Audio Deepfake Detection using the Stationary Wavelet Transform

Master Thesis

Rheinische Friedrich-Wilhelms-Universität Bonn
Computer Science

Author: Niclas Pillath

Supervisor: Dr. Moritz Wolter

Examiner: Prof. Dr. Estela Suarez
Dr. Moritz Wolter

June 2024

Preface

This thesis is about the detection of audio deepfakes using the stationary wavelet transform. The relevance of deepfake detection in today's digital landscape cannot be overstated. I am very happy to be able to contribute to the improvement of machine-based detection methods. Although this work did not result in a new detector that enhances current capabilities, we have answered the question of whether the stationary wavelet transform is suitable as an input for CNN-based detectors.

I would like to thank my supervisor Dr. Moritz Wolter for introducing me to wavelet transforms and for providing countless suggestions for my experiments. I also want to thank Dr. Bartosz Kostrzewa for his tips on managing large datasets in a HPC environment. Finally, I wish to sincerely thank Prof. Dr. Estela Suarez for her valuable feedback and the opportunity to compute at JSC. Having one of the world's most powerful supercomputers available for a master's thesis is not something taken for granted. It was a truly great experience.

Contents

1 Introduction **2**

2 Theoretical Background **4**

2.1 Wavelet Transforms 4

2.1.1 Continuous Wavelet Transform 7

2.1.2 Discrete Wavelet Transform 8

2.1.3 Discrete Wavelet Packet Transform 15

2.1.4 Stationary Wavelet Transform 17

2.2 Convolutional Neural Networks 19

2.2.1 Convolutional Layer 20

2.2.2 Detector Layer 23

2.2.3 Pooling Layer 24

2.2.4 Residual Learning 24

3 Experiments **28**

3.1 Setup 29

3.1.1 Environment 29

3.1.2 Implementation 30

3.1.3 Configuration 30

3.2 Dataset 30

3.2.1 Preprocessing 31

3.3 Fingerprints 33

3.4 1-dimensional vs 2-dimensional CNNs 36

3.5 Transfer Learning with ResNet-50 38

3.6 Wide Residual CNNs 42

3.6.1 Evaluation 44

3.7 Comparison to DWPT models 47

4 Conclusion **53**

References **54**

Appendices **58**

A Complete Evaluation Results **59**

B No log-transform **71**

Abstract

Recent advances in generative modelling have uncovered new avenues for creative expression, while also raising the potential of malicious misuse. Deepfakes pose significant risks to personal privacy, information integrity and cybersecurity. One possible countermeasure against deepfakes are robust detection models based on deep learning. In the context of audio, convolutional architectures have demonstrated good recognition rates. The raw audio data is usually transformed into a suitable format for convolutional neural networks. Current state-of-the-art models employ Fourier or wavelet transforms. This thesis investigates the detection of audio fakes using deep convolutional networks that process stationary wavelet transform inputs. The results show functional detector models, but also reveal limitations of the stationary wavelet transform.

List of Abbreviations

ANOVA	Analysis of Variance
AST	Audio Spectrogram Transformer
ASV	Automatic Speaker Verification
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
CWT	Continuous Wavelet Transform
DFT	Discrete Fourier Transform
DWPT	Discrete Wavelet Packet Transform
DWT	Discrete Wavelet Transform
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
JSC	Jülich Supercomputing Centre
DCNN	Dilated Convolutional Neural Network
LCNN	Light Convolutional Neural Network
MRA	Multiresolution Analysis
ReLU	Rectified Linear Unit
STFT	Short-Time Fourier Transform
SWT	Stationary Wavelet Transform

Wavelets[1]

`haar`	Haar
`db`	Daubechies
`sym`	Symlet
`coif`	Coiflet

[1]Please refer to the manual of PyWavelets [24] for more information.

Chapter 1

Introduction

Recent advances in deep learning have marked the beginning of a new era in synthetic media creation. This has uncovered new avenues for creative expression, while also raising the potential for malicious misuse. As the quality and accessibility of these technologies continue to improve, it is becoming increasingly evident that they will have a profound impact on society.

The introduction of generative adversarial networks (GANs) by Goodfellow et al. in 2014 was the start of this new era. The researchers proposed a novel training framework in which a generative model competes with an adversary, called the discriminative model. The generative model learns to produce samples that are similar to those in the dataset, while the discriminative model learns to distinguish these generated samples from the original samples in the dataset. Goodfellow et al. demonstrated competitive results on image datasets and revealed the potential of this framework [11]. Since their introduction, GANs have been continuously developed and improved, and they continue to play a key role in today's research [12]. They already reached a point where it is now possible to produce very realistic images [18] and audio waveforms [19, 25, 3].

In addition to GANs, however, there exist numerous other neural network architectures and learning frameworks that are employed for generative modeling. Popular approaches include variational autoencoders and fully-visible belief networks [12]. Recently, diffusion models have emerged as a promising alternative to the previously mentioned approaches, pushing the state of the art in image and video generation tasks [33].

Apart from their legitimate applications, it is evident that the advancement of generative models presents a potential for malicious exploitation. Synthetic audio, images, or videos that appear realistic but have been fabricated by a deep neural network are commonly referred to as *deepfakes*. One form of deepfakes involves the imitation of individuals without their knowledge or consent [44], which can be used for various criminal activities, such as spear phishing or bypassing biometric authentication systems. In 2020, an audio deepfake of a company director was used to authorize money transfers. The bank employee recognized the company director's voice and was thus successfully deceived [28]. Although such threats mainly affect individuals, it is plausible that larger audiences could also be targeted by deepfakes. And indeed, deepfakes are already being used today to widely spread misinformation. This erodes the overall trust in digital content. The relevance of deepfake detection in today's digital landscape cannot be overstated. Deepfakes pose significant risks to personal privacy, information integrity and cybersecurity [44]. Therefore, it is crucial to understand the extent of these threats and to find suitable countermeasures.

In order to raise public awareness and sensitize people to deepfakes, private efforts, such as blogs and podcasts, as well as institutional projects have emerged. It is worth mentioning the "Detect Fakes" research program[1], launched at the MIT Media Lab in 2019. In addition to the publications, the program provides a website that tests visitors in their image deepfake detection capabilities. Human detection capabilities have been studied using speech [28] and video [13] deepfakes. The results generally indicate limited detection capabilities. The difficulty for humans to detect deepfakes motivates machine-based methods, especially those based on machine learning. Challenges like ASVspoof [38] represent a key motivator in this regard.

Recently, the majority of the leading deepfake detection models employ deep learning. The current state of the art mainly involves convolutional neural networks. However, other architectures, such as variational autoencoders or recurrent neural networks, are also used [44]. In general, the various approaches can be distinguished based on the features employed for their prediction. These include biometric features such as eye blinking or facial expressions, and media features, such as inconsistencies and artifacts. In addition, features specific to a generative model can also be used [44]. To illustrate, researchers discovered that GAN-based models leave a fingerprint, that can be extracted from the samples they generate [29]. This enables not only the detection of a deepfake, but also its attribution to a specific GAN model [43].

In the context of this thesis, we will limit ourselves to audio deepfakes. We investigate their detection using convolutional neural networks, which have already demonstrated good recognition rates in the existing literature. It is a common practice to transform the raw audio data into a suitable format for convolutional neural networks. One possible option is the transformation into a time-frequency representation. Available methods, such as Fourier and wavelet transformations, have already been investigated and showed promising results. However, the stationary wavelet transform remains unexplored, which motivates the topic of this thesis.

Chapter 2 establishes the theoretical foundations for the wavelet transforms and convolutional neural networks. Then, in Chapter 3, a series of experiments will examine different approaches to audio deepfake detection using convolutional neural networks. The corresponding digital content and source code can be found here[2]. This thesis places particular emphasis on the stationary wavelet transform and its comparison to other methods. Finally, Chapter 4 presents a summary of our findings and assesses the utility of the stationary wavelet transform in the context of audio deepfake detection.

[1] https://detectfakes.kellogg.northwestern.edu

[2] https://gitlab.jsc.fz-juelich.de/suarez1/niclas_master

Chapter 2

Theoretical Background

2.1 Wavelet Transforms

We will restrict our notation to the one-dimensional case, as we will only be working with audio. However, it should be noted that the presented techniques can be extended to any number of dimensions. Let us consider a possibly infinite real-valued function $x(\cdot)$ of an ordered independent variable u. While it may seem intuitive to consider u as time and view its origin as a continuous range of values, in practice, u is usually from a discrete set of values. Then, signal $x(\cdot)$ is sampled in evenly spaced intervals, known as *sample rate.*
The well-known Fourier transform,

$$\hat{x}(f) := \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x(u)\, \mathrm{e}^{-ifu}\, du\,, \tag{2.1}$$

is capable of transforming a continuous signal $x(u)$ into a function $\hat{x}(f)$ of frequency f. The conversion from the time domain into the frequency domain yields a representation of the frequency content of the original function [4, Chapter 1.1]. Similarly, the discrete Fourier transform (DFT),

$$X(f) := \sum_{u=-\infty}^{\infty} x_u\, \mathrm{e}^{-i2\pi fu}\,, \tag{2.2}$$

transforms a discretized infinite signal $\{x_u : u \in \mathbb{Z}\}$ into a function $X(f)$ of frequency f [31, Chapter 2.2]. In this context it is important to mention the *Nyquist frequency*, which is the highest frequency that can be represented by a discretized signal. Given a sampling interval Δt, the Nyquist frequency is defined as $f_{\mathcal{N}} := 1/(2\Delta t)$ [31, Page 87].

The result of the standard Fourier transform does not contain any time related information. Suppose we were interested in the frequency content locally in time instead, which is commonly referred to as *time-frequency localization.* Then the standard approach would be to use the windowed Fourier transform,

$$\hat{x}(f,\tau) := \int_{-\infty}^{\infty} x(u)g(u-\tau)\mathrm{e}^{-ifu}\, du\,, \tag{2.3}$$

often referred to as Short-Time Fourier Transform (STFT). The basic idea is to use a windowing function g that is translated using τ to extract slices of x before taking its Fourier transform. In practice, the frequencies f and the locations τ are assigned discrete, evenly spaced values. There are many possible choices for the windowing function, one popular choice is the Gaussian function [4, Chapter 1.2].

The wavelet transform is another technique to yield a time-frequency description of some signal $x(u)$. It does, however, have a few important differences compared to the windowed Fourier transform, which we will discuss later. First, we will define wavelets, the indispensable component of any type of wavelet transform. The term wavelet is a neologism that originated from the French word "ondelette", which translates to "small wave". Formally, a wavelet is a real-valued function $\psi(\cdot)$ satisfying the following two basic properties.

I) The integral of $\psi(\cdot)$ is zero:

$$\int_{-\infty}^{\infty} \psi(u)\, du = 0 \tag{2.4}$$

II) The square of $\psi(\cdot)$ integrates to unity:

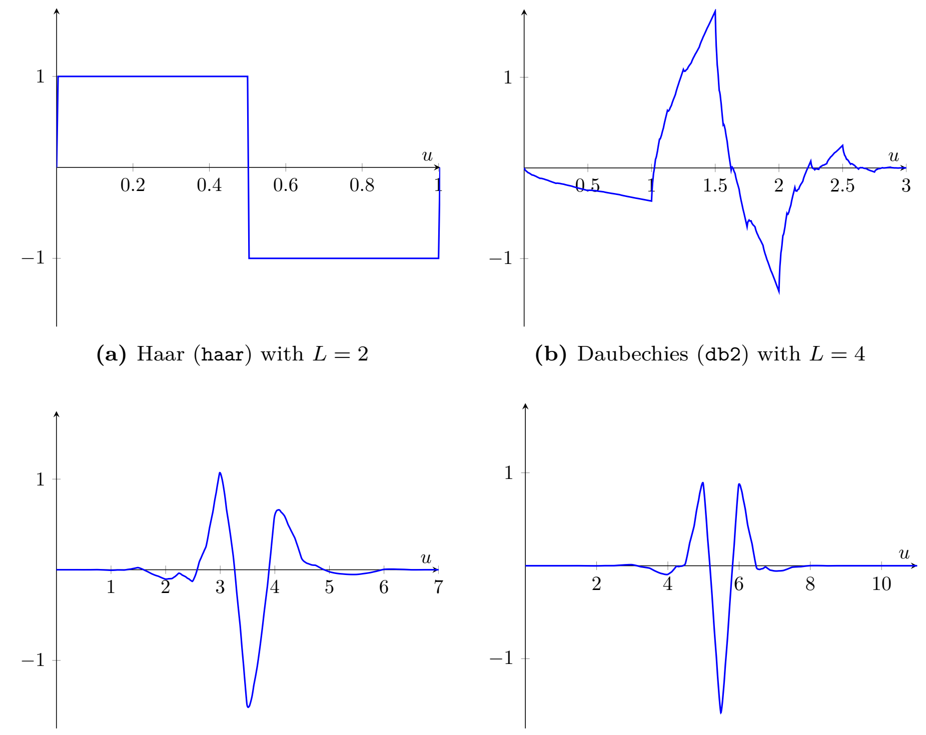
$$\int_{-\infty}^{\infty} \psi(u)^2\, du = 1 \tag{2.5}$$

The importance of the aforementioned properties becomes clear when considering their intuition. For some $0 < \epsilon < 1$, there must be an interval $[-T, T]$ of finite length such that $\int_{-T}^{T} \psi(u)^2\, du > 1 - \epsilon$ holds, if (2.5) is satisfied. Hence, for ϵ approaching zero, $\psi(u)$ can only deviate insignificantly from zero outside the interval $[-T, T]$. At the same time, (2.5) also forces $\psi(u)$ to make some excursions away from zero. However, (2.4) requires, that any excursion above zero must be canceled out by an excursion below zero, because otherwise the integral of $\psi(u)$ would not be zero. Combined with the observation, that the nonzero activity of $\psi(u)$ is restricted to only a relatively small portion of the real axis, every $\psi(u)$ satisfying properties (2.4) and (2.5) resembles a small wave. This gives us a perfect recipe for constructing wavelets [31]. Figure 2.1 displays a selection of popular wavelets. The choice of a suitable wavelet is not straightforward and is dependent on the intended application.

We will now formulate the basic concept of any wavelet transform before diving deeper into different types of wavelet transforms. Let us begin by introducing two new variables, scale λ and translation t. These two parameters will be used to create copies of a given wavelet $\psi(u)$. Formally, this can be defined as

$$\psi_{\lambda,t}(u) := \psi\left(\frac{u-t}{\lambda}\right). \tag{2.6}$$

Intuitively, the scaling parameter λ dilates the wavelet and the translation parameter t will move the center of the wavelet. Note, that as λ changes, the wavelet will analyze different frequency ranges. Small values for λ cover high



(a) Haar (`haar`) with $L = 2$ **(b)** Daubechies (`db2`) with $L = 4$

(c) Symlet (`sym4`) with $L = 8$ **(d)** Coiflet (`coif2`) with $L = 12$

Figure 2.1: The $\psi(u)$ functions from a selection of popular wavelets. Their respective filter lengths are denoted by parameter L.

frequencies and large values for λ cover low frequencies of an input signal. Because every wavelet $\psi_{\lambda,t}(u)$ can be seen as a child from $\psi(u)$, the original wavelet $\psi(u)$ is also referred to as the *mother wavelet* [4, Chapter 1.2].

The wavelet transform utilizes individual $\psi_{\lambda,t}(u)$ as analyzing functions, similar to the function g used in the windowed Fourier transform. However, g solely translates a fixed-size window. In contrast, $\psi_{\lambda,t}$ scales with λ and thus inevitably also changes the size of the analyzing window, because the nonzero activity of $\psi(u)$ is limited. Consequently, wavelets are capable of naturally forming windows of frequency-dependent size. Compared to g, where the size of every window is the same, independent of the frequency that is being analyzed, wavelets are therefore able to better capture short-lived high frequency phenomena [4, Chapter 1.2].

Now, analogous to (2.3), by taking the inner products of $x(u)$ with the analyzing functions $\psi_{\lambda,t}(u)$, that traverse the time and frequency domain of $x(\cdot)$, we yield a time-frequency description of the input signal. At the same time, this raises the question on how to choose the scaling parameter λ and translation parameter t. We will discuss the different approaches in the following subsections.

2.1.1 Continuous Wavelet Transform

The first, and perhaps also most intuitive approach, is to select λ and t from the continuum of real numbers $\mathbb{R}$, except $\lambda = 0$. For simplicity, we will restrict λ to positive values only. Formally, this results in the following family of child wavelets $\mathcal{F} := \left\{\psi_{\lambda,t} \mid \lambda \in \mathbb{R}_{>0}, t \in \mathbb{R}\right\}$, where each child wavelet is defined as

$$\psi_{\lambda,t}(u) := \frac{1}{\sqrt{\lambda}}\, \psi\left(\frac{u-t}{\lambda}\right). \tag{2.7}$$

The continuous wavelet transform (CWT) is then defined by

$$\mathcal{W}_x(\lambda,t) := \int_{-\infty}^{\infty} x(u)\psi_{\lambda,t}(u)\, du\,. \tag{2.8}$$

It is important to note, that the continuous wavelet transform preserves all information of $x(\cdot)$. Additionally, if $x(\cdot)$ has finite energy and if $\psi(\cdot)$ satisfies the so-called *admissibility* condition, the input signal can be reconstructed given its CWT [31, Chapter 1.2].

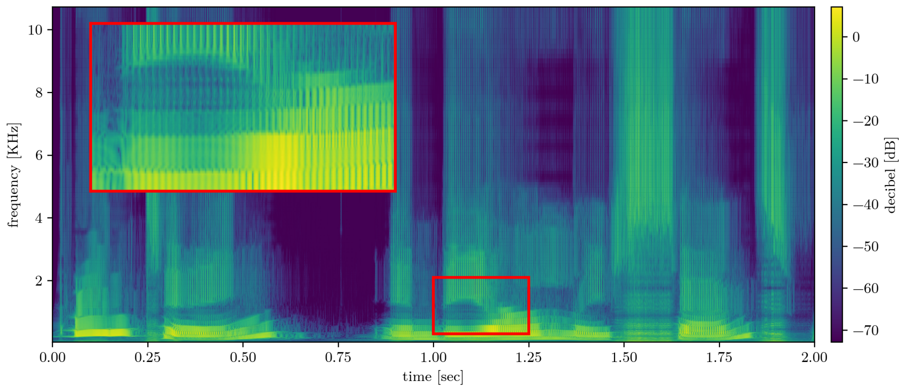


Figure 2.2: Scalogram of a two-second slice of an utterance obtained from a CWT using 256 scales and the Shannon wavelet. The large red rectangle represents a magnified view of the smaller red rectangle.

The transformation of a one-dimensional signal into a two-dimensional time-frequency description inevitably results in a considerable amount of redundancy in the output. Figure 2.2 serves to illustrate redundancies in the output of the CWT. The time-frequency description of the CWT is well-suited for analysis with the human eye. Generally, it can observed, that there are only small differences of adjacent CWT coefficients. This sparks the idea of only considering a subset of the $\mathcal{W}_x(\lambda,t)$ computed by the CWT [31, Page 12 f].

2.1.2 Discrete Wavelet Transform

As indicated, the discrete wavelet transform (DWT) can be thought of as a subsampling of the $\mathcal{W}_x(\lambda, t)$ values. Consequently, we want to restrict λ and t to discrete values only. For the scaling parameter, we naturally chose $\lambda = \lambda_0^m$, with $\lambda_0 \neq 1$ and $m \in \mathbb{Z}$. For the translation parameter, we chose $t = nt_0$, with $t_0 > 0$ and $n \in \mathbb{Z}$. Then, we modify the representation of a child wavelet such that

$$\psi_{m,n}(u) := \frac{1}{\sqrt{\lambda_0^m}}\, \psi\left(\frac{u - nt_0\lambda_0^m}{\lambda_0^m}\right). \tag{2.9}$$

Note, that the choice of λ_0 and t_0 depend on the mother wavelet $\psi(u)$. The natural choice of $\lambda_0 = 2$ and $t_0 = 1$ does not suit every wavelet [4].

However, the discrete wavelet transform can also be formulated independently of its continuous counterpart. This formulation describes an iterative approach to the computation of the DWT. For this, let us introduce filters. We define a real-valued filter $\{h_l : l = 0, \dots, L-1\}$ of length $L \in 2\mathbb{N}_{\neq 0}$ with $h_0 \neq 0$ and $h_{L-1} \neq 0$. Additionally, we define $h_l := 0\ \forall l \notin L$ such that $\{h_l\}$ is an infinite series with at most L nonzero values. If $\{h_l\}$ satisfies the following three properties, it is called a *wavelet filter*.

I) The wavelet filter sums to zero:

$$\sum_{l=0}^{L-1} h_l = 0 \tag{2.10}$$

II) The wavelet filter has unit energy:

$$\sum_{l=0}^{L-1} h_l^2 = 1 \tag{2.11}$$

III) The wavelet filter must be orthogonal to even shifts:

$$\sum_{l=0}^{L-1} h_l h_{l+2n} = 0 \quad \forall n \in \mathbb{N}_{\neq 0} \tag{2.12}$$

Note that properties (2.11) and (2.12) together are referred to as the *orthonormality property* of wavelet filters [31, Chapter 4.2].

For the remainder of this section, let us redefine the input signal $x(\cdot)$ as vector $\mathbf{X}$ of length $N = 2^J$. Moreover, for convenience, we define $N_j := N/2^j\ \forall j \in \mathbb{N}$. In addition, let us quantify the notion of *periodized filters* for future reference. Given an infinite filter $\{a_t : t = \dots, -1, 0, 1, \dots\}$, the values for the filter periodized to length N are defined by

$$a_t^\circ := \sum_{n=-\infty}^{\infty} a_{t\,+\,nN}\ , \text{ for } t = 0, \dots, N-1\,. \tag{2.13}$$

Intuitively, the filter $\{a_t\}$ is divided into blocks of N coefficients, which are then stacked and finally added to form the periodized filter [31, Chapter 2.6]. If we use a periodized filter, we always specify its length.

