Verarbeitung von Sequenzen mit Neuronalen Netzen

Steffen Heike Georg Illium

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität München

vorgelegt von
Steffen Heike Georg Illium

Tag der Einreichung: 31.10.2023

Verarbeitung von Sequenzen mit Neuronalen Netzen

Steffen Heike Georg Illium

Dissertation

an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität München

vorgelegt von
Steffen Heike Georg Illium

1. Berichterstatter:	Prof. Dr. Claudia Linnhoff-Popien
2. Berichterstatter:	Prof. Dr. Jörg Hähner
Tag der Einreichung:	31.10.2023
Tag der Disputation:	02.08.2024

Eidesstattliche Versicherung

(siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Steffen Heike Georg Illium

Augsburg, der 10. September 2024

Zusammenfassung

Die Verarbeitung von geordneten Daten (Sequenzen) hat seit Längerem in vielen Bereichen der Informatik und damit auch in unserem alltäglichen Leben Einzug gehalten. Sei es bei offensichtlichen Anwendungsfällen, wie der Verarbeitung von Messreihen und der Vorhersage von Trends, z. B. in meteorologischen Systemen, als auch bei nicht offensichtlichen Anwendungen, wie der Interpretation nicht sequenzieller Datentypen. Wie in vielen anderen Teilbereichen der Informatik hat die Anwendung selbstlernender Algorithmen auch hier aufgrund ihrer ausgezeichneten Generalisierungsfähigkeiten jüngst zugenommen. Dabei ist das Lernen an viele Einflüsse gebunden, so eignen sich viele bekannte Algorithmen z. B. für einen konkreten Anwendungsfall, verfehlen aber ihr Leistungsziel bei artverwandten Datensätzen. Hier müssen passende Hyperparameter-Konfigurationen oder andere, geeignetere *Architekturen* gefunden werden. Die vorliegende Arbeit bearbeitet Fragestellungen aus diesem Teilbereich der Anwendung tiefer neuronaler Netzwerke im Kontext sequenzieller Daten (*Deep Neural Network*) in fünf Kapiteln.

In der vorliegenden Arbeit wird zunächst die generelle Funktionsweise des Informationstransports im sog. *Hidden Vector* untersucht. Dies spielt bei der Verarbeitung von Sequenzen typischen *Architektur* sog. *Recurrent Neural Network* eine besondere Rolle. Dabei lässt sich eine harte Grenze der sog. *Memory Horizon* feststellen, die mit der Anzahl an verwendeten *Neurons* korreliert. Auch verwandte *Zelle-Architekturen* werden hierbei untersucht.

Anschließend beschäftigt sich diese Arbeit mit alternativen *Architekturen* in der Sequenzverarbeitung im Bereich der Audio-Klassifizierung. So werden zwei *Architekturen* für die Verarbeitung von Mel-Spektrogrammen von Audio-Signalen aus *Real-World*-Datasets eingeführt und im Kontext bestehender Ansätze, unter Beachtung verschiedener *Data Augmentation (DA)*-Methoden, evaluiert. Dann werden etablierte *Data Augmentation*-Methoden in einem neuen Verfahren („Transport“) zusammengefasst und evaluiert. Erste Ergebnisse lassen die Hoffnung zu, dass dieser neue Ansatz nicht nur die Varianz steigern, sondern auch einen Leistungsgewinn im Bereich der Audio-Klassifizierung erzielen kann.

Schließlich werden *Self-Replicating Neural Network* betrachtet, deren Lerndaten sich fortlaufend (sequenziell) verändern, bis diese auf sich selbst abgebildet werden können. Durch die Erweiterung dieses Forschungsfeldes auf Aufgaben aus dem Bereich des *Deep Learning* ist das Phänomen des natürlichen *Dropout* (*Pruning*), durch einen aus solchen *Particle Network* zusammengesetzten Organismus (*Organism Network*) zu beobachten.

Abstract

The processing of ordered data (sequences) has been used for a long time in many areas of computer science, and thus also in our everyday life. Be it in obvious use cases, such as the processing of series of measurements and the prediction of trends, meteorological systems e.g., as well as in non-obvious applications, such as the interpretation of non-sequential data types. As in many other subfields of computer science, the use of self-learning algorithms has recently increased due to their excellent generalization capabilities. Thereby, learning is bound to many influences, e.g., many known algorithms are suitable for a concrete use case, but fail to achieve their performance goal on related or slightly altered data sets. Then, only suitable hyperparameter configurations or entirely other more appropriate architectures must be found. This thesis addresses issues from this subfield of applying deep neural networks in the context of sequential data in five chapters.

In this thesis, we first examine the general operation of information transport in hidden vectors. A typical problem when dealing with architectures (Recurrent Neural Networks), specialized in processing sequences. A hard limit, the so-called *memory horizon*, can be found, which correlates with the number of neurons used. Related cell types are also examined here.

Next, this thesis deals with alternative architectures in sequence processing in the field of audio classification. Thus, two candidates for processing mel spectrograms of audio signals from real-world datasets are introduced and evaluated in the context of existing approaches, considering different data augmentation techniques. Then, established data augmentation methods are combined and evaluated in a newly proposed method („transport“). Initial results give hope that this new approach can not only increase variance, but also gain performance in audio classification.

Finally, we consider self-replicating neural networks, which learn on continuously changing data sources (sequentially) until it can be mapped onto itself. By extending this field of research to tasks from the image classification domain, the phenomenon of natural pruning can be observed.

Inhaltsverzeichnis

Danksagung

Einen ganz besonderen Dank möchte ich zunächst an Frau Dr. Claudia Linnhoff-Popien richten, die mich während der Betreuung der vorliegenden Dissertation stets unterstützt und uns als Lehrstuhl mit vielen einmaligen Erlebnissen und unerwarteten Möglichkeiten immer wieder überrascht hat. Ebenso möchte ich meinem Zweitgutachter Prof. Dr. Jörg Hähner für seine Geduld und die Inspiration danken. Ohne Ihre spannende Vorlesung hätten mich selbstlernende Systeme wahrscheinlich zu spät interessiert. Auch dem Kommissionsvorsitzenden Prof. Dr. Schubert und dem Ersatzprüfer Prof. Dr. Schmidt möchte ich für ihre Unterstützung danken.

Mein Dank geht natürlich auch an meine Kollegen des Lehrstuhls für mobile und verteilte Systeme der LMU München für eine kreative, konstruktive und einfach wundervolle Zeit mit euch allen. Den folgenden Personen möchte ich aber einen besonderen Dank aussprechen: Dr. Robert Müller für die stete gegenseitige Unterstützung in Projekten und Forschung sowie die gemeinsame Zeit. Dr. Thomas Gabor für die kontinuierliche Unterstützung, die Diskussion unserer manchmal auch verrückt anmutenden Ideen sowie die inspirierenden Worte. Dr. Andreas Sedlmeier für die Geduld, die Unterstützung und das Nachhaken bei noch so kleinen Ungenauigkeiten sowie für sein technisches Fachwissen. Michael Kölle dafür, dass er scheinbar Tag und Nacht arbeitet und ein ausgesprochen guter Kollege ist. Dr. Markus Friedrich für seine immer positive Art, die gemeinsame Zeit und Forschung. Fabian Ritz für seine übersprudelnde Energie und Begeisterung. Maximilian Zorn, dass er ein hervorragender und starker Student und dann ein guter Kollege war. Kyrill Schmid für seine stets unterstützende und aufmerksame Art. Thomy Phan für die lehrreichen Gespräche, gerade zu Beginn meiner Promotion. Sowie Dr. Sebastian Feld, der mich an der LMU vorgestellt hat. Danke.

Zuletzt möchte ich meiner Familie danken, die mir diesen Weg ermöglicht hat. Allen voran meinen Großeltern, die wirklich gute Menschen sind oder waren. Auch meinen Eltern möchte ich danken, die mich immer geduldig unterstützen. Mein Dank gilt aber auch meiner Freundin Hanna, auf die ich besonders in harten Zeiten zählen kann, die mir die Welt zeigt und die mit viel Liebe und Geduld bei mir ist.

Vielen Dank, dass ich meinen Weg bis hierhin mit euch allen teilen durfte.

1 Einleitung und Motivation

Die Welt ist im Wandel. Das bedeutet, wir leben auf der Achse einer strukturierten, gerichteten und topologisch aufgebauten Dimension: der Zeit. Diese Beobachtung schlägt sich in allen uns umgebenden Dingen wieder: Die Uhr tickt, ein Schritt folgt dem anderen. Weder ist die Frequenz dabei für alle Dinge gleich, noch ist die inhärente Funktion für jedes Phänomen konstant. Im Bestreben, die Welt um uns herum abzubilden, ist unausweichlich ein Format entstanden, welches diese Eigenschaften implizit vereint: die Sequenz. Seien es Messdaten einer Wetterstation, Nullen und Einsen einer Bitfolge, Einträge der Planeten- und Sternenbeobachter der Renaissance oder Noten in einer Klaviatur, all diese Dinge haben eines gemeinsam: Informationen stecken nicht nur in einem einzigen Datenpunkt, sondern in der Reihung dieser. Sequenzen ermöglichen es, aus wenigen Beobachtungen erweiterte Schlüsse zu ziehen. Eine Messung der Temperatur sagt z. B. etwas über den aktuellen Umgebungszustand aus. Erhebt man allerdings schon mehr als einen Datenpunkt, kann durch die Beobachtung der Veränderung im Kontext der Folge, aus einer einfachen eine erweiterte Aussage („es ist kälter geworden“) getroffen werden. Durch wieder mehr Messpunkte kann sogar ein Trend („es wird kälter werden“) abgeleitet werden. Die Erfassung deutlich längerer Zeiträume führt schließlich zur Etablierung von statischen Modellen zur Erklärung oder Modellierung des hier beobachteten, bzw. abgetasteten Phänomens. Es erschließt sich, dass das Analysieren der verborgenen Funktion auch im Kontext moderner informationstechnischer Verfahren eine übergeordnete Rolle spielt.

Wie viele andere Bereiche auch, wird die Forschung an sequenziellen Daten jüngst durch Methoden des *Machine Learning (ML)* revolutioniert. *Neural Network (NN)* erlauben es, sich den in Datenreihen verborgenen, teilweise komplexen Funktionen mit immer höherer Genauigkeit anzunähern. So werden Muster erkannt, durch die ein *Model* etabliert (gelernt) werden kann, welches es erlaubt, verallgemeinerte Aussagen über neue Eingaben zu treffen (*Generalization*) oder Vorhersagen zu berechnen (*Forecasting*). Dies führt gerade im Bereich von meteorologischen Simulationen zu immer neuen und detaillierteren *Model* [22]. Doch Wetter- bzw. Klimadaten sind, wie bereits erwähnt, nur ein Anwendungsfall solcher *Deep Neural Network (DNN)*, so können unter anderem Zufallsfolgen, Audio-Aufnahmen, Bilder oder Generationen selbst-lernender NNs als Sequenz analysiert werden. So haben sich unterschiedliche *Models* (z. B. *Recurrent Neural Network (RNN)*, *Transformer*) und Verfahren etabliert (Mel-Spektrogramm), welche an die Besonderheiten der Sequenzverarbeitung

angepasst sind. Diese Arbeit möchte sich nun mit verschiedenartigen Sequenzen oder Sequenzen-verarbeitenden Konzepten im Kontext von NNs beschäftigen, um mehrere Fragen zu beantworten. Inwieweit und wie lange bleiben z. B. Informationen über viele Verarbeitungsschritte hinweg erhalten und wie wirken sich verschiedene *Architekturen* auf diese Fähigkeit aus? Ist die Leistungsfähigkeit (*Performance*) von NN-*Models* auf allen Datensätzen gleich gut und welche Rolle spielt DA? Können *Attention*-basierte NNs auch auf als Folge interpretierte Mel-Spektrogramme angewendet werden und wie schneiden solche *Models* im Vergleich zu etablierten Verfahren ab? Kann die Neu-Einbettung (*Transport*) von Informationen in ihrem eigenen Kontext einen Vorteil gegenüber vergleichbaren DA-Methoden bringen? Und welche Anwendungsfälle ergeben sich durch sequenziell lernende *Self-Replicating Neural Network (SRNN)*? Diese und andere Fragen sollen in der vorliegenden Arbeit behandelt werden. Dies geschieht in mehreren Kapiteln, deren Struktur im Folgenden erläutert wird.

1.1 Struktur

Nach einer allgemeinen Einführung der gemeinsamen technischen Grundlagen (Seite 9) gestaltet sich die Arbeit in fünf Inhaltskapiteln. Zunächst erfolgt die Untersuchung (Seite 29) von RNN-Varianten hinsichtlich ihrer Informationstransportfähigkeit (*Memory Horizon*), um zu klären, wie groß ein Sequenz verarbeitendes NN sein sollte, um den Einfluss singulärer Ereignisse auf eine Funktion hinreichend abzubilden. Durch die Einführung eines grundlegenden Experiments und einer neuen Blickweise auf die Analyse der Informationstransporteigenschaften verschiedener RNN-Varianten soll eine Grundlage zur besseren Implementation solcher *Architekturen* geschaffen werden.

Eine mögliche Anwendung solcher und anderer NN-*Architekturen* findet sich im zweiten Themenkapitel wieder. Die Mel-Spektrogramm-basierte Audiodaten-Klassifikation ist motiviert durch die zunehmende Verfügbarkeit von Audiodaten und den Bedarf an automatisierter Klassifizierung und Segmentierung dieser. Das Anwendungsfeld dieser Methoden liegt primär, aber nicht nur, in der Spracherkennung und der akustischen Überwachung und ist hier von essenzieller Bedeutung. Nach der Einführung des Konzepts (Seite 49) und der Präsentation erster Experimente an einem *Real-World*-Dataset, bei dem auch ersichtlich wird, welche Rolle DA hat, wird im Kontext dieses Datenformates ein neues *Attention*-basiertes Verfahren in Kapitel 5 eingeführt und gegen etablierte Ansätze evaluiert.

Wie sich gezeigt hat, spielt DA vorwiegend in Bezug auf *Real-World*-Datasets eine übergeordnete Rolle im Kontext der Anwendung von NNs. Daher soll in diesem nächsten Kapitel (Seite 99) ein Überblick über existierende Methoden etabliert, deren Eigenschaften herausgearbeitet und in einem neuen DA-Verfahren vereint werden. Durch die Verwendung von Voronoi-Diagrammen, einer geometrischen Methode zur Partitionierung des Raums, soll die Vielfalt

in Trainingsdaten für CNNs mittels *Feature*-Transport erhöht werden. Es zeigt sich, dass dieses Verfahren geeignet ist, um die Varianz der mit VP trainierten *Models* gegenüber anderen Methoden zu senken.

Das fünfte Inhaltskapitel (Seite 119) beschäftigt sich mit der Weiterentwicklung von sog. SRNNs, einer neuartigen Form von NN, die ihre eigenen Trainingsdaten sequenziell anpassen und so stabile, replikative Zustände (*Weight*-Vektoren) erreichen können. Um eine Brücke von der bereits etablierten, allerdings noch theoretischen Forschung, hin zu einer möglichen Anwendung solcher Werkzeuge zu schaffen, wird eine als kollaborativ interpretierte Struktur von Individuen eingeführt: das *Organism Network (ON)*. Während der experimentellen Exploration ist ein Phänomen, das „natürliche *Pruning* “, zu beobachten, was in ersten Experimenten eine effizientere Ausdünnung der NN-*Weights* ermöglicht als vergleichbare Methoden. Diese Entdeckung ist von besonderem Interesse, da durch dieses Verfahren NNs, bei annähernd gleichbleibender Leistung, ausgedünnt und so die Rechenanforderungen reduziert werden können.

Insgesamt liefert diese Arbeit durch die hier betrachteten Themen, einen breiten Blick sowohl auf Bereiche der Grundlagenforschung als auch auf die Anwendung von NNs im Kontext sequenzieller Daten oder der sequenziellen Verarbeitung dieser. Dabei ist das Ziel, sowohl das Verständnis der vorgestellten *Models* zu verbessern als auch effektivere oder effizientere Methoden für die Anwendung von NNs zu entwickeln.

1.2 Vorveröffentlichungen des Autors

Die im Rahmen dieser Arbeit aufgegriffenen, zusammengefassten und erweiterten Forschungsinhalte sind zum Zeitpunkt der Anfertigung dieser Arbeit bereits durch den Autor auf internationalen Fachkonferenzen vorgestellt, diskutiert und veröffentlicht worden. Das erweiterte Fachgebiet der Themen umfasst dabei ML durch DNN, sowie fachspezifische Themen im Kontext der Analyse von Audiodaten und der *Society of Artificial Life*.

In diesem Abschnitt soll auf die entsprechenden wiss. Veröffentlichungen eingegangen, der Eigenanteil des Autors dabei herausgestellt und die Mitarbeit weiterer beteiligter Personen abgegrenzt werden. Die folgende Aufzählung ist thematisch sortiert und folgt damit dem Erscheinen der Inhalte in den nachfolgenden Inhaltskapiteln Kapitel 3, 4, 5, 6 & 7.

Zu Beginn sei Dr. Claudia Linnhoff-Popien in der Rolle der Doktormutter des Autors und als konstruktive Diskussionspartnerin im Rahmen der nachfolgend zitierten Arbeiten genannt. Sie stand immer beratend zur Seite und war eine konstruktive Diskussionspartnerin, vorwiegend dann, wenn sie als Co-Autorin genannt ist.

- **„Empirical Analysis of Limits for Memory Distance in Recurrent Neural Networks“** [105]

  Diese Arbeit beschäftigt sich mit der Untersuchung von Informationstransport in RNNs und Varianten (*Long Short-Term Memory (LSTM)* & *Gated Recurrent Unit (GRU)*). Dabei wird im Besonderen der sog. *Memory Horizon (MH)*, also eine natürliche Grenze des Informationsflusses in verschiedenen RNN-*Architekturen* untersucht. Die einzelnen Kerninhalte dieser Publikation sind dabei zugleich Teil der Kerninhalte von Kapitel 3. Die Eigenanteile des Autors umfassen die grundsätzliche Idee, nach eingehender Recherche, sowie der Umsetzung initialer Experimente und Grafiken sowie der Diskussion der Ergebnisse inkl. der finalen Ausarbeitung. Thore Schillmann, Dr. Robert Müller sowie Dr. Thomas Gabor waren sowohl an der Recherche zum Thema, Diskussion und Konzept beteiligt, Dr. Thomas Gabor dabei in besonderem Maße. Im Rahmen seiner Bachelorarbeit arbeitete Thore Schillmann an den Experimenten mit und führte sie in einem erweiterten Lauf aus.

- **„Surgical Mask Detection with Convolutional Neural Networks and Data Augmentations on Spectrograms“** [101]

  Diese Arbeit beschäftigt sich vordergründig mit der binären Audio-Klassifikation eines *Real-World*-Dataset. Im Kern wird allerdings die Exploration von DA-Methoden im Kontext sequenzieller Daten (hier Audio) in Form von Mel-Spektrogrammen behandelt. Überdies wird ein bestehender Ansatz im Kontext dieses Datensatzes evaluiert und durch die Überführung in eine neue NN-*Architektur* hinterfragt. Die einzelnen

Kerninhalte dieser Publikation sind dabei zugleich Teil der Kerninhalte von Kapitel 4. Die Eigenanteile des Autors an dieser Arbeit umfassen die grundsätzliche Idee sowie deren Umsetzung in zahlreichen Experimenten und Grafiken sowie die finale Ausarbeitung. Dr. Robert Müller sowie Dr. Andreas Sedlmeier waren sowohl an der Recherche zum Thema als auch der Diskussion des Konzepts maßgeblich beteiligt.

- **„Visual transformers for primates classification and covid detection“** [102]

  Anhand zweier *Real-World*-Datasets wird die Arbeit an *Real-World*-Datasets weiter diskutiert und Strategien zum Umgang mit aufkommenden Problemen erörtert. Ferner wird zum ersten Mal der sog. *Vision Transformer (ViT)*, im Kontext von Mel-Spektrogramm basierter Audio-Klassifikation, implementiert und evaluiert. Die Inhalte dieser Publikation nehmen zudem Inhalte aus [101] auf und entwickeln diese in Teilen weiter und sind dabei zugleich Teil der Kerninhalte von Kapitel 5. Die Eigenanteile des Autors an dieser Arbeit umfassen die grundsätzliche Idee sowie deren Umsetzung in initialen und extensiven Experimenten und Anfertigung von Grafiken sowie die finale Ausarbeitung. Dr. Robert Müller sowie Dr. Andreas Sedlmeier waren auch hier sowohl an der Recherche zum Thema als auch der Diskussion des Konzepts beteiligt.

- **„VoronoiPatches: Evaluating a New Data Augmentation Method“** [104]

  In dieser Arbeit wird nach einer extensiven Recherche zu DA im Bereich der Bild-Klassifikation das Konzept eines *Voronoi-Diagramm* basierten DA-Verfahrens durch *Feature*-Transport entwickelt. Die einzelnen Kerninhalte dieser Publikation sind dabei zugleich Teil der Kerninhalte von Kapitel 6. Die Eigenanteile des Autors umfassen hier die Idee und thematische Einordnung von VP, die Konzeption der Experimente und Grafiken, sowie weite Teile der finalen Ausarbeitung. Gretchen Griffin bearbeitete im Rahmen ihrer Bachelorarbeit Teile dieser Veröffentlichung, wie die Recherche der *Related Work*, die Durchführung der Experimente und Anfertigung der Grafiken. Michael Kölle, Maximilian Zorn arbeiteten bei der finalen Ausarbeitung mit. Alle gelisteten Autoren waren an der Diskussion beteiligt.

- **„Self-replication in neural networks“** [67]

  Diese Arbeit führt das Konzept der SRNN (neuronaler Fixpunktgeneratoren) ein. Dabei entwarf Dr. Thomas Gabor das ursprüngliche Konzept, die formale Notation und die Analysemethoden. Er führte auch die wichtigsten Experimente durch. Die Eigenanteile des Autors beziehen sich auf einen Beitrag zur Entwicklung der fortgeschrittenen Konzepte, der Unterstützung der praktischen Experimente und der Durchführung weiterer Analysen auf Basis der Ergebnisse. Andy Mattausch führte im Kontext eines Praktikums vorläufige Experimente durch und lieferte zusammen

mit Prof. Dr. Lenz Belzner einen großen Beitrag für die Diskussionen des ursprünglichen Konzepts.

- **„Goals for Self-Replicating Neural Networks“** [69]

  Diese Arbeit führt das Konzept von Nebenzielen (*Goals*) im Kontext der, von Gabor et al. definierten SRNN, ein [67]. Teile der Definition von *Goals* und SRNN in dieser Veröffentlichung finden sich auch in Abschnitt 7.3 wieder. Der Eigenanteil des Autors umfasst die Mitarbeit an Konzept, Diskussion und finaler Ausarbeitung. Weitere Teile der Arbeit sind den übrigen Autoren zuzuschreiben.

- **„Constructing Organism Networks from Collaborative Self-Replicators“** [103]

  Die letzte Arbeit beschäftigt sich mit der Erweiterung des Konzepts der SRNN [67] und der *Self-Replication (SR)-Goals* [69] durch die Einbettung von *Particle Network (PN)* in eine übergreifende Struktur. Infolge der Experimente wird ein neuartiges Verfahren zum *Pruning* von NN-*Weights* beobachtet und untersucht. Die einzelnen Kerninhalte dieser Publikation sind dabei zugleich Teil der Kerninhalte von Kapitel 7. Die Eigenanteile des Autors umfassen hier die Idee, die Implementierung sowie die Konzeption und Durchführung der Experimente und der Planung der Grafiken. Maximilian Zorn war an der Implementierung und experimentellen Phase beteiligt und wirkte an der finalen Ausarbeitung mit. Zusammen mit Dr. Thomas Gabor unterstützten beide das Konzept und die Diskussion, bei der auch Cristian Lenta und Michael Kölle beteiligt waren.

1.3 Formalien

Im Folgenden wird auf die in dieser Arbeit verwendete Notation und Bildsprache eingegangen.

1.3.1 Notation

Da es sich bei den hier verwendeten Notationen zumeist um mathematische Operationen in oder im Kontext von mehrdimensionalen Matrizen handelt, findet zur eindeutigen Unterscheidung die folgende Notation Verwendung:

Symbol	Operation	Symbol	Bedeutung
$\cdot$	*Multiplikation*	$\times$	Seitenverhältnis
$\odot$	*Skalarprodukt*	x^T	Transposition
$*$	*Kreuzkorrelation*		

Fachworte werden in englischer Sprache verwendet und, wenn möglich, direkt mit dem deutschen Äquivalent (*Equivalent*) genannt. Manche Begriffe wurden gewählt, um eine eindeutige Unterscheidung bereitzustellen. Als Beispiel hier bezeichnet „Parameter“ die Parameter einer ausformulierten Funktion, welche aktiv übergeben und die Funktionsweise maßgeblich steuern. *Weights*, also die Gewichte eines DNN, werden im Deutschen auch als „Parameter“ übersetzt. Zur besseren Unterscheidung innerhalb dieser Arbeit werden daher die lernbaren Gewichte eines DNN als *Weights* angesprochen.

1.3.2 Modell-Abbildungen

In dieser Arbeit werden Abbildungen von NN-Modellen (*Models*) verwendet, um die *Architektur*, d. h. den inneren Aufbau (Folge und Verknüpfungen von sog. Modulen) dieser darzustellen. Abbildung 1.1 ist hierfür beispielhaft.

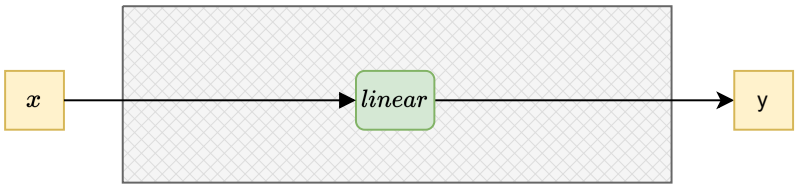


Abbildung 1.1: Als Beispiel: Einfaches NN-Modul. Hier, eine voll vermaschte Schicht ohne Activation-Function; *Feed Forward Neural Network (FFNN)*.

Neben der generellen Struktur wird so auch die Beschaffenheit der einzelnen Module eines *Models* veranschaulicht. Dies folgt dem in Abbildung 1.2 dargestellten Schema, bei dem z. B. orange Kreise arithmetische Funktionen im Allgemeinen darstellen. Die genaue Funktion ist in der Mitte des Symbols (hier x; *Matrix-Multiplikation (MM)*) angegeben.

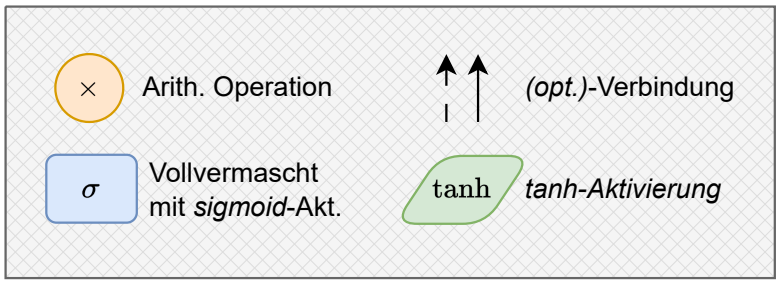


Abbildung 1.2: Verwendetes Schema der Abbildung NN-*Models*.

2 Methodische Grundlagen

In diesem Kapitel soll zunächst die allgemeine technische Grundlage dieser Arbeit eingeführt werden. Dabei wird ein generelles Verständnis informationstechnischer Verfahren vorausgesetzt, sodass sich dieses Grundlagenkapitel vorwiegend auf Methoden des ML im Kontext von NN konzentrieren kann. Spezielle Konzepte, wie unterschiedliche NN-*Architekturen*, werden dann jeweils in Grundlagen-Abschnitten der folgenden Kapitel eingeführt: Abschnitt 3.2, 2.4.1, 6.1 & 7.3. Beginnend *Machine Learning* wird abschließend eine Überleitung zu *Deep Learning (DL)*-Verfahren und dem NN-Training etabliert. Dann wird das generelle Konzept der sog. *Hyperparameter*, sowie Besonderheiten im Umgang mit DNN, eingeführt, um dann im Anschluss eine grundlegende Definition von sequenziellen Daten zu formulieren.

2.1 *Machine Learning* Models

Ganz grundsätzlich kann man zwei verschiedene Gruppen von ML-*Model* unterscheiden: *generative* und *discriminative Models*. Als *generative* werden statistische *Models* dann bezeichnet, wenn sie in der Lage sind, eine bedingte Wahrscheinlichkeit $P(Y|X)$ (*posterior*) einer unbekannten Zielfunktion $f : x \rightarrow y, y \in Y, x \in X$ zu approximieren. Hierzu wird zunächst die gemeinsame Wahrscheinlichkeit von $P(X, Y)$ beobachtet, um dann die *Bayes'sche Regel* anzuwenden; so ergibt sich [174]:

$$P(Y|X) = \frac{P(Y) \cdot P(X|Y)}{P(X)} \tag{2.1}$$

oder anders formuliert:

$$Posterior = \frac{\text{(Y-Prior)} \cdot \text{(X geg. Y)}}{\text{X-Prior}} \tag{2.2}$$

Rein durch die gemeinsame Beobachtung $(X \cap Y)$ sowie die individuelle Beobachtung X kann durch Hinzunahme einer Vorvermutung (*prior*, oft $\mathcal{N}(0, 1)$) eine Aussage bezüglich y, bei neuen Beobachtungen von x getroffen werden. [119, 174]

Discriminative Models hingegen sind darauf ausgelegt, $P(Y|X)$ direkt über *Weights* w zu modellieren. In einem iterativen Prozess (*Weight*-Optimierung,

Training) wird durch die stetige Minimierung eines *Loss*-Terms w angepasst, also ein Optimierungsproblem gelöst. Gegeben sei ein Datensatz $D = \{(y_i, x_i), i = 1, \ldots, |D|\}$, ist ein linearer binärer *Classifier* $f_w(x)$ [176, 249] nach [139] ausgedrückt durch:

$$f_w(x) = w_0 + \sum_{j=1}^{x} w_j x_j \tag{2.3}$$
$$\text{mit } \langle w_j, w_0 \rangle \in \mathbb{R}$$

Man möchte dabei w^* finden, sodass ein *Loss* $\mathcal{L}$, gegeben X, minimiert wird:

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}(m_i(w)) \tag{2.4}$$

Das Beispiel der *Support Vector Machine (SVM)* [235] versucht, auf Basis der Trainingsdaten, eine möglichst gute Annäherung in Form einer Separationslinie zu schaffen, sodass eine Transformationsfunktion $h : X \to Y$ entsteht, durch die zukünftig Entscheidungen getroffen werden können.

Dieser Anpassungsprozess, also das *Fitting* von h an die Eingabedaten, wird auch als Lernvorgang bezeichnet. Wie eine mögliche Separierung eines höherdimensionalen Datensatzes (hier zufällig generierte Datenpunkte, als Mitglieder zweier Klassen) durch eine SVM im zweidimensionalen Raum aussieht, ist in Abbildung 2.1 dargestellt. Im Falle von nichtlinearen Modellen haben sich hingegen Gradienten-basierte Verfahren etabliert, wie der nächste Abschnitt zeigt.

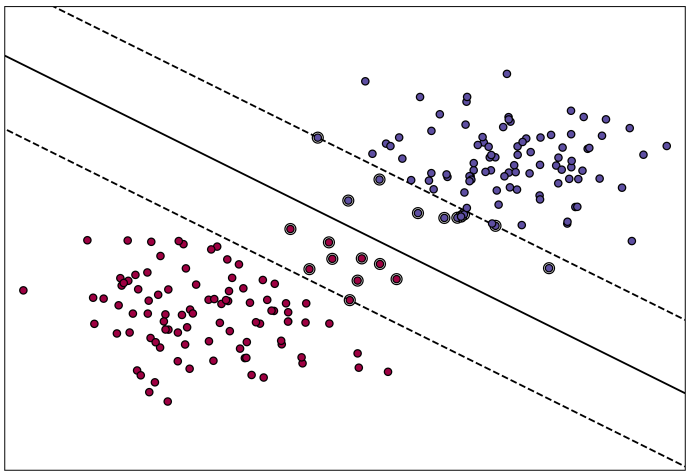
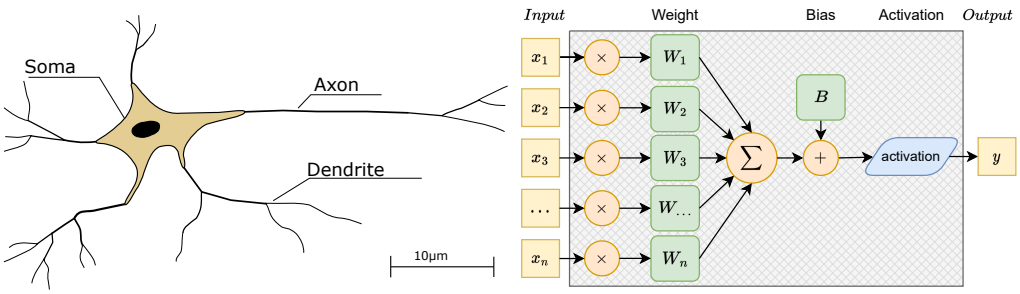


Abbildung 2.1: Separierung eines Trainingsdatensatzes durch eine lineare SVM.

2.2 Perceptron

Neural Networks (NNs) haben sich als ein erstklassiger Ansatz zur Approximation komplexer Funktionen etabliert und sind selbst als solche zu begreifen [56, 131]. Durch Verfahren wie (Un-)*Supervised Learning* oder *Reinforcement Learning* können Probleme gelöst werden, für die fest kodierte Algorithmen kompliziert zu erstellen wären oder lange Verarbeitungszeiten aufweisen. Ferner verfügen NNs über die *Generalization*-Fähigkeit, was ihnen erlaubt, gelernte Merkmale (*Features*) in zuvor ungesehenen Daten zu entdecken. Anders ausgedrückt, können diese Funktionsapproximationen ähnliche Eingangsmuster auf ähnliche Ausgangsmuster abbilden [200].

Das *Perceptron* (auch *Feed-Forward Network*) wurde als früheste *Architektur* nach biologischem Vorbild motiviert [3, 131]. Ein gerichteter Graph mit parametrisierten arithmetischen Operationen als Knoten und Verbindungen als Kanten soll eine Vernetzung ähnlich der eines biologischen *Neurons* nachahmen. Dabei wird die Verbindungsstärke der ankommenden Signale (*Input*, x), durch Gewichte (Parameter θ; eng. *Weight* w & (optional) *Bias* b) abstrahiert. Die Summe dieser Signale wird anschließend durch eine Activation-Function weiter in den *Output* $\hat{y}$ transportiert, $y = f_{wb}(x) : R^{(|x|)} \rightarrow R$:



(a) Biologisches Neuron (b) Mathematische Abstraktion

Abbildung 2.2: Biologisches Neuron (**a**) sowie die mathematische Abstraktion (**b**) nach Suzuki [225]

$$\hat{y} = sign(\sum_{i} = 1^{|x|} w_i \cdot x_i) \tag{2.5}$$

Das Optimierungsziel dieser Funktion $f_{wb}(x)$ ist formuliert als:

$$\underset{w,b}{\operatorname{argmin}} \mathcal{L} = \sum_{i} = 1^{|x|} (y - \hat{y})^2 \tag{2.6}$$

Wobei w jeweils angepasst wird wie folgt:

$$w \Leftarrow w + \alpha(y - \hat{y}) \cdot x \tag{2.7}$$
$$\text{mit } \alpha \in \mathbb{R}^+ \leq 1 \ (\textit{Lernrate})$$

Durch mehrfach randomisierte Durchgänge durch den D (Epochen) entstand so ein *Model*, welches die Zuordnung $\hat{y} \in [-1, 1]$ (eine Form der *Binary Classification*) gegeben eines linear separierbaren Datensatzes D vornehmen konnte. Eine SVM z. B. kann so durch die Wahl des richtigen *Loss* simuliert werden. [3]

2.3 *Deep Learning* durch *Neural Networks*

Deep Learning (DL) kann als konsequente Weiterentwicklung dieses Prinzips gesehen werden. Durch die Hinzunahme weiterer *Neurons*-Einheiten C (oder auch *Zelle*) entsteht eine gleichgeschaltete Schicht L (*Layer*), welche als Funktion $L^C : \mathbb{R}^{(|x|)} \to \mathbb{R}^{(|y|)}; \hat{y} = L^C(x)$ parallelisiert auf einen *Input* x angewendet wird. Da es sich hier um eine Vollvermaschung zwischen allen Elementen des *Input*-Vektors und L handelt, spricht man auch von einem *Fully-Connected* Network. Die Größe von $\hat{y}$ ist hier durch die Anzahl der Zellen $|C|$ angegeben. Werden schließlich mehrere dieser *Layer* hintereinander platziert, also parallelisierte Einheiten in sequenzieller Folge auf einen *Input* x angewendet, entsteht ein gerichteter, nichtzyklischer *Computational Graph (CG)*, das *Neural Network (NN)*:

$$\hat{y} = f_\theta(x) = L_n(L_{n-1}(L_1(x))) \tag{2.8}$$

Man spricht dann auch von tiefen neuronalen Netzwerken[1].

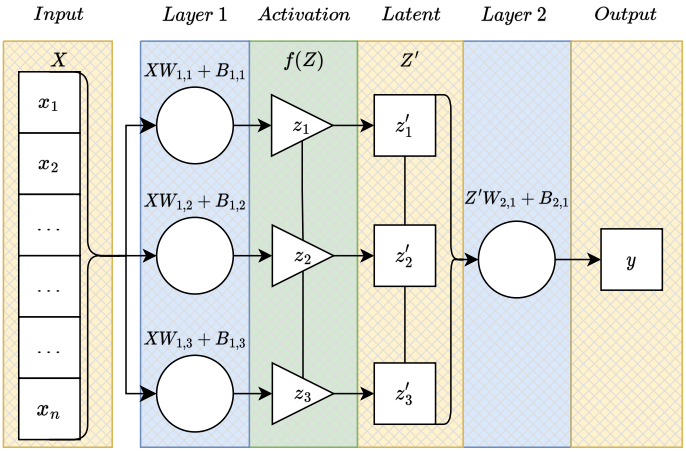


Abbildung 2.3: Einfaches NN mit 2 Schichten und 4 *Neurons* sowie einer arbiträren Activation-Function.

Obwohl DNNs über vergleichsweise viele freie *Weights* θ (*Weight* & *Bias*) verfügen, können diese dennoch effizient durch *Backpropagation (BP)* [200] trainiert werden. Hierfür wird im Allgemeinen ein Gradient auf Basis einer Verlustfunktion $\mathcal{L}$ (*Loss*) berechnet und auf die NN-*Weights* $\theta = (w, b)$ (*Weights* & *Biasses*) im CG durchgereicht. BP, kann immer dann angewendet werden,

[1] Genauer ist dieser Begriff im Allgemeinen nicht definiert.

wenn das NN als Funktion differenzierbar ist, sodass die *Kettenregel* (eng. *Chain Rule*) auf jeden *Layer* und dort auf jedes *Weight* in jedem *Neuron*[2] rekursiv angewendet werden kann. [34] Gegeben der Eingabe x eines Datensatzes $D = \langle x_i, y_i \rangle, \ldots, \langle x_n, y_n \rangle$, mit $n = |D|$ entsprechen die *Weights* θ einer zusammengesetzten, nichtlinearen Funktion F so zu optimieren, dass x auf die Ausgabe $\hat{y} = f_\theta(x) = F(x, \theta) : \mathbb{R}^{(|x|)} \to \mathbb{R}^{(|y|)}$ abgebildet und $|y - \hat{y}|$ minimiert wird [3, 202]:

$$\hat{y} = f_\theta(x) = l_{|L|}(\ldots l_2(l_1(x, \theta_1), \theta_2) \ldots, \theta_{|L|}) \quad \text{wobei } |L| \in \mathbb{N}= \text{ Layeranzahl} \tag{2.9}$$

In Abhängigkeit der zu lösenden Aufgabe wird eine *Loss*-Funktion $\mathcal{L}$ formuliert, welche den *Model-Output* $\hat{y}$, gegeben einem *Input* x, mit einem *Target* y vergleicht: z. B. $L_f = \mathcal{L}(y, \hat{y}) = |y - \hat{y}|; L \in \mathbb{R}$. Dieser numerische Fehlerwert wird anschließend für das Gradienten-Update von θ verwendet. Da alle Pfade des CG statisch, gerichtet und nichtzyklisch sind, kann durch eine partielle Ableitung der Beitrag jedes *Weight* $w_1, \ldots, w_{|w|} \subset \theta$ rekursiv berechnet werden [3]. Da es hier lediglich darum gehen soll, das Prinzip von BP zu demonstrieren, macht diese sehr simple Version der *Chain Rule* dabei die Annahme, dass kein *Bias* existiert und lediglich alle *Weights* einer *Zelle* c des vorletzten *Layer* L_n optimiert werden sollen:

$$\frac{\partial L_f}{\partial c_r} = \Delta(c_r, \hat{y}) = \sum_{r=1}^{|c|} f'(z) \cdot w_{c_r} \cdot \Delta(c, \hat{y}) \tag{2.10}$$

Ganz allgemein erfolgt die Optimierung innerhalb einer Epoche durch *Stochastic Gradient Descent (SGD)*:

$$\theta = \underset{\theta}{\operatorname{argmin}} \frac{1}{D} \sum_{i=1}^{D} (\mathcal{L} f_\theta(x_i)) \tag{2.11}$$

Die *Performance* dieser so trainierten NN-*Architekturen* übersteigt (bei korrekter Implementierung) zumeist die *Performance* traditionellerer linearer ML-Methoden. Dies ist z. B. dann der Fall, wenn die Dimensionalität der Eingabedaten zunimmt (z. B. Bilder, Audioaufnahmen, Zeitserien, Filme). Hier haben sich solche approximativen Verfahren als besonders geeignet erwiesen. Abschließend sei noch angemerkt, dass bislang keine Methoden existieren, welche eine perfekte Generalisierungsfähigkeit des so entstehenden f_θ garantieren. Bewegt sich der NN-*Input* vollständig außerhalb (*out-of-distribution*) des Beobachtungsraums, können ungewollte und falsche Ausgaben forciert werden. Dies geschieht z. B. durch die gezielte Manipulation einzelner *Pixel* in Bilddaten,

[2] In diesem Kontext auch als *Node* bezeichnet.

welche von Menschen mit dem bloßen Auge nicht als solche identifiziert werden kann [30, 82, 120].

Wichtige Faktoren des NN-Trainings und Verfahren werden in den folgenden Absätzen vorgestellt, dabei wird der Fokus auch auf die sog. *Hyperparameter (HP)* gelegt, welche maßgeblich das NN-Lernverhalten moderner *Architektur* beeinflussen.

2.3.1 Activation-Functions

Konzeptionell bilden Aktivierungsfunktionen, nach ihrem biologischen Vorbild, den Anregungszustand einer biologischen Zelle anhand der ankommenden Signale ab. Die Activation-Function, welche dieses Verhalten am deutlichsten widerspiegelt, also die Transformation des Wertezustands einer Zelle in einem NN in einen binären *Output*, ist die logistische Sigmoid-Funktion (vgl. Abbildung 2.4, rot). Diese Eigenschaft machen sich NNs zur binären Klassifikation im *Output-Layer* zunutze.

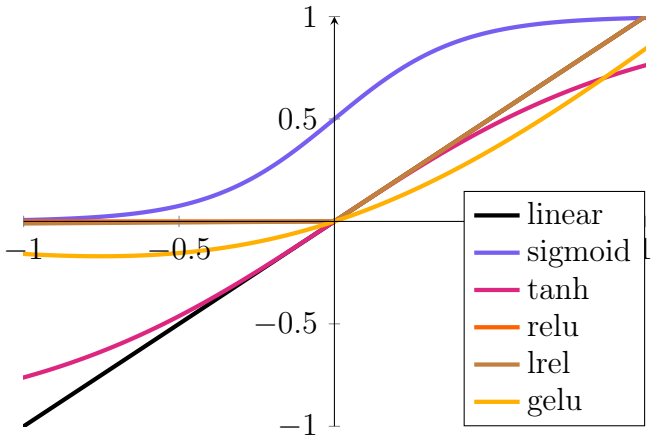


Abbildung 2.4: Darstellung der in dieser Arbeit verwendeten Activation-Function*s*

Praktisch transformieren Activation-Function den *Neuron-Output* z (*Logit*) auf nicht lineare Weise und sorgen, durch das Einbringen von Variationen, dafür, dass z. B. auch nicht linear separierbare Funktionen abgebildet werden können. Sogenannte *Squashing Functions* „zwängen“ z in einen bestimmten Wertebereich, während *Rectifier* das Signal „aufrichten“, indem der negative Bereich der Funktion abgeschwächt oder gar vollständig unterdrückt wird ($(z \leq 0) = 0$). Darüber hinaus haben Activation-Function*s* eine Vielzahl nützlicher Eigenschaften, so wird z. B. durch das Eindämmen des Wertebereichs vor jedem *Layer* unter anderem BP unterstützt, was einem möglichen *Vanishing Gradient Problem (VGP)* (vgl. Abschnitt 2.3.4.1) entgegenwirkt. [3] Die heute zur Verfügung stehenden Varianten haben jeweils ihre eigenen Vor- und Nachteile:

Etwa findet die Sigmoid-Funktion $\sigma(x) : \mathbb{R} \to \mathbb{R}^{[0,1]}$ in den inneren Schichten L_{N-2} tiefer NNs kaum noch Anwendung, da diese über Zeit den Gradienten abschwächt ($lim_{x\to-\infty} f(x) = 0$, ab ca. fünf *Layer* [49, 76]), zum anderen ist die Berechnung der Exponentialfunktion (vgl. Tabelle 2.1) und deren Ableitung rechenintensiver als die Alternativen.

Tabelle 2.1: Activation-Functions

Linear	$f(x)$
Sigmoid	$\sigma(x) = \frac{1}{1+\exp^{-x}}$
Tanh	$Tanh(x) = \frac{\exp(x)+\exp(-x)}{\exp(x)-\exp(-x)}$
ReLU	$Relu(x) = max(0, x)$
LREL	$LRel(x) = max(a \cdot x, x)$
GELU	$GELU(x) = x \odot P(\mathcal{N}(0, 1) \leq x)$
Softmax	$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, i = 1, \ldots, K$

Deswegen etablierte sich zunächst *Hyperbolic Tangent Function (Tanh)* in der Community, welche den Wertebereich zwischen $[-1; 1]$ und deren Ableitung symmetrisch um null platziert, sodass es zunächst nicht zu den gleichen Phänomenen wie bei Sigmoid kam. In tieferen Schichten (nach der ersten) zeigt sich allerdings wieder dasselbe Verhalten des nachlassenden Gradienten [76].

Rectified Linear Unit (ReLU) [77, 172] (vgl. Abbildung 2.4, orange, Tabelle 2.1) und spätere Varianten folgten dann, unter anderem wegen des kostengünstigeren Designs und der beschleunigten Konvergenz [49, 129].

$$Relu(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{sonst} \end{cases} \tag{2.12}$$

Bis vor wenigen Jahren verwendete man ReLU zumeist im Aufgabenbereich der *Computer Vision*. Da ReLU im negativen Bereich der Funktion nicht differenzierbar ist, kann zuteilen das Phänomen der sog. *Dying Neurons* beobachtet werden, bei dem die *Weights* einzelner *Neurons* den Wert *null* annehmen.

$$LRel(x) = \begin{cases} x & \text{if } x \geq 0 \\ a \cdot x & \text{sonst} \end{cases} \tag{2.13}$$

Hier versucht *Leaky ReLU (LREL)* mit einer einfachen Anpassung entgegenzusteuern: Anstatt einer Beschneidung des negativen Bereichs $x \leq 0$, steuert ein Parameter $a \in \mathbb{R}$ (*Negative Slope*) wie viel des ursprünglichen Signals in den negativen Bereich „durchsickert“. Obwohl Maas, Hannun und Ng in seinen ersten Ergebnissen keine Leistungssteigerung gegenüber ReLU darstellen konnte, findet LREL häufig Anwendung [157]. Aggarwal geht hingegen davon

aus, dass die durch ReLU unter Umständen eingeführte *Sparsity* als eine Art Pruning-Strategie gewertet werden könnte [3].

Gaussian Error Linear Unit (GELU) [93], hingegen ist eine Art „weiche" ReLU-Variante, welche als parametrisierte Activation-Function den *Slope* in ihrem negativen Funktionsbereich über erlernbare *Weights* steuern kann. Tabelle 2.1 gibt die Funktion nach Hendrycks und Gimpel wieder, welche auch durch eine Vereinfachung angenähert werden kann:

$$0.5 \cdot x(1 + \tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)))$$

Die in den negativen Wertebereich geschwungene Activation-Function hat in Untersuchungen die negativen Eigenschaften der (erfolgreichen) ReLUs-Activation-Function scheinbar überwunden und findet daher oft im Kontext moderner *Transformer-Architekturen* (vgl. Abschnitt 5.3.2) Verwendung (z. B. [50]).

Zur Vollständigkeit sei an dieser Stelle *Softmax* zu nennen. Diese Transformationsfunktion wird vorwiegend in NN-*Output-Layer* verwendet, um das Verhältnis der Klassenwahrscheinlichkeiten der *Logits* in den Koordinatenraum eines $n-1$ *Simplex* zu überführen, sodass [212]:

$$1 = \sum_{2}^{K} Softmax\left(f_\theta(x)\right); f_\theta : \mathcal{R}^- \rightarrow \mathcal{R}^K$$

2.3.2 Trainingsablauf

Wie in Abschnitt 2.3 beschrieben, wird im iterativen Optimierungsprozess eines NN (Training) über mehrere *Epochen* hinweg θ schrittweise durch BP angepasst, um $\mathcal{L}$ auf D zu minimieren. Eine Epoche bezeichnet dabei einen dieser Optimierungsschritte auf D, in der die *Generalization* verstärkt und negative Effekte wie *Catastrophic Forgetting* (das konstante Überschreiben vergangener Gradientenanpassungen) minimiert werden sollen. Die Umsetzung der gradientenbasierten BP übernimmt dabei ein *Optimizer* wie z. B. *Stochastic Gradient Descent (SGD)* [25, 26, 124, 194] oder *Adam* [125]. Das NN-Training wird dann als beendet angesehen und gestoppt, wenn die zu approximierende Funktion ihren *Grenzwert* erreicht hat, also $L = \mathcal{L}(f_\theta(x), y)$ sich über mehrere Epochen hinweg nicht mehr (deutlich) verbessert (vgl. Abschnitt 2.3.4.2). Das Konzept der mathematischen *Konvergenz* wird im Kontext des DL meist eher salopp verwendet und beschreibt im weitesten Sinne eine Annäherung von $\mathcal{L}$ an den Grenzwert $\lim_{L \rightarrow L^*} f_\theta(D)$, wobei L^* geg. D, f_θ, alle HP und aller Konfigurationen, das (nicht lokale) Optimum der Funktion beschreibt:

Im Falle der Minimierung:

$$L^* = \mathcal{L}(f_\theta(x_n), y_n) \leq \mathcal{L}(f_\theta(x), y) \text{ für alle } \langle x, y \rangle \in D, n \leq |D| \tag{2.14}$$

Im Falle der Maximierung:

$$L^* = \mathcal{L}(f_\theta(x_n), y_n) \geq \mathcal{L}(f_\theta(x), y) \text{ für alle } \langle x, y \rangle \in D, n \leq |D| \tag{2.15}$$

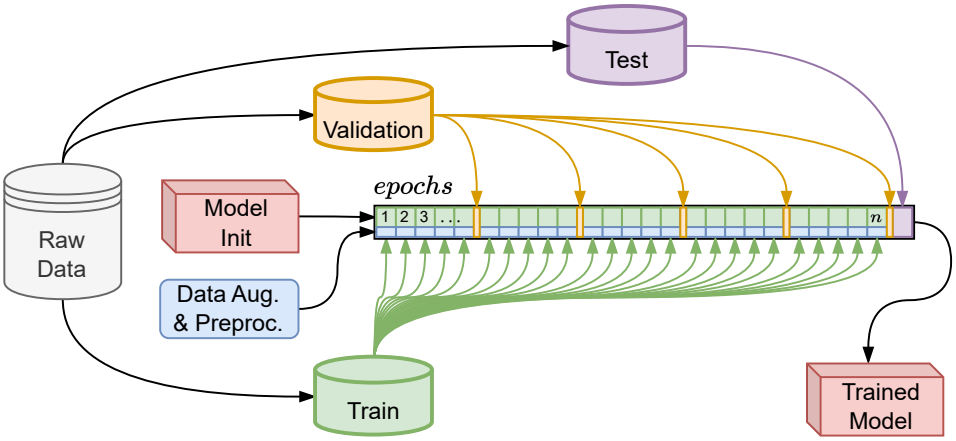


Abbildung 2.5: Trainingsablauf eines *Neural Network*s. Die Sequenz vertikaler Kästchen beschreibt die Epochenfolge und demonstriert die Frequenz zwischen Trainings-, Validierungs- und Testdatensatz.

Es können aber auch andere Metriken zur Beurteilung der NN-*Performance* zum Einsatz kommen. Diese Arbeit führt solche zum Zeitpunkt ihrer Relevanz in den nachfolgenden Kapiteln ein. Metriken im Allgemeinen sollten jedoch immer nur als Approximation der wirklichen *Performance* verstanden werden [3].

2.3.3 *Learning Rate* & Schedule

Ein weiterer Faktor, der aktiv das Maß der *Weight*-Anpassung jeder BP-Operation steuert, ist die *Learning Rate (LR)*(α), welche im Verlauf des Trainings durch einen *Schedule* kontinuierlich angepasst werden kann. Es existieren individuelle und oft an die NN-*Architektur* angepasste Strategien, die funktional an den Integer $n \in N_{epoch}$ der aktuellen Epoche gekoppelt sind:

Tabelle 2.2: *Learning Rate-Schedule*

linear annealing	lineare Reduzierung der LR
exponential	exponentielle Reduzierung der LR
cyclic annealing	zyklische Reduzierung der LR
cosine annealing with restarts	Kos. Absenkung der LR, Wiederkehrend $lr = (1 + \cos(\frac{n}{N_{epoch}}\pi)), n \in N_{epoch}$ [151]

Andere Verfahren versuchen hier nicht nur die zugrunde liegende *Software-Architektur*, sondern auch Beschränkungen der *Hardware* zu berücksichtigen [177].

2.3.4 *Hyperparameter*

Neben der Lernrate (LR) existieren viele weitere sog. *Hyperparameter (HP)*, welche das Lernverhalten eines NN maßgeblich beeinflussen und zumeist auf Basis von Expertenwissen (auch *Domain Knowledge*) ausgewählt werden. Die wichtigsten und in dieser Arbeit verwendeten HP sollen nun in diesem Unterabschnitt behandelt werden.

2.3.4.1 Hindernisse im *Neural Network*-Training

In diesem Abschnitt soll in Kürze auf Probleme eingegangen werden, die im Verlauf eines DNN-Trainings auftreten können. Gegenmaßnahmen, welche in dieser Arbeit zum Einsatz kommen, werden anschließend in Abschnitt 2.3.4.2 vorgestellt.

Overfitting
Im Extremfall kann *Overfitting* dafür sorgen, dass keine *Generalization* des NN zustande kommt und das trainierte *Model* die approximierte Funktion auf neuen (aber ähnlichen) Datenpunkten nicht aufrechterhalten kann, auch wenn diese sich rein statistisch innerhalb der Verteilung des Trainingsdatensatzes bewegen. Ein *Model* ist dann überangepasst, wenn es einen hohen *Bias* aufweist, d. h. es bildet die zum Trainingszeitpunkt bereitgestellten Daten gut ab, verliert aber auf ungesehenen Eingaben seine Interpretierbarkeit, was einer hohen Varianz entspricht. Die Gesamtleistung wird hier als schlecht angesehen. Die Höhe der Varianz eines *Models* kann als eine Funktion seiner Größe (*Weights*) betrachtet werden, d. h., dass unter der Annahme endlicher Stichproben die Varianz eines Modells mit der Anzahl seiner *Weights* zunimmt. [3, 31]

Catastrophic Forgetting
Catastrophic Forgetting äußert sich als unsteter Lernprozess, der sich durch eine

nicht kontinuierlich fallende, bzw. nicht konvergierende Lernkurve äußert. Die häufigste Ursache ist das „Überschreiben“ schon gelernter Zuordnungen während des Trainingsprozesses. Auch die Leistungsmessung auf Validierungsdatensätzen schwankt in diesen Fällen stark oder verbleibt auf einem initial schlechten Niveau. Dieses Phänomen kann als Folge der Wahl unpassender HP für den Lernprozess (LRs) oder beim Optimieren auf kontinuierlichen Daten-Streams auftreten.

Vanishing Gradient Problem

In DL bezeichnet das VGP einen Abfall oder starken Anstieg der Höhe des Gradienten während der Ausführung der BP. Zunächst in *tiefen* Netzen bei Verwendung von Sigmoid oder Tanh als Activation-Function beobachtet [76], tritt dieses Problem primär im Kontext von RNNs auf (vgl. Abschnitt 3.2.1). Bengio, Simard, Frasconi et al. beobachtet, dass kurze Abhängigkeiten (*short-term*) im Eingabevektor eines sequentiell aktivierten RNN, das Lernverhalten von *long-term* Beziehungen („weit“ auseinander liegende Abhängigkeiten in Sequenzen) stark stören können [16]. Genauso wie sich das Produkt von $t \cdot k, (t, k) \in \mathbb{R}$ dem Wert 0 annähern (*vanish*) oder in Richtung ∞ entwickeln kann (*explode*), verhält es sich auch für das Produkt von Matrizen [183] oder eben die Anpassung der Gewichtsmatrix durch den Gradienten.

2.3.4.2 Lösungsansätze

Ganz allgemein beschreibt die *Regularisierung* eine Menge von mathematischen und informationstechnischen Konzepten zur Anpassung der Charakteristik des *Weight*-Spaces (*Weight-Space*) durch eine Steuerung des Gradienten. Dabei ist die Anwendung einer *Regularisierung* nicht immer eindeutig, sondern kann auf die Lösung mehrerer Probleme abzielen. So soll z. B. VGP verhindert, die *Generalization* gestärkt und/oder *Overfitting* verhindert werden. In diesem Abschnitt werden für diese Arbeit relevante Methoden zur *Regularisierung* und weitere Lösungsansätze für die zuvor beschriebenen Probleme vorgestellt.

Minibatch

Aktuelle Arbeiten setzen vorwiegend auf das *Mini-Batch*-Verfahren, bei dem ein *Weight*-Update immer nur für den $mean(Gradient)$ einer unabhängig und identisch (eng. *independent, identically distributed (i. i. d.)*) Teilmenge D_{rand} von D berechnet wird [20], wobei $D > D_{rand} > 1$. Dieses Verfahren stellt einen Kompromiss zwischen dem *Weight*-Update nach der Verarbeitung jedes einzelnen *Samples* (online) sowie dem gemittelten *Weight*-Update nach der Verarbeitung aller verfügbaren Trainingsdaten (*Batch*) dar. Grundsätzlich ist es durch *Mini-Batch* möglich, eine neutrale Abschätzung des Gradienten zu erhalten, welcher das zu minimierende Problem optimiert. Aus praktischer Sicht können die zur Verfügung stehenden Ressourcen (Rechenleistung & Memory)

so optimal ausgenutzt werden [80]. Die Größe der Mini-*Batches* ist je nach Datensatz und NN-*Architektur* dabei neu zu bestimmen (= HP), wobei sich überwiegend der Einsatz von Zweierpotenzen ($2^n; n > 4$) beobachten lässt, wofür allerdings keine mathematische oder evidenzbasierte Erklärung existiert. Je nach verwendeter Literatur können unterschiedliche Interpretationen des Begriffs *Batch* aufkommen: Diese Arbeit verwendet *Batch* als ein zufällig bestimmtes Subset von D, welches für ein *Weight*-Update (*mini-batch gradient descent*) verwendet wird:

$$\theta = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} f_\theta(x_i) \tag{2.16}$$

Standardisierung

Es stellte sich heraus, dass ein NN-Training deutlich schneller Konvergenz erreicht, wenn der *Input* standardisiert, also $mu = 0, \sigma = 1$ aufweist [135, 247].

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \tag{2.17}$$

$$\text{mit } E[x] = \frac{1}{N} \sum_{i=1}^{N} x_i,\ Var[x] = \frac{1}{N} \sum_{i=1}^{N} (x_i - E[x])^2$$

Ioffe und Szegedy machen sich diese Beobachtung zunutze und übertragen das Prinzip auf jeden *Layer* eines NN im Training. Da eine Reihe von negativen Beobachtungen gemacht wurden [107], entschied man sich dazu, die Transformation zwar auf Ebene eines Mini*Batch* x auszuführen, diese aber vom vollständigen Trainingsdatensatz X abhängig zu machen ($\hat{x} = \mathrm{Norm}(x, X)$). *Batch Normalization (BN)* transformiert eine *Batch* mit k-Dimensionen und führt eine parametrisierte (γ, β) Skalierungsfunktion (2.18) ein, um die Nichtlinearität der Activation-Function*s* aufrechtzuerhalten:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)} \tag{2.18}$$

Dropout

Um gegen das in Abschnitt 2.3.4.1 beschriebene *Overfitting* anzugehen, wurden verschiedene Praktiken entwickelt. Eine davon ist die Applikation von *Dropout* [219], was als eine Möglichkeit der NN-*Regularisierung* durch ein „Verrauschen“ der *Neuron*-Aktivität gesehen werden kann [219]. Ein zufällig bestimmter Anteil p (HP) des *Neurons-Inputs* wird deaktiviert = 0, sodass ein „verrauschter“ *Output* $\tilde{y}_{l+1}$ des nachfolgenden *Layer* entsteht, wobei *Dropout*

auf jeden *Layer* $l \in L$ angewendet werden kann:

$$
\begin{aligned}
r_l &\sim Bernoulli(p) \\
\tilde{y}_l &= r_l \cdot y_l \\
\tilde{z}_l &= w_l \odot \tilde{y}^l + b_l \\
\tilde{y}_l &= f(\tilde{z}_l)
\end{aligned}
\tag{2.19}
$$

Durch die so entstehende Stochastizität im Trainingsprozess lernt das NN sich nicht auf das Vorhandensein aller *Neuron* zu verlassen und verteilt die Gewichte für verschiedene Aufgaben gleichmäßig über viele *Neurons* in einem *Layer*. Hinton et al. beschreibt dies als eine extreme Form des sog. *Bagging* bzw. *Ensemble Bootstrapping* [95]: Durch das parallele Training mehrerer NN-*Models*, also die jeweils zu einem Zeitpunkt gleichzeitig aktiven Pfade im CG, werden für verschiedene Aufgaben mehrere, Gewichte verwendet, was zu einer natürlichen *Regularisierung* der Gewichte führt.

