Bachelorarbeit in der Informatik

# Trennung von Musikquellen mit TFC-TDF U-Net Modellen

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Methoden des Maschinellen Lernens

Adriano Piu, (Matrikelnummer: 5966275)
adriano.piu@student.uni-tuebingen.de,

Bearbeitungszeitraum: von 01.02.2025 - bis 02.06.2025

Gutachter: Prof. Dr. Philipp Hennig, Universität Tübingen

# Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich diese schriftliche Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

---

Adriano Piu (Matrikelnummer 5966275), 02. Juni 2025

# Contents

# 1 Einleitung

Die Musikquellentrennung ist ein aktives Forschungsfeld an der Schnittstelle zwischen Signalverarbeitung und maschinellem Lernen. Ziel ist es, aus einem komplexen Musiksignal einzelne Bestandteile, wie Gesang, Schlagzeug, Bass oder weitere Instrumente, als isolierte Spuren zu extrahieren. Diese Technologie bietet zahlreiche praktische Anwendungsmöglichkeiten. Sie kann unter anderem in der Musikproduktion zum Remixen oder Samplen, zur Unterstützung in der Musikpädagogik, zum Beispiel zum instrumentenspezifische Üben, eingesetzt werden. Zudem ermöglicht sie die automatische Erstellung von Karaoke-Versionen und kann in der Musikforschung zur Transkription oder Analyse musikalischer Strukturen genutzt werden.

Trotz ihrer Vielseitigkeit stellt die Musikquellentrennung eine technisch äußerst anspruchsvolle Aufgabe dar. Musiksignale bestehen aus mehreren überlagerten Quellen, die sich sowohl zeitlich als auch spektral stark überlappen. Zusätzlich erschwert die hohe Variabilität, etwa durch unterschiedliche Instrumentierungen, Aufnahmetechniken oder stilistische Eigenheiten, die Trennung einzelner Komponenten. Klassische signalverarbeitende Verfahren stoßen hier schnell an ihre Grenzen. Aus diesem Grund kommen zunehmend Methoden des maschinellen Lernens zum Einsatz. Neuronale Netze sind in der Lage, durch Training auf großen Mengen an Audiomaterial charakteristische Muster einzelner Klangquellen zu erkennen und voneinander zu unterscheiden.

Im Rahmen dieser Arbeit wird das Ziel der Musikquellentrennung mithilfe des Modells **TFC-TDF-UNet V3** verfolgt. Das Modell kombiniert Time-Frequency Convolutions (TFC) mit Time-Distributed Fully Connected (TDF) Layern, um zeitlich-frequente Strukturen in Musikdaten besonders effektiv zu erfassen.

TFC-TDF-UNet V3 wurde als Beitrag zur *Sound Demixing Challenge 2023* entwickelt und erzielte dort auf mehreren Leaderboards hervorragende Ergebnisse. Als typischer Challenge-Beitrag liegt jedoch keine vollständige, zusammenhängende Dokumentation der Modellarchitektur oder des Trainingsprozesses vor. Viele Implementierungsdetails sind nur im Code ersichtlich oder kaum beschrieben. Dies erschwert sowohl die wissenschaftliche Nachvollziehbarkeit als auch die Nutzung erheblich.

Darüber hinaus zeichnet sich das Modell durch einen hohen Trainingsaufwand aus. Das ursprünglich vorgesehene Training über mehrere Checkpoints und große

Datenmengen ist zeitintensiv und nutzt in der vorhandenen Implementierung die verfügbare Rechenleistung nicht effizient aus.

Vor diesem Hintergrund verfolgt die vorliegende Bachelorarbeit drei zentrale Ziele: Zum einen soll die Modellarchitektur vollständig rekonstruiert und dokumentiert werden, um Klarheit und Reproduzierbarkeit zu schaffen. Dadurch soll eine fundierte Grundlage für wissenschaftliche Analysen und weiterführende Experimente entstehen. Zum anderen wird der Trainingsprozess modularisiert und systematisch reproduzierbar gemacht, sodass unterschiedliche Modellvarianten effizient getestet und verglichen werden können. Schließlich wird die Trainingspipeline gezielt, unter anderem durch die Einführung von Parallelisierungsstrategien wie Distributed Data Parallel (DDP), optimiert, um die Trainingsdauer signifikant zu reduzieren, ohne dabei Leistungseinbußen hinzunehmen.

Diese Arbeit leistet somit einen Beitrag zur verbesserten Nachvollziehbarkeit, Nutzbarkeit und Skalierbarkeit eines aktuellen Deep-Learning-Modells im Bereich der Musikquellentrennung.

# 2 Stand der Technik

## 2.1 Neuronale Netze und Deep Learning

### 2.1.1 Neuronale Netze

Ein neuronales Netz [LBH15] ist ein rechnerisches Modell, das vom Aufbau und der Funktionsweise des menschlichen Denkens inspiriert ist. Es besteht aus vielen miteinander verbundenen künstlichen Neuronen, die in Schichten (Layern) organisiert sind. Die grundlegende Struktur eines neuronalen Netzes umfasst mindestens drei Schichten: eine Eingabeschicht, eine oder mehrere verborgene Schichten (Hidden Layers) und eine Ausgabeschicht. Jedes Neuron empfängt Signale von Neuronen der vorherigen Schicht, gewichtet diese mit trainierbaren Parametern (Gewichten), summiert sie, wendet eine Aktivierungsfunktion an und leitet das Ergebnis an die nächste Schicht weiter. Auf diese Weise lassen sich komplexe, nichtlineare Zusammenhänge in den Eingabedaten modellieren.

Tiefe neuronale Netze nutzen die Eigenschaft aus, dass viele natürliche Signale kompositionelle Hierarchien sind, in denen höherstufige Merkmale durch die Komposition niedrigerstufiger Merkmale gewonnen werden. In Bildern bilden lokale Kombinationen von Kanten Motive, Motive setzen sich zu Teilen zusammen und Teile bilden Objekte. Ähnliche Hierarchien existieren in Sprache und Text von Lauten zu Silben, Wörtern und Sätzen.

Die häufigste Form des maschinellen Lernens ist das Supervised Learning (überwachtes Lernen). Bei dieser Methode wird der Maschine ein großer Datensatz von Beispielen gezeigt, die jeweils mit der korrekten Kategorie oder dem gewünschten Output markiert sind. Während des Trainings wird ein Input gezeigt, und die Maschine produziert einen Output, zum Beispiel in Form eines Vektors von Scores für verschiedene Kategorien. Ziel ist es, dass die gewünschte Kategorie den höchsten Score erzielt. Eine objektive Funktion misst den Fehler (Loss) zwischen den Output-Scores und dem gewünschten Muster der Scores. Die Maschine passt dann ihre internen einstellbaren Parameter an, um diesen Fehler zu reduzieren.

Um die Gewichte korrekt anzupassen, berechnet der Lernalgorithmus einen Gradientenvektor. Der Gradient einer skalaren Funktion $f : \mathbb{R}^n \to \mathbb{R}$, z. B. einer Kostenfunktion in einem neuronalen Netz, ist ein Vektor, der alle partiellen Ableitungen von $f$ bezüglich ihrer Eingabevariablen enthält:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n$$

Der Gradient zeigt die Richtung des steilsten Anstiegs der Funktion $f$ an und wird in neuronalen Netzen verwendet, um die Änderung der Kostenfunktion in Bezug auf die Modellparameter zu berechnen. Dieser Gradientenvektor gibt für jedes Gewicht an, um wie viel der Fehler zunehmen oder abnehmen würde, wenn das Gewicht leicht erhöht würde. Die Gewichte werden dann in die entgegengesetzte Richtung des Gradientenvektors angepasst, um den Fehler zu reduzieren.

In der Praxis wird oft die Methode des Stochastic Gradient Descent (SGD) [HRS16] verwendet. Dabei wird der Input-Vektor für eine kleine Anzahl von Beispielen gezeigt, die Outputs und Fehler werden berechnet, der durchschnittliche Gradient für diese Beispiele wird ermittelt und die Gewichte entsprechend angepasst. Dieser Prozess wird für viele kleine Sätze von Beispielen aus dem Trainingssatz wiederholt, bis die objektive Funktion im Durchschnitt nicht mehr abnimmt. SGD findet trotz seiner Stochastizität oft überraschend schnell gute Gewichte. Nach dem Training wird die Leistung des Systems auf einem separaten Testset gemessen, um seine Generalisierungsfähigkeit, also die Fähigkeit, sinnvolle Antworten auf neue Inputs zu geben, die es während des Trainings noch nie gesehen hat, zu überprüfen.

Der Algorithmus zur Berechnung der Gradienten für die Gewichte eines Multilayer-Netzwerks ist die Backpropagation [RHW86]. Die Idee ist, dass der Gradient der objektiven Funktion in Bezug auf den Input eines Moduls berechnet werden kann, indem man vom Gradienten in Bezug auf den Output dieses Moduls (oder den Input des nachfolgenden Moduls) rückwärts arbeitet. Die Backpropagation-Gleichung kann wiederholt angewendet werden, um Gradienten durch alle Module zu propagieren, beginnend vom Output an der Spitze, wo das Netzwerk seine Vorhersage trifft, bis ganz nach unten, wo der externe Input eingespeist wird.

### 2.1.2 Tiefere Netze schaffen (Deep Learning)

Die Erweiterung neuronaler Netze durch zusätzliche Schichten ist ein zentrales Konzept im Deep Learning. Durch zunehmende Tiefe können neuronale Netze immer komplexere Funktionen approximieren und schrittweise höhere Abstraktionsebenen aus den Eingabedaten lernen. Beispielsweise können in tieferen Netzwerken einfache Kantenmuster in frühen Schichten zu komplexen Formen oder sogar semantischen Konzepten in späteren Schichten zusammengesetzt werden. Doch mit zunehmender Tiefe wachsen auch die Herausforderungen beim Training solcher Modelle erheblich. Beim Training tiefer Netze wird der Loss über Backpropagation

schichtweise rückwärts durch das Netzwerk propagiert. In sehr tiefen Architekturen kann es dabei zu zwei Problemen kommen. Beim Gradient Vanishing werden die Gradienten in frühen Schichten zunehmend kleiner, sodass die Gewichte kaum noch angepasst werden. Das Netz lernt in diesen Bereichen nur sehr langsam oder gar nicht mehr [PMB13]. Umgekehrt können bei der Gradient Explosion die Gradienten exponentiell anwachsen, was zu instabilen Gewichtsaktualisierungen und letztlich zum Abbruch des Trainings führen kann. Diese Probleme behindern die effiziente Optimierung tiefer Modelle und waren lange ein limitierender Faktor bei der Entwicklung tiefer Netzarchitekturen.

Eine zentrale Innovation zur Stabilisierung sehr tiefer Netze sind Residual Networks (ResNets) [HZRS15a]. ResNets umgehen das Gradient-Vanishing-Problem durch sogenannte Shortcut Connections, bei denen der Input einer Schicht direkt an spätere Schichten weitergeleitet wird. Formal wird nicht eine Funktion $f(x)$ gelernt, sondern eine Residualfunktion $f(x) = H(x) - x$, sodass das Netzwerk effektiv lernt:

$$H(x) = f(x) + x$$

Diese einfache Änderung ermöglicht das Training von Netzen mit hunderten Schichten und hat sich als Standardansatz für sehr tiefe Architekturen etabliert.

Neben architektonischen Anpassungen sind auch geeignete Optimierungsverfahren entscheidend für den erfolgreichen Trainingsverlauf. Der Adam-Optimizer [KB17] kombiniert Vorteile der adaptiven Lernratenregelung mit Momentum-basierten Verfahren. Adam berechnet für jedes Gewicht individuell angepasste Lernraten auf Basis gleitender Mittelwerte von Gradienten und deren Quadraten. Dies führt zu schnellerer Konvergenz, besonders bei verrauschten oder hochdimensionalen Daten, wie sie bei Audio- oder Bilddaten typisch sind.

In Deep-Learning-Modellen stellen Aktivierungsfunktionen ein zentrales Element dar, um nicht-lineare Zusammenhänge in Daten zu modellieren. Ihre Hauptaufgabe besteht darin, die linearen Transformationen, die durch die gewichteten Verbindungen innerhalb eines neuronalen Netzwerks entstehen, durch nicht-lineare Operationen zu ergänzen. Erst durch diese Nicht-Linearität wird es dem Netzwerk ermöglicht, komplexe Funktionen zu approximieren und tiefere Architekturen mit höherer Ausdruckskraft zu realisieren. Die am häufigsten verwendete Aktivierungsfunktion ist die Rectified Linear Unit (ReLU) [Aga19]

$$\mathrm{ReLU}(x) = \max(0, x)$$

Sie gibt den Eingabewert $x$ zurück, falls dieser positiv ist. Andernfalls ist die Ausgabe 0.

