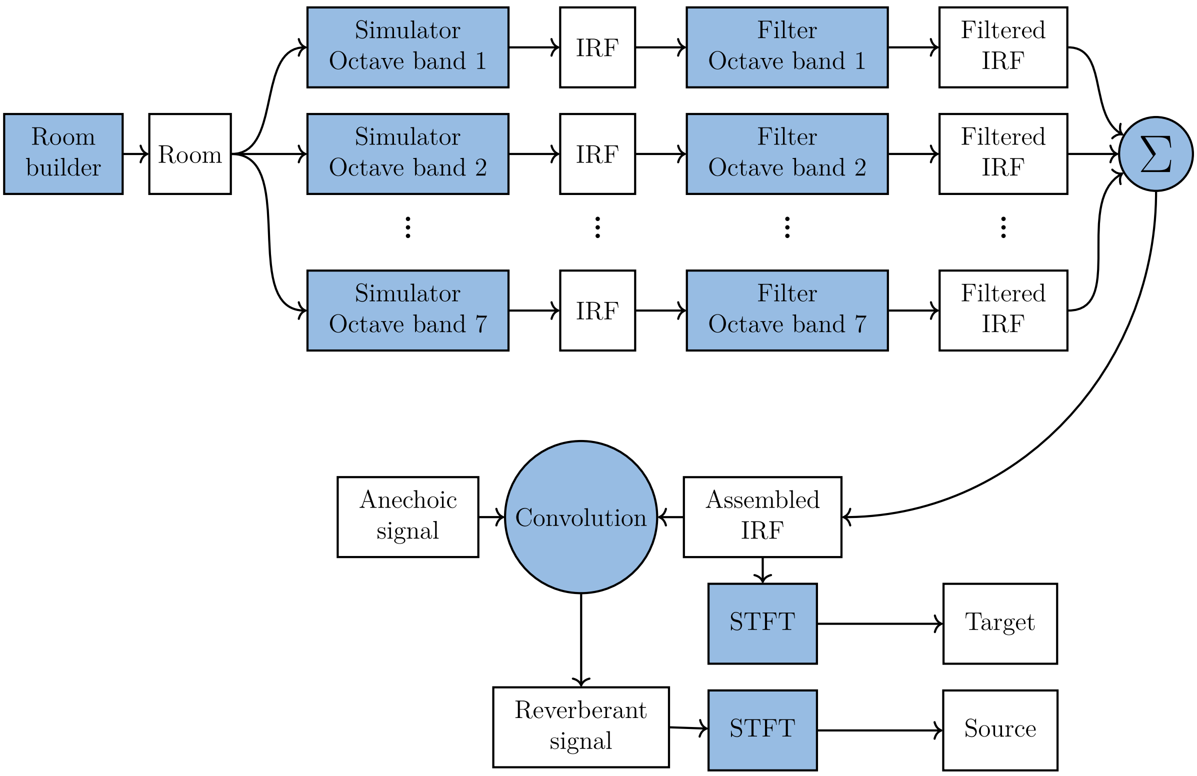
CHALMERS
UNIVERSITY OF TECHNOLOGY



Acoustic impulse response function estimation using machine learning

Estimating the room impulse response function from reverberant sound using deep neural networks

Master's thesis in Complex Adaptive Systems

ERIK LARSSON, FELIX VIBERG

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Acoustic impulse response function estimation using machine learning

Estimating the room impulse response function from reverberant sound using deep neural networks

ERIK LARSSON, FELIX VIBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Acoustic impulse response function estimation using machine learning
Estimating the room impulse response function from reverberant sound using deep neural networks
ERIK LARSSON, FELIX VIBERG

Supervisor: Jens Ahrens, Division of Applied Acoustics at Chalmers University of Technology
Supervisor: Måns Larsson, Department of Electrical Engineering at Chalmers University of Technology
Examiner: Fredrik Kahl, Department of Electrical Engineering at Chalmers University of Technology

Master's Thesis 2019
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The figure visualizes the method for generating data-target pairs used for training and validation of a network.

Typeset in LaTeX
Digital Gothenburg, Sweden 2019

Acoustic impulse response function estimation using machine learning
Estimating the room impulse response function from reverberant sound using deep neural networks
ERIK LARSSON, FELIX VIBERG
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis work discusses the possibility to extract acoustic impulse response functions from reverberant sound generated in virtual rooms, using deep learning. We propose frameworks that combine deep autoencoder networks, learning impulse response function features using point cloud representations, with convolutional networks, extracting latent representations of impulse response functions from reverberant sound. These networks take either MFCCs or full time-series as input. We also propose purely convolutional networks that both take STFT data as input and produce STFT data. Special emphasis is put on how to obtain and represent simulated data. Results from listening tests are presented that show that the impulse response function can be approximated from reverberant signals within reason. When presented with a simulated reference signal, a signal generated by our framework and a signal generated using simulations, the listeners were unable to group the simulated signals in 55% of the cases on average.

Keywords: Acoustics, Convolutional Neural Networks, Deep Learning, Image-Source Model, IRF, MFCC, RIR.

Acknowledgements

We want to thank Måns Larsson for his support and for that he was willing to share his expertise related to the machine learning aspects of this thesis work.
We also want to thank Jens Ahrens for his support and guidance related to the acoustics aspects of this thesis work, and for his contributions to the field of acoustics that were of importance for us.
We would also like to thank all friends and relatives that helped us with evaluation of our results, that is, took the roll as listeners and completed our listening test. Listeners include Anshuman Dubey, Christoffer Ekdahl, Heléna Frenell, Dundar Göc, Lars Jansson, Oscar Johanneson, Rickard Johansson, Svante Karlsson, Hanna Larsson, the Jonas Larssons, Lisa Larsson, Magnus Larsson, Per Liljesson, Susanna Nemeth, Jenny Noreklint Larsson, Jens Nilsson, Sara Nilsson, Mokhtar Mapar, Emil Reinholdtsson, Mina Warringer, Love Westlund Gotby and Massimiliano Zanoli.

Erik Larsson and Felix Viberg, Gothenburg, May 2019

Contents

1

Introduction

You are standing in an unknown room, blindfolded. Everything is completely silent until you hear a voice. You can not tell exactly where the origin is but you hear how the sound fills the room, and then fades away.

Based on this event, can you tell exactly what the soundscape of the room looks like? Without any knowledge of neither the room nor the sound source it is an impossible task as it means separating the mixed source signal and impulse response function of the room, without having any prior information of the two unknowns. But this is the task that this thesis is concerned with; approximating impulse response functions of unknown rooms based on reverberant sounds from unknown sources within it. In a mathematical sense, this means taking the output C of a convolution between two unknowns A and B such that

$$C = A * B \tag{1.1}$$

and extracting A from it. This is an underdetermined problem, in most cases impossible to solve exactly. But if we were to add additional knowledge of typical properties related to A and B we may have a chance of making an educated guess. In the scenario painted above, you actually do have additional information about the two unknowns since the sound originated from a person and you know how people typically sound. Thus you can compare this sound to your mental model of speech. You have been in rooms of many shapes and sizes before and some subconscious faculty of your brain will compare the reverberation of this room to previous ones. Thus you create a sense of the situation you are currently in.

In this thesis, the problem presented is approached as a statistical problem where deep learning models are trained to perform extraction of room impulse response functions from reverberant signals. The main idea is that given enough samples of rooms and source signals, a deep neural network might become able to separate the temporality of sound events from the persistent nature of impulse response functions. This idea is investigated through two separate approaches where one is focused around the timings and amplitudes of the early reflections in the impulse response function, while the other appreciates the importance of allowing frequencies to behave in different ways. To make the two data-driven approaches viable, a lot of data is required. Therefore the reverberant signals used to train the models are comprised of impulse response functions of simulated rooms with realistic geometrical properties, mixed with recorded signals free from reverberation, i.e. *anechoic* signals.

The ability to make on-the-fly approximations of room impulse response functions could have several uses of varying importance. It could be used to improve the level of realism achieved in augmented reality applications by having sound emission from virtual sources altered by the current soundscape in real time. A successful model could also be used for psychological experiments where one might want to trick the human perception system. Or it could be used to obtain an estimate of the acoustic properties given any specific room, without the need for time-consuming simulations and tedious measurements.

Typically, statistical problems are evaluated with statistical means, but numerically measuring the *perceived realism* of a sound is anything but trivial. Therefore this thesis combines the results from numerical measurements with perception tests conducted by human listeners. The listeners were presented with a reference audio signal obtained from simulation, and two other signals that either were simulated equivalently to the reference or generated using a deep neural model. The tests showed that the listeners confused the two signals to a large degree.

1.1 Related work

Estimation of room impulse response functions and other related acoustic properties are heavily researched areas. The traditional approach to such estimations can be divided in two different ways. With complete knowledge of the geometric properties of a room, i.e. its shape and the absorption of its surface materials, simulations can be performed with near perfect results, perceptually. The downside to this approach is that it is time-consuming, requires specialized software and that it de facto requires the precise geometric properties of a room which can be impractical to acquire. The other typical way of estimating a room's impulse response function is by creating some known sound event and measuring the response. This is a fast method but it too requires specialized measurement tools while also polluting the room with possibly unwanted sounds. An example of the second approach is [Abe+], where approximation of the impulse response functions of highly reverberant chambers was performed by recording the sound of a popping balloon. Related to the estimation of room impulse response functions is to instead find the key features of it, this is done in [Ken+08] where approximation of reverberation time, clarity and other acoustic parameters was performed by training an artificial neural network on the envelope spectrum of reverberant sound.

Much research has been conducted in topics related to the problem investigated in this thesis. Acoustic scene classification is a similar problem where an algorithm is developed to output a label on the type of scene that a sound is recorded in. In [Val+16] a convolutional neural network was trained on mel-scaled spectrograms to perform acoustic scene classification. [HL16] use a similar approach but also show that accuracy can be improved by extracting additional features from the spectral data. Similarly scene classification can be conducted using other back ends than artificial neural networks, for instance; [PLF12] conducts scene classification on mel-scaled spectrograms using Gaussian mixture models.

Various authors have conducted classification using learnable frameworks on other labels than acoustic scenes. [PCL12] describes a model that classifies artificial reverberation plugins given artificially reverberant sound. They, too, use a Gaussian mixture model as back end.

1.2 Outline and reading guidelines

The first chapter following this introduction (Chapter 2) is of theoretical character. Here the most important mechanics relating acoustics and machine learning are reviewed. In Chapters 4 through 5, two families of models are presented. Chapter 4 describes implementation and presents results related to a point-based approach and Chapter 5 is equivalent for a spectrogram-based counterpart. A comparison between the models is presented in Chapter 6 and the differences between the two models are discussed in Chapters 7. Finally, in Chapter 8, the conclusions from this project is presented.

2

Theory

The following sections gives brief theory reviews of relevant topics related to this thesis work. Acoustics related topics are presented first, in Section 2.1, followed machine learning topics in Section 2.2.

2.1 Acoustics

The following sections gives a review of the theoretical background in acoustics required by the reader in order to understand the implementations related to acoustics in this thesis work. The review includes the various aspects related to impulse response functions, MFCCs, STFT, filtering, the Griffin-Lim phase estimation algorithm and the image-source model.

2.1.1 Acoustic impulse response functions

An acoustic impulse response function is the function that transfers a signal to its reverberant counterpart, or *response*, when the signal is passed through an acoustically visible system. Impulse response functions are modelled in different ways depending on the system's time dependency, if the system is linear of not or if the signal is represented in discrete or continuous time. For this work, only linear time-invariant systems are considered and the signals are discrete. Thus the impulse response function is the function, h, that produce the response, y, when convolved with the system input signal, x, such that

$$y_n = \sum_{k=-\infty}^{\infty} x_k h_{n-k}. \tag{2.1}$$

2.1.2 The Mel Frequency Cepstrum

The Mel Frequency Cepstrum, or the *MFC*, commonly used in speech recognition applications and music analysis such as genre classification, is a compressed representation of sound signals. The MFC is based on a linear cosine transform of the logarithmic power spectrum on the nonlinear mel frequency scale, all of which explained in the following sections.

2.1.2.1 Mel scale

The mel scale relates the perceived pitch of a signal to the actual frequency. This is useful since the human perception of a sound's pitch does not map linearly with frequency. For instance, we perceive the "pitch distance" between two low frequency signals and two high frequency signals as different even though the frequency distances are equal if measured in Hertz. The mel scale was first derived 1937 using listening experiments and provides a perceptually more linear scale [SVN37]. Popular conversion formulas between f hertz and m mels are presented in Equations (2.2a) below.

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

$$M^{-1}(m) = f = 700\left(10^{\frac{m}{2595}} - 1\right) \tag{2.2b}$$

2.1.2.2 Cepstrum

Given the frequency spectrum of a signal, the cepstrum corresponding to the signal is the result of taking the inverse Fourier transform of the logarithm of the spectrum. Representing the frequency spectrum using the Fourier transform, the cepstrum can be calculated using Equation (2.3) below.

$$\text{cepstrum of } f(t) = \mathcal{F}^{-1}\Big\{ \log\Big(\mathcal{F}\{f(t)\}\Big)\Big\} \tag{2.3}$$

The cepstrum conveniently maps convolution to addition in the cepstral domain, that is, before taking the inverse Fourier transform but after taking the natural logarithm. Operations in the cepstral domain are often referred to *quefrency alanysis*, continuing to wordplay. Although the cepstrum itself is of little interest in this thesis work the cepstral coefficients mapped to the mel scale are used to calculate the MFC coefficients. The concept of the cepstrum was first introduced in a 1963 paper by Bogert et al [BP 63].

2.1.2.3 The discrete cosine transforms

The discrete cosine transforms, first conceived by Nasir Ahmed in 1974 [ANR74], are often used in compression algorithms due to their properties of half revolutions that allow non repeating signal segments to be approximated using few coefficients. There are several types of discrete cosine transforms, type-II as used in this report, is presented in Equation (2.4) below, transforming the N samples of x to N samples of X.

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \quad k = 0, ..., N-1 \tag{2.4}$$

2.1.2.4 MFC Coefficients

Given a signal, the corresponding MFC Coefficients are calculated the following way [DM80]:

1. Take the Fourier transform of a windowed signal excerpt.
2. Using a filter bank of triangular overlapping windows – map the powers of the spectrum onto the Mel scale according to Equation (2.2a), giving a list of *mels* and corresponding powers.
3. Take the logarithm of the powers at the *mels*.
4. Generate a set of coefficients using the discrete cosine transform on the list of logarithmic powers, for instance using type-II according to Equation (2.4).
5. Repeat step 1 to 4 for the entire signal to obtain a two dimensional representation.

The retrieved coefficients corresponds to mel-frequency components over time. The first coefficient is a constant and can thus be discarded.

2.1.2.5 Delta features

The first and second order delta features are the first and second order *numerical* derivatives of the MFCC coefficients respectively. These are typically used as supplementary acoustic features when dealing with noisy data, increasing the performance of e.g. speech recognition applications [KKS11]. One way to perform one-dimensional delta feature extraction is by applying a Savitzky-Golay filter to the data. The filters are applied by convolving the data, x, with a set of Savitzky-Golay coefficients, c, as

$$y_i = \sum_{i=-m}^{m} c_i x_{j+i}. \tag{2.5}$$

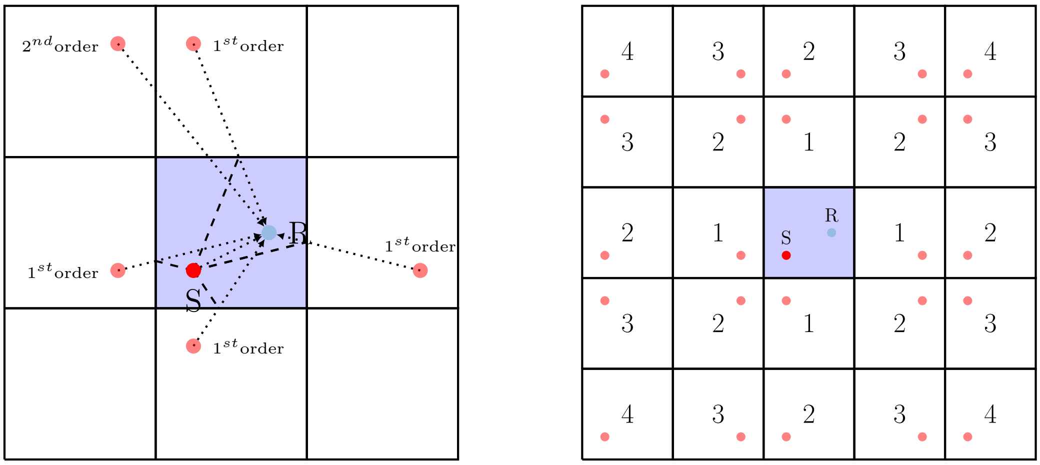
The coefficients c are found as polynomial coefficients using the linear least-squares method in a sliding window centered at x_i with width $2m + 1$. The process of fitting the data within the sliding convolution window is simplified by the linearity of the least-squares method [SE51] [PT90]. The filtered output for index i, y_i, is the approximated delta features at each time-step.

2.1.3 The image-source model

The image-source model is a simple and well-known technique used for impulse response assembly. The technique provides a way to find paths between two points inside a bounded simple geometry and involves first calculating mirror source positions outside the system boundary and, in the simplest case, calculating the number of reflections needed to traverse the system. For impulse response assembly, a delta function signal is delayed proportionally to the distance between the mirror source and the receiver, ignoring boundaries, and the amplitude of the signal is reduced proportionally to the order and corresponding absorption coefficients. The delays and amplitudes of the delta functions form the complete simplified impulse response function. The technique is based on tiling copies of the geometry and deducing where to place a "mirror source" within each tile. The method assumes that all signals propagate in straight lines at equal speed and that the energy decreases with the distance travelled as $1/r^2$. Further it assumes that sound is perfectly reflected at the boundaries with an amplitude reduction proportional to the absorption coefficient of the boundary regardless of frequency [AB79]. Figure 2.1 visualizes the fundamental properties of the image-source model. The image-source model is the means of simulation used for this thesis work and forms the baseline for presented results.

2.1.4 Filters

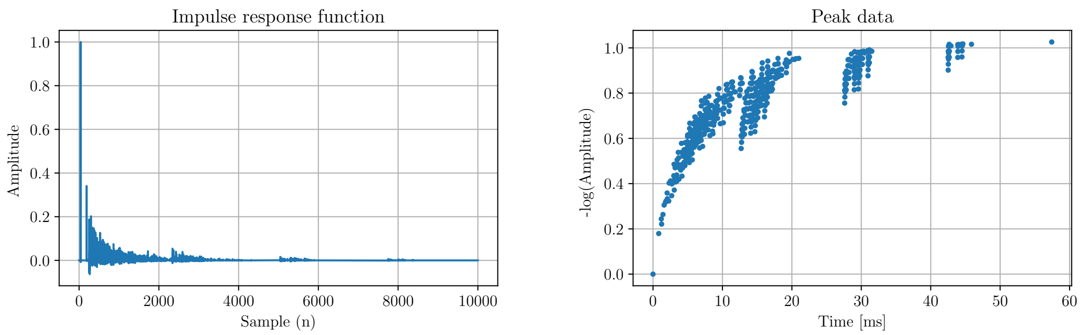
In signal processing, filtering is the process of removing some unwanted feature of the signal. If a signal is viewed as a collection of superposed signals each with fixed frequency, that is, as a Fourier series, then a low-pass filter would suppress the coefficients for any frequency component with a frequency higher than the cut-off frequency. A low-pass filter lets constituents in a signal with low frequency pass through the filter unaltered. Analogously, a high-pass filter lets constituents of a signal with frequencies higher than the cut-off frequency pass through the filter. A band-pass filter is a filter with two cut-off frequencies that attenuates any constituent with a frequency that does not reside between the cut-off frequencies.



(a) The figure shows the positions of the source, S, and receiver, R, and some paths related to the positions.

(b) The figure shows the mirror image pattern that corresponds to the initial source position and the reflection orders.

Figure 2.1: The figures visualize the image-source model and how it, due to the emerging pattern, is generalized for simple room geometries.



2.1.4.1 Butterworth filter

The Butterworth filter, conceived in 1930 by S Butterworth [But+30], is a set of filters designed to have a flat frequency response over the passband - the range in frequency space that corresponds to the passed signal constituents. The Butterworth filter can have different orders that corresponds to the number of degrees in a polynomial function used for the definition. A high order Butterworth filter leads to a steeper attenuation outside the cut-off frequencies but in electronics higher orders also correspond to a larger amount of components in the actual filter. In digital filtering and mathematics, however, Butterworth filters of arbitrary order can be designed.

A Butterworh filter of the n:th order has the amplitude response

$$|H_n(j\omega)| = \frac{1}{\sqrt{1+\left(\frac{\omega}{\omega_c}\right)^{2n}}}, \tag{2.6}$$

where ω is the angular frequency and ω_c is the cutoff-frequency.

Using ω_c=1 and introducing $s = j\omega$ we can write

$$H_n(s)H_n(-s) = \frac{1}{1+\left(\frac{s}{j}\right)^{2n}}, \tag{2.7}$$

with the $2n$ poles where $(s/j)^{2n} = -1$ such that

$$s_k = \exp\left(\frac{j\pi}{2n}(2k+n-1)\right) \tag{2.8}$$

for $k = 1, 2, ..., 2n$. Which leads to that $H_n(s)$ can be written

$$H_n(s) = \frac{1}{(s-s_1)(s-s_2)...(s-s_n)} = \frac{1}{B_n(S)} \tag{2.9}$$

where $B_n(s)$ is the n:th order Butterworth polynomial.

2.1.5 Short-time Fourier transform and the Overlap-add method

The (discrete) short-time Fourier transform, or $STFT$, introduced in 1977 [All77], is used to obtain the frequency spectrum of a signal. The procedure is based on windowed fast Fourier transforms performed on short excerpts of the signal, the lengths and positions of which are determined by the window size. The window is usually slid over the signal using a hop length smaller than the size of the window, such that the windows overlap. The two-dimensional complex spectrogram is then obtained by simply appending the resulting spectra to form a two-dimensional matrix. The discrete STFT is given by the following equation:

$$STFT\{x[n]\}(m,\omega) = X(m,\omega) = \sum_{x=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n}. \tag{2.10}$$

where $w[n]$ is a window function and $x[n]$ the signal to be transformed.

In order to be able to reconstruct a time-signal from a spectrogram resulting from an STFT, the time-frames in the forward transformation must overlap to a certain degree depending on the type of window used. An STFT spectrogram is generally considered to be reconstructable perfectly if the forward transformation meets the, Constant Overlap-add, or *COLA*, constraint. The COLA constraint states that the sum of the overlapping windows must equal unity, that is;

$$\sum_{m \in \mathbb{Z}} w(n - mR) = 1, \forall n \in \mathbb{Z},$$

where R is the hop length of the transform.

One can show that Hanning windows gives constant overlap, hence meets the COLA constraint, for hop lengths of $(M-1)/2$ and $(M-4)/2$, where M is the number of samples in each time-frame. With the COLA constraint met, the inverse STFT is the result of the inverse Fourier transform of the complex vector at each time, and addition for the overlapping time signal outputs.