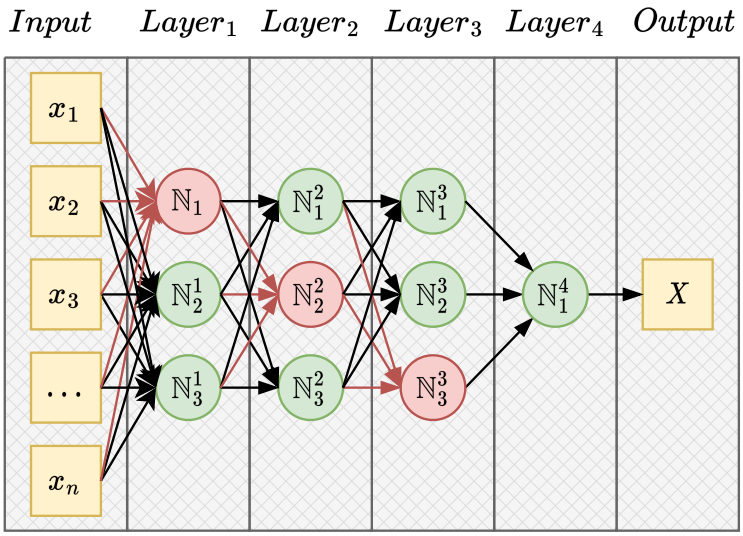


Abbildung 2.6: Abbildung eines einfachen NN mit 4 Schichten mit $3-3-3-1$ *Neurons*. Zur von *Dropout* wurden die 0-Werte produzierenden *Neurons rot* markiert.

HP-Tuning

Um die Auswahl passender HPs überhaupt zu finden oder einzugrenzen, ist es möglich, einen Suchraum festzulegen und mit automatisierten Verfahren abtasten zu lassen. Der einfachste Ansatz ist die sog. *Random Search*, bei dieser wird der HP-Raum stochastisch (zufällig) erkundet und das resultierende *Model* anschließend evaluiert. *Grid Search (GS)* hingegen ist der Versuch, den HP-Raum iterativ zu durchsuchen, indem HP vom Typ Integer um 1 erhöht, während Gleitkommazahlen (eng. *Float*) in regelmäßigen Intervallen

probiert werden. Es existieren auch komplexere Suchverfahren auf Basis genetischer Algorithmen oder statistischen Verfahren wie der *Tree-Structured Parzen Estimator (TPE)* [18](vgl. Abschnitt 5.3.3).

Early Stopping

Durch ein frühzeitiges Beenden, also weniger Epochen als initial geplant, des NN-Trainings, ist es möglich *Overfitting* zu verhindern. Um diese Entscheidung treffen zu können, wird neben dem Trainings-*Loss* auf D, parallel in einem größeren Intervall ein zusätzlicher Fehler $\mathcal{L}_{val}$ auf dem Validierungsdatensatz D_{val} berechnet. *Early Stopping (ES)* beendet das Training, sobald sich eine Metrik der Wahl für einen zuvor bestimmten Zeitraum nicht weiter verbessert. [3]

Data Augmentation

Neben Methoden zur *Regularisierung* durch einen direkten Einfluss auf den Gradienten im Lernprozess, basiert eine andere Strategie darauf, durch künstlich in die Trainingsdaten eingebrachte Störungen (sog. DA-Methoden), entgegenzusteuern. Auf diesem Wege kann die NN-*Robustness* gegenüber Invarianzen von Merkmalen in D und darüber hinaus gestärkt werden. Die Grundidee besteht in der Anpassung von Menge und Vielfalt eines ansonsten begrenzten Datensatzes (z. B. Menge der Trainingsdaten, Gleichgewicht der Klassen, Entropie innerhalb einer Klasse). Im Allgemeinen treffen DA dabei die Annahme, dass durch die Störung der Originaldaten zusätzliche, das NN-Training unterstützende Informationen generiert werden. Es wird angenommen, dass ein durch DA erweiterter D, eine vollständigere Beobachtung aller *möglichen* Ausprägungen darstellt, d. h. die Lücke zur realen Welt schließt und *Generalization* fördert.

Als Hauptstrategien können hier Störung, Rekombination oder Synthese zusätzlicher Trainingsbeispiele genannt werden. Roh, Heo und Whang liefert eine breite Analyse über viele Aspekte dieses Verfahrens, dabei wird deutlich, dass eine passende DA-Methode den Trainingserfolg von NNs deutlich steigern kann. Dies gilt vorwiegend für kleinere Datensätze, wie sie in Kapitel 4 verwendet werden. Größere Datensätze hingegen werden durch DA eher gestört und können negativ beeinflusst aus einem Training heraustreten[70]. Im Speziellen sind mal zwei Subkategorien zu unterscheiden: Bei der *Data Warping Method* werden vorhandene Daten transformiert, um den Umfang eines Datensatzes zu vergrößern. Bei *Oversampling Methods* hingegen werden völlig neue Daten synthetisiert [215]. Eine gute Übersicht über bestehende DA-Methoden für Anwender aus dem Bereich der *Computer Vision* findet sich in „A survey on Image Data Augmentation for Deep Learning“ [215]. Aufgrund der Tatsache, dass DA-Methoden je nach Domäne jeweils speziell ausgewählt und parametrisiert werden müssen, folgt eine erneute Erwähnung in Kapitel 6 und Abschnitt 4.3.3.

2.4 Sequenzielle Daten

Diese Arbeit wird sich in den folgenden Kapiteln mit unterschiedlichen Arten von Sequenzen und deren Verarbeitung beschäftigen. Der relevante methodische Hintergrund soll an dieser Stelle eingeführt werden. Beginnend mit einer allgemeinen Definition von Sequenzen, wird im Anschluss im Speziellen auf Audio-Daten eingegangen.

2.4.1 Im Allgemeinen

In der Informatik versteht man unter sequenziellen Daten (eng. *time series data*) eine strukturierte Datenfolge, bei welcher die Position einzelner Datenpunkte innerhalb einer Folge oder in Relation zueinander eine wichtige Rolle spielt. Dabei erschließt sich diese Folge meist aus technischen oder semantischen Gegebenheiten, welche die inhärente Ordnung definieren. Oft liegt dieser Ordnung die Zeit als Index zugrunde, z. B. bei fortlaufenden Messungen, sodass eine Zeitreihe eine Sammlung von äquidistanten Beobachtungen in chronologischer Abfolge darstellt. [58]: $D = (x_1, \ldots, x_n)$. Dabei ist es unerheblich, ob das Intervall der Messungen (Distanz) ein stetiges ist oder zu einem Teil der Daten selbst wird: $D_{timed} = \{\langle x_1, t_1\rangle, \ldots, \langle x_n, t_n\rangle\}$. Wichtig hierbei ist, dass es sich bei den in dieser Arbeit betrachteten sequenziellen Daten immer um einen diskreten Datentyp handelt. Dabei ist unerheblich, ob z. B. eine Messung ein ursprünglich kontinuierliches Signal erfasst hat. Eine Anhebung der Auflösung ist in diesem Fall nur durch ein gesteigertes Messintervall möglich. Als praktische Beispiele für sequenzielle Daten können unter anderem ganz allgemein Messreihen, Texte, Tonaufnahmen, Musik sowie Trajektorien von Koordinaten oder Zustandsräumen genannt werden. Über diese eingängigen Beispiele hinaus können auch andere strukturierte Datentypen, wie z. B. Bilder oder Punktwolken [64] als eine Folge interpretiert und als eine solche verarbeitet werden [50, 86, 162, 239]. Fawaz et al. gibt einen Überblick über Varianten und die Geschichte der Sequenzverarbeitung [60] im Kontext von NN.

Im Folgenden soll dies am Beispiel der Verarbeitung von sequenziellen Audiodaten genauer betrachtet werden. Dabei wird auf die Beschaffenheit von Audiodaten sowie Besonderheiten dieses Datenformats eingegangen. Im Zuge dessen werden auch Möglichkeiten der Transformation in andere mögliche Repräsentationsformen beschrieben.

2.4.2 Audio Sequenzen

Unter Klang versteht man ein mechanisches Phänomen, das durch die Störung eines Übertragungsmediums (Luft) auftritt. Weisen die dabei entstehenden Änderungen im Luftdruck, welche als *Wellen* mit einer Frequenz f_a von einem Punkt zum anderen wandern, die richtigen Charakteristiken auf, können sie von

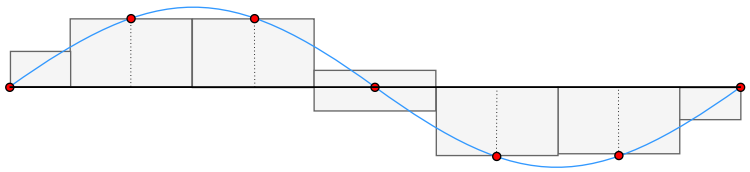


Abbildung 2.7: Beispielhafte Darstellung des Prinzips der Abtastung einer kontinuierlichen Welle zum Erhalt einer diskreten Repräsentation.

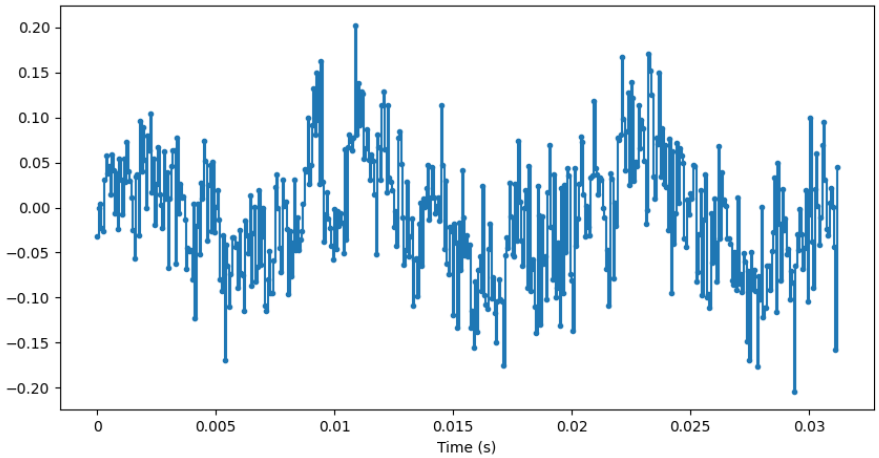


Abbildung 2.8: Beispiel einer Wellenformdarstellung eines Audiosignals.

dem menschlichen Ohr erfasst werden. [234]. Bleibt über einen Zeitraum T die Amplitudenhöhe A und die Frequenz f_a der Schwingung konstant, kann diese periodische Welle als z. B. eine Sinus Funktion der Zeit angegeben werden [191]: $x = f(t) = A \cdot \sin(f_a \cdot t)$. Die vom menschlichen Ohr wahrgenommene Lautstärke ist dann gleich Energie unter der Kurve dieser Welle, die als Schalldruckpegel definiert ist [246]. Durch eine äquidistante Messung dieser kontinuierlichen (Schall-)Welle lässt sich dann eine diskrete, digitale Repräsentation mit einer definierten Auflösung ableiten. Eine beispielhafte Darstellung findet sich in Abbildung 2.7. Die Genauigkeit dieser diskreten Annäherung an eine zusammengesetzte kontinuierliche Funktion der Zeitdomäne wird sowohl durch die Dauer der Messung $0 < s < \inf, s \in \mathbb{R}^+$ als auch durch die Häufigkeit innerhalb von s, die SR, definiert wird, wobei $sr \in \mathbb{N}^+$. Die Sample-Frequenz $F_s = 1/sr$ entspricht dann der Unterteilung einer Zeiteinheit in sr Messungen [123].

Eine Audio-Sequenz x entspricht dann einer Folge von Einzelmessungen x_t, wobei jeder Datenpunkt die Amplitudeninformationen, also die Lautstärke oder Energie zu einem bestimmten Zeitpunkt $t < (sr \cdot s) < \inf$ modelliert: $x = [x_0, \ldots, x_t], x \in \mathbb{R}, t \in \mathbb{N}^+$. Abbildung 2.8 zeigt eine typische Visualisierung eines Amplitudensignals. Möchte man so ein bestimmtes Phänomen z. B. im für Menschen hörbaren Spektrum erfassen, müssen nach Shannon bestimmte Annahmen getroffen werden: Das *Shannon Theorem* besagt, dass jede Welle für

eine Zeiteinheit mindestens doppelt so oft abgetastet werden muss, wie die größte zu erfassende Frequenz $f_{sr} = 1/2 \cdot f_a$, wobei die Zeiteinheit im Audiobereich typischerweise einer Sekunde entspricht. Da das menschliche Ohr gewöhnlicherweise Schwingungen der Luft in einem Frequenzbereich von bis zu $f_a \leq 22kHz$ auflösen kann, sollte die SR für Aufnahmen bei $40.1kHz$. [117] nach [213]

2.4.3 Mel-Spektrogramm und die Fourier-Transformation

Da die wenigsten uns umgebenden Geräusche (z. B. Sprache oder Musik) als eine immer stehende kontinuierliche Welle auftreten, wird hier von einer zusammengesetzten Funktion gesprochen. Diese bildet sich aus der Summe der vorhandenen Sinus- und Kosinus-Komponenten (sog. Basisfunktionen) in Abhängigkeit der Zeit. Für z. B. Analysezwecke schafft es die *Fourier Transformation (FT)*, die akustischen Teilkomponenten, also den Anteil der vorhandenen Frequenz f_a zu einem Zeitpunkt $t \in T$ sowie deren Amplitude A_n herauszuarbeiten [28]. Eine Erweiterung dieses Konzepts, die *Discrete Fourier Transformation* (DFT), erlaubt die Anwendung der FT auf diskrete Sequenzen mit regelmäßigen, also äquidistanten Intervallen [28, 218].

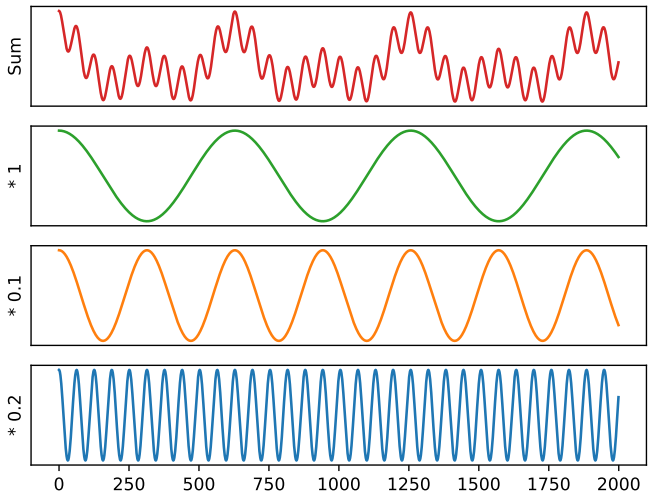


Abbildung 2.9: Skalierte beispielhafte Wellen-Dekomposition.

Durch die Kombination der Basisfunktionen in Gleichung 2.21 & 2.20 ergibt sich so die Amplitudenfolge der Sinus- und Kosinus-Wellen des abgetasteten Signals, in einem betrachteten Analysefenster N bilden [218]:

$$\begin{aligned} C_{f_a}[t] &= \cos(2\pi f_a t/N) \\ S_{f_a}[t] &= \sin(2\pi f_a t/N), \\ \text{wobei } f_a &\in 0, \ldots, N/2; t \in 0, \ldots, N-1. \end{aligned} \tag{2.20}$$

Zusammengesetzt ergibt sich so der Anteil eines Frequenzspektrums $p_f a$ eines diskretisierten Signals x der Länge T, wobei $t \in T$:

$$p_f a[t] = \sum_{f_a=0}^{N/2} Re\bar{X}[f_a] \cos(2\pi f_a t/N) + \sum_{f_a=0}^{N/2} Im\bar{X}[f_a] \cos(2\pi f_a t/N), \tag{2.21}$$

$$\text{wobei } Im\bar{X}[f_a] = \sin\text{-Amp.}, \tag{2.22}$$

$$Re\bar{X}[f_a] = \cos\text{-Amplitudenvektoren darstellen}$$

Zur Vollständigkeit: Möchte man hingegen ein Signal der Frequenzdomäne analysieren, stellt man diese Transformation nach $ReX[f_a]$ bzw. $ImX[f_a]$ um [218]:

$$ReX[f_a] = \sum_{t=0}^{N-1} x[t] \cos(2\pi f_a t/N) \tag{2.23}$$

$$ImX[f_a] = -\sum_{t=0}^{N-1} x[t] \sin(2\pi f_a t/N) \tag{2.24}$$

wobei:

$$Re\bar{X}[f_a] = \frac{ReX[f_a]}{N/2} \tag{2.25}$$

$$Im\bar{X}[f_a] = -\frac{ImX[f_a]}{N/2}$$

Die Aneinanderreihung solcher Analysefenster (eng. *Short-Time Fourier Transformation (STFT)*), kann durch zwei Parameter gesteuert werden. Zum einen die Segmentlänge $n_{fft} = N$, zum anderen die Sprungweite *hop_length* (gemessen in t), um die das Fenster verschoben wird. Die Erweiterung *Fast Fourier Transformation (FFT)* [41] reduziert die Komplexität der hierzu notwendigen Berechnungen, wenn die N einer Zweierpotenz entspricht. n_{fft} wird dabei als Kompromiss (*Trade-Off*) zwischen Frequenzauflösung (bei längeren Segmenten) und Zeitauflösung (begünstigt durch kürzere Segmente) gewählt [218]. Je nach Domäne und Datensatz unterschiedlich hat sich z. B. für die Analyse von Sprache eine Fensterbreite von $20ms$ etabliert [211].

Zum besseren Verständnis wird die durch STFT gewonnene zwei-dimensionale Frequenz-Amplituden-Matrix durch die *Mel-Scale* [221] in eine, dem menschlichen Gehör angepasste und zusammengefasste Wertebereiche, verschoben. Durch eine Konvertierung in ein Graustufenbild, wobei niedrige Amplitude hell, und hohe Amplitude dunkel dargestellt werden, erhält man ein Bild dieses Spektrogramms. Anders ausgedrückt beschreibt ein Mel-Spektrogramm im Audiobereich für einen Zeitraum $t - t_{N-1}$ den Anteil eines Frequenzspektrums

und überführt diesen in die Mel-Skala [78, 146]. Ein weiterer Vorteil ergibt sich aus der Möglichkeit, dieses so entstehende Format in visueller Form zu erschließen.

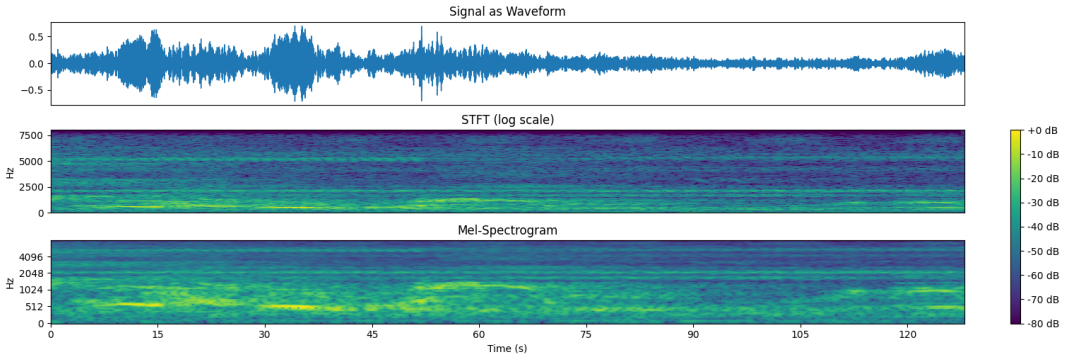


Abbildung 2.10: Waveform, STFT & Mel-Spektrogramm

Nachdem beschrieben wurde, wie ganz allgemein und im speziellen Audiodaten sowie daraus resultierende Mel-Spektrogramme aufgebaut sind, wird in den folgenden Abschnitten darauf eingegangen, wie dieses Datenformat durch NN verarbeitet werden kann. Hierbei soll der Fokus auf in der Praxis anwendbare *Architekturen* und ihre Besonderheiten gelegt werden. Rein formal sei an dieser Stelle angemerkt, dass die gewählten Notationen aufgrund des stark schwankenden Kontextes immer nur in den Grenzen des jeweiligen Abschnitts relevant sind.

So untersucht Kapitel 3 natürliche Grenzen des Informationstransports in RNN unter der Betrachtung von i. i. d. Daten. Dann beschäftigt sich Kapitel 4 mit der Verarbeitung von Audiodaten, also äquidistanter Messungen einer kontinuierlichen zusammengesetzten Schwingung, durch NNs. Dabei zeigt sich, dass DA-Methoden einen erheblichen Einfluss auf die *Performance* von NNs haben können. Die Ergebnisse dieses Kapitels werden zudem in Kapitel 5 wieder aufgegriffen und vor dem Hintergrund des ViT weiter diskutiert. Diese neuartige Art von *Attention*-basierten NN-*Models* erlaubt es, Bilder als Sequenzen zu betrachten. Kapitel 6 versucht diesen Werkzeugkasten zu erweitern und verfolgt dabei den neuen Ansatz des *Feature*-Transports in *Samples* x. Zuletzt fällt der Blick in Kapitel 7 auf SRNN, die in der Lage sind, ihre eigenen Trainingsdaten im Verlauf des Optimierungsprozesses anzupassen. Dabei wird eine neuartige Variante des *Prunings* beobachtet.

3 Der *Memory Horizon* in RNN-Architekturen

In der Grundkonzeption sollen RNN-*Architekturen* in der Lage sein, Beziehungen zwischen Datenpunkten über Sequenzen hinweg zu modellieren. Von Anwendungen abseits dieser Konzeption wird zumeist abgesehen, da *Architektur*-bedingt scheinbar kein verlustfreier Informationstransport durch den *Hidden Vector (HV)* eines RNN erfolgen kann. Ganz allgemein können diese NN-*Architektur* z. B. nicht zählen oder bestimmte wichtige Datenpunkte bis in die Ausgabeschicht durchreichen (Transport).

In diesem ersten Kapitel sollen genau diese Vorurteile genauer untersucht werden. Es wird gezeigt, dass, wenn keine unmittelbare Beziehung zwischen aufeinanderfolgenden Datenpunkten vorhanden ist (z. B. wenn sich Datenpunkte zufällig verhalten), RNNs dennoch in der Lage sind, gezielt einzelne Datenpunkte zu reproduzieren. Es wird beobachtet, dass Elemente fester Positionen einer Sequenz bis ans Ende der Verarbeitungskette mit einem minimalen Fehler durchgereicht werden können. Die Experimente in diesem Kapitel schließen auch weitere (modernere) Arten von *Recurrent Neural Network*s ein. Es wird gezeigt, dass die Strecke dieses beobachteten Datentransports stark begrenzt und durch *Architektur* und Größe der verwendeten rekurrenten *Zellen* definiert ist. Es besteht also eine harte Grenze (weit unterhalb der informationstheoretischen Grenze) für die maximale Transportdistanz, unterhalb derer RNNs in der Lage sind, Elemente einer *Input*-Sequenz über lange Zeit als latente Repräsentation aufrechtzuerhalten. In diesem Kapitel wird dieser MH beschrieben und im Kontext der Verarbeitung zufälliger Sequenzen untersucht.

Contribution 3.0.1: Memory Horizon in RNN

Beiträge zur Wissenschaft in diesem Kapitel umfassen die Einführung des Konzepts des *Memory Horizon (MH)*, Definition und Durchführung von Experimenten zur Analyse des Informationstransports in RNN-Varianten (RNN, LSTM, GRU, LSTM). Sowie die Demonstration und Einordnung des MH im Kontext vergleichbarer *Architekturen* (*Memory Horizon* & *Echo State Network*). Vorveröffentlichung durch Illium. et al. in „Empirical Analysis of Limits for Memory Distance in Recurrent Neural Networks“ [105]

3.1 Einführung

Eine der entscheidenden Einschränkungen von NNs besteht darin, dass diese *Architektur* direkt auf die Struktur der Eingabedaten angepasst werden muss. So hat z. B. ein initialisiertes (und trainiertes) FFNN, welches im *Input-Layer* an n-große Vektoren angepasst wurde, keine Möglichkeit m-große *Sample* zu verarbeiten $(n \neq m) \in \mathbb{N}^+$. Um dies z. B. während des Trainings zu ermöglichen, muss von reinen Gradienten-basierten Trainingsmethoden abgewichen werden [220]. Aber auch durch den Einsatz z. B. evolutionärer Algorithmen ist es aufgrund der starren NN-*Architekturen* nicht möglich, Eingabedaten über eine maximale Größe (Länge/Höhe/Breite) hinaus zu verarbeiten.

Anwendungsfälle wie die Klassifizierung unterschiedlich langer sequenzieller Zeitreihen haben so das Konzept des *Recurrent Neural Network (RNN)* [200] motiviert. Diese *Architektur* wendet eine parametrisierbare *Zelle* (vgl. *Neuron*) sukzessive auf einen Datenstrom an, wobei durch einen *Hidden Vector (HV)*-Vektor h eine direkte Verbindung zu dem eigenen jeweils früheren Zustand h_{t-1} besteht. So können Informationen jedes Elements x_t des gesamten *Input*-Vektors x mit in die Berechnung einbezogen werden. Diese sequenzielle Art der Anwendung erlaubt die Verarbeitung von x variabler Länge. Weitere Einzelheiten zum inneren Aufbau von RNN-Zellen finden sich in Abschnitt 3.2.1.

Eine modernere und je nach Anwendungsgebiet weit verbreitete Erweiterung der RNNs-*Architektur* stellt das LSTM [96] dar. Bei diesem Ansatz werden zusätzliche Verbindungen (und *Weights*) eingeführt, welche die Weitergabe von Zustandsinformationen zwischen den Zeitschritten der Verarbeitung explizit steuern können. Eine genauere Beschreibung der Funktionsweise folgt in Abschnitt 3.2.2. Basierend auf dieser *Architektur* entwickelten Cho et al. die *Gated Recurrent Unit (GRU)*, welche mit weniger *Weights* (bzgl. LSTM) in vielen Anwendungen eine vergleichbare Leistung aufweist [37, 39].

All diesen *Architekturen* ist gemein, dass sie einen oder mehrere HVs mit sich führen, welche während der Verarbeitung einer Sequenz Informationen zu Korrelationen in Form einer Repräsentation mit sich tragen. Ein RNN hat so die Möglichkeit, die zugrunde liegende Funktion zu approximieren. Doch wie verhält es sich, wenn die Menge an Eingabedaten keine natürliche Korrelation oder inhärente Funktion trägt, also $x_t = \mathcal{U}(0, 1)$ für jedes $t, \ldots, q$ i. i. d. verteilt ist?

Da RNNs konzipiert wurden, Abhängigkeiten in sequenziellen Daten zu erkennen und zu modellieren, könnte ihre Fähigkeit, mit i. i. d. oder unkorrelierten Eingaben umzugehen, von geringer Bedeutung sein. Folgt man allerdings der Reservoir-Computing-Literatur, scheint diese Fähigkeit eher ein integraler Bestandteil des RNN-Speichers und damit der Leistungsfähigkeit zu sein. Für Aufgaben, bei denen die Ausgabe stark von der gesamten Eingabehistorie abhängen soll, benötigt es einen eindeutigen, klar unterscheidbaren verborgenen Zustand für jede Eingabe. Werden jedoch unkorrelierte Daten verwendet,

kann die Gesamtmenge an Informationen, welche ein RNN in seiner latenten Aktivierungsdynamik speichern kann, annähernd bestimmt werden.

In diesem Kapitel soll daher gezeigt werden, dass der theoretische Transport von Elementen randomisierter Sequenzen durch RNN-artige Strukturen im HV auch in der Praxis erlernt werden kann [27]. Hierbei wird eine deutlich erkennbare Grenze, der MH, für die Distanz, welche diese Transportfähigkeit überwinden kann, erarbeitet. Entsprechende Experimente werden in Abschnitt 3.3 vorgestellt und für verschiedene Ausprägungen von RNN-*Architektur* in Abschnitt 3.4 ausgeführt. Verwandte Arbeiten zu dieser Thematik sind in Abschnitt 3.5 aufgeführt und werden direkt mit den Ergebnissen in diesem Kapitel diskutiert. Es stellt sich abschließend die Frage, wie RNN-HPs gewählt werden müssen, um spezifische Aufgaben zu bewältigen, dies wird am Ende dieses Kapitels kurz diskutiert (vgl. Abschnitt 3.6).

3.2 Methodische Grundlagen

In diesem Abschnitt werden zunächst die drei populärsten Arten von RNN-*Architekturen* umrissen, welche dann in Abschnitt 3.4 durch Experimente zu ihren jeweiligen Transportgrenzen untersucht werden.

3.2.1 *Recurrent Neural Network*

RNN, zuerst in der Literatur von Rumelhart, Hinton und Williams erwähnt, beschreibt *Neural Network*, welche speziell zur Modellierung von Abhängigkeiten in sequenziellen Daten eingeführt wurden [205]. Genauer beschreibt diese erste Arbeit das Prinzip hinter RNN als einen Mechanismus, der das Problem zu lösen versucht, ein sequenzielles Programm zu finden, das einen sequenziellen *Input* auf einen sequenziellen *Output* abbildet [200]. Die Vorteile gegenüber FFNN liegen hierbei auf der Hand: Durch eine sequenzielle Aktivierung eines Systems können deutlich längere Sequenzen von weniger *Zelle* bearbeitet werden und erkannte Muster können gegenüber eines *Input*-Vektors $x = \langle x_t \rangle_{1 \leq t \leq q, t \in \mathbb{N}}$ der Länge $q \in \mathbb{N}$ innerhalb dieser an unterschiedlichen Positionen t vorkommen. Wenn FFNNs Funktionen entsprechen, können RNNs durch Zyklen als dynamisches System gesehen werden, welches durch den latenten dynamischen Speicher h_{t-1} auch ohne *Input* $x_t = 0$ einen internen Zustand h_t aufrechterhalten können [154].

Während immer jeweils ein Element x_t verarbeitet wird, transportiert der HV h die Historie der vorhergegangenen Berechnungen h_{t-1} und verrechnet diese mit den jeweils aktuellen Elementen x_t der *Inputs*-Sequenz [131]. Gegeben eines sequenziellen *Inputs* x und einer nicht-linearen Activation-Function $g = tanh$ ist der Ausgangszustand für jeden Zeitschritt $h_t = g(W \odot x_t + U \odot h_{t-1})$ [39]. Der *Output* h_t steht also immer für den jeweils nächsten Schritt

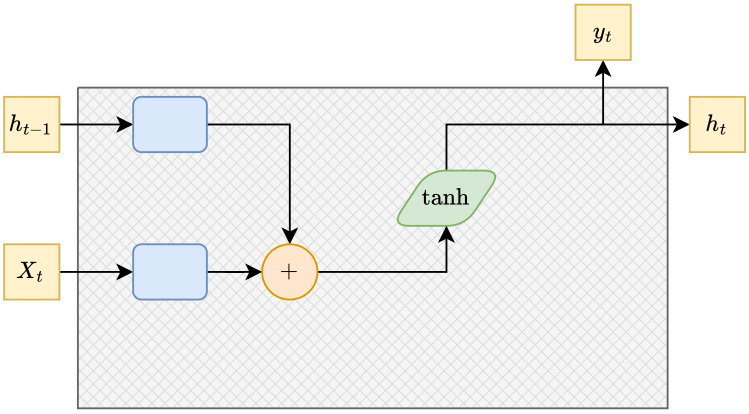


Abbildung 3.1: Die interne Struktur einer RNN *Zelle*.

als h_{t-1} zur Verfügung. RNNs können als eine einzige Schicht neuronaler *Zellen* betrachtet werden, die entlang der *Inputs*-Sequenz mehrfach angewendet werden, während alle aufeinanderfolgenden Berechnungen die jeweils vorherige Berechnung mit betrachten (vgl. Abbildung 3.1). In der Praxis werden zu jedem Zeitschritt die *Weights* der gleichen *Zelle* angewendet, bzw. teilen sich die in Reihe geschalteten *Zellen* die gleiche Gewichtsmatrix (W, U). So wird jede Position des *Output*-Vektors h durch die Kombination aller zuvor ausgewerteten *Inputs* beeinflusst. Die in Abschnitt 3.3 verwendete Version der RNN ist auch als *Elman Network* [54] bekannt und wird durch das verwendete Framework mit zusätzlichem *Bias* b wie folgt definiert:

$$h_t = \tanh(x_t \odot W_x + b_x + h_{t-1} \odot W_h + b_h) \tag{3.1}$$

Die Literatur [3, 253] nennt oft noch einen weiteren Schritt, durch den der HV nicht direkt offengelegt wird:

$$y_t = W_y \odot h_t \tag{3.2}$$

Abschließend ist noch auf eine Besonderheit hinzuweisen: Aufgrund der Zyklen im CG und der damit geteilten *Weights*, muss die BP bzw. die Berechnung des Gradienten auf RNNs angepasst werden. Es wird dabei angenommen, dass die verschiedenen zeitlichen Ausprägungen von w_h, w_x, w_y (und *Biasses*) unabhängig voneinander existieren, um einen Gradienten für jeden Zeitschritt berechnen zu können. Anschließend werden die jeweiligen Anteile zu einem Gesamtgradienten aufaddiert, um eine einheitliche Aktualisierung für jedes *Weight* zu berechnen [3, 253]. Spricht man von *truncated Backpropagation Through Time (BPTT)* [113, 165, 195, 200, 245] referenziert man ein Verfahren, bei dem diese Summe nur die letzten τ-Schritte einbezieht [3]. Lillicrap und Santoro vergleicht das Verfahren der BPTT durch RNNs mit dem sog. „Credit Assignment“ in kognitiven Prozessen.

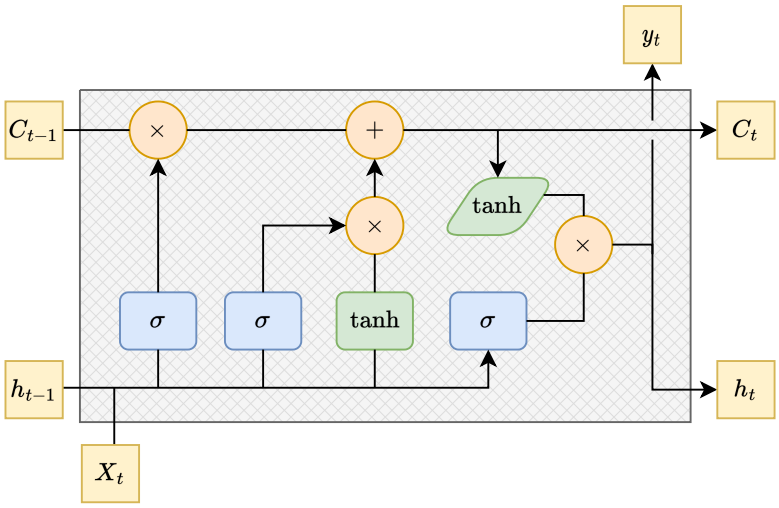


Abbildung 3.2: Die interne Struktur einer LSTM Zelle.

3.2.2 *Long Short-Term Memory*

Weil reguläre RNNs für ihre Probleme beim Verständnis von Langzeitabhängigkeiten und das VGP (vgl. Abschnitt 2.3.4.1) bekannt sind [16], wurde die *Long Short-Term Memory (LSTM)*-Einheit [96] eingeführt. Der Hauptunterschied zu RNN-*Zellen* liegt in der Einführung eines zusätzlichen HV c und der Kombination von drei parametrisierten Entscheidungs-*Gates* (Sigmoid σ aktivierte *Fully-Connected-Layer*), durch welche die Manipulation des Informationsflusses innerhalb der LSTM-Zelle durch die beiden HVs und über alle Zeitschritte der *Inputs*-Sequenz hinweg erlernt wird. Die drei *Gated Memory Vector (GMV)* können wie folgt beschrieben werden [3, 96]:

$$
\begin{aligned}
i_t &= \sigma(w_t \odot [h_{t-1}, x_t] + b_t) \\
f_t &= \sigma(w_f \odot [h_{t-1}, x_t] + b_f) \\
o_t &= \sigma(w_o \odot [h_{t-1}, x_t] + b_o) \qquad && [,] = \text{Concatenation} \\
\\
\widetilde{c_t} &= tanh(w_c \odot [h_{t-1}, x_t] + b_c); && \widetilde{c_t} = \textit{Hidden Vector} \\
c_t &= f_t * c_{t-1} + i_t * \widetilde{c_t}; && c_t = \text{Zustand der Zelle} \\
h_t &= o_t * tanh(c_t); && h_t = \text{Finale Ausgabe}
\end{aligned}
\tag{3.3}
$$

i_t: reguliert den Einfluss neuer Informationen (x_t, *Input Gate*)

f_t: reguliert den Einfluss des bestehenden HV (h_{t-1}, *Forget Gate*)

o_t: reguliert die Informationsweitergabe an h_{t+1} (*Output Gate*)

Gleichung 3.3 sowie Abbildung 3.2 beschreiben den Einsatz von i_t, f_t & o_t innerhalb der LSTM-*Zelle*.

Durch die relativ häufige Verwendung der Sigmoid-Aktivierungen (vgl. Ta-

belle 2.1) in den *Gates* sind insbesondere BP-Operationen mit hohen Rechenkosten verbunden. Auch steigt die Gesamtzahl an trainierbaren *Weights* auf $N = 4 \cdot (m * n + n^2 + n)$, wobei m = *Input*-Dimension und n = Zell-Dimension. Diese Vervierfachung im Vergleich zu einem RNN [152] wirkt sich auch auf die Anzahl an arithmetischen Operationen, welche nötig sind, um den deutlich komplexeren Informationsfluss abzubilden, innerhalb der Zelle aus.

3.2.3 *Gated Recurrent Unit*

Cho et al. schlagen mit der *Gated Recurrent Unit* eine RNN-Variante vor, welche sich im Gegensatz zu LSTM durch einen weniger komplexen Informationsfluss und eine Reduktion der benötigten *Weights* auszeichnet. Hier entscheidet das *Reset-Gates* r, ob der Zustand des *Hidden Vector* h_{t-1} ignoriert wird, dann entscheidet ein *Update-Gates* z, ob dieser durch die neu hinzukommenden Informationen x_t aktualisiert werden soll. Kurz gesagt steuert GRU durch die *Gates* r, z aktiv, wie viel Information jede Einheit beim Lesen oder Erzeugen einer Sequenz in die jeweilige Operation einfließen lässt [3, 37][1]:

$$
\begin{aligned}
r_t &= \sigma(W_r \odot x_t + U_r \odot h_{(t-1)} + b_r) \\
z_t &= \sigma(W_z \odot x_t + U_z \odot h_{(t-1)} + b_z) \\
\tilde{h_t} &= \tanh(W_{\tilde{h}} \odot x_t + r_t * (U_{\tilde{h}} \odot h_{(t-1)} + b_n)) \\
h_t &= z_t * h_{(t-1)} + (1 - z_t) * \tilde{h_t} \\
\text{wobei } * &= \text{Element-wise } \textit{Hadamard} \text{ Product}
\end{aligned}
\tag{3.4}
$$

Im Vergleich zu LSTMs ist der HV hier zusätzlich exponiert, heißt der *Output* y_t entspricht dem Wert des *Hidden Vector* zum Zeitpunkt h_t und die parametrisierten σ Entscheidungs-*Gates* werden von vier auf lediglich zwei reduziert. Dadurch sinkt sowohl die Gesamtzahl der *Weights* pro *Layer* als auch die für die Berechnung von σ benötigten arithmetischen Operationen (vgl. [37, 39], Abbildung 3.3). Obwohl Chung et al. in einem empirischen Testaufbau GRU nicht gegenüber LSTM bevorzugten, merken die Autoren an, dass die *Performance* stark vom Datensatz und der entsprechenden Aufgabe abhängig ist.

[1] Cho et al. unterschlägt im Original „aus Gründen der Einfachheit“ die genauen *Bias*-Positionen [37]. Daher wird hier die Implementation des verwendeten Frameworks angegeben.

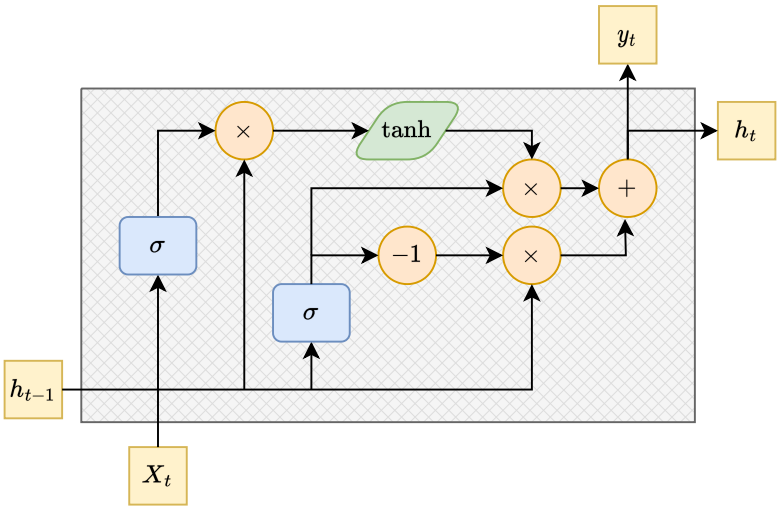


Abbildung 3.3: Die interne Struktur einer GRU Zelle.

3.2.4 *Echo State Network*

Das auch als *Reservoir Computing* bekannte Verfahren wurde unabhängig voneinander von Jaeger als *Echo State Network* und Maass, Natschläger und Markram als *Liquid State Machine* entwickelt. *Reservoir* leitet sich hier von einem Modul ab, welches durch zufällige Verknüpfungen zwischen zufällig parametrisierten nichtlinearen *Zellen* definiert ist. In diesem System können ebenfalls Zyklen auftreten, sodass ein latenter Zustand und damit eine nichtlineare Transformation der Historie über mehrere Aktivierungen hinweg aufrechterhalten werden kann. Diese Verbindungen sind allerdings nicht vollständig, sondern weisen lediglich um etwa 1% Konnektivität auf. Ein *Fully-Connected-Layer* mit *Weights* W_o liest diesen Zustand aus und berechnet den *Model-Output*. Im Gegensatz zum RNN werden bei *Echo State Networks (ESNs)* lediglich die *Output-Layer-Weights* durch lineare Regression, welche durch den *Mean Squared Error (MSE)* approximiert werden kann, trainiert. Die *Reservoir-Weights* hingegen sind ab dem Zeitpunkt der randomisierten Initialisierung fixiert. (vgl. Abbildung 3.4) [112, 153, 154].

$$\begin{aligned} h_t &= \tanh(w_x \odot x_t + Wx(t-1)) \\ y_t &= w_o \oplus h_t \end{aligned} \tag{3.5}$$

Gut initialisierte ESNs weisen nach einer anfänglichen Einschwingphase eine Garantie (*Echo State Property (ESP)*) dafür auf, dass der Aktivierungszustand eine Funktion der *Inputs*-Historie ist. In Gleichung 3.6 bezeichnet E eine *Echo-Funktion* $E = (e_1, ..., e_N)$, wobei $e_t : U^{-\mathbb{N}} \to \mathbb{R}$ die *Echo-Funktion* der i-ten Einheit ist:

$$h_t = E(..., x_{t-1}, x_t) \tag{3.6}$$

Aufgrund der fixierten und randomisierten w_h des Reservoirs spielt die Güte

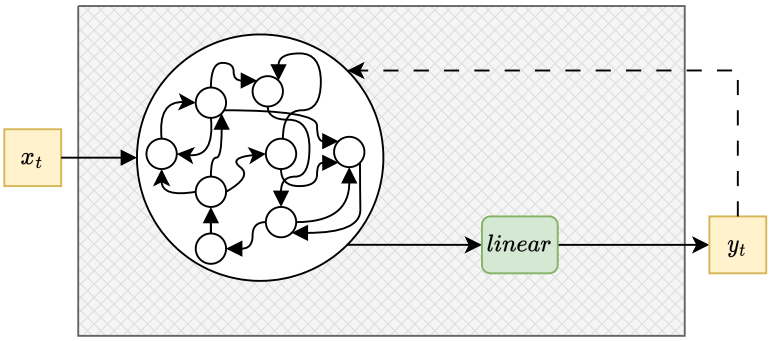


Abbildung 3.4: Ein ESN mit allen möglichen Arten von Verbindungen. Die Legende gibt an, welche Verbindungen fest, trainierbar und optional sind.

deren Initialisierung eine wichtige Rolle. Daher wurden mehrere Bedingungen für gute *Echo States* vorgeschlagen, von denen der bekannteste besagt, dass der spektrale Radius und die Norm von w_h kleiner als Eins sein müssen [112], was aber wiederum von anderen Autoren [161, 250] als weder ausreichend noch notwendig beschrieben wurde.

Im Kontext dieses Kapitels ist Folgendes interessant: Die Menge an Informationen des vergangenen *Inputs* $x_{t-n}n < t$, die h_t des ESN gespeichert werden kann, wird oft als Kurzzeitgedächtnis bezeichnet, dessen Fassungsvermögen gemessen und als *Memory Capacity (MC)* bezeichnet werden kann [112]. MC ist durch die Summe des Korrelationskoeffizienten zwischen *Input* und *Output* definiert [59]:

$$MC = \sum_{k=1}^{K=\infty} MC_k = \sum_{k=1}^{\infty} \frac{cov^2(x(t-k), \tilde{x}(t-k))}{Var(x(t)) * Var(\tilde{x}(t-k))} \tag{3.7}$$
$$\text{mit } \tilde{x} = w_o \oplus x_t, K \approx |y_t|$$

Das Konzept der ESNs hatte seine Blütezeit deutlich vor der *Deep Learning*-Revolution der 2010er-Jahre, als viele RNN-Varianten entstanden und das Problem des VGP z. B. durch *Gradient Clipping (GC)* verringert wurde. Dennoch wurden auch danach „tiefe“ ESNs-Varianten vorgeschlagen, um die Vorteile der nichtlinearen Schichtungen in NNs im Kontext neuer NN-*Architekturen* anzuwenden (z. B. [108, 140, 141, 147, 233, 240]). *Reservoir Computing (RC)* ist auch heute noch ein aktives Forschungsgebiet, da die *Models* schneller zu trainieren sind und aufgrund der festen *Weights* des Reservoirs eine präzise Beschreibung der dynamischen Eigenschaften des HV ermöglichen.

3.3 Experimenteller Aufbau

Um die Informations-Übertragungsgrenzen aus vorhergegangenen Zuständen $h_{t-n}; 1, \ldots, n \in \mathbb{N}^+$ durch rekurrente Verbindungen in RNNs aufzudecken, soll eine Aufgabe konstruiert werden, bei der das RNN gezwungen ist, ein bestimmtes Element der x_t *Input*-Sequenz x zu reproduzieren. Wie zuvor motiviert, impliziert dies, dass keine Korrelation zwischen den Elementen der *Input*-Sequenz bestehen darf, diese also i. i. d. sein müssen. Es soll keine Möglichkeit bestehen, x_{t-n} durch $\langle x_{t-n} \cap x \rangle$ zu rekonstruieren, ohne x_{t-n} vollständig durch eine Repräsentation in allen latenten Zuständen $\langle h_{t-n}, \ldots, h_T \rangle$ bis zur finalen Ausgabe y_T weiterzugeben. Eine Steigerung des Schwierigkeitsgrades ist möglich, indem das Experiment mit Sequenzen zufälliger Länge durchgeführt wird, so müssen immer die p letzten Werte vorgehalten werden. Die zu trainierende Aufgabe (vgl. Experiment 3.3.1) ist dann die Reproduktion von x_p in einem Trainingsaufbau für p wobei $-q < p; p, \ldots, -1 \in -\mathbb{N}$. Dies soll das kanonische Experiment sein, um den Informationstransport oder die *Memory Horizon (MH)* von RNNs zu messen:

Experiment 3.3.1: Random Memorization Task

q sei eine Zufallszahl, die gleichmäßig aus $[10; 15] \subset \mathbb{N}$ gezogen wird (i. i. d.). $x = \langle x_t \rangle_{1 \leq t \leq q, t \in \mathbb{N}}$ sei eine Folge von Zufallszahlen x_t, die jeweils gleichmäßig aus $[0; 1] \subseteq \mathbb{R}$ gezogen werden. Sei $\mathcal{R}$ ein RNN, das auf die Folge x sequenziell mit dem Ergebnis $y_T = \mathcal{R}(x) : \mathbb{R}^T \to \mathbb{R}$ angewendet wird. Der *Transport des Elements* x_p, wobei $-q < p; p, \ldots, -1 \in \mathbb{N}$, ist geg. durch die zu minimierende Zielfunktion $f_p(\mathcal{R}, x) = |x_{q+1-p} - \mathcal{R}(x)|$.

Die zufällige Länge der Sequenz ist hierbei entscheidend, andernfalls könnte ein RNN, das darauf trainiert ist, die x_p zu reproduzieren, den Einfluss der Elemente beginnend mit x_1 so einstellen, dass am Ende y_T immer noch ein annähernd korrekter Wert im HV gehalten werden kann. Durch Experiment 3.3.2 wird als eine Überprüfung dieser These definiert:

Experiment 3.3.2: Fixed-Length Random Memorization Task

$q = 10$. Alle weiteren Definitionen folgen Experiment 3.3.1.

Da Experiment 3.3.1 & 3.3.2 in starkem Kontrast zu den üblichen Anwendungen von RNNs stehen, bei denen explizit eine Korrelation zwischen den Elementen in x vorhanden ist, modelliert Experiment 3.3.3 die Gegenthese zu Experiment 3.3.1:

Experiment 3.3.3: Correlated Memorization Task

$q = 10$. $v = \langle v_t \rangle_{1 \leq t \leq q, t \in \mathbb{N}}$ sei eine Folge von Zufallszahlen v_t, die jeweils gleichmäßig aus $[0; 1] \subseteq \mathbb{R}$ gezogen werden. $x = \langle x_t \rangle_{1 \leq t \leq q, t \in \mathbb{N}}$ sei eine Folge, sodass $x_1 = v_1$ und $x_t = \alpha_t \cdot x_{t-1} + (1 - \alpha_t) \cdot v_t$ für $2 \leq t \leq q$ sowie ein Korrelationsparameter $\alpha_t = \frac{t}{q}$. Alle weiteren Definitionen folgen Experiment 3.3.1.

Intuitiv definiert Experiment 3.3.3 damit eine Folge x, sodass jedes Element x_t in einem (parametrisierten) Maße (α_t) von dem vorherigen Element x_{t-1} und damit von der gesamten vorhergehenden Sequenz $x_{[1,t-1)}$ abhängt. $\alpha_t = \frac{t}{q}$ ist so gewählt, dass das erste Element x_1 der Folge x immer noch mit dem letzten Element x_q korreliert, während der Einfluss der Zufallselemente in x in der Folge abnimmt. Durch diese Verkettung der Informationen sollte ein RNN in der Lage sein, weiter entfernte Elemente (in der Vergangenheit) zu reproduzieren, indem es die Korrelationsfunktion erlernt. Dieses Experiment entspricht eher der ursprünglich beabsichtigten Anwendung von RNNs.

3.4 Durchführung und Ergebnisse

In diesem Kapitel werden die Ergebnisse der Durchführung der in Abschnitt 3.3 formulierten Experimente vorgestellt.

3.4.1 Random Memorisation

Um Experiment 3.3.1 zu evaluieren, werden mehrere Experimente mit verschiedenen Konfigurationen an RNNs, LSTMs und GRUs durchgeführt. Dabei wird eine Konfiguration immer durch die Anzahl der *Layer* l und der *Zellen* c angegeben. Aus Gründen der Vergleichbarkeit zwischen den *Zelle*-Typen, wird davon abgesehen, komplexere *Architekturen* oder moderne Netzwerktrainingstechniken wie GC [183] oder *Weight Decay (WD)* [90, 150] zu betrachten. So werden nur rekurrente *Fully-Connected-Layer* untersucht, wobei die letzte Ausgabe summiert wird, wenn die Anzahl der *Zelle* $c > 1$ ist:

$$y_T = \begin{cases} \sum y_T, & \text{if } c > 1 \\ y_T, & \text{sonst} \end{cases} \tag{3.8}$$

Die Trainingsdaten x, v, y werden durch Stichproben aus einem *Mersenne Twister (MT)* Pseudo-Zufallszahlengenerator [160] erzeugt, dabei wird jede Konfiguration für bis zu 5000 Epochen für 10 verschiedene *Seeds* trainiert. Gradienten-Updates werden durch SGD mit dem *truncated* BP-Algorithmus vorgenommen, wobei das *Loss* das y_t (finalen *Output*) und den Wert des zu

reproduzierenden Elements vergleicht[2]: $\mathcal{L} = \sum_{i=1}^{D}(y_T - x_p)^2$

Ergebnisse des Zelltyps RNN

Zunächst werden 10 unabhängige Trainingsläufe für jede zu reproduzierende

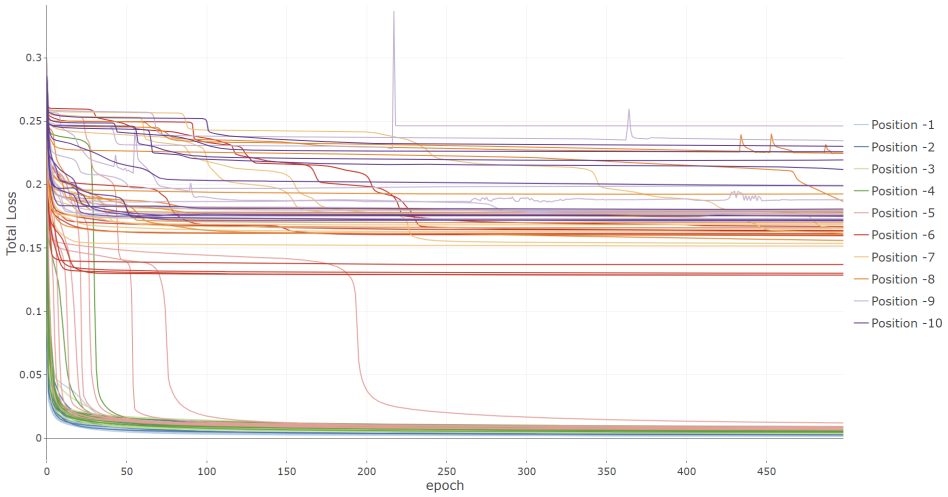


Abbildung 3.5: Absoluter *Loss* für RNN mit $l = 1$ und $c = 5$. Jede Farbe entspricht 10 verschiedene Trainingsläufe für die zu reproduzierenden Elementpositionen p.

Position $[p-1, \ldots, p-10]$ durchgeführt, wobei $l = 1$ *Layer* und $c = 5$ *Zellen* entspricht. Abbildung 3.5 präsentiert als Metrik den zusätzlich aufgezeichneten *Mean Absolute Error* für ein RNN über den Verlauf des Trainings: Die Reproduktion der Positionen $-1 \leq p \leq -4$ wird schnell erlernt, sodass sich die meisten Linien am unteren Rand der Grafik überschneiden. Die Reproduktion von $p = -5$ scheint schwieriger erlernbar zu sein, da einige Durchläufe deutlich mehr Zeit (x-Achse) benötigen, um einen Fehler nahe null zu erreichen. Es kommt aber in absehbarer Zeit zur Konvergenz. Trainingsläufe mit dem Ziel der Reproduktion der Positionen $-6 \leq p \leq -8$ sind nicht in der Lage, mit einem *Loss* $> 0,15$, die ursprünglichen Eingaben zuverlässig zu reproduzieren. Allerdings scheint etwas an Informationen über den ursprünglichen Wert des zu reproduzierenden Sequenzelements durchgereicht zu werden. Denn, bei den Durchläufen der Positionen $p = 9$ und $p = 10$ scheint der *Loss* bei etwa $0,25$ dann zu stagnieren. Dieser *Loss* entspricht bei dem *Input* von $x_t \in [0; 1] \subseteq \mathbb{R}$; dem Mittelwert eines konstanten *Outputs* von $\mathcal{R}(\mathbf{x}) = 0.5$, was dem durchschnittlichen *Loss* von $0,25$ bei einer Sequenz von Zufallszahlen entspricht. Wenn also keine Reproduktion erlernbar ist, und wird das im Folgenden als *Guess* bezeichnet. Dieses Verhalten kann bei allen RNN-Konfigurationen beobachtet werden.

Um die Auflösung des vermuteten MH zu erhöhen, wird Experiment 3.3.1

[2]Die Angabe der Summe ist hier irrelevant, da zwei Skalare verglichen werden.

auf weitere Konfigurationen ausgeweitet. Der Suchraum zur Beobachtung des MH umfasst $l \in [1;5]$ *Layer* und $c \in [1;20] \subseteq \mathbb{N}$ *Zellen* pro *Layer*. Mit diesen Parameterkonfigurationen wird untersucht, ob und wie es dem jeweiligen *Model* gelingt, Elemente $x_p, p \in [1;20] \subset \mathbb{N}$ zur letzten Position des Ausgabevektors zu transportieren.

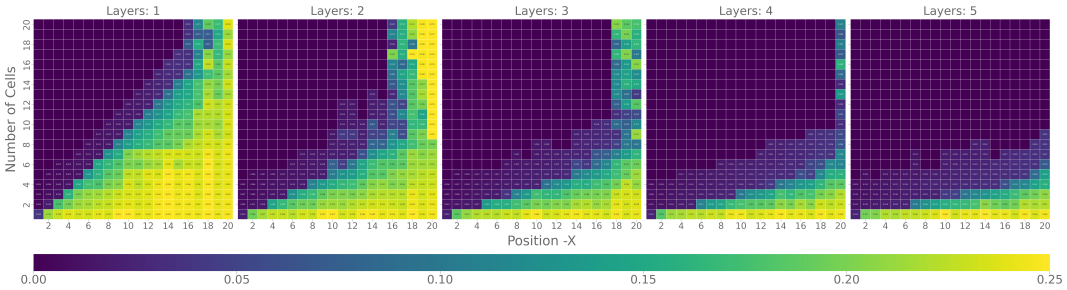


Abbildung 3.6: *Mean Absolute Error* für verschiedene Konfigurationen von RNN mit $l \in [1;5]$ und $c \in [1;20]$, *Seeds* $s \in [1;10]$.

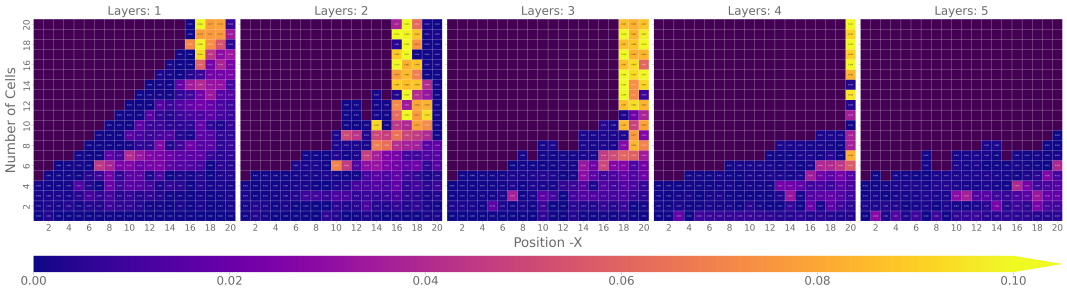


Abbildung 3.7: Varianz für verschiedene Konfigurationen von RNN mit $l \in [1;5]$ und $c \in [1;20]$, *Seeds* $s \in [1;10]$.

Die Ergebnisse für *Zellen* des Typs RNN sind in Abbildung 3.6 dargestellt, durch die Visualisierung des durchschnittlichen *Mean Absolute Error (MAE)* (*Loss*) über 5 Trainingsläufe, als farbgebendes Element. Die Farbskala ist dabei so gewählt, dass $Loss = 0.25$, wie zuvor beschrieben, den Maximalwert (gelb) und $Loss = 0.0$ eine perfekte Übereinstimmung (lila) repräsentiert. Es kann beobachtet werden, dass $l = 1$ & $c = 1$ (unten links) lediglich den letzten Wert der Zufallsfolge erfolgreich reproduzieren kann. Hier muss lediglich die Identitätsfunktion ohne Einfluss des HV abgebildet werden (vgl. Abschnitt 3.2.1), was kein Problem für RNN (und Varianten) darstellt. Ferner ist zu erkennen, die Schwierigkeit der in Experiment 3.3.1 umrissenen Aufgabe monoton mit der zu rekonstruierenden Position p ansteigt (scharfe Grenze im Farbverlauf). Dies kann als Bestätigung der Gültigkeit der Annahme in Experiment 3.3.1 gewertet werden, da das RNN hier erfolgreich gezwungen wird, individuelle Elemente der Eingabesequenz über einen gewissen Zeitraum hinweg $p_{-n;-1}$ in seinem HV zu repräsentieren. Dies zeigt auch, dass eine *Generalization* nicht möglich ist.

Das vielleicht interessanteste Ergebnis ist, dass die Anzahl der Instanzen, die zu einem durchschnittlichen *Loss* von etwa $[0,05; 0,2]$ führen (hellblau, grün), relativ gering ist: Für eine genauere Untersuchung ist die Varianz der Experimente in Abbildung 3.7 abgetragen. So ist zu erkennen, dass bestimmte Elemente der Eingabesequenz, mit einer Varianz nahe null (blau) entweder sehr genau oder kaum besser als *Guess* reproduziert werden können. Dies impliziert eine starke Korrelation zwischen der Menge an Informationen, die ein RNN transportieren kann und der jeweils gewählten Konfiguration. Ferner ist zu beobachten, dass breitere (*Zellen*) und tiefere (*Layer*) RNNs eindeutig bei der Lösung des Problems in Experiment 3.3.1 helfen. Es ist zu beachten, dass die Menge an *Zellen* mit der Anzahl an *Layer* linear skaliert, sodass eine Konfiguration mit $c = 5, l = 3$ über insgesamt 15 *Zellen* verfügt.

Ergebnisse des Zelltyps LSTM

Abbildung 3.9 zeigt die Ergebnisse für Experimente, bei denen LSTM-*Zellen* für die Lösung des Problems in Experiment 3.3.1 verwendet werden. Der dazugehörige MAE ist in Abbildung 3.8 dargestellt. Es ist zu erkennen, dass LSTMs diese Aufgabe schlechter als RNN-*Models* lösen. So kann das Element der Eingabesequenz an Position $p = -10$ nicht zufriedenstellend (MAE ≥ 0.04) gelöst werden. Wie in Abschnitt 3.2.2 beschrieben, kann vermutet werden, dass die erhöhte *Weight*-Anzahl und das rechenintensive *Model*-Training insgesamt mehr Trainingszeit erfordert, als im Rahmen dieses Experiments durchgeführt werden konnte. Da diese relativ kleine Aufgabe große Rechenressourcen verbraucht, scheinen LSTMs zumindest bei (kleineren Konfigurationen) für die Lösung des Problems in Experiment 3.3.1 ineffizient zu arbeiten.