Alternativen wie die Gaussian Error Linear Unit (GELU) bieten weichere Übergänge und können je nach Anwendungsfall Vorteile in der Modellierung bieten [HG23].

$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

wobei $\Phi(x)$ die kumulative Verteilungsfunktion (CDF) der Standardnormalverteilung ist:

$$\Phi(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

Daraus ergibt sich:

$$\mathrm{GELU}(x) = \frac{x}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

Alternativ kann GELU auch durch eine Näherung ausgedrückt werden:

$$\mathrm{GELU}(x) \approx 0.5x\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right]\right)$$

Durch den Einsatz solcher nicht-linearer Funktionen wird verhindert, dass das gesamte Netzwerk lediglich eine lineare Abbildung bleibt. Die Nicht-Linearität verleiht dem Modell seine Ausdrucksstärke und befähigt es dazu, komplexe Entscheidungsgrenzen zu lernen.

Die Initialisierung der Netzwerkgewichte ist ein weiterer kritischer Faktor beim Training tiefer Netze. Eine zu kleine Initialisierung kann zum Gradient Vanishing führen, eine zu große zum Gradient Explosion. Methoden wie die Xavier-Initialisierung [GB10] oder He-Initialisierung wählen die Anfangswerte der Gewichte so, dass die Varianz der Ausgaben in jeder Schicht konstant bleibt, was die Stabilität des Trainings erhöht. Die He-Initialisierung [HZRS15b] wurde speziell für Netze mit Rectified Linear Unit (ReLU) Aktivierungsfunktionen entwickelt und zielt darauf ab, die Varianz der Ausgangssignale über alle Schichten hinweg konstant zu halten.

Mathematisch erfolgt die He-Initialisierung der Gewichte $W$ einer Schicht durch Ziehen aus einer Zufallsverteilung mit folgender Varianz:

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{\text{in}}}}\right),$$

wobei $n_{\text{in}}$ die Anzahl der eingehenden Verbindungen (bzw. Neuronen der vorherigen Schicht) bezeichnet. Die Wahl der Varianz $\frac{2}{n_{\text{in}}}$ ist darauf zurückzuführen, dass ReLU-Aktivierungen den Erwartungswert und die Varianz der Ausgaben asymmetrisch beeinflussen, da sie alle negativen Eingabewerte auf null setzen.

Die He-Initialisierung wird typischerweise bei Netzwerken eingesetzt, die ReLU oder Varianten wie Leaky ReLU als Aktivierungsfunktion verwenden. Für Netzwerke mit symmetrischen Aktivierungsfunktionen wie *tanh* oder *sigmoid* wird hingegen meist die Xavier-Initialisierung bevorzugt.

Ein weiterer Stabilitätsmechanismus ist die Normalisierung der Zwischenwerte im Netzwerk. Besonders verbreitet ist die Batch Normalization [IS15], bei der die Ausgabe einer Schicht normalisiert wird, bevor sie an die nächste Schicht

weitergegeben wird. Dies beschleunigt das Training, wirkt regulierend und reduziert das Risiko von Gradient Vanishing/Explosion zusätzlich. Für jede Aktivierung x berechnet sich die normalisierte Ausgabe $\hat{x}$ wie folgt:

$$\hat{x} = \frac{x - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}}} + \varepsilon}$$

Dabei sind $\mu_{\text{batch}}$ und $\sigma_{\text{batch}}$ der Mittelwert und die Varianz des aktuellen Mini-Batches, und $\varepsilon$ ist eine kleine Konstante zur Vermeidung von Division durch Null. Zusätzlich werden zwei lernbare Parameter $\gamma$ (Skalierung) und $\beta$ (Verschiebung) eingeführt, um der Normalisierung Flexibilität zu geben:

$$y = \gamma\hat{x} + \beta$$

In neueren Ansätzen werden auch Instance Normalization [UVL17] eingesetzt. Diese wurde ursprünglich für Stiltransferaufgaben eingeführt und normalisiert jede Trainingsinstanz einzeln. Sie berechnet für jede Instanz und jeden Kanal eigene Mittelwerte und Varianzen. Die Formel ist analog zur Batch Normalization, aber die Mittelwerte und Varianzen beziehen sich auf räumliche Dimensionen innerhalb einer einzelnen Probe:

$$\hat{x}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2}}$$

Auch hier erfolgt anschließend eine Transformation mit $\gamma$ und $\beta$. Instance Normalization ist besonders nützlich bei Aufgaben mit stark variierenden Stil- oder Intensitätsmustern, da sie unabhängig von der Batch-Größe und anderen Beispielen funktioniert.

Das Tiefermachen neuronaler Netze eröffnet enormes Potenzial zur Leistungssteigerung, insbesondere bei komplexen Aufgaben wie der Musikquellentrennung. Um jedoch mit sehr tiefen Architekturen erfolgreich arbeiten zu können, ist ein Zusammenspiel aus architektonischen Innovationen (wie ResNets), stabilen Optimierungsverfahren (z.B. Adam), sinnvoller Gewichtinitialisierung und effektiver Normalisierung unerlässlich.

### 2.1.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (auch ConvNets oder CNNs) [LBBH98] verbinden die Struktur klassischer neuronaler Netze mit der mathematischen Operation der Konvolution (auch Faltung genannt). Sie sind speziell dafür konzipiert, Daten zu verarbeiten, die in Form mehrdimensionaler Arrays vorliegen. Beispiele hierfür sind Farbbilder, die typischerweise als drei 2D-Arrays (einer pro Farbkanal) dargestellt werden, Audiosignale in Form von 1D-Zeitreihen oder Spektrogramme als 2D-Daten. Auch Videos können als 3D-Arrays modelliert werden, wobei zwei Dimensionen den Raum und eine Dimension die Zeit repräsentieren.

Eine zentrale Komponente von CNNs sind die Faltungsschichten (Convolutional Layers), die speziell zur Extraktion lokaler Merkmale aus strukturierten Daten wie Bildern oder Audiosignalen entwickelt wurden. Dabei wird eine Filterbank (Kernel) über das Eingabefeld bewegt und mittels Faltung auf kleine lokale Bereiche angewendet. Die diskrete 2D-Faltung einer Eingabe $X$ mit einem Filter $K$ kann formal wie folgt beschrieben werden:

$$S(i,j) = (X * K)(i,j) = \sum_m \sum_n X(i+m, j+n) \cdot K(m,n)$$

Das Ergebnis dieser Faltung ist eine neue Feature Map, die lokale Muster oder Strukturen wie Kanten oder Texturen extrahiert.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 20 & 30 & 40 & 0 \\ 0 & 50 & 60 & 70 & 80 & 0 \\ 0 & 90 & 100 & 110 & 120 & 0 \\ 0 & 130 & 140 & 150 & 160 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \approx \begin{bmatrix} 15.6 & 26.7 & 33.3 & 24.4 \\ 36.7 & 60 & 70 & 50 \\ 63.3 & 100 & 110 & 76.7 \\ 51.1 & 80 & 86.7 & 60 \end{bmatrix}$$

Figure 2.1: Beispiel einer Convolution auf einer $4x4$ Feature Map durch einen $3 \times 3$ Glättungsfilter (Mittelwertfilter). Durch Padding (das Hinzufügen von Werten um eine Eingabematrix) wird die Ausgabe nicht kleiner. Der Stride-Wert beträgt hierbei 1.

Ein zentrales Konzept bei CNNs ist das sogenannte Parameter Sharing. Anstatt für jedes Neuron eines Layers eigene Gewichte zu lernen, wird ein und derselbe Filter über das gesamte Eingabebild verschoben. Dadurch werden die gleichen Muster an unterschiedlichen Positionen erkannt, was die Anzahl der zu lernenden Parameter drastisch reduziert und das Modell effizienter macht. Dieser Mechanismus führt zur Translation Invariance. Einer wichtigen Eigenschaft bei der Verarbeitung von Bild- oder Audiodaten. Das bedeutet, dass das Netzwerk in der Lage ist, ein gelerntes Muster unabhängig von dessen Position im Eingaberaum zu erkennen.

CNNs nutzen die Lokalität der Informationen, indem sie kleine Filtergrößen (z.B. $3 \times 3$) verwenden, um nur nahegelegene Pixel oder Datenpunkte miteinander zu verknüpfen. Diese Struktur spiegelt die Tatsache wider, dass Merkmale wie Farben, Texturen oder spektrale Muster oft lokal begrenzt auftreten.

Nach jeder Faltung wird eine nicht-lineare Aktivierungsfunktion angewendet, um komplexe Zusammenhänge zu erfassen. Die vollständige Differenzierbarkeit der Faltungsoperation ermöglicht ein effizientes Training über Backpropagation und Optimierungsverfahren wie Adam.

CNNs stellen eine leistungsfähige Architektur für die Verarbeitung strukturierter Daten dar. Durch lokale Faltung, Parameterteilung und translationale Invarianz sind sie besonders effizient im Erkennen wiederkehrender Muster. Ihre Fähigkeit zur automatischen Merkmalsextraktion macht sie unverzichtbar für zahlreiche Anwendungsbereiche, insbesondere dort, wo Daten als Raster- oder Zeit-Frequenz-Darstellungen vorliegen.

### 2.1.4 Multiscale Processing und Bildsegmentierung

Natürliche Bilder enthalten Informationen auf verschiedenen Skalenebenen. Feine Details wie Texturen oder Kanten treten auf kleinen Skalen auf, während größere Strukturen wie Objekte, Formen oder Hintergründe auf höheren Abstraktionsebenen erkannt werden. Multiscale Processing bezeichnet Strategien zur gleichzeitigen Verarbeitung und Integration von Informationen über unterschiedliche Auflösungsebenen hinweg. Dies ist insbesondere bei Aufgaben wie der Bildsegmentierung oder Objekterkennung essenziell, bei denen sowohl globale Kontextinformationen als auch lokale Details eine Rolle spielen.

Eine Möglichkeit, multiskalige Informationen zu erfassen, ist die Nutzung von verschiedenen Filtergrößen in Faltungsschichten. Kleine Filter (z. B. $3\times3$) erfassen feine Details, während größere Filter (z. B. $7\times7$ oder $11\times11$) breitere Strukturen analysieren.

Zusätzlich spielt der Stride eine wichtige Rolle. Der Stride bestimmt, wie weit sich der Filter bei jedem Schritt über das Bild bewegt. Ein Stride von 1 bedeutet, dass der Filter jedes Pixel nacheinander verarbeitet, während ein Stride von 2 das Bild in größeren Sprüngen durchläuft und die räumliche Auflösung reduziert. Dies kann effizient sein, um großflächige Muster zu erfassen oder die Anzahl der Berechnungen zu verringern.

Alternativ kann das Eingabebild auf verschiedenen Auflösungsebenen verarbeitet werden, z. B. durch Downsampling oder pyramidale Darstellungen (z. B. Gaußsche oder Laplacesche Pyramiden). Diese Methoden ermöglichen eine hierarchische Analyse des Bildinhalts [BA83].

Die Bildsegmentierung ist eine zentrale Aufgabe der Bildverarbeitung, bei der das Ziel ist, jedem Pixel eines Bildes eine semantische Klasse zuzuweisen. Im Gegensatz zur Objekterkennung, bei der die Position und Klasse einzelner Objekte (z.B. durch Bounding Boxes) erkannt werden, liefert die Segmentierung eine feingranulare Aufteilung des Bildes, häufig in Form eines sogenannten Segmentation Maps $S \in \mathbb{R}^{H\times W\times C}$, wobei $H$ und $W$ die Bildhöhe und -breite, und $C$ die Anzahl der Klassen darstellt [LDG+17].

Die Herausforderung bei beiden Aufgaben liegt darin, sowohl die Position globaler Objektstrukturen als auch deren feine Kanten und Übergänge korrekt zu erkennen. Besonders bei kleinen Objekten oder detailreichen Szenen ist es notwendig,

Information auf mehreren Skalen gleichzeitig zu berücksichtigen.

#### U-Net-Architekturen

Eine besonders effektive Architektur zur Lösung dieser Herausforderung ist das U-Net [RFB15], ursprünglich eingeführt für die biomedizinische Bildsegmentierung. Der Name leitet sich von der symmetrischen U-förmigen Struktur des Netzwerks ab, das aus zwei Hauptpfaden besteht:

- Einem **Encoder**, der durch abwechselnde Faltung und Downscaling schrittweise tiefere, abstraktere Repräsentationen erzeugt.
- Einem **Decoder**, der durch Transposed Convolutions oder Upsampling diese Repräsentationen wieder in hochauflösende Feature Maps rekonstruiert.