We will now apply the wavelet filter on $\mathbf{X}$ by iterating over $t = 0, \ldots, N_1 - 1$ and computing

$$\mathbf{W}_{1,t} := \sum_{l=0}^{L-1} h_l \, \mathbf{X}_{2t\,+\,1\,-\,l \bmod N} \,. \tag{2.14}$$

Note, that in (2.14) we take every other value of $\mathbf{X}$, which is commonly referred to as *downsampling* by two, which is also denoted by the $\downarrow 2$ operator. Analogously, the *upsampling* operator $\uparrow 2$ can be defined as a function, that inserts zeros before every value of its input [31, Chapter 4.2].
The wavelet filter output is the sequence $\{\mathbf{W}_{j,t}\}$, called *wavelet coefficients*, where $j = 1, 2, \ldots, J$ denotes the *decomposition step.* The coefficients of one decomposition step form a group, which is also referred to as j-th level coefficients. So far, we only defined the computation of the first level wavelet coefficients $\{\mathbf{W}_{1,t}\}$. In order to state the computation of the remaining ones up to level J, we introduce a second filter, called *scaling filter.*

The scaling filter $\{g_l\}$ reverses $\{h_l\}$ and then changes the sign of the wavelet filter values with even indices, formally defined as

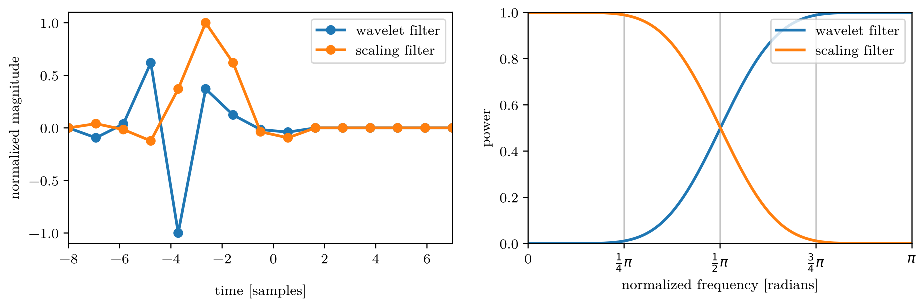
$$g_l := (-1)^{l+1} h_{L-1-l} \,. \tag{2.15}$$

Note the inverse relationship $h_l = (-1)^l g_{L-1-l}$. Additionally, it is important to note, that properties (2.11) and (2.12) are also satisfied by $\{g_l\}$. Thus the scaling filter also satisfies the orthonormality property [31, Chapter 4.3].
Similar to the computation of the first level wavelet coefficients, we can now apply the scaling filter on $\mathbf{X}$ by iterating over $t = 0, \ldots, N_1 - 1$ and computing

$$\mathbf{V}_{1,t} := \sum_{l=0}^{L-1} g_l \, \mathbf{X}_{2t\,+\,1\,-\,l \bmod N} \,. \tag{2.16}$$

The scaling filter output is the sequence $\{\mathbf{V}_{j,t}\}$, called *scaling coefficients*, with again $j = 1, 2, \ldots, J$. Observe, that the output is downsampled in the same way as in (2.14) [31, Chapter 4.2]. Before proceeding with the computation of the remaining wavelet and scaling coefficients, let us examine some properties of the wavelet filter and the scaling filter.

The representation $\{h_l\}$ is referred to as the *impulse response sequence* of the wavelet filter. Alternatively, its representation in the frequency domain is given by the *frequency response function*, denoted by $H(f)$. Furthermore, let $\mathcal{H}(f) := |H(f)|^2$ be the *squared gain function* of the wavelet filter. Analogously, we denote $G(f)$ and $\mathcal{G}(f)$ as the frequency response function and the squared gain function for the scaling filter $\{g_l\}$, respectively [31, Chapter 2.3]. The values of



(a) Impulse responses **(b)** Squared gain of frequency responses

Figure 2.3: Analysis of the filters for the Symlet (`sym4`, $L = 8$) wavelet. The plot on the left shows the impulse responses of the wavelet and scaling filters. The plot on the right shows the squared gain of their frequency responses. Both plots show the filters for the first decomposition step, that is for level $j = 1$.

$H(f)$ and $G(f)$ can be computed using an algorithm for the DFT. To illustrate, the values of $H(f)$ are given by

$$H(f) := \sum_{l=-\infty}^{\infty} h_l \, \mathrm{e}^{-i2\pi fl} = \sum_{l=0}^{L-1} h_l \, \mathrm{e}^{-i2\pi fl} \,, \tag{2.17}$$

according to equation (2.2). Note, that the equality in equation (2.17) holds because, by definition, the non-zero activity of $\{h_l\}$ is limited to indices $0 \leq l \leq L - 1$ [31, Chapter 4.2].

Figure 2.3 provides an example of the impulse and frequency responses of a wavelet and scaling filter for level $j = 1$. A closer examination of Figure 2.3a shows the inverse relationship between the wavelet and scaling filter. Of particular interest, however, is Figure 2.3b, which presents the squared gain functions of the filters' frequency responses. Here we can observe, that the wavelet filter can be regarded as an approximation to a *high-pass* filter, which preserves high frequency components while attenuating low frequency components. Conversely, we can also observe, that the scaling filter can be regarded as an approximation to a *low-pass* filter, which preserves low frequency components while attenuating high frequency components.

Generally, the *pass-band* of a filter describes the preserved frequencies. In this thesis we normalize frequencies to the interval $[-\pi, \pi]$. Due to the periodic nature of the DFT (i.e., $\mathcal{H}(-f) = \mathcal{H}(f)$), it is sufficient to analyze the interval $[0, \pi]$. Referring back to Figure 2.3b, we can observe an approximation to the pass-band $\frac{1}{2}\pi \leq |f| \leq \pi$ and $0 \leq |f| \leq \frac{1}{2}\pi$ for the wavelet filter and the scaling filter, respectively. It is important to note, that different wavelets lead to different approximations with ultimately different frequency responses. In general, it can be observed that the approximation of the aforementioned pass-bands improve with increasing filter lengths. This is the case, for example, with

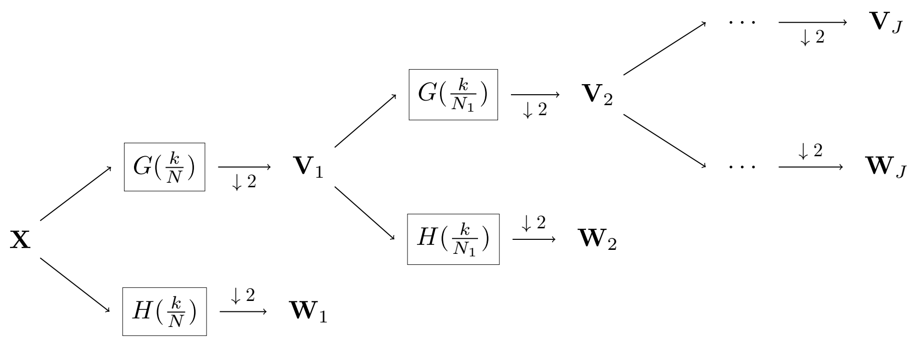


Figure 2.4: Flow diagram illustrating the analysis of $\mathbf{X}$ into the wavelet coefficients $\mathbf{W}_1$, $\mathbf{W}_2$, ..., $\mathbf{W}_J$ and the scaling coefficients $\mathbf{V}_1$, $\mathbf{V}_2$, ..., $\mathbf{V}_J$ using the discrete wavelet transform. The filtering operation of the wavelet and scaling filters is expressed as a sampling of their frequency response functions.

wavelets from the Daubechies family [31, Page 73].

The calculation of the remaining wavelet and scaling coefficients can be described as a cascade of filtering and downsampling operations, as illustrated in Figure 2.4. Input $\mathbf{X}$ is transformed into $N/2$ first level wavelet coefficients $\mathbf{W}_1$ using the wavelet filter, and into $N/2$ first level scaling coefficients $\mathbf{V}_1$ using the scaling filter. For $j = 2, \dots, J$, the j-th level transforms vector $\mathbf{V}_{j-1}$ of length N_{j-1} into the vectors $\mathbf{W}_j$ and $\mathbf{V}_j$, each of length N_j, by filtering all elements of $\mathbf{V}_{j-1}$ separately with the wavelet filter and scaling filter and then downsampling by two. Formally, j-th level wavelet and scaling coefficients are defined for $t = 0, \dots, N_j - 1$ by

$$\mathbf{W}_{j,t} := \sum_{l=0}^{L-1} h_l \, \mathbf{V}_{j-1,\, 2t+1-l \bmod N_{j-1}} \tag{2.18}$$

$$\mathbf{V}_{j,t} := \sum_{l=0}^{L-1} g_l \, \mathbf{V}_{j-1,\, 2t+1-l \bmod N_{j-1}} \, . \tag{2.19}$$

This aforementioned scheme is repeated for every subsequent level j in order to compute the wavelet and scaling coefficients up to level $j = J$ [31, Chapter 4.6].

In the flow diagrams, we express the circular filtering operations as a sampling of their frequency response functions. For all $k = 0, \dots, L-1$ and for each level $j = 1, \dots, J$, we sample the frequency response functions $H(\cdot)$ and $G(\cdot)$ at the locations $\frac{k}{N_{j-1}}$ [31, Page 33].

Figure 2.5 presents the squared gain functions of frequency responses, continuing the example provided in Figure 2.3. For the wavelet filter and the scaling filter, frequency responses of the first three decomposition steps were computed using the DFT. We can identify the pass-bands of the filters, namely $\frac{1}{2}\pi \leq |f| \leq \pi$ for the first level wavelet filter, $\frac{1}{4}\pi \leq |f| \leq \frac{1}{2}\pi$ for the second level wavelet filter,

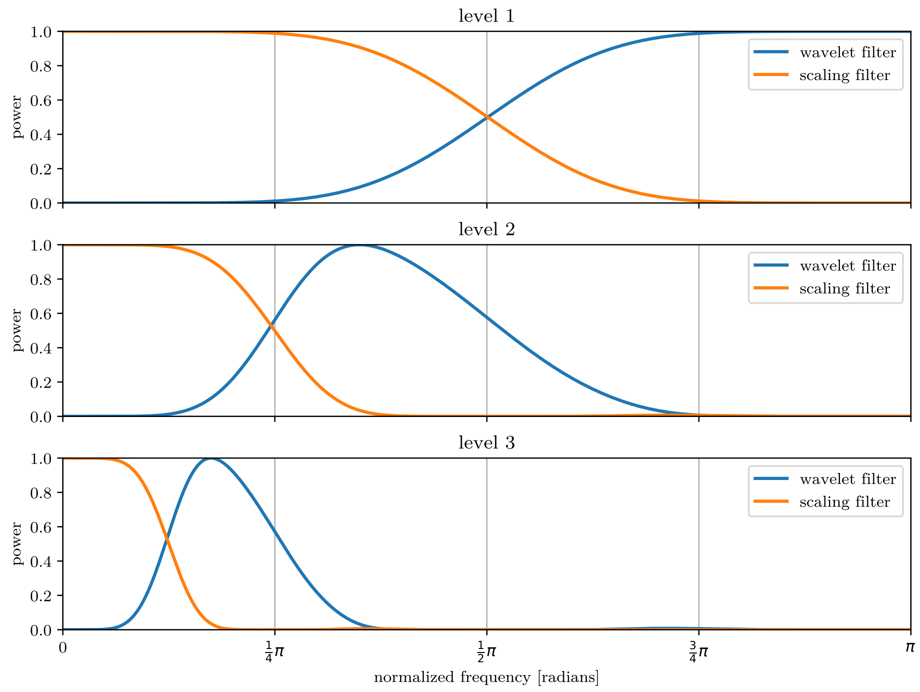


Figure 2.5: Squared gain functions of the frequency responses for the wavelet and scaling filters of the Symlet (`sym4`, $L = 8$) wavelet for the first three decomposition steps. The plot on the top represents level $j = 1$, the plot in the middle represents level $j = 2$, and the plot in the bottom represents level $j = 3$.

$\frac{1}{8}\pi \leq |f| \leq \frac{1}{4}\pi$ for the third level wavelet filter, and $0 \leq |f| \leq \frac{1}{8}\pi$ for the third level scaling filter.

Every level of the DWT can also be associated with a scale that we introduced as part of the CWT. The scale of the output from the wavelet filter at level $j = 1, \ldots, J$ is defined by $\tau_j := 2^{j-1}$. In practice, it is common to relate a scale τ_j to its physical scale $\tau_j \Delta t$ by taking into account the interval Δt between observations [31, Page 59]. In addition, let $\lambda_j := 2^j$ define the scale of the output from the scaling filter at level $j = 1, \ldots, J$. It can be argued that the j-th level wavelet coefficients are associated with changes in weighted averages over scale τ_j and that the j-th level scaling coefficients are associated with weighted averages on scale λ_j [31, Chapter 4.1]. This property is of interest from a mathematical and statistical perspective, but its intricacies will not be further discussed.

Considering the coefficients as vectors suggests that the calculation can also be expressed in matrix notation. For $j = 1, \ldots, J$, let us define two matrices $\mathcal{B}_j$ and $\mathcal{A}_j$, each of shape $N_j \times N_{j-1}$. The rows of $\mathcal{B}_j$ contain circularly shifted versions of the wavelet filter $\{h_l\}$ periodized to length N_{j-1}. Similarly, the rows of $\mathcal{A}_j$ contain circularly shifted versions of the scaling filter $\{g_l\}$ periodized to length N_{j-1}. Consequently, following the previously described scheme, we can compute the j-th level wavelet coefficients using $\mathbf{W}_j = \mathcal{B}_j \mathbf{V}_{j-1}$ and the j-th level scaling coefficients using $\mathbf{V}_j = \mathcal{A}_j \mathbf{V}_{j-1}$. Expanding these terms with $\mathbf{V}_0 := \mathbf{X}$ as follows

$$\begin{aligned}
\mathbf{W}_j &= \mathcal{B}_j \mathbf{V}_{j-1} \\
&= \mathcal{B}_j \mathcal{A}_{j-1} \mathbf{V}_{j-2} \\
&= \mathcal{B}_j \mathcal{A}_{j-1} \mathcal{A}_{j-2} \mathbf{V}_{j-3} \\
&= \mathcal{B}_j \mathcal{A}_{j-1} \mathcal{A}_{j-2} \dots \mathcal{A}_1 \mathbf{V}_0 \\
\mathbf{V}_j &= \mathcal{A}_j \mathbf{V}_{j-1} \\
&= \mathcal{A}_j \mathcal{A}_{j-1} \mathbf{V}_{j-2} \\
&= \mathcal{A}_j \mathcal{A}_{j-1} \mathbf{A}_{j-2} \mathbf{V}_{j-3} \\
&= \mathcal{A}_j \mathcal{A}_{j-1} \mathcal{A}_{j-2} \dots \mathcal{A}_1 \mathbf{V}_0
\end{aligned}$$

leads to

$$\begin{bmatrix} \mathcal{W}_1 \\ \mathcal{W}_2 \\ \vdots \\ \mathcal{W}_j \\ \vdots \\ \mathcal{W}_J \\ \mathcal{V}_J \end{bmatrix} \mathbf{X} = \begin{bmatrix} \mathcal{B}_1 \\ \mathcal{B}_2 \mathcal{A}_1 \\ \vdots \\ \mathcal{B}_j \mathcal{A}_{j-1} \dots \mathcal{A}_1 \\ \vdots \\ \mathcal{B}_J \mathcal{A}_{J-1} \dots \mathcal{A}_1 \\ \mathcal{A}_J \mathcal{A}_{J-1} \dots \mathcal{A}_1 \end{bmatrix} \mathbf{X} = \begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_2 \\ \vdots \\ \mathbf{W}_j \\ \vdots \\ \mathbf{W}_J \\ \mathbf{V}_J \end{bmatrix} = \mathbf{W}\,. \tag{2.20}$$

Here, we introduce the matrices,

$$\begin{aligned}
\mathcal{W}_j &:= \mathcal{B}_j \mathcal{A}_{j-1} \dots \mathcal{A}_1\,, \\
\mathcal{V}_J &:= \mathcal{A}_J \mathcal{A}_{J-1} \dots \mathcal{A}_1\,,
\end{aligned}$$

mainly for the sake of better readability [31].

It is important to mention, that the discrete wavelet transform is an orthonormal transform. We already stated the orthonormal property of the wavelet filter $\{h_l\}$ and the scaling filter $\{g_l\}$. Thereby, the construction of the matrices $\mathcal{B}_j$ and $\mathcal{A}_j$ implies that their respective rows are orthonormal. The orthonormality property can be stated using inner products

$$\langle \mathcal{B}_{k\bullet}, \mathcal{B}_{k'\bullet} \rangle = \begin{cases} 1, & \text{if } k = k' \\ 0, & \text{otherwise} \end{cases} \,. \tag{2.21}$$

Here, $\mathcal{B}_{k\bullet}$ denotes the k-th row vector of $\mathcal{B}$. However, the orthonormality property has to hold also for the column vectors $\mathcal{B}_{\bullet k}$ of $\mathcal{B}$. The orthonormality can be shown for both matrices $\mathcal{B}_j$ and $\mathcal{A}_j$. Generally, an input can easily be reconstructed from its orthonormal transform. Hence, the input and its transform can be considered to be two representations of the same mathematical entity [31, Chapter 3].

The fact, that the DWT is an orthonormal transform, has remarkable mathematical implications, which enable a wide range of applications. Besides the important property that any input $\mathbf{X}$ can be reconstructed given the complete coefficient matrix $\mathbf{W}$, the DWT also preserves the energy its input, because every orthonormal transform preserves the energy of the input. In addition, its coefficients enable the *energy decomposition*, formally

$$\|\mathbf{X}\|^2 = \sum_{j=1}^{J} \|\mathbf{W}_j\|^2 + \|\mathbf{V}_J\|^2 \,, \tag{2.22}$$

which partitions the energy of $\mathbf{X}$ into segments associated with different scales and times [31]. The energy decomposition is very similar to the statistical technique called analysis of variance (ANOVA) [31, Page 19].

Furthermore, the wavelet and scaling coefficients of the DWT can be used in the *multiresolution analysis* (MRA). We define vectors $\mathcal{D}_j := \mathcal{W}_j^T \mathbf{W}_j$ and $\mathcal{S}_j := \mathcal{V}_j^T \mathbf{V}_j$ for every level $j = 1, \ldots, J$ of the DWT. Input $\mathbf{X}$ can be expressed as

$$\mathbf{X} = \sum_{j=1}^{J} \mathcal{D}_j + \mathcal{S}_J \,. \tag{2.23}$$

We know that $\mathbf{W}_j = \mathcal{W}_j \mathbf{X}$ represents the portion of the analysis $\mathbf{W} = \mathcal{W}\mathbf{X}$ attributable to scale τ_j. Conversely, $\mathcal{W}_j^T \mathbf{W}_j$ represents the portion of the synthesis (reconstruction) $\mathbf{X} = \mathcal{W}^T \mathbf{W}$ attributable to scale τ_j. Intuitively, the values in $\mathcal{D}_j$ are associated with changes in $\mathbf{X}$ at scale τ_j and the values in $\mathcal{S}_j$ are associated with the averages at scale λ_j. That is why the vectors $\mathcal{D}_j$ and $\mathcal{S}_j$ are also referred to as j-th level *wavelet detail* and *wavelet smooth*, respectively [31, Chapter 4.1].

So far, we assumed a sample size $N = 2^J$ and studied the calculation of the DWT coefficients $\mathbf{W}$ up to level $j = J$. However, it is also possible to stop the decomposition after $J_0 < J$ steps. This practice is commonly referred to as *partial* DWT. The result of a J_0-level partial DWT is closely related to the coefficients of $\mathbf{W}$. Precisely, all $\mathbf{W}_j$ for $1 \leq j \leq J_0$ are subvectors of $\mathbf{W}$, while the scaling coefficients $\mathbf{V}_{J_0}$ simply replace the last $N/2^{J_0}$ coefficients of $\mathbf{W}$. In practice, the partial DWT offers the flexibility to not further explore large scales [31, Chapter 4.7].

2.1.3 Discrete Wavelet Packet Transform

The previous section presented the DWT, which decomposes an input signal into wavelet and scaling coefficients. However, the wavelet coefficients $\mathbf{W}_j$ for some level j are not further analyzed by the DWT, as illustrated by the filtering scheme in Figure 2.4. Thus, the analysis of high frequency components of a signal stops once it has been high-passed using the wavelet filter. This section presents the discrete wavelet packet transform (DWPT), which can be considered as an extension of the DWT.

The discrete wavelet packet transform recursively filters its coefficients of the previous level. For every level $j = 0, \dots, J$, let $\mathbf{W}_{j,n}$ denote the DWPT coefficient vector of level j indexed by $n = 0, \dots, 2^{j-1} - 1$. Furthermore, let $\mathbf{W}_{j-1,n,t}$ denote the t-th element of $\mathbf{W}_{j,n}$. In order to recursively compute the $\mathbf{W}_{j,n}$ given the previous coefficients, we require a formula for selecting the correct filter given some level j and some index n. Formally, we define new filter values

$$a_{n,l} := \begin{cases} g_l, & \text{if } n \text{ is even} \\ h_l, & \text{if } n \text{ is odd} \end{cases} \quad \text{and} \quad b_{n,l} := \begin{cases} h_l, & \text{if } n \text{ is even} \\ g_l, & \text{if } n \text{ is odd} \end{cases} . \tag{2.24}$$

Now, the j-th level DWPT coefficients at index n are defined by iterating over $t = 0, \dots, N_j - 1$ and computing

$$\begin{aligned} \mathbf{W}_{j,2n,t} &:= \sum_{l=0}^{L-1} a_{n,l} \mathbf{W}_{j-1,\, n,\, 2t+1-\, l \bmod N_{j-1}} \,, \\ \mathbf{W}_{j,2n+1,t} &:= \sum_{l=0}^{L-1} b_{n,l} \mathbf{W}_{j-1,\, n,\, 2t+1-\, l \bmod N_{j-1}} \,. \end{aligned} \tag{2.25}$$

Likewise, the computation of the DWPT coefficients can also be expressed using the previously defined matrices $\mathcal{B}_j$ and $\mathcal{A}_j$:

$$\begin{aligned} \mathbf{W}_{j,2n,t} &:= \begin{cases} \mathcal{A}_j \mathbf{W}_{j-1,n}, & \text{if } n \text{ is even} \\ \mathcal{B}_j \mathbf{W}_{j-1,n}, & \text{if } n \text{ is odd} \end{cases} , \\ \mathbf{W}_{j,2n+1,t} &:= \begin{cases} \mathcal{B}_j \mathbf{W}_{j-1,n}, & \text{if } n \text{ is even} \\ \mathcal{A}_j \mathbf{W}_{j-1,n}, & \text{if } n \text{ is odd} \end{cases} . \end{aligned} \tag{2.26}$$

In words, if n in $\mathbf{W}_{j-1,n}$ is even, we use the scaling filter to obtain $\mathbf{W}_{j,2n}$ and the wavelet filter to obtain $\mathbf{W}_{j,2n+1}$. And vice versa, if n in $\mathbf{W}_{j-1,n}$ is odd, we use the wavelet filter to obtain $\mathbf{W}_{j,2n}$ and the scaling filter to obtain $\mathbf{W}_{j,2n+1}$. We can illustrate this pattern in a *wavelet packet tree*, shown in Figure 2.6. Each level j of the DWPT decomposes the frequency spectrum into 2^j individual intervals of equal size. Moreover, each DWPT coefficient can be assigned to a specific band of frequencies and a particular interval of time, similar to the time-frequency localization in the STFT [31, Chapter 6.1].