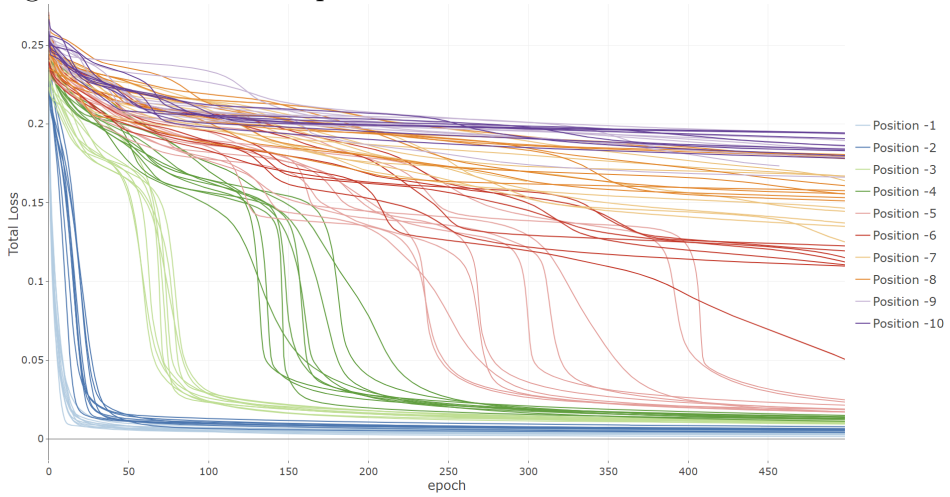


Abbildung 3.8: Absoluter *Loss* für LSTM mit $l = 1$ und $c = 5$. Jede Farbe entspricht 10 verschiedene Trainingsläufe für die zu reproduzierenden Elementpositionen p.

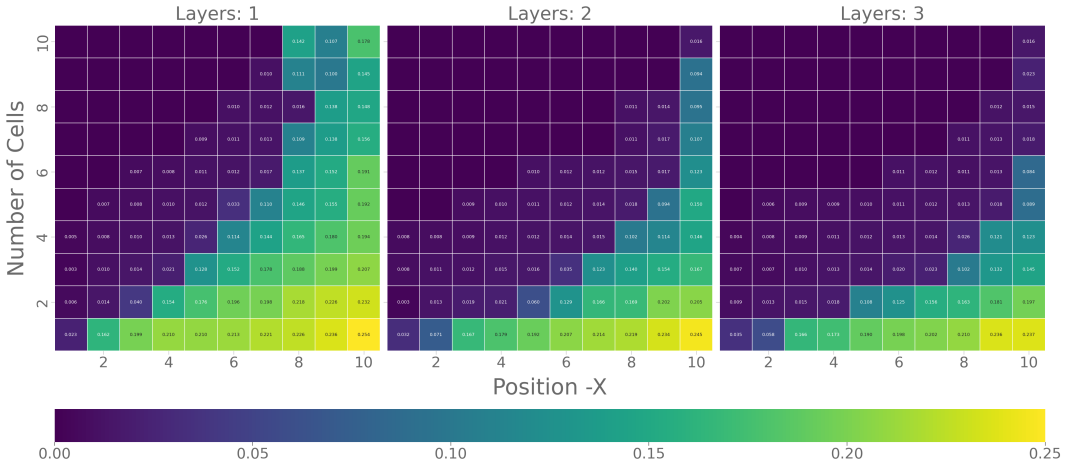


Abbildung 3.9: *Mean Absolute Error* für verschiedene Konfigurationen von LSTMs mit $l \in \{1, 2, 3\}$, $c \in [1; 10]$, *Seeds* $s \in [1; 10]$.

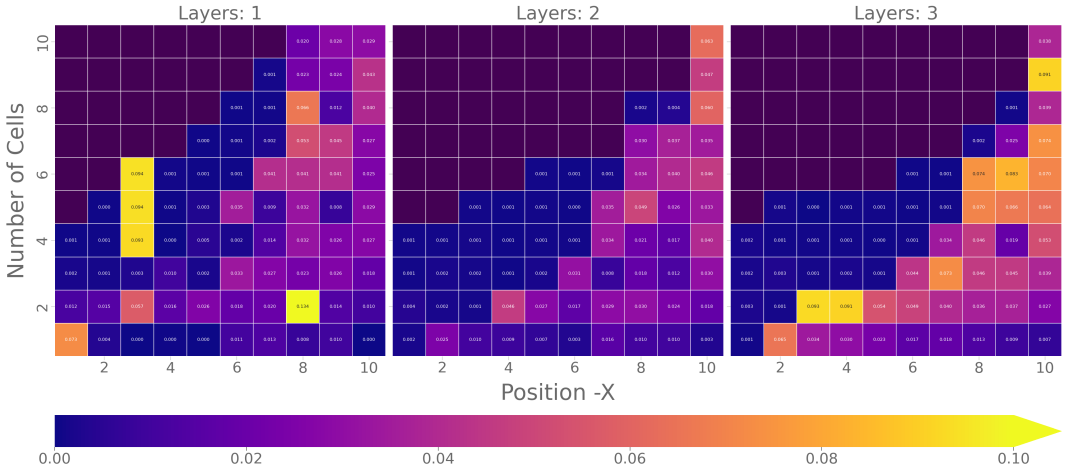


Abbildung 3.10: Standardabweichung für verschiedene Konfigurationen von LSTMs mit $l \in \{1, 2, 3\}$, $c \in [1; 10]$, *Seeds* $s \in [1; 10]$.

Ergebnisse des Zelltyps GRU

Schließlich wird auch GRU durch Experiment 3.3.1 in Abbildung 3.11 untersucht, dabei sind Ähnlichkeiten in der Charakteristik zu LSTMs zu erkennen. Grundsätzlich auffällig ist jedoch die geringere Varianz der Messungen. Eine erste Vermutung wäre, dies auf die reduzierte Komplexität dieser *Zelle*-Variante zurückzuführen. In allen weiteren Experimenten verhält sich GRU, auch ohne explizite Visualisierung an dieser Stelle, immer gleich. GRUs scheinen aufgrund weniger *Weights* ($|c| = 3 \cdot (mn + n^2 + n)$) einen interessanten Kandidaten für

effiziente praktische Anwendungen darzustellen [47].

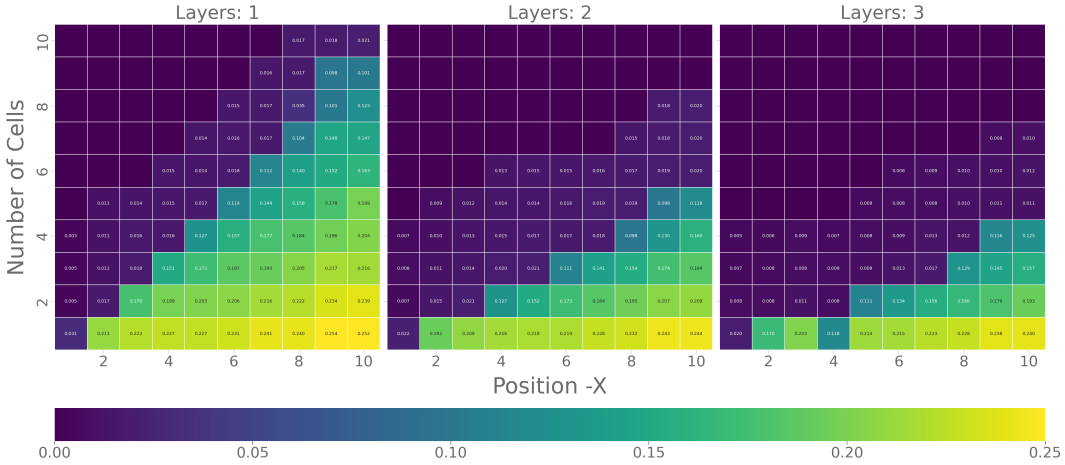


Abbildung 3.11: Mittelwert für verschiedene Konfigurationen von GRUs mit $l \in \{1, 2, 3\}$, $c \in [1; 10]$, *Seeds* $S \in [1; 10]$.

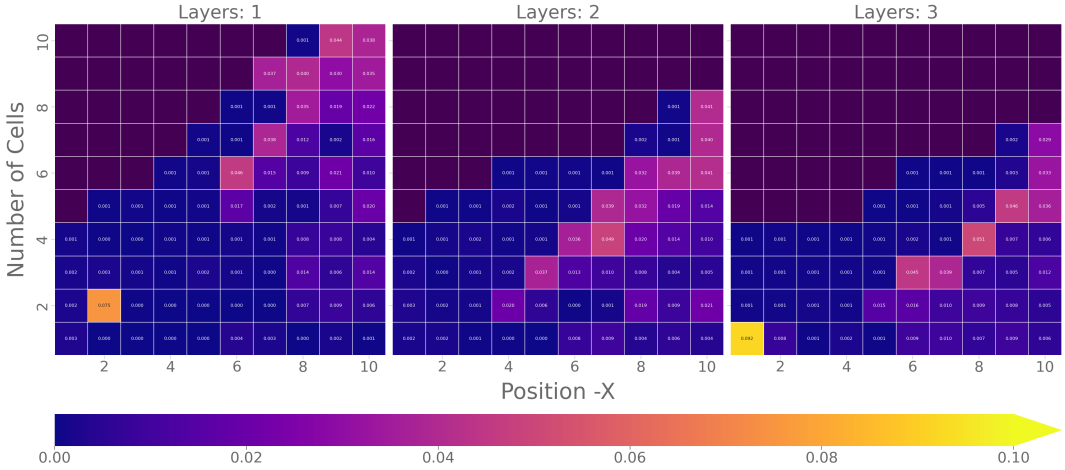


Abbildung 3.12: Standardabweichung für verschiedene Konfigurationen von GRUs mit $l \in \{1, 2, 3\}$, $c \in [1; 10]$, *Seeds* $S \in [1; 10]$.

Zusammenfassung

Die Ergebnisse dieses Kapitels weisen insgesamt darauf hin, dass *Jaegers* [111] obere Schranke für das ESN-Kurzzeitgedächtnis (MC), auch im Zusammenhang mit durch BP trainierten nichtlinearen RNN-Varianten (RNN, LSTM, GRU), gültig ist. Bei tiefen, mehrschichtigen RNN-Varianten konnte diese theoretische maximale MC bzgl. der hier untersuchten Konfigurationen nicht bestätigt werden, stattdessen wird hier eine Obergrenze von $MC \leq N-(l-1)$ mit $N = c*l$ beobachtet. Da der *Model-Output* nur aus dem HV der letzten Schicht erzeugt wird, könnte dies auf eine technische Einschränkung zurückzuführen sein. Auch Jaeger geben an, in Experimenten mit i. i. d.-Sequenzen an linearen RNNs nicht die volle theoretische MC, sondern 80 % davon erreicht zu haben. Die Vermutung liegt nahe, dass eine ESNMC auf die zufällige und unkontrollierbare Natur des zyklischen Reservoirs stark beeinflusst ist. Ein zufälliger Initialisierungszustand

und damit die Ausgangsposition für ein gradienten-basiertes NN-Training könnte einen vergleichbaren Einfluss auf die RNN-MC haben.

3.4.2 Fixed-Length Random Memorization

An dieser Stelle soll nun Experiment 3.3.2 ausgewertet werden. Im Gegensatz zu Experiment 3.3.1 wird die Länge der Eingabesequenz auf $q = 10$ fixiert. So ist es nun theoretisch möglich, die Element-Positionen in x von Anfang an zu „zählen“ und abzuschätzen, zu welchem Zeitpunkt t_n das Element der Eingabesequenz in den HV übernommen werden soll. Es müssen nun also nicht mehr Elemente mehrerer Zeitschritte im HV mitgeführt werden, sodass diese Aufgabe konzeptionell einfacher zu lösen sein sollte.

Die experimentellen Ergebnisse in Abbildung 3.13 bestätigen diese Einschätzung. Allerdings scheint diese einfachere, abzählbare Aufgabe einer gewissen Speicherbegrenzung zu unterliegen. Während jede Position, welche zuvor (*Random Memorization*) leicht zu reproduzieren war, auch in diesem Szenario lernbar ist, erscheinen weitere reproduzierbare Instanzen auf der rechten Seite des Plots. $p = 10$ für Konfigurationen mit $l = 1$ und $c = 2$, sowie $p \geq 8$ für Konfigurationen mit $l = 2$ und $c = 2$. Diese Beobachtung deutet darauf hin, dass ein RNN, wie auch in der gängigen Fachliteratur besprochen, nicht in der Lage ist zu „zählen“, sondern einen gewissen Bereich (vom Anfang der Eingabe, bzw. von der Ausgabe zurück) nutzen kann, um Informationen über Elemente der Gesamtsequenz gezielt zu reproduzieren. Für LSTMs und GRUs konnte durch Experimente das gleiche Verhalten bestätigt werden. Dieses Experiment hat in seiner Struktur Ähnlichkeiten mit den Untersuchungen von Jaeger, ist allerdings für keines der beschriebenen Experimente oder Ergebnisse deckungsgleich.

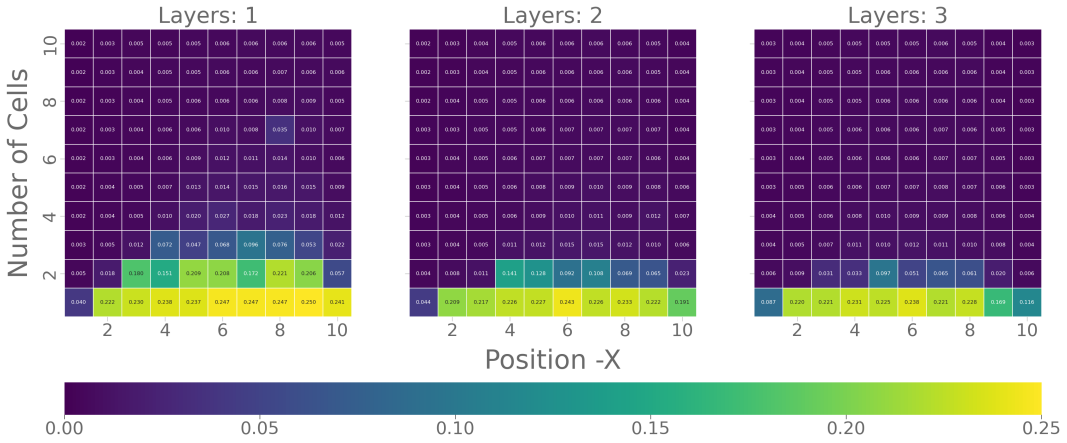


Abbildung 3.13: *Mean Absolute Error* für verschiedene Konfigurationen von RNNs mit $l \in \{1, 2, 3\}$ Schichten, mit $c \in [1; 10]$ *Zellen*, Sequenzlänge $q = 10$, *Seeds* $S \in [1; 10]$.

3.4.3 Correlated Memorization

Da die Dynamik des MH in den Konfigurationen $l \in [1;5]$ und $c, p \in [1;20]$ für i. i. d.-Eingabesequenzen in Abschnitt 3.4 untersucht wurde, soll nun als ein Gegentest explizit das Verhalten für voneinander abhängige Daten betrachtet werden. Jaeger teilen diese Sichtweise [111] im Kontext von ESN. Wie in Experiment 3.3.3 beschrieben, wird durch eine künstliche Verkettung solch eine Abhängigkeit in i. i. d.-Sequenzen hervorgerufen. Abbildung 3.14 & 3.15 zeigen jeweils die Ergebnisse für die Experimente mit RNN- bzw. LSTM-*Zellen*. Hier ist zu erkennen, dass die Reproduktion, bis auf den Fall $c = 1$, sämtlicher Positionen kein Problem für die RNN-Varianten darstellt. Sowohl RNN als auch LSTM lösen das Problem ohne Probleme und erkennen die induzierte Abhängigkeit. Aufgrund der scheinbar direkten Bestätigung der in Experiment 3.3.3 beschriebenen Vermutung und der mit den Experimenten einhergehenden Rechenzeit, wurde nur ein eingeschränkter Teilbereich des hier untersuchten Konfigurationsraums betrachtet.

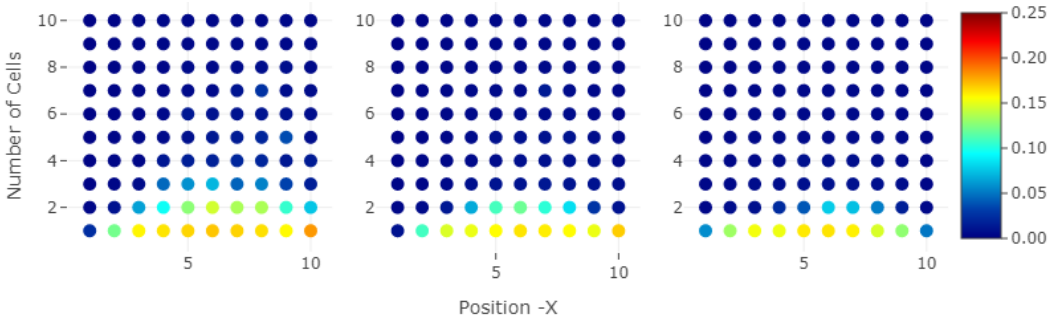


Abbildung 3.14: *Mean Absolute Error* für verschiedene Konfigurationen von RNNs mit $l \in \{1, 2, 3\}$ Schichten, mit $c \in [1;10]$ *Zellen*, Sequenzlänge $q = 10$, *Seeds* $S \in [1;10]$.

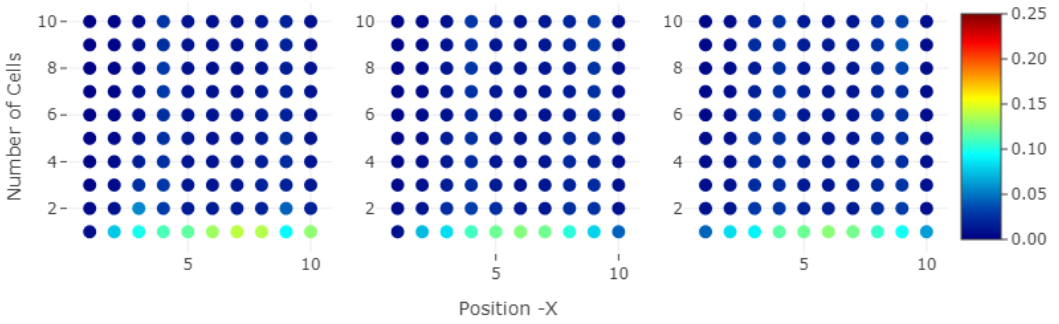


Abbildung 3.15: *Mean Absolute Error* für verschiedene Konfigurationen von LSTMs mit $l \in \{1, 2, 3\}$ Schichten, mit $c \in [1;10]$ *Zellen*, Sequenzlänge $q = 10$, *Seeds* $S \in [1;10]$.

3.5 Diskussion im Kontext verwandter Arbeiten

Die Experimente in diesem Kapitel legen nahe, dass ein explizit implementierter Zählmechanismus, auch im Kontext zufälliger i. i. d.-Sequenzen (d. h. reinem Rauschen), nicht unbedingt in allen Fällen notwendig ist. Dies steht im Gegensatz zu den von Hochreiter und Schmidhuber gemachten Beobachtungen, welche jeweils eine zusätzliche Hilfestellung benötigten, um positionsabhängige Informationsverarbeitung umzusetzen. Gleichzeitig kann durch das hier eingeführte Experiment die nach Cleeremans, Servan-Schreiber und McClelland unbedingte Abhängigkeit für einen Informationstransport widerlegt werden (vgl. [40]).

Erste Experimente zur Fähigkeit eines RNN, Informationen über eine gewisse Distanz in der Eingabesequenz zu transportieren, wurden von Cleeremans, Servan-Schreiber und McClelland durchgeführt [40]. Indem sie ein RNN mit einem endlichen Automaten verglichen [75], trainierten sie ein RNN, um eine beliebige, aber festgelegte reguläre Grammatik zu erkennen. Interessanterweise kamen sie zu dem Schluss, dass RNNs nur dann Informationen über weitreichende Distanzen in h aufrechterhalten können, wenn diese als „kritische Ereignisse“ zu jedem Zeitschritt relevant sind. Dies steht nicht unbedingt im Gegensatz zu den Ergebnissen aus Abschnitt 3.4, welche am Beispiel von i. i. d.-Sequenzen als Eingabe durchgeführt wurden. Die in diesem Kapitel untersuchten RNN-Varianten mussten Informationen für eine gewisse Zeitspanne aufrechterhalten, d. h. x_t musste für p-Schritte transportiert und dann als y_T bereitgestellt werden, ohne zu wissen, wann genau y_t eintritt.

Experiment 3.3.1 betrachtet neben Sequenzen mit Zufallszahlen auch in der Länge variable *Input*, um ein Element x_i an einer bestimmten Position zu reproduzieren. Dabei wird auf jegliche Form von Hinweisen, wie z. B. eine zusätzliche Markierung, verzichtet. Zunächst erscheinen die Experimente 4 und 5 von Hochreiter und Schmidhuber hier von Bedeutung [96], allerdings sind die Experimente in Abschnitt 3.3 so konstruiert, dass auf maximale Signalentfernung und nicht auf Speicherausdauer im Allgemeinen getestet wird. Im Gegensatz dazu sind die Experimente von Hochreiter und Schmidhuber so konstruiert, dass bei einem *Input* von zufällig generierter Länge nach der binären Antwort auf die Frage, ob ein bestimmtes Symbol vor einem anderen kommt, gesucht wird. Damit testen sie, ob der Gradient genügend rekurrente Schritte aufrechterhalten kann, sodass die beiden Informations-Bits entdeckt werden können. Bei einem zufälligen *Input* von zufälliger Länge muss in Experiment 3.3.1 dagegen ein vom *Output* n-Schritte entferntes Element reproduziert werden. Ferner wurden hier auch klassische RNNs untersucht, die über keinen GMV verfügen. Auch vermuten sie, dass gradientenbasierte Ansätze unter der praktischen Unfähigkeit leiden, diskrete Zeitschritte präzise zu zählen und gehen daher von der Notwendigkeit eines zusätzlichen Zählmechanismus aus [96]. Die Experimente in diesem Kapitel zeigen jedoch, dass ein expliziter Zählmechanismus selbst für Modelle, die auf zufälligen Sequenzen (d. h. reinem Rauschen)

trainiert wurden, nicht unbedingt notwendig ist.

Bengio, Simard, Frasconi et al. führen ähnliche Experimente durch, um das VGP zu untersuchen. Dabei geben ihre Ergebnisse Aufschluss über die Stabilität des Informationstransports unter dem Einfluss von i. i. d.-Daten. Im Gegensatz zu den Experimenten in Abschnitt 3.3 wurde dabei allerdings ein reines „Auswendiglernen“ nicht berücksichtigt. [16] So findet sich dort eine umfassendere Analyse (bzgl. der Anzahl an *Neuron*) sowie die Betrachtung weiterer RNN-Varianten, die einfacher aufgebaut sind und gleichzeitig wertvolle Einblicke in die Natur des Informationstransports liefern.

Ein direkter Vergleich der Vermutungen und Beobachtungen in diesem Kapitel ist von Jaeger in Grundzügen und in Bezug auf ESN bereits formuliert. Es ist zu betonen, dass sich die in Experiment 3.3.1 definierten Annahmen bzgl. der Bedeutung eines *Inputs* mit variabler Länge deutlich von seinen Experimenten unterscheiden. Auch die Betrachtung (neuerer) RNN-Varianten und nichtlinearer Aktivierungen hat so bisher nicht stattgefunden. Wenig erforscht sind auch Zufallssequenzen von variabler Länge als *Input* für RNN-Varianten.

Da die Fragestellungen in diesem Kapitel leicht mit Arbeiten zum Thema *Generalization* und *Overfitting* zu verwechseln sind, soll an dieser Stelle explizit darauf hingewiesen werden, dass der hier formulierte Kapazitätsbegriff MC ein völlig anderer ist. Obwohl methodisch nicht verwandt, liefern Zhang et al. eine Analyse, die im Wortlaut recht ähnlich klingt: Sie betrachten die Kapazität eines FFNN als die Anzahl der *Input-Output*-Paare der Trainingsmenge, welche im Anschluss an ein erfolgreiches Training exakt reproduziert werden können. Dieses Kapazitätsvermögen wird abschließend an einem ganzen Datensatz von *Input*-Vektoren erhoben. Die Experimente in Abschnitt 3.3 betrachten hingegen die Speicherkapazität des HV während eines einzelnen sequenziellen Aufrufs an einem einzelnen *Input*-Vektor, was eine andere Konstellation darstellt. Zwar korrelieren die von ihnen beschriebenen Probleme auch mit der in verschiedenen Strukturen angeordneten *Weight*-Anzahl, allerdings können z. B. die Trainingsdaten in den Experimenten in Abschnitt 3.3 vernachlässigt werden, sobald RNNs trainiert sind.

Weitere interessante Experimente wurden von [72] und später [85] zu LSTM-Varianten durchgeführt, dort wurden allerdings z. B. durch *Peephole Connections* Änderungen an der *Architektur* vorgenommen. Dann kann noch die Klasse von empirischen Studien genannt werden, die mittels visueller Auswertungen die innere Arbeitsweise des LSTM verdeutlichen (z. B. [121]).

Abschließend sei noch zu erwähnen, dass auch Gonon, Grigoryeva und Ortega die Speicher- und Vorhersagekapazitäten von linearen und nichtlinearen RNNs (für unabhängige und nicht unabhängige Eingaben) erforscht hat. Der mathematische und weniger empirische Ansatz der Autoren definiert eine obere Schranke für die jeweilige Fähigkeit und verallgemeinert gleichzeitig die Aussagen von Jaeger zu dessen Ergebnissen zu MC. [79]

3.6 Zusammenfassung der Ergebnisse

In diesem Kapitel wurden Experiment 3.3.1 sowie dessen logische Erweiterungen (Experiment 3.3.2 & Experiment 3.3.3) vorgestellt, um einen Einblick in die Transport- bzw. Speicherdynamik des HV bei RNN-Varianten zu erhalten. In den anschließenden Experimenten stellte sich heraus, dass (gegen die Intuition) klassische RNNs, LSTM- und GRU-Netze in der Lage sind, einzelne Elemente zufälliger Eingangssequenzen je nach Konfiguration und *Architektur* über eine gewisse Distanz durch den HV (hier h) zu transportieren. Dabei ließ sich eine leicht (weil linear) erkennbare Grenze zwischen dieser Distanz und der Anzahl an *Weights* bzw. *Zellen* erkennen: MH. Diese Grenze stimmt mit der von *Jaeger* für ESN formulierten MC annähernd überein.

Obwohl diese Experimente, aufgrund der benötigten Rechenleistung, nur einen begrenzten Umfang haben, ist der von der Anzahl an *Zellen* abhängige Trend deutlich zu erkennen. Da in diesen Experimenten nicht durch GC oder andere Regularisierungstechniken eingegriffen wurde, ist ein deutlich negativer Einfluss auf die untersuchten RNN-*Architekturen*, durch VGP ausgelöste Effekte, zu erkennen.

In der Praxis wird das Verhältnis zwischen der Länge der Eingabesequenz und der Anzahl der verwendeten RNN-Varianten so gewählt, dass der hier untersuchte MH in der Praxis keine wirkliche Rolle spielt. Weiterhin wurde gezeigt, dass eine vorhandene Korrelation in den Trainingsdaten den MH deutlich über den beobachteten linearen Zusammenhang hinaus heben kann. Dies bestätigt auch Jaeger in Bezug auf ESNs. Dennoch ist zu hoffen, dass die hier dargelegten Experimente das Verständnis über das Lernverhalten von RNN-Varianten beleuchten und so die oft zitierte *Black-Box* der *Neural Network*s in gewisser Hinsicht öffnen können.

4 Audio-Klassifikation von Real-World-Datasets

Nach der Betrachtung der Informations-Transport-Grenzen, bzw. des MH in RNN-Varianten, soll in diesem Kapitel ein Fokus auf die praktische Anwendung von *Neural Network*s im Kontext sequenzieller Audiodaten gelegt werden.

Die Forschung im Bereich der Interpretation von menschlicher Sprache ist seit jeher ein aktives Forschungsfeld. Die Motivation reicht dabei von einem Wunsch der Unterstützung von Menschen mit Behinderungen, Monetarisierungsabsichten im Endkundengeschäft (B2C) (z. B. Google Assistant[1], Amazon Alexa[2]) oder schlicht Visionen der Popkultur[3] über viele Branchen hinweg. Als neueste und sehr medienwirksame Ausprägung kann *ChatGPT*[4], ein modernes Sprachmodell, genannt werden. Dieses generiert, als natürliche menschliche Kreationen erscheinende, strukturierte Texte oder auch Code auf Basis von Eingaben (sog. *Prompts*).

Ein verwandter Forschungszweig befasst sich mit der Verarbeitung von Aufnahmen menschlicher Sprache in Form sequenzieller Audiodaten. Tandel, Prajapati und Dabhi liefert einen Überblick über verschiedene Anwendungsfälle, verwendete Techniken, *Performance*-Metriken und Datensätze [228]. Neben der Spracherkennung kommen, oft als Vorstufe, weitere Verfahren zum Einsatz, die es z. B. erleichtern, Text aus rohen Audiodaten zu extrahieren. Man unterscheidet dabei zwischen der *Audio-Event-Detection* [11], bei der primär entschieden werden soll, wann und für wie lange etwas Bestimmtes passiert (Sprache setzt ein, oder hört auf) und der *Audio-Segmentation*, bei der homogene (zusammenhängende) Sequenzbereiche (zeit- oder frequenzbasiert) herausgearbeitet werden [65], und der Klassifikation, bei der *Samples* fest definierter Klassen zugeordnet oder neue gebildet werden sollen [24]. Dabei finden diese Verfahren nicht exklusive Verwendung, sondern können auch innerhalb einer Pipeline miteinander kombiniert werden [214]. Da sich Audioaufnahmen als ein scheinbar günstiges Werkzeug zum Zwecke der Überwachung und Anomalieerkennung von z. B. Maschinen [168–170] und wilden Tieren [258] verwenden lässt, ist das Erkennen des eigentlichen Signals vor etwaigen Hintergrundgeräuschen (*background*, *noise*) von besonderer Bedeutung [13]. Dabei hat sich hier die

[1] Google Assistant - `https://assistant.google.com/`
[2] Amazon Alexa - `https://developer.amazon.com/en-US/alexa`
[3] Science-Fiction, z. B. *2001: Odyssee im Weltraum*
[4] OpenAI ChatGPT - `https://openai.com/blog/chatgpt/`

Klasse der verwendeten Algorithmen, wie in anderen informationstechnischen Bereichen auch, stark in Richtung von ML und DNN entwickelt.

Wie zuvor beschrieben erfasst das Format der Audiodaten eine kontinuierliche Funktion der Zeit und diskretisiert diese mit einer hinreichenden Auflösung. Dabei kann das Ursprungssignal nicht nur in seine Frequenzanteile zerlegt und visuell erkundet werden, sondern auch für DL-basierte Verfahren ein Mehrwert geschaffen werden. Mel-Spektrogramme führen durch ihre Eigenschaften eine räumliche Komponente ein und erschließen als eine Art Vorverarbeitungsschritt rohe Audiodaten für DNN aus dem Forschungsfeld der *Computer Vision*.

Wie zu Beginn (Kapitel 4) motiviert, soll der Schwerpunkt in diesem Kapitel auf die praktische Anwendung von NN auf Audiodaten am Beispiel von *Real-World*-Datasets eingegangen werden. Dabei soll der Fokus auch auf Schwierigkeiten und Besonderheiten im Kontext des Datenformates sowie des hier untersuchten Datensatzes liegen. Neben der Darstellung und Erweiterung der in „Surgical Mask Detection with Convolutional Neural Networks and Data Augmentations on Spectrograms“ veröffentlichten Inhalte dient dieses Kapitel auch als erweitertes Grundlagenkapitel und Vorbereitung für das folgende. Der hierfür benötigte technische Hintergrund zur Verarbeitung sequenzieller Audiodaten sowie die allgemeine Struktur der hier verwendeten NN-*Architektur* wurde zuvor ganz generell in Abschnitt 2.4.1 besprochen.

Contribution 4.0.1: SSC für Mel-Spektrogramme

Beiträge zur Wissenschaft in diesem Kapitel umfassen die Vorstellung eines *SubSpectralNet (SSN)*-basierten *Classifier* (SSC) zur Klassifizierung von sequenziellen Audiodaten auf Basis von Mel-Spektrogrammen, sowie der Evaluation dieses und anderer NN-*Architekturen* auf einem *Real-World*-Dataset. Vorveröffentlichung durch Illium et al. in „Surgical Mask Detection with Convolutional Neural Networks and Data Augmentations on Spectrograms“ [101]

Nach einer kurzen Einführung und Motivation in Abschnitt 4.1, werden (in 4.2) verwandte Arbeiten vorgestellt. Dann folgt in 4.4 eine Beschreibung der hier verwendeten Methoden, sowohl zur auf Audiodaten angepassten *Data Augmentation*, als auch der *Neural Network*s. Nach einem Blick auf Besonderheiten in den Praktiken zur Verarbeitung sequenzieller Audiodaten durch NN, werden die experimentellen Ergebnisse auf einem *Real-World*-Dataset demonstriert (4.6) und in 4.7 diskutiert.

4.1 Einführung

Gerade im Forschungsfeld des ML (aber auch darüber hinaus) hat sich ein Habitus entwickelt, welcher immer kleiner werdende *Performance*-Gewinne an

scheinbar bessere Verfahren knüpft und diese über bestehende Modelle, *Architekturen*, Trainingskomponenten oder -Verfahren knüpft. Dabei handelt es sich oft um kleinere wissenschaftliche Beiträge bzw. Änderungen am Gesamtkonzept, was es schwierig macht, in der Praxis die richtigen Schlüsse zu ziehen oder eine Entscheidung hinsichtlich der für die eigenen Fragestellungen passenden Verfahren zu treffen. Betrachtet man solche NN-*Architekturen* abseits der häufig zitierten und untersuchten Datensätze, ergibt sich oft ein anderes Bild in diesen sozusagen fremden Domänen.

Audiodaten sind deswegen, gerade in der Praxis, ein spannendes Forschungsfeld, weil das ubiquitäre Vorkommen von Ton und der hinterlegten Semantik zu vielen möglichen Einsatzgebieten führt. So können nicht nur industrielle Maschinen [168–170, 190] und Wildtiere [102, 166, 208, 258] kosteneffektiv und nicht-invasiv überwacht werden, sondern auch menschliche Stimme [115, 127, 180, 211] oder der Kontext dieser [14, 55, 101, 206, 209] durch moderne Verfahren verarbeitet werden. Neben der großen Menge an HP auf verschiedenen Ebenen (*Preprocessing (PP)*, DA, *Architektur*, *Model*, *Performance*-Metriken), sind dabei auch Besonderheiten hinsichtlich des NN-*Model*-Trainings und im PP zu beachten. Es stellt sich die Frage, inwieweit sich gängige *Architekturen* in praktischer Anwendung auf *Real-World*-Dataset trainieren lassen und welche Besonderheiten dabei beachtet werden sollten. Diese Punkte sollen exemplarisch am Beispiel einer einfachen binären Klassifikationsaufgabe besprochen werden. Vor diesem Hintergrund bietet es sich zusätzlich an, statt der gängigen Forschungsdatensätze einen *Real-World*-Dataset für die Betrachtungen in diesem Kapitel in Erwägung zu ziehen. Diese weisen in der Regel besondere Eigenschaften auf, was wiederum eine reale, praktische Anwendung annähert.

In diesem Kapitel sollen im Kontext der Audio-Klassifikation verschiedene Algorithmen vorgestellt und die Schwierigkeiten in der Arbeit auf *Real-World*-Datasets herausgearbeitet werden. Dabei wird unter anderem ein Blick auf infrage kommende DA-Methoden (vgl. Abschnitt 2.3.4.2) geworfen. Daraufhin werden Möglichkeiten aufgezeigt, wie Mel-Spektrogramme, also eine Audio-Repräsentation der Frequenzdomäne, als Sequenzen interpretiert werden können und wie dies die *Model-Performance* im Vergleich zu CNN-basierten Verfahren befördert.

4.2 Verwandte Arbeiten

In diesem Unterkapitel werden zunächst verwandte Arbeiten aus dem Bereich der Audio-Klassifikation sowie der Datensatzerweiterung durch DA im Bereich der Audioverarbeitung vorgestellt.

4.2.1 Audio Classification

Wie eingangs motiviert, umfasst das Forschungsfeld der Audioklassifikation eine Vielzahl unterschiedlicher und nicht trivialer Aufgaben. Hierzu gehört unter anderem sog. die Multi-Label-Klassifikation zur Vorhersage von Noten in Musikaufnahmen [230] oder die Vorhersage von Genre-Schlagworten (Tags) für Lieder (Songs) [48]. Spezialisierte Anwendungen können z. B. als Werkzeuge in einer höheren Pipeline genutzt werden, um z. B. die Klassifikation von Umgebungsgeräuschen [187] oder die Klassifikation von akustischen Szenen [53] zu übernehmen. Typischerweise wird hier versucht, ein einzelnes Label für einen gesamten Audioclip vorherzusagen.

Wie in den meisten anderen Forschungsfeldern auch, hat sich die Forschung im Bereich der Audio-Klassifikation weitgehend von der vormals verbreiteten Verwendung manuell konstruierter Merkmale (z. B. auf der Grundlage von *Mel-Frequenz-Cepstral-Koeffizienten (MFCC)*) wegbewegt. So verlagert sich die Entwicklung neuer Ansätze und Lösungsstrategien zunehmend auf die Einbeziehung von Gradienten basierten Lernverfahren durch *Deep Neural Network*, um automatisch aufgabenrelevante Merkmale (Features) zu erlernen (z. B. *Visual Geometry Group (VGG)*[216]). Innerhalb dieser Gruppe der DNN-basierten Audioklassifizierung kann weiter nach der Art der verwendeten Eingabedaten unterschieden werden:

i. Lernen aus Spektrogrammmerkmalen
ii. Lernen aus unkomprimierten (*raw*) Audiodaten

Mel-Spektrogramm basierte Audio-Klassifikation durch CNNs

Als valide Strategie hat sich der Einsatz von CNN in Kombination mit durch FFT berechneten Log-Mel-Spektrogrammen etabliert. Dieses Datenformat, das dem eines monochromen Bildes (2-dimensional, *Höhe*; *Breite*, vgl. Abschnitt 2.4.3) ähnelt, kann durch bestehende DL-Verfahren, wie z. B. CNNs, aus dem Bereich der Bildverarbeitung verarbeitet werden. Aufgrund der Fülle an Ansätzen für den Umgang mit Bilddaten liegt so die Annäherung der beiden Domänen nahe. Die *Performance* solcher NN-*Architektur* z. B. zum Zwecke der Klassifizierung von Umwelt- und Stadtgeräuschen auf Basis von log-skalierten Mel-Spektrogrammen wird beispielsweise von Piczak untersucht [187]. Die Autoren zeigen, dass ein tiefes CNN klassische Methoden, welche sich auf manuell konstruierte Merkmale (*Features*) stützen, übertrifft.

Sowohl Haubrick und Ye als auch Bear et al. sind als Beispiele für Arbeiten zu nennen, welche *Real-World*-Datasets durch den Mel-Spek.-CNN Ansatz klassifizieren. Dabei implementiert Haubrick und Ye, neben anderen Ansätzen, mehrere parallele, *Binary-Classifiers*, um eine Entscheidung bzgl. der durch ein CNN extrahierten *Feature* zu treffen. Bear et al. hingegen setzt auf eine ähnliche Strategie mit zwei *Multi-Class-Classifiers*.

Ein interessanter Gedanke ist, dass nicht nur das Trainingsverfahren, sondern die *Architektur* von CNNs selbst ein wichtiges Element bei der Suche nach guten Merkmalsrepräsentationen ist [188]. So zeigen die Autoren durch die Leistungsbewertung von untrainierten, d. h. zufällig gewichteten Netzwerken, dass diese selbst ohne Optimierungsschritte eine beeindruckende Genauigkeit bei verschiedenen Audioklassifizierungs-Aufgaben erreichen und damit sogar ein vergleichbares MFCC-basiertes Verfahren übertreffen können.

Palanisamy, Singhania und Yao stellt die Frage, warum dies so ist und beantwortet sie mit der Visualisierung der Gradienten-Aktivität, auf zufällig initialisierten und auf Bilddaten vortrainierten *Architekturen* [179], die wiederum zu besseren Ergebnissen kommen als. Einen ähnlichen Weg geht Müller et al. mit vortrainierten CNNs, allerdings zum Zwecke der Anomalieerkennung [170]. Darüber hinaus finden sich viele weitere Beispiele und Anwendungsfälle, wie die CNN basierte Audio-Klassifikation im Kontext von Videosoundtracks [94].

Eine andere erwähnenswerte *Architektur* (*SampleCNN*) macht sich die Eigenschaften besonders kleiner Kernel zunutze, um damit sehr effizient Musik-Klassifikationen vorzunehmen [138].

Raw-Audio basierte Audio-Klassifikation durch DL

Wie in Kapitel 3 dargestellt, ist die Verarbeitung sehr langer Sequenzen, wie sie z. B. bei Audioaufnahmen entstehen, durch DNN mit einem erheblichen Aufwand verbunden. Während bei FFNN sehr viele *Neurons* benötigt werden, tritt bei RNNs sehr wahrscheinlich das *Vanishing Gradient Problem* auf. Zwar wurden *Architekturen* entwickelt, welche zur Verarbeitung roher Audiodaten verwendet werden können (z. B. [42, 43, 48, 97, 115, 180, 202]), allerdings hat sich eine bis heute nicht gezeigt, dass diese Ansätze überlegen sind. Im Gegensatz zu den zuvor besprochenen Ansätzen wird von Dieleman und Schrauwen eine Untersuchung der *Performance* von CNNs vorgestellt, welche direkt auf rohen Audiosignalen arbeiten [48]. In dem genannten Beispiel sollen nützliche Merkmale für eine automatische Markierungsaufgabe gelernt werden. Die Ergebnisse zeigen, dass es zwar möglich ist, direkt von rohen Audiosignalen zu lernen, dass aber NN, welche Spektrogramm-basierte Eingaben verwenden, besser abschneiden als solche, die rohe Audiosignale als Eingabe verwenden. Obwohl Dai et al. zeigt, dass es für *End-to-end*-Lernansätze möglich ist, mit der *Performance* von CNNs gleichzuziehen, erreichen die Autoren dies nur, indem sie sehr tiefe NN verwenden, die folglich deutlich mehr Rechenleistung benötigen [42]. Lee et al. untersucht ein Jahr zuvor, wie *Sample-CNN* auf rohen Audiodaten arbeitet. Dabei zeigt er auf drei unterschiedlichen Datensätzen einen Vergleich zu aktuellen Methoden, übertrifft diese aber nicht.

Die hier genannten Arbeiten unterstreichen sowohl die Bedeutung von CNN-basierten *Architekturen* im Kontext von Audio-Klassifizierungs-Aufgaben sowie die Validität der dazugehörigen Transformation von rohen Audiodaten in log skalierte Mel-Spektrogramme.

4.2.2 Data Augmentation im Bereich der Audioverarbeitung

Der Einsatz von DA im Bereich der Audioverarbeitung und im Besonderen bei Klassifizierungsaufgaben ist alles andere als neu. Zum Beispiel wurde schon eine Kombination aus Störung in der Darstellung der Vokaltraktlänge (Vocal Tract Length Perturbation (VTLP)), Geschwindigkeitsveränderungen sowie Tempoverschiebung (Time Warping) eingesetzt [127]. Geschwindigkeitsstörungen erwiesen sich nach Ko et al. wohl als der hilfreichste DA-Ansatz.

Weitere Techniken, wie die teilweise Maskierung von zusammenhängenden Bereichen entlang der Zeit- und Frequenzachse (auch bekannt als *SpecAugmentation* [181]), führen auf den ersten Blick zu einem Informationsverlust in Mel-Spektrogrammen. Die Maskierung erfolgt hier durch das gezielte Setzen von Nullwerten (Löschen), es existieren aber auch Versionen, welche diese Bereiche mit Gaussian *Noise (GN)* füllen. In Kombination mit einer Tempoverschiebung wurde so gezeigt, dass es möglich ist, durch diese Art der DA eine Überanpassung des Modells gezielt zu vermeiden.

Aguiar, Costa und Silla führt neben Anderen verschiedene DA-Methoden zusammen, um die Klassifizierung von Musikgenres zu verbessern: Lautstärke, Rauschen, Time Warping und Tonhöhenverschiebung (Pitch Shift). Anschließend lernen CNN-*Models* die Klassifizierungsaufgaben [1, 4, 5, 109, 204].

Diese und vergleichbare Ansätze der Forschung im Bereich ML und/oder DL haben gezeigt, dass *Data Augmentation* eine valide Strategie darstellt. In Abschnitt 4.3.3 werden die in dieser Arbeit verwendeten Methoden der DA vorgestellt, unter deren Einfluss die Arbeitsweise der NN-*Architekturen* (Abschnitt 4.4.2) untersucht wird.

4.3 Methodische Grundlagen

In diesem Kapitel sollen zunächst die methodischen Grundlagen der Mel-Spektrogramm basierten Verarbeitung von Sequenzen durch NN eingeführt werden, wobei der Fokus auf NN-*Architekturen* liegt. Bereits in Abschnitt 2.4.2 wurde auf die Besonderheiten von Audio-Daten sowie Transformationsmöglichkeiten eingegangen.

4.3.1 *Convolutional Neural Network (CNN)*

In Abgrenzung zu der in Abschnitt 2.3 besprochenen Verarbeitung ungeordneter vektorisierter Daten, existieren viele weitere Datenformate, in denen Information durch die Positionierung der einzelnen Vektorelemente zueinander oder relativ zur Folge kodiert ist. Beispiele sind ganz generell Signale oder Sequenzen

(1D), monochrome Bilder und Audio-Spektrogramme (2D), Farbbilder und monochrome Videos (3D) oder Farb-Videos (4D).

Auch kann es vorkommen, dass sich die Positionierung gewisser Merkmale (*Feature*) als zusammenhängende Menge von Elementen (bzgl. der Semantik oder der Position zueinander), z. B. bei Bildern oder Filmen, nicht immer an der gleichen Position befinden, oder eine Verformung stattgefunden hat. Im Kontext von den RNNs in Kapitel 3 wurde dies bereits erwähnt. Zwar können FFNN in der Theorie für die Verarbeitung solcher Datenformate zum Einsatz kommen, doch praktisch ergibt sich ein enormer Trainingsaufwand durch die stark steigende Anzahl der für diese Aufgaben benötigten NN-*Weights* [134]. Um die durch die Struktur dieser Formate gegebenen Informationen besser ausnutzen zu können, wurden unter anderem das *Convolutional Neural Network (CNN)* entwickelt, welches explizit die Nachbarschaftsbeziehungen in $[1, 2, 3]$-dimensionalen Arrays modelliert [131, 132, 134] CNNs sind wie FFNNs auch als eine Struktur aus unterschiedlichen sukzessiven *Layer* aufgebaut, wobei der wichtigste Bestandteil dabei der *Convolution-Layer* ist:

Convolution

Konzeptionell arbeiten *viele* parallele, parametrisierte und räumliche Filtereinheiten in einem sog. *Kernel*, um *Feature Maps (FMs)* zu erzeugen, welche nach dem Vorbild des *Neocognitron* [66] für die Optimierung des *Losses* relevante Merkmale herausstellen. Abbildung 4.1 zeigt dies visuell. LeCun et al.

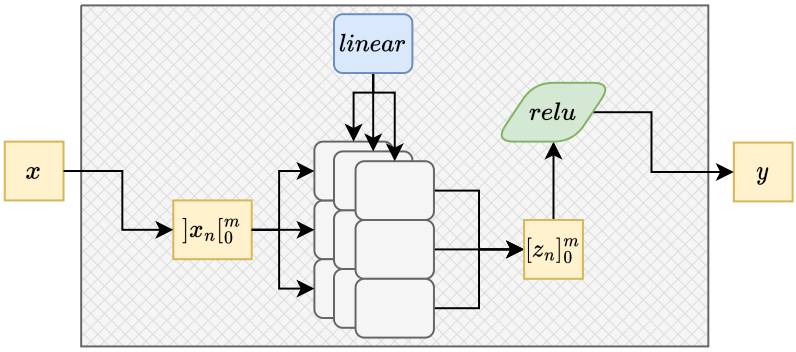


Abbildung 4.1: Struktur *Convolution-Zelle* nach Fukushima und Miyake [66]

erweitern diese Idee durch das Erlernen der *Convolution-Kernel* (*Weights*) mittels *Backpropagation (BP)* [133, 200]. Der größte Unterschied zwischen *Convolution-* und *Fully-Connected-Layer* liegt in den „geteilten“ (eng. shared) *Weights* [200]. Diese Strategie, also die Erzeugung mehrerer Pfade (im CG) durch die gleichen *Weights*, beschleunigt den Lernprozess, mit sehr geringem zusätzlichen Rechenaufwand, deutlich [3, 133, 222].

In seiner einfachsten Form seien drei Vektoren gegeben:

$$\begin{aligned} &\text{Input} = \underset{N\times N}{\overrightarrow{x}}, \text{ Weight} = \underset{M\times M\times Q}{\overrightarrow{w}} \;\&\; \mathit{Bias} = \underset{Q}{\overrightarrow{b}} \\ &\text{mit } , N \in \mathbb{N}^+, N > M \in \mathbb{N}^+, Q = |\mathit{Filter}| \in \mathbb{N}^+, \end{aligned} \tag{4.1}$$

dann ist eine parametrisierte 2-D *Convolution* definiert, als eine Funktion:

$$\begin{aligned} &NN(\overrightarrow{x}) : R^{N\times N} \rightarrow R^{\tilde{M}\times\tilde{M}\times Q}, \\ &\text{mit } \tilde{M} = (M - J + 1) \end{aligned} \tag{4.2}$$

welche die Operation:

$$\underset{\tilde{M}\times\tilde{M}\times Q}{\overrightarrow{z}} = \overrightarrow{x} * \overrightarrow{w} + \overrightarrow{b}, \tag{4.3}$$

berechnet, wobei $*$ = *sliding* dot product ($\odot$).

Die hier beschriebene mathematische Faltungsoperation (*Convolution*) kann als sequenziell ausgeführte (*sliding*) Operation verstanden werden, die ein *dot product* $\odot$ (vgl. Abschnitt 2.3) über das gesamte Volumen des Filters darstellt. Da der Filter dabei mehrfach bewegt und angewendet wird, spricht man hier von dem oben genannten *Kernel*. Abbildung 4.2 verdeutlicht diesen Vorgang, bei dem der *Kernel* über alle zulässigen räumlichen Positionen des *Input*-Vektors bewegt wird [3][5].

In der Praxis hat diese veränderte *Architektur* deutliche Vorteile, so neigen Filter in den vorderen Schichten dazu, eher primitive *Feature* zu erkennen (z. B. Texturen), während spätere Filter sich komplexere Kompositionen aus diesen erschließen. Ferner ist diese *Architektur* äquivariant zu *Feature*-Translationen, d. h., dass diese an unterschiedlichen Positionen des *Inputs* platziert und sogar gedreht oder verzerrt erscheinen können, ohne dass sich die extrahierten Features ändern. Weiterhin wird $*$ nicht etwa durch einen *Loop* realisiert, sondern stellt eine parallelisierte und damit hochperformante Operation dar.

Eine der bekanntesten Arbeiten zur visuellen Erschließung der *Convolution Arithmetics* liefern Dumoulin und Visin [51]. Diese Arbeit hilft die möglichen *Kernel*-Ausprägungen, resultierenden FM und zusätzlichen *Convolution*-HPs zu verstehen. Die da wären [3, 132, 134, 222]:

- k *Kernel* = Größe $M \times M$ der angewendeten (quadratischen) Filterebene
  Filter = Anzahl der angewendeten Filter
- s *Stride* = Schrittweite, die der *Kernel* versetzt bzw. $*$ ausgeführt wird
- p *Padding* = Zugabe (0-Werte) im Randbereich von $\overrightarrow{x}$

Geht man davon aus, dass alle diese HPs verwendet werden, verändert sich die

[5] Bei diesen beiden Darstellungen wird explizit auf die Darstellung von 3-D Farbbildern, höheren *Convolution*-Operationen, *Stride*, *Padding* oder nicht quadratische *Kernel* verzichtet.

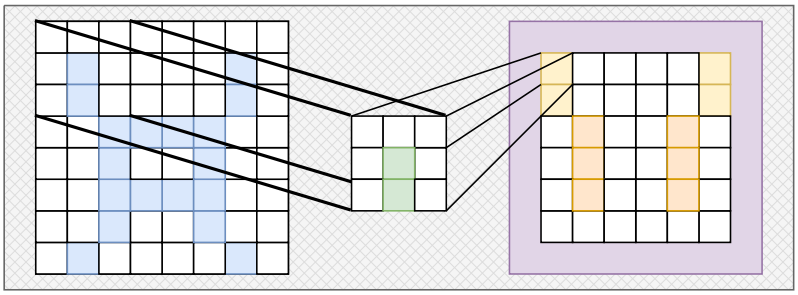


Abbildung 4.2: Funktionsweise der *Convolution* nach Fukushima und Miyake & LeCun et al. [66, 132]. Ein Kernel zur Erkennung von vertikalen Strichen wird beispielhaft appliziert.

Seitenlänge $\tilde{M}$ von $\overrightarrow{y}$ geg. eines (quadratischen) *Input* i nach angewendeter *Convolution* wie folgt:

$$\tilde{M} = \lfloor \frac{i + 2p - k}{s} \rfloor + 1 \tag{4.4}$$

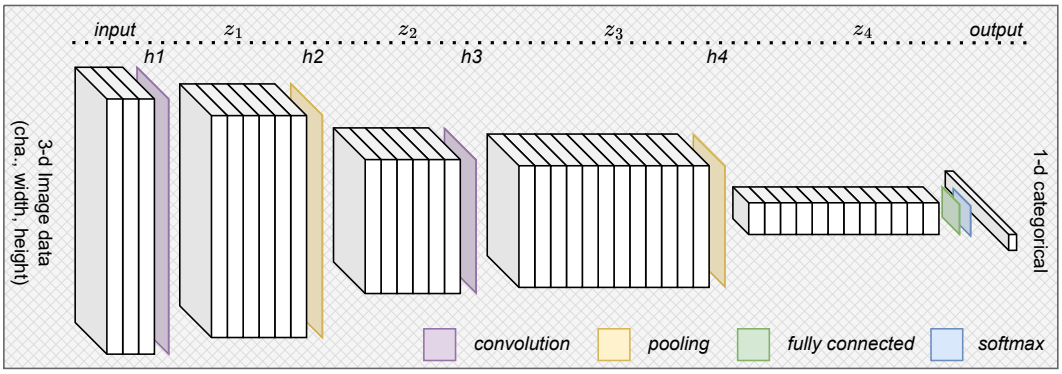


Abbildung 4.3: Typischer Aufbau eines CNN am Beispiel des *ConvNet* nach LeCun et al. [132]

Pooling

Zusätzlich zu *Convolution* kommt ein von Fukushima und Miyake beschriebenes Verfahren zur Reduktion der Auflösung der resultierenden FM zum Einsatz: *Pooling*. Hierbei werden (ebenfalls in einem parallelisierten Bearbeitungsfenster der Größe $N \times N$) Werte durch eine Funktion aus $[max, mean]$ zusammengefasst (daher auch *Subsampling*[6] genannt), sodass sich die Auflösung, also die Größe der Dimension, welche die Fläche der strukturierten Informationsebene konstituieren, um einen Faktor $\in \mathbb{N}^+$ reduziert. Die Anzahl der vorhandenen Dimensionen wird dabei nicht verändert. Auch *Pooling* definiert einen Parameter (*Stride* s), welcher das Intervall der *Pooling-Kernel*-Anwendung beeinflusst.

[6] $Subsampling = \neg Interpolation$

Dieser wird gewöhnlich so gewählt, dass sich die *Kernel* nicht überlappen und die Auflösung der resultierenden FM halbiert wird. *Pooling* kann auch als Werkzeug zur Kantenglättung oder Erhöhung des Kontrastes verwendet werden. Dazu werden *Padding*- und *Stride*-Parameter so gewählt, dass sich die Größe der FM nicht verändert. Im Vergleich zu *Convolution* verändert sich dabei die (quadratische) Seitenlänge eines *Inputs* der Größe $N \times N$ wie folgt [3]:

$$\tilde{M} = (N - M)/s \tag{4.5}$$

In „Handwritten digit recognition with a back-propagation network“ wird zum ersten Mal die Struktur des heute als *ConvNet* bekannten Bild-Klassifizierungs-Netzwerks (CNN) mit allen Modulen beschrieben. Abbildung 4.3 zeigt die nach LeCun et al. typische, auch als *LeNet* bekannte *Layer*-Folge aus *Convolution*, *Pooling*, Activation-Function und *Fully-Connected-Layer* LeCun et al.

4.3.2 Residual Skip Connection

Im Kontext von DL wurden die sog. *Skip Connections* nach Graves verwendet, um das VGP in sehr „tiefen“ NNs abzuschwächen [84]: Eine Eingabe x_t wird mit allen *Layer* h_t und diese wiederum mit der Ausgabe y_t der verwendeten RNN-Variante verknüpft. Wie auch in anderen Arbeiten aus dieser Zeit [226] sind die hier verwendeten Verbindungen parametrisiert (*Fully-Connected*) und werden im Training optimiert.

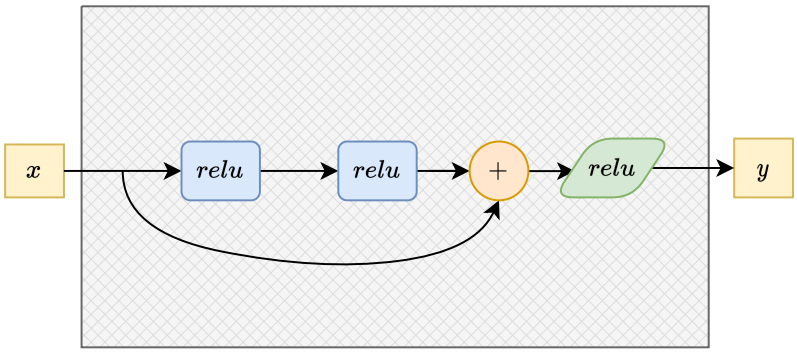


Abbildung 4.4: Funktionsweise der *Residual Skip Connection* nach He et al. [92]. Statt x, y kann hier auch eine andere Konfiguration oder Position im *Model* gegeben sein, z. B. h_i, h_{i+2}

In „Deep residual learning for image recognition“ ist hingegen eine nicht parametrisierte Variante zu finden, die *Residual Skip Connection* [92], welche namensgebend für das heute als *ResNet* bekannte *Model* ist: In einer (tiefen) CNN-Struktur werden sukzessive folgende Paare aus *Convolution-Layer*, mit einer Verbindung umgangen, sodass der *Input* dieses Pärchens auf dessen *Output*

addiert wird. Es werden also immer zwei Layer übersprungen:

$$y = F(x, W_i) + x$$

Parametrisiert werden diese Verbindungen nur dann, wenn es durch eine veränderte Form $y \neq x$ nötig ist, eine Anpassung vorzunehmen [92]:

$$y = F(x, W_i) + W_s \cdot x$$

Abbildung 4.4 zeigt diese durch *Residual Skip Connection* umschlossenen „Bausteine“ (*Building Blocks*).

4.3.3 *Data Augmentation*

Wie in Abschnitt 2.3.4.2 ausgeführt, gestaltete sich DA als ein praktikabler Ansatz zur Reduktion von Überanpassungen von DL-*Models* an kleine, unterrepräsentierte Trainingsdatensätze. Hier eingesetzt, soll DA den Nachteil und damit den *Bias* durch Mask Augsburg Speech Corpus-*Dataset (MASC)* reduzieren und so die *Performance* der untersuchten *Architekturen* besser vergleichbar machen. Da sich die Klasse der infrage kommenden NN durch die in Abschnitt 4.2 motivierte Auswahl an CNN-*Models* eingrenzen lässt, können auch auf diese *Architekturen* angepasste DA-Methoden eingesetzt werden. Gerade in den vergangenen Jahren wurden die hier verwendeten Ansätze außerordentlich gründlich vorgestellt und empirisch untersucht [5, 109, 204], daher folgt an dieser Stelle eine Übersicht zur Vollständigkeit. Diese sind beispielhaft in Abbildung 4.11g dargestellt und werden für die in Abschnitt 4.5 beschriebenen Experimente einzeln oder in Kombination verwendet. Die Symbole $\langle n, m, o \rangle = p_{hp}, 0 \leq p_{hp} \geq 1, p_{hp} \in \mathbb{R}$ und der folgenden Paragrafen bezeichnen die freien DA-*Hyperparameter* der jeweiligen Methode und sind nicht übergreifend zu verstehen. Gleiches gilt für $s_i \in S, i \in i, \ldots |S_x|, i \in N^+$, was jeweils ein zeitliches „Segment“ aller möglichen zeitlichen Segmente eines *Sample* des Datensatzes $x_j \in x, j, \ldots, |X|, j \in N^+$ bezeichnet. Frequenzsegmente werden mit $f_q \in F, q \in q, \ldots |S_y|, q \in N^+$ angesprochen, wobei $S = S_y \times S_x = x_j$ alle möglichen Datenpunkte (Matrix-Elemente) beschreibt.

Geschwindigkeits-Manipulation (Speed)

Speed beschleunigt (streckt) oder bremst (staucht) eine Audioaufnahme [127]. Hierzu wird zunächst ein zufällig ausgewählter Startpunkt s_i bestimmt: $i \sim \mathcal{U}(0, |S|)$. Anschließend wird (ebenfalls zufällig) eine Fenstergröße $w \sim \mathcal{U}(0, n)\text{mit}w < (|S| - i), w \in \mathbb{R}^+$ bestimmt. Die Geschwindigkeit der zusammenhängenden Sequenz der Datenpunkte innerhalb dieses zufällig ausgewählten Fensters $[s_i, \ldots, s_{\lfloor w \times |S| \rfloor}]$ wird anschließend durch einen Faktor $a \sim \mathcal{U}(m, o)$ angepasst. Im Gegensatz zu Implementierungen, welche zwischen Datenpunkten in Mel-Spektrogrammen interpolieren, wurde hier ein Ansatz verwendet, der

auf rohen Audiodaten arbeitet. So können Interpolationsfehler in den zumeist niedrig aufgelösten Mel-Spektrogrammen ausgeschlossen werden.

Lautstärkeanpassung (Loudness)

Eine weitere Möglichkeit, um zusätzliche Diversität in Audiodaten zu erzeugen, ist die Manipulation der Lautstärke bzgl. bestimmter Frequenzen oder ganzer Zeitabschnitte. Hierbei geht es weniger darum, die Gesamtlautstärke des *Samples* S anzupassen, da *Normalization*-Funktionen (vgl. Abschnitt 2.3.4.2) im NN-Trainingsprozess diesen Effekt wieder ausgleichen würden. Als Folge dessen bietet es sich an, eine relative Anhebung der Lautstärke mit einem Faktor $l \sim \mathcal{U}(0, n)$ vorzunehmen, der steuert, um wie viel Prozent die Intensität eines Datenpunktes in Abhängigkeit von seinem ursprünglichen Wert verstärkt wird $S + S \cdot l$. So wird erreicht, dass laute Signale deutlich lauter werden, während leise Signale weniger stark verstärkt werden. Dieser im Gegensatz zur globalen Anhebung aller Datenpunkte S stehende Ansatz ist in Abbildung 4.11f visualisiert.

