ERKENNUNG VON AUDIO-DEEPFAKES MITHILFE VON KONTINUIERLICHEN WAVELET-TRANSFORMATIONEN

KONSTANTIN GASENZER

Geboren am 11. September 1999 in Heidelberg

Bachelorarbeit Informatik

Betreuer: Dr. Moritz Wolter
Zweitgutachter: Prof. Dr. Reinhard Klein

Institut für Informatik II – Visual Computing

Mathematisch-Naturwissenschaftliche Fakultät
Rheinische Friedrich-Wilhelms-Universität Bonn

10. März 2023

INHALTSVERZEICHNIS

1 Theoretische Grundlagen 1
- 1.1 Zeitkontinuierliche Signale und Signalräume 1
  - 1.1.1 Signalräume 1
- 1.2 Fourieranalyse 3
  - 1.2.1 Fourierreihe periodischer Signale 3
  - 1.2.2 Fouriertransformation 4
  - 1.2.3 Wichtige Eigenschaften der FT 4
- 1.3 Kurzzeit-Fouriertransformation (STFT) 4
  - 1.3.1 Zeit-Frequenz-Analyse 5
- 1.4 Kontinuierliche Wavelet-Transformation (CWT) 6
  - 1.4.1 Definition der CWT 6
  - 1.4.2 Zeit-Skalen-Analyse 7
  - 1.4.3 Vorüberlegungen zur Implementierung der CWT 8
  - 1.4.4 Wavelets für die Audiosignalanalyse 9
- 1.5 Audio-Deepfakes und maschinelles Lernen 10
  - 1.5.1 Künstliche Neuronale Netze 10
  - 1.5.2 Backpropagation, Training und Validierung 11
  - 1.5.3 Faltungsnetze 15
  - 1.5.4 Modellinterpretation: Attribution 18
  - 1.5.5 Generierung und Erkennung von Deepfakes 19

2 Numerische Experimente 21
- 2.1 Entwicklungsumgebung und Implementierung 21
- 2.2 Der WaveFake-Datensatz 22
- 2.3 Untersuchung der Datensätze 22
  - 2.3.1 Spektro- und Skalogramme der Audio-Deepfakes 22
  - 2.3.2 Durchschnittliche Signalenergien der Datensätze 26
- 2.4 Deepfake-Erkennung 28
  - 2.4.1 Datenvorverarbeitung, Training und Testen 28
  - 2.4.2 Hyperparameter für die binäre Klassifikation 28
  - 2.4.3 Ergebnisse der Erkennung aus bekannten Quellen 30
  - 2.4.4 Ergebnisse der Erkennung aus unbekannten Quellen 35
  - 2.4.5 Ergebnisse des Vergleichs der STFT und CWT 37
  - 2.4.6 Ergebnisse der Modellinterpretation mit integrierten Gradienten 39

3 Schlussfolgerungen und Ausblick 41

A Anhang 43

Literatur 49

ABKÜRZUNGEN

FT	Fouriertransformation (engl.: fourier transform)
FFT	schnelle Fouriertransformation (engl.: fast fourier transform)
iFFT	inverse schnelle Fouriertransformation (engl.: inverse fast fourier transform)
STFT	Kurzzeit-Fouriertransformation (engl.: short time fourier transform)
WFT	gefensterte Fouriertransformation (engl.: windowed fourier transform)
DWT	diskrete Wavelettransformation (engl.: discrete wavelet transform)
DCT	diskrete Kosinustransformation (engl.: discrete cosinus transform)
CWT	kontinuierlichen Wavelettransformation (engl.: continuous wavelet transform)
LFCC	Linear Frequency Cepstral Coefficients (dt.: Lineare-Frequenz-Cepstrum-Koeffizienten)
CNN	Faltungsnetz (engl.: convolutional neural network)
ReLU	rektifizierende Aktivierungsfunktion (engl.: rectified linear unit)
GAN	Generative Adversarial Network (dt.: erzeugendes gegnerisches Netzwerk)
EER	Equal Error Rate (dt.: Gleichfehlerrate)
aEER	Average Equal Error Rate (dt.: durchschnittliche Gleichfehlerrate)
IG	Integrated Gradients (dt.: Integrierte Gradienten)
LDN	LearnDeepNet
LN	LearnNet
ODN	OneDNet

DANKSAGUNG

Gerne möchte ich hier meinen Dank all denjenigen ausdrücken, die mich auf dem Weg zum Abschluss dieser Arbeit begleitet haben. Zunächst bedanke ich mich bei meinem Betreuer Dr. Moritz Wolter für die vielen investierten Stunden, die ständige Begleitung und die interessanten Diskussionen. Des Weiteren möchte ich mich bei Prof. Dr. Reinhard Klein für die Zweitkorrektur bedanken. Darüber hinaus geht mein Dank an das High Performance Computing and Analytics Lab der Universität Bonn, dessen Großrechner *Bender* ich freundlicherweise für diese Arbeit verwenden konnte.

Last but not least gilt ein großer Dank meinen Eltern Maren Gasenzer, Thomas Gasenzer und meiner Freundin Franziska Munz für die Unterstützung im Schreibverlauf und die vielen Gespräche über Wavelets und Deepfakes, sowie auch meiner Schwester Sophia für die Bachelorarbeitskerze, die eine ständige Begleiterin war.

EINFÜHRUNG

„Hello everybody, and welcome to MIT 6.S191, the official introductory course on deep learning, taught here at MIT“. Mit diesem Satz beginnt das Einführungsvideo der Vorlesung *Deep Learning* am Massachusetts Institute of Technology auf der Videoplattform YouTube im Jahr 2022 [Ami22]. Bemerkenswert ist, dass diese Ansprache nicht vom Kursleiter selbst, sondern vom ehemaligen Präsident der Vereinigten Staaten Barack Obama gehalten wird, der in einer Videobotschaft zu sehen ist. Er beendet seine Ansprache mit dem Satz: „[...] In fact, this entire speech and video are not real and were created using deep learning and artificial intelligence“ [Ami22]. Dies ist ein sehr aktuelles Beispiel eines erzeugten Videos. Besonders in den letzten Jahren ist im Bereich des maschinellen Lernens unter den sogenannten *generativen künstlichen neuronalen Netzwerken* ein großer Zuwachs an performanten Technologien zu erkennen, die Bild-, Audio-, Text- und Videomaterial realistisch erzeugen können [Zha22]. Diese Technologien beschränken sich jedoch nicht nur auf den Einsatz z. B. in Apps auf mobilen Endgeräten, zur Bildbearbeitung oder zur Generierung von automatischen Antworttexten, sondern sie ebnen auch den Weg für eine Fülle von neuen Anwendungsgebieten. Darunter fällt auch die Möglichkeit, digitale Fakes, also gefälschte Inhalte zu erzeugen [Par12]. Insbesondere gehören dazu Audiosignale, die in Reinform oder als Teil von Videos im Zeitalter des Internets von großer Bedeutung für die Vermittlung von Inhalten sind. In der Literatur werden mit Deepfakes die mit Methoden des maschinellen Lernens (engl. Machine Learning) generierten Bild-, Audio- oder Videodaten bezeichnet, die realistisch erscheinen, jedoch nicht in der Realität aufgezeichnet wurden [Zha22]. Dazu werden Methoden des Deep Learning verwendet, ein Teilgebiet des maschinellen Lernens, welches sich mit mehrschichtigen Technologien beschäftigt.

Gefälschte Bild- und Audiosignale, die realistisch wirken und kaum von realen Daten zu unterscheiden sind, können weitreichende Implikationen für die Privatsphäre der Menschen, den Diskurs in demokratischen Gesellschaften oder auch die Sicherheit eines Landes haben [Zha22]. Insbesondere als Mittel der Manipulation sind Deepfakes nicht nur im politischen, sondern auch im privaten Kontext verbreitet und werden durch simple Schnittstellen wie Webseiten und Apps auch für viele Menschen einfach zugänglich. Diese Problematik macht die Diversifizierung neuer Deepfake-Erkennungsmethoden im Bereich der wissenschaftlichen Forschung zu einem relevanten Ziel.

Gängige Erkennungsalgorithmen basieren auf der Analyse verschiedenster Merkmale, die den Daten bzw. Signalen zugrunde liegen [Zha22]. Um die komplexen Informationen eines solchen Signals extrahieren zu können, werden in der Physik und Datenverarbeitungstechnik lineare Abbildungen verwendet. Als Standardmethode kommt die *Fouriertransformation* zum Einsatz, welche die spektralen Koeffizienten liefert, die mit den jeweiligen Frequenzen überlagert das analysierte Signal formen. Als wesentliches Problem er-

gibt sich im Anschluss an eine Fouriertransformation jedoch, dass sich die Information über eine zeit-räumliche Änderung etwa bestimmter Schwingungsformen nur durch eine Rücktransformation rekonstruieren lässt. In der Auseinandersetzung mit diesem Problem wurde zum Einen die sogenannte Kurzzeit-Fouriertransformation entwickelt, die nur kurze Abschnitte eines Signals Fourier-transformiert. Weiterhin wurden als Alternative, um dasselbe Ziel zu erreichen sogenannte Wavelets [Dau92] formalisiert, *kleine Wellenfunktionen*, die in der sogenannten Wavelettransformation zum Einsatz kommen und dort wie die Kurzzeit-Fouriertransformation eine Analyse von kontinuierlichen Signalen im Zeit- und Frequenzbereich möglich machen [Mer20].

Die vorliegende Arbeit widmet sich den beiden Lösungsansätzen und ist auf die Analyse von Audio-Deepfakes und die Bewertung der Deepfake-Erkennung von Audiodaten mittels tiefer neuronaler Netze fokussiert. Hierbei wird die kontinuierliche Wavelettransformation verwendet und mit der Kurzzeit-Fouriertransformation verglichen. Das Hauptaugenmerk liegt dabei auf verschiedenen Architekturen neuronaler Netze und Wavelets.

In Kapitel 1 werden zunächst die theoretischen Grundlagen der Signaltheorie und die Unterschiede zwischen den Transformationen vorgestellt. Daran schließt sich eine Erläuterung wichtiger Grundlagen aus dem Bereich des maschinellen Lernens an, und es werden im Speziellen der Aufbau und die Funktionsweise von *tiefen* neuronalen Netzen und Faltungsnetzen eingeführt. Im experimentellen Teil in Kapitel 2 wird dann ein Datensatz aus originalen Audiodateien und generierten Audio-Deepfakes unter Nutzung der Kurzzeit-Fourier- und der Wavelettransformation analysiert und die entsprechenden Resultate miteinander verglichen. Anschließend werden die vorgestellten neuronalen Netze verschiedener Kapazitäten auf dem wavelettransformierten Datensatz trainiert, um die Deepfakes als solche erkennen zu können. Die Ergebnisse werden vorgestellt und mit dem aktuellen Forschungsstand verglichen. Dabei wird auch die Fähigkeit der neuronalen Netze zur Generalisierung auf Audio-Deepfakes aus verschiedenen Quellen getestet, auch solchen, auf die das Netz nicht trainiert wurde. Um die theoretisch herausgearbeiteten Unterschiede aus der Verwendung der verschiedenen Transformationen experimentell zu testen, werden die neuronalen Netze mittels beider Verfahren separat trainiert und ihre Genauigkeit miteinander verglichen. Abschließend erfolgt ein Ausblick.

1 THEORETISCHE GRUNDLAGEN

In Natur und Technik lassen sich zeitkontinuierliche Signale in vielen Bereichen beobachten, beispielsweise in Schwingungen elektrischer Spannungen und Ströme sowie in Druckverläufen von Schallwellen [Mer20]. Diese Signale und ihre Analyse mittels Fourier- und Wavelettransformationen werden in den folgenden Abschnitten formal beschrieben, um eine Grundlage für die Analyse von Deepfake-Audiodaten zu vermitteln.

Vorausgesetzt werden Grundkenntnisse der Linearen Algebra sowie zu komplexen Zahlen und Funktionen.

1.1 ZEITKONTINUIERLICHE SIGNALE UND SIGNALRÄUME

Ein kontinuierliches Signal lässt sich als eine Funktion $f : \mathbb{R}^n \to \mathbb{C}$ einer oder mehrerer unabhängiger Variablen darstellen. Als Teilgebiet der Audiosignalverarbeitung beschränkt sich die Untersuchung der kontinuierlichen Wavelettransformation (engl.: continuous wavelet transform) (CWT) in dieser Arbeit auf eindimensionale zeitkontinuierliche Signale, also Funktionen der Zeit [OW92, S. 8].

1.1.1 *Signalräume*

Zeitkontinuierliche Signale lassen sich als Vektor eines linearen Vektorraumes interpretieren. Um ein solches Signal zeitlich und räumlich beschreiben zu können, werden von Normen induzierte metrische Vektorräume eingeführt. Das folgende Unterkapitel orientiert sich dabei an Mertins [Mer20].

Vielfach verwendete Normen für zeitkontinuierliche Signale sind sogenannte $\mathcal{L}_p$-Normen für messbare Funktionen f und ein $p \in [1, \infty)$, welche innerhalb der Signalgrenzen a und b betrachtet werden, mit der Form [Mer20]

$$||f||_{\mathcal{L}_p} = \left[\int_a^b |f(t)|^p \, \mathrm{d}t \right]^{1/p}. \tag{1.1}$$

Mithilfe der $\mathcal{L}_p$-Normen lassen sich die Lebesgueschen Räume definieren als [Mer20]

$$\mathcal{L}_p(\mathbb{R}) := \{ f : \mathbb{R} \to \mathbb{C} \mid f \text{ messbar und } ||f||_{\mathcal{L}_p} < \infty \}, \tag{1.2}$$

worin die Messbarkeit bedeutet, dass den Urbildern aller abgeschlossenen Untermengen von $\mathbb{C}$ ein Maß zugeordnet ist. Dies ist für die in dieser Arbeit untersuchten stetigen Signale stets der Fall. In der Signalverarbeitung ist insbesondere der $\mathcal{L}_2$-Raum interessant, auf dem sich das Skalarprodukt zweier Signale $x, y \in \mathcal{L}_2(a, b) \equiv \mathcal{L}_2([a, b])$ definieren lässt als [Mer20]

$$\langle x, y \rangle = \int_a^b x(t) \, y^*(t) \, \mathrm{d}t \tag{1.3}$$

Warum ist der $\mathcal{L}_2$ interessant? Als vollständiger linearer Raum mit Skalarprodukt ist $\mathcal{L}_2(a,b)$ ein sogenannter Hilbertraum [Mer20, S. 30]. Aufgrund ihrer starken mathematischen Struktur sind Hilberträume nicht nur in der Signalverarbeitung, sondern allgemein in der Analysis von wichtiger Bedeutung [Wer18, S. 219].

Da $\mathcal{L}_2(a,b)$ vollständig ist [Wer18, S. 221], lässt sich das mathematische Hilfsmittel der orthogonalen Projektion für ein Signal $f \in \mathcal{L}_2(a,b)$ anwenden und sich dieses mittels des Skalarprodukts als Linearkombination der Basisvektoren φ_i mit $i \in \{1 \dots n\}$ eines vollständigen orthonormalen Funktionensystems approximieren über [Mer20]

$$f_n = \sum_{i=1}^{n} \langle f, \varphi_i \rangle \varphi_i \quad , \quad f = \lim_{n \to \infty} f_n \,. \tag{1.4}$$

Bei $\langle f, \varphi_i \rangle$ handelt es sich um das Skalarprodukt wie in (1.3) definiert und somit um die orthogonale Projektion von f auf φ_i. Je größer $n \in \mathbb{N}$ ist, desto genauer ist die Annäherung. Komplexe Exponentialfunktionen gehören zu der Klasse der vollständigen orthonormalen Funktionensysteme und dienen somit als probates Mittel in der Wahl der Basisvektoren φ_i. So kann ein zeitkontinuierliches Signal mittels einer Reihenentwicklung in kompakter Form dargestellt werden, die eine Analyse der im Signal enthaltenen Frequenzen ω_i ermöglicht.

Da das Skalarprodukt sowohl für die CWT als auch für künstliche neuronale Netze eine maßgebliche Rolle spielt, sei noch kurz erwähnt, dass das Skalarprodukt aus (1.3) allgemein für einen Vektorraum $\mathbb{C}^{n \times 1}$ der Dimension $n \in \mathbb{N}$ und $\mathbf{x}, \mathbf{y} \in \mathbb{C}^{n \times 1}$ definiert ist als [Mer20]

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{y}^H \mathbf{x} \,, \tag{1.5}$$

wobei es sich bei $\mathbf{x}$ und $\mathbf{y}$ um $(n \times 1)$-Matrizen handelt und $\mathbf{x}^H$ die adjungierte Matrix von $\mathbf{x}$ ist. Das Standardskalarprodukt in Matrixdarstellung ist außerdem von dem Hadamard-Produkt, gekennzeichnet durch $\odot$, zweier Matrizen abzugrenzen, da es keine elementweise Summe bildet, sondern die Matrizen elementweise multipliziert. Für zwei $(m \times n)$-Matrizen ist es definiert als [VA07]

$$\forall i \in \{1, 2, \dots, m\} : \forall j \in \{1, 2, \dots, n\} : (\mathbf{x} \odot \mathbf{y})_{ij} = x_{ij} \cdot y_{ij} \,, \tag{1.6}$$

wobei $m, n \in \mathbb{N}$. In dieser Arbeit ist x_{ij} das Element einer Matrix $\mathbf{x}$ in der Zeile i und Spalte j. Zur visuellen Abgrenzung sind Matrizen **fett** gedruckt und Skalare nicht.
Ein weiteres für diese Arbeit wichtiges mathematisches Hilfsmittel ist die *Faltung*. Für zwei Funktionen $f, g \in \mathcal{L}_1(\mathbb{R})$ ist sie $\forall y \in \mathbb{R}$ definiert durch [For12]

$$(f * g)(y) := \int_{\mathbb{R}} f(x) g(y - x) \, \mathrm{d}x \,. \tag{1.7}$$

1.2 FOURIERANALYSE

1.2.1 *Fourierreihe periodischer Signale*

Eine mit der Fouriertransformation eng zusammenhängende Signalkonstruktion gemäß Gleichung (1.4) ist die Fourier-Reihenentwicklung periodischer Signale. Die Periode T eines periodischen Signals beschreibt die in festen Abständen auftretende Wiederholung des bisherigen Signalverlaufs, also $\forall t \in \mathbb{R} : x(t) = x(t+T)$ für ein in der Zeit t kontinuierliches Signal $x(t)$. In der Fourieranalyse werden meist trigonometrische Funktionen wie Sinus- und Kosinusfunktionen als Basis der Fourierreihe gewählt. Beide haben die Periode $T = 2\pi$ und werden daher 2π-periodisch genannt. Da sich Sinus- und Kosinusfunktionen aufgrund der Eulerschen Formel mittels der komplexen Exponentialfunktion mit imaginärer Zahl i kompakt darstellen lassen, wird diese Form in dieser Arbeit verwendet. Die Basisvektoren der Fourierreihe für ein im Zeitintervall $[-\frac{T}{2}, \frac{T}{2}]$ betrachtetes periodisches Signal $x(t) \in \mathcal{L}_2(-\frac{T}{2}, \frac{T}{2})$ der Zeit t ergeben sich somit als [Mer20] [Mer20]

$$\varphi_k(t) = \frac{1}{\sqrt{T}} \, \mathrm{e}^{\frac{2\pi \mathrm{i} k t}{T}} \, . \tag{1.8}$$

Bei $\frac{T}{k}$ mit $k \in \mathbb{Z} \setminus \{0\}$ handelt es sich um die Periodenlänge, also den räumlichen Abstand von sich wiederholenden Signalverläufen. Der in der Exponentialfunktion enthaltene Term $\frac{k}{T}$ beschreibt also die Frequenz der k-ten Oberschwingung, eines Vielfachen der Frequenz $f = \frac{1}{T}$ der Grundschwingung [Mes15, S. 150]. Dies ergibt mit (1.4) die Fourierreihe, eine gewichtete Superposition von Grund- und Oberschwingungen in der Zeit $t \in \mathbb{R}$, mit der Kreisfrequenz $\omega = \frac{2\pi}{T}$ der Grundschwingung und den sogenannten Fourierkoeffizienten x_k, der Form [Mer20]

$$x(t) = \sum_{k=-\infty}^{\infty} x_k \, \mathrm{e}^{\mathrm{i}\omega k t} \, . \tag{1.9}$$

Wenn im folgenden von Fourieranalyse eines (Audio-)Signals gesprochen wird, ist damit insbesondere die Berechnung dieser Fourierkoeffizienten x_k gemeint, und damit die Annäherung des Signals mithilfe der Fourierdarstellung. Im Anschluss an die Analyse lässt sich mittels (1.9) das Audiosignal rekonstruieren [Kai94]. Im betrachteten Intervall können die Fourierkoeffizienten des periodischen Signals mittels der Umkehrtransformation (Projektion von $x(t)$ auf die Basisfunktionen, vgl. (1.4)) [Mer20]

$$x_k = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} x(t) \, \mathrm{e}^{-\mathrm{i}\omega k t} \, \mathrm{d}t \tag{1.10}$$

berechnet werden. Das zugehörige Frequenz-Amplituden-Diagramm nennt sich Linien*spektrum* und ist diskret [Mes15].

Im Bereich der Signalverarbeitung handelt es sich jedoch meist um aperiodische Signale. Der Signalverlauf wiederholt sich also nicht, weshalb die Periode unendlich wird. Da die Oberfrequenzen im Abstand $\frac{1}{T}$ zueinander

stehen, entsteht bei $T \to \infty$ aus dem diskreten Linienspektrum ein kontinuierliches Spektrum [Mes15], das auch als Fouriertransformation (engl.: fourier transform) (FT) bekannt ist.

1.2.2 *Fouriertransformation*

Wie zuvor erwähnt, geht für den Grenzübergang $T \to \infty$ auch $k \to \infty$, weshalb sich das Linienspektrum aus x_k in (1.10) durch Integration über die Zeit t in ein kontinuierliches Spektrum verwandelt. Dieses ist definiert über die Fouriertransformierte [Mes15]

$$X(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x(t)\, \mathrm{e}^{-\mathrm{i}\omega t}\, \mathrm{d}t\,. \tag{1.11}$$

Anschaulich wird ein zeitabhängiges Signal $x(t)$ in eine frequenzabhängige Funktion $X(\omega)$ transformiert, weshalb die Berechnung der Transformierten aus dem Signal auch Fouriertransformation genannt wird [Mer20]. Analog zur Fourierreihe kann mithilfe der Fouriertransformierten das ursprüngliche Signal rekonstruiert bzw. als Superposition gewichteter Exponentialfunktionen dargestellt werden. Aus der Fourierreihe wird das zu (1.11) symmetrische Fourierintegral [Mes15]

$$x(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} F(\omega)\, \mathrm{e}^{\mathrm{i}\omega t}\, \mathrm{d}\omega\,. \tag{1.12}$$

1.2.3 *Wichtige Eigenschaften der FT*

[Mer20] ZEITVERSCHIEBUNG Wird ein Signal um $\tau \in \mathbb{R}$ verschoben, also $x(t) \to x(t-\tau)$, drückt sich dies im Spektrum als Vorfaktor aus, $X(\omega) \to \mathrm{e}^{-\mathrm{i}\omega\tau}\, X(\omega)$.

FREQUENZVERSCHIEBUNG Die Amplitude eines Signals lässt sich durch Multiplikation z. B. mit einer komplexen Exponentialschwingung modulieren (Amplitudenmodulation). Die Modulation $\mathrm{e}^{\mathrm{i}\omega_0 t} x(t)$ mit imaginärer Zahl i und Grundfrequenz ω_0 resultiert in einer Frequenzverschiebung $X(\omega - \omega_0)$ des Spektrums.

FALTUNG Für zwei Signale $x(t), y(t)$ und ihre Fouriertransformierten $X(\omega)$ und $Y(\omega)$ gilt, dass die Fouriertransformierte $Z(\omega)$ der Faltung $z(t) = (x * y)(t)$ der Multiplikation der Fouriertransformierten von x, y entspricht, also $Z(\omega) = X(\omega)Y(\omega)$.

1.3 KURZZEIT-FOURIERTRANSFORMATION (STFT)

Als Möglichkeit zeitkontinuierliche Audiosignale zu analysieren, wurden die Fourierreihe und FT eingeführt. Wie gezeigt, lassen sich mithilfe der Fourierreihe jedoch nur periodische Signale als zusammengesetzte Oberschwingungen darstellen, für Audiosignale im Allgemeinen wird daher auf die FT zurückgegriffen. In der Anwendung ist die FT jedoch unpraktikabel, da die Periode

unendlich ist und daher über die gesamte Zeit integriert werden muss. Dies lässt sich auch anschaulich auf den Umstand zurückführen, dass das Spektrum $X(\omega)$ zeitlich lokale Frequenzinformationen in der Phase codiert [Kai94].

