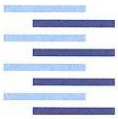

Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorthesis

Sven Ehlers

Störgeräuschreduktion von Mikrofonsignalen mittels DNN-basierter Zeit-Frequenz-Maskierung

*Fakultät Technik und Informatik*
*Department Informations- und Elektrotechnik*

*Faculty of Engineering and Computer Science*
*Department of Information and Electrical Engineering*

Sven Ehlers

# Störgeräuschreduktion von Mikrofonsignalen mittels DNN-basierter Zeit-Frequenz-Maskierung

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Hans Peter Kölzer
Zweitgutachter : Prof. Dr. Klaus Jünemann

Abgegeben am 11. Juni 2020

**Sven Ehlers**

**Thema der Bachelorthesis**

Störgeräuschreduktion von Mikrofonsignalen mittels DNN-basierter Zeit-Frequenz-Maskierung

**Stichworte**

Mehrschichtiges neuronales Netz, Matlab, Störgeräuschreduzierung, Sprache, Maskierung, Maske, Spektrogramm, FFT, STFT

**Kurzzusammenfassung**

Diese Arbeit befasst sich mit der Optimierung und des Vergleichs verschiedener Systeme zur Störgeräuschreduktion bzw. zur Hervorhebung von Sprache in Mikrofonsignalen auf Basis von neuronalen Netzen und Maskierungsverfahren im Zeit-Frequenz-Bereich.

**Sven Ehlers**

**Title of the paper**

Noise reduction of microphone signals by using DNN based Time-Frequency-Masking

**Keywords**

Deep Neural Network, Matlab, Noisereduction, Speech, Masking, Mask, Spektrogramm, FFT, STFT

**Abstract**

This paper is about the optimization and comparison of several systems for noise reduction and speech enhancement in microphone signals based on neural networks and Time-Frequency-Masking.

# Abbildungsverzeichnis

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

In vielen industriellen Anlagen, aber auch Orten des alltäglichen Lebens, sind hohe Umgebungslautstärken aufgrund des Einsatzes von Maschinen und Transportsystemen nicht zu vermeiden. Dennoch ist es besonders in diesen Bereichen häufig erforderlich, dass bestimmte Klänge, insbesondere von Menschen erzeugte Sprache, von elektronischen Systemen erfasst werden können, um bspw. Geräte mittels Sprachbefehlen steuern zu können. Da einfache Mikrofonsysteme jedoch nicht zwischen Umgebungsgeräuschen und menschlicher Sprache unterscheiden können, gestaltet sich eine ausreichend gute bzw. entstörte Erfassung der Sprache als schwierig.

Bisherige Lösungen erreichten eine Hervorhebung von gezielten verbalen Steuereingaben mithilfe von Richtmikrofonen in Form von Mikrofon-Arrays. Jedoch kommen diese Systeme an ihre Grenzen, wenn die Störgeräusche in einem ähnlichen Frequenzbereich sind wie die Sprachbefehle.

Menschen sind in der Lage die Laute anderer Menschen zu erkennen und gegenüber unwichtigen Umgebungsgeräuschen hervorzuheben. Dieser Umstand verleitet zu dem Ansatz, diese Fähigkeit durch ein technisches System nachzuahmen. Da die Strukturen menschlicher Sprache zu komplex für die analytische Beschreibung essentieller Sprachmerkmale ist, liegt es nahe, zur Erkennung von menschlicher Sprache, ein neuronales Netz einzusetzen. Bestehende Systeme setzen neuronale Netze zur Übersetzung von gesprochener Sprache in geschriebene Sprache ein. Für die beschriebene Störgeräuschproblematik ist es jedoch lediglich notwendig, gesprochene Sprache an sich und nicht dessen Wortzusammensetzung zu erkennen. Es existieren Ansätze zur Separierung von akustischen Mischsignalen, bestehend aus den Stimmen mehrerer Personen, in einzelne akustische Signale. Da diese Ansätze sehr zuverlässig dazu in der Lage sind Stimmen voneinander zu unterscheiden, werden einige dieser Ansätze in dieser Arbeit auf die Problematik der Störgeräuschreduktion adaptiert.

## 1.2 Zielsetzung

Durch Variation verschiedener Prozessabläufe wie Datenaufbereitung, Quantisierungs- bzw. Codierungsform der Audiosignale und Training eines neuronalen Netzes sollen verschiedene Gesamtsysteme zur Störgeräuschreduktion verglichen und optimiert werden. Insbesondere soll dabei ein vermeintlich bestes Grundkonzept ermittelt werden.

# 2 Theoretische Grundlagen

## 2.1 Spektrogramm

Ein Spektrogramm stellt den zeitlichen Verlauf eines Signals in drei Dimensionen dar. Die horizontale Dimension ist hierbei die Zeit und die vertikale Dimension die Frequenz. Die Amplitude der Zeit-Frequenz-Komponente wird in der dritten Dimension in Form von Helligkeit oder Farbe repräsentiert. In Abbildung 1 ist beispielhaft das Spektrogramm einer, mit einer Pan-Flöte gespielten, Melodie [16] dargestellt.

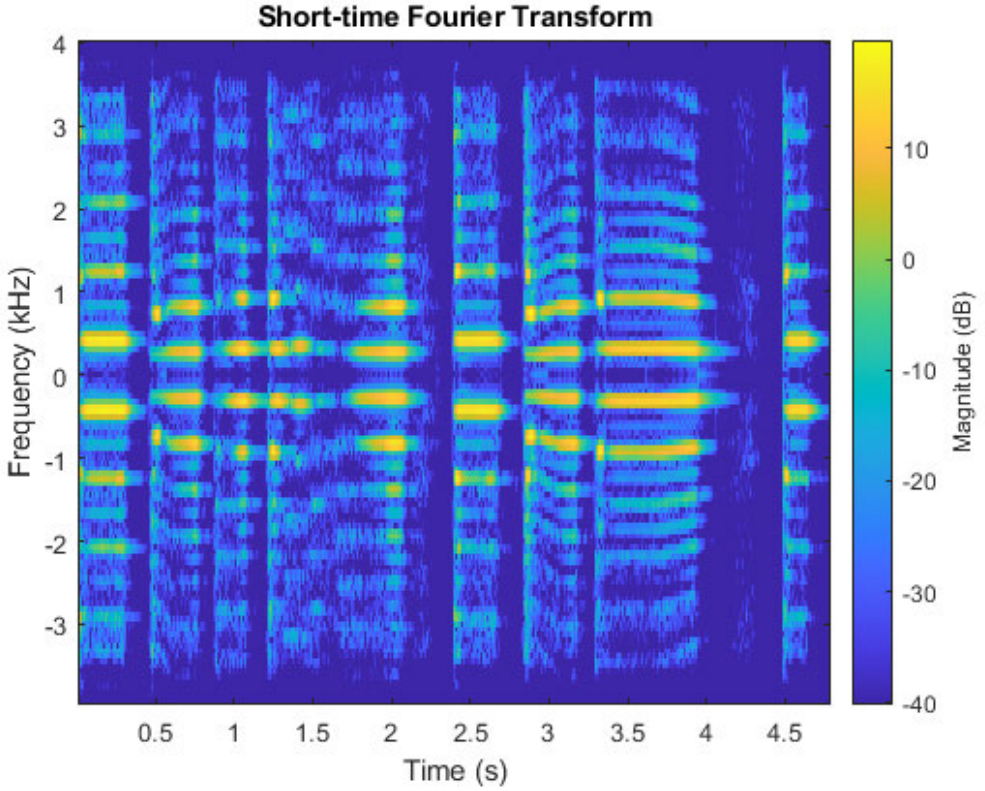


Abbildung 1: Das Spektrogramm einer Audiosequenz mit Pan-Flöte

Spektrogramme werden genutzt, wenn in einer Anwendung nicht nur die spektrale Zusammensetzung eines Gesamtsignals von Bedeutung ist, sondern auch dessen zeitliche Veränderung innerhalb des Gesamtsignals. Das Verfahren zur Berechnung eines Spektrogramms nennt sich Short Time Fourier Transformation (STFT). [6, Seite 300]

### 2.1.1 DFT bzw. FFT

Da Signale in der Praxis nur mit endlichen Abtastraten aufgezeichnet und verarbeitet werden können, kommt zur Berechnung der spektralen Zusammensetzung die diskrete Fouriertransformation (DFT) zum Einsatz. Diese berechnet für ein diskretes und periodisches Zeitsignal das dazugehörige diskrete und periodische Spektrum. Aufgrund der Periodizität des Spektrums wird nur die Grundperiode berechnet.

Vor der Berechnung der DFT wird das Signal mit einer Fensterfunktion gewichtet. Dies beeinflusst zwar das Spektrum, verhindert jedoch, dass durch scharfe Abschnittkanten am Anfang und am Ende des Zeitsignals hohe Frequenzen im Spektralbereich erzeugt werden. In Abbildung 2 wird exemplarisch die sogenannte Hanning-Fensterfunktion zur Fensterung verwendet. Im Rahmen dieser Arbeit wird ausschließlich diese Fensterfunktion verwendet.

Für ein endliches und diskretes Signal $S(t_i)$ mit der Länge $N$ berechnet sich die DFT inklusive vorheriger Fensterung mit $f_{Fenster}(t_i)$ durch

$$A(f_k) = \sum_{i=0}^{N-1} e^{-2\pi j \cdot \frac{ik}{N}} \cdot S(t_i) \cdot f_{Fenster}(t_i) \text{ [6, Seite 114].} \quad (1)$$

Aufgrund von komplex konjugiert gerader Symmetrie gilt $A(f_k) = \overline{A(f_{-k})}$, sodass die DFT $\frac{N}{2}$ nicht redundante Frequenz-Amplituden enthält [6, Seite 116]. Ist $F_s$ die Abtastrate des Signals $S(t_i)$, so folgt für die dargestellten Frequenzen $f_k = k * \frac{Fs}{N}$ und das Auflösungsvermögen $\Delta f = \frac{Fs}{N}$.
In Abbildung 2 ist veranschaulicht dargestellt wie die DFT inklusive der Fensterung durchgeführt wird und welche Symmetrien im Spektrum entstehen. Um Aliasing-Effekten vorzubeugen, muss das Nyquist-Kriterium erfüllt sein, welches besagt, dass die höchste Frequenz im Signal kleiner als die Hälfte der Abtastrate sein muss [6, Seite 98].

Unter der Bedingung, dass die Fensterbreite eine Zweierpotenz beträgt, ergibt sich die Möglichkeit einer besonders effizienten Methode zur Berechnung der DFT. Diese effiziente Methode wird Fast Fouriertransformation (FFT) genannt [6, Seite 172].

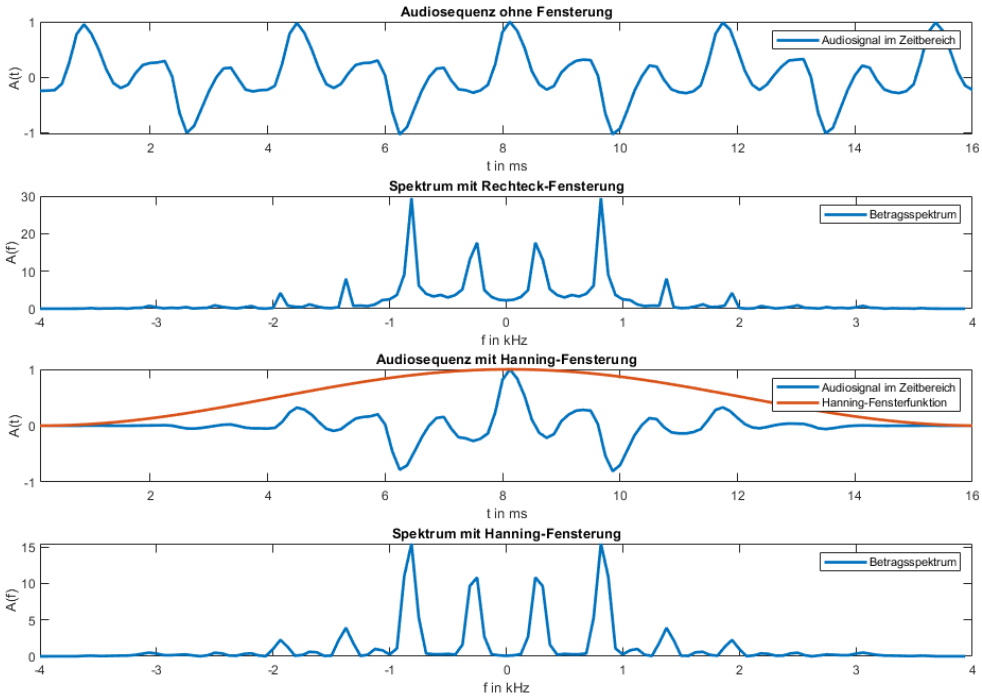


Abbildung 2: Die Diskrete Fouriertransformation und Fensterung eines Audiosignals

### 2.1.2 STFT

Die STFT berechnet für äquidistante Zeitpunkte die spektrale Zusammensetzung der, den Zeitpunkt umgebenden, Region. Dabei können sich die Regionen überschneiden.
Die Breite der Regionen bzw. die Fensterbreite der DFT, die für diese Regionen ausgeführt wird, sollte so gewählt werden, dass eine der Anwendung genügende Bandbreite mit ausreichend hoher Auflösung generiert werden kann [6, Kapitel 8].

## 2.2 Deep Neural Network (DNN)

Ein DNN oder allgemeiner ein neuronales Netz ist ein mathematisches und technisch realisierbares Konstrukt, welches derart konfiguriert werden kann, dass in einer Sammlung von Informationen bestimmte Charakteristika identifiziert werden können. Die Charakteristika werden nicht unmittelbar vom Programmierer abstrahiert und implementiert, sondern vom DNN selber im einem Trainingsprozess erlernt. Besonders nützlich sind DNNs bei Anwendungen, in denen die Charakteristika nicht offensichtlich sind bzw. analytisch nicht bestimmt werden können [5, Kapitel 2].

### 2.2.1 Komponenten und Aufbau

Die Grundkomponenten, aus denen ein neuronales Netz besteht, sind sogenannte Perzeptronen bzw. Neuronen und die Verbindungen zwischen diesen. Mithilfe der Graphentheorie lässt sich ein neuronales Netz als gerichteter Graph mit seinen Neuronen als Knoten und den Verbindungen als Kanten beschreiben. Dabei besteht jedes Neuron aus einer Eingangsfunktion, einer Ausgangs- bzw. Aktivierungsfunktion und einem reellen Bias-Wert. Die Eingangsfunktion vollzieht die Summation der Eingangswerte und dem Bias-Wert des Neurons. Die Verbindungen zwischen den Neuronen besitzen jeweils eine individuelle Gewichtung.
In dieser Arbeit liegt der Fokus auf sogenannten ″Feed Forward Neural Networks″. Ein solches neuronales Netz setzt sich aus einer Eingangsebene und gerichtet verbundenen Ebenen aus Neuronen zusammen. Die Knoten der Eingangsebene sind jeweils mit jedem Neuron der Folgeebene verbunden. Ein solches Netzwerk ist in Abbildung 3 im Allgemeinen dargestellt [5, Kapitel 4.1].

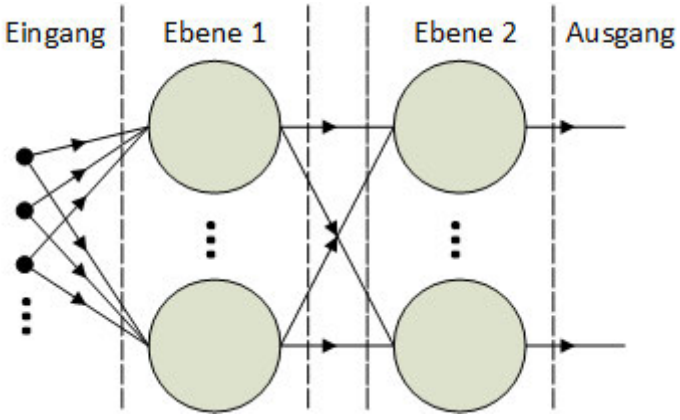


Abbildung 3: Schaubild eines Feed Forward Neural Network mit 2 Ebenen

Besitzt das neuronale Netz mehr als eine Ebene mit Neuronen, bezeichnet man es als ″Deep Neural Network″(DNN). Die inneren Ebenen sind in diesem Fall sogenannte ″Hidden Layer″.

### 2.2.2 Anwendung

Mithilfe von Abbildung 4 wird die Funktionsweise des DNN im Anwendungsfall beschrieben. Dazu wird ein DNN mit $N_E \in \mathbb{N} > 1$ Ebenen und $N_{K_i} \in \mathbb{N} > 0$ Knoten in Ebene $i \in \mathbb{N}$ mit $0 \leq i < N_E$ herangezogen. Innerhalb einer Ebene $i$ werden die Knoten mit $K_{ij}$ und $0 < j \in \mathbb{N} < N_{K_i}$ durchnummeriert. Das Gewicht der Verbindung zwischen Knoten $K_{ij}$ und Knoten $K_{(i+1)k}$ ist gegeben durch $w_{ijk} \in \mathbb{R}$ mit $0 < k \in \mathbb{N} < N_{K_i}$. Die Ebene 0 ist die Eingangsebene und besitzt als einzige keine Neuronen, sondern nur Einspeisepunkte. Die Knoten aller anderen Ebenen sind Neuronen mit Ein- und Ausgangsfunktion. Die Ausgänge der Neuronen $K_{(N_E-1)}$ sind zugleich die Ausgänge des DNN [5, Kapitel 4.2].