Verschiebung (Shift)

Im Kontext der *Computer Vision* durch DL ist es von besonderer Bedeutung, trainierte *Models* invariant gegenüber Translation zu gestalten. Im Audiobereich bestehen ähnliche Anforderungen, so kann es sein, dass eine Beispielaufnahme zu unterschiedlichen Zeitpunkten betrachtet wird. Um diese möglichen temporalen Verschiebungen zu simulieren, sollen die Mel-Spektrogramme der Trainingsaufnahmen entweder positiv oder negativ entlang der Zeitachse verschoben werden. Der Grad der Verschiebung t (*Shift*) wird dabei zufällig in Prozent der Länge der Zeitdimension bestimmt: $t \sim \mathcal{U}(0, n)$. Die Verschieberichtung d (*Direction*) hingegen kann durch $d \sim \mathcal{B}(0.5)$ festgelegt werden (vgl. Abbildung 4.11e). Die aus dieser Verschiebung resultierenden „leeren“ Datenpunkte können entweder mit *null*-Werten oder Gaussian *Noise* gefüllt werden.

Verrauschen (Noise)

Wie bei der Arbeit mit Bilddaten im *End-to-end*-Learning, hat es sich auch im Audiobereich als nützlich herausgestellt, *Samples* durch Addition von einfacher GN zu stören. Dies ist darin begründet, dass Audioaufnahmen oft mit einem Anteil an Hintergrundrauschen durch den Einfluss in Hardware und den Umständen der Probenentstehung (z. B. Aufnahmen bei niedriger Beleuchtung (Bild) oder leisen Quellen (Ton)), entstehen. Die hier verwendete Implementierung addiert GN $g_n \sim \mathcal{N}(0, n)$, auf das Mel-Spektrogramm $(S + S \cdot g_n)$ wie in Abbildung 4.11c dargestellt.

Maskierung (SpecAugment)

Ren et al. haben gezeigt, dass durch eine einfache Okklusion (*null*-Werte (vgl.

Abschnitt 6.2.1) oder GN) sowohl auf der Zeitachse als auch über Frequenzbänder hinweg (vertikale und horizontale Balken mit zufälliger Breite) bereits eine Leistungssteigerung erzielt werden kann. Man spricht hier in einem allgemeineren Kontext auch von *Maskierung*, wobei sich im Audiobereich ein anderer Name durchgesetzt hat: *SpecAugment* [193]. In der hier verwendeten Implementierung (vgl. Abbildung 4.11d), dargestellt in Abbildung 4.11d, wird zunächst ein zufälliger Startpunkt $s_i \sim \mathcal{U}(0, |S_y|)$ bestimmt, gefolgt von der relativen Fensterbreite $w \sim \mathcal{U}(0, n)$. Die Fensterhöhe entspricht allen verfügbaren Werten auf der anderen Spektrogramm-Achse. Die so ausgewählten Datenpunkte $s_i, \ldots, s_{i+w \cdot |S_y|} \in S_y$ können dann wie zuvor beschrieben überschrieben werden (*null*-Werte oder GN). Dies wird für beide Segmentgruppen $\langle S_y, S_x \rangle$ durchgeführt.

4.4 Methoden

Dieser Abschnitt beschäftigt sich mit den Methoden und ausgewählten Verfahren praktischer Anwendung von CNN auf sequenziellen Audiodaten auf Basis von Log-Mel-Spektrogrammen. Subsequent und im Kontext des gewählten *Real-World*-Dataset, welches hier ebenfalls vorgestellt und motiviert wird, werden zunächst die verwendeten DA Methoden (Abschnitt 4.3.3) und anschließend die NN-*Architekturen* (Abschnitt 4.4.2) besprochen.

4.4.1 MASC-Dataset

Hinsichtlich der Menge an Variationen der menschlichen Stimme und ihren unterschiedlichen Modi durch Lautstärke, Geschwindigkeit und Tonfall, verspricht der Mask Augsburg Speech Corpus-*Dataset (MASC)* [163], einen Beitrag zur Verbesserung von Spracherkennung im medizinischen Umfeld zu liefern. Auch erlaubt dieser stark unterrepräsentierte Datensatz einen Blick, auf in der Praxis relevante Besonderheiten im Training von NN auf sequenziellen Audiodaten. Der MASC-Datensatz besteht aus vor-segmentierten Audioaufnahmen menschlicher Stimme, mit und ohne getragener medizinischer Maske (OP – nicht FFP, eine Sekunde ohne Überlappung) mit einer Gesamtdauer von $10h9min14sec$. Mohamed et al. gibt an, dass Aufnahmen von 32 Personen (deutsche MuttersprachlerInnen, Alter von 20 bis 41 Jahren, Durchschnittsalter 25,6 Jahre $\pm 4,5$) erfasst worden sind, welche verschiedene sprachliche Aufgaben ausführen. Unweigerlich entsprechen ca. 10 *Stunden* der hier eingesprochenen menschlichen Stimme, nur einer eher kleinen Stichprobe aus der Gesamtverteilung der insgesamt möglichen Klangbilder, was eine Evaluation der *Generalization* der im Folgenden untersuchten NN-*Models* erlaubt. Für dieses *Real-World*-Dataset ist *Binary Classification* als Aufgabe vorgeschlagen, dem hier gefolgt werden soll. Die Erforschung der Anwendbarkeit von DL-Methoden ist deswegen wichtig, weil medizinische Masken eine Notwendigkeit im medizinischen Umfeld sind,

gleichzeitig aber die Klangfarbe eines Sprechenden verändern und Gesprochenes weniger deutlich wahrgenommen werden kann oder ein Sprechender nicht mehr verwechslungsfrei identifiziert werden kann [163]. MASC unterscheidet sich von anderen Datensätzen, da eine deutliche Relevanz in der Anwendung gegeben ist, und enthaltene Stichproben (*Sample*) die mögliche Variationsmenge stark unterrepräsentieren.

4.4.2 NN-Model Architectures

In diesem Abschnitt werden vielversprechende NN-*Architekturen* ausgewählt, welche die in Abschnitt 2.4.2 präsentierten weiter ausformulieren und im Kontext von MASC vergleichen. Um einen repräsentativen Überblick über die Arbeitsweisen dieser zu erhalten, sind die *Architekturen* so gewählt, dass diese eine stetige Weiterentwicklung von typischen CNN-*Architekturen* bis zu auf Audiodaten spezialisierten CNN-Varianten darstellen.

Um Vergleichbarkeit unter den *Architekturen* zu etablieren, wird zunächst eine Basiskonfiguration (*Default*) bzgl. der Anzahl an *Neurons* und *Layer* sowie kritischer Lern-Parameter, für alle folgenden *Models* und Experimente festgelegt. Aufgrund von Besonderheiten in den jeweiligen NN-*Architekturen*, kann es zu kleineren Unterschieden in der Gesamtparameteranzahl kommen, daher werden diese in Abschnitt 4.5 mit angegeben. Die Unterschiede der CNN-*Architekturen* werden im jeweiligen Abschnitt aufgezeigt.

Basiskonfiguration (*Default*)

Als Basiskonfiguration wird hier eine CNN-*Architekturen* (vgl. Abschnitt 4.3.1) mit quadratischen 3×3 *Kernel* verwendet. Dies ist eine übliche Größe zur Erkennung und Unterscheidung von Feinheiten in texturierten Oberflächen [3, 134], dies erscheint aufgrund der feinen Struktur von Mel-Spektrogrammen logisch. Die Filtergrößen der aufeinander folgenden *Convolutions* sind dann mit $[32, 64, 128, 64]$ aufsteigend gestaltet, was der gängigen Praxis zur Differenzierung von räumlichen Zusammenhängen auf mehreren Stufen entspricht (*Feature*-Extraktion). Durch die Reduzierung der letzten *Convolution-Layer*-Filter (auf 64), wird die Anzahl der benötigten *Weights* des nachfolgenden *Fully-Connected-Classifier*, aufgrund der $|C_{l_n}| \times |C_{l_n+1}|$-Beziehung zwischen den Schichten (vgl. Abschnitt 2.3), um die Hälfte reduziert. Diese Überlegung ist dahingehend wichtig, da viele DL-*Models* mit einer großen Anzahl an erlernbaren *Weights* dem Risiko des *Overfittings* ausgesetzt sind, also Trainingsdaten auswendig lernen (vgl. Abschnitt 2.3.4.1). Dies konnte in ersten Versuchen auch hier beobachtet werden.

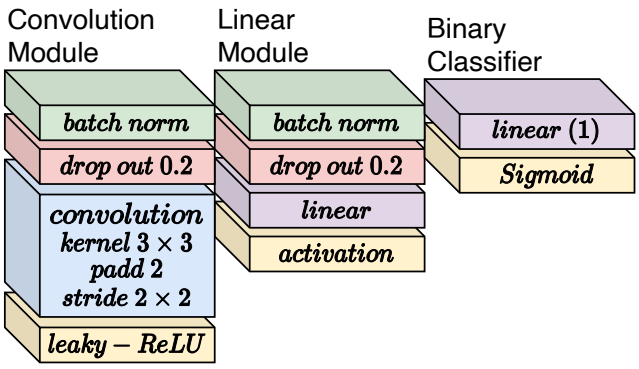


Abbildung 4.5: Beschreibung der Modulblöcke

Zur weiteren Reduktion der Auflösung zwischen den *Convolution-Layer* (und damit einer weiteren Reduktion der benötigten *Weights*) setzt diese Implementierung *Convolutions* mit $striding = 2$ (Schrittweite, vgl. Abschnitt 4.3.1) ein. Eine Alternative hierzu stellt das in Abschnitt 4.3.1 vorgestellte max-*Pooling* dar, welches lokal benachbarte *Pixel*-Werte durch die Auswahl des Maximalwertes zusammenfasst. Dieses Verfahren führt allerdings zusätzliche arithmetische Operationen im CG ein, auf welche so verzichtet werden kann. Eine technisch notwendige Größenanpassung wird durch ein 2×2 *zero-Padding* (ein Rahmen aus 0-Werten), aufrechterhalten und ist nur zur Vollständigkeit erwähnt.

Nach den subsequenten *Convolution-Layer* schließt sich nahtlos ein *Classifier*, als aus vierschichtiges *Fully-Connected*-NN, mit jeweils $[128, 256, 128, 1]$ *Neurons* an. Weiterhin soll die Möglichkeit eines *Overfitting* weiter reduziert werden, weshalb zwischen *Layer* mit optimierbaren *Weights* jeweils ein *Dropout* von 0.2 eingeführt wird (vgl. Abschnitt 2.3.4.2). Um gegen das Phänomen der *Dying Neurons* vorzugehen, wird LREL [157](vgl. Abschnitt 2.3.1) als Activation-Function verwendet. Da es hier um einen *Real-World*-Dataset handelt, der zuvor nicht standardisiert oder normalisiert wurde, wird eine *Batch-Normalization* [107], auch zwischen allen parametrisierten Schichten eingeführt. Der *Model-Output* wird schließlich durch eine Sigmoid-Funktion in den Wertebereich zwischen $[0; 1]$ transportiert (vgl. Abschnitt 2.3.1). Zum Training kommt ein *Binary Cross Entropy (BCE)-Loss* zum Einsatz, welches die binäre Kreuzentropie zwischen dem *Target* $y \in \mathbb{R}$ und dem *Model-Output* $\hat{y} \in \mathbb{R}$ misst: $\mathcal{L} = \log(1 + \exp(-y \cdot \hat{y}))$

ConvClassifier (CC)

Dieses *Model* bildet die *Architektur* von *Default* als *Baseline* ab und kombiniert (Abbildung 4.6) grundlegende Strategien aus dem Bereich der CNN basierten Bildklassifizierung. Vier CNN-*Layer* reduzieren die (räumliche) Auflösung des Bildes, während diese lernen, immer mehr Filterebenen zu nutzen. Mehrere *Fully-Connected-Layer* lernen anschließend eine nicht lineare Separierung der so extrahierten Hauptmerkmale (*Feature*) innerhalb der *Samples*.

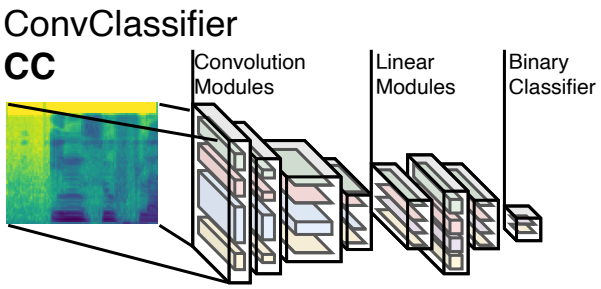


Abbildung 4.6: *ConvClassifier (CC)*

SubSpectralNet (SSN)

Im Gegensatz zur gängigen CNN Implementierung soll mit der folgenden *Architektur* untersucht werden, wie sich ein auf Mel-Spektrogramme spezialisiertes CNN verhält. In Anlehnung an Phaye, Benetos und Wang wird ein *Model* implementiert, welches vier voneinander getrennte, aber parallele CNN auf der Grundlage der *Baseline* (*default*) zusammenführt [185]. Der *Input* jedes dieser Sub-Netzwerke umfasst eine jeweils andere, nicht überlappende Menge an Mel-Bändern $= 64/4$ (vgl. Abbildung 4.7). Aneinandergereiht (*Konkatenation*), werden diese Band-weisen Vorhersagen (Sigmoid-aktiviert, nicht *Logit*) an den *Classifier* nach *Default*-Vorbild weitergegeben (vier *Fully-Connected Layer* mit $[128, 256, 128, 1]$ *Neurons*). Der individuelle *Loss* $\mathcal{L}_n, 5 > n \in \mathbb{N}^+$ jedes dieser Sigmoid-aktivierten *Sub-Classifier*, also die bandweisen Vorhersagen, wird einzeln durch BCE berechnet. Der Gesamt-*Model-Loss*, ergibt sich nach dem Vorbild von Phaye et al., aus der Kombination und der gleichzeitigen BP: $\mathcal{L}_{SSN} = 1/n \sum_{n=1}^{4} |\mathcal{L}$

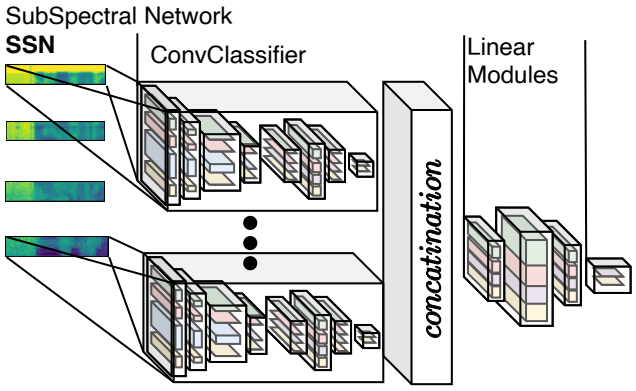


Abbildung 4.7: SubSpectralClassifier (SSN)

SubSpectralClassifier (SSC)

Um den *Performance*-Gewinn durch die zusätzlichen und teuren Ableitungen der Exponentialfunktionen in den 4 *Losses* weiter zu untersuchen, wird ein weiteres Band-weises CNN implementiert. *Default* wird (vgl. SSN) bandweise

angewendet, während das globale Klassifizierungs-Subnetz den aneinandergereihten *Outputs* der *Convolution*-Operationen aufnimmt, wie in Abbildung 4.8 dargestellt. Im Gegensatz zu SSN muss somit der globale *Classifier* (anstatt der lokalen) lernen, welche Merkmale für eine finale Entscheidung zu verwenden sind, während die Filterebenen der *Sub*-CNNs nicht geteilt werden und daher für eigene Frequenzbereiche zuständig sind. Die Menge der *Neurons* im *Classifier* folgt *Default*. Die letzte NN-Ebene ist Sigmoid-aktiviert während das BCE-*Loss* den *Model*-Fehler berechnet.

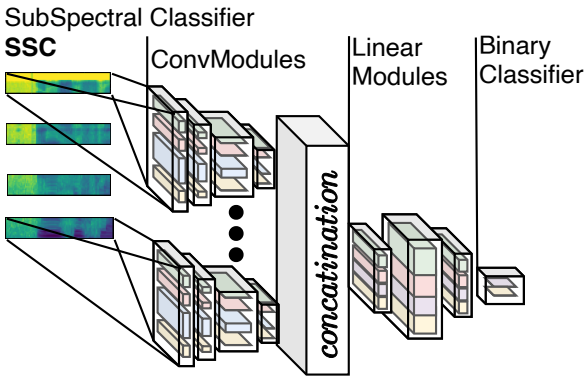


Abbildung 4.8: *SubSpectralClassifier (SSC)*

ResidualConvClassifier (RCC)

Nach dem Vorbild von Ford et al., Ren et al. soll eine *Residual Skip Connection*-basierte Variante eines CNNs zur Einordnung der übrigen Ergebnisse herangezogen werden. Die hier verwendete Implementierung überspannt mit *Residual Skip Connections* immer zwei gleichartig gestaltete *Convolution*-Blöcke[7] (Abbildung 4.9). Der *Fully-Connected-Classifier* folgt auch hier *Default*.

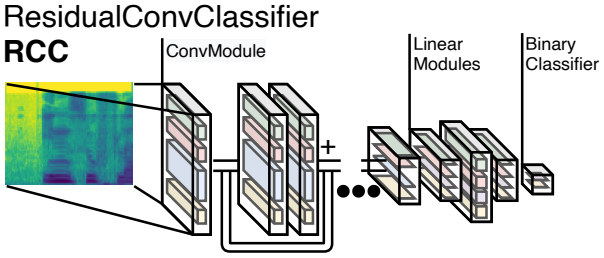


Abbildung 4.9: ResidualConvClassifier (RCC)

Ensemble (E)

Abschließend soll, inspiriert von vielen bestehenden Beispielen (z. B. [95, 175]), getestet werden, ob die hier trainierten *Models* sich auf unterschiedliche *Features* in den *Samples* eingestellt haben. Hierzu wird über einen *mean Vote*,

[7] „Block“ meint hier eine Kombination aus *Normalization*, *Convolution*, *Dropout* & Activation-Function.

also eine Mehrheitsabstimmung die Gesamtaussage der vier *Models*, wie in Abbildung 4.10 dargestellt ermittelt. Die Zusammenstellung unterschiedlicher *Models* ist eine gängige Praxis in *Real-World*-Application, zur Steigerung der primären *Performance*.

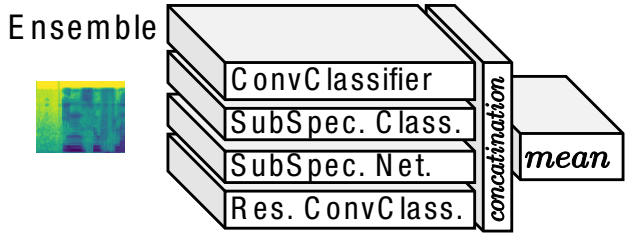
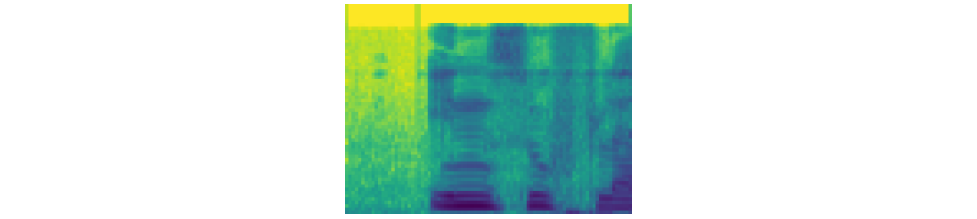


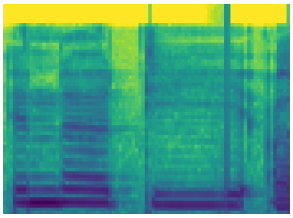
Abbildung 4.10: Ensemble

4.4.2.1 *Hyperparameter*-Tuning

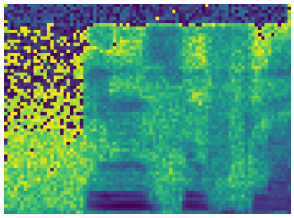
Die *Performance* der hier betrachteten Ansätze zur praktischen Anwendung von CNNs im Kontext sequenzieller Audiodaten, ist offensichtlich massiv und wie motiviert, mit Absicht durch den gewählten Datensatz beeinflusst. Um diesen induzierten *Bias* zu minimieren, empfiehlt es sich sowohl eine möglichst breite sowie freie und damit repräsentative HP-Optimierung durch eine Rastersuche (vgl. Abschnitt 2.3.4.2) vorzunehmen. Dies bezieht nicht nur die HP-Auswahl für *Default*, sondern auch die DA- und Trainingsparameter mit ein. Da sich die *Architektur* der betrachteten *Models* nur immer hinsichtlich bestimmter Aspekte unterscheidet, erscheint die Parametrisierung von SSN, SSC & *ResidualConvClassifier (RCC)* mit den für *ConvClassifier (CC)* gefundenen HPs eine gute Vergleichsgrundlage zu sein.



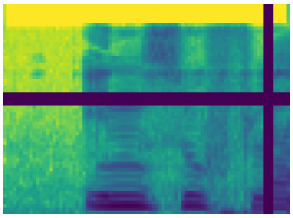
(a) Raw / Original



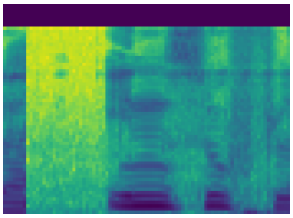
(b) Speed Variation



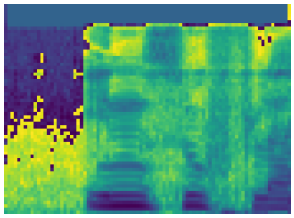
(c) Gaussian Noise



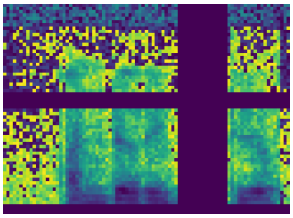
(d) Masking (SpecAug)



(e) Temporal Shift



(f) Loudness



(g) Kombination

Abbildung 4.11: Einige Beispiele für, durch DA-Methode (gestörte) Mel-Spektrogramme (Abschnitt 4.3.3), wie sie für das Training der *Models* zum Einsatz kamen.

4.5 Experimenteller Aufbau

Das hier beschriebene Experiment wird unter kontrollierten Bedingungen durchgeführt, d. h. alle *Models* sind mit denselben Trainingsdaten und HPs trainiert. Weiterhin werden nur sog. Fixed-*Seed*-Zufallsoperationen verwendet[8].

Die Details der einzelnen Schritte werden in den nächsten Abschnitten beschrieben, wobei das Training der *Models End-to-end* erfolgt und wie folgt angewendet wird:

[8]Dies umfasst alle verwendeten *Python*-Bibliotheken: *Random*, *NumPy*, *PyTorch*.

Experiment 4.5.1: NN Training - MASC

- *Speed* & Log-Mel-Spektrogramm
- Für jeden *Seed*:
  - Vorverarbeitung
  - Training Start für 200 Epochen
  - Für jeden *Batch*:
    - *Data Augmentation*
    - *Model*-Anwendung (*Inference*)
    - *Backpropagation*
- Evaluation & Vergleich

DA-Parameter	Value
$Speed - Factor$	$= 0.7$
$Speed - Size$	$= 0.3$
$Mask$	$= 0.2$
$Noise$	$= 0.4$
$Shift$	$= 0.3$
$Loudness$	$= 0.4$

4.5.1 Vorverarbeitung

Wie in Abschnitt 4.1 motiviert und in Abschnitt 4.4 vorgeschlagen, werden die rohen Audiodaten in Bezug auf *Speed* verändert. Aufgrund der deutlich gesteigerten Ressourcenanforderung der Anwendung von DA-Methoden auf rohen Audiodaten, muss dieser Schritt im Gegensatz zu den sonstigen *Online*-DA-Verfahren als *Preprocessing*-Schritt implementiert werden, wobei immer mindestens ein rohes (nicht transformiertes) Exemplar jedes *Sample* im Trainingsdatensatz beibehalten wird. Somit kann die für das *Model*-Training und die weiteren DA-Schritte zur Verfügung stehende Menge an Trainingsdaten vervierfacht werden. In ersten Versuchen hat sich gezeigt, dass sich durch eine weitere synthetische Vergrößerung des Trainingsdatensatzes keine weitere *Performance*-Steigerung einstellt, während aber die Trainingszeit weiter linear wächst.

Die Log-Mel-Transformation[9] wird mit einer Fensterbreite von n-fft=512 bei einer Sprungweite von hop-length=256 für n-mels=64 Mel-Frequenzbänder, was eine Mel-Spektrogramm-Auflösung von $64 \cdot 87$ Pixeln pro *Samples* ergibt, ausgeführt. Diese Parameter haben in ersten Versuchen die besten Ergebnisse geliefert. Um Unregelmäßigkeiten im Datensatz auszugleichen, werden alle Mel-Spektrogramme lokal, also innerhalb jedes *Sample*, normalisiert ($\mu = 0, \sigma = 1$, vgl. Abschnitt 2.3.4.2).

4.5.2 Data Augmentation und Model-Training

Die hier verwendeten DA-Methoden[10], werden wie in Abschnitt 4.4 besprochen umgesetzt. Die DA-HPs werden auf Basis einer breiten *Grid Search* bestimmt,

[9] *librosa*: `https://librosa.github.io/-10.5281/zenodo.3606573`
[10] *nlpaug*`https://nlpaug.readthedocs.io/en/latest/` [156]

wobei die jeweiligen HP-Suchräume auf Basis von Vorerfahrungen aus verschiedenen Experimenten bestimmt wurden (*Educated Guess*). Das Ziel dabei ist es, Kombinationen zu finden, welche möglichst starke Störungen in die Trainingsdaten einbringen, also die Entropie in diesem kleinen *Real-World*-Dataset synthetisch zu erhöhen, ohne ein erfolgreiches Training zu unterbinden (vgl. Abschnitt 2.3.4.2). Hierbei mussten auch Wechselwirkungen zwischen den DA-Methoden mit berücksichtigt werden. Passende Parameter werden wie in Experiment 4.5 spezifiziert verwendet.

Zugunsten der Vergleichbarkeit und Reproduzierbarkeit werden diese Parameter bei allen *Model-Seed*-Kombinationen im Training gleichgehalten. DA-Operationen werden *Online*, also bei der randomisierten Auswahl von *Samples* eines *Batch* im Training, durchgeführt. Bei der Ausführung der Validierung kommt es zu keiner Störung durch DA-Methoden.

4.6 Durchführung und Ergebnisse

Bestehende, externe *Baselines* dieses Datensatzes [163, 208] verwenden als vergleichende Metrik die *Unweighted Average Recall (UAR)* (auch als *Balanced Error Rate (BER)* [198] bekannt). Diese eignet sich besonders gut für die Bewertung von *Classifier*, wenn die Menge der *Samples* der verfügbaren Klassen im Ungleichgewicht ist [159]. Gleichung 4.6 gibt zum besseren Verständnis die UAR im Kontext von MASC an [163], wobei $N_{masc} = |x|_{masc}$ die Menge an *Samples* (je Datensatz) angibt, die als „mask“ gelabelt sind und umgekehrt, mit TP_{mask} der „True-Positive“ in der Evaluation.

$$\begin{aligned} \textit{Binary}\ \text{UAR} &= \frac{1}{2} \cdot \left(\frac{TP}{FN + TP} + \frac{TN}{TN + FP} \right) \\ &= \frac{1}{2} \cdot \left(\frac{TP_{mask}}{N_{mask}} + \frac{TP_{clear}}{N_{clear}} \right) \end{aligned} \tag{4.6}$$

In Tabelle 4.1 ist die *Performance* der in Abschnitt 4.4.2 eingeführten NN-*Architekturen* unter Einfluss der ausgewählten DA-Methoden aus vgl. Abschnitt 4.3.3. Als Vergleichswert wird auch die *Performance* auf einem nicht durch DA manipulierten Datensatz dargestellt („raw“). Die jeweils erreichte UAR-*Performance* (vgl. Abschnitt 4.5) über alle *Models* sowie die dazugehörige Varianz über fünf *Seeds* in Prozent angegeben.

Model-Performance auf „rohen“ Trainingsdaten

Erstaunlicherweise kann schon ein einfaches, „konventionelles“, auf den normalisierten Mel-Spektrogrammen trainiertes CNN (vgl. Abschnitt 4.4.2; $kernel = 3 \times 3$, nicht überlappend, $stride = 2$) bereits eine *Performance* (CC-„UAR“

Tabelle 4.1: Vergleich der Resultate: *Model-Performance* Leistungen unter dem Einfluss verschiedener DA-Methoden.

	CC	SSN	SSC	RCC	**Max.**
Raw	64.71 ± 0.30	63.54 ± 0.26	65.82 ± 0.39	64.53 ± 0.53	65.82
Speed	64.71 ± 0.40	63.54 ± 0.91	66.40 ± 0.34	64.07 ± 0.77	66.40
Noise	63.62 ± 0.22	61.35 ± 0.67	64.48 ± 0.46	63.38 ± 0.41	64.48
Loudn.	64.31 ± 0.43	62.87 ± 0.64	65.65 ± 0.50	64.14 ± 0.37	65.65
Shift	65.40 ± 0.55	66.31 ± 0.70	$\mathbf{68.20} \pm 0.44$	64.38 ± 0.78	**68.20**
Mask.	64.27 ± 0.34	62.27 ± 0.33	65.10 ± 0.32	64.30 ± 0.45	65.10
Fuse	65.03 ± 0.41	63.49 ± 0.79	66.35 ± 0.34	64.12 ± 0.42	66.35
Max.	65.40	66.31	**68.20**	64.38	
Model Param.	633,219	1,801,621	1,533,321	1,095,171	

=64.71) erreichen, die mit der dem Datensatz zugeordneten UAR-*Baseline*, einer Kombination aus *Auto-Encoder (AE)* und SVM (*max.* 64.4), konkurriert. Interessanterweise verhält sich der *ConvClassifier (CC)* damit stabiler (*variance* = 0.3) als der RCC, (64.53 ± 0.53) bei einer (leicht) besseren *Performance*. RCC sollte durch die zusätzlichen *Residual Skip Connections* und mehr *Weights* bessere Voraussetzungen haben, nutzt dies aber nicht. §overfit wurde nicht beobachtet.

Auch überraschend ist der deutliche Unterschied zwischen SSN [185] (63.54 ± 0.26) und dem hier als Gegenthese konstruierten SSC (65.82 ± 0.39). Zwar fällt die Varianz für SSN etwas geringer aus (-0.13), aber es ist auch nicht schwer, mit Konsequenz schlecht zu sein. Überraschenderweise übertrifft damit der SSC, trainiert auf einem unterrepräsentierten Datensatz, ohne zusätzliche DA-Einflüsse, die anderen CNN-basierten *Models*. Dies spricht dafür, dass das Lernen der *Features* pro Mel-Band einen deutlichen Vorteil gegenüber einem Ansatz mit vollständig geteilten *Weights* aufweist. Es sei abermals darauf hingewiesen, dass die initiale HP-Suche lediglich für CC ausgeführt wurde. Unterschiedliche Frequenzen scheinen in der Tat unterschiedliche Merkmale zu besitzen, was die Angaben von Phaye, Benetos und Wang bestätigt. Ein über mehrere *Losses* mit zusätzlichen rechenintensiven Sigmoid-Funktionen konstruiertes Lernziel scheint darüber hinaus, zumindest in Bezug auf dieses *Real-World*-Dataset, keinen Mehrwert zu schaffen.

Model-Performance durch *Data Augmentation*

Betrachtet man nun die *Performance* der Modelle, welche auf Daten unter dem Einfluss von DA trainiert wurden, entsteht ein ähnliches Bild. SSC tritt immer, mal mit mehr, mal mit weniger Abstand, als in diesem Fall (MASC)

am besten geeignete *Architektur* hervor. Auch die anderen Modelle scheinen sich in gleichem Maße anzupassen, sodass sich die Unterschiede zueinander in einem ähnlichen Maße verhalten. Eine Ausnahme stellt CC unter dem Einfluss von „Temporal Shift“ (vgl. Abschnitt 4.3.3) dar. Zwar ist diese DA-Methodeauch für dieses *Model* förderlich, allerdings nicht im gleichen Maße wie bei den anderen. Dies ist in Tabelle 4.2 dargestellt. Hier zeigt sich, dass die DA-Methodenfür sich genommen zumeist keinen positiven Einfluss auf das *Model*-Training haben. Lediglich „Temporal Shift“ tritt als für diesen

Tabelle 4.2: Vergleich der Resultate: *Model-Performance* Leistungen unter dem Einfluss verschiedener DA-Methoden.

	CC	SSN	SSC	RCC
Raw	64.71	63.54	65.82	64.53
Speed	0	-0.01	+0.58	-0.46
Noise	-1.09	-2.19	-1.34	-1.15
Loudn.	-0.4	-0.67	-0.17	-0.39
Shift	**+0.69**	**+2.77**	**+2.38**	**-0.15**
Mask.	-0.44	-1.27	-0.72	-0.23
Fuse	+0.32	-0.05	+0.53	-0.41

Datensatz beste und zuverlässigste DA-Strategie hervor. Diese Aussage bezieht sich auf den Vergleich mit den anderen (Abschnitt 4.3.3) implementierten DA-Methodenund sollte im Zusammenhang mit anderen Trainingsdaten auch zu anderen Ergebnissen führen, wie sich in Kapitel 5 zeigen wird.

Insgesamt kommt man zu der Beobachtung, dass alle *Models* robust gegenüber den in die Trainingsdaten eingebrachten Störungen sind, denn alle *Models* verhalten sich innerhalb ihres Leistungsbereichs (mit einer Abweichung von bis zu 5%). Eine Kombination („fuse“) der DA-Methodenführt im Vergleich zum *Model*-Training mit „Shift“ zu einer zu starken Störung der Trainingsdaten, sodass nur eine geringe Veränderung der *Performance* gemessen werden kann. CC und SSC können aber selbst hier einen kleinen positiven Einfluss mitnehmen (max. +0.53%).

Der *ResidualConvClassifier (RCC)* scheint als *Model* selbst sehr stabil zu sein, da die durch DA eingebrachte Störung die Leistung des Modells meist nur in geringem Maße beeinflusst (Ausnahme „Noise“: -1.15%). Interessanterweise kann der RCC keinen positiven Effekt aus den DA-Methodenziehen.

Zur Vollständigkeit wurden auch typische Modell-Ensembles (dargestellt in Abbildung 4.10) untersucht. Dabei zeigte sich allerdings keine Leistungssteigerung. Ein „Bagging“ bzw. „Ensemble Bootstrapping“ [95], also ein Training unterschiedlicher *Models*, auf sich überschneidenden Untermengen von Trainingsdaten ($NN_E(x) = 1/I * I \sum_{0 \to i}^{I} (NN_i(x|\Theta_{i|s}); s \in S)$), könnte hier, wie

Tabelle 4.3: Vergleich zwischen der mit den Datensatz-*Baselines* sowie der in Abschnitt 4.4.2 eingeführten *Models*. Die *Model-Performance* wird anhand der UAR gemessen.

Baseline Models	**UAR**	
	Dev	**Test**
DeepSpectrum + SVM (ResNet50)	63.4	70.8
S2SAE + SVM (Fused)	64.4	66.6
ComParE functionals + SVM (C=10^{-3})	62.3	67.8
ComParE BoAW + SVM (N=2k)	64.2	67.7
Fusion	-	71.8
Our Models		
ConvClassifier + Shift	65.4	-
SpectralNet Network + Shift	66.3	-
SpectralNet Classifier + Shift	68.2	71.5
Residual Conv. Classifier + Shift	64.5	-
Ensemble over best Models; Mean Vote	68.0	-

„Fusion“ und andere *Model*-Kombinationen (vgl. Tabelle 4.2), von Vorteil sein. Zum Zeitpunkt der Ausführung dieser Experimente konnte dies nicht überprüft werden.

4.7 Diskussion und Zusammenfassung

In diesem Kapitel wurden verschiedene CNN-basierte NN-*Models* (Abschnitt 4.4.2) zur Verarbeitung und Klassifizierung von Audiodaten eines *Real-World*-Dataset in Form von Mel-Spektrogrammen vorgestellt. Dabei lag der Fokus auf der Vorstellung und Gegenüberstellung der unterschiedlichen *Architekturen*, sowie im Kontext passende, ausgewählte DA-Methoden. Ferner wurden Ansätze und deren Kombination im Umgang mit *Real-World*-Dataset eingeführt und an entsprechender Stelle verwendet.

Im Training auf einem schwierigen *Real-World*-Dataset (MASC) an der Aufgabe der binären Klassifikation zur Erkennung chirurgischer Masken hat sich gezeigt, wie sich besagte *Models* in einer praktischen Anwendung verhalten. So sollte ein möglichst reales Setting geschaffen werden, während die Einflüsse der DA-Methoden einen Einblick in die Generalisierungsfähigkeit und *Robustness* der *Models* zulassen sollten.

Als *Baseline* diente ein einfaches CNN (mit *Stride*) der sog. CC. Diese Basis-*Architektur* wurde in drei speziellen Varianten betrachtet: **SSN** ahmte

die Eigenschaften des von Phaye, Benetos und Wang vorgestellten Ansatzes nach, um kleinere Sub-Netze (mit einem *Loss*) auf unterschiedlichen Frequenzbereichen zu etablieren. **SSC** folgte einem ähnlichen Ansatz, aber sparte die zusätzlichen Sub-*Classifier* und die zusätzlichen *Sigmoid*-basierten *Losses* aus. **RCC** legt *Residual Skip Connections* um paarweise *Convolution*-Blöcke, um ein tieferes, aber (in der Theorie) stabiler trainierbares *Model* zu erhalten.

Es konnte insgesamt festgestellt werden, dass „einfache“ CNN-*Models* unter dem Einfluss von *Temporal Shift* (Abschnitt 4.3.3) eine wettbewerbsfähige Einstiegsstrategie darstellen.

Insgesamt erschien der RCC zwar nicht als leistungsstärkstes, jedoch als insgesamt robustestes *Architektur* hervorzutreten. Der Einfluss der verschiedenen DA-Methoden konnte als vergleichsweise gering beobachtet werden.

Auch scheint die Strategie der Kombination von Sub-*Model* eigener Frequenzbereiche erfolgreich vielversprechend für praktische Anwendungen zu sein. So erzielte der in diesem Kapitel (Vorveröffentlichung in „Surgical Mask Detection with Convolutional Neural Networks and Data Augmentations on Spectrograms“[101]) vorgestellte *SubSpectralClassifier (SSC)* bessere Ergebnisse bei einer vergleichbaren Varianz (CC), wobei diese recht simple *Architektur* mit geringeren Berechnungskosten (*compute*) arbeitet, als das durch mehrere Exponentialfunktionen aufgeladene Vorbild (SSN [185]).

Es sollte allerdings beachtet werden, dass die hier geäußerten Beobachtungen jeweils nur im Kontext des verwendeten Datensatzes valide sind. Dies gilt selbstverständlich jeweils auch für andere wissenschaftliche Veröffentlichungen, die diesen Aspekt gerne auslassen. So sollten rein statisch gesehen, deutlich breitere Studien aufgesetzt werden, um von *Performance*-Steigerungen in wissenschaftlichen Arbeiten berichten zu können. Leider ist dies teuer, sodass sich nicht jedes Institut einen Beitrag wie z. B. ViT [50] leisten kann. Dennoch können auch viele kleinere Studien das Forschungsfeld im Laufe der Zeit weiterbringen.

5 Attention-basierte Sequenzverarbeitung

Nachdem in Kapitel 4 die generelle *Performance* von NN-*Models* der Bilderkennung sowie auf Audio-Daten spezialisierte *Architekturen* im Kontext von *Real-World*-Dataset eingeführt wurden, soll nun ein *Model* vorgestellt und im Kontext der Audio-Klassifizierung eingeführt werden, das Konzepte aus dem Bereich des *Natural Language Processing (NLP)* mit Ansätzen der Bildverarbeitung verbindet.

In diesem Kapitel wird erstmals der sog. ViT, welcher, wie andere *Transformer* auch, den *Attention*-Mechanismus einsetzt, in den Anwendungskontext der Verarbeitung sequenzieller Audiodaten gebracht. Ferner wird gezeigt, wie dieser Ansatz in Kombination mit Mel-basierten DA-Methoden und einer Anpassung der Stichprobenentnahme im Training (*Weighted Sampling (WS)*), eine adäquate *Performance* auf *Real-World*-Datensätzen erzielt. Außerdem soll der Einfluss verschiedener Parameterkonfigurationen auf das Trainingsverhalten beobachtet werden.

Contribution 5.0.1: ViT für Mel-Spektrogramme

Beiträge zur Wissenschaft in diesem Kapitel umfassen die Einführung und Evaluation eines *Attention*-basierten *Classifiers* nach dem Vorbild des *Vision Transformer* zur Klassifizierung von sequenziellen Audiodaten auf *Real-World*-Datensätzen. Vorveröffentlichung durch Illium et al. in „Visual transformers for primates classification and covid detection“ [102]

5.1 Einführung

Wie das vorangegangene Kapitel sowie Abschnitt 2.4.1 gezeigt haben, ist es möglich, durch Mel-Spektrogramme repräsentierte Audiosignale erfolgreich zur Verarbeitung mit DL-*Architektur* zu verwenden. Und nicht nur das, auch hat sich gezeigt, dass dieser Forschungsbereich viele Einflüsse aus dem Forschungsfeld der *Computer Vision* erfährt. Bemerkenswerterweise liefern einige dieser *Architekturen* wie z. B. das CNN (vgl. Abschnitt 4.3.1) in dieser Domäne eine erste wettbewerbsfähige Grundlage für viele Problemstellungen *out-of-the-box* [55]. Dies setzt natürlich voraus, dass eine geeignete Kombination von HP

gefunden werden kann, wie es auch zuvor schon in Kapitel 4 demonstriert wurde. In den Fällen, in denen neben rohem Audiomaterial zusätzlich geschriebener Text bereitgestellt wird, übernehmen oft Algorithmen bzw. *Models* des *Natural Language Processing (NLP)* zusätzliche Aufgaben. Dabei entstehende Repräsentationen (*Embedding*, *Hidden Vector*), welche kombiniert auf unterschiedliche Arten kombiniert werden, um anschließend z. B. von einem *Classifier* ausgewertet zu werden. Man spricht hier auch von *Multi-Modal-Models*[1]. Jüngst erobern sog. *Transformer* [236] (vgl. Abschnitt 5.3.2) durch die geschickte Verwendung des sog. *Attention*-Mechanismus das Forschungsfeld des NLPs. Durch ihre parallelisierte Arbeitsweise und den Verzicht auf rekurrente Verbindungen können deutlich größere *Model* stabil trainiert werden und längere Beziehungen in den sequenziellen Datenformaten aufgegriffen und verarbeitet werden.

Indem Dosovitskiy et al. Bilder als *Embedding*-Sequenzen interpretiert, kombiniert der ViT [50] die Bereiche der Bild- und Sprachverarbeitung auf geschickte Art und Weise. Im Allgemeinen greift der ViT auf eine Arbeitsweise zurück, welche der einer dynamischen *Convolution* ähnelt [87], daher erschließt es sich, als naheliegend, auch andere Domänen auf die Anwendbarkeit dieses *Models* zu untersuchen.

In diesem Kapitel soll aufgrund dieser Schlussfolgerung untersucht werden, ob und wie der Ansatz des ViT auch auf Mel-Spektrogramme (vgl. Abschnitt 2.4.3), am Beispiel der Audio-Klassifikation, angewendet werden kann. Dabei werden verschiedene Varianten ins Auge gefasst und evaluiert und HP-Konfigurationen untersucht.

5.2 Verwandte Arbeiten

Da bereits ein enormer Teil an konzeptionell ähnlichen, verwandten Arbeiten in Abschnitt 4.2 behandelt wurde, konzentriert sich dieses Unterkapitel auf den Forschungsbereich der auf *Attention* basierenden Algorithmen und deren Einsatzgebiete.

Attention [12, 155], aus dem Forschungsfeld des NLP stammend, wurde auch schon zuvor im Kontext der Bild- und Audioverarbeitung eingesetzt. Dabei wurde zumeist auf eine Kombination der beiden Verfahren gesetzt [14, 35, 184, 248]. Der von Dosovitskiy et al. eingeführte *Vision Transformer (ViT)* weicht davon ab, indem lediglich *Attention*- und (*Fully-Connected*) *Multi-Layer Perceptron (MLP)*-Module Verwendung finden. Als großer Unterschied zu ViT ist die Wahl der *Patch*-Form sowie der Kontext der Audiodaten zu nennen, der zuvor so noch keine Betrachtung fand. Zusätzlich könnte die horizontale Abtastung dazu genutzt werden, um Audio-Segmente variabler Länge zu verarbeiten. Tao, Shanxu und Changsong scheint eine dem ViT ähnelnde *Architektur* zu implementieren, verwendet die *Attention*-Module allerdings nicht in einer

[1] Beispiel: „PaLi“ - `https://sites.research.google/pali/?ref=blog.roboflow.com`

Verarbeitungslinie im *Transformer*-Stil, sondern baut einen zusammengesetzten *Attention*-Vektor auf, welcher jeweils auf die Zwischenschritte der Verarbeitung blickt [229]. Han et al. stellt den Vergleich zwischen *Attention* und dynamischen *Convolution* an, eine These, die (wie sich später zeigt) auch von *Mixer* [232] in Bezug auf MLPs geteilt wird. Der *Perceiver* kann durch ein iteratives Bottleneck, das stark an den HV der LSTM-*Architektur* erinnert, die die quadratische Abhängigkeit von $K\&Q$ umgehen [114]. Andere Ansätze setzen auf Attention, um audiovisuellen *Input* zu verarbeiten (z. B. [171]). Parallel zu der Vorveröffentlichung wurde ein direkt vergleichbarer Ansatz entwickelt, auf den zu einem späteren Zeitpunkt genauer eingegangen wird [78].

5.3 Methodische Grundlagen

In diesem Abschnitt sollen die für dieses Kapitel benötigten Grundlagen *Attention*-basierter NN-*Architekturen* eingeführt werden. Beginnend mit dem *Attention*-Mechanismus, wird über die *Multi-Head-Attention* im Anschluss die allgemeine Funktionsweise des *Transformers* eingeführt.

5.3.1 Attention-Mechanismus

Als *Attention* wird ein Mechanismus bezeichnet, der nach Bahdanau, Cho und Bengio NN-*Models* „entlastet“, indem in einem NN-*Layer* „entschieden“ wird, welchen Teilen eines Vektors Aufmerksamkeit geschenkt werden sollte [12]. Die ursprüngliche Idee entstand aus der Motivation der Unterstützung der im Kontext des NLP eingesetzten *Models*, welche schwer und aufwendig zu trainieren waren. Eine parametrisierte (v, W & U) Bewertungsfunktion wird definiert, welche eine Aussage über den wahrscheinlichen Stellenwert jedes intermediären HV (h_j) eines vorgeschalteten RNN zulässt (vgl. Gleichung 5.1). Wie im DL üblich sind die *Weights* dieser Bewertungsfunktion durch ein Gradienten-basiertes Verfahren und BP optimierbar.

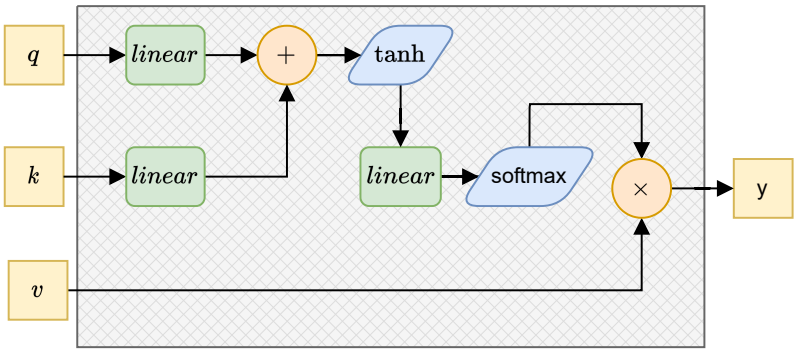


Abbildung 5.1: Funktionsweise der *Attention* nach Bahdanau, Cho und Bengio [12]

Historisch gewachsen, ist der *Attention*-Mechanismus mit drei Eingabevektoren (*Inputs*) definiert: *Query* Q, *Key* K und *Value* V. Je nach Einsatzort, innerhalb einer NN-*Architektur*, wird diesen Vektoren eine unterschiedliche (semantische) Bedeutung zugewiesen, aber immer die Bewertungsfunktion aus Q & K bedient. Durch die Verrechnung mit V entsteht ein kontextualisierter Vektor c_i, welcher für eine Aufgabe wie z. B. eine Übersetzung verwendet werden kann. Das Konzept dieser Bewertungsfunktion (a_{ij}), sowie der Kontextualisierung von V, ist in Abbildung 5.1 umrissen, wohingegen die Einbettung in die ursprüngliche Formulierung, als parallelisiertes Modul eines Sprach-*Models* in Gleichung 5.1 zu finden ist. Man beachte die Abhängigkeit der Formulierung von T sowie die doppelte Verwendung von h_j als V und K.

$$
\begin{aligned}
e_{ij} &= a(s_{i-1}, h_j) \\
&= v_a^T \tanh(W_a s_{i-1} + U_a h_j), \\
a_{ij} &= \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \quad (softmax) \\
c_i &= \sum_{j=1}^{T_x} a_{ij} h_j
\end{aligned}
\tag{5.1}
$$

wobei $W_a \in \mathbb{R}^{n \times n}, U_a \in \mathbb{R}^{n \times 2n}$ und $v_a \in \mathbb{R}^n$. [12]

Attention wurde zunächst etabliert, um den *Decoder* im Kontext einer rekurrenten Encoder-Decoder-*Architektur* zu unterstützen und in diesem Kontext noch als *Alignment Model* mit einer *Score Function* bezeichnet. Heute hingegen, ist diese Methode auch unter *Bahdanau-* oder *Additive-Attention* bekannt.

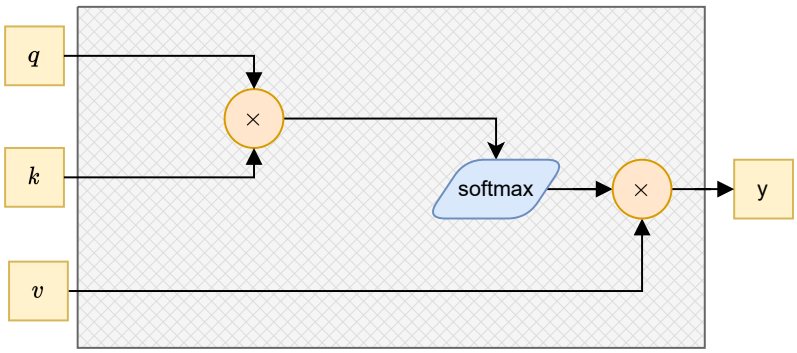


Abbildung 5.2: Funktionsweise der multiplikativen *Attention* nach Luong, Pham und Manning [155]

Luong, Pham und Manning entwickelten auf dieser Grundlage die als *Multiplikative-* oder *Luong-Attention* bekannte Variante (vgl. Gleichung 5.3, *concat*),

bei der die Bahdanau-*Attention* vereinfacht und Rechenpfade verkürzt wurden [155]. Dabei unterscheidet sich die *Luong-Attention* in der Art und Weise, wie zusätzliche erlernbare *Weights* eingesetzt und der HV der umgebenden RNN Verwendung findet. Zusätzlich stellte er auch die, später als *dot-* oder *Intra-Attention* bekannte Variante, vor. Welche auf eine elementweise Direktverknüpfung des aktuellen HV h_s mit dem zuletzt erzeugten Ausgabevektor h_t setzt. Interessant ist, dass *Attention* im Gegensatz zu RNN-*Models* eine direkte Verbindung zwischen Ein- und Ausgabe (*Layer* oder *Model*) aufweist, welche direkt mit den *Skip Connections* nach Graves verglichen werden können.

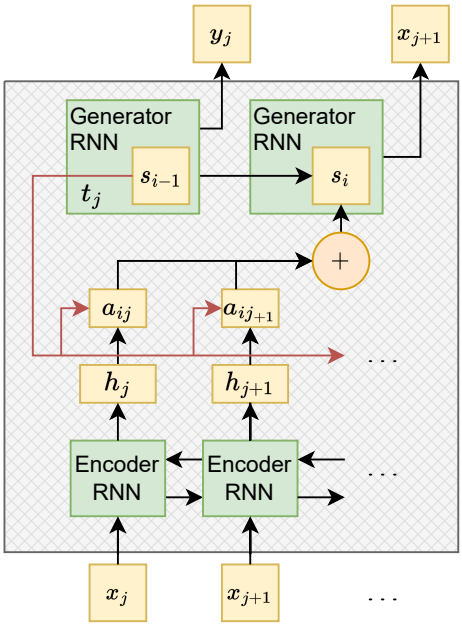


Abbildung 5.3: Einbettung von *Attention* in eine RNN Struktur nach Bahdanau, Cho und Bengio [12]

$$score(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & dot \\ h_t^\top W_a \bar{h}_s & general \\ v_a^\top \tanh([h_t; \bar{h}_s]) & concat \end{cases} \tag{5.2}$$

$$a_t(s) = softmax(score(h_t, \bar{h}_s)) \tag{5.3}$$

wobei $\top$ eine Transposition angibt.

5.3.2 Transformer

Im Gegensatz zu dem sonst üblichen Ansatz, *Attention* als einen zusätzlichen Mechanismus zur Bewertung von HVs einer *Encoder-Decoder*-Struktur zu verwenden, präsentiert Vaswani et al. zum ersten Mal eine NN-*Architektur* zur

Verarbeitung von Sequenzen, welche ganz ohne Rekurrenz auskommt. Der *Transformer* setzt dabei auf eine tiefe Netzstruktur, die sowohl im *Encoder* als auch im *Decoder* auf wiederkehrende *Attention*-Module setzt. Diese sind in Abbildung 5.5 dargestellt.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}V\right) \tag{5.4}$$

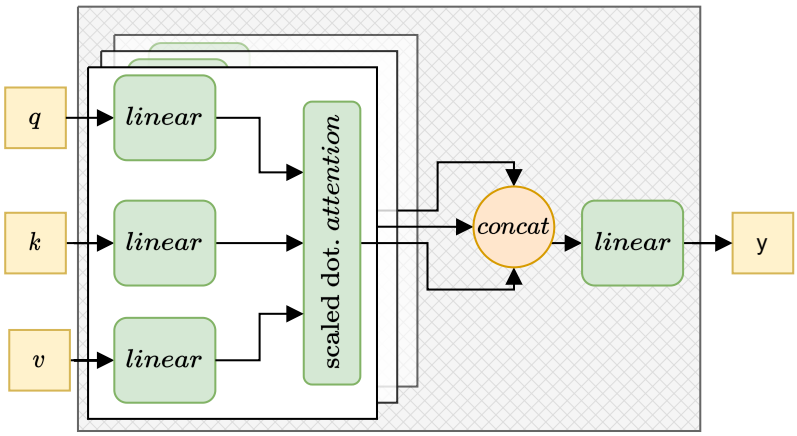


Abbildung 5.4: Aufbau der *MultiHead Scaled Dot Attention* nach Vaswani et al. [236]

Die hier zum Einsatz kommende Form von Self-*Attention* entspricht weitestgehend der *dot-Attention* nach Luong, Pham und Manning, wobei als Q, K, V-Vektor im *Encoder* jeweils die gleichen Vektoren zum Einsatz kommen. Dies wird auch als *intra-Attention* bezeichnet, da hier die Bedeutung jedes einzelnen Elements in den Bezug zu allen anderen Elementen einer Sequenz gesetzt wird. Durch den in Abbildung 5.4 dargestellten parallelen Einsatz der *Attention*-Module, auch *Multi-Head* genannt, wird es dem *Model* ermöglicht, auf unterschiedliche *Features* gleichzeitig zu „achten“. Dies ist vorwiegend durch eine zusätzliche Skalierung innerhalb der Berechnung der *Attention*, wie in Gleichung 5.4 dargestellt, möglich. Die Einsatzweise dieser Module kann dabei mit den Filtereinheiten eines dynamischen CNN mit einem 1×1-großen Kernel verglichen werden [236]. Auch der *Transformer* erhält den Gradienten durch die tiefen Strukturen dieser NN-*Architektur* aufrecht, indem *Residual Skip Connection* um einzelne Module gespannt werden. Im Laufe der vergangenen Jahre wurde diese *Architektur* in nahezu allen Forschungsfeldern, die sich mit DL beschäftigen, implementiert. Eine breite Übersicht liefert Wen et al. [244].

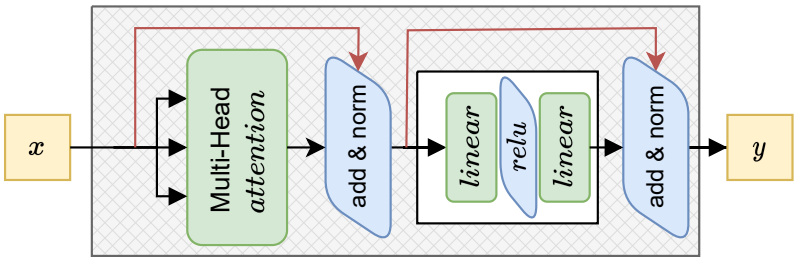


Abbildung 5.5: Typischer Aufbau des *Transformers Encoders* nach Vaswani et al. [236]

5.3.3 *Tree-Structured Parzen Estimator (TPE)*

Wie in Abschnitt 2.3.4.2 erwähnt, existieren neben den linearen oder zufälligen Abtastverfahren für HP-Suchräume auch deutlich komplexere Ansätze. Dies ist sinnvoll, da nicht jeder Parameter die gleiche Relevanz (Importance [99]) für ein durch ein *Model* zu lösendes Optimierungsproblem besitzt. Unterschiedliche Parameter können z. B. unterschiedliche Charakteristiken hinsichtlich des Einflusses auf die *Model-Performance* und der zugrunde liegenden Funktion oder auch ihres Wertebereichs $V \in [\mathbb{R}, \mathbb{R}^+, \mathbb{N}]$ sowie der Werteverteilung ($[uniform,\ log-uniform,\ quantized\ log-uniform,\ categorical\ variables]$) aufweisen. Betrachtet man den Instanziierungs-Prozess (*Init*) von NNs, kann zudem angenommen werden, dass einige *Model*-HP hierarchische Beziehungen zueinander haben [18]. Lineare Suchstrategien wie GS machen daher in so einem heterogenen Suchraum nur wenig Sinn.

Tree-Structured Parzen Estimator (TPE) nutzt daher die Annahme des hierarchischen *Model*-Entstehungsprozesses, um $p(\beta|y)$ durch die Annäherung zweier Dichteverteilungen $l(\beta), g(\beta)$ zu modellieren (vgl. Gleichung 5.5), wobei x einem Parameter aus dem HP-Raum $\mathcal{B}$ und y dem Resultat einer Fitnessfunktion (*Score*, z. B. NN-*Loss*) entspricht. Dabei bildet $l(\beta)$ die Dichteverteilung für Fälle ab, in denen $\beta^{(i)}$ so gewählt ist, dass $y = f(\beta_i); i, \ldots, |B|$ geringer als ein dynamisch formuliertes Optimum $y*$, während $g(\beta)$ durch die übrigen Fälle $y \leq y^*$ definiert wird. Zusätzlich wird für jedes β eine Prior mit eigenen Charakteristiken $[uniform\ in\ (a, b),\ gaussian,\ log-uniform]$ angenommen [18, 19, 182]. So kann TPE verwendet werden, um den vollen heterogenen HP-Raum hinsichtlich $y*$ zu optimieren, wobei $y*$ in diesem Prozess aktualisiert werden muss.

$$p(\beta|y) = \begin{cases} l(\beta) & \text{if } y < y^* \\ g(\beta) & \text{if } y \geq y^* \end{cases} \tag{5.5}$$

5.4 Experimenteller Aufbau

In diesem Abschnitt wird zunächst das Modell des ViT eingeführt und die Besonderheiten in Bezug auf die Anwendung auf sequenzielle Audiodaten besprochen. Dann folgt die Ansprache der zum Vergleich herangezogenen NN-*Architekturen* sowie grundlegenden Beschaffenheiten. Abschließend werden die weiteren verwendeten Methoden zur Durchführung der Experimente in Abschnitt 5.5, welche bisher nicht in Kapitel 2 & 2.4.2 vorgestellt wurden, behandelt.

5.4.1 Datasets

Die Ergründung der Anwendbarkeit des (im Kontext der Audio-Klassifizierung) neuartigen ViT erfolgt anhand zweier aktueller *Real-World*-Datensätze, eine *Binary-* und eine *Multi-Class Classification*, welche zunächst in diesem Abschnitt vorgestellt werden.

Primates-*Datensatz (PRS)*

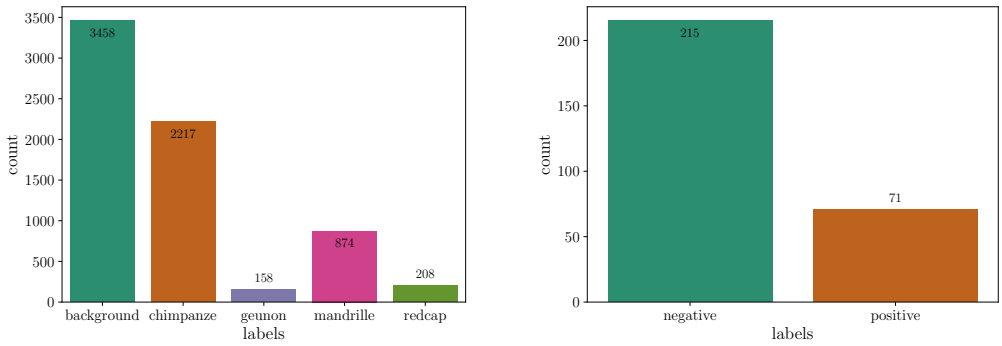
Zwerts et al. erhoben die Daten des Primates-*Datensatz (PRS)*, welcher aus akustischen Aufnahmen aus dem *Mefou* Primatenschutzgebiet[2] besteht. Das Ziel hierbei war es, eine auf Audioaufnahmen basierende, nicht-invasive Möglichkeit zu schaffen, um verschiedene bedrohte Affenarten zu identifizieren und zu zählen [258]. Dieser Datensatz enthält Audioaufnahmen mit den folgenden fünf vordefinierten *Labels*, wodurch eine *Multi-Class Classification* angeboten wird:

- *background* (Hintergrundgeräusche; Urwald)
- *chimpanzee* (Schimpansen)
- *guenon* (Meerkatzen)
- *mandrill* (*Mandrillartige*)
- *redcap* (Halsbandmangabe; *Rotkopfmangabe*)

Wie leicht zu erkennen ist, handelt es sich hier entweder um eine Primatenart oder um Hintergrundgeräusche (*background*), also Aufnahmen aus dem natürlichen Lebensraum der Primaten. Die hier enthaltene Klassifizierungsaufgabe besteht also darin, je *Samples* zu entscheiden, ob eine Aufnahme im Datensatz einen Primatenlaut enthält und wenn *ja*, welcher Spezies dieser zuzuordnen ist. Wie Zwerts et al. angibt, weist dieser *Real-World*-Dataset eine Besonderheit auf: Da es nicht möglich ist, mit totaler Sicherheit in dieser Umgebung wirklich immer nur die beabsichtigten Geräusche aufzuzeichnen, können leichte Vermischungen unter den Klassen entstehen. Datensätze, welche diese Eigenschaft

[2]Zentralafrika: `https://goo.gl/maps/QKPrxNNBcnFznY5g7` (Google Maps)

aufweisen, werden, nicht nur, aber auch, als eng. *weakly labeled* bezeichnet [197, 224]. Es können also z. B. als *background* gekennzeichnete *Samples* schwache Primatenlaute enthalten, bzw. enthalten alle *Samples*, welche Primaten zugeordnet sind, immer die Hintergrundgeräusche ihrer „natürlichen“ Umgebung. Dies sorgt für eine erhöhte Verwechslungsgefahr im Lernprozess und muss entsprechend berücksichtigt werden.