2.1.6 Phase reconstruction and the Griffin-Lim Algorithm

In order to reconstruct a time-signal from an STFT spectrogram, estimated using machine learning or otherwise, the phase information of the spectrogram must be present for the reconstruction to be correct. This means that the spectrogram must be complex, which is often not the case. However, the phase information can somewhat be reconstructed if the windows of the STFT are overlapping, using only the magnitude of the STFT spectrogram [1] by utilizing the Griffin-Lim algorithm [GM84]. The Griffin-Lim algorithm is based on that some phase information is implicitly encoded into the magnitude spectrogram because of the overlapping windows.

The Griffin-Lim Algorithm states that

Algorithm 0: Griffin-Lim Algorithm for phase estimation. Let S be the STFT-operator and $|Y|$ the given STFT magnitude spectrogram.

Result: Phases x

Guess x;
while *Not converged* **do**
  $X \leftarrow S(x)$; `// Calculate the STFT of` x
  $Z \leftarrow |Y| \exp(i\angle X)$; `// Set STFT magnitude to` $|Y|$
  $x \leftarrow S^{-1}(Z)$; `// Compute new phase information` x
end

[1]This is important since the type of networks explained in Chapter 5 generates the magnitude of the spectrogram only.

2.1.7 Octave bands

Due to the same reasons that lead to the use of the Mel scale, that the human perception of a sound's pitch does not map linearly with frequency, sound frequency is often split into regions, or bands, that span roughly equal parts of the perceived pitch spectrum. However, for instance when measuring the frequency dependency of materials Octave bands are often used. An Octave band is a range on the frequency axis where the upper frequency limit of the range, f_{max}, is twice the frequency of the lower limit, f_{min}. The mid-point f_c then becomes $\sqrt{2}f_{min} = f_c = f_{max}/\sqrt{2}$. A table of standard mid-point frequencies and limits are presented below.

Table 2.1: The table lists the first Octave bands and their limits rounded to nearest integer

Octave bands and their center frequencies		
Lower limit	Center frequency	Upper limit
11	16	22
22	31.5	44
44	63	88
88	125	177
177	250	355
355	500	710
710	1000	1420
1420	2000	2840
2840	4000	5680
5680	8000	11360
11360	16000	22720

2.2 Machine Learning

The following sections gives a review of machine learning theory knowledge required by the reader in order to understand the implementations related to machine learning in this thesis work. The review includes basic theory behind neural networks and deep learning including layer-types, optimization and loss functions, as well as aspects related to training on point-clouds. This section also gives a brief review of why and how to use autoencoders.

2.2.1 Neural networks

Artificial neural networks are computing systems that can learn to solve specific tasks, inspired by the biological counterpart found in brains. Such biological neural networks are composed of large numbers of nerve cells called *neurons*. The neurons are connected to *dendrites* that transfer input signals to the neuron cores. The neuron cores, if stimulated above certain thresholds, outputs response signals via *axons* that connects to other dendrites belonging to other neurons. This arrangement permits highly complex information transfer networks composed of small simple units. In an artificial neural network the neuron is replaced by the *perceptron*, an even simpler unit since it cannot have any time dependency, which must not be the case for the neuron. The perceptron sums a set of inputs and pass the result through an *activation function*, introducing nonlinearity to the system, to generate an output. A simple artificial neural network can for instance be composed of a single perceptron that sums a set of inputs and pass the result through the Heaviside step function which determines the network output. A more complex network that enable more complex inference might be composed of two layers of perceptrons where the output of the perceptrons in the first layer is the input of the perceptrons in the second layer. Further the connections between perceptrons may be related by certain weights and the perceptrons might add biases to the summation result before applying the activation function.

A perceptron can be modelled according to Equation 2.11 below.

$$y = \Phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.11}$$

where y is the output from a perceptron, w_i is the weight associated with input x_i, N is the total number of inputs, b is the bias, and Φ is the activation function.

A system encapsulating several layers, each with its own set of nonlinear activation functions and weights, constitutes what is typically referred to as an artificial neural network. An example of which is visualized in Figure 2.3. Artificial neural network can approximate functions that yield sought output given a sample from an input class. For instance, a neural network might separate the two subsets A_1 and A_2 where A_1 contains visual representations of some object, for instance a dog or a car, whereas A_2 does not. In order to permit a neural network to separate the sets requires that the weights and biases of the network are calibrated correctly. This is achieved through a series of updates referred to as *training* or *learning*. A common way of updating the weights and biases is through *backpropagation* which is described in Section 2.2.2.3.

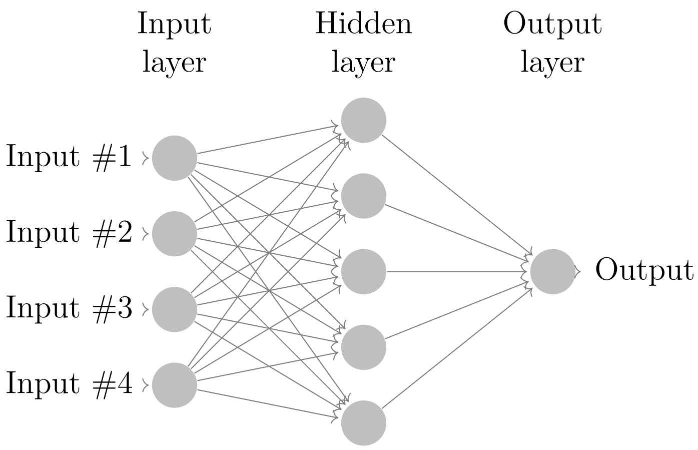


Figure 2.3: The figure shows a graphical representation of a small neural network. Here, each circle represents a simple perceptron that can be associated with a set of weights, a bias and an output, see Equation (2.11).

2.2.2 Loss functions and learning

In order to evaluate performance of a network, its output has to be compared to a sought output, a *target*. The function or group of functions that measures distance between network output and target is referred to as a *loss function*. In the following sections the loss functions used in this thesis work are described.

2.2.2.1 Mean squared error

The Mean squared error can be described as the average distance between two sets of data points that maps one-to-one.

Given sets X and Y, construct d_{MSE} such that, if

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix}^T \qquad Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_N \end{bmatrix}^T, \tag{2.12}$$

then,

$$d_{MSE} = \frac{1}{N}\sum_n l_n, \text{ with } l_n = (x_n - y_n)^2 \tag{2.13}$$

2.2.2.2 Chamfer distance

The Chamfer distance is a measure of the overlap of two unordered sets that permits the datasets to be of different size. The Chamfer distance is small if all points in one set are close to points in the other set. The Chamfer distance is easily symmatrized and in this thesis, the symmetric Chamfer distance is used.

The symmetric Chamfer distance, d_C, between two sets, $\mathbb{X}$ and $\mathbb{Y}$, is defined as

$$d_C(\mathbb{X}, \mathbb{Y}) = \frac{1}{2}\left(\frac{1}{n(\mathbb{X})}\sum_{\vec{x}\in\mathbb{X}}\min_{\vec{y}\in\mathbb{Y}} \mathrm{d}(\vec{x}, \vec{y}) + \frac{1}{n(\mathbb{Y})}\sum_{\vec{y}\in\mathbb{Y}}\min_{\vec{x}\in\mathbb{X}} \mathrm{d}(\vec{x}, \vec{y})\right), \tag{2.14}$$

where $\mathrm{d}(\vec{x}, \vec{y})$ is the Euclidean distance between the points $\vec{x}$ and $\vec{y}$ and $n(\cdot)$ denotes the cardinality of a set.

2.2.2.3 Stochastic gradient descent and weight update

Gradient descent is an iterative method for finding a local minimum of a differentiable multivariate function and makes use of the first order derivative at the point of evaluation. Since the gradient points in the direction of the steepest function value increase, it follows that for an infinitesimal step in the opposite direction, the function value is decreased. Equation (2.15) defines the iteration that, for small enough steps, η, decreases the function value until a minima is reached.

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \eta\frac{\partial\tilde{\mathbf{y}}}{\partial\mathbf{w}_n} \tag{2.15}$$

For artificial neural networks with weights $\mathbf{w}$, input $\mathbf{x}$ and output $\mathbf{y}$ such that for every layer; $\mathbf{y} = \mathbf{w}\cdot\mathbf{x}$, Equation (2.15) can be used to locate minima in weight space. Using $\tilde{\mathbf{y}} = \mathbf{d}(\mathbf{F}(\mathbf{y}, \mathbf{w}))$ where $\mathbf{d}(\cdot)$ is a differentiable loss and $\mathbf{F}(\mathbf{y}, \mathbf{w})$ defines the operation of the network the chain rule will apply. Intermittent derivatives can be used to update weights for each layer which means that the gradients must only be calculated for a single error propagating through the network. This way of updating weights is commonly referred to as *backpropagation*.

For artificial neural networks it is common to update the weights stochastically where stochasticity stems from passing random collections of input data through the network. Given a *batch* of K samples, the stochastic weight update is given in Equation (2.16).

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \eta\sum_{k=1}^{K}\frac{1}{K}\frac{\partial\tilde{\mathbf{y}}_\mathbf{k}}{\partial\mathbf{w}_n} \tag{2.16}$$

A detailed example of how to update weights in a fully connected neural network with one hidden layer can be found in Appendix B.1.

2.2.2.4 Optimization methods

As was explained in Section 2.2.2.3 the update of weights during optimization is based on the gradient of the loss function with respect to the weights. Machine learning frameworks for deep learning often, however, use more refined methods for weight updates than the ones given in Equations (2.15) and (2.16). The optimization algorithms often include a momentum term that stores previous update directions and weigh them against the new update to permit the descent to easier traverse past local minima in weight space thus speeding up convergence. An example of momentum based optimization is the commonly used SGD algorithm that stems from a 1951 paper by Robbins and Monro [RM51]. Adam [KB14] is another method for stochastic gradient descent that store and use running averages of both the gradients and the second moments of the gradients. The algorithm first calculates the first and second order moment estimate vectors according to

$$\begin{aligned} m_w^{(t+1)} &\leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w H^{(t)} \\ v_w^{(t+1)} &\leftarrow \beta_2 m_w^{(t)} + (1-\beta_2)(\nabla_w H^{(t)})^2 \end{aligned}$$

where β_i are decay terms and $H^{(t)}$ are the calculated loss for time step t. The moments are then divided with a bias correction term according to

$$\begin{aligned} \hat{m}_w &= \frac{m_w^{(t+1)}}{1-\beta_1^{t+1}} \\ \hat{v}_w &= \frac{v_w^{(t+1)}}{1-\beta_2^{t+1}}. \end{aligned}$$

This counteracts zero-initialization, note here that in the denominator the superscripts are exponent factors. The weights are updated according to

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_m} + \epsilon},$$

where η is the step size and ϵ is a small bounding number typically in the order of 10^{-8}.

2.2.3 Activation functions

Activation functions that introduce nonlinearity are necessary in order to allow artificial neural network to find nontrivial solutions to complicated problems. Not using activation functions in an artificial neural network is to assume that the problem is linear.

2.2.3.1 ReLU

The ReLU or *Rectified Linear Unit* is one of many common activation functions and frequently used throughout this thesis project, see Equation (2.17).

$$f(x) = \max(0, x), \quad f : R \rightarrow [0, \infty[\tag{2.17}$$

2.2.3.2 Tanh

Tanh or *Hyperbolic Tangent* is a common activation function defined on a different range than the ReLU activation function according to Equation (2.18).

$$f(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \quad f : R \rightarrow]-1, 1[\tag{2.18}$$

2.2.3.3 Leaky ReLU

The Leaky ReLU activation function is similar to the ReLU activation function but permits small gradients for negative input. See Equation (2.19).

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}, \quad f : R \rightarrow]-\infty, \infty[\tag{2.19}$$

2.2.4 Pooling

Pooling is a non-linear operation for spatial down-sampling of the data. It is performed by dividing the data into a number of regions and performing some operation on each region. One common type of pooling is max-pooling which takes the maximum value in the region and discards the rest. Another type is average-pooling which takes the average of the region. The data can be divided into subregions in different ways depending on the selected kernel size and stride, that is, the area to be pooled and the spacing between two target areas. Figure 2.4 shows an example of max-pooling on two-dimensional data with a single feature dimension using kernel size $[2 \times 2]$ and stride 2.

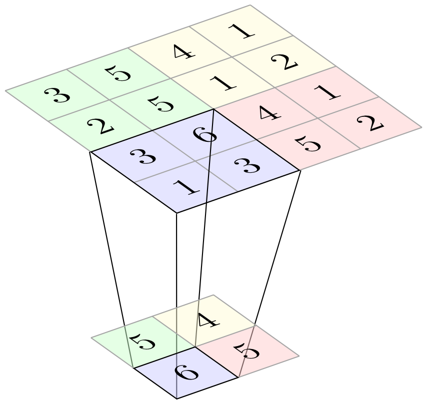


Figure 2.4: The figure visualizes max-pooling performed on two-dimensional data with a single feature dimension. The pooling is performed with kernel size $[2 \times 2]$ and stride 2 on the upper layer, which reduces the spatial size by a factor of 4. The bottom layer is the output of the operation.

2.2.5 Deep learning

Artificial neural networks consisting of many layers between input and output are typically referred to as *deep* networks and belong to a subfield of machine learning referred to as *deep learning*. Deep learning permits higher levels of abstractions of input data and thus more refined outputs. Since training deep neural networks requires that many simple calculations are carried out, it is not feasible to train using CPUs. Instead large models are typically trained using GPUs, utilizing the parallellizing capabilities of the many cores in a GPU.

2.2.6 Convolutional networks

Convolutional neural networks (CNNs) are a type of translation invariant artificial neural networks that utilize weight sharing, commonly used in computer vision applications. This is done by convolving learnable numerical kernels with the data and feeding the result of this convolution to subsequent layers. By design, this means that features in the data can yield large activations upon convolution independent of the positions of the features within the data. What features yield large activations depend on the parameters of the kernels, which are tuned during the training phase. The process of generating an output by convolving a single two-dimensional kernel with two-dimensional input data with a single feature dimension is visualized in Figure 2.5. The filter can be of arbitrary shape as long as it it smaller than the data dimensions.

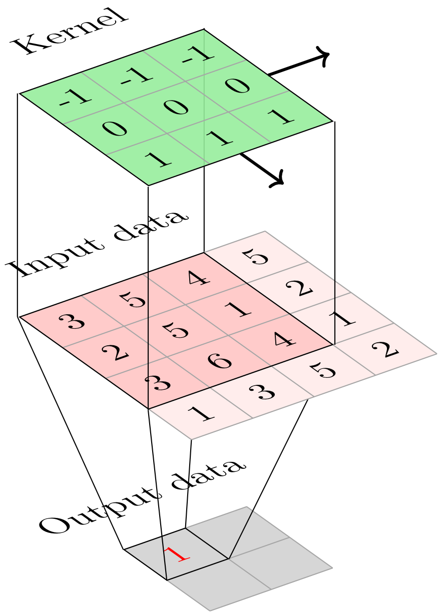


Figure 2.5: The figure shows the process of convolving two-dimensional input data with a single two-dimensional kernel to generate output data. In a convolutional layer of a neural network, a large amount of kernels are used in order to generate an output with a larger feature dimension. The kernel weights are tuned during the training process.

2.2.7 Autoencoders

Autoencoders are a type of artificial neural networks that are used to find efficient encodings of a given type of data. An autoencoder network is composed of an encoder and a decoder, with a narrow bottleneck in between. The task of the encoder is to find a small representation of the input data without losing any important information, where the size of the latent representation is governed by the size of the bottleneck. The decoder takes the latent representation of the data and reconstructs it into its original form.

The performance of an autoencoder given some compression ratio is typically measurable. Depending on how the data is represented one may for instance use the Mean squared error, as presented in Section 2.2.2.1, and compare the pixel-wise distance between input and output. The figure obtained is referred to as the *reconstruction loss.*

2.2.8 Point clouds and one-by-one convolution

Treating data as multi-dimensional point clouds provides an efficient way to represent sparse, irregular data without discarding any of the positional information. A point cloud is an unordered set of data points that can represent any object with the precision of the data type that stores the point's position, instead of the resolution of a raster.

When using a d-dimensional non regular point cloud as input to a convolutional neural network there is generally no reason to use a kernel size larger than one. Convolutional layers with kernel size larger than one activates in accordance with the values of adjacent elements, adjoinability being a never realized property of non-rasterized data. Because of the lack off correlation between coordinate order and value order a convolutional filter can not be assumed to activate in any regularized manner for unsorted data. Solutions to the problem of using unsorted data in convolutional networks may be to rasterize the data on a d-dimensional grid in order to store proximity information in an easily retrievable manner. Unfortunately, rasterization also means that loss of precision is prone to occur depending on the data type used. Encoding point clouds using a series of one-by-one convolutional layers ahead of a decoder can qualitatively be regarded as passing the data through a learnable nonlinear function that updates such that the represented data is easily separatable for the decoder, following the ideas by Lin et. al. [LCY13].

3

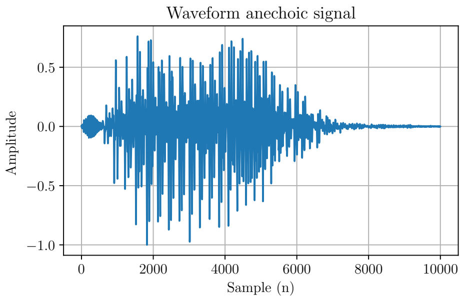
Generating data

In this thesis, all data used for training models is derived from reverberant audio signals. A reverberant signal is the result of convolving an anechoic signal with a room impulse response function. A large amount of data is necessary to be able to train the types of models used in this thesis, therefore it is impractical and time-consuming to measure the IRFs from real rooms. Luckily, given that the geometrical properties of a room is known, a perceptually good IRF can be generated by simulatory means. The steps taken to achieve these simulations are described in Sections 3.2 and 3.3. The generated IRF can then be convolved with anechoic audio signals to form a reverberant signal. The recording of anechoic audio signals is described in Section 3.1.

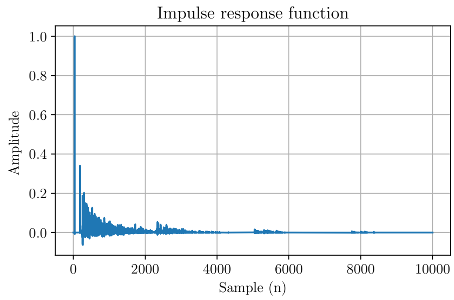
3.1 Recording anechoic audio

A set of short waveform segments with audio data was taken from a longer recording of the authors' voices captured in the Anechoic Chamber at the Department of Technical Acoustics at Chalmers University of Technology, using the built-in microphones on a MacBook Pro Retina late 2013. An algorithm was written that searches signals for transient sounds, that is, short excerpts containing information onset and complete or near complete decay. An example of such transient signal can be seen in Figure 3.1a. The algorithm uses `onset_detect` from the Python package LibROSA[1] [McF+15] for onset detection and splits the signal at the onsets, $t = t_{o,i}$ and at $t = t_{o,i} + f_s/4$ giving excerpts of 250 ms duration. Signal excerpts that contain information that do not decay within the time window are discarded by the algorithm. The algorithm controls for decay using the sum of the tail of the absolute of the signal and compares the sum to a threshold value. The value used for the threshold was 2 for the last 1000 samples, for data between -1 and 1 sampled at 44100 samples per seconds.

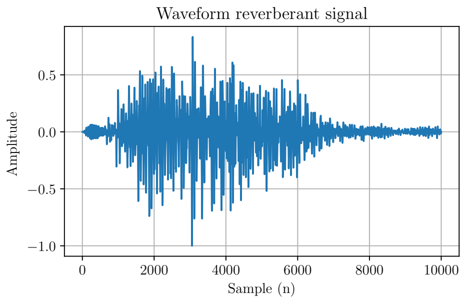
[1]https://librosa.github.io



(a) Signal



(b) Impulse response function



(c) Reverberant signal

Figure 3.1: The figure shows convolution of an anechoic signal and an impulse response function to obtain a reverberant signal. Figure (a) shows a transient excerpt of a longer signal recorded in an anechoic chamber. Figure (b) shows a simulated impulse response function. Figure (c) shows the result of the convolution.

3.2 Generating virtual rooms in python

A process for generating rooms of various shapes was developed. The process creates rooms by generating two rectangles of random but realistic size and either taking the union or the difference of the two rectangles, each with 50% probability. The rectangles were created sequentially such that limitations of the second rectangle could be calculated due to the shape of the first. The second rectangle were placed with its corner of origin inside the first rectangle and its dimensions were controlled such that its second corner, describing the rectangle completely, were not situated near the borders of the first rectangle. This counteracts unwanted cases such as those with very thin corridors near the edges of the room. Further, if the second rectangle was randomized in such a way that it ended up being completely inside the first rectangle the possibility of taking the difference of the rectangles were removed such that these cases always ended up being rectangular rooms. The reason for this is partly that O-shaped rooms are extra prone to heavy information loss during simulations and partly because that they are extra prone to spawning source and microphone positions that are not directly acoustically visible which is an unwanted case - as shall be discussed further in proceeding sections. The resulting geometric shape, with between four and eight corners, describes the floor of a virtual room and were thus extruded such that a volume was obtained, describing the room geometry completely. The geometrical shapes were produced-, and geometrical operations were performed, using the geometry module in the Python package Shapely[2] [Gil18]. All dimensions were generated randomly within realistic boundaries; typical for medium sized rooms. An example of a room generated as the union of two rectangles is shown in Figure 3.2.

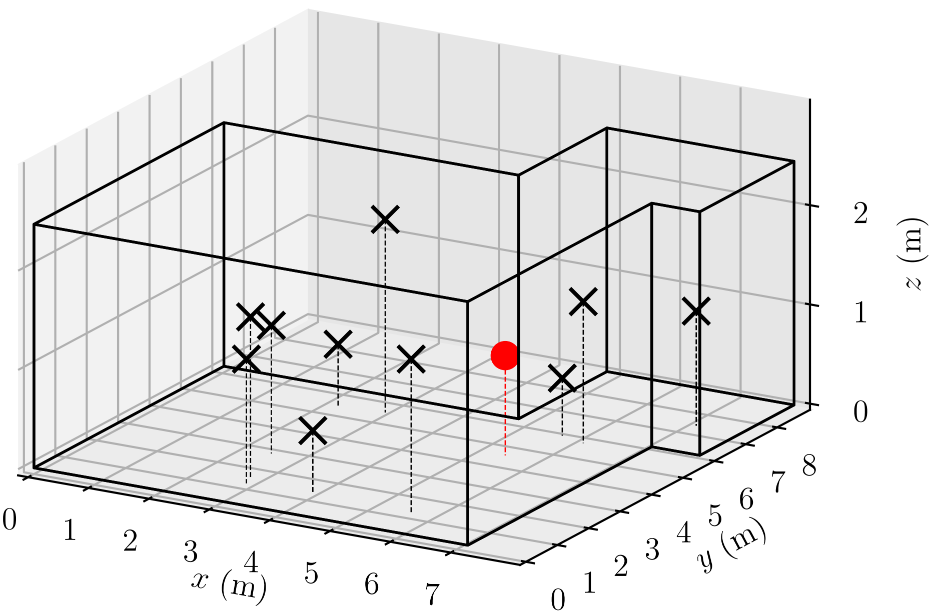


Figure 3.2: The figure shows a randomly generated room with one source position (the dot) and 10 microphone positions (the crosses). The room shown are the union of two rectangles

3.3 Simulating sound propagation with the image-source model

Simulation of the propagation of sound was performed using the image-source model [AB79]. In order to perform a simulation, a room geometry was generated as described in Section 3.2. Further, each surface of the room was given an absorption coefficient chosen within a given realistic interval. A single source position and 10 microphone positions were generated such that they were all within the boundaries of the room with a minimum distance away from boundaries. This was done due to that the simulation software did not handle the proximity effects that occur in reality when sources or receivers are close to surfaces. Further, the microphone positions and source position were ensured to be in direct visibility of each other, thus

[2]https://pypi.org/project/Shapely/

guaranteeing presence of one direct sound, which proved to be of importance for acceptable results.