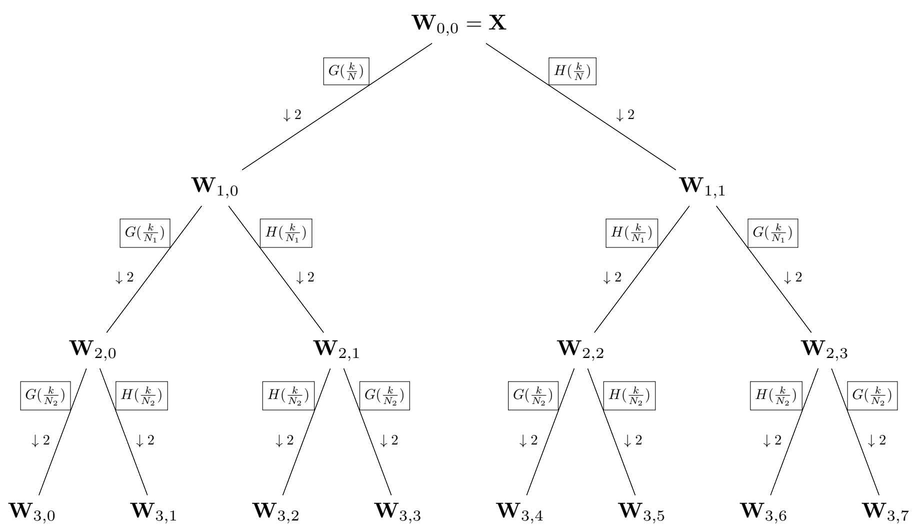


Figure 2.6: Flow diagram illustrating the analysis of $\mathbf{X}$ using the wavelet packet transform up to level $j = 3$. The ordering of the coefficient vectors at each level j represent the so-called frequency ordering, where index $n = 0, \ldots, 2^j - 1$ reflects the ordering of the decomposed frequency bands.

While Figure 2.6 shows an complete wavelet packet transform up to level $j = 3$, it is also possible compute a only subset of coefficients. For instance, computing the coefficient vectors $\mathbf{W}_{1,1}$, $\mathbf{W}_{2,1}$, $\mathbf{W}_{3,1}$, and $\mathbf{W}_{3,0}$ would be equivalent to computing a 3-level DWT. On the other hand, if we were particularly interested in analyzing the upper frequencies of a signal, for example, we could only compute coefficient vectors $\mathbf{W}_{1,0}$, $\mathbf{W}_{2,3}$, $\mathbf{W}_{3,4}$, and $\mathbf{W}_{3,5}$. It is convenient to identify nodes in the wavelet packet tree as tuples from the family $\mathcal{N} := \{(j, n) : j = 1, \ldots, J_0\,; n = 0, \ldots, 2^j - 1\}$ with $J_0 \leq J$.

The decomposition of any $\mathbf{W}_{j-1,n}$ into $\mathbf{W}_{j,2n}$ and $\mathbf{W}_{j,2n+1}$ is an orthonormal transform. This can be shown, for instance in equation (2.26), where $\mathcal{B}_j$ and $\mathcal{A}_j$, by definition, both satisfy the orthonormality property. In order to make a more general statement about orthonormal transforms in the DWPT, we introduce *disjoint dyadic decompositions*. If for each node (j, n) in the wavelet packet tree, we either decompose it into two child nodes $(j+1, 2n)$ and $(j+1, 2n+1)$ or we do not further decompose it, then the result is a disjoint dyadic decomposition. Ultimately, it can be shown, that any disjoint dyadic decomposition extracted from a wavelet packet tree yields an orthonormal transform [31, Chapter 6.1].

2.1.4 Stationary Wavelet Transform

The stationary wavelet transform (SWT) presents an interesting alternative to the previously discussed transforms. From the previous sections we know that the wavelet coefficients can be regarded as changes in weighted averages associated to a specific scale and that the scaling coefficients can be regarded as weighted averages associated to a specific scale. It is important to recognize, that the intervals over which these averages are computed are fixed a priori by the start point and the length of the input signal. Therefore, the averages may not align well with interesting features of the input. Furthermore, small translations of the input signal can yield quite different wavelet coefficients and thereby also different averages. These considerations serve as the primary motivation for the SWT [31, Chapter 5.1].

The SWT attempts to address the issues associated with the a priori fixed filter positions. Intuitively, this can be achieved by avoiding the downsampling of the filter output and thus calculating coefficients for all possible filter positions. This motivates the key difference between the SWT and the DWT: The coefficients are not downscaled. Therefore, given some input length N, every level of the SWT yields coefficient vectors of length N [31, Chapter 5.3]. Nevertheless, this approach also entails a greater computational effort, as the coefficients for all possible filter positions must be computed [31, Chapter 5.0].

In order to avoid confusion with preceding definitions, we attach a tilde sign to definitions related to the SWT. Let the j-th level wavelet and scaling coefficients be denoted by vectors $\tilde{\mathbf{W}}_j$ and $\tilde{\mathbf{V}}_j$, respectively. Similar to the previous sections, we denote $\tilde{\mathbf{W}}_{j,t}$ and $\tilde{\mathbf{V}}_{j,t}$ as the t-th element of $\tilde{\mathbf{W}}_j$ and $\tilde{\mathbf{V}}_j$, respectively. We will define the computation of $\tilde{\mathbf{W}}_j$ and $\tilde{\mathbf{V}}_j$ by applying the corresponding filters on input $\mathbf{X}$. This is different from the previous sections, where we used the coefficient vectors of level $j-1$.
For this, we first redefine the wavelet and scaling filters, such that they relate $\tilde{\mathbf{W}}_j$ and $\tilde{\mathbf{V}}_j$ directly to $\mathbf{X}$. The wavelet filter for level j, denoted by $\{h_{j,l}\}$, is constructed by inserting $2^{j-1}-1$ zeros between the filter values of $\{h_l\}$. The scaling filter for level j, denoted by $\{g_{j,l}\}$, is constructed analogously [31, Page 59 f]. Note, that for some level j, both filters have width $L_j := (2^j-1)(L-1)+1$. Let $\{\tilde{h}_{j,l}\}$ be defined such that $\tilde{h}_{j,l} := h_{j,l}/2^{j/2}$ and let $\{\tilde{g}_{j,l}\}$ be defined such that $\tilde{g}_{j,l} := g_{j,l}/2^{j/2}$. Then the j-th level coefficients are given by iterating over $t = 0, \dots, N-1$ and computing

$$\begin{aligned} \tilde{\mathbf{W}}_{j,t} &:= \sum_{l=0}^{L_j-1} \tilde{h}_{j,l} \mathbf{X}_{t-l \bmod N} \, , \\ \tilde{\mathbf{V}}_{j,t} &:= \sum_{l=0}^{L_j-1} \tilde{g}_{j,l} \mathbf{X}_{t-l \bmod N} \, . \end{aligned} \tag{2.27}$$

The outputs are not downscaled, which yields N wavelet and scaling coefficients for every level j [31, Chapter 5.4]. This results in a highly redundant representation [31, Chapter 5.0].

Unlike the DWT, the SWT is not an orthonormal transform. Nevertheless, it is still possible to employ the SWT coefficients in a multiresolution analysis and a energy decomposition. It can be shown, that the SWT filters have zero phase properties, which enable the alignment of events in $\mathbf{X}$ with events in the detail and smooth vectors of the corresponding MRA. In addition, circularly shifting the input also circularly shifts the values of the detail and smooth vectors [31, Chapter 5.12]. This is why the SWT is also referred to as a shift-invariant or time-invariant DWT [31, Chapter 5.0]. Another useful property of the SWT, is that the coefficients are defined for any input size N [31, Chapter 5.4].

Typically, a decomposition using the SWT follows a filtering scheme, similar to the one of the DWT. However, it is also possible to extend the idea of the SWT to the DWPT. This enables the DWPT to inherit the aforementioned properties [31, Chapter 6.6].

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs), famously introduced by LeCun et al. in 1989 [23], specialize neural networks to work exceptionally well with hierarchical grid-structured data, such as images. CNNs extend the concept of neural feedforward networks, inherit their layered structure and are also trained with the well-known backpropagation algorithm. The distinguishing feature of CNNs is their ability to efficiently capture spatial hierarchies and patterns through convolutional and pooling layers [23, 10].

CNNs were among the first deep neural networks to perform well and were also some of the first neural networks to be deployed in commercial applications. Since their introduction, CNNs continued to evolve, various different architectures have been presented, and significant improvements regarding scalability and representational capacity have been achieved [10, Chapter 9.11].
In 2012, Krizhevsky et al. won the popular ImageNet challenge using a CNN-based architecture, marking a significant breakthrough for CNNs. Their proposed model demonstrated best performance on both image classification and single-object localization tasks [20, 34].

As with feedforward networks, CNNs are typically described by dividing them into layers, with each layer successively transforming its input units into output units. A convolutional layer performs a special form of transformation that serves as the foundation of CNNs. Figure 2.7 depicts the key components of CNNs, which will be explained in the following subsections.

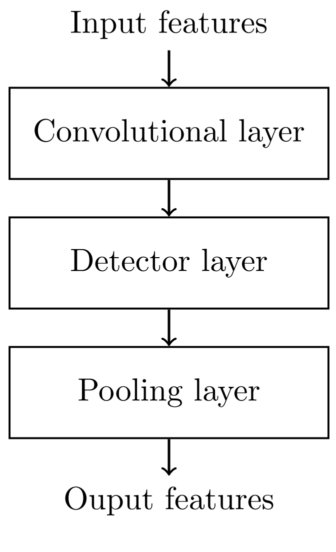


Figure 2.7: Components of a convolutional neural network.

2.2.1 Convolutional Layer

Convolution, in general terms, describes an operation using functions $x(\cdot)$ and $w(\cdot)$ to produce a third function, denoted by $(x * w)(\cdot)$. Over a continuous space it is defined by $(x*w)(t) := \int x(a)w(t-a)\,da$. Similarly, the discrete convolution is defined by $(x*w)(t) := \sum x(a)w(t-a)$. In the context of CNNs, however, we use a slightly different notion, as their inputs are typically of two-dimensional and discrete nature. Let $\mathbf{I} \in \mathbb{R}^2$ be a two-dimensional input of shape $I \times J \in \mathbb{N} \times \mathbb{N}$, such as an image and let $\mathbf{K} \in \mathbb{R}^2$ be a two-dimensional parameterized filter of shape $M \times N \in \mathbb{N} \times \mathbb{N}$. Note, that $\mathbf{I}$ and $\mathbf{K}$ are commonly called *input tensor* and *kernel*, respectively. Let a tuple $(x,\, y) \in \{(x,\, y) : x, y \in \mathbb{N}\}$ denote an arbitrary position. We can consider $\mathbf{I}$ and $\mathbf{K}$ as functions where we use position $(x,\, y)$ to query the value of their finite two-dimensional set of values. We assume zero values for all undefined positions $(x,\, y)$. For positions (i, j), the convolution of $\mathbf{I}$ with $\mathbf{K}$ is defined as

$$(\mathbf{I} * \mathbf{K})(i,j) := \sum_m \sum_n \mathbf{I}(i-m, j-n)\mathbf{K}(m,n)\ . \tag{2.28}$$

In practice, many neural network libraries implement the convolution of $\mathbf{I}$ with $\mathbf{K}$ by computing

$$(\mathbf{I} * \mathbf{K})(i,j) := \sum_m \sum_n \mathbf{I}(i+m, j+n)\mathbf{K}(m,n)\ , \tag{2.29}$$

which is a related function known as *cross-correlation* [10, Chapter 9.1].

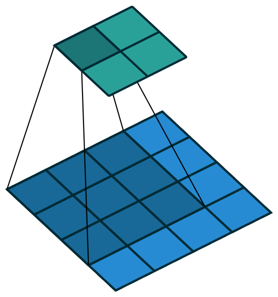
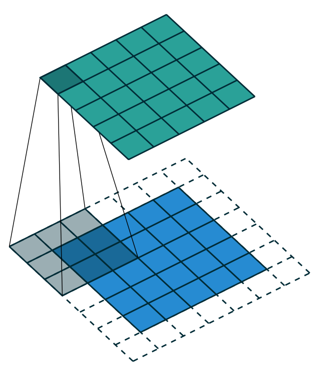


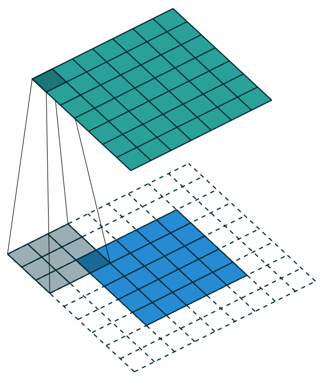
Figure 2.8: Illustration of a two-dimensional convolution with no padding. The convolution of an input tensor (blue) of shape 4×4 with a kernel (dark blue) of shape 3×3 results in an output tensor (cyan) of shape 2×2. This illustration is taken from "A guide to convolution arithmetic for deep learning" [5].

Figure 2.8 depicts a two-dimensional convolution. Here, the convolution effectively reduces the spatial sizes of the output. More precisely, a kernel of shape $M \times N$, that is only allowed to visit positions (i, j) such that position $(i-m, j-n)$ is defined on $\mathbf{I}$, reduces the output to size $I-M+1 \times J-N+1$. This kind of convolution is sometimes called *valid* convolution. The consequence of valid convolutions in multiple convolutional layers is that the output size is successively reduced, which limits the number of convolutional layers that can be stacked in the network. Although the choice of small kernel sizes mitigates this effect, overall the expressive power of the network is significantly constrained [10, Chapter 9.5]. Moreover, it can be observed that that positions near the border of the input are visited less often than positions near the center.

The aforementioned side effects can be alleviated by using *padding*, which is defined as adding zeros around the input. This effectively lets us control the size of the output without modifying the shape of the kernel. One common padding technique involves adding exactly as many zeros around the input tensor as are needed to produce an output of equal size. This, however, still may under-represent features near the border, because elements near the border have less influence on the kernel compared to elements near the center of the input.
Another padding technique involves adding enough zeros to ensure that every element in the input is visited equally many times. This enlarges the output size to $I + M - 1 \times J + N - 1$ [10, Chapter 9.5]. Both padding techniques are illustrated in Figure 2.9.



(a) same padding



(b) full padding

Figure 2.9: Illustration of convolutions using two different padding techniques. The transparent squares with dashed borders indicate the padding values, that have been added to the input tensor (blue). The illustrations are taken from "A guide to convolution arithmetic for deep learning" [5].

A convolutional layer offers significant advantages compared to traditional fully-connected layers, in particular *sparse interactions*, *parameter sharing* and *equivariant representations*. According to equation (2.28), it is clear that multiple neighboring input units affect the output of the convolution at position (i, j). These neighboring input units are also referred to as the *receptive field* of the output units. Conversely, multiple neighboring output units are affected by the same input units. This property is known as sparse interactions and occurs when the kernel size is smaller than the input size. In contrast, in a fully-connected layer, an output units is affected by all input units. Sparse interactions in a convolutional layer result in fewer parameters, which in turn requires fewer operations for computing the output and reduces memory requirements. Besides that, it is also worth mentioning, that even though direct connections in a single convolutional layer are very sparse, units in deeper convolutional layers can be indirectly connected with a large portion of the input. This enables the network to efficiently describe complex interactions [10, Chapter 9.2].

Every kernel value can be regarded as an individual parameter. When computing the output of a convolutional layer, each parameter is used at every position of the input, except perhaps positions near the border depending on the padding used. This concept is commonly referred to as parameter sharing. After the calculation of the output, the parameters of the kernel will be adjusted. Parameter sharing in a convolutional layer results in an additional property, called equivariance to translation. Intuitively, this property states that a translated input to a convolutional layer yields a equally translated output. However, convolution is not naturally equivariant to other transformations, such as changes in scale or rotation [10, Chapter 9.2].

As part of the research in CNNs, additional variants of convolutional layers have been developed. The *strided convolution* skips positions of the input during convolution, instead of computing the convolution over all positions of the input. The result can be seen as downsampling the output of the full convolution. Note, that it is also possible to define different strides for the different dimensions of the input [10, Chapter 9.5].

The *dilated convolution* increases the receptive field of output units without changing the kernel size and thus without increasing the number of parameters in the kernel. In practice, this is achieved by modifying the convolution operator to apply the kernel to a different range of the input. Intuitively, this can be thought of as inserting spaces between the kernel values. Note, that similar to the strided convolution, it is also possible to define different spacings for the different dimensions of the kernel. An increase in the kernel size in order to increase the receptive field results in a loss of resolution. In contrast, dilated convolutions do not result in a loss of resolution. Moreover, dilated convolutions exponentially increase the receptive field of the network while the number of parameters grows linearly [42, 5]. Both variations are illustrated in Figure 2.10.

Furthermore, the *tiled convolution* first divided the input into smaller subsections, called tiles, before performing a convolution operation over the tiles using a set of independent kernels. This offers a compromise between a fully-connected layer and a convolutional layer [10, Chapter 9.5].

We will conclude this subsection by briefly discussing an important practical aspect of convolutional layers, namely their use of *multichannel convolutions*. It is common to regard an image as a 3-dimensional input tensor, where the first

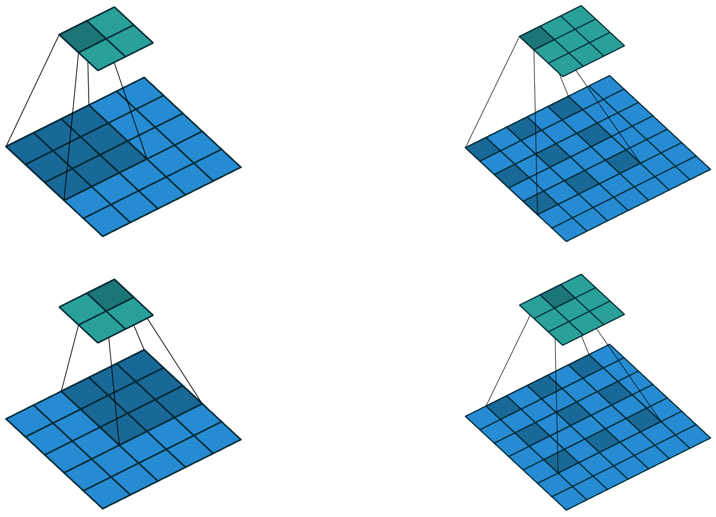

(a) strided convolution **(b)** dilated convolution

Figure 2.10: Illustration of strided and dilated convolutions. The left column demonstrates a strided convolution where the kernel of shape 3×3 skips one valid position of the 5×5 input. The right column demonstrates a dilated convolution, where the receptive field of an output unit is increased by 2 in both dimensions. All illustrations are taken from "A guide to convolution arithmetic for deep learning" [5].

dimension defines the color channel and the remaining two dimensions define the width and height of the image. A convolutional layer then transforms the input with C_i channels into an output with C_o channels by using a 4-dimensional kernel. The first two dimensions of such an kernel represent the strength of the connection between the respective input and output channels. The remaining two dimensions represent the row and column offset between the input and output units, as defined in equation (2.28) [10, Chapter 9.5].

2.2.2 Detector Layer

The output units returned from a convolutional layer are linear combinations of the input units and the parametrized kernel, which can be observed in equation (2.28). This is why the output units of the convolutional layers are also referred to as linear activations. In order to introduce nonlinearity between layers, the output units are passed into a nonlinear activation function. This extends a linear model to represent nonlinear functions [10, Chapter 6.0] and causes the loss function to become nonconvex [10, Chapter 6.2].

It is worthwhile to have a closer look at the various activation functions that are commonly used in practice. Figure 2.11 depicts the rectified linear unit (ReLU), the LeakyReLU, the hyperbolic tangent, and the Sigmoid function.

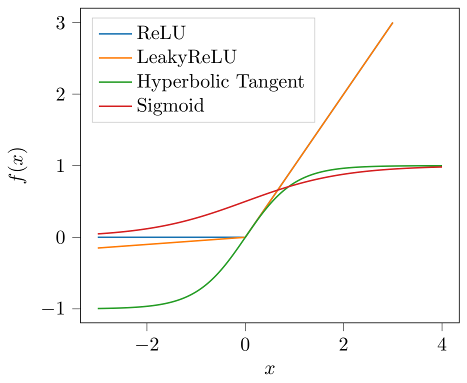


Figure 2.11: Popular activation functions. The ReLU and its variations are commonly used in detector layers of CNNs. The hyperbolic tangent and Sigmoid functions are primarily used in the output layers.

2.2.3 Pooling Layer

The final central component of CNNs is the pooling layer, which uses a pooling function to further modify the output units. More precisely, the pooling function replaces the features at a certain location with a summary statistic of nearby features. This process further reduces the number of output units resulting in a more compact representation. Furthermore, it is easy to see that this helps to make the output units almost invariant to small translations of the input. This is especially useful if we are more concerned with the presence of certain features rather than their specific locations. Additionally, pooling layers are commonly used for handling inputs of varying size [10, Chapter 9.3].

There are numerous types of pooling functions and it is also straightforward to create new ones. Commonly used pooling functions include average pooling and max pooling. Some theoretical work provides guidance as to which types of pooling should be used in different scenarios [10, Chapter 9.3].

2.2.4 Residual Learning

Empirical evidence indicates that an increase in learnable parameters of a network is not sufficient to enhance performance. Rather, the number of layers in a network must also be increased. A deep neural network is capable of learning a greater variety of functions. Assuming a function can be described using many simpler functions, the intuition behind a deep network is that more layers help to represent parts of the complex function. In addition, the existing literature suggests that greater depth of neural networks appears to result in better generalization [10, Chapter 6.4.1]. However, the introduction of additional layers to a neural network has also been observed to present drawbacks, which will now be discussed in detail.

Deep convolutional networks have enabled important advances in image classification, but have proven to be more difficult to train. This is attributable to the fact that simply stacking more additional layers introduces several problems. While the use of normalized initialization and normalization layers was effective in addressing the problem of vanishing and exploding gradients, the issue of degradation remained unresolved. Residual learning addresses the degradation problem. Its concept was first introduced by He et al. in 2016 in their well-known research paper "Deep Residual Learning for Image Recognition" [14].

As the depth of a network increases, its accuracy gets saturated. This is to be expected, given that a more complex model is more likely to be able to learn all the intricacies of the training data. However, at a certain point during training, the accuracy of the deep network degrades rapidly, which is commonly known as the *degradation problem*. It is important to note, that this is not caused by overfitting, because the training accuracy also degrades. The residual learning approach successfully addresses the degradation problem and proves to simplify the training of deep neural networks [14].

Let $H(x)$ be the desired underlying mapping to be fitted by a few stacked layers, with x denoting the inputs to the first of these layers. Instead of letting these layers to directly approximate $H(x)$, the residual learning approach lets these layers to approximate a residual function $F(x) := H(x) - x$. Thus, the approximated output function becomes $F(x)+x$. It is possible to asymptotically approximate both representations $H(x)$ and $F(x) + x$, provided that multiple non-linear layers can approximate arbitrarily complex functions. However, the ease of learning is very different, as shown by the researchers [14].