(a) PRS (b) CCS

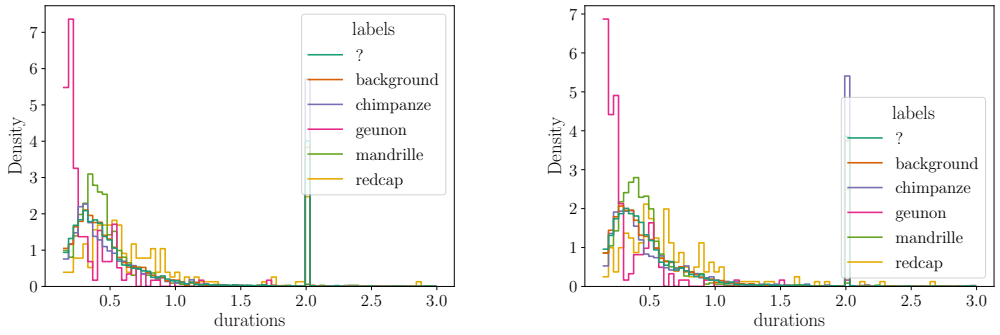
Abbildung 5.6: Statistiken den Primates-*Datensatz (PRS)* und den COVID-19 Cough-*Datensatz (CCS)*: **5.6a** und **5.6b** zeigen jeweils die Anzahl der *Samples* pro Klasse.

Als zweite, für das Training negative Eigenschaft dieses Datensatzes, lässt sich neben den vermischten *Labels* das Ungleichgewicht in der Anzahl an vorhandenen Trainings-*Samples* pro Klasse, wie in Abbildung 5.6a dargestellt, nennen. Betrachtet man die Anzahl der *Samples* sowie deren Verteilung über die Klassen, sind Trainings- und Validierungsdatensatz vergleichbar aufgestellt. Dafür fällt aber eine leichte Schwankung bezüglich der Länge der Audioaufnahmen in Abbildung 5.7a & 5.7b auf, wobei die allgemeine Verteilung miteinander vergleichbar sind. Anders verhält es sich in diesem Punkt bezüglich des Testdatensatzes, hier sind Abweichungen in der Stichprobenlänge hinsichtlich der Anzahl der sehr kurzen *Samples* ($<= 0,3$ Sekunden) festzustellen.

Ferner und am problematischsten ist hingegen die schwankende Dauer der Aufnahmen (vgl. Abbildung 5.7a) über alle Klassen hinweg. Besonders auffällig ist hier die Klasse der *Guenon*, sie ist nicht nur unterrepräsentiert, sondern kommt auch mit relativ kurzen Aufnahmedauern. Weitere einführende Erläuterungen dieses Datensatzes sind verfügbar [206, 258].

COVID-19 Cough-*Datensatz (CCS)*

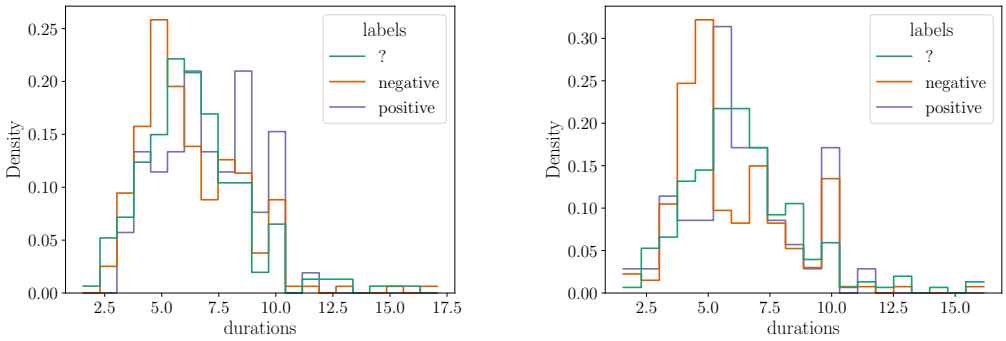
Neben der eben vorgestellten Sammlung an „natürlichen“ Primatenlauten, ist der aus dem „aktiven“ Zeitraum der *Coronavirus disease 2019 (COVID-19)*-Pandemie stammende COVID-19 Cough-*Datensatz (CCS)* [29], ein interessanter Kandidat für die Betrachtung der praktischen Anwendbarkeit des ViT im Kontext sequenzieller Audiodaten. Dieser *Real-World*-Dataset umfasst Stichproben



(a) Länge der Trainingsaufnahmen (b) Länge der Entwicklungsaufnahmen

Abbildung 5.7: **5.7a** & **5.7b** zeigen die Verteilung der Länge der Audioaufnahmen pro Klasse im Primates-*Datensatz (PRS)*. Dauer des Testsatzes: *?*).

atmender und hustender Personen mit und ohne aktiver (nachgewiesen) *Severe acute respiratory syndrome - Coronavirus 2 (SARS-CoV-2)* Infektion, welche durch ein *Crowdsourcing*-Verfahren (*Android-* & Web-Anwendung) erhoben worden sind. Aus diesen Aufnahmen ergibt sich eine binäre Klassifikationsaufgabe, bzw. die Frage, ob das Husten und Atmen von *negativ* und *positiv* getesteten Personen genutzt werden kann, um eine eventuelle Klangbildveränderung durch SARS-CoV-2, zu erlernen. Vergleicht man die Stichproben mit denen des PRS-Datensatzes, weisen die CCS-Aufnahmen die fünffache Länge auf (bis zu 16 Minuten, vgl. Abbildung 5.6b). Beim Blick auf die Stichprobenlängenverteilung in Abbildung 5.8a & 5.8b sind kleinere Unterschiede zwischen dem Trainings- und Validierungsdatensatz festzustellen. Es kann festgestellt werden, dass dieser *Real-World*-Dataset hinsichtlich der binären Klassifikationsaufgabe nicht nur unausgewogen ist, sondern auch die Anzahl der Stichproben insbesondere im Hinblick auf die üblichen Größenordnungen im Bereich des *Deep Learning*, mit 215 positiven gegen 71 negative Stichproben eher gering als eher kleiner Datensatz betrachtet werden kann. Führt man sich die in Kapitel 4 beschriebene Spanne an möglichen Variationen der Klangbilder der menschlichen Stimme vor Augen, erscheint das hier zu lernende Problem durch die gegebenen Daten deutlich unterrepräsentiert. Weitere einführende Erläuterungen finden sich z. B. in der zum Datensatz veröffentlichten Begleitliteratur [29, 206]. So ist z. B. auch eine variierende Gehegegröße und damit eine variierende Distanz zwischen Aufnahmegeräten und Primatenarten zu entnehmen.



(a) Trainingsdatensatz (b) Entwicklungsdatensatz

Abbildung 5.8: **5.8a** & **5.8b** zeigen die Verteilung der Länge der Audioaufnahmen pro Klasse im CCS. Dauer des Testsatzes: *?*

5.4.2 ViT & NN-Model Architectures

Wie in Abschnitt 5.1 motiviert, soll hier überprüft werden, wie und in welcher Art und Weise der von Dosovitskiy et al. eingeführte *Vision Transformer (ViT)* auch im Kontext von Mel-Spektrogrammen zur Klassifikation von sequenziellen Audiodaten Anwendung finden kann.

Vision Transformer (ViT)

In Abschnitt 5.3.1 wurde bereits der für die Funktionsweise des ViT [50] maßgebliche *Attention*-Mechanismus [12, 155], sowie in Abschnitt 5.3.2 der für Aufgaben aus dem Bereich des NLP entwickelte *Transformer* [236] eingeführt.

Zum besseren Verständnis der ViT-Arbeitsweise kann die Betrachtung der ursprünglichen Arbeitsweise des *Transformers* im Kontext des NLP herangezogen werden. Dort wird jedes Element eines einzelnen Satzes (Wörter, Satzzeichen & Symbole) mit jedem anderen Element der Sequenz (Satz) und sich selbst in (eine quadratische) Beziehung gesetzt. Da ein NN auf Zahlen und nicht auf Buchstaben arbeitet, hat es sich etabliert, Wörter als Vorverarbeitungsschritt in ein *Embedding* zu überführen. Diese parametrisierten und erlernbaren Transformationsfunktionen stellen eine sinnvolle Beziehung zwischen einzelnen Wörtern her und betten diese abschließend in ihren eigenen Vektorraum ein, der sich aus dem hierzu verwendeten Datensatz (*Korpus*) konstituiert.

Anstatt von Worteinbettungen wendet der ViT das gleiche Prinzip auf als Sequenz interpretierte Bilder an. Zunächst wird, ähnlich wie bei *Convolution-Layer*, der *Input* zunächst in immer gleich große, nicht überlappende Bereiche, *Patches* unterteilt. Diese sind mit CNN-*Kernels* (vgl. Abschnitt 4.3.1), sowie dem dazugehörigen Abtastverfahren vergleichbar, wobei das Konzept der *Strides* im ViT-Kontext bisher nicht untersucht wurde. Anschließend erfolgt die Einbettung auf einer per-*Patch*-Basis, welche im Laufe des Trainings mit optimiert, sprich gelernt wird.

In Anlehnung an die ursprüngliche ViT-Implementierung wird auch hier ein, als Stapel von *Multi-Head Self-Attention* (vgl. Abbildung 5.4 & Gleichung 5.8) Blöcken realisierte *Architektur*, deren Grundidee die parallele Ausführung mehrerer *Self-Attention*-Operationen ist, etabliert. Diese werden dann subsequent in einer einzigen d/n_{heads}-großen Einbettung zusammengeführt, wobei d der Größe dieses Vektors und n_{heads} der Anzahl an parallelen *Attention*-„Köpfen" entspricht. Gleichung 5.7 & 5.8 definieren diese Operation.

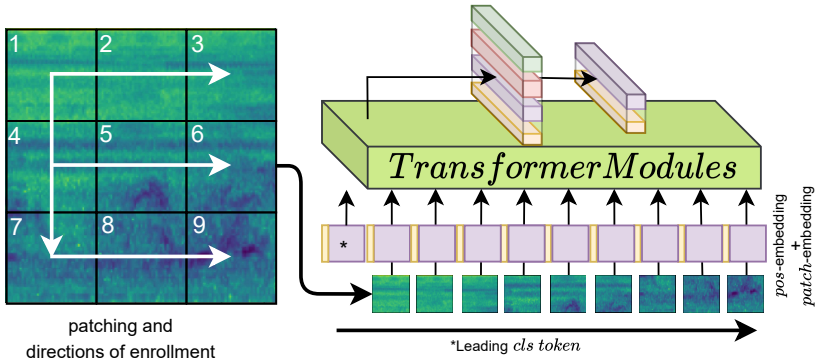


Abbildung 5.9: *Vision Transformer (ViT)* Aufbau der *Architektur* und Anwendung auf Mel-Spektrogrammen

Geg. sei die arbiträre *Input*-Sequenz $\underset{N\times D}{x}$ von N-Elementen mit je D Werten, sowie der *Weight*-Matrix $\underset{D\times(3\times D_h)}{W_{qkv}}$, wobei D_h die Größe jedes *Head-Embedding* darstellt:

$$\begin{aligned} z_{qkv} &= x \odot W_x \\ [q,k,v] &= z_{qkv} \\ A(q,k,v) &= softmax(\frac{Q\cdot K^T}{\sqrt{D_h}}) \\ SA(z) &= A \odot z \end{aligned} \tag{5.6}$$

Hierbei ist anzumerken, dass $[\cdot,\cdot,\cdot]$ die Zerlegung eines Vektors in drei gleiche Sub-Vektoren angibt. *Multi-Head Self-Attention* erhält man für k-Köpfe mit $i \in \mathbb{N}^+, i \leq k$ und der *Weight*-Matrix $\underset{(k\times D_h)\times D}{W_{msa}}$ aus:

$$head_i = SA(qW_{iq}, kW_{ik}, vW_{iv}) \tag{5.7}$$

$$MHSA(q,k,v) = [head_1, \ldots, head_k] \odot W_{msa} \tag{5.8}$$

Die Art und Weise, wie der ViT [50] die reihen-, dann spaltenweise Segmentierung der Eingabedaten vornimmt, wirkt zunächst zufällig (vgl. Abbildung 5.9). Aber wie auch der *Convolution-Layer* arbeitet ein *Attention*-Modul parallelisiert auf einem *Input*. Da wichtige *Features* bei regulären Bildern über die

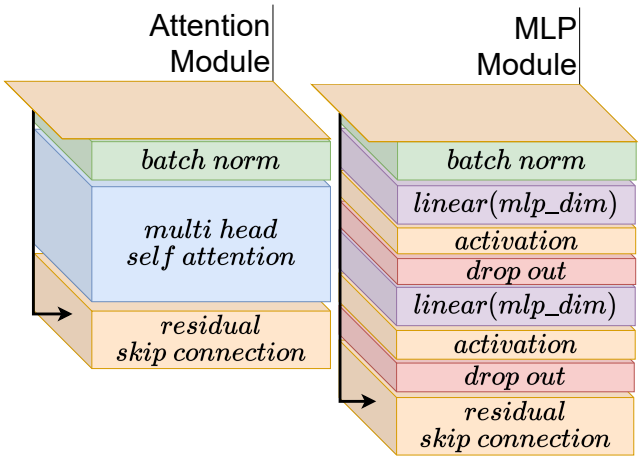


Abbildung 5.10: Beschreibung der Transformer-Komponenten

zweidimensionale Fläche (das Bild) und alle (drei) Farbkanäle verteilt sein können, bietet sich dieses Verfahren an. Zudem spielt eine Reihenfolge des *Inputs* bei der *Transformer-Architektur* keine Rolle, da die quadratische Beziehung im *Attention*-Modul immer gleich und unabhängig arbeitet. Dies hat Dosovitskiy et al. hinreichend untersucht und bei der Vorstellung des ViT beschrieben.

Vertical Vision Transformer (VViT)

Eine typische Größe in CNNs stellt der quadratische *Convolution-Kernel* dar. Bis heute existieren nur wenige einflussreiche Studien zu CNNs mit anderen *Kernels*-Varianten, die von der typischen $n \times n$-Form abweichen. Zwar entstehen immer noch Studien zu Alternativen (z. B. [145]), bislang hat sich allerdings noch keine von diesen in der *Computer Vision* durchgesetzt, weshalb sich bis heute die Verwendung quadratischer $n \times n$ gewissermaßen als der Standard anzusehen ist (vgl. Abschnitt 4.3.1).

Bei der Verarbeitung von Sequenzen in Form von Mel-Spektrogrammen durch CNN lässt sich Ähnliches beobachten: Zwar bestehen (auch einflussreiche) Arbeiten zu verschiedenen *Kernel*-Formen (z. B. [148, 189]), durchgesetzt haben sich diese allerdings bislang nicht. Das verwundert, denn aufgrund der generellen *Gestalt* der aus Audioaufnahmen generierten Mel-Spektrogramme können gewisse Annahmen getroffen werden. Pons et al. stellt eingängigerweise fest, dass es *Models* mit kleinen (quadratischen) *Kernels* schwerfällt, den gesamten harmonischen Umfang der Frequenzbänder zu erfassen [189]. Im Gegensatz zu Bilddaten mit heterogenen Topologien und scheinbar zufälligen Rotationen und Platzierungen der Hauptmerkmale (*Feature*) folgen Mel-Spektrogramme immer einer inhärenten Ordnung entlang der Frequenz- und der Zeitachse.

Auf Basis dieser Gedanken soll zusätzlich zu der Anwendbarkeit der *Attention*-basierten ViT-*Architekturen* untersucht werden, inwieweit sich andere Abtastverfahren, sprich *Kernel*-Formen, auf die *Performance* dieser auswirken. Eine Variante kann hier intuitiv genannt werden: die horizontale Abtastung

entlang der Frequenz-Dimension, welche alle Mel-Bänder gleichzeitig zu einem kleinen Zeitausschnitt ins Auge fasst. Es soll also ein Fenster mit der Auflösung $res = [s_i, \ldots, s_{i+n}] \times |S_y|, \langle i, y \rangle \in \mathcal{N}$, wobei S_y der vollen Anzahl an Elementen der Mel-Dimension, S_x der Anzahl an Elementen eines schmalen Bereichs der Zeit-Dimension entspricht, definiert werden. Dieses wird auf allen möglichen Fenster-Positionen, entweder überlappend oder bündig aneinandergereiht und das Spektrogramm so unterteilt. Weitere Details zur genauen Implementierung finden sich in Abschnitt 5.5.

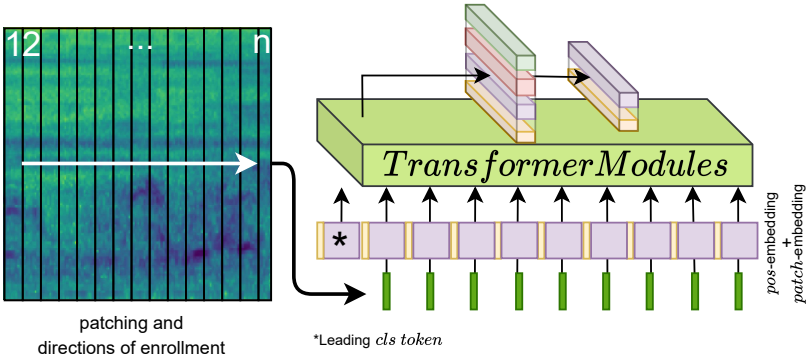


Abbildung 5.11: VerticalVisionTransformer (ViT): Übersicht der *Architektur* und Anwendung

CNN-*Baseline*

Um einen Vergleichswert für die Bewertung der *Performance* der hier betrachteten Anwendung von ViTs bereitzustellen, wird als NN-*Baseline* ein *Convolutional Neural Network (CNN)* nach dem *LeNet*-Vorbild [134], wie in Abbildung 5.12 dargestellt, realisiert. Der grundlegende Aufbau gestaltet sich durch die Aneinanderreihung von (vier) CNN-Modulen, von denen jedes, wie in Abschnitt 5.5 angegeben, aufgebaut ist. Es folgt ein MLP-basierter *Classifier*, welcher die letzte FM als *Input* erhält. Ganz allgemein folgt dieses *Model* der schon in Abschnitt 4.4.2 beschriebenen *Architektur*. Als Unterschied ist lediglich der letzte *Fully-Connected Output-Layer* zu nennen. Dieser wird in Abhängigkeit der Aufgabe (PRS / CCS), durch eine Sigmoid- oder eine Softmax-Activation-Function, angepasst.

SubSpectralClassifier (SSC)

Als zusätzliche Vergleichswerte soll der schon aus der Vorstudie (vgl. Abschnitt 4.4.2 [101]) bekannte *SubSpectralClassifier (SSC)* herangezogen werden, da sich diese *Architektur* ungewöhnlich deutlich hinsichtlich der gemessenen UAR-*Performance* von den übrigen *Architekturen* abgehoben hat und somit einen starken Vergleichswert für einen ViT darstellt. In Kürze: wie in Abbildung 5.13 dargestellt, werden vier *Convolution*-Module auf verschiedene, sich nicht überlappende Mel-Bereiche je der Größe $\lfloor |S_y|/4 \rfloor$ trainiert. Anschließend

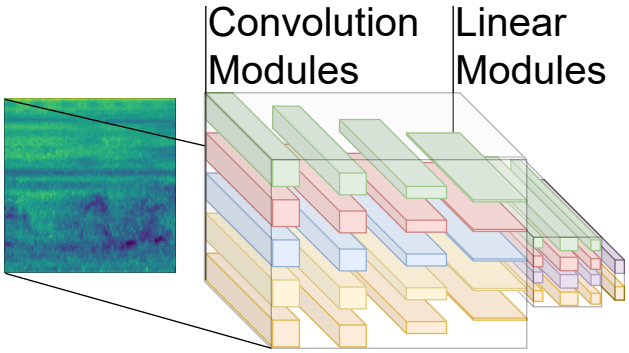


Abbildung 5.12: Architektur der Convolutional *Baseline* (CNN)

durchlaufen diese verketteten bandweisen FMs einen *Classifier*. Implementierungsdetails finden sich in Abschnitt 5.5.

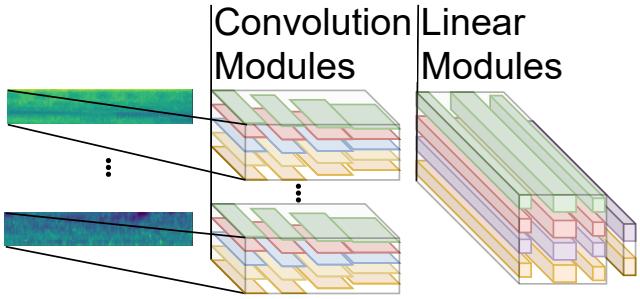


Abbildung 5.13: Architektur des SubSpectral Classifier (SSC)

Multiclass-*Unweighted Average Recall (UAR)*

Wie auch in Kapitel 4, wird die Güte der Klassifikation für die beiden Datensätze (Primates-*Datensatz* und COVID-19 Cough-*Datensatz*) durch *Unweighted Average Recall (UAR)* ausgedrückt. Der allgemeine Fall mit beliebig vielen Klassen c der UAR wird berechnet durch:

$$UAR = 1/c \cdot \sum_{c=1}^{C} \frac{TP_c}{TP_c + FN_c} \tag{5.9}$$

5.4.3 *Neural Network*-Training

In diesem Abschnitt soll beschrieben werden, wie das Training der in Abschnitt 5.4.2 beschriebenen *Models* auf den zuvor Abschnitt 5.4.1 eingeführten Datensätzen konzipiert ist und welche Besonderheiten dabei zu beachten bzw. wie diese zu behandeln sind, damit ein Training auf solcher *Real-World*-Datasets

erfolgreich verlaufen kann. Es empfiehlt sich z. B. wie zuvor in Kapitel 4 auch, aufgrund der teilweise stark unterschiedlichen Klassengrößen (*Samples* pro Klasse) Maßnahmen zu ergreifen, um diese Eigenschaft des Datensatzes nicht zu einem Problem der *Models* werden zu lassen. Aktuell haben sich im Bereich des DL (allgemein) besonders zwei Methoden durchgesetzt, um genau gegen dieses Problem vorzugehen: Focal-Loss [144] und Oversampling.

Focal Loss

Das *Focal-Loss*, welches auch in der parallel verfassten Studie von Müller, Illium und Linnhoff-Popien am Primates-*Datensatz* untersucht wurde, kann wie folgt beschrieben werden: Grundsätzlich verschiebt dieses Verfahren den *Model-Loss* so, dass als Resultat der durch BP angewendete Gradient in jeder *Batch* durch leicht zu erlernende Beispiele ab- und durch schwer zu erlernende Beispiele aufgewertet wird. Je größer der Parameter γ gewählt wird, desto stärker verlagert sich die Gewichtung hin zu den schwer zu erlernenden Beispielen im Trainingsdatensatz.

$$FL(p_t) = -(1 - p_t)^{\gamma} \log(p_t) \tag{5.10}$$

Random Oversampling (ROS)

Da die Frequenz, mit der ein NN *Samples* im Training zu Gesicht bekommt, eine wesentliche Rolle bei der Anpassung der *Weights* spielt, kann zur Verbesserung der *Model-Performance*, während des zufälligen Zusammenstellens einer *Batch*, eingegriffen werden. *Random Oversampling (ROS)* [116][3] ist dabei die denkbar einfachste Methode, dieses Problem anzugehen: *Samples* aus unterrepräsentierten Klassen werden innerhalb einer Epoche mehrfach aus dem Trainingsdatensatz ausgewählt, durch DA manipuliert und in der aktuellen *Batch* aufgenommen, sodass ein NN keinen *Bias* durch häufiger vertretende Klassen erfährt. Anders ausgedrückt können durch ROS Defizite in der Datenverteilung pro Klasse (und Länge im Audio- oder NLP-Bereich) ausgeglichen werden. Nimmt man z. B. an, dass gleich verteilte Klassen als erste Näherung eine bessere Lerngrundlage (als die Ausgangssituation mit überrepräsentierten Klassen) darstellen, errechnet sich die Gesamtzahl der Stichproben $S_{tot} = max(len(S_{class})) \cdot n\_classes$. Durch eine Anpassung der Auswahlwahrscheinlichkeit (p) im zufälligen Prozess der *Batch*-Zusammenstellung kann so das Training aktiv beeinflusst werden: $p = 1/len(S_{class})$ Während der ersten Untersuchungen (mit allen *Models*) stellte sich heraus, dass beide hier behandelten Datensätze (PRS und CCS) durch das starke Ungleichgewicht der Klassen (vgl. Abschnitt 5.4.1) mindestens eine der beiden Methoden benötigen, da sonst kein Lernerfolg erzielt werden kann.

[3] früheste nachzuvollziehende Erwähnung; ohne Akronym.

Ausgleich der *Sample*-Längen

Wie Abschnitt 5.4.1 gezeigt hat, sind nicht nur die Klassen unausgeglichen, sondern auch die Verteilungen der erhaltenen *Samples* hinsichtlich der Länge der Aufnahmen. Dies kann unter Umständen zum Lernen von Abkürzungen (*shortcuts* [71]) führen, sodass ein *Model* zwar auf *Benchmark*-Datensätzen (oder Validierungsdaten) gute Ergebnisse liefert, allerdings schlecht generalisiert. Die Studie von [168] hat für diesen Fall gezeigt, dass die Klassifikation von sequenziellen *Real-World*-Daten deutlich von der Aggregation sequenzieller Auswertungen profitiert. Daher wird hier, um den Bias der unterschiedlichen Klassenlängen zu reduzieren, auf das durch Müller et al. beschriebene Verfahren zurückgegriffen, welches auf Sub-*Sample*-Ebene trainiert und evaluiert.

Definition 5.4.1: Sub-Segment Training

Geg. sei ein NN $NN_\theta(s_{n\to m}) : \mathbb{R}_{\times|s_{n\to m}|} \to \mathbb{R}$, einer Sequenz x_i aus Datensatz D mit $i, \ldots, |D|$ und $\langle s_{n\to m}\rangle \subset x_i$ mit $n < |x_i|, m \leq n < |x_i|$. Dann sei $s_{n\to m}$ ein Sub-Segment der Länge $d, d \in N^+$, wobei $m = n + d$.
Dann ist die Gesamtaussage eines NN-*Model*, trainiert auf $s_{n\to m}$: $\hat{NN}(x_i) = f([NN_\theta(s_{1\to m}), NN_\theta(s_{1+j\to m+j}), \ldots, NN_\theta(s_{n\to|x_i|})]$ mit $f \in (mean, max)$. Sprungweite j sollte so definiert sein, dass $0 = |x_i| \bmod j$.

Data Augmentation mit Mel-Spektrogrammen

Als DA-Methoden kommen die in Abschnitt 4.3.3 beschriebenen Verfahren zum Einsatz, wobei auf *SpeedAugmentation* verzichtet werden muss, da der Zugewinn an zuvor beobachteter *Performance* in keinem Verhältnis zum zusätzlichen Rechenaufwand steht (vgl. Tabelle 4.2). Die zuvor genannten Parameter werden im Zuge der Experimente in Abschnitt 5.6 als *Model*-HPs behandelt und durch TPE auf die jeweilige Kombination aus Datensatz (PRS oder CCS) und *Model* angepasst. Angaben, die nachfolgend auf Ziehungen aus Zufallsverteilungen hinweisen, werden bei jeder Auswahl eines *Sample* aus dem jeweiligen Datensatz ausgeführt und sind unabhängig.

5.5 Experimenteller Aufbau

In den folgenden Paragrafen wird kurz auf die Parameter der oben erwähnten Methoden eingegangen. Es werden z. B. die sonstigen NNs-HPs sowie das *Model*-Trainings im Allgemeinen besprochen.

Hyperparameter-Optimierung

Um eine hinreichende Abdeckung des HP-Raums zu erreichen, werden 400Durchläufe pro NN-*Architektur*, mit jeweils 200Epochen und Unterstützung von TPE [6, 18], durchgeführt. Dies ist, gemessen an der Zahl an HPs,

keine erschöpfende Suche, kann aber aufgrund der zur Verfügung stehenden Ressourcen (*compute*) nicht ausgeweitet werden. TPE hilft hier, auch mit mittleren Stichprobengrößen, nützliche HP-Konfigurationen zu finden. Da es sich z. B. bei der Sub-Segment-Länge um einen *Hyperparameter* handelt, empfiehlt es sich, die beste Einstellung für jede der hier verwendeten Kombinationen aus Datensätzen (PRS und CCS) und *Models* CC, SSC, ViT, *Vertical Vision Transformer (VViT)* innerhalb der HP-Suche zu bestimmen. Grundsätzlich ergibt sich so z. B. für PRS eine kürzere Sample-Länge ($\sim$ 0.4) als für die Sub-Segment-Samples des CCS-Datensatzes ($\sim$ 1.2) in Sekunden.

NN-Model-Training

Alle NN-*Architekturen* werden *End-to-end* durch *AdamW* [149], als Optimierungsalgorithmus für die Gradienten-basierte BP (vgl. Abschnitt 2.3), 200 Epochen trainiert. Dabei wird auf ES verzichtet, aber das *Model* mit der besten Validierungs-*Performance* ausgewählt. ROS erweitert die unterrepräsentierten Klassen, während die aufgezählten DA-Methoden die Entropie erhöhen. Für alle *Models* werden die *Seeds* $[0; 19]$ festgelegt, welche dann im Training verwendet werden. Alle zuvor definierten HP werden wie angegeben durch TPE für alle NN-*Architekturen* (CNN, SSC, ViT & VViT; vgl. Abschnitt 5.4) optimiert und entsprechend verwendet, sodass jeweils die bestmöglichen Voraussetzungen zur Beurteilung der *Performance* durch die UAR geschaffen sind. Abbildung 5.18 gibt eine Übersicht über die Relevanz dieser Parameter.

NN-*Model*-Parameter

Zum Vergleich wird für alle *Models*, sofern nicht anders angegeben, ein ähnlicher Parameterbereich gewählt. Zwar kann es zu Schwankungen und unterschiedlichen *Model*-Größen, bedingt durch die Einführung zusätzlicher *Weights* z. B. durch Änderungen der Netzwerk-*Architektur*, kommen, generell wird aber darauf geachtet, dass die *Models* in vergleichbaren Größenordnungen agieren. Für beide Aufgaben (PRS und CCS) werden die in Tabelle 5.1 gezeigten Standardparameter definiert.

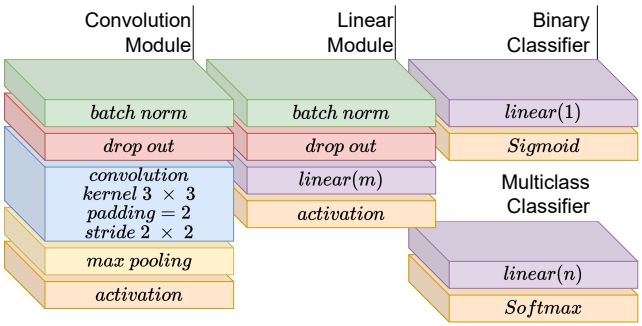


Abbildung 5.14: Beschreibung und Abfolge der (CNN) Modul-Blöcke

CNN-*Baseline*

Alle *Convolution*-Module sind gleichförmig strukturiert und implementieren die im Folgenden gelisteten Elemente in der Reihenfolge der Aufzählung: **1)** BN (Abschnitt 2.3), **2)** *Dropout* (Abschnitt 2.3), **3)** *Convolution* (Abschnitt 4.3.1), **4)** max-*Pooling* (Abschnitt 4.3.1), **5)** Aktivierungsfunktion Abschnitt 2.3.1. Dies ist auch in Abbildung 5.14 („Convolution Module“) dargestellt.

Auf *Stride* wird im Gegensatz zu den *Models* in Abschnitt 4.4.2 verzichtet, da *Pooling* zu besseren Ergebnissen geführt hat.

Die Filtereinheiten der sich wiederholenden *Convolution*-Blöcke sind wie folgt gestaltet: $\{32, 64, 128, 64\}$, wobei die Fenstergröße der *Kernels* mit einer Größe von 3×3 konservativ gestaltet sind, um auch Feinheiten der Mel-Spektrogramme zu erfassen (vgl. Abschnitt 4.4). Durch den Einsatz von *Zero*-spad $= 1$ kann das Seitenverhältnis jeder resultierenden FM beibehalten werden (vgl. Abschnitt 4.3.1). Dies vereinfacht die Größenberechnungen der Tensoren zwischen den Modulen und sorgt dafür, dass *Max-Pooling* immer auf Matrizen arbeitet, die eine gerade Kantenlänge aufweisen. Der *Pooling-Kernel* ist mit 2×2 so gewählt, dass die Auflösung, also die Größe der resultierenden *Feature*-Map, immer halbiert wird.

Ferner werden *Fully-Connected-Layer* mit $[128, 64, n]$ *Neurons* eingesetzt, die (wenn nicht anders angegeben) folgendermaßen aufgebaut sind: **1)** BN (Abschnitt 2.3), **2)** *Dropout* (Abschnitt 2.3), **3)** *Fully-Connected-Layer* (Abschnitt 2.3), **4)** Aktivierungsfunktion Abschnitt 2.3.1.

SubSpectralClassifier (SSC)

Die SSC-HPs entsprechen denen des CNN (volle Übereinstimmung). Abweichend von der zuvor beschriebenen Implementierung (vgl. Abschnitt 4.4.2) wird so z. B. die ReLU- zugunsten der neueren GELU-Activation-Function ausgetauscht.

Vision Transformer (ViT)

Nach einer einfachen *Batch Normalization* folgt eine Schicht mit sog. skaliertem *Dot-Product Attention* (vgl. Abschnitt 5.3.1). Die aus der *ResNet-Architektur* [92] bekannten *Skip*-Verbindungen kommen hier zum Einsatz, um

Tabelle 5.1: Geteilte Standardparameter

	PRS	CCS
n_logits	5	1
loss,	*ce_loss*,	*bce_loss*,
dropout	0.2	0.2
activation,	*gelu*	*gelu*

den ursprünglichen Vektor stärker in die weiteren Berechnungen einzubeziehen (vgl. Abbildung 5.10). Das Ergebnis dieser dynamischen Konvolution wird dann von einem MLP-Modul weiterverarbeitet: Nach einer Normalisierung (BN) folgen zwei *Fully-Connected-Layer*, welche beide jeweils nach GELU-Aktivierung *Dropout* erfahren (vgl. Abbildung 5.10). Auch dieser Block macht sich die zuvor beschriebenen *Skip*-Verbindungen zunutze. Die in dieser Arbeit verwendete *Architektur* schließt, im Vergleich zum ursprünglichen Ansatz [50], noch ein zusätzliches MLP-Modul an, um die resultierende Ausgabe des ViT durch einen *Classifier* als *Feature*-Map zu interpretieren (vgl. Abbildung 5.10).

Vertical Vision Transformer (VViT)

Die Implementierung des VViT folgt dem grundsätzlichen ViT-Design, allerdings wird das Abtastverfahren wie in Abschnitt 5.4.2 an die inhärente Struktur der Mel-Spektrogramme angepasst. Anstatt dem „Zeilen-dann-Spalten"-Prinzip (vgl. Abbildung 5.9 vs. 5.11), schiebt sich ein Fenster in der Höhe der Mel-Bandelemente horizontal entlang der zeitlichen Achse ($n\_mels \times patch\_size, stride = 1$). Erweiterte Experimente haben gezeigt, dass die Breite der überlappenden Felder zwischen 5 und 9 Pixeln am zuverlässigsten funktioniert. Zwar sorgt dieses Abtastverfahren zur Entstehung von mehr und größeren Elementen in der Sequenz der Eingabedaten (im Vergleich zum ursprünglichen ViT), dafür wird der gesamte harmonische Umfang über alle Mel-Bänder beibehalten.

5.6 Durchführung und Ergebnisse

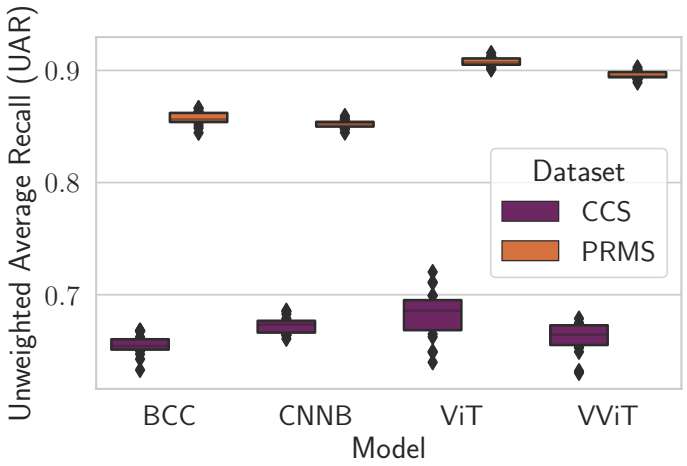


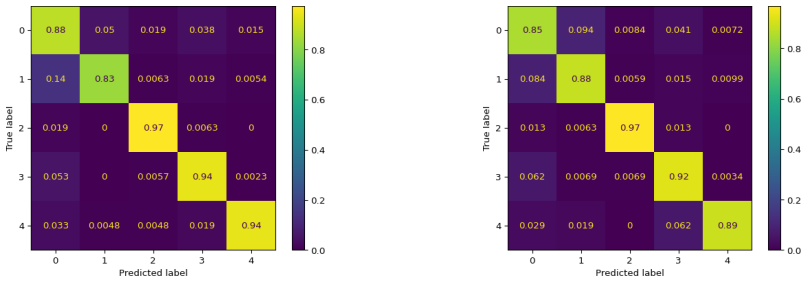
Abbildung 5.15: *Performance* der Trainierten *Models* gemessen als *Unweighted Average Recall (UAR)* und gemittelt über 20 *Seeds*

Auswertungen auf Basis des Entwicklungsdatensatzes zeigen, dass die meis-

Tabelle 5.2: Die Ergebnisse der Experimente im Vergleich zu den mit den *Baselines* der Datensätze PRS & CCS.

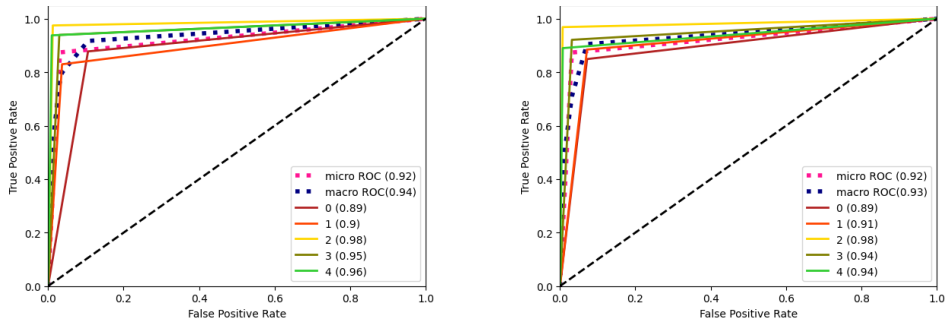
Dataset	**primates**		**coughing**	
Model	Devel	Test	Devel	Test
Deep Spectrum	81.3	78.8	63.3	64.1
End2You	72.70	70.8	61.8	64.7
OpenSMILE	82.4	82.2	61.4	65.5
OpenXBOW	83.3	83.9	64.7	**72.9**
AuDeep	84.6	86.1	67.6	67.6
Fusion	-	87.5	-	73.9
CNN	85.9	-	68.6	67.4
SSC	86.6	84.2	66.8	72.0
ViT	**91.5**	**88.3**	**72.0**	69.9
VViT	90.3	87.2	67.8	68.6

ten der, im Kontext der Datensätze bestehenden Vergleichswerte, selbst von den einfachsten CNN-*Architekturen* (CNN & SSC) übertroffen werden, wobei *OpenXBOW* [207] und *AUDeep* [63] als die stärksten Konkurrenten identifiziert werden können. Je nach Aufgabe (PRS, CCS) übertreffen die *Attention*-basierten (ViT & VViT) sowohl die bestehenden als auch die zusätzlich trainierten Vergleichswerte (CNN & SSC; vgl. Abbildung 5.15). Die *Model-Performance* auf dem Testdatensatz erscheint hingegen in den meisten Fällen zu stagnieren. An dieser Stelle soll darauf hingewiesen werden, dass die im Rahmen dieser Arbeit evaluierten *Model-Architekturen* nur auf den Trainingsdatensätzen trainiert wurden, wohingegen im Gegensatz dazu der Begleitliteratur der Datensätze [206] zu entnehmen ist, dass alle bestehenden Vergleichswerte, auch Daten aus dem Entwicklungsdatensatz für das Training der dazugehörigen Ansätze verwendet wurden.



(a) ViT (b) VViT

Abbildung 5.16: Konfusionsmatrizen zu ViT & VViT



(a) *Receiver Operating Characteristic (ROC)* plot

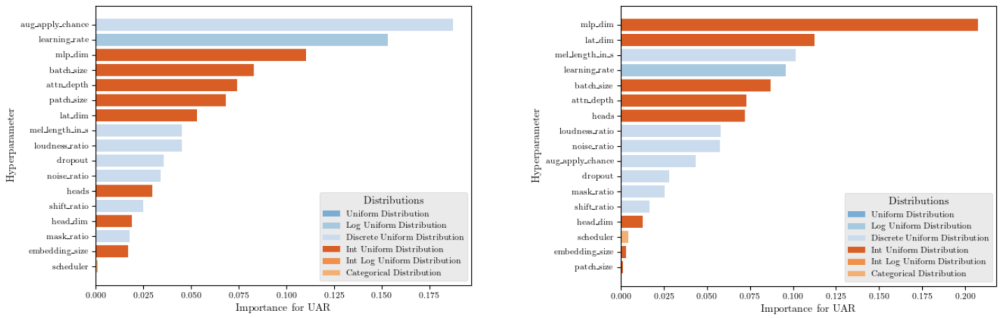
(b) ROC plot

Abbildung 5.17: ROC Darstellungen zu ViT & VViT

Für den Primates-*Datensatz (PRS)* ist eine hohe Verwechselungsrate zwischen der Klasse *background* (0) und *chimpanzee* (1) zu beobachten. Dies gilt sowohl für die beiden *Attention*-basierten Modelle, als auch für CNN & SSC und für Experimente in zusätzlichen Studien abseits dieser Arbeit. Die Darstellung der Konfusionsmatrizen in vgl. Abbildung 5.16 und die ROC-Kurven in Abbildung 5.17 unterstreichen diese Beobachtung. Die Begleitliteratur bestätigt diese unsauberen *Labels*, so können sich Schimpansen schwach unter die Hintergrundklasse gemischt haben [258]. Ein Effekt, der in anderen Veröffentlichungen zu diesem Datensatz bereits zu unterdrücken versucht wurde. [128]

In Bezug auf den CCS-Datensatz kann eine hohe Falsch-Negativ-Rate (bis zu $FNR = 60\%$) über alle getesteten Modelle hinweg festgestellt werden. Dies ist im Hinblick auf eine medizinische Anwendbarkeit schwierig, da hier zumeist eine hohe Spezifität über einer hohen Sensitivität erforderlich ist. Da sowohl die verfügbaren Vergleichswerte als auch die in dieser Arbeit erhobenen Resultate starke Schwankungen zwischen *Entwicklungs-* und *Test*-Datensatz aufweisen, entfällt eine weitere Betrachtung der Ergebnisse. Dieser *Real-World*-Dataset scheint aktuell keine Aussagekraft für die Erforschung und Anwendbarkeit des ViT zu liefern, was eine gängige Gefahr bei der Betrachtung neuer Datensätze ist. Neben der Betrachtung der *Performance* der *Attention*-basierten *Models* wurde außerdem die Bedeutung der *Hyperparameter* beider Varianten (ViT & VViT) analysiert. Abbildung 5.18 stellt die Relevanz der einzelnen HP auf das Lernverhalten dar. Daten zu dieser Metrik wurden während der HP-Suche aggregiert und durch das verwendete Framework[4] erhoben [99]. Mit der Einführung der *Transformer-Architektur* wird in der Literatur zumeist der Einfluss und zu Teilen auch die Notwendigkeit eines passenden *Schedule*-Konzeptes eruiert. Die Untersuchungen im Zuge der HP-Optimierung scheinen darauf hinzudeuten, dass eine solche Betrachtung im Falle der ViT-Varianten keinen Einfluss auf den Trainingserfolg im Kontext des PRS-Datensatzes zu haben scheinen. Abbildung 5.18 zufolge ist es deutlich wichtiger, eine geeignete

[4] *Optuna* - `https://optuna.org/`



(a) ViT parameter importance (b) VViT parameter importance

Abbildung 5.18: Sensitivität der ViT (5.18a) & VViT (5.18b) HP für das Primates-*Datensatz* bestimmt durch die *Importance*-Analyse durch *fANOVA* [99]

LR zu finden, sowie die für den Datensatz passende Anzahl an insgesamt verfügbaren *Weights* der *Fully-Connected-Layer* (*mlp_dim* & *lat_dim*) zu definieren.

Bei näherer Betrachtung fällt die geringe Relevanz der Größe der initialen Einbettungsschicht (*embedding_size*), sowie der Anzahl an *Attention*-Heads *head_dim* für beide *Models* auf. Dies erscheint überraschend, da diese beiden Parameter steuern, wie viel Information auf jeder einzelnen Schicht parallel ausgewertet werden kann: Je niedriger die anfängliche Projektionsdimension ausfällt, desto höher die anfängliche Informationskomprimierungsrate, bzw. desto stärker muss das *Embedding* ausfallen. Es ist interessant zu beobachten, dass die TPE-HP-Optimierung der DA-Parameter nicht nur durch die jeweiligen Datensätze beeinflusst ist, sondern auch von der jeweiligen NN-*Architektur* abhängt, auch wenn diese sich nur geringfügig unterscheiden.

Die hier getroffene Beobachtung, also die geringere Gesamtleistung des VViT, lässt unter anderem den Schluss zu, dass der Gesamtkontext über viele Mel-Bänder hinweg weniger relevant ist, als die quadratische, nicht überlappende Unterteilung des ViT.

5.7 Diskussionen und Zusammenfassung

Dieser Abschnitt hat nicht nur zum ersten Mal den *Vision Transformer* im Kontext der Verarbeitung sequenzieller Audiodaten (in Form von Mel-Spektrogrammen) untersucht, sondern auch die *out-of-the-box* Anwendbarkeit auf *Real-World*-Datasets unterstellt und nachgewiesen. Dabei konnte sich dieses *Attention*-basierte *Model* deutlich von den parallel implementierten *Models* abheben, wobei für alle *Architekturen* jeweils das *Model* mit der höchsten gemessenen *Performance* aus 20*Seeds* genannt wurde. Zusätzlich wurde *Domain-Knowledge* verwendet, um andere naheliegende *Patch*-Formen zu berücksichtigen, was sich

aber (zumindest im Kontext dieser Datensätze) als nicht erfolgreich herausstellte.

Zukünftige Arbeiten sollten andere, von der Standardimplementierung des ViT abweichende, Strukturen untersuchen, um von den neuesten Errungenschaften aus dem Forschungsfeld der *Attention*-basierten *Architekturen* zu profitieren. So gibt es z. B. deutliche Fortschritte im Bereich der Verarbeitungseffizienz, was in Zukunft vor dem Hintergrund der *Green-AI* [210] eine übergeordnete Rolle spielen wird. Eine relevante Arbeit in diesem Feld sind unter anderem der *Performer* [38], der eine Annäherung an die Arbeitsweise des *Attention*-Mechanismus erreicht, ohne explizit die dafür notwendige quadratische Matrix (vgl. Abschnitt 5.3.2) konstruieren zu müssen. Einen ähnlichen Gedanken verfolgen auch andere Ansätze, welche sowohl das Aufstellen als auch die Berechnungen der quadratischen Matrix zu reduzieren versuchen [242, 251]. Ferner haben wiss. Arbeiten gezeigt, dass es vielleicht sogar nicht notwendig ist, den Umweg über *Attention* oder *Convolution* zu gehen, so schafft es der *Mixer* vollständig auf *Fully-Connected-Layer* zu setzen [232]. Wen et al. hinterfragt den Einsatz von *Transformer*-artigen *Architekturen* im Kontext von generativen Aufgaben zur Vorhersage von Zeitserien [244]. Welche Relevanz dies für Aufgaben der *Computer Vision* und Audio-Klassifizierung haben wird, ist zum aktuellen Zeitpunkt nicht bekannt. In eine ähnliche Richtung geht Verma, der von einer vergleichbaren *Performance* durch eine Kombination von *Embeddings* aus AE auf unterschiedlichen *Patches* berichtet.

Eine interessante parallele Entwicklung schlägt ebenfalls die Verwendung des ViT für die Verarbeitung von Mel-Spektrogrammen vor [78] [5]. Gong, Chung und Glass kommt auf den von ihnen verwendeten Datensätzen (*AudioSet*, *ESC-50* & *Speech Commands V2*) allerdings zu dem Ergebnis, dass eine nicht überlappende horizontale Abtastung auch von Vorteil sein kann. Weiterhin werden für die *Weight*-Initialisierung vortrainierte *Models* aus der *Computer Vision* in Betracht gezogen. Forschungsergebnisse legen nahe, dass dieser domänenübergreifende Einsatz möglich ist und zu guten Ergebnissen führen kann [167].

[5] Erstveröff. durch Gong, Chung und Glass bzw. Illium et al. zu Interspeech 2021

6 Feature-Transport als *Data Augmentation*

Angetrieben von *Big Data* und der Verfügbarkeit leistungsfähiger Hardware, haben DNN dank der jüngsten Entwicklungsschritte bemerkenswerte Leistungen in vielen Bereichen der Informatik erreicht und bestehende Ansätze verdrängt. Mit der Tiefe und Breite der dort eingesetzten NN-*Architekturen* steigt oft auch die für das Training notwendige Menge an Daten [3]. Dabei können die für das Training verwendeten Datenquellen allerdings z. B. durch Fehler bei der Beschaffung, Falschetikettierung sowie dem Einbinden unterrepräsentierter oder unausgewogener Klassen fehlerbehaftet sein [101, 102]. Im Umgang mit *Real-World*-Datasets kommt, wie auch die letzten Kapitel gezeigt haben, DA-Methodenaufgrund der Effektivität und einfachen Anwendungsmöglichkeit [215] (neben anderen Regularisierungsverfahren) zum Einsatz. Im Laufe der Zeit wurden so verschiedene Ansätze (z. B. Okklusion, Rekombination, Fragmentierung) und Strategien der Anwendung entwickelt [215]. In diesem Kapitel soll nun aufgrund der zuvor gemachten Erfahrungen eine logische Kombination bestehender DA-Methoden vorgeschlagen und evaluiert werden: VP.

Contribution 6.0.1: Voronoi-basierter Feature-Transport

Beiträge zur Wissenschaft in diesem Kapitel umfassen die Einführung, Motivation und erste Evaluation des Konzepts des *Feature*-Transports auf Basis eines räumlichen Segmentierungsverfahrens als *Data Augmentation (DA)*-Methode. Dazu erfolgt auch eine Evaluation und Erklärung des Parameterraums. Durch diese Methode kann Varianz reduziert und *Overfitting* minimiert werden. Vorveröffentlichung durch Illium. et al. in „VoronoiPatches: Evaluating a New Data Augmentation Method“ [104]

Die Struktur dieses Kapitels gliedert sich wie folgt: Zunächst wird das Konzept Voronoi-Diagramme in Abschnitt 6.1 vorgestellt. Dann werden bestehende und verwandte Arbeiten sowie die dort verwendeten DA-Methodenin Abschnitt 6.2 vorgestellt und diskutiert. Anschließend wird in Abschnitt 6.3 der neu entwickelte Ansatz (VP), der zur Evaluation verwendete Datensatz sowie der Versuchsaufbau vorgestellt. Die Ergebnisse der Experimente werden daraufhin in Abschnitt 6.5 präsentiert. Abschnitt 6.7 fasst die Ergebnisse des Kapitels zusammen und gibt einen kurzen Ausblick auf mögliche Anknüpfungspunkte.

6.1 Methodische Grundlagen

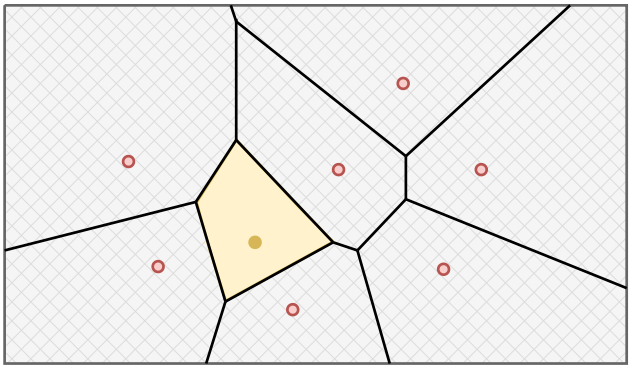


Abbildung 6.1: Ein einfaches Voronoi-Diagramm für acht Generatorpunkte: $V(P)$ mit $|P| = 8$ nach [8]. Die ockerfarbene Voronoi Region ist hier dem ockerfarbenen Generator zuzuordnen.

Voronoi-Diagramme sind geometrische Strukturen, welche die Unterteilung eines Raums durch eine endliche Menge eindeutiger und isolierter Punkte sog. („Generatorpunkte“) definieren. Jeder andere Punkt im Raum ist dann jeweils seinem nächstgelegenen Generator zuzuordnen. Gruppiert man diese Punkte nun nach zugeordnetem Generator, ist es möglich, die Flächen (*Regions*) eines Voronoi-Diagramms zu beschreiben [178]. Diese Definition ist zunächst auf zweidimensionale Voronoi-Diagramme (im euklidischen Raum) beschränkt, wie sie auch in diesem Kapitel verwendet werden, es gibt allerdings auch höherdimensionale Formulierungen [178].

Definition 6.1.1: Voronoi-Polygon

S sei eine Menge von $n \geq 3$ Generator $p, q, r \ldots$ im euklidischen Raum $\mathbb{R}^2$. Der Abstand d zwischen einem beliebigen Punkt $x = (x_1, x_2)$ und dem Generator $p = (p_1, p_2)$ ist gegeben durch:
$d(p, x) = \sqrt{(p_1 - x_1)^2 + (p_2 - x_2)^2}$ Betrachtet man die Generatoren p und q, so kann eine äquidistante Gerade als $D(p, q) = \{x | d(p, x) \leq d(q, x)\}$ definiert werden
Ein Voronoi-Polygon, das zu einem Generator $p \in S$ gehört, $VR(p, S)$, ist die Schnittmenge der Halbebene $D(p, q)$, wobei q über alle p in S reicht: $VR(p, S) = \bigcap_{q \in S, q \neq p} D(p, q)$

Ein Voronoi-Polygon $VP(p) = VR(p, S)$ besteht also aus allen Punkten $x \in \mathbb{R}^2$, für die p die minimale Distanz besitzt. Die Grenzlinien der dabei entstehenden Regionen werden auch als *Kanten* bezeichnet, welche durch ihre Endpunkte (eng. *Vertices*) begrenzt werden. Auf diese Weise entsteht ein konvexes Polygon, das, wie in Abbildung 6.1 abgebildet, begrenzt oder unbegrenzt sein kann. Eine Besonderheit gilt es zu beachten: Alle Punkte,

die im kontinuierlichen Raum genau auf einer Kante liegen, werden beiden angrenzenden Regionen und damit zwei Generatoren zugeordnet. *Vertices* können so bis zu drei Generatoren zugeordnet werden. Die Regionen eines Voronoi-Diagramms bilden zusammen eine polygonale Partition einer Ebene, $V(P)$ [8, 9].

6.2 Verwandte Arbeiten

Nach der Einführung der Methodik zu Voronoi-Diagrammen sowie der bereits erfolgten Vorstellung des Konzepts der DA (Kapitel 4), folgt in diesem Abschnitt ein Überblick über einige zentrale Arbeiten auf dem Gebiet der *Data Augmentation* im Kontext von DNNs.

6.2.1 Maskierung

Methoden, die das Prinzip der Maskierung anwenden, maskieren Teile der Eingabedaten, um eine Veränderung eines *Feature*- oder Objekt-Kontexts hervorzurufen (regionaler *Dropout*, Okklusion). Dabei soll eine Erkennung eines Objekts anhand dessen Struktur, im Gegensatz zu einer Erkennung des (Objekt-) Kontexts, erzwungen werden. Als zwei der jüngsten Methoden aus dem Bereich der Maskierung sind *Cutout* [46] und *Random Erasing (RE)* [256] zu nennen. Beide Verfahren maskieren eine große, zusammenhängende Region je *Sample* (vgl. Abbildung 6.2, 1A & 1B). Je nach Parametrisierung und Verteilung der Objekt-Größenordnungen über die Trainingsdaten werden unter Umständen entweder einzelne *Feature* oder auch ganze Objekte verdeckt.

Der gleichen Motivation folgend, arbeitet *Hide-and-Seek (HaS)*, indem vielfältige Kombinationen kleinerer, benachbarter Regionen in einem Gittermuster aus *Samples* entfernt werden. Dabei können sich allerdings auch größere zusammenhängende Regionen bilden, bzw. einer Maskierung zum Opfer fallen [217]. Dies stellt auch das größte Problem solcher DA-Methodendar, da so durch eine flächige Maskierung das Haupt-*Feature* eines *Samples* im schlimmsten Fall gar nicht oder eben komplett verdeckt werden kann. *GridMask* [36] versucht, diese beiden Extreme zu vermeiden, indem es gleichmäßig verteilte Regionen entfernt (vgl. Abbildung 6.2, 1C, 2A).

Während die Verwendung einfacher orthogonaler Formen und Muster ein gemeinsames Merkmal maskierender DA-Methodenist, zeigt *Voronoi Decomposition-based Random Region Erasing (VDRRE)* einen möglichen Vorteil der Entfernung komplexerer Formen [2]. Für die Erkennung und Klassifizierung von Gesichtslähmungen evaluierte VDRRE die Verwendung von Voronoi-Mosaiken (vgl. Abbildung 6.2, 3A & B).

Über die reine Variation der Formen der Verdeckungsmasken hinaus zeigt sich, dass es auch möglich ist, anstatt der üblichen 0-Werte, Alternativen in

Betracht zu ziehen. Die Optionen reichen hier von Zufallswerten und gemittelten Datensatz-*Pixel*-Werten über die schon bekannten schwarzen (0) oder weißen (1) Maskierungen [2, 36, 46, 217, 252, 256]. Doch auch hier kann das zentrale Objekt des *Samples* bzw. die Haupt-*Features*, wie bei den anderen Methoden auch überschrieben werden, was zu einer Trennung des *Labels* führt [215].

6.2.2 Daten-Rekombination

Im Vergleich zu maskierenden DA-Methodenhat sich unter anderem die lineare oder auch nichtlineare Rekombination von Trainingsdaten ebenfalls als effiziente DA-Strategie erwiesen [227, 252]. Leider ist bislang noch unklar, wodurch genau der *Performance*-Gewinn beim Einsatz dieser Strategie zu erklären ist, da, wie Abbildung 6.2 (4A & B + Zeile 5) zeigt, *Sample* so ihre eindeutige semantische Zuordnung verlieren. [223]

Nichtlineare Mischverfahren wie z. B. Random Image Cropping and Pasting *(RICAP)* [227] und *CutMix* [252] kombinieren mehrere ($2 \dots 4$) zusammenhängende Bildbereiche. Die entsprechenden *Labels* werden proportional zur Fläche jedes verwendeten Bildes gemischt. Der Unterschied zwischen den beiden Ansätzen liegt dabei in der Art der genauen Rekombinationsstrategie. RICAP mischt vier Bilder in einem zwei mal zwei Raster [227], wohingegen *CutMix* einen bestimmten Bildbereich mit dem aus einem anderen *Sample* belegt (vgl. Abbildung 6.2, 5B) [252].

Lineare Mischmethoden wie z. B. *Mixup* [255], *Between-class Learning* [231], und *SamplePairing* [106] kombinieren zwei Trainingsbilder durch die Berechnung des Mittelwertes sich überschneidende *Pixel*-Werte [215]. Inoue schildert, dass dies zu besseren Ergebnissen führt, als das Mischen von Bildern innerhalb von Klassen. Dieses in Abbildung 6.2, 4B & C, 5C dargestellte Verfahren bezieht dabei immer den gesamten Trainingsdatensatz mit ein. Obwohl die so entstehenden Bilder oft keinen semantischen Sinn ergeben, sind diese Verfahren erstaunlich effektiv und führen zu einem deutlichen *Performance*-Gewinn [223]. Summers und Dinneen demonstrieren dies durch die Anwendung einer Reihe von nichtlinearen (aber auch linearen) Methoden, indem *Mixup* oder *Between-class Learning* als Vorstufe vor einer nichtlinearen Methode angewendet [223] wird. Es ist anzumerken, dass Yun et al. Kritik an diesen Methoden äußert, da lineare Methoden „unnatürliche Artefakte“ einführen, welche NN-*Model* irritieren können [252]. Davon abgesehen wird auch dieser Ansatz aktiv weiter erforscht und weiterhin adaptiert [237].