With the geometric properties of the room set up, the simulations were performed using the Python package Pyroomacoustics[3] [SBD17], which includes implementations of the image-source model for three-dimensional geometries. As explained in Section 2.1.3, the simulations rely on summing signals having traversed the system using less than some fixed amount of reflections, or *orders*. The maximum order greatly influence the required amount of time needed to complete a simulation.

3.4 Data generation process for Peak-data and Signal-data

With methods for generation of rooms and making simulations, room impulse response functions could be obtained for each of the source-microphone pairs. An example of a resulting impulse response function can be seen in Figure 3.1b.

A complete framework was developed for generating training- and evaluation databases consisting of input data of two different types as well as of corresponding target data to be used for training the deep neural models with which this report is concerned. The steps for generating such databases includes generating room geometries, simulating sound propagation inside the rooms and extracting features from the resulting impulse responses and reverberant signals. Figure 3.3 is a diagram that visualizes the database generation method that was used. The figure describes a framework that yields two different types of input data and one target data type. The framework outputs reverberant signals represented both as MFCCs and time-series to be used with networks that take such data as input. A database was defined as a catalog of .npy-files, containing input data and target data, and a .csv-file storing paths to the associated files.

[3]https://pypi.org/project/pyroomacoustics/

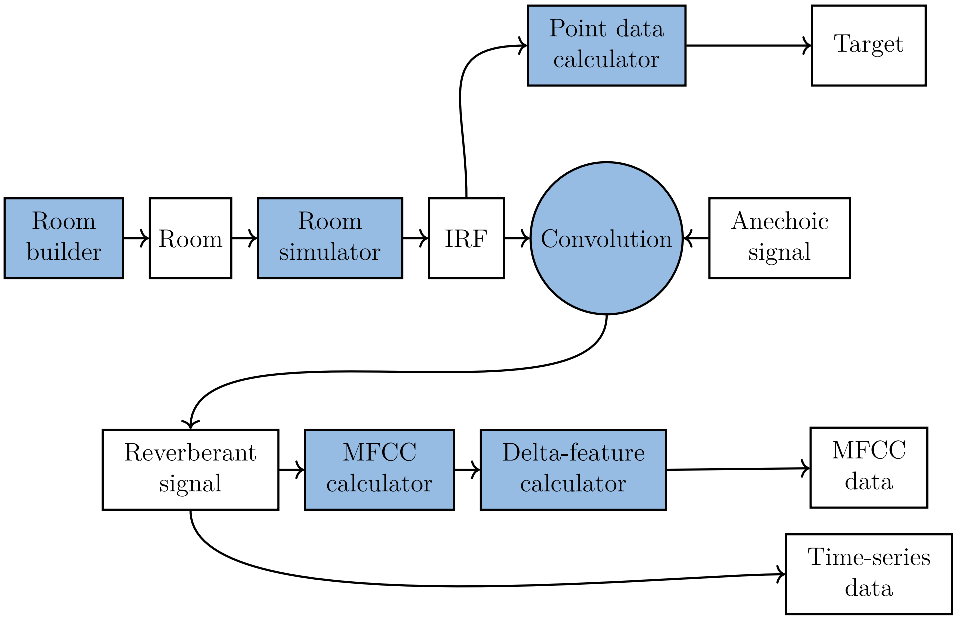


Figure 3.3: The figure shows a diagram that visualizes the method for generating input-target pairs used for training and validation of the networks. The methods described in this thesis work use either MFC coefficients or time-series as input.

3.5 Expanding data generation for frequency dependence

The framework described in the previous sections does not produce frequency dependent output since the image-source model implementation used does not have methods to handle this kind of output. However, simulating each situation several times for different absorption coefficients, each set corresponding to a different frequency octave band, and filtering the output for the corresponding frequencies generalizes the process for frequency dependent impulse response function retrieval. The retrieved impulse response function can not be described completely by the use of timings and amplitudes since each peak would have a different shape - more on this topic in the next section.

In order to include frequency dependence into an already existing framework, explained in Section 3.4, the database generation process had to be altered to encompass the multiple simulations needed for each network training sample.

3.5.1 Simulation orders and room geometry

The image-source model generation process for impulse response functions inevitably results in that most of the samples lie at amplitudes near zero; at least in the early part of the impulse response function. It is only at, or close to, samples that maps to incident times that deviate heavily from zero amplitude. This fact is preferable, or at least convenient, when working with peak data due to that it leads to small representations such that each peak data time, and amplitude, can be mapped to input or output neurons directly. This means that the zeros in the impulse response function are discarded while packaging the data. For frequency dependent impulse response functions, however, each simulated incident results in peaks of more exotic shape instead of delta functions, therefor frequency dependence cannot be described using only amplitude and timing for each incident. To account for this, the short-time Fourier transform of the signal and impulse response function was used as source and target data, respectively.

To ensure that every short-time Fourier transform time frame holds enough relevant information the orders used for simulation in the image-source model had to be increased considerably. Contrary to the frequency independent model, for which it was desirable to be able to store the impulse response function using as little data as possible, the number of incidents does not affect the memory size of the resulting impulse response function for the frequency dependent model since the STFT matrix is independent of the underlying signal information density. Increasing the orders of simulation, however, leads to an increase in simulation time. Since the simulation needs to be run once per octave band in order to yield one input data representation, also increasing the simulation orders would seem to inevitable increase the simulation time further. However, limiting the number of floor corners to four enables the image-source model to be much more efficient and simplifies the calculations considerably such that the image-source model can utilize the generalization explained in the theory section (see Figure 2.1), but in three dimensions. For rectangular floor shapes the number of orders could be increased by a factor of six with kept simulation duration even though the simulation had to be run once for each of the seven octave bands - suggesting an efficiency increase of around a factor of 50 for rectangular-floor rooms compared to rooms with six or eight floor corners. The room geometries for the frequency dependent model are therefor all rectangular. Although rectangular, unfurnished rooms with surfaces that reflect acoustic waves in an idealized manner might seem to limit the possibility of realistic output for measured reverberant sound significantly, input from generally shaped rooms would still produce output when passed through a deep neural model due to the nature of artificial neural networks. The hope is thus that the trained model generalizes to arbitrarily shaped rooms.

3.5.2 Materials

Instead of randomly choosing one single realistic absorption coefficient for each surface of the virtual room, a realistic set of frequency dependent absorption coefficient functions were chosen. The number of coefficients at each surface were limited to seven, corresponding to the octave bands with centers around between 32 Hertz and 8000 Hertz as listed in Table 2.1. A list of materials with known absorption over the frequency spectrum were used from which coefficients for each surface were randomly drawn with the certain limitation that each material could only be drawn for certain surfaces, for instance the material `wooden-floor` can only be drawn for floors whether `concrete-porous` can be drawn for all surfaces; walls, ceilings and floors. The measured octave band sets of absorption coefficients are collected from the work of [Vor08].

3.5.3 Simulation and filtering

To obtain an impulse response function sample the system was simulated several times, varying the absorption coefficients of the virtual room for each simulation. Each simulation resulted in a non frequency dependent impulse response function that was then filtered using the frequency band limits that corresponded to the coefficients used. In other words, given the absorption coefficients that corresponds to some certain octave band, the room was simulated using the coefficients and the resulting signal was then passed through the band-pass filter that covers the octave band. The resulting signals were superposed to construct the wide-band impulse response function. Figure 3.4 below shows a part of an impulse response function as its various frequency components.

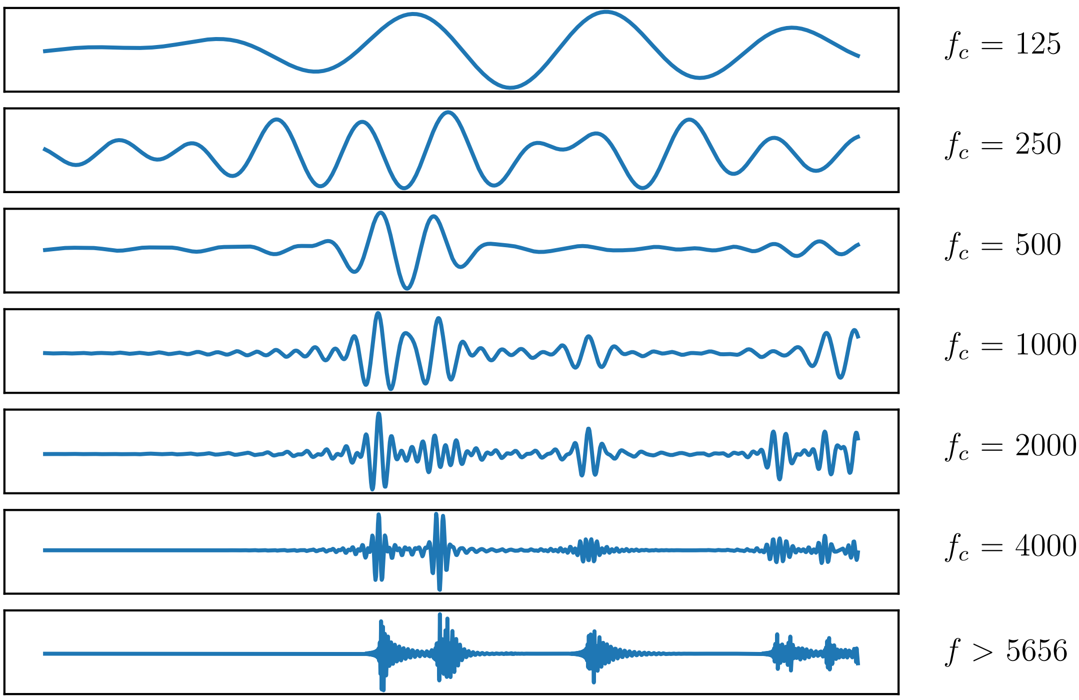


Figure 3.4: The figures shows the filtered constituents that makes up the complete frequency dependent impulse response function when superposed. Note that the y-axes of the figures are rescaled for the sake of clarity.

3.6 Data generation process for spectrogram-data

A complete framework was developed for generating training- and evaluation databases consisting of frequency dependent input- and output data. The steps for generating such databases includes generating rectangular room geometries, simulating sound propagation inside the room for seven different frequency bands, filtering and summing the results and taking the short-time Fourier transform of the convolution of the sum and an anechoic signal.

Figure 3.5 is a diagram that visualizes the database generation method that was used. The framework outputs reverberant signals and impulse response functions represented as their respective STFT spectrogram.

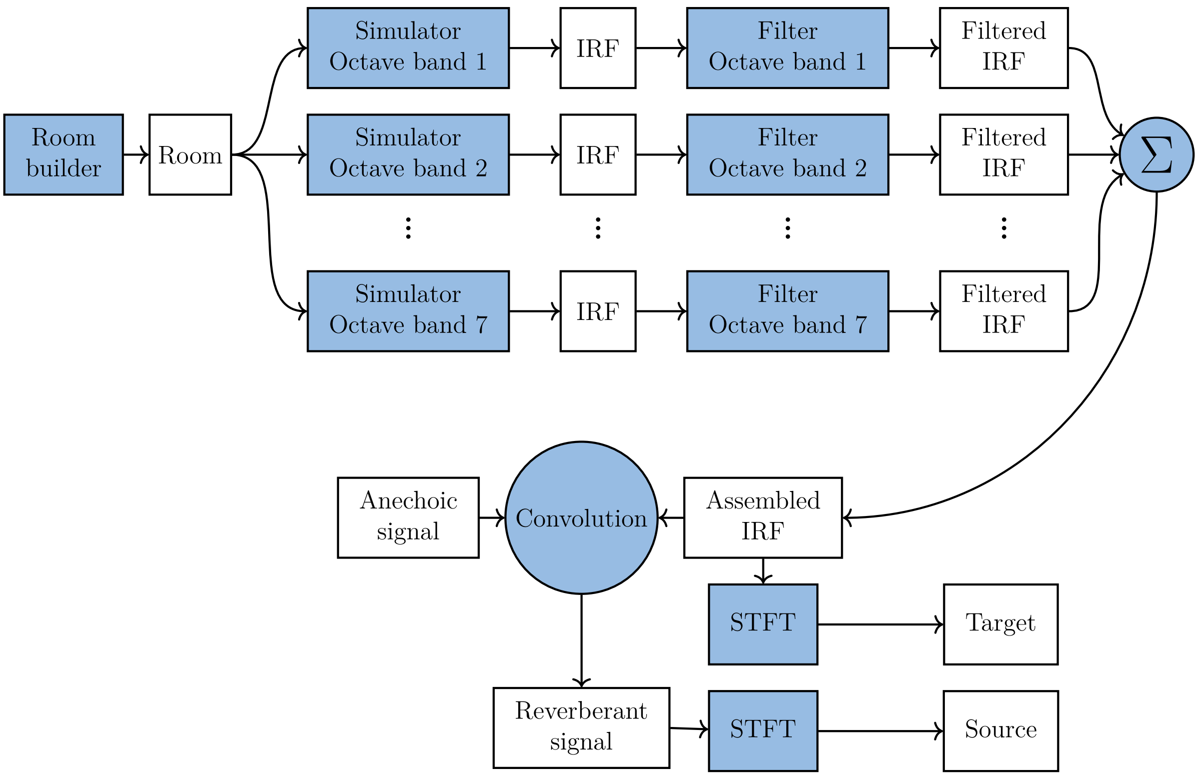


Figure 3.5: The figure shows a diagram that visualizes the method for generating input-target pairs used for training and validation of the frequency dependent networks.

4

Point-based representation

A framework was developed to approximate impulse response functions based on reverberant signals, where the impulse response functions are represented as points in a two-dimensional space. This chapter describes the implementation of the elements essential to enable such framework. The following sections describe data and signal pre- and post processing, different network architectures, and presents some of the results.

4.1 Data processing

This section describes necessary steps to prepare data generated as explained in Section 3.4 in order to use a point-based representation together with deep neural models.

4.1.1 Data pre-processing

For training some of the models used during this work, the input data consisted of mel-frequency cepstral coefficients and its first and second derivatives, called delta features, see Section 2.1.2.5. MFCCs have been proved to encode perceptually relevant data from audio signals [DM80] and is a common way to represent data when applying machine learning to acoustics related tasks [Hyd+17]. Extracting delta features and including them in the input data has been shown to increase performance of some networks [HL16].

The MFCCs and the corresponding delta features were extracted from the generated reverberant signals using the algorithm described in Section 2.1.2.4.

The corresponding target data for each input consisted of sets of two-dimensional points where the dimensions were the negative log amplitude and the time of arrival of the early reflections in the impulse response function. The negative logarithm was taken to keep the span of the amplitude positive with increased signal to noise ratio. This *peak data*, described in Section 4.1.1.2, is convertible into the full synthetic impulse response function represented in the time-domain. An example of peak data is shown in Figure 4.1 and an example of the extracted MFCCs is shown in Figure 4.2.

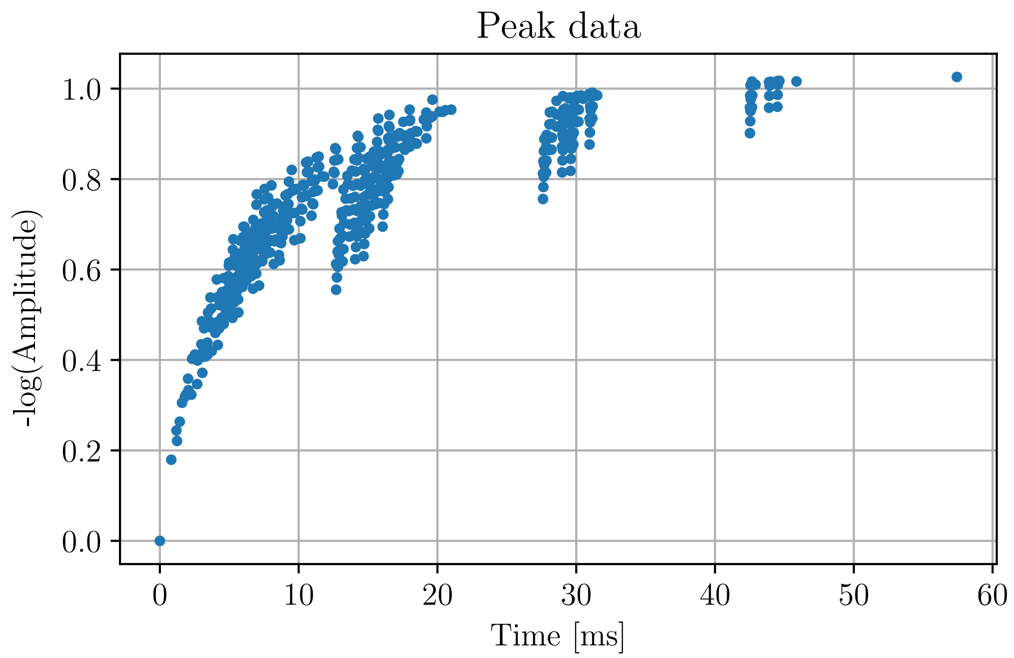


Figure 4.1: The figure shows the corresponding peak data retrieved from the impulse response function in Figure 3.1b according to Section 4.1.1.2.

4.1.1.1 Input data processing

The input data was built using the short anechoic signals of the authors' voices as described in Section 3.1. The equally long signals were convolved with the impulse response functions obtained from simulation using the virtual rooms as described in Section 3.2, yielding reverberant signals of **different** lengths depending on the impulse response lengths. In order to keep the dimensions equal, which is required for the network architectures used, the signals were zero-padded i.e. extended with silence. Depending on model, the MFCCs and delta-features were either calculated for the signals as described in Section 2.1.2.4, or the signals were left unaltered to be used directly for the training phase.

4.1.1.2 Target data processing

The target data generated using the image-source model and the room geometries as described in Section 3.2 was composed of sampled timings of delta function incidence, $\boldsymbol{t}$, and the amplitudes of which, $\boldsymbol{\alpha}$. The vectors in the database were rescaled for

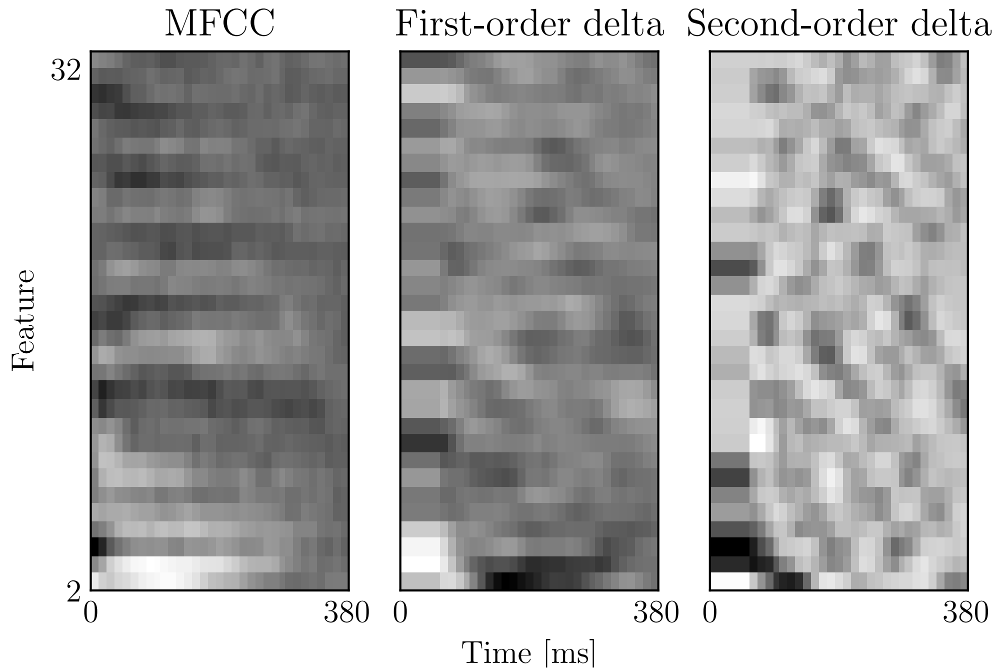


Figure 4.2: The figure shows an MFCC matrix with corresponding first- and second order delta features as well as an RGB representation; combining the three. The RGB image is rescaled such that all channels are normalized which is not the case during training or inference but only included here to highlight general dependence between channels.

the training process such that both timings and amplitude approximately spanned a square unit area in output space. Here follows the procedure used to normalize peak data.

Given the sample rate, f_s, the maximum number of reflections used during simulation, n, and approximate room dimensions, a typical impulse response time in sample space can be calculated. Assuming simple rooms with mean free path, l, the mean path length of late receivers becomes $l(n+1)$. Using the speed of sound, c_s, a mean impulse response length can be calculated as $t_{avg,max} = f_s l(n+1)/c_s$. Assuming that the direct signal path length is l, the time of incidence can be approximated using $t_{free} = f_s l/c_s$. Thus, given that the timing vectors span approximately $\boldsymbol{t} \in [t_{free}, t_{avg,max}]$ they can be normalized accordingly. The amplitude vector spans $\boldsymbol{\alpha} \in [\epsilon, 1]$ where ϵ is a small number that depends on room size and absorption coefficients. Since $\boldsymbol{\alpha}$ tends to ϵ quickly with increasing time, the logarithm is taken giving $-\log(\boldsymbol{\alpha}) \in [0, -\log(\epsilon)]$. Ignoring signal decay due to air resistance gives that $\epsilon \approx a^n$ where a is the mean absorption coefficient of the system boundaries in the simulation. $\boldsymbol{\alpha}$ can then be normalized such that $\boldsymbol{\alpha}_{normalized} = -\log(\boldsymbol{\alpha})/\log a$. The resulting vectors $\boldsymbol{t}_{normalized}$ and $\boldsymbol{\alpha}_{normalized}$ are what are referred to in this thesis work as *peak data* and represents impulse response functions. An example of peak data is shown in Figure 4.1. Since it is not of utmost importance for network performance that each sample spans *exactly* unit areas, the entire databases were normalized in bulk and the process was thus easily reversible for recovery of the full simulated impulse response functions as explained in Section 4.1.2.2.

4.1.2 Post-processing

This section explains the operations made to the network output in order to obtain refined results. Peak data representations as used here are highly simplified compared to simulated impulse response functions and even more so compared to completely realistic counterparts.