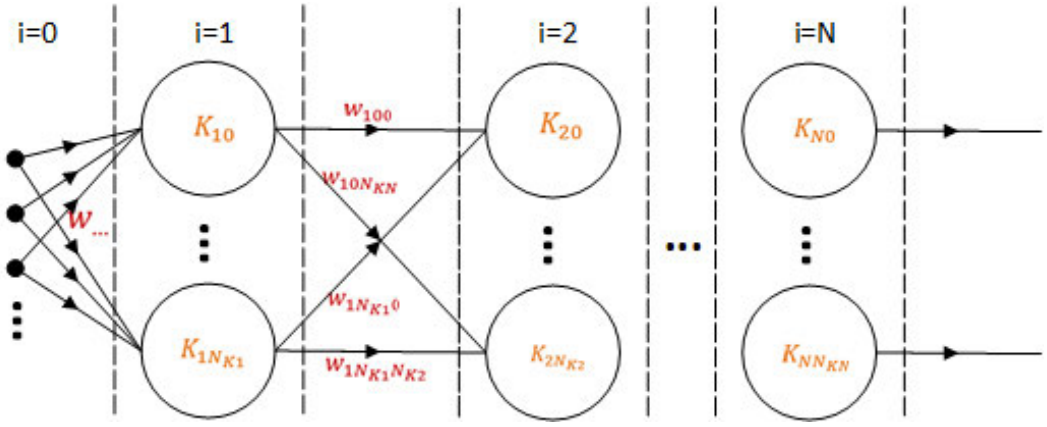


Abbildung 4: Schaubild - Anwendung eines DNN

Ein Merkmalsvektor $M \in \mathbb{R}$ wird zunächst in die Eingangsebene des DNN eingespeist. Dabei wird jedem Knoten $K_{0j}$ der Wert $M_j$ als Ausgangswert zugeführt.
Für $i = 0$ werden die Werte aller Knoten $K_{ij}$ über die Verbindungen an alle Knoten $K_{(i+1)k}$ weitergeleitet und dabei mit dem Gewicht $w_{ijk}$ der jeweiligen Verbindung multipliziert. In jedem Neuron $K_{(i+1)k}$ werden die Eingangswerte und der Bias-Wert aufsummiert und anschließend an die Aktivierungsfunktion $f_{akt}$ weitergegeben. Die Aktivierungsfunktion ist eine Abbildung mit $f_{akt} : \mathbb{R} \mapsto \mathbb{R}$ und kann für jeden Knoten individuell gewählt werden. Der Ausgangswert der Aktivierungsfunktion ist dann gleichzeitig der Ausgangswert des Neurons $K_{(i+1)j}$. Dieser Vorgang wird für $i = 1...(N_E - 1)$ nacheinander ebenso durchgeführt, sodass im letzten Schritt für $i = N_E - 1$ am Ausgang der letzten Ebene der Ergebnisvektor $A \in \mathbb{R}^{N_{K(N_E-1)}}$ erzeugt wird [5, Kapitel 4.2].
Der Ausgangswert eines Neurons $K_{(i+1)j}$ lässt sich also beschreiben durch

$$K_{(i+1)k} = f_{akt}(b + \sum_{j=1}^{N_{K_i}} w_{ijk} \cdot K_{ij}) \tag{2}$$

Die gezielte Konfiguration der Verbindungsgewichte $w_{ijk}$ ermöglicht die Nutzung des DNN als System zur Mustererkennung bzw. Musterverarbeitung. Dabei führen spezifische Informations-Muster im Merkmalsvektor $M$ zu einer spezifischen Ausgabe im Ergebnisvektor $A$. Der Prozess der Approximation der optimalen Konfiguration der Verbindungsgewichte wird als Training bezeichnet.

### 2.2.3 Trainingsprozess

Die manuelle Konfiguration der Verbindungsgewichte ist theoretisch möglich, jedoch sehr langwierig und somit unpraktikabel. Stattdessen kommt eine Datenmenge zum Einsatz, welche Merkmalsvektoren und die dazugehörigen erwarteten Ergebnisvektoren enthalten. Anhand dieser Daten soll das DNN darauf trainiert werden, die gewünschte Funktionalität zu erfüllen. Die Daten werden zunächst in sogenannte Trainings- und Validierungsdaten separiert. Üblicherweise bestehen so die gesamten Daten aus ca. 85% Trainingsdaten und 15% Validierungsdaten. Ein Trainingsschritt besteht darin, einen Merkmalsvektor aus den Trainingsdaten in das neuronale Netz zu führen und den erzeugten Ergebnisvektor mit dem erwarteten Ergebnisvektor zu vergleichen. Anschließend werden die Verbindungsgewichte des DNN so manipuliert, dass die Abweichung des erzeugten Ergebnisvektors zu dem erwarteten Ergebnisvektors geringer wird. In einer sogenannten Epoche wird dieser Trainingsschritt für jede Merkmals- und Ergebnisvektor Kombination nacheinander durchgeführt. Ein vollständiges Training kann über mehrere Epochen stattfinden. Dabei werden die bereits verwendeten Trainingsdaten erneut verwendet, können jedoch in ihrer Reihenfolge durchmischt werden. In regelmäßigen Abständen wird der Trainingserfolg mithilfe der Validierungsdaten in einem Validierungsschritt überprüft. Zu diesem Zweck werden die Merkmalsvektoren der Validierungsdaten in das DNN eingeführt und die erzeugten Ergebnisvektoren mit den erwarteten Ergebnisvektoren verglichen. Sollte sich die Abweichung bzw. der Fehler zwischen erzeugten und erwarteten Ergebnisvektoren über mehrere Validierungsschritte hinweg erhöhen, sollte das Training beendet werden, und die Konfiguration der Verbindungsgewichte im Validierungsschritt der geringsten Abweichung als Ergebnis für die approximierte optimale Konfiguration des DNN genommen werden. In dem Fall, dass die Fehler in den Validierungsschritten steigen und in den Trainingsschritten weiter sinken, wird von sogenanntem ″Overfitting″ gesprochen. Beim Auftreten von ″Overfitting″wird das DNN zu speziell auf die Trainingsdaten trainiert und verliert an genereller Anwendbarkeit, wie an der Erhöhung der Validierungsfehler zu erkennen ist.
Das Verfahren zur quantitativen Berechnung der Verbindungsgewichte in den Trainingsschritten unterscheidet sich je nach verwendeter Trainingsmethode. Die für diese Arbeit relevante Trainingsmethode wird im weiteren Verlauf dieser Arbeit vorgestellt.

### 2.2.4 Trainingsmethoden

Die Trainingsmethode bestimmt quantitativ inwiefern im Trainingsprozess der Unterschied zwischen erwartetem Ergebnisvektor und berechnetem Ergebnisvektor zu einer Änderung der Verbindungsgewichte führt. Für die Quantifizierung des Unterschieds wird als Fehlerfunktion üblicherweise die Summe der quadratischen Fehler aller Ausgangsneuronen genommen. Das heißt für einen Trainingsschritt ist der Fehler bzw. die Abweichung $E$, mit dem berechneten Ergebnisvektor $A_{ber}$ und dem erwarteten Ergebnisvektor $A_{ideal}$, gegeben durch

$$E = \sum_i (A_{ber,i} - A_{ideal,i})^2 \, [5, Seite 41]. \tag{3}$$

In mehrschichtigen neuronalen Netzen, also DNNs, ist eine Möglichkeit der Berechnung der verbesserten Gewichte die sogenannte Fehler-Rückpropagation. Dabei wird der Fehler an der Ausgabeebene sukzessive auf die Verbindungsgewichte versteckter Ebenen zurückgeführt, sodass die Ableitung des Ausgabefehlers nach einem Verbindungsgewicht nicht nur für die Ausgangsebene, sondern auch für versteckte Ebenen bestimmt werden kann. Mithilfe dieser Ableitung bzw. der Steigung des Ausgabefehlers an der Stelle des momentanen Verbindungsgewichtes kann das Gewicht so geändert werden, dass der Fehler minimiert wird.
Bei diesem Verfahren gilt z.B. nach einem Trainingsschritt für die Änderung eines Verbindungsgewichtes $\Delta w_{ijk}$ zwischen Neuron $j$ der Ebene $i$ und Neuron $k$ der Folgeebene $i+1$,

$$\Delta w_{ijk} = -\frac{\eta}{2} \cdot \frac{\partial E}{\partial w_{ijk}} [5, Seite 64], \tag{4}$$

wobei $\eta$ ein Maß für die Stärke der Gewichtsveränderung in jedem Trainingsschritt ist und typischerweise z.B. 0.2 beträgt.

In dieser Arbeit kommt eine Trainingsmethode zum Einsatz, die ebenfalls auf der Ableitung des Fehlers nach den Verbindungsgewichten und der Fehler-Rückpropagation basiert, sich jedoch von Gleichung 4 unterscheidet. Die zum Einsatz kommende Methode heißt ″Adaptive Moment Estimation″(Adam) und berechnet die Änderung des Verbindungsgewichtes nach Trainingsschritt $l$ mit

$$\Delta w_{ijk}^{(l)} = -\frac{\eta \cdot m_l}{\sqrt{v_l} + \epsilon} \tag{5}$$

und

$$\begin{aligned} m_l &= \beta_1 m_{l-1} + (1-\beta_1) \cdot \frac{\partial E}{\partial w_{ijk}}^{(l)} \\ v_l &= \beta_2 v_{l-1} + (1-\beta_2) \cdot \left( \left( \frac{\partial E}{\partial w_{ijk}} \right)^2 \right)^{(l)} [7]. \end{aligned} \tag{6}$$

Dabei sind $\eta$, $\beta_1$, $\beta_2$ und $\epsilon$ einstellbare Parameter. Aufgrund der rekursiven Berechnungsformel der Gewichtsänderung besitzt diese Trainingsmethode eine Erinnerung an vorhergehende Trainingsschritte und kann als gleitender Mittelwert $m_l$ der Fehlerableitung und als gleitender Mittelwert $v_l$ des Quadrats der Fehlerableitung aufgefasst werden. Durch Platzierung von $v_l$ im Nenner der Gewichtsveränderung, wird die Gewichtsveränderung kleiner, wenn diese zum Großteil aus Rauschen in der Fehlerableitung besteht.

Zusätzlich kann das sogenannte Batch-Training angewandt werden, bei dem bei gleichbleibenden Verbindungsgewichten eine Menge an Trainingsschritten durchgeführt wird und erst danach die Gewichtsveränderungen aller Trainingsschritte durchgeführt werden. Dieses Verfahren bewirkt, dass statistische Fluktuationen geglättet werden und das Training insgesamt stabilisiert wird.

### 2.2.5 Aktivierungsfunktionen

Für die Wahl der Aktivierungsfunktion haben sich verschiedene Funktionen etabliert. Behandelt werden im Folgenden jene, welche in dieser Arbeit Anwendung finden. Dazu gehören die Sigmoid-, die ReLU- und die ELU-Aktivierungsfunktion.
Im Folgenden wird der Bias-Wert separat zum restlichen Funktionsargument aufgeführt, um zu verdeutlichen, dass der Bias-Wert als horizontale Verschiebung der Aktivierungsfunktion aufgefasst werden kann.

#### Sigmoid

Die Sigmoid-Funktion wird beschrieben durch

$$f(x+b) = \frac{1}{1+e^{-(x+b)}} \quad \text{[5, Seite 45, Logistische Funktion].} \tag{7}$$

Sie zeichnet sich besonders dadurch aus, dass sie stetig differenzierbar ist und somit die Ableitung des Ausgabefehlers nach den Verbindungsgewichten leichter bestimmt werden kann.

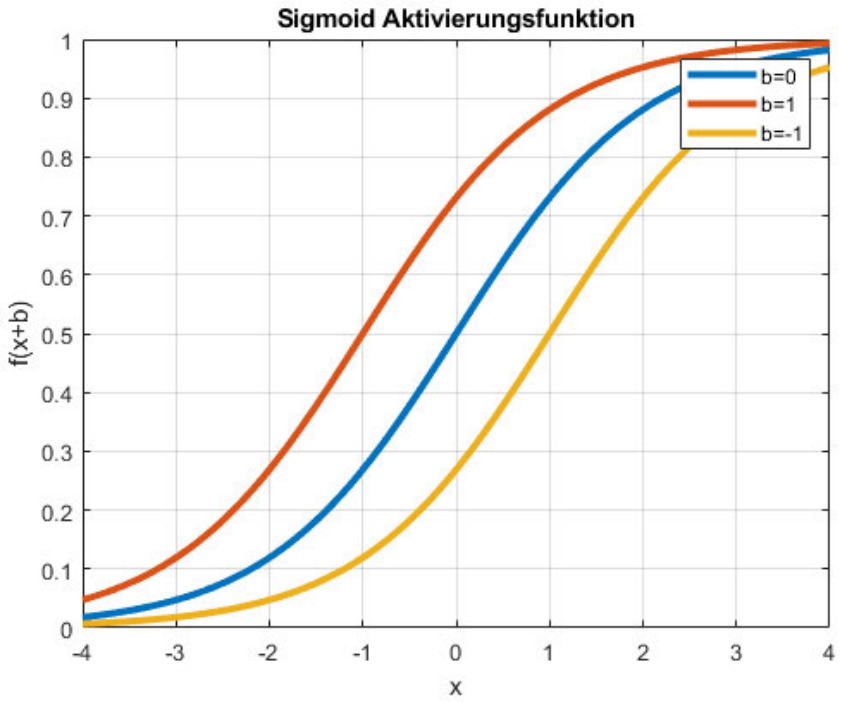


Abbildung 5: Graph der Sigmoid Aktivierungsfunktion

**ReLU**

Die ReLU-Funktion wird beschrieben durch

$$f(x+b) = \begin{cases} 0 & |x < -b \\ x+b & |x \geq -b \end{cases} [8, reluLayer]. \tag{8}$$

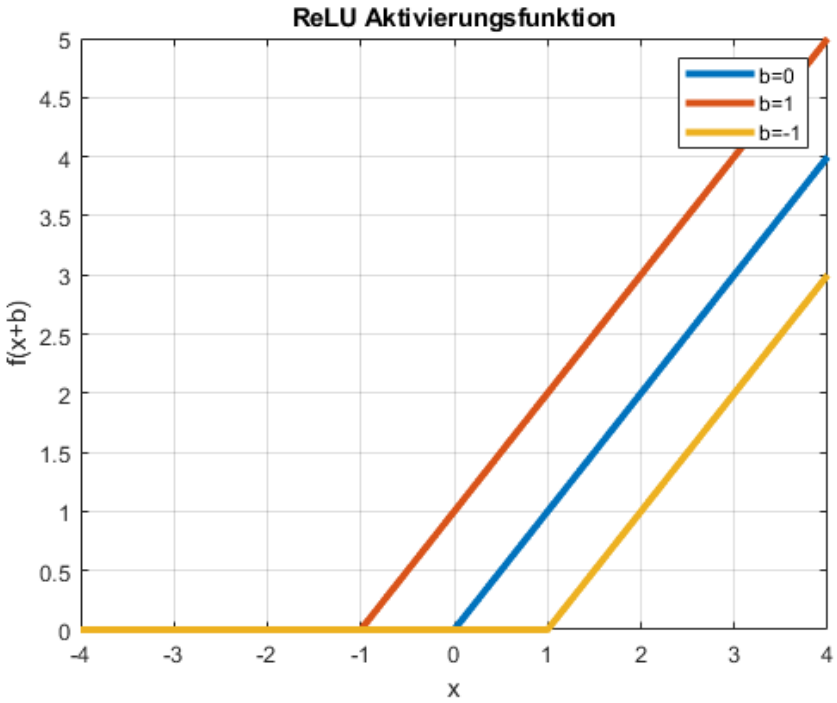


Abbildung 6: Graph der ReLU Aktivierungsfunktion

**ELU**

Die ELU-Funktion wird beschrieben durch

$$f(x+b) = \begin{cases} \alpha \cdot (e^{(x+b)} - 1) & |x < -b \\ x + b & |x \geq -b \end{cases} [8, eluLayer]. \tag{9}$$

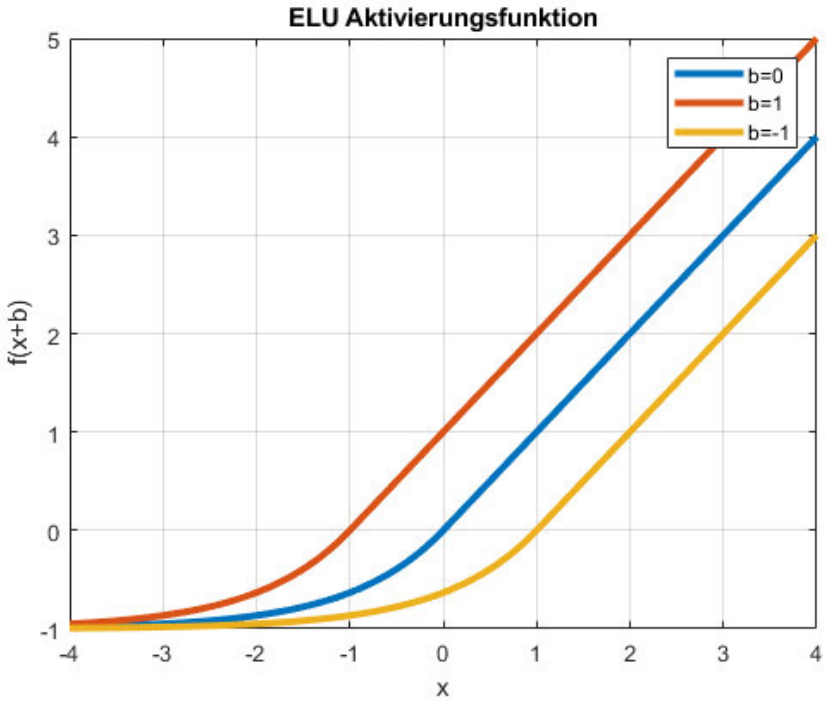


Abbildung 7: Graph der ELU Aktivierungsfunktion

# 3 Stand der Technik

## 3.1 Source Separation von Andrew J.R. Simpson

Die Grundlage für diese Arbeit stellt eine Veröffentlichung von Andrew J.R. Simpson mit dem Titel ″Probabilistic Binary-Mask Cocktail-Party Source Separation in a Convolutional Deep Neural Network″. Die Bezeichnung ″Convolutional″im Titel ist insofern irreführend, dass es sich in der Veröffentlichung nicht um ein ″Convolutional″ DNN handelt, sondern um ein gewöhnliches DNN.
Die Veröffentlichung beschreibt ein System zur Separierung der gesprochenen Texte zweier Stimmen. Dabei soll mithilfe eines DNN ein, aus zwei Stimmen bestehendes, Mischsignal in zwei separate Signale getrennt werden, wobei die getrennten Signale jeweils nur eine Stimme enthalten sollen.

Das Grundkonzept des Systems ist, dass das im System enthaltene DNN die spezifischen Strukturen einer Stimme erlernen soll, sodass anhand der DNN-Ausgabe eine Extraktion dieser Strukturen möglich sein soll. Das DNN wird darauf trainiert, anhand des Spektogramms des Mischsignals, eine sogenannte ″Binary Mask″ vorherzusagen, welche dazu genutzt wird ein Signal mit einer einzelnen Stimme aus dem Mischsignal zu extrahieren.