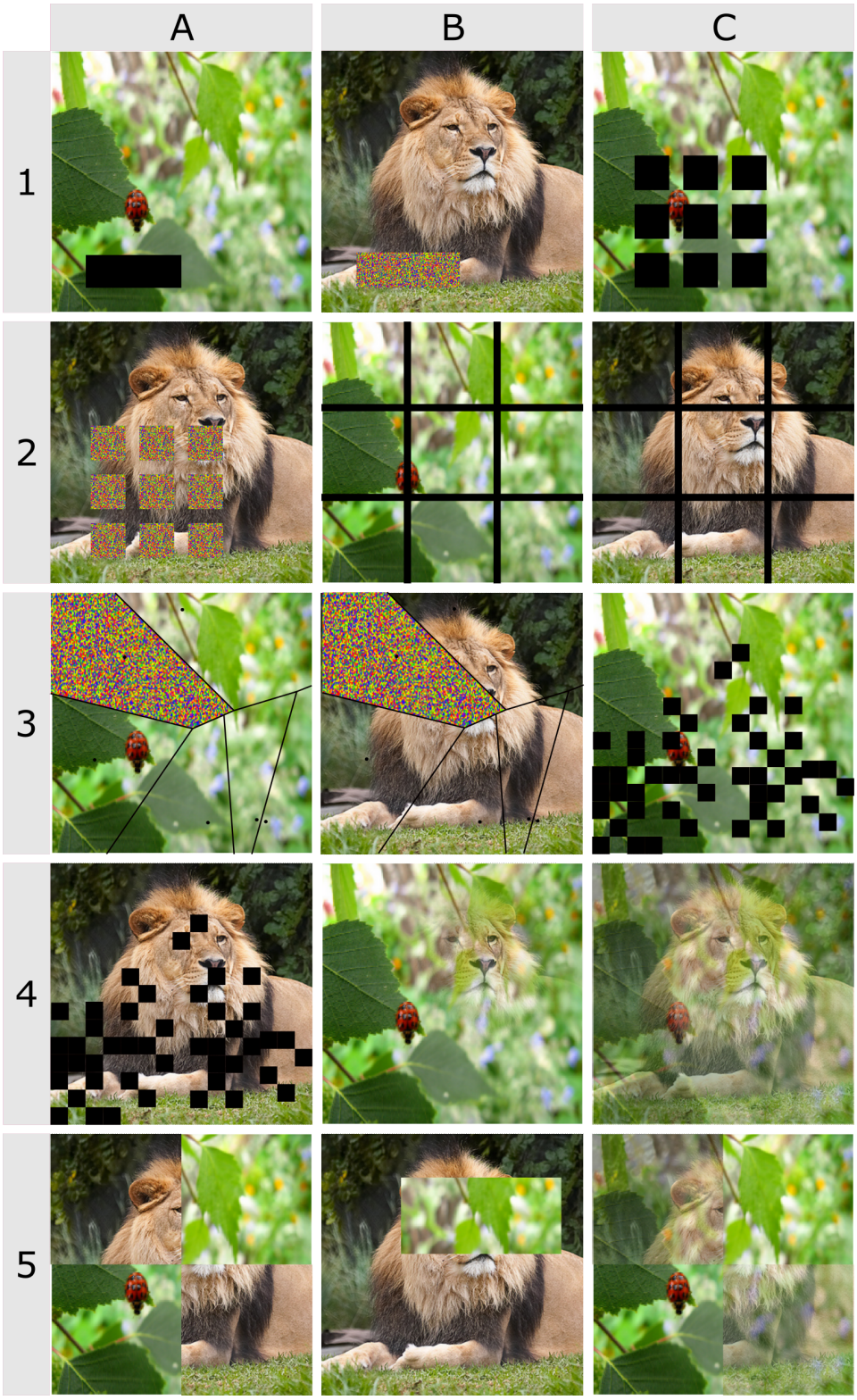


Abbildung 6.2: Kombinierter Überblick über bestehende DA-Methoden: **Zeilen 1–3 + 4A:** Maskierungsmethoden; **4A & B + Zeile 5:** (Lineare) Kombinationsmethoden.

6.2.3 Fragmentierung

Fragmentierung kann als Nebeneffekt von Pixelartefakten auftreten, welche durch den Einsatz von Maskierungen oder von nichtlinearer Rekombination entstehen [136, 227]. Es zeigen sich oft abrupte Übergänge innerhalb eines *Samples* z. B. an den Rändern von Maskierungen oder Rekombinationen. Einerseits können Maskierungs- und Rekombinationsmethoden NN daran hindern, sich im Lernprozess auf „einfache“ *Features* zu fokussieren oder Abkürzungen (*simple solutions*) zu finden. Andererseits können solche künstlich erzeugten Grenzlinien auch hinderlich sein, indem sie die Aufmerksamkeit eines *Models* im Training einfangen. Lee et al. zeigt Beispiele hierfür, indem er den Lernprozess eines NN, das sich auf den Bereich innerhalb solcher künstlichen Grenzen beschränkt, analysiert [136]. *SmoothMix* [136] arbeitet gegen dieses Phänomen an, indem es den Übergang zwischen zwei überblendeten Bildern verwischt. Hierzu werden Alphawertmasken auf zwei Bilder angewendet, welche dann im Anschluss kombiniert werden, was zu weichen (*smooth*) Übergängen führt. Dieser Ansatz kann auch als Addition zu bestehenden Rekombinationsmethoden, wie z. B. *CutMix* Anwendung finden.

Um neue, hinreichend realistische Trainingsbilder zu erstellen, werden durch *Cut, Paste and Learn (CPL)* [52] Objekte auf zufällige Hintergründe „aufgeklebt“, während Objektkanten entweder überblendet oder aufgeweicht werden. Untersuchungen haben gezeigt, dass die resultierenden Bilder zwar für das menschliche Auge fragmentiert erscheinen, allerdings im Vergleich zu manuell zusammengestellten *Samples* zu besseren Trainingsergebnissen führen [52].

Während *SmoothMix* und CPL versuchen, künstlich eingeführte Grenzen zu minimieren, verwendet *MeshCut* [118] hingegen eine gitterförmige Maske, um *Samples* gezielt zu fragmentieren und so mit Absicht solche Grenzen einzuführen (vgl. Abbildung 6.2, 2B & C). Die Idee dahinter ist, dass auf diese Weise ein NN-*Model* im Training Objekte anhand vieler kleinerer Teile kennenlernt und so gezwungen ist, sich auf größere, mehrere Fragmente überspannende *Features* eines Objekts zu konzentrieren. Im Gegensatz zu *SmoothMix* und *CPL* zeigt so *MeshCut*, dass absichtlich erzeugte Grenzen die Modellleistung auch verbessern können.

6.3 Methoden

Basierend auf den in Abschnitt 6.2 vorgestellten Arbeiten und durch die Erfahrung mit DA-Methodenaus Kapitel 4 & 5 können die folgenden Beobachtungen gemacht werden: **1)** orthogonale Kanten (horizontal & vertikal) können von NNs im Training genutzt werden, um „Abkürzungen“ durch einen Datensatz zu lernen. **2)** Künstlich eingeführte, abrupte Grenzlinien wirken sich allerdings nicht grundsätzlich immer schlecht auf die resultierende *Model-Performance* aus. **3)** Voronoi-Diagramme [2] ermöglichen nicht orthogonale Übergänge und

zufällige Formen. **4)** Die Okklusion vieler kleiner Bereiche ist einer großflächigen Verdeckung zu bevorzugen (HaS [217]), da so die Chance steigt, Hauptmerkmale nicht zu verdecken. **5)** Die Synthese neuer *Samples* durch „Aufkleben“ von Objekt-Merkmalen in einem anderen Kontext (CPL) scheint von Vorteil zu sein. **6)** Flächen, welche in Okklusion-basierten Ansätzen verdeckt werden, tragen keine zusätzliche Information mehr, wenden allerdings Rechenkapazität auf.

Als logische Konsequenz dieser Beobachtungen erscheint ein Ansatz, welcher versucht, diese Erkenntnisse zu vereinen. Wünschenswert wären also viele präferiert kleine aber auch große Störungen, die im Gegensatz zu einer Maskierung wie 0-Werte oder GN, Informationen in ihrem natürlichen Kontext neu positionieren. In Kombination mit dem Konzept der nicht orthogonalen Kanten (Voronoi-Diagramm) sowie HaS, wird also ein Ansatz vorgeschlagen, der gleichzeitig eine neue Kategorie von *Data Augmentation*-Methoden darstellt: *Transport*. Dieser Abschnitt soll diese kombinierte Methode (VP) nun einführen, den Evaluationsdatensatz vorstellen und den Versuchsaufbau der Experimente beschreiben.

6.3.1 *Voronoi Patches (VP)*

VP setzt, wie zuvor motiviert, auf die Erweiterung bestehender Trainingsdatensätze durch einen nicht linearen *Feature*-Transport innerhalb eines *Samples*. Zu diesem Zweck wird vorgeschlagen, ein *Sample*, durch eine Voronoi-Diagramms-Partitionierung, in eine Menge konvexer Polygone zu unterteilen. So entstehende Flächen könnten dann zufällig ausgewählt, aus dem Original kopiert und wieder zufällig auf anderen Generatorpunkten platziert (geklebt) werden (vgl. Abbildung 6.3). Als Resultat ist ein neues *Sample* zu erwarten, das (größtenteils) die gleichen Informationen wie das Original enthält, allerdings nicht orthogonale Grenzlinien an vielen Stellen einführt. VP kann Merkmale eines Objekts im

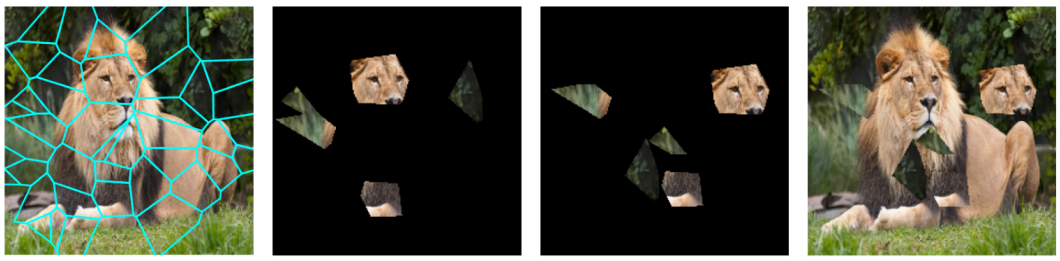


Abbildung 6.3: **Von links:** Beispielbild; das dazugehörige Voronoi-Diagramm (50 Generatorpunkte); 5 zufällig ausgewählte Polygone werden „kopiert“ und an zufällige Orte *transportiert*; das resultierende neue *Sample*.

Original gleichzeitig verdecken und an einen anderen Ort transportieren, so steigt die Chance, dass die Verknüpfung zwischen ursprünglichem *Label* und

Sample erhalten bleibt, wobei Ort, Anzahl und Sichtbarkeit jedes Mal variieren, wenn VP angewendet wird. Dieses gerade umrissene Verfahren wird durch Algorithmus 1 genauer beschrieben. Im weiteren Verlauf dieses Kapitels werden die so entstehenden zusammenhängenden Flächen auch als *Patches* bezeichnet. Wie sich zeigen wird, können sich, aufgrund der möglichen Variationen von Form, Größe und endgültiger Position der Voronoi-Polygone, die ausgewählten und verschobenen Elemente sogar überlappen oder gar nicht bewegen.

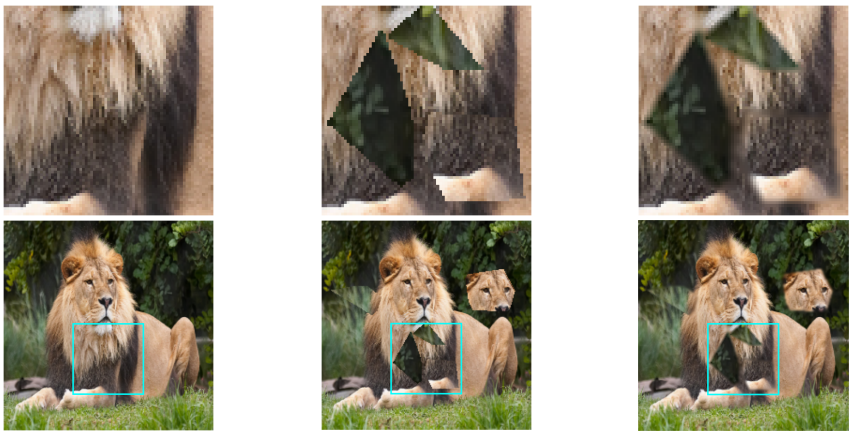


Abbildung 6.4: Darstellung der Übergänge von *Patch* zu Hintergrund; **links:** Originalbild; **mitte:** Transportierter *Patch*; **rechts:** Geglätteter Übergang (**unten** Gesamt; **oben** Ausschnitt).

In seiner initialen Form ist VP mit drei anpassbaren HPs definiert: **1)** *generators* steuert die Anzahl und damit die ungefähre Größe der, durch eine Voronoi-Partition, erzeugten Polygone. Dabei führen wenige Generatorenpunkte zu kleineren *Patches* und umgekehrt. **2)** *patches* bestimmt die Menge der zu transportierenden *Patches*. Dabei ist zu bemerken, dass das Verschieben vieler oder weniger, aber großer *Patches* dazu führen kann, dass zentrale Objekte in einem *Sample* vollständig verdeckt oder nicht mehr identifizierbar sind. Eine Garantie, die Verknüpfung zwischen Hauptmerkmal und *Label* immer aufrechtzuerhalten, gibt es auch durch VP nicht. **3)** *smooth* steuert die Art des Übergangs zwischen den bereits verschobenen *Patches* hin zum Original-*Sample*. Diese können, wie in Abbildung 6.4 dargestellt, entweder mit harten Grenzlinien belassen oder im Anschluss geglättet werden.

Algorithmus 1: Voronoi*Patches* als Pseudocode

```
Input  : sample, generators, patches, smooth
Output : aug
begin:
    aug ← copy of sample;
    //Partition of a plane into regions (polygons p ∈ P),
    //defined by vertices v ∈ V. p:=[vertex v], v:=(x, y)
    polygons ← Voronoi(generators);
    centroids ← [mean(p) in polygons];
    for range(patches) do
        p ← random(polygons);
        c ← random(centroids);
        Move p such that arith.Mean(p) = c;
        for x, y in p do
            aug[x, y] ← sample[x, y];
            if smooth then
                s ← gauss.Filter(aug);
                borders ← calc_border();
                for xy_b in borders do
                    aug[xy_b] ← s[xy_b];
```

6.3.2 Datensatz

Dieser Abschnitt beschreibt in Kürze den für die Evaluation von VP verwendeten Datensatz, welcher auf Basis zweier Faktoren, der Menge an Trainingsbeispielen und der Größe der einzelnen *Sample*, ausgewählt wurde. Durch die Verwendung von *Samples* mit mittlerer bis großer Auflösung soll sichergestellt werden, dass mehr Variationen in den berechneten Partitionen vorhanden sind sowie ausreichend kleine Polygone erzeugt werden können.

Der *2012 ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC)-Datensatz[1], der umgangssprachlich auch als *ImageNet* bezeichnet wird, besteht aus 1,2 Millionen Trainings- und 60.000 Validierungs-*Samples*, die gemäß einer auf WordNet[2] basierenden Klassenhierarchie in 1.000 Klassen kategorisiert und mit *Labels* versehen sind [201]. Während die Auflösung der *Samples* eine hinreichende Größe aufweist, um die Funktionsweise von VP zu evaluieren, ist die Menge an *Samples* im Datensatz ein Problem.

Auf Basis dieser Abwägung fiel die Entscheidung auf den *mixed_10*-Datensatz [57], einer Teilmenge von *ImageNet*. Hier stehen 77.237 Trainings- und 3.000 Test-Daten aus zehn ausgewogenen Klassen zur Verfügung: Hund, Vogel, Insekt, Affe, Auto, Katze, LKW, Obst, Pilz und Boot (vgl. Abbildung 6.5). Da es sich bei *mixed_10* um einen ausgewogenen Datensatz handelt, kann die

[1] http://www.image-net.org/
[2] https://wordnet.princeton.edu/

Klassifizierungsgenauigkeit in % direkt als eine Annäherung an die tatsächliche *Model-Performance* interpretiert werden. Durch die Berechnung von Varianz und Entropie sind aber zusätzliche Einblicke in die Arbeitsweise von VP zu erwarten.

6.4 Experimenteller Aufbau

Wie so oft im DL-Kontext werden alle vorhandenen HP auch hier durch GS für alle Szenarien definiert. Dabei legt ein initialer *educated guess* auf Basis vorhergehender Experimente den Suchbereich jedes HPs fest. Da hier primär die Evaluation von VP im Fokus steht, soll zunächst ein DA trainierte *Baseline* definiert werden, welche dann als Vergleich dienen kann.

Baseline

Mit *SqueezeNet 1.0* wird ein Modell speziell für den Einsatz der *Multi-Class Classification* von Bilddaten, mit Trainingsdaten aus *ImageNet* entwickelt und daher auch für diese Evaluation gewählt. Obwohl dieses *Model* vergleichsweise wenige *Weights* besitzt ($\times 0.5$) [100], erreichte oder übertraf es die Top-1- und Top-5-Genauigkeit [100] des *ILSVRC*-Gewinners 2012, *AlexNet* ($\sim$ 60Millionen) [130]. Die ursprüngliche *Architektur* wird für nachfolgende Experimente lediglich durch das Einführen von BN modifiziert, da dies dem aktuellen Vorgehen beim Training von NN-*Models* entspricht, weil so der Trainingsprozess stabilisiert wird (vgl. Abschnitt 2.3.4.2). Da *Data Augmentation* (bis auf wenige Ausnahmen) *online*, also während des Trainings auf jedes *Sample* einzeln, angewendet wird, ist dies auch die einzige Möglichkeit, die Verteilung des Trainingsdatensatzes konsistent über alle *Models* und DA-Methoden hinweg zu normalisieren. Mit Referenzparametern aus der Literatur ausgestattet, soll durch eine initiale GS eine *Baseline* geschaffen werden, welche eine Vergleichsgrundlage für NN-Training mit und ohne DA liefert.

(a) *Insect* (b) *Car* (c) *Dog*

Abbildung 6.5: Objektgrößen der Klassen im *mixed_10*-Dataset weisen einen großen Unterschied auf. Beispiele der ersten drei Klassen (a – c).

Für das Training wird *Cross Entropy (CE)* als gängiges [241] *Loss* und als Optimierungsalgorithmus ADAM [126] eingesetzt. Letzterer verwendet eine

adaptive LR, was in der Praxis weniger Anpassungen erfordert und schneller konvergieren sollte [199]. Da *SqueezeNet* für *Samples* mit $224x224x3Pixel$ konzipiert wurde [100], wird die Auflösung der *Samples* angepasst und anschließend je (Farb-) Kanal durch Min-Max-Normalisierung in einen Wertebereich von $[0, 1]$ skaliert. [3]. Durch diese Anpassung besitzen die Eingabewerte den gleichen Wertebereich wie die *Models-Weights*, was auch das Lernverhalten stabilisieren kann [3]. Für dieses Basismodell werden sonst keine weiteren Transformationen oder sonstige Anpassungen vorgenommen.

Die Wahl der HP-Suchraum-Grenzen spiegelt die beiden Hauptziele von VP wider und ist durch die Eigenschaften des Datensatzes beeinflusst: 1.) Der Erhalt der Gültigkeit des *Labels* der Trainingsdaten sowie 2.) das zufällige und unregelmäßige Verdecken oder Wiederholen von Objektmerkmalen bzw. des Objektkontexts. Die in Abbildung 6.5 vorgestellte Bandbreite der Stichproben des Datensatzes (im Vergleich zu z. B. *MNIST-Dataset (MNIST)*) erschwert dabei im Besonderen die Auswahl einer optimalen Anzahl von Generatorpunkten, da das Verhältnis von Objekt zu Kontext stark variiert (vgl. Abbildung 6.5a & 6.5c). Auch der durch VP eingeführte Transport verhindert nicht das Überschreiben aller Hauptmerkmale kleiner Objekte, sodass es schwierig abzuschätzen ist, welche *Patch*-Größe benötigt wird oder wie viele *Patches* (d. h. insgesamt bewegte $Pixel^2$) notwendig sind, um einen positiven Einfluss auf die finale Modellleistung zu erzeugen. Durch erste Untersuchungen hat sich gezeigt, dass sich der HP-Suchraum wie folgt gestalten sollte: $generators = \{50, 70, 90\}$ (vgl. Abbildung 6.6), $patches = \{5, 10, 15\}$ & $smoothing = \{true, false\}$.

Abbildung 6.6: Unterschiedliche Patchgrößen durch die Variation des *generators*-HP; **links:** 50; **mittig:** 70; **rechts:** 90.

[3]Scikit-learn/Preprocessing: Min-Max-Scale

6.5 Durchführung und Ergebnisse

In diesem Abschnitt werden die gesammelten Erkenntnisse und Leistungsdaten von *Voronoi Patches (VP)* als DA-Methodeim Vergleich zu einem *Baseline-Model*, sowie vergleichbaren DA-Methoden präsentiert. Auch wird der Einfluss der HP auf den Trainingsprozess und die Gestalt der resultierenden *Patches* untersucht.

Reproduzierbarkeit der Ergebnisse

Die Reproduzierbarkeit der gesammelten Ergebnisse wird durch die Verwendung einheitlicher, zufällig bestimmter *Seeds* ($seeds \in [0, \ldots, 2^{32} - 1]$) für alle Zufallszahlengeneratoren der (parallelen) Trainingsprozesse (während der HP-Optimierung & in den zu vergleichenden Trainingsläufen) gewährleistet. Zur Messung der durchschnittlichen *Performance* werden alle trainierten *Models* und DA-Methoden jeweils mit HPs einer breiten Rastersuche (vgl. Abschnitt 2.3.4.2) ausgestattet. So werden nicht beliebige Algorithmen und *Models* miteinander verglichen, sondern die jeweils besten Kombinationen auf der für die Experimente verfügbaren Hardware. Da diese auch als limitierenden Faktor herauszustellen ist, verläuft das Training jeweils bis zur Konvergenz der *Baseline* (100 Epochen, max. gemessene Genauigkeit 80,1%).

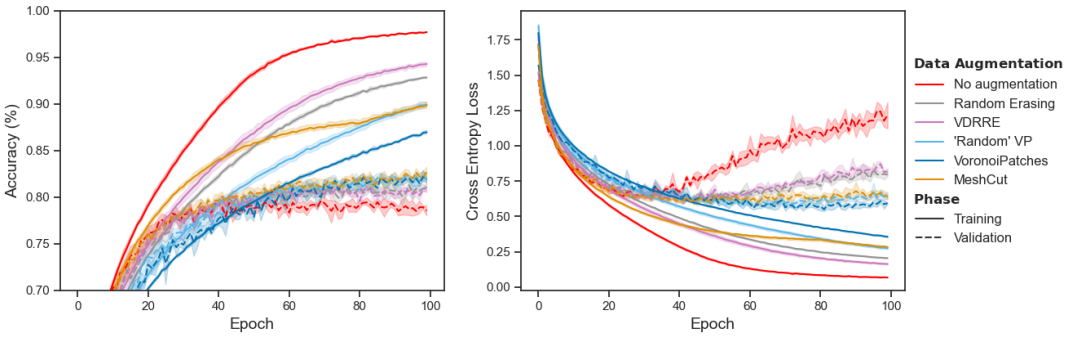


Abbildung 6.7: Durchschnittliche Leistung; **links:** *Genauigkeit* der Klassifizierung von Trainings- und Validierungsdaten über den Trainingsverlauf hinweg (100 Epochen). **rechts:** CE-*Loss* für Trainings- und Validierungsdaten. Die Unterschiede im *Overfitting* sind sichtbar

Gemessene Ergebnisse

Grundsätzlich kann für alle während des Trainings variierten VP-HP Kombinationen eine Leistungsverbesserung gegenüber der *Baseline*, im Bereich von 1,3-3,5% (vgl. Tabelle 6.1) festgestellt werden. Obwohl sich die Messungen durchweg recht ähnlich verhalten, variiert die Leistung der einzelnen Klassen erwartungsgemäß. Überraschenderweise sind *Car* und *Truck* mit einem Leistungsdefizit von 10-30% für alle *Models* am schwierigsten zu unterscheiden. Aus

Tabelle 6.1: Gemessene Leistungsdaten im Vergleich; VP als DA-Methode.

	accuracy	*macro-recall*	*macro-precision*
Baseline	80.9%	80.9%	81.0%
Voronoi Patches	83.3%	83.5%	83.3%
random-RE	82.7%	82.7%	82.9%
black-RE	82.6%	82.6%	83.0%
MeshCut	83.8%	83.8%	83.8%
VDRRE	82.5%	82.5%	82.7%

diesem, auch in Abbildung 6.8 dargestellten, Grund steht *ImageNet* derzeit in der Kritik, mit zu stark abweichenden *Labels* (*noisy*) ausgestattet zu sein [21][4].

Abbildung 6.8: Klassifizierungsanomalien: Als Krankenwagen genutzte Transporter sind als *Car* markiert (**unten links**), von der Polizei genutzte Transporter jedoch als *Truck* (**mittig**). Polizei-*Combi* mit *Truck* als *Label* (**unten rechts**).

Hauptergebnisse

1) Abbildung 6.9 zeigt die gemessene Varianz für alle untersuchten DA-Methoden. Es ist festzustellen, dass über alle Trainingsläufe hinweg, der Einsatz von VP zu *Models* mit der geringsten Varianz führt. Das bedeutet, dass ein so trainiertes NN-*Model*, sich in seiner Anwendung stabiler verhält, als ein vergleichbares, welches auf nicht gestörten Trainingsdaten trainiert wurde.
2) Zudem ist in Abbildung 6.7 ein weiterer interessanter Aspekt zu beobachten: Bei allen in diesem Experiment trainierten *Models* (mit und ohne DA) kam es wohl zu einer mal mehr, mal weniger starken Ausprägung von *Overfitting*.

[4]Dies stellte sich erst im Laufe der Experimente heraus

VP allerdings, zeigte unter allen Trainingsläufen das geringste Ausmaß dieser Überanpassung an die Trainingsdaten. Dies lässt sich am wachsenden Abstand des CE-*Loss* in Abbildung 6.7 (rechts) erkennen.

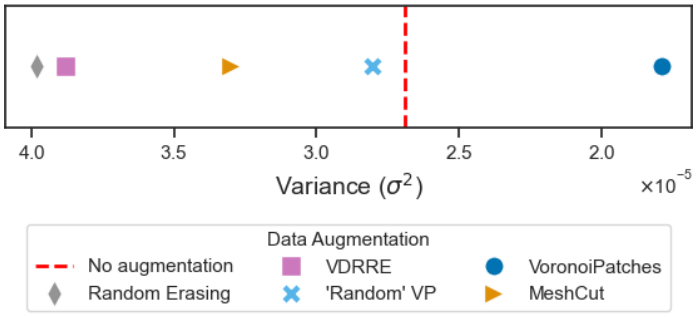


Abbildung 6.9: Durchschnittliche Varianz auf dem Testdatensatz (std) für, durch verschiedene DA-Methoden gestörte Trainingsdaten.

Weiche Kanten

Entgegen der gängigen Literatur im Forschungsfeld der DA-Methodekann keine eindeutige Verbesserung der *Model-Performance* bei einer Anwendung von VP mit glatten gegenüber abrupten Übergängen ($smooth = false$) festgestellt werden. Dies kann mehrere Ursachen haben: Zum einen kann entweder die Breite des Übergangsbereichs nicht optimal gewählt sein oder die Pixelartefakte haben einen positiven Effekt (vgl. Jiang et al.[118]). In zukünftigen Arbeiten sollte dies als weiterer HP eingeführt und entsprechend untersucht werden.

Vergleich mit VDRRE

Um die Gültigkeit und den Nutzen der Anwendung von VP zu verifizieren, erfolgt auch ein Vergleich mit RE & VDRRE. Für RE kommen dabei zwei mögliche *Pixel*-Werte in Erwägung [256] *schwarz* (0) und *random* (i. i. d.[0,1)). Abbildung 6.7 zeigt eine durchschnittliche Verbesserung der Genauigkeit von VP gegenüber RE von 0.6 %-0.7 %. Die VDRRE-Parameter werden hingegen entsprechend den Literaturvorschlägen gesetzt [2]. Abayomi-Alli et al. berechnet sechs Regionen (*Patches*), von denen eine zufällig mit GN gefüllt wird (Okklusion). Dabei zeigt sich, dass die gemessene Leistung von VDRRE mit RE vergleichbar ist. Inspiriert durch die Okklusion mittels GN wurde eine zusätzliche VP-Version mit zufällig gefüllten Feldern getestet. Dieser Ansatz ist direkt mit VDRRE vergleichbar, setzt aber auf viele, kleinere und verteilte *Patches*. In diesen Experimenten erzielt *random*-VP bessere Ergebnisse als VDRRE und RE, wobei es so an VP herankommt. *MeshCut* (DA-*Schedule* & HPs nach Jiang et al.) hingegen verhält sich auf dem hier gewählten Datensatz schlechter als *Random* VP.

Entropie-Analyse zu VP

Für weitere Einblicke in die Arbeitsweise von VP kann die eingebrachte Störung im Kontext der Trainingsdaten gemessen werden. Eine Möglichkeit hierfür, ist die Messung der Entropie H [83], welche die Wahrscheinlichkeitsverteilung jedes Bildes x (mit $P := prob.$, $K := classes$) vor und nach der Anwendung von VP einbezieht:

$$H(x) = -\sum_{k \in K} P(k) * \log(P(k)) \tag{6.1}$$

Bei einer Wahrscheinlichkeitsverteilung von 10 Werten (für zehn Klassen) werden so Werte im Bereich von $0.00 - 3.32$ (obere, bzw. untere Grenze) gemessen, wobei der untere Grenzwert einer perfekten Klassifizierung und der obere Grenzwert die perfekt ausgeglichene Wahrscheinlichkeit über alle Klassen ausdrückt (= schlechtes Klassifizierungsergebnis). Abbildung 6.10 zeigt die aggregierten Entropie-Mittelwerte, sodass der Einfluss verschiedener DA-Methodendirekt verglichen werden kann [83]. Es ist festzustellen, dass, je mehr Daten in einem *Sample* durch VP transportiert werden, desto größer die zu erwartende Unsicherheit ist. Diese Störgröße kann durch eine Anpassung der *Patch*-Flächen (HP) in einen Stör-Bereich zwischen RE mit entweder *schwarzen* oder *random Pixel* gebracht werden. Die eher niedrige Entropie von RE mit *random Pixel* kann ein Indiz für artifiziell eingeführte Artefakte und die implizite Verwechslungsgefahr durch latente Muster in GN sein. Es sei noch darauf hingewiesen, dass auch die örtliche Verteilung der initialen Generatorenpunkte, als eine stochastische Komponente, Einfluss auf die Varianz über alle *Patch*-Größen ausübt.

Weiterführende Analysen zu VP

Wie beschrieben, korreliert die Anzahl der Generatorenpunkte direkt mit der durchschnittlichen Größe der entstehenden *Patches*. So ergibt die Ermittlung der durchschnittlichen $Pixel^2$ (2D-Fläche) 932, 673 bzw. 528 $Pixel^2$ für jeweils 50, 70 & 90 Generatorenpunkte (vgl. Abbildung 6.6). Es ist festzustellen, dass die Verwendung entweder vieler kleiner Patches oder weniger großer Patches einen positiven Einfluss auf die NN-*Model* Leistung im Kontext des verwendeten Datensatzes hat (5 vs. 15 *Patches*). Die beste Leistung wird mit 10.095 $Pixel^2$ ($Generatoren = 70, Patches = 15, smooth = False$) erreicht, was im Durchschnitt einer Neukombination von 20.1 % der Bilddaten (pro *Sample*) entspricht. Werden durch VP pro *Sample* mehr als ca. 10.000 $Pixel^2$ transportiert, kommt es hingegen zu einem *Performance*-Verlust. Der Transport vieler kleinerer *Patches* durch VP wirkt sich aber besser auf die *Performance* aus, als die Maskierung eines großen zusammenhängenden Rechtecks durch RE. Zum Vergleich entspricht die durchschnittliche Größe der durch RE[5] erzeugten, Maskierung (Rechteck) einer Fläche von 10.176 $Pixel^2$ und damit

[5]HP nach Zhong et al.

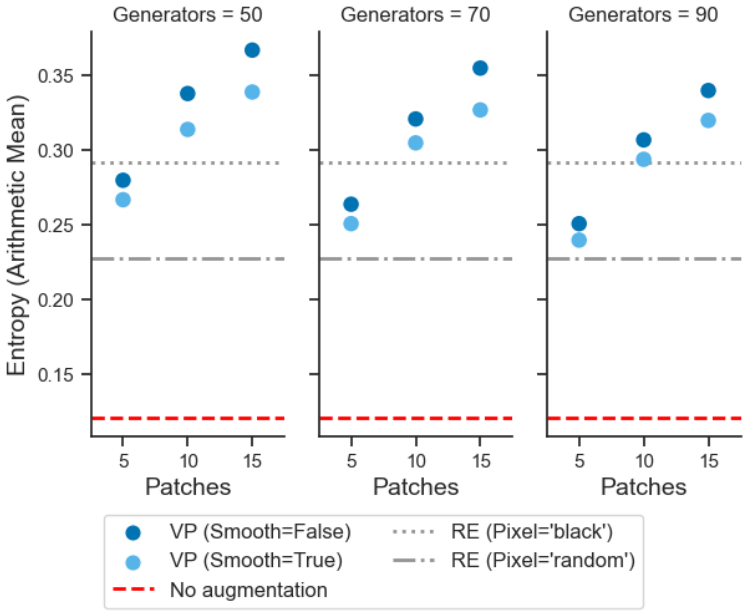


Abbildung 6.10: Messung der durchschnittlich eingebrachten Entropie H (vgl. Gleichung 6.1) durch VP, RE und nicht veränderten *Samples*.

durchschnittlich 20.3 % pro *Sample*.

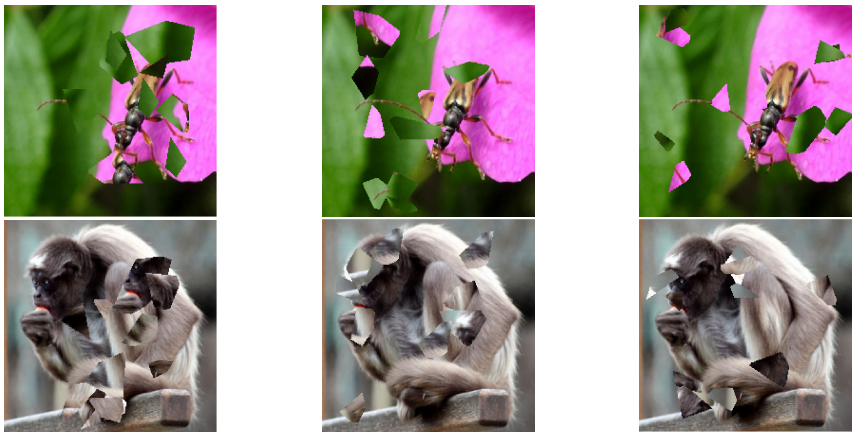


Abbildung 6.11: Visueller Vergleich verschiedener Patchgrößen (**von links:**) 50, 70, 90 Generatorenpunkte). Es wurden immer 10 *Patches* transportiert, die Kanten sind geglättet.

Zusätzlich kann neben der *Patch*-Fläche auch die *Structural SIMilarity Index (SSIM)* zwischen Original- und durch VP gestörten Bildern berechnet werden. SSIM wurde für die Messung der Bildqualität ([0-1]) durch den Vergleich von Luminanz, Kontrast und Strukturen zweier Bilder entwickelt. Ein Ergebnis von 1.0 drückt dabei eine perfekte Übereinstimmung aus. [243] Insgesamt wurden die SSIM-Werte von 1.000 *Samples* zunächst berechnet und dann aggregiert (arithmetisches Mittel). Dabei zeigt sich, dass, wenn die Anzahl der Patches

erhöht oder die Anzahl der Generatoren verringert wird, die Ähnlichkeit von Original- und augmentiertem Bild abnimmt. Dies ist in Abbildung 6.12 für verschiedene HP-Konfigurationen dargestellt. Vergleicht man diese Messungen, dann kann die Beziehung zwischen SSIM und Entropie als annähernd linear bezeichnet werden, da mit der strukturellen Differenz (SSIM) in den durch VP beeinflussten *Samples* auch die Entropie der *Model-Prediction* zunimmt.

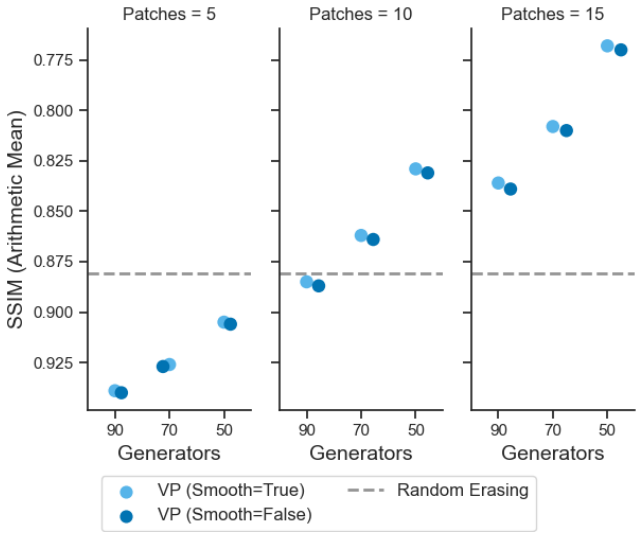


Abbildung 6.12: Durchschnittliche Strukturveränderung (SSIM) nach Anwendung von VP und RE.

6.6 VP als DA-Methode auf Mel-Spektrogrammen

Wie in Kapitel 4 & 5 erörtert wurde, bietet sich zum Zwecke der Sequenzverarbeitung durch neuronale Netzwerke das Format der Mel-Spektrogramme als alternative Repräsentation für Audiodaten an. Im Kontext dieses Kapitels stellt sich nun die Frage, inwieweit sich VP auch für die Anwendung als DA-Methode dieses Datenformats eignet. Um dies zu untersuchen, werden im Folgenden entsprechende Experimente durchgeführt.

6.6.1 Visuelle Analyse

Abbildung 6.13 zeigt (rechts unverändert) den Einfluss von VP auf ausgewählte Mel-Spektrogramme des PRS-Datensatzes. Zunächst bewegte und dann eingesetzte *Patches* sind deutlich rot markiert, dabei ist zu erkennen, dass lokale Strukturen vollkommen aus ihrem Kontext gerissen werden. So wird durch VP nicht nur Information innerhalb eines benachbarten Mel-Bandes, was

einer temporalen Modulation entsprechen würde, bewegt. *Patches* können auf anderen Bändern landen, was einer Frequenz-Modulation entsprechen würde. Die gleichzeitige Veränderung in Zeit- und Frequenzdimension stellt sich als deutliche Störung der Trainingsdaten heraus. Dabei scheint die Größe der resultierenden *Patches* keine Rolle zu spielen.

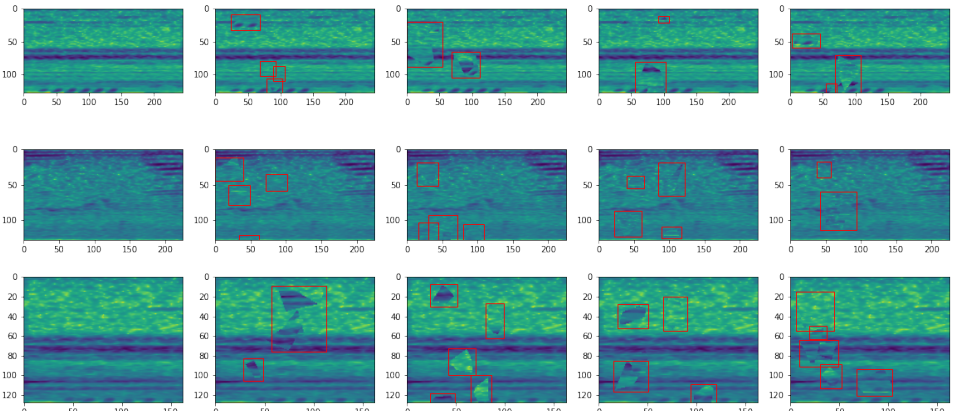


Abbildung 6.13: VP auf Mel-Spektrogrammen: **links:** Originales Spektrogramm. **nach rechts:** Anwendung des VP-Algorithmus mit verschiedenen Generatoren (Anzahl und bewegte).

6.6.2 Experimentelle Ergebnisse

Zusätzlich zu der visuellen Analyse in Abschnitt 6.6.1 werden auch weitere bekannte *Models* (CC, SSC, ViT & VViT) aus Kapitel 4 & 5 jeweils auf dem PRS-Datensatz trainiert. Wie zuvor auch, wird je *Model* durch TPE eine HP Suche angestoßen, um vergleichbare *Models* zu trainieren. Da die neuesten Ergebnisse für den PRS-Datensatz vorlagen und VP direkt in die vorhandene Pipeline eingebaut werden konnte, wurde dieser Datensatz für diesen ersten Überblick bevorzugt. Wie Tabelle 6.2 zeigt, stellte sich ein Leistungszuwachs im Vergleich zu den in Abschnitt 5.6 gewonnenen Ergebnissen ein. Die einzige Ausnahme bildet VViT, hier ist auf dem PRS-Test-Datensatz keine Steigerung festzustellen. Dies könnte mit der veränderten Art der Mel-Spektrogramm-Unterteilung zusammenhängen, die es dem *Attention*-basierten *Model* nicht ermöglicht, Artefakte leicht auszublenden, da Bruchlinien und damit Artefakte eher erkannt werden als durch die quadratischen *Convolution-Kernel* von CC, SSC oder auch ViT.

Insgesamt sind diese Ergebnisse erste Anhaltspunkte, dass sich trotz des gesteigerten Rechenaufwands für VP das Einbinden dieser DA-Methodelohnen könnte. Zum aktuellen Zeitpunkt und aufgrund der begrenzten Experimente kann allerdings noch keine sichere Aussage getroffen werden. Abbildung 6.14 könnte den zuvor geäußerten Verdacht untermauern. Für die hier trainierten

Tabelle 6.2: VP auf Mel-Spektrogrammen trainiert auf dem PRS-Datensatz

Model	CC		SSC		ViT		VViT	
VP	✓	✗	✓	✗	✓	✗	✓	✗
Score	**0.880**	—	**0.896**	0.842	**0.896**	0.883	0.850	**0.872**

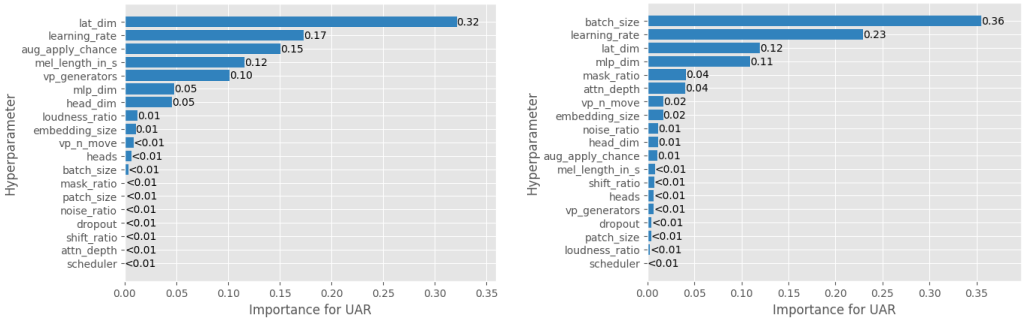


Abbildung 6.14: Parameter Einfluss während der HP-Suche **links:** ViT; **rechts:** VViT

ViT-*Models* (links) scheint der HP *vp_generators*, also die Anzahl an Generatorpunkten, zusammen mit der *aug_apply_chance* eine hohe Bedeutung zu haben. Die hier trainierten VViT scheinen im Gegenteil wenig Einfluss durch die vorhandenen DA-Methodenzu erfahren (geringe *aug_apply_chance*). Als Ausnahme ist die *mask_ratio* zu erwähnen, die ganze Mel-Bänder und diskrete Zeitabschnitte schwarz (== 0) färbt.

6.7 Diskussion und Zusammenfassung

In diesem Kapitel sollte untersucht werden, wie sich die Rekombination von Information innerhalb eines Trainings*Sample* auf den Trainingsprozess an sich, sowie die resultierende *Performance* des NN-*Models* auswirkt. Hierzu wurde *Voronoi Patches (VP)* eingeführt, eine DA-Methode und -Kategorie (Transport), um *Overfitting* für CNNs zu reduzieren. Ein weiteres Ziel war die Minimierung des Informationsverlusts durch den Transport von nicht orthogonalen *Patches* als Methode der *Data Augmentation*.

Hierbei stellte sich heraus, dass VP vergleichbare DA-Methoden hinsichtlich der Modellvarianz bei Vorhersagen übertrifft und der Tendenz zur Überanpassung entgegensteuert. Eine generelle Steigerung der *Performance* konnte hingegen nicht erreicht werden. In zusätzlichen Experimenten wurde zudem der Einfluss der neu eingeführten HP weiter analysiert. Dabei wurde z. B. gezeigt, welche Entropie VP im Vergleich zu anderen, vergleichbaren DA-Methodeeinträgt und wie sich einzelne Parameter auf den Trainingsprozess auswirken.

Im Anschluss wurde auch ein Blick auf vorhergegangene Kapitel geworfen, in denen erste Experimente hinsichtlich der Anwendbarkeit von VP auf Mel-Spektrogramm basierte Audio-Klassifizierung stattfanden. Noch lässt sich hier keine genaue Aussage treffen, erste Ergebnisse lassen aber hoffen.

Verbesserungsmöglichkeiten und offene Fragen im Kontext von Transport basierten DA-Methoden umfassen die Optimierung fließender (*smooth*) Übergänge sowie die Untersuchung verschiedener *Pixel*-Werte oder *Patch*-Formen.

7 *Self-Replicating Neural Networks* und deren Anwendung

Nachdem sich die vorhergehenden Kapitel mit typischen Anwendungsfällen von NNs beschäftigt haben, soll in diesem letzten Kapitel über den sprichwörtlichen Tellerrand hinausgeblickt werden. Im Rahmen dieser Arbeit waren sequenzielle Daten bisher lediglich als Folge von Werten innerhalb eines *Samples* in einem unveränderlichen oder stationären Datensatz definiert. Dies soll an dieser Stelle aufgeweicht werden, um eine andere Form des sequenziellen Lernens durch NN zu ermöglichen.

Anhand *Self-Replicating Neural Network (SRNN)* soll gezeigt werden, wie NN ihre eigenen nichtstationären und sequenziell manipulierbaren Trainingsdaten nutzen können, nicht nur, um eine Form der Selbstreplikation [68, 69] zu ermöglichen, sondern auch, um als eine Menge von Individuen (in einer größeren Struktur) eine höhere Aufgabe zu lösen. Die durch SRNN approximierte Funktion wird hierzu als „sich wie ein NN-*Weight* verhalten“ interpretiert. Dabei stellt sich heraus, dass diese Art des Optimierungsprozesses zu einer impliziten Reduktion unwichtiger Kanten (*Natural Pruning*) führt.

Contribution 7.0.1: *Organism Network*-basiertes Natural Pruning

Beiträge zur Wissenschaft in diesem Kapitel umfassen die Erweiterung des Baukastens der SRNN um eine sinnvolle und „gemeinsame“ Aufgabe innerhalb einer festen Struktur des *Organism Networks (ONs)*. Den Erhalt des SRNN-Interfaces und damit der Operatoren aus dem Baukasten des *artificial chemistry of life system*, sowie die Entdeckung des Verfahrens zum *Natural Pruning* von *Neural Network (NN)-Weights*. Vorveröffentlichung durch Illium et al. in „Constructing Organism Networks from Collaborative Self-Replicators“ [103], mit Einflüssen aus „Self-Replication in Neural Networks“ [68] & „Goals for Self-Replicating Neural Networks“ [69].

Die Struktur dieses Kapitels gliedert sich wie folgt: Abschnitt 7.3 gibt eine Zusammenfassung des relevanten Hintergrunds auf dem Gebiet der *Self-Replicating Neural Network*, bevor das Konzept und die *Architektur* von *Organism Networks* in Abschnitt 7.4 vorgestellt werden. Experimente mit verschiedenen *Auxiliary Tasks* unterschiedlicher Komplexität und die Untersuchung der dar-

aus resultierenden Leistungen finden sich in Abschnitt 7.5 & 7.6. Abschließend werden in Abschnitt 7.2 verwandte Arbeiten erörtert und in Abschnitt 7.7 maßgebliche Erkenntnisse zusammengefasst.

7.1 Einleitung

Self-Replication (SR) wird seit jeher als einer der zentralen Aspekte des Lebens angesehen [122]. Daher erscheint es nicht als verwunderlich, dass auch in der jüngeren Literatur ein starkes Interesse an diesem Themenfeld zu erkennen ist. Die Dominanz der *Neural Networks (NNs)* in vielen informationstechnischen Disziplinen hat die Frage aufkommen lassen, ob solch ein Mechanismus, nach natürlichem Vorbild, wohl einen möglichen positiven Einfluss auf Lernverhalten, *Generalization* oder *Robustness* bedingen würde. Eine andere Motivation möchte hingegen einfach nur Entitäten erschaffen, die sich näher an natürlichen Vorbildern und Prozessen orientieren, um, aber nicht nur, Beschränkungen, welche den derzeit verwendeten *Architekturen* und deren Trainingsmethoden innewohnen, mithilfe von SR umgehen zu können. Verbesserungspotenzial liegt beispielsweise in der Verringerung der Volatilität (Schwankung/Unsicherheit) im Trainingsprozess oder der Schaffung neuer Möglichkeiten des Informationsaustauschs zwischen einzelnen oder Gruppen von NNs. [33, 69, 192]

Zwar wurden bereits *Self-Replicating Neural Networks (SRNNs)*-Varianten mit zusätzlichen Aufgaben (*Auxiliary Task (AT)*) ausgestattet, wodurch diese fähig sind, mehr als nur *Self-Replication* zu erlernen (vgl. [33, 69, 192]). In all diesen Fällen sind die Experimente jedoch so konstruiert, dass SRNNs diese ATs allein oder mit indirekter Unterstützung ihrer Nachbarn (andere SRNNs in einem geteilten „Lebensraum“) während des Trainings ausführt [69].

In diesem Kapitel soll die Nützlichkeit von SRNNs weiter untersucht werden, indem auf den bestehenden Vorarbeiten [67–69] aufgebaut wird. Individuelle *Self-Replicating Neural Network*s sollen nicht nur als benachbarte Individuen, sondern als wertvolle Mitglieder einer Gruppe gesehen werden, die fähig sind, zusammen eine *Auxiliary Task* zu lösen. Hierzu wird im Folgenden eine Struktur (*Organism Network (ON)*) definiert, welche von außen, wie ein gewöhnliches NN erscheint, jedoch im Kern aus verschiedenen untergeordneten Einheiten (*Particle Network (PN)*) aufgebaut ist. Auch werden durch diesen gestaffelten Aufbau die biologisch inspirierte Metapher rund um die Entwicklung aus dem Bereich der *Self-Replicating Neural Network*s, wie die Zellen in einem mehrzelligen biologischen Organismus, weiterentwickelt.

Da der *Weight*-Space des *Organism Network* mit allen PNs aktuell noch stark anwächst und die gemessene *Performance* des ON (noch) nicht mit NNs vergleichbarer Größe mithalten kann, wird in diesem Kapitel ON zunächst eingeführt und in verschiedenen Experimenten evaluiert.

7.2 Verwandte Arbeiten

SR als Forschungsgebiet im Kontext von NNs fand in den vergangenen Jahren zunehmend Beachtung. Chang und Lipson etablierten selbstreplizierende NNs, sog. *NN-quine*, welche die Autoren am *MNIST-Dataset (MNIST)* evaluierten. Diese scheinen allerdings im Verlauf des Trainings ihren „Anreiz“ zu verlieren, die SR-Fähigkeit zu erlangen oder zu erhalten [33]. Der Ansatz nach Gabor et al. verfolgt dagegen den Weg der parallelen Optimierung auf beiden (SR und AT) Aufgaben und erzielt damit ein gegenteiliges Ergebnis [67–69]. Die Autoren Randazzo, Versari und Mordvintsev entwickelten ein Verfahren, bei dem sie Variationen von (durch *Backpropagation (BP)* erzeugten) *Weights*, durch SR auf *fertile* Agenten übertragen. Das Training der SR-Eigenschaft wird dort durch die Einführung eines sog. *sink-loss* erreicht.

Die Einbettung von SRNNs in größere Strukturen scheint bisher nicht erfolgt zu sein. Die Idee, NN-*Neurons* durch wieder kleinere NNs abzubilden, wurde hingegen bereits vorgestellt [32], wobei sich die Autoren auf das Thema des *kontinuierlichen Lernens* konzentrierten. Die von Camp, Mandivarapu und Estrada verwendeten NNs teilen sich einen auf einer Metaebene vortrainierten *Weights*, wodurch der Effekt des *Catastrophic Forgettings* (Verlernen von linearen Regressionsaufgaben) minimiert wird. Der Schwerpunkt dieses Ansatzes liegt allerdings auf der *Plastizität* dieser vektorisierten Synapsen und nicht auf den *Neurons* selbst.

Ba et al. präsentieren die Vorteile der Verwendung von zwei *Weight*-Partitionen mit neuartigen *schnellen Neurons* (sog. *fast weights*) für eine (laut Aussagen der Autoren) verbesserte Informations-Speicher-Kapazität. Hurtado, Raymond und Soto hingegen stellt spezifisch geteilte *Weights* als gemeinsame Wissensbasis zwischen Aufgaben von Agenten vor. Im Gegensatz dazu prüft dieses Kapitel nicht, wie gut oder lang sich NNs an Aufgaben erinnern können, sondern zielt darauf ab, von Anfang an die erlernte Zuordnung so zu erweitern, dass mehrere Aufgaben von denselben *Weights* abgedeckt werden können (*multi-purpose*). Die partielle Fixierung von *Neurons* (PN_{SR} in ON) wurde bisher weder in diesem Kapitel noch in den Vorveröffentlichungen untersucht. Beniaguev, Segev und London verwendet kleine Sub-*Neurons* um zu demonstrieren, dass (selbst kleine) mehrschichtige NN ausreichen, um die Funktionalität echter *Neurons* der Hirnrinde (Kortex) zu imitieren.

Das in Abschnitt 7.6 gezeigte *Dropout*-Verfahren ist vergleichbar mit dem Konzept der Auswahl wichtiger NN-*Weights* im Kontext des NN-*Pruning*. Das Ziel von *Pruning* ist es, Trainings- und Ausführungskosten immer größer werdender NN zu senken, dabei müssen Ansätze gefunden werden, welche die Identifizierung und Entfernung unwichtiger Teile eines Netzwerks, z. B. durch eine Null-Maske, ermöglichen.

Einen umfassenden Überblick hierzu findet sich in [23]. *Pruning* ist von speziellem Interesse für die NN-Anwendungen, da einerseits angenommen wird,

dass aktuelle NN-*Architekturen* anfällig für Überparametrisierung sind [45] und andererseits die aktuellen Forschungsergebnisse nahelegen, dass in jedem zufällig initialisierten und voll vermaschten *Feed Forward Neural Network (FFNN)* Teilmengen von *Hyperparameter (HP)*-Konfigurationen mit (mindestens) gleicher *Performance* (im Vergleich zum ursprünglichen NN) existieren [62]. Die durch *Pruning* eingesparten *Weights* können unter anderem Speicherverfahren großer NNs unterstützen [88, 173], was es zu einer nützlichen Ergänzung moderner Trainingsverfahren macht.

Der im Folgenden vorgestellte Ansatz, kleine, zufällig initialisierte Netzwerke in einer größeren Meta-Struktur zur Vorhersage von Kantengewichten zu verwenden, zeigt Ähnlichkeit mit einem Verfahren, welches das zu erwartende innere Produkt der *Neurons* nach der Evaluation weniger *Inputs* liefert [7]. Lin, Chen und Yan verwendet ebenfalls ein *Micronetzwerk* als Funktionsapproximator zwischen NN-*Layer*, allerdings im Kontext einer *Convolutional Neural Network (CNN)-Architektur*.

Zu den sonstigen *Pruning*-Ansätzen gehören das *Variational Dropout (VD)*, zur Erzeugung schmalerer NNs (im Sinne von weniger *Neurons* pro *Layer*) [164], und Variationen des *Magnitude-Pruning*, um entfernbare *Weights* über oder unter einem bestimmten Schwellenwert auszuwählen ([62, 89, 173, 257]).

Das in diesem Kapitel präsentierte Verfahren zum Identifizieren und *prunen* (entfernen) überschüssiger *Weights* durch einen Spezialisierungs-Vorgang könnte sich als Gegenstück zu einfachen Schwellenwert-*Pruning* bilden. Der Unterschied liegt darin, dass durch ONs die Trennung als Nebenprodukt des Lernprozesses entsteht, anstatt, dass eine harte Grenzlinie empirisch festgelegt wird. Dies sollte in Zukunft weiter untersucht werden.

7.3 Methodische Grundlagen

Nach der Vorstellung ähnlicher bzw. relevanter Arbeiten soll in diesem Abschnitt nun der benötigte Hintergrund zur Etablierung von *Self-Replicating Neural Network* und *Organism Network* geschaffen werden.

7.3.1 *Self-Replicating Neural Networks*

Zunächst werden die bestehenden Definitionen im Kontext des SRNN weitestgehend übernommen und alle relevanten Grundlagen kurz zusammengefasst. Diese sind bereits veröffentlicht [67–69] und bilden eine wichtige Basis dieses Kapitels.

Eine allgemeine Einführung in die Struktur und Funktionsweise sowie das Training von NNs findet sich in 2.3. Arbeiten, welche den (nötigen) Formalismus zu NNs einführen, sind zitiert [67–69, 103, 131]. Um SR zu ermöglichen,

definieren Gabor et al. das verwendete NN, sodass es eine Zuordnung lernen kann, welche (korrekt abgefragt) alle internen *Weights* preisgibt. Die in diesem Abschnitt verwendeten FFNN-*Architekturen*, sind fixiert und werden im Training nicht verändert, daher kann die jeweilige Funktion lediglich anhand des zusammengeführten Gewichtsvektors abgeleitet werden. Da der Gewichtsvektor eines beliebigen NN immer mindestens so viele *Weights* enthält, wie die Summe der Vektorelemente V in *Input* & *Output* $|V| \leq |X| \cdot |Y|$, gestaltet es sich schwierig, ein NN zu definieren, welches alle *Weights* als Produkt einer singulären Operation (in einem einzigen Aktivierungsschritt) ausgibt. Daher ist es nötig, ein *Interface* zu definieren, welches die internen *Weights* nach einem klar erlernbaren Schema abrufen kann. Gabor et al. definieren eine lernfähige Codierung, welche es ermöglicht, *Weight*-Position (in Bezug auf *Layer*, *Neuron*, *Weight*) in NN zu adressieren. Auf dieser Basis kann ein NN das entsprechende *Weight* approximieren [67]. Diese *Weightwise Reduction (WWR)* genannte Operation ist in „Self-replication in neural networks“ zum ersten Mal definiert. Dieser Abschnitt beschreibt die sog. *Weightwise Application (WWA)*, welche die Fragestellungen in diesem Kapitel und auch die Interaktionen zwischen NNs (hier als PN referenziert) ermöglicht.

Die Anwendung (eng. Application) eines *Neural Network* auf sich selbst oder ein anderes durch WWR reduziertes NN ist wie folgt definiert:

Definition 7.3.1: ***Weightwise Application***

$\mathcal{M}, \mathcal{N}, \mathcal{O} : \mathbb{R}^4 \to \mathbb{R}$ seien NNs. Die *Weights* von $\mathcal{M}$ seien $\overline{\mathcal{M}} = \langle v_i \rangle_{0 \leq i < |\overline{\mathcal{M}}|}$. Ein NN $\mathcal{O}$ ist das Ergebnis der gewichtsweisen Anwendung von $\mathcal{N}$ auf $\mathcal{M}$, geschrieben $\mathcal{O} = \mathcal{N} \lhd_{ww} \mathcal{M}$, gdw.

$$\overline{\mathcal{O}} = \langle \mathcal{N}(v_i, L(i), C(i), E(i)) \rangle_{0 \leq i < |\overline{\mathcal{M}}|}$$

wobei $L(i)$ den zugeordneten *Layer*, $C(i)$ die zugeordnete *Zelle* und $E(i)$ die zugeordnete Position eines Kantengewichts i definiert.

Gabor et al. definiert SR als die Fähigkeit eines NN, die eigenen *Weights* durch WWA auf sich selbst zu reproduzieren. Dies wird auch als (*weightwise*) *Self-Application (SA)* bezeichnet [67]. Zwar können *Weights* durch die Grenzen der *Float*-Operationen nicht exakt reproduziert werden, doch es kann ein Abstand ε definiert werden, bis zu dem ein durch WWA erzeugtes *Weight* als „reproduziert“ angenommen werden kann. Sobald ein NN hinreichend genaue SR erreichen kann, wird es daher als ε- Replikator[1] bezeichnet [67].

[1] Vormals als „Fixpunkt“ bezeichnet, wird das tiefere Konzept in dieser Arbeit nicht verwendet und nicht eingeführt, kann aber in „Self-Replication in Neural Networks“ nachvollzogen werden [67–69].

Definition 7.3.2: ε-Replikator

Gegeben sei $\mathcal{N}$ mit *Weights* $\overline{\mathcal{N}} = \langle v_i \rangle_{0 \leq i < |\overline{\mathcal{N}}|}$. $\varepsilon \in \mathbb{R}$ sei die erlaubte Fehlerspanne. $\mathcal{N}' = \mathcal{N} \lhd_{ww} \mathcal{N}$ sei die Selbstanwendung von $\mathcal{N}$ mit *Weights* $\overline{\mathcal{N}'} = \langle w_i \rangle_{0 \leq i < |\overline{\mathcal{N}'}|}$. $\mathcal{N}$ verfügt dann über die SR-Eigenschaft (ist ein ε-Replikator), wenn für alle i gilt, dass $|w_i - v_i| < \varepsilon$. Diese werden in den folgenden Abschnitten implizit auch als PN_{SR} bezeichnet.

Wie in vielen Anwendungen werden NN-*Weights* durch, nicht nur, aber auch, *Supervised Learning* in einem gradientenbasierten Verfahren (BP) optimiert (vgl. 2.3). Beim Training selbstreplizierender NNs durch BP kann ein, das gewichtsweise *Interface* beachtender Trainingsoperator $\leftsquigarrow_{ww}$ wie folgt definiert werden:

Definition 7.3.3: *Weightwise Training*

Gegeben seien die NNs $\mathcal{M}, \mathcal{N}, \mathcal{O}$ mit $\overline{\mathcal{M}} = \langle v_i \rangle_{0 \leq i < |\mathcal{M}|}$. Ein neuronales Netz $\mathcal{O}$ ist das Ergebnis des gewichtsweisen Trainings von $\mathcal{N}$ auf $\mathcal{M}$, geschrieben $\mathcal{O} = \mathcal{N} \leftsquigarrow_{ww} \mathcal{M}$, gdw. $\mathcal{O} = \mathcal{N}$

Dies schließt die Einführung der Grundlagen der SRNNs ab. Im Folgenden wird auf das weiterführende Konzept der zuvor als *Goals* bezeichneten AT eingegangen [69].

7.3.2 Nebenziele (Goals)

Zur Formalisierung der Nebenziele sog. *Auxiliary Tasks (ATs)* werden hier die Erkenntnisse von Gabor et al. einbezogen und das Verfahren des in „Goals for Self-Replicating Neural Networks“ erweiterten *Interfaces* zur Verarbeitung von AT (*Goals*) aufgegriffen. Durch ein kombiniertes Eingabeformat $\mathcal{N} : \mathbb{R}^6 \to \mathbb{R}$ wird das SRNN *Interface* der WWA um Sekundärziele der Form $f : \mathbb{R}^2 \to \mathbb{R}$ erweitert.

Da die im folgenden Abschnitt eingeführte Struktur einen eingehenden Skalar verarbeitet $f : \mathbb{R}^1 \to \mathbb{R}$, kann die zweite Position des AT-*Interfaces* schlicht ausgelassen werden. Um diese Abänderung zum Ausdruck zu bringen, wird die Formulierung des ATs in Definition 7.3.4 aktualisiert:

Definition 7.3.4: ***Self-Replication & Auxiliary Application***

Gegeben seien die NNs $\mathcal{M}, \mathcal{N}, \mathcal{O} : \mathbb{R}^5 \to \mathbb{R}$ mit $\overline{\mathcal{M}} = \langle v_i \rangle_{0 \leq i < |\mathcal{M}|}$. Wobei $\mathcal{O}$ das Ergebnis der WWA von $\mathcal{N}$ auf $\mathcal{M}$, geschrieben $\mathcal{O} = \mathcal{N} \triangleleft_{rep} \mathcal{M}$, gdw.

$$\overline{\mathcal{O}} = \langle \mathcal{N}(v_i, L(i), C(i), E(i), 0) \rangle_{0 \leq i < |\overline{\mathcal{M}}|}$$

wobei L, C, E in Definition 7.3.1 definiert sind.
Bei einem Eingabewert $x \in \mathbb{R}$ ist der Ausgabewert $y \in \mathbb{R}$ das Ergebnis von *Auxiliary Application (AA)* des *Inputs* x, geschrieben $y = \mathcal{N} \bullet_{aux} x$, gdw.

$$y = \mathcal{N}(0, 0, 0, 0, x).$$

7.4 Methode

In diesem Abschnitt wird die in der Einführung beworbene Struktur zunächst motiviert und anschließend definiert. Eine Motivation zu einem früheren Zeitpunkt (der Einführung) wäre aufgrund des hierfür benötigten technischen Hintergrunds nicht möglich gewesen.