Ein zentrales Merkmal des U-Net ist die Verwendung von Skip Connections, bei denen die Feature Maps der Encoder-Schichten direkt an die entsprechenden Decoder-Schichten weitergeleitet und dort miteinander kombiniert (z. B. durch Konkatenation, Addition oder Multiplikation) werden. Mathematisch lässt sich dieser Schritt wie folgt beschreiben:

$$F^{(l)}_{\text{decoder}} = \text{Up}(F^{(l+1)}_{\text{decoder}}) \oplus F^{(l)}_{\text{encoder}}$$

Dabei bezeichnet $\text{Up}(\cdot)$ eine Upsampling-Operation, $F^{(l)}$ die Feature Map auf Ebene $l$ und $\oplus$ die Kombination (z. B. durch Konkatenation, Addition oder Multiplikation).

Durch die Kombination von tieferliegenden, semantisch reichen Features mit hochaufgelösten, ortsgenauen Features über die Skip Connections ermöglicht das U-Net eine effektive multiresolutionale Verarbeitung. Es werden dabei Informationen aus verschiedenen Auflösungsebenen integriert, sodass sowohl globale Kontextinformationen als auch lokale Details für die finale Vorhersage genutzt werden können. Dies adressiert eine zentrale Schwäche vieler klassischer CNN-Architekturen, bei denen durch wiederholtes Downsampling wichtige räumliche Informationen verloren gehen.

Die U-Net-Architektur stellt somit einen leistungsfähigen Ansatz dar, um hochaufgelöste, ortsgenaue Vorhersagen bei gleichzeitigem Zugriff auf tiefere kontextuelle Informationen zu ermöglichen. Ihre Fähigkeit zur multiskaligen Verarbeitung macht sie zu einer zentralen Architektur in zahlreichen modernen Deep-Learning-Anwendungen.

## 2.2 Deep Learning für Audioverarbeitung

### 2.2.1 Audio als Zeitreihe und Verarbeitung von Waveforms

Audiodaten liegen typischerweise als digitale Zeitreihen vor, sogenannte Waveforms, bei denen der Schallpegel in regelmäßigen Abständen als Amplitudenwert abgetastet

wird. Ein Mono-Audiosignal ist somit eine eindimensionale Sequenz $x(t) \in \mathbb{R}$, wobei $t$ die Zeitindizes der Abtastung bei einer festen Abtastrate (z. B. 44,100 Hz) darstellt. Die Abtastrate eines digitalen Audiosignals bestimmt, wie oft pro Sekunde der analoge Schall in diskrete Werte umgewandelt wird. Eine zentrale Rolle spielt hierbei die Nyquist-Frequenz [Sha49], die besagt, dass die maximale Frequenz $f_{\max}$ eines korrekt rekonstruierbaren Signals höchstens die Hälfte der Abtastrate $f_s$ betragen darf:

$$f_{\max} = \frac{f_s}{2}$$

Beispielsweise beträgt die Nyquist-Frequenz bei einer üblichen Abtastrate von 44,100 Hz genau 22,050 Hz, was den maximal erfassbaren Frequenzbereich für digitale Audiosysteme definiert.

Rohwellenformen sind reich an Informationen, enthalten jedoch keine expliziten Merkmale wie Tonhöhe, Timbre oder rhythmische Struktur, was die direkte Modellierung durch neuronale Netze herausfordernd macht. Für viele Aufgaben in der Audiobearbeitung, insbesondere im Kontext von Deep Learning, ist es daher üblich, die Waveform durch eine Zeit-Frequenz-Transformation aufzubereiten. Eine der am häufigsten verwendeten Methoden ist die Short-Time Fourier Transformation (STFT).

### 2.2.2 Short-Time Fourier Transformation (STFT)

Die Short-Time Fourier Transformation (STFT) [All77] zerlegt ein Audiosignal in überlappende Zeitfenster, auf die jeweils eine diskrete Fourier-Transformation angewendet wird. So entsteht ein Spektrogramm, das zeitabhängig die Energieverteilung über verschiedene Frequenzbänder abbildet. Formal ist die STFT eines Signals $x(t)$ definiert als:

$$X(n,\omega) = \sum_{m=-\infty}^{\infty} x(m) \cdot w(m-n) \cdot e^{-j\omega m}$$

Hierbei bezeichnet $w(m)$ ein Fensterfunktion (z. B. Hann oder Hamming), $n$ ist der Zeitschritt, und $\omega$ die Frequenz. Das Ergebnis ist eine komplexwertige Matrix $X(n,\omega)$, deren Betrag $|X(n,\omega)|$ als Magnitude-Spektrogramm betrachtet wird. Optional kann auch die Phase $\arg(X(n,\omega))$ weiterverarbeitet werden. Die Ergebnisse aus einer STFT werden Beispielhaft in Figur 2.2 gezeigt.

Die STFT erlaubt neuronalen Netzwerken den Zugriff auf lokale Frequenzinformationen, die für Aufgaben wie Sprachverarbeitung, Musikverständnis oder Quellentrennung von entscheidender Bedeutung sind.

Um ein im Frequenzbereich dargestelltes Signal zurück in den Zeitbereich zu transformieren, wird die Inverse Short-Time Fourier Transformation (iSTFT) verwendet.

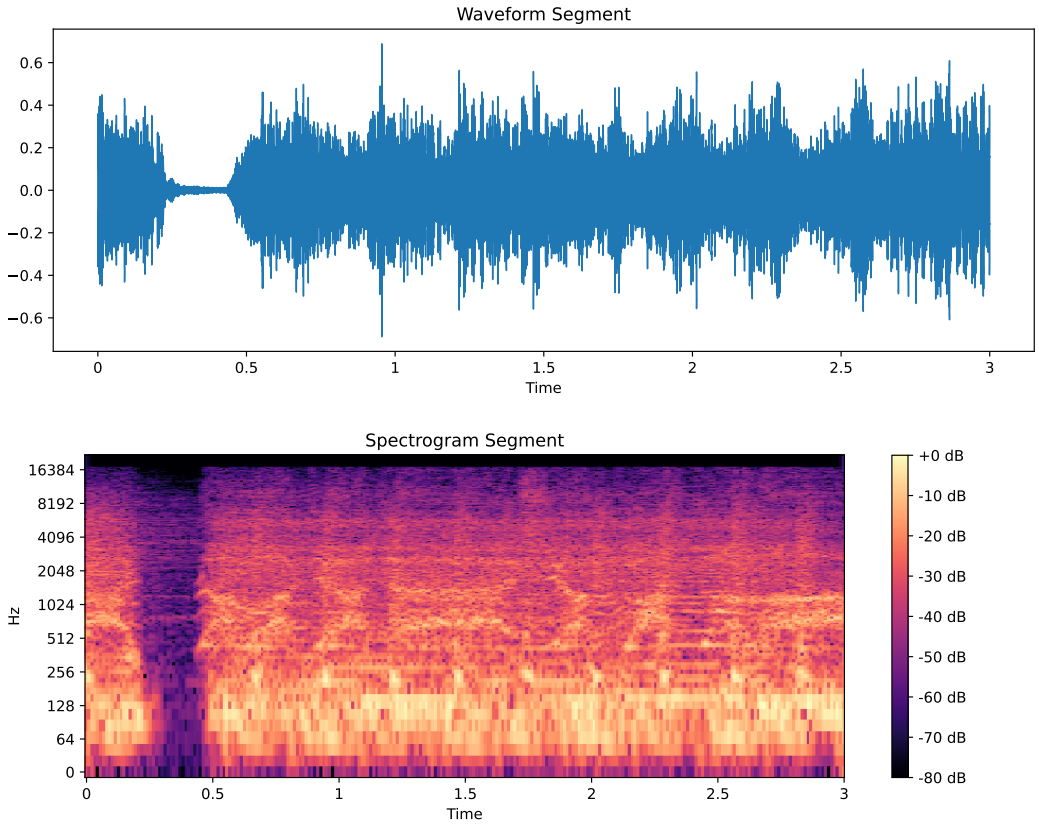


Figure 2.2: Oben: Abbild der Waveform. Unten: Das dazugehörige logarithmische Spektrogramm. Es wird ein 3 Sekunden Abschnitt des Liedes "Ill Fate" von "Hollow Ground" aus dem MUSDB18 Datensatz gezeigt [RLS+19].

Sie ist die Umkehrung der STFT und rekonstruiert das ursprüngliche Audiosignal aus den spektralen Komponenten.

$$x(t) = \sum_{n} w(n-t) \sum_{\omega} X(n,\omega) e^{j\omega t}$$

Dabei setzt sie die transformierten Zeit-Frequenz-Daten unter Berücksichtigung der Fensterfunktion $w(n)$ zurück in die ursprüngliche Zeitdarstellung.

### 2.2.3 Musikquellentrennung (Music Source Separation)

Die Aufgabe der Musikquellentrennung (MSS) besteht darin, aus einem gemischten Musiksignal einzelne semantisch getrennte Komponenten wie Gesang, Bass, Schlagzeug oder andere Instrumente zu extrahieren. Gegeben ist ein Audiosignal $x(t)$, das sich als lineare Mischung $x(t) = \sum_{i=1}^{N} s_i(t)$ mehrerer Quellsignale $s_i(t)$ zusammensetzt. Ziel ist es, aus $x(t)$ eine möglichst genaue Schätzung $\hat{s}_i(t)$ jeder Quelle

zu rekonstruieren. Dargestellt in Figur 2.3. MSS ist eine besonders anspruchsvolle

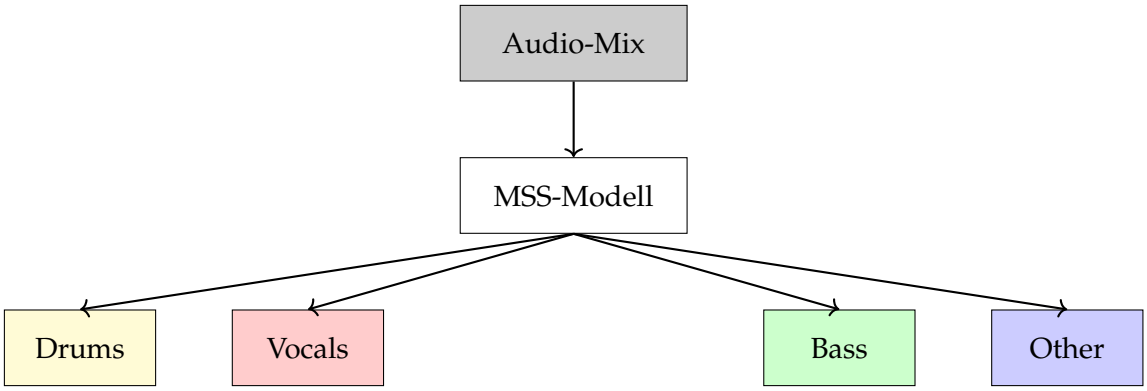


Figure 2.3: Darstellung der Musikquellentrennen. Ein Audiomix (Musikstück) wird durch ein neuronales Netzwerk in seine einzelnen Bestandteile (Bass, Vocals, Drums und andere Instrumente) zerlegt.

Form der Blind Source Separation (BSS) [AAA+23], da die Quellen, sowohl im Frequenzraum als auch in ihrer zeitlichen Struktur, stark überlappen. Darüber hinaus erfordert eine erfolgreiche Trennung sowohl die Modellierung spektraler Eigenschaften (z.B. Formanten, Instrumentenspektren) als auch die Erfassung zeitlicher Dynamiken (z.B. Rhythmus, Artikulation).

### 2.2.4 Bewertung mit Signal-to-Distortion Ratio (SDR)

Die Qualität der rekonstruierten Quellsignale wird häufig mit der Signal-to-Distortion Ratio (SDR) bewertet. Die SDR quantifiziert das Verhältnis zwischen dem rekonstruierbaren Signalanteil und der Gesamtheit der Verzerrungen (Fehler, Störungen, Artefakte):

$$\text{SDR} = 10 \cdot \log_{10}\left(\frac{||s||^2}{||s - \hat{s}||^2}\right)$$

Hierbei ist $s$ das Originalsignal und $\hat{s}$ die rekonstruierte Quelle. Ein hoher SDR-Wert zeigt an, dass das geschätzte Signal nahe am Original liegt. In wissenschaftlichen Benchmarks wie MUSDB18-HQ [RLS+19] oder dem Sound Demixing Challenge [FUL+24] werden häufig Durchschnittswerte über mehrere Quellen und Tracks angegeben.