Für das Training des DNN stehen zwei Signale mit jeweils 2 Minuten Länge und einer Abtastrate von 4 kHz zur Verfügung. Beide Signale bestehen aus gesprochenem Text, wobei eines mit einer weiblichen und das andere mit einer männlichen Stimme gesprochen wird. Außerdem wird ein drittes Signal durch Addition der beiden Einzelsignale erzeugt. Von allen drei Signalen wird ein Spektrogramm mittels der STFT berechnet, wobei für die Fensterbreite 128 Schritte, für die Schrittweite 1 Schritt und für die Fensterfunktion die Hanning-Funktion gewählt werden. Aufgrund ungerader komplex konjugierter Symmetrie des Spektrums wird das Spektrogramm von 128 Punkten in der Frequenz-Dimension auf 65 nicht redundante Punkte reduziert.
Die Beträge der reduzierten Spektrogramme der Einzelsignale werden genutzt, um die binäre Maske zu berechnen. Dafür werden die Amplituden der reduzierten Spektrogramme verglichen. Besitzt die männliche Stimme in einem Zeit-Frequenz Punkt eine höhere Amplitude als die weibliche Stimme, so erhält die binäre Maske an dieser Stelle eine 1. Ist die Amplitude der männlichen Stimme geringer als die der weiblichen Stimme, erhält die binäre Maske an der Stelle eine 0. Für den Maskenwert der männlichen Stimme $M^m_{ft}$ gilt mit den Spekrogrammen $S^m_{ft}$ der männlichen Stimme und $S^w_{ft}$ der weiblichen Stimme also die folgende Berechnungsvorschrift.

$$M^m_{ft} = \begin{cases} 1 & | \ |S^m_{ft}| > |S^w_{ft}| \\ 0 & | \ |S^m_{ft}| \leq |S^w_{ft}|. \end{cases} \tag{10}$$

Für den Maskenwert der weiblichen Stimme $M_{tf}^{w}$ gilt entsprechend

$$M_{ft}^{w} = \begin{cases} 1 & | \ |S_{ft}^{w}| > |S_{ft}^{m}| \\ 0 & | \ |S_{ft}^{w}| \leq |S_{ft}^{m}|. \end{cases} \tag{11}$$

Für das Training des DNN wird das reduzierte Spektrogramm des Mischsignals $S_{Misch}$ normiert, indem der Mittelwert auf 0 und die Standardabweichung auf 1 erzwungen werden. Anschließend werden das normierte Spektrogramm des Mischsignals und die Matrix der Maske $M$ in Abschnitte mit einer Länge von 20 Zeitschritten eingeteilt. Die Abschnitte haben dabei eine Überlappung von 10 Zeitschritten. Die Abschnitte des normierten Spektrogramms und der binären Maske, mit jeweils $65 \cdot 20 = 1300$ Zahlenwerten, werden für das Training als Merkmalsvektoren und erwartete Ergebnisvektoren aufgefasst.
Die Struktur des DNN wird die Randbedingungen ausgelegt und erhält eine Eingangsebene mit 1300 Eingängen. Zusätzlich kommen dahinter zwei Ebenen mit 1300 Neuronen mit jeweils der Sigmoid-Funktion als Aktivierungsfunktion. Der Bias der inneren Neuronen-Ebene wird in der Veröffentlichung nicht explizit erwähnt, ist jedoch aufgrund eines referenzierten Papers als 6 anzunehmen [2]. Der Bias der Ausgangsebene wird auf 0 gesetzt.

In der Anwendung wird für ein Mischsignal, genau wie für das Training, der Betrag des Spektrogramms reduziert und normiert. Das Spektrogramm wird anschließend in das trainierte DNN überführt. Die resultierende binäre Maske wird mit dem ursprünglichen komplexen Spektrogramm multipliziert und dann wieder in ein Zeitsignal überführt. Dieses Signal soll im Idealfall nur noch eine Stimme enthalten.

[1]

# 4 Konzept

Im Konzept wird das übergreifende Grundkonzept aller in dieser Arbeit untersuchten Systeme beschrieben.

## 4.1 Systemstruktur

Die allgemeine Struktur des Systems zur Störgeräuschreduzierung in Abbildung 8 setzt sich aus den Komponenten zusammen, welche alle im Rahmen dieser Arbeit getesteten Systeme gemeinsam haben. Das Blockschaltbild des Systems setzt sich aus zwei hauptsächlichen Signalpfaden zusammen. Ein Pfad (Schwarz) beschreibt das System für den Anwendungsfall, der andere Pfad (Rot) beschreibt die Systemergänzung für den Trainingsprozess.

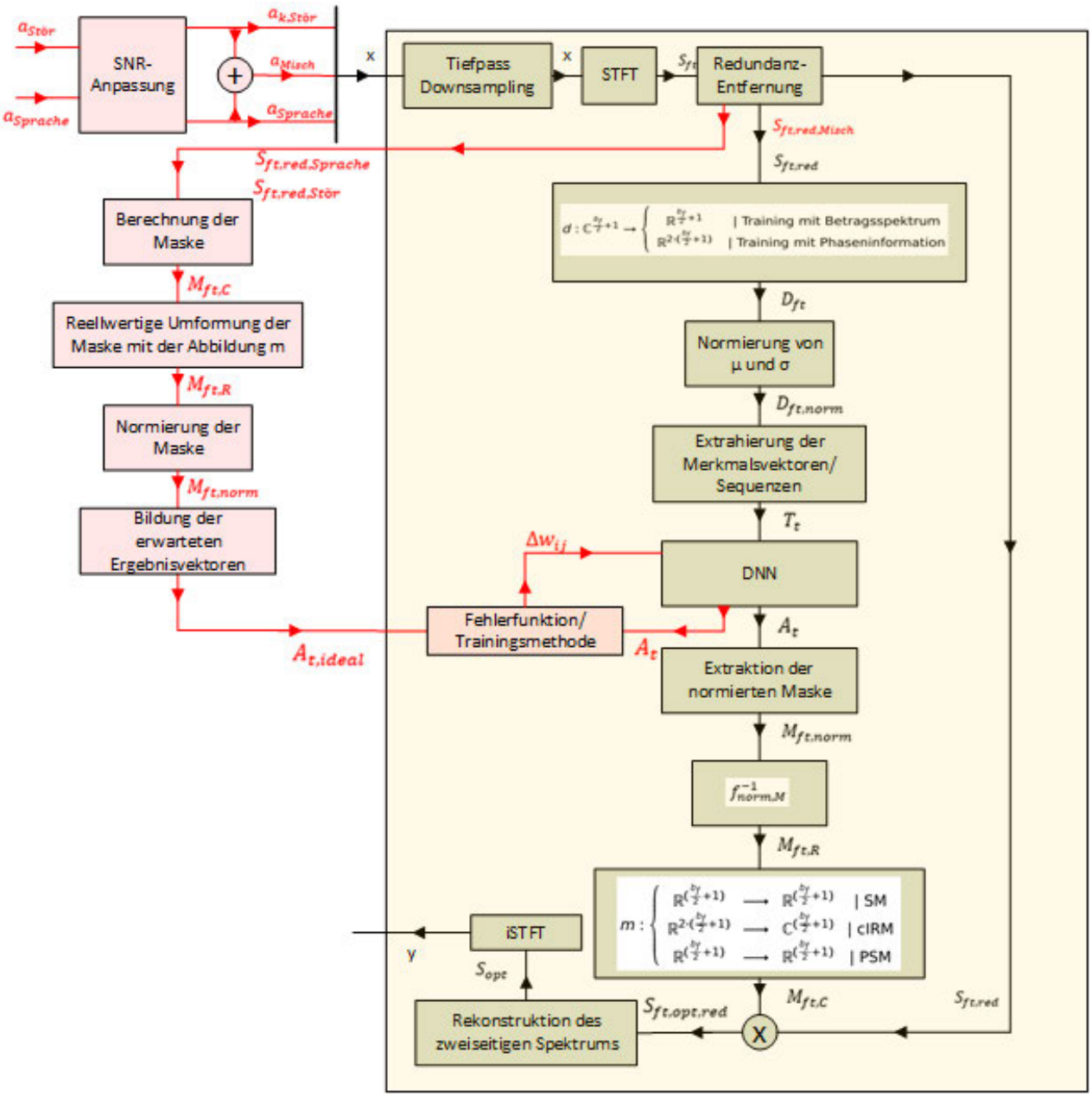


Abbildung 8: Allgemeines Blockschaltbild des Systems zur Störgeräuschreduzierung

Es wird zunächst auf die Systempfade des Grundsystems im Anwendungsfall eingegangen. Anschließend werden die für den Trainingsprozess nötigen zusätzlichen Pfade beschrieben.

**Systempfade des Grundsystems**

Das System besitzt einen Eingang für ein akustisches Signal $x$ der Länge $N$. Dieses Signal sollte zur Einhaltung des Nyquist-Kriteriums keine Frequenzen enthalten, die höher als die Hälfte der Abtastrate sind. Außerdem muss die Abtastrate größer oder gleich $8kHz$ und ein ganzzahliges Vielfaches von $8kHz$ sein.
Ein Signal mit einer höheren Abtastrate als 8kHz wird mit einem Tiefpass mit einer Grenzfrequenz von $4kHz$ gefiltert und anschließend mit $8kHz$ abgetastet.
Von dem Signal $x$ wird anschließend das Spektrogramm als Matrix mit den Matrixelementen $S_{ft} \in \mathbb{C}$ mit FFT-Breite $b_f$ und Schrittweite $s_t$ unter der Verwendung der Hanning-Fensterfunktion erzeugt. Es gilt also für die Indices der Spektrogramm-Matrix $0 < f < b_f$ und $0 < t < \frac{N}{s_t}$. Von $S_{ft}$ werden für alle Zeitspalten $t < N$ die obersten $\frac{b_f}{2} - 1$ redundanten Frequenz-Zeilen entfernt. Die reduzierte Matrix ergibt sich also mit $S_{ft,red} = S_{ft}$ für $0 < f < \frac{b_f}{2}$.
Je nach gewählter Trainingsform wird die reduzierte Matrix $S_{ft,red} \in \mathbb{C}$ in eine reelle Matrix der Spektraldaten $D_{ft,spek}$ umgewandelt. Die Umwandlung geschieht entsprechend der Vorschrift

$$D_{ft_0,spek} = d(S_{ft_0,red}) \tag{12}$$

mithilfe der Umwandlungsabbildung

$$d : \mathbb{C}^{\frac{b_f}{2}+1} \rightarrow \begin{cases} \mathbb{R}^{\frac{b_f}{2}+1} & \text{| Training mit Betragsspektrum} \\ \mathbb{R}^{2\cdot(\frac{b_f}{2}+1)} & \text{| Training mit Phaseninformation} \end{cases}, \tag{13}$$

sodass sich die Anzahl der Spalten $t$ nicht ändert, jedoch je nach Trainingsform die Anzahl der Frequenz-Zeilen $f$. Zur Allgemeinhaltung wird die Anzahl der Zeilen im Folgenden mit $f_{D,max}$ bezeichnet.

Aus der Matrix $D_{ft,spek}$ wird die Matrix der normierten Spektraldaten $D_{ft,norm}$ erzeugt, indem für alle Matrixelemente der Mittelwert auf 0 und die Standardabweichung auf 1 erzwungen werden. Die Normierung findet demnach gemäß der Berechnungsvorschrift in Gleichung 14 für alle Matrixelemente $D_{ft,norm}$ statt.

$$D_{ft,norm} = \frac{D_{ft,spek} - \mu_{D_{spek}}}{\sigma_{D_{spek}}}. \tag{14}$$

Aus der Matrix der normierten Spektraldaten $D_{ft,norm}$ werden Merkmalsvektoren $T_t$ nach der Vorschrift in Gleichung 15 extrahiert. Dabei ist $t_0$ der Zeitschritt, ab dem für den Merkmalsvektor $T_{t0}$ ein Abschnitt, mit der Breite $b_t$ in der Dimension $t$, aus den Martrixelementen $D_{ft,norm}$ extrahiert wird. Der Abstand zwischen den Abschnitten ist $\Delta t$ und kann in der Trainingsphase kleiner als $b_t$ sein, sodass sich eine Überschneidung der Abschnitte ergibt. Im Anwendungsfall wird $\Delta t = b_t$ gewählt, sodass keine Überschneidung stattfindet.

$$T_{t_0} = \begin{pmatrix} D_{ft_0,norm} \\ \vdots \\ D_{f(t_0+b_t-1),norm} \end{pmatrix} \text{ und } T_{t_0} \in \mathbb{R}^{f_{D,max} \cdot b_t} \tag{15}$$

Die Anzahl der Merkmalsvektoren ist somit gegeben durch

$$N_T = \lfloor \frac{\frac{N}{s_t - b_t}}{\Delta t} + 1 \rfloor, \tag{16}$$

wobei die restlichen Elemente von $D_{ft,norm}$ für $t \geq N_T$ ignoriert bzw. nicht verwendet werden.

Die extrahierten Merkmalsvektoren $T_t$ werden in das DNN eingegeben und es werden die Ergebnisvektoren $A_t$ erzeugt. Je nach gewählter Maske gilt für $A_t$

$$A_t \in \begin{cases} \mathbb{R}^{(\frac{b_f}{2}+1) \cdot b_t} & | \text{ SM} \\ \mathbb{R}^{2 \cdot (\frac{b_f}{2}+1) \cdot b_t} & | \text{ cIRM} \\ \mathbb{R}^{(\frac{b_f}{2}+1) \cdot b_t} & | \text{ PSM} \end{cases} \tag{17}$$

Um Allgemeinheit zu gewähren wird die Anzahl der Elemente von $A_t$ im Folgenden als $f_{A,max}$ bezeichnet. Im Trainingsprozess werden die Ergebnisvektoren $A_t$ mit den erwarteten Ergebnisvektoren $A_{t,ideal}$ verglichen und entsprechend der gewählten Trainingsmethode für das Training ausgewertet.
Im Anwendungsfall wird der Ergebnisvektor zur einer Maske transformiert. Dazu wird zunächst die Matrix der Ergebnisvektoren $A_{ft}$ bezüglich der Spaltenzahl in das Format des Eingangs-Spektrogramms $S_{ft}$ gebracht, sodass sich für die vorläufige Maske ergibt

$$M_{norm} = \begin{pmatrix} A_{00} & \cdots & A_{(f_{A,max}-b_t-1)(b_t-1)} & A_{0(b_t)} & \cdots & A_{(f_{A,max}-b_t-1)(N_T+b_t)} \\ \vdots & & \vdots & \vdots & & \vdots \\ A_{(\frac{f_{A,max}}{b_t}-1)0} & \cdots & A_{(f_{A,max}-1)(b_t-1)} & A_{(\frac{f_{A,max}}{b_t}-1)(b_t)} & \cdots & A_{(f_{A,max}-1)(N_T+b_t)} \end{pmatrix}. \tag{18}$$

Die Elemente Matrix $M_{ft,norm}$ werden anschließend je nach gewählter Maske mithilfe der Umkehrfunktion der Normierungsabbildung $f_{norm,M} : \mathbb{R} \mapsto \mathbb{R}$ in Gleichung 29, 32 oder 36 entnormt, sodass $M_{ft,R} = f^{-1}_{norm,M}(M_{ft,norm})$. Anschließend wird auf die Spalten der entnormten Maske $M_{ft,R}$ (je nach Maske) entsprechend der Gleichung 30,34 oder 38 die Umformungsabbildung

$$m : \begin{cases} \mathbb{R}^{(\frac{b_f}{2}+1)} & \longrightarrow & \mathbb{R}^{(\frac{b_f}{2}+1)} & \text{| SM} \\ \mathbb{R}^{2\cdot(\frac{b_f}{2}+1)} & \longrightarrow & \mathbb{C}^{(\frac{b_f}{2}+1)} & \text{| cIRM} \\ \mathbb{R}^{(\frac{b_f}{2}+1)} & \longrightarrow & \mathbb{R}^{(\frac{b_f}{2}+1)} & \text{| PSM} \end{cases} \tag{19}$$

angewandt, sodass für jede Spalte $t_C$ der finalen Maske $M_{ft}$ gilt $M_{ft_C} = m(M_{ft_C,R})$.
Die Elemente der finalen Masken-Matrix $M_{ft}$ werden mit den Elementen der reduzierten Spektrogramm-Matrix $S_{ft,red}$ multipliziert, sodass sich die Elemente der Matrix des entstörten Spektrogramms ergeben mit

$$S_{ft,opt,red} = M_{ft} \cdot S_{ft,red}. \tag{20}$$

Die Matrix des vollständigen optimierten Spektrogramms ist dann aufgrund komplex konjugiert ungerader Symmetrie

$$S_{opt} = \begin{pmatrix} \overline{S_{\frac{b_f}{2},red}} \\ \vdots \\ \overline{S_{1,red}} \\ S_{0,red} \\ \vdots \\ S_{\frac{b_f}{2},red} \end{pmatrix}. \tag{21}$$

Das Ausgangssignal des Systems $y$ errechnet sich dann wie in Gleichung 22 durch die inverse STFT, mit Fensterbreite $b_f$, Schrittweite $s_t$ und der inversen Hanning-Fensterfunktion.

$$y = iSTFT(S_{opt}). \tag{22}$$

### Zusätzliche Systempfade im Trainingsprozess

Zusätzlich zu den Systempfaden im Anwendungsfall kommen für den Trainingsprozess weitere Signalpfade hinzu. Diese Signalpfade sind in Abbildung 8 rot dargestellt. Die Trainings- und Validierungsdaten werden in Form von separaten Signalen $a_{Sprache}$ und $a_{Stoer}$ in das Trainingssystem eingeführt. Das Störsignal $a_{Stoer}$ wird mithilfe des Korrekturfaktors $k$ auf ein definiertes $SNR$ gegenüber dem Nutzsignal $a_{Sprache}$ gebracht, sodass

$$SNR = \frac{P_{a,Sprache}}{P_{a,k,Stoer}} = \frac{|a_{Sprache}|^2}{|a_{k,Stoer}|^2} = \frac{|a_{Sprache}|^2}{|k \cdot a_{Stoer}|^2} = \frac{|a_{Sprache}|^2}{k^2 \cdot |a_{Stoer}|^2}. \tag{23}$$

Damit ergibt sich für den Korrekturfaktor

$$k = \frac{|a_{Sprache}|}{|a_{Stoer}|} \cdot \sqrt{SNR} \tag{24}$$

mit

$$a_{k,Stoer} = k \cdot a_{Stoer}. \tag{25}$$

Aus den Signalen $a_{Sprache}$ und $a_{k,Stoer}$ wird das Mischsignal mit definiertem $SNR$ erzeugt durch

$$a_{Misch} = a_{Sprache} + a_{k,Stoer}. \tag{26}$$

Die drei Signale $a_{Misch}$, $a_{Sprache}$ und $a_{k,Stoer}$ werden parallel in den Eingangspfad des Grundsystems eingeführt. Von den drei Signalen werden die Matrizen der reduzierten Spektrogramme $S_{ft,red,Misch}$, $S_{ft,red,Sprache}$ und $S_{ft,red,Stoer}$ erzeugt. Aus den reduzierten Matrizen $S_{ft,red,Sprache}$ und $S_{ft,red,Stoer}$ wird die Maske $M_{ft}$ (je nach gewählter Maske) entsprechend den Vorschriften in Gleichung 28,31 oder 35 berechnet.
Für die Validierungsdaten werden aus dem Spektrogramm $S_{ft,red,Misch}$ des Mischsignals und der Maske $M_{ft}$ jeweils die selben $N_{AV}$ zufällig lokalisierten Abschnitte mit Breite $b_V$ herausgeschnitten, sodass das Spektrogramm $S_{ft,red,val,Misch}$ und die Maske $M_{ft,val}$ der Validierungsdaten entstehen. Der Rest des Spektrogramms $S_{ft,red,Misch}$ und der Maske $M_{ft}$ dienen als Trainingsdaten und werden im Folgenden nicht anders benamt.
Die reduzierten Spektrogramme des Mischsignals $S_{ft,red,Misch}$ und $S_{ft,red,val,Misch}$ folgen weiter dem Signalpfad des Grundsystems. Die erzeugten Masken $M_{ft,C}$ und $M_{ft,C,val}$ werden je nach gewählter Maske mit der Abbildung $m^{-1}$ in Gleichung 30, 34 oder 38 umgeformt, sodass für jede Spalte $t_R$ von $M_{ft,R}$ und jede Spalte $t_R$ von $M_{ft,val,R}$ gilt $M_{ft_R,R} = m^{-1}(M_{ft_R,C})$ und $M_{ft_R,val,R} = m^{-1}(M_{ft_R,C})$. Die Elemente $M_{ft,R}$ und $M_{ft,val,R}$ werden anschließend je nach gewählter Maske mit der Abbildung $f_{norm}$ in Gleichung 29,32 oder 36 genormt und es entsteht die finale normierte Trainingsmaske $M_{ft,norm} = f_{norm}(M_{ft,R})$ und die finale Validierungsmaske $M_{ft,val,norm} = f_{norm}(M_{ft,val,R})$.