He et al. proposed two building blocks for residual learning, namely the `basic` and the `bottleneck` block. Nevertheless, they also emphasized that the form of the residual function is flexible, but should at least comprise two or more layers. The `basic` block consists of two convolutional layers, both with kernel size k. In contrast, the `bottleneck` block comprises three convolutional layers, where the layers with kernel size 1 are responsible for reducing and restoring dimensions, leaving the layer with kernel size k as the bottleneck with smaller input and output dimensions. Every convolutional layers is followed by a batch normalization layer. Additionally, detector layers are placed between a batch normalization layer and a convolutional layer [14]. An illustration of both building blocks can be found in Figure 2.12.

To simplify the notation we define both building blocks as two separate functions:

$$\begin{gathered}\texttt{basic}(i,\, o,\, e,\, k,\, s,\, d)\,, \text{ and}\\ \texttt{bottleneck}(i,\, o,\, b,\, e,\, k,\, s,\, d)\,, \text{ with}\end{gathered}$$

$$\begin{aligned} i &= \text{input channels}\,,\\ o &= \text{output channels}\,,\\ e &= \text{expansion}\,,\\ b &= \text{bottleneck}\,,\\ k &= \text{kernel size}\,,\\ s &= \text{stride}\,,\\ d &= \text{dilation}\,. \end{aligned} \tag{2.30}$$

The standard implementation involves zero-padding both dimensions such that the output size matches exactly the input size. Furthermore, the shortcut connection involves an identity mapping, if applicable. Otherwise, a convolutional layer with a kernel size of 1 is typically used. It is crucial to acknowledge that the utilization of identity mappings offers great advantages in practice. Identity mappings do not introduce additional parameters or an increase in computational complexity [14]. This should be taken into account when designing residual networks.

He et al. developed a popular family of deep convolutional networks, known as ResNet, which leverage the benefits of residual learning. More precisely, they developed five ResNet variants with different depths ranging from 18 up to 152 convolutional layers. All networks can be divided into five groups of convolutional layers and one final fully connected layer. The first group comprises a single convolutional layer and the remaining groups are composed of varying numbers of `basic` or `bottleneck` blocks [14].

The ResNet models were originally designed to classify images and were trained on the ImageNet dataset to classify images into 1000 different categories. In 2015, the researchers won the ImageNet challenge using an ensemble of their ResNet models. Their submission report a 3.57% top-5 error rate for the classification task on the test set [14].

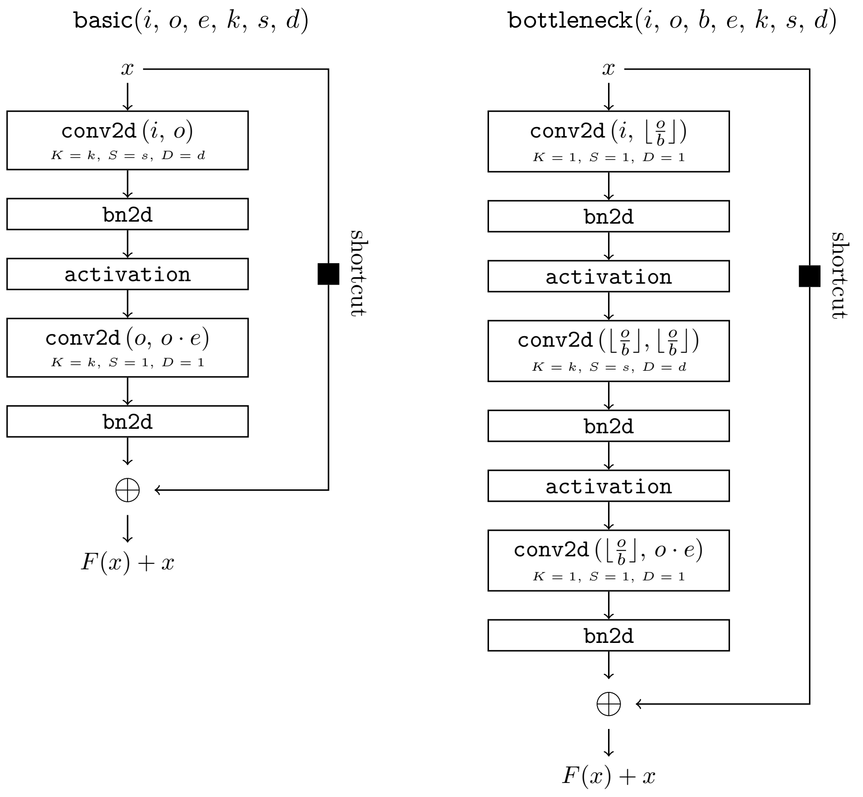

basic(i, o, e, k, s, d)
x
conv2d (i, o)
$K = k,\ S = s,\ D = d$
bn2d
activation
conv2d (o, $o \cdot e$)
$K = k,\ S = 1,\ D = 1$
bn2d
shortcut
$\oplus$
$F(x) + x$
bottleneck(i, o, b, e, k, s, d)
x
conv2d (i, $\lfloor \frac{o}{b} \rfloor$)
$K = 1,\ S = 1,\ D = 1$
bn2d
activation
conv2d ($\lfloor \frac{o}{b} \rfloor$, $\lfloor \frac{o}{b} \rfloor$)
$K = k,\ S = s,\ D = d$
bn2d
activation
conv2d ($\lfloor \frac{o}{b} \rfloor$, $o \cdot e$)
$K = 1,\ S = 1,\ D = 1$
bn2d
shortcut
$\oplus$
$F(x) + x$

Figure 2.12: Illustration of the residual building blocks utilized in the ResNet architectures [14]. The black square denotes the shortcut connection, ideally an identity mapping. Furthermore, `conv2d` denotes a two-dimensional convolutional layer, `bn2` denotes a two-dimensional batch normalization layer, and `activation` denotes an arbitrary detector layer.

Chapter 3

Experiments

We will now proceed to the practical part, namely the design and analysis of neural network based detectors for audio deepfakes. Let us begin by formally stating the problem. The objective is to determine whether an audio recording of spoken language is authentic or not. An authentic recording is defined as one that reflects an utterance of a real person. If a recording is not authentic, then the utterance was synthesized by some generator. We have already briefly mentioned generators at the beginning of this thesis, but in order to better address the given problem, generators need to be discussed in greater detail. In recent years, research into text-to-speech synthesis has made impressive progress. This has led to the emergence of numerous text-to-speech generators.

For generative tasks such as waveform synthesis, Generative Adversarial Networks (GANs) are a popular choice of network architecture. Due to dependencies at different time scales in audio data and its high temporal resolution, modelling raw audio waveforms is considered a challenging task. Therefore, lower resolution intermediate representations are often used. Thus, synthesis typically involves two steps using two distinct models. The first model creates an intermediate representation based on a given some text input, and the second model transforms the intermediate representation back into audio. This second model is also known as the *vocoder*. In the context of speech synthesis, commonly used intermediate representations include aligned linguistic features and mel-spectrograms [21].

One of the early models to produce high-quality audio was the MelGAN [21]. Both models in the GAN framework, namely the generator and the discriminator use a non-autoregressive convolutional architecture. This kind of architecture reduces the number of learnable parameters and ultimately enables very fast inference, compared to autoregressive models [21]. In a subsequent research project, MelGAN's capabilities were further enhanced [41]. It is also important to mention research in autoregressive models such as WaveRNN [17]. Besides that, flow-based models, such as WaveGlow [32] have also been proposed. The most recent generation of GAN-based models, such as BigVGAN [25] and Avocodo [3] reflect the current state of the art in generative modeling of audio waveforms.

The availability of many new generators has also led to a surge of interest in their detection. One important driving force is the ASVspoof challenge [38, 26], of which there have already been four editions. Automatic speaker verification (ASV) is used in biometric systems to verify an individual's identity based on their voice. Such systems are vulnerable to spoofing attacks, where a malicious actor deceives the system into falsely recognizing them as an authorized user. The ASVspoof challenge sets the task of developing countermeasures to spoofing attacks using synthetic, converted, and replayed speech [38, 26]. It is evident that the mentioned generator networks have the requisite capabilities for such spoofing attacks.

The contestants have thus far proposed many promising detector models, which do not only differ in terms of their network architecture, but also use a variety of different feature extraction methods. Generally, feature extraction in the context of audio data involves transforming the raw audio signals into meaningful characteristics [22]. For example, the STFT and the wavelet transforms, which we discussed in Section 2.1, are also methods for feature extraction. The latest editions of ASVspoof provided three baseline models. First, RawNet2 [16], a combination convolutional layers, residual blocks and gated recurrent units. As the name suggests, RawNet2 processes raw audio data. Second, two variants of a Gaussian mixture model (GMM) with different feature extraction approaches. We focus on the model that employs the linear frequency cepstral coefficient (LFCC) [35]. And finally, the Light CNN (LCNN), a five layer convolutional neural network that is characterized by the use of so-called Max-Feature-Maps as detector layers. The LCNN processes Fourier transform coefficients [22]. We will also use these models as baseline to our evaluation.

The primary objective of this thesis is to develop a working detector that utilizes the stationary wavelet transform. Subsequently, this detector will be compared to existing detectors and the utility of the stationary wavelet transform will be evaluated. We will begin by outlining the experimental setup and a detailed overview of the dataset subjected to our experiments. We will then look at the fingerprints of generators. This will be followed by numerous experiments that analyze and compare different detectors.

3.1 Setup

The following three subsections describe the setup of our experiments. It is crucial to provide a meticulous and comprehensive description of the details associated with experiments, in order to guarantee the reproducibility of the results. These include the hardware configuration, implementation details, and most importantly, the configuration of the numerous hyperparameters associated with deep learning.

3.1.1 Environment

All models were trained on a single node of the JUWELS Booster cluster at the Jülich Supercomputing Centre (JSC), configured with two AMD EPYC 7402 CPUs, a total of 512GB main memory, and four NVIDIA A100-SXM4 GPUs with 40GB RAM each. A single model was always trained on a single GPU. However, four models were always trained simultaneously on the four available

GPUs to maximize resource utilization. There is no node-sharing at JSC, which means that no other jobs were running on a node during the training process.

3.1.2 Implementation

All models and their associated training code have been implemented in Python using PyTorch [30] (version 2.1.2). We use the implementation of PyWavelets [24] (version 1.5.0) for the SWT and for every other wavelet transform. Alternatively, we use the implementation of wavelet transforms provided in the PyTorch-Wavelet-Toolbox [39], which markedly accelerates the computation when carried out on the GPU.

3.1.3 Configuration

This section defines the default parameters of a training run. If any parameter differs for a specific training process a model, it will explicitly be noted in the corresponding subsection. We use the cross entropy loss function weighted by the class frequencies in the training set and the Adam optimizer with a learning rate of $3{\times}10^{-4}$. We train the models using 128 samples per batch. For statistical significance, we trained every model on four different seeds ($\{1, 2, 3, 4\}$).

3.2 Dataset

Recent research into the detection of audio fakes has led to the introduction of the WaveFake dataset [6]. The researchers synthesize audio of spoken sentences from two popular reference sets, namely JSUT [36] and LJSpeech [15], using five different generative network architectures. In addition to that, the WaveFake dataset was extended by two more recent generators, as part of the work on generalizing networks for audio deepfake detection [8]. The complete list of all generators included in our dataset can be taken from Table 3.1.

The *basic5000* subset of the JSUT dataset contains 5000 utterances of everyday conversational Japanese spoken by a single female native speaker. The length of the audio files ranges from 1.4 seconds to 16.4 seconds. The LJSpeech dataset includes 13100 audio recordings of a single female native English speaker reading passages from several different books. The length of the audio files ranges from 1.1 seconds to 10.1 seconds. We label the original audio as *real* and the synthesized audio as *fake*.

Figure 3.1 displays time-frequency descriptions of a sample in the LJSpeech dataset and its corresponding synthesized version taken from the WaveFake dataset. Interestingly, one can observe subtle differences, especially in the higher frequency regions of the signal. But upon listening to the audio of this sample, it is challenging to detect these differences.

ID	Reference Set	Generator	
		Name	Published in
R1	JSUT [36]	–	–
R2	LJSpeech [15]	–	–
A1	JSUT	Multi-Band MelGAN [41]	2020
	JSUT	Parallel WaveGAN [40]	2020
A2	LJSpeech	Full-Band MelGAN [41]	2020
	LJSpeech	HiFi-GAN [19]	2020
	LJSpeech	MelGAN [21]	2019
	LJSpeech	Multi-Band MelGAN [41]	2020
	LJSpeech	Parallel WaveGAN [40]	2020
	LJSpeech	WaveGlow [32]	2018
B2	LJSpeech	Avocodo [3]	2023
	LJSpeech	BigVGAN [25]	2023

Table 3.1: The composition of the dataset used for our experiments. All audio files in subsets A1 and A2 are taken from the WaveFake dataset [6] and all audio files from subset B2 are taken from the WaveFake dataset extension [8].

3.2.1 Preprocessing

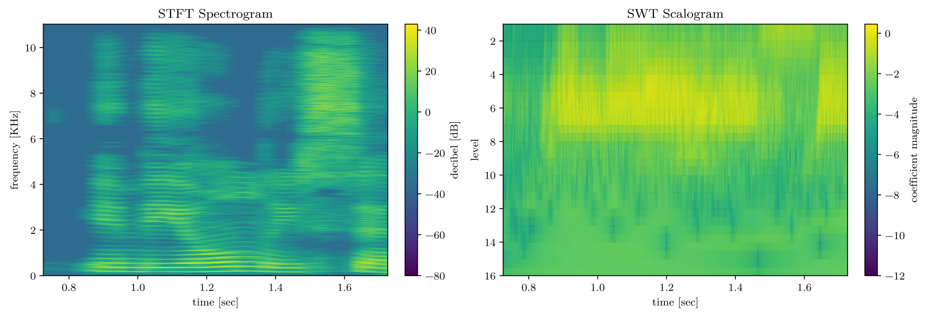
Preprocessing is essential for training of neural networks to ensure that the raw input data matches the format expected by the network. Regarding our dataset, the raw audio data requires a preprocessing such that it can be processed by a CNN. In general, thorough preprocessing has been shown to accelerate convergence and improve model robustness. This subsection describes our default preprocessing pipeline. The audio data used to train a model is subject to the following transformations in the order in which they are presented.

Resampling and Slicing

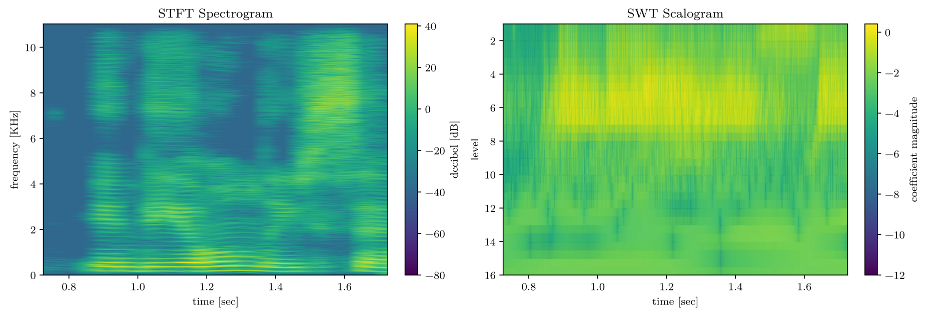
All audio files are resampled to a sample rate of 22050Hz. Once an audio file has been loaded, a slice of size N is extracted, resulting in a vector X^N. During the training phase, a random contiguous slice from the audio data is extracted. Conversely, for the purpose of validation and testing, a contiguous slice starting from index zero is used. The slicing operation is comparable to the random crop operation commonly applied to images. Both operations augment the dataset by creating varied versions of the samples.

Stationary Wavelet Transform

The one-dimensional input vector X^N is transformed using the SWT with a given wavelet. Due to the implementation of the SWT, the input vector is required to have a size that is a multiple of 2^L, where L denotes the number of decomposition steps of the SWT. The output is a two-dimensional coefficient vector $\mathbf{W}$ of shape $(L+1, N)$.



(a) Original audio from LJSpeech



(b) Synthesized audio from Full-Band MelGAN

Figure 3.1: Spectrograms (left column) and scalograms (right column) of an one-second slice from an utterance contained in the LJSpeech corpus. The first row depicts the original recording and the second row depicts its synthesized counterpart generated with Full-Band MelGAN. The spectrograms were created using the STFT with the Hann window computing 1025 frequency bands. The scalograms display the resulting coefficients from a 15-level SWT using the Daubechies (db4) wavelet.

Normalization

The normalization of the coefficients involves three steps. First, we take the logarithm of the absolute coefficient values. This reduces the range between very small and very large values, but also removes the sign from all values. Second, we zero-center the values by subtracting the mean of all coefficients in the training set. Finally, we divide all values by the standard deviation of all coefficients in the training set. Generally, this normalization ensures an even distribution of the features present in the training data, which is recommended for training CNNs [1]. Since the coefficient vectors of all samples in the training set usually did not fit into memory, we had to employ an iterative approach to computing mean and standard deviation, called Welford's online algorithm.

3.3 Fingerprints

While most recent advances on generators make real and fake voices indistinguishable for the human ear, there still exist differences detectable for machines. The question, if generators leave artificial fingerprints has been addressed by Marra et al. in their research paper [29]. Although, their work focused on image data, the proposed method can also be used for audio data. For some input X_i and some denoising filter $f(\cdot)$, one extracts a noise residual R_i by computing $R_i = X_i - f(X_i)$. The authors assume the residual to contain a deterministic non-zero component (the fingerprint) and some random noise component. Now, given N input samples, by averaging over all their residuals, one can estimate the fingerprint

$$\hat{F} = \frac{1}{N} \sum_{i=1}^{N} R_i \, . \tag{3.1}$$

Intuitively, for small N, the fingerprint is dominated by some noise component. However, as N grows, this noise component tends to vanish. Eventually, this estimate converges to a stable pattern and what remains is an approximation of the fingerprint [29].

In this work, we discovered fingerprints using the stationary wavelet transform for every reference set and for every available generator. We transformed the first 2^{15} samples ($\sim 1,48$ seconds) of every audio file using a 15-level SWT with the Haar wavelet. We then reconstructed the audio signal using the inverse SWT twice, once with all coefficients, and once with all coefficients but the first level wavelet coefficients set to zero. Since the first level wavelet coefficients contain information about the top half of the frequency spectrum, the second reconstruction gives a low-passed signal, which acts as our denoising filter. By taking the difference of these two reconstructed signals, we high-passed the audio signal and yield the desired noise residuals. Figure 3.2 depicts spectrograms of fingerprints of the LJSpeech dataset and of three different generators on LJSpeech. The generator fingerprints are easily distinguished from the reference set by their artifacts. Moreover, individual peak frequencies can be identified. These peak frequencies are also audible, but not pleasant to listen to.

The question now arises as to whether these fingerprints are also recognizable by the coefficients of the stationary wavelet transform. We transformed every fingerprint using a 15-level SWT with several different wavelets and made two important observations. First, the examination of the mean absolute coefficient magnitudes, depicted in Figure 3.3, shows differences between the fingerprint of a reference set and the fingerprint of any generator on the respective reference set. This assures that differences in the fingerprints, which we observed in Figure 3.2, are also reflected in the coefficients. However, it does not provide a clear distinction between the reference and generator fingerprints. It is also noteworthy that the magnitude and distribution of the absolute differences in coefficient magnitudes across the decomposition steps depend on the wavelet used in the SWT.

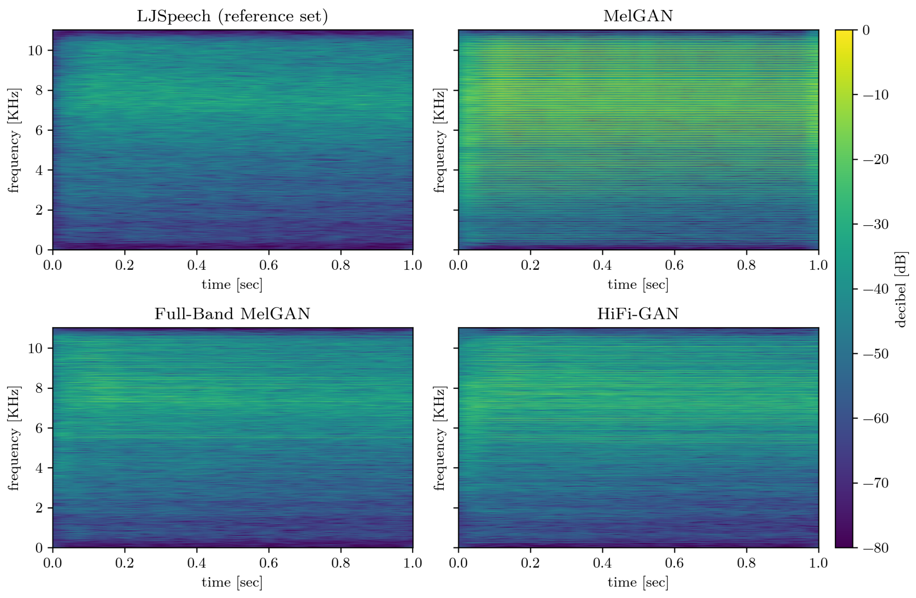
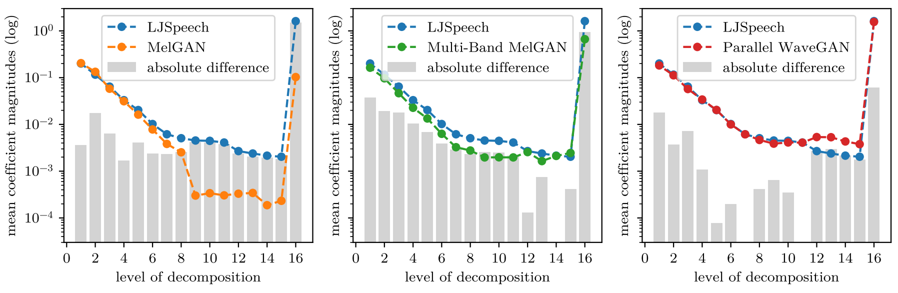


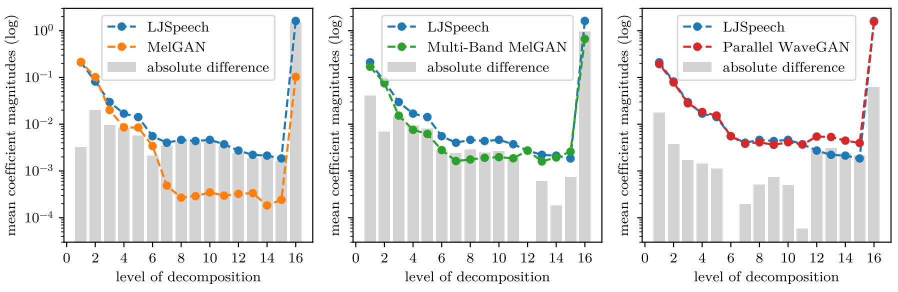
Figure 3.2: Spectrograms of four fingerprints. The top left shows the fingerprint of the reference set LJSpeech. The top right, bottom left, and bottom right show the fingerprints of MelGAN, Full-Band MelGAN, and HiFi-GAN, on LJSpeech, respectively. All spectrograms were created using the STFT with the Hann window computing 1025 frequency bands. The fingerprint of the reference set is visible in the fingerprints of the generators. The spectrograms of the generator fingerprints show noisy artifacts with distinguishable frequencies, mostly in the upper half of the frequency spectrum.