4.1.2.1 Extrapolating output data

In [Ahr19], Ahrens proposed that the human ear is insensitive to changes in the late part of the impulse response function, which can therefor be synthesized from random data without yielding a perceivable difference. Therefor, the dimension of the impulse response data passing the autoencoder was kept at low 256 x 2 data points. For some situations, primarily involving highly reverberant rooms, this number of peaks was not enough to construct naturally sounding reverberant signals due to that abrupt endings of the signal decays became audible. The impulse responses generating abrupt ending reverberant signals were therefor extrapolated to extend the late decay, see Figure 4.3. An extrapolation algorithm was constructed for this, that finds coefficients for $\boldsymbol{\alpha}'(\boldsymbol{t}') = a\log(b(\boldsymbol{t}' + c))d$ where $\boldsymbol{t}'$ and $\boldsymbol{\alpha}'$ are obtained from the network output, approximating the timings and amplitudes of the delta functions in the impulse response function. The algorithm treats the approximated function as the mean in a Gaussian random variable with probability distribution such that $\mathcal{N}(x, y|\mu_x = \boldsymbol{t}', \mu_y = \boldsymbol{\alpha}'(\boldsymbol{t}'), \sigma_x^2 = 0, \sigma_y^2 = c)$. The variance along the y-axis, c, is here assumed to be constant and equal to the variance of the late part of the network output.

4.1.2.2 Reconstructing impulse response functions from peak data

Since the network architecture outputs floating point-valued times for the incident contributions to the impulse response, and those times have a very low probability of coinciding with any sampled index, some care had to be taken in the reconstruction of the sampled impulse response functions. If this was not done, the impulse response functions, when used to create reverberant signals, introduce non-natural sounding artefacts in the resulting signals.

A solution to this problem was proposed by Peterson [Pet86]. They suggest placing Hann-windowed sinc-functions that depend on the distance between the times suggested by the network and the nearest sample times, at each position, according to

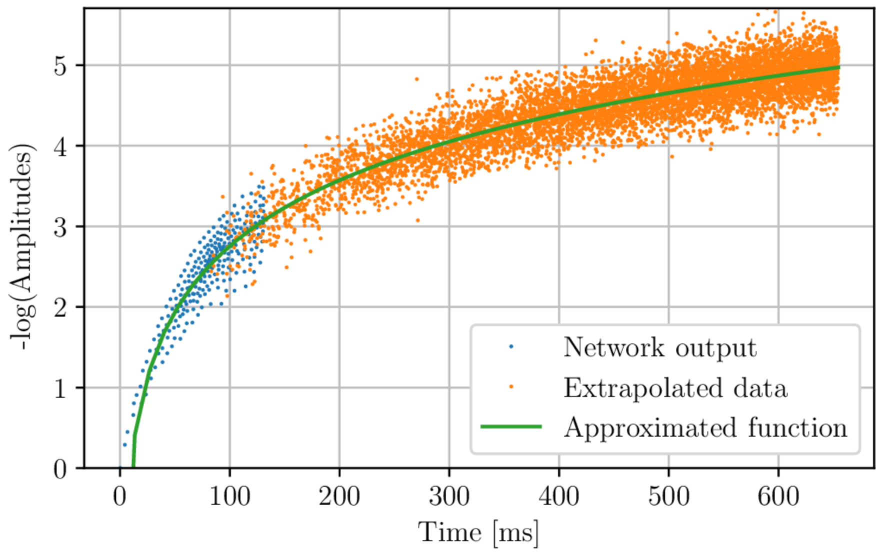


Figure 4.3: The figure shows the extrapolation process of the network output; depicted as blue dots. The fitted curve, depicted as a green line, was used as mean in a Gaussian random variable that varies in the y-direction; generating the extrapolation depicted as orange dots.

$$\delta_{LP}(n-\epsilon) = \begin{cases} \frac{1}{2}\left(1+\cos\left(\frac{2\pi(n-\epsilon)}{N_\omega}\right)\right)\operatorname{sinc}(2\pi f_c(n-\epsilon)), & \text{if} \quad -\frac{N_\omega}{2} \leq n \leq \frac{N_\omega}{2}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

where N_ω is the window size, f_c is the normalized cutoff frequency and ϵ is the distance to the nearest sample point. The equation describes the function that, when multiplied with the corresponding network output amplitudes and shifted according to the network output timings, generate the full impulse response.

4.2 Networks

Several different models were constructed. They share some fundamental properties related to the forward pass which is briefly explained in this section. The first part of each model, here referred to as the *extractor*, governs interpretation of sound signals. An extractor takes reverberant signals as input, either as time series or as MFC coefficients depending on architecture, and outputs compact, *latent*, representations stored as vectors of between 4 and 64 floating-point values, also depending on architecture. These latent representations are then passed to a shallow, fully connected network referred to as the *decoder*. Here, a decoder takes latent representations as input and generates point data outputs, that can be processed to impulse response functions, using 512 values. This type of feed-forward network structure was implemented in a few variations, all of which are explained in detail in the following sections. The extractors and decoders in this work were either trained together as single networks as visualized in Figure 4.10, or trained in two steps, as explained in Section 4.2.2 and visualized in Figure 4.6. The two-step implementations revolved around the use of autoencoders and these models are referred to as *extractor-autoencoder type networks*, further described in Section 4.2.2.

4.2.1 Extractor networks

The early part of the models, referred to as *extractors*, were implemented as feed-forward convolutional neural networks. All implemented extractors map reverberant sound signal inputs to latent representations of impulse response functions. The extractor networks described here were either trained using time-series directly, or using a two-dimensional representation with three channels. In the latter case the channels were the MFC coefficients and their first and second order delta features as described in Sections 4.1.1 and 4.1.1.1. An example of such data can be seen in Figure 4.2.

4.2.1.1 MFCC extractor networks

The MFCC extractor networks are composed of an increasing number of features and two-dimensional convolutional layers of kernel size $k = [3 \times 3]$. Delinearization was achieved by adding ReLU activation functions after each convolutional layer. Regularization possibilities were increased by the use of dropout layers. Down-sampling was implemented using max-pooling along two dimensions. A visualization of an MFCC extractor network can be seen in Figure 4.4

4.2.1.2 Time-series extractor networks

The time-series extractor networks were implemented using an increasing number of features and one-dimensional convolutional layers using kernel size one. Delinearization was achieved by adding ReLU activation functions after each convolutional layer. Regularization possibilities was increased using batch normalization. Down-sampling was realized by application of max-pooling along one dimension. A visualization of a time-series extractor network can be seen in Figure 4.5

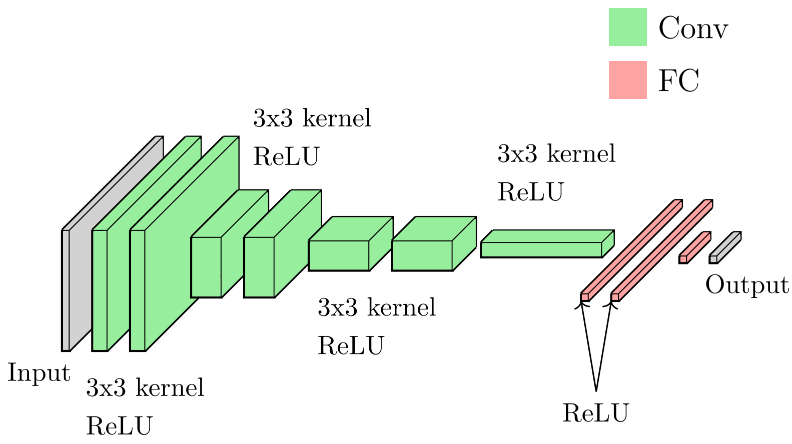


Figure 4.4: The figure shows the structure of the MFCC extractor networks. The input to the network is as described in Section 4.1.1.1. The convolutional layers are all followed by ReLU activation functions. Sub-sampling is performed by max-pooling which reduces the size of the data by a factor of 4. During training, dropout was performed after the pooling in order to improve regularization. After the final convolutional layer, average-pooling is performed which reduces the spatial dimension of the data to $[1 \times 1]$ thus preserving only the feature dimension. The first two fully connected layers are followed by ReLU activation functions. The output size of the final fully connected layer is governed by the bottleneck for the network in question. Further details about the MFCC extractor architecture is shown in Table A.5.

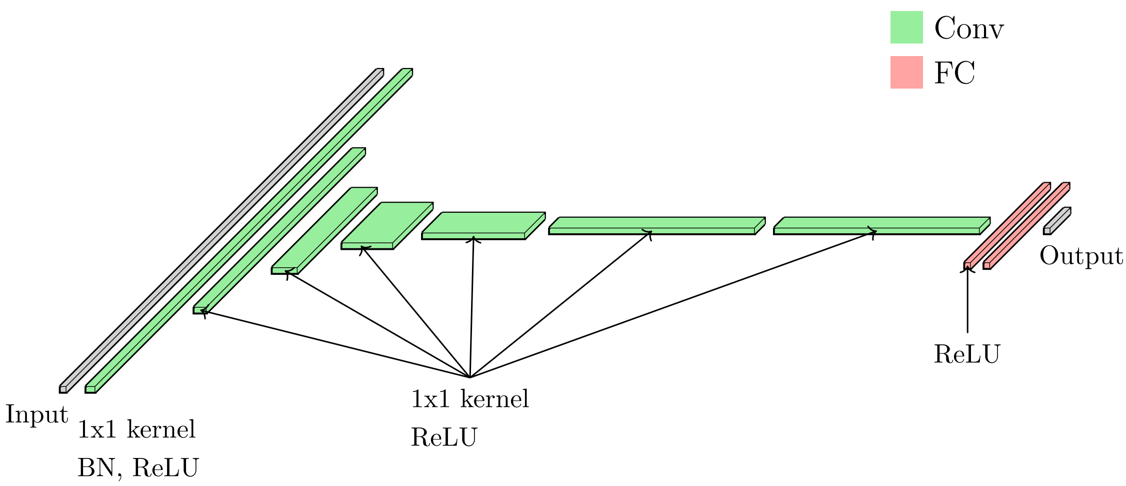


Figure 4.5: The figure shows the structure of a Time-series extractor network. The input to this kind of network is a reverberant sound signal, normalized between -1 and 1. The network is composed of a number of one-by-one convolutional layers with an increasing number of features. The data is down-sampled after each convolution by applying max-pooling in one dimension. After the convolutional layers there are two fully connected layers with an output size governed by the bottleneck for the network in question. Further details about the network architecture can be found in Table A.6.

4.2.2 Training with an autoencoder

To improve generalization and force networks to output data that resembles impulse response functions, autoencoders were implemented and trained as an initial step for some architectures. An ideal autoencoder takes some data as input and outputs an exact copy after having compressed and decompressed the data through a bottleneck. The reason this extra network was trained was in order to find a smaller representation of the data that described its important features well, such that the original data could be reconstructed from minimal features. The smallest representation of the data lies in the bottleneck layer, often referred to as the *latent space*. Autoencoders are seldom perfect and typically only works well for a certain set, or class, of data but could therefor also be exploited to guarantee proximity to a certain type of output regardless of what data is injected in the latent space.

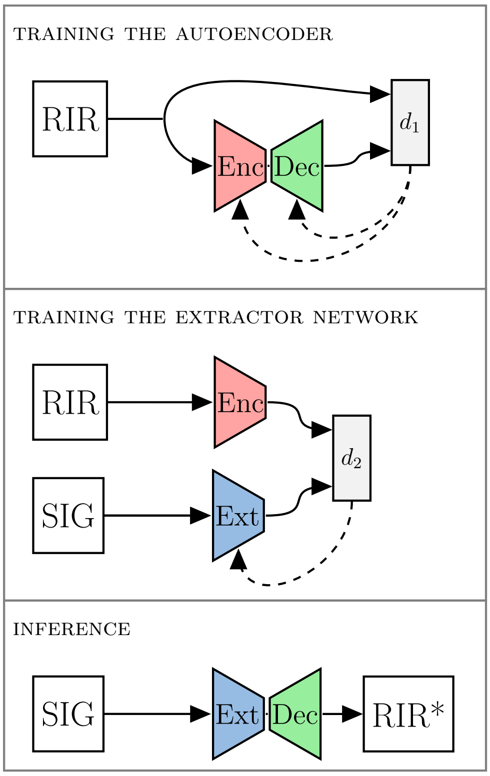


Figure 4.6: The figure shows the training scheme used to build the extractor-autoencoder pipeline and the order and combination of networks to be used for inference. The first pane shows conceptualized training of the autoencoder whose loss d_1 is minimized for *input* = *output*. The autoencoder is trained using impulse response data. The second pane shows training of the feature extractor network, depicted as Ext. The loss d_2 is minimized for agreement in the abstract latent space between target impulse response data and reverberant signal MFCCs or time-series. The third pane shows a forward pass with the combined extractor and decoder networks. For a fully optimized network, the approximated output RIR* is a perfect replication of the target RIR. The encoder network is not needed for inference.

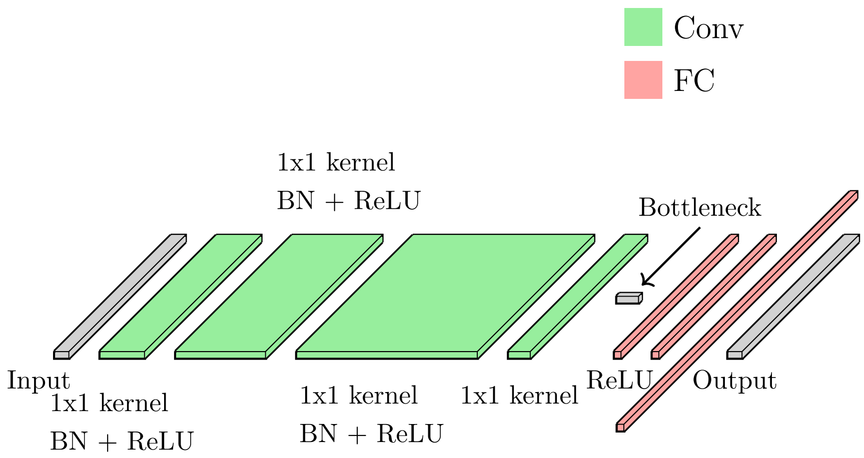


Figure 4.7: The figure shows the structure of an autoencoder network. The input to this network is a set of two-dimensional points. The early part of the network, up until and including the bottleneck, is the *encoder* part of the network, and the remaining constituents are referred to as the *decoder* part. It is composed of one-dimensional convolutional layers with kernel size one and an increasing number of features, encoding each point independently as described in Section 2.2.8. Each layer, except for the final one, is followed by batch normalization and ReLU activation functions. The size of the bottleneck governs the amount of data each IRF is encoded into. The late part of the network, after the bottleneck, is the decoder, a set of fully connected layers with ReLU activation functions. Further details about the autoencoder network structure can be found in Table A.4.

The autoencoders used during this thesis work were built using an encoder part and a decoder part. The encoder takes two-dimensional point clouds consisting of 256 two-dimensional points as input, **representing impulse response functions**, and encodes them into compressed representations of between 4 and 64 floating-point values. The decoder then up-samples the latent representations such that its dimensions equals the input data. The networks were trained using the distance between input and output, measured using the Chamfer distance as explained in Section 2.2.2.2, and weights were updated using the Adam algorithm. Due to that the data point order does not matter during Chamfer distance calculation it can not be guaranteed that the ordering of the points on either side of the autoencoder becomes equal. This is not an issue since the points can be sorted time wise prior to converting the data to time-series.

Having trained the autoencoder, a second training phase can be initialized. This phase consists of training an *extractor* (see Section 4.2.1) to take sound data as input and output a compressed representation in the latent space prior to the decoder. The encoded sound data as approximated by the extractor can then be decoded by the pre-trained decoder into peak data representations of impulse response functions. To optimize the extractor network, the losses were calculated in the latent space and weights were updated using the Adam algorithm. The losses were calculated as the MSE loss as described in Section 2.2.2.1.

4.2.2.1 Encoder

In order to avoid the problems that arise when training using data points that are not spatially ordered, as mentioned in Section 4.2.2.3, an implementation similar to PointNet [Qi+16] was developed for the encoder. The method revolves around encoding points individually and independently by the use of convolutional layers of kernel size one and an increasing number of features. A down-sampled and joint representation was achieved by adding max-pooling layers. This type of network structure was used to transfer input data to latent space in the autoencoders, see Section 4.2.2. A visualization of an encoder is shown in Figure 4.7 and consists of all layers to the left of, and including, the bottleneck.

4.2.2.2 Decoder

The late part of the models is referred to as the *decoder*. Here, decoders are shallow networks composed of a few fully-connected layers with ReLU activation functions after each layer except for the output layer. Decoders were either trained together with extractors as in Section 4.2.3, or as a part of another network called an autoencoder to be explained in Section 4.2.2. A visualization of a decoder is shown in Figure 4.7 and consists of all layers to the right of the bottleneck.

4.2.2.3 Motivating the use of one-by-one convolutions

Given data generated as explained in Section 4.1.1.2, different kinds of ordering of the points were possible. Predictive ordering is important for pattern emergence which in turn is crucial for most machine learning methods that involves larger than one-by-one convolutional layers. The first ordering that comes to mind might be that of the reflection order in the simulation model. For constant absorption the peak data was highly regular and piece-wise monotonous using this sorting which would lend predictability to a neural model. For varying absorption, however, the data became irregular, see Figure 4.8. Since many real systems contain surfaces made up of different materials each having a different absorption coefficient, low absorption variance can not be guaranteed for realistic rooms. This led to that another solution was introduced and the approach used in this project involves networks using one-by-one convolutions.

4.2.2.4 Data augmentation to combat complications during training

Random weight initialization in deep networks in the range of $[-a, a]$ around the origin results in that the output of the first forward pass of normalized data follows a Gaussian normal distribution with a variance $\sigma^2 < a$. Measuring the Chamfer loss between an unsorted sample from said distribution and an unsorted pattern that

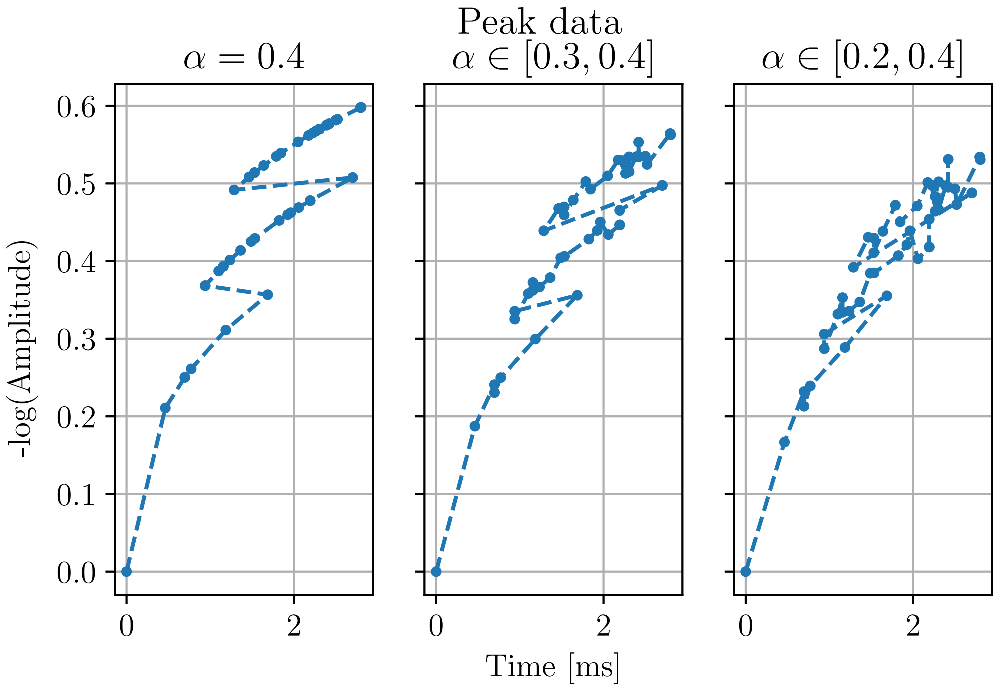


Figure 4.8: The figures show the impulse response function for a rectangular room for different ranges of absorption coefficients α. The figures show that for a constant absorption the pattern is regular and for a varying absorption the patterns are irregular.

spans a larger volume, but is guaranteed to have exactly one point at the origin typically resulted in that several points in the output data migrated towards the origin and became stationary. The explained situation arose during the training process of several network architectures that used the Chamfer loss together with an autoencoder but the problem was relieved by introduction of a random coordinate shift to the data associated with training the autoencoder such that it instead could be guaranteed that no point could be situated at the origin in neither input nor target. The coordinate shift could then gradually be removed once the points in the output data approximately spanned the volume of the target leaving very few points at the origin, better representing the patterns in the database.

4.2.3 End-to-end pipeline

End-to-end neural networks were constructed as single networks, taking MFCCs with delta features as input and returning two-dimensional point data to be compared to the peak data described in Section 4.1.1.2. Such networks were constructed for improved generalization using a bottleneck-scheme with a down-sampling MFCC extractor and an up-sampling decoder. A forward pass in this architecture is analogous to the forward pass in an extractor-autoencoder type network, as described in Section 4.2.2. A visualization of the network architecture is shown in Figure 4.9.

The training of an end-to-end network is visualized in Figure 4.10.

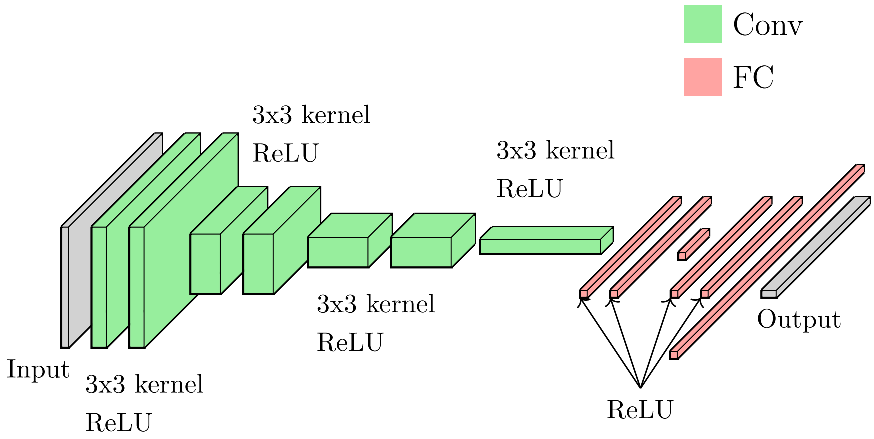


Figure 4.9: The figure shows the structure of an End-to-end network, consuming MFCC input data and outputting a peak data impulse response function directly. The convolutional layers are all followed by ReLU activation functions. Subsampling is performed by max-pooling which reduces the size of the data by a factor of 4. During training, dropout was performed after the pooling in order to improve regularization. After the final convolutional layer, average-pooling is performed which reduces the spatial dimension of the data to $[1 \times 1]$ thus preserving only the feature dimension. The first two fully connected layers are followed by ReLU activation functions. The output size of the final fully connected layer is governed by the bottleneck for the network in question. This part of the network, up until and including the bottleneck fully connected layer, is equivalent to the MFCC extractor network visualized in Figure 4.4. The final part of the network, including the last three fully connected layers where the first two are followed by ReLU activation functions, is equivalent to the decoder part of the autoencoder, visualized in Figure 4.7. Further details about the network architecture can be found in Table A.3.