Wie für die Merkmalsvektoren im Pfad des Grundsystems werden die erwarteten Ergebnisvektoren $A_{t,ideal}$ nach der Vorschrift in Gleichung 27 extrahiert. Dabei ist $t_0$ der Zeitschritt, ab dem für den Ergebnisvektor $A_{t0,ideal}$ ein Abschnitt mit der Breite $b_t$ in der Dimension $t$ aus den Maskenelementen $M_{ft,norm}$ extrahiert wird.

$$A_{t_0,ideal} = \begin{pmatrix} M_{ft_0,norm} \\ \vdots \\ M_{f(t_0+b_t-1),norm} \end{pmatrix} \text{ und } T_{t_0} \in \mathbb{R}^{f_{D,max} \cdot b_t} \tag{27}$$

Die Anzahl der Elemente eines erwarteten Ergebnisvektors $A_{t,ideal}$ ergibt sich dann entsprechend Gleichung 17.
Der selbe Vorgang wird für die Maske der Validierungsdaten vorgenommen, sodass sich die erwarteten Ergebnisvektoren der Validierungsdaten $A_{t,val,ideal}$ ergeben.

## 4.2 Masken

In dieser Arbeit kommen drei verschiedene Masken zum Einsatz. Die verwendeten Bezeichnungen für die Masken ″Complex Ideal Ratio Mask″(cIRM) und ″Phase Sensitive Mask″(PSM) stammen aus einem Paper, in dem diese Masken eingesetzt wurden [3].

### Abwandlung des Simpson-Ansatzes (SM)

Die SM nutzt zur Berechnung der Maske das Verhältnis der Betragsspektren entsprechend

$$M_{ft} = \frac{|S_{ft,red,Sprache}|}{|S_{ft,red,Sprache}| + |S_{ft,red,Stoer}|}. \tag{28}$$

Für die SM wird keine Normierung angewandt, da bereits gilt, dass $M_{ft} \in [-1, 1]$ Dementsprechend ist die Normierungsabbildung die Identität mit

$$M_{ft,norm} = f_{norm,SM}(M_{ft}) = M_{ft}. \tag{29}$$

Eine Rekombination oder Trennung der Maskenelemente ist ebenfalls nicht notwendig, da diese bereits reell sind. Dementsprechend ist in diesem Fall die Umformungsabbildung die Identität mit

$$m(M_{ft_C}) = M_{ft_C,R} \text{ bzw. } m^{-1}(M_{ft_R}) = M_{ft_R,C} \tag{30}$$

Das Kürzel ″SM″ stammt von der Bezeichnung ″Soft Mask″in einer Realisierung dieser Maskenform von Mathworks [4].

### Komplexe Maske (cIRM)

Die cIRM nutzt zur Berechnung der Maske das Verhältnis der komplexen Spektren entsprechend

$$M_{ft} = \frac{S_{ft,red,Sprache}}{S_{ft,red,Sprache} + S_{ft,red,Stoer}}. \tag{31}$$

Da aufgrund der komplexen Division gilt $M_{ft,norm} \in (-\infty, \infty)$, müssen die Maskenelemente für die Verwendung im DNN normiert werden. Deshalb gilt für die Normierungsabbildung

$$M_{ft,norm} = f_{norm,cIRM}(M_{ft,R}) = Q \cdot \frac{1 - e^{-C \cdot M_{ft,R}}}{1 + e^{-C \cdot M_{ft,R}}} \text{ [3]} \tag{32}$$

und

$$M_{ft,R} = f^{-1}_{norm,cIRM}(M_{ft,norm}) = -\frac{1}{C} \cdot log\left(\frac{Q - M_{ft,norm}}{Q + M_{ft,norm}}\right) \text{ [3]}. \tag{33}$$

Da die Maskenelemente der cIRM komplexe Werte besitzen, das DNN jedoch nur reelle Werte verarbeitet, müssen die Maskenelemente rekombiniert bzw. getrennt werden, sodass gilt

$$m(M_{ft_R,R}) = \begin{pmatrix} M_{0t_R,R} + j \cdot M_{(\frac{b_f}{2}+1)t_R,R} \\ \vdots \\ M_{\frac{b_f}{2}t_R,R} + j \cdot M_{b_f t_R,R} \end{pmatrix} \text{ und } m^{-1}(M_{ft_C}) = \begin{pmatrix} Re\left(M_{0t_C}\right) \\ \vdots \\ Re\left(M_{\frac{b_f}{2}t_C}\right) \\ Im\left(M_{0t_C}\right) \\ \vdots \\ Im\left(M_{\frac{b_f}{2}t_C}\right) \end{pmatrix} \tag{34}$$

**Phasen-sensitive Maske (PSM)**

Die PSM nutzt zur Berechnung der Maske den Realteil des Verhältnisses der komplexen Spektren entsprechend

$$M_{ft} = Re\left(\frac{S_{ft,red,Sprache}}{S_{ft,red,Sprache} + S_{ft,red,Stoer}}\right). \tag{35}$$

Da aufgrund komplexer Division gilt $M_{ft,norm} \in (-\infty, \infty)$, müssen die Maskenelemente für die Verwendung im DNN normiert werden. Deshalb gilt für die Normierungsabbildung

$$M_{ft,norm} = f_{norm,PSM}(M_{ft,R}) = Q \cdot \frac{1 - e^{-C \cdot M_{ft,R}}}{1 + e^{-C \cdot M_{ft,R}}} \text{ [3]} \tag{36}$$

und

$$M_{ft,R} = f^{-1}_{norm,PSM}(M_{ft,norm}) = -\frac{1}{C} \cdot log\left(\frac{Q - M_{ft,norm}}{Q + M_{ft,norm}}\right) \text{ [3]} \tag{37}$$

Eine Rekombination oder Trennung der Maskenelemente ist wie bei der SM nicht notwendig, da die Maskenelemente bereits reell sind. Dementsprechend ist in diesem Fall die Umformungsabbildung die Identität mit

$$m(M_{ft_C}) = M_{ft_C,R} \text{ bzw. } m^{-1}(M_{ft_R}) = M_{ft_R,C} \tag{38}$$

## 4.3 Trainingsformen

In dieser Arbeit kommen 2 Trainingsformen zum Einsatz.

### Training mit Betragsspektrum

Im Training mit dem Betragsspektrum werden nur die Amplituden der Elemente des reduzierten Spektrogramms $S_{ft,red}$ genutzt. Dementsprechend gilt für die Abbildung

$$d(S_{ft0,red}) = \begin{pmatrix} \left|S_{0t_0,norm}\right| \\ \vdots \\ \left|S_{\frac{b_f}{2}t_0,norm}\right| \end{pmatrix} \tag{39}$$

### Training mit Phaseninformation

Im Training mit Phaseninformation wird das komplexe Spektrogramm in Real- und Imaginärteil aufgeteilt und parallel in das DNN eingeführt. Dementsprechend gilt

$$d(S_{ft0,red}) = \begin{pmatrix} Re\left(S_{0t_0,norm}\right) \\ \vdots \\ Re\left(S_{\frac{b_f}{2}t_0,norm}\right) \\ Im\left(S_{0t_0,norm}\right) \\ \vdots \\ Im\left(S_{\frac{b_f}{2}t_0,norm}\right) \end{pmatrix} \tag{40}$$

# 5 Implementierung

## 5.1 Arbeitsumgebung

### 5.1.1 Entwicklungsumgebung und Bibliotheken

Zur Implementierung des Systems zur Störgeräuschreduzierung wird die Entwicklungsumgebung Matlab von Mathworks mit der Version
MATLAB 2019b verwendet. Zusätzlich importiert werden die Bibliotheken ″Control System Toolbox″für verschiedene Funktionen der Signalverarbeitung und die ″Deep Learning Toolbox″für die Konfiguration, das Training und die Anwendung eines ″Deep Neural Network″.

Als Vorlage wird ein bestehendes System von MathWorks mit dem Namen ″Cocktail Party Source Separation Using Deep Learning Networks″[1] verwendet, welches das System im bereits vorgestellten Paper mit dem Namen ″Probabilistic Binary-Mask Cocktail-Party Source Separation in a Convolutional Deep Neural Network″[4] von Andrew J.R. Simpson, in Matlab implementiert.

### 5.1.2 Datenaufbereitung

In der Datenaufbereitung werden in einem separaten Skript die zur Verfügung stehenden Trainings- und Testdaten zur Verwendung vorbereitet. Das Skript erhält Verzeichnisse von jeweils Audiodateien zu Sprach- und Störsignalen. Da die Abtastraten der Audiodateien unterschiedlich sein können, müssen alle Audiodateien zunächst geladen, durch einen Tiefpass mit der Grenzfrequenz $4kHz$ gefiltert und auf eine Abtastrate von $8kHz$ abgetastet werden. Außerdem wird der Mittelwert jedes Signals auf 0 erzwungen. Damit ergeben sich einheitlich abgetastete Signale ohne Offset und ohne Aliasing im Spektrum.
Anschließend werden alle Sprachsignale und alle Störsignale jeweils auf die gleiche Leistung normiert und in ein gemeinsames Sprachsignal und ein gemeinsames Störsignal überführt.
Zuletzt werden das Sprachsignal und das Störsignal auf die gleiche Länge geschnitten und anschließend in ein gewünschtes Verzeichnis gespeichert.

Handelt es sich bei den Daten um Trainingsdaten, werden zusätzlich vom Sprachsignal die Pausen entfernt bevor die Sprachsignale zusammengefügt werden. Dies soll verhindern, dass Abschnitte ohne Leistung im Sprachsignal, im Trainingsprozess des DNN als Sprache klassifiziert werden und somit das Ergebnis verfälschen.
Zur Entfernung der Pausen im Sprachsignal wird ein 128 Schritte breites Fenster über die Sprachsignale mit normierter Amplitude gefahren und Abschnitte, die in ihrer Leistung unter einen vorher festgelegten Schwellen-

wert geraten, werden entfernt. Für diese Arbeit wurde der Schwellenwert anhand akustischer Tests auf 0.0001 festgelegt.
Die Realisierung als Matlab-Code ist in Listing 1 zu sehen.

```
function [] = SoundPreparation(FsNorm,PowerThreshold,RemovePauses,DirectoryVoice,
    DirectoryNoise,DirectoryResult)
% Funktionen von Soundpreparation:
% Angleichen vom Leistungslevel mehrerer Sounddateien, Zusammenfuegen
% mehrerer Sounddateien, Entfernen von Pausen (Abschnitte mit niedrigem
% Leistungslevel). Pausenentfernung wird nur auf das Sprachsignal angewandt.

% FsNorm: Integer -> Abtastrate auf die alle Dateien abgetastet werden
% PowerThreshold: Double -> Energie-Schwelle fuer die Pausenerkennung
% RemovePauses: bool -> Pausen werden entfernt oder nicht
% DirectoryVoice: String -> Verzeichnis der Sprachdateien
% DirectoryNoise: String -> Verzeichnis der Stoergeraeuschdateien
% DirectoryResult: String -> Zielverzeichnis der erzeugten zusammengefuegten
% Audiodateien

% Fensterlaenge der Pausenentfernung
WindowLength = 128;

if isfolder(DirectoryVoice)
    audioFilesVoice = dir(fullfile(DirectoryVoice, '**/*.WAV'));
elseif isfile(DirectoryVoice)
    audioFilesVoice = dir(DirectoryVoice);
else
end

if isfolder(DirectoryNoise)
    audioFilesNoise = dir(fullfile(DirectoryNoise, '**/*.WAV'));
elseif isfile(DirectoryNoise)
    audioFilesNoise = dir(DirectoryNoise);
else
end

% Anzahl der gefundenen Dateien
NumberOfVoiceFiles = size(audioFilesVoice,1);
NumberOfNoiseFiles = size(audioFilesNoise,1);

%---------Sprachdateien lesen und herunterabtasten-------------------------------------

SamplesVoice = {};
for i = 1:1:NumberOfVoiceFiles
    % Audiodatei lesen
    [audioTemp,FsVoice] = audioread(strcat(audioFilesVoice(i).folder,'\',audioFilesVoice(i
    ).name));

    % Audiosignal mit Tiefpass filtern undherunterabtasten
    audioSampleFiltered = decimate(audioTemp,FsVoice/FsNorm);

    % Entfernung von linearem Offset
     audioSampleFiltered = detrend(audioSampleFiltered,0);

    SamplesVoice{end+1} = audioSampleFiltered;
end

%---------Pausenentfernung falls gefordert
    ------------------------------------------------------------
if RemovePauses
    % Normierung der Amplitude der Sprachsignale zur Vorbereitung fuer die
    % Pausenentfernung
    for i = 1:1:NumberOfVoiceFiles
       SamplesVoice{i} = SamplesVoice{i}/max(SamplesVoice{i});
    end

    % Entfernung der Pausenabschnitte
    for i = 1:1:NumberOfVoiceFiles
        maskThreshold = zeros(length(SamplesVoice{i}),1);
        for j = 1:1:length(SamplesVoice{i})-WindowLength
            if rms(SamplesVoice{i}(j:j + WindowLength - 1))^2 > PowerThreshold
               maskThreshold(j + WindowLength - 1) = 1;
```

```
            else
               maskThreshold(j + WindowLength - 1) = 0;
            end
        end
        SamplesVoice{i} = SamplesVoice{i}(maskThreshold == 1);
    end
end

% Sprachsignale auf selbe Leistung normieren
for i = 1:1:NumberOfVoiceFiles
   SamplesVoice{i} = SamplesVoice{i}/rms(SamplesVoice{i});
end

% Sprachsignale zusammenfuegen
SampleVoice = [];
for i = 1:1:NumberOfVoiceFiles
    SampleVoice = [SampleVoice;SamplesVoice{i}];
end

%---------Stoergeraeuschdateien lesen und herunterabtasten
    ------------------------------------
SampleNoise = [];
for i = 1:1:NumberOfNoiseFiles
    % Audiodatei lesen
    [audioTemp,FsNoise] = audioread(strcat(audioFilesNoise(i).folder,'\',audioFilesNoise(i
    ).name));

    % Audiosignal mit Tiefpass filtern und herunterabtasten
    audioSampleFiltered = decimate(audioTemp,FsNoise/FsNorm);

    % Entfernung von linearem Offset
     audioSampleFiltered = detrend(audioSampleFiltered,0);

    % Audiosignale auf selbe Leistung normieren
    audioSampleFiltered = audioSampleFiltered/rms(audioSampleFiltered);

    % Audiosignale zusammenfuegen
    SampleNoise = [SampleNoise; reshape(audioSampleFiltered,length(audioSampleFiltered),1)
    ];
end

% Audiosignale auf gleiche Laenge schneiden
if length(SampleNoise) < length(SampleVoice)
    SampleVoice = SampleVoice(1:length(SampleNoise));
else
    SampleNoise = SampleNoise(1:length(SampleVoice));
end

% Amplitude normieren
SampleVoice = SampleVoice/max(abs(SampleVoice));
SampleNoise = SampleNoise/max(abs(SampleNoise));

% Audiodateien abspeichern
audiowrite(strcat(DirectoryResult,'Data Voice','.wav'),SampleVoice,FsNorm);
audiowrite(strcat(DirectoryResult,'Data Noise','.wav'),SampleNoise,FsNorm);

end
```

Listing 1: Implementierung der Datenaufbereitung

### 5.1.3 Auswertungsmetriken

Das System zur Störgeräuschreduzierung wird auf das Eingangssignal $x$ angewandt. Das Signal setzt sich aus dem bekannten Sprachsignal $x_{Nutz}$ und dem bekannten Störsignal $x_{Stoer}$ gemäß $x = x_{Nutz} + x_{Stoer}$ zusammen. Am Ausgang des Systems zur Störgeräuschreduzierung wird das entstörte Signal $y$ erzeugt, welches sich auf unbekannte Weise aus dem bekannten idealen Sprachsignal $x_{Nutz}$ und dem unbekannten Störsignal $y_{Stoer}$ gemäß $y = y_{Nutz} + y_{Stoer}$ mit $y_{Nutz} = c \cdot x_{Nutz}$ zusammensetzt. Der Faktor $c \in \mathbb{R}$ berücksichtigt hierbei, dass ein konstant verstärktes oder gedämpftes ideales Sprachsignal $c \cdot x_{Nutz}$ unabhängig von $c$ als Nutzsignal von $y$ betrachtet werden kann.
Es soll nun die Qualitätsdifferenz zwischen den Signalen $x$ und $y$ quantitativ erfasst werden. Dazu kommen die 3 Maße SNR, PESQ und STOI zum Einsatz. Im Folgenden wird beschrieben wie die Berechnung dieser Maße implementiert ist. Die konkrete Programmierung ist später in der Basis-Implementierung zu sehen.