GAN fingerprints are expected to be predominantly contained in the high-frequency components of a signal [43], which are mainly represented by the first decomposition step of the SWT. While differences in coefficient magnitudes can also be observed in the first level coefficients, this raises the question of whether these differences are sufficient for a neural network to detect a generator fingerprint. In the forthcoming sections, empirical evidence will be presented in an attempt to answer this question.

The second important observation arises from the investigation of correlations between the coefficients. The inspiration for this investigation was the work of Marra et al. [29], where the correlation between the fingerprints of generator models has been studied. For each generator, the researchers sampled 1000 images using each of their chosen prompts to the model. The fingerprints were then estimated based on 512 samples. The remaining samples were used to calculate the average correlation between them and the fingerprints. A key finding is that the samples not only correlate with the fingerprints of their generator, which uses the same input prompt, but also with fingerprints of their generator using other input prompts. This, however, depends on the GAN [29].



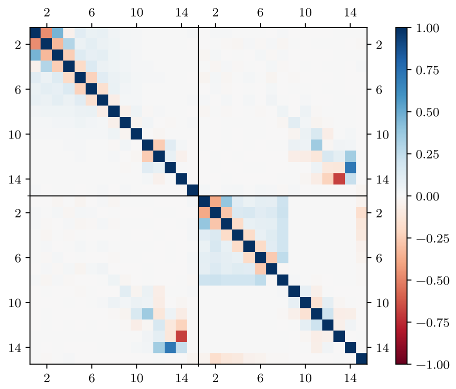
(a) SWT decompositions using the Daubechies (db2) wavelet



(b) SWT decompositions using the Coiflet (coif4) wavelet

Figure 3.3: Mean absolute coefficient magnitudes returned from a 15-level SWT decomposition of four fingerprints, namely the LJSpeech reference set, MelGAN, Multi-Band MelGAN, and Parallel WaveGAN. The gray bars show the absolute differences of the mean absolute coefficient values for every level of the decomposition. Differences of the coefficient values can be observed across all decomposition steps. Note, that decomposition level 16 represents the scaling coefficients at level 15.

We computed the correlation between SWT coefficients of the reference fingerprint and the mean SWT coefficients of the fingerprints of all generators. The results for two different wavelets are presented in Figure 3.4. First of all, the large diagonal entries signify a very strong correlation of every coefficient vector with itself, which is to be expected. It is notable, however, that there are also weak correlations between coefficient vectors, especially in neighboring vectors. Furthermore, weak correlations within the mean fingerprint of all generators indicate that there are some commonalities across all fingerprints.
Besides correlations within the fingerprints, we also observe correlations between the counterparts, shown by non-zero values in the lower left and upper right sections of both plots. What is particularly interesting here is that these correlations are more apparent in the later steps of the decomposition. At the same time, no correlations can be seen here in the first steps of the decomposition. This indicates measurable differences in the higher frequency ranges and


(a) Daubechies (db2) wavelet

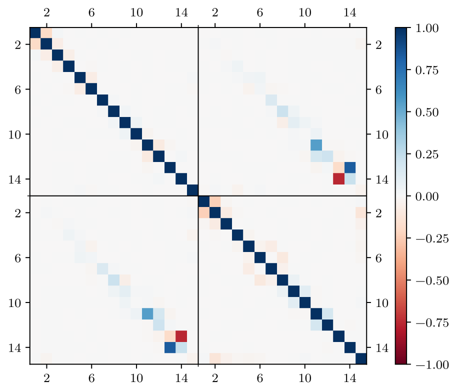

(b) Daubechies (db8) wavelet

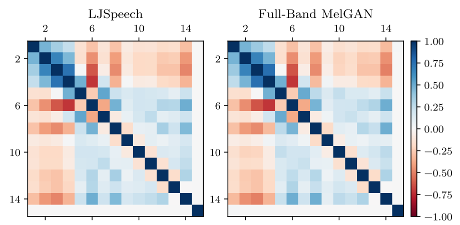
Figure 3.4: Correlations of SWT coefficients returned from 14-level decompositions using the two different wavelets from the Daubechies family. The upper left section of both plots displays correlations within the fingerprint of the LJSpeech reference set (R2). In contrast, the lower right section of both plots displays correlations within the mean coefficients of all generator fingerprints of the original WaveFake dataset (A2). The lower left and upper right sections of both plots display correlations between the respective counterparts. Values close to zero indicate no correlation, whereas values approaching one or minus one indicate a strong correlation. The rows and columns correspond to the coefficient vectors of the decompositions.

is consistent with the finding that GAN fingerprints are more prevalent in the high frequency range of a signal.

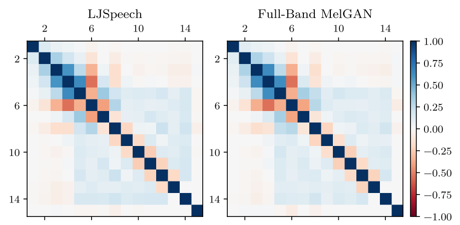
3.4 1-dimensional vs 2-dimensional CNNs

Before we delve deeper into convolutional neural networks to solve the classification task at hand, let us quickly state an interesting observation. After transforming an arbitrary audio signal into a time-frequency description, one inherently yields a two-dimensional result. It can be observed that due to the presence of harmonies or overtones in a given signal, two or more frequencies can correlate with each other. Similarly, neighboring scales obtained from a wavelet transform are also not independent of each other. Referring back to Figure 3.1, we can see such relationships between different frequencies and scales.

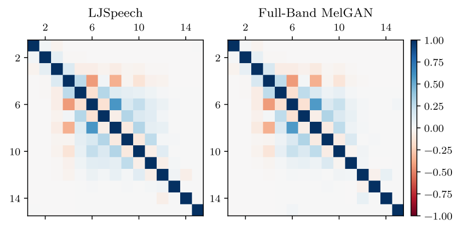
The question thus arises as to whether there is a clear advantage to the use of two-dimensional convolutional layers when processing time-frequency descriptions of audio signals. In a one-dimensional CNN, the kernel slides along the time dimension and correlations between scales are represented by connections



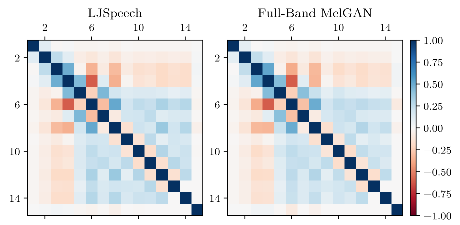
(a) Haar wavelet (`haar`)



(b) Daubechies wavelet (`db2`)



(c) Symlet (`sym4`)



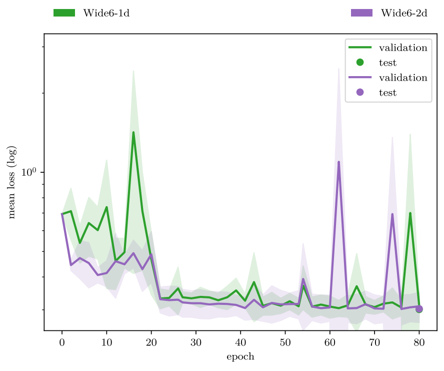
(d) Coiflet (`coif1`)

Figure 3.5: Correlations of SWT coefficients. Sample *LJ001-0001* from the LJSpeech dataset and its counterpart generated by Full-Band MelGAN were transformed by a 14-level SWT with four different wavelets. Values close to zero indicate no correlation, whereas values approaching one or minus one indicate a strong correlation. The rows and columns correspond to the coefficient vectors of the decompositions.

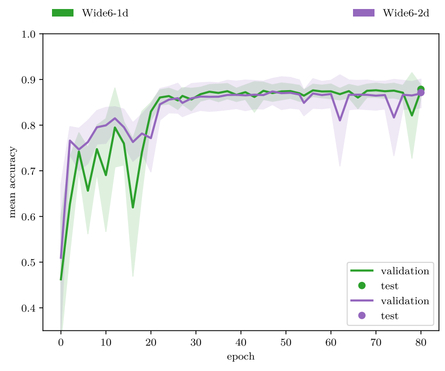
between the input and output channels of the convolutional layers. In a two-dimensional CNN on the other hand, the kernel slides along the time and the scale dimension of the input, while the channel dimension remains available for learning additional features. Consequently, the two-dimensional CNN has an enhanced expressive capability.

Figure 3.5 depicts correlations of SWT coefficients. It is expected that every coefficient vector of some level has a strong correlation to itself, hence the large diagonal entries. However, we can also observe weaker correlations between coefficient vectors of neighboring levels. While this analysis confirms our theoretical considerations on a sample basis, we also tried to gather experimental evidence using two simple convolutional networks.

Both networks comprise six convolutional layers and one final fully connected layer. The key difference between the two networks is that one, named Wide6-1d, employs one-dimensional convolutions, whereas the other, named Wide6-2d employs two-dimensional convolutions. Both networks receive the same SWT coefficients as input, which is of crucial importance to this experiment. The one-dimensional variant processes the coefficient vectors as separate input channels. In contrast, the two-dimensional variant employs the coefficient vectors as a second dimension for the kernel. Therefore, the two-dimensional variant has one additional kernel dimension and thus slightly more learnable parameters.



(a) Mean validation and test loss



(b) Mean validation and test accuracy

Figure 3.6: Training progress of Wide6-1d (green) and Wide6-2d (purple) CNNs. The plot on the left shows the progression of the mean validation loss and the final test loss. The plot on the right shows the progression of the mean validation accuracy and the final test accuracy.

Both networks have an equal sized fully connected layer, which ensures a fair and impartial comparison. The total number of learnable parameters for the Wide6-1d and the Wide6-2d networks, respectively, is 139 218 and 164 538.

We train both networks on the combined dataset R1 and A1 for 80 epochs using the default preprocessing pipeline. In addition to the default training setup we use a decaying learning rate that reduces by a factor 10 every 20 epochs. Figure 3.6 depicts the training progress for both networks.

The two-dimensional model appears to initially learn at a slightly faster rate than its one-dimensional counterpart. However, after sufficient training epochs, both models converge to similar validation loss and accuracy. In addition, the test loss and accuracy at epoch 80 are strikingly similar. Wide6-1d achieves a test accuracy of $87.84\% \pm 0.41\%$ and Wide6-2d achieves a test accuracy of $87.13\% \pm 2.69\%$. These findings demonstrate that the choice of one-dimensional or two-dimensional convolution has little impact on the results obtained. When the computational effort is a concern, a one-dimensional network should be preferred due to its reduced number of learnable parameters.

3.5 Transfer Learning with ResNet-50

Before developing custom network architectures tailored to the detection of fake audio, we attempt to leverage the transfer learning technique and benefit from an existing pre-trained model, namely ResNet-50 from the popular ResNet family. The concept of residual learning and its advantages for deep convolutional networks was presented in Section 2.2.4.

A common scenario for transfer learning involves using a pre-trained CNN as a fixed feature extractor, retaining its weights and using its output as input for a classifier network. Alternatively, the pretrained weights can be used to fine-tune the CNN further. We will now experiment with fine-tuning the pre-trained ResNet-50 to detect audio deepfakes. It can be reasonably assumed, that the degree of similarity between image data and time-frequency representations of audio signals is very small. It is, however, still possibile to benefit from pre-trained weights in practice [2].

The convolutional layers of any ResNet architecture can be organized in five groups. The first convolutional layer of each group performs a striding operation along both axes, except the one of the second group. In this group, however, there is a max-pooling operation with a stride of two. Therefore, the input size is divided by two for a total of five times along both dimensions. Here it becomes apparent, that we can not use the SWT coefficients as input to ResNet-50, unless we compute a 31-level SWT of the input, which would require a signal of length 2^{31}. This is not possible with our dataset. Even if we had such a signal, the ResNet would only reduce it to length 2^{26}. This in turn would require an extremely large fully connected layer, which is not desirable, because it induces many extra parameters.

The above considerations have led us to the decision to transform the coefficients in a way such that their shape matches the expected shape of (3, 224, 224). This makes adjusting the models fully connected layer easier, where solely the number of output channels has to be set to 2. We experimented with three different approaches to reshaping the input, which will be described in the following paragraphs.

A: Duplicates

In this approach, we begin by slicing the coefficient vector in order to match the expected width. We then repeat every sliced vector in an interleaved fashion to match the expected height. Finally, we create two copies of the coefficients, each for one of the missing color channels.

B: Chunked

Here we split the coefficient vector into chunks matching the expected width. Subsequently, 15 unique chunks are concatenated to match the expected height. This procedure is repeated for every of the three color channels.

C: Three Frequency Bands

This approach is similar to B, but before chunking we partition the coefficient vector into three frequency bands. Then we require the concatenation of 45 unique chunks in order to match the expected height.

A visualization of the differences of these three reshaping approaches can be found in Figure 3.7. It is evident that approach A inevitably creates a lot of duplicated values. However, what all of these approaches have in common is that they differ significantly from the typical input image that a ResNet would expect.

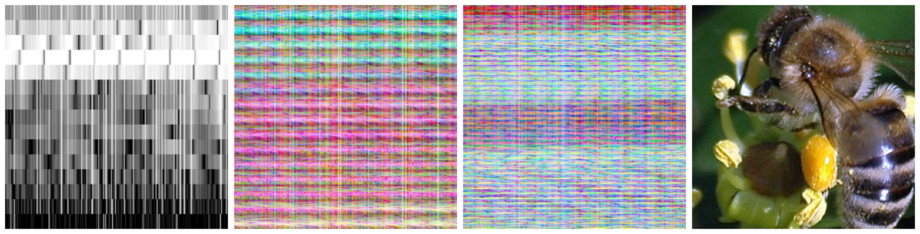
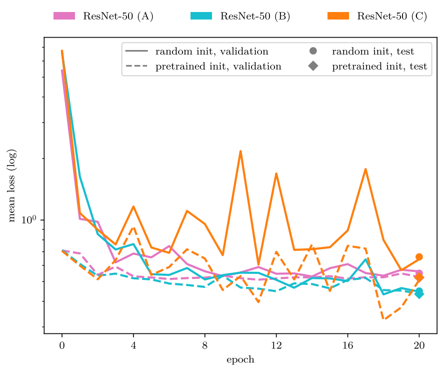


Figure 3.7: Visualization of four different inputs for the ResNet-50 model. The first three images display the same SWT coefficient vector after reshaping with our different methods (in order A, B, C). The image on the right displays a sample (*n02206856_bee*) from the ImageNet dataset.

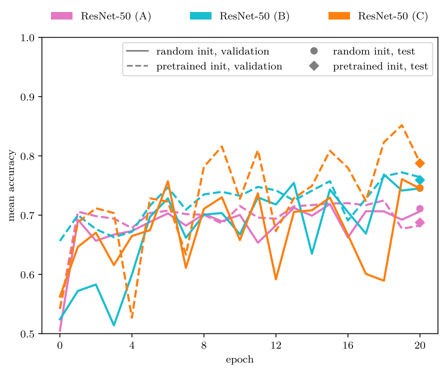
For this experiment, the existing ResNet-50 implementation of PyTorch has been used. The pre-trained model was initialized using weights, that closely reproduce the results obtained in the original paper [14]. We train one randomly initialized and one pre-trained ResNet-50 on the combined dataset R1 and A1 for 20 epochs. Before reshaping we employ the default preprocessing pipeline.

Figure 3.8 shows the progress of the loss values and accuracies during the training process. All pre-trained models start with a significantly smaller loss compared to their randomly initialized counterparts. However, the validation accuracies prior to the first training epoch do not show a similar advantage of the pre-trained models. Only the pre-trained models that receive "chunked" inputs exhibit a notably higher initial validation accuracy. The randomly initialized models converge in loss and accuracy with their pre-trained counterparts after only a few training epochs. This indicates that there is no clear benefit of having pre-trained weights. Overall, the progression of the mean validation loss and the mean validation accuracy show no discernible improvement, suggesting an inability to learn the classification task. A quick glance at the validation and test accuracies serves to confirm this conclusion. None of the models reach a mean test accuracy of 80% or above. Given the success of the ResNet-50 network in the image classification domain and its large number of learnable parameters, this raises the question, why we were unable to utilize the ResNet-50 to detect audio fakes.

The following paragraph seeks to answer to this question. We will begin by examining the nature of the inputs and the architecture of the network. As previously noted, reshaping approach A creates a lot of duplicated values. To be more precise, almost 98% of the values are duplicates, and in turn, only 2% of the values carry relevant information. Nevertheless, approaches B and C do not include any duplicates, yet still achieve only marginal improvements in performance. With regard to the pre-trained weights, it can be assumed that the characteristics in the SWT coefficients differ significantly from the learned characteristics of image inputs. Despite the well-distributed input of SWT coefficients resulting from the normalization during preprocessing, the learned



(a) Mean validation and test loss



(b) Mean validation and test accuracy

Figure 3.8: Training progress of the ResNet-50 variants. The reshaping approaches A, B, C are indicated by the colors pink, cyan and orange, respectively. Solid lines identify the randomly initialized models, dashed lines identify the pre-trained models.

distribution appears to be different. Finally, we will attempt to explain why the randomly initialized network still achieves a comparable performance. In a recent publication, Frankle et al. stated the *lottery ticket hypothesis*, which suggests that a randomly initialized dense neural network contains a subnetwork that can match the test accuracy of the original network when trained in isolation. The experiments conducted on fully-connected networks and convolutional networks, including popular architectures such as ResNet, have shown the existence of subnetworks whose initial weights of their connections are particularly suitable for isolated training. Because such subnetworks are able to achieve test accuracies comparable to the original network, Frankle et al. call them *winning tickets* [7]. It is possible for our randomly initialized models to contain subnetworks that are winning tickets.

In summary, independent of the initialization, our models began converging after only a few training epochs. The mean test accuracy of all model variants remains below 80%. These findings have led to the decision to cease further investigation of the transfer learning approach using a ResNet architecture.

3.6 Wide Residual CNNs

The previous section indicated difficulties associated with the processing of SWT coefficients for CNNs due to the shape of the coefficient vector. Recall, that the first and the second dimension of such a vector represent the decomposition steps and the sample steps, respectively. It is also important to remember that there is no downsampling in the SWT. Consequently, each decomposition step yields N coefficients. The size of the second dimension therefore is significantly larger than the size of the first dimension, as the number N of samples in the source signal was selected to such that the number of desired levels equals $L = log_2(N)$. The subsequent section will examine residual convolutional networks that are designed to process such inputs, which we refer to as wide inputs.

Recent research into audio deepfake detection presented promising results for convolutional architectures, especially in combination with time-frequency representations [22, 8]. As outlined in Section 2.2.4, deep CNNs benefit from residual learning. In this thesis we will thus try to combine convolutional architectures with residual learning. More precisely, we designed four different residual CNNs using residual blocks that closely follow the original implementation of the ResNet blocks. Two networks, named Wide-16 and Wide-24, exclusively employ `basic` residual blocks. Two additional networks, named Wide-19 and Wide-32, also utilize `bottleneck` residual blocks.

layer	output size	Wide-16	Wide-19	kernel	stride	dilation
1	(8 × 2048)	`basic`(1, 4)	`basic`(1, 4)	(3, 4)	(1, 2)	(1, 2)
		`basic`(4, 8)	`basic`(4, 8)	3	(1, 2)	1
		`basic`(8, 8)	**`bottleneck`**(8, 8, 2, 1)	3	1	1
		`maxpool`	`maxpool`	2	1	1
2	(4 × 512)	`basic`(8, 16)	`basic`(8, 16)	(3, 4)	(1, 2)	(1, 2)
		`basic`(16, 32)	`basic`(16, 32)	3	(1, 2)	1
		`basic`(32, 32)	**`bottleneck`**(32, 32, 2, 1)	3	1	1
		`maxpool`	`maxpool`	2	1	1
3	(1 × 64)	`basic`(32, 64)	`basic`(32, 64)	3	1	1
		`maxpool`	`maxpool`	2	1	1
		`basic`(64, 128)	**`bottleneck`**(64, 64, 4, 2)	3	1	1
		`maxpool`	`maxpool`	2	(1, 2)	1
4	(1 × 2)	`fc`(8192, 2)	`fc`(8192, 2)			
conv layers		16	19			
parameters		344.574	112.526			

Table 3.2: Architecture of the Wide-16 and Wide-19 networks. We use the definition of the `basic` and `bottleneck` residual blocks given in equation (2.30). The first two parameters represent the input and output channels of the layer. For better readability, we fix the expansion parameter e of the `basic` block to 1. The columns kernel, stride, and dilation represent the parameters k, s, and d, respectively. The difference between the two networks is highlighted in **bold**.

layer	output size	Wide-24	Wide-32	kernel	stride	dilation
1	(15×4096)	basic(1, 4)	basic(1, 4)	(3, 4)	(1, 2)	(1, 2)
		basic(4, 8)	basic(4, 8)	3	(1, 2)	1
		basic(8, 8)	basic(8, 8)	3	1	1
2	(15×1024)	basic(8, 16)	basic(8, 16)	(3, 4)	(1, 2)	(1, 2)
		basic(16, 32)	basic(16, 32)	3	(1, 2)	1
		basic(32, 32)	basic(32, 32)	3	1	1
3	(4×256)	basic(32, 48)	basic(32, 48)	3	2	1
		basic(48, 64)	basic(48, 64)	3	2	1
		basic(64, 64)	**bottleneck**(64, 64, 2, 1)	3	1	1
			bottleneck(64, 64, 2, 1)	3	1	1
4	(1×64)	basic(64, 96)	basic(64, 96)	3	2	1
		basic(96, 128)	basic(96, 128)	3	2	1
		basic(128, 128)	**bottleneck**(128, 128, 4, 1)	3	1	1
			bottleneck(128, 128, 4, 1)	3	1	1
5	(1×2)	fc(8192, 2)	fc(8192, 2)			
conv layers		24	32			
parameters		946.526	639.838			

Table 3.3: Architectures of the Wide-24 and Wide-32 models. We use the definition of the basic and bottleneck residual blocks given in equation (2.30) and fix the expansion parameter e of the basic block to 1. The difference between the two networks is highlighted in **bold**.