Die Maßeinheit für den SDR ist Dezibel. Das Dezibel (dB) ist eine logarithmische Maßeinheit zur Beschreibung von Lautstärken und Signalpegeln. Die Verwendung einer logarithmischen Skala ergibt sich aus der menschlichen Wahrnehmung. Menschen nehmen Lautheitsunterschiede nicht linear, sondern logarithmisch wahr. Eine

Verdoppelung der physikalischen Energie eines Schallsignals führt subjektiv nicht zu einer Verdoppelung der wahrgenommenen Lautstärke.

Die allgemeine Definition eines Pegels in Dezibel lautet:

$$L = 10 \log_{10} \left( \frac{P}{P_{\text{ref}}} \right)$$

wobei $P$ die Leistung des Signals ist und $P_{\text{ref}}$ die Referenzleistung. Alternativ kann die Amplitude $A$ eines Signals verwendet werden. Dies ist besonders relevant für Audiosignale:

$$L = 20 \log_{10} \left( \frac{A}{A_{\text{ref}}} \right)$$

Ein Pegel von $0dB$ bedeutet nicht das Fehlen von Schall, sondern dass das Signal exakt die Referenzlautstärke besitzt. Negative dB-Werte bedeuten eine geringere Lautstärke als die Referenz, positive Werte eine höhere.

### 2.2.5 Sound Demixing Challenge (SDX)

Die Sound Demixing Challenge [FUL+24] ist ein international organisierter Wettbewerb, der sich auf die Herausforderung der Musikquellentrennung spezialisiert. Ziel der Challenge ist es, leistungsstarke Deep-Learning-Modelle zu entwickeln, die in der Lage sind, aus einem gemischten Audiosignal einzelne musikalische Quellen – wie Gesang, Bass, Schlagzeug oder andere Instrumente – zu extrahieren. Die Challenge bietet dabei nicht nur einen objektiven Vergleich aktueller Ansätze, sondern stellt auch hochwertige Datensätze und einheitliche Evaluierungsmethoden zur Verfügung, um Forschungsergebnisse reproduzierbar und vergleichbar zu machen.

Die Challenge wird regelmäßig über z.B. die Plattform `www.aicrowd.com` ausgerichtet und wird von Forschungsinstitutionen und großen Unternehmen wie Sony unterstützt. Besonders herausragende Modelle können Preisgelder gewinnen, wodurch der Wettbewerb zusätzliche Anreize für Teilnehmer bietet. Aufgrund der starken Konkurrenz ist die Challenge eine ausgezeichnete Plattform, um nach kompetitiven und leistungsfähigen Modellen für die Musikquellentrennung zu suchen.

Da es sich bei der SDX um einen Wettbewerb handelt, light der Fokus der Teilnehmer vor allem darauf, hohe Scores und gute Platzierungen in den Ranglisten zu erzielen. Wissenschaftliche Dokumentation oder sauberer Code stehen dabei oft nicht im Mittelpunkt. Dies hat zur Folge, dass in der Challenge vorgestellten Modelle, darunter auch das **TFC-TDF-UNet V3**-Modell, oft nicht klar dokumentiert oder vollständig beschrieben sind, wodurch eine Reproduktion dieser Modelle erheblich erschwert wird.

Ein zentrales Ziel dieser Arbeit besteht darin, dieses Problem für das **TFC-TDF-UNet V3**-Modell zu lösen, indem dessen Architektur rekonstruiert, systematisch

dokumentiert und reproduzierbarer gemacht wird. Dies trägt dazu bei, die wissenschaftliche Nachvollziehbarkeit zu verbessern und zukünftige Forschung im Bereich der Musikquellentrennung zu erleichtern.

In der Regel umfasst die Sound Demixing Challenge mehrere Subtracks oder Leaderboards, die sich je nach Jahr und Schwerpunkt unterscheiden. In der Ausgabe SDX23 gliederte sich der Wettbewerb in die folgenden Kategorien:

- **Leaderboard A – Label-Noise Robustness:** Bewertet die Fähigkeit von Modellen, auch unter Bedingungen mit fehlerhaften Quelllabels (Label-Noise) robuste Trennungen durchzuführen.
- **Leaderboard B – Bleeding Robustness:** Fokus auf Modelle, die mit Bleeding umgehen können, also dem Übersprechen von Instrumenten zwischen den Spuren.
- **Leaderboard C – Standard Separation:** Klassische Musikquellentrennung ohne Störungen, bei der beliebige Trainingsdaten verwendet werden dürfen.

Jeder Track stellt andere Anforderungen an Modellarchitektur, Training und Generalisierungsfähigkeit.

Für die MDX-Challenge werden speziell kuratierte Datensätze bereitgestellt, darunter:

- **MUSDB18-HQ**: Ein weit verbreiteter Benchmark-Datensatz, der hochwertige Aufnahmen von Musikstücken mit separaten Stems für Gesang, Bass, Schlagzeug und andere Instrumente bietet [RLS+19].
- **SDXDB23 Label-Noise / Bleeding**: Zwei von `Moises.ai` [PAKV23] bereitgestellte Datensätze mit gezielt eingebautem *Label-Noise* bzw. *Bleeding* um die Robustheit der Modelle gegenüber Fehlern zu testen in den Datensätzen zu testen.

Die Herausforderungen wie *Label-Noise* und *Bleeding* spiegeln reale Probleme wider, wie sie im Musikproduktionsprozess regelmäßig auftreten. Dieser Datensatz besteht aus Musikstücken, denen *Label-Noise* beigefügt wird. *Label-Noise* stellt ein Problem bei der Erstellung von Trainingsdaten für die Musikquellentrennung dar. Es beschreibt die fehlerhafte oder ungenaue Zuordnungen zwischen den Audiosignalen und den zugehörigen Quelllabels. Unter *Bleeding* versteht man das unerwünschte akustische Überschreiten einer Tonquelle in eine andere, eigentlich isolierte Aufnahme. In Studioproduktionen tritt dieses Phänomen häufig auf, wenn beispielsweise der Klang eines Schlagzeugs über das Mikrofon des Gesangs aufgenommen wird Dadurch enthalten isolierte Spuren nicht ausschließlich die intendierte Quelle, sondern auch Störanteile anderer Instrumente oder Stimmen.

Diese Störfaktoren können die Modellqualität beeinträchtigen, da das neuronale Netzwerk versucht, fehlerhafte oder widersprüchliche Muster zu lernen. Dies kann zu einer geringeren Trennleistung und einer schlechteren Generalisierbarkeit auf neue, realitätsnahe Daten führen. Die Fähigkeit von Modellen, unter solchen Bedingungen

zuverlässig zu arbeiten, ist entscheidend für den praktischen Einsatz von Musikquellentrennung in industriellen Anwendungen. Die Qualität der Separationsergebnisse wird anhand des Signal-to-Distortion Ratio (SDR) gemessen.

Durch die klare Aufgabenstellung, öffentlich zugängliche Vergleichsdaten und die hohe Sichtbarkeit in der Forschungsgemeinschaft bietet die Sound Demixing Challenge eine ideale Plattform, um neue Modelle, Trainingsstrategien und Evaluierungsmethoden zu testen. Einige heute führenden Modelle, auch das TFC-TDF-UNet V3, wurden im Rahmen dieser Challenge entwickelt, evaluiert und anschließend veröffentlicht.

# 3 Das TFC-TDF-UNet V3 Modell

TFC-TDF-UNet V3 [KLJ23] ist ein Modell, das aus den Lösungen für den Music Demixing Track der Sound Demixing Challenge 2023 hervorgegangen ist. Es ist eine Weiterentwicklung des MDX-NEt Modells [KCC+21].

## 3.1 Architektur

Die Architektur folgt einem, typischen U-Net-Design mit Encoder-Decoder-Struktur und Skip-Verbindungen, die helfen, hochauflösende Informationen während der Tiefenreduktion zu bewahren. Das Modell verarbeitet komplexwertige STFT-Outputs als Eingabe und erzeugt für jede Zielquelle ein geschätztes Spektrum, das per iSTFT in die Zeitdomäne rücktransformiert wird. Der Encoder, bestehend aus TFC-TDF und Downscale-Blöcken, extrahiert zunehmend abstrakte Merkmale, während der Decoder diese in der Synthesephase nutzt, um eine präzisere Trennung zu ermöglichen. Die Skip-Verbindungen sorgen dabei für den Erhalt wichtiger Detailinformationen aus früheren Schichten. (Siehe Abbildungen 3.1, 3.5 und 3.2)

Das Modell verfolgt einen explizit multiresolutionalen Ansatz. Lokale Muster werden durch Faltungen erfasst, während TDF-Module globale Strukturinformationen innerhalb der Frequenzachse modellieren.

## 3.2 Training

### 3.2.1 Datenladen

Für das Training werden Audiodateien einzelner Instrumente und Gesangsspuren verwendet, die jeweils isoliert vorliegen. Diese dienen als Ground-Truth für die jeweilige Zielquelle. Um ein realistisches Trainingssignal zu erzeugen, werden die einzelnen Spuren zufällig kombiniert und zu einem gemeinsamen Mix summiert. Das resultierende Signal enthält somit alle Quellen überlagert. Anschließend wird das gemischte Signal in kleinere Abschnitte (Chunks) unterteilt und in Batches zusammengefasst, die dann dem Modell als Eingabe zugeführt werden.

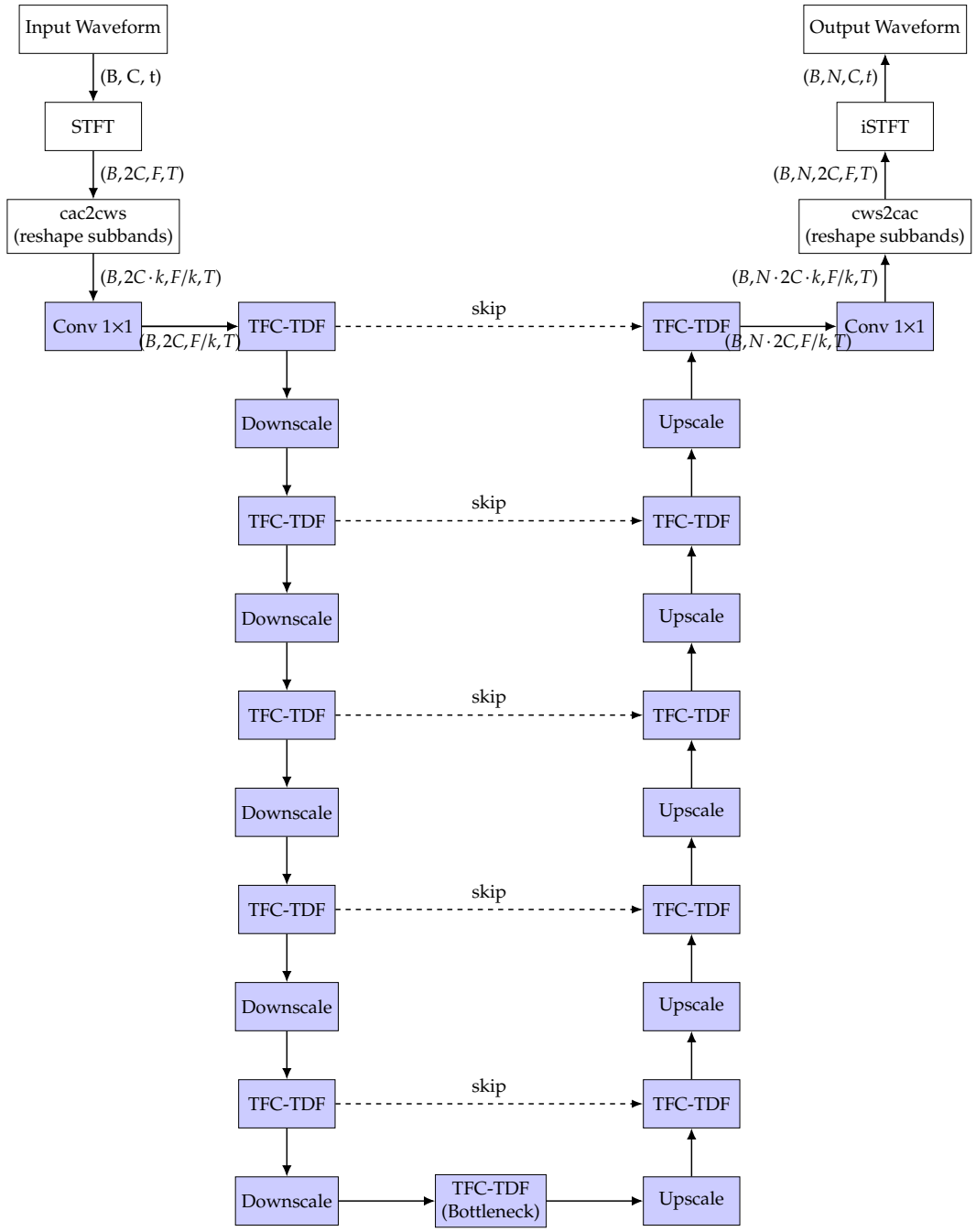


Figure 3.1: Übersicht der U-Net Struktur des TFC-TDF-UNet V3 Modells. Blau markierte Boxen enthalten trainierbare Parameter. Die TFC-TDF Blöcke werden in Figur 3.2 genauer beleuchtet. B = Batchgröße, C = Anzahl Kanäle, t = Chunkgröße, F = Frequenz-Bins, k = Anzahl der Subbänder, T = STFT Frames, N = Anzahl der Zielinstrumente