#### SNR

Für das Ausgangssignal $y$ ist das SNR definiert als das Verhältnis der Leistung $P_{y,Nutz}$ des störungsfreien Sprachsignals $y_{Nutz}$ zur Leistung $P_{y,Stoer}$ des Störsignals $y_{Stoer}$. Es gilt also

$$SNR_y = \frac{P_{y,Nutz}}{P_{y,Stoer}}, \tag{41}$$

bzw. in Dezibel

$$SNRDB_y = 10 \cdot log\big(SNR_y\big)\, dB \text{ [6, Seite 199,200].} \tag{42}$$

Da die Signale $y_{Nutz}$ und $y_{Stoer}$ nicht bekannt sind, müssen sie zunächst auf Basis der bekannten Signale $y$ und $x_{Nutz}$ hergeleitet werden. Es gilt

$$y = y_{Nutz} + y_{Stoer} = c \cdot x_{Nutz} + y_{Stoer} \Leftrightarrow y_{Stoer} = y - c \cdot x_{Nutz}. \tag{43}$$

Es wird als Ansatz gewählt, dass das Nutzsignal $y_{Nutz} = c \cdot x_{Nutz}$ über die Wahl der konstante $c$ möglichst gut approximiert wird, wenn die Leistung $P_{y,Stoer}$ minimal wird. Das heißt für ein optimales c muss die Vorschrift in Gleichung 44 erfüllt sein.

$$P_{y,Stoer} \sim |y - c \cdot x_{Nutz}|^2 \longrightarrow min \tag{44}$$

$\Rightarrow$

$$y_0^2 - 2y_0 x_{Nutz,0} c + x_{Nutz,0}^2 c^2 + \ldots + y_n^2 - 2y_n x_{Nutz,n} c + x_{Nutz,n}^2 c^2 \longrightarrow min \tag{45}$$

⇒

$$(y_0^2+\ldots+y_n^2)-2c\cdot(y_0x_{Nutz,0}+\ldots+y_nx_{Nutz,n})+c^2\cdot(x_{Nutz,0}^2+\ldots+x_{Nutz,n}^2)\longrightarrow min. \tag{46}$$

Durch Ableitung nach $c$ ergibt sich das Minimum mit

$$-2\cdot(y_0x_{Nutz,0}+\ldots+y_nx_{Nutz,n})+2c\cdot(x_{Nutz,0}^2+\ldots+x_{Nutz,n}^2)=0 \tag{47}$$

⇒

$$c=\frac{y_0x_{Nutz,0}+\ldots+y_nx_{Nutz,n}}{x_{Nutz,0}^2+\ldots+x_{Nutz,n}^2}=\frac{\sum_{i=0}^{n}y_ix_{Nutz,i}}{\sum_{i=0}^{n}x_{Nutz,i}^2}. \tag{48}$$

Mit den Gleichungen 41, 43 und 48 kann nun der optimale Wer von $c$ anhand der bekannten Signale $y$ und $x_{Nutz}$ ausgerechnet werden. Für das SNR des Ausgangssignals $y$ gilt folglich

$$SNR_y=\frac{|c*x_{Nutz}|^2}{|y-c*x_{Nutz}|^2}. \tag{49}$$

**PESQ**

PESQ steht für ″Perceptual Evaluation of Speech Quality″und beinhaltet den ″Mean Opinion Score″(MOS). Der MOS ist eine Skala von 1.0 bis 4.5 und bewertet die Sprachqualität eines Audiosignals. Umso höher der Wert, umso besser ist die Bewertung [9]. Die Messung des PESQ MOS wird in dieser Arbeit mithilfe eines externen Skripts ausgeführt [11],[12].

**STOI**

Der STOI steht für ″Short Time Objective Intelligibility″und beschreibt die durchschnittliche Verständlichkeit eines Sprachsignals. Die Verständlichkeit wird durch eine Skala von 0 bis 1 quantisiert und kann als der prozentuale Anteil der verstehbaren Abschnitte im Sprachsignal verstanden werden [10]. zur Berechnung des STOI wird ein externes Skript herangezogen [13].

## 5.2 Basis-Implementierung (SM)

In diesem Abschnitt wird die Implementierung des Basis-Framework vorgestellt, welche als Grundlage für jedes System genutzt wird. Das Basis-Framework entspricht dem System mit der SM als Maske und mit dem Training ohne Phaseninformation. Alle dargestellten Werte und die DNN-Struktur entsprechen der Standard-Konfiguration und werden, wenn nicht explizit erwähnt, über alle getesteten Systeme beibehalten.

Zu Beginn wird das System über die Eingabe verschiedener Parameter definiert. Die getesteten Systeme unterscheiden sich in der Fensterbreite der STFT ″Winlen″, der Schrittweite der STFT ″STFTStepwidth″, der Abschnittslänge ″SequenceLength″, der maximalen Anzahl der Trainingsepochen ″EpochNumber″, der Struktur des DNN, der Maske und der Trainingsdatenformatierung. Die veränderten Elemente der Struktur des DNN umfassen die Wahl der Aktivierungsfunktionen und die Anzahl der Neuronen pro Ebene. Alle weiteren Parameter bleiben über alle Systeme hinweg konstant.

```
clearvars
%% Systemkonfiguration %%
FsNorm = 8000; % Abtastrate

% Konfiguration der Trainingsdaten
PowerThreshold = 0.0001; % Energie-Schwellwert fuer die Pausenentfernung Empfohlen:
    0.00005 to 0.001
SNRdBTraining = 0; % SNR des Mischsignals fuer Training und Validation in dB
SNRTraining = 10.^(SNRdBTraining/10);% SNR des Mischsignals fuer Training und Validation
    als Verhaeltnis

% Konfiguration der Validierungsdaten
ValidationDataSize = 0.15; % Prozentualer Anteil der Validierungsdaten aus den
    Trainingsdaten
ValidationDataNumberOfFragments = 32;   % Anzahl der Abschnitte die aus den Trainingsdaten
     herausgeschnitten werden
ValidationDataFragmentSize = ValidationDataSize/ValidationDataNumberOfFragments;

% Konfiguration der Testdaten
SNRdBTest = [0];    %SNR des Mischsignals fuer den Test in dB
SNRTest = 10.^(SNRdBTest/10);   %SNR des Mischsignals fuer den Test als Verhaeltnis

% Allgemeine Einstellungen
NumberOfRuns = 10;  % Anzahl der Systemtests
ForceTrainingSoundPreparation = false; % Nutzung von vorhandenen Trainingsdaten oder
    Neugenerierung
ForceTestSoundPreparation = true; % Nutzung von vorhandenen Testdaten oder Neugenerierung

% Konfiguration der STFT
Winlen = 256;
Overlap = Winlen-2;
Window = hann(Winlen,"periodic");
STFTStepwidth = Winlen - Overlap;

% Konfiguration der Trainings und Tests
SequenceLength = 20;    % Anzahl der STFT Zeitschritte in einem Merkmalsvektor

SequenceLengthTraining = SequenceLength;
SequenceOverlapTraining = round(SequenceLength/2);
SequenceStepwidthTraining = SequenceLengthTraining - SequenceOverlapTraining;

```

```
SequenceLengthTest = SequenceLength;
SequenceOverlapTest = 0;
SequenceStepwidthTest = SequenceLengthTest - SequenceOverlapTest;

EpochNumber     = 10; % Maximale Anzahl der Trainingsepochen
miniBatchSize = 128;

% Konfiguration des DNN

    % Anzahl der Koten der Eingangsebene
    inputLayerSize = (1 + Winlen/2) * SequenceLength;

    layers = [ ...

        imageInputLayer([1 1 inputLayerSize],"Normalization","None")

        % 1. Ebene
        fullyConnectedLayer(inputLayerSize)
        BiasedSigmoidLayer(6)
        batchNormalizationLayer
        dropoutLayer(0.1)

        % 2. Ebene
        fullyConnectedLayer(inputLayerSize)
        BiasedSigmoidLayer(6)
        batchNormalizationLayer
        dropoutLayer(0.1)

        % Ausgangsebene
        fullyConnectedLayer(inputLayerSize)
        BiasedSigmoidLayer(0)

        % Fehlerfunktion
        regressionLayer

        ];


% Verzeichnisse der Audiodateien
databaseTrainingRoot = 'C:\Users\svene\Google Drive\HAW\7. Semester\Bachelorarbeit\Arbeit\
    Aktuell\VoiceSeperation\Database\TrainingData\';
databaseTrainingVoiceRoot = strcat(databaseTrainingRoot,'Voice\');
databaseTrainingNoiseRoot = strcat(databaseTrainingRoot,'Noise\');

databaseTestRoot = 'C:\Users\svene\Google Drive\HAW\7. Semester\Bachelorarbeit\Arbeit\
    Aktuell\VoiceSeperation\Database\TestData\';
databaseTestVoiceRoot = strcat(databaseTestRoot,'Voice\');
databaseTestNoiseRoot = strcat(databaseTestRoot,'Noise\');

ResultsRoot = 'C:\Users\svene\Google Drive\HAW\7. Semester\Bachelorarbeit\Arbeit\Aktuell\
    VoiceSeperation\Results\';
```

Listing 2: Initialisierung der Systemparameter

Anschließend werden die Trainingssignale je nach Bedarf generiert und geladen. Die importierten Trainingssignale werden auf ein definiertes SNR gebracht und zum Mischsgnal kombiniert.

```
%% Importiere, filtere und kombiniere Audiodaten

% Generiere Trainingsdaten mit Pausenentfernung falls keine vorhanden
if ForceTrainingSoundPreparation
    SoundPreparation(FsNorm,PowerThreshold,true,databaseTrainingVoiceRoot,
    databaseTrainingNoiseRoot,databaseTrainingRoot);
end

TrainingSampleVoice = audioread(strcat(databaseTrainingRoot,'\','Data Voice.wav'));
TrainingSampleNoise = audioread(strcat(databaseTrainingRoot,'\','Data Noise.wav'));

% Korrigiere Leistung des Stoersignals entsprechend des konfigurierten SNR
NoiseCorrection = norm(TrainingSampleVoice)/(sqrt(SNRTraining)*norm(TrainingSampleNoise));
TrainingSampleNoise = NoiseCorrection * TrainingSampleNoise;

% Berechne Mischsignal fuer das Training
TrainingSampleMix = TrainingSampleVoice + TrainingSampleNoise;

% Normiere die Amplitude
TrainingSampleMix = TrainingSampleMix/max(abs(TrainingSampleMix));

% Trainingsdatenlaenge in Sekunden
TrainingDataLengthSeconds = length(TrainingSampleMix)/FsNorm;
```

Listing 3: Importieren der Trainingsdaten

Die STFTs der drei Signale und die Maske werden je nach gewählter Maske berechnet.

```
%% STFTs und Maske

% Sprachsignal
TrainingSampleVoiceSTFT = abs(stft(TrainingSampleVoice,'Window',Window,'OverlapLength',
    Overlap,'FFTLength',Winlen));
clear TrainingSampleVoice;
TrainingSampleVoiceSTFT = TrainingSampleVoiceSTFT(Winlen/2:end,:);

% Stoersignal
TrainingSampleNoiseSTFT = abs(stft(TrainingSampleNoise,'Window',Window,'OverlapLength',
    Overlap,'FFTLength',Winlen));
clear TrainingSampleNoise;
TrainingSampleNoiseSTFT = TrainingSampleNoiseSTFT(Winlen/2:end,:);

% Berechne die Maske
TrainingSampleMask0 = TrainingSampleVoiceSTFT ./ (TrainingSampleVoiceSTFT +
    TrainingSampleNoiseSTFT + eps);
clear TrainingSampleVoiceSTFT;
clear TrainingSampleNoiseSTFT;

% Mischsignal
TrainingSampleMixSTFT0 = abs(stft(TrainingSampleMix,'Window',Window,'OverlapLength',
    Overlap,'FFTLength',Winlen));
clear TrainingSampleMix;
TrainingSampleMixSTFT0 = TrainingSampleMixSTFT0(Winlen/2:end,:);
```

Listing 4: Berechnung der STFTs und der Maske

Aus der STFT des Mischsignals und der Maske werden die Validierungs- und Trainingsdaten zufällig voneinander separiert.

```
%% Vorbereitung der Testlaeufe
timeStamp = datestr(now,'dd mmmm yyyy HH-MM-SS');
ResultsDir = strcat(ResultsRoot,timeStamp,'\');
mkdir(ResultsDir);
Results = {};
for RunNumber=1:1:NumberOfRuns
    %% Extrahiere Validierungsdaten
    TrainingSampleMixSTFT = TrainingSampleMixSTFT0;
    TrainingSampleMask = TrainingSampleMask0;
    ValidationSampleMixSTFT = [];
    ValidationSampleMask = [];

    for ValidationFragmentationCounter=0:1:ValidationDataNumberOfFragments-1

        randomNumber = rand;

        FragmentsizeCorrection = 1/(1-ValidationDataFragmentSize*
    ValidationFragmentationCounter);
        ValidationFragment = [randomNumber * (1-ValidationDataFragmentSize*
    FragmentsizeCorrection) , randomNumber * (1-ValidationDataFragmentSize*
    FragmentsizeCorrection) + ValidationDataFragmentSize*FragmentsizeCorrection];
        ValidationFragmentIndex = round(ValidationFragment * size(TrainingSampleMixSTFT,2)
    );

        ValidationSampleMixSTFT = [ValidationSampleMixSTFT, TrainingSampleMixSTFT(:,
    ValidationFragmentIndex(1):ValidationFragmentIndex(2))];
        ValidationSampleMask = [ValidationSampleMask, TrainingSampleMask(:,
    ValidationFragmentIndex(1):ValidationFragmentIndex(2))];

        TrainingSampleMixSTFT = [TrainingSampleMixSTFT(:,1:ValidationFragmentIndex(1)-1),
    TrainingSampleMixSTFT(:,ValidationFragmentIndex(2)+1:end)];
        TrainingSampleMask = [TrainingSampleMask(:,1:ValidationFragmentIndex(1)-1),
    TrainingSampleMask(:,ValidationFragmentIndex(2)+1:end)];

    end
```

Listing 5: Separation der Trainings- und Validierungsdaten

Die Trainings- und Validierungsdaten werden separat bezüglich Mittelwert und Standardabweichung normiert.

```
    %% Normiere Trainings und Validierungsdaten bezueglich Mittelwert und
    Standardabweichung

    MV = mean(TrainingSampleMixSTFT(:));
    StdDev = std(TrainingSampleMixSTFT(:));
    TrainingSampleMixSTFT = (TrainingSampleMixSTFT - MV) / StdDev;

    MV = mean(ValidationSampleMixSTFT(:));
    StdDev = std(ValidationSampleMixSTFT(:));
    ValidationSampleMixSTFT = (ValidationSampleMixSTFT - MV) / StdDev;
```

Listing 6: Normierung der Trainings- und Validierungsdaten

Aus den normierten Daten werden die Merkmalsvektoren für das Training erstellt.

```
%% Erstellung der Merkmalsvektoren fuer Training und Validierung

% Anzahl der Abschnitte
AmountTrainingSequences = floor((size(TrainingSampleMixSTFT,2) - SequenceLengthTraining)/SequenceStepwidthTraining + 1);
AmountValidationSequences = floor((size(ValidationSampleMixSTFT,2) - SequenceLengthTraining)/SequenceStepwidthTraining + 1);


% Bereite Speicherplatz fuer die Abschnitte vor
TrainingSequencesMix = zeros(Winlen/2 + 1,SequenceLengthTraining,1, AmountTrainingSequences);
TrainingSequencesMask = zeros(Winlen/2 + 1,SequenceLengthTraining,1, AmountTrainingSequences);

% Extrahiere Abschnitte aus den STFTs
for SequenceCounter = 1:1:AmountTrainingSequences
    stftIndex0 = (SequenceCounter-1)*(SequenceLengthTraining-SequenceOverlapTraining) + 1;
    stftIndex1 = (SequenceCounter-1)*(SequenceLengthTraining-SequenceOverlapTraining) + SequenceLengthTraining;

    TrainingSequencesMix(:,:,:,SequenceCounter) = TrainingSampleMixSTFT(:,stftIndex0:stftIndex1);
    TrainingSequencesMask(:,:,:,SequenceCounter) = TrainingSampleMask(:,stftIndex0:stftIndex1);
end
clear TrainingSampleMixSTFT;
clear TrainingSampleMask;

ValidationSequencesMix = zeros(Winlen/2 + 1,SequenceLengthTraining,1, AmountValidationSequences);
ValidationSequencesMask = zeros(Winlen/2 + 1,SequenceLengthTraining,1, AmountValidationSequences);
for SequenceCounter = 1:1:AmountValidationSequences
    stftIndex0 = (SequenceCounter-1)*(SequenceLengthTraining-SequenceOverlapTraining) + 1;
    stftIndex1 = (SequenceCounter-1)*(SequenceLengthTraining-SequenceOverlapTraining) + SequenceLengthTraining;

    ValidationSequencesMix(:,:,:,SequenceCounter) = ValidationSampleMixSTFT(:,stftIndex0:stftIndex1);
    ValidationSequencesMask(:,:,:,SequenceCounter) = ValidationSampleMask(:,stftIndex0:stftIndex1);
end
clear ValidationSampleMixSTFT;
clear ValidationSampleMask;

% Vektoren umformen fuer die Eingabe in das DNN
TrainingSequencesMixVectors = reshape(TrainingSequencesMix, [1 1 (1 + Winlen/2)*SequenceLengthTraining size(TrainingSequencesMix,4)]);
clear TrainingSequencesMix;
TrainingSequencesMaskVectors = reshape(TrainingSequencesMask, [1 1 (1 + Winlen/2)*SequenceLengthTraining size(TrainingSequencesMask,4)]);
clear TrainingSequencesMask;

ValidationSequencesMixVectors = reshape(ValidationSequencesMix, [1 1 (1 + Winlen/2)*SequenceLengthTraining size(ValidationSequencesMix,4)]);
clear ValidationSequencesMix;
ValidationSequencesMaskVectors = reshape(ValidationSequencesMask, [1 1 (1 + Winlen/2)*SequenceLengthTraining size(ValidationSequencesMask,4)]);
clear ValidationSequencesMask;
```

Listing 7: Erstellung der Merkmalsvektoren

Mit den merkmalsvektoren der Trainings- und Validierungsdaten wird das DNN trainiert.

```
%% Training des DNN %%

options = trainingOptions("adam", ...
"MaxEpochs",EpochNumber, ...
"MiniBatchSize",miniBatchSize, ...
"SequenceLength","longest", ...
"Shuffle","every-epoch",...
"Verbose",0, ...
"Plots","training-progress",...
"ValidationFrequency",floor(size(TrainingSequencesMixVectors,4)/miniBatchSize),...
"ValidationData",{ValidationSequencesMixVectors,ValidationSequencesMaskVectors},...
"ValidationPatience",2,...
"LearnRateSchedule","piecewise",...
"LearnRateDropFactor",0.9,...
"LearnRateDropPeriod",1,...
"ExecutionEnvironment",'multi-gpu');

VoiceSeparationNet = trainNetwork(TrainingSequencesMixVectors,
TrainingSequencesMaskVectors,layers,options);

clear TrainingSequencesMixVectors;
clear TrainingSequencesMaskVectors;
clear ValidationSequencesMixVectors;
clear ValidationSequencesMaskVectors;
```