The number of convolutional layers for our Wide-16 and Wide-19 networks is 16 and 19, respectively. In the Wide-19 network, we replace the final basic residual block of each layer with a bottleneck residual block. The number of convolutional layers for our Wide-24 and Wide-32 networks is 24 and 32, respectively. In the Wide-32 network, we replace two basic residual blocks of the final two layers with two bottleneck residual blocks. The aforementioned modifications account for the increase in convolutional layers. It is important to ensure identity mappings for the shortcut connections in order to capitalize on the computational efficiency of residual blocks. This is particularly important for the bottleneck block [14]. In doing so, the Wide-19 and Wide-32 have a considerably lower number of learnable parameters in comparison to their counterparts, despite the increase in the number of convolutional layers. In this context, it is important to emphasize that all networks have an equal sized final fully connected layer with 8192 input units and 2 output units. A comprehensive overview of the architectures can be found in Table 3.2 and Table 3.3. In our networks, we use the LeakyReLU [27] activation function for all detector layers.

In this experiment, we exclusively train our models on a single generator model and evaluate their generalization capabilities on all available generator models. Research prior to this work suggests that training a classifier on Full-Band MelGAN results in a better generalization compared to other generator models [6]. We adopt these results and proceeded to train our models on Full-Band MelGAN. The training dataset included a 70% random subset of the synthesized

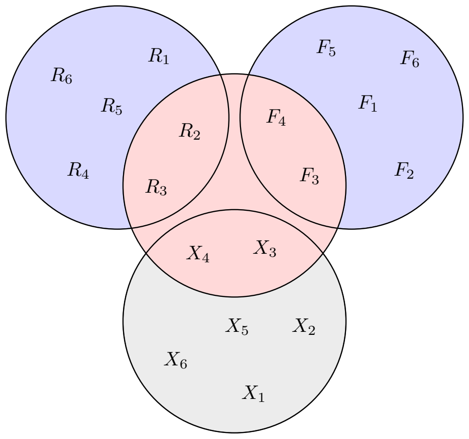


Figure 3.9: Venn diagram illustrating the composition of the evaluation dataset. The blue area represents the training set, while the red area represents the evaluation set. $R_1, \dots, R_6$ represent samples from the reference set, $F_1, \dots, F_6$ represent samples from Full-Band MelGAN, and $X_1, \dots, X_6$ represent samples from some unseen generator. Note, that the evaluation set includes the same sample IDs $(3, 4)$ for both generators.

audio by Full-Band MelGAN (A2) and of the reference samples in LJSpeech (R2). The training of a single model for 40 epochs took approximately one hour.

3.6.1 Evaluation

This subsection explains the evaluation methodology and presents the results of our models. All models were evaluated exclusively on unseen samples, which is the standard for assessing the performance of neural networks. However, we impose one additional constraint on our evaluation dataset. Since we trained our models using fake samples synthesized only by Full-Band MelGAN, there exist many unseen synthesized samples from all the other generative models in the dataset. In order to ensure a fair comparison between the generators, we map the unseen fake samples encountered during the training phase to the same fake samples of each other generator. In particular, we collect the unseen sample IDs and then select the same sample IDs for the other generative models. Hence, the model to be evaluated has to predict the same non-authentic utterances for all generators. The diagram shown in Figure 3.9 serves to illustrate this methodology.

All models that are compared within this thesis were trained exclusively on LJSpeech reference samples (R2) and synthetic samples generated by Full-Band MelGAN (A2). This applies especially to baseline models and any other model used for comparison. The evaluation of the baseline models on the original WaveFake dataset was performed by Frank et al. [6]. The evaluation of the baseline models on the extended WaveFake dataset was done by Gasenzer et al.

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-16	SWT-db4	69.08	66.29 $\pm$ 2.12	0.299	0.331 $\pm$ 0.022
	SWT-db8	68.39	67.49 $\pm$ 1.19	0.322	0.334 $\pm$ 0.011
	SWT-sym7	71.56	68.74 $\pm$ 1.72	0.274	0.312 $\pm$ 0.025
	SWT-coif9	70.04	69.06 $\pm$ 1.17	0.286	0.294 $\pm$ 0.005
Wide-19	SWT-db4	67.87	64.92 $\pm$ 3.22	0.314	0.334 $\pm$ 0.019
	SWT-db8	67.14	66.11 $\pm$ 1.11	0.325	0.346 $\pm$ 0.013
	SWT-sym7	66.87	66.27 $\pm$ 0.56	0.307	0.325 $\pm$ 0.019
	SWT-coif9	69.00	68.06 $\pm$ 0.62	0.270	0.308 $\pm$ 0.022
Wide-24	SWT-db4	68.89	67.76 $\pm$ 0.69	0.285	0.320 $\pm$ 0.021
	SWT-db8	69.69	68.58 $\pm$ 0.86	0.266	0.300 $\pm$ 0.022
	SWT-sym7	72.43	70.24 $\pm$ 2.11	0.264	0.297 $\pm$ 0.030
	SWT-coif9	70.27	69.77 $\pm$ 0.53	0.278	0.304 $\pm$ 0.018
Wide-32	SWT-db4	68.52	67.08 $\pm$ 1.30	0.276	0.312 $\pm$ 0.026
	SWT-db8	68.01	66.99 $\pm$ 0.68	0.317	0.343 $\pm$ 0.019
	SWT-sym7	70.12	68.66 $\pm$ 1.38	0.280	0.312 $\pm$ 0.028
	SWT-coif9	69.37	67.97 $\pm$ 1.18	0.291	0.313 $\pm$ 0.018
GMM [6]	LFCC	–	–	**0.062**	–
RawNet2 [6]	raw	–	–	0.363	–

Table 3.4: Evaluation results of the wide residual CNNs on the original WaveFake dataset including JSUT (R1, R2, A1, A2). The second column presents the different input types processed by the networks. In addition, the last two rows provide a comparison to the baseline models. Results highlighted in blue represent the best values within their respective columns, ignoring the baseline models.

[8]. Our models have been evaluated independently for every generator using the evaluation set. This enables us to precisely analyze the generalization to the different generator models. The final result of a model is presented using the mean over its evaluations. Since we trained our models on four different random seeds, we compute the mean, denoted by μ, and standard deviation, denoted by σ, over the evaluation results.

Besides the prediction accuracy, which is defined as the fraction of correctly predicted samples out of the total number of evaluated samples, we provide an additional evaluation metric called *equal error rate* (EER). This metric is particularly suitable for assessing the performance of binary classifiers. Consequently, it is frequently used in the evaluation of biometric security systems and is also part of the evaluation in the ASVspoof challenge [38]. Formally, it is defined as the point on the receiver operating characteristic (ROC) curve, where the false acceptance rate and the false rejection rate are equal. Intuitively, it can be described as the rate of incorrectly accepted and incorrectly rejected sam-

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-16	SWT-sym7	72.59	70.50 $\pm$ 1.25	0.263	0.292 $\pm$ 0.019
	SWT-coif9	71.47	70.34 $\pm$ 1.36	0.274	0.282 $\pm$ 0.005
Wide-19	SWT-sym7	68.89	68.06 $\pm$ 0.72	0.294	0.306 $\pm$ 0.014
	SWT-coif9	70.71	69.67 $\pm$ 0.78	0.261	0.292 $\pm$ 0.018
Wide-24	SWT-sym7	73.41	71.97 $\pm$ 1.42	0.256	0.277 $\pm$ 0.021
	SWT-coif9	71.53	71.09 $\pm$ 0.47	0.273	0.287 $\pm$ 0.011
Wide-32	SWT-sym7	71.73	70.39 $\pm$ 1.32	0.265	0.292 $\pm$ 0.023
	SWT-coif9	71.15	69.74 $\pm$ 1.02	0.276	0.295 $\pm$ 0.014
GMM [6, 8]	LFCC	–	–	**0.145**	–

Table 3.5: Evaluation results of the wide residual CNNs networks on the extended WaveFake dataset including JSUT (R1, R2, A1, A2, B2). The second column presents the different input types processed by the networks. Results highlighted in blue represent the best values within their respective columns, ignoring the GMM model.

ples. With regard to the detection of audio deepfakes, an incorrectly accepted sample means predicting a fake sample as authentic. Conversely, an incorrectly rejected sample means predicting an authentic sample as fake. The EER provides a single measure for the overall accuracy of binary classifiers. Values equal to zero indicate no wrong predictions, whereas values equal to one indicate only incorrect predictions. A value of 0.5 can be interpreted as guessing [6].

The evaluation results of all models are presented in Table 3.4 and Table 3.5. It can be observed, that the choice of the wavelet can impact the performance of the model. The Symlet and Coiflet wavelets seem to be more suitable than wavelets from the Daubechies family, as all models consistently achieve slightly higher recognition rates. Generally, wavelets with larger filter length result in better model performance. The Wide-16 and Wide-19 models achieve best recognition rates using the `coif9` wavelet. The Wide-24 and Wide-32 models on the other hand achieve best recognition rates using the `sym7` wavelet. Overall, with a mean accuracy of 70.24%$\pm$2.11%, the Wide-24 network achieves the best performance of all our models evaluated on the original WaveFake dataset. The recognition rates surpass those of RawNet2, which served as a lower baseline in several ASVspoof challenges. Frank et al. showed that RawNet2 does not generalize well on the original WaveFake dataset, hence its high EER of 0.363 when evaluated on the entire dataset [6]. Our models, on the other hand, appear to generalize well on unseen fake samples. Remember that all fake samples contained in the training set originate from Full-Band MelGAN. Therefore, the models were able to successfully detect the remaining five unseen generative networks. However, the recognition rates are considerably lower than those of the GMM on the LFCC features.

The results presented in Table 3.5 also demonstrate satisfactory performance on the extended WaveFake dataset, indicating that the models are also able to detect newer generators. The models are able to detect fake samples from BigVGAN and Avocodo without any significant loss of accuracy. Nevertheless, the recognition rates observed here are also considerably lower than those of the GMM. A comparison with the performance of more recent detectors, which will be presented in the following section of this thesis, shows that our results are not competitive.

3.7 Comparison to DWPT models

This final section of the chapter will present a comparative analysis of the potential of the stationary wavelet transformation with that of other methods. We have already compared the recognition rates of our four models processing SWT coefficients with the baseline models, RawNet2 and GMM. While these baseline models were a good starting point for our analysis, they cannot be considered the current state of the art in audio deepfake detection. We will therefore compare our results with the performance of more recent detector models.

First, we will consider the already mentioned LCNN. While this model was originally designed to process Fourier transform coefficients, Gasenzer et al. also evaluated this network processing DWPT coefficients. Interestingly, this resulted in a considerable improvement [8]. In addition, they designed the Dilated Convolutional Neural Network (DCNN), which consists of 9-layer convolutional layers and a final fully connected layer. The last two convolutional layers utilize dilated kernels. Aside from the STFT, the wavelet packet transform with a variety of wavelets was used as input to this model. Their experimental results show very good recognition rates for samples from unseen generators [8].

Apart from the GMM, our models have not been compared with any other detector architectures than convolutional neural networks. Recently, a purely attention-based architecture called Audio Spectrogram Transformer (AST) [9] has been proposed. The input to the AST consists of audio spectrograms split into 16×16 patches, which are linearly projected to one-dimensional patch embeddings. Then, a positional encoding is added to each embedding. A transformer model processes these embeddings and its output is used for classification in a final fully connected layer. The AST demonstrated good results for audio classification [9]. This suggests that it could also be used to detect audio deepfakes. Gasenzer et al. also evaluated the AST on the WaveFake dataset, which also showed good recognition rates [8].

Building upon the success of the DCNN, we have designed two residual convolutional networks, named WPT-Basic and WPT-Bottle, which also process DWPT coefficients. Please refer to Table 3.6 for details regarding their architecture. One crucial difference compared to the architecture of the networks presented in Section 3.6 is the absence of strided convolutions, because the

layer	output size	WPT-Basic	WPT-Bottle	kernel
1	(128×128)	`basic`(1, 4, 1)	`basic`(1, 4, 1)	5
		`basic`(4, 8, 2)	`basic`(4, 8, 2)	3
		`maxpool`	`maxpool`	2
2	(64×64)	`basic`(16, 24, 2)	**`bottleneck`**(16, 16, 2, 4)	3
		`basic`(24, 32, 2)	–	3
		`maxpool`	`maxpool`	2
3	(16×16)	`basic`(64, 96, 1)	**`bottleneck`**(64, 64, 2, 1)	3
		`maxpool`	`maxpool`	2
		`basic`(96, 128, 1)	**`bottleneck`**(64, 64, 4, 2)	3
		`maxpool`	`maxpool`	2
4	(1×2)	`fc`(32 768, 2)	`fc`(32 768, 2)	
conv layers		12	13	
parameters		487.314	65.122	

Table 3.6: Architectures of the WPT-Basic and WPT-Bottle models. We use the definition of the `basic` and `bottleneck` residual blocks given in equation (2.30). The stride and dilation parameter are fixed to their default value of 1. For this illustration the input for both models has shape (256×256). The difference between the two models is highlighted in **bold**.

downsampling in the DWPT can be regarded as a striding operation. Instead, we use the maximum pooling layers to reduce spatial dimensions.

In our preprocessing pipeline, we replaced the SWT with the DWPT. Specifically, we compute a 8-level DWPT which yields $2^8 = 256$ frequency bins. We chose the signal length N such that $N/2^{L-1}$ approximately matches 256. Given $L = 8$, this results in a required signal length of 32 768 samples. In this calculation we disregarded the fact that the different filter lengths of the wavelets produce different numbers of coefficients. Nevertheless, for an approximately square-shaped coefficient vector, this approach is sufficient. We trained both networks for 80 epochs. In addition to the default training configuration, we schedule the learning rate to decay by a factor of 10 at epoch 25 and 50.

Table 3.7 presents a comparison of the best performing models trained in the context of this thesis with current the state of the art in audio deepfake detection. It is clearly evident that the wide residual CNNs that process SWT coefficients are unable to match the performance of the other models. However, the residual CNNs that process DWPT coefficients, namely WPT-Basic and WPT-Bottle, show promising results. Notwithstanding a slightly lower mean accuracy compared to the DCNN, their average EER appears comparable to the state of the art. As in previous observations, wavelets with higher filter lengths lead to better recognition rates.

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-16	SWT-coif9	71.47	70.34 ± 1.36	0.274	0.282 ± 0.005
Wide-19	SWT-coif9	70.71	69.67 ± 0.78	0.261	0.292 ± 0.018
Wide-24	SWT-sym7	73.41	71.97 ± 1.42	0.256	0.277 ± 0.021
Wide-32	SWT-sym7	71.73	70.39 ± 1.32	0.265	0.292 ± 0.023
WPT-Basic	DWPT-db5	88.10	87.04 ± 0.82	0.064	0.065 ± 0.001
	DWPT-sym5	87.12	86.58 ± 0.33	0.061	0.066 ± 0.003
	DWPT-sym9	88.83	87.68 ± 0.79	0.056	0.060 ± 0.004
	DWPT-coif8	88.42	87.45 ± 0.71	0.065	0.074 ± 0.005
WPT-Bottle	DWPT-db5	86.80	85.13 ± 1.04	0.083	0.089 ± 0.005
	DWPT-sym7	87.92	86.90 ± 0.86	0.058	0.077 ± 0.012
	DWPT-sym9	90.69	88.72 ± 1.33	0.067	0.073 ± 0.005
	DWPT-coif8	90.72	89.07 ± 1.00	0.077	0.085 ± 0.007
DCNN [8]	STFT	96.46	91.72 ± 2.94	0.036	0.159 ± 0.150
	DWPT-db5	96.88	94.65 ± 1.85	0.048	0.082 ± 0.042
	DWPT-sym5	97.70	95.25 ± 3.09	0.031	**0.069 ± 0.036**
	DWPT-coif8	98.72	97.39 ± 1.80	**0.026**	0.079 ± 0.047
LCNN [35, 8]	STFT	91.65	79.21 ± 16.55	0.083	0.169 ± 0.101
	DWPT-sym5	97.46	90.12 ± 6.44	0.067	0.108 ± 0.042
AST [9, 8]	STFT	90.98	87.10 ± 2.54	0.089	0.122 ± 0.021
	DWPT-sym5	93.49	91.25 ± 1.38	0.065	0.087 ± 0.013
GMM [6, 8]	LFCC	–	–	0.145	–

Table 3.7: Comparison of our models with the state of the art. Evaluation on unseen samples of the extended WaveFake dataset including JSUT (R1, R2, A1, A2, B2). The values highlighted in blue represent the best results within our models, ignoring the results of all other models. The values highlighted in **bold** represent the current state of the art in audio deepfake detection.

We will now conclude the experimental chapter with a detailed discussion of the findings. All of the models designed in the context of this thesis were deep residual CNNs with residual blocks that closely follow the original ResNet implementation. Yet we can observe significant differences in their recognition rates. This observation suggests that one of the driving factors behind these differences must be the nature of the input. The wide residual CNNs received SWT coefficients, whereas the other two models received DWPT coefficients. It appears, that the latter are more suitable for the detection of audio fakes in our dataset. We can only speculate about the reasons for this observation.

We assume that the features present in the SWT coefficients are not sufficient to reliably distinguish between real and fake recordings. In particular, we hypothesize that the absence of valuable characteristics in the high-frequency range,

Generator	EER			
	Wide-24-sym7		WPT-Basic-sym9	
	min	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Full-Band MelGAN	0.183	0.190 $\pm$ 0.005	0.001	0.001 $\pm$ 0.001
Avocodo	0.196	0.209 $\pm$ 0.014	0.002	0.002 $\pm$ 0.000
BigVGAN	0.203	0.209 $\pm$ 0.004	0.005	0.009 $\pm$ 0.003
HiFi-GAN	0.302	0.315 $\pm$ 0.009	0.001	0.001 $\pm$ 0.001
Large BigVGAN	0.243	0.253 $\pm$ 0.007	0.074	0.098 $\pm$ 0.017
MelGAN	0.234	0.239 $\pm$ 0.004	0.009	0.013 $\pm$ 0.002
Multi-Band MelGAN	0.267	0.273 $\pm$ 0.009	0.007	0.010 $\pm$ 0.002
Parallel WaveGAN	0.170	0.181 $\pm$ 0.008	0.001	0.001 $\pm$ 0.001
WaveGlow	0.156	0.170 $\pm$ 0.008	0.001	0.001 $\pm$ 0.001
Multi-Band MelGAN (JSUT)	0.394	0.476 $\pm$ 0.073	0.187	0.216 $\pm$ 0.021
Parallel WaveGAN (JSUT)	0.346	0.533 $\pm$ 0.149	0.301	0.313 $\pm$ 0.014

Table 3.8: Recognition rates for each generator available in the extended WaveFake dataset. Wide-24-sym7 represents our best performing model processing SWT coefficients and WPT-Basic-sym9 our best performing model processing DWPT coefficients.

which are solely due to the limited frequency resolution of the SWT, makes it difficult to recognize a fingerprint. The SWT analyzes the upper half of the frequency spectrum once with the wavelet filter returning the first level wavelet coefficients. These coefficients are not further processed. Hence, the majority of high-frequency components of a signal are described by only one vector of wavelet coefficients. The DWPT, on the other hand, recursively filters all coefficients of the previous level. Given some level L, this results in 2^L coefficient vectors that analyze 2^L evenly spaced frequency bands. Half of the coefficient vectors, precisely 2^{L-1}, thus describe the upper half of the frequency spectrum. Therefore, the DWPT coefficients provide additional valuable features that are likely to assist a CNN in the recognition of a fingerprint.

Table 3.8 lists the recognition rates for each generator model available in the extended WaveFake dataset. We selected two of our best performing models, one processing SWT coefficients, namely Wide-24 on `sym7`, and one processing DWPT coefficients, namely WPT-Basic on `sym9`. It is particularly noticeable that the recognition rates on another reference dataset, such as JSUT, are significantly worse, especially for the WPT-Basic model. A different reference set contains recordings from a different setting usually with a different speaker, language, and recording equipment. It is possible that the fingerprint of a GAN on a different reference set differs significantly from the fingerprint, that the model was intended to learn. Against this, however, the evaluation of the DCNN did not yield such results [8]. Therefore, we have to assume that our models are overfitting to the LJSpeech reference set.

Network	Input	Parameters
Wide-16	*any*	344 574
Wide-19	*any*	112 526
Wide-24	*any*	946 526
Wide-32	*any*	639 838
WPT-Basic	DWPT-sym5	487 314
	DWPT-coif8	495 506
WPT-Bottle	DWPT-sym5	65 122
	DWPT-coif8	73 314
DCNN	STFT	239 015
	DWPT-db5	239 015
	DWPT-sym5	239 015
	DWPT-coif8	248 347
LCNN	STFT	–
	DWPT-sym5	3 312 450
AST	STFT	85 256 450
	DWPT-sym5	85 256 450

Table 3.9: Number of learnable parameters for the examined models. Numbers for the DCNN, LCNN and AST models are taken from Table 9 in [8].

In general, there are similarities in the recognition rates between the two models. It is noteworthy, that the WPT-Basic model only achieved lower recognition rates on Large BigVGAN samples compared to all other recognition rates. Especially the recognition rates for BigVGAN and Avocodo remained equal to other generators, which underlines the ability to detect novel generative models.

When evaluating the performance of different deep learning models, it is also useful to compare their computational complexity. Table 3.9 lists the number of learnable parameters of the different models. Our models have a large number of convolutional layers compared to the other models. However, they do not have a large number of parameters, which underlines the benefit of the residual learning framework employing identity mappings. The overall performance of CNN-based models in combination with their comparatively low number of parameters speaks for their well-known efficiency. The enormous number of parameters in the AST is not unusual for transformer models, but shows that equally good results can be achieved with significantly fewer parameters.

The remaining paragraphs are dedicated to a discussion of future work, including improvements to the proposed models. It is first and foremost important to acknowledge the possibility of better recognition rates when using SWT inputs by employing different CNN architectures. While residual blocks generally seemed to fit well, perhaps not all layers should consist of residual blocks. Because of the shape of the SWT coefficients, the first convolutional layers are required to substantially reduce the time dimension. While we mainly used strided convolutions for this, other techniques may be more suitable. The Max-Feature-Maps [22] of the LCNN could be a good start for further investigations.

In addition, the overfitting to the reference set of our models must be reduced. This could potentially be achieved by introducing dropout [37] layers. Furthermore, a different preprocessing could improve the training progress. We already experimented with omitting the log-transform step, which increased recognition rates on the trained detector but resulted in overall worse generalization (see appendix B). A proven method to improve generalization is dataset augmentation [10, Chapter 7.4]. In regard to audio data, one may consider adding noise, filters, or pitch variations. Beyond that, a multi-speaker dataset may also enhance the overall generalization capabilities to unseen inputs.

A particularly interesting direction of further investigation of the SWT is its combination with the DWPT. We have already briefly mentioned this extension of the SWT in the theoretical part of this thesis. While it is described in the literature [31, Chapter 6.6], there are currently no such implementations in practice [24]. This combination could offer the best of both worlds, namely invariance to shifts in the input and a detailed analysis across the entire frequency spectrum.