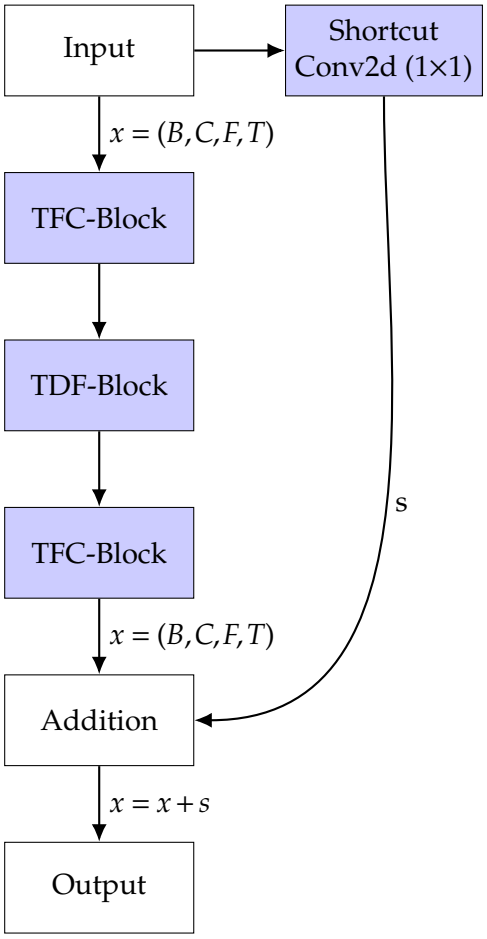


Figure 3.2: TFC-TDF-Block. Die blau markierten Boxen enthalten trainierbare Parameter. Die TFC- und TDF-Blöcke werden detailierter in Abbildung 3.5 beleuchtet.

### 3.2.2 Loss

Beim Training des TFC-TDF-UNet V3 Modells auf den *Bleeding*- und *Label-Noise*-Datensätzen kommt eine modifizierte Form des L2-Losses (Mean Squared Error) zum Einsatz. Diese Anpassung hat das Ziel, die Robustheit des Modells gegenüber ungenauen oder fehlerhaften Trainingsdaten zu erhöhen und gleichzeitig den Lernprozess zu stabilisieren.

Anstatt den mittleren quadratischen Fehler (MSE) über alle Trainingsbeispiele zu berechnen, berücksichtigt die modifizierte Variante nur jene Beispiele, deren Fehler unterhalb eines bestimmten Quantils $Q_q$ liegen. Die Loss-Funktion ist damit:

$$\mathcal{L}_{\text{masked}} = \frac{1}{|S_q|} \sum_{i \in S_q} (y_i - \hat{y}_i)^2$$

mit

$$S_q = \left\{ i \mid (y_i - \hat{y}_i)^2 < Q_q \right\}$$

Hierbei bezeichnet $y_i$ die Ground-Truth-Wellenform und $\hat{y}_i$ die vom Modell vorhergesagte Wellenform für das $i$-te Beispiel. $Q_q$ ist der Schwellenwert des $q$-Quantils der Fehlerverteilung. Nur Trainingsbeispiele, deren Fehler unterhalb dieser Schwelle liegen, werden in die Berechnung des Loss einbezogen.

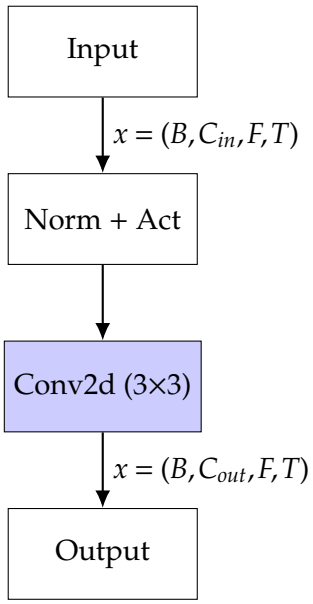


Figure 3.3: TFC-Block

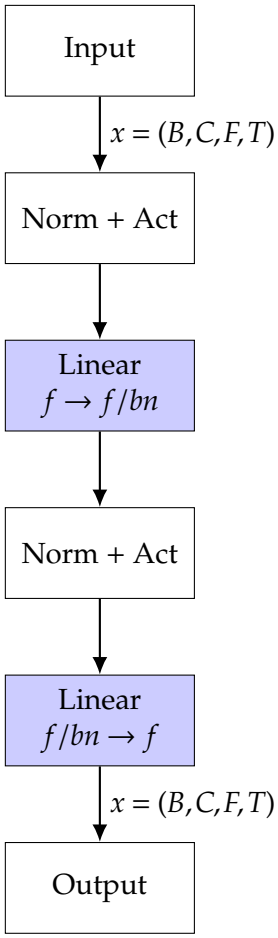


Figure 3.4: TDF-Block

Figure 3.5: Der TFC-Block ist darauf ausgelegt, lokale Muster in den zeitlichen und frequenzbasierten Dimensionen zu extrahieren. Der TDF-Block ist darauf spezialisiert, globale Beziehungen in der Frequenzdimension zu modellieren. Er verwendet voll verbundene Schichten (Linear-Layer), um die Frequenzdimension zu transformieren. $f \rightarrow f/bn$ reduziert die Frequenzdimension. Im zweiten Linear-Layer wird diese wieder rekonstruiert. Trainierbare Parameter sind blau markiert.

| Model | vocals | drums | bass | other | mean |
|---|---|---|---|---|---|
| Modell A mit Loss-Maskierung | 7.58 | 6.38 | 6.43 | 4.64 | 6.26 |
| Modell A ohne Loss-Maskierung | 6.12 | 5.31 | 5.31 | 3.45 | 5.05 |
| Modell B mit Loss-Maskierung | 7.41 | 6.20 | 6.58 | 4.69 | 6.22 |
| Modell B ohne Loss-Maskierung | 6.87 | 5.86 | 6.11 | 4.36 | 5.80 |

Table 3.1: Vergleich der Modelle A und B mit und ohne Loss-Maskierung [KLJ23].

Diese Technik ist insbesondere bei verrauschten oder teilweise fehlerhaften Datensätzen hilfreich, da sie den Einfluss von Ausreißern und inkonsistenten Labels auf den Gradientenfluss reduziert, wodurch eine bessere Musikquellentrennung erreicht werden kann, wie man in Tabelle 3.1 sehen kann.

## 3.3 Vergleich mit anderen State of the Art Modellen

### 3.3.1 Demucs und Hybrid Demucs

Demucs [Dé22] war ein bedeutender Fortschritt, da es als erstes Wellenform-basierte Methode die Spektrogramm-basierten Methoden in Bezug auf den Gesamt-SDR (6.3 dB auf MUSDB) übertraf. Demucs arbeitet direkt mit der rohen Wellenform als Eingabe und gibt pro Zielquelle ebenfalls eine rekonstruierte Wellenform zurück. Die ursprüngliche Demucs-Architektur basiert auf einem U-Net-ähnlichen Encoder-Decoder-Modell mit bidirektionalen LSTM-Schichten im Bottleneck. Hybrid Demucs kombiniert Wellenform- und Spektrogrammverarbeitung über parallele Zweige.

### 3.3.2 Band-Split RNN (BSRNN)

BSRNN [LY22] ist ein Spektrogramm-basiertes MSS-Modell, das speziell für Signale mit hoher Abtastrate entwickelt wurde. BSRNN nimmt ein komplexwertiges Spektrogramm als Eingabe und gibt für jede Zielquelle ein maskiertes Spektrogramm zurück, das über eine inverse STFT in die Zeitdomäne transformiert wird. Das Spektrogramm wird in Subbänder zerlegt, welche jeweils durch Residual-RNN-Schichten verarbeitet werden. Danach erfolgt eine Re-Kombination der Subbands über Multi-Layer-Perzeptronen (MLPs) zur Erzeugung von Zeit-Frequenz-Masken. Die Idee ist eine spektrale Spezialisierung durch Bandtrennung, kombiniert mit sequentieller Modellierung innerhalb und über Bänder hinweg. Dies erlaubt eine gezielte Behandlung frequenzspezifischer Muster.

### 3.3.3 Vergleich

Alle drei Modelle wurden zum Vergleich auf dem MUSDB18-HQ [RLS+19] Datensatz ohne externe Daten trainiert. Eine Übersicht bietet Tabelle 3.2. Obwohl BSRNN bei

| Modell | SDR Vocals | Drums | Bass | Other | Mean | Speed (x Real-Time) |
|---|---|---|---|---|---|---|
| BSRNN | 10.04 | 8.92 | 6.80 | 6.01 | 7.94 | 0.7 |
| Hybrid Demucs | 8.11 | 8.87 | 7.76 | 5.39 | 7.53 | 8.9 |
| TFC-TDF-UNet V3 | 9.22 | 8.81 | 7.36 | 6.19 | 7.90 | 15.0 |
| + Overlap Add | 9.34 | 8.96 | 7.53 | 6.32 | 8.04 | 3.9 |

Table 3.2: Vergleich ausgewählter Modelle auf dem MUSDB18-HQ Testdatensatz. Die Geschwindigkeit wurde auf dem MDX23-Evaluierungsserver gemessen. Für BSRNN wurde eine inoffizielle Implementierung genutzt [KLJ23].

einzelnen Quellen wie *Vocals* die besten SDR-Werte erzielt, hat es vergleichsweise eine zu geringe Geschwindigkeit. Hybrid Demucs zeigt solide Leistungen, erreicht jedoch nicht die gleiche Qualität der Quellentrennung und ist langsamer als TFC-TDF-UNet V3.

# 4 Reproduzierbarkeit und Modularität des TFC-TDF-UNet V3 Modells

## 4.1 Datensätze und Evaluierung

Im Rahmen dieser Arbeit werden zwei Modelle untersucht, die im Kontext der Leaderboards A und B der *Sound Demixing Challenge 2023* entwickelt wurden. Im Folgenden werden diese als Modell A und Modell B bezeichnet.

*Modell A* wird ausschließlich mit dem *Label-Noise*-Datensatz trainiert, der im Rahmen der Challenge bereitgestellt wurde. Dieser Datensatz enthält absichtlich inkonsistente oder fehlerhafte Label-Zuordnungen, um die Robustheit der Modelle gegenüber unzuverlässigen Trainingsdaten zu testen. *Modell B* hingegen wird mit dem sogenannten *Bleeding*-Datensatz trainiert, bei dem es zu akustischen Überschneidungen zwischen den Quellspuren kommt. Solche Überschneidungen stellen eine Herausforderung dar, da Trainingsdaten im Idealfall aus vollständig isolierten Quellen bestehen sollten. Das *Bleeding* simuliert realistische Studioaufnahmen, in denen Spuren nicht perfekt voneinander getrennt sind.

Für die Evaluation beider Modelle wird der Testdatensatz des *MUSDB18-HQ* [RLS+19] verwendet, der ursprünglich im Rahmen der *Signal Separation Evaluation Campaign (SiSEC)* 2018 [SLI18] veröffentlicht wurde. Der Datensatz umfasst 50 Musikstücke aus verschiedenen Genres im WAV-Format mit einer Gesamtspielzeit von etwa 3,5 Stunden. Als Metrik zur Bewertung der Modellleistung wird der *Signal-to-Distortion Ratio (SDR)* verwendet, der die Trennqualität quantifiziert, indem das Verhältnis zwischen dem rekonstruierten Signal und den darin enthaltenen Störanteilen gemessen wird.

## 4.2 Evaluierung des ursprünglichen Modells

Eine zentrale Herausforderung dieser Arbeit liegt in der Modularisierung der Trainings- und Evaluierungslogik. Die ursprüngliche Codebasis ist stark hardcodiert, was zur Folge hat, dass für jede Modellvariante oder jedes Experiment manuelle und teils tiefgreifende Änderungen am Quellcode erforderlich sind. Dieses Vorgehen erweist sich als fehleranfällig, wartungsintensiv und wenig skalierbar.

Im Rahmen dieser Arbeit wird daher die Architektur des Trainingssystems über-

arbeitet und so umgestaltet, dass zentrale Parameter und Komponenten über konfigurierbare Schnittstellen steuerbar sind. Ziel ist es, eine flexible und erweiterbare Umgebung zu schaffen, die eine einfache Integration neuer Modelle, eine systematische Durchführung von Experimenten sowie eine konsistente und reproduzierbare Evaluierung ermöglicht.

Diese Umstrukturierung verbessert nicht nur die Nachvollziehbarkeit und Effizienz der Arbeit, sondern schafft auch eine belastbare Grundlage für weiterführende Forschung und zukünftige Erweiterungen.

Eine Aufgabe dieser Arbeit besteht darin, die bestehenden Modelle lokal zu reproduzieren und die im Rahmen der Music Demixing Challenge berichteten Evaluierungsergebnisse nachzuvollziehen. Zu diesem Zweck werden die bereitgestellten Modell-Checkpoints heruntergeladen und unter identischen Testbedingungen lokal evaluiert.

Dabei zeigt sich, dass die lokal erzielten Ergebnisse sowohl für Modell A als auch Modell B in allen Kategorien niedrigere SDR-Werte aufweisen als in der Originalveröffentlichung angegeben (vgl. Tabelle 4.2). Besonders auffällig ist, dass die Unterschiede bei den Quellen *Bass* und *Vocals* deutlich ausgeprägter sind, während die SDR-Werte für *Drums* und *Other* weitgehend mit den publizierten Resultaten übereinstimmen.

Für alle nachfolgenden Vergleiche und Experimente dienen die lokal evaluierten Modelle als Referenzbasis, um konsistente und vergleichbare Rahmenbedingungen sicherzustellen. Dadurch wird gewährleistet, dass alle Ergebnisse unter identischen Voraussetzungen ermittelt werden und etwaige Leistungsunterschiede klar auf die vorgenommenen Anpassungen zurückgeführt werden können. Diese lokal reproduzierten Modelle werden im Folgenden als Referenz A und Referenz B bezeichnet.

Zur belastbaren Bewertung der tatsächlichen Leistungsfähigkeit der jeweiligen Modellarchitektur werden für jedes Modell drei unabhängige Trainingsläufe mit identischen Hyperparametern durchgeführt. Die daraus resultierenden Testergebnisse werden anschließend statistisch zusammengeführt (vgl. Fehlerbalken in Abbildung 4.1). Auf diese Weise lässt sich für beide Modelle eine Fehlerspanne bestimmen, welche die Unsicherheit der Ergebnisse quantifiziert und eine differenziertere Interpretation der Modellleistung ermöglicht.

Auch die lokal neu trainierten Modelle erreichen durchweg niedrigere SDR-Werte als in der Originalveröffentlichung angegeben und spiegeln eher den Ergebnissen der lokalen Evaluierung wieder, wie anhand der Resultate in Tabelle 4.3 ersichtlich ist.

Obwohl sämtliche Hyperparameter sowie die verwendeten Zufalls-Seed-Werte konstant gehalten werden, treten dennoch leichte Unterschiede in den Evaluierungsergebnissen zwischen den Trainingsläufen auf. Dies deutet darauf hin, dass im Train-

| Hyperparameter | Bestenliste A&B ModellA | ModellB |
|---|---|---|
| **STFT** | | |
| n_fft (Fenstergröße) | 8192 | |
| hop_length (Sprungweite) | 1024 | |
| **Modell** | | |
| # Frequenz-Bins | 4096 | |
| # Anfangskanäle | 64 | |
| Wachstumsrate | 64 | |
| # Down/Up-Skalen | 5 | |
| # Blöcke pro Skala | 2 | |
| # Subbänder | 4 | |
| TDF Batch-Norm Faktor[**?**] | 4 | |
| Normalisierung | InstanceNorm | |
| Aktivierungsfunktion | GELU | |
| # Parameter | 30M | |
| **Training** | | |
| Optimierer | Adam | |
| Lernrate | 1e-4 | |
| Batch-Größe | 6 | |
| Chunk-Größe | $\approx 6s$ | |
| Verlustmasken-Dimensionen | Batch | Batch, Zeit |
| $q$ | $\in [1/3, 1/2)$ | 0,93 |
| **Inferenz** | | |
| Chunk-Größe | $\approx 24s$ | |
| Overlap-Add-Faktor | 8 | |

Table 4.1: Konfiguration der Hyperparameter für die einzelnen Modelle des TFC-TDF-UNet v3 Modells (Wachstumsrate: die Anzahl der Kanäle wird nach jeder Down-/Upsampling-Ebene um diesen Betrag erhöht/verringert; Verlustmaskendimensionen: die $q$-Quantile werden entlang dieser Dimensionen für die Verlustmaskierung berechnet; Overlap-Add: hop_size = Chunk-Größe / Overlap-Add_Faktor) Tabelle ist aus [KLJ23] entnommen. Jedes Modell besteht aus drei identisch trainierten Checkpoints, welche nach den in der Tabelle 4.1 angegebenen Parametern, aber mit unterschiedlichen Random Seed ($seed \in [0, 1, 2]$) trainiert wurden.

ingsablauf eine Varianz besteht, die sich trotz deterministischer Einstellungen nicht vollständig vermeiden lässt. Mögliche Ursachen liegen beispielsweise in nicht-deterministischen Operationen auf GPU-Hardware oder in parallelen Berechnungsprozessen innerhalb der verwendeten Deep-Learning-Frameworks.

| | | SDR | | | | |
|---|---|---|---|---|---|---|
| Modell | | Bass | Drums | Other | Vocal | Mean |
| Original | A | 6.71 | 6.71 | 4.82 | 7.82 | 6.51 |
| Lokal | A | 5.88 | 7.07 | 4.54 | 6.98 | 6.12 |
| Original | B | 6.98 | 6.65 | 4.96 | 7.74 | 6.58 |
| Lokal | B | 5.89 | 7.27 | 4.82 | 7.05 | 6.26 |

Table 4.2: SDR-Werte der Modelle A und B laut SDX23 Leaderboard (Original A,B) [FUL+24] und die lokal evaluierten Referenzmodelle von KUIELAB (Lokal A,B) [KLJ23]. Die SDR-Werte sind auf zwei Nachkommastellen gerundet.

| | SDR | | | | |
|---|---|---|---|---|---|
| Modell | Bass | Drums | Other | Vocals | Mean |
| | 5.85 | 6.93 | 4.33 | 6.87 | 5.99 |
| A | 5.83 | 7.00 | 4.50 | 6.91 | 6.06 |
| | 5.36 | 6.24 | 3.81 | 6.21 | 5.40 |
| | 5.78 | 7.16 | 4.80 | 6.96 | 6.18 |
| B | 5.97 | 7.36 | 4.89 | 7.04 | 6.31 |
| | 5.68 | 7.01 | 4.86 | 7.21 | 6.19 |

Table 4.3: Lokal ermittelte SDR-Werte für die Modelle A und B auf Leaderboard A und B. Für jedes Modell werden drei Checkpoints mit den Hyperparametern aus Tabelle 4.1 und jeweils unterschiedlichem Seed per Checkpoint trainiert und anschließend evaluiert.

## 4.3 Protokollierung des Trainings durch Cross-Validation

Ein weiterer zentraler Aspekt dieser Arbeit ist die systematische Protokollierung und Validierung des Trainingsverlaufs. Ziel ist es, zu untersuchen, ob ein Modell bereits nach kürzerer Trainingszeit eine vergleichbare Leistung wie vollständig trainierte Modelle erzielt. Zu diesem Zweck wird eine automatische Snapshot-Funktion implementiert, die den aktuellen Zustand des Modells in festgelegten Intervallen speichert. In dieser Arbeit erfolgt die Speicherung jeweils alle 10 000 Trainingsschritte.

Ergänzend dazu kommt eine Cross-Validation-Routine zum Einsatz, die in denselben Intervallen jeden gespeicherten Snapshot auf einem festen Validierungsdatensatz evaluiert. Dieser Validierungsdatensatz basiert auf einer gekürzten Version des *MUSDB18-HQ* Test-Datensatzes [RLS+19].

Nach jeder Validierungsrunde werden zentrale Metriken wie der Validierungs-Loss, die SDR-Werte sowie die bis dahin benötigte Trainingszeit gespeichert. Diese Form der regelmäßigen Zwischenbewertung ermöglicht eine detaillierte Analyse des Lernfortschritts und kann dabei helfen, optimale Trainingszeitpunkte zu identifizieren und den Ressourcenaufwand gezielt zu minimieren.

Durch diese erweiterte Protokollierung ist es möglich, den zeitlichen Verlauf der Modellleistung präzise zu analysieren, frühzeitig optimale Snapshots zu identifizieren und Trainingszeiten gegebenenfalls zu reduzieren, ohne dabei Einbußen bei der Modellqualität in Kauf nehmen zu müssen. Gleichzeitig schafft diese Form der Validierung fundierte Basis für Folgeexperimente, etwa zum Einfluss verschiedener Modellgrößen, Datenmengen oder Regularisierungsmethoden.

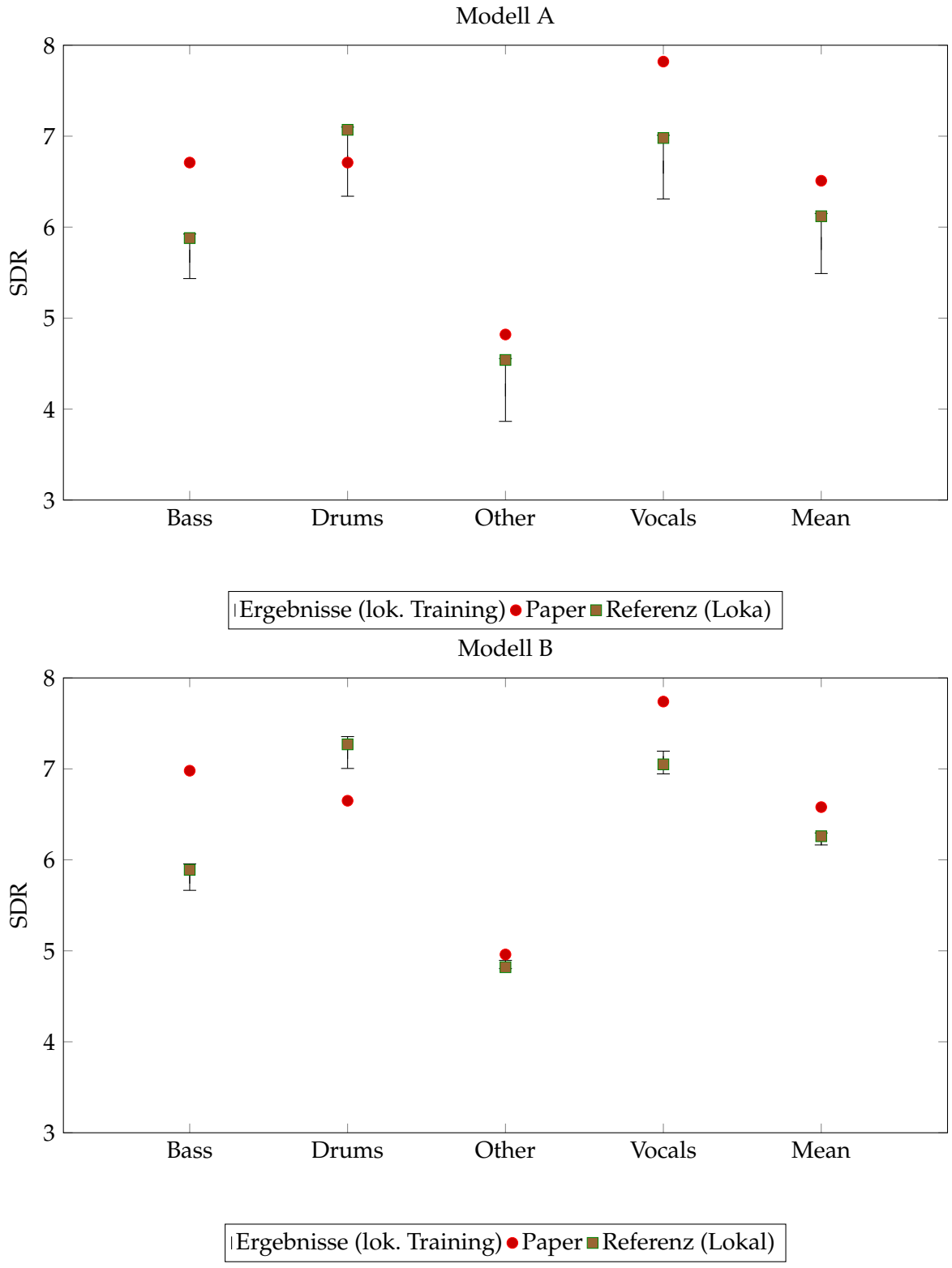


Figure 4.1: Fehlerspanne basierend auf den lokal trainierten Modellen aus Tabelle 4.3 Für jeweils Modell A und B. Ergebnisse aus 4.2 sind als einzelne Punkte angezeigt.

# 5 Effizienzsteigerung durch Optimierung des Trainingumfeldes

Ein weiteres Ziel dieser Arbeit ist die Optimierung der Trainingseffizienz. Obwohl das untersuchte Modell im Vergleich zu anderen Ansätzen bereits eine relativ hohe Inferenzgeschwindigkeit aufweist, nutzt das ursprüngliche Trainingsverfahren die verfügbaren Rechenressourcen nicht vollständig aus. Insbesondere der potenzielle Leistungsgewinn durch den Einsatz mehrerer GPUs bleibt in der bestehenden Implementierung weitgehend ungenutzt.

Ziel ist es daher, die Trainingspipeline so zu überarbeiten, dass das Training effizient auf mehreren Grafikkarten parallelisiert werden kann. Durch die gleichzeitige Nutzung mehrerer GPUs soll die Trainingszeit signifikant verkürzt werden. Im Idealfall führt die effizientere Parallelisierung sogar zu einer verbesserten Performance, beispielsweise durch die Möglichkeit, größere Batch-Größen zu verwenden oder ein stabileres Gradientenverhalten zu erzielen.

Um dieses Ziel zu erreichen, werden im Rahmen dieser Arbeit mehrere grundlegende Anpassungen an der Trainingsarchitektur vorgenommen, mit dem Ziel, die vorhandenen Hardware-Ressourcen bestmöglich auszuschöpfen und den gesamten Trainingsprozess deutlich zu beschleunigen.

## 5.1 Einführung von Distributed Data Parallel

Ein zentraler Schritt zur Steigerung der Trainingseffizienz ist die Einführung von Distributed Data Parallel (DDP) [LZV+20], einem Framework innerhalb von PyTorch, das die parallele Ausführung von Trainingsprozessen über mehrere GPUs hinweg ermöglicht. Durch den Einsatz von DDP lässt sich das Modelltraining effektiver auf mehrere Grafikkarten verteilen, was in dieser Arbeit zu einer mehr als fünffachen Beschleunigung des Trainingsprozesses, auf derselben Hardware, führt.

Ein wesentlicher zusätzlicher Vorteil von DDP besteht in der Skalierung der verfügbaren Speicherressourcen. Der kumulierte VRAM der beteiligten GPUs erlaubt es, deutlich größere Batchgrößen zu verwenden. Diese Möglichkeit wird gezielt genutzt, um die Batchgröße signifikant zu erhöhen. Größere Batches wirken sich positiv auf die Stabilität des Gradientenflusses aus und erhöhen zugleich die Effizienz pro Trainingsschritt.

In direktem Zusammenhang damit wird auch die Lernrate entsprechend angepasst und erhöht. Da größere Batchgrößen typischerweise höhere Lernraten erlauben, ohne das Optimierungsverhalten negativ zu beeinflussen, kann so das Modell nicht nur schneller, sondern auch qualitativ effektiver trainiert werden.

## 5.2 Anpassung der Batchgröße und Lernrate

Durch die Einführung von Distributed Data Parallel (DDP) lässt sich das Training des Modells effizienter auf vier GPUs parallelisieren. Diese Maßnahme ermöglicht es, die verfügbare Rechenleistung besser auszuschöpfen und gleichzeitig größere Batchgrößen zu verwenden, ohne den GPU-Speicher zu überlasten. Im Rahmen dieser Arbeit wurde, im Vergleich zur ursprünglichen Einstellung mit einer Batchgröße von 6, eine Batchgröße von 44 realisiert. Parallel dazu wurde die Lernrate auf das Fünffache des ursprünglichen Werts erhöht.

### 5.2.1 Überwachung des Trainingsablaufes mit angepassten Parametern

Zur Analyse der Auswirkungen dieser Änderungen wurden beide Modellvarianten, die angepasste und die ursprüngliche, jeweils mit identischer Seed-Initialisierung trainiert. Die Ergebnisse der Cross-Validation über den Trainingsverlauf hinweg sind in Abbildung 5.1 (Modell A) und Abbildung 5.2 (Modell B) dargestellt.

#### Modell A

Wie Abbildung 5.1 zeigt, erzielt das Modell mit großer Batchgröße (BS=44) und erhöhter Lernrate bereits nach wenigen Tausend Schritten deutlich höhere SDR-Werte als die Standardkonfiguration (BS=6, LR=0.0001). Der anfängliche starke Leistungszuwachs deutet darauf hin, dass die neuen Parameter ein schnelleres Lernen ermöglichen. Nach etwa 30 000 Trainingsschritten ist jedoch ein Leistungsmaximum erreicht. Im weiteren Verlauf sinken die SDR-Werte leicht.

Dieses Verhalten lässt sich auf zwei Effekte zurückführen. Erstens kann eine zu hohe Lernrate in Verbindung mit großen Batches dazu führen, dass das Modell das Minimum der Loss-Funktion nicht stabil halten kann.

#### Modell B

Auch beim Training von Modell B zeigt sich ein ähnliches Verhalten. In der Variante mit großer Batchgröße und erhöhter Lernrate steigen die SDR-Werte zunächst stark an und erreichen bereits nach 20 000 Schritten ein erstes Leistungshoch (siehe Abbildung 5.2). Anschließend stagnieren die Ergebnisse oder sinken leicht.

Im Gegensatz dazu zeigt das Modell mit kleiner Batchgröße einen langsameren, aber kontinuierlichen Leistungsanstieg. Ab etwa 100 000 Schritten beginnt es, das schnellere Modell zu in den Evaluierungsergebnissen zu überholen. Diese Entwicklung legt nahe, dass kleinere Batchgrößen, trotz geringerer Anfangsstabilität, langfristig zu besserer Generalisierung führen können.

### 5.2.2 Ergebnisse der Validierung

Die validierten SDR-Ergebnisse für beide Modelle sind in Tabelle 5.1 zusammengefasst. Während Modell A mit angepasster Lernrate und größerer Batchgröße nach nur 30 000 Schritten bessere SDR-Werte als die Referenz erzielt, bleibt *Modell B* unter vergleichbaren Bedingungen hinter der Referenz zurück. Insbesondere die Konfiguration mit nur zwei Checkpoints liefert bei Modell A bereits bessere Ergebnisse als das ursprüngliche Modell mit drei Checkpoints. Dies entspricht einer Reduktion des Trainingsaufwands um ca. 80 %.

#### Interpretation der Unterschiede

Die unterschiedliche Wirkung der Trainingsanpassungen bei Modell A und B lässt sich auf die Natur der zugrunde liegenden Trainingsdaten zurückführen. Modell A wurde mit dem *Label-Noise*-Datensatz trainiert. Dieser ist strukturell klarer und weist weniger komplexe Überlappungen auf.

Modell B hingegen basiert auf dem *Bleeding*-Datensatz, bei dem akustische Überschneidungen zwischen den Quellen auftreten. Diese sind schwieriger zu entkoppeln. Bei solchen komplexen Daten können aggressive Trainingsparameter wie hohe Lernraten und große Batchgrößen dazu führen, dass das Modell vor allem dominante Muster lernt und subtilere Strukturen vernachlässigt. Dadurch wird die Modellleistung im weiteren Verlauf begrenzt.

#### Zusammenfassung

Insgesamt führen die vorgenommenen Anpassungen bei Modell A zu einer signifikanten Steigerung der Trainingsgeschwindigkeit und -effizienz sowie zu einer verbesserten Modellqualität. Im Fall von Modell B lässt sich zwar ebenfalls eine deutliche Reduktion der Trainingsdauer beobachten, diese geht jedoch mit einem leichten Rückgang der Trennungsqualität um etwa 0,1 dB (SDR) einher. Um das optimale Verhältnis zwischen Trainingsaufwand und Modellleistung für Modell B zu bestimmen, sind weiterführende Experimente mit feinjustierten Hyperparametern erforderlich.

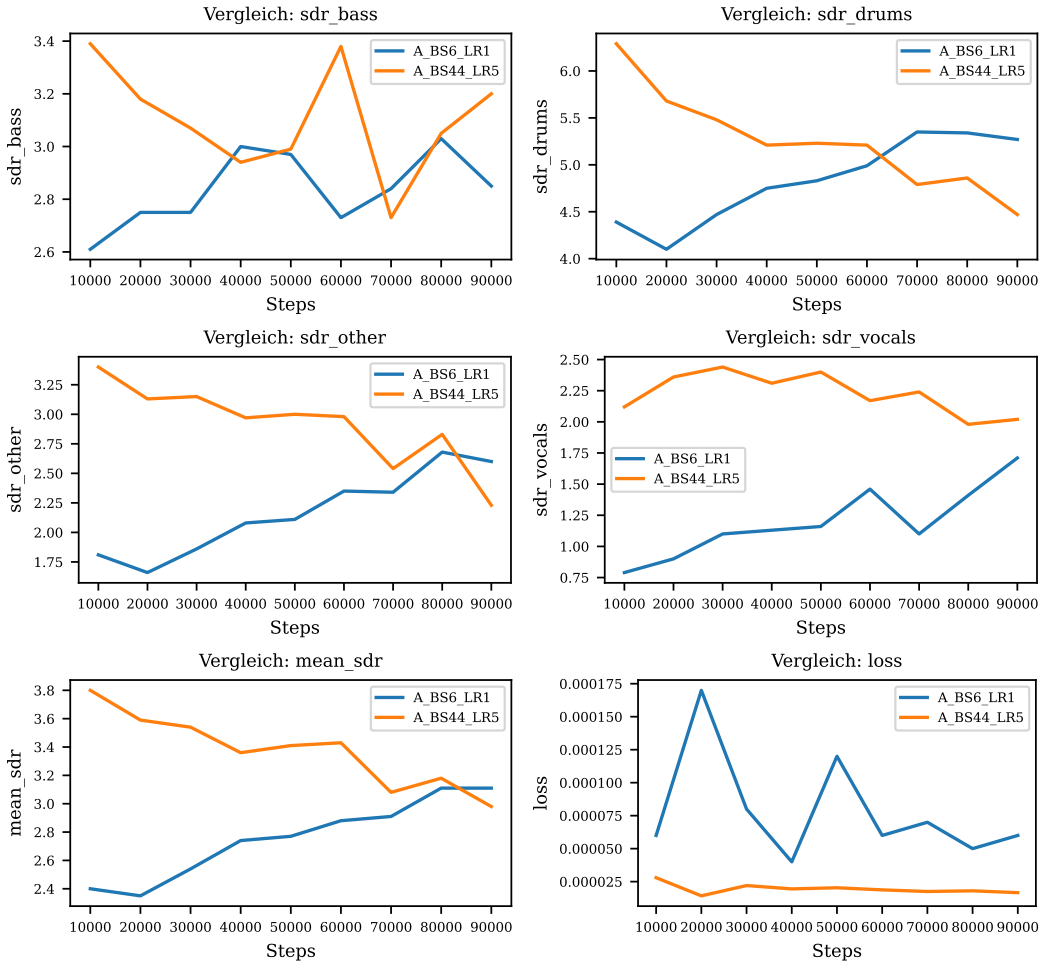


Figure 5.1: Vergleich der Metriken für Modell A über die Trainingszeit. Die Metriken wurden mit einem jeweils einem Seed und dem für die Cross-Validierung erstelltem Datensatz erstellt. (Steps=Anzahl der Trainingsschritte, A_BS44_LR5 zeigt das Modell mit höherer Batchgröße und fünfacher Lernrate, A_BS6_LR1 zeigt ein Modell mit Batchgröße von 6 und Lernrate von 0.0001)

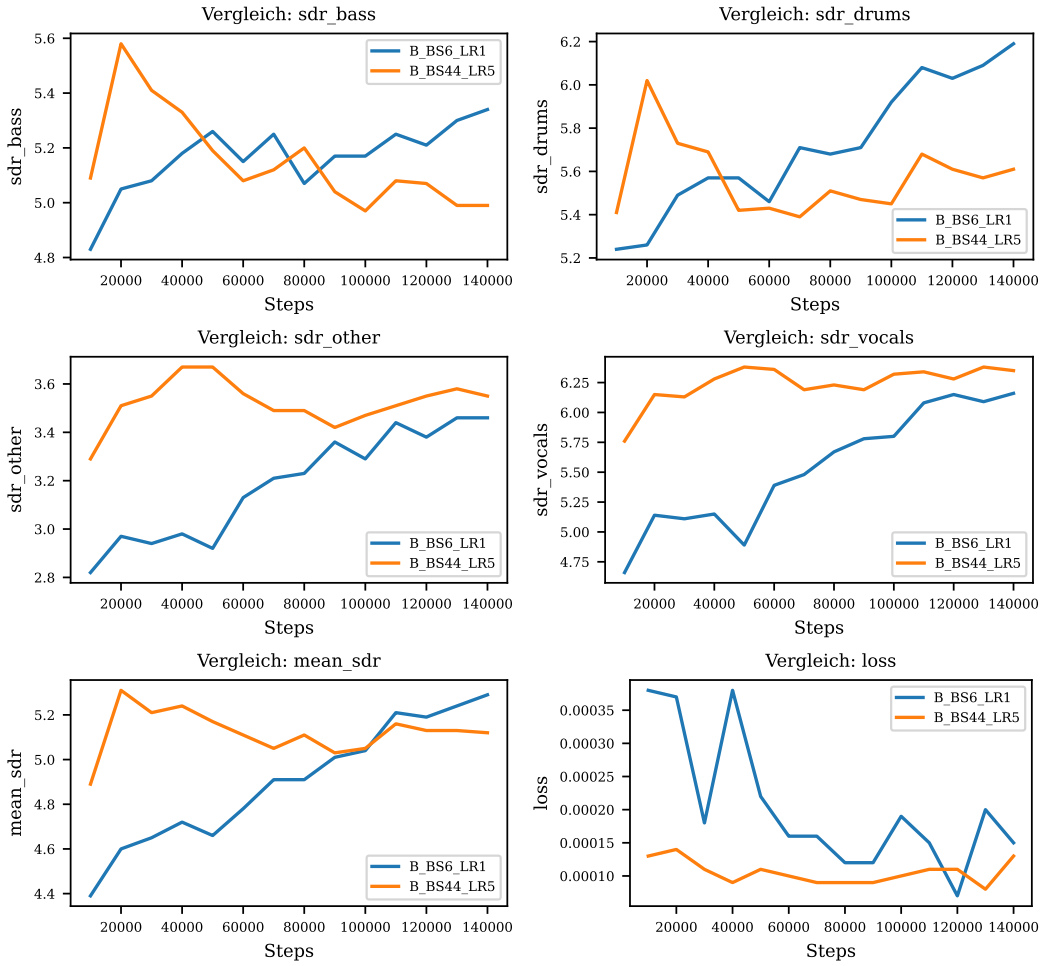


Figure 5.2: Vergleich der Metriken für Modell B über die Trainingszeit. Die Metriken wurden mit einem jeweils einem Seed und dem für die Cross-Validierung erstelltem Datensatz erstellt. (Steps=Anzahl der Trainingsschritte, B_BS44_LR5 zeigt das Modell mit höherer Batchgröße und fünfacher Lernrate, B_BS6_LR1 zeigt ein Modell mit Batchgröße von 6 und Lernrate von 0.0001)

| | | | | | SDR | | | |
|---|---|---|---|---|---|---|---|---|
| Modell | CP | BS | LR | Steps | Bass | Drums | Other | Vocals | Mean |
| Referenz A | 3 | 6 | 0.0001 | 100k | 5.88 | 7.07 | 4.54 | 6.98 | 6.12 |
| A | 3 | 44 | 0.0005 | 30k | 5.97 | 7.51 | 4.76 | 7.36 | 6.41 |
| A | 2 | 44 | 0.0005 | 30k | 5.86 | 7.25 | 4.65 | 7.25 | 6.26 |
| Referenz B | 3 | 6 | 0.0001 | 150k | 5.89 | 7.27 | 4.82 | 7.05 | 6.26 |
| B | 3 | 44 | 0.0005 | 20k | 5.91 | 7.05 | 4.77 | 6.91 | 6.16 |
| B | 2 | 44 | 0.0005 | 20k | 5.9 | 6.94 | 4.7 | 6.8 | 6.09 |

Table 5.1: Auswertung der SDRs. CP=Anzahl der Checkpoints, BS=Batchgröße, LR=Lernrate, Steps=Anzahl der Schritte (k=kilo, Anzahl wird mit 1000 multipliziert). `Referenz A` und `Referenz B` entsprechen den SDR Werten, der lokal evaluierten Originalmodellen.

# 6 Fazit und Ausblick

In dieser Bachelorarbeit wird das Ziel verfolgt, die wissenschaftliche und praktische Nutzbarkeit des TFC-TDF-UNet V3 Modells zur Musikquellentrennung gezielt zu verbessern. Im Mittelpunkt stehen dabei drei zentrale Aspekte. Die verständliche Aufarbeitung der Architektur des TFC-TDF-UNet V3 Modells, die Erhöhung der Reproduzierbarkeit und Modularität des Trainingsprozesses sowie die Optimierung der Trainingsgeschwindigkeit bei gleichbleibender Modellleistung.

Im ersten Teil dieser Arbeit wird die Architektur des TFC-TDF-UNet V3 Modells eingehend analysiert und systematisch dokumentiert. Ziel ist es, die Funktionsweise des Modells transparent darzustellen, alle relevanten Komponenten nachvollziehbar zu beschreiben und somit eine fundierte Grundlage für weiterführende Experimente und wissenschaftliche Analysen zu schaffen.

Im zweiten Teil liegt der Fokus auf der Analyse der ursprünglichen Trainingsumgebung, insbesondere im Hinblick auf nicht-deterministische Aspekte des Trainingsprozesses. Es zeigt sich, dass selbst bei identischer Konfiguration unterschiedliche Trainingsläufe zu teils stark abweichenden Ergebnissen führen. Durch die Einführung einer systematischen Snapshot-Validierung in Kombination mit Cross-Validation kann ein nachvollziehbares und reproduzierbares Trainingsverhalten etabliert werden. Parallel dazu wird die bestehende Codebasis modularisiert, sodass neue Modelle mit deutlich geringerem Aufwand trainiert und evaluiert werden können. Dies erhöht die Transparenz des Workflows und erleichtert zukünftige Forschung sowie die Erweiterung der Modellarchitektur.

Im dritten Teil der Arbeit steht die Effizienzsteigerung des Trainingsprozesses im Mittelpunkt. Die Integration von *Distributed Data Parallel* (DDP) ermöglicht die parallele Nutzung mehrerer GPUs, wodurch sich die Trainingszeit signifikant reduzieren lässt. Darüber hinaus kann der Einsatz größerer Batchgrößen in Verbindung mit einer angepassten Lernrate zu einer weiteren Beschleunigung des Trainings führen. Dabei ist jedoch zu beachten, dass eine verkürzte Trainingsdauer nicht zwangsläufig mit einer besseren Modellleistung einhergeht. Insbesondere dann, wenn die Stabilität der Optimierung oder die Generalisierungsfähigkeit beeinträchtigt wird.

Insgesamt zeigt diese Arbeit, dass durch gezielte technische und strukturelle Anpassungen sowohl die wissenschaftliche Qualität (im Sinne von Reproduzierbarkeit, Vergleichbarkeit und Modularität) als auch die praktische Effizienz (hinsichtlich Trainingsdauer und Ressourcennutzung) eines bestehenden Deep-Learning-Modells erheblich verbessert werden kann. Die vorgestellten Maßnahmen bilden eine belast-

bare Grundlage für weiterführende Forschung im Bereich der Musikquellentrennung und liefern darüber hinaus wertvolle Impulse für die allgemeine Optimierung neuronaler Netze im audiobasierten Kontext.

## 6.1 Ausblick und zukünftige Arbeiten

Aufbauend auf den in dieser Arbeit gewonnenen Erkenntnissen ergeben sich verschiedene Ansätze für weiterführende Arbeiten im Bereich der musikbasierten Quellentrennung mit Deep Learning.

Ein naheliegender Forschungsschwerpunkt liegt in der Weiterentwicklung der Loss-Funktion. In dieser Arbeit wird eine quantilbasierte Variante des L2-Losses verwendet, die gezielt stabile Trainingsbeispiele priorisiert. Zukünftige Arbeiten könnten untersuchen, inwieweit Loss-Funktionen, basierend auf spektralen oder psychoakustischen Merkmalen, zu einer subjektiv höheren Klangqualität führen, selbst wenn objektive Metriken wie der SDR unverändert bleiben.

Ein praktischer Aspekt zukünftiger Arbeiten betrifft die Komprimierung und Optimierung des Modells für den Einsatz auf ressourcenbeschränkten Geräten.

Schließlich wäre es denkbar, benutzergesteuerte Quellentrennungsmethoden zu erforschen, bei denen Nutzer gezielt steuern können, welche Quellen extrahiert werden sollen, etwa durch Textprompts, manuelle Markierungen oder Beispieleingaben. Dies könnte insbesondere für Anwendungen im Musikproduktionsbereich oder bei interaktiven Tools relevant sein.

Zusammenfassend bietet das Thema Musikquellentrennung durch Deep Learning, sowohl im Hinblick auf Modellarchitektur und Trainingsverfahren als auch in Bezug auf praktische Anwendungen und Nutzbarkeit, eine Vielzahl spannender Weiterentwicklungsmöglichkeiten.

# Bibliography


[AAA+23] S Ansari, AS Alatrany, KA Alnajjar, T Khater, S Mahmoud, D Al-Jumeily, and AJ Hussain. A survey of artificial intelligence approaches in blind source separation. *Neurocomputing*, 561, October 2023.

[Aga19] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019.

[All77] J. Allen. Short term spectral analysis, synthesis, and modification by discrete fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(3):235–238, 1977.

[BA83] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983.

[Dé22] Alexandre Défossez. Hybrid spectrogram and waveform source separation, 2022.

[FUL+24] Giorgio Fabbro, Stefan Uhlich, Chieh-Hsin Lai, Woosung Choi, Marco Martínez-Ramírez, Weihsiang Liao, Igor Gadelha, Geraldo Ramos, Eddie Hsu, Hugo Rodrigues, Fabian-Robert Stöter, Alexandre Défossez, Yi Luo, Jianwei Yu, Dipam Chakraborty, Sharada Mohanty, Roman Solovyev, Alexander Stempkovskiy, Tatiana Habruseva, Nabarun Goswami, Tatsuya Harada, Minseok Kim, Jun Hyung Lee, Yuanliang Dong, Xinran Zhang, Jiafeng Liu, and Yuki Mitsufuji. The sound demixing challenge 2023 – music demixing track. *Transactions of the International Society for Music Information Retrieval*, 7(1):63–84, 2024.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[HG23] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

[HRS16] Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent, 2016.

[HZRS15a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[HZRS15b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[KCC+21] Minseok Kim, Woosung Choi, Jaehwa Chung, Daewon Lee, and Soonyoung Jung. Kuielab-mdx-net: A two-stream neural network for music demixing, 2021.

[KLJ23] Minseok Kim, Jun Hyung Lee, and Soonyoung Jung. Sound demixing challenge 2023 music demixing track technical report: Tfc-tdf-unet v3, 2023.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBH15] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[LDG+17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[LY22] Yi Luo and Jianwei Yu. Music source separation with band-split rnn, 2022.

[LZV+20] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.

[PAKV23] Igor Pereira, Felipe Araújo, Filip Korzeniowski, and Richard Vogl. Moisesdb: A dataset for source separation beyond 4-stems, 2023.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning

representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[RLS+19] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. MUSDB18-HQ - an uncompressed version of musdb18, December 2019.

[Sha49] C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[SLI18] Fabian-Robert Stöter, Antoine Liutkus, and Nobutaka Ito. The 2018 signal separation evaluation campaign, 2018.

[UVL17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.