Listing 8: Training des DNN

Zum Schluss wird das System getestet, indem in einer Schleife alle Testsignale geladen, das DNN darauf angewandt, die resultierende Maske mit dem Originalspektrum multipliziert und das verbesserte Mischsignal anhand der eingeführten Metriken getestet wird.

```
    %% Teste das DNN mit den gewuenschten SNRs

    TestVoiceFiles = dir(fullfile(databaseTestVoiceRoot,'**/*.WAV'));
    NumberOfTestVoiceFiles = size(TestVoiceFiles,1);
    TestNoiseFiles = dir(fullfile(databaseTestNoiseRoot,'**/*.WAV'));
    NumberOfTestNoiseFiles = size(TestNoiseFiles,1);

    for VoiceFileCount = 1:1:NumberOfTestVoiceFiles
    for NoiseFileCount = 1:1:NumberOfTestNoiseFiles
    for i=1:1:length(SNRdBTest)

        % Generiere und lade aktuelle Testsignale
        TestVoiceFileName = strcat(TestVoiceFiles(VoiceFileCount).folder,'\',
    TestVoiceFiles(VoiceFileCount).name);
        TestNoiseFileName = strcat(TestNoiseFiles(NoiseFileCount).folder,'\',
    TestNoiseFiles(NoiseFileCount).name);

        SoundPreparation(FsNorm,PowerThreshold,false,TestVoiceFileName,TestNoiseFileName,
    databaseTestRoot);

        TestSampleVoice = audioread(strcat(databaseTestRoot,'\','Data Voice.wav'));
        TestSampleNoise = audioread(strcat(databaseTestRoot,'\','Data Noise.wav'));

        % Korrigiere Leistung des Stoersignals entsprechend des konfigurierten SNR
        NoiseCorrection = norm(TestSampleVoice)/(sqrt(SNRTest(i))*norm(TestSampleNoise));
        TestSampleNoise = NoiseCorrection * TestSampleNoise;

        % Erzeuge Mischsignal
        TestSampleMix = TestSampleVoice + TestSampleNoise;

        % STFT vom Mischsignal
        TestSampleMixSTFT0 = stft(TestSampleMix,'Window',Window,'OverlapLength',Overlap,'
    FFTLength',Winlen);
        TestSampleMixSTFT0 = TestSampleMixSTFT0(Winlen/2:end,:);

        % Erzeuge Spektraldaten je nach gewaehlter Trainingsform
        TestSampleMixSTFT = abs(TestSampleMixSTFT0);

        % Normiere Trainings und Validierungsdaten bezueglich Mittelwert und
    Standardabweichung
        MV = mean(TestSampleMixSTFT(:));
        StdDev = std(TestSampleMixSTFT(:));
        TestSampleMixSTFT = (TestSampleMixSTFT - MV) / StdDev;

        % Erzeuge Merkmalsvektoren
        AmountTestSequences = floor((length(TestSampleMixSTFT) - SequenceLengthTest) /
    SequenceStepwidthTest + 1);
        TestSequencesMix = zeros(Winlen/2 + 1,SequenceLengthTest,1,AmountTestSequences);
        for SequenceCounter = 1:1:AmountTestSequences
            stftIndex0 = (SequenceCounter-1)*(SequenceLengthTest-SequenceOverlapTest) + 1;
            stftIndex1 = (SequenceCounter-1)*(SequenceLengthTest-SequenceOverlapTest) +
    SequenceLengthTest;

            TestSequencesMix(:,:,:,SequenceCounter) = TestSampleMixSTFT(:,stftIndex0:
    stftIndex1);
        end
        TestSequencesMixVectors = reshape(TestSequencesMix, [1 1 (1 + Winlen/2)*
    SequenceLengthTest size(TestSequencesMix,4)]);

        % Erzeuge Ergebnisvektoren mit dem trainierten DNN nd den erzeugten
        % Merkmalsvektoren
        MaskRaw = predict(VoiceSeparationNet,TestSequencesMixVectors);
```

```
54
55          MaskRaw = MaskRaw.';
56          MaskRaw = reshape(MaskRaw,1 + Winlen/2,numel(MaskRaw)/(1 + Winlen/2));
57
58          % Berechne die Maske je nach gewaehlter Maske
59          Mask = MaskRaw;
60
61          % Wende die Maske auf das Originalspektrum des Mischsignals an
62          MixSTFTOrigin = TestSampleMixSTFT0(:,1:length(Mask));
63          MixImprovedSTFT = MixSTFTOrigin .* Mask;
64
65          % Rekonstruiere redundante Spektrumselemente
66          MixImprovedSTFT = [conj(MixImprovedSTFT(end-1:-1:2,:)) ; MixImprovedSTFT];
67
68          % Erzeuge verbessertes Mischsignal mit der inversen STFT
69          MixImprovedSTFT(isinf(MixImprovedSTFT)) = 0;
70          MixImproved = istft(MixImprovedSTFT, 'Window',Window,'OverlapLength',Overlap,'
    FFTLength',Winlen,'ConjugateSymmetric',true);
71
72          % Definiere Bereich der getestet werden soll
73          TestArea = numel(Window):numel(MixImproved)-numel(Window);
74
75          % Speichere Resultate
76          currentTestString = strcat(num2str(RunNumber),'-',num2str(VoiceFileCount),num2str(
    NoiseFileCount),' ',num2str(SNRdBTest(i)),'dB');
77          audiowrite(strcat(ResultsDir,'Result ',currentTestString,' ','origin.wav'),
    TestSampleMix(TestArea)*4/norm(TestSampleMix(TestArea)),FsNorm);
78          audiowrite(strcat(ResultsDir,'Result ',currentTestString,' ','estimated.wav'),
    MixImproved(TestArea)*4/norm(MixImproved(TestArea)),FsNorm);
79          audiowrite(strcat(ResultsDir,'Result ',currentTestString,' ','voice.wav'),
    TestSampleVoice(TestArea)*4/norm(TestSampleVoice(TestArea)),FsNorm);
80
81          % Berechne die Auswertungsmetriken
82          c = sum(MixImproved(TestArea).*TestSampleVoice(TestArea))/(norm(TestSampleVoice(
    TestArea))^2);
83          snr value origin = snr(TestSampleVoice(TestArea),TestSampleNoise(TestArea));
84          snr value estimated = snr(c*TestSampleVoice(TestArea),MixImproved(TestArea)-c*
    TestSampleVoice(TestArea));
85          pesq value origin = pesq2 mtlb(strcat('Result ',currentTestString,' ','voice.wav')
    ,strcat('Result ',currentTestString,' ','origin.wav'),FsNorm,'nb','pesq NoResFile.exe'
    ,strcat('Results\',timeStamp));
86          pesq value estimated = pesq2 mtlb(strcat('Result ',currentTestString,' ','voice.
    wav'),strcat('Result ',currentTestString,' ','estimated.wav'),FsNorm,'nb','
    pesq NoResFile.exe',strcat('Results\',timeStamp));
87          stoi value origin = stoi(double(TestSampleVoice(TestArea)/norm(TestSampleVoice(
    TestArea))),double(TestSampleMix(TestArea)/norm(TestSampleMix(TestArea))),FsNorm);
88          stoi value estimated = stoi(double(TestSampleVoice(TestArea)/norm(TestSampleVoice(
    TestArea))),double(MixImproved(TestArea)/norm(MixImproved(TestArea))),FsNorm);
89
90          % Schreibe Analyseergebnisse in eine Datenmatrix
91          Results(end+1,:) = {strcat(currentTestString,' PESQ'),strcat(currentTestString,'
     STOI'),strcat(currentTestString,' SNR')};
92          Results(end+1,:) = {pesq value origin(1),stoi value origin,snr value origin};
93          Results(end+1,:) = {pesq value estimated(1),stoi value estimated,
    snr value estimated};
94      end
95      end
96      end
97  end
98
99  Results(end+1,1) = {strcat('Network trained by SNR of ', num2str(SNRdBTraining),' dB')};
100
101 % Schreibe alle Analyseergebnisse in eine csv-Datei
102 writecell(Results,strcat(ResultsDir,'Results','.csv'),'Delimiter',';');
```

Listing 9: Testphase

## 5.3 Variationen

Die Verwendung anderer Masken und Trainingsformen erfordert einige Anpassungen der Basis-Implementation. Diese sind in den folgenden Unterpunkten erläutert.

### 5.3.1 cIRM

Im Unterschied zur Basis-Implementierung wird im Fall der cIRM nicht der Betrag der Spektralwerte, sondern der komplexe Wert zur Berechnung der Maske benutzt. Die Anzahl der Neuronen in der Ausgangsebene ist doppelt so groß und in der vorletzten Ebene $1,5$ mal so groß wie in der Basisimplementierung. Die Anzahl der Neuronen in der Eingangsebene bleibt gleich.

```
%% STFTs und Maske

% Sprachsignal
TrainingSampleVoiceSTFT = stft(TrainingSampleVoice,'Window',Window,'OverlapLength',Overlap
    ,'FFTLength',Winlen);
clear TrainingSampleVoice;
TrainingSampleVoiceSTFT = TrainingSampleVoiceSTFT(Winlen/2:end,:);

% Stoersignal
TrainingSampleNoiseSTFT = stft(TrainingSampleNoise,'Window',Window,'OverlapLength',Overlap
    ,'FFTLength',Winlen);
clear TrainingSampleNoise;
TrainingSampleNoiseSTFT = TrainingSampleNoiseSTFT(Winlen/2:end,:);

% Berechne die Maske
TrainingSampleMask0 = TrainingSampleVoiceSTFT ./ (TrainingSampleVoiceSTFT +
    TrainingSampleNoiseSTFT);
clear TrainingSampleVoiceSTFT;
clear TrainingSampleNoiseSTFT;

% Mischsignal
TrainingSampleMixSTFT0 = abs(stft(TrainingSampleMix,'Window',Window,'OverlapLength',
    Overlap,'FFTLength',Winlen));
```

Listing 10: Berechnung der STFTs und der Maske bei der cIRM

Die komplexen Elemente der Maske werden in Real- und Imaginärteil separiert, sodass die reellen Trainingsdaten entstehen.

```
    %% Separiere komplexe Daten in Real- und Imaginaerteil

    TrainingSampleMaskReIm = [real(TrainingSampleMask);imag(TrainingSampleMask)];
    clear TrainingSampleMask;

    ValidationSampleMaskReIm = [real(ValidationSampleMask);imag(ValidationSampleMask)];
    clear ValidationSampleMask;
```

Listing 11: Separation von Imaginär- und Realteil bei der cIRM

Die separierte Maske wird wie im Konzept in Gleichung 32 normiert.

```
%% Normiere Maske mit hyperbolischer Funktion
Q = 1;
C = 0.5;

temp = exp(-C*TrainingSampleMaskReIm);
temp(temp == inf) = realmax('double');
TrainingSampleMaskReIm = Q * (1-temp)./(1+temp);

temp = exp(-C*ValidationSampleMaskReIm);
temp(temp == inf) = realmax('double');
ValidationSampleMaskReIm = Q * (1-temp)./(1+temp);

clear temp;
```

Listing 12: Normierung der cIRM

In der Testphase muss die Maske denormiert und zur komplexen Maske rekombiniert werden.

```
        % Denormiere die Maske
        MaskRaw = -log((Q-MaskRaw)./(Q+MaskRaw))/C;
        MaskRaw(MaskRaw == inf) = realmax('double');

        % Berechne die Maske je nach gewaehlter Maske
        Mask = MaskRaw(1:Winlen/2+1,:) + 1i*MaskRaw(Winlen/2 + 2:end,:);
```

Listing 13: Denormierung und Rekombination der cIRM in der Testphase

### 5.3.2 PSM

Die Maske bei der Verwendung der PSM berechnet sich durch den Realteil der cIRM.

```
%% STFTs und Maske

% Sprachsignal
TrainingSampleVoiceSTFT = stft(TrainingSampleVoice,'Window',Window,'OverlapLength',Overlap
    ,'FFTLength',Winlen);
clear TrainingSampleVoice;
TrainingSampleVoiceSTFT = TrainingSampleVoiceSTFT(Winlen/2:end,:);

% Stoersignal
TrainingSampleNoiseSTFT = stft(TrainingSampleNoise,'Window',Window,'OverlapLength',Overlap
    ,'FFTLength',Winlen);
clear TrainingSampleNoise;
TrainingSampleNoiseSTFT = TrainingSampleNoiseSTFT(Winlen/2:end,:);

% Berechne die Maske
TrainingSampleMask0 = real(TrainingSampleVoiceSTFT ./ (TrainingSampleVoiceSTFT +
    TrainingSampleNoiseSTFT));
clear TrainingSampleVoiceSTFT;
clear TrainingSampleNoiseSTFT;

% Mischsignal
TrainingSampleMixSTFT0 = abs(stft(TrainingSampleMix,'Window',Window,'OverlapLength',
    Overlap,'FFTLength',Winlen));
clear TrainingSampleMix;
TrainingSampleMixSTFT0 = TrainingSampleMixSTFT0(Winlen/2:end,:);
```

Listing 14: Berechnung der STFTs und der Maske bei der PSM

Da die Elemente der PSM bereits reell sind, müssen Real-und Imaginärteil nicht separiert werden. Jedoch müssen die Elemente wie bei der cIRM normiert werden.

```
    %% Normiere Maske mit hyperbolischer Funktion
    Q = 1;
    C = 0.5;

    temp = exp(-C*TrainingSampleMask);
    temp(temp == inf) = realmax('double');
    TrainingSampleMask = Q * (1-temp)./(1+temp);

    temp = exp(-C*ValidationSampleMask);
    temp(temp == inf) = realmax('double');
    ValidationSampleMask = Q * (1-temp)./(1+temp);

    clear temp;
```

Listing 15: Normierung der PSM

Genau wie bei der cIRM müssen die Elemente der Maske in der Testphase denormiert werden.

```
        % Denormiere die Maske
        Mask = -log((Q-Mask)./(Q+Mask))/C;
        Mask(Mask == inf) = realmax('double');
```

Listing 16: Denormierung und Rekombination der PSM in der Testphase

### 5.3.3 Training mit Phaseninformation

Das Training mit Phaseninformation erfordert für das Training und den Test das Aufteilen der komplexen Spektrogramme in Real- und Imaginärteil. Bei der SM wird das Training mit Phaseninformation nicht angewandt.
Bei der cIRM wird dafür in jeder Ebene des DNN die doppelte Neuronen-Anzahl wie in der Basis-Implementierung verwendet. Bei der PSM erhält die Eingangsebene die doppelte Anzahl und die zweite Ebene die 1, 5-fache Anzahl. Die Neuronen-Anzahl in der Ausgangsebene entspricht bei der PSM der Basis-Implementierung.

Bei der cIRM ergibt sich die Separation der Trainings- und Validierungsdaten und der Maske in Real- und Imaginärteil wie folgt.

```
%% Separiere komplexe Daten in Real- und Imaginaerteil

TrainingSampleMixSTFTReIm = [real(TrainingSampleMixSTFT);imag(TrainingSampleMixSTFT)];
clear TrainingSampleMixSTFT;
TrainingSampleMaskReIm = [real(TrainingSampleMask);imag(TrainingSampleMask)];
clear TrainingSampleMask;

ValidationSampleMixSTFTReIm = [real(ValidationSampleMixSTFT);imag(
ValidationSampleMixSTFT)];
clear ValidationSampleMixSTFT;
ValidationSampleMaskReIm = [real(ValidationSampleMask);imag(ValidationSampleMask)];
clear ValidationSampleMask;
```

Listing 17: Separation von Imaginär- und Realteil der Trainings- und Validierungsdaten bei der cIRM und Training mit Phaseninformation

Bei der PSM ist die Separation der Trainings- und Validierungsdaten entsprechend.

```
%% Separiere komplexe Daten in Real- und Imaginaerteil

TrainingSampleMixSTFTReIm = [real(TrainingSampleMixSTFT);imag(TrainingSampleMixSTFT)];
clear TrainingSampleMixSTFT;

ValidationSampleMixSTFTReIm = [real(ValidationSampleMixSTFT);imag(
ValidationSampleMixSTFT)];
clear ValidationSampleMixSTFT;
```

Listing 18: Separation von Imaginär- und Realteil der Trainings- und Validierungsdaten bei der PSM und Training mit Phaseninformation

In der Testphase müssen bei der cIRM und der PSM die Testdaten ebenfalls separiert werden. Das geschieht bei der cIRM und der PSM wie im Folgenden.

```
        StdDev = std(TestSampleMixSTFTReIm(:));
        TestSampleMixSTFTReIm = (TestSampleMixSTFTReIm - MV) / StdDev;
```

Listing 19: Separation von Imaginär- und Realteil der Trainings- und Validierungsdaten bei der cIRM und PSM mit Training mit Phaseninformation

# 6 Tests und Ergebnisse

## 6.1 Testablauf

Jedes zu testende System unterscheidet sich in mindestens einem Testparameter. In jedem System-Test werden die selben Trainings- und Testdaten verwendet. Jeder System-Test wird einmal mit männlichen Stimmen und einmal mit weiblichen Stimmen in den Trainings- und Testdaten durchgeführt. Ein System-Test umfasst 10 vollständige Trainingsdurchläufe und Testung des trainierten DNNs mit jeweils neu zufällig extrahierten Validierungsdaten. Die Auswertungsergebnisse der 10 trainierten DNNs ergeben gemittelt die Auswertungsergebnisse eines System-Tests. Jedes mit männlichen Stimmen trainierte DNN wird anhand der Audiosignale in Tabelle 1 mit männlichen Stimmen, also insgesamt 3 Sprach- und Störsignal-Kombinationen mit jeweils 30 Sekunden Länge, getestet. Entsprechend wird jedes, mit weiblichen Stimmen trainierte DNN, mit den 3 Sprach- und Störsignal-Kombinationen mit weiblicher Stimme getestet. Jedes System, inklusive der Extraktion der Validierungsdaten, ist entsprechend der Basis-Implementierung konfiguriert. Die Trainingsdurchläufe der SM umfassen jeweils 10 Epochen, die Trainingsdurchläufe der cIRM und der PSM jedoch jeweils nur 3 Epochen. Abweichungen von der Basiskonfiguration werden mit einem Kürzel gekennzeichnet.

| Sprachsignal | + | Störsignal |
|---|---|---|
| Stimmen | + | Maschine |
| Stimmen | + | Fahrzeugmotor |
| Stimmen | + | Kettensägen |

Tabelle 1: Sprach- und Störsignal-Kombinationen der Testsignale

In jedem Test wird das trainierte DNN auf eine Sprach- und Störsignal-Kombination angewendet und die Qualitätsverbesserung wird gemessen. Die Qualitätsindikatoren sind das SNR, der PESQ und die STOI.