Lastly, the AST has not yet been examined using SWT inputs. This may be worth considering in future work. Overall, the proposed options represent just a few of the many avenues for future work in audio deepfake detection. As the need for reliable detectors continues into the future, we can expect the development of many innovative and exciting solutions.

Chapter 4

Conclusion

The first part of this thesis provided a theoretical background to wavelet transforms and convolutional neural networks. The second part involved the design and examination of neural networks for the detection of audio deepfakes. Inspired by recent work, we were able to extract fingerprints of all generator networks within our dataset using the stationary wavelet transform. A detailed examination of the fingerprints revealed that these are also reflected in the coefficients of the SWT. However, as the fingerprints are mainly to be expected in the high-frequency part of a signal, it may well be that the SWT is not the most suitable method. This is due to the nature of the SWT, which does not perform a detailed analysis of high-frequency components. Instead, the upper half of the frequency spectrum is only analyzed once by the wavelet filter.

Nevertheless, we have built functional detectors based on convolutional neural networks that process SWT coefficients. However, in comparison to other detectors, they did not achieve the desired recognition rates. We suspect that this is mainly due to the lack of high frequency detail in the SWT. In addition, the shape of the SWT coefficients was found to be unfavorable for two-dimensional CNNs. The advantage of the SWT being invariant to small shifts in time does not appear to show a notable positive impact.

In summary, we concluded that the coefficients of the discrete wavelet packet transform are more suitable. First, because the DWPT captures more detailed information from high-frequency components of a signal. This additional information is represented by additional coefficients in the output of the DWPT. As hypothesized, these provide valuable features to a CNN-based detector for audio deepfakes. Second, because the shape of the DWPT coefficients is better suited for two-dimensional CNNs. The results of our experiments showed that residual CNNs processing DWPT coefficients achieve better recognition rates than residual CNNs processing an SWT input.

In general, the choice of a CNN-based detector is advantageous because they can achieve good recognition rates with a relatively small number of parameters. CNN-based detectors that process a signal transformed with the STFT or DWPT currently achieve the most promising results. Nevertheless, it should be acknowledged that other architectures, such as transformers, could potentially become more effective detectors in the future.

As generative networks continue to evolve, there is also a growing interest in enhancing the performance of detectors. This poses a great challenge for the research community and a seemingly endless number of research opportunities.

Bibliography

[1] Cs231n convolutional neural networks for visual recognition - setting up the data and the model. `https://cs231n.github.io/neural-networks-2/`. Accessed: 2024-06-25.

[2] Cs231n convolutional neural networks for visual recognition - transfer learning. `https://cs231n.github.io/transfer-learning/`. Accessed: 2024-06-25.

[3] Taejun Bak, Junmo Lee, Hanbin Bae, Jinhyeok Yang, Jae-Sung Bae, and Young-Sun Joo. Avocodo: Generative adversarial network for artifact-free vocoder. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12562–12570, 2023.

[4] Ingrid Daubechies. *Ten lectures on wavelets*. SIAM, 1992.

[5] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[6] Joel Frank and Lea Schönherr. Wavefake: A data set to facilitate audio deepfake detection. *arXiv preprint arXiv:2111.02813*, 2021.

[7] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[8] Konstantin Gasenzer and Moritz Wolter. Towards generalizing deep-audio fake detection networks. *Transactions on Machine Learning Research*, 2024.

[9] Yuan Gong, Yu-An Chung, and James Glass. Ast: Audio spectrogram transformer. *arXiv preprint arXiv:2104.01778*, 2021.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[13] Matthew Groh, Ziv Epstein, Chaz Firestone, and Rosalind Picard. Deepfake detection by human crowds, machines, and machine-informed crowds. *Proceedings of the National Academy of Sciences*, 119(1):e2110013119, 2022.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Keith Ito and Linda Johnson. The lj speech dataset. `https://keithito.com/LJ-Speech-Dataset/`, 2017.

[16] Jee-weon Jung, Seung-bin Kim, Hye-jin Shim, Ju-ho Kim, and Ha-Jin Yu. Improved rawnet with feature map scaling for text-independent speaker verification using raw waveforms. *arXiv preprint arXiv:2004.00526*, 2020.

[17] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. In *International Conference on Machine Learning*, pages 2410–2419. PMLR, 2018.

[18] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[19] Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in Neural Information Processing Systems*, 33:17022–17033, 2020.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[21] Kundan Kumar, Rithesh Kumar, Thibault De Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre De Brebisson, Yoshua Bengio, and Aaron C Courville. Melgan: Generative adversarial networks for conditional waveform synthesis. *Advances in neural information processing systems*, 32, 2019.

[22] Galina Lavrentyeva, Sergey Novoselov, Egor Malykh, Alexander Kozlov, Oleg Kudashev, and Vadim Shchemelinin. Audio replay attack detection with deep learning frameworks. In *Interspeech*, pages 82–86, 2017.

[23] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[24] Gregory Lee, Ralf Gommers, Filip Waselewski, Kai Wohlfahrt, and Aaron O'Leary. Pywavelets: A python package for wavelet analysis. *Journal of Open Source Software*, 4(36):1237, 2019.

[25] Sang-gil Lee, Wei Ping, Boris Ginsburg, Bryan Catanzaro, and Sungroh Yoon. Bigvgan: A universal neural vocoder with large-scale training. *arXiv preprint arXiv:2206.04658*, 2022.

[26] Xuechen Liu, Xin Wang, Md Sahidullah, Jose Patino, Héctor Delgado, Tomi Kinnunen, Massimiliano Todisco, Junichi Yamagishi, Nicholas Evans, Andreas Nautsch, et al. Asvspoof 2021: Towards spoofed and deepfake speech detection in the wild. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2023.

[27] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[28] Kimberly T Mai, Sergi Bray, Toby Davies, and Lewis D Griffin. Warning: humans cannot reliably detect speech deepfakes. *Plos one*, 18(8):e0285333, 2023.

[29] Francesco Marra, Diego Gragnaniello, Luisa Verdoliva, and Giovanni Poggi. Do gans leave artificial fingerprints? In *2019 IEEE conference on multimedia information processing and retrieval (MIPR)*, pages 506–511. IEEE, 2019.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[31] Donald B Percival and Andrew T Walden. *Wavelet methods for time series analysis*, volume 4. Cambridge university press, 2000.

[32] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019.

[33] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

[34] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[35] Md Sahidullah, Tomi Kinnunen, and Cemal Hanilçi. A comparison of features for synthetic speech detection. 2015.

[36] Ryosuke Sonobe, Shinnosuke Takamichi, and Hiroshi Saruwatari. Jsut corpus: free large-scale japanese speech corpus for end-to-end speech synthesis. *arXiv preprint arXiv:1711.00354*, 2017.

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

[38] Massimiliano Todisco, Xin Wang, Ville Vestman, Md Sahidullah, Héctor Delgado, Andreas Nautsch, Junichi Yamagishi, Nicholas Evans, Tomi Kinnunen, and Kong Aik Lee. Asvspoof 2019: Future horizons in spoofed and fake audio detection. *arXiv preprint arXiv:1904.05441*, 2019.

[39] Moritz Wolter, Felix Blanke, Jochen Garcke, and Charles Tapley Hoyt. ptwt - the pytorch wavelet toolbox. *Journal of Machine Learning Research*, 25(80):1–7, 2024.

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

[41] Geng Yang, Shan Yang, Kai Liu, Peng Fang, Wei Chen, and Lei Xie. Multi-band melgan: Faster waveform generation for high-quality text-to-speech. In *2021 IEEE Spoken Language Technology Workshop (SLT)*, pages 492–498. IEEE, 2021.

[42] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[43] Ning Yu, Larry S Davis, and Mario Fritz. Attributing fake images to gans: Learning and analyzing gan fingerprints. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7556–7566, 2019.

[44] Tao Zhang. Deepfake generation and detection, a survey. *Multimedia Tools and Applications*, 81(5):6259–6276, 2022.

Appendices

Appendix A

Complete Evaluation Results

Wide-16

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-16	SWT-haar	60.63	57.69 $\pm$ 2.54	0.386	0.417 $\pm$ 0.027
	SWT-db3	67.58	64.75 $\pm$ 2.83	0.297	0.342 $\pm$ 0.028
	SWT-db4	69.08	66.29 $\pm$ 2.12	0.299	0.331 $\pm$ 0.022
	SWT-db5	67.25	65.70 $\pm$ 1.33	0.327	0.341 $\pm$ 0.013
	SWT-db7	66.64	65.51 $\pm$ 1.06	0.328	0.340 $\pm$ 0.013
	SWT-db8	68.39	67.49 $\pm$ 1.19	0.322	0.334 $\pm$ 0.011
	SWT-sym3	67.58	64.75 $\pm$ 2.83	0.297	0.342 $\pm$ 0.028
	SWT-sym4	67.80	67.12 $\pm$ 0.71	0.283	0.327 $\pm$ 0.027
	SWT-sym5	69.21	67.38 $\pm$ 2.28	0.270	0.315 $\pm$ 0.040
	SWT-sym7	71.56	68.74 $\pm$ 1.72	0.274	0.312 $\pm$ 0.025
	SWT-sym8	67.95	67.20 $\pm$ 0.58	0.302	0.330 $\pm$ 0.019
	SWT-sym9	68.64	67.91 $\pm$ 0.68	0.301	0.315 $\pm$ 0.012
	SWT-coif3	67.97	66.12 $\pm$ 2.21	0.328	0.347 $\pm$ 0.014
	SWT-coif4	69.12	67.27 $\pm$ 1.56	0.279	0.314 $\pm$ 0.024
	SWT-coif5	68.62	67.17 $\pm$ 1.38	0.307	0.324 $\pm$ 0.010
	SWT-coif7	69.31	67.25 $\pm$ 1.46	0.290	0.325 $\pm$ 0.021
	SWT-coif8	69.76	68.61 $\pm$ 0.87	0.312	0.322 $\pm$ 0.011
	SWT-coif9	70.04	69.06 $\pm$ 1.17	0.286	0.294 $\pm$ 0.005
GMM [6]	LFCC	–	–	**0.062**	–
RawNet2 [16, 6]	raw	–	–	0.363	–

Table A.1: Complete evaluation results of the Wide-16 residual CNN on the original WaveFake dataset including JSUT (R1, R2, A1, A2). The second column presents the different input types processed by the networks. Results highlighted in blue represent the best values within their respective columns, ignoring the baseline models.

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-16	SWT-haar	61.45	58.85 $\pm$ 2.41	0.377	0.403 $\pm$ 0.022
	SWT-db3	69.22	66.46 $\pm$ 3.01	0.284	0.321 $\pm$ 0.025
	SWT-db4	70.42	68.07 $\pm$ 1.98	0.284	0.309 $\pm$ 0.018
	SWT-db5	68.84	67.80 $\pm$ 0.95	0.306	0.317 $\pm$ 0.009
	SWT-db7	68.84	67.60 $\pm$ 1.10	0.305	0.316 $\pm$ 0.010
	SWT-db8	70.28	69.29 $\pm$ 1.27	0.300	0.310 $\pm$ 0.011
	SWT-sym3	69.22	66.46 $\pm$ 3.01	0.284	0.321 $\pm$ 0.025
	SWT-sym4	69.73	68.75 $\pm$ 1.02	0.274	0.307 $\pm$ 0.021
	SWT-sym5	70.41	68.92 $\pm$ 1.91	0.265	0.298 $\pm$ 0.030
	SWT-sym7	72.59	70.50 $\pm$ 1.25	0.263	0.292 $\pm$ 0.019
	SWT-sym8	69.51	69.10 $\pm$ 0.35	0.287	0.307 $\pm$ 0.015
	SWT-sym9	69.67	69.38 $\pm$ 0.38	0.285	0.297 $\pm$ 0.010
	SWT-coif3	69.76	68.03 $\pm$ 2.06	0.306	0.322 $\pm$ 0.013
	SWT-coif4	70.64	69.17 $\pm$ 1.31	0.270	0.297 $\pm$ 0.019
	SWT-coif5	70.30	68.95 $\pm$ 1.46	0.288	0.303 $\pm$ 0.009
	SWT-coif7	70.75	69.04 $\pm$ 1.46	0.276	0.303 $\pm$ 0.016
	SWT-coif8	70.90	69.90 $\pm$ 0.79	0.294	0.304 $\pm$ 0.007
	SWT-coif9	71.47	70.34 $\pm$ 1.36	0.274	0.282 $\pm$ 0.005
GMM [6, 8]	LFCC	–	–	**0.145**	–

Table A.2: Complete evaluation results of the Wide-16 residual CNN on the extended WaveFake dataset including JSUT (R1, R2, A1, A2, B2). The second column presents the different input types processed by the networks. Results highlighted in blue represent the best values within their respective columns, ignoring the baseline model.

Wide-19

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-19	SWT-haar	59.01	55.29 $\pm$ 3.44	0.406	0.433 $\pm$ 0.022
	SWT-db3	67.66	64.84 $\pm$ 3.24	0.315	0.343 $\pm$ 0.021
	SWT-db4	67.87	64.92 $\pm$ 3.22	0.314	0.334 $\pm$ 0.019
	SWT-db5	66.68	64.98 $\pm$ 1.33	0.324	0.341 $\pm$ 0.018
	SWT-db7	68.35	65.37 $\pm$ 1.86	0.304	0.338 $\pm$ 0.021
	SWT-db8	67.14	66.11 $\pm$ 1.11	0.325	0.346 $\pm$ 0.013
	SWT-sym3	67.66	64.84 $\pm$ 3.24	0.315	0.343 $\pm$ 0.021
	SWT-sym4	66.94	64.89 $\pm$ 1.34	0.312	0.344 $\pm$ 0.021
	SWT-sym5	68.25	65.37 $\pm$ 1.93	0.303	0.339 $\pm$ 0.023
	SWT-sym7	66.87	66.27 $\pm$ 0.56	0.307	0.325 $\pm$ 0.019
	SWT-sym8	68.12	66.50 $\pm$ 1.88	0.307	0.327 $\pm$ 0.016
	SWT-sym9	68.10	67.56 $\pm$ 0.39	0.289	0.319 $\pm$ 0.017
	SWT-coif3	67.70	65.75 $\pm$ 1.28	0.313	0.334 $\pm$ 0.019
	SWT-coif4	67.16	65.85 $\pm$ 1.31	0.321	0.342 $\pm$ 0.015
	SWT-coif5	67.63	66.00 $\pm$ 2.20	0.326	0.337 $\pm$ 0.010
	SWT-coif7	69.07	67.43 $\pm$ 1.29	0.293	0.319 $\pm$ 0.017
	SWT-coif8	67.53	66.54 $\pm$ 1.22	0.311	0.333 $\pm$ 0.015
	SWT-coif9	69.00	68.06 $\pm$ 0.62	0.270	0.308 $\pm$ 0.022
GMM [6]	LFCC	–	–	**0.062**	–
RawNet2 [16, 6]	raw	–	–	0.363	–

Table A.3: Complete evaluation results of the Wide-19 residual CNN on the original WaveFake dataset including JSUT (R1, R2, A1, A2). The second column presents the different input types processed by the networks. Results highlighted in blue represent the best values within their respective columns, ignoring the baseline models.

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-19	SWT-haar	60.04	56.16 $\pm$ 3.72	0.405	0.423 $\pm$ 0.020
	SWT-db3	69.17	66.18 $\pm$ 3.35	0.300	0.323 $\pm$ 0.019
	SWT-db4	69.52	66.42 $\pm$ 3.26	0.300	0.315 $\pm$ 0.016
	SWT-db5	67.83	66.77 $\pm$ 0.83	0.309	0.320 $\pm$ 0.012
	SWT-db7	69.99	67.07 $\pm$ 1.84	0.288	0.318 $\pm$ 0.018
	SWT-db8	69.00	67.95 $\pm$ 1.26	0.304	0.323 $\pm$ 0.011
	SWT-sym3	69.17	66.18 $\pm$ 3.35	0.300	0.323 $\pm$ 0.019
	SWT-sym4	68.36	66.43 $\pm$ 1.29	0.299	0.324 $\pm$ 0.015
	SWT-sym5	69.98	67.17 $\pm$ 1.71	0.288	0.319 $\pm$ 0.019
	SWT-sym7	68.89	68.06 $\pm$ 0.72	0.294	0.306 $\pm$ 0.014
	SWT-sym8	69.82	68.25 $\pm$ 1.58	0.294	0.308 $\pm$ 0.012
	SWT-sym9	69.29	68.96 $\pm$ 0.27	0.283	0.303 $\pm$ 0.011
	SWT-coif3	69.27	67.30 $\pm$ 1.16	0.298	0.315 $\pm$ 0.014
	SWT-coif4	69.08	67.65 $\pm$ 1.25	0.305	0.320 $\pm$ 0.012
	SWT-coif5	69.26	67.63 $\pm$ 2.25	0.308	0.316 $\pm$ 0.010
	SWT-coif7	70.56	69.21 $\pm$ 1.04	0.281	0.301 $\pm$ 0.013
	SWT-coif8	68.89	68.22 $\pm$ 0.73	0.301	0.313 $\pm$ 0.009
	SWT-coif9	70.71	69.67 $\pm$ 0.78	0.261	0.292 $\pm$ 0.018
GMM [6, 8]	LFCC	–	–	**0.145**	–

Table A.4: Complete evaluation results of the Wide-19 residual CNN on the extended WaveFake dataset including JSUT (R1, R2, A1, A2, B2). The second column presents the different input types processed by the networks. Results highlighted in blue represent the best values within their respective columns, ignoring the baseline model.

Wide-24

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-24	SWT-haar	61.44	59.56 $\pm$ 1.14	0.363	0.403 $\pm$ 0.025
	SWT-db3	70.06	67.49 $\pm$ 1.70	0.293	0.324 $\pm$ 0.027
	SWT-db4	68.89	67.76 $\pm$ 0.69	0.285	0.320 $\pm$ 0.021
	SWT-db5	68.90	67.19 $\pm$ 1.36	0.312	0.334 $\pm$ 0.014
	SWT-db7	68.58	68.13 $\pm$ 0.42	0.309	0.324 $\pm$ 0.014
	SWT-db8	69.69	68.58 $\pm$ 0.86	0.266	0.300 $\pm$ 0.022
	SWT-sym3	70.06	67.49 $\pm$ 1.70	0.293	0.324 $\pm$ 0.027
	SWT-sym4	68.93	67.91 $\pm$ 0.87	0.252	0.304 $\pm$ 0.032
	SWT-sym5	68.62	67.45 $\pm$ 1.16	0.281	0.328 $\pm$ 0.027
	SWT-sym7	72.43	70.24 $\pm$ 2.11	0.264	0.297 $\pm$ 0.030
	SWT-sym8	69.83	69.36 $\pm$ 0.34	0.292	0.301 $\pm$ 0.010
	SWT-sym9	69.68	68.37 $\pm$ 1.05	0.288	0.320 $\pm$ 0.019
	SWT-coif3	70.15	68.77 $\pm$ 0.90	0.266	0.293 $\pm$ 0.027
	SWT-coif4	69.59	69.05 $\pm$ 0.31	0.270	0.307 $\pm$ 0.024
	SWT-coif5	72.02	69.89 $\pm$ 1.78	0.271	0.296 $\pm$ 0.020
	SWT-coif7	70.14	69.95 $\pm$ 0.16	0.296	0.308 $\pm$ 0.012
	SWT-coif8	72.05	70.18 $\pm$ 1.47	0.271	0.296 $\pm$ 0.022
	SWT-coif9	70.27	69.77 $\pm$ 0.53	0.278	0.304 $\pm$ 0.018
GMM [6]	LFCC	–	–	**0.062**	–
RawNet2 [16, 6]	raw	–	–	0.363	–

Table A.5: Complete evaluation results of the Wide-24 residual CNN on the original WaveFake dataset including JSUT (R1, R2, A1, A2). The second column presents the different input types processed by the networks. Results highlighted in blue represent the best values within their respective columns, ignoring the baseline models.

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-24	SWT-haar	62.56	60.72 $\pm$ 1.15	0.358	0.390 $\pm$ 0.020
	SWT-db3	71.32	69.03 $\pm$ 1.62	0.281	0.305 $\pm$ 0.021
	SWT-db4	70.54	69.62 $\pm$ 0.65	0.273	0.298 $\pm$ 0.015
	SWT-db5	70.53	69.00 $\pm$ 1.12	0.292	0.310 $\pm$ 0.012
	SWT-db7	70.67	70.12 $\pm$ 0.48	0.288	0.300 $\pm$ 0.012
	SWT-db8	71.21	70.49 $\pm$ 0.73	0.256	0.283 $\pm$ 0.017
	SWT-sym3	71.32	69.03 $\pm$ 1.62	0.281	0.305 $\pm$ 0.021
	SWT-sym4	70.64	69.58 $\pm$ 0.84	0.245	0.287 $\pm$ 0.025
	SWT-sym5	70.14	69.24 $\pm$ 0.80	0.272	0.306 $\pm$ 0.020
	SWT-sym7	73.41	71.97 $\pm$ 1.42	0.256	0.277 $\pm$ 0.021
	SWT-sym8	71.27	70.95 $\pm$ 0.26	0.276	0.284 $\pm$ 0.008
	SWT-sym9	70.97	69.89 $\pm$ 0.77	0.276	0.299 $\pm$ 0.014
	SWT-coif3	71.53	70.33 $\pm$ 0.85	0.257	0.279 $\pm$ 0.021
	SWT-coif4	71.18	70.63 $\pm$ 0.36	0.259	0.289 $\pm$ 0.018
	SWT-coif5	72.74	71.38 $\pm$ 1.23	0.263	0.280 $\pm$ 0.015
	SWT-coif7	71.55	71.41 $\pm$ 0.16	0.280	0.289 $\pm$ 0.008
	SWT-coif8	73.03	71.51 $\pm$ 1.08	0.261	0.280 $\pm$ 0.016
	SWT-coif9	71.53	71.09 $\pm$ 0.47	0.273	0.287 $\pm$ 0.011
GMM [6, 8]	LFCC	–	–	**0.145**	–

Table A.6: Complete evaluation results of the Wide-24 residual CNN on the extended WaveFake dataset including JSUT (R1, R2, A1, A2, B2). The second column presents the different input types processed by the networks. Results highlighted in blue represent the best values within their respective columns, ignoring the baseline model.

Wide-32

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-32	SWT-haar	59.92	58.25 $\pm$ 1.21	0.372	0.402 $\pm$ 0.025
	SWT-db3	67.97	65.74 $\pm$ 1.77	0.316	0.334 $\pm$ 0.018
	SWT-db4	68.52	67.08 $\pm$ 1.30	0.276	0.312 $\pm$ 0.026
	SWT-db5	67.37	66.51 $\pm$ 1.26	0.305	0.323 $\pm$ 0.016
	SWT-db7	67.85	67.15 $\pm$ 0.43	0.314	0.322 $\pm$ 0.007
	SWT-db8	68.01	66.99 $\pm$ 0.68	0.317	0.343 $\pm$ 0.019
	SWT-sym3	67.97	65.74 $\pm$ 1.77	0.316	0.334 $\pm$ 0.018
	SWT-sym4	68.03	66.53 $\pm$ 1.27	0.313	0.327 $\pm$ 0.012
	SWT-sym5	67.28	66.35 $\pm$ 0.79	0.289	0.330 $\pm$ 0.025
	SWT-sym7	70.12	68.66 $\pm$ 1.38	0.280	0.312 $\pm$ 0.028
	SWT-sym8	69.26	68.14 $\pm$ 0.94	0.296	0.325 $\pm$ 0.024
	SWT-sym9	70.29	68.51 $\pm$ 1.09	0.267	0.306 $\pm$ 0.024
	SWT-coif3	67.94	66.83 $\pm$ 0.87	0.298	0.324 $\pm$ 0.015
	SWT-coif4	68.84	68.14 $\pm$ 0.64	0.316	0.323 $\pm$ 0.005
	SWT-coif5	69.96	68.81 $\pm$ 0.78	0.293	0.300 $\pm$ 0.009
	SWT-coif7	69.35	68.76 $\pm$ 0.69	0.304	0.315 $\pm$ 0.010
	SWT-coif8	68.95	68.09 $\pm$ 1.24	0.305	0.318 $\pm$ 0.009
	SWT-coif9	69.37	67.97 $\pm$ 1.18	0.291	0.313 $\pm$ 0.018
GMM [6]	LFCC	–	–	**0.062**	–
RawNet2 [16, 6]	raw	–	–	0.363	–

Table A.7: Complete evaluation results of the Wide-32 residual CNN on the original WaveFake dataset including JSUT (R1, R2, A1, A2). The second column presents the different input types processed by the networks. Results highlighted in blue represent the best values within their respective columns, ignoring the baseline models.

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Wide-32	SWT-haar	61.04	59.12 $\pm$ 1.35	0.374	0.394 $\pm$ 0.019
	SWT-db3	69.53	67.50 $\pm$ 1.48	0.298	0.315 $\pm$ 0.014
	SWT-db4	70.28	68.89 $\pm$ 1.23	0.267	0.295 $\pm$ 0.021
	SWT-db5	68.90	68.18 $\pm$ 1.09	0.291	0.305 $\pm$ 0.013
	SWT-db7	69.82	69.01 $\pm$ 0.49	0.297	0.302 $\pm$ 0.005
	SWT-db8	69.92	69.00 $\pm$ 0.53	0.298	0.317 $\pm$ 0.015
	SWT-sym3	69.53	67.50 $\pm$ 1.48	0.298	0.315 $\pm$ 0.014
	SWT-sym4	69.51	68.19 $\pm$ 1.04	0.296	0.307 $\pm$ 0.009
	SWT-sym5	69.00	68.03 $\pm$ 0.80	0.280	0.311 $\pm$ 0.019
	SWT-sym7	71.73	70.39 $\pm$ 1.32	0.265	0.292 $\pm$ 0.023
	SWT-sym8	70.58	69.89 $\pm$ 0.77	0.284	0.303 $\pm$ 0.018
	SWT-sym9	71.49	70.00 $\pm$ 1.01	0.260	0.290 $\pm$ 0.019
	SWT-coif3	69.00	68.55 $\pm$ 0.43	0.289	0.304 $\pm$ 0.009
	SWT-coif4	70.56	69.84 $\pm$ 0.84	0.301	0.302 $\pm$ 0.000
	SWT-coif5	71.36	70.40 $\pm$ 0.71	0.278	0.285 $\pm$ 0.007
	SWT-coif7	70.97	70.43 $\pm$ 0.56	0.286	0.295 $\pm$ 0.007
	SWT-coif8	70.59	69.93 $\pm$ 0.91	0.289	0.298 $\pm$ 0.006
	SWT-coif9	71.15	69.74 $\pm$ 1.02	0.276	0.295 $\pm$ 0.014
GMM [6, 8]	LFCC	–	–	**0.145**	–

Table A.8: Complete evaluation results of the Wide-32 residual CNN on the extended WaveFake dataset including JSUT (R1, R2, A1, A2, B2). The second column presents the different input types processed by the networks. Results highlighted in blue represent the best values within their respective columns, ignoring the baseline model.

WPT-Basic

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
WPT-Basic	DWPT-haar	67.77	65.14 ± 2.08	0.297	0.327 ± 0.028
	DWPT-db2	80.57	79.48 ± 0.77	0.165	0.175 ± 0.009
	DWPT-db3	88.00	84.99 ± 1.79	0.108	0.114 ± 0.004
	DWPT-db4	85.67	85.09 ± 0.40	0.077	0.084 ± 0.005
	DWPT-db5	86.03	85.54 ± 0.40	0.071	0.073 ± 0.002
	DWPT-db6	84.81	84.38 ± 0.49	0.078	0.081 ± 0.002
	DWPT-db7	84.75	84.23 ± 0.40	0.076	0.085 ± 0.008
	DWPT-db8	85.12	83.96 ± 0.80	0.077	0.087 ± 0.006
	DWPT-db9	85.68	85.23 ± 0.33	0.082	0.086 ± 0.003
	DWPT-db10	85.26	84.75 ± 0.35	0.081	0.085 ± 0.003
	DWPT-sym2	80.57	79.48 ± 0.77	0.165	0.175 ± 0.009
	DWPT-sym3	88.00	84.99 ± 1.79	0.108	0.114 ± 0.004
	DWPT-sym4	86.33	85.61 ± 0.53	0.081	0.085 ± 0.003
	DWPT-sym5	86.02	85.90 ± 0.18	0.071	0.073 ± 0.001
	DWPT-sym6	86.54	86.14 ± 0.33	0.067	0.072 ± 0.003
	DWPT-sym7	86.36	85.86 ± 0.57	0.055	0.067 ± 0.008
	DWPT-sym8	86.97	86.56 ± 0.26	0.063	0.070 ± 0.005
	DWPT-sym9	87.01	86.77 ± 0.19	0.064	0.069 ± 0.004
	DWPT-sym10	86.97	86.65 ± 0.27	0.060	0.065 ± 0.005
	DWPT-coif2	85.79	85.61 ± 0.17	0.069	0.081 ± 0.008
	DWPT-coif3	86.46	85.96 ± 0.52	0.067	0.071 ± 0.003
	DWPT-coif4	86.70	86.25 ± 0.37	0.065	0.071 ± 0.004
	DWPT-coif5	86.76	86.50 ± 0.25	0.075	0.080 ± 0.006
	DWPT-coif6	86.60	86.39 ± 0.20	0.075	0.088 ± 0.008
	DWPT-coif7	86.57	86.36 ± 0.17	0.071	0.087 ± 0.010
	DWPT-coif8	86.40	86.13 ± 0.22	0.074	0.087 ± 0.007
	DWPT-coif9	86.63	86.32 ± 0.20	0.073	0.090 ± 0.012
	DWPT-coif10	86.79	86.46 ± 0.25	0.073	0.086 ± 0.008
DCNN [8]	STFT	99.88	97.98 ± 3.18	**0.001**	0.099 ± 0.178
	DWPT-db5	96.36	94.39 ± 1.45	0.048	0.083 ± 0.041
	DWPT-sym5	97.60	95.57 ± 2.58	0.032	0.066 ± 0.035
	DWPT-coif8	98.81	97.87 ± 0.92	0.026	0.121 ± 0.089
LCNN [35, 8]	STFT	99.88	98.33 ± 1.85	**0.001**	0.019 ± 0.018
	DWPT-sym5	96.89	95.34 ± 1.83	0.037	0.085 ± 0.053
AST [9, 8]	STFT	99.37	98.31 ± 1.49	0.007	**0.018 ± 0.016**
	DWPT-sym5	93.63	91.98 ± 0.98	0.065	0.081 ± 0.010
GMM [6]	LFCC	–	–	0.062	–

Table A.9: Complete evaluation results of the WPT-Basic residual CNN on the original WaveFake dataset including JSUT (R1, R2, A1, A2).

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
WPT-Basic	DWPT-haar	65.84	63.32 $\pm$ 1.94	0.309	0.339 $\pm$ 0.028
	DWPT-db2	78.27	76.63 $\pm$ 1.34	0.172	0.187 $\pm$ 0.014
	DWPT-db3	86.66	85.13 $\pm$ 1.16	0.100	0.107 $\pm$ 0.005
	DWPT-db4	87.89	86.48 $\pm$ 1.09	0.070	0.076 $\pm$ 0.007
	DWPT-db5	88.10	87.04 $\pm$ 0.82	0.064	0.065 $\pm$ 0.001
	DWPT-db6	86.34	85.50 $\pm$ 0.71	0.072	0.073 $\pm$ 0.002
	DWPT-db7	86.00	85.12 $\pm$ 0.52	0.073	0.079 $\pm$ 0.006
	DWPT-db8	86.68	85.26 $\pm$ 0.86	0.072	0.078 $\pm$ 0.004
	DWPT-db9	87.21	86.70 $\pm$ 0.48	0.073	0.075 $\pm$ 0.001
	DWPT-db10	86.79	86.43 $\pm$ 0.37	0.070	0.073 $\pm$ 0.002
	DWPT-sym2	78.27	76.63 $\pm$ 1.34	0.172	0.187 $\pm$ 0.014
	DWPT-sym3	86.66	85.13 $\pm$ 1.16	0.100	0.107 $\pm$ 0.005
	DWPT-sym4	88.15	86.61 $\pm$ 1.15	0.071	0.077 $\pm$ 0.005
	DWPT-sym5	87.12	86.58 $\pm$ 0.33	0.061	0.066 $\pm$ 0.003
	DWPT-sym6	88.14	86.77 $\pm$ 0.79	0.059	0.066 $\pm$ 0.005
	DWPT-sym7	88.48	87.17 $\pm$ 0.95	0.050	0.058 $\pm$ 0.006
	DWPT-sym8	89.11	87.60 $\pm$ 1.01	0.057	0.060 $\pm$ 0.002
	DWPT-sym9	88.83	87.68 $\pm$ 0.79	0.056	0.060 $\pm$ 0.004
	DWPT-sym10	88.17	87.08 $\pm$ 0.82	0.053	0.059 $\pm$ 0.004
	DWPT-coif2	87.24	86.53 $\pm$ 0.73	0.065	0.074 $\pm$ 0.007
	DWPT-coif3	87.60	86.85 $\pm$ 0.66	0.060	0.066 $\pm$ 0.004
	DWPT-coif4	88.35	87.52 $\pm$ 0.73	0.058	0.062 $\pm$ 0.002
	DWPT-coif5	88.37	87.64 $\pm$ 0.46	0.066	0.069 $\pm$ 0.004
	DWPT-coif6	88.52	87.85 $\pm$ 0.59	0.065	0.074 $\pm$ 0.005
	DWPT-coif7	88.17	87.36 $\pm$ 0.58	0.064	0.075 $\pm$ 0.008
	DWPT-coif8	88.42	87.45 $\pm$ 0.71	0.065	0.074 $\pm$ 0.005
	DWPT-coif9	88.32	87.43 $\pm$ 0.57	0.065	0.078 $\pm$ 0.009
	DWPT-coif10	88.53	87.56 $\pm$ 0.59	0.065	0.075 $\pm$ 0.006
DCNN [8]	STFT	96.46	91.72 $\pm$ 2.94	0.036	0.159 $\pm$ 0.150
	DWPT-db5	96.88	94.65 $\pm$ 1.85	0.048	0.082 $\pm$ 0.042
	DWPT-sym5	97.70	95.25 $\pm$ 3.09	0.031	**0.069** $\pm$ **0.036**
	DWPT-coif8	98.72	97.39 $\pm$ 1.80	**0.026**	0.079 $\pm$ 0.047
LCNN [35, 8]	STFT	91.65	79.21 $\pm$ 16.55	0.083	0.169 $\pm$ 0.101
	DWPT-sym5	97.46	90.12 $\pm$ 6.44	0.067	0.108 $\pm$ 0.042
AST [9, 8]	STFT	90.98	87.10 $\pm$ 2.54	0.089	0.122 $\pm$ 0.021
	DWPT-sym5	93.49	91.25 $\pm$ 1.38	0.065	0.087 $\pm$ 0.013
GMM [6, 8]	LFCC	–	–	0.145	–

Table A.10: Complete evaluation results of the WPT-Basic residual CNN on the extended WaveFake dataset including JSUT (R1, R2, A1, A2, B2).

WPT-Bottle

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
WPT-Bottle	DWPT-haar	55.80	54.74 $\pm$ 1.40	0.432	0.450 $\pm$ 0.018
	DWPT-db2	77.53	75.43 $\pm$ 1.91	0.211	0.225 $\pm$ 0.023
	DWPT-db3	82.14	80.03 $\pm$ 1.37	0.128	0.136 $\pm$ 0.006
	DWPT-db4	83.85	82.32 $\pm$ 0.99	0.104	0.108 $\pm$ 0.003
	DWPT-db5	83.91	82.61 $\pm$ 1.12	0.096	0.103 $\pm$ 0.007
	DWPT-db6	83.42	82.57 $\pm$ 0.80	0.090	0.107 $\pm$ 0.011
	DWPT-db7	83.81	82.78 $\pm$ 0.95	0.100	0.112 $\pm$ 0.009
	DWPT-db8	83.85	83.04 $\pm$ 0.74	0.092	0.110 $\pm$ 0.013
	DWPT-db9	84.40	83.87 $\pm$ 0.61	0.097	0.114 $\pm$ 0.016
	DWPT-db10	83.91	83.46 $\pm$ 0.49	0.105	0.118 $\pm$ 0.012
	DWPT-sym2	77.53	75.43 $\pm$ 1.91	0.211	0.225 $\pm$ 0.023
	DWPT-sym3	82.14	80.03 $\pm$ 1.37	0.128	0.136 $\pm$ 0.006
	DWPT-sym4	85.14	83.70 $\pm$ 0.95	0.096	0.102 $\pm$ 0.004
	DWPT-sym5	85.36	84.35 $\pm$ 1.09	0.082	0.093 $\pm$ 0.011
	DWPT-sym6	85.56	84.78 $\pm$ 0.69	0.071	0.083 $\pm$ 0.007
	DWPT-sym7	85.54	85.09 $\pm$ 0.40	0.067	0.088 $\pm$ 0.016
	DWPT-sym8	86.44	86.19 $\pm$ 0.18	0.064	0.084 $\pm$ 0.014
	DWPT-sym9	89.29	87.44 $\pm$ 1.07	0.078	0.085 $\pm$ 0.008
	DWPT-sym10	86.48	86.13 $\pm$ 0.28	0.074	0.086 $\pm$ 0.009
	DWPT-coif2	84.54	84.16 $\pm$ 0.35	0.086	0.095 $\pm$ 0.005
	DWPT-coif3	85.49	84.88 $\pm$ 0.60	0.078	0.094 $\pm$ 0.011
	DWPT-coif4	85.85	85.37 $\pm$ 0.37	0.090	0.104 $\pm$ 0.011
	DWPT-coif5	86.01	85.69 $\pm$ 0.38	0.095	0.107 $\pm$ 0.008
	DWPT-coif6	88.37	86.47 $\pm$ 1.10	0.095	0.109 $\pm$ 0.011
	DWPT-coif7	88.05	86.62 $\pm$ 1.08	0.094	0.106 $\pm$ 0.007
	DWPT-coif8	88.97	86.68 $\pm$ 1.36	0.099	0.106 $\pm$ 0.007
	DWPT-coif9	87.89	86.46 $\pm$ 0.88	0.099	0.110 $\pm$ 0.008
	DWPT-coif10	87.95	86.09 $\pm$ 1.15	0.106	0.112 $\pm$ 0.004
DCNN [8]	STFT	99.88	97.98 $\pm$ 3.18	**0.001**	0.099 $\pm$ 0.178
	DWPT-db5	96.36	94.39 $\pm$ 1.45	0.048	0.083 $\pm$ 0.041
	DWPT-sym5	97.60	95.57 $\pm$ 2.58	0.032	0.066 $\pm$ 0.035
	DWPT-coif8	98.81	97.87 $\pm$ 0.92	0.026	0.121 $\pm$ 0.089
LCNN [35, 8]	STFT	99.88	98.33 $\pm$ 1.85	**0.001**	0.019 $\pm$ 0.018
	DWPT-sym5	96.89	95.34 $\pm$ 1.83	0.037	0.085 $\pm$ 0.053
AST [9, 8]	STFT	99.37	98.31 $\pm$ 1.49	0.007	**0.018 $\pm$ 0.016**
	DWPT-sym5	93.63	91.98 $\pm$ 0.98	0.065	0.081 $\pm$ 0.010
GMM [6]	LFCC	–	–	0.062	–

Table A.11: Complete evaluation results of the WPT-Bottle residual CNN on the original WaveFake dataset including JSUT (R1, R2, A1, A2)

Network	Input	Accuracy [%]		Average EER	
		max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
WPT-Bottle	DWPT-haar	55.24	54.31 ± 1.36	0.437	0.453 ± 0.018
	DWPT-db2	76.88	74.10 ± 2.63	0.212	0.237 ± 0.029
	DWPT-db3	83.93	82.18 ± 1.08	0.120	0.127 ± 0.006
	DWPT-db4	86.50	84.58 ± 1.44	0.089	0.096 ± 0.004
	DWPT-db5	86.80	85.13 ± 1.04	0.083	0.089 ± 0.005
	DWPT-db6	86.14	84.89 ± 0.78	0.084	0.092 ± 0.006
	DWPT-db7	85.68	84.73 ± 0.84	0.088	0.097 ± 0.005
	DWPT-db8	86.49	85.57 ± 0.75	0.079	0.092 ± 0.010
	DWPT-db9	87.17	86.26 ± 0.75	0.085	0.095 ± 0.010
	DWPT-db10	86.90	86.09 ± 0.73	0.087	0.096 ± 0.008
	DWPT-sym2	76.88	74.10 ± 2.63	0.212	0.237 ± 0.029
	DWPT-sym3	83.93	82.18 ± 1.08	0.120	0.127 ± 0.006
	DWPT-sym4	87.54	85.35 ± 1.35	0.082	0.094 ± 0.007
	DWPT-sym5	87.15	86.06 ± 1.26	0.074	0.083 ± 0.007
	DWPT-sym6	88.03	86.16 ± 1.52	0.072	0.076 ± 0.004
	DWPT-sym7	87.92	86.90 ± 0.86	0.058	0.077 ± 0.012
	DWPT-sym8	88.70	87.75 ± 1.01	0.062	0.072 ± 0.007
	DWPT-sym9	90.69	88.72 ± 1.33	0.067	0.073 ± 0.005
	DWPT-sym10	88.39	87.35 ± 1.05	0.071	0.075 ± 0.005
	DWPT-coif2	86.88	86.07 ± 0.86	0.078	0.086 ± 0.006
	DWPT-coif3	87.68	86.61 ± 1.02	0.078	0.083 ± 0.005
	DWPT-coif4	87.91	87.25 ± 0.90	0.074	0.087 ± 0.008
	DWPT-coif5	88.32	87.39 ± 1.13	0.078	0.089 ± 0.007
	DWPT-coif6	90.39	88.72 ± 0.97	0.077	0.088 ± 0.009
	DWPT-coif7	90.01	88.73 ± 0.92	0.077	0.086 ± 0.005
	DWPT-coif8	90.72	89.07 ± 1.00	0.077	0.085 ± 0.007
	DWPT-coif9	90.06	88.52 ± 1.04	0.079	0.089 ± 0.008
	DWPT-coif10	90.04	88.46 ± 0.91	0.084	0.090 ± 0.004
DCNN [8]	STFT	96.46	91.72 ± 2.94	0.036	0.159 ± 0.150
	DWPT-db5	96.88	94.65 ± 1.85	0.048	0.082 ± 0.042
	DWPT-sym5	97.70	95.25 ± 3.09	0.031	**0.069 ± 0.036**
	DWPT-coif8	98.72	97.39 ± 1.80	**0.026**	0.079 ± 0.047
LCNN [35, 8]	STFT	91.65	79.21 ± 16.55	0.083	0.169 ± 0.101
	DWPT-sym5	97.46	90.12 ± 6.44	0.067	0.108 ± 0.042
AST [9, 8]	STFT	90.98	87.10 ± 2.54	0.089	0.122 ± 0.021
	DWPT-sym5	93.49	91.25 ± 1.38	0.065	0.087 ± 0.013
GMM [6, 8]	LFCC	–	–	0.145	–

Table A.12: Complete evaluation results of the WPT-Bottle residual CNN on the extended WaveFake dataset including JSUT (R1, R2, A1, A2, B2)

Appendix B

No log-transform

The following Table shows the evaluation results of the Wide-24 model processing SWT coefficients computed by the `sym7` wavelet. It provides a comparison between two different training configurations, namely one with and one without the log-transform preprocessing step. The model without log-transformed coefficients achieves high recognition rates on the trained generator (Full-Band MelGAN) and its successors. However it does not generalize to the other generators.

Generator	EER			
	With log-transform		Without log-transform	
	min	$\mu \pm \sigma$	min	$\mu \pm \sigma$
Full-Band MelGAN	0.183	0.190 $\pm$ 0.005	0.030	0.043 $\pm$ 0.010
Avocodo	0.196	0.209 $\pm$ 0.014	0.271	0.334 $\pm$ 0.036
BigVGAN	0.203	0.209 $\pm$ 0.004	0.559	0.609 $\pm$ 0.032
HiFi-GAN	0.302	0.315 $\pm$ 0.009	0.441	0.457 $\pm$ 0.011
Large BigVGAN	0.243	0.253 $\pm$ 0.007	0.448	0.461 $\pm$ 0.011
MelGAN	0.234	0.239 $\pm$ 0.004	0.147	0.179 $\pm$ 0.031
Multi-Band MelGAN	0.267	0.273 $\pm$ 0.009	0.102	0.114 $\pm$ 0.012
Parallel WaveGAN	0.170	0.181 $\pm$ 0.008	0.512	0.586 $\pm$ 0.051
WaveGlow	0.156	0.170 $\pm$ 0.008	0.190	0.201 $\pm$ 0.008
Multi-Band MelGAN (JSUT)	0.394	0.476 $\pm$ 0.073	0.502	0.544 $\pm$ 0.043
Parallel WaveGAN (JSUT)	0.346	0.533 $\pm$ 0.149	0.464	0.595 $\pm$ 0.093

Table B.1: Evaluation of Wide-24-sym7 showing the recognition rates for each generator available in the extended WaveFake dataset.

Declaration of Authorship

I hereby confirm that I have written this thesis myself in compliance with the rules of good research practice, that I have not used any sources other than those indicated and that I have marked all citations as such.

Bonn, 27.06.2024