Die Vorarbeiten von „Goals for Self-Replicating Neural Networks“ haben demonstriert, dass einzelne, mit dem erweiterten AT-*Interface* (Definition 7.3.4) ausgestattete, NNs nicht nur durch *Weightwise Training (WWT)* (Definition 7.3.3) die SR-Eigenschaft (Definition 7.3.2) erfüllen, sondern auch ein Nebenziel (AT) verfolgen können. Soweit bekannt, betrachten die in diesem Bereich bestehenden Experimente lediglich ein System aus in der gleichen Umgebung „lebenden“ Individuen. Diese betreiben aktiv eine Form der Selbsterhaltung (z. B. SR) und interagieren sporadisch durch wohldefinierte Operatoren [67] im Kontext eines künstlichen Chemiesystems nach biologischem Vorbild. Durch WWT sind diese SRNN aktiv in der Lage entweder nur ihre individuelle SR-Funktion [68] oder SR und eine zusätzliche *Auxiliary Task* [69] zu erlernen. Zwar ist der AT für alle SRNN in diesen Experimenten gleich definiert und es kommt durch die gelegentlichen Interaktionen zu einer gerichteten *Weight*-Manipulation, allerdings lernen alle Individuen nur für sich selbst. Sie haben keinerlei Anreiz zur Kollaboration oder einer zielgerichteten Interaktion, welche direkt oder indirekt Vorteile hinsichtlich der Erfüllung von AT bringt. Zusammengefasst kann man sagen, dass gruppenübergreifende Aufgaben und zielgerichtete Rollen im Kontext von SRNN bislang nicht erforscht sind.

In diesem Kapitel soll dieser Umstand geändert und eine kollaborative Umgebung eingeführt werden, in welcher eine Gruppe von individuellen SRNNs neben der SR eine gemeinsame, globale Aufgabe (AT) zu erlernen hat.

Die Anforderungen hierfür ergeben sich wie folgt:

1. Das SRNN-*Interface* muss erhalten bleiben, sodass SA durchgeführt und

SR erlernt werden kann.

2. SRNNs sollen noch in der Lage sein, untereinander zu interagieren, um die bestehende Forschung [68] auch in Zukunft weiterführen zu können.
3. Alle SRNNs sollen durch ihre *Auxiliary Task*-Funktionen einen globalen, höheren Zweck erfüllen.

Um diesen Anforderungen gerecht zu werden, wird AT als das Erlernen einer linearen Funktion von „multipliziere x mit einem festen Wert w“ definiert. Dies erlaubt eine hierarchische Struktur, in Anlehnung an das Konzept eines arbiträren biologischen Organismus, einzuführen, welche aus Gruppen (*Neurons*) einzelner SRNNs (*Weights*) aufgebaut ist: *Organism Network (ON)*. Durch die Verwendung des in Definition 7.3.4 veränderten *Interfaces*, ändert sich ab hier auch das Namensschema, sodass SRNNs im Kontext des ON als *Particle Networks (PNs)* bezeichnet werden.

Abbildung 7.1.a präsentiert diese PNs und das beschriebene SRNN-*Interface* (grau, grün). Indem lediglich der *Input* des ATs der ersten Ebene (grün) nach außen hin geöffnet wird, können Verbände von PNs (vgl. Abbildung 7.1.b) definiert werden. Diese Verbände ($\sum_{i=0}^{|x|}(PN_i(x_i))$) können gemeinhin als *Cell Networks (CNs)* in einem *Neural Network* gesehen werden. Um lediglich eine Exposition des AT-*Input* zu ermöglichen, wird technisch 0 an die unbenutzten SA-Eingänge übergeben. Anders ausgedrückt wird hier ein strukturierter Graph (ON, Abbildung 7.1.c) eingeführt, in dem Gruppen (CN, Abbildung 7.1.b) von Individuen (PN, Abbildung 7.1.a) miteinander verbunden sind, sodass sich jedes PN wie ein *Weight* in einem (linearen) FFNN verhält.

Definition 7.4.1: *Organism Network*

Ein *Organism Network (ON)* ist ein *Neural Network (NN)*, dessen Kanten nicht durch skalare *Weights* $E_{l,c,e}$, sondern mit *Particle Networks* $\mathcal{M}_{l,c,e}$ (definiert durch die Gewichtsvektoren der PNs $\overline{\mathcal{M}_{l,c,e}}$) besetzt sind. d. h., $\mathbf{w} = \overline{\mathcal{ON}} = \langle \mathcal{M}_{l,c,e} \rangle_{l=1,\dots,r,c=1,\dots,|L_l|,e=1,\dots,|L_{l-1}|}$.
Der *Output* eines ON, sofern ein *Input* $\mathbf{x} \in \mathbb{R}^p$ gegeben ist, lautet: $\mathbf{y} = \mathcal{ON}(\mathbf{x}) = \langle O'(r,c) \rangle_{c=1,\dots,|L_r|} \in \mathbb{R}^q$ wobei

$$O'(l,c) = \begin{cases} x_c & \text{if } l = 0, \\ \sum_{i=1}^{|L_{l-1}|} \mathcal{M}_{l,c,i}(0,0,0,0,O'(l-1,c)) & \text{sonst.} \end{cases}$$

Durch die Definition der PNs als Gewichts-Applikatoren kann das ON in der Theorie als Funktionsapproximator *End-to-end* trainiert werden, wobei jedes individuelle PN weiterhin auf seine eigene ε-SR-Eigenschaft trainieren kann. Es wird also von einem ON erwartet, die SR-Eigenschaft (O_{SR}) innerhalb eines Fehlers von $e = 10^{-5}$, wie in [67] beschrieben (vgl. Definition 7.3.2), zu erfüllen und (gleichzeitig) den globalen AT in Form von „sich-wie-ein-Gewicht-verhalten“ (vgl. Definition 7.4.2) zu erlernen.

Definition 7.4.2: Erlernen der Aufgabe

Ein *Organism Network (ON)*, zusammengesetzt aus vielen *Particle Networks (PNs)* $\mathcal{M}_1, ..., \mathcal{M}_{|\mathcal{ON}|}$[a] hat eine Aufgabe (bis zu einer Genauigkeit von ε, ζ) dann gelernt, wenn

- alle *Particle Networks* $\mathcal{M}_1, ..., \mathcal{M}_{|\mathcal{ON}|}$ ε-Replikatoren sind (vgl. Definition 7.3.2) und
- für alle spezifizierten Testdaten (x, y) die Ausgabe $y' = \mathcal{ON}(x)$ des Netzes $|y - y'| \leq \zeta$ $(=|Loss)$ erfüllt.

[a] Für diese Definition wird die interne Struktur der *Particle Networks* ignoriert und ihre Funktion als Gewichts-Applikator an den entsprechenden Kanten eines *Computational Graph (CG)* angenommen.

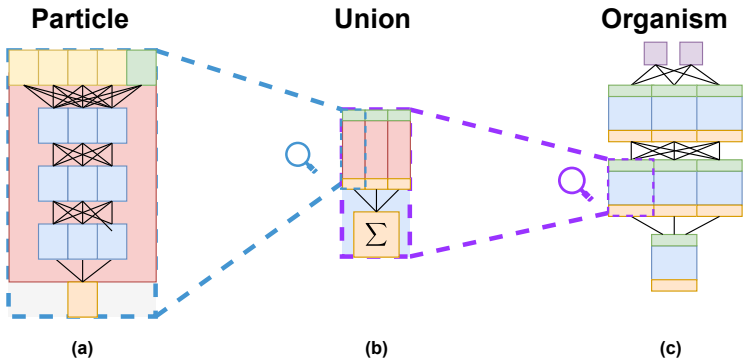


Abbildung 7.1: Schematischer Überblick über die *Architektur* der ON. **a:** Ein PN approximiert eine Gewichtsanwendung (y, orange) für ein Eingabeelement eines Vektors (x, grün), eingebettet in das Eingabeformat aus Definition 7.3.4; **b:** Eine Zelle aggregiert viele Vorhersagen (orange); **c:** Das gesamte ON (hier: zwei Eingänge, drei Schichten) umfasst mehrere schichtweise gebündelte Zellen.

Aufgrund der gegebenen hierarchischen Ordnung in [PN < CN < ON] wird die Optimierung der PNs in Richtung des kollaborativen *Auxiliary Tasks* als *global*, während sowohl *Self-Replication* als auch „sich-wie-ein-Gewicht-verhalten" als *lokale* Aufgaben betrachtet.

7.5 Float-Addition Experiment

Um den Nachweis zu erbringen, dass ein ON tatsächlich in der Lage ist, als Struktur von verketteten Individuen auf einem globalen *Auxiliary Task* (Kooperation) zu arbeiten, soll zunächst eine triviale Aufgabe, als *Sanity Check*

eingeführt, und das ON-Verhalten beobachtet werden: die Addition von zwei Gleitkommazahlen. Es hat sich gezeigt, dass diese Aufgabe als AT für ein einzelnes SRNN erlernbar ist [69]. In ersten Versuchen stellte sich heraus, dass ein *Schedule* benötigt wird, der beide Aufgaben im Lernprozess abwechselnd trainiert. Hiermit kann eine Annäherung sowohl der *globalen* AT als auch der *lokalen* SR erlernt werden. Ein ähnliches Verfahren wird z. B. auch bei *Generative Adversarial Networks (GANs)* angewendet, um die *Performance* der beiden Gegenspieler (zwei *Deep Neural Networks (DNNs)*) zu balancieren [81]. Andere *Schedules*, die zunächst eine der beiden Aufgaben fokussieren, und dann zur jeweils anderen wechseln, zeigten hier starke Charakteristiken des *Catastrophic Forgettings*. Ein ähnliches Vorgehen hat sich allerdings in vorhergehenden Arbeiten als notwendig erwiesen, damit SRNNs in der Lage sind AT zu erlernen [69].

Die hier im Folgenden präsentierten Experimente werden mit den jeweils gleichbleibenden HPs durchgeführt, wobei passende Werte durch eine hinreichend breite *Grid Search (GS)* ermittelt werden: *Stochastic Gradient Descent (SGD)* eignet sich im besonderen Maße aufgrund seiner Einfachheit für Arbeiten im Bereich der Grundlagenforschung, daher wird hier dieser *Optimizer* (*Learning Rate (LR)* $= 0.004$, *momentum* $= 0.9$) verwendet. Trainiert wird mit AT als *Target* für 50 Epochen, wobei diese wiederum in 20 *Batch* unterteilt sind. Am Ende jeder *Batch* werden zusätzlich 25 WWT-Schritte ausgeführt. Eine ausreichende (minimale) Größe von sowohl PN als auch ON (bzgl. *Zelle* & *Layer*) wird nach Untersuchungen auf $\text{PN}^{(3,3)}$ & $\text{ON}^{(3,2)}$ festgelegt.

7.5.1 *Performance*, Stabilität und *Robustness*

Um die hier gewonnenen Ergebnisse in Kontext bestehender Arbeiten [67, 69] einzuordnen und eine Vergleichbarkeit zu schaffen, wird sowohl das ON als auch die enthaltenen PNs auf ihre *ε-Self-Replication (SR)*-Eigenschaft hin überprüft sowie Konvergenzeigenschaften und *Robustness* untersucht. Abbildung 7.2 zeigt die Zeitachse aller PNs innerhalb des trainierten *Organism Network*. Im Laufe des Trainings erlangen diese ihre SR-Fähigkeit (orange).

Selbst über mehrere Durchläufe hinweg ist festzustellen, dass sowohl der *lokale Loss* des WWT als auch der *globale* AT-*Loss* innerhalb von wenigen Epochen konvergieren (vgl. Abbildung 7.3). Diese Beobachtung spricht für ein robustes und stabiles Trainingsverhalten, was aufgrund der relativ trivialen Aufgabe nicht überrascht. Gerade im Bereich von Epoche 9 bis 15 scheinen einige Instanzen deutlich schneller ihr Ziel zu erreichen als andere. Da beide gemittelten Verluste (*Self-Replication* und *Auxiliary Task*) deutlich unter dem Schwellenwert ε liegen, können die assoziierte *lokale* und *globale* Aufgabe nach Definition 7.4.2 für alle PNs als erlernt angesehen werden. Es stellt sich die Frage, warum in so einem Setting zusätzlich auf unterschiedliche Werte für *train*/*test* getestet wird, wenn die Werte für das AT-Training der

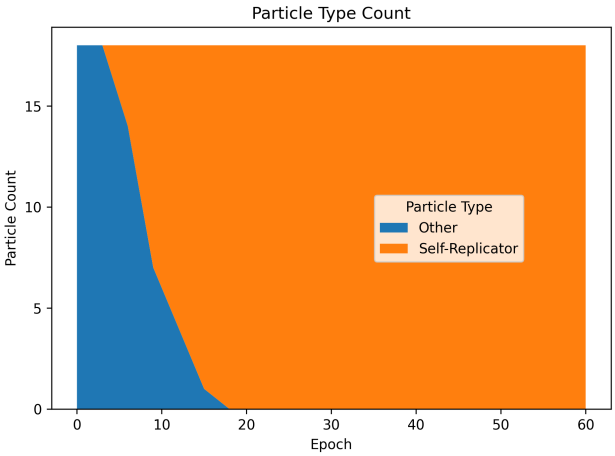


Abbildung 7.2: Verteilung der PN-Typen (PN_{SR} (orange), PN_F (blau; **Y-Achse**) über den Verlauf des Trainings in Epochen.

gleichen Wahrscheinlichkeitsverteilung $\mathcal{U}(-1, 1)$ entstammen. Die Erklärung liefert Abbildung 7.3, in der das abwechselnde Training der ATs & SR-Aufgabe durch den eingeführten *Schedule* beobachtet werden kann.

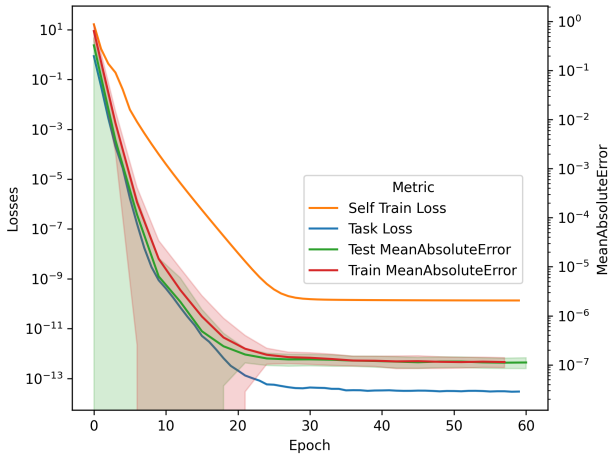


Abbildung 7.3: *Mean Squared Error (MSE)-Loss* der *lokalen* SR (**orange**) sowie der *globalen* AT (**blau**). Dazu zeigt der *Mean Absolute Error (MAE)* die Genauigkeit des AT im Training (**rot**) und Test (**grün**).

Wie durch Gabor et al. gezeigt, kann die *Robustness* der SR-Eigenschaft durch eine wiederholte SA ($\mathbf{N} \lhd_{rep} \mathbf{N}$) in SA-Zyklen gemessen werden. Dieses Experiment wird, wie in Definition 7.3.1 angegeben, für alle PN_{SR} des *Organism Network*s ausgeführt. Dabei ist festzustellen, dass die hier untersuchten PN_{SR} im Vergleich zu SRNN in vorherigen Studien bei wiederholter SA ihre SR-Eigenschaft länger beibehalten (vgl. Definition 7.3.1). Diese Beobachtung trifft

auch unter dem Einfluss von Gaussian *Noise (GN)* in der Größenordnung bis $> 10^{-5}$ zu (vgl. Abbildung 7.4). Interessanterweise ist die durchschnittliche *Robustness* pro GN-Pegel höher, als bei einfachen SRNN [67], obwohl die beobachtete Varianz sowie das .95-Konfidenzintervall deutlich größer ausfallen. Es liegt die Vermutung nahe, dass PN_{SR} stärkere ε-Replikatoren sind als individuell trainierte Varianten (SRNNs). Hier sollte in Zukunft untersucht werden, ob dies an der deutlich längeren Trainingszeit oder anderen sich aus dem NN-Training ergebenden Parametern liegt oder sich hingegen stabilere Plateaus im *Weight*-Space ergeben haben.

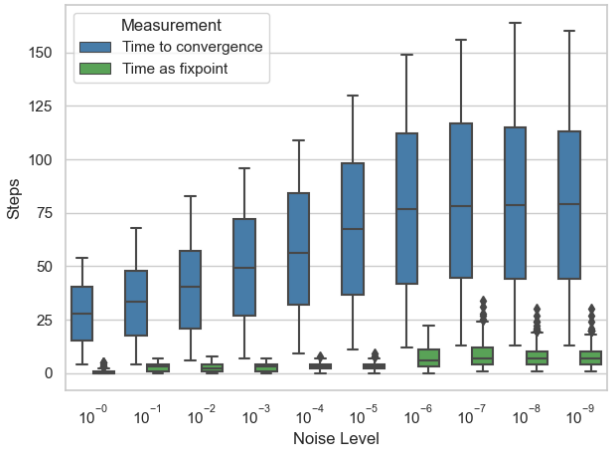


Abbildung 7.4: *Robustness* von PN_{SR}. **X-Achse:** Bereich des auf die ursprüngliche Eingabe angewendeten Rauschens. **Y-Achse:** Anzahl der SA-Schritte in PN_{SR} ihre SR-Eigenschaft bzgl. $\varepsilon = 10^{-5}$ behalten (**grün**); Schritte der Selbstanwendung, bis das Netzwerk als divergiert angesehen wird (**blau**).

7.5.2 Gewichts Re-Substitution

In diesem Abschnitt soll überprüft werden, ob sich die trainierten PNs wirklich wie *Weights* eines NN verhalten. Dazu wird eine stetige Eingabe $x_{ON_w} = (0, 0, 0, 0, 1)$, deren Form der des ON-*Interface* entspricht, eingeführt. Stimmt diese These, dann sollte der *Output* von PN(x_{ON_w}) dem *Weight*, welches jedes PN, in seiner Rolle als Teil des übergeordneten *Organism Network*, zu repräsentieren gelernt hat, entsprechen. Diesem Gedanken folgend sollte es möglich sein, *Weights* eines FFNN der gleichen Beschaffenheit (bzgl. der Anzahl der *Neurons* & *Layer* des ON) durch so erzeugte Gewichtswerte zu ersetzen, dabei sollte der absolute Fehler des ATs nahezu (*Float*-Genauigkeit) gleich bleiben.

Abbildung 7.5 präsentiert das Ergebnis dieser Untersuchung. Es zeigt sich, dass sich ein ON, mit einer Abweichung von $\leq 10^{-8}$ bei einer deutlich geringeren

Varianz, tatsächlich wie ein FFNN verhält.

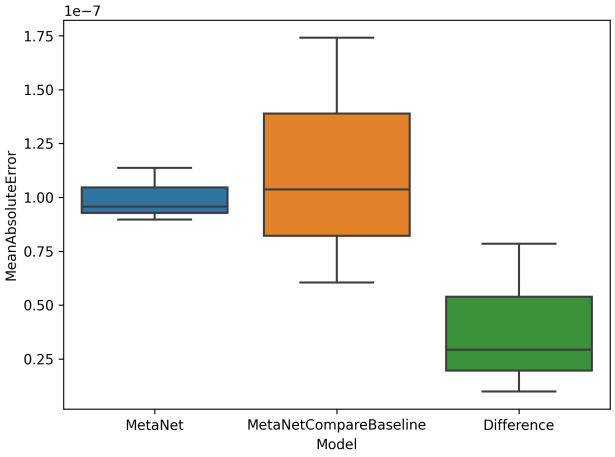


Abbildung 7.5: Absolute Fehlermarge der Additionsaufgabe (**Y-Achse**, skaliert mit $1e-7$; Definition 7.3.4) nach dem Ersetzen von *Weights* in einem FFNN gleicher Größe (bzgl. Anzahl der *Neurons* und *Layer*) durch „gelernte Gewichtsapplikation“ von $pn_i \in PN_0^i$ (vgl. Experiment 7.6.1)

7.6 MNIST-Klassifizierungsexperiment

Wie der vorherige Abschnitt gezeigt hat, ist ein ON grundsätzlich geeignet, um einfache Aufgaben zu lösen. Nun soll untersucht werden, ob auch größere d. h. *globale* ATs gelöst werden können. Hierzu soll eine MNIST-Klassifizierungsaufgabe als AT betrachtet werden und damit das Forschungsfeld der SRNNs näher an etablierte *Machine Learning (ML)* Aufgaben herangeführt werden. In diesem Kontext soll auch der Einfluss von nichtlinearen Activation-Function*s* auf das ON untersucht werden.

7.6.1 Versuchsaufbau

Da (aktuell) die Trainingszeit und die Rechenkosten mit wachsendem *Input*-Vektor und jedem zusätzlichen PN schnell ansteigen, wird für Experimente im Rahmen dieser Arbeit eine auf (15×15)-*Pixel* reduzierte Variante von MNIST verwendet. *MNIST-Dataset* besteht aus 70.000 Bildern standardisierter handgeschriebener Ziffern und gilt seit vielen Jahren als Standard-Benchmark für viele Forschungsfragen im Bereich des *Deep Learning (DL)* [44].

Um die MNIST-Klassifizierungsaufgabe erlernen zu können, muss das ON zunächst angepasst werden. Die benötigte ON-Größe (bzgl. der *Layer* & *Zellen*) wird durch eine Rastersuche anhand eines linearen FFNN (Stellvertreter)

ermittelt und so auf $(3, 3)$ festgelegt. Nach jeder ON-Schicht wird zusätzlich eine Activation-Function platziert, um den Anforderungen der deutlich komplexeren Aufgabe gerecht zu werden[2]. Als Activation-Function fällt hier die Wahl auf *Gaussian Error Linear Unit (GELU)* [93] (vgl. Abschnitt 2.3.1), da diese parametrisierte Funktion im Gegensatz zu *Rectified Linear Unit (ReLU)* im negativen Wertebereich ein Gradienten-Signal zulässt. So wird das Phänomen der *Dying Neurons* reduziert, was im Kontext von SRNN wahrscheinlich zur Entstehung trivialer *Fixpunkte* führen würde (vgl. [67]). MNIST als Klassifizierungsaufgabe ist deutlich schwieriger zu erlernen, als die einfache Addition aus Abschnitt 7.5. Daher wird der Anteil der *lokalen* WWT-Schritte gegenüber dem *globalen* AT-Training auf ein Verhältnis von 5:1 angehoben. Die Parametrisierung des *Optimizers* wird nicht verändert und alle Experimente mit einem zurückgehaltenen 10.000-Stichproben-Testsatz durchgeführt [44].

Da nach ersten Versuchen ein sehr ungewöhnliches Verhalten beobachtet wird und MNIST bekanntermaßen viele 0-Werte enthält, liegt die Überlegung nahe, jedem Bild GN der Größe 10^{-4} hinzuzufügen. So soll sichergestellt werden, dass das beobachtete Verhalten nicht einfach darauf zurückzuführen ist, dass MNIST an diesen Stellen immer mit Null-Werten belegt ist (vgl. Abbildung 7.10). Durch eine automatische Überprüfung können sog. *triviale Fixpunkte* (z. B. wenn alle *Weights* $== 0$ werden) für alle beobachteten PN_{SR} in den folgenden Experimenten ausgeschlossen werden.

7.6.2 Ergebnisse: Leistung, Stabilität und *Robustness*

Abbildung 7.6 zeigt sowohl den *Loss* beider Aufgaben sowie die auf MNIST gemessene Genauigkeit (*Accuracy (ACC)*). Es ist zu beobachten, dass sich das SR-*Loss* schneller als erwartet verbessert, während die ACC stetig bis zur Konvergenz wächst. Diese Situation soll in den nächsten Abschnitten weiter untersucht werden. Auch wenn die Ergebnisse nicht den Leistungen des *State-of-the-Art (SOTA)* entsprechen, demonstriert eine Testgenauigkeit von > 0.8 durch ein eher kleines NN, klar die Fähigkeiten des ONs, und beweist, dass es möglich ist, auch komplexere Aufgaben zu betrachten.

Das Ergebnis der Experimente zur *Robustness* mit den auf MNIST trainierten PNs ist in Abbildung 7.7 dargestellt. Es ist zu sehen, dass das zuvor sehr hohe Maß an *Robustness*, hinsichtlich der SA-Fähigkeit, deutlich zurückgegangen und nun eher mit den Ergebnissen der etablierten Literatur vergleichbar ist [67–69]. Dies steht im Gegensatz zu den Ergebnissen aus Abschnitt 7.5.1 und stützt die Annahme, dass mit einem fokussierten und längeren WWT sehr robuste und stabile SRNN erreicht werden können. Trotz eingehender HP-Suche erlangt nur etwa ein Viertel aller PNs die SR-Eigenschaft. Damit erfüllen diese PNs nicht

[2]Die Anforderung eines Reviewers wurde hier erfüllt. Experimente mit einer linearen Variante zeigen aber das gleiche, im Folgenden beschriebene, Verhalten

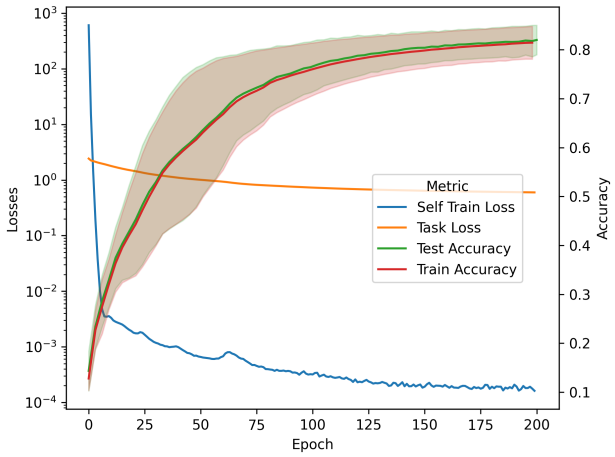


Abbildung 7.6: *Loss* (**linke x-Achse**) vs. Genauigkeit (**rechte y-Achse**) sowohl für die lokale SR-Aufgabe als auch für die *globale* MNIST-Klassifikationsaufgabe ($n_{seeds} = 3$) im Verlauf des Trainings ($epochs = 200$, **x-Achse**).

das Ziel der Aufgabe nach Definition 7.4.2, obwohl, wie in Abbildung 7.9 zu sehen ist, der WWT-*Loss* schnell konvergiert.

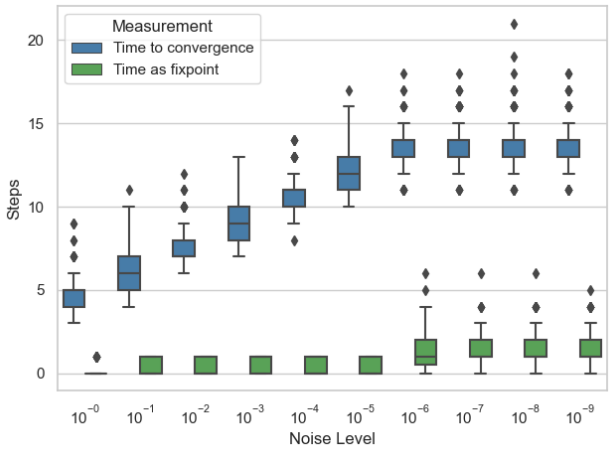


Abbildung 7.7: Die *Robustness* von PNs, welche die SR-Eigenschaft PN_{SR} durch $\lhd_{ww}$ (vgl. Definition 7.3.1) gelernt haben. **x-Achse:** Bereich des auf die ursprüngliche Eingabe angewendeten Rauschens. **y-Achse:** Schritte der Selbstanwendung PNs behalten ihre SR-Eigenschaft, wenn $\varepsilon = 10^{-5}$ (**grün**); SA-Schritte, nach denen PNs als divergieren (**blau**).

7.6.3 Dropout-Test

Da ein gleichzeitiges Lösen beider Aufgaben (SR & AT) durch die PNs des ON im Rahmen der MNIST-Experimente nicht möglich ist, sollen die Eigenschaften und daraus resultierenden Fähigkeiten der PNs an dieser Stelle weiter untersucht werden. Es ist anzunehmen, dass das Erlernen von MNIST-Klassifizierung komplexer ist als das einfache Beispiel einer *globalen* Aufgabe aus Abschnitt 7.5.1. Die folgenden Thesen liegen nach eingehender Analyse der Ergebnisse in Abschnitt 7.6.2 nahe:

- PNs (d. h. *Weights* in einem FFNN), an Positionen mit wenig Information für die Lösung des *Auxiliary Task*, stehen unter dem Einfluss eines kleineren Gradienten.
- Sie erlangen die SR-Eigenschaft früher als ihre nicht SR-Nachbarn (PN_F), da sie (PN_{SR}) im Verlauf des Trainings weniger durch AT beeinflusst werden.
- Diese Annahme würde bestätigt werden, wenn der Einfluss dieser *Weights* auf die Klassifizierungsaufgabe (MNIST-AT) nur marginal wäre.

Um diese Annahmen zu überprüfen, wird ein gruppenweiser *Dropout*-Test definiert, welcher zeigen soll, wie sich die FFNN-*Performance* durch das Wegfallen (0-Wert) von Kantengewichten an Positionen der jeweiligen Gruppe PN $\in [PN_{SR}, PN_F]$ verhält. Dieses Vorgehen entspricht einem „Gewichts-weisen *Dropout* “ (vgl. Abschnitt 2.3.4.2). Anders ausgedrückt soll ein vergleichba-

res FFNN durch *Pruning* ausgedünnt werden, sodass der Einfluss der beiden Gruppen $\langle PN_{SR}, PN_F \rangle$ auf die ACC des MNIST-AT veranschaulicht wird (vgl. Experiment 7.6.1).

Experiment 7.6.1: Dropout Test

$\mathcal{ON}$ sei ein *Organism Network*, sodass $\overline{\mathcal{ON}} = \langle \mathcal{PN}_i \rangle_{i=1,...,|\overline{\mathcal{ON}}|}$ der *Weight*-Vektor von $\mathcal{PN}_i ist : \mathbb{R}^5 \to \mathbb{R}$ für $i = 1, ..., |\overline{\mathcal{ON}}|$ ist. $T : (\mathbb{R}^5 \to \mathbb{R}) \to \{PN_{SR}, PN_F\}$ sei die Bezeichnung aller PNs, sodass jedes SR-fähige $\mathcal{PN}$ durch $T(\mathcal{PN}) = \mathcal{PN}_{SR}$ und jedes andere $\mathcal{PN}$ durch $T(\mathcal{PN}) = PN_F$ gekennzeichnet ist. $\mathcal{Z} : \mathbb{R}^5 \to \mathbb{R}$ sei ein 0-Netzwerk, sodass der *Output* $\mathcal{Z}(\_, \_, \_, \_, \_) = 0$ für jeden *Input* entspricht. Das Organismus-Netzwerk nach dem *Dropout* $\mathcal{ON}'$ für $\mathcal{ON}$ mit Bezug auf den Typ $t \in \{PN_{SR}, PN_F\}$ ist gegeben durch $\overline{\mathcal{ON}} = \langle D(i) \rangle_{i=1,...,|\overline{\mathcal{ON}}|}$, wobei

$$D(i) = \begin{cases} \mathcal{Z} & \textit{wenn } T(\mathcal{PN}_i) = t, \\ \mathcal{PN}_i & \textit{sonst.} \end{cases}$$

Abbildung 7.8 zeigt die Ergebnisse von Experiment 7.6.1 als MNIST-ACC, nach der Ausführung des Experiments für alle Gruppen von $PN \in \{PN_{SR}, PN_F\}$, im Vergleich zur gesamten ON-*Performance*. Zur allgemeinen Überraschung ist festzustellen, dass ein PN_{SR}-*Dropout* zu keiner merklichen Veränderung der gemessenen ACC führt ($> 0,001$ ACC). Als Gegentest wurde ein einfaches 20%-*l1-norm Pruning* durchgeführt, bei dem sich zeigt, dass der Ansatz des *Prunings* durch PN_{SR}-*Dropout* nicht nur mehr *Weights* des FFNN entfernt (+5%), sondern auch eine deutlich höhere Performance beibehält. Der Vergleich mit weiteren *Pruning*-Verfahren steht noch aus.

Durch einen PN_F-*Dropout* wird eine MNIST-Klassifikation hingegen sofort unmöglich. Dies ist angesichts des vergleichsweise großen Anteils von $PN_F \in ON$ und ihrer Konnektivität (vgl. Abbildung 7.11) ein zu erwartendes Ergebnis, da nach einem PN_F-*Dropout*, sprich aller Kanten in einem *Layer*, keine Informationsweitergabe im CG möglich sein sollte.

Es ist also festzustellen, dass PNs, welche in der Lage sind, SR zu erlernen („Spezialisten"), offensichtlich keinen Anteil an der Lösung der Gruppenaufgabe (AT) tragen. Ganz pragmatisch gedacht, könnte man so auf die Idee kommen, dass diese *Weights* in einem FFNN einfach gestrichen werden können (*Pruning*). In dem hier vorgestellten Fall entspräche dieses Vorgehen einer Reduktion des FFNN-*Weight*-Raums um ca. 25%. Beachtet man, dass die HP des Vergleichsnetzes empirisch bestimmt wurden, ist dies ein beträchtlicher Wert. Dieses Experiment 7.6.1 dient diesen Ergebnissen nach der formalen Bestätigung der oben aufgestellten Hypothese.

An dieser Stelle sollte jedoch ein Mangel im Training nicht ausgeschlossen werden, da ein langsamer, aber allmählicher Trend zu mehr Selbstreplikatoren

(PN_{SR}) (vgl. Abbildung 7.9) zu erkennen ist, was auf eine unzureichende Trainingszeit hindeuten könnte. Eine eindeutige Konvergenz könnte also zu diesem Zeitpunkt im Training bislang einfach nicht stattgefunden haben. Erste Versuche in Richtung dieser Vermutung konnten allerdings auch mit deutlich mehr WWT und anderen Implementierungen, z. B. durch *Residual Skip Connection*, lineare Activation-Function*s* oder gewichtete *Loss*-Funktionen keine besseren Ergebnisse erzielen.

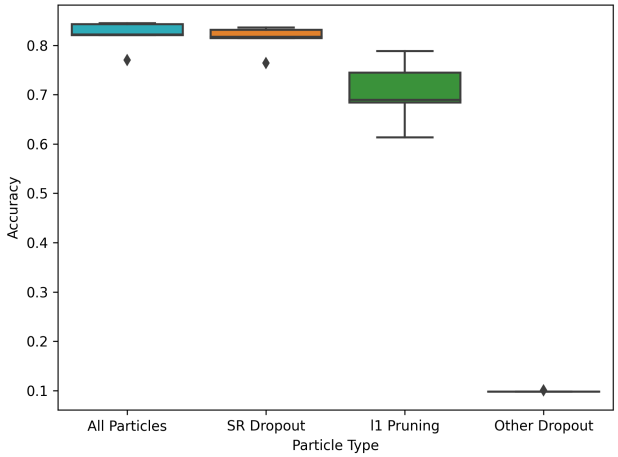


Abbildung 7.8: Genauigkeit (**y-Achse**) nach *Dropout* für die Typen $PN_p \in \{PN_{SR}, PN_F\}$ im Vergleich zum Gesamtorganismus (ON) und $l1 - Norm\text{-}Pruning$ **x-Achse**.

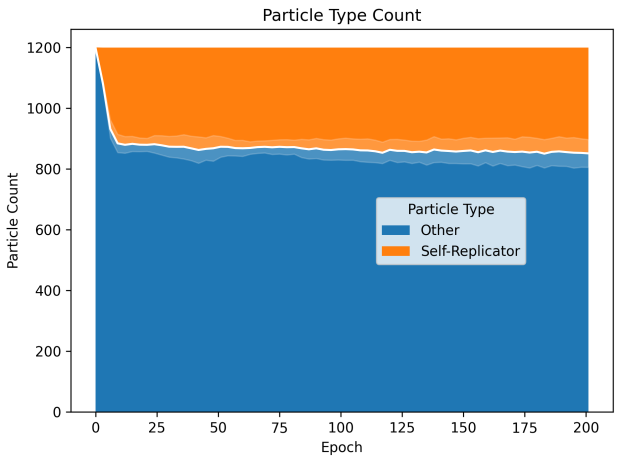
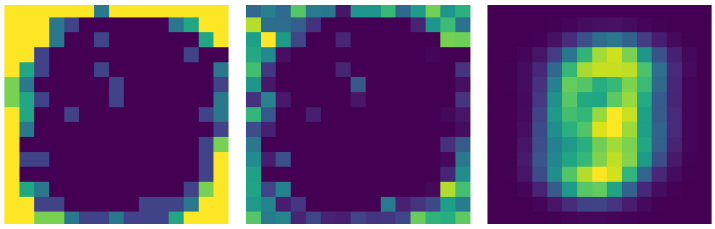


Abbildung 7.9: Verteilung der PNs-Typen (PN_{SR} (orange), PN_F (blau); **y-Achse**) über den Verlauf des Trainings in Epochen (**x-Achse**).

7.6.4 Visuelle Analyse

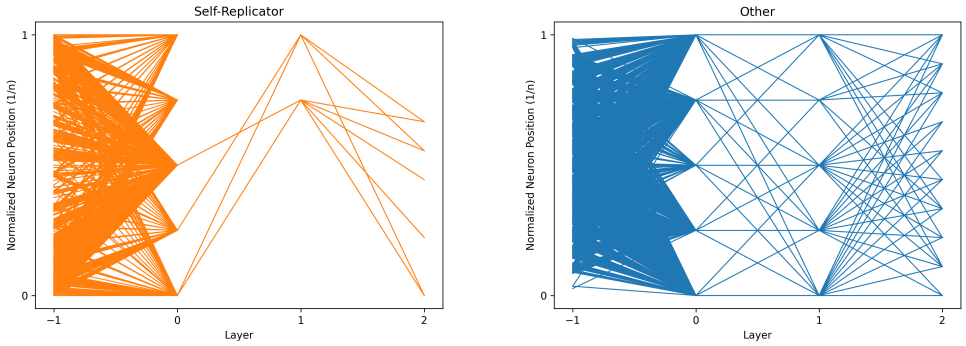
Da auf Bilddaten (hier MNIST) auch eine qualitative visuelle Analyse möglich ist, soll im Folgenden die Positionierung der PN_{SR} untersucht werden. Abbildung 7.10.a stellt die Positionen der PN_{SR} als *Heatmap* dar, während Abbildung 7.10.b hingegen die Summe der für den jeweiligen *Pixel* zuständigen Kantengewichte visualisiert. Zum Vergleich ist in Abbildung 7.10.c der Mittelwert pro Pixel des Testdatensatzes dargestellt. Es ist deutlich zu erkennen, dass die SR-Spezialisten (PN_{SR}) hauptsächlich für die Vorhersage der *Weights* in den „leeren“ Bereichen des Bildes (an den Rändern) verantwortlich sind. Die Kombination dieser Visualisierungen unterstützt die These, dass PN_{SR} vermutlich aufgrund eines *Vanishing Gradient* im Training (vgl. Abschnitt 2.3) lediglich „unwichtigen“ *Input*-Elementen, mit geringem Informationsgehalt bzgl. der MNIST-Klassifikation, zugeordnet sind. In diesen Bereichen ist mit einer schnelleren Konvergenz hin zu SR zu rechnen. Dies ist deswegen interessant, weil explizit durch eine additive Störung mit GN versucht wird, den 0-Werten in diesem Bereich entgegenzuwirken.



a b c

Abbildung 7.10: Position gefundener PN_{SR} als Heatmap. **links:** Pro Pixel Anzahl von PN_{SR} über alle PN-Gruppen (Zellen) in der ersten ON-Schicht; **Mitte:** Pro-Pixel-Summe der *equal-weight-value* von PN_{SR} über alle $\mathcal{PN}$-Gruppen (Zellen) in der ersten ON-Schicht; **rechts:** Mittelwert pro Pixel des MNIST-Datensatzes zum Vergleich.

Abbildung 7.11 visualisiert die Netzwerkkonnektivität für jeden PN-Typ (PN_{SR}, PN_F). Dabei zeigt sich eine starke Häufung von PN_{SR} im ersten NN-*Layer* (nicht ausschließlich), was die oben formulierte These weiter unterstützt. Es kann vermutet werden, dass diese Positionierung direkt mit dem Informationsgehalt des jeweiligen Eingabepixels korreliert.

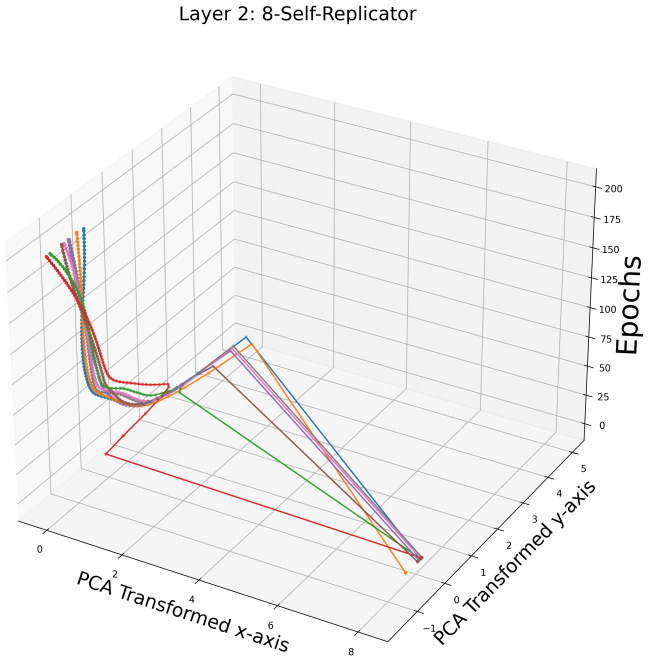


(a) (b)

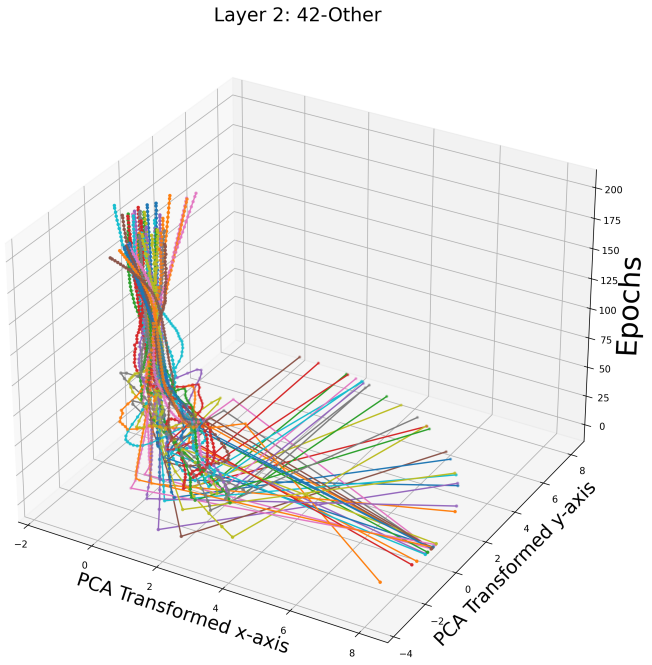
Abbildung 7.11: Verbindungsfähigkeit pro Organismengruppe: **(a)** PNs, die das Merkmal der *Self-Replication* (PN_{SR}) erlernt haben; **(b)** PNs, die das globale *Auxiliary Task* (PN_F) gelernt haben. ($Layer_{-1}$ = ON-*Input*.)

7.6.5 Untersuchung der Gewichtstrajektorie im PCA-Raum

Abschließend soll in Anlehnung an Gabor et al. [67–69] soll in diesem Unterkapitel ein Blick auf die *Principal Component Analysis (PCA)*-transformierten Gewichtstrajektorien für jede PN-Gruppe (vgl. Abbildung 7.12b) geworfen werden. Entlang der meisten *Layer* ist festzustellen, dass die PN_F-Gewichtstrajektorien schnell eine spezifische Region ihres PCA-Gewichtsraums erreichen. Dies ist im Kontext der Vorarbeiten (vgl. [68, 69]) das zu erwartende Verhalten. Im gleichen PCA-Raum betrachtet, zeigt sich, dass die PN_{SR}-Trajektorien zumeist entlang einer einzigen Achse entwickeln (vgl. Abbildung 7.12a.a; $layer = 2$). Dies ist recht ungewöhnlich und wurde zuvor nicht beobachtet, kann aber auch in anderen Untersuchungen so gesehen werden. Welche Rückschlüsse sich auf das Lernverhalten ziehen lassen, ist bislang nicht untersucht. Ferner konnten keine weiterführenden Erkenntnisse aus den PCA-Trajektorien der stark besiedelten ersten Schicht ($15 \times 15 \times 5$ PNs; vgl. Abbildung 7.11) entnommen werden.



(a) 8 PN_{SR}, welche die *Self-Replication*-Eigenschaft erworben haben



(b) 42 PN_F, welche den *Auxiliary Task* gelernt haben.

Abbildung 7.12: Gewichtstrajektorien im PCA-Raum über den Verlauf des Trainings.

7.7 Zusammenfassung und zukünftige Arbeit

In diesem letzten Kapitel wurden Möglichkeiten ergründet, aus sequenziell lernenden SRNN einen gemeinsamen und kooperierenden Organismus (ON) zu konstruieren und zu untersuchen, um das Forschungsfeld der SRNN dem allgemeinen Bereich der DL-Forschung näherzubringen. Der in Abschnitt 7.4 beschriebene Ansatz verwendet die von Gabor et al. eingeführte Erweiterung der SRNN durch sog. *Goals* (hier AT [67]), um SRNN (hier PN) für die Approximation von Kantengewichten eines kollaborativen CG (NN) fungieren zu lassen. Hierbei wurde sich für einen *End-to-end*-Trainingsansatz entschieden, der einen abwechselnden *Schedule* zwischen den verschiedenen Aufgaben ausführt. Es wurde gezeigt, dass sowohl ein einfacher *Proof-of-Concept* (*Float*-Addition) als auch eine nicht triviale Aufgabe (MNIST) gelöst werden können. Allerdings sollte die zu diesem Zeitpunkt noch mit deutlichem Rechenaufwand verbundene *Architektur* in den nächsten Iterationsstufen verbessert werden können.

Während der Untersuchung dieser *Architektur* hat sich herausgestellt, dass PNs auch zu robusten ε-Replikatoren werden können, d. h. SRNN, die sowohl SR beherrschen als auch einen AT innerhalb eines *Organism Network*s erlernen können. Dieser Doppelstatus wird allerdings immer schwieriger zu erreichen, wenn PN in komplexeren Gruppenaktionen eingesetzt werden. Anhand der MNIST-Bildklassifizierung stellt sich heraus, dass diese Doppelfähigkeit vorwiegend bei PNs auftritt, welche für Bereiche des Input-Vektors mit geringem Informationsgehalt verantwortlich sind. Da PNs so definiert sind, dass ihr AT-*Output* eine Annäherung an eine *Weights*-Applikation in einem FFNN darstellt, ist es möglich, dieses *Weight* durch geschickte Aktivierung preiszugeben. Nach einem selektiven *Dropout* (sog. *Pruning*) von *Weights* an PN_{SR}-Positionen ist lediglich ein minimaler Verlust an Test-ACC zu beobachten. Damit wird die *Pruning*-Leistung eines einfachen $l1 - Norm\ Pruning$ übertroffen.

Zukünftige Untersuchungen sollten weitere Aspekte des self-*Pruning* innerhalb des ON erforschen. Weiterhin könnten PNs auch an anderer Stelle Verwendung finden und z. B. als *Zelle*-Aggregatoren (die Summenfunktion ersetzend) oder Approximatoren für Activation-Function*s* werden. Ferner ist die Untersuchung der von Gabor et al. definierten Operator-basierten Interaktion zwischen PNs ein offenes Forschungsfeld. Hier wäre interessant herauszufinden, ob und in welchem Maße z. B. das *Pruning* oder die *Robustness* des ON, bzw. der Lernprozess positiv beeinflusst oder sogar gesteuert werden kann. Da die hier untersuchten NN noch relativ simpel gehalten wurden, könnten weitere *Architekturen*, die zu erlernende Funktion und das Trainingsregime (*Schedule*) aus dem etablierten Bereich des DL in die weitere Betrachtung einfließen. Obwohl gezeigt wurde, dass der hier entdeckte *Pruning*-Ansatz auf MNIST gute Ergebnisse liefert, wäre es interessant, RGB-Datensätze (z. B. *CIFAR10-Datensatz (CIFAR)*) zu untersuchen. Diese Überlegungen sowie weitere Vergleiche mit den in Abschnitt 7.2 erwähnten *Pruning*-Strategien bleiben als Forschungsfrage für zukünftige Arbeiten an dieser Stelle unbeantwortet.

8 Zusammenfassung und Ausblick

Dieses Kapitel wird nun als Abschluss einen Rückblick auf die zurückliegenden Inhalte geben, gewonnene Erkenntnisse noch einmal zusammenfassen und anschließend einen Ausblick für die jeweiligen Teilbereiche wagen.

8.1 Zusammenfassung und Ergebnisse

Diese Arbeit führte zunächst in 2 die gemeinsamen methodischen Grundlagen der folgenden Kapitel ein. Dabei wurde die grundlegende Arbeitsweise von NNs und Besonderheiten im Umgang mit diesen behandelt. Auch wurde auf den Datentyp der „Sequenz“ eingegangen und im Speziellen der Umgang mit Audio-Sequenzen und deren Transformation in Mel-Spektrogramme behandelt.

Anschließend folgten fünf Inhaltskapitel (3, 4, 5, 6 & 7), welche sich mit dem Kontext der Verarbeitung von sequenziellen Daten durch NN beschäftigten. Dabei wurden unterschiedliche Perspektiven angenommen, um detaillierte Einblicke in jeweils unterschiedliche Teilbereiche zu werfen. Durch Experimente wurden zuvor aufgestellte Thesen bekräftigt oder widerlegt und neue Einblicke in die Funktionsweisen und das Lernverhalten von NNs generiert.

Beginnend mit Kapitel 3 wurde zunächst das Konzept einer natürlichen Grenze für den Informationstransport in *Recurrent Neural Networks (RNNs)* und Varianten (LSTM, GRU) durch klare Methodik offengelegt. Dieser *Memory Horizon (MH)* wurde bereits in der Vergangenheit unter anderem Namen und im Kontext einer anderen NN-*Architektur* untersucht. Die hier gemachten Betrachtungen fassen allerdings moderne RNN-Varianten in Kombination mit nichtlinearen Activation-Function*s* in Auge. Auch wird die bestehende Methodik durch einen neu eingeführten experimentellen Aufbau erweitert. Mittels Zufallssequenzen (*independent, identically distributed (i. i. d.)*) variabler Länge konnten die hier betrachteten RNN-Varianten gezwungen werden, Elemente einer Sequenz so lange wie möglich in ihrem *Hidden Vector (HV)* zu repräsentieren und wiederzugeben. Für eine praktische Anwendung stellt sich jedoch die Frage, wie relevant diese Beobachtung ist. Zudem wurden jüngst, vorrangig im Bereich des *Natural Language Processing (NLP)*, immer mehr RNN-basierte NNs durch *Transformer* abgelöst.

In Kapitel 4 (und später in 5) wurde die praktische Verarbeitung sequenzieller Audiodaten auf Basis von Mel-Spektrogrammen im Kontext von *Real-World*-Datasets durch den Einsatz unterschiedlicher Verfahren und NN*Architekturen* demonstriert. Dabei wurden unter anderem Methoden aufgezeigt, die den Umgang mit unterrepräsentierten Klassen, *weakly* oder *noisy Labels* und unausgewogenen Datensätzen behandeln. Durch *Data Augmentation (DA)* und eine breite HP-Suche konnten verschiedene CNN-basierte *Models* miteinander verglichen werden. Dabei wurde die generelle Funktionsweise des *SubSpectralNet (SSN)* hinterfragt. Dabei handelt es sich um eine *Architektur*, mit dem Fokus auf die Etablierung disjunkter *Embedding*-Räume je betrachtetem Mel-Spektrogrammbereich. Hierzu wurde eine einfachere Variante eingeführt, die das Konzept der getrennten Betrachtung von Mel-Bändern übernahm, aber im Kontext des hier betrachteten Datensatzes eine *Performance*-Steigerung aufweisen konnte.

Kapitel 5 führte dann den aus dem *Computer Vision* bekannten *Vision Transformer (ViT)* in den Kontext der Mel-Spektrogramm-basierten Audio-Verarbeitung ein. Diese *Attention*-basierte NN-*Architektur* interpretiert Bilder, also 2D- bzw. 3D-Arrays, als Sequenzen, um lokale Repräsentationen in den Kontext ihrer Nachbarn zu setzen. Dieses *Model* war den als *Baseline* bereitgestellten CNN-Varianten in den ersten Experimenten und im Kontext des betrachteten Datensatzes überlegen. Auch wurden hier alternative *Patch*-Layouts in Betracht gezogen, die in dieser Arbeit bislang allerdings nicht zu den erhofften Leistungssteigerungen geführt haben.

In Kapitel 6 wurde dann aufgrund der zuvor gemachten Erfahrungen das Themenfeld der DA weiter beleuchtet und durch eine ausführliche Hintergrundrecherche zugänglich gemacht. Auf Basis der so gewonnenen Erkenntnisse entstand durch die Rekombination verschiedener Konzepte eine neue DA-Methode, welche auf räumlichen Informationstransport geometrischer Polygone mit nicht trivialen Kanten setzt. Dabei zeigte sich zwar, dass durch *Voronoi Patches (VP)* keine allgemeine Leistungssteigerung im Vergleich zu den betrachteten Alternativen erzielt werden konnte. Allerdings konnte die generelle Tendenz des *Overfittings* der betrachteten *Models* reduziert werden. Durch erweiterte Experimente wie der Untersuchung der Entropieveränderung und einer *Structural SIMilarity Index*-Analyse konnte ein tiefer Einblick in das Konzept des Informationstransports als DA-Methodegewährt werden.

Abschließend, in Kapitel 7 wurde über den Tellerrand traditioneller Sequenzverarbeitung hinweggeblickt, um sich mit NNs zu beschäftigen, die ihren eigenen Trainingsdatensatz sequenziell verändern. Im Detail wurde eine neue *Architektur* auf Basis SRNN zusammengesetzter ONs vorgestellt und dessen Anwendbarkeit evaluiert. Dieser Versuch, das Forschungsfeld dieser selbstreplizierenden Module näher an die NN-Forschung zu bringen, endete mit der Entdeckung von *Natural Pruning*, welches in einem ersten Experiment einem vergleichbaren Ansatz, L1-*Pruning*, überlegen war. Zugleich wurden die Werkzeuge der SRNN um

wichtige und zentrale Fragen erweitert, ohne die bestehenden Operatoren des *Artificial Chemistry of Life* überflüssig zu machen. So können nicht nur völlig neue Anwendungsfälle erschlossen werden, sondern auch bestehende Konzepte der SRNN weiter erforscht werden.

8.2 Ausblick und offene Forschungsfragen

Nach einem Rückblick auf die vorhergegangenen Kapitel soll an dieser Stelle auf Anknüpfungspunkte für zukünftige Forschung an den behandelten Themen hingewiesen werden. Angefangen mit Kapitel 3 stellt sich, wie auch oben zuvor schon angesprochen, die noch offene Frage, welche Implikationen die beobachtete Grenze des Informationstransports, der MH, für reale Anwendungsfälle auf *Real-World*-Datasets hat. Kann diese Grenzlinie z. B. auch bei nicht i. i. d. verteilten Sequenzen beobachtet werden? Diese Frage ist nicht einfach zu beantworten, da gerade bei RNN-*Architekturen* ein Einfluss des *Vanishing Gradient Problem (VGP)* ausgeschlossen werden sollte. Hier könnte ein Datensatz mit sequenzieller Korrelation und gelegentlichen Extremereignissen als Ausgangsbasis genutzt werden, um z. B. eine Anomalieerkennung als Testszenario zu etablieren. Dabei sollte jedoch darauf geachtet werden, dass es wichtig ist, die stets vorhandenen (und meist gewünschten) *Generalization*-Bestrebungen eines NN, klar von dem Informationsverlust durch MH abzugrenzen. Neben diesen Überlegungen stellt sich natürlich auch die Frage, inwieweit weitere RNN *Zelle*-Varianten von diesem Phänomen betroffen sind. Zwar wurden z. B. die zum Zeitpunkt der Experimente aktuellen und populären Varianten in Betracht gezogen, andere blieben allerdings noch außen vor [73, 74, 196], vor allem, weil diese explizite Zählmechanismen implementieren. Weiterhin könnten auch moderne „tiefe“ *Recurrent Neural Network-Architekturen* betrachtet werden, welche behaupten, für bestimmte Aufgaben besser geeignet zu sein als RNN oder *Transformers*. Das Feststellen des MH in solchen und anderen *Architekturen* könnte in Zukunft als nützliches Auswahlkriterium fungieren.

Zwar wurden in Kapitel 4 & 5 jeweils neuartige NN-*Architektur* eingeführt und in ersten Experimenten evaluiert, gerade für *SubSpectralClassifier (SSC)* sind hier noch weitere Experimente auf weiter verbreiteten Datensätzen nötig. Mögliche Kandidaten hierfür wären *AudioM-MNIST* [110] (gesprochene Zahlen von $0-9$) oder *Urban8K* [203] (Stadtgeräusche in hinreichender Auflösung), welches nur aufgrund der beschränkten Ressourcen unbeachtet blieb. Neben allen möglichen an eine *Architektur* gebundenen HPs zu betrachten, kann auch Architektur als solche betrachtet werden. Jeweils im Kontext des jeweiligen Datensatzes gehen Ansätze wie *AutoML*[1] bereits diesen Weg. Der in dieser Arbeit und in [78] untersuchte Ansatz, Mel-Spektrogramme mit ViT zu analysieren, zeigte in beiden Arbeiten eine jeweilige Überlegenheit im Kontext der betrachte-

[1] *AutoML* - `https://www.automl.org/automl/`

ten Datensätze. Dieser scheinbare Entwicklungsvorsprung ist allerdings kritisch und grundsätzlich im Kontext von [232, 244] zu hinterfragen. Auch, weil mit *Transformer*-basierten *Architekturen* deutlich mehr Rechenleistung und damit Energie solchen *Models* zur Verfügung gestellt wird.

Die in Kapitel 6 vorgestellte, jüngste Veröffentlichung [104] lässt noch einige Fragen offen. Zwar wurde eine breite Analyse durchgeführt, wodurch wichtige Eigenschaften der vorgestellten DA-Methode(VP) aufgedeckt werden konnten, die definierten Parameter lassen aber noch Platz für weitere Experimente. So kann der „weiche“ Übergang zwischen *Patch* und Bild weiter untersucht werden. Die Breite dieses Übergangs wurde im Kontext des Datensatzes empirisch bestimmt, könnte aber als HP interpretiert und im Training weiter optimiert werden. Zum aktuellen Zeitpunkt ist der Algorithmus nicht sehr performant, sodass man mit mehr Trainingszeit als mit vergleichbaren DA-Ansätzen rechnen muss. Die hier präsentierten Ergebnisse lassen den Faktor, der *wall clock time*, also wie lange ein Algorithmus in Echtzeit rechnet, jedoch außer Acht, sodass die berichteten Ergebnisse ihre Gültigkeit behalten. Weiterhin könnte z. B. durch *Attention* oder eine andere Heuristik unterstütztes Verfahren genutzt werden, um nach einigen Epochen explizit nur *Features* zu verschieben. Vor allem im Kontext von ViT sollte diese DA-Methodenoch untersucht werden, da hier die Analyse der *Feature* innerhalb der Informationsfläche einer leicht anderen Interpretation unterliegt. Zur Vollständigkeit sei hier erwähnt, dass weitere Untersuchungen mit deutlich gesteigertem Rechen-Budget auf anderen Datensätzen das hier gemachte Bild vervollständigen könnten.

ON konnten in Kapitel 7 zeigen, dass durch ein alternierendes Training auf unterschiedlichen Aufgaben ein natürlicher *Pruning*-Vorgang auftritt. Dies sollte noch weiter erforscht werden, da es interessant wäre, ob andere Mischungen von Aufgaben zu gleichen Ergebnissen führen oder explizit SR hierfür verantwortlich ist. Da es bislang nicht gelungen ist, die SR-Eigenschaften in ONs im Training auf MNIST bei allen PNs zu etablieren, sollte hier natürlich weiter untersucht werden, wie das Gradientenverhältnis zwischen den beiden Aufgaben in Einklang gebracht werden kann. Wünschenswert wäre, eine *Architektur* zu erhalten, die Informationen über sich selbst lernen und wiedergeben kann, ohne dass dies ihre einzige Aufgabe ist. Diese Arbeit hat einen wichtigen Schritt in diese Richtung geliefert, ist aber bislang nicht an diesem Ziel angekommen. Auch wurde bislang nicht versucht, bestehende Operatoren des SRNN-Werkzeugkastens im Kontext von ON wieder einzubinden. Erste nicht veröffentlichte Experimente zeigten, dass es aufgrund des aufrechterhaltenden *Interfaces* Interaktionen zwischen den PNs zulässt. Bislang konnte hier aber kein Mehrwert nachgewiesen werden, was ein noch offener Punkt für zukünftige Betrachtungen ist. Abschließend sei auch darauf hingewiesen, dass das hier beobachtete *natural Pruning* unbedingt anderen *Pruning*-Verfahren gegenübergestellt werden sollte. Wichtig wäre es, den zentralen Mechanismus hinter dem Phänomen herauszuarbeiten, um zukünftig sich selbst regulierende (bzgl. *Layer*-Breite) NNs trainieren zu können.

Abkürzungsverzeichnis

AA	Auxiliary Application	125
ACC	Accuracy	132
AE	Auto-Encoder	70
AT	Auxiliary Task	120
BCE	Binary Cross Entropy	63
BER	Balanced Error Rate	69
BN	Batch Normalization	20
BPTT	Backpropagation Through Time	32
BP	Backpropagation	121
CN	Cell Network	126
CC	ConvClassifier	66
CCS	*COVID-19 Cough*-Datensatz	83
CE	Cross Entropy	108
CG	Computational Graph	127
CIFAR	CIFAR10-Datensatz	140
CNN	Convolutional Neural Network	122
COVID-19	Coronavirus disease 2019	83
CPL	Cut, Paste and Learn	104
DA	Data Augmentation	142
DL	Deep Learning	131
DNN	Deep Neural Network	128
ES	Early Stopping	22
ESN	Echo State Network	35
ESP	Echo State Property	35
FFNN	Feed Forward Neural Network	122
FFT	Fast Fourier Transformation	26
FM	Feature Map	55
FT	Fourier Transformation	25
GAN	Generative Adversarial Network	128
GC	Gradient Clipping	36
GELU	Gaussian Error Linear Unit	132
GMV	Gated Memory Vector	33
GN	*Gaussian* Noise	130
GRU	Gated Recurrent Unit	154
GS	Grid Search	128
HaS	Hide-and-Seek	101
HP	Hyperparameter	122

HV Hidden Vector . . . 141
i. i. d. independent, identically distributed . . . 141
LREL Leaky ReLU . . . 15
LR Learning Rate . . . 17
LSTM Long Short-Term Memory . . . 154
MAE Mean Absolute Error . . . 129
MASC *Mask Augsburg Speech Corpus*-Dataset . . . 59
MC Memory Capacity . . . 36
MFCC Mel-Frequenz-Cepstral-Koeffizienten . . . 52
MH Memory Horizon . . . 141
MLP Multi-Layer Perceptron . . . 76
ML Machine Learning . . . 131
MM Matrix-Multiplikation . . . 8
MNIST MNIST-Dataset . . . 121
MSE Mean Squared Error . . . 129
MT Mersenne Twister . . . 38
NLP Natural Language Processing . . . 141
NN Neural Network . . . 120
ON Organism Network . . . 120
PCA Principal Component Analysis . . . 138
PN Particle Network . . . 120
PP Preprocessing . . . 51
PRS *Primates*-Datensatz . . . 82
RCC ResidualConvClassifier . . . 66
RC Reservoir Computing . . . 36
RE Random Erasing . . . 101
ReLU Rectified Linear Unit . . . 132
RICAP *Random Image Cropping and Pasting* . . . 102
RNN Recurrent Neural Network . . . 141
ROC Receiver Operating Characteristic . . . 96
ROS Random Oversampling . . . 90
SA Self-Application . . . 123
SARS-CoV-2 Severe acute respiratory syndrome - Coronavirus 2 . . . 84
SGD Stochastic Gradient Descent . . . 128
SSIM Structural SIMilarity Index . . . 114
SOTA State-of-the-Art . . . 132
SRNN Self-Replicating Neural Network . . . 120
SR Self-Replication . . . 120
SSC SubSpectralClassifier . . . 143
SSN SubSpectralNet . . . 142
STFT Short-Time Fourier Transformation . . . 26
SVM Support Vector Machine . . . 10
Tanh Hyperbolic Tangent Function . . . 15
TPE Tree-Structured Parzen Estimator . . . 22

UAR Unweighted Average Recall 69
VD Variational Dropout 122
VDRRE Voronoi Decomposition-based Random Region Erasing . 101
VGG Visual Geometry Group(*ausgeschriebene* Kürzel) 52
VGP Vanishing Gradient Problem 143
ViT Vision Transformer 142
VP Voronoi Patches 142
VViT Vertical Vision Transformer 92
WD Weight Decay 38
WS Weighted Sampling 75
WWA Weightwise Application 123
WWR Weightwise Reduction 123
WWT Weightwise Training 125

Fachwortverzeichnis

Activation-Function

wörtlich: Aktivierung; Die Anwendung einer *Aktivierungsfunktion* (vgl. Abschnitt 2.3.1) auf einen Tensor. (vgl. Abschnitt 3.2).

Dense-Layer

Vollvermaschter *Layer*. *siehe Fully-Connected*

Architektur

Beschreibt die Art und Weise, wie *Neurons* in *Layer* zusammengefasst und untereinander verbunden werden. Wird in der deutschsprachigen Literatur auch als „Topologie“ bezeichnet, was aber aufgrund möglicher Zyklen im CG eines NN als unzutreffend erscheint. *vgl. Model*

Attention

wörtlich: Aufmerksamkeit; Beschreibt ein Verfahren zur Kontextualisierung von Vektoren in NNs im Kontext von NLP. Durch die Berechnung eines quadratischen, parametrisierten Verhältnisses zwischen *Key- K* & *Query- Q* wird ein *Score* berechnet, durch welches der *Value*-Vektor V mit zusätzlichen Informationen bereichert werden kann.

Baseline

wörtlich: Grundlinie; Vergleichsgrundlage zur Evaluation von Modellen aus dem Bereich des ML.

Batch

Submenge eines Trainingsdatensatzes, der in einer Epoche als Grundlage für ein *Weight*-Update zusammengestellt wird (vgl. Abschnitt 2.3.2).

Bias

1. Addierbare *Weights* in einem *Neuron*, welche die Ausgabe steuern (vgl. Abschnitt 2.3). *2.* Die *Voreingenommenheit* eines (statistischen) Modells.

Binary Classification

wörtlich: Binäre Klassifikation; Beschreibt die Aufgabe im Gegensatz zur *Multi-Class Classification*, zu unterscheiden, welcher von zwei möglichen Klassen ein *Sample* zugeordnet ist.

Catastrophic Forgetting

wörtlich: Katastrophales Vergessen; Überschreiben schon gelernten *Wissens* während des NN-Trainings (vgl. Abschnitt 2.3.4.1).

Chain Rule

Kettenregel zur Bestimmung des partiellen Anteils eines *Weight* am *Loss* eines NN (vgl. Abschnitt 2.3).

Classifier

wörtlich: *Klassifizierer*; Ein *Model* bzw. eine Funktion mit dem Ziel ein *Mapping* zu erlernen, das ein *Sample* genau einer von n Klassen zuordnet. *siehe Binary Classification*, *Multi-Class Classification*

Computer Vision

wörtlich: Computer Vision; Beschreibt das Anwendungs- und Forschungsfeld der Bildverarbeitung, z. B. Segmentierung und Klassifizierung

Convolution

wörtlich: Konvolution; Parametrisierte, mathematische Faltung zur Extraktion von Merkmalsinformationen (*Features*, vgl. Abschnitt 4.3.1).

Decoder

Aneinanderreihung von *Layer* zur Generierung künstlicher Daten Y aus einer latenten Repräsentation Z, sodass $Y = P(Z|\Theta)$. Oft in Verbindung mit einem *Encoder* in Verwendung.

Dropout

wörtlich: Rausfallen; **1.** Förderung der Generalisierungsfähigkeiten durch (zufälliges) Abschalten mehrerer zufällig bestimmter NN-*Neurons* durch das Ersetzen der *Weights* mit 0 ($w_n = [0]$) (vgl. Abschnitt 2.3.4). **2.** Klasse von DA-Methoden, welche auf die Okklusion von Bereichen der Trainingsdaten setzt. Fördert die *Generalization* durch Verlagerung des Fokus auf Merkmale der zentralen Objekte, anstatt des Objekt-Kontextes.

Embedding

wörtlich: Einbettung; Resultat einer strukturerhaltenden Funktion $f : X \to Y$

Encoder

Aneinanderreihung von *Layer* zur Extraktion von *Feature*, bzw. der Konstruktion einer den Datensatz abbildenden Repräsentation. Approximiert (nach erfolgreichem Training) die nicht lineare Abbildung der Trainingsdaten X auf einen *Latenten Raum* Z, sodass $Z = P(X|\Theta)$. Oft in Verbindung mit einem *Decoder* in Verwendung.

End-to-end

wörtlich: *Ende-zu-Ende*; Beschreibt den im DL gängigen Ansatz, *Models* für mehrere Epochen am Stück zu trainieren, wobei lediglich *Input*.

Feature

Merkmale; im Kontext von DL sind hiermit mathematische Muster gemeint, die über einen Datensatz verteilt wiederkehrend in Erscheinung treten. In tieferen Ebenen eines NN nähern sich *Feature* den nicht linearen Hauptkomponenten eines Datensatzes an. Kann mit einer nicht linearen PCA verglichen werden.

Float

Gleitkommazahl

Forecasting

wörtlich: Vorhersagen; Wendet ein *Model* auf eine Folge als Eingabe an, um deren weiteren Verlauf vorherzusagen.

Fully-Connected

wörtlich: vollvermascht, vollständig verbunden; Gibt an, dass alle *Neurons* eines *Layer* sich mit allen *Neurons* des vorherigen *Layer* verbinden. - *siehe* Dense-Layer, *Linear-Layer*

Gate

wörtlich: Gatter; Eine Art (parametrisierte) Schleuse, die den Einfluss eines Informationsstroms (Vektor) steuert.

Generalization

Der Erhalt der generellen Funktionsweise einer erlernten Funktion auf ungesehenen Daten.

Input

Eingabe in eine Funktion oder einen Algorithmus.

Interface

(Software) Schnittstelle

Kernel

Fenster, Kernel; **1.** Bearbeitungsfenster mit Höhe und Breite, das über Eingabedaten bewegt wird, um eine Funktion sequenziell auf die Daten in diesem anzuwenden. **2.** Mathematisch - Abbildung von Eingabedaten in einen Skalarproduktraum.
(vgl. Abschnitt 4.3.1)

Label

wörtlich: *Etikett*; die aus einem zuvor gekennzeichneten Datensatz stammenden Zuordnungen eines Beispieldatenpunkts.

Layer

Schicht in einem NN; besteht aus einer Vielzahl von *Neurons* oder einer Funktion auf einem Vektor und stellt eine parallelisierte Operation (vgl. Abschnitt 2.3) im CG dar.

Linear-Layer

Vollvermaschter *Layer*. *siehe Fully-Connected*.

Logit

Pure, lineare Ausgabe eines *Layer*; Das zu erwartende innere Produkt der *Neurons*.

Loss

Berechneter Fehler, der zum Training von NN durch einen *Optimizer* mittels BP eingesetzt wird (vgl. Abschnitt 2.3).

Mapping

Eine Zuordnung $(1 \to 1 | m \to 1 | m \to n; m, n \in \mathbb{N})$, wird im Kontext des DL oft als die gelernte Funktion eines *Model*, welches eine Menge von *Input* auf einen *Output* abbildet.

Model

1. Statistisches Model **2.** Trainiertes NN $\neq$ NN-*Architektur*

Multi-Class Classification

wörtlich: Mehrklassenklassifikation; Beschreibt die Aufgabe im Gegensatz zur *Binary Classification*, zu unterscheiden, welcher von n möglichen Klassen ein *Sample* zugeordnet ist.

Natural Pruning

wörtlich: Natürliches Ausscheiden, Ausdünnen; Auftreten eines Effekts, der mit dem Auslassen von Kanntengewichten eines NN (*Pruning*) vergleichbar ist.

Neuron

Funktionale Einheit, die in der gängigsten Form durch eine MM und eine Addition definiert ist. Dies kann aber auch weitaus komplexere Formen annehmen Abschnitt 3.2.2, 3.2.1, 3.2.3, 2.3 & 4.3.1.

Normalization

Standardisierung; Lineare Transformation von Daten, sodass diese einen Erwartungswert von $\mu = 0$ und eine Varianz von $\sigma = 1$ ergeben (vgl. Abschnitt 2.3.4).

Optimizer

Optimierungsalgorithmus, der sich im Kontext von NN gradientbasierte Verfahren zunutze macht, um ein *Loss* zu optimieren (minimieren oder maximieren) (vgl. Abschnitt 2.3.2).

Output

Ausgabe einer Funktion oder eines Algorithmus.

Overfitting

Überanpassung; Die approximierte Funktion eines NN wird so stark an die tatsächliche Verteilung der Datenpunkte des Trainingsdatensatzes angepasst, dass keine Generalisierung mehr möglich ist und die Leistung auf nicht-gesehene Eingaben (deutlich) abnimmt (vgl. Abschnitt 2.3.4.1).

Padding

wörtlich: Polsterung; Auffüllung einer Matrix mit zusätzlichen Werten, meist im Randbereich (vgl. Abschnitt 4.3.1). Entweder es wird 0 verwendet oder benachbarte Werte kopiert oder interpoliert.

Patch

wörtlich: *Flecken*; oft im Kontext von Verfahren der Bildverarbeitung verwendet; zusammenhängende Länge von *Pixel*.

Performance

wörtlich: Performanz, Leistungsfähigkeit; Gibt die Güte einer Funktion hinsichtlich einer Metrik, oft im Vergleich zu artverwandten Konzepten an.

Pixel

wörtlich: *Bildpunkt*; oft im Kontext von Verfahren der Bildverarbeitung verwendet; Werte (-Tuple) das die (Farb-) Intensität einer Zelle in einem Rasterformat beschreibt. Viele Pixel ergeben ein Bild.

Pooling

wörtlich: Bündelung; Reduziert die räumliche *Auflösung* (die Anzahl der Elemente in den als *Höhe* & *Breite* verstandenen Dimensionen) von 2D-Matrizen z. B. bei Bilddaten (vgl. Abschnitt 4.3.1) im Bereich eines definierten *Kernel* Als Bündelungsverfahren findet oft eine der folgenden Funktionen Verwendung: *max*, *mean*, *min*.

Pruning

Reduktion eines NN-Parameterraums, auch Beschneiden oder Ausdünnen (vgl. Abschnitt 7.3).

Real-World

Die Anwendung eines *Models* oder eines Algorithmus außerhalb der zuvor beobachteten Forschungsumgebung. Wird auch in Zusammenhang mit

Datensätzen verwendet, beschreibt dann, dass die Art der Erhebung und die Quelle an sich nicht störungsfreien Laborbedingungen entspricht. Klassen können unterrepräsentiert, Labels schwach und ein starkes Hintergrundrauschen vorhanden sein.

Regularisierung

Das aktive Einschränken der NN-*Weights* (θ). Dient als Methode zur Unterstützung der *Generalizations*

Residual Skip Connection

Layer übergreifende Verbindung zur Aufrechterhaltung des Gradienten.

Robustness

wörtlich: Robustheit; Maß der Stabilität von Funktionen oder *Models*.

Sample

Trainingsbeispiel s in einem Datensatz S (vgl. Abschnitt 2.3)

Schedule

Trainingsplan, durch den die Frequenz oder Intensität des NN-Trainings gesteuert wird; z. B. durch die *Abkühlung* der *Learning Rate (LR)* (vgl. Abschnitt 2.3.2).

Seed

wörtlich: *Saatschlüssel*; Initialisierungswert eines Zufallsgenerators. Wird im Kontext von NN verwendet, um den stochastischen Einflüssen unterliegenden Prozess des Trainings reproduzierbar zu gestalten (vgl. Abschnitt 2.3).

Stride

wörtlich: Schritt(weite); Parameter, der festlegt, wie weit der *Kernel* bei jeder Applikation bewegt werden soll (vgl. Abschnitt 4.3.1).

Supervised Learning

Überwachtes Lernen

Target

Vergleichswert für die *Loss*-Funktion; Trainingsziel.

Transformer

NN-*Model-Architektur*, welche nur auf *Attention-Layer* setzt.

Weight

Untermenge der Gesamtmenge der freien und optimierbaren Parameter (θ) eines NN. Eine weitere Untermenge ist der sog. *Bias* (vgl. Abschnitt 2.3).

Zelle

siehe Neurons.

Abbildungsverzeichnis

Literatur

[1] O. O. Abayomi-Alli, R. Damaševičius, A. Qazi, M. Adedoyin-Olowe und S. Misra. „Data Augmentation and Deep Learning Methods in Sound Classification: A Systematic Review“. In: *Electronics* 11.22 (2022), S. 3795.

[2] O. O. Abayomi-Alli, R. Damaševičius, R. Maskeliūnas und S. Misra. „Few-Shot Learning with a Novel Voronoi Tessellation-Based Image Augmentation Method for Facial Palsy Detection“. In: *Electronics* 10.8 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10080978. URL: https://www.mdpi.com/2079-9292/10/8/978.

[3] C. C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. 1st. Springer Publishing Company, Incorporated, 2018. ISBN: 3319944622.

[4] R. L. Aguiar, Y. M. G. Costa und C. N. Silla. „Exploring Data Augmentation to Improve Music Genre Classification with ConvNets“. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. Juli 2018, S. 1–8. DOI: 10.1109/IJCNN.2018.8489166.

[5] R. L. Aguiar, Y. M. Costa und C. N. Silla. „Exploring data augmentation to improve music genre classification with convnets“. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, S. 1–8.

[6] T. Akiba, S. Sano, T. Yanase, T. Ohta und M. Koyama. „Optuna: A next-generation hyperparameter optimization framework“. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, S. 2623–2631.

[7] E. Amid, R. Anil, W. Kotłowski und M. K. Warmuth. „Learning from Randomly Initialized Neural Network Features“. In: *arXiv preprint arXiv:2202.06438* (2022).

[8] F. Aurenhammer. „Voronoi Diagrams—a Survey of a Fundamental Geometric Data Structure“. In: *ACM Comput. Surv.* 23.3 (Sep. 1991), S. 345–405.

[9] F. Aurenhammer, R. Klein und D.-T. Lee. *Voronoi Diagrams and Delaunay Triangulations*. 1st. USA: World Scientific Publishing Co., Inc., 2013. ISBN: 9789814447638.

[10] J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo und C. Ionescu. „Using fast weights to attend to the recent past“. In: *Advances in neural information processing systems* 29 (2016).

[11] E. Babaee, N. B. Anuar, A. W. Abdul Wahab, S. Shamshirband und A. T. Chronopoulos. „An overview of audio event detection methods from feature extraction to classification“. In: *Applied Artificial Intelligence* 31.9-10 (2017), S. 661–714.

[12] D. Bahdanau, K. Cho und Y. Bengio. „Neural machine translation by jointly learning to align and translate“. In: *arXiv preprint arXiv:1409.0473* (2014).

[13] A. Bansal und N. K. Garg. „Environmental Sound Classification: A descriptive review of the literature“. In: *Intelligent Systems with Applications* (2022), S. 200115.

[14] J. L. Bautista, Y. K. Lee und H. S. Shin. „Speech Emotion Recognition Based on Parallel CNN-Attention Networks with Multi-Fold Data Augmentation“. In: *Electronics* 11.23 (2022), S. 3935.

[15] H. L. Bear, T. Heittola, A. Mesaros, E. Benetos und T. Virtanen. „City classification from multiple real-world sound scenes“. In: *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE. 2019, S. 11–15. DOI: 10.1109/WASPAA.2019.8937271.

[16] Y. Bengio, P. Simard, P. Frasconi et al. „Learning long-term dependencies with gradient descent is difficult“. In: *IEEE transactions on neural networks* 5.2 (1994), S. 157–166.

[17] D. Beniaguev, I. Segev und M. London. „Single cortical neurons as deep artificial neural networks“. In: *Neuron* 109.17 (2021), S. 2727–2739.

[18] J. Bergstra, R. Bardenet, Y. Bengio und B. Kégl. „Algorithms for hyper-parameter optimization“. In: *Advances in neural information processing systems* 24 (2011).

[19] J. Bergstra, D. Yamins und D. Cox. „Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures“. In: *International conference on machine learning*. PMLR. 2013, S. 115–123.

[20] D. P. Bertsekas et al. „Incremental gradient, subgradient, and proximal methods for convex optimization: A survey“. In: *Optimization for Machine Learning* 2010.1-38 (2011), S. 3.

[21] L. Beyer, O. J. Hénaff, A. Kolesnikov, X. Zhai und A. v. d. Oord. „Are we done with imagenet?“ In: *arXiv preprint arXiv:2006.07159* (2020).

[22] K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu und Q. Tian. „Accurate medium-range global weather forecasting with 3D neural networks“. In: *Nature* 619.7970 (2023), S. 533–538.

[23] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle und J. Guttag. „What is the state of neural network pruning?“ In: *Proceedings of machine learning and systems* 2 (2020), S. 129–146.

[24] D. Bonet-Solà und R. M. Alsina-Pagès. „A comparative survey of feature extraction and machine learning methods in diverse acoustic environments“. In: *Sensors* 21.4 (2021), S. 1274.

[25] L. Bottou. „Online Algorithms and Stochastic Approximations“. In: *Online Learning and Neural Networks*. Hrsg. von D. Saad. revised, oct 2012. Cambridge, UK: Cambridge University Press, 1998. URL: http://leon.bottou.org/papers/bottou-98x.

[26] L. Bottou, F. E. Curtis und J. Nocedal. „Optimization methods for large-scale machine learning“. In: *Siam Review* 60.2 (2018), S. 223–311.

[27] N. Boulanger-Lewandowski, Y. Bengio und P. Vincent. „Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription“. In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. 2012, S. 1881–1888.

[28] R. N. Bracewell. *The Fourier transform and its applications*. McGraw-Hill, 1978.

[29] C. Brown, J. Chauhan, A. Grammenos, J. Han, A. Hasthanasombat, D. Spathis, T. Xia, P. Cicuta und C. Mascolo. „Exploring automatic diagnosis of covid-19 from crowdsourced respiratory sound data“. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, S. 3474–3484.

[30] T. B. Brown, D. Mané, A. Roy, M. Abadi und J. Gilmer. „Adversarial patch“. In: *arXiv preprint arXiv:1712.09665* (2017).

[31] K. P. Burnham und D. R. Anderson. *Model selection and multimodel inference : a practical information-theoretic approach*. 2nd. New York, NY: Springer-Verlag, 2002. ISBN: 0387953647.

[32] B. Camp, J. K. Mandivarapu und R. Estrada. „Continual learning with deep artificial neurons“. In: *arXiv preprint arXiv:2011.07035* (2020).

[33] O. Chang und H. Lipson. „Neural network quine“. In: *Artificial Life Conference Proceedings*. MIT Press. 2018.

[34] Y. Chauvin und D. Rumelhart. *Backpropagation: Theory, Architectures, and Applications*. Developments in connectionist theory. Lawrence Erlbaum Associates, Incorporated, 1994. ISBN: 9780203763247. URL: https://books.google.de/books?id=P%5C_ZMwgEACAAJ.

[35] K. Chen, J. Wang, F. Deng und X. Wang. „iCNN-Transformer: An improved CNN-Transformer with Channel-spatial Attention and Keyword Prediction for Automated Audio Captioning“. In: *Proc. Interspeech 2022* (2022), S. 4167–4171.

[36] P. Chen, S. Liu, H. Zhao und J. Jia. „GridMask Data Augmentation“. In: *CoRR* abs/2001.04086 (2020). URL: https://arxiv.org/abs/2001.04086.

[37] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk und Y. Bengio. „Learning phrase representations using RNN encoder-decoder for statistical machine translation“. In: *arXiv preprint arXiv:1406.1078* (2014).

[38] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser et al. „Rethinking Attention with Performers“. In: *arXiv e-prints* (2020), arXiv–2009.

[39] J. Chung, C. Gulcehre, K. Cho und Y. Bengio. „Empirical evaluation of gated recurrent neural networks on sequence modeling“. In: *arXiv preprint arXiv:1412.3555* (2014).

[40] A. Cleeremans, D. Servan-Schreiber und J. L. McClelland. „Finite state automata and simple recurrent networks“. In: *Neural computation* 1.3 (1989), S. 372–381.

[41] J. W. Cooley und J. W. Tukey. „An algorithm for the machine calculation of complex Fourier series“. In: *Mathematics of computation* 19.90 (1965), S. 297–301.

[42] W. Dai, C. Dai, S. Qu, J. Li und S. Das. *Very Deep Convolutional Neural Networks for Raw Waveforms*. 2016. arXiv: `1610.00087 [cs.SD]`.

[43] W. Dai, C. Dai, S. Qu, J. Li und S. Das. „Very deep convolutional neural networks for raw waveforms“. In: *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2017, S. 421–425.

[44] L. Deng. „The mnist database of handwritten digit images for machine learning research“. In: *IEEE Signal Processing Magazine* 29.6 (2012), S. 141–142.

[45] M. Denil, B. Shakibi, L. Dinh, M. Ranzato und N. De Freitas. „Predicting parameters in deep learning“. In: *Advances in neural information processing systems* 26 (2013).

[46] T. Devries und G. W. Taylor. „Improved Regularization of Convolutional Neural Networks with Cutout“. In: *CoRR* abs/1708.04552 (2017). URL: `http://arxiv.org/abs/1708.04552`.

[47] R. Dey und F. M. Salemt. „Gate-variants of gated recurrent unit (GRU) neural networks“. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE. 2017, S. 1597–1600.

[48] S. Dieleman und B. Schrauwen. „End-to-end learning for music audio“. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, S. 6964–6968.

[49] B. Ding, H. Qian und J. Zhou. „Activation functions and their characteristics in deep neural networks“. In: *2018 Chinese control and decision conference (CCDC)*. IEEE. 2018, S. 1836–1841.

[50] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly et al. „An image is worth 16x16 words: Transformers for image recognition at scale“. In: *arXiv preprint arXiv:2010.11929* (2020).

[51] V. Dumoulin und F. Visin. „A guide to convolution arithmetic for deep learning“. In: *ArXiv e-prints* (März 2016). eprint: 1603.07285.

[52] D. Dwibedi, I. Misra und M. Hebert. „Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection“. In: *CoRR* abs/1708.01642 (2017). URL: http://arxiv.org/abs/1708.01642.

[53] H. Eghbal-Zadeh, B. Lehner, M. Dorfer und G. Widmer. „CP-JKU submissions for DCASE-2016: A hybrid approach using binaural i-vectors and deep convolutional neural networks“. In: *IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events (DCASE)* (2016).

[54] J. L. Elman. „Finding structure in time“. In: *Cognitive science* 14.2 (1990), S. 179–211.

[55] D. Elsner, S. Langer, F. Ritz, R. Mueller und S. Illium. „Deep Neural Baselines for Computational Paralinguistics“. In: *Proc. Interspeech 2019* (2019), S. 2388–2392.

[56] A. P. Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[57] L. Engstrom, A. Ilyas, H. Salman, S. Santurkar und D. Tsipras. *Robustness (Python Library)*. 2019.

[58] P. Esling und C. Agon. „Time-series data mining“. In: *ACM Computing Surveys (CSUR)* 45.1 (2012), S. 1–34.

[59] I. Farkaš, R. Bosák und P. Gergel'. „Computational analysis of memory capacity in echo state networks“. In: *Neural Networks* 83 (2016), S. 109–120.

[60] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar und P.-A. Muller. „Deep learning for time series classification: a review“. In: *Data Mining and Knowledge Discovery* 33.4 (2019), S. 917–963.

[61] L. Ford, H. Tang, F. Grondin und J. Glass. „A Deep Residual Network for Large-Scale Acoustic Scene Analysis“. In: *Proc. Interspeech 2019* (2019), S. 2568–2572.

[62] J. Frankle und M. Carbin. „The lottery ticket hypothesis: Finding sparse, trainable neural networks“. In: *arXiv preprint arXiv:1803.03635* (2018).

[63] M. Freitag, S. Amiriparian, S. Pugachevskiy, N. Cummins und B. Schuller. „audeep: Unsupervised learning of representations from audio with deep recurrent neural networks“. In: *The Journal of Machine Learning Research* 18.1 (2017), S. 6340–6344.

[64] M. Friedrich, S. Illium, P.-A. Fayolle und C. Linnhoff-Popien. „A Hybrid Approach for Segmenting and Fitting Solid Primitives to 3D Point Clouds." In: *VISIGRAPP (1: GRAPP)*. 2020, S. 38–48.

[65] Z. Fu, G. Lu, K. M. Ting und D. Zhang. „A survey of audio-based music classification and annotation". In: *IEEE transactions on multimedia* 13.2 (2010), S. 303–319.

[66] K. Fukushima und S. Miyake. „Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, 1982, S. 267–285.

[67] T. Gabor, S. Illium, A. Mattausch, L. Belzner und C. Linnhoff-Popien. „Self-replication in neural networks". In: *ALIFE 2019: The 2019 Conference on Artificial Life*. MIT Press. 2019, S. 424–431.

[68] T. Gabor, S. Illium, M. Zorn, C. Lenta, A. Mattausch, L. Belzner und C. Linnhoff-Popien. „Self-Replication in Neural Networks". In: *Artificial Life* (Juni 2022), S. 205–223. ISSN: 1064-5462. DOI: 10.1162/artl_a_00359. URL: https://doi.org/10.1162/artl%5C_a%5C_00359.

[69] T. Gabor, S. Illium, M. Zorn und C. Linnhoff-Popien. „Goals for Self-Replicating Neural Networks". In: *ALIFE 2021: The 2021 Conference on Artificial Life*. MIT Press. 2021.

[70] J. Geiping, M. Goldblum, G. Somepalli, R. Shwartz-Ziv, T. Goldstein und A. G. Wilson. „How Much Data Are Augmentations Worth? An Investigation into Scaling Laws, Invariance, and Implicit Regularization". In: *arXiv preprint arXiv:2210.06441* (2022).

[71] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge und F. A. Wichmann. „Shortcut learning in deep neural networks". In: *Nature Machine Intelligence* 2.11 (2020), S. 665–673.

[72] F. Gers. „Long short-term memory in recurrent neural networks". Diss. Universität Hannover, 2001.

[73] F. A. Gers und J. Schmidhuber. „Recurrent nets that time and count". In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. Bd. 3. IEEE. 2000, S. 189–194.

[74] F. A. Gers, N. N. Schraudolph und J. Schmidhuber. „Learning precise timing with LSTM recurrent networks". In: *Journal of machine learning research* 3.Aug (2002), S. 115–143.

[75] C. L. Giles, C. B. Miller, D. Chen, H.-H. Chen, G.-Z. Sun und Y.-C. Lee. „Learning and extracting finite state automata with second-order recurrent neural networks". In: *Neural Computation* 4.3 (1992), S. 393–405.

[76] X. Glorot und Y. Bengio. „Understanding the difficulty of training deep feedforward neural networks“. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop und Conference Proceedings. 2010, S. 249–256.

[77] X. Glorot, A. Bordes und Y. Bengio. „Deep sparse rectifier neural networks“. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop und Conference Proceedings. 2011, S. 315–323.

[78] Y. Gong, Y.-A. Chung und J. Glass. „AST: Audio Spectrogram Transformer“. In: *Proc. Interspeech 2021*. 2021, S. 571–575. DOI: 10.21437/Interspeech.2021-698.

[79] L. Gonon, L. Grigoryeva und J.-P. Ortega. „Memory and forecasting capacities of nonlinear recurrent networks“. In: *Physica D: Nonlinear Phenomena* 414 (2020), S. 132721.

[80] I. Goodfellow, Y. Bengio und A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[81] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville und Y. Bengio. „Generative adversarial networks“. In: *Communications of the ACM* 63.11 (2020), S. 139–144.

[82] I. J. Goodfellow, J. Shlens und C. Szegedy. „Explaining and harnessing adversarial examples“. In: *arXiv preprint arXiv:1412.6572* (2014).

[83] I. J. Goodfellow, Y. Bengio und A. Courville. *Deep Learning*. MIT Press, 2016.

[84] A. Graves. „Generating sequences with recurrent neural networks“. In: *arXiv preprint arXiv:1308.0850* (2013).

[85] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink und J. Schmidhuber. „LSTM: A search space odyssey“. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2017), S. 2222–2232.

[86] K. Gregor, I. Danihelka, A. Graves, D. Rezende und D. Wierstra. „Draw: A recurrent neural network for image generation“. In: *International conference on machine learning*. PMLR. 2015, S. 1462–1471.

[87] Q. Han, Z. Fan, Q. Dai, L. Sun, M.-M. Cheng, J. Liu und J. Wang. „On the connection between local attention and dynamic depth-wise convolution“. In: *arXiv preprint arXiv:2106.04263* (2021).

[88] S. Han, H. Mao und W. J. Dally. „Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding“. In: *arXiv preprint arXiv:1510.00149* (2015).

[89] S. Han, J. Pool, J. Tran und W. Dally. „Learning both weights and connections for efficient neural network“. In: *Advances in neural information processing systems* 28 (2015).

[90] S. Hanson und L. Pratt. „Comparing biases for minimal network construction with back-propagation“. In: *Advances in neural information processing systems* 1 (1988).

[91] P. Haubrick und J. Ye. „Robust audio sensing with multi-sound classification“. In: *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*. IEEE. 2019, S. 1–7. DOI: 10.1109/PERCOM.2019.8767402.

[92] K. He, X. Zhang, S. Ren und J. Sun. „Deep residual learning for image recognition“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, S. 770–778.

[93] D. Hendrycks und K. Gimpel. „Gaussian error linear units (gelus)“. In: *arXiv preprint arXiv:1606.08415* (2016).

[94] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss und K. Wilson. *CNN Architectures for Large-Scale Audio Classification*. 2016. arXiv: 1609.09430 [cs.SD].

[95] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever und R. R. Salakhutdinov. „Improving neural networks by preventing co-adaptation of feature detectors“. In: *arXiv preprint arXiv:1207.0580* (2012).

[96] S. Hochreiter und J. Schmidhuber. „Long short-term memory“. In: *Neural Computation* 9.8 (1997), S. 1735–1780.

[97] Y. Hoshen, R. J. Weiss und K. W. Wilson. „Speech acoustic modeling from raw multichannel waveforms“. In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2015, S. 4624–4628.

[98] J. Hurtado, A. Raymond und A. Soto. „Optimizing reusable knowledge for continual learning via metalearning“. In: *Advances in Neural Information Processing Systems* 34 (2021), S. 14150–14162.

[99] F. Hutter, H. Hoos und K. Leyton-Brown. „An Efficient Approach for Assessing Hyperparameter Importance“. In: *Proceedings of the 31st International Conference on Machine Learning*. Hrsg. von E. P. Xing und T. Jebara. Bd. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, Juni 2014, S. 754–762. URL: https://proceedings.mlr.press/v32/hutter14.html.

[100] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally und K. Keutzer. „SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size“. In: *CoRR* abs/1602.07360 (2016). URL: http://arxiv.org/abs/1602.07360.

[101] S. Illium, R. Müller, A. Sedlmeier und C. Linnhoff-Popien. „Surgical Mask Detection with Convolutional Neural Networks and Data Augmentations on Spectrograms“. In: *Proc. Interspeech 2020* (2020), S. 2052–2056.

[102] S. Illium, R. Müller, A. Sedlmeier und C.-L. Popien. „Visual transformers for primates classification and covid detection“. In: *22nd Annual Conference of the International Speech Communication Association, INTERSPEECH 2021*. 2021, S. 4341–4345.

[103] S. Illium, M. Zorn, C. Lenta, M. Kölle, C. Linnhoff-Popien und T. Gabor. „Constructing Organism Networks from Collaborative Self-Replicators“. In: *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2022, S. 1268–1275.

[104] S. Illium., G. Griffin., M. Kölle., M. Zorn., J. Nüßlein. und C. Linnhoff-Popien. „VoronoiPatches: Evaluating a New Data Augmentation Method“. In: *Proceedings of the 15th International Conference on Agents and Artificial Intelligence - Volume 3: ICAART*. INSTICC. SciTePress, 2023, S. 350–357. ISBN: 978-989-758-623-1. DOI: 10.5220/0011670000003393.

[105] S. Illium., T. Schillman., R. Müller., T. Gabor. und C. Linnhoff-Popien. „Empirical Analysis of Limits for Memory Distance in Recurrent Neural Networks“. In: *Proceedings of the 14th International Conference on Agents and Artificial Intelligence - Volume 3: ICAART,* INSTICC. SciTePress, 2022, S. 308–315. ISBN: 978-989-758-547-0. DOI: 10.5220/0010818500003116.

[106] H. Inoue. „Data Augmentation by Pairing Samples for Images Classification“. In: *CoRR* abs/1801.02929 (2018). URL: http://arxiv.org/abs/1801.02929.

[107] S. Ioffe und C. Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *International Conference on Machine Learning*. 2015, S. 448–456.

[108] M. Ishaq, S. Kwon et al. „A CNN-assisted deep echo state network using multiple time-scale dynamic learning reservoirs for generating short-term solar energy forecasting“. In: *Sustainable Energy Technologies and Assessments* 52 (2022), S. 102275.

[109] B. K. Iwana und S. Uchida. „An empirical survey of data augmentation for time series classification with neural networks“. In: *Plos one* 16.7 (2021), e0254841.

[110] Z. Jackson. „Free spoken digit dataset (fsdd)“. In: *Technical report, Technical report* (2016).

[111] H. Jaeger. „Short term memory in echo state networks. gmd-report 152“. In: *GMD-German National Research Institute for Computer Science*. Citeseer. 2002.

[112] H. Jaeger. „The “echo state” approach to analysing and training recurrent neural networks-with an erratum note“. In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148.34 (2001), S. 13.

[113] H. Jaeger. „Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and theëcho state networkäpproach“. In: (2002).

[114] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman und J. Carreira. „Perceiver: General perception with iterative attention“. In: *International conference on machine learning*. PMLR. 2021, S. 4651–4664.

[115] N. Jaitly und G. Hinton. „Learning a better representation of speech soundwaves using restricted boltzmann machines“. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2011, S. 5884–5887.

[116] N. Japkowicz et al. „Learning from imbalanced data sets: a comparison of various strategies“. In: *AAAI workshop on learning from imbalanced data sets*. Bd. 68. Citeseer. 2000, S. 10–15.

[117] A. J. Jerri. „The Shannon sampling theorem—Its various extensions and applications: A tutorial review“. In: *Proceedings of the IEEE* 65.11 (1977), S. 1565–1596.

[118] W. Jiang, K. Zhang, N. Wang und M. Yu. „MeshCut data augmentation for deep learning in computer vision“. In: *PLoS One* 15.12 (2020), e0243613.

[119] A. A. Johnson, M. Q. Ott und M. Dogucu. *Bayes Rules!: An Introduction to Applied Bayesian Modeling*. CRC Press, 2022.

[120] D. Kang, Y. Sun, D. Hendrycks, T. Brown und J. Steinhardt. „Testing robustness against unforeseen adversaries“. In: *arXiv preprint arXiv:1908.08016* (2019).

[121] A. Karpathy, J. Johnson und L. Fei-Fei. „Visualizing and understanding recurrent networks“. In: *arXiv preprint arXiv:1506.02078* (2015).

[122] S. A. Kauffman et al. *The origins of order: Self-organization and selection in evolution*. Oxford University Press, USA, 1993.

[123] A. P. Kefauver und D. Patschke. *Fundamentals of digital audio*. Bd. 22. AR Editions, Inc., 2007.

[124] J. Kiefer. „J. Wolfowitz Stochastic estimation of a regression function“. In: *Annals of Mathematical Statistics* 23 (1952), S. 462–466.

[125] D. P. Kingma und J. Ba. „Adam: A method for stochastic optimization“. In: *arXiv preprint arXiv:1412.6980* (2014).

[126] D. P. Kingma und J. Ba. „Adam: A Method for Schocastic Optimization“. In: *ArXiv* abs/1412.6980 (2014).

[127] T. Ko, V. Peddinti, D. Povey und S. Khudanpur. „Audio augmentation for speech recognition“. In: *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.

[128] M. Kölle, S. Illium, M. Zorn, J. Nüßlein, P. Suchostawski und C. Linnhoff-Popien. „Improving Primate Sounds Classification Using Binary Presorting for Deep Learning“. In: *International Conference on Deep Learning Theory and Applications*. Springer. 2023, S. 19–34.

[129] A. Krizhevsky, I. Sutskever und G. E. Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems*. 2012, S. 1097–1105.

[130] A. Krizhevsky, I. Sutskever und G. E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *Commun. ACM* 60.6 (Mai 2017), S. 84–90.

[131] Y. LeCun, Y. Bengio und G. Hinton. „Deep Learning“. In: *Nature* 521.7553 (2015), S. 436.

[132] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard und L. Jackel. „Handwritten digit recognition with a back-propagation network“. In: *Advances in neural information processing systems* 2 (1989).

[133] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard und L. D. Jackel. „Backpropagation applied to handwritten zip code recognition“. In: *Neural computation* 1.4 (1989), S. 541–551.

[134] Y. LeCun, L. Bottou, Y. Bengio und P. Haffner. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11 (1998), S. 2278–2324.

[135] Y. A. LeCun, L. Bottou, G. B. Orr und K.-R. Müller. „Efficient backprop“. In: *Neural networks: Tricks of the trade*. Springer, 2012, S. 9–48.

[136] J. Lee, M. Zaheer, M. Astrid und S.-I. Lee. „SmoothMix: a Simple Yet Effective Data Augmentation to Train Robust Classifiers“. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Juni 2020, S. 3264–3274.

[137] J. Lee, T. Kim, J. Park und J. Nam. „Raw waveform-based audio classification using sample-level CNN architectures“. In: *arXiv preprint arXiv:1712.00866* (2017).

[138] J. Lee, J. Park, K. L. Kim und J. Nam. „Samplecnn: End-to-end deep convolutional neural networks using very small filters for music classification“. In: *Applied Sciences* 8.1 (2018), S. 150.

[139] A.-M. Legendre. „New methods for the determination of orbits of comets“. In: *Courcier, Paris* (1805).

[140] X. Li, F. Bi, X. Yang und X. Bi. „An Echo State Network With Improved Topology for Time Series Prediction“. In: *IEEE Sensors Journal* 22.6 (2022), S. 5869–5878. DOI: 10.1109/JSEN.2022.3148742.

[141] Z. Li und G. Tanaka. „Multi-reservoir echo state networks with sequence resampling for nonlinear time-series prediction“. In: *Neurocomputing* 467 (2022), S. 115–129.

[142] T. P. Lillicrap und A. Santoro. „Backpropagation through time and the brain“. In: *Current opinion in neurobiology* 55 (2019), S. 82–89.

[143] M. Lin, Q. Chen und S. Yan. „Network in network“. In: *arXiv preprint arXiv:1312.4400* (2013).

[144] T.-Y. Lin, P. Goyal, R. Girshick, K. He und P. Dollár. „Focal loss for dense object detection“. In: *Proceedings of the IEEE international conference on computer vision*. 2017, S. 2980–2988.

[145] G. Liu, K. Zhang und M. Lv. „ASKs: Convolution with any-shape kernels for efficient neural networks“. In: *Neurocomputing* 446 (2021), S. 32–49.

[146] S. Liu, A. Mallol-Ragolta, E. Parada-Cabaleiro, K. Qian, X. Jing, A. Kathan, B. Hu und B. W. Schuller. „Audio self-supervised learning: A survey“. In: *Patterns* 3.12 (2022), S. 100616.

[147] Y. Liu, L. Shan, D. Yu, L. Zeng und M. Yang. „An echo state network with attention mechanism for production prediction in reservoirs“. In: *Journal of Petroleum Science and Engineering* 209 (2022), S. 109920.

[148] C. Lordelo, E. Benetos, S. Dixon und S. Ahlbäck. „Investigating kernel shapes and skip connections for deep learning-based harmonic-percussive separation“. In: *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE. 2019, S. 40–44.

[149] I. Loshchilov und F. Hutter. „Decoupled Weight Decay Regularization“. In: *International Conference on Learning Representations*. 2018.

[150] I. Loshchilov und F. Hutter. „Decoupled weight decay regularization“. In: *arXiv preprint arXiv:1711.05101* (2017).

[151] I. Loshchilov und F. Hutter. „Sgdr: Stochastic gradient descent with warm restarts“. In: *arXiv preprint arXiv:1608.03983* (2016).

[152] Y. Lu und F. M. Salem. „Simplified gating in long short-term memory (LSTM) recurrent neural networks“. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE. 2017, S. 1601–1604.

[153] M. Lukoševičius. „A practical guide to applying echo state networks“. In: *Neural Networks: Tricks of the Trade: Second Edition* (2012), S. 659–686.

[154] M. Lukoševičius und H. Jaeger. „Reservoir computing approaches to recurrent neural network training“. In: *Computer science review* 3.3 (2009), S. 127–149.

[155] M.-T. Luong, H. Pham und C. D. Manning. „Effective approaches to attention-based neural machine translation“. In: *arXiv preprint arXiv:1508.04025* (2015).

[156] E. Ma. *NLP Augmentation*. `https://github.com/makcedward/nlpaug`. 2019.

[157] A. L. Maas, A. Y. Hannun und A. Y. Ng. „Rectifier nonlinearities improve neural network acoustic models“. In: *Proc. icml*. Bd. 30. 1. 2013, S. 3. DOI: `10.1109/WASPAA.2019.8937271`.

[158] W. Maass, T. Natschläger und H. Markram. „Real-time computing without stable states: A new framework for neural computation based on perturbations“. In: *Neural computation* 14.11 (2002), S. 2531–2560.

[159] C. D. Manning. *Introduction to information retrieval*. Syngress Publishing, 2008.

[160] M. Matsumoto und T. Nishimura. „Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator“. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998), S. 3–30.

[161] N. M. Mayer. „Echo state condition at the critical point“. In: *Entropy* 19.1 (2017), S. 3.

[162] V. Mnih, N. Heess, A. Graves et al. „Recurrent models of visual attention“. In: *Advances in neural information processing systems* 27 (2014).

[163] M. M. Mohamed, M. A. Nessiem, A. Batliner, C. Bergler, S. Hantke, M. Schmitt, A. Baird, A. Mallol-Ragolta, V. Karas, S. Amiriparian et al. „Face mask recognition from audio: The MASC database and an overview on the mask challenge“. In: *Pattern Recognition* 122 (2022), S. 108361.

[164] D. Molchanov, A. Ashukha und D. Vetrov. „Variational dropout sparsifies deep neural networks“. In: *International Conference on Machine Learning*. PMLR. 2017, S. 2498–2507.

[165] M. Mozer. „A Focused Backpropagation Algorithm for Temporal Pattern Recognition“. In: *Complex Systems* 3 (Jan. 1995).

[166] R. Müller, S. Illium und C. Linnhoff-Popien. „A Deep and Recurrent Architecture for Primate Vocalization Classification.“ In: *Interspeech*. 2021, S. 461–465.

[167] R. Müller, S. Illium, F. Ritz und K. Schmid. „Analysis of Feature Representations for Anomalous Sound Detection“. In: *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,* INSTICC. SciTePress, 2021, S. 97–106. ISBN: 978-989-758-484-8. DOI: `10.5220/0010226800970106`.

[168] R. Müller, S. Illium, F. Ritz, T. Schröder, C. Platschek, J. Ochs und C. Linnhoff-Popien. „Acoustic Leak Detection in Water Networks“. In: Jan. 2021, S. 306–313. DOI: 10.5220/0010295403060313.

[169] R. Müller, S. Illium, F. Ritz, T. Schröder, C. Platschek, J. Ochs und C. Linnhoff-Popien. „Acoustic leak detection in water networks“. In: *arXiv preprint arXiv:2012.06280* (2020).

[170] R. Müller, F. Ritz, S. Illium und C. Linnhoff-Popien. „Acoustic anomaly detection for machine sounds based on image transfer learning“. In: *arXiv preprint arXiv:2006.03429* (2020).

[171] A. Nagrani, S. Yang, A. Arnab, A. Jansen, C. Schmid und C. Sun. „Attention bottlenecks for multimodal fusion“. In: *Advances in Neural Information Processing Systems* 34 (2021), S. 14200–14213.

[172] V. Nair und G. E. Hinton. „Rectified linear units improve restricted boltzmann machines“. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, S. 807–814.

[173] S. Narang, E. Elsen, G. Diamos und S. Sengupta. „Exploring sparsity in recurrent neural networks“. In: *arXiv preprint arXiv:1704.05119* (2017).

[174] A. Ng und M. Jordan. „On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes“. In: *Advances in neural information processing systems* 14 (2001).

[175] T. Nguyen, A. Fuchs und F. Pernkopf. „Acoustic Scene Classification Using Deep Mixtures of Pre-trained Convolutional Neural Networks“. In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. 2019, S. 871–875.

[176] T. Nguyen und S. Sanner. „Algorithms for direct 0–1 loss optimization in binary classification“. In: *International conference on machine learning*. PMLR. 2013, S. 1085–1093.

[177] H. Oh, H. Kim und J. Seo. „Scheduling Optimization Techniques for Neural Network Training“. In: *arXiv preprint arXiv:2110.00929* (2021).

[178] A. Okabe, B. Boots, K. Sugihara und S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. 2nd. Series in Probability and Statistics. John Wiley und Sons, Inc., 2000.

[179] K. Palanisamy, D. Singhania und A. Yao. „Rethinking CNN models for audio classification“. In: *arXiv preprint arXiv:2007.11154* (2020).

[180] D. Palaz, R. Collobert und M. M. Doss. „Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks“. In: *arXiv preprint arXiv:1304.1018* (2013).

[181] D. S. Park, Y. Zhang, C. Chiu, Y. Chen, B. Li, W. Chan, Q. V. Le und Y. Wu. „Specaugment on Large Scale Datasets“. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, S. 6879–6883.

[182] E. Parzen. „On estimation of a probability density function and mode“. In: *The annals of mathematical statistics* 33.3 (1962), S. 1065–1076.

[183] R. Pascanu, T. Mikolov und Y. Bengio. „On the difficulty of training recurrent neural networks“. In: *International conference on machine learning*. Pmlr. 2013, S. 1310–1318.

[184] H. Phan, P. Koch, F. Katzberg, M. Maass, R. Mazur und A. Mertins. „ATTENTION-BASED CNN WITH GENERALIZED LABEL TREE EMBEDDING FOR AUDIO SCENE CLASSIFICATION“. In: *A=(A+ AT)* 2 (), S. 2.

[185] S. S. R. Phaye, E. Benetos und Y. Wang. „SubSpectralNet–using sub-spectrogram based convolutional neural networks for acoustic scene classification“. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, S. 825–829.

[186] S. S. R. Phaye, A. Sikka, A. Dhall und D. R. Bathula. „Multi-level dense capsule networks“. In: *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part V 14*. Springer. 2019, S. 577–592.

[187] K. J. Piczak. „Environmental sound classification with convolutional neural networks“. In: *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE. 2015, S. 1–6.

[188] J. Pons und X. Serra. „Randomly weighted CNNs for (music) audio classification“. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, S. 336–340.

[189] J. Pons, O. Slizovskaia, R. Gong, E. Gómez und X. Serra. „Timbre analysis of music audio signals with convolutional neural networks“. In: *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE. 2017, S. 2744–2748.

[190] H. Purohit, R. Tanabe, K. Ichige, T. Endo, Y. Nikaido, K. Suefusa und Y. Kawaguchi. „MIMII DATASET: SOUND DATASET FOR MALFUNCTIONING INDUSTRIAL MACHINE INVESTIGATION AND INSPECTION“. In: *Acoustic Scenes and Events 2019 Workshop (DCASE2019)*. 2019, S. 209.

[191] R. H. Randall. *An introduction to acoustics*. Courier Corporation, 2012.

[192] E. Randazzo, L. Versari und A. Mordvintsev. „Recursively Fertile Self-replicating Neural Agents“. In: *ALIFE 2021: The 2021 Conference on Artificial Life*. MIT Press. 2021.

[193] Y. Ren, D. Liu, Q. Xiong, J. Fu und L. Wang. „Spec-resnet: a general audio steganalysis scheme based on deep residual network of spectrogram“. In: *arXiv preprint arXiv:1901.06838* (2019).

[194] H. Robbins und S. Monro. „A stochastic approximation method“. In: *The annals of mathematical statistics* (1951), S. 400–407.

[195] A. J. Robinson und F. Fallside. *The utility driven dynamic error propagation network*. Bd. 11. University of Cambridge Department of Engineering Cambridge, 1987.

[196] P. Rodriguez, J. Wiles und J. L. Elman. „A recurrent neural network that learns to count“. In: *Connection Science* 11.1 (1999), S. 5–40.

[197] Y. Roh, G. Heo und S. E. Whang. „A survey on data collection for machine learning: a big data-ai integration perspective“. In: *IEEE Transactions on Knowledge and Data Engineering* 33.4 (2019), S. 1328–1347.

[198] A. Rosenberg. „Classifying skewed data: Importance weighting to optimize average recall“. In: *Thirteenth Annual Conference of the International Speech Communication Association*. 2012.

[199] S. Ruder. „An overview of gradient descent optimization algorithms“. In: *ArXiv* abs/1609.04747 (2016).

[200] D. E. Rumelhart, G. E. Hinton und R. J. Williams. „Learning representations by back-propagating errors“. In: *nature* 323.6088 (1986), S. 533–536.

[201] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg und L. Fei-Fei. „ImageNet Large Scale Visual Recognition Challenge“. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), S. 211–252. DOI: 10.1007/s11263-015-0816-y.

[202] J. Salamon und J. P. Bello. „Deep convolutional neural networks and data augmentation for environmental sound classification“. In: *IEEE Signal processing letters* 24.3 (2017), S. 279–283.

[203] J. Salamon, C. Jacoby und J. P. Bello. „A dataset and taxonomy for urban sound research“. In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, S. 1041–1044.

[204] J. Schlüter und T. Grill. „Exploring data augmentation for improved singing voice detection with neural networks.“ In: *ISMIR*. 2015, S. 121–126.

[205] J. Schmidhuber. „Deep learning in neural networks: An overview“. In: *Neural networks* 61 (2015), S. 85–117.

[206] B. Schuller, A. Batliner, C. Bergler, C. Mascolo, J. Han, I. Lefter, H. Kaya, S. Amiriparian, A. Baird, L. Stappen et al. „The INTERSPEECH 2021 Computational Paralinguistics Challenge: COVID-19 Cough, COVID-19 Speech, Escalation & Primates“. In: *Proceedings INTERSPEECH 2021, 22nd Annual Conference of the International Speech Communication Association*. Brno, Czechia: ISCA, Sep. 2021.

[207] B. Schuller. „Openxbow: introducing the passau open-source crossmodal bag-of-words toolkit“. In: (2017).

[208] B. Schuller, A. Batliner, C. Bergler, E.-M. Messner, A. Hamilton, S. Amiriparian, A. Baird, G. Rizos, M. Schmitt, L. Stappen, H. Baumeister, A. D. MacIntyre und S. Hantke. „The INTERSPEECH 2020 Computational Paralinguistics Challenge: Elderly Emotion, Breathing & Masks“. In: *Proceedings of Interspeech*. Shanghai, China, Sep. 2020, 5 pages.

[209] B. Schuller, A. Batliner, S. Steidl und D. Seppi. „Recognising realistic emotions and affect in speech: State of the art and lessons learnt from the first challenge“. In: *Speech communication* 53.9-10 (2011), S. 1062–1087.

[210] R. Schwartz, J. Dodge, N. A. Smith und O. Etzioni. „Green ai“. In: *Communications of the ACM* 63.12 (2020), S. 54–63.

[211] S. Sen, A. Dutta und N. Dey. *Audio processing and speech recognition: concepts, techniques and research overviews*. Springer, 2019.

[212] M. Sensoy, L. Kaplan und M. Kandemir. „Evidential deep learning to quantify classification uncertainty“. In: *Advances in neural information processing systems* 31 (2018).

[213] C. E. Shannon. „Communication in the presence of noise“. In: *Proceedings of the IRE* 37.1 (1949), S. 10–21.

[214] G. Sharma, K. Umapathy und S. Krishnan. „Trends in audio signal feature extraction methods“. In: *Applied Acoustics* 158 (2020), S. 107020.

[215] C. Shorten und T. M. Khoshgoftaar. „A survey on Image Data Augmentation for Deep Learning“. In: *Journal of Big Data* 6.1 (2019), S. 1–48. DOI: 10.1186/s40537-019-0197-0. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0.

[216] K. Simonyan und A. Zisserman. „Very deep convolutional networks for large-scale image recognition“. In: *arXiv preprint arXiv:1409.1556* (2014).

[217] K. K. Singh, H. Yu, A. Sarmasi, G. Pradeep und Y. J. Lee. „Hide-and-Seek: A Data Augmentation Technique for Weakly-Supervised Localization and Beyond“. In: *CoRR* abs/1811.02545 (2018). URL: http://arxiv.org/abs/1811.02545.

[218] S. W. Smith et al. *The scientist and engineer's guide to digital signal processing*. 1997.

[219] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever und R. Salakhutdinov. „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“. In: *Journal of Machine Learning Research* 15.56 (2014), S. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[220] K. O. Stanley und R. Miikkulainen. „Evolving neural networks through augmenting topologies“. In: *Evolutionary computation* 10.2 (2002), S. 99–127.

[221] S. S. Stevens, J. Volkmann und E. B. Newman. „A scale for the measurement of the psychological magnitude pitch“. In: *The journal of the acoustical society of america* 8.3 (1937), S. 185–190.

[222] E. Stow und P. H. Kelly. „Convolutional kernel function algebra“. In: *Frontiers in Computer Science* 4 (2022).

[223] C. Summers und M. J. Dinneen. „Improved Mixed-Example Data Augmentation“. In: *CoRR* abs/1805.11272 (2018). URL: http://arxiv.org/abs/1805.11272.

[224] Y.-Y. Sun, Y. Zhang und Z.-H. Zhou. „Multi-label learning with weak label“. In: *Proceedings of the AAAI conference on artificial intelligence*. Bd. 24. 1. 2010, S. 593–598.

[225] K. Suzuki. *Artificial Neural Networks*. Rijeka: IntechOpen, Jan. 2013. DOI: 10.5772/3409. URL: https://doi.org/10.5772/3409.

[226] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke und A. Rabinovich. „Going deeper with convolutions“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, S. 1–9.

[227] R. Takahashi, T. Matsubara und K. Uehara. „RICAP: Random Image Cropping and Patching Data Augmentation for Deep CNNs“. In: *Proceedings of The 10th Asian Conference on Machine Learning*. Hrsg. von J. Zhu und I. Takeuchi. Bd. 95. Proceedings of Machine Learning Research. PMLR, Nov. 2018, S. 786–798. URL: https://proceedings.mlr.press/v95/takahashi18a.html.

[228] N. H. Tandel, H. B. Prajapati und V. K. Dabhi. „Voice recognition and voice comparison using machine learning techniques: A survey“. In: *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE. 2020, S. 459–465.

[229] C. Tao, D. Shanxu und C. Changsong. „Forecasting power output for grid-connected photovoltaic power system without using solar radiation measurement“. In: *The 2nd International Symposium on Power Electronics for Distributed Generation Systems*. IEEE. 2010, S. 773–777.

[230] J. Thickstun, Z. Harchaoui und S. Kakade. *Learning Features of Music from Scratch*. 2016. arXiv: 1611.09827 [stat.ML].

[231] Y. Tokozume, Y. Ushiku und T. Harada. „Between-class Learning for Image Classification“. In: *CoRR* abs/1711.10284 (2017). URL: http://arxiv.org/abs/1711.10284.

[232] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit et al. „Mlp-mixer: An all-mlp architecture for vision“. In: *Advances in Neural Information Processing Systems* 34 (2021).

[233] W. Ullah, T. Hussain, Z. A. Khan, U. Haroon und S. W. Baik. „Intelligent dual stream CNN and echo state network for anomaly detection“. In: *Knowledge-Based Systems* 253 (2022), S. 109456.

[234] A. Uncini. *Digital Audio Processing Fundamentals*. Bd. 21. Springer Nature, 2023.

[235] V. N. Vapnik. „The support vector method“. In: *International Conference on Artificial Neural Networks*. Springer. 1997, S. 261–271.

[236] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser und I. Polosukhin. „Attention is all you need“. In: *arXiv preprint arXiv:1706.03762* (2017).

[237] S. Venkataramanan, E. Kijak, L. Amsaleg und Y. Avrithis. „AlignMixup: Improving Representations By Interpolating Aligned Features“. In: *arXiv preprint arXiv:2103.15375* (2021).

[238] P. Verma. „Large Scale Audio Understanding without Transformers/-Convolutions/BERTs/Mixers/Attention/RNNs or....“ In: *arXiv preprint arXiv:2110.03183* (2021).

[239] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. Courville und Y. Bengio. „Renet: A recurrent neural network based alternative to convolutional networks“. In: *arXiv preprint arXiv:1505.00393* (2015).

[240] H. Wang, Y. Liu, P. Lu, Y. Luo, D. Wang und X. Xu. „Echo state network with logistic mapping and bias dropout for time series prediction“. In: *Neurocomputing* 489 (2022), S. 196–210.

[241] Q. Wang, Y. Ma, K. Zhao und Y. Tian. „A Comprehensive Survey of Loss Functions in Machine Learning“. In: *Annals of Data Science* 9 (Apr. 2022). DOI: 10.1007/s40745-020-00253-5.

[242] S. Wang, B. Z. Li, M. Khabsa, H. Fang und H. Ma. „Linformer: Self-attention with linear complexity“. In: *arXiv preprint arXiv:2006.04768* (2020).

[243] Z. Wang, A. Bovik und H. Sheikh. „Structural Similarity Based Image Quality Assessment“. In: *Digital Video Image Quality and Perceptual Coding, Marcel Dekker Series in Signal Processing and Communications* (Nov. 2005). DOI: 10.1201/9781420027822.ch7.

[244] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan und L. Sun. „Transformers in time series: A survey“. In: *arXiv preprint arXiv:2202.07125* (2022).

[245] P. J. Werbos. „Generalization of backpropagation with application to a recurrent gas market model“. In: *Neural Networks* 1.4 (1988), S. 339–356. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(88)90007-X. URL: https://www.sciencedirect.com/science/article/pii/089360808890007X.

[246] R. Westwater. „Digital audio presentation and compression“. In: *Handbook of Multimedia Computing* 5 (1998), S. 135.

[247] S. Wiesler und H. Ney. „A convergence analysis of log-linear training“. In: *Advances in Neural Information Processing Systems* 24 (2011).

[248] Y. Wu, H. Mao und Z. Yi. „Audio classification using attention-augmented convolutional neural network“. In: *Knowledge-Based Systems* 161 (2018), S. 90–100.

[249] X. Yan. *Linear Regression Analysis: Theory and Computing*. World Scientific Publishing Company Pte Limited, 2009. ISBN: 9789812834119. URL: `https://books.google.de/books?id=MjNv6rGv8NIC`.

[250] I. B. Yildiz, H. Jaeger und S. J. Kiebel. „Re-visiting the echo state property“. In: *Neural networks* 35 (2012), S. 1–9.

[251] W. Yu, C. Si, P. Zhou, M. Luo, Y. Zhou, J. Feng, S. Yan und X. Wang. „Metaformer baselines for vision“. In: *arXiv preprint arXiv:2210.13452* (2022).

[252] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe und Y. Yoo. „CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features“. In: *CoRR* abs/1905.04899 (2019). URL: `http://arxiv.org/abs/1905.04899`.

[253] A. Zhang, Z. C. Lipton, M. Li und A. J. Smola. „Dive into deep learning“. In: *arXiv preprint arXiv:2106.11342* (2021).

[254] C. Zhang, S. Bengio, M. Hardt, B. Recht und O. Vinyals. „Understanding deep learning requires rethinking generalization“. In: *arXiv preprint arXiv:1611.03530* (2016).

[255] H. Zhang, M. Cissé, Y. N. Dauphin und D. Lopez-Paz. „mixup: Beyond Empirical Risk Minimization“. In: *CoRR* abs/1710.09412 (2017). URL: `http://arxiv.org/abs/1710.09412`.

[256] Z. Zhong, L. Zheng, G. Kang, S. Li und Y. Yang. „Random Erasing Data Augmentation“. In: *CoRR* abs/1708.04896 (2017). URL: `http://arxiv.org/abs/1708.04896`.

[257] M. Zhu und S. Gupta. „To prune, or not to prune: exploring the efficacy of pruning for model compression“. In: *arXiv preprint arXiv:1710.01878* (2017).

[258] J. A. Zwerts, J. Treep, C. S. Kaandorp, F. Meewis, A. C. Koot und H. Kaya. „Introducing a central african primate vocalisation dataset for automated species classification“. In: *arXiv preprint arXiv:2101.10390* (2021).