Die Trainingsdaten bestehen aus Sprach- und Störsignalen mit ca. 240 Sekunden Länge. Die Sprachsignale stammen aus der sogenannten TIMIT-Datenbank [14] und werden jeweils doppelt verwendet. Die Verzeichnisse der Sprachsignale der Trainingsdaten sind in Tabelle 2 dargestellt.

| Männliche Stimmen | Weibliche Stimmen |
|---|---|
| TRAIN/DR1/MCPM0/*.WAV | TRAIN/DR1/FCJF0/*.WAV |
| TRAIN/DR1/MDAC0/*.WAV | TRAIN/DR1/FDAW0/*.WAV |
| TRAIN/DR1/MDPK0/*.WAV | TRAIN/DR1/FDML0/*.WAV |
| TRAIN/DR1/MEDR0/*.WAV | TRAIN/DR1/FECD0/*.WAV |
| TRAIN/DR1/MGRL0/*.WAV | TRAIN/DR1/FETB0/*.WAV |
| TRAIN/DR1/MJEB1/*.WAV | TRAIN/DR1/FJSP0/*.WAV |

Tabelle 2: Verzeichnisse der TIMIT-Datenbank der Sprachsignale der Trainingsdaten

Die Störsignale der Trainingsdaten setzen sich 10 Sekunden langen Ausschnitten aus 24 verschiedenen Audiodateien zusammen. Die genaue Zusammensetzung der Störsignale ist in Tabelle 3 dargestellt.

| Nr. | Abschnitt in Sek - Sek | Nr. | Abschnitt in Sek - Sek |
|---|---|---|---|
| 1 | 0 - 10 | 20 | 0 - 10 |
| 3 | 2 - 12 | 21 | 5 - 15 |
| 4 | 1 - 11 | 22 | 0 - 10 |
| 7 | 4-14 | 23 | 0 - 10 |
| 9 | 0 - 10 | 24 | 10,5 - 20,5 |
| 11 | 0 - 10 | 25 | 0 - 10 |
| 12 | 0 - 10 | 26 | 0 - 10 |
| 13 | 0 - 10 | 27 | 5 - 15 |
| 14 | 0 - 10 | 28 | 0 - 10 |
| 16 | 1 - 11 | 30 | 1 - 11 |
| 17 | 10 - 20 | 31 | 0 - 10 |
| 19 | 0 - 10 | 32 | 4 - 14 |

Tabelle 3: Zusammensetzung der Störsignale der Trainingsdaten mit Nummerierung der Liste im Literaturverzeichnis [15].

Die Sprachsignale für die Testungen werden durch die Audiodateien in Tabelle 4 zusammengesetzt. Das Signal für den Test mit männlicher Stimme und das Signal für die Tests mit weiblicher Stimme sind so jeweils 30 Sekunden lang.

| Männliche Stimmen | Weibliche Stimmen |
|---|---|
| TEST/DR1/MDAB0/SA1.WAV | TEST/DR1/FAKS0/SA1.WAV |
| TEST/DR1/MJSW0/SI1010.WAV | TEST/DR1/FDAC1/SI844.WAV |
| TEST/DR1/MREB0/SI2005.WAV | TEST/DR1/FELC0/SI2016.WAV |
| TEST/DR1/MRJO0/SX104.WAV | TEST/DR1/FJEM0/SX94.WAV |
| TEST/DR1/MSJS1/SX279.WAV | - |
| TEST/DR1/MSTK0/SA1.WAV | - |

Tabelle 4: Verzeichnisse der TIMIT-Datenbank der Sprachsignale der Testdaten

Die 3 Störsignale für die Testungen, mit jeweils 30 Sekunden Länge, bilden sich aus den Audiodateien in Tabelle 5. Dazu wird für jedes Störsignal jeweils eine Audiosequenz mit 10 Sekunden Länge aus Tabelle 5 drei mal hintereinander geschnitten.

| Nr. | Abschnitt in Sek - Sek | Bezeichnung |
|---|---|---|
| 5 | 0 - 10 | Fahrzeugmotor |
| 6 | 0 - 10 | Maschine |
| 8 | 11 - 21 | Kettensägen |

Tabelle 5: Zusammensetzung der Störsignale der Testdaten mit Nummerierung der Liste im Literaturverzeichnis [15].

## 6.2 Bewertung der Testergebnisse

In diesem Abschnitt werden alle signifikanten Testergebnisse für alle getesteten Systeme aufgelistet. In Tabelle 6 sind die verwendeten Kürzel zur Abweichung der Systemkonfiguration zu der Basis-Implementierung und den Variationen aufgelistet und beschrieben. Die Testergebnisse für weibliche Stimmen sind leider nicht vorhanden, da die Testungen zu viel Zeit in Anspruch genommen haben. Ebenso sind für das Training mit Phaseninformation nur die Testergebnisse der optimierten Konfiguration vorhanden und nicht die der Basiskonfiguration.

| Kürzel | Beschreibung |
|---|---|
| - | Keine Änderung zum Basissystem bzw. zur Variation |
| STFT2 | STFT-Schrittweite 2 |
| FFT256 | STFT-Fensterbreite 256 |
| SEQ40 | Abschnittslänge 40 |
| ReLU | Verwendung der ReLU-Aktivierungsfunktion in Ebene 1 und 2 |
| ELU | Verwendung der ELU-Aktivierungsfunktion in Ebene 1 und 2 |

Tabelle 6: Relation zwischen Kürzel und System-Parametern

Die Darstellung der Ergebnisse ist in 15 Tabellen mit jeweils allen Testergebnissen zu einer Maske und eines Störsignals aufgegliedert.

### Ergebnisse der SM im Training ohne Phase

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | 5.84 | 0.41 | 0.08 | - | - | - |
| STFT2FFT128SEQ20 | 5.85 | 0.43 | 0.09 | - | - | - |
| STFT2FFT128SEQ40 | 6.16 | 0.45 | 0.09 | - | - | - |
| STFT2FFT256SEQ40 | 6.93 | 0.48 | 0.11 | - | - | - |
| STFT2FFT256SEQ40ReLU | 3.71 | 0.11 | 0.07 | - | - | - |
| STFT2FFT256SEQ40ELU | 4.42 | 0.28 | 0.10 | - | - | - |

Tabelle 7: Ergebnisse der SM mit Training ohne Phase für ″Maschine″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | 4.41 | 0.24 | 0.04 | - | - | - |
| STFT2FFT128SEQ20 | 4.42 | 0.23 | 0.04 | - | - | - |
| STFT2FFT128SEQ40 | 4.57 | 0.24 | 0.04 | - | - | - |
| STFT2FFT256SEQ40 | 5.42 | 0.30 | 0.05 | - | - | - |
| STFT2FFT256SEQ40ReLU | -0.18 | -0.12 | -0.07 | - | - | - |
| STFT2FFT256SEQ40ELU | 1.52 | 0.06 | 0.00 | - | - | - |

Tabelle 8: Ergebnisse der SM mit Training ohne Phase für ″Fahrzeugmotor″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | 4.70 | 0.23 | 0.02 | - | - | - |
| STFT2FFT128SEQ20 | 4.90 | 0.21 | 0.03 | - | - | - |
| STFT2FFT128SEQ40 | 4.70 | 0.30 | 0.04 | - | - | - |
| STFT2FFT256SEQ40 | 4.99 | 0.34 | 0.04 | - | - | - |
| STFT2FFT256SEQ40ReLU | 4.66 | 0.09 | -0.01 | - | - | - |
| STFT2FFT256SEQ40ELU | 4.94 | 0.12 | 0.01 | - | - | - |

Tabelle 9: Ergebnisse der SM mit Training ohne Phase für ″Kettensägen″

### Ergebnisse der cIRM im Training ohne Phase

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | 3.21 | 0.34 | 0.07 | - | - | - |
| STFT2FFT128SEQ20 | 4.59 | 0.32 | 0.06 | - | - | - |
| STFT2FFT128SEQ40 | 5.50 | 0.32 | 0.06 | - | - | - |
| STFT2FFT256SEQ40 | 5.30 | 0.31 | 0.06 | - | - | - |
| STFT2FFT256SEQ40ReLU | 5.12 | 0.43 | 0.06 | - | - | - |
| STFT2FFT256SEQ40ELU | 5.16 | 0.47 | 0.07 | - | - | - |

Tabelle 10: Ergebnisse der cIRM mit Training ohne Phase für ″Maschine″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | 3.10 | 0.21 | 0.03 | - | - | - |
| STFT2FFT128SEQ20 | 4.64 | 0.28 | 0.04 | - | - | - |
| STFT2FFT128SEQ40 | 5.00 | 0.36 | 0.04 | - | - | - |
| STFT2FFT256SEQ40 | 4.90 | 0.41 | 0.03 | - | - | - |
| STFT2FFT256SEQ40ReLU | 4.08 | 0.27 | 0.02 | - | - | - |
| STFT2FFT256SEQ40ELU | 3.97 | 0.26 | 0.03 | - | - | - |

Tabelle 11: Ergebnisse der cIRM mit Training ohne Phase für ″Fahrzeugmotor″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | 1.87 | 0.22 | 0.01 | - | - | - |
| STFT2FFT128SEQ20 | 3.57 | 0.25 | 0.02 | - | - | - |
| STFT2FFT128SEQ40 | 4.33 | 0.28 | 0.02 | - | - | - |
| STFT2FFT256SEQ40 | 2.82 | 0.27 | 0.00 | - | - | - |
| STFT2FFT256SEQ40ReLU | 3.52 | 0.37 | -0.01 | - | - | - |
| STFT2FFT256SEQ40ELU | 3.71 | 0.36 | -0.00 | - | - | - |

Tabelle 12: Ergebnisse der cIRM mit Training ohne Phase für ″Kettensägen″

### Ergebnisse der cIRM im Training mit Phase

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | - | - | - | - | - | - |
| STFT2FFT128SEQ6 | 5.22 | 0.11 | 0.04 | - | - | - |

Tabelle 13: Ergebnisse der cIRM mit Training mit Phase für ″Maschine″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | - | - | - | - | - | - |
| STFT2FFT128SEQ6 | 5.24 | 0.44 | 0.05 | - | - | - |

Tabelle 14: Ergebnisse der cIRM mit Training mit Phase für ″Motor″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | - | - | - | - | - | - |
| STFT2FFT128SEQ6 | 4.86 | 0.30 | 0.03 | - | - | - |

Tabelle 15: Ergebnisse der cIRM mit Training mit Phase für ″Kettensägen″

### Ergebnisse der PSM im Training ohne Phase

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | 5.54 | 0.38 | 0.07 | - | - | - |
| STFT2FFT128SEQ20 | 5.74 | 0.42 | 0.08 | - | - | - |
| STFT2FFT128SEQ40 | 5.54 | 0.35 | 0.07 | - | - | - |
| STFT2FFT256SEQ40 | 6.24 | 0.37 | 0.07 | - | - | - |
| STFT2FFT256SEQ40ReLU | 5.75 | 0.44 | 0.08 | - | - | - |
| STFT2FFT256SEQ40ELU | 5.79 | 0.45 | 0.09 | - | - | - |

Tabelle 16: Ergebnisse der SM mit Training ohne Phase für ″Maschine″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | 4.55 | 0.28 | 0.03 | - | - | - |
| STFT2FFT128SEQ20 | 4.28 | 0.27 | 0.03 | - | - | - |
| STFT2FFT128SEQ40 | 4.76 | 0.35 | 0.03 | - | - | - |
| STFT2FFT256SEQ40 | 5.18 | 0.39 | 0.03 | - | - | - |
| STFT2FFT256SEQ40ReLU | 4.45 | 0.24 | 0.04 | - | - | - |
| STFT2FFT256SEQ40ELU | 4.28 | 0.24 | 0.04 | - | - | - |

Tabelle 17: Ergebnisse der SM mit Training ohne Phase für ″Fahrzeugmotor″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | 3.84 | 0.27 | 0.02 | - | - | - |
| STFT2FFT128SEQ20 | 4.37 | 0.31 | 0.03 | - | - | - |
| STFT2FFT128SEQ40 | 3.96 | 0.24 | 0.02 | - | - | - |
| STFT2FFT256SEQ40 | 4.04 | 0.31 | 0.01 | - | - | - |
| STFT2FFT256SEQ40ReLU | 4.07 | 0.37 | 0.00 | - | - | - |
| STFT2FFT256SEQ40ELU | 4.07 | 0.38 | 0.01 | - | - | - |

Tabelle 18: Ergebnisse der SM mit Training ohne Phase für ″Kettensägen″

### Ergebnisse der PSM im Training mit Phase

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | - | - | - | - | - | - |
| STFT1FFT128SEQ6 | 5.22 | 0.11 | 0.04 | - | - | - |

Tabelle 19: Ergebnisse der SM mit Training mit Phase für ″Maschine″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | - | - | - | - | - | - |
| STFT1FFT128SEQ6 | 5.24 | 0.44 | 0.05 | - | - | - |

Tabelle 20: Ergebnisse der SM mit Training mit Phase für ″Motor″

| Training und Test mit | männlichen Stimmen | | | weiblichen Stimmen | | |
|---|---|---|---|---|---|---|
| System-Kürzel | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* | Δ*SNR* in *dB* | Δ*PESQ* | Δ*STOI* |
| STFT1FFT128SEQ20 | - | - | - | - | - | - |
| STFT1FFT128SEQ6 | 4.86 | 0.30 | 0.03 | - | - | - |

Tabelle 21: Ergebnisse der SM mit Training mit Phase für ″Kettensägen″

## Beispielhafte Trainingsverläufe

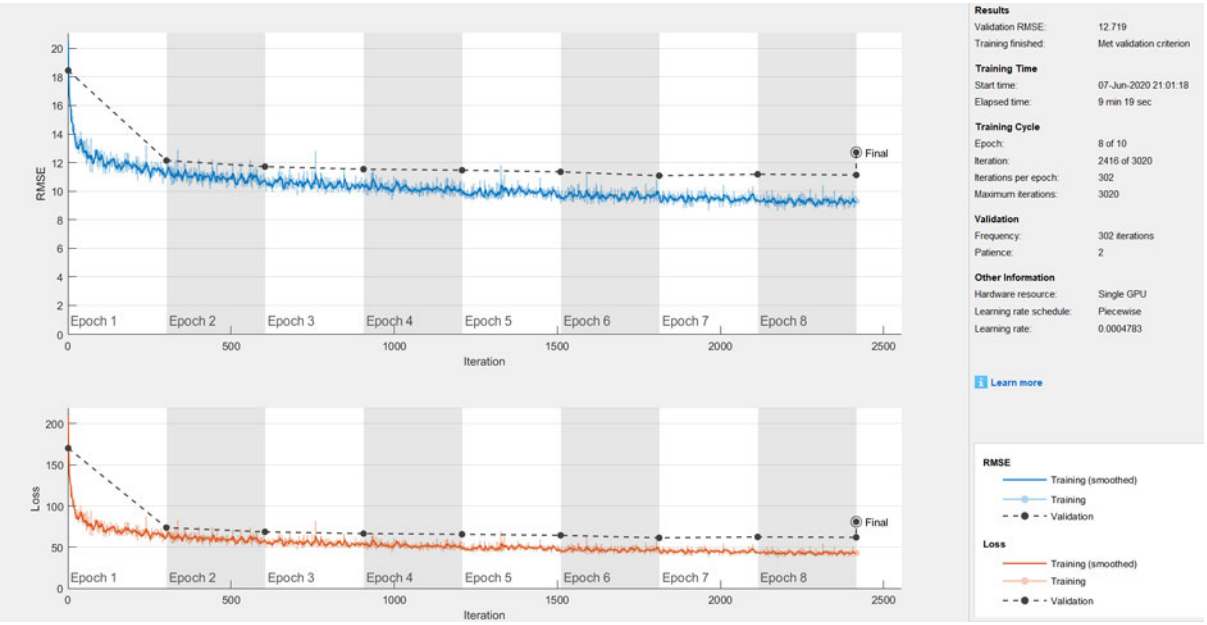


Abbildung 9: Exemplarischer Trainingsverlauf bei Maskierung mit der SM

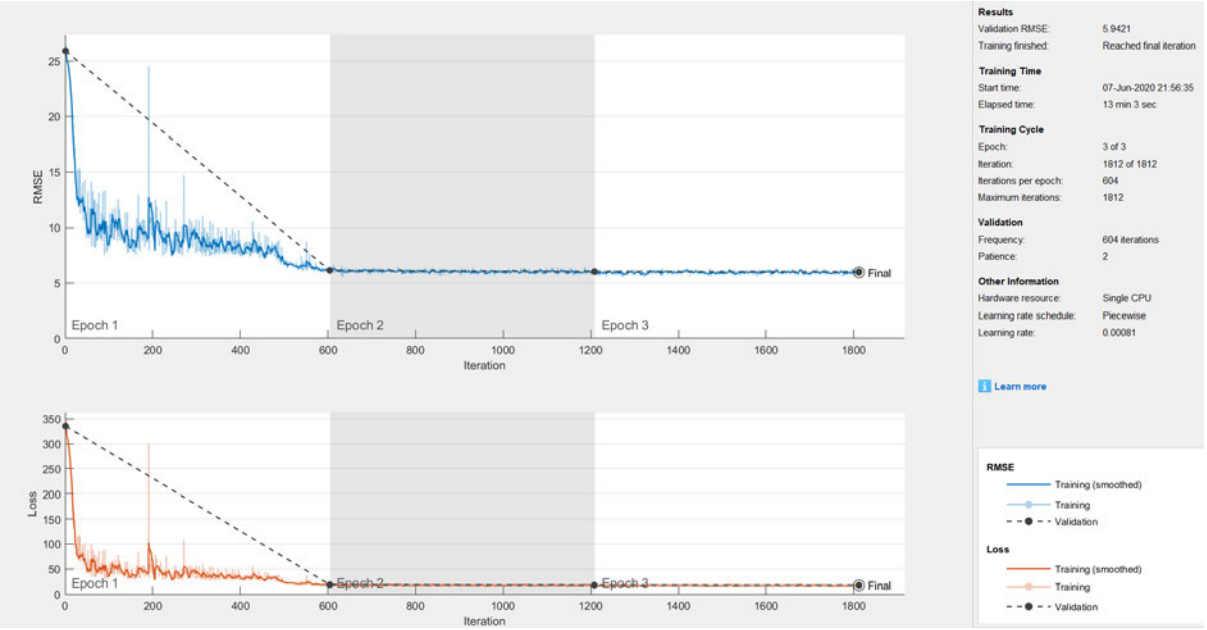


Abbildung 10: Exemplarischer Trainingsverlauf bei Maskierung mit der cIRM

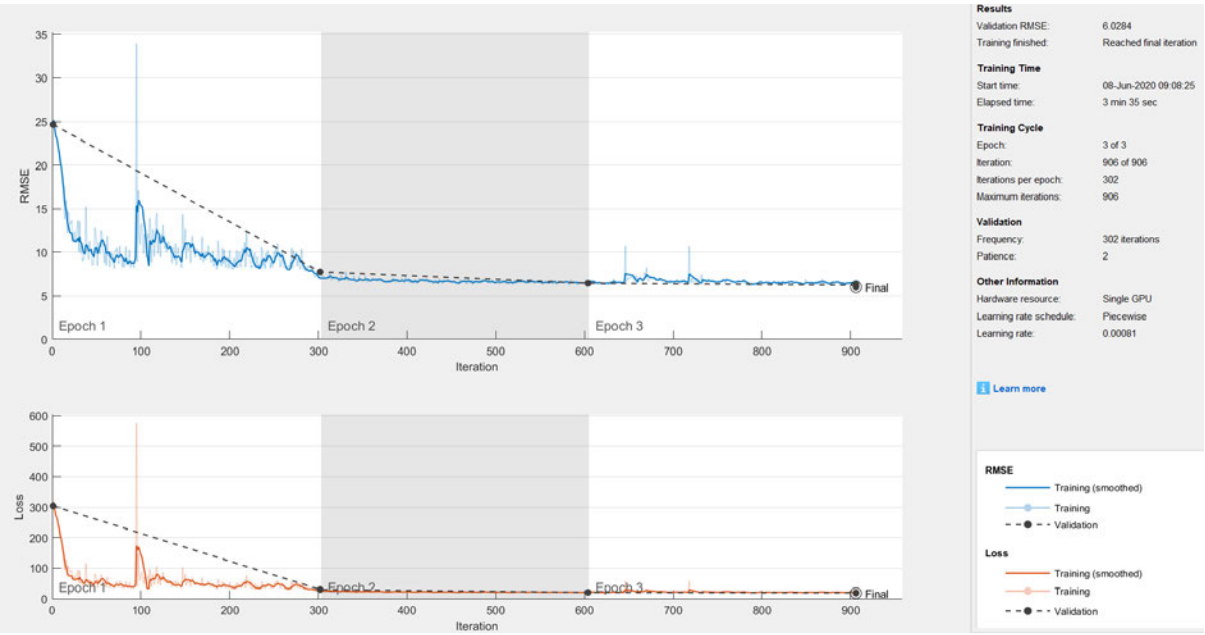


Abbildung 11: Exemplarischer Trainingsverlauf bei Maskierung mit der PSM

## 6.3 Auswertung und Vergleich

Zur Übersicht und zur Vergleichbarkeit sind die besten bzw. für die Auswertung relevanten Ergebnisse jeder Maske und Trainingsform in Abbildung 12, 13 und 14 grafisch dargestellt. Die Farben in den Balkendiagrammen beziehen sich wie folgt auf die Störsignale. Blau für Maschine, Orange für Motor, Grau für Kettensägen und Gelb für den Durchschnitt. Das Kürzel ″Comp″steht hierbei für das Training mit Phaseninformation.