Für eine genauere Zeit-Frequenz-Analyse eines Audiosignals, hat sich die gefensterte Fouriertransformation (engl.: windowed fourier transform) (WFT) als nützlich erwiesen, mithilfe derer die Spektralanteile eines Signals bestimmten Zeitabschnitten zugeordnet werden können. Als auf zeitkontinuierliche Signale beschränkte FT wird im folgenden die Kurzzeit-Fouriertransformation (engl.: short time fourier transform) (STFT) als Spezialfall der WFT betrachtet (eine genauere Diskussion der Unterschiede in den Begrifflichkeiten finden sich im Anhang A) [Mer20].

Die zeitkontinuierliche STFT eines Signals $x \in \mathcal{L}_2(\mathbb{R})$ ist allgemein definiert als [Mer20]

$$\mathcal{F}_x^{\gamma}(\tau, \omega) = \langle x, \gamma_{\tau,\omega} \rangle = \int_{-\infty}^{\infty} x(t)\, \gamma^*(t-\tau)\, \mathrm{e}^{-\mathrm{i}\omega t}\, \mathrm{d}t\,, \tag{1.13}$$

mit der um τ verschobenen Fensterfunktion $\gamma^*(t-\tau) \in \mathcal{L}_2(\mathbb{R})$ und der Kreisfrequenz ω. Um eine möglichst gute Näherung der in einem Zeitbereich vorkommenden Frequenzen zu erhalten, wird das Signal mit der Fensterfunktion multipliziert, sodass Werte außerhalb eines gewissen *Fensters* auf Null gesetzt werden [Mer20]. Innerhalb des analysierten Fensters wird näherungsweise von einer Zeit-Invarianz des Signals ausgegangen, was entsprechende lokale Frequenzinformationen hervorhebt [KSW08].

1.3.1 *Zeit-Frequenz-Analyse*

Anschaulich lässt sich das Analysefenster $\gamma_{\tau,\nu}(t) = \gamma(t-\tau)\, \mathrm{e}^{\mathrm{i}\nu t}$ als Schwingung interpretieren, die mit der Kreisfrequenz ν im Bereich $\{t \,|\, |\gamma(t-\tau)| \neq 0\}$ oszilliert [Kai94]. Um mithilfe dieses Fensters eine möglichst genaue Zeit- und Frequenzauflösung dieser Schwingung zu erhalten, wird das Spektrum der STFT genauer betrachtet. Mithilfe der sogenannten Parsevalschen Identität lässt sich die STFT eines Signals mit dem zeit- und frequenzverschobenen Fenster $\gamma_{\tau,\nu}$ und dessen Transformierter $\Gamma_{\tau,\nu}$ definieren über [Mer20]

$$\mathcal{F}_x^{\gamma}(\tau, \nu) = \frac{1}{2\pi} \langle X, \Gamma_{\tau,\nu} \rangle = \frac{\mathrm{e}^{-\mathrm{i}\nu\tau}}{2\pi} \int_{-\infty}^{\infty} X(\omega)\, \Gamma^*(\omega-\nu)\, \mathrm{e}^{\mathrm{i}\omega t}\, \mathrm{d}\omega \tag{1.14}$$

mit den Fouriertransformierten X des Signals x und Γ des Fensters γ, sowie der Zeitverschiebung τ und der Frequenzverschiebung um ν. Das Signal hat dabei die Frequenz ω und der Vorfaktor entstammt der Frequenzverschiebung (siehe 1.2.3). [Kai94] stellt eine wichtige Erkenntnis hervor, die sich aus dieser Beziehung ableiten lässt: die Ähnlichkeit zu (1.13). Diese ergibt sich nach [KSW08] aus der Parsevalschen Identität, nach der $\langle x, \gamma_{\tau,\nu} \rangle = \frac{1}{2\pi} \langle X, \Gamma_{\tau,\nu} \rangle$ gilt. Das Fenster lässt sich also im Zeit und Frequenzbereich verschieben und bleibt dabei gleich groß, was bedeutet, dass die Verwendung einer Fensterfunktion nicht nur im Zeitbereich eine Bestimmung der Frequenzen an bestimmten Zeitpunkten ermöglicht, sondern auch im Frequenzbereich die Bestimmung der Zeitpunkte, an denen bestimmte Frequenzen auftauchen [KSW08].

Für eine präzise lokale Zeit-Frequenz-Analyse mit hoher Auflösung könnte nun im Prinzip das Fenster immer weiter *verkleinert* werden, um damit die Auflösung in der Zeit- und Frequenzdomäne zu erhöhen. Es lässt sich zeigen, dass die Fenstergröße nicht mit τ oder ν skaliert, sondern mit der Zeitdauer Δ_τ und Frequenzbandbreite Δ_ν, also der Form der Kurve $\gamma_{\tau,\nu}$ [KSW08]. Aus der Heisenbergschen Unschärferelation folgt $\Delta_\tau \Delta_\nu \geq \frac{1}{2}$, was es unmöglich macht, das Fenster unbegrenzt weiter zu verkleinern, ohne entweder Zeit- oder Frequenzinformation zu verlieren [Mer20]. Demnach lässt sich nicht gleichzeitig eine beliebig genaue Frequenz- *und* eine beliebige Zeitauflösung erreichen [Mer20, S. 298ff]. Die Wahl der Fensterfunktion bestimmt also Δ_τ und Δ_ν und somit das Verhältnis zwischen Zeit- und Frequenzauflösung [Kai94].

1.4 KONTINUIERLICHE WAVELET-TRANSFORMATION (CWT)

1.4.1 *Definition der CWT*

[Dau92] Die verschobene und gestreckte Variante $\psi^{a,b}$ eines sogenannten Mutterwavelets $\psi \in \mathcal{L}_2(\mathbb{R})$ ist definiert für $a \in \mathbb{R} \setminus \{0\}$ und $b \in \mathbb{R}$ als Funktion $\psi^{a,b} : \mathbb{R} \to \mathbb{C}$

$$\psi^{a,b}(t) := |a|^{-\frac{1}{2}} \, \psi\left(\frac{t-b}{a}\right). \tag{1.15}$$

Unter der Annahme, dass $\int_{\mathbb{R}} \psi(t)\,\mathrm{d}t = 0$, lässt sich die kontinuierliche Wavelettransformation eines Signals $x \in \mathcal{L}_2(\mathbb{R})$ mit der Notation von [Mer20] definieren als [Dau92]

$$\mathcal{W}_x(a,b) = \langle x, \psi^{a,b} \rangle = |a|^{-\frac{1}{2}} \int_{-\infty}^{\infty} x(t)\, \psi^*\left(\frac{t-b}{a}\right) \mathrm{d}t\,. \tag{1.16}$$

Invertierung der CWT [Mer20]

Anschaulich entspricht die CWT des Signals x den Koeffizienten einer Reihenentwicklung (siehe 1.2.1) und somit kann das Signal mithilfe der CWT analog zur FT aus einer gewichteten Superposition der Wavelets $\psi^{a,b}$ rekonstruiert werden [Dau92, S. 8]. Um diese Eigenschaft zu erfüllen, muss das Wavelet ψ ein stückweise stetiges Energiesignal sein, für dessen Fouriertransformierte $\Psi(\omega)$ gelte, dass [Mer20]

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\Psi(\omega)|^2}{|\omega|}\,\mathrm{d}\omega < \infty\,. \tag{1.17}$$

Dies gilt nur, wenn $|\Psi(0)| = \int_{\mathbb{R}} \psi(t)\,\mathrm{d}t = 0$ und $|\Psi(\omega)|$ nach außen hin schnell abfällt [Mer20]. Ein Wavelet ist also mittelwertfrei und nur in einem begrenzten Bereich ungleich Null, woraus sich der Name „kleine Welle“ für Wavelet ableiten lässt [JMP97]. Außerdem lässt sich das ursprüngliche Signal aus den Waveletkoeffizienten rekonstruieren gemäß [Mer20]

$$x(t) = C_\psi^{-1} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathcal{W}_x(a,b)\, \psi^{a,b}(t)\, \frac{1}{a^2}\, \mathrm{d}a\,\mathrm{d}b\,. \tag{1.18}$$

Statt dass das Signal x wie bei der STFT aus 1.3 mit einer verschobenen und amplitudenmodulierten Fensterfunktion $\gamma_{\tau,\omega} = \gamma(t-\tau)\mathrm{e}^{\mathrm{i}\omega t}$ skalarmultipliziert

wird, tritt nun die Analysefunktion $\psi^{a,b}$, das Wavelet, an diese Stelle. Darin liegt der Hauptunterschied der beiden Transformationen. Während bei der STFT das immer gleiche „starre" Fenster $\gamma_{\tau,\omega}$ mit τ und ω zeit- und frequenzverschoben wird, ist bei der CWT der Zeitparameter b an den entsprechenden Frequenzparameter a angepasst.

Nach [Mer20] kann die Fouriertransformierte $\Psi_{a,b}$ des in Zeit und Frequenz verschobenen Wavelets $\psi_{a,b}$ für eine alternative Definition der CWT herhalten. Die Fouriertransformierte ergibt sich direkt aus den Eigenschaften der FT bezüglich der Zeitverschiebung und Amplitudenmodulation (siehe 1.2.3) als [Mer20]

$$\Psi_{a,b}(\omega) = |a|^{\frac{1}{2}} \, \mathrm{e}^{-\mathrm{i}\omega b} \, \Psi(\omega) \,, \tag{1.19}$$

wobei Ψ die Fouriertransformierte des Mutterwavelets ψ ist. Die restlichen Parameter entsprechen denen in (1.15). Dadurch lässt sich die CWT eines Signals x wie bei der STFT in (1.14) definieren als inverse Fouriertransformation [Mer20]

$$\mathcal{W}_x(a,b) = \frac{1}{2\pi} \langle X, \Psi_{a,b} \rangle = \frac{|a|^{\frac{1}{2}}}{2\pi} \int_{-\infty}^{\infty} X(\omega) \Psi^*(a\omega) \, \mathrm{e}^{\mathrm{i}\omega b} \, \mathrm{d}\omega \tag{1.20}$$

der Fouriertransformierten X des Signals x. Diese Definition ist relevant für die numerische Implementierung der CWT, die in Abschnitt 1.4.3 diskutiert wird.

1.4.2 *Zeit-Skalen-Analyse*

Eine besondere Eigenschaft für Wavelets ist, dass neben der Zeit eben die Variable der *Skala* eingeführt wird, sodass $\psi^{a,b}$ mit der Skala a und der Zeitlokalisierung b *skaliert*. Durch eine Veränderung von b wird das Wavelet so verschoben, dass $\mathcal{W}_x(a,b)$ Auskunft über Frequenzinformationen um $t = b$ gibt. Wird die Skala a verändert, wirkt sich das in $\mathcal{W}_x(a,b)$ wie ein mathematischer *Zoom* in der Frequenzdomäne am Zeitpunkt $t = b$ aus. Es kann also gezeigt werden, dass die Zeit- und Frequenzauflösung von der Skala a abhängt. Da die zeitliche Auflösung mit abnehmendem $|a|$, also zunehmender Frequenz größer wird, eignet sich die CWT besonders, um hochfrequente Anteile des Signals in Bezug auf die Zeit zu analysieren [Dau92]. Für niedrige Frequenzen (großes $|a|$) steigt die Frequenzauflösung [KSW08]. Diese Eigenschaft macht die CWT attraktiv für die Signalanalyse, da Unstetigkeitsstellen im Signal von der restlichen Funktion isoliert werden können. Dagegen reagiert z. B. die klassische FT stark auf einzelnen Unstetigkeitsstellen eines Signals und auch die STFT kann nur in einem festen lokalen Fenster Auskunft über die Stetigkeit des Signals geben [VK07, S. 321f]. [LMR98]

Translations- & Affininvarianz [KSW08]

Als spezielle Form der STFT hat die Wavelettransformation auch die Eigenschaft der Translationsinvarianz. Deshalb gilt bei einem mit τ in der Zeit t verschobenen Signal $x_\tau(t) = x(t-\tau)$ für dessen Wavelettransformierte $\mathcal{W}_{x_\tau}(a,b) = \mathcal{W}_x(a, b-\tau)$, wobei $\mathcal{W}_x$ die Wavelettransformierte von x ist. Weiterhin ist die Transformation affin-invariant, was bedeutet, dass eine Skalierung des Signals mit s, also $x_s(t) = |s|^{-\frac{1}{2}} x(\frac{t}{s})$ zu einer Skalierung der Wavelettransformierten führt, also $\mathcal{W}_{x_s}(a,b) = \mathcal{W}_x(\frac{a}{s}, \frac{b}{s})$.

Zeit-Frequenz-Darstellung [Mer20]

Mithilfe eines Skalogramms, definiert als $|\mathcal{W}_x(a,b)|^2$, lässt sich die aus der Wavelettransformierten gewonnene Information über die Verteilung der Signalenergie visualisieren. Dies ähnelt dem Spektrogramm $|\mathcal{F}_x^\gamma(\tau,\omega)|^2$ der gefensterten Fouriertransformierten $\mathcal{F}_x^\gamma$, welches dieselbe Information für die STFT darstellt.

1.4.3 *Vorüberlegungen zur Implementierung der CWT*

Digitalen Audiosignalen liegen als abgetastete, also diskrete Signale vor, weshalb einige Vorüberlegungen für die Implementierung der CWT anzustellen sind. Die folgenden Abschnitte sind [JMP97] entnommen.

Da es sich bei der computergestützten Analyse von Signalen um Datenverarbeitung mit beschränkter Rechen- und Speicherkapazität handelt, ist in jedem Fall eine Diskretisierung bzw. Abtastung der kontinuierlichen Signale notwendig. Unendlich *feine* Signaldarstellungen sind hingegen nicht möglich. Damit erzeugt die kontinuierliche Wavelettransformation prinzipiell auch kein kontinuierliches Spektrum. Die numerische CWT ist aber dennoch von der diskrete Wavelettransformation (engl.: discrete wavelet transform) (DWT) abzugrenzen, da diese die Signalanalyse auf diskrete Zeitschritte und Skalen beschränkt, wohingegen die numerische CWT eine näherungsweise kontinuierliche Analyse durchführt. Dafür werden jedoch Methoden der diskreten Fouriertransformation verwendet. Während die CWT eine Verteilung der Signalenergie in Zeit und Skala erzeugt, handelt es sich bei der DWT um eine Menge angeordneter Filter an diskreten Zeitpunkten und Frequenzen, die diskrete Koeffizienten erzeugen [NS+97]. Das Verhältnis der DWT zur CWT ähnelt also dem oben beschriebenen Verhältnis zwischen Fourierreihe und Fouriertransformation.

Wie in Abschnitt 1.4.1 definiert, handelt es sich bei der CWT um eine Faltung des Signals mit einem τ-translatierten und skalierten Wavelet-Fenster. Um diese Transformation algorithmisch umzusetzen, kann das durch die Skala a parametrisierte Faltungsintegral berechnet werden. Diese Faltungen werden jedoch meist im Frequenzbereich durchgeführt, da sich dadurch Verfahren wie die sogenannte schnelle Fouriertransformation (engl.: fast fourier transform) (FFT) und inverse schnelle Fouriertransformation (engl.: inverse fast fourier transform) (iFFT) nutzen lassen [Mer20]. Es handelt sich um effiziente Algorithmen zur Berechnung der Fouriertransformierten einer Funktion mithilfe der diskreten Fouriertransformation. Diese Methode zur Berechnung der CWT ist aufgrund von (1.20) möglich und hilfreich, da sich, wie in 1.2.3 definiert, die Faltung im Frequenzbereich zu einer Multiplikation der Fouriertransformierten reduziert.

Es existieren bereits Implementierungen der CWT, die verschiedene Verfahren verwenden. Die in dieser Arbeit verwendete Methode basiert auf (1.20). Es werden also für eine diskrete Menge an Skalen a mittels der FFT die Fouriertransformierten des Signals und des Mutterwavelets berechnet und miteinander multipliziert, was einer Fensterung des Signals mit dem Wavelet entspricht. Danach wird die inverse Fouriertransformation auf das Produkt angewendet und das Ergebnis mit $\sqrt{|a|}$ skaliert. Unter Verwendung von bereits existierenden Algorithmen zur Berechnung der FFT und der iFFT und unter Einbindung

von Routinen des Moduls PyWavelets von [Lee+19] ist dieser Algorithmus in Python implementiert worden. Aus dem skalierten Ergebnis der iFFT lassen sich die Koeffizienten direkt ablesen.

1.4.4 *Wavelets für die Audiosignalanalyse*

Um Wavelets als Basisfunktionen für die Analyse und Synthese von Audiosignalen verwenden zu können, werden einige Anforderungen an sie gestellt. Die wichtigste, bereits genannte Eigenschaft ist die Mittelwertfreiheit, die durch die Zulässigkeitsbedingung (1.17) garantiert wird. Daraus folgt außerdem, dass die Signalenergie normiert ist, also $\int_{-\infty}^{\infty} |\psi|^2 \, \mathrm{d}t = 1$ [KSW08]. Weiterhin sorgt C_ψ in (1.17) dafür, dass eine Rücktransformation möglich ist, wodurch sichergestellt werden kann, dass durch die Anwendung der CWT auf ein Signal keine Informationen verloren gehen [Sad96]. Unter Einhaltung dieser Anforderungen lassen sich beliebige Wavelets konstruieren. Da aber bereits eine große Menge an verschiedenen Wavelets existieren, die sich im Bereich der Signalverarbeitung als nützlich erwiesen haben, wird auf diese zurückgegriffen. Im Folgenden sollen häufig verwendete Wavelets vorgestellt werden.

Das Morlet-Wavelet ist definiert als [Dau92]

$$\psi(t) = \pi^{-\frac{1}{4}} \, \mathrm{e}^{-\frac{t^2}{2}} \left(\mathrm{e}^{-\mathrm{i}\omega t} - \mathrm{e}^{-\frac{\omega^2}{2}} \right), \tag{1.21}$$

wobei ω die Kreisfrequenz und t die Zeit ist. Meist wird das analytisch kompaktere *Morlet-Pseudowavelet* angeführt, welches im Folgenden synonym verwendet wird [Sad96]. Es ist definiert als [NS+97]

$$\psi_{\mathrm{morl}}(t) = \frac{1}{\sqrt{\pi f_b}} \, \mathrm{e}^{-\frac{t^2}{f_b}} \, \mathrm{e}^{\mathrm{i}\omega t} = \frac{1}{\sqrt{\pi f_b}} \, \mathrm{e}^{-\frac{t^2}{f_b}} \, \mathrm{e}^{2\pi \mathrm{i} f_c t}, \tag{1.22}$$

wobei ω die Kreisfrequenz und $f_b \in \mathbb{R}^+$ die Bandbreite des Wavelets beschreibt [JMP97] sowie $f_c = \frac{\omega}{2\pi}$ die Mitten- oder Zentrumsfrequenz des Wavelets. Die Wahl von f_b beeinflusst bei diesem Wavelet die Frequenzauflösung. Für höhere Frequenzen kann durch ein größeres f_b eine bessere Frequenzauflösung erreicht werden und für niedrigere Frequenzen bieten sich kleinere Werte für f_b an [NS+97]. Für die Sprach- und Musikanalyse wird $\omega \in [5, 6] \frac{1}{\mathrm{s}}$ empfohlen [Sad96][JMP97], was $f_c \in [0.8, 0.95]$ Hz entspricht.

Ein weiteres Wavelet aus $\mathcal{L}_2(\mathbb{R})$, welches in der Fachliteratur und auch in Anwendungen zu finden ist, ist das *Mexican-Hat*-Wavelet. Es ist definiert als [LMR98]

$$\psi_{\mathrm{mexh}}(t) = \frac{2}{\sqrt{3} \pi^{\frac{1}{4}}} (1 - t^2) \, \mathrm{e}^{-\frac{t^2}{2}} \tag{1.23}$$

und hat im Vergleich zu ψ_{morl} eine kleinere Bandbreite.

Da es sich bei ψ_{morl} um eine modulierte Gauß-Funktion und bei ψ_{mexh} um die zweite Ableitung einer Gauß-Funktion handelt, lassen sich beide vergleichen. Das restliche Unterkapitel orientiert sich an der Argumentation und den Ergebnissen von [QS15]. Im hochfrequenten Bereich beispielsweise hat ψ_{mexh}

eine bessere Frequenz-Auflösung Δ_ω als ψ_{morl}. Unter weiterer Betrachtung der Zeitauflösung Δ_τ ist jedoch die Fläche $\Delta_\omega \Delta_\tau$ des Morlet-Wavelets über alle Skalen kleiner als die des Mexican-Hat-Wavelets, was in Anbetracht von Heisenbergs Unschärferelation ein Vorteil ist. Dadurch kann mit dem Morlet-Wavelet im Mittel eine bessere Zeit-Frequenzauflösung erreicht werden. Im Vergleich zu ψ_{morl} und ψ_{mexh} schneidet beispielsweise das orthonormale *Meyer*-Wavelet in der CWT schlechter ab und wird daher nicht näher betrachtet. Sowohl die Frequenz- und Zeitauflösung als auch dessen Fensterauflösung $\Delta_\omega \Delta_\tau$ ist geringer. Ein weiteres Wavelet, das dem Morlet-Wavelet ähnelt, ist das Shannon-Wavelet, welches definiert wird als [TB98, S. 63]

$$\psi_{\text{shan}}(t) = f_b^{-\frac{1}{2}} \frac{\sin(f_b \pi t)}{\pi t} \, \mathrm{e}^{\mathrm{i}2\pi f_c t} \tag{1.24}$$

mit der Bandbreite f_b, der Zentrumsfrequenz f_c und der Zeit t. Durch die enthaltene Dirichlet-Funktion flacht das Wavelet wesentlich langsamer in der Zeit ab als das Morlet-Wavelet [TB98].

1.5 AUDIO-DEEPFAKES UND MASCHINELLES LERNEN

Nachdem die STFT und CWT ausführlich eingeführt und verschiedene Wavelets vorgestellt wurden, werden im folgenden Teil die Grundlagenliteratur zu künstlichen neuronalen Netzen dargestellt und die für den experimentellen Teil wichtigen Grundlagen geschaffen. Eine Architektur für das Erzeugen von Deepfakes im Audiobereich ist z. B. das WaveNet, welches nach entsprechendem Training realistische Sprachsequenzen von menschlichen Stimmen generieren kann [Zha22]. Nachfolgend werden zunächst künstliche neuronale Netze formal eingeführt und Methoden des Trainings und der Validierung solcher Netze vorgestellt. Des Weiteren wird auf Faltungsnetze im Speziellen eingegangen, die später als Architektur in der Klassifikation zum Einsatz kommen. Zuletzt wird der aktuelle Forschungsstand zur Deepfakeerkennung in Audiodaten behandelt.

1.5.1 *Künstliche Neuronale Netze*

Deep Learning ist ein Teilgebiet des Machine Learning und befasst sich mit speziellen künstlichen neuronalen Netzen, die mindestens eine Zwischenschicht besitzen. Ursprünglich von der Neurowissenschaft und Biologie inspiriert, sind künstliche neuronale Netze in der heutigen Forschung vielfach angewendete Datenverarbeitungsmethoden [GBC16]. Die folgenden Abschnitte orientieren sich maßgeblich an dem Buch von Daniel Kriesel [Kri07].

Künstliche neuronale Netze bestehen anschaulich aus einzelnen technischen Neuronen j, die ein oder mehrere Eingangssignale x entgegennehmen, diese über eine Propagierungsfunktion f_{prop} aufsummieren und in eine Aktivierungsfunktion f_{act} geben. Die Neuronen sind in l Schichten angeordnet und jeweils mit Neuronen mehrerer Schichten verbunden. Das Neuron wird je nach Eingangssignal entsprechend dieser Funktionen *aktiviert*, was bei dem Neuron j

der l-ten Schicht als Aktivierung a_j^l bezeichnet wird. Dabei wird die Ausgabe o_j^l eines Neurons meist der Aktivierung a_j^l gleichgesetzt. Ein klassisches *Feedforward-Netz* besitzt eine Eingabeschicht für die Eingangssignale, damit verbunden jeweils die nächsttieferen Zwischenschichten bis zur Ausgabeschicht. Die *Tiefe* dieser neuronalen Netze ergibt sich aus der Menge der Schichten. *Feedforward* bedeutet, dass die Verbindungen nur in eine Richtung gehen. Außerdem sind die Verbindungen zwischen zwei Neuronen i, j verschiedener Schichten mit $w_{i,j}^l$ gewichtet und mit einem *Bias* b_j^l ausgestattet, sodass das Netz lernen kann. Es wird von einem *dicht besetzten* oder *vollvernetzten* Netz gesprochen, wenn alle Neuronen jeder Schicht mit jeweils allen Neuronen der angrenzenden Schichten verbunden sind. Es existieren noch weitere Netzarchitekturen, wovon für diese Arbeit neben dem einfachen Feedforward-Netz noch das Faltungsnetz (siehe Abschnitt 1.5.3) relevant ist.

Mathematisch lässt sich ein künstliches neuronales Netz als Funktionsapproximator beschreiben, der aus der Eingabefunktion jede Funktion y für die gegebenen Maßstäbe beliebig genau annähern kann. Diese Universalitätseigenschaft spiegelt sich in der Grundfunktion eines neuronalen Netzes wieder: der Fähigkeit zu lernen und dadurch zu *generalisieren*. Bei dem Trainingsprozess werden dem neuronalen Netz eine Menge an Trainingsbeispielen als Eingangssignal gegeben, welche durch das Netz propagieren und zu einer Ausgabe der Neuronen in der Ausgabeschicht führen. Mithilfe eines Lernalgorithmus, der Backpropagation, werden die Gewichte und der Bias so angepasst, dass eine festgelegte *Kostenfunktion* minimiert wird. Die Konstruktion dieser Kostenfunktion erfolgt auf Basis der gewünschten Ausgabe $y(x)$, die das neuronale Netz haben soll [Nie15]. Mit dem Ziel, den Funktionswert der Kostenfunktion möglichst zu minimieren, wird das Training nach einigen (im Weiteren erläuterten) Kriterien abgeschlossen [Nie15]. Mögliche Kostenfunktionen werden im nächsten Unterkapitel behandelt.

Das in dieser Arbeit angewandte Lernparadigma wird *überwachtes Lernen* genannt, da die Trainingsdaten als Audio-Deepfakes bzw. als reale Audiodaten gekennzeichnet sind. Dadurch kann die vom Netz erzeugte Ausgabe in jedem Trainingsdurchlauf mit der Kennzeichnung verglichen werden und entsprechend können dann die Gewichte und der Bias trainiert werden. Dieses Lernparadigma kommt in sogenannten Klassifikatoren zum Einsatz, die binär zwischen einem Audio-Deepfake und einem originalen Audiosignal unterscheiden, diese also in die *Klassen* „Fake“ und „Nicht-Fake“ einteilen können. Dies wird im folgenden als Klassifizierung bezeichnet. Wird dem neuronalen Netz nun ein unbekanntes Beispiel derselben Klasse eingegeben, ist es bis zu einem gewissen Maß in der Lage, dieses Beispiel richtig zu klassifizieren. Das bedeutet, dass das eingegebene Signal zu einer ähnlichen Ausgabe der Ausgabeneuronen führt, wie die Trainingsbeispiele derselben Klasse [Kri07].

1.5.2 *Backpropagation, Training und Validierung*

Die Lern- und Generalisierungsfähigkeit von neuronalen Netzen wird nicht nur im Deep Learning zur Generierung von Deepfakes verwendet, sondern soll auch in dieser Arbeit genauer analysiert, beschrieben und angewandt werden,

um Audio-Deepfakes von realen Audiodaten zu separieren. Dazu werden im Folgenden einige technische Grundlagen des Lernprozesses sowie Kriterien, nach denen die Netze trainiert werden, aufgeführt. Der folgende Abschnitt ist [Nie15] entnommen.

Wie in 1.5.1 eingeführt, wird beim überwachten Lernen eines neuronalen Netzes eine Kostenfunktion C minimiert. Diese beschreibt im Allgemeinen, wie stark die Ausgabe des Netzes von der gewünschten Ausgabe y abweicht und gibt so der Anpassung der Gewichte, dem Lernprozess, eine Richtung. Darüber hinaus lässt sich ein Zusammenhang zwischen der Anpassung der Gewichte und der Ausgabe und somit der Kostenfunktion herstellen, denn eine kleine Änderung Δw der Gewichte führt zu einer Änderung der Kostenfunktion in entsprechender Höhe. Daraus leitet sich der Lernalgorithmus der Backpropagation ab. Genauer handelt es sich um ein Gradientenabstiegsverfahren, bei dem der Gradient der Kostenfunktion nach den Gewichten und dem Bias verwendet wird, um die Änderung der Gewichte und des Bias zu finden, die zu einer Minimierung der Kostenfunktion führen. Dies soll nun formalisiert werden.

Bei dem Zusammenhang zwischen Neuronen verschiedener Schichten miteinander handelt es sich um sogenannte *affine Transformationen*, die als Matrixmultiplikation geschrieben werden kann [DV16]. Im folgenden werden Matrizen in großen Buchstaben und fett und auch Vektoren fett gedruckt. Der Operator $\odot$ steht für das Hadamard-Produkt (1.5) zweier Matrizen. Seien n_x die Anzahl der Neuronen der Schicht l und n_y die Anzahl der Neuronen der vorhergehenden Schicht $l-1$. Dann beschreibt ein Eintrag $w^l_{j,i}$ der Gewichtsmatrix $\mathbf{W}^l \in \mathbb{R}^{n_x \times n_y}$ der Schicht l die Gewichtung der Verbindung des i-ten Neurons der Schicht $l-1$ mit dem j-ten Neuron der l-ten Schicht. Formal lässt sich aus der mit dieser Matrix gewichteten Summe $\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$ der Schicht l die Aktivierung $\mathbf{a}^l \in \mathbb{R}^{n_x}$ über

$$f_{\text{act}} : \mathbf{z}^l \mapsto \mathbf{a}^l \quad \text{mit} \quad a^l_j = f_{\text{act}}\left(\sum_i w^l_{j,i}\, a^{l-1}_i + b^l_j \right) \tag{1.25}$$

definieren, wobei $\mathbf{b}^l \in \mathbb{R}^{n_x}$ den Bias beschreibt und f_{act} eine skalare Aktivierungsfunktion bezeichnet. Gängige Aktivierungsfunktionen für eine Eingabe x sind die logistische Funktion $\sigma(x) = (1 + \mathrm{e}^{-x})^{-1}$, auch Sigmoid-Funktion genannt, und der damit eng verwandte Tangens Hyperbolicus $\tanh(x) = 2\sigma(2x) - 1$, sowie die sogenannte rektifizierende Aktivierungsfunktion (engl.: rectified linear unit) (ReLU), die $\max(0, x)$ verwendet [GBC16]. Da beim Lernprozess die Ableitung der Aktivierungsfunktion gebildet wird, kann für die Zwischenschichten eine Funktion von Vorteil sein, deren Ableitung konstant hoch ist, sobald ein Neuron aktiviert wurde, damit das Lernverfahren nicht aufgrund eines zu geringen Gradienten stagniert [GBC16]. Dies ist für ReLU der Fall, denn deren Ableitung ist 1 für positive x und 0 für negative x. In dieser Arbeit beschränkt sich f_{prop} auf die gewichtete Summe aller Eingabesignale a^{l-1} der vorherigen Schicht. Dieses Prinzip lässt sich auch auf andere Netzarchitekturen übertragen [DV16].

Wie eben erwähnt lässt sich die Anpassung der Gewichte vom Gradienten der Kostenfunktion C her ableiten: An der Ausgabeschicht kann zunächst der Fehler in der letzten Schicht L mit dem Gradienten ∇ als $\boldsymbol{\delta}^L = \nabla_{\mathbf{a}^L} C \odot f'_{\text{act}}(\mathbf{z}^L)$

berechnet werden, wobei $f'(z) = \frac{\mathrm{d}f}{\mathrm{d}z}$ die skalare Ableitung der Aktivierungsfunktion beschreibt. Daraufhin wird der berechnete Fehler in die vorherige Schicht zurückpropagiert, indem statt der ursprünglichen Gewichtsmatrix ihre Transponierte $(\mathbf{W}^{L-1})^\top$ mit dem Fehler der Ausgabeschicht multipliziert wird. Damit ergibt sich für die vorletzte Schicht die Gleichung [Nie15] $\delta^{L-1} = [(\mathbf{W}^L)^\top \delta^L] \odot f'_{\text{act}}(\mathbf{z}^{L-1})$ für den von der Kostenfunktion abhängigen Fehler. Dieses Prinzip lässt sich bis zur ersten Schicht iterativ durchführen, weshalb der Algorithmus auch Backpropagation (*wörtl. übers.: Zurückpropagierung*) genannt wird. Allgemein gilt daher [Nie15]

$$\delta^l = [(\mathbf{W}^{l+1})^\top \delta^{l+1}] \odot f'_{\text{act}}(\mathbf{z}^l) \tag{1.26}$$

für jede Schicht l und ihre nachfolgende Schicht $l+1$. Aus diesen Gleichungen folgt schlussendlich die Ableitung der Kostenfunktion nach den Gewichten bzw. nach dem Bias als [Nie15]

$$\frac{\partial C}{\partial w^l_{j,i}} = a^{l-1}_i \delta^l_j \quad , \quad \frac{\partial C}{\partial \mathbf{b}^l} = \delta^l \,. \tag{1.27}$$

Der beschriebene Algorithmus ist nochmal in 1 zu sehen. Dabei ist η die skalare *Lernrate*, welche die Geschwindigkeit, mit der die Gewichte angepasst werden sollen, bestimmt.

Algorithmus 1 Backpropagation-Lernalgorithmus [Nie15]

Require: Eingabe $\mathbf{x}$ des Trainingsbeispiels
  Setze Aktivierung der Eingabeschicht $\mathbf{a}^1 = \mathbf{x}$
  for $l = 2, \ldots, L$ **do**
    $\mathbf{a}^l = f_{\text{act}}(\mathbf{z}^l)$
  end for
  Berechne Abweichung (Fehler) δ^L der Ausgabeschicht L des Netzes als $\delta^L = \nabla_{\mathbf{a}^L} C \odot f'_{\text{act}}(\mathbf{z}^L)$
  for $l = L-1, \ldots, 2$ **do**
    $\delta^l = [(\mathbf{W}^{l+1})^\top \delta^{l+1}] \odot f'_{\text{act}}(\mathbf{z}^l)$
  end for
  $w^l_{j,i} \leftarrow w^l_{j,i} - \eta a^{l-1}_i \delta^l_j$
  $b^l_j \leftarrow b^l_j - \eta \delta^l_j$

Eine gängige Kostenfunktion C ist die mittlere quadratische Abweichung, die definiert ist als [Wol21]

$$C(\mathbf{y}, \mathbf{a}^L) = \frac{1}{n} \sum_j^n (a^L_j - y_j)^2 \tag{1.28}$$

für die gewünschte Ausgabe $\mathbf{y}$ (häufig *target* genannt), der Aktivierung bzw. Ausgabe $\mathbf{a}^L$ der Ausgabeschicht L und der Anzahl n der Ausgangskanäle (z.B. 2 für ein binäres Klassifikationsproblem). Die Funktion erzeugt eine feste Ausgabe für die aktuelle Netzkonfiguration. In Bezug auf die Änderung der Aktivierung $\mathbf{a}^L$ ergibt sich der Gradient der Kostenfunktion dann als [Nie15][Wol21]

$$\frac{\partial C(\mathbf{y}, \mathbf{a}^L)}{\partial a^L_j} = a^L_j - y_j. \tag{1.29}$$

[Nie15] Die in dieser Arbeit verwendete Kostenfunktion ist die Kreuzentropie und ist definiert als [Wol21]

$$C(\mathbf{y}, \mathbf{a}^L) = -\sum_j^n (y_j \ln a_j^L) + (1 - y_j) \ln(1 - a_j^L) \tag{1.30}$$

und wird bei Klassifikationsproblemen häufig verwendet. Wird die Sigmoid-Funktion $f_{\text{act}}(x) = (1 + e^{-x})^{-1}$ als Aktivierungsfunktion gewählt, kann es zu nur langsamen Lernschritten kommen, da die Veränderung der Gewichte direkt von der Ableitung der Aktivierungsfunktion abhängt und diese an den Rändern der Sigmoid-Funktion sehr klein wird. Bei der Kreuzentropie hängt der Gradient der Kostenfunktion dagegen nur noch von der Ausgabe des Netzes ab, was die Lernschritte gleichmäßiger macht.

Statt des klassischen hat sich das stochastische Gradientenabstiegsverfahren bewährt [Nie15, S. 22f]. Dabei wird ein neuronales Netz statt auf den gesamten Trainingsbeispielen nur auf wechselnd zufällig ausgewählten kleinen Mengen (engl. *Batches*) an Beispielen trainiert. Dies wird in mehreren *Epochen* durchgeführt, wobei eine Epoche ein Durchlauf aller Trainingsbeispiele beschreibt. Da in dieser Arbeit das in Abschnitt 2.1 beschriebene Framework PyTorch [Pas+19] verwendet wird, welches den Backpropagation-Algorithmus implementiert, wird im Folgenden nicht weiter auf technische Details eingegangen. In den nächsten Absätzen werden abschließend das Vorgehen, verwendete Metriken zur Evaluierung und die Optimierung des Trainings neuronaler Netze eingeführt. Sofern nicht anders angegeben, ist der Inhalt des restlichen Kapitels [Nie15] entnommen.

Das Training neuronaler Netze zielt darauf ab, möglichst effizient und schnell eine passende Kostenfunktion zu minimieren, sodass das Netz gut generalisiert. Jeder Trainingsepoche wird dafür ein Testzyklus angeschlossen, bei der ein Teil der Gesamtdaten, die Testmenge, in das Netz gegeben und die Ausgabe ermittelt wird. Dabei lässt sich die *Genauigkeit* des Netzes als Anteil richtig klassifizierter Beispiele der Gesamtmenge beschreiben. Eine weitere Metrik zur Evaluation eines binären Netzes ist die Equal Error Rate (dt.: Gleichfehlerrate) (EER) [FS21].

EER [Pow20], [Yam+19] Die Testmenge wird dafür in zwei Klassen *positiv* (P) und *negativ* (N) geteilt, was bei binären Aufgaben trivial ist. Das Ergebnis der Klassifikation eines Netzes für ein Testdatum wird *falsch positiv* (FP) genannt, wenn es fälschlicherweise aussagt, dass es sich um die bestimmte Klasse handelt. Das Ergebnis wird wiederum als *falsch negativ* (FN) bezeichnet, wenn es fälschlicherweise aussagt, dass es sich bei dem Testdatum nicht um eine bestimmte Klasse handelt. Daraus lässt sich die Falschakzeptanzrate (FAR) über $\text{FAR} = \frac{\text{FP}}{\text{N}}$ und die Falschrückweisungsrate (FRR) über $\text{FRR} = \frac{\text{FN}}{\text{P}}$ berechnen. Da die Ausgabe des Netzes in Form von Wahrscheinlichkeiten zwischen 0 und 1 für die jeweiligen Klassen zurückgegeben wird, kann die FRR und FAR für verschiedene Schwellenwerte berechnet werden. Der Schwellenwert bestimmt, ab welcher Wahrscheinlichkeit ein Testergebnis der P-Klasse oder der N-Klasse zugeordnet wird. Die EER ist schließlich definiert als der Punkt, an dem für einen bestimmten Schwellenwert die FRR und FAR übereinstimmen [FS21]. Der beste Wert für die EER ist hierbei 0 und der schlechteste Wert 1.

Wird das Training unbegrenzt fortgeführt, steigt die Genauigkeit in Bezug auf die Testmenge, da die gewünschte Ausgabefunktion immer genauer approximiert wird. Dies führt zu sogenanntem Overfitting (dt.: Überanpassung). Es kann beobachtet werden, dass die Generalisierungsfähigkeit von überangepassten Netzen sinkt. Dieses Phänomen kann mithilfe verschiedener Methoden erkannt und verringert werden, damit das Netz die Trainingsdaten sozusagen nicht nur stur auswendig lernt und dann keine Abweichungen bei der Klassifizierung mehr zulässt. Eine dieser Methoden ist die Erzeugung einer von der Trainingsmenge disjunkten Validierungsmenge, auf der die Genauigkeit des Netzes ebenfalls berechnet wird. Dadurch kann Overfitting mithilfe von *Early Stopping* minimiert werden, wobei das Training gestoppt wird, sobald die Genauigkeit der Validierungsmenge nicht mehr weiter steigt.

Weiterhin führt das Vergrößern des Trainingsdatensatzes oftmals zu geringerer Überanpassung. Eine andere Möglichkeit, Overfitting zu verringern und die Genauigkeit des Netzes zu erhöhen ist *Regularisierung*. Sowohl technisch als auch anschaulich zielt Regularisierung darauf ab, neuronale Netze während des Trainings robuster gegen Rauschen zu machen und dabei das Lernen einer einfachen und musterorientierten Modellierung der Trainingsdaten zu forcieren. Methoden wie *Weight Decay* und L_1-Regularisierung sowie *Dropout* [Sri+14] haben sich bewährt. Für Weight Decay und L_1-Regularisierung wird der Kostenfunktion ein Regularisierungsterm angefügt, damit das Netz kleine Gewichte vorzieht und die Gewichte nur auf den für das Muster relevanten Teil der Neuronen verteilt. Dropout bedeutet, dass beim Training zufällig wechselnd einzelne Neuronen vollständig vom Training ausgeschlossen werden, damit sich die Gewichte eines Netzes nicht zu sehr auf einzelne Neuronen konzentrieren.

Neben Early Stopping und Regularisierung ist die Wahl der Kostenfunktion sowie der Aktivierungsfunktion von Bedeutung für die Genauigkeit eines neuronalen Netzes. Für genauere Informationen zu Vor- und Nachteilen der verschiedenen Funktionen wird auf [Nie15] und [Kri07] verwiesen. Auf die für diese Arbeit gewählten Parameter und Aktivierungs- sowie Kostenfunktionen wird in Kapitel 2 genauer eingegangen.

1.5.3 *Faltungsnetze*

Neben dicht besetzten neuronalen Netzen wird in dieser Arbeit eine leicht abgewandelte Art verwendet: das Faltungsnetz (engl.: convolutional neural network) (CNN). Nach der Einführung in 1998 von LeCun [GBC16, S. 330] hat sich im Jahre 2012 mit der erfolgreichen Anwendung der CNNs in der ImageNet-Challenge ein weiterer Forschungsbereich des Deep Learning aufgetan [DV16]. CNNs haben sich als performantere und genauere Modelle herausgestellt, die eine hierarchische Repräsentation der Trainingsdaten lernen und im Vergleich zu dicht besetzten Feedforward-Netzen eine ausgeprägtere Generalisierungsfähigkeit besitzen [GBC16]. Da CNNs sowohl im Bereich der Computer Vision [Mas+18], Speech Recognition und besonders bei der Erkennung und Generierung von Deepfakes vielfach eingesetzt werden [Zha22; Kho+21; Wol+22], soll

der Fokus dieser Arbeit auf den Faltungsnetzen liegen. Der folgende Abschnitt ist [DV16] entnommen.

Wie bei dicht besetzten Netzen handelt es sich bei Faltungsnetzen um affine Transformationen der Eingabedaten. Diese können als mehrdimensionale Matrix vorliegen und werden im Weiteren als *Tensor* bezeichnet. Statt der einfachen Matrixmultiplikation (siehe (1.25)) wird die diskrete Faltung mit einem *Kernel* bzw. *Filter* verwendet. Als lineare Transformation hat die diskrete Faltung den Vorteil, die verschiedenen Eigenschaften der Eingabedaten zu bewahren, während die einfache Matrixmultiplikation alle Dimensionen gleich behandelt. Das bedeutet, dass beispielsweise Farbkanäle von Bildern oder Stereokanäle von Audiodaten sowie die Bild- und Audiodimensionen durch die Faltung im Lernprozess nicht verloren gehen. Neben der erwähnten Performanz und Genauigkeit benötigen CNNs aufgrund dieser Faltungsoperation reduzierte Speicherkapazität, was daran liegt, dass die diskrete Faltung statt dicht besetzten dünn besetzte Matrizen verwendet (engl. Sparse Weights) und durch sogenanntes Parameter Sharing dieselben Parameter für mehrere Faltungen verwendet [GBC16, S. 335].

Damit dieses Modell lernen kann, müssen trainierbare Gewichte vorhanden sein. In Analogie zu (1.25) sind die Gewichte im dünn besetzten Kernel enthalten. Mithilfe von Backpropagation (1) kann das CNN Merkmalsrepräsentationen lernen, die an verschiedenen Stellen der Eingabedaten vorkommen [Nie15]. In vorderen Schichten lernt es einfache Merkmale wie Ecken und Kanten, in tieferen Schichten dann bereits komplexere, zusammengesetzte Merkmale [LeC+98], was oben bereits als Lernen hierarchischer Repräsentationen bezeichnet wurde. Handelt es sich bei den Eingabedaten beispielsweise um Tensoren fourier- oder wavelettransformierter Signale, können Faltungsnetze komplex aufgebaute Merkmale im Zeit-Frequenz- oder Zeit-Skalenbereich der Daten lernen. Im Folgenden wird der Aufbau und die Funktionsweise eines Faltungsnetzes erläutert.

Ein Faltungsnetz besteht aus mehreren Faltungsschichten. Um eine Klassifizierung von extrahierten Merkmalen möglich zu machen, werden den Faltungsschichten weitere vollvernetzte Neuronenschichten angehängt. Eine klassische Faltungsschicht wiederum gliedert sich in drei Teile: Faltung, Aktivierung und Pooling [Nie15]. Die Eingabe des Tensors x in ein CNN wird als Input-Feature-Map (engl. für Merkmalskarte) bezeichnet und wird mittels der diskreten Faltung in eine Output-Feature-Map transformiert [DV16].

Da es sich bei fourier- und wavelettransformierten Audiosignalen um zweidimensionale Tensoren handelt [GBC16, S. 361], können diese als Bild aufgefasst werden. Die transformierten Signale in dieser Arbeit dienen wiederum als Eingabe für die Faltungsnetze, weshalb besonders die diskrete Faltung von Bilddaten von Interesse ist. Diese lässt sich nach [Wol21, S. 11] für das Eingabesignal $\mathbf{X} \in \mathbb{R}^{n_r \times n_c}$ mit n_r Zeilen und n_c Spalten und Einträgen $x_{i,j}$ und das Ausgabesignal $\mathbf{Y}$ mit Einträgen $y_{i,j}$ an jedem Pixel mit Zeilen-Spalten-Koordinaten (i, j) und einem zweidimensionalen Kernel $\mathbf{K} \in \mathbb{R}^{k_r \times k_c}$ mit Einträgen $k_{i,j}$ definieren als [Wol21]

$$y_{i,j} = (x * k)_{i,j} = \sum_{m}^{k_r} \sum_{n}^{k_c} x_{i+m,j+n}\, k_{m,n} \tag{1.31}$$

mit Anzahl der Zeilen k_r und Anzahl der Spalten k_c des Kernels. Genau genommen handelt es sich hier um eine Korrelation, eine Faltung mit einem gespiegelten Kernel. Die klassische Faltung wird jedoch oft als Korrelation implementiert [GBC16, S. 333] und wird hier daher synonym verwendet. Falls der Kernel die Ränder des Eingabesignals überlappt, ist ein sogenanntes *Padding* (dt.: Auffüllung) notwendig, bei dem z. B. Nullwerte hinzugefügt werden, sodass $i + m \leq n_r$ bzw. $j + n \leq n_c$.

Anschaulich wird das als Raster angeordnete Eingangssignal mit dem Kernel *abgetastet* und so gefiltert. Dies ist in Abbildung 1.1 beispielhaft mit einer (4×4)-Input-Feature-Map und einem (3×3)-Kernel zu sehen.

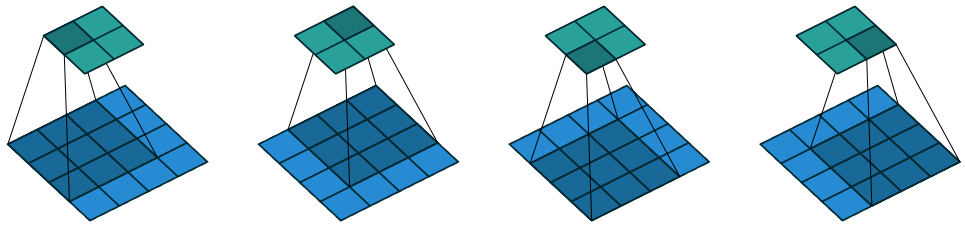


Abbildung 1.1: Faltung einer (4×4)-Input-Feature-Map (blau) mit einem (3×3)-Kernel (dunkelblau) in eine (2×2)-Output-Feature-Map (grün) [DV16, S. 14]

Um die Gradienten im Backpropagation-Algorithmus 1 einfach berechnen zu können, führt [DV16] die transponierte Faltung ein, bei der zirkuläre Matrizen der Kernelelemente $k_{i,j}$ verwendet werden. Diese Matrix ist so aufgebaut, dass eine Multiplikation der Zirkulärmatrix mit einem aus der zweidimensionalen Eingabe ausgerollten Vektor (engl. flattend) genau der Faltung derselben entspricht [DV16]. Der Eingabevektor ist also eine eindimensionale Matrix. Dies lässt sich an einem Beispiel illustrieren, welches direkt am Beispiel von [DV16, S. 20] angelehnt ist. Sei $\mathbf{K}'$ eine beispielhafte (2×2)-Kernelmatrix und $\mathbf{X}'$ ein zweidimensionales Eingabesignal der Größe (3×3). Im Folgenden stehen die Indizes i, j für die Zeile i und die Spalte j. Dann lässt sich die Faltungsmatrix in zyklischer Form schreiben als

$$\begin{pmatrix} k_{0,0} & k_{0,1} & 0 & k_{1,0} & k_{1,1} & 0 & 0 & 0 & 0 \\ 0 & k_{0,0} & k_{0,1} & 0 & k_{1,0} & k_{1,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{0,0} & k_{0,1} & 0 & k_{1,0} & k_{1,1} & 0 \\ 0 & 0 & 0 & 0 & k_{0,0} & k_{0,1} & 0 & k_{1,0} & k_{1,1} \end{pmatrix} \tag{1.32}$$

wobei $k_{i,j}$ den Einträgen von $\mathbf{K}'$ entsprechen [DV16]. Der Inputvektor wäre dann $(x_{00}\, x_{01}\, x_{02}\, x_{10}\, x_{11}\, x_{12}\, x_{20}\, x_{21}\, x_{22})^T$, mit der Bezeichnung $\mathbf{M}^T$ für die Transponierte einer Matrix $\mathbf{M}$. Die Multiplikation $\mathbf{K}'\mathbf{X}'$ erzeugt eine Output-Feature-Map, die ein vierdimensionaler Vektor ist, welcher wieder in eine (2×2)-Matrix überführt werden kann [DV16]. Diese gleicht dem Ergebnis einer klassischen Faltung der Eingabematrix. Natürlich lässt sich diese zirkuläre Struktur auf Kernel größerer Dimensionierung übertragen.

Unter Verwendung dieser Notation, stellt Dumoulin [DV16] fest, reduziert sich die Backpropagation darauf, die Transponierte von $\mathbf{K}$ mit dem Gradienten der Kostenfunktion zu multiplizieren, was den Rechenaufwand und die Komplexität verringert. In dieser Arbeit wird die implementierte Variante der Faltungsschichten von PyTorch [Pas+19] verwendet, weshalb nicht genauer auf mathematische Details eingegangen wird.

Bei dieser Herangehensweise handelt es sich um Matrixmultiplikationen, weshalb die Dimensionen dieser Matrizen von Belang sind. Im Kontext der Faltungsnetze ist daher besonders die *Stride* (dt.: Schrittweite), ein vorgegebener Abstand, mit der der Kernel über das Eingabebild iteriert, sowie das Padding, mit der die Input-Feature-Map in Höhe und Breite vergrößert werden kann, erwähnenswert [GBC16]. Die gewählten Werte für Stride und Padding beeinflussen die Output-Feature-Map und werden in Kapitel 2 aufgeführt.

Da unterschiedliche Kernels mit derselben Input-Feature-Map gefaltet werden können, entstehen entsprechend viele Output-Feature-Maps, die verschiedene Merkmale hervorheben [Wol21]. Analog zu dicht besetzten Netzen wird daraufhin eine Aktivierungsfunktion auf diese Feature-Maps angewendet, meist ReLU [Nie15]. Danach findet das Pooling statt, bei dem die Feature-Maps mit einer Pooling-Funktion gefiltert werden, die ähnlich der Faltung die Daten transformiert, um die relevanten Informationen zu extrahieren. Eine häufig verwendete Funktion ist das Max-Pooling, bei der nur der Wert der größten Ausgabe eines festen Abschnitts der Feature-Map weiterverwendet wird [Nie15]. Dies wird analog zur beispielhaften Faltung in Abbildung 1.1 in festen Abständen durchgeführt, allerdings eben mit einer Poolingfunktion statt einer gewichteten Aufsummierung eines Kernels. Dadurch werden die Feature-Maps wie bei der Faltung verkleinert, was die zu lernenden Parameter ebenfalls verringert und Rechenzeit sowie Speicherplatz einspart [GBC16].

1.5.4 *Modellinterpretation: Attribution*

[STY17] Die Wahl der sogenannten Hyperparameter eines neuronalen Netzes umfasst unter anderem die Zusammenstellung seiner einzelnen Komponenten, durch welche die Eingaben verarbeitet werden. Um das erzeugte Modell besser verstehen zu können, hat sich im Laufe der Entwicklung von tiefen neuronalen Netzen die Methode der *Integrated Gradients (dt.: Integrierte Gradienten) (IG)* hervorgetan, welche im folgenden vorgestellt werden soll. Diese gehört zur Methodik der *Attribution* (dt.: *Attribuierung, Zuordnung*): Ein tiefes neuronales Netz, welches durch eine Funktion $F : \mathbb{R}^n \to [0,1]$ beschrieben werden kann, gibt eine Vermutung $F(x)$ zur Klasse eines Inputs $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ zurück. Die Attribution zu dieser Vermutung ist ein Vektor $A(x, x') = (a_1, \ldots, a_n) \in \mathbb{R}^n$ relativ zu einem neutralen Vergleichswert $x' \in \mathbb{R}^n$, dessen Elemente a_i den Beitrag von x_i zu F gewichten [STY17, Definition 1]. Als Ansatz kann die Attribution also aufzeigen, welcher Teil der Eingabe das Netz besonders stark bei der Entscheidung zum Einteilen in eine Klasse beeinflusst. Integrierte Gradienten sind definiert als Wegintegral [STY17]

$$\mathrm{IG}_i(x) := (x_i - x_i') \cdot \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha(x - x'))}{\partial x_i} \, \mathrm{d}\alpha \tag{1.33}$$

auf dem geraden Weg zwischen dem Eingabevektor $x \in \mathbb{R}^n$ und dem Vergleichsvektor $x' \in \mathbb{R}^n$ und werden über die Gradienten $\frac{\partial F(x)}{\partial x_i}$ der i-ten Kanaldimension (z. B. Farbkanal, Audiokanal) berechnet. Für Bilder wäre ein Vergleichsvektor z. B. ein schwarzes Bild derselben Dimensionen und für Audiodaten der Nullvektor. Numerisch lassen sich die IG einfach annäherungsweise implementieren als Summe [STY17]

$$\mathrm{IG}_i^{\mathrm{aprx}}(x) := (x_i - x_i') \cdot \frac{1}{m} \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m} \cdot (x - x'))}{\partial x_i} \tag{1.34}$$

mit $m \in \mathbb{N}$ Schritten. Eine Schrittzahl von $m = 300$ genügt als Näherung an das Wegintegral [STY17]. Aus der Intensität der integrierten Gradienten an spezifischen Punkten lassen sich Rückschlüsse auf das Eingabesignal ziehen: Je größer die Intensität an einem Punkt ist, desto relevanter kann die dort vorliegende Information für die Klassifizierung durch das Netz sein.

1.5.5 *Generierung und Erkennung von Deepfakes*

Im Bereich der Deepfake-Generierung haben sich neben CNNs auch andere Netzarchitekturen als besonders performant hervorgetan, darunter rückgekoppelte neuronale Netze, Long-Short-Term-Memory-Netze und das Generative Adversarial Network (dt.: erzeugendes gegnerisches Netzwerk) (GAN) [Zha22]. Speziell im Audiobereich finden sich verschiedene generative Modelle, darunter sogenannte Flow-basierte Ansätze, probabilistische Diffusionsnetzwerke sowie auch GANs [FS21]. Da sich diese Arbeit zwar mit Datensätzen befasst, die unter Verwendung dieser Netzwerkarchitekturen erzeugt wurden, diese jedoch nicht im Einzelnen untersucht werden, soll nicht weiter auf die Details dieser Netze eingegangen werden.

Sowohl in der klassischen Signalverarbeitung als auch im Deep Learning existieren verschiedenste Ansätze zur Erkennung von Audiodeepfakes. Der Fokus soll hier auf Deep-Learning-Algorithmen liegen. Diese unterscheiden sich in den untersuchten akustischen Merkmalen, auf welche die neuronalen Netze trainiert werden, sowie in den Klassifikations-Architekturen, die in den neuronalen Netzen verwendet werden. Wie bereits in Abschnitt 1.5.3 angesprochen, haben sich Architekturen wie tiefe dicht besetzte Netze und Faltungsnetze als performante und genaue Modelle durchgesetzt. Daneben finden sich Erkennungsansätze mithilfe von Gaussian Mixture Models (GMM), die eine Wahrscheinlichkeitsverteilung aus den Trainingsdaten lernen, Support Vector Machines und rückgekoppelten neuronalen Netzwerken, sowie Kombinationen aus CNNs und rückgekoppelten Netzen und spezielle Formen von CNNs wie beispielsweise Temporal Convolution Networks. Weiterhin haben sich bestimmte Merkmale von Audiosignalen hervorgetan, die als Input für Deep-Learning-Algorithmen dienen und deren Generalisierungsfähigkeit erhöhen. Zu diesen Merkmalen gehören neben der mittleren quadratischen Signalenergie, Chroma-Features oder der Schwerpunktwellenlänge auch beispielsweise die Mel Frequency Cepstral Coefficients (MFCC), wobei es sich um spezielle periodische Komponenten der menschlichen Stimme handelt.

[Kho+21]

[FS21]

Dabei wird aus dem Eingangssignal mittels STFT ein Spektrogramm erzeugt. Dieses wird mit einem Mel-Filter skaliert, was einer modifizierten logarithmischen Skalierung der Frequenzen entlang der Achse entspricht. Das resultierende Mel-skalierte Spektrogramm wird logarithmiert und dann die diskrete Kosinustransformation (engl.: discrete cosinus transform) (DCT) darauf angewendet. Frank und Schönherr [FS21] experimentieren in ihrer Arbeit sowohl mit MFCCs als auch mit sogenannten Linear Frequency Cepstral Coefficients (dt.: Lineare-Frequenz-Cepstrum-Koeffizienten) (LFCC), die sich nach Frank et al. von MFCCs insofern unterscheiden, dass statt einem Mel-Filter ein linearer Filter verwendet wird, um höherfrequente Informationen herauszufiltern. In ihrer Arbeit nutzen [FS21] beide Merkmale, um mit einem Gaussian-Mixture-Model Audio-Deepfakes in ihrem WaveFake-Datensatz zu erkennen und erreichen mittels der LFCC-Merkmale eine hohe Genauigkeit. Darüber hinaus zeigen sie, dass ihre Klassifikatoren besonders im hochfrequenten Bereich der Audiodeepfakes operieren. Dies motiviert die in dieser Arbeit durchgeführte Audiosignalanalyse mittels der CWT, da sie im Vergleich zur STFT präziser Zeit- und Frequenzinformationen gleichzeitig liefern kann (siehe 1.4.2).

Zudem unterscheiden [Wol+22] zwischen Deepfakeerkennungs-Verfahren, die im Frequenzbereich operieren, beispielsweise mithilfe der DCT oder FT, und Verfahren, die im Pixelbereich arbeiten, bei denen z. B. Bilder direkt in CNNs eingegeben werden. [Wol+22] schlagen hingegen eine Methode vor, die mithilfe von Wavelet-Paketen beide Bereiche kombiniert, um sowohl eine hohe Frequenzauflösung als auch Zeitauflösung zu erreichen und so die Deepfakeerkennung zu erleichtern.

Darüber hinaus finden sich in der Audiosignalverarbeitung diverse Ansätze zur Erkennung von speziellen Merkmalen in Signalen mithilfe der CWT und CNNs, beispielsweise zur Erkennung von Motorgeräuschen [Wan+17], fehlerhaften elektrischen Leitungen von Spannungsverteilungssystemen [Guo+18] oder Herzgeräuschen [MLL18]. Des Weiteren nutzen [Kac+22] die CWT erfolgreich, um mit CNNs die neurologische Sprachstörung Dysarthrie aus Stimmaufnahmen zu klassifizieren. Dabei nutzen sie die hohe Frequenzauflösung der Transformation für tiefe Frequenzen, in denen die für das Krankheitsbild typischen Artefakte auftreten. Ferner untersuchen [FAK22] Audiodeepfakes direkt anhand der Mel-Spektrogramme statt auf spezifischen Merkmalen wie MFCCs und implementieren ein eigenes neuronales Netz, welches diskrete Wavelettransformationen innerhalb der Netzstruktur verwendet. Diese Ansätze sowie die Ergebnisse von [Wol+22] und [FS21] motivieren die Anwendung der Wavelettransformation auf Audiodaten, um diese zu analysieren und mittels eines CNN als Deepfake klassifizieren zu können. Dies wird in Kapitel 2 experimentell untersucht.

2 NUMERISCHE EXPERIMENTE

In Kapitel 1 wurden die Grundlagen zur Signalanalyse mit Fourier- und Wavelettransformation eingeführt und die Verwendung im Bereich der Deepfake-Erkennung motiviert. Nachfolgend wird dieser mathematische Hintergrund nun auf Methoden der Numerik übertragen, um zu untersuchen, inwiefern die kontinuierliche Wavelettransformation die Deepfakeerkennung mit Methoden des maschinellen Lernens ermöglicht. Vorbereitend werden zunächst die Python-Module und der verwendete Datensatz vorgestellt. Anhand von Spektrogrammen und Skalogrammen werden dann die STFT und CWT der Audiodaten dieses Datensatzes verglichen. Im Weiteren werden in Abschnitt 2.4.3 verschiedene CNNs in PyTorch implementiert und mithilfe der transformierten Audiosignale trainiert, die dem vorgestellten Datensatz entstammen. Durch die Verwendung verschiedener Wavelets in der CWT, sowie lernender Wavelets werden abschließend in Abschnitt 2.4.4 die trainierten Klassifikatoren hinsichtlich ihrer Robustheit gegen die verschiedenen Audioquellen des Datensatzes miteinander verglichen und dies den Ergebnissen von [FS21] gegenübergestellt.

2.1 ENTWICKLUNGSUMGEBUNG UND IMPLEMENTIERUNG

Der gesamte Programmcode, der in dieser Arbeit zur Transformation von Audiosignalen, zur Klassifikation von Audio-Deepfakes und Generierung von Diagrammen verwendet wurde, ist im GitHub-Repository [GW23] einsehbar und soll der Nachvollziehbarkeit der Ergebnisse dienen. Zur Entwicklung wurde die Umgebung *Visual Studio Code* und die Programmiersprache Python der Version 3.10 verwendet. Die wichtigsten Python-Module, die miteinbezogen wurden, sind *PyWavelets* [Lee+19], *PyTorch* [Pas+19], *TorchAudio* [Yan+22], die *PyTorch-Wavelet-Toolbox* [Wol21] und *NumPy* [Har+20].

PyWavelets implementiert diverse Wavelets, die DWT und CWT, und kommt extensiv in Kombination mit der PyTorch-Wavelet-Toolbox zum Einsatz, die verschiedene Transformationen wie die schnelle Wavelettransformation mit dünn besetzten Matrizen und die CWT umsetzt. Im Vergleich zu PyWavelets kommen hier jedoch PyTorch-Tensoren statt NumPy-Arrays zum Einsatz. Damit baut dieses Modul eine Brücke zwischen Wavelettransformationen und PyTorch, was die Verwendung der Funktionen von PyTorch ermöglicht und vereinfacht.

Als umfassendes, performantes und weit verbreitetes Framework im Bereich des maschinellen Lernens in Python wird PyTorch gewählt. Durch eine flexible Schnittstelle wird die Implementierung von individuell konfigurierbaren CNNs möglich, aber auch das Laden von Eingabedaten und die Optimierung der Modelle ist in PyTorch einfach gehalten. Insbesondere wird PyTorch verwendet, weil es die Parallelisierung von Tensor-Berechnungen und die Hardwarebeschleunigung durch Verwendung von Grafikprozessoren (engl.: graphics processing unit, GPU) unterstützt. [Pas+19]

2.2 DER WAVEFAKE-DATENSATZ

In dieser Arbeit werden ausschließlich sprachbasierte Audiosignale untersucht, die dem WaveFake-Datensatz [FS21] entspringen. Die folgenden Informationen sind aus der Beschreibung des Datensatzes [FS21, S. 4f] entnommen. Er beinhaltet 177 985 Audiodateien im 16-Bit PCM_S-Wav-Format mit einer Abtastrate von 22050 Hz. Die Audiosignale haben nur einen Kanal, was die Verwendung vereinfacht. Es handelt sich dabei um Audio-Deepfakes, die von sieben verschiedenen Netzwerkarchitekturen erzeugt wurden. Sie heißen MelGAN (Mel), MelGAN Large (L-Mel), Multi-Band-MelGAN (MB-M), Full-Band-MelGAN (FB-M), Parallel-WaveGAN (PWG), HiFi-GAN (HiFi) und WaveGlow (WG). Um die verschiedenen neuronalen Netze miteinander zu vergleichen, haben [FS21] diese Architekturen verwendet, um damit durch sogenannte Text-To-Speech-Vocoder die zwei öffentlichen Datensätze LJSpeech [IJ17] und JSUT [STS17] zu reproduzieren. Der WaveFake-Datensatz besteht damit aus 13 100 Audiodateien aus LJSpeech für jedes der sieben aufgezählten Netzarchitekturen und 5 000 Dateien aus JSUT für je Multi-Band-MelGAN und PWG, sowie theoretisch 16 285 Sequenzen, die von einem auf LJSpeech trainierten PWG aus Sätzen in Textform erzeugt wurden (die Bezeichnung *TTS* für diesen Datensatz wird hier von den Autor*innen übernommen). Der öffentlich zugängliche Teil des Datensatzes besteht jedoch nur aus 177 983 unterschiedlichen Dateien. Dies ist jedoch zu vernachlässigen, da es sich um zwei fehlende Dateien des TTS handelt und dieser Teil des Datensatzes unwesentlich für die vorliegende Arbeit ist. Bei den Audiodateien in LJSpeech handelt es sich um Sprachsequenzen mit einer durchschnittlichen Länge von 6 Sekunden, in der eine englisch sprechende Person Sätze aus nicht-fiktionalen Büchern vorliest. Weiterhin entsprechen die Audiosequenzen aus JSUT japanischen Sätzen, die von einer Person mit Japanisch als Muttersprache vorgelesen werden.

2.3 UNTERSUCHUNG DER DATENSÄTZE

2.3.1 *Spektro- und Skalogramme der Audio-Deepfakes*

Im Rahmen dieser Arbeit wird das experimentelle Vorgehen von [FS21] nachgestellt, um Unterschiede zwischen realen Audiosignalen und Audio-Deepfakes herauszuarbeiten. Wie bei [FS21] werden die Audiodateien mit Methoden aus TorchAudio aus den Wav-Dateien in PyTorch-Tensoren geladen. Die Tensoren repräsentieren damit Matrizen aus dem reellen Intervall $[-1, 1]^{1\times|S|}$, wobei S der Menge der abgetasteten Werte (engl. Samples) und jedes Sample der Amplitude des Signals an einem bestimmten Zeitpunkt entspricht. In Kombination mit der *Abtastrate* A in der Einheit Hertz, die als Häufigkeit der Abtastung der Amplitude eines Signals pro Zeiteinheit definiert ist [OW92], kann das i-te Sample $s_i \in S, i \in \{1, 2, \ldots, |S|\}$ als Amplitude am Abtastzeitpunkt iA^{-1} in Sekunden ab Zeitpunkt $t = 0$ interpretiert werden.

Wie bereits in Abschnitt 1.5.5 eingeführt, haben [FS21] im weiteren experimentellen Verlauf Spektrogramme zu den Audiosignalen aus LJSpeech und den Deepfakes aus ihrem WaveFake-Datensatz erzeugt. Diese Spektrogramme wur-

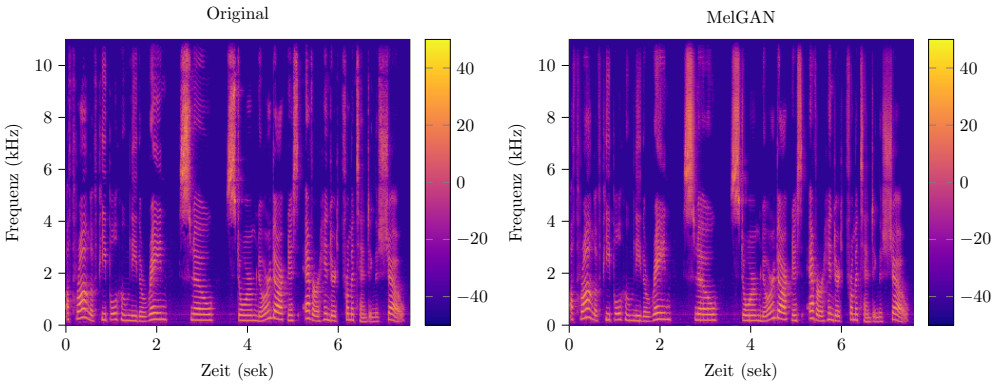


Abbildung 2.1: Spektrogramme des Satzes „This was more particularly the practice in London.“ als Originalaufnahme (links) aus LJSpeech und des Audio-Deepfakes (rechts) von [FS21] der Bezeichnung *LJSpeech 008-0217*. Es handelt sich um die STFT des rohen Audiosignals, wobei als Fensterfunktion das Von-Hann-Fenster mit einer Fensterlänge von 1023 Samples gewählt wurde. Es werden 512 Frequenzbereiche berechnet (n_FFT= 1023). Im hohen Frequenzbereich sind die Koeffizienten des Audio-Deepfakes im Vergleich zum Original verwaschen. Die Koeffizienten in den tieferen Frequenzen zeigen hingegen beim Deepfake horizontale Artefakte (z. B. bei Sekunde 3).

den hier in Anlehnung an diese Publikation generiert. Da nicht deutlich wurde, welche Methoden und Parameter dort für die Erzeugung der Spektrogramme verwendet wurden und wie skaliert wurde, wird in dieser Arbeit auf die Standard-Implementierung der STFT in PyTorch zurückgegriffen, welche über TorchAudio direkt einen Tensor in Zeit-Frequenz-Darstellung zurückgibt und das Ergebnis der STFT des Audiosignals repräsentiert. In Abbildung 2.1 sind zwei solche Spektrogramme zu sehen. Bei dem linken Spektrogramm handelt es sich um ein Signal aus LJSpeech, rechts daneben ist das vom MelGAN erzeugte Pendant aus dem WaveFake-Datensatz zu sehen. Weitere Spektrogramme zu den Audio-Deepfakes der anderen Netzwerkarchitekturen sind im Anhang A zu finden. Wie [FS21] beobachten, sind deutliche Unterschiede zwischen dem Original und dem Audiodeepfake zu erkennen. Besonders im hochfrequenten Bereich wirken die Fourierkoeffizienten des Deepfakes verwaschen, da Frequenzinformationen zu fehlen scheinen (siehe z. B. bei Sekunde 2 und 3). Auch im niederfrequenten Bereich sind Artefakte zu erkennen, beispielsweise ca. bei Sekunde 3 des Audio-Deepfakes.

Statt der STFT wird nun die CWT eingesetzt. Analog zu Abbildung 2.1 stellt Abbildung 2.2 die Koeffizienten der CWT für die Datei *LJSpeech 008-0217* und dem von MelGAN erzeugten Deepfake vergleichend dar. Als Mutterwavelet wurde das Shannon-Wavelet mit mittlerer Frequenz $f_c = 0.87$ Hz und Bandbreite $f_b = 0.001$ Hz verwendet. Da es sich nun nicht mehr um eine Zeit-Frequenz-Darstellung, sondern eine Zeit-Skalen-Darstellung handelt, sind Skalogramme nicht direkt mit Spektrogrammen vergleichbar. Dennoch ist eine

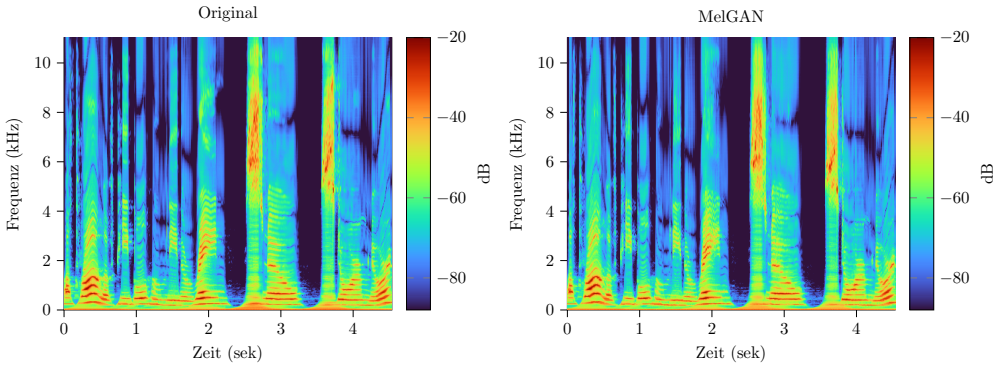


Abbildung 2.2: Skalogramme des Satzes „This was more particularly the practice in London.“ als Originalaufnahme (links) aus LJSpeech und des Audio-Deepfakes (rechts) von [FS21] der Bezeichnung *LJSpeech 008-0217*. Es handelt sich um die CWT des rohen Audiosignals, wobei als Wavelet das Shannon-Wavelet mit $f_b = 0.001$ Hz und $f_c = 0.87$ Hz gewählt wurde. Es wurden die CWTs mit 512 verschiedenen Skalen berechnet. Im hohen Frequenzbereich sind die Koeffizienten des Audio-Deepfakes im Vergleich zum Original teils niedriger und wirken verwaschener. Die Koeffizienten in den tieferen Frequenzen zeigen hingegen beim Deepfake horizontale Artefakte (z. B. bei Sekunde 1.5). Im Vergleich zur STFT in Abbildung 2.1 sind generell in den höheren Frequenzen weniger Frequenz-Informationen zu erkennen.

Zuordnung der Skalen a zu Frequenzen f_p möglich, da diese über die mittlere Frequenz f_c des verwendeten Mutterwavelets durch [KSW08, S. 116]

$$f_p = \frac{f_c}{a} \tag{2.1}$$

im Zusammenhang mit den Skalen stehen [KSW08][LMR98, Kap. 1.3.1]. Dies lässt sich damit erklären, dass bei der Berechnung der CWT das Skalarprodukt konzeptionell die Ähnlichkeit des Wavelets mit dem Signal berechnet [Mer20]. Da das Mutterwavelet eine feste mittlere Frequenz besitzt, die es charakterisiert, ist das Skalarprodukt des Wavelets mit dem Signal an der Stelle am größten, wo sich die mittlere Frequenz des Wavelets der Frequenz des Signals annähert.

In PyWavelets ist mit der Methode `scale2frequency` die Umrechnung von Skalen zu Frequenzen und mit `frequency2scale` die Umrechnung von Frequenzen zu Skalen gemäß (2.1) implementiert. Dafür wird das verwendete Wavelet mit übergeben. Für die Erzeugung der Skalogramme in Abbildung 2.2, die vergleichbar mit Spektrogrammen sind, wird eine lineare Frequenzskala erzeugt und über `frequency2scale` in Skalen für das entsprechende Wavelet übersetzt. Diese werden der Methode `cwt` der PyTorch-Wavelet-Toolbox übergeben, die zu den Skalen die CWT an allen abgetasteten Werten berechnet. Die Frequenzinformationen werden nur bis zu einer maximalen Frequenz von 11025 Hz betrachtet, was der Nyquistfrequenz $f_n = \frac{A}{2}$ entspricht [JMP97, S. 1488], die beschreibt, ab welcher analysierten Frequenz das sogenannte Aliasing eintritt.

Aliasing [Mer20, S. 100f]

Letzteres führt zu einer erschwerten Unterscheidbarkeit der Frequenzanteile des Signals an allen Abtastwerten, da Frequenzen oberhalb von f_n als Alias-Effekte von Frequenzen unterhalb von f_n auftreten. Dadurch ist eine eindeutige

Signalrekonstruktion nicht mehr möglich. Um Aliasing zu vermeiden, werden die zu analysierenden Frequenzen unterhalb von f_n gewählt.

In Abbildung 2.2 sind wie in Abbildung 2.1 deutliche Unterschiede zwischen der Originalaufnahme und dem Audiodeepfake zu erkennen. Diese Unterschiede zeigen sich bei allen sieben Netzwerkarchitekturen in unterschiedlicher Stärke. Im gesamten analysierten Frequenzbereich fehlen Frequenzinformationen, die sich im Bereich von $1 - 5$ kHz sichtbar durch Verzerrungen der enthaltenen Frequenzen (z. B. bei Sekunde 2 gut zu erkennen) und im Bereich von $5 - 11$ kHz durch Verschmierung beziehungsweise schwächere Koeffizienten äußern.

Vergleich der STFT und CWT

Um den in Abschnitt 1.4.2 theoretisch eingeführten Unterschied zwischen der STFT und der CWT grafisch zu verdeutlichen, wurde die vom MelGAN generierte Audiodatei *LJSpeech 008-0217* mithilfe dieser beiden Transformationen untersucht. Abbildung 2.3 und 2.4 zeigen den Unterschied visuell: Während das Spektrogramm der STFT mit einer Fensterfunktion kurzer Länge auf der linken Seite der Abbildung eine hohe Zeitlokalisierung der Frequenzen andeutet, ist die Frequenzauflösung klein. Bei Verwendung eines längeren Fensters (rechts) steigt die Frequenzauflösung und die Zeitauflösung sinkt. Dies ist mit Hilfe der Heisenbergschen Unschärferelation, wie in Abschnitt 1.3.1 diskutiert, zu erklären.

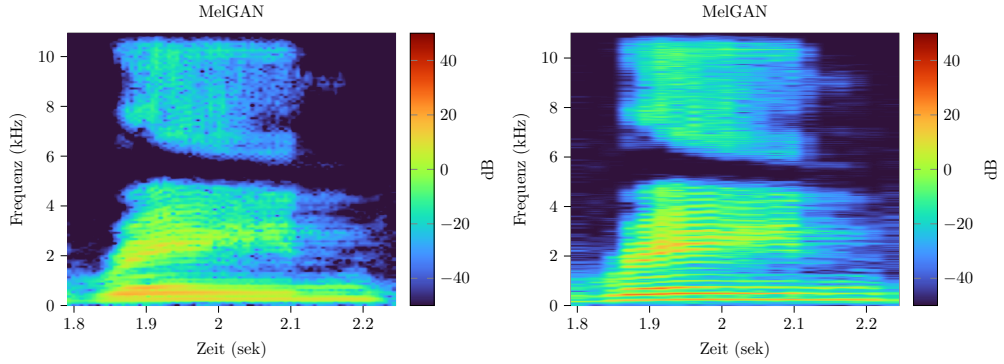


Abbildung 2.3: Analyse eines Audio-Deepfakes mit der STFT und verschiedenen Fensterlängen: kurzes Fenster (links) mit Fensterlänge von 255 Samples, langes Fenster (rechts) mit Fensterlänge von 1023 Samples. Es handelt sich um einen Zeitausschnitt des rechten Spektrogramms in Abbildung 2.1, wobei nur die Fensterlänge variiert wurde. Für das kurze Fenster ist die Zeitlokalisierung der Frequenzen hoch, jedoch die Frequenzlokalisierung gering. Im rechten Spektrogramm liegt für das lange Fenster das Gegenteil vor.

Im Vergleich dazu wird mithilfe der CWT im rechten Diagramm von Abbildung 2.4 deutlich, dass sowohl eine hohe Zeit- als auch eine hohe Frequenzauflösung erreicht werden kann. Es wird ersichtlich, dass besonders im niederfrequenten Bereich die Frequenzauflösung hoch ist, wobei leichte Verschmierungseffekte im zeitlichen Bereich zu erkennen sind, und im hochfrequenten Bereich die Zeitauflösung steigt. Diese Beobachtung deckt sich mit den theoretischen Vorüberlegungen und der Beobachtung im Beispiel in [Mer20, S. 317] und bestärkt die These, dass die CWT eine kompaktere und genauere Zeit-

Frequenzanalyse von Audiodaten möglich machen kann als die ihr ähnlichen STFT.

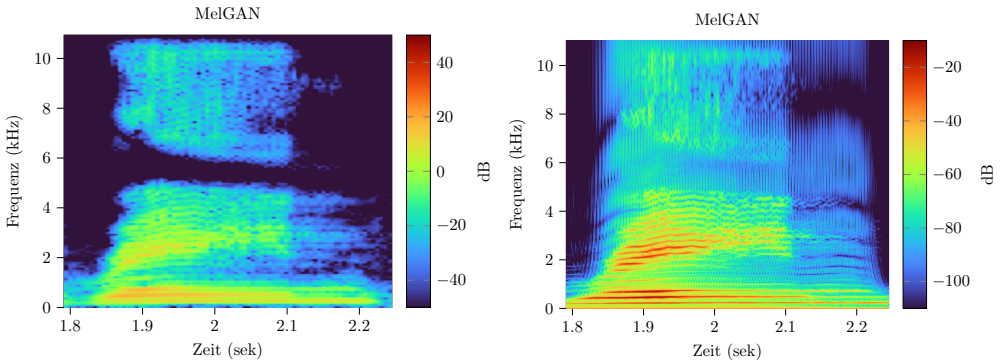


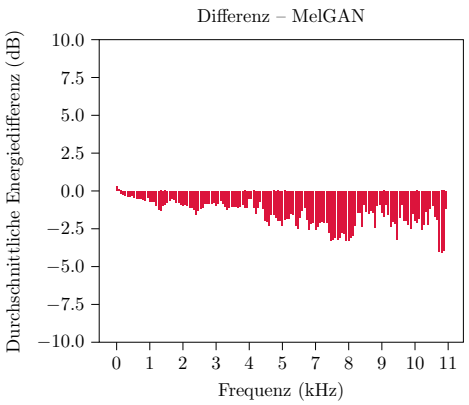
Abbildung 2.4: Analyse eines Audio-Deepfakes mit verschiedenen Transformationen: Spektrogramm der STFT mit einem kurzem Fenster (links) und einer Fensterlänge von 255 Samples, Skalogramm der CWT mit Shannon-Wavelet (rechts) mit $f_b = 0.001$ Hz und $f_c = 0.87$ Hz. Es handelt sich um Zeitausschnitte des rechten Spektrogramms in Abbildung 2.1 mit kürzerem Fenster und des rechten Skalogramms in Abbildung 2.2. Für das kurze Fenster ist die Zeitlokalisierung der Frequenzen hoch, jedoch die Frequenzlokalisierung gering. Im rechten Skalogramm ist sowohl im tiefen Frequenzbereich eine hohe Frequenzauflösung zu erkennen, was an der Trennbarkeit der Stimmartefakte zu sehen ist (wenig Verschmierung). Im hohen Frequenzbereich ist die Zeitauflösung hingegen höher als z. B. im Vergleich zwischen der STFT mit kurzem und langen Fenster in 2.1. Die CWT schafft damit einen Trade-Off zwischen Zeit- und Frequenzauflösung. Gleichzeitig treten bei der CWT des Originals und des Deepfakes gleichermaßen zeitlich horizontale Artefakte über viele Skalen auf, die vermutlich auf die Bandbreite des Wavelets zurückzuführen sind.

2.3.2 *Durchschnittliche Signalenergien der Datensätze*

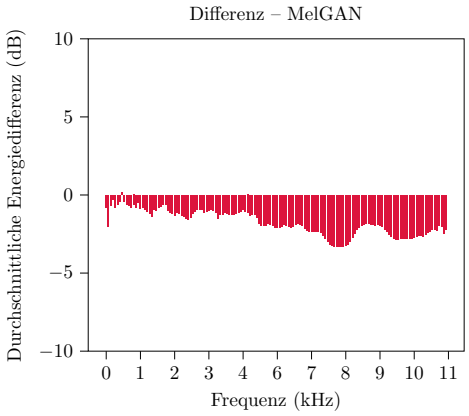
Des Weiteren wird analog zu [FS21] mit der STFT für jeden Datensatz die durchschnittliche Signalenergie pro Frequenzabschnitt berechnet. Dies wird für den Originaldatensatz ebenfalls durchgeführt. Die Differenz der beiden Durchschnittsenergien wird in den Abbildungen 2.5a und 2.5c in Balkendiagrammen für die Architekturen MelGAN und FB-MelGAN gezeigt. Die Ergebnisse von [FS21] konnten reproduziert werden und wie auch bereits [FS21] beobachten, sind im Durchschnitt zwischen den Originalaufnahmen und den generierten Daten große Unterschiede zu erkennen. Gerade im hochfrequenten Bereich zeigen sich deutliche Differenzen.

Dieser Analyseansatz wird nun auf die CWT ausgeweitet, indem statt der STFT die kontinuierliche Wavelettransformation für die Analyse verwendet wird. Die resultierenden Diagramme für die von MelGAN und FB-MelGAN generierten Datensätze sind in den Abbildungen 2.5b und 2.5d zu sehen. Weitere Resultate für die restlichen Netzarchitekturen sind im Anhang A zusammengestellt. Es ist deutlich zu erkennen, dass die CWT eine ähnliche Verteilung der

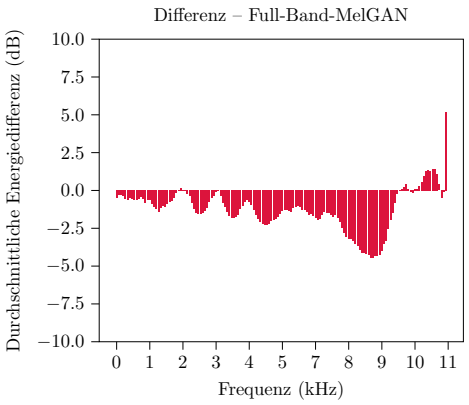
Signalenergie erzeugt und im Vergleich zur STFT im höherfrequenten Anteil der Signale glattere Differenzverteilungen ergibt.



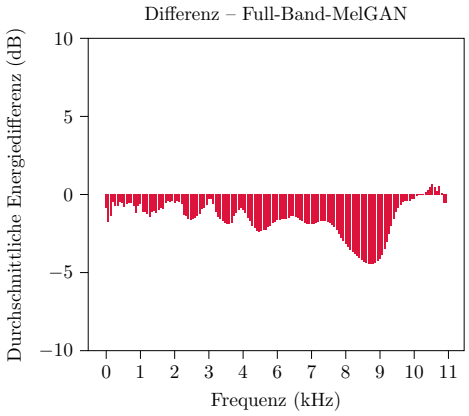
(a) LJSpeech-MelGAN/STFT



(b) LJSpeech-MelGAN/CWT



(c) LJSpeech-FB-MelGAN/STFT



(d) LJSpeech-FB-MelGAN/CWT

Abbildung 2.5: Differenz der Signalenergien von 13 100 Audiodateien aus LJSpeech und denjenigen von MelGAN und FB-MelGAN generierten Gegenstücken. Die STFTs wurden mit dem Hann-Fenster und einer Fensterlänge von 150 Samples über 150 Frequenzbereiche berechnet. Für die CWT wurde das komplexe Morlet-Wavelet mit $f_b = 3.3$ Hz und $f_c = 4.17$ Hz verwendet und die Transformation über 150 Skalen berechnet. Die Ähnlichkeit der STFT und CWT ist eindeutig zu erkennen, wobei die Energiedifferenzen im hochfrequenten Bereich der Audio-Deepfakes aus MelGAN bei der CWT höher ausfallen als bei der STFT. Für die Audio-Deepfakes aus FB-MelGAN ergeben sich hingegen für die STFT höhere Differenzen als für die CWT. Generell scheinen die Differenzen für die CWT zwischen 6 und 11 kHz glatter als für die STFT.

2.4 DEEPFAKE-ERKENNUNG

Um die Performanz und Genauigkeit verschiedener Klassifikatoren zu untersuchen, wird im Rahmen dieser Arbeit mithilfe des PyTorch-Moduls eine Trainings- und Testumgebung erstellt. In dieser Umgebung werden drei verschiedene künstliche neuronale Netze mit dem transformierten WaveFake-Datensatz trainiert und deren Testgenauigkeit bestimmt. Im folgenden Unterkapitel werden die verwendeten Hyperparameter der verschiedenen Modelle und die Ergebnisse der Klassifikation des WaveFake-Datensatzes vorgestellt. Des Weiteren werden in Abschnitt 2.4.3 die Modelle mittels eines Testdatensatzes über die Metriken Genauigkeit und EER evaluiert, um die Performanz der Modelle mit den Ergebnissen aus [FS21] zu vergleichen. Die Robustheit der Modelle wird anhand der Average Equal Error Rate (dt.: durchschnittliche Gleichfehlerrate) (aEER) verglichen, wobei die EER jedes Modells gegen die Testdatensätze aller Fake-Datensätze zu LJSpeech aus WaveFake kreuzvalidiert werden und über die Teilergebnisse der Durchschnitt berechnet wird. Abschließend werden in Abschnitt 2.4.6 die IG der Klassifikatoren über einen Teil der Testdaten ausgewertet.

2.4.1 *Datenvorverarbeitung, Training und Testen*

Um eine umfassende Untersuchung des WaveFake-Datensatzes zu gewährleisten, werden zwei verschiedene Ansätze bei der Datensatz-Wahl betrachtet:
(1) Im ersten Ansatz werden die von den GAN-Architekturen erzeugten Subdatensätze jeweils in Kombination mit dem LJSpeech-Datensatz in eine Trainings-, Validierungs- und Testmenge geteilt. Alle drei Mengen sind disjunkt. Für diese Untermengen ergeben sich für zwei verschiedene Datensätze mit je 13 100 Audiosequenzen eine Trainingsmenge aus 231 768 Audiodateien, einer Validierungsmenge aus 33 107 Sequenzen und eine Testmenge aus 66 185 Audiosequenzen. Der Fokus liegt dabei auf der einzelnen Architektur.
(2) Weiterhin werden im zweiten Ansatz alle Subdatensätze mit dem LJSpeech-Datensatz kombiniert, wobei stets die gleiche Anzahl an generierten und echten Audiodateien vorhanden sind. Es ergibt sich ein Datensatz derselben Größe wie zuvor, mit 50 % LJSpeech-Daten und jeweils ca. 7.14 % der Daten von MelGAN, MelGAN Large, Multi-Band-MelGAN, Full-Band-MelGAN, Parallel-WaveGAN, HiFi-GAN und WaveGlow. Dieser besteht aus 231 755 Trainingssequenzen, 33 095 Validierungssequenzen und 66 225 Testsequenzen. Die Abtastrate von 22050 Hz wird beibehalten. Die aus den Audiodateien geladenen Tensoren werden in Stücke der Länge 0.5 s zerschnitten und gespeichert.

2.4.2 *Hyperparameter für die binäre Klassifikation*

Es werden drei verschiedene künstliche neuronale Netze verwendet. Das erste, 2D-CNN *LearnDeepNet (LDN)*, besitzt sechs zweidimensionale Faltungsschichten mit einer Kernelgröße von 3×3 und einer Stride von 2 (siehe Tab. 2.1 links), gefolgt von einer dicht besetzten Schicht. Das 1D-CNN *OneDNet (ODN)* besitzt sechs eindimensionale Faltungsschichten, mit einer Stride von 2 und

der Kernelgröße 53 in der ersten Schicht und eine Kernelgröße von 3 in den nachfolgenden Schichten (siehe Tab. 2.1 Mitte). Zuletzt handelt es sich bei dem *LearnNet (LN)* um eine reduzierte Form von *LearnDeepNet*, worin die letzten beiden Faltungsschichten außerdem einer Stride von 3 verwenden (siehe Tab. 2.1 rechts). Im Unterschied zu den 2D-CNNs wird beim OneDNet die Frequenzdimension des transformierten Signals als Kanaldimension aufgefasst, woraufhin die eindimensionale Faltung nur noch entlang der Zeitdimension berechnet wird. Bei LDN und LN hingegen findet die Faltung entlang der Zeit- und der Frequenzachse statt. Außerdem wird in allen Netzen nach jeder Faltungsschicht mithilfe von Batch-Normalisierung [IS15] regularisiert. Die Wahl der Stride und der Batch-Normalisierung orientiert sich an den Blöcken aus dem in der Bilderkennung häufig eingesetzten *ResNet* [He+16]. Wie aus der Tabelle ersichtlich wird, ist das 1D-CNN ODN mit 303 012 trainierbaren Parametern mit dem 2D-CNN LDN vergleichbar, das einen Parameterraum von 303 364 besitzt. LearnNet hingegen besitzt eine reduzierte Anzahl von 180 772 Parametern.

2D-CNN *LDN*		1D-CNN *ODN*		2D-CNN *LN*	
CWT	(1, 150)	CWT	(1, 150)	CWT	(1, 150)
Conv2d	(1, 32)	Conv1d	(150, 32, 53)	Conv2d	(1, 32)
BN2d	(32, 32)	BN1d	(32, 32)	BN2d	(32, 32)
ReLU		ReLU		ReLU	
Conv2d	(32, 32)	Conv1d	(32, 32, 3)	Conv2d	(32, 32)
BN2d	(32, 32)	BN1d	(32, 32)	BN2d	(32, 32)
ReLU		ReLU		ReLU	
Conv2d	(32, 32)	Conv1d	(32, 32, 3)	Conv2d	(32, 64)
BN2d	(32, 32)	BN1d	(32, 32)	BN2d	(64, 64)
ReLU		ReLU		ReLU	
Conv2d	(32, 64)	Conv1d	(32, 64, 3)	Conv2d	(64, 128)
BN2d	(64, 64)	BN1d	(64, 64)	BN2d	(128, 128)
ReLU		ReLU		ReLU	
Conv2d	(64, 128)	Conv1d	(64, 64, 3)		
BN2d	(128, 128)	BN1d	(64, 64)		
ReLU		ReLU			
Conv2d	(128, 128)	Conv1d	(64, 64, 3)		
BN2d	(128, 128)	BN1d	(64, 64)		
ReLU		ReLU			
		AvgPool1d	(2, 2)	AvgPool2d	(2, 2)
Dense	(21888, 2)	Dense	(5440, 2)	Dense	(39168, 2)

Tabelle 2.1: Netzarchitekturen der drei verwendeten CNNs LearnDeepNet, OneDNet und LearnNet. In der jeweils linken Spalte ist die Funktion der Schichten aufgelistet und rechts daneben in Klammern die Zahl der Eingangs- und Ausgangskanäle, gefolgt von der Größe des Faltungskernels (falls abweichend von dem Standard-Kernel der Größe 3).

Jedem Netz ist außerdem eine Schicht vorangestellt, in der die Audiosequenzen mithilfe der CWT in eine Zeit-Skalen-Darstellung transformiert werden. Es werden insgesamt 150 Skalen des gewählten Wavelets berechnet, die in gleichen Abständen mit (2.1) aus einer linearen Frequenzskala gewählt sind. Die unter-

suchte Frequenzskala ist, aufgrund des in Abschnitt 2.3.1 erläuterten Aliasings oberhalb der Nyquist-Frequenz von 11025 Hz, begrenzt auf 1000 Hz bis 9500 Hz. Es wird hauptsächlich das komplexe Morlet-Wavelet mit Bandbreite $f_b = 3.3$ Hz und Zentrumsfrequenz $f_c = 4.17$ Hz (Abk.: cmor 3.3-4.17) untersucht, aber auch solche mit anderen Parametern f_b, f_c sowie das Shannon-Wavelet werden mit diesem Wavelet verglichen, um die Möglichkeiten verschiedener Filter auszutesten. Als letztes steht eine vollvernetzte Schicht, die zwei Ausgabekanäle besitzt. Diese werden als die Wahrscheinlichkeiten interpretiert, dass das Eingabesignal entweder der Klasse„Fake“ oder „Nicht-Fake“ angehört.

In den Basisexperimenten, deren Ergebnisse im nächsten Abschnitt vorgestellt werden, sind f_b und f_c fest gewählt. Darüber hinaus wird in dieser Arbeit mit trainierbaren Bandbreiten und Zentrumsfrequenzparametern experimentiert. Die Netze werden mit einer Minibatchgröße von 128 und einer Lernrate von $\eta = 0.0001$ über 10 Epochen trainiert. Als Optimierungsalgorithmus wird Adam [KB14] und als Kostenfunktion die Kreuzentropie verwendet. Als Aktivierungsfunktion kommt ReLU zum Einsatz. Um die Generalisierungsfähigkeit zu erhöhen, werden verschiedene Regularisierungsmethoden verwendet. Neben Batch-Normalisierung (BN1d und BN2d) wird ein Weight Decay von $\lambda = 0.0001$ angewandt. Außerdem wird Early Stopping eingesetzt, wenn die Validierungsgenauigkeit der Netze 100 % erreicht oder 10 Epochen durchgelaufen sind. Um reproduzierbare und statistisch signifikante Ergebnisse zu erzeugen, werden alle Modelle mit den festen Seeds 0 bis 4 auf einem Rechner unter Rocky Linux 8.6, mit einer NVIDIA-A40-GPU mit 48 GB RAM und unter Nutzung von 8 Kernen einer Intel Xeon(R) Silver 4309Y 8C/16T CPU mit 2.8 GHz und 384 GB RAM trainiert und getestet. Das Training eines Modells dauert für 10 Epochen ca. 6.5 Stunden.

2.4.3 *Ergebnisse der Erkennung aus bekannten Quellen*

Die Ergebnisse der Experimente sind den nachfolgenden Tabellen zu entnehmen, wobei alle Experimente jeweils fünf mal durchgeführt wurden, weshalb der Mittelwert μ und die Standardabweichung σ angegeben werden. Sofern nicht anders beschrieben, ist die Quelle der Trainings- und Testdaten für jedes Modell entweder das entsprechend bekannte GAN selbst oder LJSpeech (nach Ansatz (1) aus Abschnitt 2.4.1). Die Ergebnisse für die verschiedenen Modelle sind in Tabellen 2.2 und für verschiedene Wavelets in Tabelle 2.3 und 2.4 dargestellt.

Vergleich der Netze

Wie der Tabelle 2.2 zu entnehmen ist, haben die trainierten Modelle allgemein für alle GANs eine sehr hohe Genauigkeit und eine geringe EER. Im Vergleich zum OneDNet und dem LearnNet hat das LearnDeepNet für alle Subdatensätze jeweils die höchste Genauigkeit und konvergiert somit am weitesten. Das LDN-Modell für den MelGAN (L)-Datensatz diskriminiert die Audio-Deepfakes im Schnitt mit einer sehr hohen Genauigkeit von 99.99 ± 0.02 % von den Originaldaten. Dagegen wird der FB-MelGAN-Datensatz von dem entsprechenden LDN-Modell mit einer kleineren Genauigkeit von 97.53 ± 0.98 % klassifiziert. Für das OneDNet-Modell ist erkennbar, dass es auf MelGAN und MelGAN (L) zum LDN vergleichbar weit konvergiert, für die restlichen Subdatensätze

jedoch eine um ca. 4 bis 9 % geringere Genauigkeit erreicht. Die geringste Genauigkeit hat der Klassifikator des HiFi-GAN-Datensatzes.

	Genauigkeit [%]		EER	
Methodik	max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
OneDNet ODN (1D)				
MelGAN	99.96	98.96 ± 1.57	0.000	0.003 ± 0.003
MelGAN (L)	99.98	99.85 ± 0.14	0.000	0.001 ± 0.001
MB-MelGAN	97.42	94.73 ± 2.61	0.013	0.032 ± 0.032
FB-MelGAN	93.87	85.76 ± 5.48	0.055	0.105 ± 0.105
HiFi-GAN	97.99	95.84 ± 1.62	0.016	0.026 ± 0.026
WaveGlow	99.60	91.11 ± 9.58	0.003	0.011 ± 0.011
PWG	99.20	92.51 ± 10.84	0.002	0.017 ± 0.017
LearnDeepNet LDN (2D)				
MelGAN	99.99	99.92 ± 0.12	0.000	0.000 ± 0.000
MelGAN (L)	**99.99**	99.99 ± 0.02	**0.000**	0.000 ± 0.000
MB-MelGAN	99.83	99.57 ± 0.32	0.002	0.002 ± 0.000
FB-MelGAN	98.43	97.53 ± 0.98	0.015	0.017 ± 0.002
HiFi-GAN	99.46	99.26 ± 0.23	0.004	0.006 ± 0.002
WaveGlow	99.87	99.78 ± 0.07	0.001	0.002 ± 0.001
PWG	99.87	99.81 ± 0.05	0.001	0.002 ± 0.001
LearnNet LN (2D)				
MelGAN	99.95	99.72 ± 0.25	0.001	0.001 ± 0.001
MelGAN (L)	99.98	99.97 ± 0.01	0.000	0.000 ± 0.000
MB-MelGAN	98.72	98.46 ± 0.17	0.010	0.012 ± 0.002
FB-MelGAN	94.04	93.07 ± 0.72	0.060	0.064 ± 0.004
HiFi-GAN	93.13	90.69 ± 1.46	0.060	0.065 ± 0.004
WaveGlow	99.15	98.82 ± 0.43	0.008	0.012 ± 0.004
PWG	99.10	98.85 ± 0.29	0.009	0.009 ± 0.001

Tabelle 2.2: Ergebnisse der binären Audioklassifikation mit 1D- und 2D-Faltungsnetzen mit dem komplexen Morlet-Wavelet cmor 3.3-4.17 für alle Subdatensätze von WaveFake. Die besten Ergebnisse sind fett markiert. Es wird der Trainingsansatz (1) aus Abschnitt 2.4.1 verwendet. Die Testgenauigkeit und die Equal Error Rate (EER) auf dem Testdatensatz ist gemittelt über fünf Durchläufe mit den Seeds 0 bis 4, wodurch das Mittel μ und die Standardabweichung σ angegeben werden kann. Generell wird für alle LDN-Modelle eine hohe Genauigkeit von fast 100 % und geringe EER erreicht, was dafür spricht, dass die Netze in einem tiefen Faltungsnetz dieser Form Audio-Deepfakes klassifizieren können. Das 1D-Faltungsnetz schneidet im Vergleich für die Subdatensätze aus MB-MelGAN, FB-MelGAN und HiFi-GAN, WaveGlow und PWG schlechter ab als das LDN. Das flachere LearnNet erreicht ähnliche Testgenauigkeiten für MelGAN, MelGAN (L), WaveGlow und PWG.

Um einen Vergleich zu LDN und ODN anzustellen, sind die Ergebnisse für das flachere Netz LN im dritten Block ebenfalls dargestellt. Zusätzlich wurde die Abtastrate des Datensatzes für das LN beim Vorbereiten von 22050 Hz

auf 16000 Hz verringert. Die Testgenauigkeiten der MelGAN-, MelGAN (L)-, MB-MelGAN-, WaveGlow- und PWG-Modelle auf dem LearnNet sind wenige Promillepunkte geringer als bei den LDN-Modellen, aber im Durchschnitt wesentlich höher als beim ODN. Die LN-FB-MelGAN und HiFi-GAN-Modelle schneiden mit im Schnitt $93.07 \pm 0.72\,\%$ und $90.69 \pm 1.46\,\%$ Testgenauigkeit schlechter ab als für das tiefere Netz LDN.

Die großen Differenzen der Testgenauigkeiten zwischen den zweidimensionalen und eindimensionalen Faltungsnetzen könnten darauf hindeuten, dass 2D-Faltungen auf den Zeit-Skalen-Daten der CWT eindeutigere Merkmale hervorbringen als 1D-Faltungen. Es handelt sich bei den LDN- und ODN-Modellen um zwei Netzarchitekturen gleicher Kapazität, was die Vergleichbarkeit unterstützt und die Schlussfolgerung zulässt, dass zumindest die in dieser Arbeit gewählte Netzstruktur des LDN besser abschneidet als die des ODN. Da jedoch die Wahl dieser Netzstrukturen an neuronalen Netzen aus anderen Kontexten orientiert ist, sollte der Zusammenhang zwischen den Testgenauigkeiten der Experimente und den Netzstrukturen als rein spekulativ bewertet werden.

Vergleich der Wavelets

Des Weiteren geht aus Tabelle 2.3 und 2.4 (siehe nächste Seite) für die LearnDeepNet-Architektur unter Nutzung verschiedener Wavelets hervor, dass mit dem komplexen Morlet-Wavelet cmor 3.3-4.17 für alle Generatoren die höchste Genauigkeit erreicht wird. Im Schnitt liegt die Genauigkeit höher als 99 % für dieses Wavelet. Des Weiteren ist zu beobachten, dass alle Modelle für die drei verwendeten Wavelets beim Training auf MelGAN und MelGAN (L) die Testdaten fast vollständig richtig klassifizieren, bei FB-MelGAN jedoch eine erkennbar geringere Genauigkeit erreichen.

Auch für die Trainingsmenge, die zur Hälfte aus echten Audiodaten und zur anderen Hälfte aus Teilen der generierten Subdatensätze von WaveFake besteht (Ansatz 2), erreichen die Modelle, die das cmor 3.3-4.17-Wavelet verwenden, die höchste Genauigkeit von im Schnitt $96.18 \pm 1.00\,\%$. Dies ist in Tabelle 2.4 dargestellt. Mit $90.33 \pm 1.17\,\%$ Genauigkeit folgen die LDN-Modelle, die das Shannon-Wavelet shan 0.01-0.40 verwenden. Das cmor 4.60-0.87-Wavelet, welches im ersten Ansatz (siehe Tabelle 2.3) für alle Subdatensätze ähnlich abschneidet wie das Shannon-Wavelet, ist für den zweiten Ansatz zwar mit 93.59 % maximaler Genauigkeit eine höhere Genauigkeit als beim Shannon-Wavelet erreicht, jedoch ist die mittlere Standardabweichung im Vergleich mit $\pm 4.95\,\%$ sehr viel höher. Dies könnte ein Hinweis darauf sein, dass das cmor 3.3-4.17-Wavelet im Vergleich zu cmor 4.6-0.87 allgemeinere Zeit-Frequenz-Informationen für alle Generatoren herausfiltert, die für das anschließende CNN von Interesse sind.

Im Vergleich zur STFT existiert für die CWT eine große Menge verschiedener Wavelets, die unterschiedliche Zeit-Skalen-Informationen herausfiltern können. Die Ergebnisse in den Tabellen 2.3 und 2.4 legen die vage Vermutung nahe, dass durch Änderung der Fensterfunktion auch die Performanz von CNNs verbessert werden kann.

	Genauigkeit [%]		EER	
GAN/Wavelet	max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
MelGAN				
cmor 4.6-0.87	99.97	99.94 ± 0.05	0.000	0.000 ± 0.000
cmor 3.3-4.17	**99.99**	99.92 ± 0.12	**0.000**	0.000 ± 0.000
shan 0.01-0.4	99.97	99.93 ± 0.06	0.000	0.000 ± 0.000
MelGAN (L)				
cmor 4.6-0.87	99.97	99.92 ± 0.07	0.000	0.001 ± 0.000
cmor 3.3-4.17	**99.99**	99.99 ± 0.02	**0.000**	0.000 ± 0.000
shan 0.01-0.4	99.99	99.98 ± 0.01	0.000	0.000 ± 0.000
MB-MelGAN				
cmor 4.6-0.87	99.35	99.05 ± 0.24	0.005	0.006 ± 0.001
cmor 3.3-4.17	**99.83**	99.57 ± 0.32	**0.002**	0.002 ± 0.000
shan 0.01-0.4	99.17	98.78 ± 0.33	0.007	0.011 ± 0.003
FB-MelGAN				
cmor 4.6-0.87	97.14	96.17 ± 1.03	0.026	0.029 ± 0.002
cmor 3.3-4.17	**98.43**	97.53 ± 0.98	**0.015**	0.017 ± 0.002
shan 0.01-0.4	94.65	93.31 ± 0.89	0.053	0.058 ± 0.004
HiFi-GAN				
cmor 4.6-0.87	99.01	98.71 ± 0.27	0.010	0.012 ± 0.002
cmor 3.3-4.17	**99.46**	99.26 ± 0.23	**0.004**	0.006 ± 0.002
shan 0.01-0.4	98.36	97.99 ± 0.25	0.016	0.018 ± 0.002
WaveGlow				
cmor 4.6-0.87	99.52	99.35 ± 0.15	0.004	0.005 ± 0.001
cmor 3.3-4.17	**99.87**	99.78 ± 0.07	**0.001**	0.002 ± 0.001
shan 0.01-0.4	99.68	99.57 ± 0.14	0.003	0.004 ± 0.001
PWG				
cmor 4.6-0.87	99.73	99.65 ± 0.05	0.003	0.003 ± 0.000
cmor 3.3-4.17	**99.87**	99.81 ± 0.05	**0.001**	0.002 ± 0.001
shan 0.01-0.4	99.71	99.57 ± 0.13	0.003	0.004 ± 0.001

Tabelle 2.3: Gegenüberstellung der Ergebnisse der binären Audioquellenklassifikation unter Verwendung verschiedener Wavelets und des LDN für alle Subdatensätze. Die besten Ergebnisse sind pro Generator fett gekennzeichnet. Es wird der *Trainingsansatz (1)* aus Abschnitt 2.4.1 verwendet. Die linke Spalte gibt den Subdatensatz von WaveFake an, auf dem das LDN unter Nutzung des darunter angegebenen Wavelets trainiert und getestet wird. Alle Modelle, die das cmor 3.3-4.17 Wavelet verwenden, erreichen nach dem Training auf den verschiedenen Subdatensätzen die höchste Genauigkeit.

Dabei ist zu erwähnen, dass die Wahl des Wavelets cmor 3.3-4.17 nicht willkürlich getroffen, sondern das Ergebnis eines Optimierungsprozesses ist. Dabei werden f_b und f_c als trainierbare Parameter in die Optimierung des neuronalen Netzes mit eingebunden und mit einer Lernrate von $\eta = 0.01$ trainiert. Es werden dann die Werte der trainierten Wavelet-Parameter extrahiert, die nach zehn Epochen erreicht sind. Auf dem Subdatensatz von HiFi-GAN ergibt sich aus den Anfangsparametern $f_c = 0.87, f_b = 4.6$ eben jenes komplexe Morlet-Wavelet cmor 3.3-4.17. Dieses wird wiederum als statisches Wavelet für einen

	Genauigkeit [%]		EER	
Wavelet	max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
cmor 4.60-0.87	93.59	89.56 ± 4.95	0.062	0.072 ± 0.009
cmor 3.30-4.17	**97.07**	96.18 ± 1.00	**0.029**	0.033 ± 0.003
shan 0.01-0.40	91.47	90.33 ± 1.17	0.085	0.088 ± 0.003

Tabelle 2.4: Gegenüberstellung der Ergebnisse der binären Audioquellenklassifikation unter Verwendung verschiedener Wavelets für *Trainingsansatz (2)* aus 2.4.1, bei dem die Trainingsmenge zu 50 % aus LJSpeech und zu 50 % aus den Subdatensätzen der Generatoren, mithilfe derer LJSpeech reproduziert wurde, stammt. Als Modell wurde das LDN verwendet. Die besten Ergebnisse sind fett gekennzeichnet. Die linke Spalte gibt den Subdatensatz von WaveFake an, auf dem das LDN unter Nutzung des darunter angegebenen Wavelets trainiert und getestet wurde. Mit der höchsten Genauigkeit und der geringsten Standardabweichung schneiden die Modelle, die das cmor 3.3-4.17 verwenden, am besten ab.

neuen Trainingsdurchlauf verwendet und ergibt die bereits vorgestellten Ergebnisse, die besser ausfallen, als die für das cmor 4.60-0.87-Wavelet. Im Rahmen dieser Arbeit kann nicht validiert werden, dass dieses Vorgehen reproduzierbar ist. Es basiert jedoch auf der Idee, dass der Optimierungsalgorithmus von neuronalen Netzen genutzt werden kann, um Wavelets zu finden, welche die durch die Generatoren erzeugten Artefakte besser herausfiltern.

Wie im nachfolgenden Abschnitt 2.4.4 noch ausführlicher diskutiert wird, kann jedoch kein besseres Ergebnis für das generelle Training mit dem LN und lernenden Wavelets erreicht werden. Die oben geschilderten Ergebnisse weisen allerdings bereits darauf hin, dass durch Anpassung des Wavelets, prinzipiell bessere Ergebnisse erreicht werden können.

Vergleich mit [FS21]

Im Folgenden soll noch mit den Testergebnissen von [FS21] verglichen werden. Tabelle 2.5 zeigt die EER des Ansatzes von [FS21], die Subdatensätze mit einem Gaussian-Mixture-Model und LFCC-Features zu klassifizieren. Im Rahmen dieser Arbeit werden die veröffentlichten Daten mithilfe der von [FS21] vortrainierten Netze erneut evaluiert und den in Tabelle 2 der Publikation angegebenen Werten vorgezogen, da die Bezeichnung der Spalten und Zeilen dort nicht eindeutig erscheint. Die eigens erzeugten Werte weichen jedoch kaum von den Diagonalwerten der Tabelle 2 ab, weshalb eine fehlerhafte Bezeichnung der Spalten und Zeilen angenommen wird. Dem gegenüber werden die Ergebnisse der Experimente mit dem LDN gestellt. In beiden Ansätzen wird der Klassifikator auf demselben Subdatensatz trainiert und getestet, es handelt sich also um eine bekannte Quelle. Es ist zu erkennen, dass die Audioquellenklassifikation mit dem 2D-CNN und der CWT mit cmor 3.3-4.17 für die Subdatensätze MelGAN, MelGAN (L) und HiFi-GAN eine deutlich niedrigere EER ergibt als der Ansatz von [FS21] und damit besser abschneidet. Die EER für WaveGlow, PWG, MB- und FB-MelGAN sind nah an den Werten von [FS21].

	EER		
	GMM STFT-LFCC [FS21]	LearnDeepNet cmor 3.30-4.17	
GAN	min	min	$\mu \pm \sigma$
MelGAN	0.149	**0.000**	0.000 ± 0.000
MelGAN (L)	0.044	**0.000**	0.000 ± 0.000
MB-MelGAN	**0.002**	*0.002*	0.002 ± 0.000
FB-MelGAN	0.025	**0.015**	0.017 ± 0.002
HiFi-GAN	0.029	**0.004**	0.006 ± 0.002
WaveGlow	0.003	**0.001**	0.002 ± 0.001
PWG	0.001	**0.001**	0.002 ± 0.001

Tabelle 2.5: Vergleich der Ergebnisse der Audio-Deepfake-Erkennung mittels eines Gaussian Mixture Models und STFT-basierten LFCC-Merkmalen [FS21] mit den Ergebnissen des LDN. Die besten Ergebnisse für jeden Generator sind fett gekennzeichnet. Sofern möglich, ist der Durchschnitt und die Standardabweichung über fünf Trainingsdurchläufe angegeben. In beiden Ansätzen wurde der Klassifikator auf demselben Generator trainiert und getestet. Das LDN aus dieser Arbeit klassifiziert für alle GANs bis auf MB-MelGAN deutlich mehr Testdaten richtig, als der Ansatz von [FS21]. Besonders groß ist dieser Unterschied bei MelGAN und MelGAN (L).

2.4.4 *Ergebnisse der Erkennung aus unbekannten Quellen*

Im folgenden Abschnitt wird die Robustheit der Klassifikatoren getestet. Dahinter steht die Annahme, dass in der Realität verschiedenste Generatoren von Audio-Deepfakes zum Einsatz kommen und sich diese stetig weiterentwickeln. Die auf einem der Subdatensätze trainierten Modelle werden dafür mit den Testmengen der untersuchten Generatoren getestet und somit festgestellt, ob die Modelle auf Deepfakes anderer Generatoren generalisieren können. Dies wird im folgenden als Kreuzvalidierung bezeichnet und orientiert sich an dem Vorgehen von [FS21].

In Tabelle 2.6 sind die Ergebnisse dieser Evaluierung für das LDN und das ODN dargestellt. Die linke Spalte benennt die jeweilige Trainingsmenge und die rechten Spalten geben jeweils die EERs auf den Testmengen der in der obersten Zeile bezeichneten Subdatensätze aus WaveFake an. Sofern die Trainings- und Testdatensätze demselben Subdatensatz entspringen, ist der Eintrag grau dargestellt. Für jedes Modell wurde außerdem der beste Wert fett gekennzeichnet, also diejenige EER, die am geringsten ist. Um die Robustheit der trainierten Audioklassifikatoren mit den Ergebnissen von [FS21] zu vergleichen, sind die aEER dieser Modelle in Tabelle 2.7 gegenübergestellt.

Tabelle 2.6 zeigt, dass alle trainierten LDN- und ODN-Modelle eine sehr geringe mittlere EER auf dem bekannten Datensatz und eine im Vergleich hohe EER für unbekannte Quellen erreichen. Dies lässt vermuten, dass die Modelle spezifische Generator-abhängige Merkmale lernen, die sich nicht auf andere GANs übertragen lassen. Sowohl das MelGAN- als auch das MelGAN (L)-Modell erreichen für LDN und ODN auf der jeweils anderen Testmenge die

Training	EER auf Testmenge						
LDN	Mel	L-Mel	MB-M	FB-M	HiFi	WG	PWG
Mel	0.000	**0.115**	0.483	0.504	0.481	0.461	0.481
L-Mel	**0.237**	0.000	0.510	0.490	0.495	0.498	0.512
MB-M	0.470	0.481	0.002	0.438	0.409	0.323	**0.185**
FB-M	0.330	0.345	**0.201**	0.017	0.255	0.266	0.251
HiFi	0.400	0.341	0.322	0.383	0.006	0.183	**0.135**
WGlow	0.465	0.438	0.418	0.504	0.426	0.002	**0.222**
PWG	0.403	0.425	0.420	0.475	0.412	**0.297**	0.002
ODN							
Mel	0.003	**0.254**	0.473	0.495	0.477	0.478	0.460
L-Mel	**0.398**	0.001	0.506	0.494	0.513	0.504	0.502
MB-M	0.473	0.478	0.032	0.450	0.443	0.349	**0.234**
FB-M	0.392	0.406	**0.272**	0.105	0.335	0.334	0.292
HiFi	0.409	0.389	0.380	0.456	0.026	0.192	**0.173**
WGlow	0.450	0.422	0.428	0.510	0.433	0.011	**0.277**
PWG	**0.360**	0.378	0.469	0.491	0.457	0.379	0.017

Tabelle 2.6: Ergebnisse der Kreuzvalidierung der binären Audioquellenklassifikation mit dem LDN und ODN und cmor 3.3-4.17 für alle Subdatensätze von WaveFake. Die trainierten Modelle (linke Spalte) wurden mit den Testmengen aller Subdatensätze kreuzvalidiert. Die besten Ergebnisse mit den unbekannten Testdatenquellen sind zeilenweise fett markiert, Ergebnisse bei gleicher Trainings- und Testmenge sind grau eingefärbt. Die LDN-Modelle erreichen im Schnitt geringere EERs als die ODN-Modelle können daher als robuster angenommen werden. Generell overfitten alle Modelle auf der Testmenge des bekannten Generators und generalisieren wenig auf unbekannte Quellen.

geringste EER, was sich vermutlich darauf zurückführen lässt, dass MelGAN (L) ein MelGAN mit einem größeren rezeptiven Feld [FS21] ist und sich beide daher sehr ähnlich sind. Des Weiteren ist sowohl bei den LDN- als auch ODN-Modellen zu beobachten, dass MB-MelGAN, HiFi-GAN und WaveGlow die geringsten EERs bei der PWG-Testmenge erreichen. Dies kann darauf hindeuten, dass die von PWG erzeugten Audiodaten ähnliche Artefakte besitzen, wie die von MB-MelGAN, HiFi-GAN und WaveGlow. Für HiFi-GAN ist dies auch an den geringen EER auf den Testmengen von MB-MelGAN und WaveGlow zu erkennen. Bei WaveGlow und MB-MelGAN hingegen ist keine Verbesserung der EER auf der Testmenge von HiFi-GAN zu sehen.

Für den Vergleich verschiedener Ansätze sind die aEERs auf den Subdatensätzen von WaveFake für die LDN- und ODN-Modelle sowie für den GMM-LFCC- und RawNet2-Ansatz von [FS21] in Tabelle 2.7 aufgelistet. Die GMM-LFCCs schneiden für alle Subdatensätze mit Abstand am besten ab, wobei das Minimum für das FB-MelGAN-Modell mit einer aEER von 0.068 erreicht wird. Der RawNet2-Ansatz, bei dem ein modifiziertes Faltungsnetz auf nicht transformierten Audiodaten trainiert wird, schneidet für alle Generatoren in Bezug auf die Erkennungsrate mit einer geringeren EER besser ab als die in dieser

	aEER der Trainingsmenge						
Methode	Mel	L-Mel	MB-M	FB-M	HiFi	WG	PWG
GMM-LFCC [FS21]	0.145	0.140	0.130	0.068	0.088	0.094	0.152
RawNet2 [FS21]	0.346	0.330	0.185	0.185	0.237	0.207	0.176
LN, cmor 3.3-4.17	0.360	*0.378*	0.307	0.237	0.288	*0.307*	0.321
LN, learnWav	*0.356*	0.381	*0.307*	*0.236*	0.280	0.308	0.322
LDN, cmor 3.3-4.17	0.361	0.392	0.330	0.238	*0.253*	0.354	0.348
LDN, cmor 4.6-0.87	0.424	0.434	*0.293*	0.266	0.268	0.313	*0.314*
ODN, cmor 3.3-4.17	0.377	0.417	0.351	0.305	0.289	0.361	0.365

Tabelle 2.7: Vergleich der Robustheit der verschiedenen Netze LDN, ODN (siehe Tab. 2.6) mit 22050 Hz Abtastrate und LN mit 16 kHz Abtastrate, mit und ohne gelernte Wavelets, unter Nutzung verschiedener Wavelets bei unbekannten Audioquellen, mit den Ergebnissen von [FS21]. Die besten eigenen Ergebnisse sind kursiv markiert. Die GMM-LFCCs von [FS21] erreichen für alle Subdatensätze die geringsten EERs auf allen unbekannten Quellen, einschließlich der bekannten Quelle. Das eigene LN mit lernenden Wavelets hat im Vergleich höhere aEERs, schneidet jedoch mit den geringsten eigenen Werten überwiegend besser ab als die LDN- und ODN-Modelle.

Arbeit verwendeten Netze. Außerdem ist zu beobachten, dass das LDN mit dem cmor 3.3-4.17-Wavelet im Schnitt besser abschneidet als das ODN.

Wie in Abschnitt 2.4.3 beschrieben, schneidet das LN in Bezug auf die Testgenauigkeit schlechter ab, als das tiefere LDN. In Tabelle 2.7 zeigt das Ergebnis der Kreuzvalidierung der verschiedenen LN-Modelle, dass bis auf das HiFi-GAN-Modell alle Klassifikatoren eine niedrigere aEER erreichen, als die LDN-Modelle. Dies lässt vermuten, dass die Kombination aus Verringerung der Abtastrate auf 16 kHz und die Verwendung eines flacheren Netzes zu einer höheren Robustheit gegen unbekannte Quellen führt. Dennoch ist zu erwähnen, dass dies auch an der Wahl der inneren Netzstruktur liegen kann.

Des Weiteren sind in Tabelle 2.7 die aEER der LN-Modelle notiert, die mit einem dynamischen Wavelet trainiert wurden. Als Anfangsparameter werden die Werte von cmor 3.3-4.17 gewählt. Die Wavelet-Parameter f_b, f_c sind trainierbare Gewichte des Netzes und lernen mit einer Lernrate von $\eta = 0.00001$. Für die Trainingsmengen von MelGAN, MB-MelGAN, FB-MelGAN und HiFi-GAN werden kleinere aEER erreicht als für die LN-Modelle mit statischen Waveletparametern. Allerdings unterscheiden sich die Werte nicht signifikant voneinander, weshalb dieser Effekt auch zufällig sein kann. Dennoch ist zu vermerken, dass die Ergebnisse mit dynamischen, lernenden Wavelets nicht schlechter werden, sondern vergleichbare Werte erreichen.

2.4.5 *Ergebnisse des Vergleichs der STFT und CWT*

Zuletzt werden in einem separaten Durchlauf die STFT und CWT für das bereits ausführlich getestete LDN evaluiert. Für die STFT wird das sogenannte Von-Hann-Fenster [BT58] als Fensterfunktion gewählt (Standardeinstellung in PyTorch) und für die CWT das komplexe Morlet-Wavelet cmor 3.3-4.17. Die

Fenstergröße der STFT ist 150 und die Fenster werden ohne Lücke verschoben, damit die Ausgabedimensionen mit den der CWT übereinstimmen. Tabelle 2.8 vergleicht die Testgenauigkeiten aller Modelle, die auf den Subdatensätzen von WaveFake (linke Spalte) trainiert werden. Trainings- und Testmenge entstammen derselben Quelle. Im Vergleich zu den vorherigen Experimenten werden die Skalen der CWT nicht auf ein Frequenzband beschränkt, sondern liegen gleichverteilt im zugehörigen Frequenzbereich $[1, 11025]$ Hz.

	Genauigkeit [%]		EER	
Training	max	$\mu \pm \sigma$	min	$\mu \pm \sigma$
STFT				
MelGAN	100.0	99.97 ± 0.04	0.000	0.000 ± 0.000
MelGAN (L)	100.0	99.97 ± 0.05	0.000	0.000 ± 0.000
MB-MelGAN	99.95	99.93 ± 0.02	0.000	0.001 ± 0.001
FB-MelGAN	99.69	99.55 ± 0.14	0.003	0.004 ± 0.004
HiFi-GAN	99.83	99.79 ± 0.04	0.002	0.002 ± 0.002
WaveGlow	99.99	99.97 ± 0.02	0.000	0.000 ± 0.000
PWG	99.98	99.93 ± 0.04	0.000	0.001 ± 0.001
CWT				
MelGAN	99.99	99.98 ± 0.02	0.000	0.000 ± 0.000
MelGAN (L)	100.0	99.98 ± 0.01	0.000	0.000 ± 0.000
MB-MelGAN	99.89	99.52 ± 0.53	0.001	0.002 ± 0.002
FB-MelGAN	98.75	98.11 ± 0.56	0.013	0.015 ± 0.015
HiFi-GAN	99.94	97.88 ± 2.51	0.000	0.002 ± 0.002
WaveGlow	99.94	99.82 ± 0.16	0.001	0.001 ± 0.001
PWG	99.90	99.81 ± 0.07	0.001	0.002 ± 0.002

Tabelle 2.8: Vergleich der Genauigkeit und EER von LDN unter Nutzung der Transformationen STFT und CWT. Für die CWT wurde das cmor 3.3-4.17 Wavelet und für die STFT das Von-Hann-Fenster als Fensterfunktion gewählt. Das analysierte Frequenzband umfasst für beide Transformationen 150 Frequenzen in $[1, 11025]$ Hz. Für das gewählte Wavelet schneidet die CWT für MelGAN und MelGAN (L) im Schnitt besser und für die restlichen Subdatensätze schlechter ab als die STFT.

Für die gewählte Konfiguration erreichen die Modelle, welche die CWT verwenden, für MelGAN und MelGAN (L) mit durchschnittlich 99.98 ± 0.2 die höchsten Testgenauigkeiten und sind damit 0.01 % höher als die STFT-Modelle. Für die restlichen Subdatensätze sind alle Testgenauigkeiten der STFT-Modelle zwischen 0.5 und 1.9 Prozentpunkte höher als bei der CWT.

Zuletzt wird auch für diese beiden Ansätze die Robustheit der Modelle durch Kreuzvalidierung mit den verschiedenen Testmengen untersucht. Die Ergebnisse finden sich in Tabelle 2.9. Es ist zu erkennen, dass die aEER aller Trainingsmengen für die STFT-Modelle geringer sind als für die CWT-Modelle. Das Minimum erreicht das FB-MelGAN-Modell mit einer aEER von 0.103, was den RawNet2-Ansatz von [FS21] um 0.082 übertrifft und nur um 0.035 Punkte größer ist als der beste Wert der GMM-LFCCs [FS21] (siehe Tabelle 2.7). Die

geringste aEER für die CWT-Modelle werden ebenso auf der Trainingsmenge von FB-MelGAN mit durchschnittlich 0.266 erreicht.

Es lässt sich schlussfolgern, dass die Modelle unter dem Hann-Fenster mit der STFT deutlich besser auf unbekannte Quellen generalisieren können, als die Modelle, welche die CWT mit cmor 3.3-4.17 verwenden. Da dieses Experiment nur ein Wavelet verwendet, welches bereits in den vorausgegangenen Experimenten eine geringe Generalisierungsfähigkeit aufgewiesen hat, bleibt der Vergleich beschränkt auf dieses Wavelet.

	aEER der Trainingsmenge						
Methode	Mel	L-Mel	MB-M	FB-M	HiFi	WG	PWG
LDN, STFT	0.318	0.373	0.253	**0.103**	0.132	0.296	0.233
LDN, CWT	0.371	0.398	0.304	0.266	0.332	0.335	0.316

Tabelle 2.9: Vergleich der Robustheit von LDN unter Nutzung der Transformationen STFT und CWT. Für die CWT wurde das cmor 3.3-4.17 Wavelet und für die STFT das Hann-Fenster als Fensterfunktion gewählt. Das analysierte Frequenzband umfasst für beide Transformationen 150 Frequenzen in $[1, 11025]$ Hz. Für das gewählte Wavelet schneidet die CWT für alle Subdatensätze schlechter ab als die STFT.

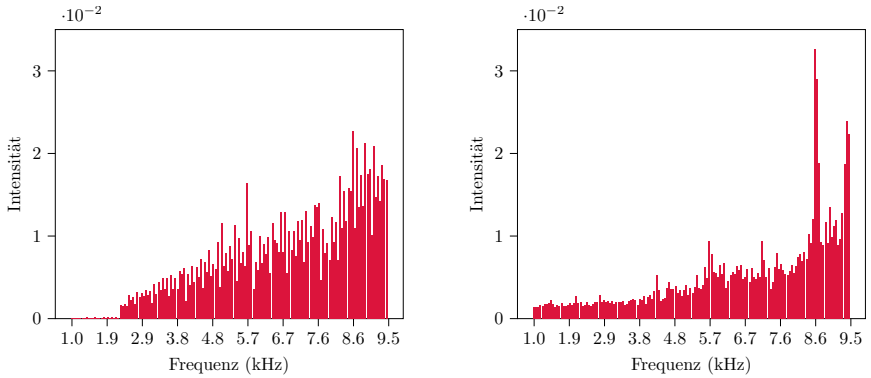
2.4.6 *Ergebnisse der Modellinterpretation mit integrierten Gradienten*

Wie in Abschnitt 1.5.4 motiviert, wird zum besseren Verständnis der trainierten Netze die Attributionsmethode der integrierten Gradienten angewendet. Dazu werden auf allen LDN- und ODN-Modellen für 5056 der CWT-transformierten Fakeaudiodaten aus den entsprechenden Testmengen der bekannten Quellen die integrierten Gradienten berechnet. Es werden dafür die Modelle aus den vorherigen Abschnitten verwendet und über jeweils fünf Netze mit den Seeds 0 bis 4 der Durchschnitt gebildet. Die Ergebnisvektoren werden über die Zeitdimension gemittelt und als Verteilung der Frequenz in Abbildung 2.6 dargestellt. Die Diagramme können als Interpretation herangezogen werden, welche Frequenzen für die trainierten Modelle am stärksten zur Klassifizierung beitragen. Die IG der Modelle, die auf den restlichen Subdatensätzen trainiert wurden, sind im Anhang A zu finden.

In allen Abbildungen 2.6a-2.6d werden im Bereich zwischen 8 und 10 kHz größere Intensitäten für die integrierten Gradienten verzeichnet. Auch der Frequenzbereich um 5.7 kHz erreicht im Vergleich höhere Werte. In Auseinandersetzung mit den IG von Modellen, die andere Wavelets verwenden, scheint letzteres Phänomen für dieses Wavelet vermehrt aufzutreten. Des Weiteren ergeben sich für den Frequenzbereich von 1 bis 2 kHz kaum erhöhte Intensitäten für die IG.

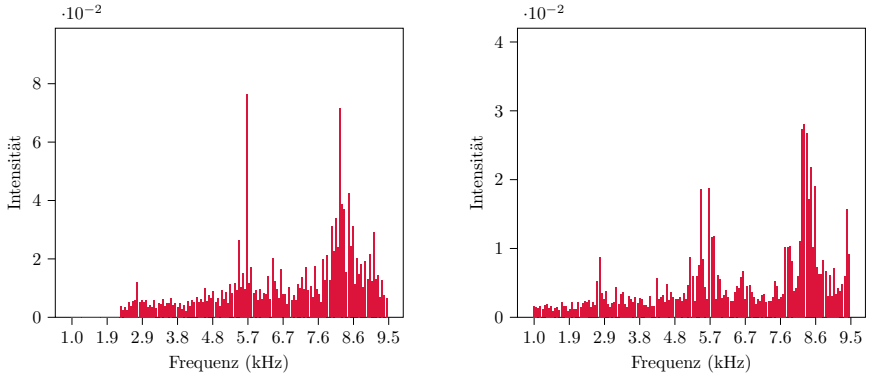
Wie bereits [FS21] herausarbeiten, kann auch für die im Rahmen dieser Arbeit trainierten Deepfake-Klassifikatoren geschlussfolgert werden, dass die Klassifizierung des Audio-Deepfakes als ein solches stärker von hochfrequenten Artefakten in den Daten beeinflusst wird. Dies kann als Hinweis darauf dienen, dass die untersuchten Generatoren besonders in den hochfrequenten

Teilen der Deepfakes erkennbare Artefakte hinterlassen, wie auch [Wol+22] sie beobachten.



(a) IG des LDN-HiFi-GAN

(b) IG des ODN-HiFi-GAN



(c) IG des LDN-WaveGlow

(d) IG des ODN-WaveGlow

Abbildung 2.6: Absolute IG pro Frequenz von vier Modellen mit cmor 3.3-4.17 über je 5056 Fake-Testdaten. Für alle Modelle wurden die Werte von je fünf Modellen mit Seeds 0 bis 4 gemittelt.

3 SCHLUSSFOLGERUNGEN UND AUSBLICK

Im Rahmen dieser Arbeit wurden die Kurzzeit-Fouriertransformation (STFT) und die kontinuierliche Wavelettransformation (CWT) zunächst theoretisch eingeführt und dann deren Anwendung auf zeitkontinuierliche Audiosignale motiviert sowie deren Unterschiede hervorgehoben. Als alternative Transformation wurde die CWT zur Analyse eines Audio-Deepfake-Datensatzes WaveFake von [FS21] als Ersatz für die Standardmethode STFT eingesetzt. Des Weiteren wurden drei verschiedene Versionen eines Faltungsnetzes vorgestellt und auf CWT-transformierten Trainingsdaten verschiedener Deepfake-Generatoren trainiert und getestet, wobei zwei Morlet-Wavelets und ein Shannon-Wavelet zum Einsatz kamen. In ausführlichen Experimenten wurden die Versuche von [FS21] zur Erkennung von Audio-Deepfakes auf die Nutzung der CWT und STFT statt komplexeren Linear-Frequency-Cepstral-Coefficients-Merkmalen (LFCC) reduziert und vergleichend gegenübergestellt.

Dabei konnten bei der Analyse des WaveFake-Datensatzes mit der CWT deutliche Unterschiede zwischen den Audio-Deepfakes und den originalen Signalen festgestellt werden, wobei die Differenz der Signalenergien in den hochfrequenten Informationen für alle untersuchten Generatoren wesentlich höher ausgefallen ist, was den Beobachtungen von [FS21] entspricht und die Ähnlichkeit der STFT zur CWT aufzeigt.

Darüber hinaus konnten die numerischen Experimente zunächst allgemein zeigen, dass tiefe neuronale Netze erfolgreich Deepfakes in wavelettransformierten Audiosignalen erkennen können. Dabei erreichte das eigens konstruierte CNN LearnDeepNet auf dem komplexen Morlet-Wavelet cmor 3.3-4.17 die höchste Testgenauigkeit und erkannte im Schnitt mehr Audio-Deepfakes aus bekannten Quellen als ein CNN mit eindimensionalen Faltungen. In Bezug auf die Generalisierungsfähigkeit der Faltungsnetze auf Deepfakes aus unbekannten Quellen konnte festgestellt werden, dass eine Reduzierung der trainierbaren Gewichte einen positiven Effekt hat. Daraus lässt sich ableiten, dass die verwendeten Netzarchitekturen vermutlich auf Generator-spezifische Merkmale overfitten und dadurch die Artefakte anderer Generatoren schlechter erkennen. Dennoch konnte in einer Analyse der integrierten Gradienten der trainierten Modelle beobachtet werden, dass die Modelle für einen Großteil der Generatoren vermehrt Informationen im hochfrequenten Bereich der Audiosignale zur Klassifizierung heranziehen. Dies deckt sich mit aktuellen Forschungsergebnissen [FS21][Wol+22] und auch mit der anfänglichen Analyse des Datensatzes.

Im Vergleich zu den Versuchen von [FS21] konnten mit CNNs auf den CWT-transformierten Audiodaten eine signifikant höhere Testgenauigkeit erreicht werden als mit Gaussian-Mixture-Models (GMM) auf den spezifischeren Audiomerkmalen LFCC. Auf STFT-transformierten Daten konnten wiederum vergleichbar gute Testgenauigkeiten erreicht werden. Im Anbetracht der Gene-

ralisierungsfähigkeit haben die GMM auf den LFCCs am besten abgeschnitten, gefolgt vom CNN-STFT-Ansatz. Beide Ansätze zeigten signifikant höhere Erkennungsraten als der eigene Ansatz auf CWT-transformierten Daten.

Zusammenfassend kann festgestellt werden, dass die CWT in Kombination mit CNNs durchaus als möglicher Erkennungsansatz für Audio-Deepfakes geeignet ist. Die Überanpassung an die Merkmale eines Generators führt zwar zu einer geringen Generalisierungsfähigkeit, welche jedoch wiederum durch die Erweiterung des Trainingssatzes auf Audio-Deepfakes aus verschiedenen Quellen verbessert werden kann. Allgemein sind die Ergebnisse positiv, dennoch ist eine genauere Auseinandersetzung mit der Wavelet-Transformation in Kombination mit den CNNs vonnöten, um deren volles Potential ausschöpfen zu können. Aufgrund des hohen Ressourcenaufwands für die Trainings- und Testprozesse auf dem WaveFake-Datensatz und durch Vergleich der einzelnen Generatoren untereinander konnten nur wenige Wavelets und drei verschiedene Netzarchitekturen miteinander verglichen werden. Eine genauere Abstimmung der Waveletparameter mit den vorhandenen Audiosignalen könnte allerdings von großer Bedeutung für die weitere Forschung in diesem Bereich sein. Wie in der Arbeit bereits angesprochen, kann auch die Anpassung der Netzarchitekturen einen großen Einfluss auf die Genauigkeit und Generalisierungsfähigkeit der Modelle haben. Aber auch eine ausführliche Untersuchung der adaptiven Implementierung der Bandbreite und der Mittenfrequenz des verwendeten Wavelets im CNN könnte Auskunft über die Anwendungsmöglichkeiten der CWT im maschinellen Lernen geben. Darüber hinaus könnte außerdem der LFCC-Ansatz von [FS21] mit Skalogrammen erweitert werden, um die Generalisierungsfähigkeit in Abhängigkeit von der verwendeten Transformation ausführlicher zu untersuchen.

Es wird deutlich, dass der Deepfake-Erkennungsansatz mit der CWT mit einem primitiven Deep-Learning-Ansatz bereits vielversprechende Ergebnisse liefert und auch über diese Arbeit hinaus Perspektiven eröffnet, die Raum für Weiterentwicklung und Verbesserung geben.

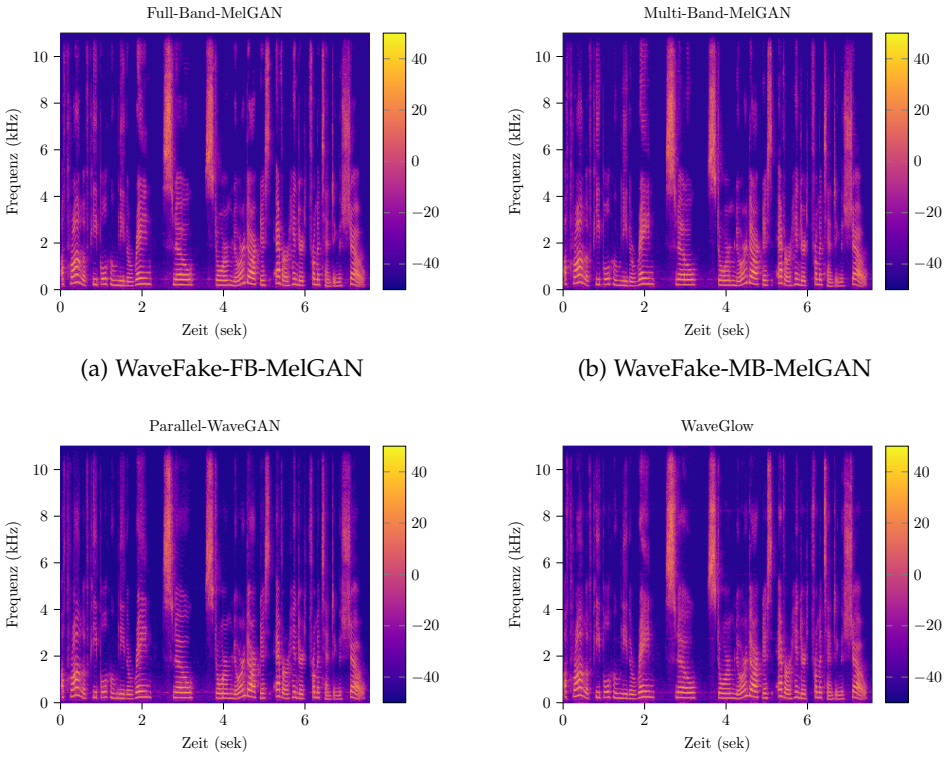
A

ANHANG

A1 GEFENSTERTE VS. KURZZEIT-FOURIERTRANSFORMATION

Da sich in der Literatur keine klare Abgrenzung zwischen STFT und WFT finden lassen konnte und es sich bei einer STFT um ein Produkt aus der Fouriertransformierten und einer Fensterfunktion handelt, werden die beiden Begriffe im folgenden analog zu [FS98] synonym verwendet.

A2 SPEKTROGRAMME DER SUBDATENSÄTZE VON WAVEFAKE



(a) WaveFake-FB-MelGAN

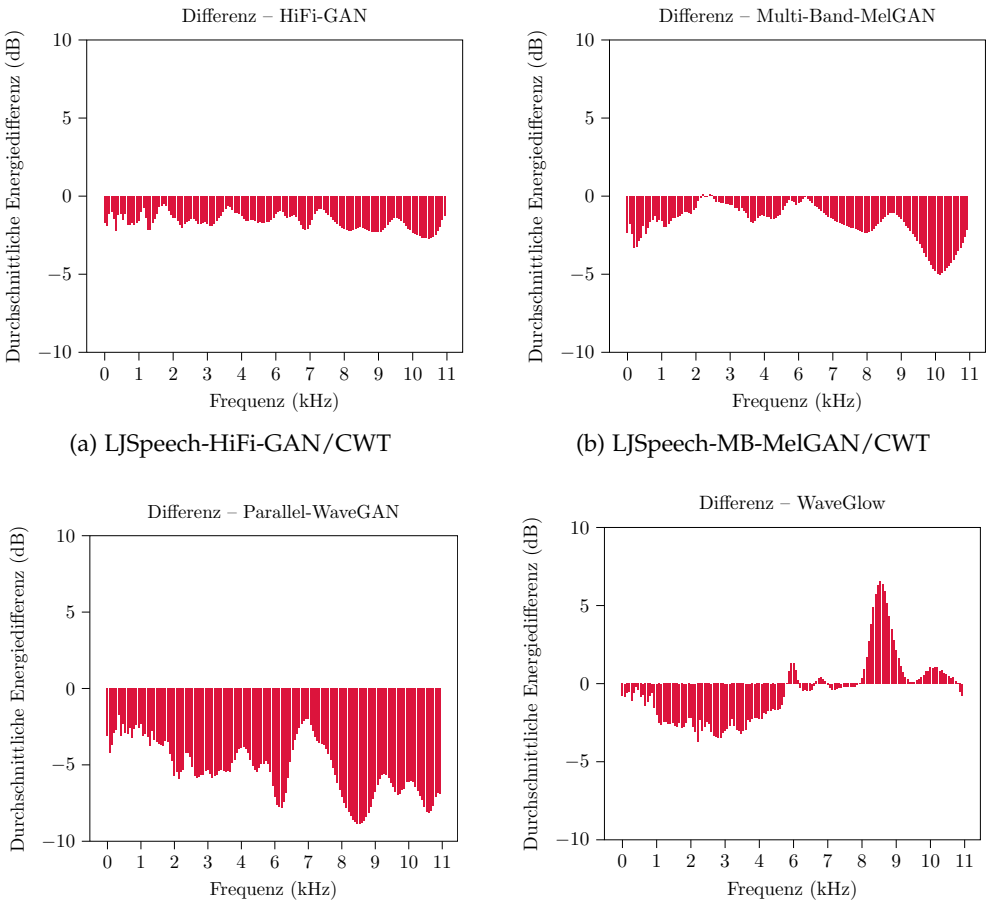
(b) WaveFake-MB-MelGAN

(c) WaveFake-PWG

(d) WaveFake-WaveGlow

Abbildung A.1: Spektrogramme des Audio-Deepfakes *LJSpeech 008-0217* verschiedener Generatoren aus dem WaveFake-Datensatz von [FS21]. Dies ist eine ergänzte Abbildung zu Abb. 2.1 und verwendet dieselben Parameter.

A3 ZEIT-FREQUENZ-ANALYSE



(a) LJSpeech-HiFi-GAN/CWT

(b) LJSpeech-MB-MelGAN/CWT

(c) LJSpeech-PWG/CWT

(d) LJSpeech-WaveGlow/CWT

Abbildung A.2: Differenz der Signalenergien zwischen 13 100 Audiodateien aus LJSpeech und den Audio-Deepfakes von HiFi-GAN, MB-MelGAN, PWG und WaveGlow (WG). Dies ist eine ergänzte Abbildung zu Abb. 2.5 und verwendet dieselben Parameter.

A4 WEITERE ERGEBNISSE DER KREUZVALIDIERUNG DER AUDIOKLASSIFIKATION

	EER auf Testmenge							
Training	Mel	L-Mel	MB-M	FB-M	HiFi	WG	PWG	aEER
Mel	0.000	0.457	0.501	0.513	0.496	0.471	0.531	0.424
L-Mel	0.470	0.001	0.523	0.508	0.500	0.529	0.506	0.434
MB-M	0.462	0.478	0.006	0.386	0.309	0.203	0.207	0.293
FB-M	0.439	0.458	0.215	0.029	0.215	0.281	0.230	0.266
HiFi	0.477	0.451	0.304	0.381	0.012	0.140	0.113	0.268
WGlow	0.435	0.405	0.367	0.503	0.284	0.005	0.190	0.313
PWG	0.458	0.447	0.420	0.465	0.246	0.157	0.003	0.314

Tabelle A.1: Ergebnisse der Kreuzvalidierung der binären Audioquellenklassifikation mit dem 2D-Faltungsnetz *LearnDeepNet* und cmor 4.6-0.87 für alle Subdatensätze von WaveFake. Die trainierten Modelle (linke Spalte) wurden mit den Testmengen aller Subdatensätze kreuzvalidiert. Die Ergebnisse bei gleicher Trainings- und Testmenge sind grau eingefärbt. Diese Ergebnisse entsprechen dem Ansatz in Tab. 2.6 für das LDN mit einem anderen Wavelet.

	EER auf Testmenge							
Training	Mel	L-Mel	MB-M	FB-M	HiFi	WG	PWG	aEER
Mel	0.000	0.254	0.505	0.499	0.508	0.497	0.516	0.397
L-Mel	0.407	0.000	0.499	0.496	0.501	0.503	0.508	0.416
MB-M	0.443	0.453	0.011	0.432	0.429	0.369	0.265	0.343
FB-M	0.302	0.335	0.345	0.058	0.35	0.346	0.285	0.289
HiFi	0.381	0.373	0.425	0.418	0.018	0.270	0.182	0.295
WGlow	0.480	0.457	0.448	0.507	0.437	0.004	0.273	0.372
PWG	0.343	0.355	0.451	0.463	0.376	0.299	0.004	0.327

Tabelle A.2: Ergebnisse der Kreuzvalidierung der binären Audioquellenklassifikation mit dem 2D-Faltungsnetz *LearnDeepNet* und shan 0.01-0.4 für alle Subdatensätze von WaveFake. Die trainierten Modelle (linke Spalte) wurden mit den Testmengen aller Subdatensätze kreuzvalidiert. Die Ergebnisse bei gleicher Trainings- und Testmenge sind grau eingefärbt. Diese Ergebnisse entsprechen dem Ansatz in Tab. 2.6 für das LDN mit einem anderen Wavelet.

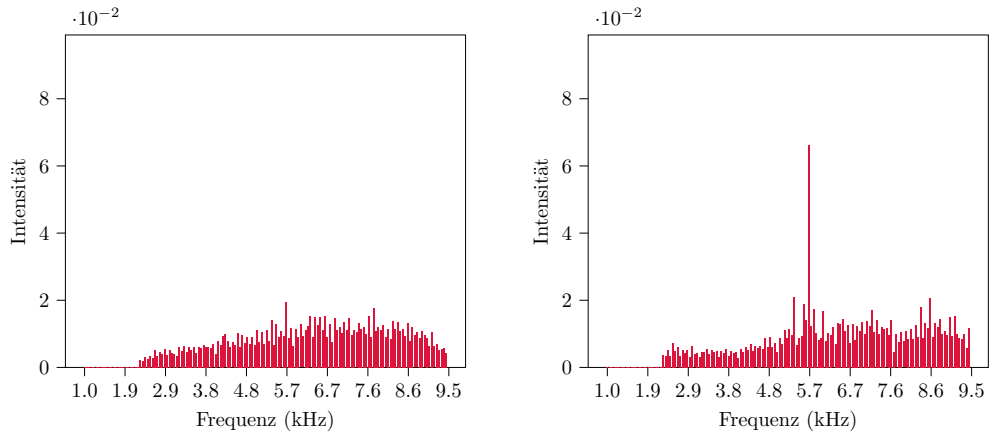
	EER auf Testmenge						
Training	Mel	L-Mel	MB-M	FB-M	HiFi	WG	PWG
Mel	0.001	**0.140**	0.467	0.474	0.492	0.475	0.472
L-Mel	**0.245**	0.000	0.451	0.463	0.488	0.531	0.467
MB-M	0.417	0.368	0.012	0.365	0.403	0.321	**0.262**
FB-M	0.284	0.231	**0.205**	0.064	0.344	0.257	0.269
HiFi	0.453	0.355	0.299	0.358	0.065	**0.224**	0.265
WGlow	0.349	0.358	0.373	0.418	0.366	0.011	**0.276**
PWG	0.424	0.419	0.304	0.431	0.407	**0.250**	0.009

Tabelle A.3: Ergebnisse der Kreuzvalidierung der binären Audioquellenklassifikation mit dem 2D-Faltungsnetz *LearnNet*, cmor 3.3-4.17 und einer verkleinerten Abtastrate von 16 kHz für alle Subdatensätze von WaveFake. Die trainierten Modelle (linke Spalte) wurden mit den Testmengen aller Subdatensätze kreuzvalidiert. Die besten Ergebnisse der dem Modell fremden Testdaten sind zeilenweise fett markiert, Ergebnisse bei gleicher Trainings- und Testmenge sind grau eingefärbt. Diese Ergebnisse entsprechen einer Aufschlüsselung der aEER aus dem Ansatz in Tab. 2.7 für das LN.

	EER auf Testmenge						
Training	Mel	L-Mel	MB-M	FB-M	HiFi	WG	PWG
Mel	0.001	**0.114**	0.469	0.470	0.491	0.471	0.476
L-Mel	**0.251**	0.000	0.450	0.462	0.489	0.536	0.480
MB-M	0.417	0.370	0.012	0.365	0.409	0.322	**0.250**
FB-M	0.258	**0.199**	0.215	0.064	0.349	0.286	0.284
HiFi	0.425	0.347	0.281	0.357	0.064	**0.220**	0.267
WGlow	0.341	0.351	0.366	0.426	0.371	0.014	**0.288**
PWG	0.426	0.408	0.291	0.437	0.412	**0.271**	0.010

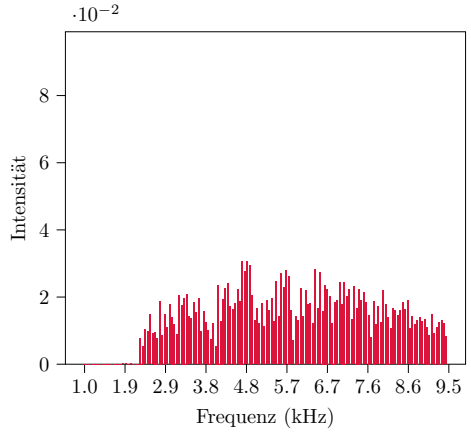
Tabelle A.4: Ergebnisse der Kreuzvalidierung der binären Audioquellenklassifikation mit dem 2D-Faltungsnetz *LearnNet*, cmor 3.3-4.17, lernenden Waveletparametern und einer verkleinerten Abtastrate von 16 kHz für alle Subdatensätze von WaveFake. Die trainierten Modelle (linke Spalte) wurden mit den Testmengen aller Subdatensätze kreuzvalidiert. Die besten Ergebnisse der dem Modell fremden Testdaten sind zeilenweise fett markiert, Ergebnisse bei gleicher Trainings- und Testmenge sind grau eingefärbt. Diese Ergebnisse entsprechen einer Aufschlüsselung der aEER aus dem Ansatz in Tab. 2.7 für das LN, learnWav.

A5 MODELL-INTERPRETATION MIT INTEGRIERTEN GRADIENTEN

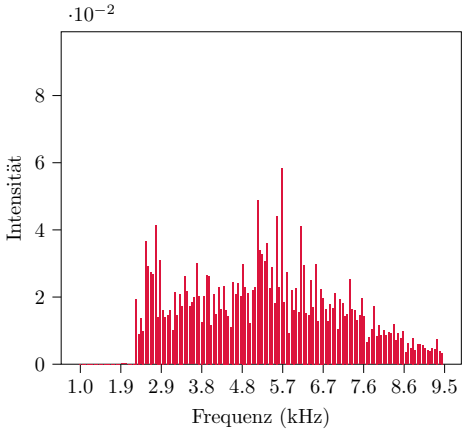


(a) IG des FB-MelGAN-Modells

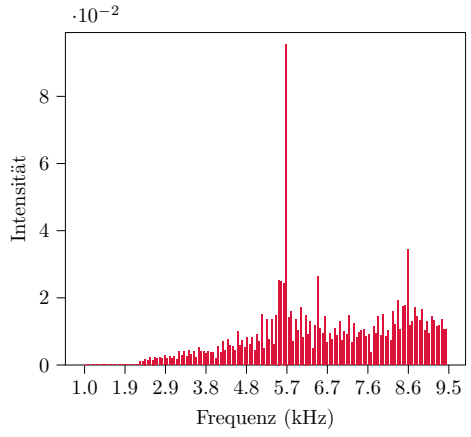
(b) IG des MB-MelGAN-Modells



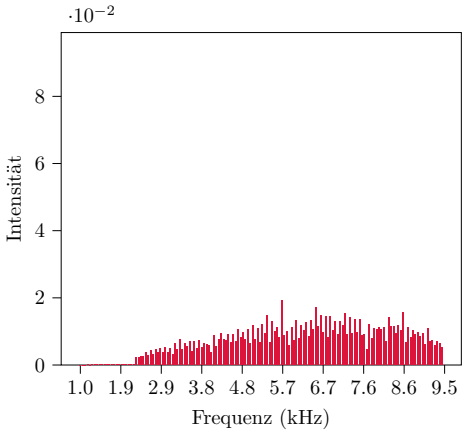
(c) IG des MelGAN-Modells



(d) IG des Large-MelGAN-Modells

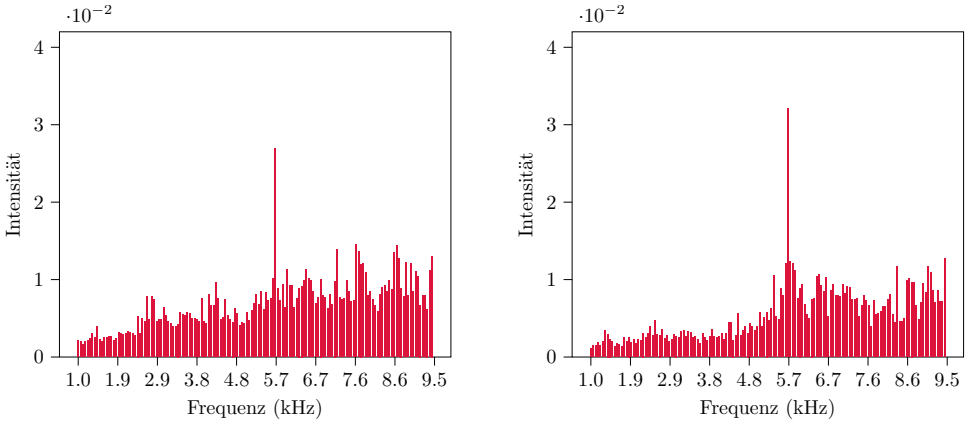


(e) IG des PWG-Modells



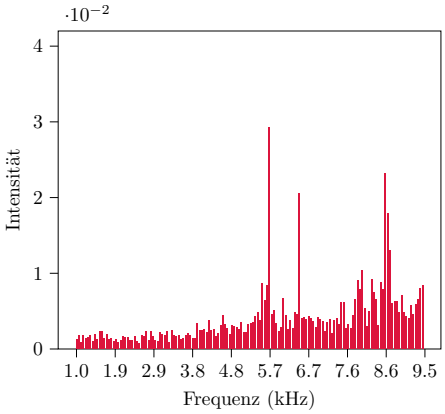
(f) IG des Ansatzes (2) aus 2.4.1

Abbildung A.3: Absolute IG pro Frequenz der trainierten LearnDeepNet-Modelle über je 5056 Fake-Testdaten. Für die Modelle wurden die Werte von je fünf Modellen mit Seeds 0 bis 4 gemittelt. Dies ist eine ergänzende Abbildung zu Abb. 2.6.

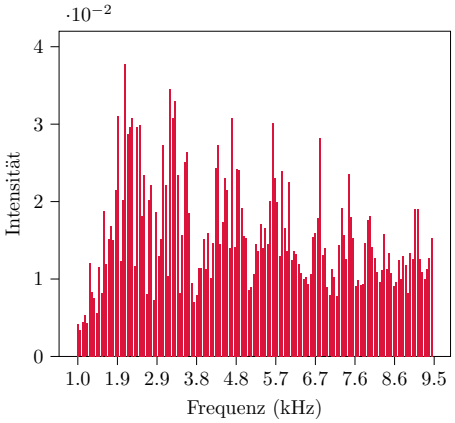


(a) IG des FB-MelGAN-Modells

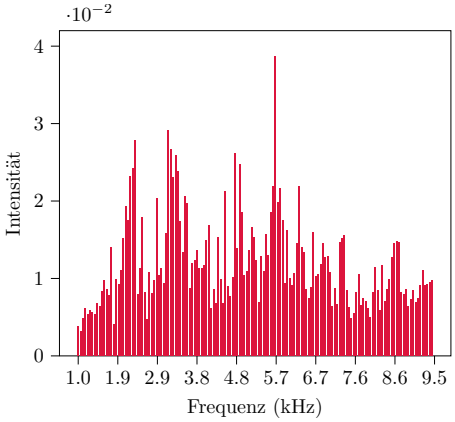
(b) IG des MB-MelGAN-Modells



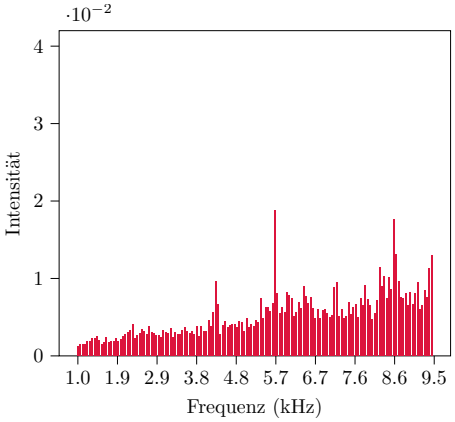
(c) IG des PWG-Modells



(d) IG des MelGAN-Modells



(e) IG des Large-MelGAN-Modells



(f) IG des Ansatzes (2) aus 2.4.1

Abbildung A.4: Absolute IG pro Frequenz der trainierten OneDNet-Modelle über je 5056 Fake-Testdaten. Für die Modelle wurden die Werte von je fünf Modellen mit Seeds 0 bis 4 gemittelt. Dies ist eine ergänzende Abbildung zu Abb. 2.6.

[Ami22] Amini, A. (2022): *MIT Introduction to Deep Learning | 6.S191*. YouTube. URL: https://www.youtube.com/watch?v=7sB052Pz0sQ&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI (Abg.: 7. März 2023).

[BT58] Blackman, R. B., Tukey, J. W. (1958): „The measurement of power spectra from the point of view of communications engineering—Part I". *Bell System Technical Journal*. 37 (1). Wiley Online Library. DOI: 10.1002/j.1538-7305.1958.tb03874.x.

[DV16] Dumoulin, V., Visin, F. (2016): „A guide to convolution arithmetic for deep learning". arXiv. DOI: 10.48550/arxiv.1603.07285.

[Dau92] Daubechies, I. (1992): *Ten Lectures on Wavelets*. 9. Auflage. Society for Industrial und Applied Mathematics (SIAM). DOI: 10.1063/1.4823127.

[FAK22] Fathan, A., Alam, J., Kang, W. (2022): „Multiresolution Decomposition Analysis via Wavelet Transforms for Audio Deepfake Detection". In: *Speech and Computer: 24th International Conference (SPECOM), Gurugram, Indien, November 14–16*. Vol. 13721. S. 188–200. Springer, Cham. DOI: 10.1007/978-3-031-20980-2_17.

[FS21] Frank, J., Schönherr, L. (2021): „WaveFake: A Data Set to Facilitate Audio Deepfake Detection". arXiv. DOI: 10.48550/arxiv.2111.02813.

[FS98] Feichtinger, H. G., Strohmer, T. (Hrsg.) (1998): *Gabor analysis and algorithms: Theory and applications*. 1. Auflage. Birkhäuser, Boston. DOI: 10.1007/978-1-4612-2016-9.

[For12] Forster, O. (2012): *Analysis 3: Integralrechnung im Rn mit Anwendungen*. 3. Auflage. Vieweg und Sohn Verlagsgesellschaft GmbH. DOI: 10.1007/978-3-8348-2374-8.

[GBC16] Goodfellow, I., Bengio, Y., Courville, A. (2016): *Deep Learning*. MIT Press. URL: http://www.deeplearningbook.org (Abg.: 3. März 2023).

[GW23] Gasenzer, K., Wolter, M. (2023): *Erkennung von Audiodeepfakes mithilfe von kontinuierlichen Wavelet-Transformationen*. URL: https://github.com/gan-police/wavelet-audiodeepfake-detection_code.

[Guo+18] Guo, M.-F., Zeng, X.-D., Chen, D.-Y., Yang, N.-C. (2018): „Deep-Learning-Based Earth Fault Detection Using Continuous Wavelet Transform and Convolutional Neural Network in Resonant Grounding Distribution Systems". *IEEE Sensors Journal*. 18 (3). IEEE. DOI: 10.1109/JSEN.2017.2776238.

[Har+20] Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J. (2020): „Array programming with NumPy“. *Nature*. 585 (7825). Nature Publishing Group. DOI: 10.1038/s41586-020-2649-2.

[He+16] He, K., Zhang, X., Ren, S., Sun, J. (2016): „Deep Residual Learning for Image Recognition“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. S. 770–778.

[IJ17] Ito, K., Johnson, L. (2017): *The LJ Speech Dataset*. URL: https://keithito.com/LJ-Speech-Dataset/ (Abg.: 3. März 2023).

[IS15] Ioffe, S., Szegedy, C. (2015): „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *Proceedings of the 32nd International Conference on Machine Learning, Lille, Frankreich, 07–09 Juli*. Vol. 37. S. 448–456. PMLR.

[JMP97] Jordan, D., Miksad, R. W., Powers, E. J. (1997): „Implementation of the continuous wavelet transform for digital time series analysis“. *Review of Scientific Instruments*. 68 (3). American Institute of Physics. DOI: 10.1063/1.1147636.

[KB14] Kingma, D. P., Ba, J. (2014): „Adam: A method for stochastic optimization“. arXiv. DOI: 10.48550/arxiv.1412.6980.

[KSW08] Kiencke, U., Schwarz, M., Weikert, T. (2008): *Signalverarbeitung*. Oldenburg Verlag, München. DOI: 10.1524/9783486593570.

[Kac+22] Kachhi, A., Therattil, A., Patil, A. T., Sailor, H. B., Patil, H. A. (2022): „Significance of Energy Features for Severity Classification of Dysarthria“. In: *Speech and Computer: 24th International Conference (SPECOM), Gurugram, Indien, November 14–16*. Vol. 13721. S. 325–337. DOI: 10.1007/978-3-031-20980-2_28.

[Kai94] Kaiser, G. (1994): *A friendly guide to wavelets*. 1. Auflage. Birkhäuser. DOI: 10.1007/978-0-8176-8111-1.

[Kho+21] Khochare, J., Joshi, C., Yenarkar, B., Suratkar, S., Kazi, F. (2021): „A Deep Learning Framework for Audio Deepfake Detection“. *Arabian Journal for Science and Engineering*. 47 (3). Springer. DOI: 10.1007/s13369-021-06297-w.

[Kri07] Kriesel, D. (2007): *Ein kleiner Überblick über Neuronale Netze*. URL: http://www.dkriesel.com (Abg.: 3. März 2023).

[LMR98] Louis, A. K., Maaß, P., Rieder, A. (1998): *Wavelets*. 2. Auflage. Teubner, Stuttgart. DOI: 10.1007/978-3-322-80136-4.

[LeC+98] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998): „Gradient-based learning applied to document recognition“. *Proceedings of the IEEE*. 86 (11). IEEE. DOI: 10.1109/5.726791.

[Lee+19] Lee, G., Gommers, R., Waselewski, F., Wohlfahrt, K., O'Leary, A. (2019): „PyWavelets: A Python package for wavelet analysis“. *Journal of Open Source Software*. 4 (36). The Open Journal. DOI: 10.21105/joss.01237.

[MLL18] Meintjes, A., Lowe, A., Legget, M. (2018): „Fundamental Heart Sound Classification using the Continuous Wavelet Transform and Convolutional Neural Networks“. In: *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. S. 409–412. DOI: 10.1109/EMBC.2018.8512284.

[Mas+18] Masi, I., Wu, Y., Hassner, T., Natarajan, P. (2018): „Deep Face Recognition: A Survey“. In: *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. S. 471–478. DOI: 10.1109/SIBGRAPI.2018.00067.

[Mer20] Mertins, A. (2020): *Signaltheorie*. 4. Auflage. Springer. DOI: 10.1007/978-3-658-29648-3.

[Mes15] Meschede, D. (Hrsg.) (2015): *Gerthsen Physik*. 25. Auflage. Springer, Berlin. DOI: 10.1007/978-3-662-45977-5.

[NS+97] Najmi, A.-H., Sadowsky, J. (1997): „The continuous wavelet transform and variable resolution time-frequency analysis“. *Johns Hopkins APL Technical Digest*. 18.

[Nie15] Nielsen, M. A. (2015): *Neural Networks and Deep Learning*. Vol. 25. Determination Press San Francisco, CA, USA. URL: http://neuralnetworksanddeeplearning.com/ (Abg.: 3. März 2023).

[OW92] Oppenheim, A. V., Willsky, A. S. (1992): *Signale und Systeme*. 2. Auflage. VCH Verlagsgesellschaft.

[Par12] Parkin, S. (2012): *The rise of the Deepfake and the threat to democracy*. Guardian News und Media. URL: https://www.theguardian.com/technology/ng-interactive/2019/jun/22/the-rise-of-the-deepfake-and-the-threat-to-democracy (Abg.: 3. März 2023).

[Pas+19] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. (2019): „PyTorch: An Imperative Style, High-Performance Deep Learning Library“. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc.

[Pow20] Powers, D. M. W. (2020): „Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation“. arXiv. DOI: 10.48550/arxiv.2010.16061.

[QS15] Qin, J., Sun, P. (2015): „Applications and comparison of continuous wavelet transforms on analysis of A-wave impulse noise“. *Archives of Acoustics*. 40. Committee on Acoustics PAS. DOI: 10.1515/aoa-2015-0050.

[STS17] Sonobe, R., Takamichi, S., Saruwatari, H. (2017): „JSUT corpus: free large-scale Japanese speech corpus for end-to-end speech synthesis“. arXiv. DOI: 10.48550/arxiv.1711.00354.

[STY17] Sundararajan, M., Taly, A., Yan, Q. (2017): „Axiomatic Attribution for Deep Networks“. In: *International conference on machine learning*. Vol. 70. S. 3319–3328.

[Sad96] Sadowsky, J. (1996): „Investigation of signal characteristics using the continuous wavelet transform“. *Johns Hopkins APL Technical Digest*. 17.

[Sri+14] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014): „Dropout: a simple way to prevent neural networks from overfitting“. *The journal of machine learning research*. 15. JMLR.

[TB98] Teolis, A., Benedetto, J. J. (1998): *Computational signal processing with wavelets*. Vol. 182. 1. Auflage. Birkhäuser. DOI: 10.1007/978-3-319-65747-9.

[VA07] Voigt, C., Adamy, J. (2007): *Formelsammlung der Matrizenrechnung*. Oldenbourg Wissenschaftsverlag GmbH. DOI: 10.1524/9783486719802.

[VK07] Vetterli, M., Kovačević, J. (2007): *Wavelets and subband coding*. 2. Auflage. Prentice Hall.

[Wan+17] Wang, L.-H., Zhao, X.-P., Wu, J.-X., Xie, Y.-Y., Zhang, Y.-H. (2017): „Motor fault diagnosis based on short-time Fourier transform and convolutional neural network“. *Chinese Journal of Mechanical Engineering*. 30. Springer. DOI: 10.1007/s10033-017-0190-5.

[Wer18] Werner, D. (2018): *Funktionalanalysis*. 8. Auflage. Springer. DOI: 10.1007/978-3-642-21017-4.

[Wol21] Wolter, M. „Frequency Domain Methods in Recurrent Neural Networks for Sequential Data Processing“. Diss. Rheinische Friedrich-Wilhelms-Universität Bonn, 2021. URL: https://hdl.handle.net/20.500.11811/9245.

[Wol+22] Wolter, M., Blanke, F., Heese, R., Garcke, J. (2022): „Wavelet-packets for deepfake image analysis and detection“. *Machine Learning*. Springer. DOI: 10.1007/s10994-022-06225-5.

[Yam+19] Yamagishi, J., Todisco, M., Sahidullah, M., Delgado, H., Wang, X., Evans, N., Kinnunen, T., Lee, K. A., Vestman, V., Nautsch, A. (2019): „ASVspoof 2019: Automatic Speaker Verification Spoofing and Countermeasures Challenge Evaluation Plan“. *ASV Spoof*.

[Yan+22] Yang, Y.-Y., Hira, M., Ni, Z., al. (2022): „TorchAudio: Building Blocks for Audio and Speech Processing“. In: *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. S. 6982–6986. DOI: 10.1109/ICASSP43922.2022.9747236.

[Zha22] Zhang, T. (2022): „Deepfake generation and detection, a survey“. *Multimedia Tools and Applications*. 81. Springer. DOI: 10.1007/s11042-021-11733-y.