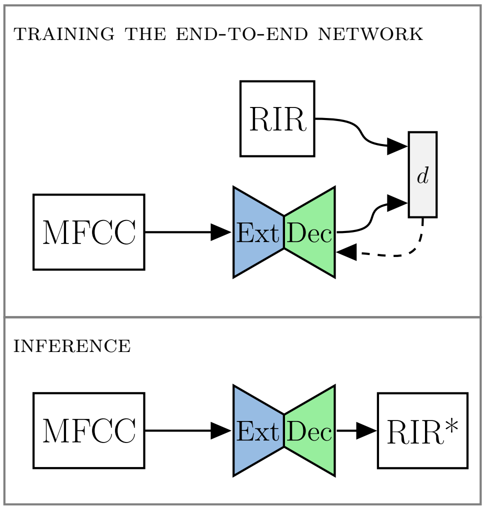


Figure 4.10: The figure shows the general scheme for training and inference of an end-to-end network pipeline. The first pane shows training of such end-to-end network, where the input, consisting of MFCCs and delta features, is passed through the network. The loss d is then measured as the Chamfer distance between the output and the target RIR. The second pane shows a forward pass with the end-to-end network. For a fully optimized network, the approximated output RIR* is a perfect replication of the target RIR.

4.3 Training results

In order to evaluate the performance of the different architectures associated with point-based representation, a database composed of 10^4 input-target pairs for training and 10^3 input-target pairs for evaluation was produced. The simulations used virtual rooms with random dimensions and absorption coefficients as presented in Table A.1. Table A.1 also presents various other parameter values associated with database generation.

4.3.1 End-to-end network

The end-to-end network was trained as described in Section 4.2.3 for 200 epochs. The lowest validation Chamfer loss achieved was 0.61, which is considerably lower than the Chamfer Distance metrics for the extractor-decoder networks presented in further down in Table 4.2. A sample of the network output can be seen in Figure 4.11, which shows the major problem regarding this network architecture; the overall shape resembles a room impulse response function *but* the large collection of data points situated in (0,0) would mean that the output signal becomes unrealistic. This is because clusters of points superpose in reassembly of the impulse response function. The proposed solution to this problem, as presented in Section 4.2.2.4, is not viable when training end-to-end network since that the decoder is not trained separately, hence the translation of the training target would be impossible to generalize.

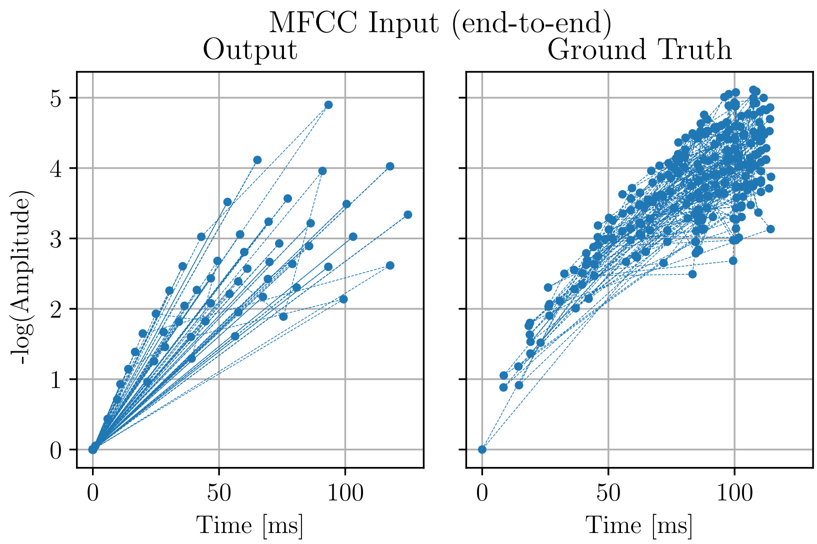


Figure 4.11: The figure shows a sample output from an end-to-end network. The network took MFCC data as input. The figure also shows the target associated with the specific sample. The points are connected with lines to highlight the fact that the network fails to produce realistic output since many of the points are situated in (0,0).

4.3.2 Extractor-autoencoder networks

The extractor-autoencoder models were trained in two separate steps as described in Section 4.2.2, results of which are presented in the following sections.

4.3.2.1 Autoencoders

Table 4.1 lists the results associated with the autoencoder networks. Figure 4.12 shows a sample output from an autoencoder with a bottleneck of 32, together with the target peak data.

The Chamfer distance as measured for the autoencoder network sets the limit for the performance of the extractor-autoencoder model. This is due to that the extractor is trained against the latent space and it will generally not be the case that the extractor produces something yielding a lower loss, in output space, than the target latent representation. The Chamfer distance metric for the extractor-autoencoders in Table 4.2 can thus be related to the Chamfer losses in Table 4.1 using this insight.

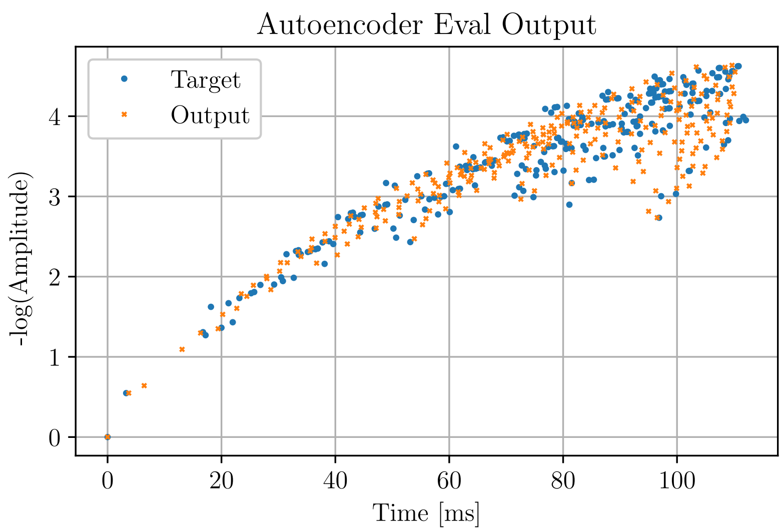


Figure 4.12: The figure shows a sample output from an autoencoder network associated with a bottleneck size of 32. It can be seen in the figure that the autoencoder represents the target well for this typical case, as it is able to produce details on different scales.

Table 4.1: The table shows the lowest obtained losses for different Autoencoders. The loss that was used to decide weight updates was the Chamfer distance. The loss presented here is the mean of the Chamfer distances between all input-target pairs for the entire training dataset as specified in Table A.1. The losses also represent the lowest achievable Chamfer distances for the extractor-autoencoder models presented in Table 4.2. For autoencoders, the loss is minimized for equality between input and output.

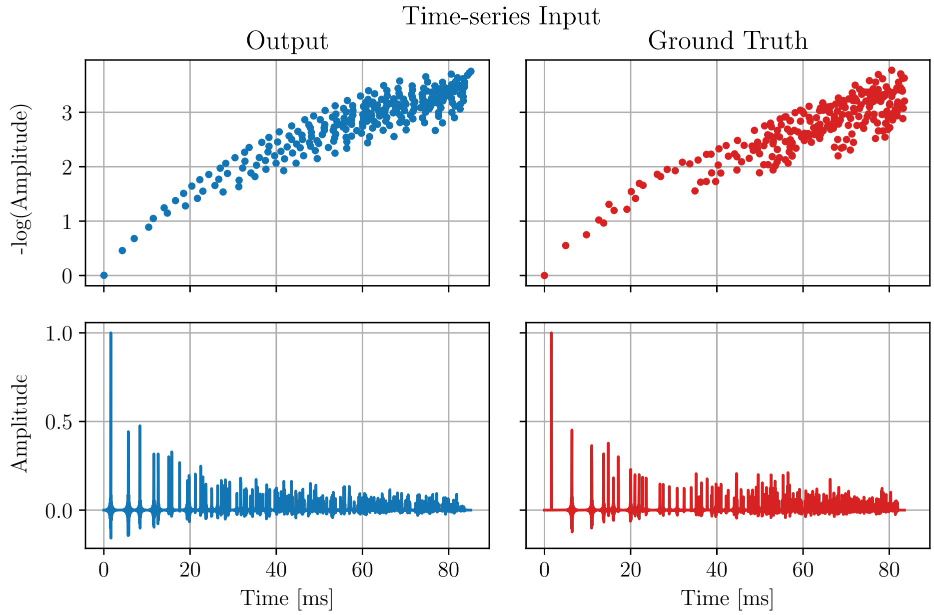
Autoencoder Validation Loss Results		
Network Architecture	Bottleneck	Chamfer loss
MFCC Input	4	0.08817
	8	0.07815
	16	0.06817
	32	0.06302
	64	**0.06030**

4.3.2.2 Extractors

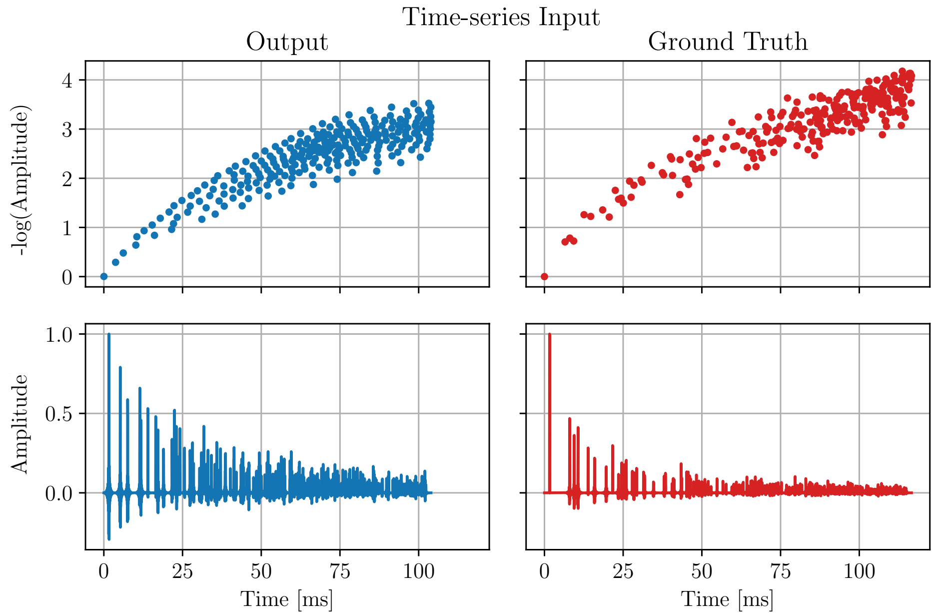
Table 4.2 lists the losses obtained from training different extractor-autoencoder type models. Figures 4.13 and 4.14 show samples of impulse response function peak data output for the different networks, both of which produce similar outputs which are likely to resemble impulse response functions. Ranging from the networks using the smallest latent space representation (a bottleneck of 4 units) to the networks using the largest latent space representation (a bottleneck of 64 units), the amount of details in the output increase.

Table 4.2: The table shows the lowest obtained losses for different combinations of extractors, autoencoders and input data formats. The loss that was used for the extractors was the MSE-loss in latent space. For the extractor-autoencoder networks, the loss is minimized latent space, between the extractor result and the latent target representation. The targets were impulse response functions as specified in Section 4.1.1.2. The presented distances were calculated as the mean of the Chamfer distance between all input-target pairs, **in output space**, for the entire dataset as specified in Table A.1. Note that, due to that the autoencoders are not ideal, a zero-loss MSE-loss does not correspond to vanishing Chamfer distance.

Extractor Validation Loss Results			
Network Architecture	Bottleneck	MSE loss	Chamfer Distance
MFCC Input	4	2.03	**1.19**
	8	2.13	1.36
	16	2.47	1.45
	32	2.35	**1.35**
	64	2.34	1.36
Time-series Input	16	1.62	1.23

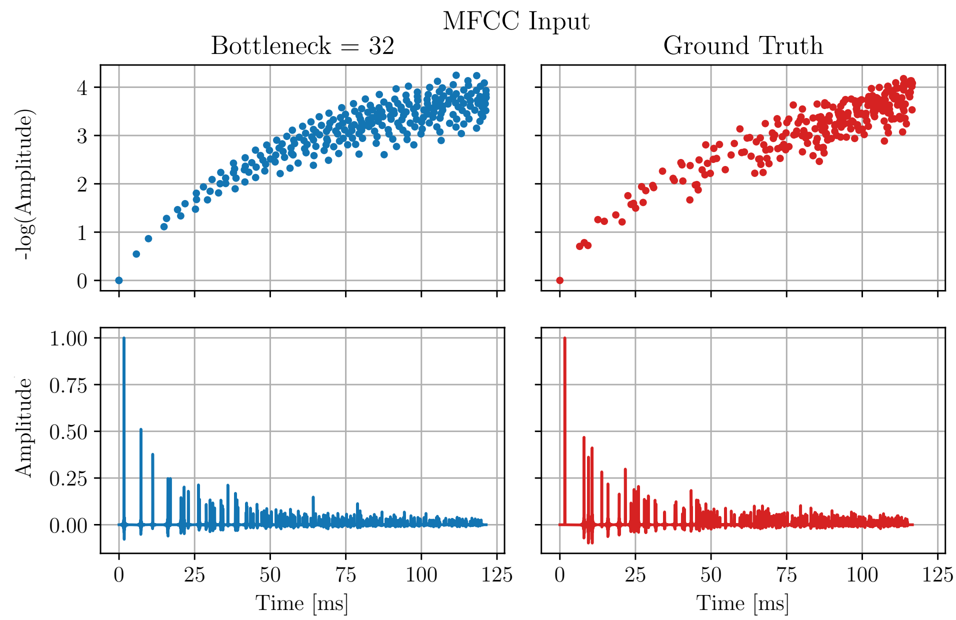


(a) Acceptable output and corresponding target.

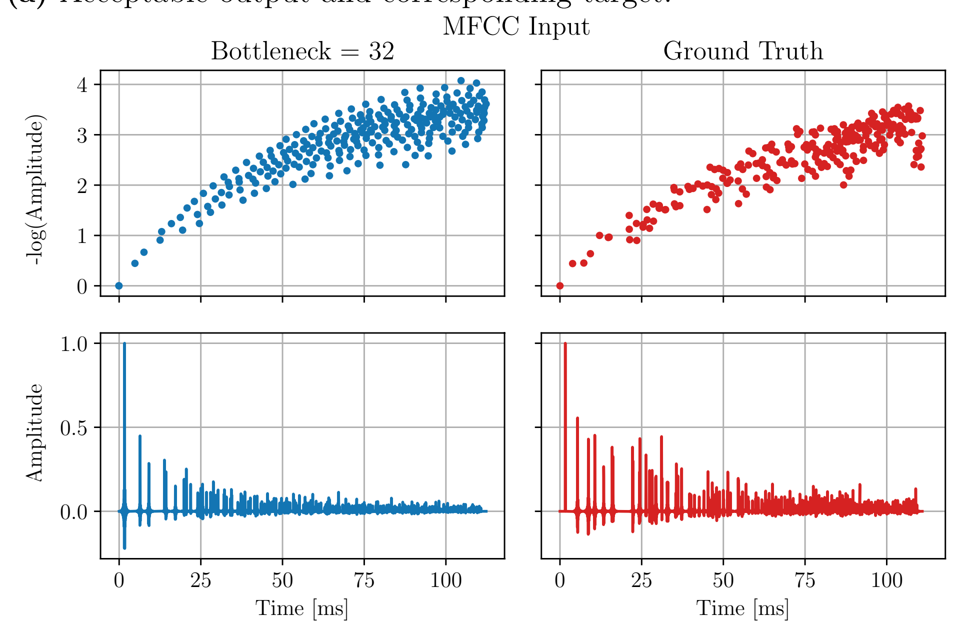


(b) Poor output and corresponding target

Figure 4.13: The figure shows sample outputs from an extractor-autoencoder network. The network took time-series as input data. The figure also shows the target associated with each output. In figure (a) the output represents the overall shape of the target well, but lacks some levels of detail whereas in figure (b) the network fails to produce output that overlaps with the target, especially if it is of too short duration. Also note that the points closest to the x-axis are the most important in terms of generating the full impulse response function since these corresponds to the highest amplitudes.



(a) Acceptable output and corresponding target.



(b) Poor output and corresponding target

Figure 4.14: The figure shows sample outputs from an extractor-autoencoder network with a bottleneck size of 32. The network took MFCC data as input. The figure also shows the target associated with each output. Analogous to the time-series case; note that in figure (a) the output represents the overall shape of the target well, but lacks some levels of detail whereas in figure (b) the network fails to produce output that overlaps with the target. Further, also note that the points closest to the x-axis are the most important in terms of generating the full impulse response function since these corresponds to the highest amplitudes.

5

Spectrogram-based representation

A framework was developed to approximate spectrograms of frequency dependent impulse response functions based on reverberant speech signals represented as spectrograms. This chapter describes the implementations of the elements essential to enable such framework. The following sections describe data and signal pre- and post-processing, two different network architectures, and present some of the results.

5.1 Data processing

This section describes the pre- and post-processing performed in order to enable training of deep neural models on the spectrogram-based data generated as described in Section 3.6 and visualized in Figure 3.5.

5.1.1 Data pre-processing

With the data being generated as the STFT spectrograms of some signals, some necessary steps were taken in order to prepare it for the network models. As described in Section 2.1.6, the phase information of a frequency spectrogram, if generated under certain prerequisites, can be reconstructed using only its magnitude. This fact means that the phase information of the training data can be discarded, effectively reducing the amount of data required to process by a factor of two. Thus, the absolute value of the complex-valued STFT spectrograms, for the input as well as the output data, were taken.

5.1.1.1 Input data pre-processing

To enhance some of the features in the magnitude spectrogram, the logarithm was taken. The pre-processing of the input data can be written as

$$\text{Input data} = \log(|\text{STFT}_{\text{Rev}}|). \tag{5.1}$$

5.1.1.2 Target data pre-processing

The first peak in the sought impulse response functions corresponds to the direct sound from the source to the receiver. This means that the amplitude of the first peak must be equal to one. Therefore, the magnitude spectrograms of the target data impulse response functions were normalized in the range $[0, 1]$ and the pre-processing of the target data can be written as

$$\text{Target data} = \frac{|\text{STFT}_{\text{IRF}}|}{\max(|\text{STFT}_{\text{IRF}}|)}. \tag{5.2}$$

5.1.2 Post-processing

To reconstruct a time-domain impulse response function from the network-generated STFT magnitude spectrograms, two steps must be taken. Firstly, the phase information must be reconstructed. This was done using the Griffin-Lim algorithm according to Algorithm 0 in Section 2.1.6. Secondly, with phase information present, the complex-numbered STFT spectrogram can be transformed to a time-domain signal using the inverse STFT.

5.2 Networks

In this section, two types of networks are presented that generate STFT magnitude spectrograms of the impulse response functions, based on the STFT magnitude spectrograms of reverberant sound (see Sections 3.6 and 5.1.1). This data is in a sense related to single-channel images, but with the important distinction that the dimensions of the data do not encode the same information, i.e. time and frequency. The implication of this is that the use of rectangular kernels can not be motivated in the same sense as it would have been using images as input data. Another important factor for the design of the networks is the fact that there is a clear correlation between the frequency bands of the input data and the output data.

The networks have in common that they are both fundamentally convolutional neural networks typically used for computer-vision tasks, but that they handle the dissociation of the dimensions in different ways.

5.2.1 Time convolutional downsampling-upsampling network

The time convolutional downsampling-upsampling network was inspired by the work of [PLS16] and typical generative neural models used for image data. It consists of two parts, where the first one encodes and subsamples the data into a small representation with many feature dimensions (a bottleneck) and the second one upsamples the data into a single-channel, two-dimensional representation to form the output. Throughout the network, care has been taken to preserve the correlation between the frequency sub-bands of the input and the output data, therefore most of the convolutional layers used in this network structure is comprised of one-dimensional kernels that are convolved with the spectrogram data over time, thus encoding each frequency sub-band independently. All pooling is also performed only in the time-domain. A visualization of the network structure is shown in Figure 5.1 and the particulars of each layer can be found in Table A.7.

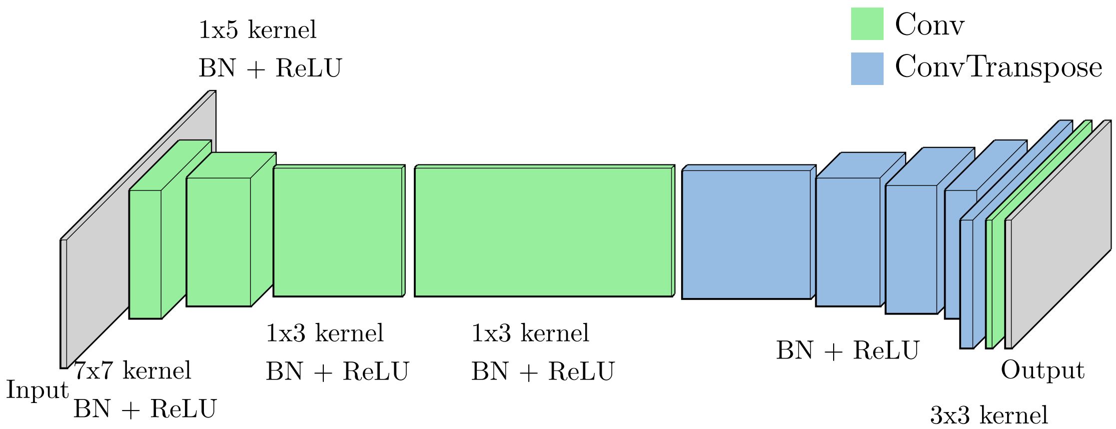


Figure 5.1: The figure shows the structure of the time convolutional upsampling downsampling network. The input to the network is the logarithm of the STFT spectrogram of reverberant signal, as described in Sections 3.6 and 5.1.1. Except for the first layer, the convolutional kernels are one-dimensional, operating along the time dimension. The convolutional layer are all followed by batch normalization and ReLU activation functions. Sub-sampling is performed by max-pooling over the time domain only - reducing time dimension in several steps until few elements remain. Up-sampling is performed for the time-axis using transposed convolutional layers. Further details about this architecture is shown in Table A.7.

5.2.2 Dimension-swapping convolutional networks

The dimension-swapping convolutional network structure is a continuation of the time convolutional downsampling-upsampling network but with the design goal of avoiding transposed convolutional layers. The use of transposed convolutions often comes with the cost of spatially regular artefacts in the output of such layers. Also, the size of the output data from a transposed convolutional layer is governed by the stride of the kernel and by padding the data, which can prove to be an inconvenient way to control the size of the data if some very specific size is sought. The size of the data within the dimension-swapping convolutional network is controlled solely by the number of kernels used in the convolutional layers, utilizing the fact that the number of kernels can be set arbitrarily, leading to that the shape of the data can be set arbitrarily. The process is done by transposing the data prior to a convolutional layer such that the feature dimension and the time dimension are swapped, and then transposing the output of the layer back to the original order.

The design of the dimension-swapping convolutional network structure can be motivated by inspection of the output data. An impulse response function can be regarded as the effect a room has on a delta function, thus, the problem at hand can be viewed as finding the sharpest transient in each frequency band and tracing it over time. But *where* this information is encoded in the input data is not necessarily the same for all frequency bands, and the density of this information is likely not to be constant over the entirety of the input data. The sub-sampling technique of pooling handles both of these issues bluntly as it essentially assumes that the information can be sampled at a constant rate over the data. Instead, allowing a learnable filter to select the most important time-samples arbitrarily from all of the samples assumes nothing of the information density. In order to design a network able to operate on the premises of the above statement; placing two-dimensional convolutional filters in the plane of frequency and features is one option. Designing a network like this permits an output time domain for each frequency where every sample is a nonlinear function of a linear combination of the entire input time domain.

It was found that the activation function used at the convolutional layer that change the length of the time dimension is of great importance for the performance of the network. The use of ReLU activation functions sometimes resulted in spectrograms equal to zero for all frequencies at certain times. This problem was relieved by the use of activation functions that only maps zero itself to zero, such as the hyperbolic tangent or leaky ReLU.

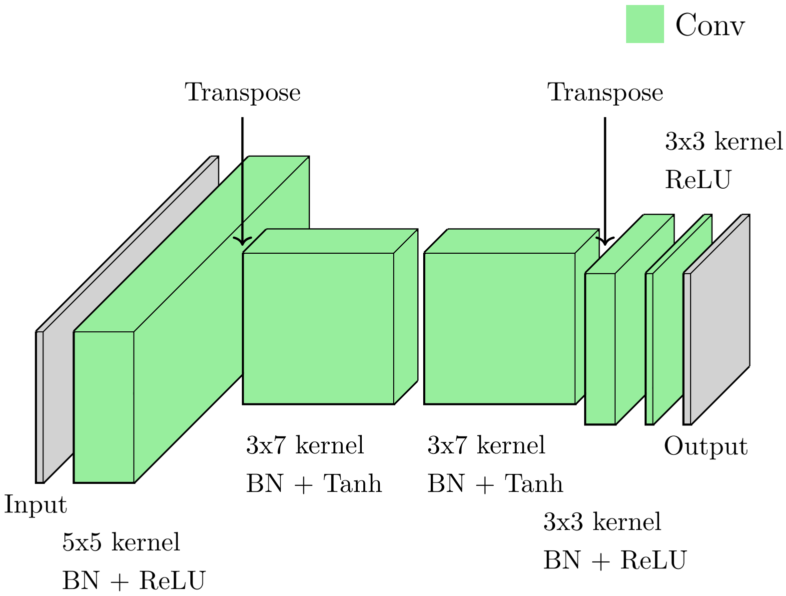


Figure 5.2: The figure shows the structure of the dimension-swapping convolutional network. The input to the network is the logarithm of the STFT spectrogram of a reverberant signal, as explained in Sections 3.6 and 5.1.1. The convolutional kernels are two-dimensional. The convolutional layers are followed by ReLU activation functions when convolutions are performed in the time-frequency plane, and by tanh activation functions when convolutions are performed in the frequency-feature plane. Sub-sampling is performed by changing the length of the time-axis in the transposed perspective. Further details about the dimension-swapping convolutional network architecture is shown in Table A.8.

5.3 Training results

In order to evaluate the performance of the different architectures associated with spectrogram-based representation, a database composed of 10^4 input-target pairs for training and 10^3 input-target pairs for evaluation was produced. The simulations used rectangular virtual rooms with random dimensions and measured, frequency dependent absorption coefficients as presented in Table A.1. Figures 5.3 through 5.8 shows network evaluation outputs from the frequency dependent models. The figures show that the overall shape of the training targets are captured and recreated by the network. The figures also show that the recreation is not perfect along the time-axis, in fact, some of the time-related fine details are lost. The network learns that the energy in all impulse response functions must decrease over time, the network sometimes can not reproduce cases where the impulse response function have gaps along the time-axis which is especially noticeable in Figure 5.4. Another problem with this model is that the initial peak in the impulse response function is not always of the same magnitude as the target which is important for good results due to that the scaling of reverberance in the output signal depends on the ratio between fist reflection and the rest of the impulse response function. The reason for this is failure to estimate the amplitude at the peak which only spans a very short time, the relative error might be approximately equal across time, but errors are most prominent if situated at the initial peak.

Table 5.1: The table shows the validation loss for the two networks based on spectrogram input and output, calculated as the mean MSE loss for a complete epoch of 10^3 samples.

Validation Loss Results	
Network Architecture	MSE loss
Time convolutional downsampling-upsampling network	$6.02 * 10^{-3}$
Dimension-swapping convolutional network	$5.17 * 10^{-3}$

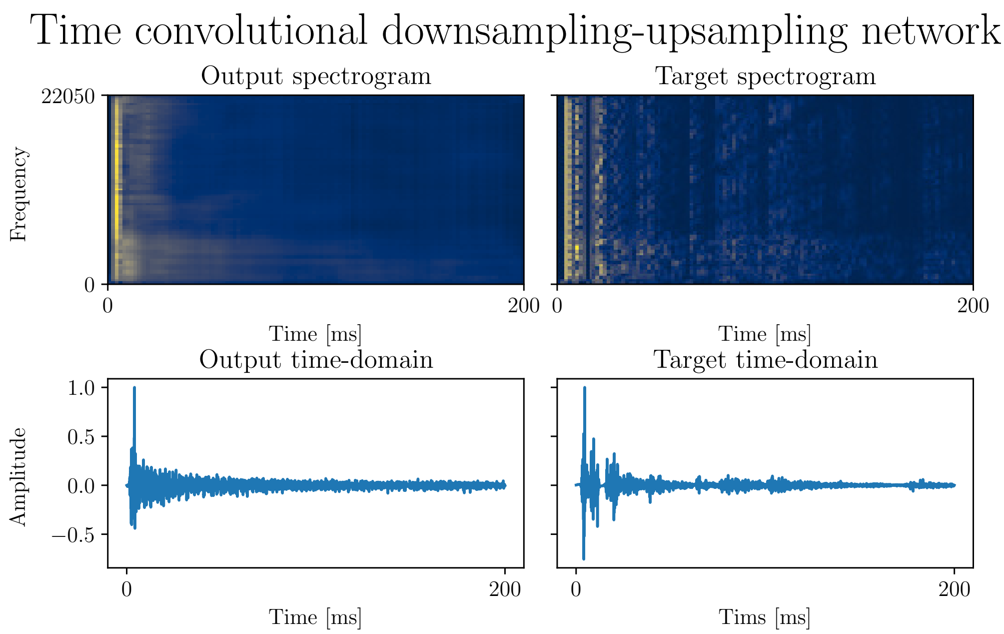


Figure 5.3: The figure shows network output from the time convolutional downsampling-upsampling network in both time and frequency-time domain. The figure shows that network is lacking the fine details of the impulse response function but that the output is still approximated within reason. The network output is representative and shows that the output amplitude is similar but smoother which is the typical behaviour of the network.

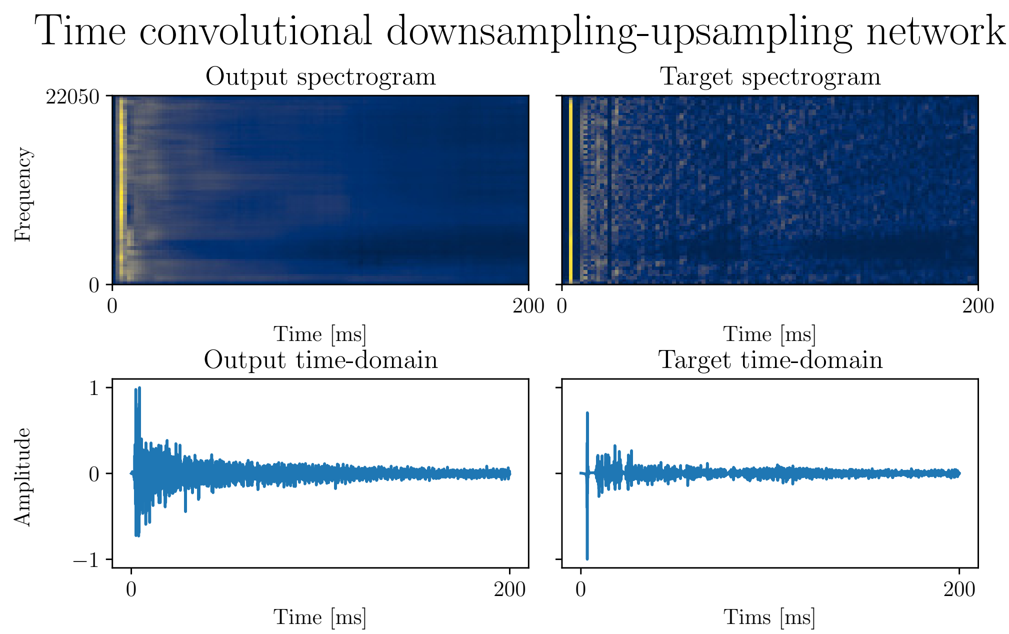


Figure 5.4: The figure shows network output from the time convolutional downsampling-upsampling network in both time and frequency-time domain. That the network often fails to produce details may sometimes alter the shape of the envelope of the time-domain impulse response to a large extent.

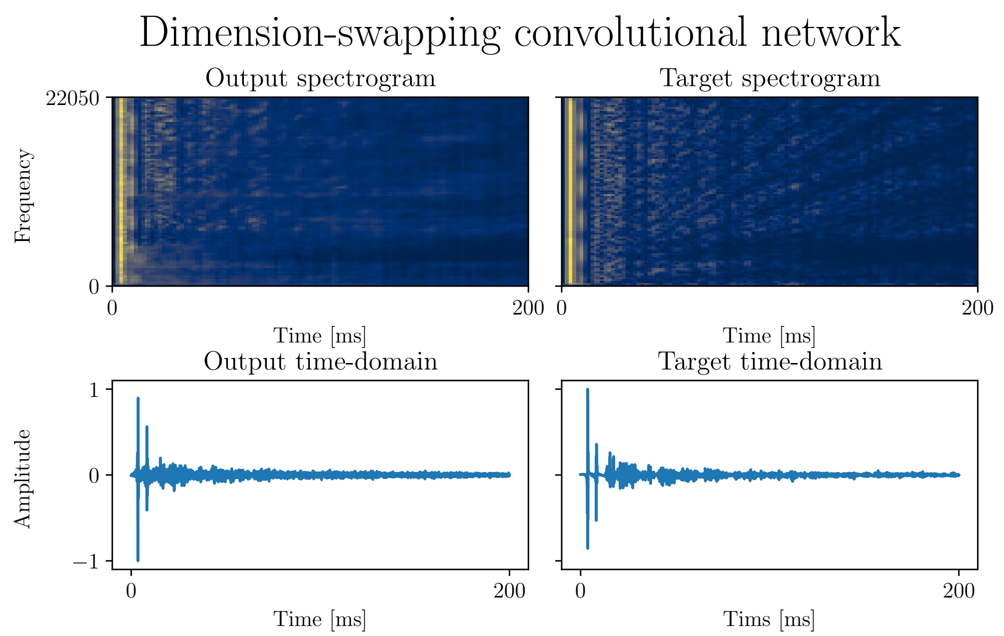


Figure 5.5: The figure shows network output from the dimension-swapping convolutional network in both time and frequency-time domain. The figure shows that the short impulse response function is approximated well, note the slower decay for low frequencies - represented in both target and output.

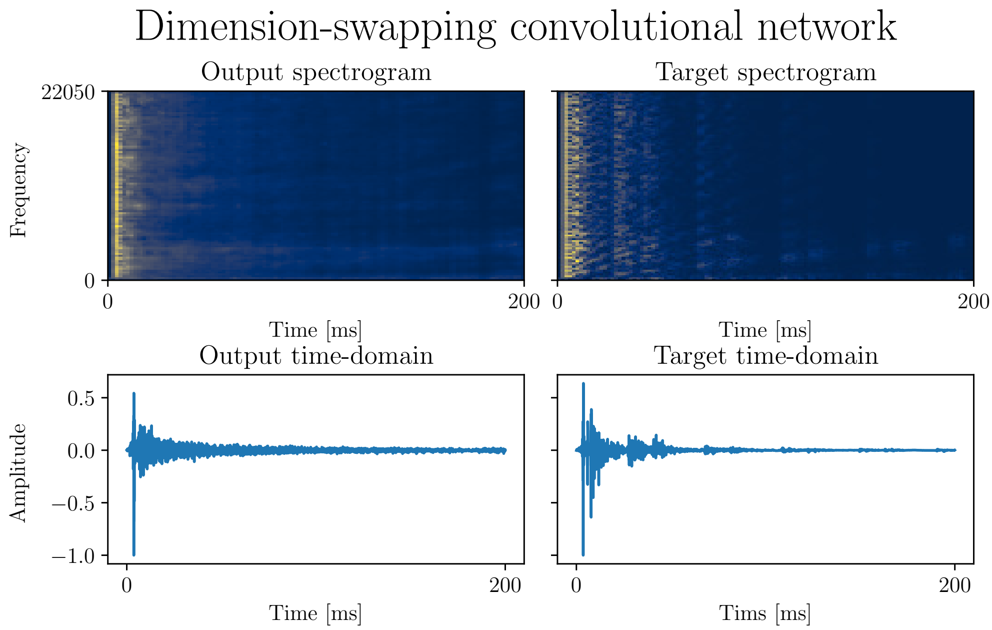


Figure 5.6: The figure shows network output from the dimension-swapping convolutional network in both time and frequency-time domain. The figure shows that the impulse response function is approximated within reason. The network output is representative and shows that the output amplitude is similar but smoother which is the typical behaviour of the network.

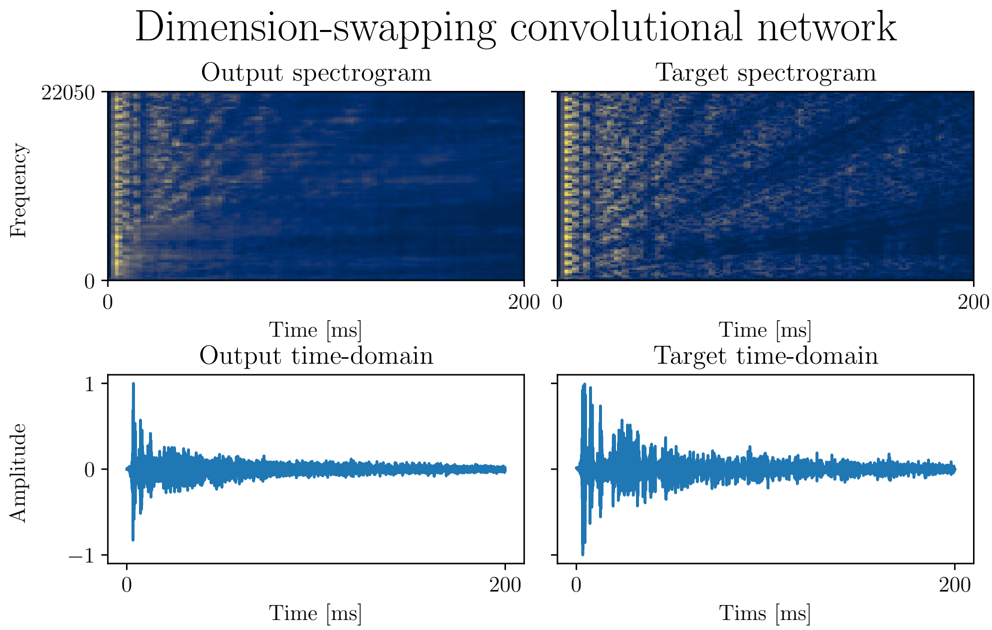


Figure 5.7: The figure shows network output from the dimension-swapping convolutional network in both time and frequency-time domain. The figure shows that the long impulse response function is approximated within reason.

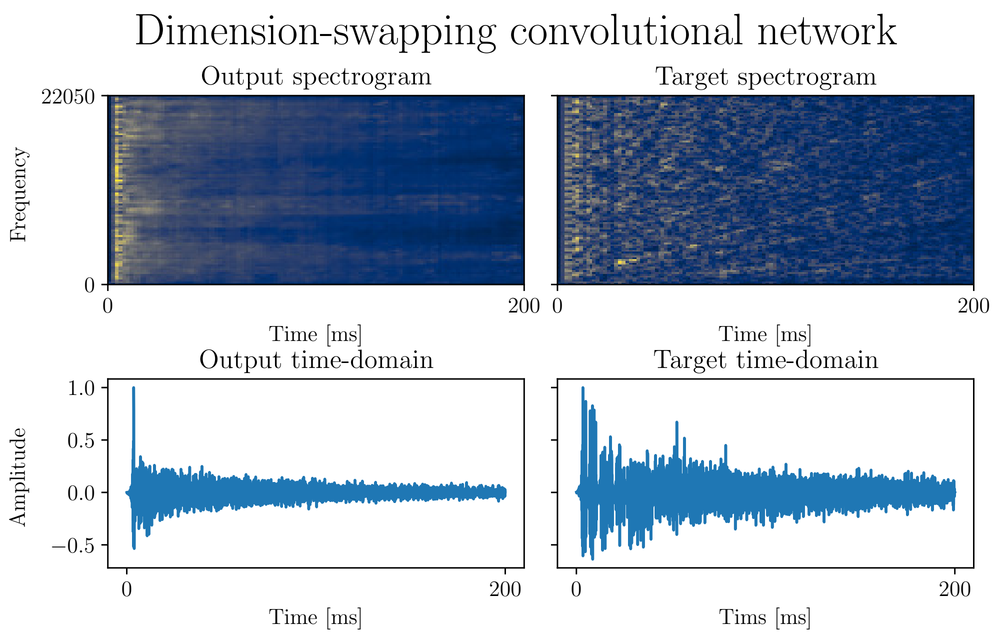


Figure 5.8: The figure shows network output from the dimension-swapping convolutional network in both time and frequency-time domain. The figure shows that the impulse response function is approximated poorly.

6

Comparing models and perceptual evaluation

Because of the differences in the data representation used for the network structures presented in Chapters 4 and 5, it is difficult to compare the models by numerical means. Also, the validation loss of any particular model does not necessarily represent *how* the deviation from a perfect model is perceived by a person listening to its output. For these reasons, the models were compared and evaluated perceptually, by conducting tests where the score was set by human listeners. Listening tests are a commonly produced metric of the performance of a synthetization within the realm of the acoustic sciences. Likewise, the human perception system can be regarded as the ultimate jury in many cases that aims for realistic perception and it should be appreciated as such in this specific case.

6.1 Comparing point-based and spectrogram-based models

Representative, high performing networks within each family of models were selected for comparison. These were the non-frequency dependent extractor-autoencoder network with MFCC input and the frequency dependent dimension-swapping convolutional networks. Since the networks accept different types of input data, outputs can not be produced based on the exact same simulations. But to make the outputs as comparable as possible, simulations were made for the same virtual room, selecting parameters carefully. For the frequency dependent network, the simulation was made for 60 orders of reflections and for seven frequency bands each corresponding to a set of absorption coefficients. For the non-frequency dependent network, the room was simulated for eight orders of reflections and the absorption coefficient was set corresponding to the frequency band that centers around 1000 Hz being approximately at the prominence of human speech.

During the test, the listeners were presented with 10 different situations such as the one in Figure 6.1. The listeners were asked to decide which of the two sounds B' and B'' had the most similar characteristics to the sound A', and set the slider to represent how certain they were of their decision. The test suggests higher performance of the model that is voted for the most. The test was devised as follows:

- Convolve signal A, drawn from the set U, with a simulated target impulse response function to obtain the reverberant signal A'.
- Convolve signal B, drawn from the set $U \setminus \{A\}$, with the frequency dependent network output that corresponds to the target impulse response function to obtain the reverberant signal B'.
- Convolve signal B with the frequency independent network output that corresponds to the target impulse response function to obtain the reverberant signal B''.
- Have a human listener decide whether he or she thinks that B' or B'' belongs to the same system as A'.

Figure 6.1: The figure shows the interface of the listening test conducted to test the performance of the frequency dependent and the non-frequency dependent models. The top audio player holds the reference signal A', the two bottom audio players holds the generated signals B' and B''.

The results from the test, shown in Figures 6.2 and 6.3, show clearly that output from the frequency dependent model emulate reference sounds with a higher degree of certainty. The authors also perceive output from the frequency dependent model as clearly more realistic. The mean of all answers was equal to 1.04, where a figure of 3 would correspond to that all listeners prefer the frequency dependent model, and a figure of -3 would correspond to that all listeners prefer the frequency independent model. The mean suggests that the average listener clearly favor the frequency dependent model. A metric called doubt was also calculated as $\text{doubt} = \text{mean}(1 - |\text{answers}|/3)$ where *answers* are in the range $[-3, 3]$. The doubt factor gives a measure in the range $[0, 1]$ on how uncertain the listeners were of their decision on average. A figure of 0 corresponds to the listeners being completely certain that they made a correct choice, and inversely a figure of 1 corresponds to complete uncertainty. For this test the doubt factor was 0.2, which suggests that the listeners were relatively certain of their decisions.

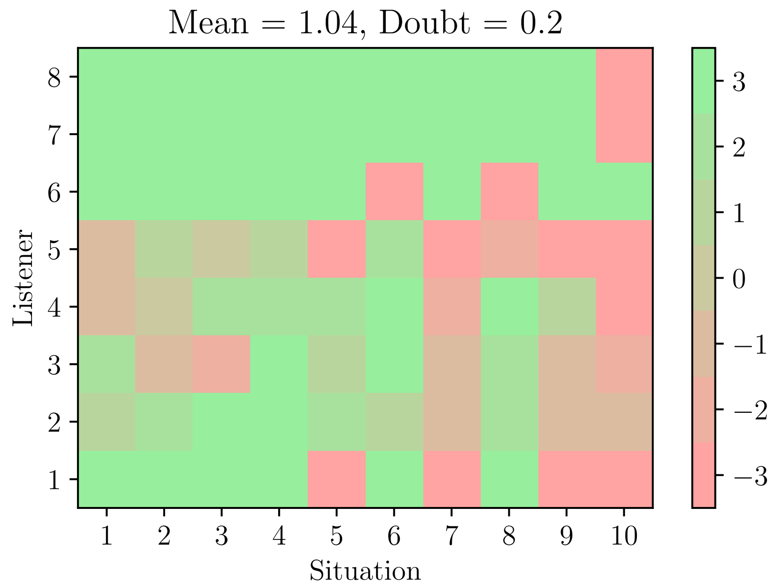


Figure 6.2: The figure shows the listening test result from the comparison test between the frequency dependent and independent models. Green and red squares represent cases where the listener choose the frequency dependent model output and frequency independent model output respectively. The mean of 1.04 suggests that the listeners perceive output from the frequency dependent model as more similar to the reference sound than output from the frequency independent model. The doubt factor of 0.2 suggests that the average listener is relatively certain of their choice. Situation 10 shows a very poor result for the frequency dependent model. For this situation, where the reverberation time happens to be short, both models fails to produce realistic output; producing long impulse response functions instead. This is likely due to that the drawn anechoic sound have some property that causes the reverberant input to seem more reverberant than it actually is.

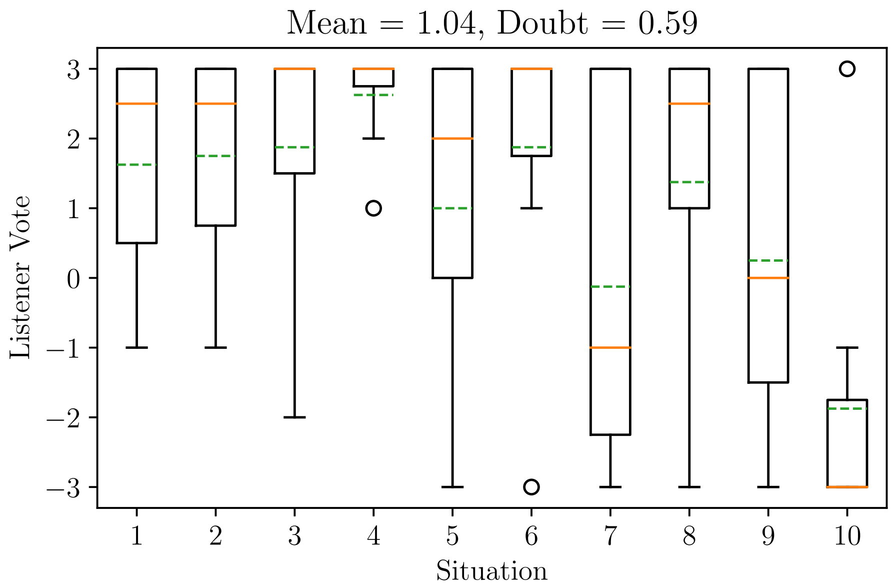


Figure 6.3: The figure shows compiled results in the form of a box plot of the data as presented in Figure 6.2. The dashed green line corresponds to the mean of the data and the solid red line corresponds to the median, the dots are outliers. The box plot shows that the listeners are inclined to choose output from the frequency dependent model instead of from the frequency independent model in the majority of cases.

6.2 Perceptual evaluation of the frequency dependent model

This test was performed to deduce if the output generated by the dimension-swapping convolutional network is convincing enough that a person believes it is real. The interface of the test presented to the listeners can be seen in figure 6.1. In the test, the listeners were asked to decide which of the two sounds B' and B'' had the most similar characteristics to the sound A', and set the slider to represent how certain they were of their decision. The test was constructed analogously to the comparison test in section 6.1, with the difference that the sound B'' was generated as an anechoic signal convolved with the target impulse response function.

The tests were conducted in two stages. In the first stage, the phases of the generated STFT spectrograms of the impulse response functions were randomized, the results of which can be seen in the bottom matrix of Figure 6.4 and in Figure 6.6. The second stage used the improved impulse response reconstruction where the phases were generated using the Griffin-Lim algorithm as explained in Section 2.1.6, the results of which can be seen in the top matrix of Figure 6.4 and in Figure 6.5.

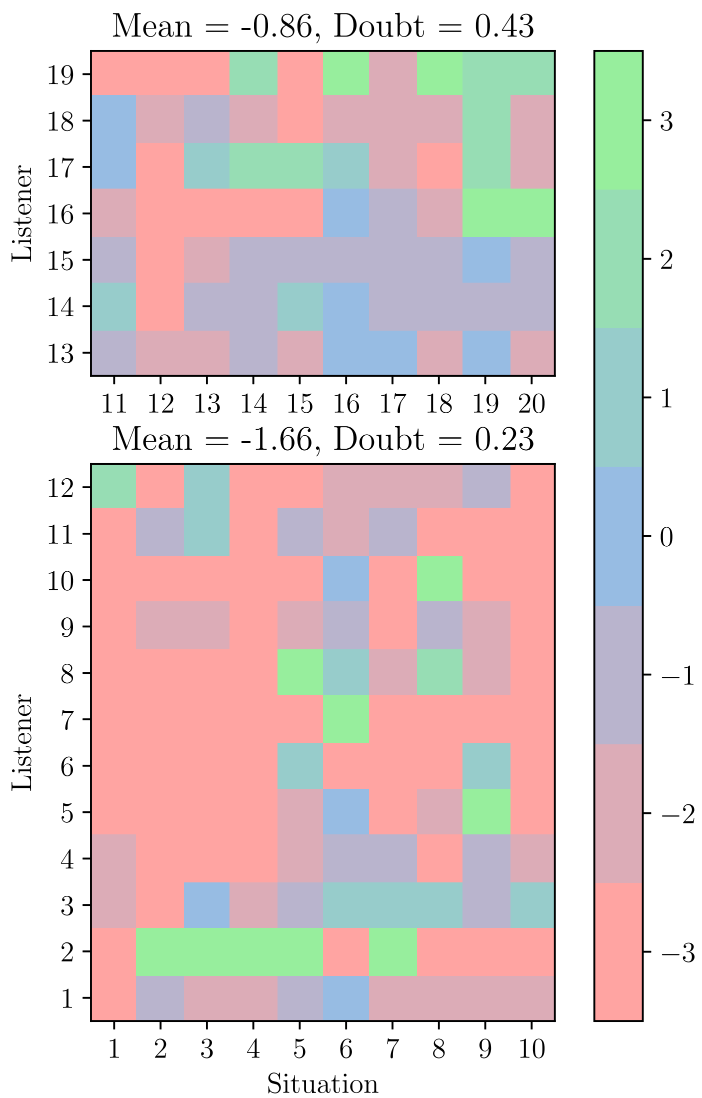


Figure 6.4: The figure shows compiled results stemming from the proximity test for a network of the dimension-swapping convolutional type. Green colored elements means that the listener preferred the network output to the simulated impulse response function. During the test, the listeners were presented with a slider which they were instructed to set according to the degree of certainty that they made the correct choice, as explained in Section 6.1. The figure shows test results from two phase generation variants. For the bottom matrix the impulse response function phases were drawn randomly and for the top matrix; phases were generated using the Griffin-Lim algorithm as explained in Section 2.1.6. The figure shows that test results improve when phases are generated using the algorithm as listeners were more inclined to state lower certainty.

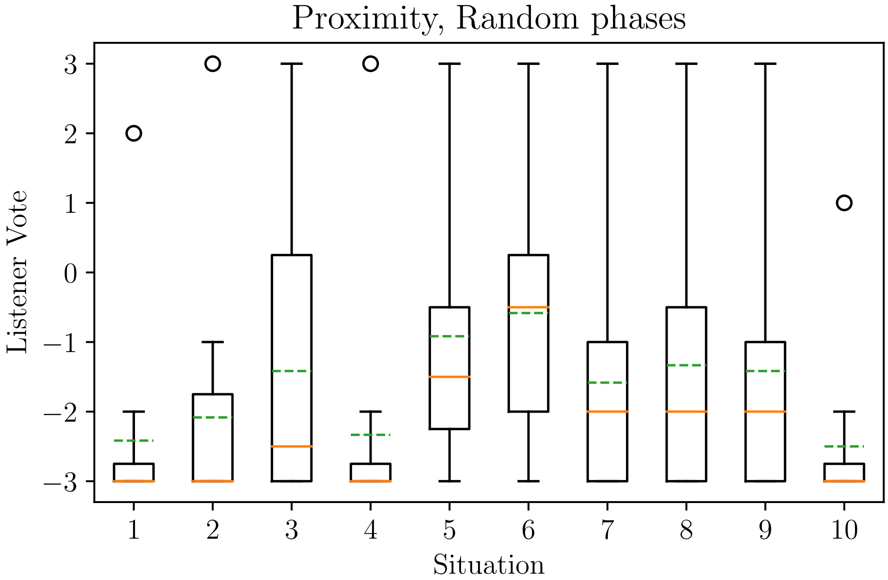


Figure 6.5: The figure shows compiled results in the form of a box plot of the data as presented in Figure 6.4, for the bottom matrix - corresponding to synthetization using random phases. The dashed green line corresponds to the mean of the data and the solid red line corresponds to the median, the dots are outliers. The box plot shows that the listeners are inclined to choose the simulated synthetization in most cases.

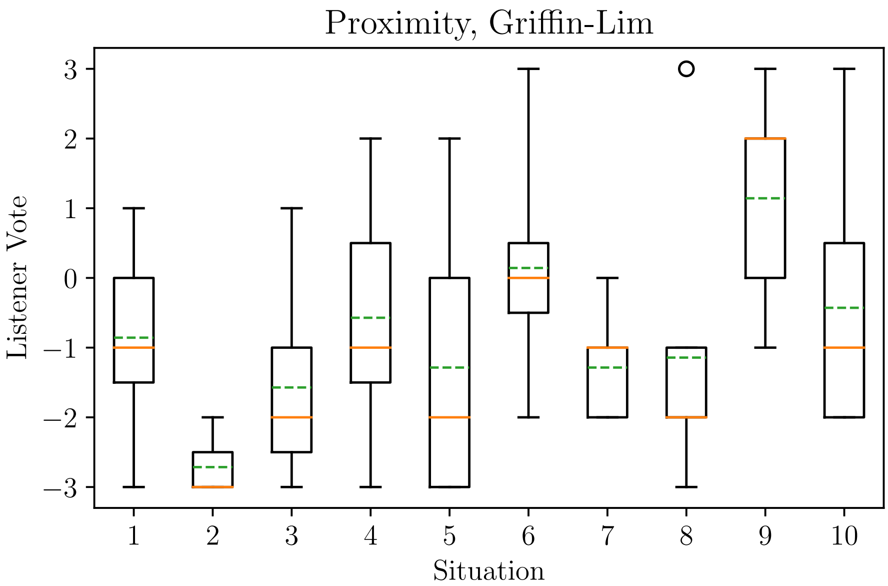


Figure 6.6: The figure shows compiled results in the form of a box plot of the data as presented in Figure 6.4, for the top matrix - corresponding to synthetization using phases generated using the Griffin-Lim algorithm as explained in Section 2.1.6. The dashed green line corresponds to the mean of the data and the solid red line corresponds to the median, the dots are outliers. The box plot shows that the listeners are inclined to choose the simulated synthetization in most cases.

As can be seen in Figure 6.4, for the dimension-swapping convolutional network with phases generated using the Griffin-Lim algorithm the mean was -0.86; an improvement from -1.66 that corresponds to output from the counterpart with random phases. The doubt factor also increased from 0.23 to 0.43, suggesting that the phases generated by the algorithm increase realism in the output. These results suggest that the output from the dimension-swapping convolutional network with phases generated using the Griffin-Lim algorithm sometimes produce reverberant signals that are perceived as realistic. One thing to note is that the test has a built-in assumption that the highest possible realism achievable using the framework is the realism of image-source model results. This is however debatable, discussed in Chapter 7.

6.3 Comparing model and reality

Given that the human perception of sound depends on other sensory input, such as vision [PK11], suggests that a test should be conducted that studies the psychoacoustic influence imposed by vision on the perception of the reverberant signal generated by the framework. The visual test studies whether synthetic sound signals generated using the network output corresponds well to the reverberance that listeners expect an acoustic room to generate when exposed to a signal and was devised as follows.

- Calculate a 'real' impulse response function using measurements inside an acoustic room and photograph the setup used that corresponds to the impulse response.
- Convolve signal A, drawn from the set U, with the **measured** target impulse response function to obtain the reverberant signal A'.
- Convolve signal A, with the network output that corresponds to the target impulse response function to obtain the reverberant signal A''.
- Have a human listener decide whether he or she thinks that A' or A'' belong to the image captured in the first step.

Figure 6.7 shows the design och the visual listening test and Figure 6.8 shows the result of the test. The *confusion* metric presented in the title for the result matrix denotes the number of answers that corresponds to network output divided with the total number of answers. The confusion was 31.6%.

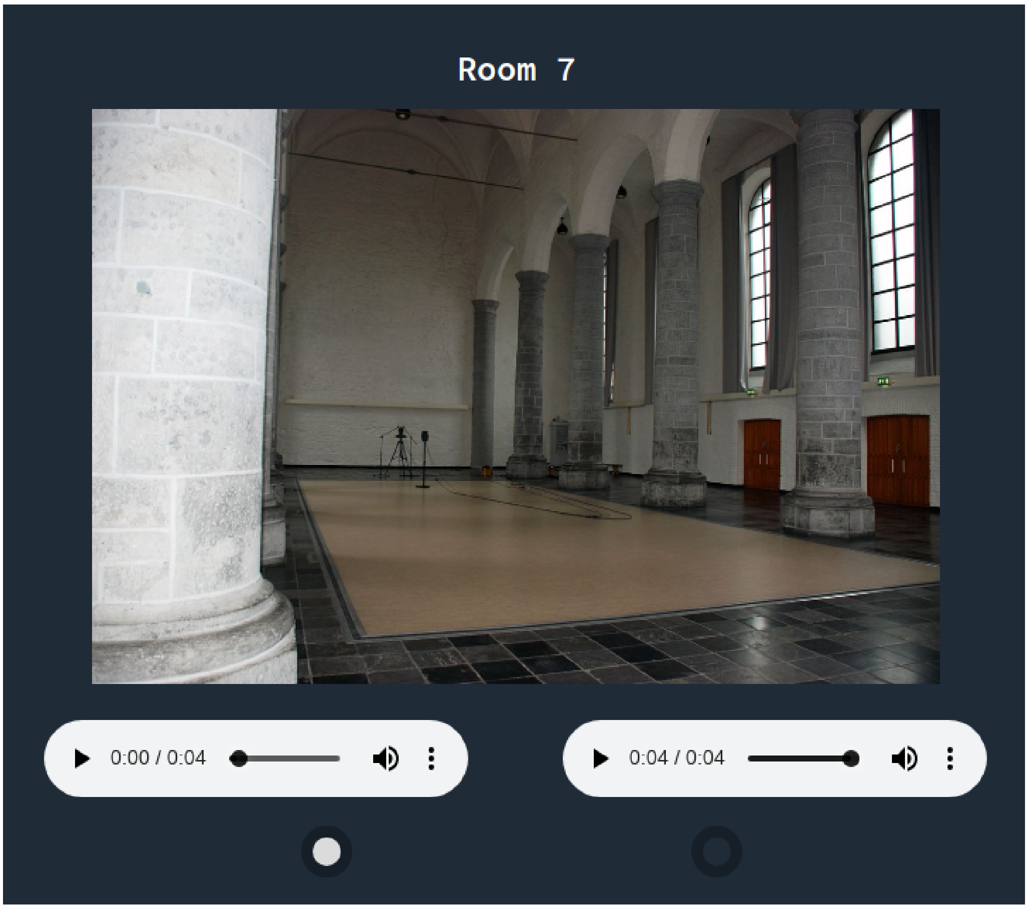


Figure 6.7: The figure shows the design of the visual test. The two audio players hold signals A' and A''.

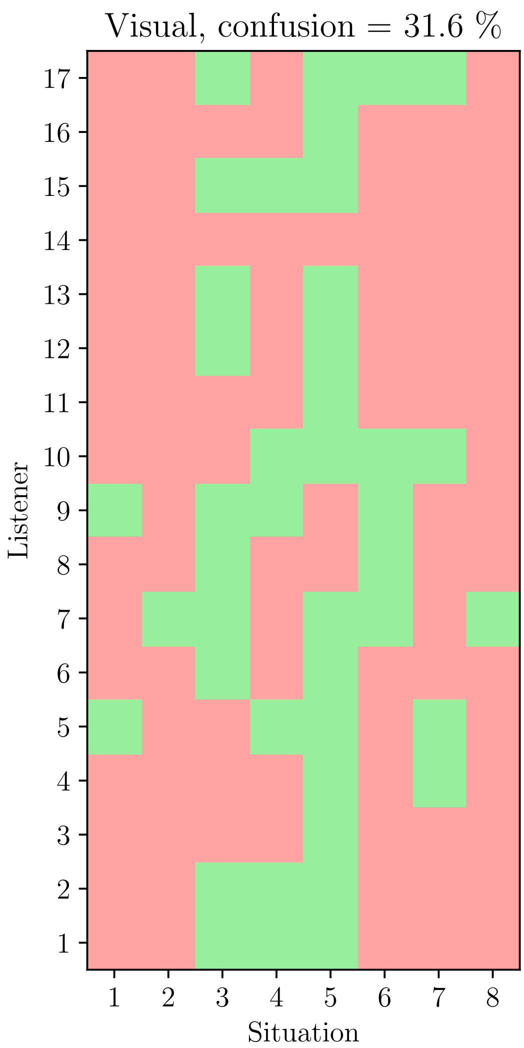


Figure 6.8: The figure shows the result of the visual test, where listeners chose between sound A' and A'' as explained above. Green elements correspond to answers where the listener chose the reverberant signal generated by the network and red elements corresponds to answers where listeners chose the reverberant signal generated using the measured impulse response function. The figure shows that the listeners were reluctant to choose network generated output for situations 1, 2 and 8 which all depicts situations with typically short reverberation.

7

Discussion

In this chapter the results presented in Chapters 4.3 and 5.3, as well as in the framework comparison chapter, Chapter 6, are discussed and further analyzed. The efforts associated with this report that did not end in firm results and that therefor did not qualify for the implementation chapters are also briefly mentioned here.

7.1 General outcome

The output from the networks produced shows that approximation of the simulated impulse response function used to generate reverberant signals can be conducted to a certain extent, in the settings that are explained in previous chapters. The image-source model, although providing easy-to-understand assembly of impulse response functions, is, however, idealized and does not produce completely realistic impulse response functions to be used for database generation. We claim that the frameworks that are the product of this thesis work, to some extent, can yield output that a listener perceives as similar to that of a measured impulse response. Results could not be compiled such that they show that the above claim holds true using the scheme described in Chapter 6. In other terms, using the testing scheme described in Chapter 6 with measured impulse response functions for reverberant signal generation results in that a listener can identify the network every time. This fact is underwhelming but expected due to that, during this thesis work, we have learned that the human hearing system is better than we initially thought at identifying oddities in any signal when presented with a reference signal. We expect that the human hearing system is sufficiently good such that a measured reverberant signal stemming from a system that can be modelled to arbitrary accuracy using the image-source model, and a signal simulated in the virtual environment are easily separated. The visual test that was conducted where listeners were presented with a scene and corresponding sound the results of which can be found in Figure 6.8, is focused around the mental model of system influence based on visual input and shows that some confusion occur. This suggests that users of this framework might find the produced sound as *sufficiently good*, at least in some cases.

We have showed that the output from networks based on the peak data that the majority of this project revolved around are outperformed by spectrogram based counterparts and therefor probably also by MFCC-based networks as the similarities between STFT spectrograms and MFCC spectrograms are significant. In hindsight it would be wise to also have created networks that take MFCC spectrograms as input in order to compare them to the spectrogram based networks but this is instead suggested for future work.

We would like to emphasize that one of the main results of this report are not presented in any of the results sections but instead in the implementation sections and in Appendix A. The six networks presented in this report are themselves the product of persistent evaluation of many other networks and long hours of training and should be regarded as one of the main contributions from us for anyone wanting to continue this work. The dimension-swapping convolutional network might also be useful any in other area of deep learning research where the input data is convolved with the target data in some way. We have not encountered this network type before.

7.2 Future work

The results presented in this report show that it is possible to make estimations of impulse responses based only on received reverberant signals affected by acoustic spaces. However, for a similar scheme to be usable in a real-world setting the framework provided with this report work must be advanced further for several reasons. Firstly, the framework described in this document is only trained, and tested on, single sound sources at a time and thus using infinite signal-to-noise ratio. Future work should therefor include tests on several sound sources at once. Secondly, the database used for training was generated using simulatory means and the network performance limit is thus that of the simulation software. A test for whether the performance of network trained using this database is enough for a user to feel immersed is not conducted and should therefor be prioritized for future work. If reasonable results in terms of immersibility is not met, future work could include better simulatory means, for instance the use of more sophisticated software.

It was found during network optimization trials for some peak data network that the performance of the extractor, at least in some cases, inversely depends on the degree of training of the autoencoders. This suggests that the autoencoders, during training, at some point starts to encode features that cannot be extracted from the reverberant signal. Existence of un-extractable features is expected due to that the problem that this thesis work discuss is somewhat ill-posed. The reverse proportionality, however, suggests that there is an optimum for the extractor performance that not necessarily coincides with the optimum of the autoencoder performance. This interesting result impose an implication that might be circumvented by implementing a feedback loop used during training so that the extractor-autoencoder pair can be trained in unison.

In some situations it might be possible to include room parameters in the input data without having to calculate them manually. An example of this could be a head-mounted device that carries the model, that might also be able to approximate the volume of the room, how far away sources are, or, in some cases, give a 3D-representation of the entire system. We suggest that the performance gains in this situation are studied further.

7.3 Limitations due to the data generation process

The underlying anechoic signals that formed the databases used during the project, as described in Chapter 3, consisted of short segments of our voices recorded in an anechoic setting with essentially no noise. This means that all results presented in this thesis are based on training and evaluating on data from this particular setting. However, the performance of the models in other acoustic settings, such as low-SNR situations, sound events with long decay, sounds containing different frequency ranges etc, can not be guaranteed. Whether or not this poses as a limitation on the models was not investigated for the interest of time.

The use of the peak-data, generated as described in Section 3.4, was motivated by the assumption that frequency independent room impulse response functions would generate sufficiently realistic output. Thus, it would be sufficient to describe the frequency impulse response functions only by the timings and amplitudes of the reflections. This simplification proved to generate output with noticeably different characteristics compared to frequency dependent impulse response functions. Expanding the image-source model with the method described in Section 3.6 provides more realistic impulse response functions that takes care of the lack of frequency dependence. Using spectrogram representations is thus an improvement. Another aspect is that the spectrogram-based framework generalize to databases with frequency response functions of arbitrary realism. The image-source model can therefor be replaced in the database generation process with a method that generate even better impulse response functions. A database based on measured impulse response functions can also be used.

7.3.1 Loss functions

The loss functions used during training of the point-based networks can not be assumed to reflect goodness of the impulse response functions. It can be said that a zero-loss result is ideal, but perceptually it is unclear what the human auditory system perceives as realistic, thus the loss function result is only an approximation of the perceived proximity. The poor results shown in the sample output of the end-to-end network (Figure 4.11) is not reflected in the Chamfer loss figure presented in

Section 4.3.1. This is due to that the Chamfer loss has several local minima that lie in output sets outside of what resembles somewhat realistic impulse response functions. The end-to-end networks were prone to update towards one of those minima due to that the decoder part could update freely and was not restricted to always output data that resembles impulse response functions. This suggests that the use of an autoencoder is a way to clean output space from unwanted data representations that could otherwise result from a network that tries to *outsmart* its creator.

8

Conclusion

The purpose of this thesis was to investigate the possibility to approximate room impulse response functions from reverberant sound under the assumption that the reverberant signal can be produced through convolution of an anechoic signal and a room impulse response.

The proposed solutions are based on deep neural models that takes reverberant signals as input, represented as time-series, MFCC coefficients and STFT spectrograms, and output approximations of impulse response functions as timings and amplitudes of incident reflection (called *peak data*) or STFT spectrograms. The reverberant signals were generated by simulatory means using convolution between: an anechoic transient speech signal and; an impulse response function generated for realistically sized empty rooms using realistic absorption coefficients. The image-source model was used for generation of frequency independent impulse response functions, and an improved version of the image-source model for frequency dependent impulse response functions. The anechoic speech signals were recorded in an anechoic chamber.

The performance of the models was evaluated perceptually by human listeners through listening tests that collect certainty of agreement between reference signals and signals produced using network output. The results show that approximation is possible in this limited case.

This thesis could be regarded as a demonstration that estimation of impulse response functions based on reverberant signals is possible to some extent, varying with the limitations imposed on the data. This work could be used as a motivation to perform further research on the topic.

Bibliography

[But+30] Stephen Butterworth et al. "On the Theory of Filter Amplifiers". In: *Wireless Engineer* 7.6 (1930), pp. 536–541.

[SVN37] S. S. Stevens, J. Volkmann, and E. B. Newman. "A Scale for the Measurement of the Psychological Magnitude Pitch". In: *The Journal of the Acoustical Society of America* 8.3 (Jan. 1937), pp. 185–190.

[RM51] Herbert Robbins and Sutton Monro. *A Stochastic Approximation Method.* Tech. rep. 3. 1951, pp. 400–407.

[SE51] Abraham Savitzky and Marcel J E. *Smoothing and Differentiation of Data by Simplified Least Squares Procedures.* Tech. rep. 2. 1951, p. 1832.

[BP 63] J.W. Tukey B.P. Bogert M.J.R. Healy. "The Quefrency Alanysis of Time Series for Echoes: Cepstrum, Pseudo-Autocovariance, Cross-Cepstrum, and Saphe Cracking". In: *M. Rosenblatt (Ed.), Proc. of the Symp. on Time Series Analysis, Wiley, NY* (1963), pp. 209–243.

[ANR74] N. Ahmed, T. Natarajan, and K.R. Rao. "Discrete Cosine Transform". In: *IEEE Transactions on Computers* C-23.1 (Jan. 1974), pp. 90–93.

[All77] J. Allen. "Short term spectral analysis, synthesis, and modification by discrete Fourier transform". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 25.3 (June 1977), pp. 235–238.

[AB79] Jont B. Allen and David A. Berkley. "Image method for efficiently simulating small-room acoustics". In: *The Journal of the Acoustical Society of America* 65.4 (Apr. 1979), pp. 943–950.

[DM80] S. Davis and P. Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.4 (Aug. 1980), pp. 357–366.

[GM84] Daniel W Griffin A N D Jae S Lim and Senior Member. *Signal Estimation from Modified Short-Time Fourier Transform.* Tech. rep. 2. 1984, p. 36.

[Pet86] Patrick M Peterson. "Simulating the response of multiple microphones to a single acoustic source in a reverberant room". In: *Citation: The Journal of the Acoustical Society of America* 80 (1986), p. 1527.

[PT90] William H Press and Saul A Teukolsky. "Savitzky-Golay Smoothing Filters". In: *Citation: Computers in Physics* 4 (1990), p. 669.

[Oli06] Travis E Oliphant. *Guide to NumPy.* Tech. rep. 2006.

[Hun07] John D. Hunter. “Matplotlib: A 2D Graphics Environment”. In: *Computing in Science & Engineering* 9.3 (May 2007), pp. 90–95.

[Oli07] Travis E. Oliphant. “Python for Scientific Computing”. In: *Computing in Science & Engineering* 9.3 (May 2007), pp. 10–20.

[Ken+08] Paul Kendrick et al. “Monaural room acoustic parameters from music and speech”. In: *The Journal of the Acoustical Society of America* 124.1 (July 2008), pp. 278–287.

[Vor08] Michael. Vorländer. *Auralization : fundamentals of acoustics, modelling, simulation, algorithms and acoustic virtual reality*. Springer, 2008, p. 335.

[McK10] Wes McKinney. *Data Structures for Statistical Computing in Python*. 2010.

[KKS11] Kshitiz Kumar, Chanwoo Kim, and Richard M. Stern. “Delta-spectral cepstral coefficients for robust speech recognition”. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2011, pp. 4784–4787.

[PK11] Etienne Parizet and Vincent Koehl. “Influence of Train Colour on Loudness Judgments”. In: *Acta Acustica united with Acustica* 97.2 (Mar. 2011), pp. 347–349.

[PCL12] Nils Peters, Jaeyoung Choi, and Howard Lei. *Matching Artificial Reverb Settings to Unknown Room Recordings: A Recommendation System for Reverb Plugins*. 2012.

[PLF12] Nils Peters, Howard Lei, and Gerald Friedland. “Name that room”. In: *Proceedings of the 20th ACM international conference on Multimedia - MM ’12*. New York, New York, USA: ACM Press, 2012, p. 841.

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. “Network In Network”. In: (Dec. 2013).

[KB14] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (Dec. 2014).

[McF+15] Brian McFee et al. “librosa: Audio and Music Signal Analysis in Python”. In: *Proceedings of the 14th Python in Science Conference*. 2015, pp. 18–24.

[HL16] Yoonchang Han and Kyogu Lee. “Acoustic scene classification using convolutional neural network and multiple-width frequency-delta data augmentation”. In: (July 2016).

[PLS16] J Pons, T Lidy, and X Serra. “Experimenting with musically motivated convolutional neural networks”. In: *2016 14th International Workshop on Content-Based Multimedia Indexing (CBMI)*. 2016, pp. 1–6.

[Qi+16] Charles R. Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: (Dec. 2016).

[Val+16] Michele Valenti et al. *Detection and Classification of Acoustic Scenes and Events*. Tech. rep. 2016.

[Hyd+17] Rakib Hyder et al. “Acoustic Scene Classification using a CNN-SuperVector system trained with Auditory and Spectrogram Image Features”. In: (2017).

[PGL17] Adam Paszke, Sam Gross, and Adam Lerer. *Automatic differentiation in PyTorch*. 2017.

[SBD17] Robin Scheibler, Eric Bezzam, and Ivan Dokmanić. “Pyroomacoustics: A Python package for audio room simulations and array processing algorithms”. In: (Oct. 2017).

[Gil18] Sean Gillies. *Shapely Documentation Release 1.7a1*. Tech. rep. 2018.

[Ahr19] Jens Ahrens. “Auralization of Omnidirectional Room Impulse Responses Based on the Spatial Decomposition Method and Synthetic Spatial Data”. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2019, pp. 146–150.

[Abe+] Jonathan S Abel et al. *Estimating Room Impulse Responses from Recorded Balloon Pops*. Tech. rep.

A

Tables

Table A.1: The table lists the values used to generate the database used for evaluation of peak data models.

Database generation settings	
Setting	Value
Train samples:	10^4 (10^3 rooms)
Val samples:	10^3 (10^2 rooms)
Number of receivers per room:	10
Range of permitted wall lengths:	2 m to 12 m
Range of permitted ceiling height:	2 m to 4 m
Sampling rate:	44100 samples/s
Max order or reflections:	8
Range of permitted absorption coefficients:	0.01 to 0.4
Number of MFC coefficients:	40
FFT window length:	2048
Window hop length:	512
Least number of incidents:	256
Maximum number of MFCC vectors per audio sample:	128

Table A.2: The table lists the values used to generate the database used for evaluation of spectrogram models.

Database generation settings	
Setting	Value
Train samples:	10^4 (10^3 rooms)
Val samples:	10^3 (10^2 rooms)
Number of receivers per room:	10
Range of permitted wall lengths:	3 m to 15 m
Range of permitted ceiling height:	2 m to 4 m
Sampling rate:	44100 samples/s
Max order or reflections:	60
Range of permitted absorption coefficients:	0.01 to 0.99
Number of samples per STFT frame:	256
Window hop length:	64

Table A.4: The table lists the operations performed in the autoencoder network, as described in Section 4.2.2. In this case the size of the bottleneck is set to $b = 16$. For different sizes of the bottleneck b, the number of filters in the final convolutional layer would be b instead, thus changing its output data shape to $[b \times 1 \times 256]$.

Operation	Output data shape
2D point cloud input data	$2 \times 1 \times 256$
$[1 \times 1]$ conv, 32 filters, ReLU, BN	$32 \times 1 \times 256$
$[1 \times 1]$ conv, 64 filters, ReLU, BN	$64 \times 1 \times 256$
$[1 \times 1]$ conv, 128 filters, ReLU, BN	$128 \times 1 \times 256$
$[1 \times 1]$ conv, 16 filters	$16 \times 1 \times 256$
$[1 \times 1 \times 256]$ max-pooling	$16 \times 1 \times 1$
Fully connected, ReLU	$256 \times 1 \times 1$
Fully connected, ReLU	$256 \times 1 \times 1$
Fully connected	$512 \times 1 \times 1$
Reshape	$2 \times 1 \times 256$

Table A.3: The table lists the operations performed in the end-to-end network with MFCC as input data, as described in Section 4.2.3. In this case the size of the bottleneck is set to $b = 16$.

Operation	Output data shape
MFCCs and delta features input data	$3 \times 40 \times 128$
$[3 \times 3]$ conv, 32 filters, $[1 \times 1]$ zero-padding, ReLU	$32 \times 40 \times 128$
$[3 \times 3]$ conv, 32 filters, $[1 \times 1]$ zero-padding, ReLU	$32 \times 40 \times 128$
$[2 \times 2]$ max-pooling	$32 \times 20 \times 64$
$(p = 0.25)$ dropout	$32 \times 20 \times 64$
$[3 \times 3]$ conv, 64 filters, $[1 \times 1]$ zero-padding, ReLU	$64 \times 20 \times 64$
$[3 \times 3]$ conv, 64 filters, $[1 \times 1]$ zero-padding, ReLU	$64 \times 20 \times 64$
$[2 \times 2]$ max-pooling	$64 \times 10 \times 32$
$(p = 0.25)$ dropout	$64 \times 10 \times 32$
$[3 \times 3]$ conv, 128 filters, $[1 \times 1]$ zero-padding, ReLU	$128 \times 10 \times 32$
$[3 \times 3]$ conv, 128 filters, $[1 \times 1]$ zero-padding, ReLU	$128 \times 10 \times 32$
$[2 \times 2]$ max-pooling	$128 \times 5 \times 16$
$(p = 0.25)$ dropout	$128 \times 5 \times 16$
$[3 \times 3]$ conv, 256 filters, $[1 \times 1]$ zero-padding, ReLU	$256 \times 5 \times 16$
$[5 \times 16]$ average-pooling	$256 \times 1 \times 1$
Fully connected, ReLU	$256 \times 1 \times 1$
Fully connected, ReLU	$256 \times 1 \times 1$
Fully connected	$16 \times 1 \times 1$
Fully connected, ReLU	$256 \times 1 \times 1$
Fully connected, ReLU	$256 \times 1 \times 1$
Fully connected	$512 \times 1 \times 1$
Reshape	$2 \times 1 \times 256$

Table A.5: The table lists the operations performed in the MFCC extractor network, as described in Section 4.2.1.1. In this case the size of the bottleneck is set to $b = 16$. For different sizes of the bottleneck b, the network would be identical, apart from the output data shape of the final layer, which would be $[b \times 1 \times 1]$.

Operation	Output data shape
MFCCs and delta features input data	$3 \times 40 \times 128$
$[3 \times 3]$ conv, 32 filters, $[1 \times 1]$ zero-padding, ReLU	$32 \times 40 \times 128$
$[3 \times 3]$ conv, 32 filters, $[1 \times 1]$ zero-padding, ReLU	$32 \times 40 \times 128$
$[2 \times 2]$ max-pooling	$32 \times 20 \times 64$
$(p = 0.25)$ dropout	$32 \times 20 \times 64$
$[3 \times 3]$ conv, 64 filters, $[1 \times 1]$ zero-padding, ReLU	$64 \times 20 \times 64$
$[3 \times 3]$ conv, 64 filters, $[1 \times 1]$ zero-padding, ReLU	$64 \times 20 \times 64$
$[2 \times 2]$ max-pooling	$64 \times 10 \times 32$
$(p = 0.25)$ dropout	$64 \times 10 \times 32$
$[3 \times 3]$ conv, 128 filters, $[1 \times 1]$ zero-padding, ReLU	$128 \times 10 \times 32$
$[3 \times 3]$ conv, 128 filters, $[1 \times 1]$ zero-padding, ReLU	$128 \times 10 \times 32$
$[2 \times 2]$ max-pooling	$128 \times 5 \times 16$
$(p = 0.25)$ dropout	$128 \times 5 \times 16$
$[3 \times 3]$ conv, 256 filters, $[1 \times 1]$ zero-padding, ReLU	$256 \times 5 \times 16$
$[5 \times 16]$ average-pooling	$256 \times 1 \times 1$
Fully connected, ReLU	$256 \times 1 \times 1$
Fully connected, ReLU	$256 \times 1 \times 1$
Fully connected	$16 \times 1 \times 1$

Table A.6: The table lists the operations performed in the time series extractor network, as described in Section 4.2.1.2. In this case the size of the bottleneck is set to $b = 16$. For different sizes of the bottleneck b, the network would be identical, apart from the output data shape of the final layer, which would be $[b \times 1 \times 1]$.

Operation	Output data shape
Time series input data	$1 \times 1 \times 65536$
$[1 \times 1]$ conv, 2 filters, BN, ReLU	$2 \times 1 \times 65536$
$[1 \times 2]$ max-pooling	$2 \times 1 \times 32768$
$[1 \times 1]$ conv, 4 filters, ReLU	$4 \times 1 \times 32768$
$[1 \times 2]$ max-pooling	$4 \times 1 \times 16384$
$[1 \times 1]$ conv, 8 filters, ReLU	$8 \times 1 \times 16384$
$[1 \times 2]$ max-pooling	$8 \times 1 \times 8192$
$[1 \times 1]$ conv, 16 filters, ReLU	$16 \times 1 \times 8192$
$[1 \times 2]$ max-pooling	$16 \times 1 \times 4096$
$[1 \times 1]$ conv, 32 filters, ReLU	$32 \times 1 \times 4096$
$[1 \times 2]$ max-pooling	$32 \times 1 \times 2048$
$[1 \times 1]$ conv, 64 filters, ReLU	$64 \times 1 \times 2048$
$[1 \times 2]$ max-pooling	$64 \times 1 \times 1024$
$[1 \times 1]$ conv, 64 filters, ReLU	$64 \times 1 \times 1024$
$[1 \times 1024]$ max-pooling	$64 \times 1 \times 1$
Fully connected, ReLU	$64 \times 1 \times 1$
Fully connected	$16 \times 1 \times 1$

Table A.7: The table lists the operations performed in the time convolutional downsampling-upsampling network, as described in Section 5.2.1. The abbreviations used in this table are C=convolutional layer, CT=convolutional transpose layer, c=number of kernels, k=kernel size, p=zero-padding, op=output zero-padding, s=stride, BN=batch normalization.

Operation	Output shape
STFT magnitude spectrogram input data	$1 \times 65 \times 372$
C(c=8, k=$[7 \times 7]$, p=$[3 \times 3]$), BN, ReLU	$8 \times 65 \times 372$
Max-pooling($[1 \times 5]$)	$8 \times 65 \times 74$
C(c=32, k=$[1 \times 5]$, p=$[0 \times 2]$), BN, ReLU	$32 \times 65 \times 74$
Max-pooling($[1 \times 4]$)	$32 \times 65 \times 18$
C(c=64, k=$[1 \times 3]$, p=$[0 \times 1]$), BN, ReLU	$64 \times 65 \times 18$
Max-pooling($[1 \times 4]$)	$64 \times 65 \times 4$
C(c=128, k=$[1 \times 3]$, p=$[0 \times 1]$), BN, ReLU	$128 \times 65 \times 4$
CT(c=64, k=$[1 \times 3]$, p=$[0 \times 1]$, s=$[1 \times 2]$, op=$[0 \times 1]$), BN, ReLU	$64 \times 65 \times 8$
CT(c=32, k=$[1 \times 3]$, p=$[0 \times 1]$, s=$[1 \times 2]$), BN, ReLU	$32 \times 65 \times 15$
CT(c=16, k=$[1 \times 3]$, p=$[0 \times 1]$, s=$[1 \times 2]$), BN, ReLU	$16 \times 65 \times 29$
CT(c=8, k=$[1 \times 3]$, p=$[0 \times 1]$, s=$[1 \times 2]$), BN, ReLU	$8 \times 65 \times 57$
CT(c=4, k=$[1 \times 3]$, p=$[0 \times 1]$, s=$[1 \times 2]$, op=$[0 \times 1]$), BN, ReLU	$4 \times 65 \times 116$
C(c=1, k=$[5 \times 5]$, p=$[2 \times 2]$), ReLU	$1 \times 65 \times 116$

Table A.8: The table lists the operations performed in the dimension-swapping convolutional network, as described in Section 5.2.2. The abbreviations used in this table are C=convolutional layer, c=number of kernels, k=kernel size, p=zero-padding, BN=batch normalization.

Operation	Output shape
STFT magnitude spectrogram input data	$1 \times 129 \times 372$
C(c=16, k=[5×5], p=[2×2]), BN, ReLU	$16 \times 129 \times 372$
Transpose(1, 3)	$372 \times 129 \times 16$
C(c=116, k=[3×7], p=[1×3]), BN, Tanh	$116 \times 129 \times 16$
C(c=116, k=[3×7], p=[1×3]), BN, Tanh	$116 \times 129 \times 16$
Transpose(1, 3)	$16 \times 129 \times 116$
C(c=8, k=[3×3], p=[1×1]), BN, ReLU	$8 \times 129 \times 116$
C(c=1, k=[3×3], p=[1×1]), BN	$1 \times 129 \times 116$

B

Equations

B.1 Weight update in a fully connected neural network

Given outputs O_i and targets ζ_i for a certain input sample μ the distance between O and ζ can be calculated according to Equation (B.1) below. In order to minimize the distance H, the weights and biases corresponding to the perceptrons must be updated in the right direction. Given an ANN with input ξ_k, hidden neurons V_j, thresholds θ_j and Θ_i, weights $w_{j,k}$ and $W_{i,j}$, and activation function g, we can define the forward pass producing O_j, according to Equations (B.2) and (B.3).

$$H = \frac{1}{2}\sum_{\mu}\sum_{i}(\zeta_i^{(\mu)} - O_i^{(\mu)})^2 \tag{B.1}$$

$$V_j^{(\mu)} = g(b_j^{(\mu)}), \quad b_j^{(\mu)} = \sum_k w_{j,k}\xi_k^{(\mu)} - \theta_j \tag{B.2}$$

$$O_j^{(\mu)} = g(b_i^{(\mu)}), \quad b_i^{(\mu)} = \sum_j W_{i,j}V_j^{(\mu)} - \Theta_i \tag{B.3}$$

Introducing the learning rate η the weight updates can be expressed according to Equation (B.4). Similar expressions for the remaining weights and biases exists and the learning rate acts analogously, see Equation (B.5).

$$W_{i,j} \leftarrow W_{i,j} + \delta W_{i,j}, \text{ where, } \delta W_{i,j} = -\eta\frac{\partial H}{\partial W_{i,j}} \tag{B.4}$$

$$\delta w_{j,k} = -\eta\frac{\partial H}{\partial w_{j,k}} \quad \delta\Theta_i = -\eta\frac{\partial H}{\partial \Theta_i} \quad \theta_j = -\eta\frac{\partial H}{\partial \theta_j} \tag{B.5}$$

Differentiating both sides of Equation (B.1) according to Equation (B.6) yields Equation (B.7) where $g'(b_i^{(\mu)})$ and $V_j^{(\mu)}$ are results of applying the chain rule to $O_m^{(\mu)}$. Here the expression for O was replaced with the expression for the output of the hidden layer and then reinserted for clarity.

$$\frac{\partial H}{\partial W_{i,j}} = \frac{\partial}{\partial W_{i,j}} \left\{ \frac{1}{2} \sum_{\mu} \sum_{m} \left[\zeta_m^{(\mu)} - \underbrace{g(\sum_n W_{m,n} V_n^{(\mu)} - \Theta_m)}_{O_m^{(\mu)}} \right]^2 \right\} \tag{B.6}$$

$$\frac{\partial H}{\partial W_{i,j}} = - \sum_{\mu} \underbrace{(\zeta_i^{(\mu)} - O_i^{(\mu)}) g'(b_i^{(\mu)})}_{\delta_i^{(\mu)}} V_j^{(\mu)} \tag{B.7}$$

Using similar techniques for the remaining scaled gradients in Equation (B.5) the expressions in Equation (B.8) are obtained and it is realized that the calculations can be generalized for networks of arbitrary depths.

$$\delta W_{i,j} = \eta \sum_{\mu} \delta_i^{(\mu)} V_j^{(\mu)} \tag{B.8}$$

$$\delta w_{j,k} = \eta \sum_{\mu} \delta_j^{(\mu)} \xi_k^{(\mu)}, \text{ with, } \delta_j^{(\mu)} = \sum_i \delta_i^{(\mu)} g'(b_j^{(\mu)}) W_{i,j} \tag{B.9}$$

$$\delta \Theta_i = -\eta \sum_{\mu} \delta_i^{(\mu)} \tag{B.10}$$

$$\delta \theta_j = -\eta \sum_{\mu} \delta_j^{(\mu)} \tag{B.11}$$

C

Implementation details

The framework used for this thesis work was implemented using Python 3.7 and a plurality of packages for data handling, calculations and network handling. There are several machine learning frameworks built for, or ported to, Python, such as Tensorflow, Keras, Caffe and PyTorch and the framework described in this report revolves around PyTorch[1] [PGL17]. PyTorch was chosen due to the ease of use and prototyping capabilities that stems from the possibility of letting gradient calculation be handled automatically. PyTorch supports Tensor computations with strong GPU acceleration due to that PyTorch Tensors can be used on GPUs that support CUDA. Other packages used during this projects are listed in Table C.1

Table C.1: The table lists the packages used that had greatest influence on the project.

Python packages excluding PyTorch	
Package	**Area of use**
NumPy[2] [Oli06]	Array operations and general math
SciPy[3] [Oli07]	Various numerical operations
MatPlotLib[4] [Hun07]	Plotting and figure handling
LibROSA[5] [McF+15]	Audio analysis
Pyroomacoustics[6] [SBD17]	Target generation through acoustic simulations
Pandas[7] [McK10]	CSV-file operations
Shapely[8] [Gil18]	Defining geometric objects (rooms)

[1] https://pytorch.org
[2] https://numpy.org
[3] https://scipy.org
[4] https://matplotlib.org
[5] https://librosa.github.io/librosa/
[6] https://pypi.org/project/pyroomacoustics/
[7] https://pandas.pydata.org
[8] https://pypi.org/project/Shapely/

C.1 Training and hardware

Given the number of parameters in each of the networks, training had to be conducted using graphics processors and run for an extended period of time. An Nvidia GTX 1060 6GB GPU was used for the entirety of this project, the speed of which proved to be sufficient.