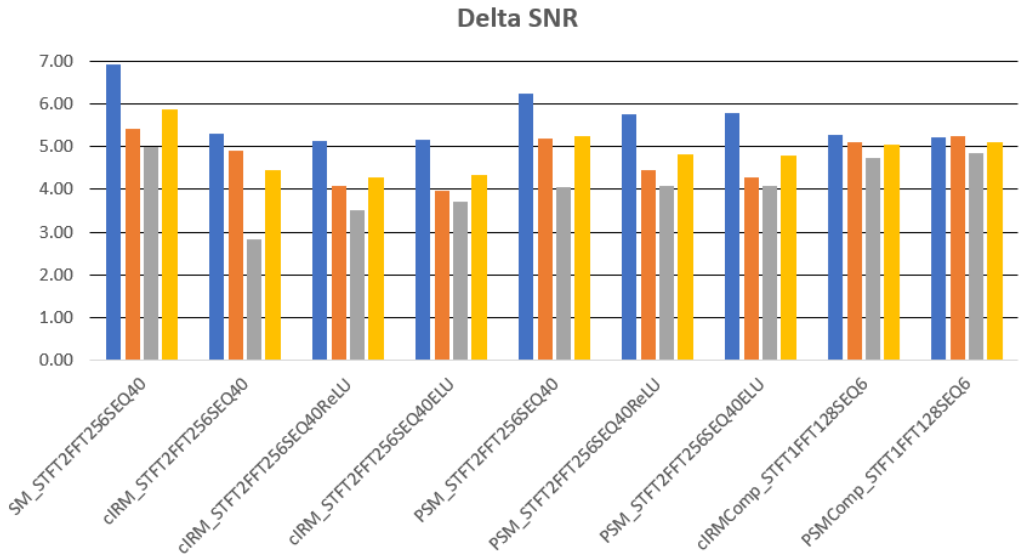


Abbildung 12: Übersichtsgrafik der besten SNR-Werte aller Masken und Trainingsformen

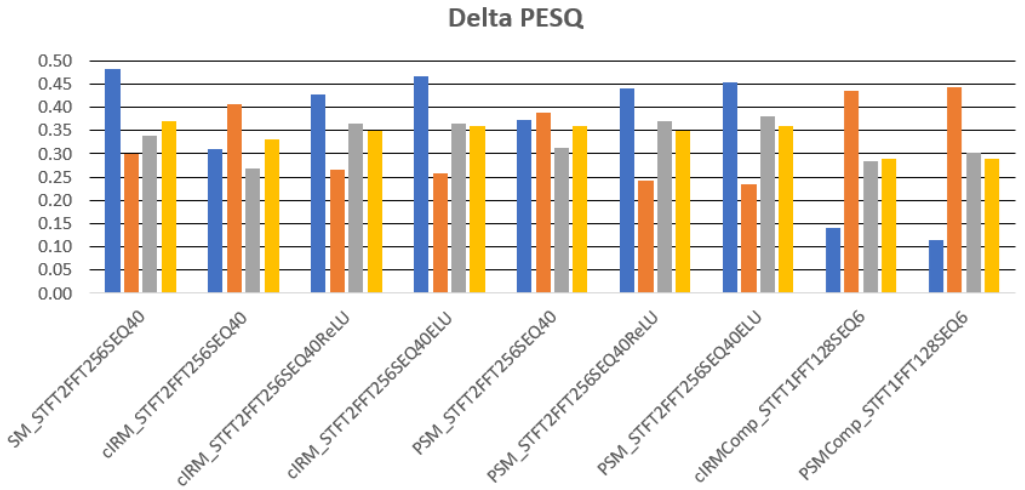


Abbildung 13: Übersichtsgrafik der besten PESQ-Werte aller Masken und Trainingsformen

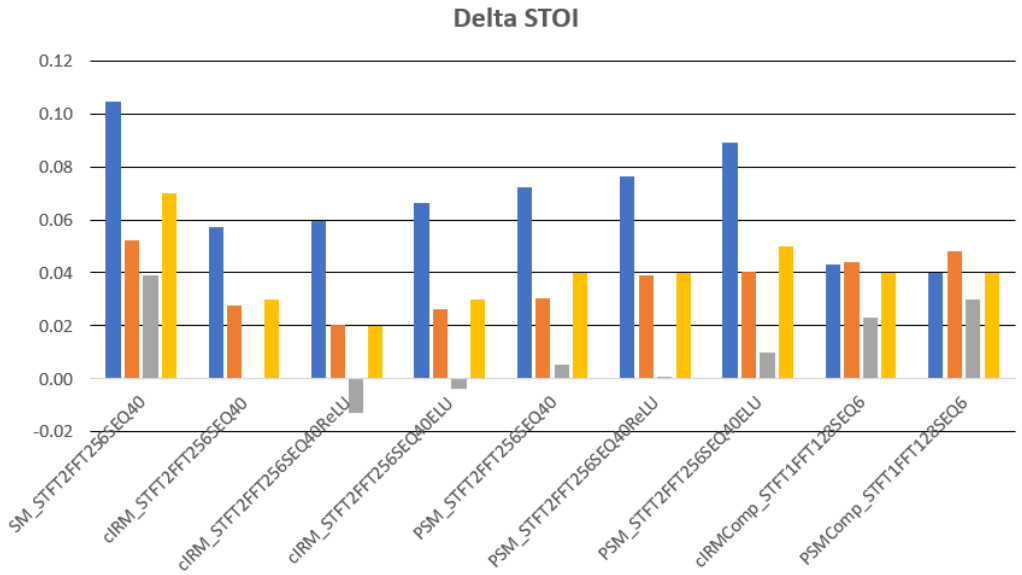


Abbildung 14: Übersichtsgrafik der besten STOI-Werte aller Masken und Trainingsformen

Generell lässt sich beobachten, dass die Entstörung der Mischsignale je nach Störgeräusch über alle Messparameter sehr unterschiedlich gut ausfällt und deshalb der Durchschnitt der Messergebnisse über die verschiedenen Störgeräusche keine bedeutende Aussage über die Qualität eines Systems im Einsatz in einer spezifischen Störgeräuschumgebung liefert. Für die Bewertung eines Systems mit dem Anspruch der Generalisiertheit sollte das System mit deutlich mehr Störgeräuschen getestet werden als es in dieser Arbeit getan wird.
In allen Messparametern und Masken lässt sich das Störgeräusch der Maschine am besten dämpfen. Insbesondere die SM lieferte in dieser Hinsicht die besten Ergebnisse.
Komplizierter ist die Bewertung der Testergebnisse für das Störgeräusch des Motors. Im SNR und STOI liefert die beste Konfiguration der SM die besten Ergebnisse für das Störgeräusch des Motors. Im PESQ jedoch dominierten die cIRM und die PSM im Training mit Phaseninformation, welche ebenfalls sehr gute bzw. zur SM sehr ähnliche Werte im SNR und STOI liefern. Gute Werte im PESQ und im SNR werden ebenfalls von der PSM im Training ohne Phaseninformation erreicht. Je nach Priorität ist diese Variante also gegenüber der PSM im Training ohne Phaseninformation zu bevorzugen, da diese deutlich weniger Rechenaufwand mit sich zieht.
Das Störgeräusch der Kettensägen lässt sich im SNR und STOI am besten von der SM dämpfen. Im PESQ jedoch liefert eine Konfiguration der PSM die besten Ergebnisse. Hier muss ebenfalls eine Priorisierung gewählt werden.

Die Sigmoid-Aktivierungsfunktion liefert bei allen Masken die Messergebnisse mit der geringsten Varianz über die verschiedenen Störgeräusche.

Die optimalen Konfigurationswerte unterscheiden sich stark zwischen dem Training ohne Phaseninformation und dem Training mit Phaseninformation. Während das Training ohne Phaseninformation bei einer Abschnittslänge von 40 und einer STFT-Schrittweite von 2 optimal ist, so ist die optimale Abschnittslänge beim Training mit Phaseninformation bei 6 mit der STFT-Schrittweite 1.

Die SM und die PSM profitieren stark von einer Erhöhung der FFT-Auflösung von Fensterbreite 128 auf 256, während die cIRM sich dadurch im SNR und STOI verschlechtert.

Abschließend lässt sich beurteilen, dass die cIRM sowohl im Training ohne Phaseninformation, als auch im Training mit Phaseninformation gegenüber der SM und der PSM, aufgrund einer eher schlechten Leistung bei hohem Rechenaufwand, abfällt. Je nach vorherrschendem Störsignal und Priorität bezüglich Qualitätsmerkmalen ist die PSM im Training mit oder ohne Phaseninformation der SM vorzuziehen. In den meisten Fällen jedoch punktet die SM mit den besten Messwerten.
Aufgrund des schnellen Erreichens eines Plateaus und des Ausbleiben von Overfitting im Training der cIRM und der PSM lässt sich jedoch vermuten, dass diese Masken bei einer geeigneteren Wahl der Trainingsdaten oder anderer Systemparameter, bessere Ergebnisse erzielen könnten.

# 7 Fazit

### Zusammenfassung

Es wurden nicht alle Ziele dieser Arbeit erreicht, jedoch ließen sich einige Erkenntnisse aus den Testergebnissen ableiten. Die cIRM fällt im Allgemeinen gegenüber den beiden anderen Masken SM und PSM in ihrer Leistungsfähigkeit ab. Es stellte sich also heraus, dass der Fokus auf relevante Informationen durch eine geeignete Maskenwahl effektiver sein kann, als die Verwendung einer Maske, welche die gesamten Informationen abbildet (cIRM).
Das Training mit Phaseninformation sollte zwar aufgrund des hohen Rechenaufwands mit Bedacht eingesetzt werden, konnte jedoch für bestimmte Störgeräuschumgebungen eine Verbesserung gegenüber dem Training ohne Phaseninformation erreichen.
Der Umstand, dass im Trainingsprozess der cIRM und der PSM sehr früh ein Plateau erreicht wurde, ließ vermuten, dass das Potential dieser Masken nicht ausgeschöpft werden konnte.
Für die nötigen Testungen der Untersuchung der Unterschiede in der Trainierbarkeit von männlichen und weiblichen Stimmen fehlte leider die Zeit.

### Aussicht

Es hat sich gezeigt, dass neuronale Netze in der Störgeräuschreduzierung sehr gute Ergebnisse erzielen können. Dementsprechend kann auf Basis der vorgestellten Konzepte ein echtzeitfähiges System zur Störgeräuschreduzierung entwickelt werden.
In der Regel bietet es sich an das System auf eine spezielle Störgeräuschumgebung zu trainieren. Um trotzdem ein generalisiertes System zu erhalten, könnten je nach Leistungsbeanspruchung mehrere solche speziell trainierter Systeme in ein System integriert werden, welches eine Störgeräuschumgebung erkennt und das wirksamste System zur Störgeräuschreduzierung adaptiv selektiert und auf die gestörte Umgebung anwendet.

# 8 Literatur

[1] Andrew J.R. Simpson *Probabilistic Binary-Mask Cocktail-Party Source Separation in a Convolutional Deep Neural Network*.
University of Surrey, Erhältlich von: arXiv.org
[Zugriff am 11. Februar 2020]

[2] Andrew J.R. Simpson *Abstract Learning via Demodulation in a Deep Neural Network*.
University of Surrey, Erhältlich von: arXiv.org
[Zugriff am 31. Mai 2020]

[3] Williamson, Donald S. / DeLiang Wang *Time-Frequency Masking in the Complex Domain for Speech Dereverberation using Denoising*.
Erhältlich von: web.cse.ohio-state.edu/~wang.77/papers/Williamson-Wang.taslp17.pdf
[Zugriff am 11. Februar 2020]

[4] MathWorks *Cocktail Party Source Separation Using Deep Learning Networks*.
MathWorks, Erhältlich von: mathworks.com
[Zugriff am 29. Mai 2020]

[5] Kruse, Rudolf / Christian Borgelt / Frank Klawonn / Christian Moewes / Georg Ruß / Matthias Steinbrecher
(2011): *Computational Intelligence*, Deutschland: Vieweg+Teubner Verlag | Springer Fachmedien Wiesbaden GmbH

[6] Alfred Mertins
(2020): *Signaltheorie*, 4. u. überarb. Aufl., Deutschland, Wiesbaden: Springer Vieweg

[7] MathWorks *Matlab Syntax-Beschreibung: trainingOptions*.
MathWorks, Erhältlich von: mathworks.com/help/deeplearning/ref/trainingoptions.html
[Zugriff am 29. Mai 2020]

[8] MathWorks *Matlab NN-Ebenen*.
MathWorks, Erhältlich von: de.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html
[Zugriff am 06. Juni 2020]

[9] OPTICOM GmbH *PESQ*
OPTICOM GmbH, Erhältlich von: opticom.de/technology/pesq.php
[Zugriff am 31. Mai 2020]

[10] Taal, Cees H. / Richard C. Hendriks / Richard Heusdens / Jesper Jensen
*An Algorithm for Intelligibility Prediction of Time-Frequency Weighted Noisy Speech*
-, Erhältlich von: cas.et.tudelft.nl/pubs/Taal2011_1.pdf
[Zugriff am 31. Mai 2020]

[11] Arkadiy Prodeus *pesq2_mtlb.m* Erhältlich von: mathworks.com/matlabcentral/fileexchange/47333-pesq-matlab-driver?w.mathworks.com
[Zugriff am 31. Mai 2020]

[12] Jacob Donley *pesq_NoResFile.exe* Erhältlich von: github.com/JacobD10/SoundZone_Tools/blob/master/pesq_NoResFile.exe
[Zugriff am 31. Mai 2020]

[13] Delft University of Technology *stoi.m*
Delft University of Technology, Signal & Information Processing Lab, Erhältnlich von: github.com/JacobD10/SoundZone_Tools/blob/master/stoi.m
[Zugriff am 31. Mai 2020]

[14] Garofolo, John S., et al. *TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1*. Web Download. Philadelphia: Linguistic Data Consortium, 1993

[15] Lewente / coneybeare / felix.blume / willybilly1984 / Filmscore / Ferdinger / ModulationStation / reklamacja / Nizerg / senorstudy / HonorHunter / Jack_Master / radwoc / DrZoom / shaderrow / blukotek / DCElliott / LeCASUAL / freesoundjon01 / NLM / KToppMod / ecodios / biholao / jone_oost / gadzooks / mma_official / jordir / ivolipa / daveincamas / ptrflr / esperri / ivolipa / Cobus190244
*Audiodateien im WAV-Format*
Erhältlich von:
freesound.org/people/Lewente/sounds/393398/
freesound.org/people/coneybeare/sounds/91129/
freesound.org/people/felix.blume/sounds/133991/
freesound.org/people/willybilly1984/sounds/345336/
freesound.org/people/Filmscore/sounds/268526/
freesound.org/people/Ferdinger/sounds/146261/
freesound.org/people/ModulationStation/sounds/132851/
freesound.org/people/reklamacja/sounds/446157/
freesound.org/people/Nizerg/sounds/232995/
freesound.org/people/senorstudy/sounds/437286/
freesound.org/people/HonorHunter/sounds/271662/
freesound.org/people/Jack_Master/sounds/202409/
freesound.org/people/radwoc/sounds/256807/
freesound.org/people/DrZoom/sounds/185990/

freesound.org/people/shaderrow/sounds/437202/
freesound.org/people/blukotek/sounds/424940/
freesound.org/people/DCElliott/sounds/381453/
freesound.org/people/LeCASUAL/sounds/442414/
freesound.org/people/freesoundjon01/sounds/322202/
freesound.org/people/NLM/sounds/178314/
freesound.org/people/KToppMod/sounds/328141/
freesound.org/people/ecodios/sounds/119977/
freesound.org/people/biholao/sounds/370276/
freesound.org/people/jone_oost/sounds/366706/
freesound.org/people/gadzooks/sounds/22855/
freesound.org/people/mma_official/sounds/351783/
freesound.org/people/jordir/sounds/372872/
freesound.org/people/ivolipa/sounds/337446/
freesound.org/people/daveincamas/sounds/495060/
freesound.org/people/ptrflr/sounds/397883/
freesound.org/people/esperri/sounds/118972/
freesound.org/people/ivolipa/sounds/345995/
freesound.org/people/Cobus190244/sounds/492836/

[16] CarlosCarty
*Aufnahme einer Panflöte*
Erhältlich von:
freesound.org/people/CarlosCarty/sounds/454883/

# A Anhang

## A.1 Beigefügte CD

Auf einer beigefügten CD befinden sich diese Arbeit im .pdf-Format, der gesamte verwendete Matlab-Code und die verwendeten externen Skripts.

Hiermit versichere ich, Sven Erik Ehlers, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort: Datum: Unterschrift: