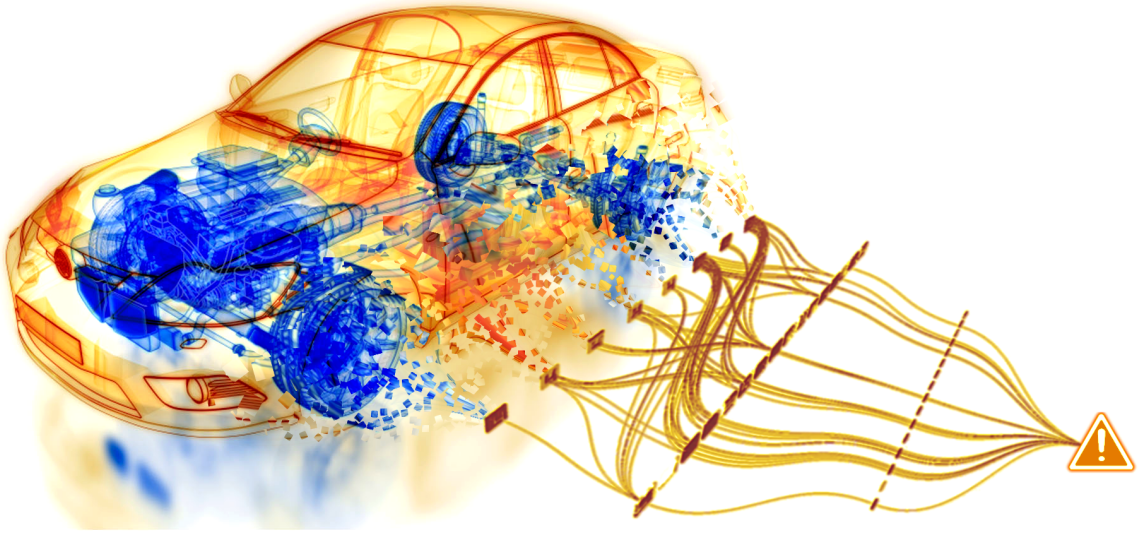
Dämpferdefektdiagnose mittels der Deep Learning Architektur Convolutional Neural Networks

Diagnosing Damper Defects using the Deep Learning Architecture Convolutional Neural Networks



Wissenschaftliche Arbeit zur Erlangung des Grades

Master of Science (M.Sc.)

an der Fakultät für Maschinenwesen der Technischen Universität München

Betreut von	Univ.-Prof. Dr.-Ing. Markus Lienkamp Thomas Zehelein, M.Sc. Lehrstuhl für Fahrzeugtechnik
Eingereicht von	Thomas Hemmert-Pottmann, B.Sc. Jamnitzerstraße 8 81543 München
Eingereicht am	22. November 2018 in Garching

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Aufgabenstellung

Dämpferdefektdiagnose mittels der Deep Learning Architektur Convolutional Neural Networks

Am Lehrstuhl für Fahrzeugtechnik wird an einem Verfahren zur automatisierten Diagnose des aktuellen Fahrwerkzustands gearbeitet. Hierbei sollen gängige Sensorsignale wie Beschleunigungen, Gierrate, Lenkwinkel und Raddrehzahlen verwendet werden, um den Zustand des Fahrwerks und eventuell vorhandene Komponentendefekte bzw. Eigenschaftsveränderungen zu identifizieren.

In dieser Arbeit sollen Änderungen der Dämpfercharakteristik mittels der Deep Learning (DL)-Architektur Convolutional Neural Network (CNN) identifiziert werden. Hierfür sollen zunächst bestehende CNN-Diagnosekonzepte aus der Literatur recherchiert werden. Anschließend sollen mehrere Diagnosekonzepte umgesetzt und die Performance evaluiert werden. Hierfür sollen unter anderem die Aspekte der Anzahl der Trainingsdaten sowie die Notwendigkeit der Vorverarbeitung von Trainingsdaten berücksichtigt werden. Durch eine iterative Evaluation der Performance soll eine Weiterentwicklung der gewählten Diagnosekonzepte stattfinden. Zum Training der Konzepte stehen bereits Messdaten zur Verfügung und können darüberhinausgehend eingefahren werden.

Folgende Punkte sind durch Herrn Thomas Hemmert-Pottmann zu bearbeiten:

- Recherche und Einarbeitung in Diagnoseverfahren des Deep Learnings und der Netzwerkarchitektur von Convolutional Neural Network (CNN)
- Auswahl und Umsetzung geeigneter Diagnosekonzepte in einer geeigneten Entwicklungsumgebung
- Evaluation der Performance der Diagnosekonzepte
- Weiterentwicklung der gewählten Konzepte

Die Ausarbeitung soll die einzelnen Arbeitsschritte in übersichtlicher Form dokumentieren. Der Kandidat verpflichtet sich, die Arbeit selbständig durchzuführen und die von ihm verwendeten wissenschaftlichen Hilfsmittel anzugeben.

Die eingereichte Arbeit verbleibt als Prüfungsunterlage im Eigentum des Lehrstuhls und darf Dritten nur unter Zustimmung des Lehrstuhlinhabers zugänglich gemacht werden.

Ausgabe: 15. Juni 2018

Abgabe: 15. Dezember 2018

Univ.-Prof. Dr.-Ing. Markus Lienkamp

Thomas Zehelein, M.Sc.

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Geheimhaltungsverpflichtung

Herr/Frau: **Hemmert-Pottmann, Thomas**

Gegenstand der Geheimhaltungsverpflichtung sind alle mündlichen, schriftlichen und digitalen Informationen und Materialien die der Unterzeichner vom Lehrstuhl oder von Dritten im Rahmen seiner Tätigkeit am Lehrstuhl erhält. Dazu zählen vor allem Daten, Simulationswerkzeuge und Programmcode sowie Informationen zu Projekten, Prototypen und Produkten.

Der Unterzeichner verpflichtet sich, alle derartigen Informationen und Unterlagen, die ihm während seiner Tätigkeit am Lehrstuhl für Fahrzeugtechnik zugänglich werden, strikt vertraulich zu behandeln.

Er verpflichtet sich insbesondere:

- derartige Informationen betriebsintern zum Zwecke der Diskussion nur dann zu verwenden, wenn ein ihm erteilter Auftrag dies erfordert,
- keine derartigen Informationen ohne die vorherige schriftliche Zustimmung des Betreuers an Dritte weiterzuleiten,
- ohne Zustimmung eines Mitarbeiters keine Fotografien, Zeichnungen oder sonstige Darstellungen von Prototypen oder technischen Unterlagen hierzu anzufertigen,
- auf Anforderung des Lehrstuhls für Fahrzeugtechnik oder unaufgefordert spätestens bei seinem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik alle Dokumente und Datenträger, die derartige Informationen enthalten, an den Lehrstuhl für Fahrzeugtechnik zurückzugeben.

Eine besondere Sorgfalt gilt im Umgang mit digitalen Daten:

- Für den Dateiaustausch dürfen keine Dienste verwendet werden, bei denen die Daten über einen Server im Ausland geleitet oder gespeichert werden (Es dürfen nur Dienste des LRZ genutzt werden (Lehrstuhllaufwerke, Sync&Share, GigaMove).
- Vertrauliche Informationen dürfen nur in verschlüsselter Form per E-Mail versendet werden.
- Nachrichten des geschäftlichen E-Mail Kontos, die vertrauliche Informationen enthalten, dürfen nicht an einen externen E-Mail Anbieter weitergeleitet werden.
- Die Kommunikation sollte nach Möglichkeit über die (my)TUM-Mailadresse erfolgen.

Die Verpflichtung zur Geheimhaltung endet nicht mit dem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik, sondern bleibt 5 Jahre nach dem Zeitpunkt des Ausscheidens in vollem Umfang bestehen. Die eingereichte schriftliche Ausarbeitung darf der Unterzeichner nach Bekanntgabe der Note frei veröffentlichen.

Der Unterzeichner willigt ein, dass die Inhalte seiner Studienarbeit in darauf aufbauenden Studienarbeiten und Dissertationen mit der nötigen Kennzeichnung verwendet werden dürfen.

Datum: 15. Juni 2018

Unterschrift: _____

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Garching, den 22. November 2018

_____

Thomas Hemmert-Pottmann, B. Sc.

Abstract

Autonomous Driving evolves from future vision to reality. One major challenge is transitioning the driver's responsibility for condition monitoring to an automated system which keeps track of the system's health. Chassis components have a huge impact on the vehicle's stability. Therefore, monitoring and diagnosing any defects is a prerequisite for automation levels 4 to 5. We observe increased data availability in modern vehicles as well as advances in the field of machine learning. Many data-driven fault detection and isolation (FDI) systems use machine learning algorithms with hand-crafted features. Instead, Convolutional Neural Networks (CNN) can replace the cumbersome and error-prone process of feature engineering. This thesis investigates the general applicability and influencing factors of CNN for the automated diagnosis of chassis components, i.e. dampers. It is shown that CNN can replace feature engineering with their feature learning capabilities. Additionally, the proposed approach shows improved fault classification results.

Zusammenfassung

Eine große Herausforderung beim autonomen Fahren ist die Übertragung der Verantwortung für die Zustandsüberwachung vom Fahrer auf ein Computersystem. Weil Fahrwerkskomponenten maßgeblich die Fahrzeugstabilität beeinflussen, ist deren Überwachung und die Diagnose etwaiger Defekte wichtig und Voraussetzung für die höchsten Automationsstufen. Die hohe Datenverfügbarkeit in modernen Autos sowie aktuelle Trends im Bereich des maschinellen Lernens ermöglichen eine datenbasierte Fehlererkennung von Fahrwerkskomponenten. Anstatt in fehleranfälliger, manueller Arbeit aus den Daten Merkmale für die Fehlererkennung zu extrahieren, kann diese Aufgabe durch Convolutional Neural Networks (CNN) übernommen werden. Diese Arbeit untersucht daher den Einsatz sowie Einflussfaktoren von CNN zur automatisierten Diagnose von Fahrwerkskomponenten am Beispiel von Schwingungsdämpfern. Es wird gezeigt, dass CNN für die Aufgabe geeignet sind und bessere Ergebnisse erreichen als ein System mit manuell extrahierten Merkmalen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ABS	Antiblockiersystem
ANFIS	Adaptive Neuro-Fuzzy Inference System
BN	Batch Normalization
BRN	Batch Renormalization
C	Convolutional Layer
CAN	Controller Area Network
CNN	Convolutional Neural Network
cReLU	concatenated Rectified Linear Unit
CT	Computertomographie
DBN	Deep Believe Network
DFT	Diskrete Fourier Transformation
DL	Deep Learning
ECU	Electronic Control Unit
ELU	Exponential Linear Unit
EMD	Empirical Mode Decomposition
ES	Envelope Spectrum
ESP	Elektronisches Stabilitäts-Programm
FC	Fully Connected
FDI	Fault Detection and Isolation
FFT	Fast Fourier Transformation
FN	Falsch Negative
FP	Falsch Positive
FQ	Fehlerquote
GAF	Gramian Angular Field
GAP	Global Average Pooling
GPS	Global Positioning System
GPU	Graphics Processing Unit
HHT	Hilbert-Huang Transformation
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IMF	Intrinsic Mode Function
IT	Informationstechnik
KKR	Korrektklassifizierungsrate
k-NN	k-nearest Neighbor

KI	Künstliche Intelligenz
KNN	Künstliche Neuronale Netze
ML	Machine Learning
MLP	Multi Layer Perceptron
MP	Max-Pooling Layer
MP’	Mean-Pooling Layer
MPLS	Multi-way Partial Least Squares
NiN	Network-in-Network
PCA	Principal Component Analysis
PNN	Probabilistic Neural Network
RBFN	Radial Basis Function Network
ReLU	Rectified Linear Unit
RFE	Recursive Feature Elimination
RN	Richtig Negative
RNN	Recurrent Neural Networks
RP	Richtig Positive
SE	Squeeze-and-Excitation
sELU	scaled Exponential Linear Unit
SGD	Stochastic Gradient Descent
SM	Softmax-Output
SNR	Signal to Noise Ratio
SS	Subsampling Layer
STFT	Short-Time Fourier Transformation
SVM	Support Vector Machine
WP	Wavelet Packet
WPI	Wavelet Packet Energy Image
WPT	Wavelet Packet Transformation
WT	Wavelet Transformation

Formelzeichen

Formelzeichen	Einheit	Beschreibung
a_{X}	$\frac{\mathrm{m}}{\mathrm{s}^2}$	Fahrzeug-Längsbeschleunigung
a_{Y}	$\frac{\mathrm{m}}{\mathrm{s}^2}$	Fahrzeug-Querbeschleunigung
b	-	Größe eines Mini-Batches
$\boldsymbol{b}$	-	Bias-Terme eines Layers
c	-	Anzahl der Datenkanäle
C	-	Kardinalität eines Inception(-artigen) Moduls (Anzahl paralleler Zweige)
d_{in}	-	Tiefe des Eingangsvolumens von einem Layer
e	-	Spatial Extent eines Filters
E	-	Kostenfunktion
f_{d}	-	Tiefe eines (Convolutional) Filterkernels
f_{h}	-	Höhe eines (Convolutional) Filterkernels
f_{s}	Hz	Abtastrate
f_{w}	-	Breite eines (Convolutional) Filterkernels
$\boldsymbol{F}$	-	Convolutional Filterkernel
h	-	Höhe des Eingangsvolumens eines Layers
$\boldsymbol{I}$	-	Eingangsvolumen eines Layers
k	-	Anzahl Filterkernel eines Convolutional Layers
n_{FL}	$\frac{1}{\mathrm{min}}$	Raddrehzahl vorne links
n_{FR}	$\frac{1}{\mathrm{min}}$	Raddrehzahl vorne rechts
n_{RL}	$\frac{1}{\mathrm{min}}$	Raddrehzahl hinten links
n_{RR}	$\frac{1}{\mathrm{min}}$	Raddrehzahl hinten rechts
p	-	Padding-Größe oder -Modus
p_r	-	Dropout-Rate

r	-	Reduktionsrate im Squeeze-and-Excitation (SE)-Block
s	-	Stride eines Convolutional Kernels
$\boldsymbol{t}$	-	Wahre (true) Label
$\boldsymbol{v}$	-	Gruppierungsvektor bei der gruppierten Depthwise Convolution
$\boldsymbol{w}$	-	Gewichte eines Neurons im neuronalen Netz
$\boldsymbol{W}$	-	Gewichte eines Layers
$\boldsymbol{x}$	-	Eingang eines neuronalen Netzes
$\boldsymbol{y}$	-	Ausgabe eines neuronalen Netzes
α	-	Skalierung bei der Leaky ReLU-Funktion
δ	$^\circ$	Lenkwinkel
ϵ	-	Lernrate
ϑ	-	Schwellwert einer Aktivierungsfunktion
λ	-	L2-Regularisierungsstärke (Weight Decay)
μ	-	Mittelwert
ρ	-	Verfall-Faktor beim RMSProp-Optimierer
σ	-	Aktivierungsfunktion
ς	-	Standardabweichung
$\dot{\psi}$	$\frac{^\circ}{\mathrm{s}}$	Gierrate

1 Einleitung und Motivation

Das autonome Fahren ist eine zunehmend realistischer werdende Zukunftsvision [1–3]. Den Schlüssel zu diesem Technologieschritt stellen moderne Fahrerassistenzsysteme sowie die Informationstechnik (IT) dar [2, S. 61]. Durch die Ersetzung des Fahrers bei der Fahraufgabe geht allerdings die Verantwortung für die Zustands- und Systemüberwachung auf ein Computersystem über [1, S. 3]. Dabei spielt die zuverlässige Diagnose inbesondere von Fahrwerkskomponenten eine entscheidende Rolle, denn das Fahrwerk beeinflusst maßgeblich die Fahrzeugstabilität beim Fahren [3, S. 1]. Eine defekte Fahrwerkskomponente, bspw. ein Dämpfer, kann sich vor allem in kritischen Fahrsituationen negativ auf die Fahrzeugstabilität auswirken [4]. Deshalb müssen Fehler frühzeitig diagnostiziert werden.

In modernen, hochvernetzten Fahrzeugen stehen jederzeit große Datenmengen zur Verfügung [2, S. 21, 3, S. 658f]. Diese Daten werden häufig für Fahrerassistenzsysteme genutzt, können jedoch ebenfalls zur Fehlerdiagnose herangezogen werden [5–11]. Der konventionelle Ansatz zur datenbasierten Fehlerdiagnose besteht aus den Schritten Datenakquisition, Merkmalsextraktion und Fehler-Klassifikation [12]. Die Merkmalsextraktion trägt maßgeblich zur Leistungsfähigkeit eines solchen Diagnosesystems bei [13]. Dafür werden wiederum neben Signalverarbeitungs-Techniken Expertenwissen und menschliche Arbeit benötigt [14]. Dieser Arbeitsanteil ist mühsam und fehleranfällig, weshalb es nahe liegt, den Schritt der Merkmalsextraktion zu automatisieren. Eine Möglichkeit dafür stellen Methoden der „künstlichen Intelligenz (KI)“ dar.

Das allgemeine Interesse am maschinellen Lernen („Machine Learning (ML)“), einem Teilgebiet der KI, wächst seit vielen Jahren stetig, seit den letzten drei jedoch besonders stark (Abbildung 1.1). In Verbindung mit der steigenden Verfügbarkeit von Daten führt dies dazu, dass

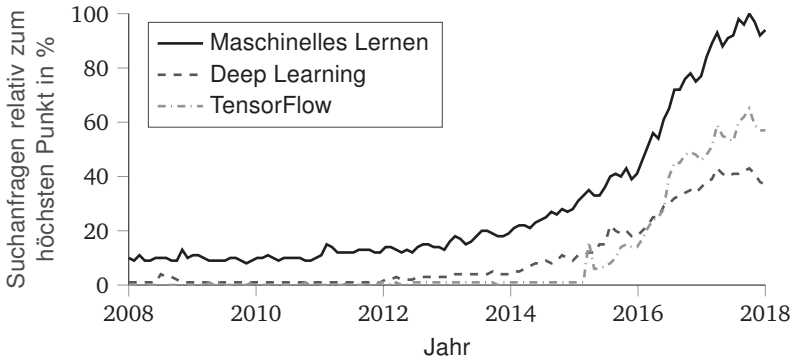


Abbildung 1.1: Beliebtheit der Themen „Maschinelles Lernen“, „Deep Learning“ und „TensorFlow“ in den letzten zehn Jahren laut Google Trends (jeweils zum September)

im Bereich von ML Software-Bibliotheken wie TensorFlow [15], Torch [16], Theano [17] etc. ständig wachsen. Die Entwicklung von Deep Learning (DL)-Algorithmen, die einen Teilbereich

des ML abbilden, wird damit zunehmend einfacher. DL-Ansätze können die für die Diagnose benötigten Merkmale eigenständig erlernen, was einer der größten Vorteile dieser Verfahren ist [18]. Für diverse Anwendungen, u.a. bedingt durch die Erfolge in der Bildverarbeitung [19, 20], hat sich die DL-Architektur Convolutional Neural Network (CNN) als Stand der Technik herauskristallisiert [21]. Diese ist als eine spezielle Art von künstlichen neuronalen Netzen (KNN) anzusehen. Im Bereich der intelligenten Fehlerdiagnose von mechanischen Komponenten weisen CNN oftmals bessere Leistungen auf als Methoden, die auf eine manuelle Merkmalsextraktion setzen [22].

1.1 Problemstellung und Zielsetzung

Das Fahrwerk spielt hinsichtlich Fahrsicherheit, Fahrkomfort, Fahrdynamik und Agilität die wesentlichste Rolle im Auto [3, S. 1]. Komponentendefekte im Fahrwerk wie bspw. defekte Federn oder Dämpfer können sich daher stark auf das Fahrverhalten auswirken. Semi-aktive Dämpfer sorgen bspw. für eine bessere Querkraftabstützung sowie eine Bremswegverkürzung [23, S. 3.1 – 161]. Diese Eigenschaften werden durch einen Defekt reduziert oder gehen gänzlich verloren. Die frühzeitige Erkennung ist deswegen wichtig. Sie gestaltet sich insbesondere bei Dämpfern jedoch als schwierig. Die Eigenschaftsänderungen im Übergang vom intakten zum defekten Dämpfer verlaufen meist schleichend und werden selbst von erfahrenen Fahrern nur schwer erkannt [5]. Zusätzlich wird dies bspw. durch Carsharing-Konzepte erschwert, bei denen der Fahrer in seinen gelegentlichen Fahrten häufig das Fahrzeug wechselt [6]. Dadurch ist er nicht in der Lage ein Defekt-Verhalten als solches zu erkennen, weil die Fahrzeuge bereits im Intakt-Zustand unterschiedliches Fahrverhalten zeigen.

Obwohl Dämpferdefekte zusammen mit gebrochenen Achsfedern einen großen Anteil der Fahrzeugmängel ausmachen [24], existieren nur wenige Systeme zur Überwachung der Dämpfer. Nach Merk [5, S. 3] bietet einzig das Start-Up Carfit ein Fehlerdetektions- und -isolationssystem (Fault Detection and Isolation (FDI)-System) für Dämpfer an, wobei zusätzlich Beschleunigungssensoren an den Dämpfern verbaut werden müssen.
Für ein FDI-System ist es zeitgemäß, auf zusätzliche Sensorik zu verzichten und stattdessen neuartige, datenbasierte Algorithmen einzusetzen. Die kostengetriebene Entwicklung des Massenprodukts „Automobil“ führt ohnehin dazu, dass Funktionen zunehmend in Software umgesetzt werden und die IT im Auto eine bedeutende Rolle einnimmt [2]. Mit einem zuverlässigen Diagnosesystem für Fahrwerkskomponenten kann auf teure Wartungen verzichtet und die Fahrzeugsicherheit erhöht werden.

In dieser Arbeit wird der prototypische Entwurf eines DL-basierten FDI-Systems für Fahrwerkskomponenten untersucht. Das System wird am Beispiel von Dämpferdefekten entwickelt und soll in der Lage sein, Änderungen der Dämpfercharakteristik, d. h. Defekte, zu identifizieren. Zur Datenerfassung bzw. -gewinnung wird ausschließlich auf vorhandene Antiblockiersystem (ABS)- und Elektronisches Stabilitäts-Programm (ESP)-Sensorik zugegriffen. Zur Diagnose sollen CNN eingesetzt werden. Hierfür werden bestehende CNN-Diagnosekonzepte zur Klassifikation mechanischer Defekte aus der Literatur recherchiert. Anschließend sollen mehrere Diagnosekonzepte umgesetzt und die Performance evaluiert werden. Hierfür werden unter anderem die Aspekte der Anzahl der Trainingsdaten sowie die Notwendigkeit der Vorverarbeitung von Trainingsdaten berücksichtigt. Das System mit der besten Leistungsfähigkeit soll anschließend hinsichtlich ausgewählter Aspekte optimiert werden.

1.2 Abgrenzung der Arbeit

Im Bereich der Erkennung von Maschinenfehlern (Fehlererkennung) werden die Ausreißererkennung („Anomaly Detection“) sowie Fehlerdetektion und -isolation (FDI) unterschieden (Abbildung 1.2). Der größte Unterschied zwischen den beiden Verfahren liegt darin, dass bei der Anomaly Detection vorab keine Beispiele für alle möglichen Zustände vorliegen müssen, sondern lediglich Daten von normalen bzw. tolerierten Betriebsbedingungen. Dies ist gleichzeitig ein großer Vorteil der Anomaly Detection, da es oftmals schwierig ist, Daten unter fehlerhaften Bedingungen bzw. im „Defekt“-Zustand aufzunehmen [25]. In dieser Arbeit liegen Daten im Defekt- und Intakt-Zustand vor. Um diese vollständig auszunutzen, wird nicht auf die Anomaly Detection ausgewichen. Mit der Erfassung von Ausreißern (Anomalien) in der Dämpfercharakteristik befasst sich eine parallel laufende Masterarbeit [26].

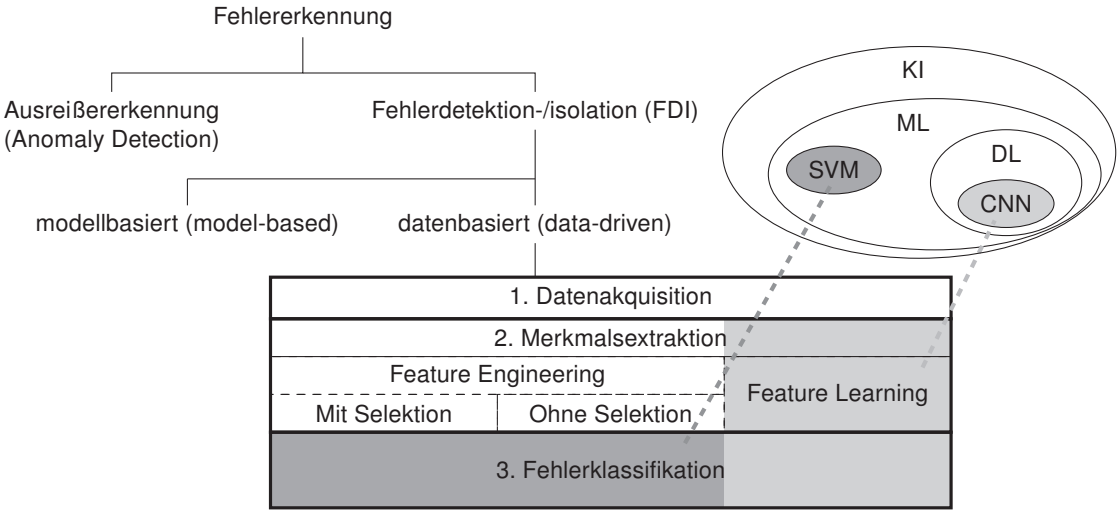


Abbildung 1.2: Übersicht zur Einordnung des zu entwickelnden FDI-Systems. Exemplarisch sind im Venn-Diagramm nach [27, S. 9] Support Vector Machine (SVM) und CNN als Stellvertreter der jeweiligen Gruppe (ML bzw. DL) eingezeichnet. Ein CNN vereint Merkmalsextraktion und Fehlerklassifikation in einer einzigen Architektur.

Systeme zur Fehlererkennung können in modell- („model-based“) und datenbasierte („data-driven“) Ansätze unterteilt werden (Abbildung 1.2) [28–31]. Zur FDI von Fahrwerkskomponenten, wie sie in dieser Arbeit angestrebt ist, existieren bereits in beiden Bereichen Diagnosesysteme. Einige davon setzen auf ein Signal- bzw. physikalisches Modell [32–35] (modellbasiert), andere auf ML-Algorithmen [5–11] (datenbasiert).
Anstatt einem Computer eine umfangreiche Liste mit Regeln und Zusammenhängen zur Problemlösung zu geben (modellbasiert), wird ihm beim ML ein Modell bzw. Konzept gegeben. Dieses kann auf Basis von Beispielen und wenigen Anweisungen angepasst werden, wenn ein Fehler gemacht wird [36, S. 4]. Der Algorithmus soll durch die Erfahrung „lernen“ und das Problem durch eine Hierarchie von Konzepten (Transformationen) verstehen, wobei jedes Konzept eine Beziehung zu einem einfacheren Konzept besitzt. Komplizierte Zusammenhänge können in einfache Konzepte zerlegt werden. Je komplizierter ein Zusammenhang ist, desto *tiefer* wird die Struktur. [27, S. 1f] Dieser Ansatz wird daher Deep Learning (DL) genannt. Eine Übersicht zu relevanten Arbeiten einschließlich einer umfangreichen Literatursammlung zum Thema DL wird von Schmidhuber [37] gegeben.

Durch die hohe Datenverfügbarkeit und aktuelle Trends auf dem Gebiet des Deep Learnings beschränkt sich diese Arbeit ausschließlich auf datenbasierte Ansätze zur Entwicklung des

Diagnosesystems. Diese Methoden zur Fehlererkennung technischer Systeme lassen sich prinzipiell nach manueller Merkmalskonstruktion bzw. -extraktion durch Expertenwissen („Feature Engineering“) mit oder ohne Reduktion der Merkmale („Features“) sowie dem automatischen Erlernen von Merkmalen („Feature Learning“) unterteilen [18, S. 332–334] (Abbildung 1.2). Die ersten beiden Ansätze werden üblicherweise mit herkömmlichen ML-Algorithmen, d. h. „flachen“ Architekturen, verfolgt, während für das Erlernen von Merkmalen DL-Architekturen eingesetzt werden.

Chen et al. [38] zeigen, dass bei der Entwicklung von Diagnosesystemen mit der DL-Architektur CNN auf die Verwendung des automatischen Feature-Lernens verzichtet werden kann. Die Schlüsseleigenschaft zur Leistungssteigerung von CNN-basierten Methoden ist jedoch die Nutzung optimaler Merkmale, die das CNN selbst erlernt hat. Diese Eigenschaft ermöglicht die Maximierung der Klassifikationsgenauigkeit [22]. In einer Getriebe-Anwendung [12] zeigt ein Vergleich von CNN-Architekturen, die manuell extrahierte Merkmale als Eingangsdaten verwenden, keine große Verbesserung gegenüber ML-Anwendungen, darunter konventionelle KNN (Multi Layer Perceptron (MLP)) und SVM, die mit denselben Merkmalen trainiert wurden. Erst die Nutzung der Netzeigenschaft, Merkmale selbst zu erlernen, brachte eine deutliche Verbesserung der Klassifikationsgenauigkeit [12, S. 6–9].

In einer vorangegangenen Arbeit am Lehrstuhl für Fahrzeugtechnik wurde die Dämpferdefektdiagnose bereits mit einem ML-Ansatz untersucht [5, 6]. Mithilfe einer SVM wurde gezeigt, dass ausschließlich auf Basis von ABS- und ESP-Sensorik eine Klassifikation unterschiedlicher Dämpferdefekte möglich ist. Die dafür verwendeten Merkmale müssen manuell extrahiert werden und lassen sich mit unterschiedlichen Algorithmen reduzieren. Die vorliegende Arbeit knüpft dort an. Es wird untersucht, inwiefern der Anteil menschlicher Arbeit in Form von Merkmalsextraktion und -selektion durch die Verwendung einer CNN-Architektur reduziert werden kann. Eine weitere Masterarbeit beschäftigt sich mit derselben Fragestellung, setzt jedoch auf einen (DL-)Ansatz mit Auto-Encodern [39].

1.3 Aufbau der Arbeit

Im folgenden Kapitel werden Grundlagen zu KNN und CNN vermittelt, die für das Verständnis dieser Arbeit notwendig sind. Außerdem wird auf Diagnoseanwendungen im automotiven Bereich eingegangen, wobei der Schwerpunkt auf ML-basierten Ansätzen liegt. Die Ergebnisse der Literaturrecherche zu DL-Anwendungen in der Automobilbranche sowie CNN-basierten Diagnosesystemen – insbesondere zur Diagnose mechanischer Komponenten – werden anschließend vorgestellt und zusammengefasst. Kapitel 3 beinhaltet Ausführungen zum Vorgehen beim Entwurf des FDI-Systems. Es wird die Datengewinnung rekapituliert und sowohl auf die Hauptuntersuchungsaspekte als auch Details zu den implementierten Architekturen eingegangen. Die Ergebnisse der untersuchten Aspekte werden im anschließenden Kapitel vorgestellt und ausgewählte Architekturen hinsichtlich weiterer Einflussparameter optimiert. Auf Basis neu eingefahrener Daten wird das entworfene FDI-System validiert und die Robustheit geprüft. In den letzten Kapiteln werden die gewonnenen Ergebnisse diskutiert, die Arbeit zusammengefasst und ein Ausblick auf mögliche Folgetätigkeiten gegeben.

2 Stand der Wissenschaft

In diesem Kapitel werden benötigte Grundlagen vermittelt und der Stand der Wissenschaft in Bezug auf automotive FDI-Systeme evaluiert. Die Konzepte und Begriffe von neuronalen Netzen werden als erstes in ihren Grundzügen eingeführt. Dies dient als Basis für die Vorstellung des KNN-basierten DL-Ansatzes CNN im darauffolgenden Unterkapitel 2.2. Darauf folgt eine Übersicht zu verschiedenen Diagnoseanwendungen, die Ansätze des maschinellen Lernens (Unterkapitel 2.3) nutzen. Es schließt sich eine Übersicht von DL-Ansätzen im Automobilbereich an (Unterkapitel 2.4). Im vorletzten Unterkapitel wird der Stand der Wissenschaft für die Aufgabe der CNN-basierten Diagnose bei industriellen Maschinenbau-Anwendungen erörtert und zusammengefasst (Unterkapitel 2.5). Auf Basis der Erkenntnisse wird die zu behandelnde Aufgabenstellung hergeleitet (Unterkapitel 2.6).

2.1 Grundlagen Neuronaler Netze

Das Konzept künstlicher Neuronaler Netze wird üblicherweise auf die Analogie zum Hirn eines Menschen oder Tieres zurückgeführt. Die entstandenen Modelle im Bereich des maschinellen Lernens sind jedoch nicht als realistisches Modell biologischer Funktionen ausgelegt, sondern können gelegentlich genutzt werden, um Hirnfunktionen nachzuvollziehen. Das Verstehen des Gehirns auf algorithmischer Ebene fällt in den Bereich der Computer-Neurowissenschaft, während KNN als Grundlage des Deep Learnings zur computergestützten Lösung von Aufgaben dienen, die eine Form von „Intelligenz“ benötigen. [27, S. 13–16]

2.1.1 Neuronen und Aktivierungsfunktionen

Die Kernidee eines KNN ist, dass eine Vielzahl einfacher Recheneinheiten zusammen ein intelligentes Verhalten nachahmen kann [27, S. 16]. Eine einzelne dieser simplen Recheneinheiten wird als Neuron, in einfacher Form als lineares Perzeptron [40], bezeichnet [36, S. 8f]. Jedes lineare Perzeptron kann durch ein einzelnes Neuron ausgedrückt werden, andersherum gilt der Zusammenhang nicht [36, S. 9]. Ein einfaches Neuron besitzt mehrere Eingänge x_1 bis x_k, die mit Gewichten w_1 bis w_k multipliziert werden, und einen Bias b, der als konstanter Korrekturwert gilt (Abbildung 2.1).

Angelehnt an Nervenzellen mit synaptischen Verbindungen, kann die Aktivierung

$$\varphi(\boldsymbol{x}) = b + \sum_{i=1}^{k} x_i w_i = \boldsymbol{x}^T \boldsymbol{w} + b \tag{2.1}$$

berechnet werden [42, S. 120–126, 27, S. 166f]. Die Aktivierung stellt die Anregung des Perzeptrons oder Neurons dar. Nur wenn sie einen Schwellwert ϑ überschreitet, wird das Neuron aktiv.

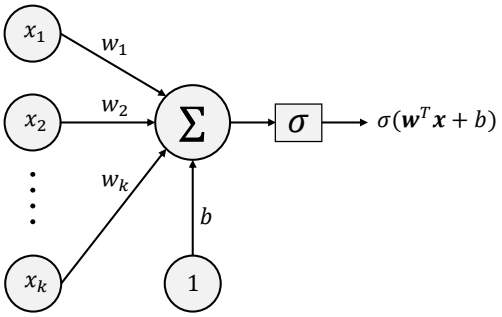


Abbildung 2.1: Darstellung eines künstlichen Neurons in Anlehnung an [41, S. 63]

Ursprünglich wurde die Schwellwertfunktion

$$\sigma_{\text{schwell}}(\boldsymbol{x}) = \begin{cases} 1, \text{ falls } \varphi(\boldsymbol{x}) \geq \vartheta \\ 0, \text{ falls } \varphi(\boldsymbol{x}) < \vartheta \end{cases} \tag{2.2}$$

binär formuliert, wobei $y = \sigma(\boldsymbol{x}) = f(\boldsymbol{x}, \boldsymbol{w}, b, \vartheta)$ die Ausgabe des Perzeptrons darstellt und von der Eingabe $\boldsymbol{x}$, den Gewichten $\boldsymbol{w}$, dem Bias b sowie dem Schwellwert ϑ abhängt [42, S. 126]. Da für das Training eines aus Neuronen aufgebauten Netzes differenzierbare Funktionen notwendig sind [43, S. 3-9], wurden unterschiedliche Schwellwertfunktionen, Aktivierungsfunktionen genannt, eingeführt.

Sigmoid Als biologisch inspirierte Aktivierungsfunktion wurde die (logistische [27, S. 65]) Sigmoid-Funktion $\sigma_{\text{sigmoid}}(\varphi) : \mathbb{R} \mapsto [0, 1]$

$$\sigma_{\text{sigmoid}}(\varphi) = \frac{1}{1 + \mathrm{e}^{-\varphi}} \tag{2.3}$$

vielfach in KNN eingesetzt [41, S. 71]. Das Hauptproblem der Funktion ist die Sättigung für große Werte der Aktivierung φ, d. h. die Funktion flacht ab und wird unempfindlich gegenüber geringen Änderungen [41, S. 71f, 27, S. 66]. Das führt zu verschwindend kleinen Gradienten bei Aktivierungen fern des Ursprungs, was wiederum nachteilig beim Training eines Netzes sein kann (Problem verschwindender Gradienten bzw. „vanishing gradients problem“) [41, S. 72]. Als anders skalierte Version des Sigmoid wird der hyperbolische Tangens eingesetzt. Die Funktion nähert um den Ursprung die Identitätsfunktion besser an ($\sigma_{\tanh}(0) = 0$ während $\sigma_{\text{sigmoid}}(0) = \frac{1}{2}$) und wird daher gegenüber der Sigmoid-Funktion bevorzugt [27, S. 189f].

Rectified Linear Unit (ReLU) Während die beiden vorangegangenen Aktivierungsfunktionen meist in KNN mit wenigen Layern, d. h. flachen neuronalen Netzen, eingesetzt werden, kommen in tieferen Architekturen aufgrund des Problems der verschwindenden Gradienten meist Funktionen der Rectified Linear Unit (ReLU)-Familie zum Einsatz. Die ReLU-Funktion stellt eine rechentechnisch effiziente, nicht-lineare Aktivierungsfunktion der Form

$$\sigma_{\text{relu}}(\varphi) = \max(0, \varphi) \tag{2.4}$$

dar und erreicht in $\mathbb{R}^+$ keine Sättigung. Das Abbildungs-Intervall ist $[0, \infty)$. Dadurch werden verschwindende Gradienten umgangen, was die Funktion attraktiv für DL-Architekturen macht [36, S. 15, 41, S. 74]. Andererseits können durch eine entsprechende Gewichts-Konfiguration von

$\boldsymbol{w}$ „tote“ Neuronen entstehen, die unabhängig vom Eingang $\boldsymbol{x}$ immer die Ausgabe $\sigma_{\text{relu}} = 0$ erzeugen. Die Neuronen können zur effizienteren Berechnung zwar entfernt werden, es kann allerdings zu Einbußen bei der Gesamt-Genauigkeit des Netzes kommen [41, S. 74f]. Um das Aussterben von Neuronen zu verhindern, existieren spezielle Erweiterungen der ReLU-Funktion wie bspw. Leaky ReLU oder Parameterized ReLU, auf die nicht weiter eingegangen wird.

2.1.2 Feedforward Neural Networks

Obwohl ein einzelnes Neuron bereits einfache Zusammenhänge wie ein logisches `AND` oder `OR` abbilden kann [43, S. 3-5f], lassen sich komplexere Lernaufgaben nicht damit lösen. Dafür werden Neuronen in Schichten („layern“) angeordnet und miteinander vollständig verknüpft („Fully Connected (FC)“), wodurch Künstliche Neuronale Netze (KNN) entstehen. KNN werden Feedforward Neural Networks bzw. Multi Layer Perceptron (MLP) genannt, wenn Informationen vom Eingang $\boldsymbol{x}$ zum Ausgang $\boldsymbol{y}$ fließen und keine Rückkopplungen („feedback connections“) des Ausgangs zurück ins Modell existieren. Sind solche Feedback-Schleifen vorhanden, heißen diese Netze rekurrente Neuronale Netze („Recurrent Neural Networks (RNN)“). [27, S. 163]
Die Layer eines Netzes enthalten ein oder mehrere Neuronen. Die erste und letzte Schicht werden Input- bzw. Output-Layer genannt, während alle Schichten dazwischen als versteckte Schichten („hidden layer“) bezeichnet werden [41, S. 63f]. Die Dimension der Input- und Output-Layer hängen von der Lernaufgabe ab, während die Weite eines Modells, d. h. die Dimension(en) der hidden Layers, beliebig gewählt werden kann. Es ist weder notwendig noch empfohlen, dass jede Schicht die gleiche Anzahl Neuronen besitzt, vielmehr kann das Netz durch einen „schmalen“ Layer („Bottleneck Layer“) zu einer komprimierten Darstellung der Daten gezwungen werden [36, S. 11]. Typischerweise sind die verdeckten Schichten vektorwertig. Ein Layer kann als einfache Vekor-Vektor-Funktion betrachtet werden oder als Komposition vieler parallel arbeitender Einheiten, die jeweils eine Vektor-Skalar-Funktion darstellen. [27, S. 164]

Abbildung 2.2 zeigt ein einfaches neuronales Netz mit vier Layern, das aus künstlichen Neuronen aufgebaut ist. Die Aktivierungsfunktion σ wurde in Abbildung 2.2 nicht genauer spezifiziert. Sie

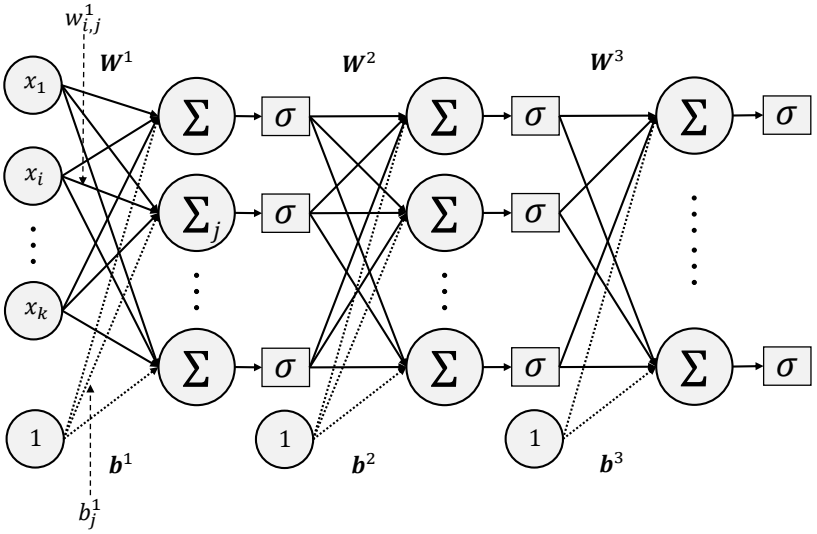


Abbildung 2.2: Darstellung eines neuronalen Netzes in Anlehnung an [41, S. 63]

kann für jedes Neuron unterschiedlich gewählt werden, in der Praxis wird jedoch üblicherweise eine einheitliche Funktion pro Layer verwendet [41, S. 64]. Das Gewicht $w_{i,j}^{(k)}$ repräsentiert die gewichtete Kante zwischen dem i-ten Neuron im k-ten Layer mit dem j-ten Neuron im $k+1$-ten

Layer, exemplarisch dargestellt im ersten, d. h. dem Input-Layer. Zwischen zwei Schichten mit jeweils mehr als einem Neuron ergibt sich statt des Vektors $\boldsymbol{w}$ eine Matrix $\boldsymbol{W}$, die jeweils alle Gewichte des entsprechenden Layers (durchgezogene Pfeile) beinhaltet. Die Bias-Terme (gepunktete Pfeile) werden in einem Vektor $\boldsymbol{b}^{(\cdot)}$ Layer-weise zusammengefasst, wobei der Exponent $(\cdot)$ den Layer indiziert. Anhand des dargestellten Beispiels lässt sich leicht die Ketten-Struktur von Feedforward Netzen erkennen [27, S. 163f]. Ein Modell des Netzes kann durch

$$f(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}^3\sigma\left(\boldsymbol{W}^2\sigma\left(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1\right) + \boldsymbol{b}^2\right) + \boldsymbol{b}^3\right) \tag{2.5}$$

beschrieben werden [41, S. 63f].

Feedforward Netze besitzen die Eigenschaft, dass bereits mit einem hidden Layer und einer endlichen Anzahl Neuronen jede beliebige, kontinuierliche Funktion angenähert werden kann [41, S. 64, 44, S. 363]. Insbesondere in Klassifizierungsproblemen ist diese Eigenschaft wichtig. Mehrklassen-Probleme, bei denen die Klassen nicht linear separierbar sind, stellen eine andere Herausforderung dar. Hierfür muss eine Transformation gefunden werden, welche die Klassen unterscheidbar macht. Unter der Annahme, dass $\Phi_{\text{ideal}}(\boldsymbol{x})$ eine solche Transformation darstellt und mit dem Wissen, dass ein Feedforward Neural Network ein universelles Annäherungsverfahren darstellt, kann ein MLP entworfen werden, das in der Lage ist $\Phi_{\text{ideal}}(\boldsymbol{x})$ zu approximieren. Mit anderen Worten: Anstatt $\varphi_{\text{ideal}}(\boldsymbol{x})$ zu entwickeln, wird mit einem KNN ein Satz von Parametern (die Gewichte aller Kanten im Neuronen-Netzwerk) erlernt, die $\varphi_{\text{ideal}}(\boldsymbol{x})$ approximieren. Der größte Vorteil dieser Netze besteht darin, dass demnach die Transformation nicht entwickelt werden muss, sondern lediglich ein geeigneter Satz sog. Hyperparameter festgelegt wird. Dazu zählen die Anzahl der hidden Layer, die Anzahl der Neuronen je Layer und die Wahl der Aktivierungsfunktion. [41, S. 63–65, 27, S. 164f]

Softmax Output Als spezielle Form von Output-Layer existiert der Softmax Output-Layer. Er liefert als Ausgabe-Vektor die Wahrscheinlichkeitsverteilung zwischen sich gegenseitig ausschließenden Klassen, indem er bei der Berechnung des Outputs eines Neurons – im Gegensatz zu anderen Layer-Arten – die Ausgabe der anderen Neuronen im selben Layer berücksichtigt. Die Summe aller Ausgänge eines Softmax Layers wird dazu auf 1 normiert, sodass eine starke Vorhersage zu einem Vektor mit einem einzigen Eintrag nahe 1 und den restlichen nahe 0 führt. Ein Anwendungsbeispiel ist die Erkennung von handgeschriebenen Ziffern, bei dem die Label 0 bis 9 sich gegenseitig ausschließen und eine Vorhersage mit 100 % Gewissheit z. B. aufgrund verschiedener Handschriften unwahrscheinlich ist. [36, S. 15]

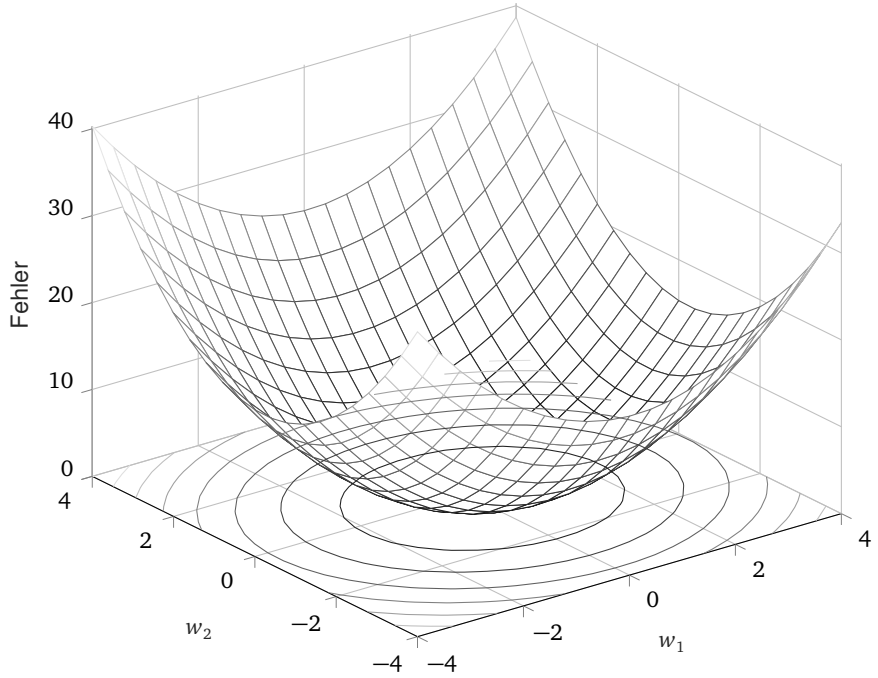
2.1.3 Training von Neuronalen Netzen

Beim MLP werden Gewichte „erlernt“, die in ihrer durch den Entwickler festgelegten Architektur für ein gegebenes Problem eine möglichst optimale Lösung annähern. Dieser Prozess wird üblicherweise Training genannt [36, S. 17, 27, S. 164]. Während des Trainingsprozesses wird dem Netz eine große Anzahl von Beispielen (Trainingsdaten) präsentiert. Die Gewichte werden iterativ angepasst, um die Abweichung zwischen Netzvorhersage und Trainingsbeispiel zu minimieren [36, S. 17].

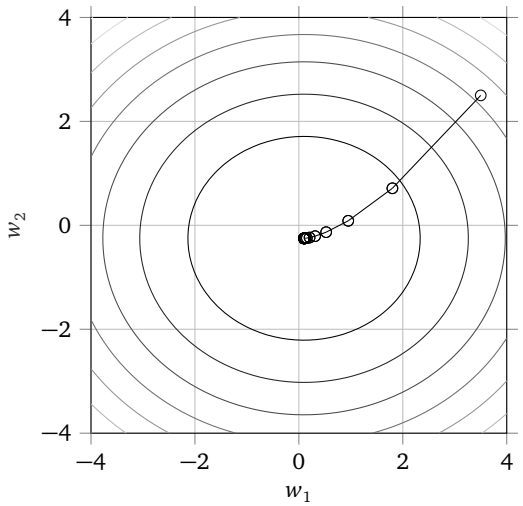
Als Gütemaß für dieses Optimierungsproblem kann beispielsweise der quadratische Fehler

$$E = \frac{1}{2}\sum(\boldsymbol{t} - \boldsymbol{y})^2 \tag{2.6}$$

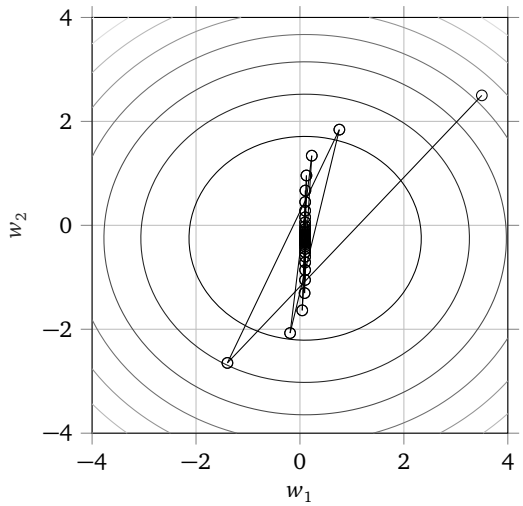
herangezogen werden. t stellt den wahren („true“) Wert der Trainingsbeispiele dar, während y die Ausgabe des MLP beschreibt. Dieses Gütemaß sorgt dafür, dass alle Fehler positiv sind und große Abweichungen gegenüber kleineren stärker ins Gewicht fallen [43, S. 3-6]. Für ein lineares Neuron mit zwei Eingängen kann ein quadratischer Fehlerraum über die beiden Gewichte aufgetragen werden (Abbildung 2.3(a)). In diesem Raum entsprechen Punkte in der



(a) 3-D Visualisierung mit Höhenlinien des quadratischen Fehlers für ein lineares Neuron in Anlehnung an [36]



(b) Konvergenz bei guter Lernrate: Es werden wenige Berechnungsschritte benötigt.



(c) Konvergenz bei (zu) großer Lernrate. Der Algorithmus springt mehrfach am Optimum vorbei und benötigt deshalb viele Berechnungsschritte.

Abbildung 2.3: Visualisierung des Gradient Descent-Algorithmus' in den Höhenlinien der Fehleroberfläche, Startpunkt oben rechts. Erstellung angelehnt an [45].

horizontalen Ebene unterschiedlichen Konfigurationen der Gewichte, während die Höhe den resultierenden Fehler repräsentiert. Es ergibt sich eine quadratische Schale. Die Höhenlinien sind in der w_1-w_2-Ebene dargestellt. Je näher sie aneinander liegen, desto steiler ist die Steigung.

Das größte Gefälle bzw. die größte Steigung wird in orthogonaler Richtung zu den Höhenlinien erreicht. Der zugehörige Richtungsvektor wird Gradient genannt. [36, S. 20] Um sich in Richtung des Minimums zu bewegen, wird der negative Gardient verwendet [27, S. 80].
Zu Anschauungszwecken kann das quadratische Fehlermaß leicht verzerrt werden; die Höhenlinien sind dadurch Ellipsen statt Kreise. Zur Anpassung der Gewichte im Sinne einer Minimierung des Fehlers wird folgendermaßen vorgegangen [36, S. 20]: Die Gewichte werden zufällig initialisiert, der Algorithmus steht demnach an einem zufälligen Punkt in der Karte der Höhenlinien (Abbildung 2.3(b)). Die Berechnung des (negativen) Gradienten liefert die Richtung für den stärksten Gradientenabstieg („Gradient Descent“), daher wird ein Schritt in diese Richtung vorgenommen. Der Algorithmus nähert sich sukzessive dem Minimum bis dieses erreicht ist.

Learning Rate Neben den Gewichten und den Hyperparametern für die Netz-Architektur werden weitere Parameter für das Training eines KNN benötigt. Einer ist die Lernrate („learning rate“), die – anschaulich gesprochen – repräsentiert wie weit der Algorithmus sich orthogonal zur Höhenlinie bewegt [36, S. 21]. Die Entfernung sollte idealerweise vom Gradienten der Fehleroberfläche abhängen, da nahe am Minimum kurze Schritte besser sind, um ein „Vorbeischießen“ am Minimum zu vermeiden (Abbildung 2.3(c)).

Der Gradient wird daher mit einem positiven Faktor ϵ, der Lernrate, skaliert. Diese wird üblicherweise zu einer kleinen Konstante gewählt [27, S. 82]. Im Beispiel aus Abbildung 2.3(b) wurde eine gut geeignete Lernrate gewählt; der Algorithmus konvergiert schnell zum Minimum. Bei zu kleinen ϵ kann der Trainingsprozess lange dauern, während große Lernraten zu Divergenz führen können. In Abbildung 2.3(c) wurde sie grenzwertig groß gewählt; der Algorithmus springt mehrfach am Minimum vorbei, konvergiert jedoch.

Backpropagation Beim Training, d. h. der Anpassung der Parameter eines KNN, fließt die Information vom Input-Layer über die hidden Layer vorwärts durch das Netzwerk bis zum Output-Layer. Dieser Ablauf wird als „forward propagation“ bezeichnet [27, S. 197]. In der letzten Schicht wird eine Ausgabe berechnet, die anschließend durch einen Vergleich mit dem Trainingsbeispiel zur Anpassung der Gewichte genutzt werden kann.
Bei Netzen ergibt sich gegenüber dem einfachen Neuron jedoch das Problem, dass für die Neuronen in den versteckten Schichten kein wahrer Wert für das Trainingsbeispiel zur Verfügung steht. Die Deltaregel [36, S. 22f], die die Stärke der Anpassung der einzelnen Gewichte eines einfachen Neurons bestimmt, lässt sich demnach nicht für Neuronen in hidden Layern anwenden. Stattdessen kann mittels des Backpropagation-Algorithmus', alternativ Back-Propagation oder backprop [27, S. 197], der Fehler, abhängig von der Aktivität der Neuronen, beim Training rückwärts durch das Netz propagiert werden [43, S. 3-10]. Über die Aktivitätsveränderung eines versteckten Neurons kann berechnet werden wie stark sich der Fehler verändert. Daher lässt sich der Einfluss einer einzelnen gewichteten Kante auf den Fehler bestimmen [36, S. 23]. Der Backpropagation-Algorithmus stellt die Grundlage zur Berechnung der Gradienten dar, während ein weiterer Algorithmus wie Gradient Descent im Parameterraum der Gewichte ein Minimum des Gütemaßes (der Fehler-Funktion) sucht [27, S. 198].

2.1.4 Optimierungen

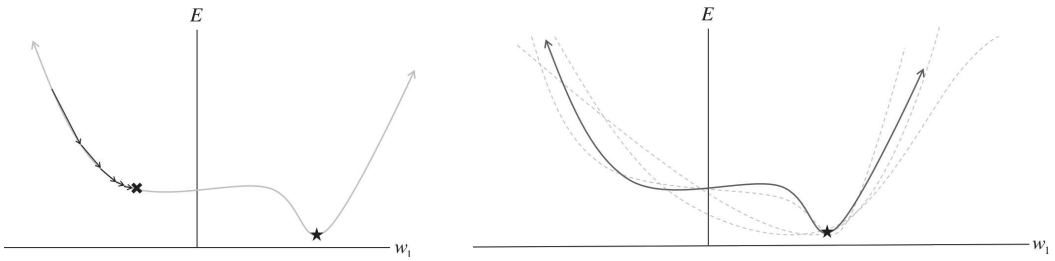
Vorangehend wurde erläutert, dass das Netztraining als ein Minimierungsproblem dargestellt werden kann. Die Gewichte werden zufällig initialisiert und anschließend zur Minimierung einer Kostenfunktion angepasst. Der Vorgang stellt in KNN ein nicht konvexes Optimierungspro-

blem dar [27, S. 171]. Bei diesem existieren verschiedene Minima und der Gradient Descent-Algorithmus garantiert nicht, dass er genau zu einem dieser Minima konvergiert. In vielen Fällen findet er jedoch schnell einen Wert nah genug am Minimum der Kostenfunktion, um nützlich zu sein [27, S. 148f]. Die Chance auf vorzeitiges Stagnieren des Algorithmus' an einem Sattelpunkt der Fehlerfunktion, d. h. einem Punkt mit Wert 0 des Gradienten, der kein Minimum darstellt [27, S. 80f], kann durch Anpassungen bzw. Erweiterungen der vorgestellten oder die Einführung neuer Algorithmen minimiert werden. Einige davon werden in diesem Abschnitt vorgestellt.

Abwandlungen des Gradient Descent-Algorithmus

Der Backpropagation-Algorithmus aus dem vorherigen Abschnitt benutzt eine Version des Gradientenabstiegs namens „batch gradient descent" [36, S. 25], auch bekannt als „Batch Learning" [46, S. 13–15]. Das Konzept dieses Ansatzes sieht vor, dass zur Berechnung der Fehleroberfläche [36, S. 25] bzw. des Gradienten der komplette Datensatz verwendet wird [46, S. 13]. Der Ansatz erhält seinen Namen folglich daher, dass zur Anpassung der Gewichte der gesamte Stapel („batch") der Daten durchlaufen werden muss [46, S. 13]. Insbesondere bei steigenden Datenmengen ist dies durch die damit einhergehende Erhöhung von Speicher- und Rechenbedarf problematisch [27, S. 148].

Stochastic Gradient Descent (SGD) Alternativ kann bei jeder Iteration der Gradient mit nur einem (bspw. zufällig gewählten) Trainingsbeispiel geschätzt werden. Der Ansatz nach diesem Prinzip nennt sich Stochastic Gradient Descent (SGD) [36, S. 25–27, 19, S. 13–15]. Es wird von „Online" statt „Batch" Learning gesprochen [46, S. 13]. Durch eine wahrscheinlichkeitsbasierte Schätzung bewegt sich der Algorithmus nicht mehr präzise entlang des wahren Gradientenabstiegs. Anhand eines einzelnen Gewichts mit zufälliger Initialisierung kann der Unterschied auf der Fehleroberfläche visualisiert werden (Abbildung 2.4). Beim Batch Learning



(a) Batch Gradient Descent (b) Stochastic Gradient Descent

Abbildung 2.4: Visualisierung des Konvergenzverhaltens am Beispiel eines einzelnen Gewichts w_1 nach [36, S. 26]

(Abbildung 2.4(a)) kann der Algorithmus in einer flachen Region stagnieren (markiert durch ein Kreuz) und das globale Minimum (mit einem Stern gekennzeichnet) wird nicht erreicht. Dagegen wird die Fehleroberfläche in Abbildung 2.4(b) jeweils nur auf Basis eines einzigen Trainingsbeispiels geschätzt (verschiedene Schätzungen sind gestrichelt angedeutet). Flache Regionen können wesentlich besser vermieden werden. SGD-basierte Backpropagation-Algorithmen können daher bestimmte Vorteile mit sich bringen, darunter [46, S. 13, 27, S. 149]:

- SGD-basiertes Lernen ist in vielen Anwendungen wesentlich schneller als Batch Learning.
- Oft werden damit bessere Lösungen erreicht.

- Die Rechenkosten für den Fehler sind beim SGD unabhängig vom Umfang des Datensatzes.

Gleichzeitig ist die Schätzung des Gradienten bzw. der Fehleroberfläche anhand eines einzelnen Trainingsbeispiel einer der größten Nachteile vom SGD-basierten Lernen. Die Schätzung kann eine ungeeignete Approximation darstellen, sodass die Berechnung nach diesem Algorithmus mehr Zeit in Anspruch nehmen kann [36, S. 27].

Mini-Batch Gradient Descent Als Mittelweg zwischen dem Batch Learning mit komplettem Trainingsdatensatz $\mathbb{D}$, bestehend aus d Trainingsbeispielen, und dem Online Learning auf Basis des i-ten Trainingsbeispiels $\boldsymbol{z}^{(i)} = (\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ aus $\mathbb{D}$, kann die Methode namens „mini-batch gradient descent“ aufgefasst werden. Dabei wird jede Anpassung der Gewichte über eine Untermenge $\mathbb{B} \subset \mathbb{D}$ durchgeführt, die als Minibatch bezeichnet wird.

Ein Minibatch enthält die Datenpaare $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(b)}$, wobei die Größe b des Minibatches einen weiteren Hyperparameter des Netzes darstellt [36, S. 27, 27, S. 148]. Für effiziente, Graphics Processing Unit (GPU)-basierte Berechnungen wird b üblicherweise als Potenz von zwei zwischen 32 und 256 gewählt [27, S. 272]. Eine einmalige Iteration über alle Trainingsbeispiele wird beim Training neuronaler Netze als Epoche bezeichnet [27, S. 239, 41, S. 274]. Eine Epoche besteht damit aus $\frac{d}{b}$ Iterationen bzw. Gewichtsanpassungen [36, S. 31]. Mithilfe des Minibatch-basierten Verfahrens kann ein Kompromiss zwischen den beiden bisher vorgestellten Verfahren eingestellt werden; die Effizienz bzw. Genauigkeit des Batch Learning wird mit der Vermeidung lokaler Minima beim SGD kombiniert [36, S. 27]. Das durch SGD eingeführte Rauschen, mit dem lokale Minima umgangen werden können, führt in der Praxis jedoch dazu, dass die Lernrate während des Trainings angepasst werden muss. Denn selbst beim Erreichen eines angestrebten Minimums verschwindet das Rauschen nicht gänzlich. Im Vergleich dazu fällt beim Batch Learning der wahre Gradient der Kostenfunktionen stetig ab und erreicht schließlich den Wert Null, während sich der Algorithmus auf das Minimum zubewegt bzw. dieses erreicht. Daher kann beim Batch Learning eine konstante Lernrate verwendet werden. [27, S.286]

Methoden zweiter Ordnung Während die vorgestellten Methoden die erste Ableitung nutzen, existieren ebenfalls Ansätze, in denen die zweite Ableitung der Kostenfunktion eingesetzt wird, um die Richtung für den Optimierungsschritt zu bestimmen [47, S. 9-10–9-15 und 12-19–12-22]. Dazu gehören bspw. die Methode des „conjugate gradient descent“ und der Gauss-Newton-Algorithmus [46, S. 31–45]. Auf diese klassischen Methoden zweiter Ordnung wird nicht weiter eingegangen, da sie im Jahr 2012 als unpraktikabel in fast allen nützlichen Fällen [46, S. 46] und 2017 nach wie vor als unpopulär in der Praxis bezeichnet wurden [36, S. 78].

Momentum

Zur Verbesserung des Konvergenzverhaltens kann ein Verfahren namens „Momentum“ [48] eingesetzt werden. Der Algorithmus minimiert über den Trainingsprozess mit stochastischen Gradientenverfahren den (negativen) Einfluss von Gradienten, die nicht in Richtung Minimum zeigen. Es wird ein gleitender Mittelwert über die vorangegangenen Gradienten gebildet, wobei die Historie von neu zu alt exponentiell abfallend gewichtet wird [36, S. 74]. Dieser Wert fungiert als eine Art Gedächtnis, über das der Algorithmus besser in Richtung des Minimums konvergieren kann. Bei jeder Aktualisierung der Gewichte wird die Anpassung der vorherigen Iteration mit dem aktuellen Gradienten kombiniert. Über den zusätzlichen Hyperparameter

$\alpha \in [0, 1)$ wird eingestellt, wie groß der Einfluss der zuvor verwendeten Geschwindigkeit auf das neue Gewichtsupdate ist [36, S. 74f]. Je größer α relativ zu ϵ gewählt wird, desto mehr der vorherigen Gradienten haben Einfluss auf die neue Richtung [27, S. 288]. α kann ebenfalls über die Iterationen angepasst werden, die Anpassung von α ist jedoch nicht so wichtig wie die Verringerung von ϵ über den Trainingsprozess [27, S. 290].

Als Variante des Momentum-Algorithmus' wurde das Nesterov Momentum [49] eingeführt. Beim Batch Learning kann dadurch die Konvergenz enorm beschleunigt werden, SGD-basierte Algorithmen erfahren bezüglich Konvergenzgeschwindigkeit keine Verbesserung [27, S. 292].

Adaptive Lernrate

Zur Anpassung der Lernrate existieren unterschiedliche Algorithmen, die in ihren Grundzügen nur umrissen werden können. Die Beschreibungen dieses Unterabschnitts sind angelehnt an [36, S. 78–82, 27, S. 298–302].

Als Anpassung von AdaGrad [50] wurde RMSProp [51] entwickelt. Während ersterer Algorithmus gute Konvergenzeigenschaften bei konvexen Funktionen aufweist und die gesamte Historie des quadrierten Gradienten nutzt, setzt letzterer einen exponentiell abnehmenden Mittelwert über ein Zeitfenster ein, um weit zurückliegende Werte zu verwerfen. Damit ist er besser für nicht-konvexe Probleme geeignet. RMSProp hat sich als effizienter Optimierungsalgorithmus für tiefe KNN herausgestellt und ist in der Praxis als Standard etabliert. Er führt zusätzlich einen Verfall-Faktor (decay factor) ρ ein, über den die Fensterlänge gesteuert wird.

Als weiterer Algorithmus zur Anpassung der Lernrate ist Adam [52] („adaptive moments“) weit verbreitet. Er kann als Kombination von RMSProp mit dem Konzept des Impulses („momentum“) angesehen werden. Bei RMSProp entstehen jedoch Abweichungen im Gradienten durch die Initialisierung, die bei Adam durch einen Korrekturfaktor berücksichtigt werden. Adam ist generell robust gegenüber der Wahl der Hyperparameter. Lediglich die Lernrate ist in bestimmten Fällen anzupassen.

2.1.5 Regularisierung

Die zentrale Herausforderung beim maschinellen Lernen ist der Transfer antrainierten Wissens auf neue Daten. Es muss sichergestellt werden, dass der Algorithmus auch bei neuen, beim Lernen nicht gesehenen Trainingsbeispielen gute Ergebnisse liefert. Diese Eigenschaft wird als Generalisierungsfähigkeit („generalization“) bezeichnet. [27, S. 107] Sie wird mit Test- oder Validierungsdaten überprüft, während für die Anpassung der Gewichte Trainingsdaten verwendet werden [27, S. 101, 117f].
Bei schwacher Ausprägung der Generalisierungsfähigkeit neigen KNN entweder zu Under- oder Overfitting. Ist ein KNN nicht ausreichend komplex, können Zusammenhänge in den Daten nicht richtig abgebildet werden; das Netz passt sich zu wenig an (Underfitting). Dagegen kann ein zu komplexes Netz dazu neigen, mehr als nur die Zusammenhänge zu erlernen, bspw. das Rauschen im Trainingsdatensatz (Overfitting). [43, S. 3-16] Durch Veränderung der Kapazität eines Netzes kann zwischen dem Hang zum Under- oder Overfitting variiert werden [27, S. 108f]. Mit der Anzahl an Neuronen steigen Kapazität und Komplexität eines Netzes. Insbesondere bei DL-Architekturen, in denen eine Vielzahl von Layern mit großen Mengen von Neuronen zum Einsatz kommen, führt die Modellkomplexität dazu, dass Maßnahmen zur Minderung bzw. Prävention von Overfitting ergriffen werden müssen [36, S. 29]. Diese Art von Strategien wird unter dem kollektiven Term Regularisierung („regularization“) geführt [27, S. 221].

L2-Regularisierung ist die am häufigsten verwendete Regularisierungsmethode [36, S. 34, 53, S. 94]. Für jedes Gewicht wird der Fehler- bzw. Kostenfunktion der Term $\frac{1}{2}\lambda w^2$ mit Regularisierungsparameter λ hinzugefügt, um extreme Gewichte zu bestrafen [53, S. 94]. Die Wahl von λ bestimmt, wie stark Overfitting vermieden werden soll. Ein $\lambda = 0$ impliziert, dass keine Regularisierung erfolgt. Das Netz wird durch den zusätzlichen Term der Gütefunktion gezwungen, alle Eingänge ein wenig zu berücksichtigen statt nur wenige viel [36, S. 34]. Die L2-Regularisierung führt letztendlich dazu, dass jedes Gewicht linear auf den Wert 0 abfällt, weshalb diese Methode als „weight decay“ bezeichnet wird [36, S. 34, 27, S. 224–226].

Data Augmentation Als eine Art Regularisierungs-Technik kann die künstliche Vermehrung („augmentation“) von Daten helfen, Overfitting zu reduzieren [20, S. 87–88, 27, S. 233f]. Für eine bessere Generalisierung wird die Verwendung von mehr Realdaten vorgeschlagen [54, S. 462, 27, S. 414]. In der Praxis ist diese Ressource jedoch limitiert, weshalb künstlich zusätzliche Daten erzeugt werden können. Bei einigen Anwendungen ist die Erzeugung dieser Daten schwierig, in anderen dagegen einfach: Insbesondere bei bildverarbeitenden Architekturen kann eine höhere Robustheit durch Verwendung zusätzlicher Bilder mit Verschiebungen um einige Pixel, Rotationen, Beschneidungen oder eingefärbten Bereichen erreicht werden. Das Einbringen von Rauschen (wie durch die Einfärbung bestimmter Bereiche) kann daher ebenfalls als eine Art Data Augmentation und somit als Regularisierung angesehen werden. [27, S. 233f]

Dropout [55] ist eine gänzlich andere Methode zur Vermeidung von Overfitting und in der Praxis mit tiefen KNN mittlerweile weit verbreitet [36, S. 36, 20, S. 87f]. Beim Dropout-Algorithmus wird eine Menge von Subnetzen, die durch den Ausschluss bestimmter Neuronen aus einer Basis-Architektur entstehen, gleichzeitig trainiert [27, S. 251]. Die Subnetze teilen sich die Parameter, was zu einer Reduktion des Speicherbedarfs führt [27, S. 253], und die Ausgabe aller Netze wird zu einem Schätzwert kombiniert. Während des Trainings wird ein Neuron (im Input- oder hidden Layer) nur mit einer Wahrscheinlichkeit p_r aktiv sein oder anderenfalls zu Null gesetzt. p_r ist ein weiterer Hyperparameter, der meist zu 0,5 gewählt wird [36, S. 103, 27, S. 256]. Mit Dropout wird verhindert, dass ein Netz zu stark von einem oder einer Kombination mehrerer Neuronen abhängt [36, S. 36]. Für eine eingehendere Beschreibung des Algorithmus' und dessen Erweiterungen sei auf [27, S. 251–261] verwiesen.

Early Stopping wird bei DL-Architekturen aufgrund seiner Effektivität und Einfachheit verwendet [27, S. 240]. Beim Training eines Netzes wird ein stetiger Rückgang des Trainingsfehlers, d. h. der Fehlerfunktion mit den Trainingsdaten, beobachtet. Dagegen steigt mit der Zeit der Validierungsfehler, d. h. die Fehlerfunktion mit Validierungsdaten [43, S. 3-16f, 27, S. 239] (Abbildung 2.5). Der Validierungsfehler wird zur Abschätzung der Generalisierungsfähigkeit herangezogen.
Early Stopping sieht vor, jedes mal, wenn der Validierungsfehler gesunken ist, eine Kopie der Modellparameter zu speichern. Kann über eine festgelegte Anzahl von Epochen keine weitere Verbesserung des Validierungsfehlers erreicht werden, wird der gespeicherte Parametersatz ausgegeben. [27, S. 239] Mit Early Stopping wird der Trainingsprozess zu dem Zeitpunkt beendet, an dem die Generalisierung aufhört und das Erlernen individueller Datenpunkte (Overfitting) beginnt [43, S. 3–16f].

Nachdem bereits Trainings- und Validierungsdatensätze angesprochen wurden, erfolgt eine kurze Einführung zu verschiedenen Datensätzen im Zusammenhang mit neuronalen Netzen.

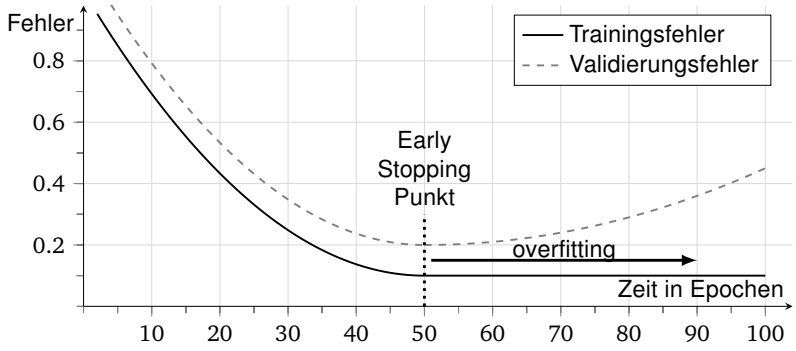


Abbildung 2.5: Qualitative Lernkurven mit Trainings- und Validierungsdaten nach [43, S. 3-17, 56, S. 17]

2.1.6 Datensätze

Es ist nicht zielführend, ein Modell (KNN) anhand der Daten zu evaluieren, mit denen es trainiert wurde, weil dadurch Overfitting nicht erkannt werden würde [36, S. 31]. Im Umfeld des maschinellen Lernens hat sich daher die Einteilung der Datenmenge in Trainings-, Validierungs- und Testdaten etabliert [36, S. 31f, 27, S. 118] (Abbildung 2.6). Möglicherweise missverständlich ist,

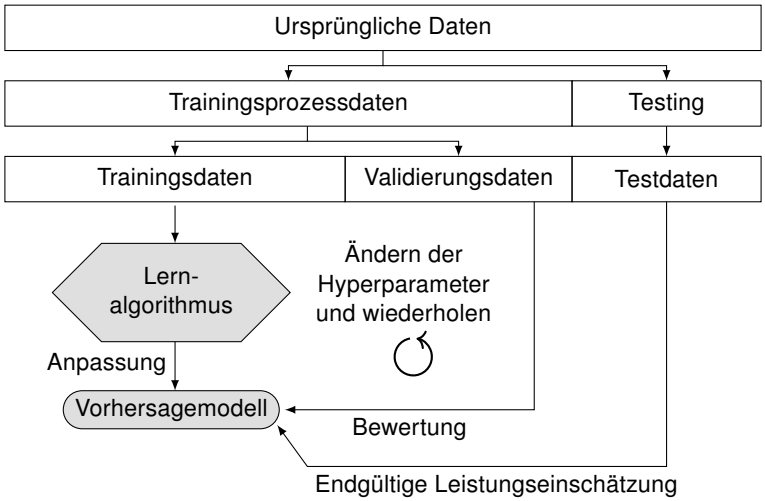


Abbildung 2.6: Einteilung der Daten nach [53, S. 206]

dass Trainings- und Validierungsdaten für das Training verwendet werden, wenn auch auf unterschiedliche Art und Weise. Sie sind zwei disjunkte Mengen, die einer übergeordneten Menge für den gesamten Trainingsprozess entstammen und typischerweise nach dem Verhältnis Trainings- zu Validierungsdaten 80:20 % eingeteilt werden [27, S. 118]. Die Trainingsdaten werden für das Erlernen der Gewichte und Bias-Terme verwendet, die Validierungsdaten dagegen zur Schätzung des Generalisierungsfehlers, wodurch die Hyperparameter entsprechend angepasst werden können. Diese werden anhand der Validierungsdaten „trainiert“. [27, S. 118] Ein bekanntes Verfahren zum Training bzw. zur Optimierung der Hyperparameter ist die Rastersuche („grid search“), bei dem für jeden Hyperparameter eine endliche Menge von Optionen existiert und das Modell mit jeder Permutation von Hyperparametern trainiert wird. [36, S. 32, 53, S. 216f]

Die obige Dateneinteilung wird auch als zweifache Kreuzvalidierung („two-fold cross-validation“) bezeichnet [53, S. 205]. Nachteilig hierbei ist, dass die Einschätzung der Leistung stark von der Einteilung der Trainingsprozessdaten in Trainings- und Validierungsdaten abhängt. Alternativ kann die robustere k-fache Kreuzvalidierung durchgeführt werden, bei der die Trainingsdatenmenge in k Teilmengen aufgeteilt wird. $k-1$ dieser Teilmengen werden für das Erlernen der

Gewichte genutzt, eine Teilmenge zur Validierung. Es entstehen k Modelle und k Gütemaße, die anschließend zu einer durchschnittlichen Leistung kombiniert werden [53, S. 206-208]. Insbesondere bei kleinen Datenmengen, die für eine einfache Einteilung in Trainings-/Validierungsdaten keine genaue Schätzung des Generalisierungsfehlers zulassen, wird dieses Verfahren empfohlen [27, S. 120].

(Un-)Ausgeglichene Datensätze Daten können nicht immer unter allen Bedingungen aufgezeichnet werden (bspw. aus Sicherheits- oder Schadensbegrenzungs-Gründen), was dazu führt, dass ein Datensatz unausgeglichen („imbalanced“) sein kann [57, S. 334–335]. In dem Fall ist die Menge der Datenbeispiele nicht gleichmäßig über die Klassen verteilt [58, S. 1264]. Unausgeglichene Datensätze werden teilweise wenig beachtet, obwohl die Folgen für das Netztraining schwerwiegend sein können [59, S. 350]. Wenn der Datensatz nicht ausgeglichen ist, führt dies zu einer Verzerrung und höherer Klassifizierungswahrscheinlichkeit für die überrepräsentierte Klasse [60, S. 1184]. Ein binäres Klassifizierungsmodell (zwei Klassen) in Form eines MLP kann bspw. ganz einfach 99 % Klassifizierungsgenauigkeit erreichen, wenn der Datensatz extrem unausgeglichen (bspw. 1000:1) ist, indem er immer die überrepräsentierte Klasse als Ergebnis liefert. Bei der Aufbereitung und Einteilung unausgeglichener Datensätze sind daher geeignete Maßnahmen wie bspw. Über- oder Unterabtastung [58, S. 1266-1270] zu ergreifen. Zudem sollte für unausgeglichene Datensätze die Wahl der Metrik sorgfältig überdacht sein.

Normierung und Standardisierung In Kombination mit dem Gardientenabstieg führt die Standardisierung zu schnellerer Konvergenz [53, S. 64f]. Durch Standardisierung wird den Daten eine Standardnormalverteilung mit Mittelwert 0 und Standardabweichung 1 verliehen. Mit einem Vektor $\boldsymbol{x_j}$, dessen Elemente die j-ten Eingangswerte aller Trainingsbeispiele n sind, wird die standardisierte Repräsentation

$$\boldsymbol{x_j}' = \frac{\boldsymbol{x_j} - \mu_j}{\varsigma_j} \tag{2.7}$$

mit Mittelwert μ_j und Standardabweichung ς_j berechnet. Sie erhält im Gegensatz zur Min-Max-Skalierung bzw. Normierung auf das Intervall $[0, 1]$ nach

$$x_{\text{norm}}^{(i)} = \frac{x^{(i)} - x_{\min}}{x_{\max} - x_{\min}} \tag{2.8}$$

Informationen über Ausreißer und sorgt für größere Robustheit. $x_{\min}$ und $x_{\max}$ bezeichnen den kleinsten bzw. größten Wert des Eingangs und $x^{(i)}$ das i-te Trainingsbeispiel. [53, S. 138f] Es wird eine Zentrierung des Mittelwerts nahe 0 empfohlen, um die Konvergenzgeschwindigkeit zu erhöhen [46, S. 16f]. Daher ist die Normierung auf das Intervall $[-1, 1]$ denkbar. Die Wahl sollte zur Vermeidung einer Sättigung der Aktivierungsfunktion jedoch von dieser abhängen [43, S. 3-14].

2.2 Grundlagen Convolutional Neural Networks (CNN)

CNN sind künstliche Neuronale Netze, die in mindestens einem Layer anstelle einer gewöhnlichen Matrixmultiplikation eine Faltungsoperation einsetzen [27, S. 321]. Dieses Unterkapitel behandelt neben der Motivation grundlegende Bausteine eines CNN sowie einige weitverbreitete

CNN-Architekturen zur Klassifikation. Zu Beginn (1990er Jahre) wurden CNN u.a. zur Erkennung handgeschriebener Ziffern [61, 62] oder zur Gesichtserkennung [63] eingesetzt. Darüber hinaus wurden viele weitere CNN-Architekturen entwickelt [64–69], die speziell in der Bildverarbeitung bzw. Computer Vision einsetzbar sind und auf die nicht weiter eingegangen werden kann. Durch steigende Rechenleistung, erhöhte Verfügbarkeit von Daten und verbesserte Algorithmen beschränkt sich der Einsatz mittlerweile jedoch nicht mehr ausschließlich auf visuelle Aufgaben [54, S.359].

Durch die Beschränkung des Umfangs dieser Arbeit können nicht alle Grundlagen ausführlich behandelt werden. Für ein vertieftes Studium wird stattdessen auf [27, 36, 41, 53, 54] und für die Studie spezieller Architekturen (Abschnitt 2.2.3) auf die jeweilige Veröffentlichung [62, 70–81] verwiesen.

2.2.1 Motivation

Bei Klassifikationsaufgaben wie der Objekterkennung werden einem herkömmlichen (flachen) ML-System Merkmale (Features) als Eingangsdaten übergeben, die durch Menschen erarbeitet und ausgewählt wurden. Dabei wird auf die Expertise derjenigen vertraut, die bestimmen, welche Merkmale für ein ML-System wichtig sein könnten. [36, S. 82f] Das Ziel von DL-Systemen wie CNN besteht daher darin, diesen mühsamen und letztendlich einschränkenden Schritt der Merkmalsextraktion und -selektion zu eliminieren [36, S. 85]. Tiefe KNN sind für diesen Zweck hervorragend geeignet, da jeder Layer des Netzes durch das Erlernen und Aufbauen von Features dazu beiträgt, eine Repräsentation der Eingangsdaten zu erreichen [36, S. 85]. CNN sind dadurch in der Lage ihre eigene Merkmalsextraktion zu synthetisieren [62, S. 7]. Sie sind eine spezielle Art von KNN, die insbesondere zur Verarbeitung von Raster-artigen Daten verwendet werden. Beispielsweise kann eine Zeitreihe als 1-D-Raster angesehen werden, bei dem zu regelmäßigen Zeitpunkten Datenpunkte aufgenommen werden. Ein Bild dagegen ist ein 2-D-Raster aus Pixeln. [27, S. 321]

Die Funktionsprinzipien von CNN werden auf den Aufbau der Sehrinde des menschlichen Gehirns zurückgeführt [53, S. 489, 54, S. 359]. Mit Experimenten an Katzen [82, 83] konnten weitreichende Erkenntnisse über den Aufbau der Sehrinde bzw. des visuellen Kortexes gewonnen werden, die in sog. Multi-Stage Hubel-Wiesel [84] Architekturen eingeflossen sind und CNN maßgeblich beeinflusst haben [19, S. 254]. Wichtige Erkenntnisse sind [54, S. 360]:

- Neuronen der Sehrinde besitzen lokale rezeptive Felder (Erfassungsbereiche).
- Diese Felder können sich für unterschiedliche Neuronen überlappen und zusammen decken sie das gesamte visuelle Feld ab.
- Einige Neuronen reagieren auf horizontale Linien, andere auf vertikale.
- Manche Neuronen besitzen größere rezeptive Felder als andere.
- Diese Neuronen reagieren auf komplexere Muster, die aus grundlegenden Mustern zusammengesetzt sind.

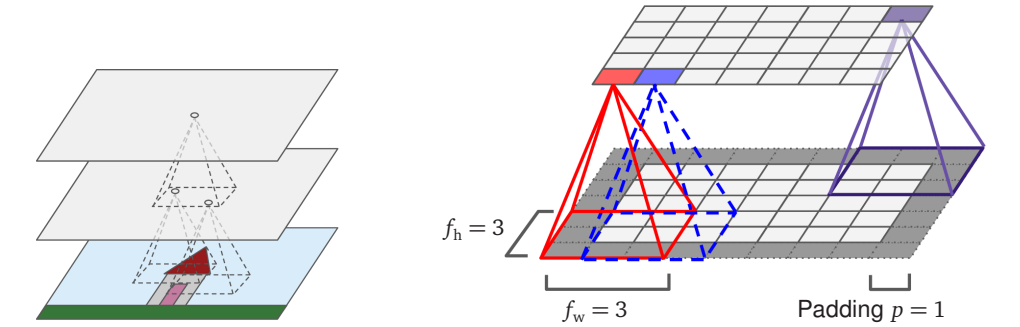
Daraus entstand die Idee, dass in höheren Schichten angeordnete Neuronen auf die Aktivierung benachbarter, tiefer angeordneter Neuronen reagieren [54, S. 360]. Inspiriert dadurch setzen CNN drei grundlegende architektonische Ideen um: Lokale rezeptive Felder („receptive fields“), geteilte Gewichte („shared weights“) und räumliche oder zeitliche Unterabtastung („subsampling“) [62, S. 2283].

2.2.2 CNN-Layer

Insbesondere angetrieben durch Wettbewerbe wie die ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [85] werden ständig neue CNN-Architekturen entwickelt, von denen einige in Abschnitt 2.2.3 behandelt werden. Die grundlegenden Bausteine dieser Netze bleiben jedoch weitestgehend unverändert [27, S. 321] und werden nachfolgend erläutert. Es wird nur auf den Fall zweidimensionaler Eingangsdaten (Bilder) eingegangen, um die Darstellung zu erleichtern.

Convolutional Layer

Konvolutionsschichten („convolutional layer“, kurz „conv layer“) stellen den wichtigsten Baustein von CNN dar [54, S. 361] und sind die Namensgeber für ebendiese Netze. Im Gegensatz zum hidden Layer eines MLP, bei dem jedes Neuron vollständig mit allen Neuronen der davor- und dahinterliegenden Schicht verbunden ist, sind Neuronen im Convolutional Layer nur mit den Pixeln ihres rezeptiven Feldes verknüpft [54, S. 361] (Abbildung 2.7(a)). Diese Eigenschaft wird als gezielte Verknüpfung („sparse connectivity“) bezeichnet [27, S. 325, 53, S. 490].



(a) Rezeptive Felder (Filter) über mehrere Schichten

(b) Filter mit stride $s = 1$ und Zero-Padding $p = 1$ („same“)

Abbildung 2.7: Illustration verschiedener Vorgänge in Convolutional Layern nach [54, S. 362]. Die unterste Schicht stellt den Input-Layer dar, während darüber ein bis zwei Convolutional Layer folgen. Eingezeichnet sind die rezeptiven Felder einzelner Neuronen. Die Neuronen im zweiten Convolutional Layer sind wiederum nur mit Neuronen innerhalb eines kleinen Rechtecks im ersten Convolutional Layer verbunden.

Ein Convolutional Layer berechnet durch eine Faltungsoperation sog. Merkmalskarten („feature maps“) [27, S. 322f]. Das rezeptive Feld kann als Filter („kernel“) angesehen werden, der Filterhöhe- und -breite f_h bzw. f_w besitzt. Diese werden häufig identisch gewählt und als „spatial extent“ e bezeichnet. Ein Filter bekommt wie im MLP einen Bias-Term, wodurch der Filter aus $f_\mathrm{h} f_\mathrm{w}$ Gewichten und einem Bias-Term besteht, die während des Trainings angepasst werden. Er wird schrittweise über das Bild bewegt (Abbildung 2.7(b)). Die Schrittweite, üblicherweise als „stride“ s bezeichnet, kann größer gleich 1 und für jede Dimension unterschiedlich gewählt werden, was zu einer Reduktion der Ausgabe-Bildgröße führt [54, S. 362f]. Beispielhaft ist eine Faltungsoperation dargestellt (Abbildung 2.8). Für jede Position der Filtermaske wird das Skalarprodukt aus Eingangsbild und Kernel gebildet und als Summe in die resultierende Feature Map I' eingetragen. Nach der Faltung wird i.d.R. eine Aktivierungsfunktion auf die Feature Map angewendet.

Das Zero-Padding, auch „Padding“, kann angewendet werden, um die Dimension des Eingabebilds zu beeinflussen [53, S. 492]. Beim Zero-Padding wird der Rand des Bildes mit Nullen aufgefüllt, um eine vollständige Berechnung zu ermöglichen. In der Praxis können unterschied-

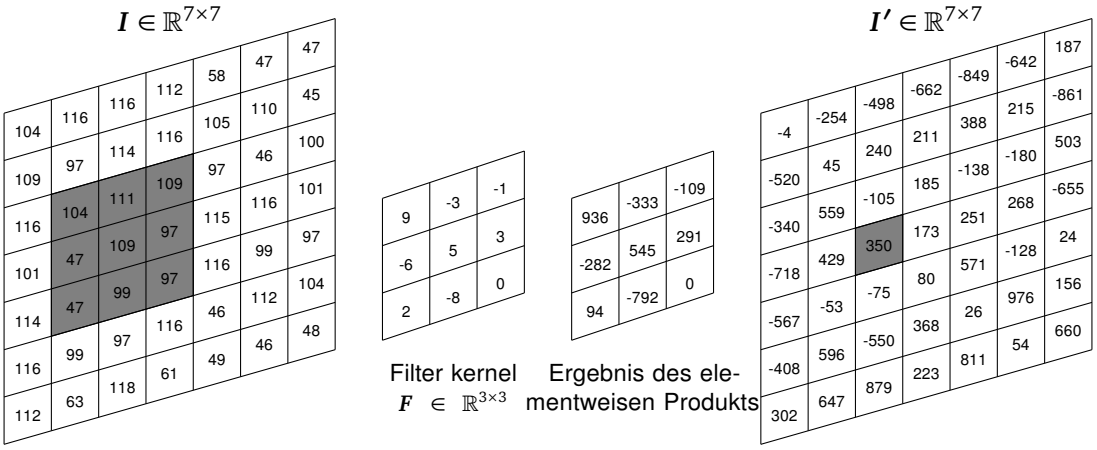


Abbildung 2.8: Anwendung einer diskreten Faltung mit einem 3×3 Filter und Stride $s = 1$ auf ein Bild der Größe 7×7. Darstellung nach [56, S. 3]. Im Gegensatz zu Abbildung 2.7(b) ist das Zero-Padding nicht explizit eingezeichnet.

liche Modi eingesetzt werden (Abbildung 2.9). Da das Full-Padding zu einer Vergrößerung

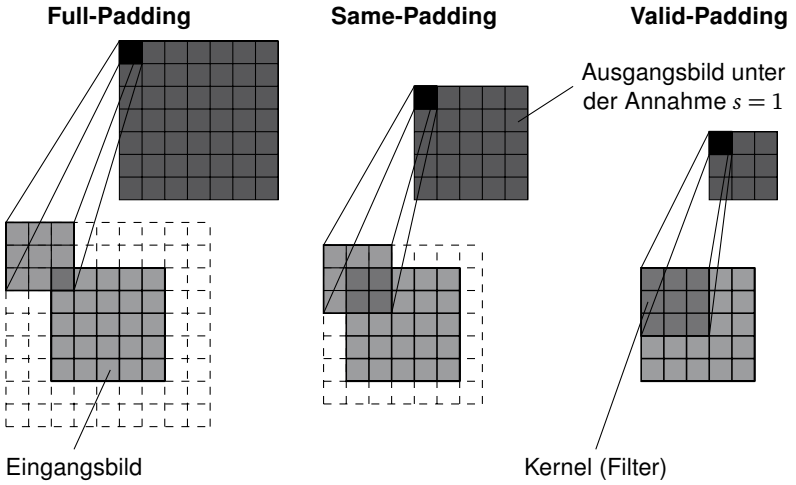


Abbildung 2.9: Darstellung unterschiedlicher Padding-Strategien nach [53, S. 495] mit einer Filtergröße $e = 3$.

des Bildes führt, wird es in CNN nur selten eingesetzt. Bei Same-Padding ist die Größe von Eingangs- und Ausgangsbild identisch, weshalb es in CNN am gebräuchlichsten ist, während Valid-Padding das Padding p zu Null setzt, was zu einer Verkleinerung des Bildes führt. [53, S. 494]

Bisher wurde der Convolutional Layer nur für ein einfaches, flaches Bild erläutert. In der Realität besitzt ein Bild jedoch meist mehrere (Farb)-Kanäle und damit neben Höhe h_{in} und Breite w_{in} eine Tiefe d_{in}, weshalb ein Convolutional Layer ein Eingangsvolumen besitzt [36, S. 91] (Abbildung 2.10). Außerdem werden statt eines einzigen Filters (Kernels) meist k Filter verwendet, um an der gleichen Position mehrere Merkmale zu extrahieren [62, S. 2283]. Die Tiefe eines Filters entspricht beim typischen Convolutional Layer der Tiefe des Eingangsvolumens [36, S. 90]. Damit dehnt sich das rezeptive Feld eines Filters über alle eingehenden Kanäle aus [54, S. 364]. Die Faltungsoperation wird dann wie in Abbildung 2.8 für jeden Kanal einzeln durchgeführt und die Ergebnisse durch Matrixsummierung addiert [53, S. 503]. Für jeden weiteren Filter entsteht eine neue Feature Map.

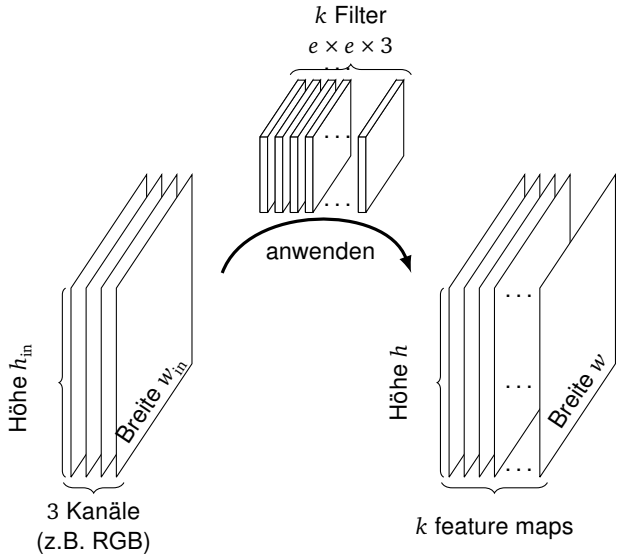


Abbildung 2.10: Anwendung eines Convolutional Layers mit k Filtern mit Größe $e \times e \times 3$, Stride $s = 1$, Padding $p = 1$ auf Eingangsdaten mit drei Kanälen. Darstellung nach [56, S. 6].

Ein Convolutional Layer besitzt unter der Annahme symmetrischer Filtermasken ($f_{\text{h}} = f_{\text{w}} = e$) $kd_{\text{in}}e^2$ Parameter und k Bias-Terme. Im Gegensatz dazu würden für einen vollständig verknüpften hidden Layer eines MLP $(eed_{\text{in}})(eek) = kd_{\text{in}}e^4$ anfallen [53, S. 504]. Dieser Unterschied kommt daher, dass die Parameter eines Filters für jeden Punkt der Feature Map gleich sind und der Filter lediglich über das gesamte Bild bewegt wird. Filter werden über den gesamten Bereich repliziert, um Merkmale in jedem Teil eines Bildes zu erkennen [36, S. 91]. Die Eigenschaft wird „parameter sharing“ genannt [27, S. 326]. Parameter Sharing in Verbindung mit Sparse Connectivity führt zu einer enormen Einsparung von Speicher- und Rechenressourcen gegenüber dem herkömmlichen MLP [53, S. 491, 27, S. 325ff].

Subsampling Layer

Ein Convolutional Layer setzt zwei der drei architektonischen Ideen von CNN um. Die dritte Idee, eine räumliche oder zeitliche Unterabtastung, wird von Subsampling Layern bzw. Pooling Layern erfüllt. Das Ziel eines Subsampling Layers ist die Verkleinerung des eingehenden Layers, um die Anzahl der Parameter und damit Rechenzeit und Speicherplatz zu verringern [54, S. 369]. Wie beim Convolutional Layer besteht eine Sparse Connectivity, d. h. jedes Neuron eines Pooling Layers ist nur mit einer begrenzten Anzahl von Pixeln oder Neuronen im vorherigen Layer verbunden. Er besitzt ebenfalls die Hyperparameter Spatial Extent e, Stride s und Padding p, jedoch keine Parameter, die während des Trainings angepasst werden. [54, S.369]

In konventionellen CNN kommen zwei Arten von Pooling-Operationen zum Einsatz: Max-Pooling und Mean-Pooling, auch als Average-Pooling bezeichnet [53, S. 500] (Abbildung 2.11). Beim Max-Pooling wird eine $\max$-Funktion auf das rezeptive Feld angewendet, beim Mean-Pooling die mean-Funktion [53, S. 500, 54, S. 369f]. Bei mehreren Kanälen wird die Operation auf jeden Kanal einzeln angewendet [54, S. 370]; die Tiefendimension bleibt erhalten. Ein Subsampling Layer dezimiert bei $s = e > 1$ die Größe der Eingabedaten mit der Pooling-Weite, d. h. dem Spatial Extent e der verwendeten Filtermaske (im Beispiel wird aus einem 4×6–Bild durch 2×2–Pooling ein 2×3–Bild) [86].
Beim Pooling Layer entsteht eine (lokale) Verschiebungs- und Störungs-Invarianz [62, S. 2284], da kleine Veränderungen in einer lokalen Nachbarschaft nicht zu einem deutlich anderen

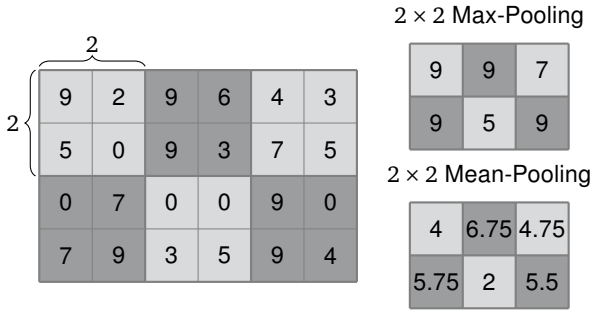


Abbildung 2.11: Anwendung einer 2×2 Max- und Mean-Pooling Operation mit $s = 2, p = 0$. Die Darstellung ist angelehnt an [56, S. 8].

Ergebnis führen [27, S. 330f, 53, S. 500]. Diese Eigenschaft ist insbesondere nützlich, wenn die reine Anwesenheit eines Features wichtiger ist als die exakte Position [27, S. 331].

Global Average Pooling Layer Als spezielle Art von Subsampling Layer wurde nach [87] erstmals mit [88] ein Global Average Pooling (GAP)-Layer eingeführt. Dieser kann als Mean-Pooling Layer mit Valid-Padding verstanden werden, in dem die Größe des Filterkernels der Größe der eingehenden Feature Maps entspricht ($f_\mathrm{h} = h_\mathrm{in}, f_\mathrm{w} = w_\mathrm{in}$) [54, S. 378]. Dadurch reduziert sich die Dimension durch einen GAP-Layer von $h_\mathrm{in} \times w_\mathrm{in} \times d_\mathrm{in}$ auf $1 \times 1 \times d_\mathrm{in}$ [87]. Jede eingehende Feature Map (Matrix) wird damit auf ein Skalar reduziert. Ein GAP-Layer kann den MLP-Teil eines CNN komplett [88] oder teilweise [70, 72] ersetzen. Dadurch wird die Anzahl der zu trainierenden Parameter und damit das Risiko von Overfitting drastisch reduziert [54, S. 378].

Normalization Layer

Nahezu jedes CNN wird aus den bisher vorgestellten Bausteinen zusammengesetzt. In tiefen KNN und damit in vielen CNN werden zusätzlich seit der Veröffentlichung von Ioffe, Szegedy [89] sog. Batch Normalization (BN)-Layer verwendet [90, S. 1]. BN-Layer stellen eine adaptive Reparamtrierung des Modells dar und tragen wesentlich zur Optimierung tiefer KNN bei [27, S. 309]. Sie begegnen dem Problem der internen Kovariatenverschiebung („internal covariate shift“): Während die Layer nahe dem Output-Layer bzw. deren Gewichte auf Basis des Eingangs tieferer Layer angepasst werden, werden diese tieferen Layer ebenfalls angepasst, wodurch sich die Verteilung des Eingangs verändert [54, S. 284]. Die Parameter der höhergelegenen Layer erfahren daher eine Verschlechterung [56, S. 9f], was die Konvergenzgeschwindigkeit beeinträchtigt.
Buduma, Locascio [36, S. 100] vergleichen die Parameterverteilung verschiedener Layer mit aufeinander gestapelten Blöcken. Die Struktur ist stabil solange der Turm aus Blöcken ordentlich gestapelt ist. Bei zufälligen Verschiebungen unterschiedlicher Blöcke wird er instabil. Ebendieses Phänomen kann beim Training von tiefen KNN auftreten [36, S. 100], da alle Layer gleichzeitig mit Werten angepasst werden, die unter der Annahme konstanter Funktionen berechnet wurden, durch die Anpassung jedoch nicht konstant bleiben [27, S.309].

Ein BN-Layer standardisiert daher den Eingang des Layers wie in Gleichung (2.7) beschrieben und benutzt zur Schätzung von Mittelwert und Standardabweichung einen (gesamten) Mini-Batch [54, S. 284]. Mit jedem Layer werden mehrere trainierbare Parameter eingeführt, darunter jeweils einer zur Skalierung und Verschiebung [54, S. 285]. Bei Verwendung eines BN-Layers kann auf die Bias-Terme im umgebenden Layer verzichtet werden, da diese redundant zum vek-

torwertigen Verschiebungsparameter sind [27, S. 312]. Die Frage, an welcher exakten Position ein BN-Layer eingefügt werden sollte (vor oder nach der Aktivierungsfunktion) ist viel diskutiert und nicht umfassend beantwortet [56, S. 10]. Die Autoren der Originalveröffentlichung [89, S. 5] schlagen einen BN-Layer unmittelbar vor der Aktivierungsfunktion vor.

Bei der Verwendung von BN können Aktivierungsfunktionen eingesetzt werden, die eine Sättigung erreichen [89, S. 2]. Weiterhin wird behauptet, dass durch den regularisierenden Charakter von BN auf weitere Regularisierung durch Dropout und L2-Regularisierung verzichtet werden kann. Darüber hinaus kann die Lernrate signifikant erhöht werden. [36, S. 106–107] Der Algorithmus wurde in einer weiteren Veröffentlichung [90] als Batch Renormalization (BRN) erweitert, der insbesondere besser für kleine und „non-i.i.d. (non-independent and identically distributed)“ Mini-Batches geeignet ist.

2.2.3 CNN-Architekturen und -Blöcke

Maßgeblich geprägt wurde die grundlegende Struktur von CNN durch die Veröffentlichung der sog. LeNet-5 [62, 91] Architektur [54, S. 361]. Mit der Zeit wurden viele Varianten abgeleitet und es sind neue „Bausteine“ für CNN entstanden. Als gutes Maß für die rapide Entwicklung kann die ILSVRC [85] angesehen werden, bei der Bilder klassifiziert werden: 2010 lag die Top-5 Fehlerrate noch bei fast 30 %, während zuletzt (2017) ein Wert von 2,25 % erreicht wurde [54, S. 371, 81, S. 1].

Klassische, sequentielle Architekturen Typische CNN-Architekturen stapeln einen bis mehrere Convolutional Layer (üblicherweise mit ReLU-Aktivierungsfunktion) gefolgt von einer Max-Pooling Operation und wiederholen diese Struktur mehrfach [54, S. 371] (Abbildung 2.12). Mit

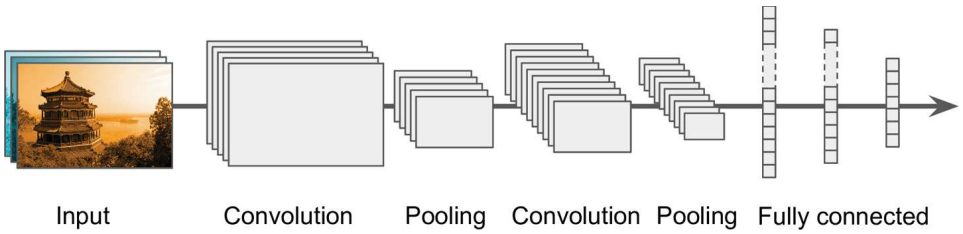


Abbildung 2.12: Typischer Aufbau eines konventionellen CNN nach [54, S. 371]

zunehmender Anzahl von Filtern in den Convolutional Layern werden die nachfolgenden Layer tiefer, während die Max-Pooling Operationen die Layer in Höhe und Breite verkleinern. An die Struktur aus Convolutional und Max-Pooling Layern wird ein klassisches Feedforward Netz (MLP) aus mehreren vollständig verknüpften (FC) Layern (meist ebenfalls mit ReLU-Aktivierung) angehangen. Der letzte Layer wird in vielen Architekturen als Softmax-Output zur Klassifikation verwendet [54, S. 371].

Als erste CNN-basierte Architektur konnte mit AlexNet [20] bei der ILSVRC 2012 ein großer Durchbruch erzielt werden, indem das Netz die Fehlerrate von 26,2 % auf 15,3 % reduzierte. Neuerungen bei AlexNet waren die Verwendung von ReLU-Aktivierungen und Dropout sowie eine GPU-optimierte Implementierung. Das Netz besitzt ca. 60 Millionen Parameter, wovon der Großteil für das MLP abfällt. Es alterniert wie LeNet-5 Convolutional und Max-Pooling Layer.

Während AlexNet die Hyperparameter k, e und s der Convolutional Layer noch variiert, wurde eine Art „Richtlinie“ zuerst durch das VGGNet [71] festgelegt. Dadurch wird der Entwurf eines Netzes „einfacher“, obwohl es gleichzeitig tiefer wird. Es setzt ausschließlich 3×3-Filter mit

Stride $s = 1$ ein, appliziert Max-Pooling nach zwei bis drei Convolutional Layern und verdoppelt anschließend die Anzahl der Filter k.
Eine Motivation für die Reduktion von e bei zwei konsekutiven Convolutional Layern ist die Reduktion von Parametern unter Beibehaltung des effektiven rezeptiven Feldes. Mit zwei 3×3-Filtern in Reihe kann ebenfalls ein rezeptives Feld von 5×5 Pixeln abgedeckt werden [56, S. 4], wobei die Aktivierungsfunktion nach dem ersten Convolutional Layer eine gehaltvollere Repräsentation erzeugt [36, S. 100]. Gleichzeitig werden nur $2 \cdot 3 \cdot 3 = 18$ statt $5 \cdot 5 = 25$ Parameter benötigt.
Beim VGGNet kommt das gleiche MLP wie bei AlexNet zum Einsatz, jedoch wesentlich mehr Convolutional Layer. Dadurch besitzt es je nach Ausprägung (acht bis elf Conv Layer) zwischen 133 und 144 Millionen Parameter. Erwähnenswert sind daher Netzarchitekturen, die auf eine möglichst effiziente und/oder parameterarme Implementierung abzielen. Dazu zählen das SqueezeNet [79], das mit einem Bruchteil der Parameter von AlexNet die gleiche Klassifikationsgenauigkeit erreicht, sowie MobileNet [80], das insbesondere für mobile und eingebettete Bildverarbeitungsanwendungen optimiert ist.

Inception Module Um sowohl global als auch lokal verteilte Merkmale besser erfassen zu können, wurde mit GoogLeNet [70] ein neuer CNN-Block eingeführt, der aus mehreren Convolutional Layern zusammengesetzt ist (Abbildung 2.13). Die Notation „$3 \times 3 + 1$ (S)“ verweist

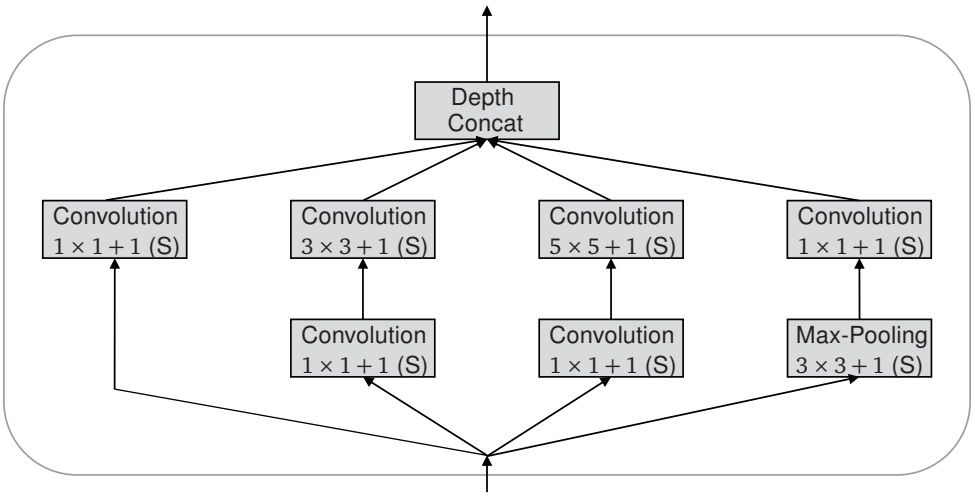


Abbildung 2.13: Inception-Modul nach [54, S. 375]

auf einen Filter mit $e = 3$, $s = 1$ und Same-Padding. Durch die Verwendung verschieden großer Kernel können unterschiedlich große Muster (Features) erfasst werden. Die Feature Maps der jeweiligen Zweige besitzen durch Same-Padding eine identische Höhe und Breite, wodurch sie durch eine Verkettung („concatenation“) in der Tiefendimension einfach aneinander gereiht werden können. Eine 1×1-Convolution dient als „bottleneck layer“ insbesondere der Dimensions- und Datenreduktion. Drei der vier Zweige stellen ein eigenes Netzwerk aus zwei Convolutional Layern dar, das jeweils über den gesamten Input angewendet wird, wodurch komplexe Features unterschiedlicher Größe erfasst werden. [54, S. 376] Die Anzahl der Zweige im Sub-Netzwerk wird durch einen weiteren Hyperparameter – die Kardinalität C – bestimmt [56, S. 12]. Der Ansatz wurde in einer weiteren Veröffentlichung verfeinert [73], indem beispielsweise die 3×3-Convolution durch eine Kombination aus 1×3- und 3×1-Convolutional Layern zur Parameterreduktion ersetzt wird.

Residual Unit Mit zunehmender Tiefe eines Netzes wird das Problem der verschwindenden Gradienten präsenter [27, S. 317f, 92, S. 253], weshalb bei GoogLeNet [70] versucht wurde, das Training durch zusätzliche „auxiliary classifier“ zu unterstützen. Die Autoren von ResNet, He et al. [72], wählten einen revolutionäreren Weg. Sie entwickelten das sog. residual learning, bei dem „skip connections“ (auch „shortcut connections“) verwendet werden, um einen möglichst ungestörten Fluss des Gradienten zu ermöglichen. Das Eingangssignal $\boldsymbol{X}_l$ eines Layers wird dem Ergebnis eines weiter „unten“ (in Richtung Output-Layer) angeordneten Layers $\boldsymbol{X}_{l+1}$ hinzugefügt (Abbildung 2.14 links). Statt einer Zielfunktion $h(\boldsymbol{X})$ wird das Netz gezwungen eine Funktion

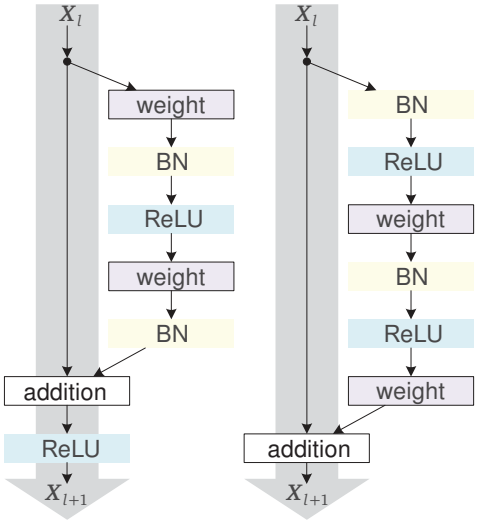


Abbildung 2.14: Darstellung unterschiedlicher Residual Units nach [76, S. 2]: Original (links) und Full Pre-Activation (rechts). Ein „weight“ Layer ist ein beliebiger Layer mit trainierbaren Parametern.

$f(\boldsymbol{X}) = h(\boldsymbol{X}) - \boldsymbol{X}$ zu lernen, was dazu führt, dass insbesondere zu Beginn des Trainingsprozesses bei einer Initialisierung der Gewichte um Null die Identitätsfunktion modelliert wird („identity mapping“) [54, S. 378f]. Dadurch kann das Training tiefer Netze beträchtlich beschleunigt bzw. ermöglicht werden. Darüber hinaus wurde die Reihenfolge verschiedener Layer (Convolution, BN, ReLU) des Identity Mapping genauer untersucht, wobei sich das Identity Mapping der Form „full pre-activation“ als beste Variante herausstellte [76] (Abbildung 2.14 rechts).
Die Autoren des DenseNet, Huang et al. [77], haben die Idee des Identity Mapping aufgegriffen und den Output jedes Layers zum Output jedes tiefergelegenen Layers addiert, um damit die Wiederverwendung von Features zu erzwingen. In einer anderen Veröffentlichung [78] wurde die Anwendung von Dropout auf ganze Layer in einem ResNet untersucht, um ein Netz mit stochastischer Tiefe zu erzeugen. Die Ideen von ResNet wurde darüber hinaus mit dem Inception-Modul kombiniert, wodurch Architekturen wie Inception-ResNet [75] und ResNeXt [74] entstanden sind.

Squeeze-and-Excitation Block

Mit dem Squeeze-and-Excitation (SE)-Block haben die Gewinner der ILSVRC „Classification“ 2017, Hu et al. [81], einen neuen Block entwickelt, um Wechselwirkungen zwischen unterschiedlichen Kanälen zu modellieren. Dieser kann unter geringem Aufwand in bestehende Architekturen eingebracht werden und führt zu einer vergleichsweise geringen Steigerung der Komplexität des Netzes [81, S. 3f]. Durch einen GAP-Layer werden die eingehenden Informationen kanalweise komprimiert („squeeze“) und anschließend in einer Abfolge von zwei vollständig verknüpften

Schichten mit ReLU- bzw. Sigmoid-Aktivierung, d. h. einem internen MLP, interpretiert („excitation“). Die eingehenden Feature Maps werden mit dieser Abfolge am Ausgang des Blocks rekalibriert bzw. skaliert (Abbildung 2.15).

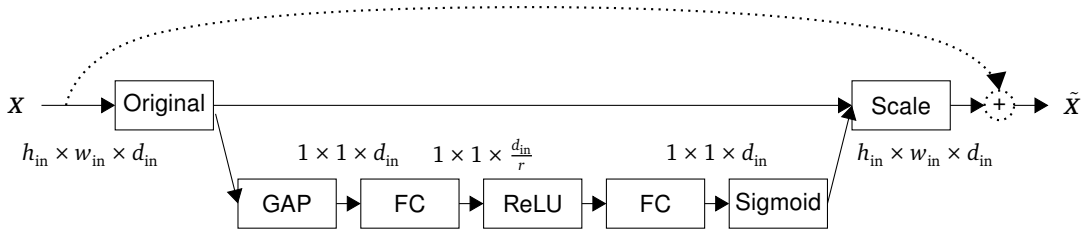


Abbildung 2.15: Darstellung eines SE-Blocks in Anlehnung an [81, S. 4], der einen beliebigen Baustein (Original) umschließt. Die Skip Connection und die anschließende Addition (gepunktet) werden nur bei einer Residual Unit angewendet. Zusätzlich sind die Dimensionen des durchfließenden Tensors X angegeben, der durch den SE-Block skaliert wird ($\tilde{X}$).

Die „reduction rate“ r stellt einen neu eingeführten Hyperparameter des SE-Blocks dar, über den die Stärke der Kompression eingestellt wird, da der erste FC-Layer als Bottleneck Layer eingesetzt wird. Als Standardwert verwenden die Autoren $r = 16$ [81, S. 4].

2.3 Automotive Diagnoseanwendungen mit maschinellem Lernen

In diesem Unterkapitel werden unterschiedliche Diagnoseanwendungen bzw. FDI-Systeme vorgestellt. Weil die Anwendungsbereiche der Fehlerdiagnose mit ML-Algorithmen vielseitig sind, wird primär auf automotive Anwendungen mit Feature Engineering-Konzepten eingegangen. Dies ist die Domäne, in der sich das zu entwickelnde Diagnosesystem befindet. Es wurden für diverse Komponenten eines Fahrzeugs bereits ML-Algorithmen zur Fehlerdiagnose verwendet.

Elektrische Antriebskomponenten wurden sowohl auf Basis von Simulationen [93, 94] als auch anhand einer Prüfstandsanwendung [95] mithilfe unterschiedlicher ML-Algorithmen diagnostiziert. Salim et al. [93] nutzen ein vibro-akustisches Modell eines Permanentmagnet-Synchronmotors zur Datengenerierung und ein KNN zur Klassifikation drei unterschiedlicher Zustände, davon zwei als „defekt“ gekennzeichnet.

Murphey et al. [94] nutzen ein Simulationsmodell, das den normalen Betrieb eines elektrischen Antriebs sowie Einzelfehler von Schaltungen und Folgefehler von Kurzschlüssen abbildet. Ein ML-Algorithmus optimiert den Parameterraum des Simulationsmodells, um daraus repräsentative Trainingsdaten für ein KNN zu generieren, das die Klassifikation auf Basis von Merkmalen der Trainingsdaten übernimmt. Zusätzlich wurde das mit Simulationsdaten trainierte System anhand von Prüfstandsmessungen validiert. Der Ansatz zeigt daher die Möglichkeit auf, ein ML-basiertes Diagnosesystem mit Simulationsdaten zu trainieren und in einer realen Applikation einzusetzen.

Kang, Kim [95] klassifizieren mit einem SVM-basierten Diagnosesystem sieben verschiedene Zustände eines Induktionsmotors. Dazu werden Beschleunigungsdaten in ein 2-D-Graustufenbild übersetzt, indem die Wavelet-Koeffizienten einer Dilatations- und Translationsparameter-Matrix berechnet und in Grauwerte umgerechnet werden. Anschließend werden mittels Nachbarschafts-Strukturkarten („neighborhood structure maps“) Textur-Merkmale aus diesen Bildern extrahiert und zur Klassifikation an eine SVM übergeben. Mit dem Ansatz wird eine durchschnittliche

Rate „wahrer Positiver“ („True Positives“) von 100 % erreicht, was im Vergleich zu fünf anderen modernen Algorithmen die höchste Klassifikationsgenauigkeit darstellt [95, S. 12]. Selbst unter Hinzufügen von (weißem Gauß-)Rauschen erzielt der ML-basierte Ansatz die besten Ergebnisse.

Sensordefekte auf Fahrzeugebene werden in einem Hardware-unabhängigen System von-Gonzalez et al. [10, 96] diagnostiziert. Die Klassifikation erfolgt auf Basis eines Adaptive Neuro-Fuzzy Inference System (ANFIS) [96] bzw. einer Fuzzy-Logik [10] in Kombination mit einem auto-assoziativen KNN. Die Messdaten umfassen zehn verschiedene physikalische Größen wie bspw. Raddrehzahlen oder Gierrate, die in einem Fahrsimulator bei normalen Betriebsbedingungen generiert wurden. Durch die Korrelation der verwendeten Variablen ist das vorgeschlagene Diagnosesystem in der Lage, fehlerhaft ausgegebene Signale zu identifizieren.
Eine Plausibilisierung von Signalverläufen nehmen Guo et al. [97] vor. Das auf einer Fuzzy-Logik basierende System übernimmt die Datenkonvertierung und -segmentierung, die Merkmalsextraktion und -kombination sowie die Klassifikation der Fahrzeugdaten. Damit werden vom System Abweichungen zum erwarteten Verhalten erkannt.

Einen Ansatz für die Fehlererkennung und -diagnose im regenerativen Bremssystem eines hybriden Elektro-Fahrzeugs liefern Sankavaram et al. [9]. Sie verwenden wie in [93, 94] ein Simulationsmodell zur Generierung der Daten und simulieren in verschiedenen Szenarien u.a. Defekte der Sensoren von Batteriestrom, -temperatur, -ladestatus, Motorstrom und -drehzahl. Residuen, d. h. Abweichungen zwischen realen Messungen und Erwartungswerten, dienen als Merkmale [9, S. 4]. Diese werden mittels Principal Component Analysis (PCA) und Multi-way Partial Least Squares (MPLS) reduziert und anschließend mit verschiedenen datengesteuerten ML-Algorithmen klassifiziert, darunter SVM, k-nearest Neighbor (k-NN) und neuronale Wahrscheinlichkeitsnetzwerke („Probabilistic Neural Network (PNN)“). Mit SVM- und k-NN-basierten Architekturen erreichen die Autoren eine Klassifikationsgenauigkeit von $100\,\%$ [9, S. 9]. Es wird die Reduktion der Datenmenge von 559 MB auf 12 KB hervorgehoben, die eine ressourcenschonende Implementierung auf einer Electronic Control Unit (ECU) zulassen würde, jedoch nicht umgesetzt wird.

Die Diagnose von automotiven Dämpfer- und Fahrwerkdefekten wird nur in wenigen Veröffentlichungen mit ML-Algorithmen untersucht [5–8]. In diesen Fällen kommen stets „flache“ Architekturen zum Einsatz und Merkmale werden manuell extrahiert statt automatisch erlernt.
Hardier [7] entwickelt ein Viertelfahrzeug-basiertes Simulationsmodell, um Beschleunigungen und Dämpferweg für das Intakt- und Verschleißverhalten eines Dämpfers zu generieren. Ein verschlissener bzw. defekter Dämpfer wird nach Hardier [7] durch ein Hysterese-Verhalten insbesondere in der Expansions-Phase charakterisiert. Um Abweichungen vom Sollzustand zu erkennen, wird ein rekurrentes Radial Basis Function Network (RBFN) eingesetzt und die Daten werden zusätzlich mit einem Rauschen beaufschlagt.

Ein weiterer Ansatz zur Diagnose von Dämpferdefekten stammt aus dem lokomotiven Bereich: Tang et al. [11] setzen eine SVM zur Detektion defekter Dämpfer ein, wobei Merkmale aus Zeit- und Frequenzbereich mithilfe einer rekursiven Merkmals-Elimination vorselektiert werden.
Der Ansatz von Liu et al. [8] basiert wie [10, 96, 97] auf einer Fuzzy-Logik und beschäftigt sich ausschließlich mit Defekten von Federn. Die umgesetzte Methode wird anhand von sechs typischen Fehlern der vorderen Federung eines VW Jetta verifiziert und liefert durchgehend zum Fehlerbild konsistente Ergebnisse.

Die Eingangs erwähnte Arbeit von Merk [5] (auch [6]) ist ebenfalls hier einzuordnen. Er untersucht

die Detektion und Isolation (Klassifikation) von Dämpferdefekten mit einem SVM-basierten FDI-System. Aus unterschiedlichen Systementwürfen wird für beide Aufgaben ein Multiclass-Binary-Konzept entworfen und optimiert, das mit zwei unterschiedlichen Datensätzen eine Genauigkeit von 73 bzw. 93 % in der Isolation und 85 bzw. 94 % in der Detektion erreicht [5, S. 66]. Wie in allen o.g. Veröffentlichungen werden manuell extrahierte Merkmale verwendet. Die Features wie spektrale Leistungsdichte [32] und Frequenzband-Energie stammen aus dem Zeit- und Frequenzbereich. Eine Auswahl (Reduktion) aller Features vor der Klassifikation erfolgt durch die Anwendung einer Recursive Feature Elimination (RFE).

2.4 Deep Learning-Ansätze im automotiven Umfeld

Wie das vorangegangene Unterkapitel zeigt, wurden herkömmliche ML-Ansätze, inbesondere Fuzzy-basierte FDI-Systeme, erfolgreich für verschiedene Bereiche im Fahrzeug entwickelt. Nach Chen et al. [38, S. 2] wurden bis 2015 zwar unterschiedliche DL-Netzwerke für verschiedene Anwendungen eingeführt, jedoch nur wenige zur Fehlerdiagnose eingesetzt. Im automotiven Kontext lässt sich diese Aussage nach jetzigem Erkenntnisstand bis heute erweitern: Es werden DL-Architekturen erfolgreich bspw. zur

- Erkennung von Fahrzeugen [98, 99], Verkehrszeichen [100] und Fußgängern [101, 102],
- Steuerung autonomer Fahrzeuge auf Basis von Kameradaten [103, 104],
- videobasierten Erkennung von Schlaglöchern [105],
- Indoor-Lokalisierung von Fahrzeugen mithilfe von Fahrzeug-Infrastruktur-Kommunikation [106],
- Überwachung des Fahrer-Zustands [107, 108],
- Vorhersage der Verkehrsflussgeschwindigkeit [109, 110] und von Fahrmanövern [111],
- Verschleißvorhersage von Polymerelektrolytbrennstoffzellen [112] und
- Detektion von Ausrichtungsfehlern zwischen mehrkanaligen Datenströmen [113]

eingesetzt. Damit decken DL-Ansätze bereits ein breites Anwendungsspektrum ab. Hinsichtlich der Fehlerdiagnose existieren abgesehen von Wälzlager- und Getriebe-Applikationen (Unterkapitel 2.5) jedoch nur wenige Ansätze [114–116].

Die Arbeit von Yin, Zhao [114] ist der lokomotiven Branche zuzuordnen, denn die Autoren nutzen ein Deep Believe Network (DBN) zur Fehlerdiagnose der Bordeinrichtung von Hochgeschwindigkeitszügen. Kanarachos et al. [115] dagegen liefern kein System zur Fehlerklassifikation, sondern lediglich zur Detektion von Anomalien. Sie erkennen anhand von Smartphone-Daten auf Basis eines nichtlinearen, auto-regressiven KNN, ob eine Straße Unregelmäßigkeiten, z. B. in Form von Schlaglöchern, aufweist [115, S. 300f].
Damit bleibt einzig die Veröffentlichung von Liao et al. [116]. Sie untersuchen die CNN-basierte Klassifikation von Zahnrad- und Wälzlagerschäden in einem automotiven Fünfganggetriebe. Es werden zwei Schadensarten in das System eingebracht: Zwei unterschiedlich große Pittings in der Außenseite eines Wälzlager-Innenrings und drei verschieden starke Schäden (Zahnbruch) am Rad der Schrägverzahnung des fünften Gangs. Mit 32×32 Pixel großen Bildern, die eine

Zeit-Frequenz-basierte Darstellung des verwendeten Beschleunigungssignals repräsentieren, wird eine Klassifikationsgenauigkeit von über 99 % erreicht.

Keine der gefundenen Veröffentlichung thematisiert die Diagnose von automotiven Fahrwerkskomponenten wie Federn oder Dämpfern mit einem DL-Ansatz. Herkömmliche ML-Ansätze für die Diagnose, inbesondere Fuzzy-basierte FDI-Systeme, wurden dagegen erfolgreich für verschiedene Bereiche im Fahrzeug entwickelt.
Für den industriellen Bereich, in dem mechanische Komponenten wie Wälzlager zum Einsatz kommen, wurden mittlerweile diverse DL-Ansätze erforscht [117–124]. Eine Zusammenfassung (Stand 2017) über den erfolgreichen Einsatz von DL-Architekturen im Bereich der Fehlerdiagnose von mechanischen Komponenten geben Jing et al. [12, S. 2].

Da diese Arbeit einen CNN-basierten Ansatz anstrebt, wird im Folgenden nur auf Veröffentlichungen mit ebendieser Architektur eingegangen und die Literatur insbesondere aufgrund der Nähe zum behandelten Thema auf die Diagnose mechanischer Komponenten eingegrenzt.

2.5 CNN-Architekturen zur Fehlerdiagnose mechanischer Komponenten

In der Medizintechnik existieren in diversen Bereichen erfolgreiche Diagnoseanwendungen von CNN [125–129], was unter anderem auf die Verwendung von Bildmaterial wie Computertomographie (CT)-Aufnahmen zurückzuführen ist. Im industriellen Maschinenbau wurden erste CNN-basierte Ansätze bspw. zur Prädiktion der verbleibenden Nutzungsdauer von Flugzeugtriebwerken [130] sowie Werkzeugmaschinen [131] und zur Vorhersage der Abnutzung von Werkzeugen eingesetzt [132]. Die CNN-basierte Fehlerdiagnose von mechanischen Bauteilen wird erst seit wenigen Jahren erforscht und die meisten Veröffentlichungen befassen sich mit Wälzlagern [13, 18, 59, 116, 133–143].

Die Architektur typischer CNN ist für Eingangsdaten in Matrixdarstellung (Bilder) ausgelegt [62]; Messdaten liegen jedoch meist als Zeitreihe (Zeilenvektor) vor. Um Vektoren statt Matrizen für das Netztraining verwenden zu können, sind nur wenige Anpassungen des CNN notwendig [144, S. 669, 145, S. 50]. Alternativ können die Daten mit geeigneten Methoden auf eine 2-D-Darstellung transformiert werden. Dadurch existieren unterschiedliche Ansätze, bei denen die Daten entweder in Form eines Vektors (1-D) [21, 135, 144, 145] oder einer Matrix (2-D) [12, 18, 116, 141] vorliegen. In anderen Anwendungen werden höherdimensionale wie bspw. 3-D-CNN-Ansätze [146, 147] umgesetzt.

Nachfolgend wird zur besseren Trennung vorhandener Ansätze einerseits eine Unterscheidung nach Dimension der Eingangsdaten der Netze vorgenommen und andererseits die Vorverarbeitung der Daten unterschieden.

2.5.1 Zeitbasierte 1-D-Architekturen

Inspiriert durch die hohe Klassifikationsrate von CNN bei der Verwendung von Zeitsignalen wie Sprachaufnahmen [86, S.2] beschäftigen sich einige Veröffentlichungen mit der möglichst direkten Nutzung der (Roh-)Daten zur CNN-basierten Fehlerdiagnose [12, 21, 134–139, 144, 148]. Es kann bereits eine minimale Datenaufbereitung ausreichen, um gute Vorhersageergebnisse zu erzielen [86, S. 3, 149, S. 195].

Ince et al. [135] diagnostizieren Lagerschäden von Induktionsmotoren auf Basis des gemessenen Motorstroms, der mit einem Bandsperrfilter zweiter Ordnung vorverarbeitet wird. Das „adaptive“ 1-D-CNN besitzt eine sequentielle Architektur mit drei Convolutional- und Subsampling- sowie zwei FC-Layern. Es wird demonstriert, dass die einfache Architektur, vor allem begründet durch skalare Operationen, echtzeitfähig ist [135]. Der Vergleich mit drei ähnlich komplexen, weitverbreiteten Klassifizierern (MLP, RBFN, SVM) zeigt, dass mit 1-D-CNN hohe Klassifikationsgenauigkeiten (über 97 %) erzielt werden können ohne Merkmale händisch extrahieren zu müssen [135]. Die Autoren setzen mittels Fast Fourier Transformation (FFT) und Wavelet Packet (WP)-Decomposition extrahierte Merkmale ein. Eren [134] nutzt dieselbe adaptive Architektur, um Fehler in Wälzlagern auf Basis von Schwingungsdaten zu detektieren. In beiden Veröffentlichungen werden die Daten jedoch nur in einem einzigen, stationären Betriebspunkt erhoben. Die adaptive 1-D-CNN-Architektur wird darüber hinaus zur Diagnose von Stahlstreben-Strukturen eingesetzt [21, 22, 145].

Jia et al. [59] nutzen unaufbereitete Beschleunigungsdaten zur Fehlerdiagnose der Lagerung in einem Hydraulikprüfstand. Der Fokus liegt auf unausgeglichenen Datensätzen und Normalization Layern. Sie stellen einen Bezug zur Frequenzdomäne her, indem gezeigt wird, dass die Filter des ersten Convolutional Layers in der Lage sind unterschiedliche charakteristische Frequenzen abzubilden, was zu der Klassifikationsgenauigkeit von über 95 % beiträgt.
„Tiefere“ Architekturen unter Verwendung von Rohdaten (gemessene Beschleunigung an einem Wälzlager) zur mechanischen Fehlerdiagnose wurden von Zhang et al. [137, 139] vorgeschlagen. Im ersten Ansatz [137] werden in den vorderen Schichten des Netzes große Filtermasken eingesetzt, die hochfrequentes Rauschen besser als kleine Kernel unterdrücken sollen. Zusätzlich wird neben BN ein statistischer Ansatz zur Domänenadaption eingesetzt, um eine Verbesserung der Klassifikationsgenauigkeit zu erreichen. Problematisch ist, dass der Algorithmus statistische Kenntnisse über die gesamten Testdaten erfordert. In der Praxis sind diese Kenntnisse jedoch nur teilweise vorhanden und deshalb wird von Teilen der Daten auf die Gesamtheit der Daten geschlossen [137, S. 20]. In einem zweiten Ansatz [139] werden daher zwei Methoden eingesetzt, um die Fähigkeit zur Generalisierung und die Domänenadaption zu verbessern: Erstens wird Dropout im ersten Convolutional Layer verwendet, um den Rohdaten Rauschen „hinzuzufügen“. Zweitens wird beim Netztraining die Größe der Mini-Batches extrem klein gewählt (sie entspricht in der Veröffentlichung der Anzahl der Klassen, d. h. zehn).

Während die meisten Veröffentlichungen nur ein einziges Signal verwenden, zeigt die Veröffentlichung von Jing et al. [148], dass mehrere Kanäle mit Rohdaten in einer Architektur fusioniert werden können. Sie kombinieren Beschleunigungs-, Geräusch- und Strommessungen sowie eine errechnete Winkelgeschwindigkeit in einem CNN zur Fehlerdiagnose von Planetengetrieben. Die Trainingsdaten wurden bei unterschiedlichen Drehzahlen im lastfreien Betrieb aufgenommen. Die Datenfusion wird einerseits weggelassen, d. h. Netze werden auf Basis einzelner Sensorsignale trainiert, und andererseits zu Vergleichszwecken auf Daten-, Merkmals- und Entscheidungsebene untersucht. Zusätzlich stellen die Autoren einen Vergleich des Ansatzes zu KNN und SVM mit manuell extrahierten Merkmalen an, wobei die beste durchschnittliche Genauigkeit mit über 99 % durch den Einsatz eines tiefen CNN und Datenfusion auf Rohdatenebene erreicht wird.

2.5.2 Frequenzbasierte 1-D-Architekturen

Alternativ zur Zeitdomäne wird mehrfach die Darstellung der Daten im Frequenzbereich gewählt [12, 119, 148]. Diese Darstellung liegt bei rotierender Maschinerie nahe, da hierdurch die

einzelnen Komponenten am besten von einander getrennt bzw. auseinandergehalten werden können und ihr jeweiliger (Verschleiß-)Zustand abgeschätzt werden kann [119, S. 307].

(Fast) Fourier Transformation Jing et al. [12] transformieren Ausschnitte (Sequenzen) der gemessenen Beschleunigung eines Getriebes in drei Betriebspunkten mittels FFT ins Frequenzspektrum, um mit diesen Eingangsdaten ein CNN als Condition Monitoring System zu trainieren. Es werden zwei Datensätze genutzt, die an Prüfständen durch Abtastung im zweistelligen Kilohertz-Bereich gewonnen wurden. Die Länge der Trainingsbeispiele wird von 1024 bis 12288 Datenpunkte variiert, wobei das beste Ergebnis je nach Datensatz mit einer Länge von 4096 bzw. 6144 Datenpunkten erreicht wird. Das Training mit den Daten im Frequenzbereich führt zu einer verbesserten Klassifikationsgenauigkeit (99 %) gegenüber der Verwendung von Trainingsdaten im Zeitbereich (48 %) [12, S. 6]. Zusätzlich untersuchen die Autoren kombinierte Zeit-Frequenz-Daten. Sie erzielen damit jedoch keine Verbesserung. Stattdessen schlagen sie eine 2-D-Zeit-Frequenz-Darstellung vor, bspw. erzeugt durch eine Short-Time Fourier Transformation (STFT).

Hüllkurvenspektrum Appana et al. [133, S. 190–191] nutzen Hüllkurvenspektren („Envelope Spectrum“ (ES)), um eine drehzahlunabhängige Fehlerdiagnose zu ermöglichen. Als Daten liegen aufgenommene Geräuschemissionen eines Wälzlagers in vier unterschiedlichen Zuständen bei variierender Drehzahl vor [133, S. 194]. Auf das Rohsignal wird eine Hilbert-Transformation und anschließend eine Fourier Transformation angewendet, woraus das ES resultiert [133, S. 192]. Auffällig ist der kleine Datensatz mit nur 360 Trainingsbeispielen pro Drehzahlstufe, mit dem bei 20-facher Kreuzvalidierung eine durchschnittliche Klassifikationsrate von über 90 % erreicht wird [133, S. 196]. Bei genauerer Betrachtung der Messdaten fällt auf, dass die verschiedenen Fehler bereits in den Rohsignalen zu deutlich wahrnehmbaren Charakteristiken führen.

2.5.3 Zeitbasierte 2-D-Architekturen

Manche Vorteile von CNN gelten nicht bei der Verwendung von 1-D-Daten [139, S. 440]. Insbesondere die Verringerung der benötigten Parameter bei zwei konsekutiven Convolutional Layern gilt nicht im 1-D-Fall, denn mit zwei 3×1 Faltungen wird zwar ein effektives rezeptives Feld von 5×1 Datenpunkten abgedeckt, es werden jedoch $2 \cdot 3 = 6$ statt 5 Parameter benötigt. Die Einsparung bei 2-D-Convolutions liegt für das Beispiel bei $1 - \frac{18}{25} = 28\,\%$. Daher werden unterschiedliche Transformationsverfahren eingesetzt, um übliche (2-D-)CNN-Architekturen verwenden zu können.

2-D-Graustufenbild Schwingungssignale besitzen periodische Anteile, daher ist ein Datensatz zum Zeitpunkt t nicht nur zu den benachbarten Datenpunkten korreliert, sondern auch zu Datenpunkten in anderen Intervallschritten [143, S. 1]. Für die Anwendung zur Fehlerdiagnose von Rollenkugellagern mit Schwingungssignalen schlagen Zhang et al. [143] daher die Transformation der 1-D-Daten auf ein Datenbild vor. Durch die zeilenweise Darstellung können weiter entfernte Datenpunkte einbezogen und periodische Merkmale besser erkannt werden [143, S. 4]. Eine einfache Transformation lässt sich vornehmen, indem das Rohsignal bspw. in n gleiche Teile zerlegt und anschließend jedes dieser Teile zeilenweise untereinander angeordnet wird [143, S. 2]. Durch die Transformation eines (2400×1)–Signals auf ein 60×40

Pixel großes Bild erreichen die Autoren eine durchschnittliche Klassifikationsgenauigkeit von über 99 %. Ähnlich gute Ergebnisse werden mit einem Rohdaten-basierten 1-D-CNN und einem herkömmlichen KNN mit FFT-basierten Merkmalen erreicht [143, S. 4].

Wen et al. [150] setzen mit derselben Motivation Graustufenbilder zur Fehlerdiagnose mit einer „LeNet5"-Architektur [91] und drei unterschiedlichen Datensätzen ein: Beschleunigungsdaten von einem Motorlager, einer selbstansaugenden Kreiselpumpe sowie einer Axialkolbenpumpe. Die Ergebnisse werden mit ML-Verfahren (MLP, SVM) und DL-Verfahren (Sparse Filter, CNN [140], sowie herkömmliche und hierarchische DBN) verglichen, wobei die LeNet5-Architektur die besten Ergebnisse erreicht. Es fällt auf, dass die SVM mit durchschnittlich 87,45 % gerade die Klassifizierungsgenauigkeit des herkömmlichen DBN erreicht, während das MLP mit 67,70 % die schlechteste Leistung zeigt und die DL-Ansätze in einem gemeinsamen Bereich liegen (98,1–99,79 %) [150, S. 5994].

2-D-Repräsentation der Zeitreihe und Gramian Angular Field Krummenacher et al. [60] untersuchen die Verwendung einer 2-D-Repräsentation der Zeitreihe (Graph bzw. Plot) und Gramian Angular Field (GAF) zur Detektion von Unrundheit sowie Abplattungen von Eisenbahnwagenrädern. GAF wurden nach [60] in [151] zur zweidimensionalen Kodierung von Zeitreihen vorgeschlagen, weil damit zeitübergreifende Zusammenhänge erfasst und die Klassifikationsergebnisse im Zusammenhang mit CNN verbessert werden können.
Mit der Datenrepräsentation als GAF wird die gleiche Klassifizierungsgenauigkeit (87 %) erreicht wie mit dem Feature Engineering-basierten Ansatz der SVM, während unter Verwendung der 2-D-Repräsentation der Zeitreihe 89 % erzielt wurden. In beiden Fällen konnte aufgrund der CNN-Eigenschaft des Feature Learnings auf die manuelle Extraktion von Merkmalen verzichtet werden. Die verwendeten Architekturen sind auf die spezielle Anwendung mit verteilten Messstellen entlang der Schienen zugeschnitten.

2.5.4 Zeit-Frequenz-basierte 2-D-Architekturen

Da die Signale Rauschanteile enthalten, sind Algorithmen, die Informationen aus der Zeitdomäne transformieren, möglicherweise nicht robust genug, um Merkmale aus den Bildern extrahieren zu können [95, S. 2]. Daher wurden auch frequenzbasierte 2-D-Architekturen untersucht. Zeit-Frequenz-Methoden repräsentieren ein Signal gleichzeitig in der Zeit- und Frequenzdomäne, wobei unterschiedliche Transformationen zum Einsatz kommen.

Diskrete Fourier Transformation (DFT) Angelehnt an die rein frequenzbasierten 1-D-Verfahren nutzen Janssens et al. [18] in einem „flachen Bild" zwei Zeilen mit Signalen von zwei Beschleunigungssensoren, die zueinander rechtwinklig an einem Getriebegehäuse montiert sind. Die bei konstanter Geschwindigkeit aufgenommenen Messdaten werden in einminütige Fenster eingeteilt, die sich zu 50 % überlappen. Mit einer diskreten Fourier Transformation (DFT), die üblicherweise mit einem FFT-Algorithmus umgesetzt wird [152, S. 165], werden die Daten in den Frequenzbereich transformiert und zum Training eines flachen CNN verwendet. Dieses besteht nur aus einem Convolutional Layer (ohne anschließendes Pooling) und einem MLP-Layer. Im Vergleich mit einem Ansatz, bei dem Merkmale manuell extrahiert werden, zeigt der CNN-basierte Ansatz eine Verbesserung der Klassifikationsgenauigkeit von ca. 6 %, wobei der Algorithmus die Merkmale eigenständig extrahiert bzw. erlernt [18, S. 343].

Short-Time Fourier Transformation (STFT) Die STFT stellt eine visuelle Repräsentation der Daten dar, wobei auf der Abszisse die Zeit und auf der Hochachse die Frequenz abgebildet wird. Die Farbe des Bildes kodiert die Amplitude der Frequenz. [142, S. 4] Durch die Verwendung eines zeitlich verschiebbaren Analysefensters können Teile des Signals außerhalb eines bestimmten Bereichs (dem Analysefenster) unterdrückt und eine Fourier-Transformation für die Extraktion des lokalen Spektrums eingesetzt werden. [153, S. 285]
Die STFT kommt in mehreren CNN-basierten Diagnoseanwendungen zum Einsatz [13, 116, 142]. Einen Vergleich mit herkömmlichen Architekturen oder anderen Transformationsverfahren liefern Verstraete et al. [142]. Sie diskutieren die Verwendung von STFT, Wavelet Transformation (WT) und Hilbert-Huang Transformation (HHT) zur Vorverarbeitung der gemessenen Beschleunigung aus zwei verschiedenen Datensätzen an einem Wälzlager. HHT-Bilder kommen in [142] zum ersten Mal zur Fehlerdiagnose zum Einsatz [142, S. 2]. Der Einfluss einer Skalierung von einem 96×96 Pixel großen Bild auf das Format 32×32 wird ebenso untersucht wie der Einfluss von künstlich eingebrachtem Rauschen. Es zeigt sich, dass der Informationsverlust durch Verkleinerung sich negativ auf alle Ergebnisse, am stärksten jedoch auf die mit HHT-Vorverarbeitung auswirkt [142, S. 7]. Die vorgeschlagene Architektur, die den Prinzipien eines VGGNet [71] folgt, wird als ausschlaggebend für eine höhere Robustheit gegen Rauschen herausgestellt [142, S. 12]. Es erfolgt zusätzlich ein Vergleich zu unterschiedlichen Feature Engineering-Ansätzen (MLP, SVM). Die Klassifikationsgenauigkeit ist im Vergleich mit allen anderen Architekturen bei der CNN-basierten Methode am höchsten. Die HHT als Vorverarbeitung führt zu den schlechtesten Ergebnissen. Mit Eingangsdaten in Form von Spektrogrammen (STFT) liegt die Klassifikationsgenauigkeit hinter der mit Daten als Skalogrammen (WT) [142, S. 8–15]. Liao et al. [116] bewerten die WT ebenfalls besser als die STFT.

Wavelet Packet Energy Image (WPI) Der von Ding und He [154] vorgestellte Ansatz zur Klassifikation von zehn Verschleiß- bzw. Defektzuständen eines Spindellagers unter vier verschiedenen Lasten nutzt eine CNN-Architektur mit einer Frequenz-Energie-basierten 2-D-Darstellung der Trainingsdaten. Das Netz besteht aus sechs hidden Layern, davon drei Convolutional, zwei Max-Pooling- und einem sog. „Multiscale“–Layer. Die Feature Maps des dritten Convolutional Layers und des zweiten Max-Pooling Layers werden ungefaltet zusammen als Eingang für den Multiscale-Layer verwendet [154]. Ein Trainingsbeispiel wird mittels Wavelet Packet Transformation (WPT) in eine Reihe von Frequenz-Unterräumen geteilt und die Koeffzienten anschließend zur Rekonstruktion mehrerer Signale genutzt, deren Energieanteile in ein Wavelet Packet Energy Image (WPI) übertragen werden. Dadurch wird ein neuer lokaler Zusammenhang zwischen nahegelegenen WP-Knoten geschaffen [154]. In der Veröffentlichung führt die Verwendung eines 32×32 Pixel großen WPI zu einer durchschnittlichen Klassifikationsgenauigkeit von über 98 % [154].

2.5.5 Zusammenfassung von CNN-Diagnoseanwendungen mit mechanischen Bauteilen

Die Quellen aus Unterkapitel 2.5 werden in Tabelle 2.1 zusammengefasst. Es wird deutlich, dass bereits viele verschiedene CNN-Architekturen zur mechanischen Fehlerdiagnose zum Einsatz gekommen sind. Sowohl für 1-D- als auch 2-D-Architekturen existieren Ansätze mit Trainingsdaten im Zeit- und Frequenzbereich.
Die meisten Anwendungen beziehen sich auf Wälzlager, wobei die Netztopologien von flachen Varianten mit einem C und anschließendem MLP- bzw. FC-Layer [18] bis hin zu 11-schichtigen

Tabelle 2.1: Tabellarische Zusammenfassung intelligenter CNN-Diagnoseanwendungen bei mechanischen Teilen. Die Tabelle ist nach Input-Dimension und Komplexität der Netztopologie sortiert. Die Abkürzungen bezeichnen Convolutional Layer (C), FC, Max-Pooling Layer (MP), Mean-Pooling Layer (MP'), Softmax-Output (SM)

Quelle	Dimension	Zeitbasiert	Frequenzbasiert	Wälzlager	Andere	Netztoplogie	Aktivierung
Jing et al. [12]	1-D	x	x		x	C-MP-FC-SM	k.A.
Abdeljaber et al. [21, 22] und Avci et al. [145]	1-D	x			x	C-SS-C-SS-FC-FC	tanh
Appana et al. [133]	1-D		x	x		C-MP-C-MP-FC-SM	ReLU
Ince et al. [135]	1-D	x		x		C-SS-C-SS-C-SS-FC-FC	k.A.
Eren [134]	1-D	x		x		C-SS-C-SS-C-SS-FC-FC	k.A.
Zhang et al. [138]	1-D	x		x		C-MP-C-MP-FC-SM	ReLU
Jing et al. [148]	1-D	x			x	C-MP-C-MP-C-FC-SM	ReLU
Jia et al. [59]	1-D	x		x		C-MP-C-MP-FC-FC-SM („DNCNN“)	ReLU, sigmoid
Zhang et al. [137]	1-D	x		x		C-MP-C-MP-C-MP-C-MP-C-MP-FC-SM („WDCNN“)	ReLU
Zhang et al. [139]	1-D	x		x		C-MP-C-MP-C-MP-C-MP-C-MP-FC-SM („TICNN“)	ReLU, sigmoid
Pan et al. [136]	1-D	x		x		3 Split, 3 Predict, 3 Update, 1 MP, 1 FC („LiftingNet“)	ReLU
Janssens et al. [18]	2-D		x	x		C-FC	tanh
Hoang, Kang [141]	2-D	x		x		C-SS-C-SS-FC	ReLU
Zhang et al. [143]	2-D	x		x		C-MP-C-MP-FC-SM	ReLU
Liao et al. [116]	2-D		x	x	x	C-MP'-C-MP'-FC	sigmoid
Ding, He [154]	2-D		x		x	C-MP-C-MP-C-MS-SM	sigmoid
Dong et al. [13]	2-D		x	x		2 Stufen: C-MP-C-MP–DBN	ReLU
Wen et al. [150]	2-D	x		x	x	C-MP-C-MP-C-MP-C-MP-FC-FC	k.A.
Guo et al. [140]	2-D	x		x		C-MP-C-MP-C-MP-FC-FC-SM (4x)	sigmoid
Krummenacher et al. [60]	2-D	x	x		x	Cyclic Permutation Network, MIL-DNN	pReLU, tanh
Verstraete et al. [142]	2-D		x	x		C-C-MP-C-C-MP-C-C-MP-FC-FC	ReLU

Netzwerken [142] reichen. Einige Veröffentlichungen spezifizieren die Subsampling Layer (SS) nicht weiter, die meisten nutzen jedoch MP oder MP' zur Dimensionsreduktion. In vielen der o.g. Veröffentlichungen wird ein SM zur Klassifikation genutzt. Zusätzlich ist dargestellt, welche Aktivierungsfunktion für das jeweilige Netz gewählt wurde.

2.6 Herleitung der Aufgabenstellung

Die Masterarbeit von Merk [5] zeigt, dass die Dämpferdefektdiagnose mit Methoden des maschinellen Lernens ohne Einbringung zusätzlicher Sensorik in gering-dynamischen Situationen mit einer signifikanten Genauigkeit möglich ist. Es wird jedoch ein Großteil der Arbeit, die Merkmalsextraktion und -selektion, vom Menschen übernommen bzw. vorgegeben.

DL-Ansätze stellen eine gute Möglichkeit dar, diesen Schritt zu automatisieren. Insbesondere die DL-Architektur CNN ist in der Lage, optimale Merkmale selbstständig zu synthetisieren [62, S. 7]. Das bedeutet, dass Features nicht mehr durch Expertenwissen abgeleitet werden müssen [18]. Obwohl bspw. in [12] eine flache CNN-Architektur mit nur jeweils einem Convolutional, Sub-

sampling und FC-Layer gewählt wurde, zeigt der Vergleich mit Feature Engineering-basierten Modellen, dass das selbstständige Erlernen von Merkmalen zu einer signifikanten Verbesserung der durchschnittlichen Klassifikationsgenauigkeit führen kann (bspw. $99,33\,\%$ bei CNN gegenüber $85,46\,\%$ mit SVM) [12]. Es ist daher besonders interessant CNN als FDI-System zu untersuchen, da einerseits aufwändige Arbeitsschritte entfallen und andererseits verbesserte Ergebnisse erwartet werden können.

Die Literaturrecherche zeigt, dass unterschiedliche CNN-Architekturen in diversen Prüfstandsanwendungen erfolgreich zur mechanischen Defektdiagnose eingesetzt wurden. Dagegen wurde die Verwendung von Daten unter Realbedingungen (keine Prüfstandsanwendung) nur in wenigen Veröffentlichungen untersucht [21, 22, 60, 116]. Davon kann einzig [116] der Automobilindustrie zugeordnet werden. Die Autoren Liao et al. [116] fokussieren mit der Wälzlager- und Zahnraddiagnose allerdings den Antriebsstrang. Die Diagnose von Fahrwerkskomponenten wie Federn und Dämpfern mit DL-Architekturen wurden dagegen nicht untersucht. Die Untersuchung eines DL-Ansatzes mit realen Fahrzeugdaten zur Diagnose von Fahrwerkskomponenten erfolgt damit erstmalig in dieser Masterarbeit.

Zusätzlich kann festgestellt werden, dass aktuelle Trends bzw. Entwicklungen wie Inception-, Residual- oder SE-Bausteine bisher nicht in CNN-Architekturen für industrielle FDI-Systeme eingeflossen sind. Im Vergleich zur Bildverarbeitung wurde die Netztopologie flach gehalten (Tabelle 2.1), weshalb der Einfluss bzw. die Übertragbarkeit unterschiedlicher CNN-Architekturen zu untersuchen ist.

Problematisch ist die Verwendung von mehr als einem Sensorsignal, wodurch es zu einer Erhöhung der Unsicherheit und Ungenauigkeit sowie unterschiedlichem Rauschverhalten kommt [148]. Studien haben jedoch belegt, dass Ende-zu-Ende Systeme, d. h. die Verwendung von Rohdaten als Input, umsetzbar sind [86] und sich zur Defektdetektion mehrere Sensorsignale in einer CNN-Architektur fusionieren lassen [148]. Die Fusion der Daten kann auf unterschiedliche Art und Weise erfolgen, weshalb verschiedene Fusionsarten untersucht werden. Zusätzlich wird eine Untersuchung des Einflusses unterschiedlicher Sensorsignale für das FDI-System angestellt.

Die Literaturrecherche zeigt weiterhin, dass hinsichtlich der Vorverarbeitung von Daten kein Trend erkennbar ist: Die Länge einer Datensequenz für ein Trainingsbeispiel reicht von wenigen Hundert [135] bis zu mehreren Tausend [12, 139] Datenpunkten. Ebenso variiert bei den 2-D-Architekturen die Bildgröße von 21×21 [141] über 96×96 [142] bis hin zu 200×200 Pixeln [143]. Die Aufbereitung der Daten wurde bereits mit DFT/FFT, WT, STFT, WPI, HHT und weiteren Verfahren untersucht, wobei verschiedene Veröffentlichungen zu unterschiedlichen Ergebnissen kommen und sich kein überlegenes Verfahren abzeichnet. Ein Aspekt dieser Arbeit besteht daher in der Untersuchung der Notwendigkeit der Datenvorverarbeitung.

Die Anzahl der Trainingsbeispiele wird oft als ausschlaggebend für die Qualität der Ergebnisse von DL-Algorithmen dargestellt [54, S. 22f, 27, S. 415]. Veröffentlichungen aus dem Bereich der CNN-basierten Fehlerdiagnose im industriellen Umfeld zeigen, dass kleine Datenmengen mit nur wenigen hundert Trainingsbeispielen pro Klasse für hohe Klassifikationsgenauigkeiten ausreichen können [12, 133, 141, 154]. Ein weiterer Aspekt dieser Arbeit ist daher die Untersuchung des Einflusses der Datensatzgröße.

3 Vorgehen

In diesem Kapitel werden Untersuchungen zur Entwicklung des CNN-basierten FDI-Systems behandelt. Der verwendete Datensatz und dessen Entstehung sowie die Bewertungsmetrik werden vorgestellt (Unterkapitel 3.1 und 3.2). Zum Vergleich unterschiedlicher Einflüsse wird ein Referenzmodell entwickelt (Unterkapitel 3.3). Anhand dessen werden die Einflüsse der Datenkanalauswahl, Datenfusionsebene und Datenaufbereitung untersucht. Die Ausgestaltung der Untersuchungen der unterschiedlichen Aspekte wird in Unterkapitel 3.4 beschrieben. Die Signale werden in Vektor- sowie Matrixdarstellungen transformiert und als Eingangsdaten für den Input Layer verwendet, was unterschiedliche Netzarchitekturen zur Folge hat. Die Architektur wird daher ebenfalls variiert, wobei Implementierungsdetails Unterkapitel 3.5 zu entnehmen sind.

3.1 Erläuterung der Messdaten

Diese Arbeit umfasst Untersuchungen zur CNN-basierten Defektdiagnose von Fahrwerkskomponenten am Beispiel von Dämpfern. Merk [5] hat die Dämpferdefektdiagnose bereits mit einem datenbasierten Ansatz untersucht. Es wird an [5] angeknüpft und die Datensätze werden übernommen. Zur besseren Nachvollziehbarkeit wird die verwendete Messkette und das Verfahren zur Datenvorauswahl in diesem Unterkapitel kurz erörtert. Für eine ausführliche Beschreibung sei auf [5, S. 29–36] verwiesen.

3.1.1 Messkette

Als Versuchsträger bei der Datengewinnung dient das Lehrstuhl-eigene Forschungsfahrzeug BMW 650i Gran Coupé mit semi-aktivem Fahrwerk. Durch das semi-aktive Fahrwerk lässt sich die Dämpfercharakteristik variabel einstellen und das Fahrzeugverhalten kann bspw. für Fahrmodi wie „Komfort“ oder „Sport“ angepasst werden [3, S. 679]. Die Anpassung der Dämpferkennlinie erfolgt über die Bestromung des Dämpferventils [3, S. 300f]. Durch Veränderung des Ventilstroms wird das „Defekt-Verhalten“ eines Dämpfers nachgestellt. Zur Verbesserung der Generalisierungsfähigkeit wurden Messfahrten auf unterschiedlichen Untergründen mit unterschiedlichen Fahrzeuggeschwindigkeiten gefahren, nach [5, S. 32] analog zu [32]. Die Bestromung einzelner und aller Dämpfer wurde variiert, um Einzeldefekte (bspw. Dämpfer vorne links defekt) und Mehrfachdefekte (bspw. alle defekt) zu simulieren. Referenzmessungen für den „Intakt-Fall“ wurden ebenfalls durchgeführt (alle Dämpfer intakt bzw. passives Fahrwerk intakt).

Mit dem Zugriff auf die Bus-Kommunikation stehen Daten von ABS- und ESP-Steuergerät zur Verfügung (Abbildung A.1). Eine ausführliche Beschreibung kann [5, S. 30f] entnommen werden. Merk [5] benutzt zur Defektdiagnose insbesondere die Controller Area Network (CAN)-Signale aller Raddrehzahlen sowie errechnete Längs- und Querbeschleunigung am Fahrzeugschwer-

punkt. Die Abtastrate der verwendeten Signale liegt bei ca. 50 Hz. Durch Eigenheiten wie Buslastschwankungen der CAN-Kommunikation stehen die Signale nicht zu den exakt gleichen Zeitpunkten zur Verfügung, weshalb eine Abtastratenkonvertierung (Resampling) erfolgt. [5, S. 34]

3.1.2 Dämpferdefekte

In [5] wurden zwei Datensätze mit realen Fahrdaten zusammengestellt. Diese wurden mit verschieden ausgeprägten Dämpfercharakteristiken durch unterschiedliche Bestromung der Dämpferventile erzeugt (Tabelle 3.1). Für den K1|K2|FF-Datensatz wird das Dämpferventil im

Tabelle 3.1: Übersicht der Ventilbestromungen in den Datensätzen „K1|K2|FF“ und „DD“ nach [5, S. 33f]

Datensatz	Zustand (Label)	Vorderachse		Hinterachse	
		Zugstufe	Druckstufe	Zugstufe	Druckstufe
K1|K2|FF	Intakt	1,6 A	1,6 A	1,6 A	1,6 A
	Defekt	0,1 A	0,1 A	0,1 A	0,1 A
DD	Intakt	0,6 A	1,2 A	0,8 A	1,2 A
	Defekt	1,6 A	1,6 A	1,6 A	1,6 A

Intakt-Zustand maximal bestromt (1,6 A), während ein Defekt einem Strom von 0,1 A an Zug- und Druckstufe entspricht. Die maximale Bestromung von Zug- und Druckstufe resultiert in einem maximal geöffneten Ventil und damit einem besonders weichen Fahrwerk [5, S. 30]. Der Intakt-Zustand bildet damit das am weichsten einstellbare Fahrwerk ab und die Bestromung für den „Defekt“-Fall führt zu einem maximal steifen Dämpfer.
Beim DD-Datensatz werden für einen Einzeldefekt die Zug- und Druckstufe des entsprechenden hydraulischen Verstelldämpfers mit 1,6 A bestromt. Die beim DD-Datensatz gewählte Bestromung für den Intakt-Fall bildet laut Herstellerangabe einen passiven Dämpfer nach [5, S. 33]. Mit der Bestromung des DD-Datensatzes ist die gewählte Dämpfercharakteristik realistischer als beim K1|K2|FF-Datensatz, weil ein verschlissener oder defekter Dämpfer zu reduzierten Dämpferkräften und damit zu einer flacheren Kennlinie führt [4], was in einem weichen Dämpfer resultiert.

3.1.3 Datenvorauswahl

Beim K1|K2|FF-Datensatz besteht das Problem, dass dieser unausgeglichen ist (Abbildung 3.1). Die Klassifikationsgenauigkeit ist aufgrund der Unausgeglichenheit kein verlässliches Maß zur

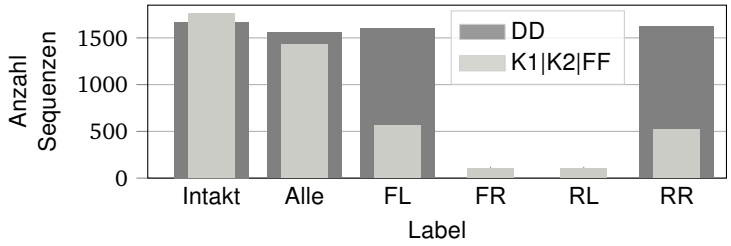


Abbildung 3.1: Histogramme der beiden Datensätze aus [5] mit einer Sequenzlänge von 128 Datenpunkten bei 50 Hz, d. h. Sequenzdauer 2,56 s. Die Label „FL“/„FR“ bzw. „RL“/„RR“ stehen für „Front Left/Right“ und „Rear Left/Right“ und geben die Position der Einzeldefekte an. „Alle“ entspricht dem Fall, dass alle vier Dämpfer defekt sind (Mehrfachdefekt) und bei „Intakt“ weisen alle Dämpfer den Intakt-Zustand auf.

Beurteilung der Klassifikationsgüte. Zusätzlich wurde der K1|K2|FF-Datensatz mit einer stark unterschiedlichen Dämpferbestromung erzeugt, die nicht dem realen Defekt entspricht. Der DD-Datensatz enthält nur zwei mögliche Einzeldefekte, er ist jedoch ausgeglichen (Abbildung 3.1). Weiterhin sind die beiden abgebildeten Einzeldefekte ausreichend, um die Eignung von CNN zur Diagnose zu demonstrieren. Aufgrund der vorangegangenen Überlegungen wird von den beiden Datensätzen aus [5] ausschließlich der DD-Datensatz für den verbleibenden Teil dieser Arbeit verwendet.

Als vorgelagerter Schritt der Vorverarbeitung („Preprocessing“) werden die Messungen zugeschnitten und durch sog. „Enable“-Bedingungen gefiltert. Eine Messdatei wird in möglichst viele Sequenzen mit einer festen Sequenzlänge von bspw. 128 Datenpunkten geteilt, wobei geprüft wird, ob die Bedingungen

- durchschnittliche Fahrzeuggeschwindigkeit über $30\,\frac{\mathrm{km}}{\mathrm{h}}$
- durchschnittlicher Betrag der Längsbeschleunigung unter $1\,\frac{\mathrm{m}}{\mathrm{s}^2}$
- durchschnittlicher Betrag der Querbeschleunigung unter $1\,\frac{\mathrm{m}}{\mathrm{s}^2}$

für jede Sequenz erfüllt sind. Dadurch wird der Datensatz – abgesehen von Grenzfällen – auf Konstantfahrten reduziert. Je nach gewünschter Länge der Sequenz variiert die Anzahl der Datenbeispiele, die aus einer Messung gewonnen werden können. Abbildung 3.1 wurde mit einer Sequenzlänge von 128 Datenpunkten generiert. Nach der Vorauswahl mit dem Enable-Fenster reduzieren sich die Messdaten des DD-Datensatz auf 6455 Datenbeispiele. Bei Verdopplung der Länge verringert sich die Anzahl auf 3300 Beispiele (51,1 %), bei erneuter Verdopplung (512 Datenpunkte) auf 1670 (25,9 %). Es fällt auf, dass die Gesamtdatenmenge mit Vergrößerung des Enable-Fensters zunimmt ($6455 \cdot 128 = 826.240$ während $1670 \cdot 512 = 855.040$ Datenpunkte). Das bedeutet, dass bei längeren Sequenzen mehr dynamische Grenzfälle in den Datensatz aufgenommen werden. Dies erschwert möglicherweise die Klassifikation der entsprechenden Sequenzen, ist jedoch hinsichtlich Praxistauglichkeit realistischer.

3.2 Bewertungsmetriken

Zur Bewertung der Qualität des FDI-Systems können verschiedene Metriken verwendet werden. Gängige Methoden zur Evaluation von ML-Systemen sind Korrektklassifizierungsrate (KKR) bzw. Klassifikationsgenauigkeit („accuracy“), Relevanz („precision“) und Sensitivität („sensitivity“ / “recall“) [54, S. 82–90, 60, S. 1183]. Daneben existieren auch kombinierte Maße wie das F1-Maß [53, S. 220]. Für die Bestimmung dieser Kennzahlen werden die Vorhersagen des Modells mit den tatsächlichen Klassen aus den beschrifteten Daten kombiniert und in Kategorien eingeteilt, die zusammen die Wahrheits- bzw. Konfusionsmatrix bilden (Abbildung 3.2). Diese ist eine quadratische Matrix mit der Unterteilung Richtig Positive (RP), Falsch Positive (FP), Falsch Negative (FN) und Richtig Negative (RN). Die Konfusionsmatrix stellt die Leistung eines Lernalgorithmus’ übersichtlich dar [53, S. 220]. Zum besseren Verständnis folgen Beispiele für die Kategorien:

- RP: Ein Dämpfer ist intakt (positiv) und das System hat dies richtig kategorisiert (Treffer / Intakt-Zustand erkannt).
- FN: Ein Dämpfer ist defekt (negativ) und das System hat ihn fälschlicherweise als intakt kategorisiert (Fehler verpasst).

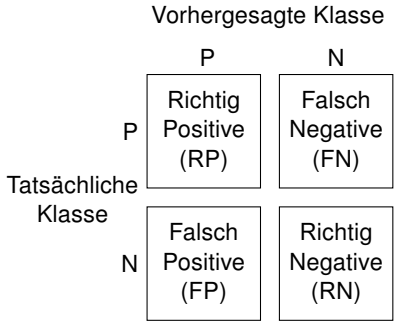


Abbildung 3.2: Wahrheits- oder Konfusionsmatrix zur Darstellung der Leistung eines ML-Algorithmus' nach [53, S. 220]

- FP: Ein Dämpfer ist intakt (positiv) und das System hat ihn fälschlicherweise als defekt kategorisiert (falscher Alarm).
- RN: Ein Dämpfer ist defekt (negativ) und das System hat ihn richtig als defekt kategorisiert (Fehler erkannt).

Die Vorhersagen gehen unterschiedlich in verschiedene Metriken ein. Der Klassifikationsfehler (Fehlerquote (FQ)) und die Klassifikationsgenauigkeit (KKR) liefern eine generelle Einschätzung für die Missklassifikation und werden nach

$$\mathrm{FQ} = \frac{\mathrm{FP} + \mathrm{FN}}{\mathrm{FP} + \mathrm{FN} + \mathrm{RP} + \mathrm{RN}} \tag{3.1}$$

bzw.

$$\mathrm{KKR} = \frac{\mathrm{RP} + \mathrm{RN}}{\mathrm{FP} + \mathrm{FN} + \mathrm{RP} + \mathrm{RN}} = 1 - \mathrm{FQ} \tag{3.2}$$

berechnet [53, S. 222]. Für unausgeglichene Datensätze werden die Richtig- bzw. Falsch-Positiv-Rate empfohlen, die stark mit Relevanz und Sensitivität korreliert sind [53, S. 223]. Die Relevanz berechnet sich aus den richtig Positiven (RP) im Verhältnis zu richtigen und falschen Positiven (RP + FP), während die Sensitivität im Nenner richtig Positive und falsche Negative summiert (RP + FN). Aus der Konfusionsmatrix lassen sich alle genannten Metriken bestimmen.
Je nach Auslegungsziel des Systems kann es sinnvoll sein, das Modell hinsichtlich einer bestimmten Metrik zu optimieren. Aus Gründen der Vergleichbarkeit zu [5] und weil ein ausgeglichener Datensatz vorliegt, wird in dieser Arbeit die Korrektklassifizierungsrate (KKR), Klassifikationsgenauigkeit oder nur „Genauigkeit", als Bewertungsmaß verwendet.

3.3 Entwicklung eines Baseline-Modells

Für die Untersuchungspakete im nächsten Unterkapitel wird ein Referenzmodell („Baseline"-Modell) entwickelt, anhand dessen der Einfluss der jeweils betrachteten Veränderung gemessen werden kann. Gleichzeitig kann anhand des Baseline-Modells festgestellt werden, ob ein CNN prinzipiell zur Dämpferdefektdiagnose geeignet ist. Wird in keiner Konstellation von Hyperparametern eine signifikante Klassifikationsgenauigkeit erreicht, ist davon auszugehen, dass CNN die zugrundeliegenden Zusammenhänge nicht abbilden können.

3.3.1 Standardparameter

Damit Datenkanalauswahl, Preprocessing, Architektur etc. eingehend untersucht werden können, ist die Festlegung verschiedener Hyperparameter notwendig. Es existieren Ansätze zur automatischen Parameteroptimierung von CNN, die jedoch zeitintensiv sind und möglicherweise nur in ein lokales Optimum konvergieren [148, S. 5f]. Das initiale Modell basiert daher auf generellen Designprinzipien und es werden nach der Überprüfung der Kapazität mehrere Konfigurationen getestet. Zur Untersuchung der Modell-Komplexität werden (soweit möglich) Standardparameter verwendet (Tabelle 3.2).

Tabelle 3.2: Übersicht der Parameter für das Baseline-Modell

Parameter	**Wert**
Preprocessing	linearer Detrend
Anteil Testdaten in %	20
Anteil Validierung in %	16
(Initiale) Lernrate ϵ	0,001
Dropout rate p_r	0,5
Weight decay λ	0,01
Mini-batch size b	128
Maximale Epochenanzahl	450
Early Stopping Geduld	35
Early Stopping Delta	0,001
Initialisierung Bias-Terme	0,01
Initialisierung Gewichte	He [155]

Die Architektur muss für eine gute Generalisierungsfähigkeit ausreichend komplex sein (Abschnitt 2.1.5). Der Philosophie von „Occam's Razor" folgend, sollte das Modell gleichzeitig so einfach wie möglich gehalten werden [27, S. 111]. Es wird eine Modell-Einstellung mit ausreichend hoher Komplexität angestrebt, um den DD-Datensatz abzubilden. Ob das Modell eine ausreichende Kapazität besitzt, kann geprüft werden, indem die Lernkurve hinsichtlich Overfitting untersucht wird. Es empfiehlt sich, bei der ersten Untersuchung die Regularisierung einzuschränken, um die Ergebnisse nicht zu stark zu verzerren. Kann ein starkes Overfitting anhand des Trainingsdatensatzes festgestellt werden, ist das Modell grundsätzlich in der Lage die zugrundeliegenden Zusammenhänge zu erlernen. Dem Overfitting kann anschließend durch Einstellung der Regularisierungsstärke entgegengewirkt werden, um die Generalisierungsfähigkeit zu verbessern. Bei der Architekturwahl gilt es eine Balance zwischen Overfitting und Modellkomplexität zu finden [36, S. 30].

Als Baseline wird ein flaches CNN mit einem Convolutional Layer, einem Max-Pooling-Layer und anschließendem FC-Layer mit Dropout verwendet (Tabelle 3.3). Ein Max-Pooling Layer

Tabelle 3.3: Übersicht der ersten Baseline-Architektur. In allen Layern mit Aktivierungsfunktionen (außer Output) werden ReLU-Funktionen eingesetzt.

Layer	**Output Dimension**	k @ $f_\mathrm{h} \times f_\mathrm{d}$ / s	**Padding**
Input ($4 \times n_{(\cdot)}, 2 \times a_{(\cdot)}$)	256×6	-	-
Convolutional	256×32	32 @ 5×6 / 1	same
Max-Pool	128×32	2×1 / 2	valid
Fully Connected (FC)	128	-	-
	Dropout		
Output (FC)	4	-	-

schafft Robustheit gegenüber kleinen Variationen der erlernten Merkmale, reduziert jedoch gleichzeitig die Anzahl der Parameter [60, 86]. Die Filtermaske und die Anzahl der Neuronen im FC-Layer werden analog zu grundsätzlichen Empfehlungen [36, S. 97] zur Parameterreduktion klein gewählt. Es ergibt sich eine Struktur mit ca. 525.000 Parametern. Auf Details zum Training wie Optimierer, Kostenfunktion etc. wird in Unterkapitel 3.5 eingegangen.

3.3.2 Grundkapazität und Regularisierung

Die generelle Eignung (Kapazität) des Modells für die Dämpferdefektdiagnose wird durch Untersuchung der Lernkurven bestimmt (Abbildung 3.3). Deren Verlauf ähnelt im Idealfall Abbildung 2.5. Das Baseline-Modell wird mit unterschiedlich starker L2-Regularisierung trainiert,

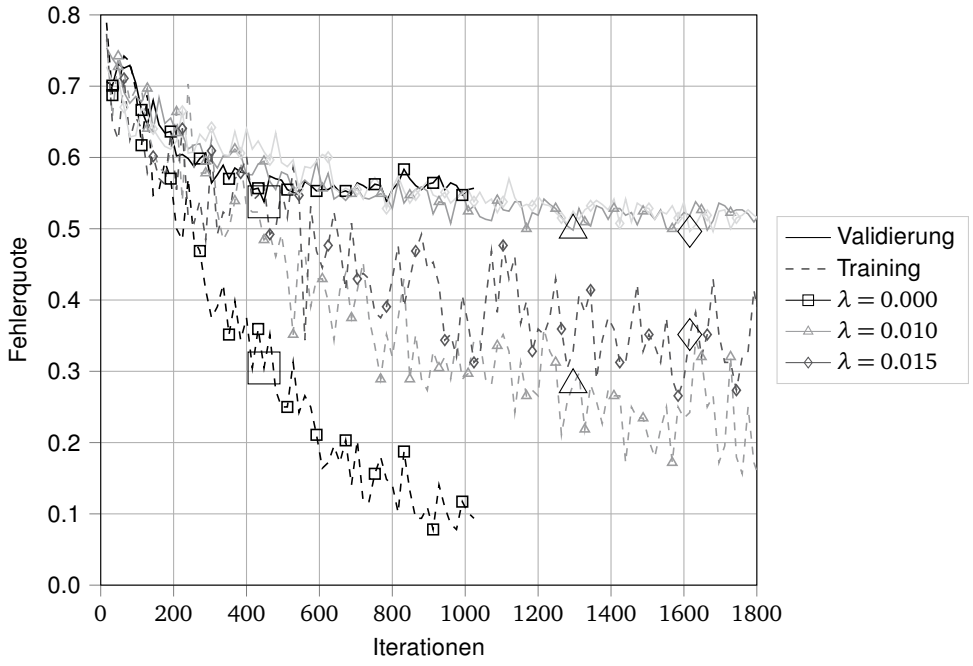


Abbildung 3.3: Lernkurven des Baseline-Modells bei unterschiedlich starker L2-Regularisierung λ. Große Marker markieren den Early Stopping Zeitpunkt. Die erreichten KKR-Werte mit Testdaten betragen für zunehmendes λ $48,36\,\%$, $58,59\,\%$ und $57,42\,\%$.

wobei die Architektur aus Tabelle 3.3 und die restlichen Parameter aus Tabelle 3.2 unverändert bleiben. Dadurch kommen Dropout mit $p = 0.5$ und Early Stopping als weitere Regularisierung zum Einsatz. Letztere führt dazu, dass die Modelle für unterschiedlich viele Epochen trainiert werden.

Trotz Dropout und Early Stopping ist eine starke Tendenz zur Überanpassung an die Trainingsdaten bei $\lambda = 0$ zu erkennen: der Klassifikationsfehler konvergiert schnell Richtung Null, während der Validierungsfehler vorzeitig auf einem Niveau stagniert. Das Baseline-Modell besitzt demnach eine ausreichende Kapazität für den DD-Datensatz. Mit zunehmender L2-Regularisierungsstärke vergrößert sich die FQ beim Trainingsdatensatz; die Graphen für Training und Validierung nähern sich an. Außerdem verschiebt sich die Dauer des Trainings nach oben, was an den Early Stopping Zeitpunkten erkennbar ist.
Die Lernkurven für den Trainingsdatensatz zeigen ein regressives Konvergenzverhalten: Die Größe der Mini-Batches und initiale Lernrate sind damit gut eingestellt. Kleinere Mini-batches hätten stärker verrauschte Lernkurven zur Folge, die zusätzlich ebenso wie eine zu klein gewählte Lernrate die Trainings-Geschwindigkeit negativ beeinflussen würde. Der Erwartungswert

des Klassifikationsproblems mit vier Klassen liegt bei 25 %. Mit einer (vorläufig) maximalen Testgenauigkeit von fast 60 % scheint ein CNN grundsätzlich für die Dämpferdefektdiagnose geeignet zu sein.

3.3.3 Abstimmung der Architektur

Um die erste Modell-Parameterwahl des Baselinemodells zu überprüfen, wird eine Rastersuche über die Modellparameter durchgeführt. Die Hyperparameter aus Tabelle 3.2 bleiben für die Untersuchung unverändert. Die Veröffentlichung von Lee et al. [149] zeigt, dass die Anzahl der Filter sorgfältig untersucht werden sollte. In deren Anwendungsfall führt eine zu geringe Filteranzahl (< 3) zu signifikant geringerer Genauigkeit, während der Einsatz von bis zu 50 Filtern kaum zusätzliche Genauigkeit gegenüber fünf Filtern einbringt [149, S. 196–197]. Es wird daher ein ausreichend großes Spektrum gewählt. Zusätzlich kann die Filtermaske beliebig groß gewählt werden, um Informationen zu erfassen, die weiter voneinander entfernt liegen [148, S. 5]. Dadurch steigt jedoch die Rechenzeit. Die Rastersuche dient der Feinabstimmung des Modells

Tabelle 3.4: Parameter der Rastersuche für die Feinabstimmung der Baseline-Architektur.

Parameter	**Varianten**
Größe der Convolutional Filtermaske e	3, 5, 9, 11, 17
Anzahl der Convolutional Kernel k	8, 16, 32, 64, 128
Anzahl der Neuronen im FC-Layer	32, 64, 128, 256

und zur Einschätzung der unterschiedlichen Parametereinflüsse. Zusätzlich kann festgestellt werden, wie gut die initiale Wahl der Architekturparameter ist.

Es zeigt sich, dass die durchschnittlichen KKR mit zunehmender Filteranzahl k bis $k = 64$ unabhängig von der Filtergröße e steigen (Abbildung 3.4 und A.2). Die Fälle mit vielen Filtern und wenigen Neuronen im FC-Layer, z. B. $k = 64$ und 32 Neuronen oder $k = 128$ und 32 bzw. 64 Neuronen bilden mit einer auffällig geringen KKR ($< 40\,\%$) eine Ausnahme dieses Trends. Dies ist darauf zurückzuführen, dass das Training in zwei der fünf Kreuzvalidierungs-Läufe nicht konvergiert. Über die Gründe können nur Vermutungen angestellt werden: Da bei jedem der fünf Kreuzvalidierungs-Durchläufe das Netz neu (zufällig) initialisiert wird, kann eine ungünstige Verteilung der initialen Gewichte nicht ausgeschlossen werden, die potenziell zu Divergenz führt. Zudem ist es möglich, dass bei der Aufteilung des Datensatzes in fünf Teilmengen eine davon stark verrauschte Beispiele oder welche mit hoher Dynamik enthält, wodurch das Training erschwert wird. Ein weiterer Grund kann die starke Regularisierung $\lambda = 0.01$ sein, die auf die Convolutional Kernel und die Gewichte im FC-Layer gleichermaßen angewendet wird. Bei weniger Neuronen fallen die Kernel stärker ins Gewicht.
Wird in weiteren Untersuchungen das Netz mit reduzierter Regularisierung und Lernrate ($\lambda = 0.003, \epsilon = 0.0003$) trainiert, ergeben sich mehr konvergierende Durchläufe, was die letzte Hypothese unterstützt. Sporadisch zeigt sich jedoch nach wie vor divergierendes Verhalten für wenige Neuronen und viele Filter, z. B. $e \in \{3, 5\}$, $k = 128$ bei 32 Neuronen im FC-Layer. Es wird daher davon ausgegangen, dass die Konstellation vieler Filter auf wenige Neuronen generell das Training (Optimierungsproblem) erschwert, weshalb diese Kombinationen für weitere Untersuchungen ausgeschlossen werden.

Für $k = 128$ liegen die meisten Ergebnisse unterhalb denen für $k = 64$, was bedeutet, dass mit einer weiteren Erhöhung keine Verbesserung zu erwarten ist. Weiterhin zeigt sich, dass mit den gewählten Hyperparametern eine Filterhöhe zwischen 5 und 11 Datenpunkten zu den höchsten

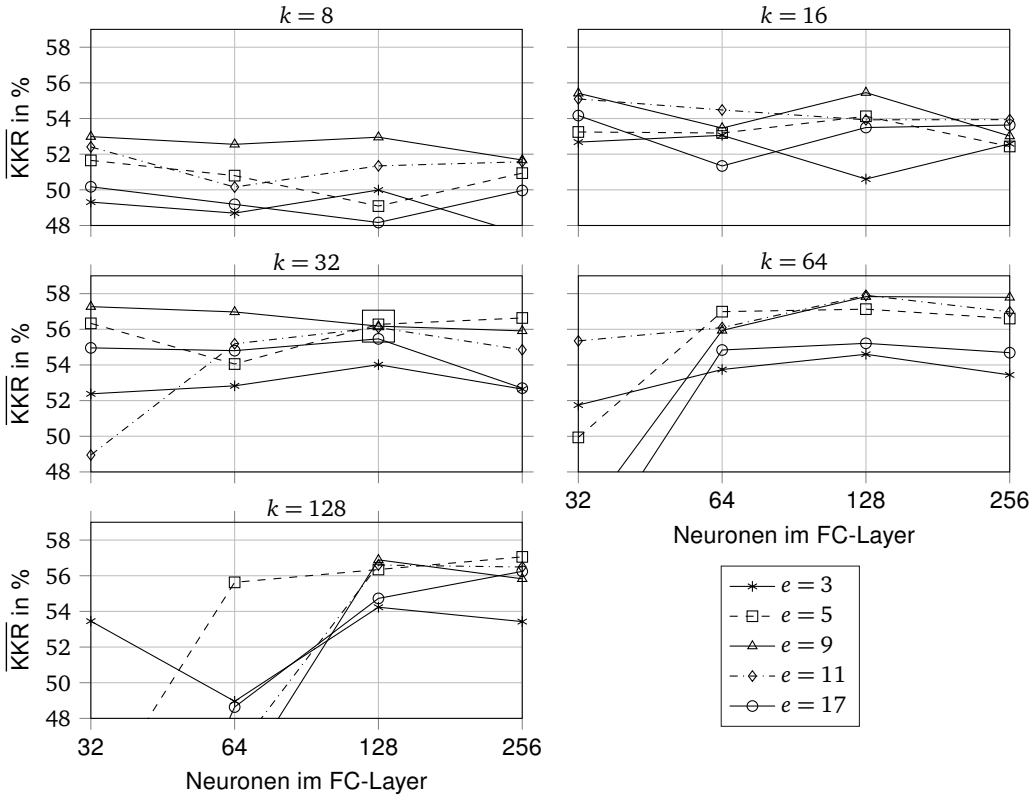


Abbildung 3.4: Fünffach kreuzvalidierte Ergebnisse der Rastersuche für die Filteranzahl bei variierender Filteranzahl im Baseline-Modell. Der große Marker markiert die Erstkonfiguration des Modells aus Tabelle 3.3.

Genauigkeiten führt (Abbildung 3.5). Zu große Filtermasken ziehen eine Verschlechterung der KKR nach sich, was den Trend kleiner Masken aus der Bildverarbeitung bestätigt. Inwiefern diese Aussage bei einer anderen Konstellation aus Hyperparametern und Daten gilt, ist in einer weiteren Arbeit zu untersuchen.

Die zu Beginn gewählte Konfiguration mit $k = 32, e = 5$ und 128 Neuronen im FC-Layer, durch größere Marker gekennzeichnet, liegt mit durchschnittlich $56,18\,\%$ unter den besten $20\,\%$ aller 100 Parameterkombinationen (Abbildung A.3). Es gilt zu bedenken, dass die Ergebnisse durch zufällige Initialisierungen und eine zufällige Reihenfolge der Trainingsdaten in den Mini-Batches einer gewissen Streuung unterliegen. Die besten 45 Ergebnisse liegen im Intervall $56 \pm 2\,\%$, wobei jedes Ergebnis aus den fünf Kreuzvalidierungs-Durchläufen gemittelt ist. Das zeigt, dass das System generell robust gegenüber Änderungen in den Modellparametern ist.
Die Ergebnisse bestätigen, dass zu Beginn eine plausible Wahl getroffen wurde, durch eine Anpassung der Parameter jedoch verbesserte Ergebnisse erreicht werden können. Die Anzahl der Filter wird für das finale Baseline-Modell auf $k = 64$ erhöht. Für weitere Untersuchungen wird die Anzahl der Neuronen im FC-Layer entsprechend oberhalb von 64 gewählt, um Divergenz-Fälle zu vermeiden. Bei gleichbleibender Filtergröße ($e = 5$) und Neuronenanzahl (128) würde dies eine Verdopplung der Parameter auf ca. 1,05 Millionen bedeuten. Bei 256 Neuronen sind die Ergebnisse vergleichbar zu denen mit 128, die Anzahl der Parameter ist jedoch doppelt so hoch. Daher wird die Neuronenanzahl von 128 beibehalten. Die Größe der Filtermaske wird auf $e = 9$ erhöht, um mehr Datenpunkte zu berücksichtigen und tiefere Frequenzen in den Daten erfassen zu können. Durch die „sparse connectivity“ des Convolutional Layers wird die Anzahl der Parameter nur um ca. 1.500 erhöht.

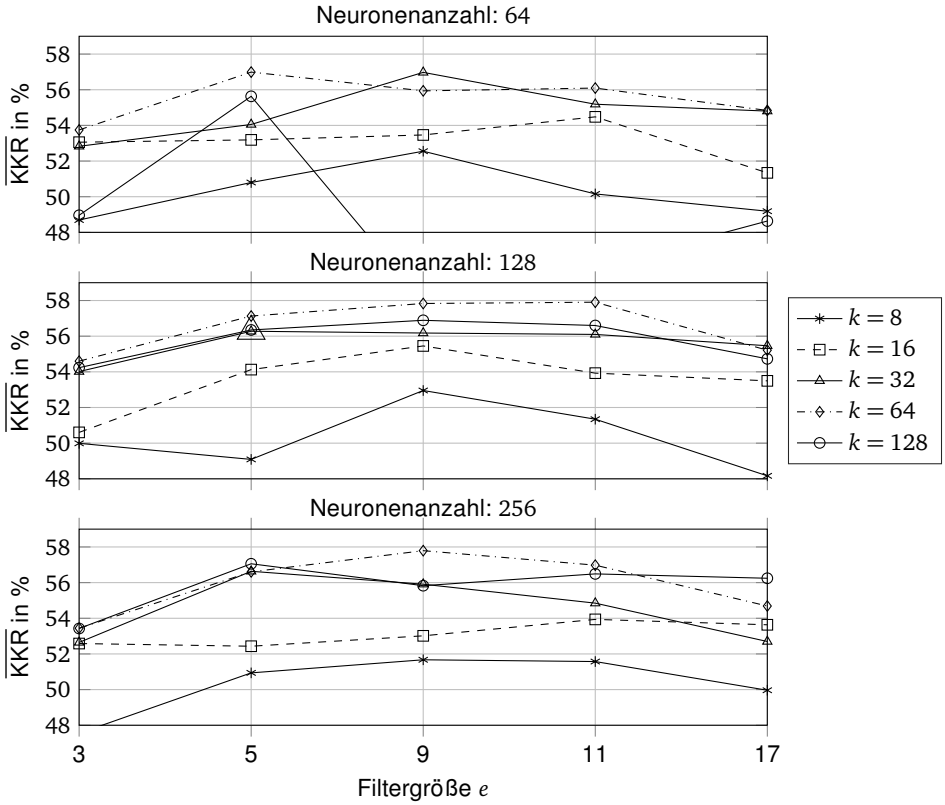


Abbildung 3.5: Fünffach kreuzvalidierte Ergebnisse der Rastersuche für die Filtergröße bei variierender Filteranzahl im Baseline-Modell. Der große Marker markiert die Erstkonfiguration des Modells aus Tabelle 3.3.

3.4 Zu untersuchende Aspekte

In den letzten Jahrzehnten wurden signifikante Verbesserungen meist durch neue Architekturen bzw. Bausteine erreicht und nur in geringem Maße durch aufwändigere bzw. bessere Algorithmen für Optimierer, Initialisierungsstrategien und andere Aktivierungsfunktionen [36, S. 80, 156]. Im ersten Schritt werden daher übergeordnete Untersuchungen wie Auswirkungen der Architektur, Datenaufbereitung etc. angestellt, während algorithmische Einflüsse in einem Optimierungsschritt für ein ausgewähltes Netz später untersucht werden (Unterkapitel 4.2). Wie in Unterkapitel 2.6 hergeleitet, werden folgende Hauptuntersuchungsaspekte festgelegt:

1. Auswahl der Datenkanäle: Welche Sensorsignale sollten als Eingangsdaten verwendet werden?
2. Datenfusionsebene: Ist es besser die Sensorsignale aneinander zu hängen oder durch unterschiedliche Faltungsoperationen (Convolutions) miteinander zu fusionieren?
3. Preprocessing: Sind Zeit-, Frequenz- oder Zeit-Frequenz-basierte Methoden am besten geeignet? Ist ein End-to-End-System umsetzbar?
4. Netzwerkarchitektur: Lassen sich „gute“ Netze aus anderen Anwendungen auf die Dämpferdefektdiagnose übertragen? Wie komplex sollte die Netztopologie sein bzw. wie beeinflusst diese die Ergebnisse?
5. Benötigte Datenmenge: Wieviele Trainingsbeispiele werden für ein gefordertes

Niveau der Klassifikationsgenauigkeit, z. B. 80 %, benötigt?

Durch die Vielfalt an Kombinationsmöglichkeiten entsteht ein großer Untersuchungsraum (Zustandsraumexplosion), der in dieser Arbeit zeit- und umfangsbedingt nicht vollständig abgedeckt werden kann. Mit der vorangegangenen Priorisierung wird daher eine Reihenfolge festgelegt, in der die Untersuchungen durchgeführt werden, um den Zustandsraum systematisch nach einer (lokal) optimalen Lösung abzusuchen. Der Ablauf folgt einem Top-Down-Prinzip ausgehend von den Daten: Es ist zu entscheiden, welche Datenkanäle überhaupt für das FDI-System zu verwenden sind. Anschließend kann die Fusionsebene untersucht werden, um festzustellen, welche Art der Kombination der Daten zu den besten Ergebnissen führt. Danach werden unterschiedliche Datenrepräsentationen in einer Vergleichsarchitektur untersucht. Diese wird im anschließenden Untersuchungspaket ersetzt. Der letzte Aspekt dient zur Abschätzung der Menge benötigter (neuer) Daten in der herausgearbeiteten Konstellation, um eine weitere Verbesserung der Klassifikationsergebnisse zu erzielen.
Es wird implizit angenommen, dass sich bspw. Datenkanalauswahl oder Preprocessing für unterschiedliche Netzwerkarchitekturen identisch auswirken. Die Gültigkeit dieser Annahmen ist in einer weiterführenden Arbeit zu untersuchen.

3.4.1 Auswahl der Datenkanäle

Nach [5, S. 28] werden bei den Messfahrten u.a. die ABS-Signale

- Raddrehzahl vorne links (n_{FL}), vorne rechts (n_{FR}), hinten links (n_{RL}) und hinten rechts (n_{RR}),
- errechnete Längs- und Querbeschleunigung (a_{X} bzw. a_{Y}) am Fahrzeugschwerpunkt,
- Gierrate ($\dot{\psi}$), gemessen durch einen Gierratensensor, und
- Lenkwinkel (δ), bezogen auf die Vorderachse,

aufgezeichnet. Merk [5] vernachlässigt für das SVM-basierte FDI-System Gierrate und Lenkwinkel, weil eine Sensitivitätsanalyse an einem Zweispurmodell geringe Auswirkungen von Dämpferdefekten auf die beiden Signale gezeigt hat. Durch Linearisierungen und zugrundeliegende Annahmen ist diese Aussage jedoch nicht allgemeingültig und in der Praxis nicht bestätigt. Deshalb werden in der Ausgangssituation alle acht Sensorsignale berücksichtigt.

In [148] brachte das Feature Learning gegenüber der manuellen Merkmalsextraktion keine Verbesserung bei Verwendung eines einzelnen Sensorsignals. Erst die Kombination mehrerer Signale führte zu einer Verbesserung gegenüber dem Feature Engineering. Weiterhin stellen die Autoren fest, dass je nach gewählten Signalen unterschiedlich gute Ergebnisse erreicht werden. Die Kombination mehrerer Datenkanäle kann in einer ungünstigen Konstellation zu einer Verschlechterung der Ergebnisse führen [148]. Daher werden verschiedene Kombinationen der verfügbaren Signale untersucht (Tabelle 3.5). Es wird davon ausgegangen, dass die Erkennung eines Defekts auf Basis eines einzelnen Sensorsignals nicht mit einer signifikanten Genauigkeit möglich ist. Um dies zu überprüfen, werden die Varianten 0, 1, 5, 6, 8 und 9 festgelegt. Der DD-Datensatz enthält nur die zwei Einzeldefekte „FL“ und „RR“. Die Varianten 0 und 1 lassen die Untersuchung zu, ob die Erkennung eines Einzeldefekts durch die Verwendung der Raddrehzahl an der entsprechenden Position beeinflusst wird. Zudem werden die Raddrehzahlen achsweise getrennt und in ihrer Gesamtheit betrachtet (Varianten 2 – 4).
Der Einfluss von Längs- und Querbeschleunigung wird ebenso wie der von Gierrate und

Tabelle 3.5: Zu untersuchende Variationen bei der Datenkanal-Auswahl

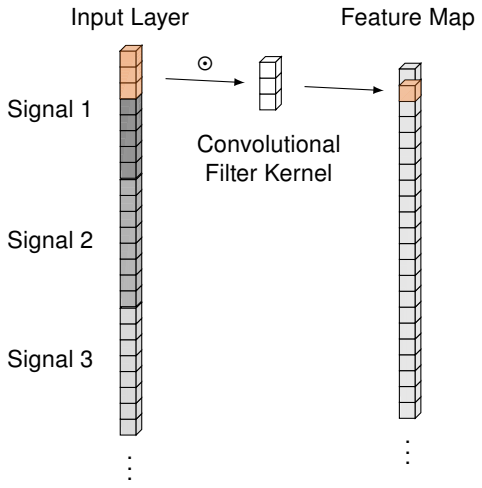
Variante	n_{FL}	n_{FR}	n_{RL}	n_{RR}	a_{X}	a_{Y}	$\dot{\psi}$	δ
0	x							
1		x						
2	x	x						
3			x	x				
4	x	x	x	x				
5					x			
6						x		
7					x	x		
8							x	
9								x
10							x	x
11					x	x	x	x
12	x	x	x	x	x	x		
13	x	x	x	x			x	x
14	x	x	x	x		x		x
15	x	x	x	x		x	x	
16	x	x	x	x	x	x	x	
17	x	x	x	x	x	x		x
18	x	x	x	x	x	x	x	x

Lenkwinkel getrennt und kombiniert untersucht (Varianten 7, 10, 11). Durch den Vergleich von zwei Sensorsignalen oder -gruppen einzeln und kombiniert kann auf den Einfluss bzw. Beitrag zum Ergebnis rückgeschlossen werden.
Es wird die gleiche Datenmenge wie bei [5] verwendet (Variante 12) und jeweils Gierrate und Lenkwinkel hinzugefügt (Varianten 16 und 17). Weil die Gierrate vom Lenkwinkel abhängt und die Längsbeschleunigung stark mit den Raddrehzahlen korreliert ist, wird Variante 14 als Konstellation mit geringer Redundanz untersucht. Der letzte Untersuchungsfall beinhaltet die Maximalausprägung und damit die Maximaldatenmenge.

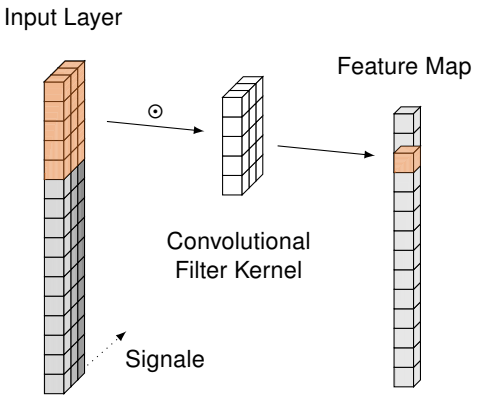
Zur Untersuchung wird ausschließlich das (1-D-)Baseline-Modell verwendet. Die Daten werden dem System jeweils im Zeit- und Frequenzbereich übergeben. Es wird angenommen, dass sich dadurch zeigt, welche Datenkanäle dem System die nützlichsten Informationen für eine Defektisolation liefern. Die Sequenzlänge wird zu 256 Datenpunkten, d. h. $5,12\,\mathrm{s}$ gewählt und ist damit vergleichbar zu [5] ($5,0\,\mathrm{s}$). Ebenso wird wie in [5] aus jedem Trainingsbeispiel der Mittelwert mit einer linearen Funktion entfernt, um die Korrelation zur Fahrzeuggeschwindigkeit insbesondere für die Raddrehzahlen zu entfernen. Die Festlegung der im weiteren Verlauf zu verwendenden Datenkanäle erfolgt durch eine gewichtete Bewertung der Ergebnisse aus Zeit- und Frequenzbereich. Die Auswertung erfolgt im Ergebnisteil.

3.4.2 Fusionsebene der Daten

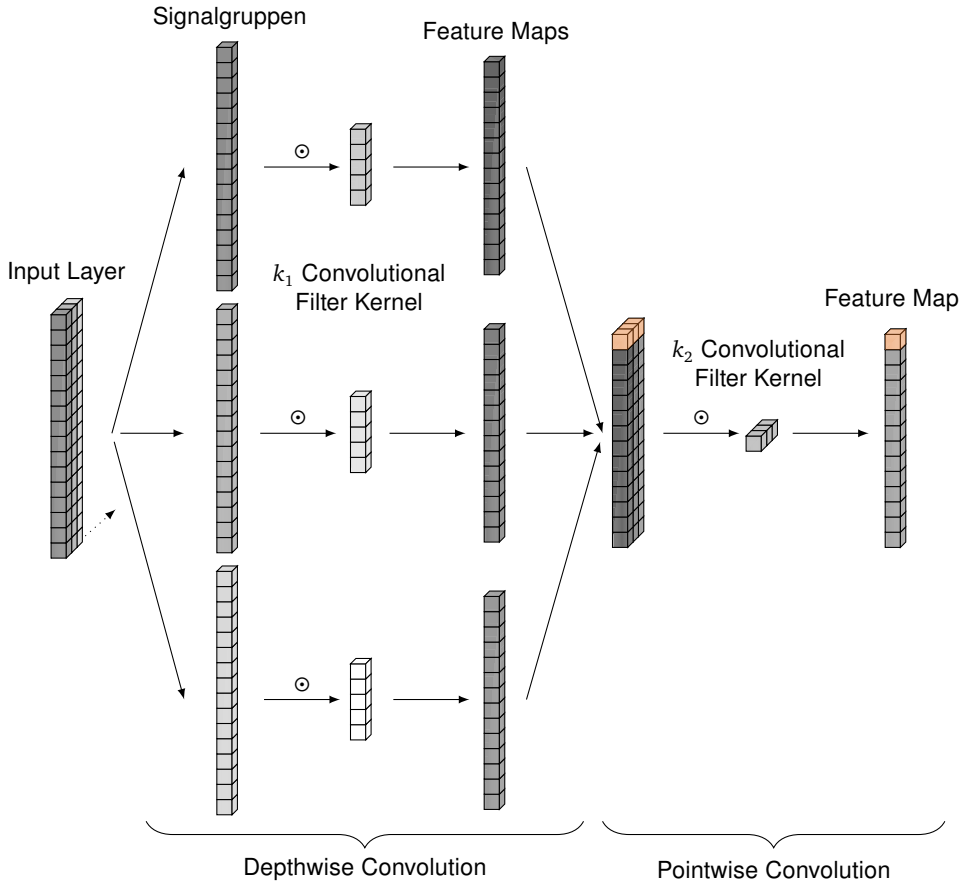
Jing et al. [148] diagnostizieren ein Planetengetriebe mithilfe mehrerer Datenkanäle anhand von Rohdaten. Diese Konstellation ist für das zu entwickelnde FDI-System ebenfalls wünschenswert. Die Autoren untersuchen die Auswirkung der Fusionsebene auf die Klassifikationsergebnisse und stellen für ihre Anwendung fest, dass die Fusion auf Rohdatenebene zum besten Ergebnis führt. Zur Verifizierung dieses Ergebnisses wird der Aspekt ebenfalls untersucht. Die Fusion der Daten lässt sich mit CNN – inbesondere für Zeitsignale (1-D-Daten) – auf unterschiedliche Arten und Weisen durchführen (Abbildung 3.6):



(a) **Option A:** Fusion auf Rohdatenebene durch Aneinanderreihung der einzelnen Signale



(b) **Option B:** Fusion auf Merkmalsebene durch eine gewöhnliche Faltung



(c) **Option C:** Fusion auf Merkmalsebene durch eine separierbare Faltung („separable convolution“), die durch Verkettung einer „Depthwise Convolution“ mit einer „Pointwise Convolution“ entsteht

Abbildung 3.6: Visualisierung der Datenfusionsebenen anhand von 1-D-Daten: Es sind jeweils nur drei Signale und ein Filterkernel dargestellt. Eine gewöhnliche Faltungsoperation wird mit ⊙ symbolisiert.

Option A Durch eine simple Aneinanderreihung aller Datenkanäle entsteht eine lange Abfolge von Datenpunkten, auf die anschließend gewöhnliche Faltungsoperationen ($\odot$) angewendet werden (Abbildung 3.6(a)). Ein Filterkernel wird über alle Signale nacheinander bewegt und so trainiert, dass er in allen Kanälen die gleichen Merkmale extrahiert. Daraus resultiert je nach Kombination aus Stride und Filtergröße eine große Anzahl von Punkten in einer Feature Map. Die Auswirkung der Signalvermischung an den Anknüpfungspunkten zweier Signale auf den Filterkernel ist nicht bekannt. Bei einem ausreichend großen Verhältnis von Sequenzlänge eines Datenkanals zur Filtergröße wird jedoch keine signifikante Beeinflussung erwartet.

Option B Für die Fusion auf Merkmalsebene existieren mehrere Möglichkeiten, wovon zwei aufgezeigt sind (Abbildung 3.6(b) und 3.6(c)). Durch die Anordnung von c Datenkanälen in der Tiefendimension lassen sich Filterkernel mit einer Tiefe $d_{\mathrm{in}} = c$ verwenden, die Merkmale über alle Kanäle gleichzeitig extrahieren (Abbildung 3.6(b)). Dies entspricht einer gewöhnlichen Faltungsoperation (Convolution). Ein Kernel erlernt dadurch kanalübergreifende Merkmale, die in einer im Vergleich zu Option A wesentlich kleineren Feature Map zusammengefasst werden.

Option C Die in dieser Anwendung stark unterschiedlich ausgeprägten Signale legen nahe, dass Merkmale möglicherweise einzeln aus den Datenkanälen extrahiert und anschließend kombiniert werden sollten. Dadurch können bspw. in den Raddrehzahlen vorhandene Merkmale unabhängig von den anderen Kanälen extrahiert werden. Eine Möglichkeit stellen „Depthwise Separable Convolutions“ [157] dar (Abbildung 3.6(c)). Dabei wird ein Eingangsvolumen $\boldsymbol{I} \in \mathbb{R}^{h_{\mathrm{in}} \times w_{\mathrm{in}} \times d_{\mathrm{in}}}$ in d_{in} einzelne Kanäle mit Dimension $h_{\mathrm{in}} \times w_{\mathrm{in}} \times 1$ aufgeteilt und jeweils mit einem Convolutional Filter $\boldsymbol{F} \in \mathbb{R}^{\mathrm{f_h} \times \mathrm{f_w} \times 1}$ verarbeitet. Es entstehen $k = k_1 d_{\mathrm{in}}$ Feature Maps, d. h. für jeden Kanal gibt es k_1 Filter, die für alle Kanäle in der Tiefe aneinander gehangen werden und anschließend mit k_2 $(1 \times 1 \times d_{\mathrm{in}})$-Convolutional Filtern vermischt werden.
In der herkömmlichen Implementierung wird das Eingangsvolumen in alle seine Einzelkanäle zerlegt und für jeden Kanal k_1 Kernel trainiert. Stattdessen kann eine Gruppierung von Signalen erwünscht sein, bei der bspw. die Raddrehzahlen in einer Gruppe zusammengefasst und mit einer gewöhnlichen Convolution verarbeitet werden, während die verbleibenden Signale jeweils einzeln mit einem Convolutional Layer transformiert werden. Alle derartig generierten Feature Maps werden anschließend durch die Pointwise Convolution ($(1 \times 1 \times d_{\mathrm{in}})$-Convolution) vermischt. Es sind verschiedene Gruppierungen der Datenkanäle denkbar, wobei nur noch Kombinationen derjenigen Signale getestet werden, die nach der Untersuchung aus Abschnitt 3.4.1 den höchsten Informationsgehalt bieten. Mögliche Kombinationen aus allen Signalen sind in Tabelle 3.6 dargestellt.

Tabelle 3.6: Auswahl möglicher Signalgruppierungen bei der gruppierten Depthwise Separable Convolution: Gruppen sind gleich eingefärbt und weiße Felder gelten als Einzelgruppen.

Variante	n_{FL}	n_{FR}	n_{RL}	n_{RR}	a_{X}	a_{Y}	$\dot{\psi}$	δ
0								
1								
2								
3								
4								
5								

Insbesondere die Raddrehzahlen können gemeinsam, achsweise getrennt oder einzeln betrachtet werden. Von einer Aufteilung der Raddrehzahlen nach der Fahrzeugseite oder einer

Gruppierung diagonal gegenüberliegender Raddrehzahlen wird keine Verbesserung erwartet. Dies ist darauf zurückzuführen, dass die Dämpferbestromung und damit die Defektcharakteristik achsweise für jeweils zwei Räder gleich ist, was das Auffinden gemeinsamer Merkmale zwischen den beiden Signalen erschwert.
Möglich ist außerdem die Trennung der Signale nach Längs- und Querdynamik. Bei anderen Varianten kann berücksichtigen werden, dass Gierrate und Lenkwinkel miteinander korreliert sind und dass Längs- und Querbeschleunigung jeweils am Fahrzeugschwerpunkt berechnet werden. Die Signale $a_\mathrm{X}, a_\mathrm{Y}$ und $\dot{\psi}$ bilden das Verhalten des Fahrzeugaufbaus ab und können deshalb ebenfalls in einer Gruppe zusammengefasst werden.

Zusammenfassend werden hinsichtlich Datenfusionsebene die folgenden Varianten getestet:

a) Fusion der Sensorsignale auf Rohdatenebene durch Aneinanderreihung der einzelnen Signale (Abbildung 3.6(a))

b) Fusion der Merkmale durch einen gewöhnlichen Convolutional Layer ($d_\mathrm{in} = c$, Abbildung 3.6(b))

c) Fusion der Merkmale durch einen Separable Convolutional Layer (Abbildung 3.6(c))

   i) Teilung des Eingangsvolumens in c Einzelkanäle (gewöhnliche Depthwise Convolution)

   ii) Teilung des Eingangsvolumens in $\boldsymbol{v}$ Teilvolumina mit Gruppierungen analog zu Tabelle 3.6 (gruppierte Depthwise Convolution). Es gilt $\sum_i v_i = c$. Der Fall $v_i = 1 \forall i \in [0, c]$ entspricht der gewöhnlichen Depthwise Convolution, während $v_0 = c$ (eine einzige Gruppe über das ganze Volumen) eine übliche Convolution abbildet.

3.4.3 Datenrepräsentation

Die Literaturrecherche zeigt, dass im Bereich der mechanischen Defektdiagnose kein Konsens zur Datenvorverarbeitung existiert. Die Daten werden in unterschiedlichen Veröffentlichungen mit diversen Verfahren aufbereitet. Pan et al. [136, S. 4973] nennen vier Eigenschaften von mechanischen Daten, die die intelligente (KI-basierte) Fehlerdiagnose schwierig machen:

- Hohe Dimension
- Zufälligkeit
- Geringe Signal to Noise Ratio (SNR)
- Unbekannte Phase

Vergleich vorhandener Verfahren Um diesen Schwierigkeiten zu begegnen, wurden zeit- und frequenzbasierte Methoden entwickelt und für verschiedene CNN-basierte FDI-Systeme eingesetzt. Diese Verfahren werden nachfolgend diskutiert und in Tabelle 3.7 zusammengefasst.

Mithilfe einer DFT können Informationen über Amplitude, Phase und Frequenz eines Signals in einfacher Form dargestellt werden. Sie wird meist mit dem als FFT bekannten Verfahren [158] implementiert und bietet auch bei geringem Signal-Rausch-Verhältnis („SNR“) eine gute Verarbeitung des Signals. [159, S. 2] Nachteilig bei der Fourier Transformation ist, dass durch die Betrachtung eines zeitlich begrenzten Ausschnitts die Frequenzauflösung sinkt. Dadurch tritt der sog. Leck-Effekt auf, bei dem durch die Annäherung unbekannter Frequenzen zusätzliche

Frequenzanteile im Spektrum auftauchen [159, S. 2].
Bei der Analyse rotierender Teile wurde das ES eingesetzt, um Anregungsquellen in Maschinen gezielt zu untersuchen [159]. Dazu werden das hochfrequente Trägersignal und das niederfrequente Modulationssignal getrennt. Als Modulation werden durch Schäden in Wälzlagern oder Zahneingriffen angeregte Schwingungen interpretiert. [159, S. 151] Zur korrekten Interpretation der Ergebnisse einer Hüllkurvenanalyse ist jedoch ein hohes Maß an Erkenntnissen über die Kinematik, darunter Wellendrehzahlen, Zahneingriffsfrequenzen, Lagergeometrie etc. der untersuchten Maschine notwendig [159, S. 159]. Eines der größten Probleme der beiden Ansätze (FFT bzw. ES) ist, dass sie stationäre sowie lineare Signale annehmen [160, S. 68].

Durch die Betrachtung eines zeitlich verschiebbaren Fensters kann die STFT auf instationäre und nicht-lineare Signale angewendet werden [160, S. 68]. Sie besitzt damit gleichzeitig einen Zeit- und Frequenzbezug [13, S. 2], weshalb sie in vielen Anwendungsbereichen eingesetzt wird [161, S. 130]. Die zeitliche Auflösung der STFT ist umgekehrt proportional zur Frequenzauflösung. Es existiert eine Unschärferelation [153, S. 288], die dazu führt, dass immer nur ein Kompromiss aus Zeit- und Frequenzauflösung gefunden werden kann, was den größten Nachteil der STFT darstellt.
Im Gegensatz zur STFT setzt die WT auf eine variierende Fenstergröße (groß für niedrige, klein für hohe Frequenzen), sodass eine gute Zeit-Frequenz-Auflösung ohne Kompromiss erreicht werden kann [162]. Der Nachteil der WT, dass die Auflösung im hochfrequenten Bereich gering sein kann, wird durch eine Erweiterung auf die WPT beseitigt [163]. Während bei der WT nur der Tiefpasskanal des in Tiefpass- und Bandpasskanal aufgespaltenen Eingangssignals rekursiv aufgespalten wird, werden bei der WPT auch die Bandpasskanäle zerlegt [154, 162]. Weil bei der Zerlegung eine Mutter-Wavelet durch Translation und Dilatation angepasst wird, ist die Wahl dieser Funktion sowie die Tiefe der Zerlegung entscheidend für die Güte der Anpassung [142, S. 5]. Beim WPI wird der ohnehin höhere Rechenaufwand im Vergleich zur STFT durch das Berechnen einer Energie auf Basis der mit modifizierten WPT-Koeffizienten rekonstruierten Signale weiter erhöht.
Die HHT ist auf beliebige Signale anwendbar, denn sie basiert im Gegensatz zu STFT und WP/WPT/WPI nicht auf der Wahl einer zugrundeliegenden Funktion, sondern nutzt einen adaptiven Ansatz [142, S. 5]. Das Signal wird durch eine Empirical Mode Decomposition (EMD) zerlegt. Daraus resultieren Intrinsic Mode Functions (IMF), auf die eine Hilbert-Spektralanalyse angewendet wird [164, S. 2]. Die Zerlegung geschieht iterativ und erfordert u.a. das Verbinden lokaler Minima und Maxima durch kubische Spline-Interpolation [164, S. 3]. Es ist daher mit hohem Rechenaufwand verbunden. Bei zu vielen Iterationen kann das sog. Sieben („Sifting“) physikalisch bedeutsame Amplitudenschwankungen verwischen, daher ist die Parameterwahl mit Vorsicht durchzuführen und bedarf „einiger Erfahrung“ [164, S. 4]. In [142] verschlechterte sich zudem im Vergleich zu den STFT- und WT-basierten CNN die Klassifikationsgenauigkeit bei der Reduktion des Eingangsbildes am meisten.

Die zeitbasierten Verfahren werden nachfolgend kurz diskutiert: Der zeitbasierte 1-D-Ansatz umfasst keinen zusätzlichen Transformationsschritt, sondern verwendet direkt die (i.d.R.) normierten und teilweise standardisierten Rohdaten. Der Vorteil dieses Verfahrens ist offensichtlich: der Rechenaufwand ist gering. Bei den 2-D-Verfahren wird der Zeilenvektor in eine Matrixdarstellung gebracht; der Aufwand ist ebenfalls gering. Die Datenmatrix liegt in Form eines Graustufenbildes, einer 2-D-Zeitreihe oder eines GAF vor. Letztere Vorverarbeitungsmethode ist durch die Transformation auf Polarkoordinaten und die anschließende trigonometrische Summenbildung über alle Punkte das rechenintensivste der zeitbasierten Verfahren. Zudem wird die Datenmenge vergrößert, da aus einer Sequenz der Länge n ein Bild der Dimension $n \times n$ entsteht.

Tabelle 3.7: Übersicht der frequenzbasierten Vorverarbeitungsmethoden zur mechanischen Fehlerdiagnose mit CNN

Dim.	Verfahren	Vorteile	Nachteile
1-D	DFT/ FFT	• Weit verbreitetes Verfahren • Gute Verarbeitung von Signalen mit geringer SNR • Effiziente Implementierung	• Nimmt Stationarität und Linearität des Signals an • Leck-Effekte • Eingeschränkte Frequenzauflösung
	ES	• Trennung von hochfrequentem Trägersignal und niederfrequentem Modulationssignal • Untersuchung der Anregungsquellen • Bestimmung von Drehzahlabhängigkeiten	• Nimmt Stationarität und Linearität des Signals an • Hohes Maß an Erkenntnissen über Kinematik des Subjekts notwendig
2-D	STFT	• Weit verbreitet auf unterschiedlichen Gebieten • Gut geeignet für stationäre Signale • Zeit- und Frequenzbezug vorhanden	• Geeignete Wahl von Zeitfenster und Fensterfunktion notwendig • Kompromiss aus Zeit- und Frequenzauflösung ist zu finden (Unschärferelation)
	HHT	• Keine Wahl einer Basis erforderlich • Geeignet für nicht-lineare und nicht-stationäre Signale • Nützlich zur Eigenschaftslokalisierung von beliebigen Signalen	• Sifting kann bedeutsame Amplitudenschwankungen verwischen • Parameterwahl bedarf Erfahrung • Anfällig bei Datenreduktion
	WPI	• WPE gut geeignet für Analyse instationärer Signale durch modulierte Fensterfunktion • Energiekonservierend	• Abhängigkeit von Wahl der Mutter-Wavelet • Wahl der Tiefe der WPT entscheidet über Repräsentationsgüte • Hoher Rechenaufwand

Zu untersuchende Verfahren Einige der o.g. Methoden zur Datenaufbereitung werden in dieser Arbeit umgesetzt und anhand des Baseline-Modells bezüglich der KKR miteinander verglichen (Tabelle 3.8).

Tabelle 3.8: Zu untersuchende Vorverarbeitungsmethoden

Verfahren	Dimension	Zeitbasiert	Frequenzbasiert	Motivation
Skalierung	1-D	x		Standard
Standardisierung	1-D	x		Standard
linearer Detrend	1-D	x		Wie Merk [5]
DFT/FFT	1-D		x	Gängiges Verfahren
Graustufenbild	2-D	x		Einfachheit, zeitbasiert
STFT	2-D	x	x	Zeit-Frequenz-basiert, Standard
WPI	2-D	x	x	Instationäre Signale, Wavelets
GAF	2-D	x	x	Phasenraumdarstellung
Rekurrenzplot	2-D	x		Phasenraumdarstellung (alternativ)

Das ES sowie die HHT werden aufgrund der benötigten Kenntnisse und Erfahrungen nicht

umgesetzt. Zusätzlich zum GAF kommen Rekurrenzplots zum Einsatz, die im Bereich der Klassifikation von Zeitreihen ebenfalls erfolgreich eingesetzt wurden [165]. Bei den 2-D-Verfahren kann das Format eines Bildes durch Skalierung (Up-/Downsampling) verändert werden. Aus Gründen der Vergleichbarkeit wird jedoch für alle 2-D-Ansätze ein quadratisches Bild angestrebt, dessen Größe je nach Sequenzlänge und Verfahren variiert. Als Referenzbeispiel für unterschiedliche Darstellungen werden die Raddrehzahlen eines Datenbeispiels dargestellt (Abbildung 3.7(a)). Anhand des Beispiels wird deutlich, dass der Datensatz nicht ausschließlich Konstantfahrten enthält, obwohl die Enable-Bedingungen erfüllt sind.

Auf Details zu den umgesetzten Methoden wird nachfolgend eingegangen; für die exakte Umsetzung wird auf den Quellcode (`data_handling.py`, `preprocessing.py`) verwiesen.

Skalierung, Standardisierung und Detrend Für eine simple Datenaufbereitung wird kein Expertenwissen benötigt [150], was für eine möglichst aufwandsarme Datenvorverarbeitung spricht. Bei sog. End-to-End-Systemen, die mit Rohdaten arbeiten, wird auf aufwändige Vorverarbeitungsschritte verzichtet, was gleichzeitig den menschlichen Arbeitsanteil reduziert. In einigen Veröffentlichungen wird zudem die Echtzeitfähigkeit des Diagnosesystems unterstrichen [133, S. 193, 21, S. 155], die mit aufwändigen Rechenverfahren nicht eingehalten werden kann.

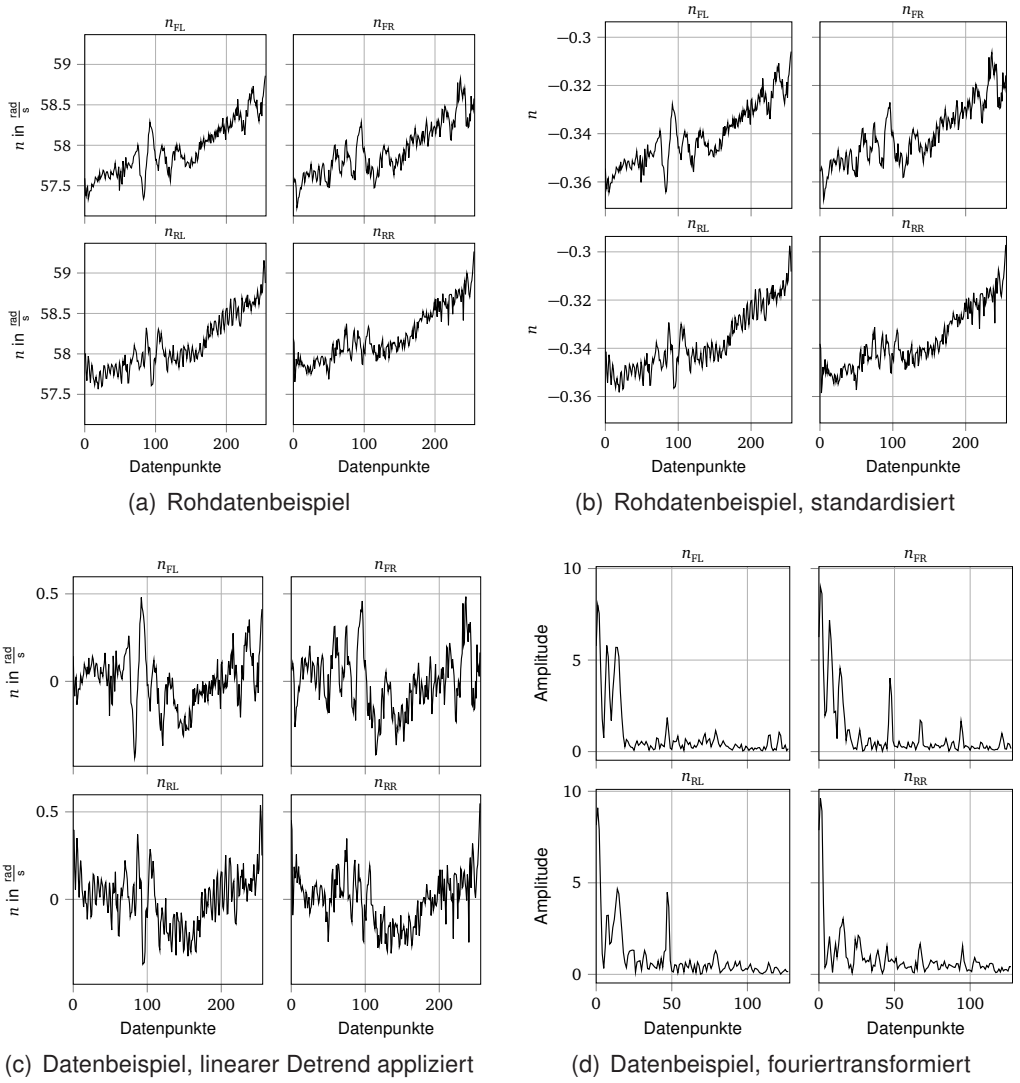
Abhängig vom Wertebereich der Eingangsdaten empfiehlt sich eine Skalierung oder Standardisierung, um eine implizite Über- oder Untergewichtung bestimmter Eingänge zu verhindern. Im Zusammenhang mit dem Backpropagation-Algorithmus, der in vielen ML-Systemen eingesetzt wird, konnte zudem mit normierten Eingangsdaten eine schnellere Konvergenz und damit verbesserte Trainingsperformance festgestellt werden [46, S. 16]. Die Skalierung und Standardisierung wurden bereits in Absatz 2.1.6 eingeführt. Nachfolgend werden Implementierungsdetails angesprochen.

Zur Skalierung bzw. Standardisierung werden die Datenbeispiele jeweils für einen Datenkanal in ihrer Gesamtheit betrachtet. Bei einer Skalierung jedes einzelnen Beispiels würden z. B. Informationen über unterschiedliche hohe Geschwinidkeiten verloren gehen. Daher wird ein einziger (langer) Vektor mit allen Trainingsbeispielen eines Datenkanals gebildet und dieser auf den gewünschten Wertebereich skaliert. Danach wird der Vektor in die einzelnen Datenbeispiele zurücktransformiert. Es werden die Skalierungen auf die Intervalle $[0, 1]$ und $[-1, 1]$ untersucht.

Merk [5] entfernt den Mittelwert eines Datenbeispiels jeweils mit einer linearen Funktion, was als „linearer Detrend“ bezeichnet wird (Abbildung 3.7(c)).

Dies ist inbesondere bei den Raddrehzahlen nützlich, um die variierende Geschwindigkeit zwischen unterschiedlichen Messfahrten zu entfernen. Durch den linearen Detrend wird aus einer Beschleunigungsfahrt aus Abbildung 3.7(a) eine Konstantfahrt mit geringer Geschwindigkeit. Dadurch werden implizit Amplitudeninformationen entfernt, Frequenzinformationen bleiben jedoch erhalten. Beide Informationsteile bleiben bei Standardisierung und Skalierung erhalten (Abbildung 3.7(b) und A.4). (Linearer) Detrend, Standardisierung sowie Skalierung können beliebig mit den nachfolgend vorgestellten Verfahren kombiniert werden.

Fast Fourier Transformation (FFT) Zeitbasierte Daten enthalten häufig einen hohen Rauschanteil und besitzen hohe Dimensionalität. Um diesen Problemen entgegenzuwirken, können Signalverarbeitungsmethoden wie die DFT eingesetzt werden. [166, S.3] Mit einer DFT, die als FFT implementiert ist, wird nur das Frequenzspektrum eines Beispiels für das Netztraining verwendet. Um Leck-Effekte zu vermeiden und eine Glättung des unter-abgetasteten Signals zum Rand



(a) Rohdatenbeispiel

(b) Rohdatenbeispiel, standardisiert

(c) Datenbeispiel, linearer Detrend appliziert

(d) Datenbeispiel, fouriertransformiert

Abbildung 3.7: Datenbeispiel in Rohform, standardisiert und nach linearem Detrend. Das Label des Beispiels ist „FL“ (defekt), die ursprüngliche Sequenzlänge beträgt 256 Datenpunkte.

herbeizuführen, wird als üblicher Standard ein von-Hann- oder Hanning-Fenster verwendet [167, S. 60f]. Bei der FFT halbiert sich die Sequenzlänge des verwendeten Datenbeispiels, da das Spektrum symmetrisch ist (Abbildung 3.7(d)).

Graustufenbild Dieser „Signal to image“-Ansatz transformiert den Zeitvektor in ein Graustufenbild. Wen et al. [150] nutzen eine 8-bit Auflösung, d. h. den Wertebereich [0, 255]. Ist die Auflösung zu gering, wird bereits bei der Vorverarbeitung ein Subsampling vorgenommen. Um dies zu vermeiden, wird das Bild stattdessen auf ein 16-bit Bild mit Wertebereich [0, 65535] transformiert. Bei der Transformation wird pro Kanal über alle Datenbeispiele hinweg skaliert. Durch die unterschiedlichen Betriebspunkte (Geschwindigkeiten) bei den Messfahrten liegen aber insbesondere die Raddrehzahlen in unterschiedlichen Wertebereichen. Zusätzlich wird daher eine Bereinigung des Mittelwerts durch linearen Detrend vorgesehen, da sonst implizit eine stärkere Gewichtung der höheren Drehzahlen erfolgt. Nach der Skalierung wird der Vektor zeilenweise von oben nach unten in ein Bild übertragen (Abbildung 3.8(a)).

Der erste Punkt des Vektors entspricht damit dem Punkt oben links, der letzte dem unten rechts.

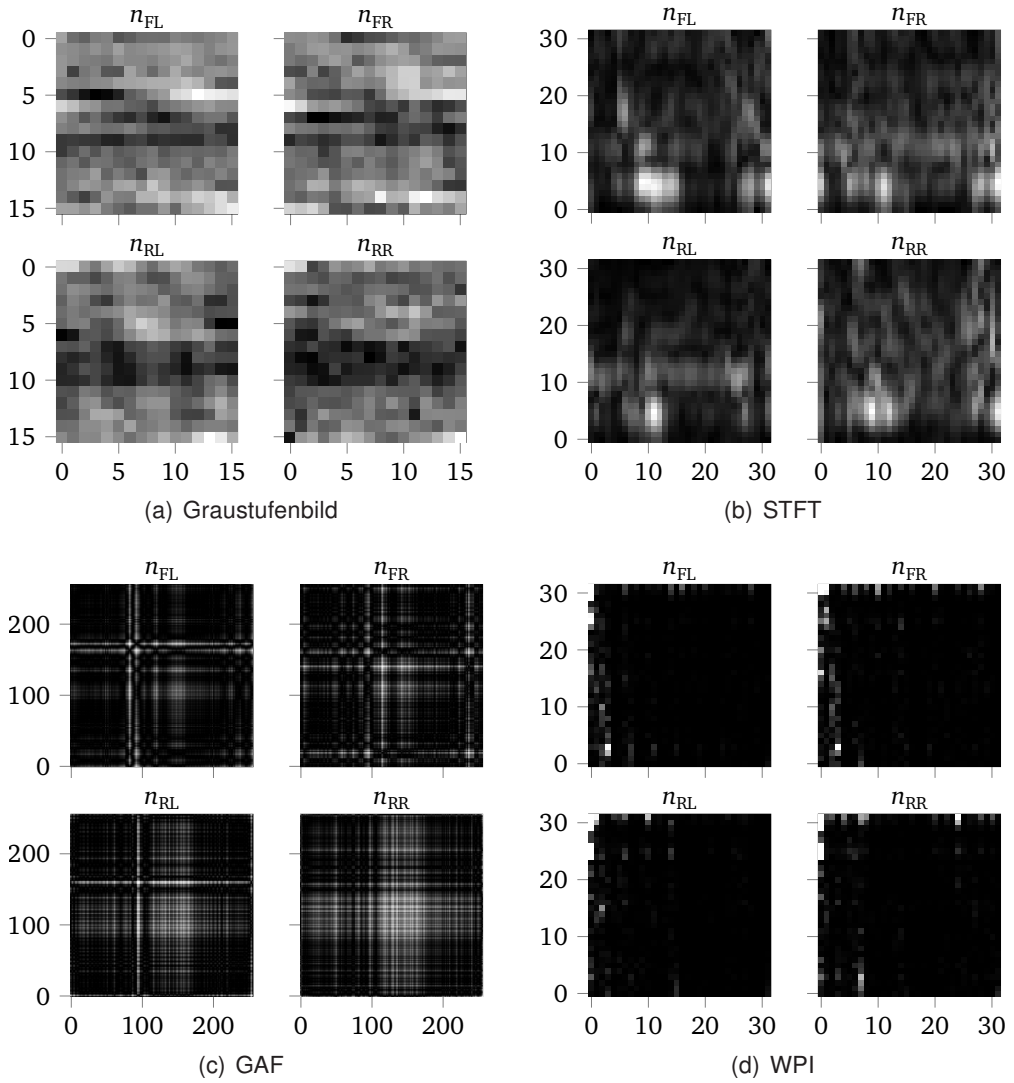


Abbildung 3.8: Datenbeispiel als Graustufenbild, STFT, Rekurrenzplot und WPI. Das Label des Beispiels ist „FL“ (defekt), die ursprüngliche Sequenzlänge 256 Datenpunkte.

Bei Verwendung von Graustufenbildern sollte für die Sequenzlänge eine gerade Potenz von 2 gewählt werden, z. B. 2^8 oder 2^{10}. Dadurch entsteht ein quadratisches Bild ohne Verschnitt von Daten, denn zur Berechnung der (quadratischen) Bildgröße wird die Quadratwurzel der Sequenzlänge abgerundet. Bei $2^7 = 128$ Datenpunkten wäre das Bild damit maximal $11 \times 11 = 121 < 128$ Pixel groß. Bei höheren Potenzen ergeben sich entsprechend höhere Verluste.

Short-Time Fourier Transformation (STFT) Datenbeispiele, die mit der FFT transformiert werden, verlieren die variierenden Informationen in der Zeitdomäne [154]. Um diesem Problem zu begegnen, kann die STFT eingesetzt werden. Bei der STFT wird ein zeitlich verschiebbares Analysefenster zur Extraktion des lokalen Spektrums mittels Fourier-Transformation verwendet [153]. Über die Parameter der STFT wird die Zeit-Frequenz-Auflösung eingestellt.

Das Verfahren wird auf jeden Datenkanal jedes Beispiels einzeln angewendet. Wie bei den bisherigen Methoden wird im ersten Schritt der lineare Trend aus den Daten entfernt. Bei der STFT erfolgt analog zur FFT eine Unterabtastung des Signals. Das Fenster kann zur Glättung mit einer Überlappung (Overlap) bewegt werden. Bei entsprechender Parameterwahl

aus Fensterlänge, Überlappung und FFT-Länge lässt sich ein 256 Datenpunkte langes Signal auf ein 33×31 großes Bild transformieren, das anschließend auf 32×32 Pixel skaliert wird (Abbildung 3.8(b)).

Gramian Angular Field (GAF) Das GAF ist eine 2-D-Darstellung des Zeitsignals, die erstmals in [151, 168] vorgeschlagen wurde und den zeitlichen Zusammenhang des Signals bewahrt. Ein Zeitsignal wird auf ein festes Intervall, z. B. $[-1, 1]$, skaliert und anschließend in Polarkoordinaten transformiert. Dazu wird der Wert als winkelförmiger Kosinus und der Zeitstempel als Radius interpretiert. Durch wiederholte Anwendung des Kosinus auf die Winkelsumme entsteht eine „Gramian Matrix“ (Abbildung 3.8(c)). Das GAF ist als Gramian Angular Summation Field nach [169] implementiert. Hierfür wird ebenfalls zuerst der lineare Trend aus dem Signal entfernt.

Rekurrenzplot Der Rekurrenzplot (“recurrence plot“) stellt die Zeitreihe im zweidimensionalen Phasenraum dar. Der Kerngedanke des Verfahrens ist die Erfassung derjenigen Punkte, an denen Trajektorien im Phasenraum in einen vorherigen Zustand zurückkehren [165]. Die Funktion ist nach [169] implementiert und berechnet die Frobeniusnorm aller Punkte zueinander. Die Zeitachse wird auf beide Seiten der entstehenden Matrix abgebildet, weshalb sich aus einem 256 Datenpunkte langen Signal ein 256×256 Pixel großes Bild ergibt (Abbildung A.5). Werden mit dem Rekurrenzplot gute Ergebnisse erzielt, ist eine Reduzierung der Datenmenge durch Skalierung (Verkleinerung) des Bildes in Erwägung zu ziehen.

Die Matrix enthält Textur-Informationen wie Diagonal-, Horizontal- und Vertikallinien sowie Topologieinformationen, die als homogen, periodisch, trendbehaftet und unterbrochen charakterisiert werden [165]. Ein Farbverlauf deutet auf Trends bzw. Verschiebungen hin, während Horizontal- und Vertikallinien eine geringe oder keine Zustandsänderung beschreiben [170].

Wavelet Packet Energy Image (WPI) Ding, He [154] benutzen zur Wälzlagerdiagnose bei instationärem Betrieb ein Bild pro Signal, das aus den WPT-Koeffizienten des Signals generiert wird (Abbildung 3.8(d)). Für ein Bild der Dimension 32×32 muss die Dekompositionsebene der WPT zu zehn gewählt werden. Mit einem Wavelet der Daubechies-Familie [171] wird das Signal durch 2^{10} Koeffizienten beschrieben. Durch sukzessive Entfernung aller Rekonstruktionskoeffizienten bis auf einen werden 2^{10} rekonstruierte Koeffizientenmengen erzeugt, deren quadratische Summe in die WPI-Matrix eingetragen wird.
Ein Großteil der Energie eines Signals steckt in den Tiefpassanteilen [172, S. 323], weshalb insbesondere diese zu hohen Werten in der WPI-Matrix beitragen. Bedingt durch die verwendeten Daten liegen die Einträge der Matrix in unterschiedlichen Größenordnungen, was dazu führt, dass wenige Pixel hell und die restlichen dunkel sind. Daher werden die Werte oberhalb einer festen Grenze abgeschnitten. Durch die Begrenzung wird gegenüber [154] die energiekonservierende Eigenschaft des Verfahrens verletzt. Inwiefern die Ergebnisse durch die Wahl der Grenze beeinflusst werden, ist in weiterführenden Studien zu untersuchen.

Das Verfahren ist insbesondere durch die Zerlegung und Rekonstruktion mittels WPT auf zehn Ebenen mit hohem Rechenaufwand verbunden. Das Preprocessing des DD-Datensatzes nimmt bei einer Sequenzlänge von 256 Datenpunkten über zwei Stunden in Anspruch, bei 128 Datenpunkten ca. viereinhalb (Intel i7-3770 mit 4 CPUs @ 3,4 Ghz, 16 GB RAM). Bei 6455 Datenbeispielen beläuft sich die Rechenzeit pro Beispiel auf über $2\,\mathrm{s}$, was mit einer Echtzeitanwendung zur Fahrwerksdiagnose nicht vereinbar ist. Zu Evaluationszwecken wird das Verfahren dennoch untersucht.

3.4.4 Wahl der Netzwerktopologien

Kapitel 2 zeigt, dass im Bereich der mechanischen Komponentendiagnose unterschiedliche CNN-Architekturen eingesetzt werden können. Während die meisten der verwendeten Netze flach sind, geht der Trend bei bildverarbeitenden CNN zu komplexen Architekturen: Skip connections, Inception Module und SE-Blöcke sind nur einige Beispiele für Bestandteile solcher Netze. Durch den zeitlich begrenzten Umfang der Arbeit und die große Auswahl an Kombinationsmöglichkeiten unterschiedlicher Layer und Blöcke von CNN können nicht beliebig viele Kombinationen untersucht werden. Es werden daher unterschiedliche, bestehende Netzwerktopologien implementiert und getestet. In diesem Abschnitt wird eine Auswahl von verschiedenen Netzen getroffen, die zur Dämpferdefektdiagnose untersucht werden (Tabelle 3.9).

Tabelle 3.9: Bewertungsmatrix für einige zur Auswahl stehende CNN-Architekturen. Eingefärbte Zellen markieren die zu untersuchenden Netze. Bewertung: 1 = unzureichend, 3 = ausreichend, 6 = gut, 9 = exzellent

		Applizierbarkeit				**Netzkomplexität**			**Ranking**				
		Standard	Aufwand	Anwendung		Parameter	Netztiefe		Aktualität	Verbreitung	Innovativität		
Architektur	Gew.	40 %				20 %			40 %				Σ
	Gew.	20 %	40 %	40 %	Σ	75 %	25 %	Σ	30 %	30 %	40 %	Σ	
AlexNet [20]	Pkt.	3	3	1		1	3		1	6	3		
	Gew.	0,6	1,2	0,4	2,2	0,6	1,2	1,8	0,3	1,8	1,2	3,3	**2,4**
SqueezeNet [79]	Pkt.	3	6	1		6	6		3	3	6		
	Gew.	0,6	2,4	0,4	3,4	3,6	2,4	6,0	0,9	0,9	2,4	4,2	**4,5**
MobileNet [80]	Pkt.	3	3	1		3	6		9	3	3		
	Gew.	0,6	1,2	0,4	2,2	1,8	2,4	4,2	2,7	0,9	1,2	4,8	**3,7**
VGGNet [71]	Pkt.	6	6	3		1	6		3	6	3		
	Gew.	1,2	2,4	1,2	4,8	0,6	2,4	3,0	0,9	1,8	1,2	3,9	**3,9**
LeNet [19]	Pkt.	6	6	6		9	3		6	9	6		
	Gew.	1,2	2,4	2,4	6,0	5,4	1,2	6,6	1,8	2,7	2,4	6,9	**6,5**
GoogLeNet [70]	Pkt.	3	1	1		3	9		3	6	6		
	Gew.	0,6	0,4	0,4	1,4	1,8	3,6	5,4	0,9	1,8	2,4	5,1	**4,0**
ResNet [72]	Pkt.	3	1	1		3	9		6	6	9		
	Gew.	0,6	0,4	0,4	1,4	1,8	3,6	5,4	1,8	1,8	3,6	7,2	**4,7**
SE-ResNeXt [81]	Pkt.	3	1	1		3	9		9	3	9		
	Gew.	0,6	0,4	0,4	1,4	1,8	3,6	5,4	2,7	0,9	3,6	7,2	**4,7**
Verstraete [142]	Pkt.	6	6	6		6	3		6	3	3		
	Gew.	1,2	2,4	2,4	6,0	3,6	1,2	4,8	1,8	0,9	1,2	3,9	**4,9**
WDCNN [137]	Pkt.	1	3	6		6	3		6	1	6		
	Gew.	0,2	1,2	2,4	3,8	3,6	1,2	4,8	1,8	0,3	2,4	4,5	**4,4**
TICNN [139]	Pkt.	3	3	9		6	3		6	1	6		
	Gew.	0,6	1,2	3,6	5,4	3,6	1,2	4,8	1,8	0,3	2,4	4,5	**4,9**
LiftingNet [136]	Pkt.	1	1	6		6	3		6	1	6		
	Gew.	0,2	0,4	2,4	3,0	3,6	1,2	4,8	1,8	0,3	2,4	4,5	**4,1**
DamNet	Pkt.	3	3	9		6	3		9	1	9		
	Gew.	0,6	1,2	3,6	5,4	3,6	1,2	4,8	2,7	0,3	3,6	6,6	**5,6**

Es werden CNN-Architekturen aus der Bildverarbeitung und der mechanischen Komponen-

tendiagnose gegenübergestellt. Von den CNN aus der Bildverarbeitung sind insbesondere die aufgeführt, die als Meilensteine in der Historie von CNN angesehen werden können (LeNet, AlexNet, VGGNet, GoogLeNet, ResNet). Zusätzlich sind SqueezeNet und MobileNet als Parameter-effiziente Varianten sowie SE-ResNeXt als Stand der Technik (ILSVRC-Gewinner 2017) aufgenommen.
Von den Diagnose-Netzen aus Abschnitt 2.5.5 werden nur vier betrachtet, was unterschiedlich begründet werden kann. Beispiel: Die adaptiven 1-D-CNN [21, 22, 134, 135, 145] werden ausschließlich mit einem einzigen Signal verwendet, sodass in der Applikation mit Stahlstreben ein Netz pro Gelenk verwendet wird. Das Training eines Netzes pro Dämpfer bedeutet eine Vervierfachung der Komplexität, die nicht erstrebenswert ist. Jia et al. [59] legen den Schwerpunkt auf unausgewogene (imbalanced) Datensätze, was auf den DD-Datensatz nicht zutrifft. Guo et al. [140] verwenden hierarchische Netze, um zusätzlich zur Defektdiagnose eine Abschätzung der Fehlerschwere abzugeben. Dies ist im jetzigen Entwicklungsstadium des FDI-Systems zur Dämpferdefektdiagnose nicht vorgesehen und benötigt weitere, nicht vorhandene Daten. Das System von Krummenacher et al. [60] ist ein speziell auf das Problem zugeschnittenes CNN-basiertes FDI-System, das nicht auf die vorliegende Anwendung übertragen werden kann. Von den übrigen Diagnose-Netzen stehen insbesondere solche im Vordergrund, die 1-D-Convolutions verwenden, da Messdaten für die mechanische Komponentendiagnose ausschließlich mit der Zeit – einem 1-D-Parameter – korrelieren [148]. Darüber hinaus wird eine speziell auf den Fall der Dämpferdefektdiagnose zugeschnittene Architektur namens DamperNet („DamNet“) entworfen.

Zur Bewertung werden drei Kategorien (Applizierbarkeit, Netzkomplexität, Ranking) unterschieden, die unterschiedlich gewichtet werden. Hinsichtlich Applizierbarkeit werden drei Punkte mit individueller Gewichtung bewertet:

- Inwiefern werden Standards verwendet oder eingeführt(hohe Wertung bedeutet mehr Standards).
- Wie aufwändig ist die Implementierung des Netzes (hohe Wertung bedeutet geringen Aufwand).
- Wie passend ist die Anwendung für die vorliegende Arbeit, d. h. wurde die Architektur bspw. schon für die Defektdiagnose verwendet.

Bei der Netzkomplexität werden die Anzahl der Parameter sowie die Tiefe des Netzes bewertet, wobei weniger Parameter und ein tieferes Netz mehr Punkte bedeutet. Die Kategorie Ranking setzt sich aus den Punkten Aktualität, Verbreitung und Innovativität zusammen und bewertet die Netze aus einer allgemeinen Perspektive.

Die geringste Wertung mit 2,4 Punkten erhält AlexNet, während das LeNet mit 6,5 Punkten am besten abschneidet. Als Untersuchungskandidaten kommen die Netze in Frage, die mindestens eine Wertung von $2,4 + \frac{6,5-2,4}{2} = 4,45$ Punkten erhalten und somit in der oberen Punktehälfte liegen.
SE-ResNeXt und ResNet erreichen dieselbe Punktzahl (4,7). Weil das SE-ResNeXt die Komponenten vom ResNet inkorporiert, wird auf eine Implementierung des ResNet verzichtet. Das WDCNN liegt mit 4,4 Punkten knapp unterhalb der Grenze, daher ist eine Untersuchung dieses Netzes ebenfalls denkbar. Es stellt allerdings die erste Version von Zhang et al. [137] zur Defektdiagnose von Wälzlagern dar und die Autoren haben das Netz in [139] zum TICNN entwickelt, das mit 4,9 Punkten untersucht werden wird. Die Auswahl aller zu untersuchenden Netze beschränkt sich damit auf die Netze mit eingefärbten Ergebniszellen.

Tabelle 3.10: Parameter der Rastersuche für die Architektur-Variation

Filtergröße (symmetrisch)	1, 3, 5, 7, 9, 11, 13
Filteranzahl (im ersten Conv Layer)	8, 16, 32, 64, 128, 256
Anzahl Blöcke Convolution(s)–Max-Pooling	1, 2, 3, 4, 5
Anzahl Convolutional Layer pro Block	1, 2, 3, 4

Anhand der Verwendung obiger Netze kann lediglich festgestellt werden, ob sich diese gut auf die Anwendung in dieser Arbeit übertragen lassen. Für verlässliche Aussagen über Einflüsse der Architektur sind die Hyperparameter wie Filteranzahl und -größe der verschiedenen Netze – bspw. SqueezeNet und LeNet – zu unterschiedlich. Ebenso variiert die Tiefe des Netzes von zwei Layern mit Gewichten (Baseline-Modell) bis zu 29 (SE-ResNeXt), was den Vergleich erschwert. Zur Untersuchung der architektonischen Einflüsse Netztiefe, Layer-Anordnung, Filtergröße und Filteranzahl wird daher eine Rastersuche anhand des Baseline-Modells durchgeführt (Tabelle 3.10).

Die Rastersuche wird in zwei Stufen durchgeführt, was zu einer Halbierung des Zustandsraums führt. Dies erlaubt eine größere Abdeckung der jeweiligen Hälften. Rechenbeispiel: Werden 4 Filtergrößen, 4 Filteranzahlen, 4 Blöcke und 3 Conv-Anzahlen getestet, ergibt das einen Zustandsraum von $4 \cdot 4 \cdot 4 \cdot 3 = 192$ Kombinationen. Stattdessen lassen sich jedoch unter geringerem Aufwand 7 Filtergrößen, 6 Filteranzahlen und anschließend 5 Blöcke und 4 Conv-Anzahlen testen ($7 \cdot 6 + 5 \cdot 4 = 62$), wenn die zweite Hälfte mit einer festen Kombination der ersten Hälfte untersucht wird.
Der erste Layer ist maßgeblich für die Merkmalsextraktion verantwortlich, weshalb er in mehreren Veröffentlichungen [70, 74, 75, 79, 81] gesondert behandelt wird. Ähnlich zur Untersuchung aus Unterkapitel 3.3 werden daher im ersten Schritt nur Filtergröße und -anzahl im ersten (und zu dem Zeitpunkt einzigen) Convolutional Layer untersucht, um eine gute Referenz für die KKR zu finden. Für eine bessere Identifikation des Tiefen-Einflusses wird das Baseline-Modell anschließend so erweitert, dass der erste Convolutional Layer identisch bleibt (mit den Parametern aus dem vorherigen Schritt). Bilden Convolutional und Max-Pooling Layer einen Block, wird einerseits der Einfluss der Blockanzahl und andererseits die Anzahl der Convolutional Layer pro Block untersucht.

3.4.5 Variation der Datenmenge

Der Einfluss der Datenmenge auf die Güte von ML-Systemen und insbesondere DL-Systemen gilt als unumstritten [27, S. 415, 54, S. 22f]. Die Effektivität von großen Datenmengen wird in mehreren Veröffentlichungen diskutiert und bestätigt [173–175]. Sind die Ergebnisse eines ML-Systems nicht zufriedenstellend, wird daher i.A. die Verwendung von mehr Daten vorgeschlagen [54, S. 462, 27, S. 414]. Dadurch kann die KKR gesteigert, Overfitting vermieden und die Generalisierungsfähigkeit des Netzes gewährleistet werden [124, S. 6, 137, S. 12]. Mit der Datenmenge nimmt jedoch die durchschnittliche Rechenzeit exponentiell zu [121, S. 3142], weshalb die Wahl des Datensatzumfangs einen Kompromiss zwischen gewünschten Netzeigenschaften und Rechenkomplexität darstellt.

Entscheidend ist die Frage, wieviel mehr Daten benötigt werden. Goodfellow et al. [27, S. 415] schlagen zur Beantwortung dieser Frage die Extrapolation von Lernkurven dar, bei denen der Generalisierungsfehler über der Trainingsdatensatzgröße abgebildet wird. Den Datensatz nur um einen Bruchteil der Gesamtanzahl von Trainingsbeispielen zu vergrößern ist im Normalfall

nicht ausreichend. Deswegen wird vorgeschlagen, dass mit Datensatzgrößen auf einer logarithmischen Skala experimentiert wird. [27, S. 415] Sun et al. [175, S. 8] fanden heraus, dass die Leistungsfähigkeit bildverarbeitender Systeme logarithmisch mit der Größe des Trainingsdatensatzes skaliert, was diesen Vorschlag unterstützt.

Die Datenmenge des FDI-Systems kann durch

- Veränderung der Sequenzlänge,
- Augmentierung von Daten und
- Einfahren neuer Daten

verändert werden. Die Anzahl der Trainingsbeispiele ist stark mit der Sequenzlänge korreliert. Steigt die Sequenzlänge, sinkt die Anzahl der Beispiele (Abschnitt 3.1.3). Dadurch stellt dies nur eine Pseudo-Variation dar, denn die effektive Datenmenge bleibt weitestgehend unverändert. Während in der Bildverarbeitung Spiegelungen, Ausschnitte und Farbverschiebungen genutzt werden, um die Anzahl der Trainingsdaten (künstlich) zu vermehren, bleibt bei Zeitsignalen nur das künstliche Einbringen von Rauschen oder die Zerlegung einer Messung mit Überlappung. Durch die Verwendung realer Messdaten ist die SNR bereits höher als bei Prüfstandsanwendungen, weshalb auf die Einbringung künstlichen Rauschens verzichtet werden sollte. Lee et al. [149] stellen fest, dass die Größe der Überlappung nicht direkt mit der Genauigkeit des Modells korreliert, weshalb aus Zeit- und Ressourcengründen auf eine Augmentierung verzichtet wird. Damit bleibt einzig das Einfahren neuer Messdaten, um den Einfluss der Datenmenge zu untersuchen.

Zusammenfassend wird für dieses Untersuchungspaket folgendes Vorgehen angestrebt:

1. Erstellung o.g. Lernkurven auf logarithmischer Skala durch Verwendung von Teilmengen bis hin zum größtmöglichen Umfang des DD-Datensatzes.
2. Extrapolation der Lernkurven zur Abschätzung der benötigten Datenmenge.
3. Einfahren neuer Messdaten im erforderlichen Umfang einschließlich Aufbereitung analog zu Merk [5].
4. Überprüfung, ob eine ähnlich hohe KKR wie beim DD-Datensatz erreicht werden kann, wenn ein Datensatz verwendet wird, der einen ähnlichen Umfang wie der DD-Datensatz besitzt.
5. Überprüfung der extrapolierten Vorhersage.

Gleichzeitig lässt sich mit einem neuen Datensatz die getroffene Wahl bzgl. Signalauswahl, Datenfusionsebene und -repräsentation sowie Netzwerktopologie überprüfen. Damit wird die Übertragbarkeit des gewählten Ansatzes validiert (Unterkapitel 4.3).

3.5 Architektur- und Implementierungsdetails

In diesem Abschnitt wird neben Implementierungsdetails auf Details zu unterschiedlichen Architekturen eingegangen. Diese sind nicht alle exakt wie in den entsprechenden Originalveröffentlichungen re-implementiert, da es insbesondere um die Konzepte und weniger um die genaue Konfiguration wie bspw. die verwendete Aktivierungsfunktion geht. Für jede der vorgestellten Architekturen werden eine 1-D- und 2-D-Variante implementiert, die sich nur in

der Dimension der Filter bei Convolutional und Pooling-Layer unterscheiden. Aus Gründen der Übersichtlichkeit und des Umfangs wird nur auf die 2-D-Variante eingegangen.

3.5.1 Implementierung

Die Implementierung erfolgt in Python 3 mit TensorFlow [15] in der Version 1.9. Die Begründung dieser Wahl ist angelehnt an [36, Kap. 3] und auf die Popularität gegenüber anderen Bibliotheken wie Theano, Torch und Caffee sowie die Flexibilität gegenüber Keras zurückzuführen. Die Liste aller in der Arbeit verwendeten Python-Module (`requirements.txt`) befindet sich im Anhang (Code B.1).

Netztraining Bei den Hauptuntersuchungsaspekten werden unterschiedliche Netze in ein Framework eingebettet, sodass das Training jeweils gleich abläuft. Die Kostenfunktion setzt sich aus L2-Regularisierung aller trainierbaren Gewichte (Convolutional und FC-Layer) und Kreuzentropie zusammen. Der Output-Layer ist durch die Wahl der „sparse softmax cross entropy“ [176] ein Softmax Output für sich gegenseitig ausschließende Klassen. Als Optimierer wird Adam [52] eingesetzt, um die Kostenfunktion zu minimieren. Standardmäßig kommen ReLU-Aktivierungsfunktionen in allen Layern bis auf den Output-Layer zum Einsatz und die Fehlerquote beim Validierungsdatensatz wird nach jeder Epoche bzgl. der Early Stopping-Kriterien untersucht. Das Training endet bei Erreichen der maximalen Epochenanzahl oder wird vorzeitig durch Early Stopping beendet. Die Initialisierungsstrategie der Gewichte folgt [155] und die Bias-Terme werden zu Beginn auf eine kleine Konstante $(0,01)$ gesetzt.

Evaluation und finales Modell Die Klassifikationsgenauigkeit wird stets anhand des Testdatensatzes angegeben, unabhängig davon, ob eine k-fache Kreuzvalidierung vorgenommen wird oder nicht. Die k-fache Kreuzvalidierung wird insbesondere dazu verwendet, ein reproduzierbares Ergebnis unter statistischen Einflüssen sicherzustellen.
Bei der Kreuzvalidierung wird die Gesamtmenge der Trainingsprozess-Daten zufällig in k Teilmengen zerlegt. Bei jedem der k Durchläufe werden $k-1$ Teilmengen für das Training und eine Teilmenge zur Validierung herangezogen, wobei von einem Durchlauf zum nächsten permutiert wird. Mit den Trainingsdaten werden die Gewichte und Bias-Terme des Netzes angepasst („trainiert“), die für jeden Durchlauf neu und (größtenteils) zufällig initialisiert werden. Dadurch unterliegen alle Ergebnisse einer Streuung. Die Validierungsdaten beeinflussen bspw. den Early Stopping Zeitpunkt und damit einen Hyperparameter des Netzes (die Epochenanzahl). Für jeden Test und jeden der k Durchläufe wird derselbe Testdatensatz verwendet, um eine vergleichbare Basis zur Bewertung zu erhalten.

Durch die Angabe eines gemittelten Ergebnisses über alle Durchläufe stellt sich die Frage, welches Modell für das finale FDI-System gewählt wird, da nicht alle gleichzeitig verwendet werden können. Eine einfache Möglichkeit ist die Wahl desjenigen Netzes (Modells) mit der höchsten KKR. Dies ist eine naheliegende und einfache Lösung für eine prototypische Implementierung, weshalb diese für dieses Entwicklungsstadium angestrebt wird. Eine weitere Option ist die Verwendung aller Modelle in einem Ensemble und die Netzvorhersagen bspw. durch Mehrheitsentscheid zu bestimmen.
Für ein ausgereifteres System kann ein Netz von der zufälligen Initialisierung aus mit der gesamten Trainingsprozess-Datenmenge (Trainings- plus Validierungs-Daten) trainiert werden, wobei als Trainingsdauer die Anzahl der Epochen durch den vorherigen Early Stopping-Zeitpunkt bestimmt wird. Alternativ lässt sich das Training für das finale FDI-System ausgehend von

den Parametern des besten Durchlaufs für wenige Epochen um den Validierungs- und/oder Testdatensatz erweitern, um eine höhere Generalisierungsfähigkeit zu erhalten. [27, S. 241f]

3.5.2 Netze aus der Bildverarbeitung

In Abschnitt 3.4.4 wurden mehrere Architekturen zur Untersuchung festgelegt, von denen einige ihren Ursprung in der Bildverarbeitung haben. Auf diese Netze wird nachfolgend eingegangen.

LeNet LeNet-5 [62, 91] kann als das erste CNN angesehen werden. Die LeNet-Architektur stellt eine simple Netztopologie dar (Abbildung 3.9) und besitzt nach wie vor Aktualität. Sie stammt aus der Bildverarbeitung und wurde bereits erfolgreich in die Defektdiagnose übertragen [150]. Die

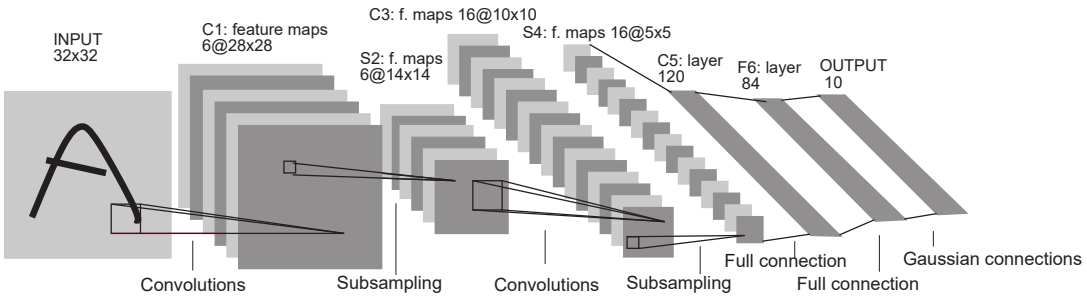


Abbildung 3.9: CNN-Architektur LeNet-5 nach [62, S. 7]. Es sei auf die Notation wie bspw. „$6 @ 28 \times 28$“ verwiesen, bei der das Format der Feature Maps und nicht wie sonst in dieser Arbeit das Format der Kernel angegeben wird

Architektur besitzt nur zwei Convolutional Layer mit wenigen Filtern und ein anschließendes MLP (Tabelle A.1). Abweichend zu [62] erfolgt die Dimensionsreduktion in der hier implementierten Variante durch Max-Pooling statt durch Mean-Pooling. Als zusätzliche Regularisierung wird neben der L2-Regularisierung Dropout eingesetzt.

SqueezeNet SqueezeNet [79] ist ein Netz aus der Bildverarbeitung, das mit 50 mal weniger Parametern als AlexNet [20] die gleiche Klassifikationsgenauigkeit erreicht. Es wird ein Baustein namens „fire module“ verwendet, in dem das Eigangsvolumen zuerst komprimiert („squeeze“) und anschließend expandiert („expand“) wird. Im squeeze-Teil kommen ausschließlich 1×1–Convolutions zum Einsatz, im expand-Teil werden 1×1– und 3×3–Convolutions eingesetzt. Das implementierte Netz setzt sich aus mehreren dieser Bausteine zusammen (Tabelle 3.11).

Am Anfang kommt ein gewöhnlicher Convolutional Layer zum Einsatz. Das SqueezeNet verzichtet vollständig auf FC-Layer, was wesentlich zur Parametereinsparung beiträgt. Stattdessen wird ein Convolutional Layer mit anschließendem GAP-Layer verwendet.

SE-ResNeXt Abweichend zur Original-Veröffentlichung von SE-ResNeXt [81] wird ein flacheres Netz (29 statt 50 Blöcke) verwendet. Die Architektur ist angelehnt an ResNeXt-29 [74, S. 8] und knüpft nach Xie et al. [74] an [72] an. Sie besteht nach dem ersten Convolutional Layer aus drei Stufen mit jeweils drei Residual-Aggregaten (Tabelle 3.12). Ein Aggregat besteht wiederum aus vier Teilen: die ersten drei Teile (Convolutional Layer) bilden einen ResNeXt-Block („bottleneck template“) und der letzte ist der SE-Block. Für die 64 Feature Maps in einem ResNeXt-Block wird ein 16×4d Template gewählt, d. h. die Anzahl der Parallelzweige (Kardinalität C) beträgt 16 und die Convolutional Layer in jedem Zweig besitzen vier Filter. Eine höhere Kardinalität ist

Tabelle 3.11: Übersicht der implementierten SqueezeNet-Architektur. In allen Layern mit Aktivierungsfunktionen (außer Output) werden ReLU-Funktionen eingesetzt.

Layer	k @ e / s	$k_{s_{1x1}}$	$k_{e_{1x1}}$	$k_{e_{3x3}}$
Input				
Convolutional	96 @ $7 \times 7 \times d_{\text{in}}$ / 1			
Max-Pool	3×3 / 2			
Fire Modul		16	64	64
Fire Modul		16	64	64
Fire Modul		32	128	128
Max-Pool	3×3 / 2			
Fire Modul		32	128	128
Fire Modul		48	192	192
Fire Modul		48	192	192
Fire Modul		64	256	256
Max-Pool	3×3 / 2			
Fire Modul		64	256	256
Dropout				
Convolutional	4 @ $1 \times 1 \times 256$ / 1			
GAP				
Output				

Tabelle 3.12: Übersicht der implementierten SE-ResNeXt-Architektur. In allen Layern mit Aktivierungsfunktionen (außer Output und SE-Block) werden ReLU-Funktionen eingesetzt.

Layer	**Aufbau**
Input	
Conv	conv, 7×7, 64, stride 2
Max-Pool	Max-Pool, 3×3, stride 2
Aggregat	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \quad C = 16 \\ \text{conv}, 1 \times 1, 128 \\ \text{SE}, [8, 128] \end{bmatrix} \times 3$
Aggregat	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \quad C = 16 \\ \text{conv}, 1 \times 1, 256 \\ \text{SE}, [16, 256] \end{bmatrix} \times 3$
Aggregat	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \quad C = 16 \\ \text{conv}, 1 \times 1, 512 \\ \text{SE}, [32, 512] \end{bmatrix} \times 3$
GAP	
FC	4 Neuronen

gegenüber mehr Filtern zu bevorzugen [74, S. 8]. Die Angabe für den SE-Block betrifft die Anzahl der Neuronen in den FC-Layern. Downsampling erfolgt wie in [74, S. 5] durch einen Stride von $s = 2$ in jeder ersten 3×3-Convolution eines Aggregats. Die Vergrößerung der Tiefendimension wird für die entsprechenden Shortcut-Connections durch Zero-Padding kompensiert. Zudem ist die „Full Preactivation“ [76] (Abbildung 2.14) implementiert, d. h. die Nichtlinearität folgt dem BN-Layer.

3.5.3 Netze aus der Defektdiagnose

Neben den Netzen aus der Bildverarbeitung werden Netze implementiert, die erfolgreich zur Defektdiagnose eingesetzt wurden.

Verstraete Verstraete et al. [142] nutzen ein Netz, das die Prinzipien von VGGNet [71] aufgreift. Es werden ausschließlich 3×3-Convolutional Layer und Max-Pooling Layer eingesetzt, wobei sich nach jeder Pooling-Operation die Anzahl der Convolutional Kernel verdoppelt (Abbildung 3.10).

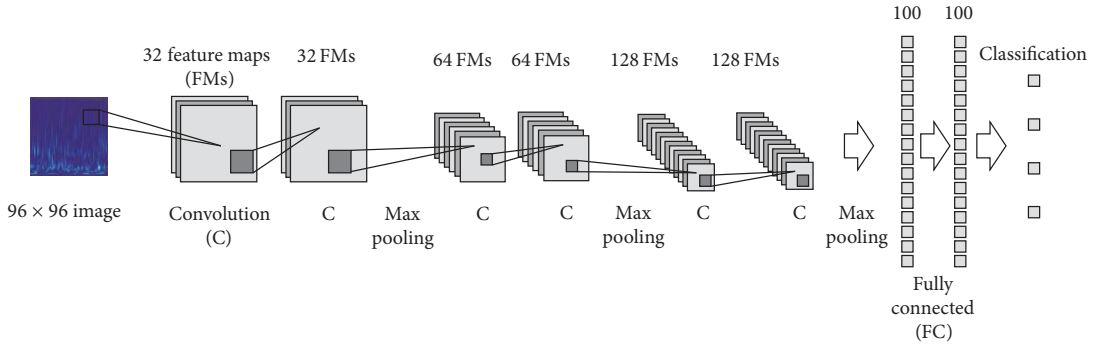


Abbildung 3.10: CNN-Architektur von Verstraete et al. [142, S. 7]

Das abgebildete Netz wird mit Dropout in beiden FC-Layern implementiert. Ansonsten folgt die Implementierung der dargestellten Architektur.

TICNN Das TICNN [139] stellt von den Netzen zur Defektdiagnose ein verhältnismäßig tiefes Netz dar, bei dem im herkömmlichen Stil Convolutional und Max-Pooling Layer alternieren (Abbildung A.6). In der Originalveröffentlichung werden eine Ensemble-Strategie sowie Dropout mit veränderbarer Dropout-Rate im ersten Convolutional Layer verwendet. Beides ist nicht implementiert. Die Verwendung von Ensembles bringt eine Vervielfachung der Komplexität, die nicht erwünscht ist und eine veränderbare Dropout-Rate muss durch Skalierung der Gewichte während des Trainings kompensiert werden. Die Idee von Dropout im ersten Convolutional Layer wird mit einer festen Dropout-Rate implementiert.
Die Tiefe des Netzes ist im Wesentlichen durch die Anzahl der Max-Pooling Layer begrenzt, denn jede 2×2–Pooling-Operation mit $s = 2$ halbiert die Dimension des Eingangsvolumens. Bei einer Sequenzlänge von 256 Datenpunkten, die durch eine FFT auf 128 reduziert wird, können maximal sechs solcher Max-Pooling Layer verwendet werden. Nach dem sechsten Subsampling Layer ist das Eingangsvolumen jedoch bereits auf eine Ausdehnung von zwei Datenpunkten minimiert. Gegenüber der Original-Veröffentlichung wird daher ein flacheres Netz mit vier Max-Pooling-Layern implementiert (Tabelle 3.13). Aufgrund der Netztiefe wird in [139] BN eingesetzt, weshalb in dieser Arbeit nach jeder Nichtlinearität ein BN-Layer folgt.

3.5.4 DamNet

Die als „DamperNet“ bezeichnete CNN-Architektur wird für diese Anwendung neu entworfen. Zur Abdeckung des Einflusses unterschiedlicher Filtergrößen wird ein neuartiger Baustein eingeführt, der an die Idee des Inception Moduls angelehnt ist: In mehreren Zweigen werden unterschiedlich große Convolutional Filter verwendet und in einem wird die Depthwise Separable Convolution eingesetzt (Abbildung 3.11). Dadurch entstehen in jedem Zweig unterschiedliche Feature Maps,

Tabelle 3.13: Übersicht der implementierten TICNN-Architektur. In allen Layern mit Aktivierungsfunktionen (außer Output) werden ReLU-Funktionen eingesetzt.

Layer	k @ e / s	**Padding**
Input		
Convolutional	16 @ $4 \times 4 \times d_{\text{in}}$ / 1	same
Batch Norm		
Dropout		
Max-Pool	2×2 / 2	valid
Convolutional	32 @ $3 \times 3 \times 16$ / 1	same
Batch Norm		
Max-Pool	2×2 / 2	valid
Convolutional	64 @ $3 \times 3 \times 32$ / 1	same
Batch Norm		
Max-Pool	2×2 / 2	valid
Convolutional	64 @ $3 \times 3 \times 64$ / 1	valid
Batch Norm		
Max-Pool	2×2 / 2	valid
FC	100 Neuronen	
Output (FC)	4 Neuronen	

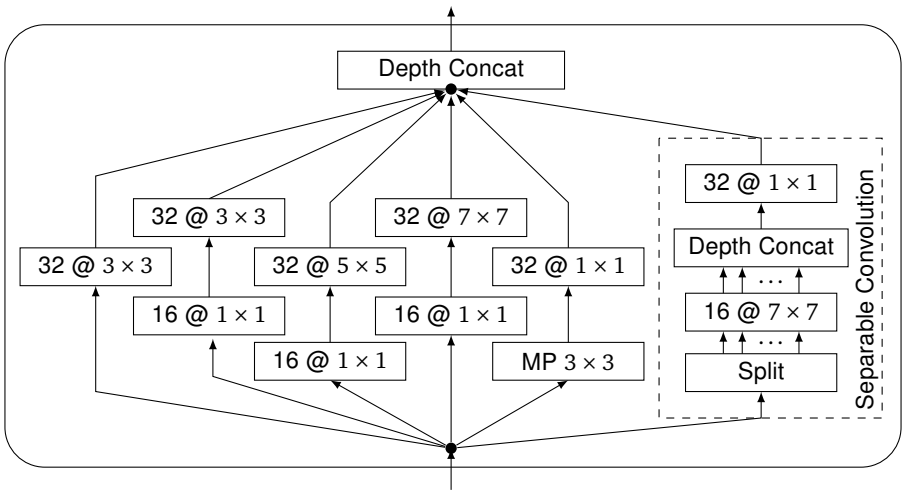


Abbildung 3.11: DamCeption Modul. In allen Layern (einschließlich Max-Pooling) wird ein Stride von $s = 1$ verwendet.

die einerseits durch gewöhnliche Faltungsoperationen mit unterschiedlich großen Filtermasken und andererseits durch die Extraktion von Features aus jedem Datenkanal einzeln und deren anschließende Kombination zustandekommen.

Der SE-Block [81] ermöglicht die Skalierung einzelner Feature-Maps durch einen Network-in-Network (NiN)-Ansatz [88]. Das interne MLP erlaubt es dem Netzwerk, eigenständig zu erlernen welche Feature Maps den größten Beitrag für eine optimale Lösung des Klassifikationsproblems leisten. Der SE-Block wird daher zusätzlich in das Netz eingebracht. An die Kombination des DamCeption-Layers mit dem SE-Block schließt sich ein Max-Pooling Layer an. Subsampling Layer verbessern die Leistung und reduzieren gleichzeitig die Anzahl der Parameter [86, S. 4].

Es folgt eine zweite Stufe von gewöhnlicher Convolution und Max-Pooling vor dem FC-Layer mit Dropout (Tabelle 3.14). Die Anzahl der Convolutional Kernel wird gegenüber den Convolutional Layern im DamCeption Layer verdoppelt und die Anzahl der Neuronen im FC-Layer identisch gewählt.

Tabelle 3.14: Übersicht der implementierten DamNet-Architektur. In allen Layern mit Aktivierungsfunktionen (außer Output und SE-Block) werden ReLU-Funktionen eingesetzt.

Layer	**Design**	**Padding**
Input		
DamLayer	(siehe Abbildung 3.11)	same
SE-Block	[192, 24]	
Max-Pool	2×2 / 2	valid
Convolutional	64 @ $3 \times 3 \times 192$ / 1	same
Max-Pool	2×2 / 2	valid
FC	64 Neuronen	
Dropout		
Output (FC)	4 Neuronen	

4 Ergebnisse, Optimierungen und Validierung

In diesem Kapitel werden die Ergebnisse der Hauptuntersuchungspakete aus Unterkapitel 3.4 präsentiert und diskutiert. Das Modell mit der Konfiguration, welche die besten Ergebnisse erzielt, wird hinsichtlich ausgewählter Aspekte optimiert. Im letzten Teil dieses Kapitels werden zur Validierung des vorgeschlagenen FDI-Systems verschiedene Untersuchungen anhand des neu eingefahrenen Datensatzes vorgenommen.

4.1 Ergebnisse der Hauptuntersuchungsaspekte

In diesem Unterkapitel wird in der Reihenfolge der Hauptuntersuchungsaspekte verfahren, um systematisch den Zustandsraum aller Modellkonfigurationen abzutasten und mit dem DD-Datensatz ein lokales Optimum hinsichtlich KKR zu finden.

4.1.1 Variation der Datenkanäle

Mit dem Ergebnis der Feinabstimmung aus Abschnitt 3.3.3 als Modell wird die Datenkanalvariation im Zeit- und Frequenzbereich durchgeführt. Erste Untersuchungen an den Ergebnissen im Frequenzbereich zeigen, dass L2-Regularisierung und Lernrate für die Darstellung im Frequenzbereich zu hoch sind. Dies äußert sich durch Divergenz in einigen Kreuzvalidierungs-Durchläufen, verrauschte Lernkurven sowie Trainingsgenauigkeiten unterhalb der Testgenauigkeiten. Abweichend zu den Standardparametern aus Tabelle 3.2 gelten – sofern nicht anders beschrieben – für die Ergebnisse dieses Kapitels daher folgende Hyperparameter:

- $\epsilon = 0.0003$
- $\lambda = 0.003$
- Maximale Epochenanzahl 650
- Early Stopping Geduld 50

Zur Demonstration der Verlässlichkeit der Ergebnisse im Zeitbereich werden beide Hyperparameterkonfigurationen aufgeführt (Tabelle 4.1). Grundsätzlich zeigen die Ergebnisse aller drei Parameter- bzw. Datenkonstellationen ähnliche Trends. Die Ergebnisse mit reduzierten Hyperparameter-Werten für den linearen Detrend sind überwiegend schlechter als mit hohem ϵ und λ. Dies ist auf die schwächere Regularisierung zurückzuführen. Die Datenrepräsentation im Frequenzbereich ist der im Zeitbereich überlegen, denn die Genauigkeiten liegen durchschnittlich $10,6\,\%$ ($\lambda = 0.01$) bzw. $13,3\%$ ($\lambda = 0.003$) über denen der anderen beiden Darstellungen (Abbildung A.7).

Tabelle 4.1: Ergebnisse der Datenkanal-Variation, fünffach kreuzvalidiert. Die Streuung wird aus Darstellungsgründen weggelassen.

Variante	n_{FL}	n_{FR}	n_{RL}	n_{RR}	a_X	a_Y	$\dot{\psi}$	δ	Test-Genauigkeit ($\overline{KKR}$)		
									(lin.) Detrend		DFT/FFT
									$\epsilon = 0.0010$ $\lambda = 0.0100$	$\epsilon = 0.0003$ $\lambda = 0.0030$	$\epsilon = 0.0003$ $\lambda = 0.0030$
0	x								28,41 %	31,92 %	43,41 %
1		x							31,05 %	29,71 %	41,49 %
2	x	x							37,54 %	35,44 %	48,91 %
3			x	x					39,54 %	39,84 %	46,85 %
4	x	x	x	x					49,40 %	47,26 %	58,05 %
5					x				30,21 %	30,46 %	38,18 %
6						x			36,67 %	31,03 %	44,09 %
7					x	x			39,06 %	34,74 %	46,66 %
8							x		27,34 %	30,91 %	45,87 %
9								x	24,87 %	25,26 %	27,35 %
10							x	x	32,79 %	37,11 %	47,02 %
11					x	x	x	x	44,22 %	36,51 %	52,78 %
12	x	x	x	x	x	x			53,31 %	46,51 %	62,30 %
13	x	x	x	x			x	x	53,55 %	49,95 %	64,76 %
14	x	x	x	x		x		x	51,84 %	48,62 %	61,28 %
15	x	x	x	x		x	x		55,74 %	49,46 %	66,91 %
16	x	x	x	x	x	x	x		54,03 %	46,27 %	68,04 %
17	x	x	x	x	x	x		x	50,15 %	45,66 %	61,85 %
18	x	x	x	x	x	x	x	x	51,66 %	43,84 %	67,61 %

Die Verwendung eines einzelnen Signals ist durchgehend der Kombination mehrerer Signale unterlegen. Der Vergleich zweier Raddrehzahlen an derselben Achse zeigt, dass die Defektposition nicht besser durch die Raddrehzahl an der entsprechenden Stelle erkannt wird (Varianten 0 – 1). Im Zeitbereich liegen die Ergebnisse bei Verwendung der Hinterachs-Signale oberhalb der bei Nutzung der Raddrehzahlen der Vorderachse. Die Kombination aller Raddrehzahlen zeigt eine deutliche Verbesserung gegenüber den achsweisen Kombinationen (3 – 4).
Wird nur die Längsbeschleunigung verwendet, liegen die Ergebnisse unterhalb denen bei alleiniger Verwendung der Querbeschleunigung. Die Kombination beider Signale führt nur zu einem leichten Anstieg der KKR gegenüber dem Einzelsignal (5 – 7). Die Nutzung des Lenkwinkels führt lediglich zum Erwartungswert bei einem Klassifikationsproblem mit vier Klassen (25 %), weshalb das Signal für weitere Untersuchungen ausgeschlossen werden kann (9). Vor allem im Frequenzbereich zeigen die Ergebnisse unter Nutzung der Gierrate dagegen gute Ergebnisse (8), die sich in Kombination mit dem Lenkwinkel (10) nur leicht verbessern. Die Kombination aller Signale ohne die Raddrehzahlen führt zu einem weiteren Anstieg der KKR, insbesondere im stark regularisierten Fall mit Daten im Zeitbereich (11).
Die Ergebnisse im letzten Block sind nach der Analyse der Einzelsignale plausibel. Mit der Auswahl wie Merk sie getroffen hat (12) werden gute Ergebnisse erzielt. Bei Hinzunahme des Lenkwinkels verschlechtern sich die Ergebnisse (14, 17, 18). Daher liegt auch die Variante mit geringer Redundanz (14) leicht unterhalb der Baseline (12). Wird stattdessen die Gierrate hinzugenommen (16), steigt die KKR. Die besten Ergebnisse werden durch Kombination der Raddrehzahlen mit den Beschleunigungen und der Gierrate erreicht (15, 16).

Eine Bewertung zur finalen Auswahl erfolgt durch Bildung des Mittelwerts über die Ergebnisse (Abbildung 4.1). Variante 15 erreicht gefolgt von 16 und 13 die höchste durchschnittliche KKR, während die restlichen Varianten aus dem untersten Block von Tabelle 4.1 als etwa gleich gut

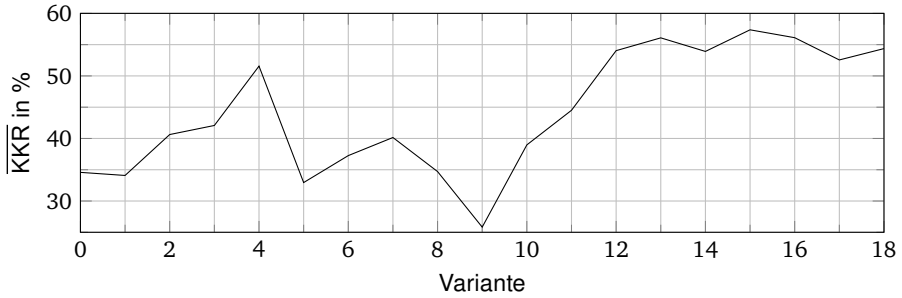


Abbildung 4.1: Gemittelte Ergebnisse der Datenkanal-Auswahl

bewertet werden können. Abgesehen von der Kombination aller vier Raddrehzahlen erreichen die anderen Varianten keine brauchbaren Ergebnisse. Variante 13 verwendet den Lenkwinkel, der jedoch keine nützliche Informationen zu enthalten scheint, weshalb diese ausgeschlossen wird. Variante 16 nutzt zusätzlich zu Variante 15 die Längsbeschleunigung. Insbesondere im Frequenzbereich scheint auf Basis des einzelnen Kanals a_{X} (Variante 5) eine Steigerung der KKR möglich zu sein. Als Erweiterung zu [5] wird daher für die folgenden Untersuchungen der Umfang der Signale auf $n_{\mathrm{FL}}, n_{\mathrm{FR}}, n_{\mathrm{RL}}, n_{\mathrm{RR}}, a_{\mathrm{X}}, a_{\mathrm{Y}}$ und $\dot{\psi}$ (Variante 16) festgelegt. Dies bedeutet eine höhere Datenmenge, die generell als vorteilhaft, hinsichtlich der Gesamtzahl der Parameter jedoch als nachteilig anzusehen ist.

4.1.2 Variation der Datenfusionsebene

Für dieses Untersuchungspaket wurden drei verschiedene Ebenen zur Datenfusion festgelegt:

- Rohdatenebene
- Merkmalsebene mittels gewöhnlicher Convolution
- Merkmalsebene mittels (Depthwise) Separable Convolution

Bei der Separable Convolution werden zusätzlich verschiedene Signalgruppierungen mit den verbleibenden Datenkanälen untersucht (Tabelle 4.2). Der erste Fall (Variante 0) bildet eine

Tabelle 4.2: Untersuchte Signalgruppierungen bei der gruppierten Depthwise Separable Convolution: Gruppen sind gleich eingefärbt und weiße Felder gelten als Einzelgruppen.

Variante	n_{FL}	n_{FR}	n_{RL}	n_{RR}	a_{X}	a_{Y}	$\dot{\psi}$
0							
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							

gewöhnliche „depthwise separable convolution“ ab, d. h. jedes Signal wird einzeln betrachtet und die extrahierten Merkmale werden anschließend durch die 1×1–Convolution fusioniert. Der Einfluss unterschiedlicher Kombinationen, z. B. das achsweise Zusammenfassen der Raddreh-

zahlen, lässt sich durch die restlichen Varianten bestimmen. Bei allen Versuchen wird die Anzahl der Filterkernel für die Pointwise (1×1) Convolution zu 64 gewählt. Dadurch unterscheiden sich die untersuchten Netze nur im ersten (und einzigen) Convolutional Layer und insbesondere die Neuronenanzahl zwischen Convolutional und Output Layer bleibt erhalten.
Die Anzahl der Kernel bei der Depthwise Convolution stellt einen Freiheitsgrad dar, der folgendermaßen gewählt wird: Jedes Signal erhält eine feste Anzahl von Filtern. Wird eine Gruppe mit mehreren Signalen gebildet, geht diese Anzahl additiv in die Gesamtzahl der Filter einer Gruppe ein. Wird bspw. $k_{\text{einzel}} = 8$ gewählt und eine Gruppe besteht aus 3 Signalen, besitzt diese Gruppe $k_i = 3 \cdot 8 = 24$ Filter.
Zusätzlich wird die Filteranzahl pro Datenkanal variiert, weil bereits bei der Rastersuche für die Hyperparameter des Baselinemodells ein Einfluss der Filteranzahl festgestellt wurde (Abbildung 4.2).

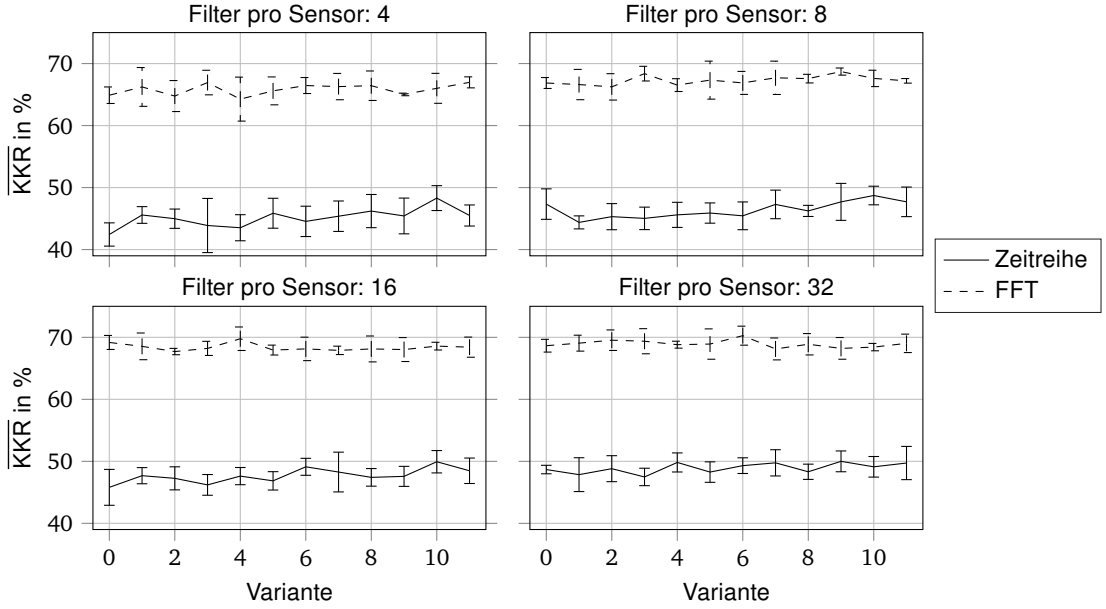


Abbildung 4.2: Ergebnisse der Datenfusionsebene mit gruppierter Depthwise Separable Convolution. Die Streuung bei der fünffachen Kreuzvalidierung ist als Fehlerbalken dargestellt.

Die erzielten Ergebnisse unterscheiden sich nur geringfügig voneinander. Jedoch kann wie in Abschnitt 4.1.1 durch Erhöhung der Filteranzahl eine Verbesserung der durchschnittlichen KKR erreicht werden. Das verdeutlicht insbesondere die Darstellung der Daten im Zeitbereich: Bei vier Filtern pro Sensor wird nur durch die Streuung bei Variante 10 eine KKR von über $50\,\%$ erreicht. Dagegen liegt bei mehreren Varianten mit 32 Filtern pro Sensor bereits der Durchschnitt bei diesem Wert. Im Frequenzbereich steigt die KKR ebenfalls mit zunehmender Filteranzahl leicht an.
Beim Training des Netzes können mit mehr Filtern mehr Merkmale aus den Signalen extrahiert werden. Liegt die Anzahl der Filter an der unteren Grenze, stehen möglicherweise weniger Informationen in Form von Merkmalen zur Verfügung. Für fundierte Aussagen ist eine Analyse der erlernten Merkmale in Abhängigkeit der Anzahl aller Merkmale erforderlich, die in einer weiterführenden Arbeit erfolgen sollte.

Für beide Datenrepräsentationen (Zeit- und Frequenzbereich) zeichnet sich keine Variante gegenüber allen anderen als herausragend ab. Der Vergleich von Variante 0 zu den übrigen Varianten aus Tabelle 4.2 legt die Vermutung nahe, dass eine Gruppierung von Signalen keinen offensichtlichen, positiven Einfluss hat. Im Vergleich zum gewöhnlichen Convolutional Layer ist

bei der (Depthwise) Separable Convolution nur im Frequenzbereich eine leichte Verbesserung zu erkennen (Tabelle 4.3).

Tabelle 4.3: Ergebnisse der Datenfusionsebene, fünffach kreuzvalidiert. Für Zeit- und Frequenzbereich gilt eine einheitliche Lernrate von $\epsilon = 0.0003$. Die Stärke der L2-Regularisierung beträgt $\lambda_{\text{Zeit}} = 0.008$ bzw. $\lambda_{\text{Frequenz}} = 0.003$. Die Implementierung des Layers für die gruppierte Separable Convolution wird durch einen Vergleich von Variante 0 und der TensorFlow-Implementierung (letzte Zeile) als tauglich angesehen.

Setup	$\overline{\textbf{KKR}}$	
	lin. Detrend	FFT
Rohdatenebene	$44,47 \pm 1.91\,\%$	$63,69 \pm 6,25\%$
Gewöhnliche Convolution	$50,99 \pm 1,51\%$	$67,07 \pm 1,63\%$
Separable Convolution (32 Filter pro Sensor, Variante 0)	$48,67 \pm 0,68\,\%$	$68,64 \pm 1,03\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 1)	$47,85 \pm 2,73\,\%$	$69,06 \pm 1,29\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 2)	$48,81 \pm 2,10\,\%$	$69,54 \pm 1,66\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 3)	$47,47 \pm 1,41\,\%$	$69,37 \pm 2,03\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 4)	$49,82 \pm 1,54\,\%$	$68,81 \pm 0,56\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 5)	$48,27 \pm 1,65\,\%$	$68,91 \pm 2,45\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 6)	$49,30 \pm 1,27\,\%$	$70,27 \pm 1,53\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 7)	$49,76 \pm 2,12\,\%$	$68,13 \pm 1,77\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 8)	$48,31 \pm 1,23\,\%$	$68,88 \pm 1,72\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 9)	$49,99 \pm 1,68\,\%$	$68,21 \pm 1,75\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 10)	$49,11 \pm 1,68\,\%$	$68,42 \pm 0,59\,\%$
Separable Convolution (32 Filter pro Sensor, Variante 11)	$49,72 \pm 2,68\,\%$	$69,03 \pm 1,49\,\%$
Gewöhnliche Convolution plus 1×1–Convolution	$50,31 \pm 1,32\,\%$	$68,67 \pm 1,84\,\%$
Separable Convolution (32 Filter pro Sensor, TensorFlow)	$51,52 \pm 2,51\,\%$	$68,44 \pm 0,92\,\%$

Die KKR bei Fusion auf Rohdatenebene liegt dagegen unterhalb der bei den übrigen Konstellationen, womit die Ergebnisse aus [148] nicht bestätigt werden. Für den Fall eines einzigen langen Vektors (Rohdatenfusion) wird jeder einzelne, flache Filter über den gesamten Vektor bewegt, um eine Feature Map zu berechnen. Dabei gelten die Gewichte des Filters für jedes Signal gleichermaßen, obwohl die Signale aus unterschiedlichen Domänen stammen. Dagegen erhält bei der Tiefenanordnung (gewöhnliche Convolution) jedes Signal ein eigenes Gewicht im tiefen Filter, was ein Vorteil zu sein scheint. Zudem gehen bei der Fusion auf Rohdatenebene temporale Zusammenhänge zwischen Signalen durch große Entfernungen zwischen den Datenpunkten verloren. Der Filter durchläuft die Signale im Vektor einzeln hintereinander, wodurch er die Datenpunkte jeder Sequenz asynchron sieht. Bei der gewöhnlichen Convolution oder der gruppierten Separable Convolution erfasst der Filter durch die Signal-Anordnung in der Tiefe alle Datenkanäle zu identischen Zeitpunkten, d. h. synchron. Die Ergebnisse deuten darauf hin, dass das synchrone Abtasten der Signale verbesserten Ergebnissen zuträglich ist.

Für einen Vergleich mit ähnlicher Netzkapazität zwischen gewöhnlichem Convolutional Layer und Separable Convolution wird das Baseline-Modell um eine 1×1–Convolution erweitert (vorletzte Zeile). Die Anzahl der Filter im ersten Convolutional Layer mit $d_{\text{in}} = c = 7$ wird zu $k_1 = 7 \cdot 32 = 224$ gewählt und die Filteranzahl für die zusätzliche Pointwise Convolution zu $k_2 = 64$. Die Ergebnisse zeigen unter Berücksichtigung der Streuung keine wahrnehmbare Verbesserung gegenüber der gewöhnlichen Convolution durch die gesteigerte Komplexität.
Zusätzlich wird untersucht, ob durch die Aneinanderreihung der Feature Maps in der Länge statt in der Tiefe oder das Verzichten auf die Pointwise Convolution Verbesserungen erzielt werden können, was sich empirisch nicht bestätigt. Die Aneinanderreihung der Feature Maps in der Länge führt zu leicht verringerten Klassifikationsgenauigkeiten. Der Verzicht auf die Pointwise Convolution wirkt sich negativ auf die Rechenzeit aus ohne die Ergebnisse zu verbessern.

Aufgrund der Tatsache, dass die Ergebnisse bei Fusion auf Rohdatenebene hinter den anderen zurückbleiben und die Anzahl der Netzparameter ca. um den Faktor Sieben ansteigt, wird diese Art der Kombination ausgeschlossen. Die Ergebnisse der beiden übrigen Fusionsebenen sind nahezu identisch, weshalb beide Fusionsarten weiter verfolgt werden. Viele CNN-Architekturen verwenden die gewöhnliche Convolution (Abschnitt 2.2.3 und 2.5). Sie wird für die Untersuchung verschiedener Architekturen sowie das Baseline-Modell als Standard verwendet. Für die Entwicklung eines neuen Netzes, das für den speziellen Anwendungsfall der Dämpferdefektdiagnose entworfen wird, wird die Separable Convolution aufgrund der positiven Ergebnisse weiter verfolgt.

4.1.3 Variation der Datenrepräsentation

Verschiedene Datenrepräsentationen werden anhand des Baseline-Modells miteinander verglichen. Weil bisher ausschließlich 1-D-Verfahren zum Einsatz gekommen sind, wird für die 2-D-Verfahren das Baseline-Modell erweitert bzw. ein 2-D-Modell erstellt. Die Parameter des 1-D-Modells werden direkt auf das 2-D-Modell übertragen: Die 64 Filterkernel des einzigen Convolutional Layers sind 9×9 (statt 9×1) Pixel groß. Es folgt ein Max-Pooling Layer mit Filtergröße $f_\mathrm{h} = f_\mathrm{w} = s = 2$ und ein FC-Layer mit 128 Neuronen.

Das 1-D- und 2-D-Baseline-Modell werden mit den unterschiedlich transformierten Daten trainiert (Tabelle 4.4). Durch die Konstellation aus Architektur und großer Datenmenge bei Rekurrenzplot und GAF ist es nicht möglich das Baseline-Modell mit den bisherigen Standard-Parametern auf einer Tesla P100 GPU (16 GB) zu trainieren. Zur Dimensionsreduktion wird daher (abweichend zu den Architekturdetails) für diese beiden Darstellungen der Stride im Convolutional Layer auf $s = 3$ erhöht und ein 4×4 Max-Pooling (statt 2×2) angewendet. Um sicherzustellen, dass die Ergebnisse nicht durch den im Verhältnis zu den Datenbildern kleinen Filterkernel negativ beeinflusst werden, wird dieser in einem weiteren Versuch auf $e = 32$ vergrößert. Zusätzlich wird für einen direkten Vergleich mit den Ergebnissen der anderen Datenrepräsentationen, die im Netz mit Standardparametern ($s = 1$, Max-Pooling mit $e = 2$) verwendet werden, das GAF auf 64×64 Pixel reduziert. Dadurch lässt sich das Baseline-Modell mit Standardparametern verwenden.

Tabelle 4.4: Ergebnisse der Variation der Datenrepräsentation, fünffach kreuzvalidiert. Für alle Versuche gilt $\epsilon = 0,0003$, $\lambda = 0,003$.

Vorverarbeitung	1-D	2-D	lin. Detrend	$\overline{\text{KKR}}$
Keine (Rohsignal)	x			$25,26 \pm 1,03\,\%$
Linearer Detrend	x		x	$47,20 \pm 2,03\,\%$
Standardisierung	x			$27,93 \pm 1,61\,\%$
Standardisierung	x		x	$44,39 \pm 2,31\,\%$
Skalierung $[0,1]$	x			$26,70 \pm 1,37\,\%$
Skalierung $[0,1]$	x		x	$25,26 \pm 1,08\,\%$
Skalierung $[-1,1]$	x			$26,70 \pm 2,09\,\%$
Skalierung $[-1,1]$	x		x	$47,24 \pm 1,29\,\%$
FFT	x		x	$65,94 \pm 1,99\,\%$
Graustufenbild		x	x	$24,32 \pm 0,38\,\%$
STFT		x	x	$73,43 \pm 1,21\,\%$
WPI		x	x	$43,48 \pm 2,52\,\%$
Rekurrenzplot		x	x	$39,49 \pm 2,26\,\%$
GAF ($s = 3$, 4×4 Max-Pooling)		x	x	$30,54 \pm 0,86\,\%$
GAF ($s = 3$, $e = 32$, 4×4 Max-Pooling)		x	x	$29,25 \pm 2,83\,\%$
GAF (64×64 Bild, Standardmodell)		x	x	$25,00 \pm 1,08\,\%$

Die Ergebnisse deuten darauf hin, dass auf Basis des Zeitsignals KKR von über 40 % nur mit vorherigem (linearem) Detrend möglich sind. Die sonstige Vorverarbeitung spielt eine untergeordnete Rolle, da die übrigen Verfahren ähnliche Ergebnisse erzielen. Ohne linearen Detrend gibt das FDI-System nur den Erwartungswert (ca. 25 %) aus, d. h. es rät, während es mit Detrend eine gemittelte KKR von ca. 45 % erreicht. Einzig bei der Skalierung auf den Wertebereich $[0, 1]$ sowie für die Darstellung als Graustufenbild wurde keine Konvergenz beim Training erreicht. Dafür wurden über die angegebene Konfiguration aus Lernrate und L2-Regularisierungsstärke hinaus verschiedene Konfigurationen getestet.
Durch den linearen Detrend werden Informationen über die Amplitude aus dem Signal entfernt, was der Güte des Systems zuträglich zu sein scheint. Die Frequenzinformation ist für die KKR ausschlaggebend, denn FFT und STFT führen mit Abstand zu den besten Ergebnissen. Das WPI schneidet vergleichbar gut wie die 1-D-Verfahren mit linearem Detrend ab, benötigt jedoch mehrere Stunden für das Preprocessing. Die Darstellung im Phasenraum durch einen Rekurrenzplot oder ein GAF liefert unabhängig von der Bild- und Filtergröße ebenfalls keine vielversprechenden Ergebnisse. Obwohl das Netz mit den Trainingsdaten Konvergenzverhalten bzw. einen Hang zum Overfitting zeigt, bleibt die KKR anhand der Testdaten unter 40 %.

Die Umsetzung eines End-to-End-Systems ist auf Basis der obigen Ergebnisse nicht zielführend. Die durchgeführten Untersuchungen zeigen, dass eine Vorverarbeitung der Daten in der vorliegenden Anwendung erforderlich ist. Aufgrund der klaren Überlegenheit der Frequenz-basierten Vorverarbeitungsmethoden werden nur diese weiterverfolgt. Obwohl die STFT die besten Ergebnisse liefert, wird die FFT für weitere Versuche nicht ausgeschlossen. Die Entscheidung ist auf die bessere Performance hinsichtlich Preprocessing (schneller), Rechenzeit (kürzer) und Speicherbedarf (geringer) des 1-D-Netzes mit FFT zurückzuführen, weshalb eine geringere KKR vertretbar ist.

4.1.4 Variation der Netzwerkarchitektur

Im ersten Teil dieses Untersuchungsaspekts wird die Übertragbarkeit unterschiedlicher CNN auf die Anwendung der Dämpferdefektdiagnose untersucht. Aufgrund der stark unterschiedlichen Netztopologien und -hyperparameter wird im zweiten Teil die systematische Variation von Modell-Hyperparametern im Baseline-Modell untersucht, um Erkenntnisse über Einflüsse wie die Netztiefe zu gewinnen. Es kommt jeweils eine 1-D- und eine 2-D-Variante zum Einsatz.

Übertragbarkeit aus anderen Anwendungen

Es zeigt sich, dass sich die tiefen Netze aus der Bildverarbeitung nicht gut auf diese Anwendung übertragen lassen (Abbildung 4.3, Tabelle A.2). Das SE-ResNeXt erreicht zwar nach wenigen Epochen eine Trainingsgenauigkeit von über $95\,\%$ (Overfitting), die KKR bleibt jedoch nur knapp oberhalb des Erwartungswerts zurück. Beim SqueezeNet werden ausschließlich Filterkernel bis zu einem spatial extent von $e = 3$ verwendet, was zu einer maximalen Genauigkeit von $48,1\,\%$ führt. Inwiefern das Ergebnis mit der Filtergröße zusammenhängen könnte, wird in der Folgeuntersuchung analysiert.
Das flache LeNet dagegen erreicht im 2-D-Fall eine durchschnittliche Genauigkeit von $73,3\,\%$ und schneidet damit im Vergleich gut ab. Wird wie in der Original-Veröffentlichung Mean-Pooling verwendet, verschlechtert sich das Ergebnis allerdings. Auf den 1-D-Fall lässt sich das LeNet, welches aus der Bildverarbeitung stammt, bedingt übertragen (KKR ca. $60\,\%$).
Die Architektur von Verstraete et al. [142] eignet sich primär für den 1-D-Fall, obwohl es in der Original-Veröffentlichung mit 2-D-Daten verwendet wird. In der 1-D-Anwendung scheint die

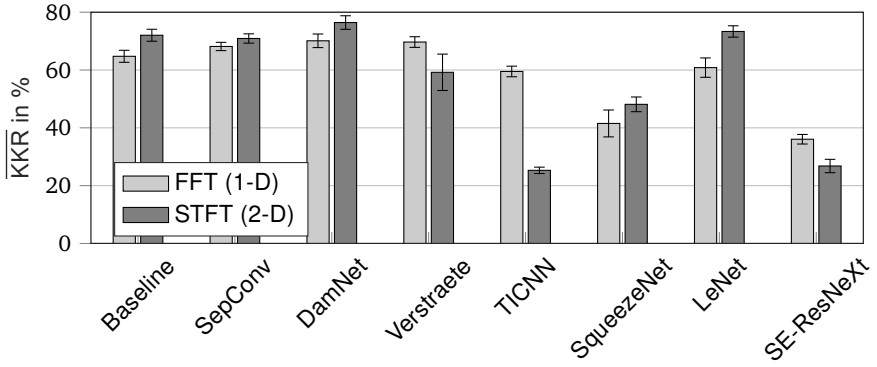


Abbildung 4.3: Ergebnisse der Architektur-Variation, fünffach kreuzvalidiert. Die Streuung bei der fünffachen Kreuzvalidierung ist als Fehlerbalken dargestellt. „SepConv“ bezeichnet das Baseline-Modell mit Depthwise Separable Convolution wie es in Abschnitt 4.1.2 verwendet wurde.

Verwendung zweier aufeinanderfolgender Convolutions vorteilhaft zu sein, da sie in DamNet und Verstraete-Modell verwendet wird, die das Feld anführen. Dieser Aspekt wird ebenfalls in der Folgeuntersuchung analysiert. Das TICNN lässt sich nur für den 1-D-Fall auf die Anwendung übertragen, im 2-D-Fall zeigt sich ein Verhalten wie beim SE-ResNeXt. Die Verringerung der Mini-Batch-Größe (ähnlich zur ursprünglichen Veröffentlichung [139]) bringt nur eine minimale Verbesserung.

Baseline-Modell, SepConv und DamNet zeigen für beide Datenrepräsentationen ähnlich gute Ergebnisse. Das DamNet zeigt von allen Netzen die beste durchschnittliche KKR in beiden Datenrepräsentationen bei gleichzeitig weniger Parametern (Tabelle 4.5).

Tabelle 4.5: Gerundete Parameteranzahl der implementierten Architekturen in Tausend bei einer ursprünglichen Segmentlänge von 256 Datenpunkten. Im 1-D-Fall wird die FFT angewendet (Halbierung der Segmentlänge), im 2-D-Fall die STFT (32×32-Bild).

Netz	**1-D**	**2-D**
Baseline	529	2134
SepConv	541	2130
DamNet (vor Optimierung)	191	437
DamNet (nach Optimierung)	187	376
Verstraete	312	415
TICNN	66	69
SqueezeNet	359	756
LeNet	69	83
SE-ResNeXt	538	570

Während bei dem Modell von Verstraete und dem SqueezeNet die Hyperparameter für das Netztraining angepasst werden mussten (Tabelle A.2), kann das DamNet ohne Anpassung für beide Datenrepräsentationen verwendet werden; es ist robuster. Für ein FDI-System wird daher das DamNet vorgeschlagen. Die Folgeuntersuchung zu architektonischen Einflüssen der Netztiefe erfolgt jedoch am Baseline-Modell, das durch seine Simplizität einen einfacheren Vergleich zulässt.

Architektonische Einflussuntersuchung mit dem Baseline-Modell

In zwei Stufen wird eine Rastersuche über die Paramter Filteranzahl, Filtergröße, Anzahl der Blöcke (Conv + Max-Pool) und Anzahl der Convolutional Layer pro Block vorgenommen. Der erste

Convolutional Layer spielt im CNN eine besondere Rolle, da dort die Rohdaten erstmalig transformiert werden. Die erste Abstimmung des Baseline-Modells und dessen ersten Convolutional Layers (Abschnitt 3.3.3) wurde auf Basis der Rohdaten (mit lin. Detrend) vorgenommen. Der Ansatz eines Rohdaten-basierten Systems wurde allerdings verworfen, weshalb die gewählten Architektur-Hyperparameter möglicherweise nicht optimal für Daten im Frequenzbereich geeignet sind. Daher erfolgt eine erneute Untersuchung der Einflüsse Filtergröße und Filteranzahl (Abbildung 4.4 und 4.5). Die Verwendung identischer Netzparameter für den 1-D- und 2-D-Fall ist dabei nicht mehr sinnvoll, da die Netze grundsätzlich unterschiedliche Hyperparameter für eine möglichst optimale Lösung des Klassifikationsproblems benötigen könnten.

Erster Convolutional Layer im 1-D-Fall Im ersten Schritt der Rastersuche werden Filtergröße und -anzahl variiert (Abbildung 4.4). Die Verwendung von Filtern mit $e = 1$ führt zu einer

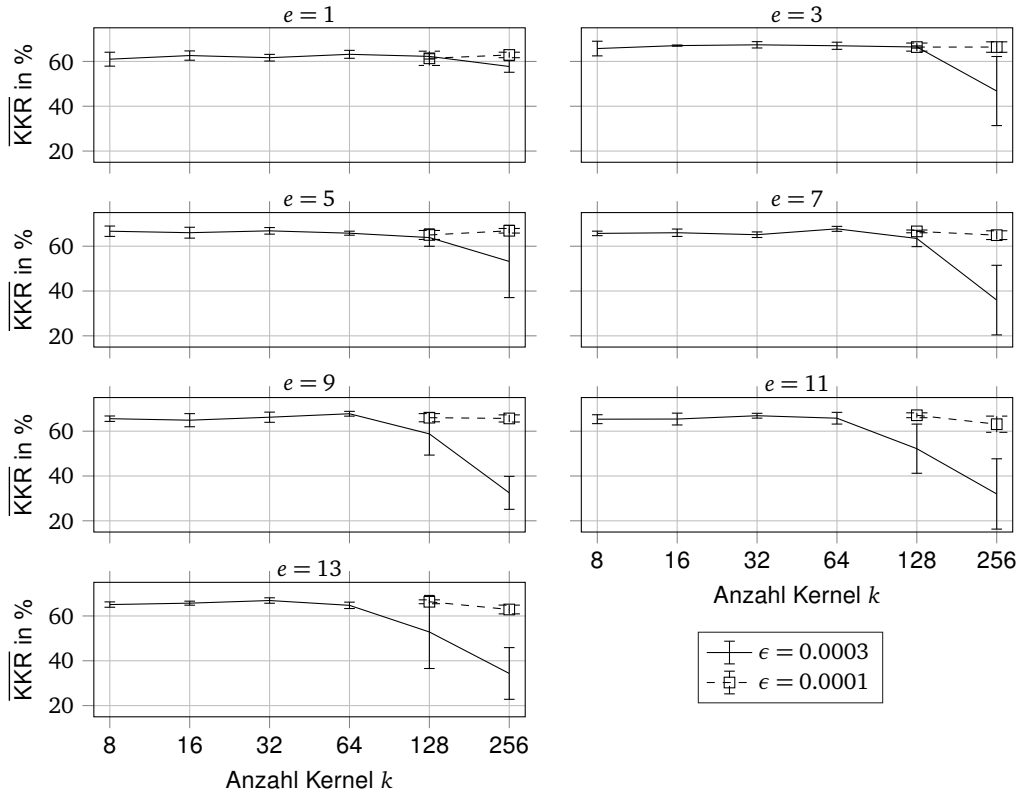


Abbildung 4.4: Ergebnisse der architektonischen Einflussuntersuchung anhand des 1-D-Baseline-Modells, fünffach kreuzvalidiert. Die Streuung bei der fünffachen Kreuzvalidierung ist als Fehlerbalken dargestellt.

leichten Verschlechterung der KKR. Abgesehen davon ist im 1-D-Fall die Wahl der Filtergröße unerheblich. Die KKR liegt bis $e = 13$ in einem gemeinsamen Bereich. Einzig für eine hohe Anzahl an Filterkerneln verschlechtert sich die KKR bei gleichbleibender Lernrate deutlich. Das Training divergiert in mindestens einem der fünf Durchläufe. Daher werden die Netze mit 128 und 256 Filtern erneut unter Verwendung einer geringeren Lernrate trainiert (Abbildung 4.4, gestrichelt). In allen Fällen kann dadurch Konvergenz sichergestellt werden. Die KKR ist jedoch bei angepasster Lernrate nach wie vor auf dem Niveau der Ergebnisse mit geringerer Filteranzahl. Das 1-D-Netz ist bei Variation von Filtergröße und -anzahl robust, denn Änderungen wirken sich kaum auf die KKR aus.

Mit steigendem k und e wächst die Anzahl der trainierbaren Parameter und damit die Rechenzeit.

Für die Untersuchung der Netztiefe werden die beiden Architektur-Hyperparameter daher zu $e = 3$ und $k = 16$ gewählt. Dieser Fall weist zusätzlich eine geringe Streuung von $0,26\,\%$ auf.

Erster Convolutional Layer im 2-D-Fall Beim 2-D-Fall kann über den Einfluss der Filteranzahl keine einfache Aussage getroffen werden. Für geringe Filtergrößen führt ein hohes k zu einer Verschlechterung der KKR, während bei großen Filtern das Gegenteil der Fall ist (Abbildung 4.5). Bei $k = 8$ zeigt sich für fast alle Filtergrößen eine leichte Verschlechterung der

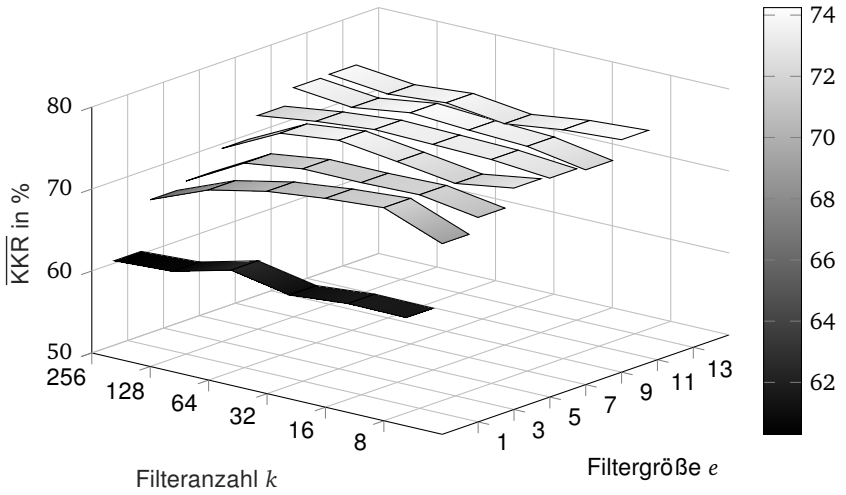


Abbildung 4.5: Ergebnisse der architektonischen Einflussuntersuchung anhand des 2-D-Baseline-Modells, fünffach kreuzvalidiert.

Ergebnisse gegenüber den Fällen $k \in \{16, 32, 64\}$, die aufgrund der Streuung jedoch nicht als signifikant zu werten ist.

Es kann allerdings ein positiver Einfluss mit steigender Filtergröße wahrgenommen werden. Die einzelnen Bänder in Abbildung 4.5 liegen auf unterschiedlichen Niveaus. Wie im 1-D-Fall führt ein Spatial Extent von $e = 1$ im 2-D-Fall ebenfalls zu der geringsten KKR. Das beste Ergebnis wird mit wenigen ($k = 8$) und großen ($e = 13$) Filtern erreicht.
Weil das erreichte Optimum am Rand des betrachteten Parameterraums liegt, wird eine zusätzliche Erweiterung in diese Richtung vorgenommen (Abbildung A.8). Lee et al. [149, S. 196] stellen in ihrer Anwendung fest, dass eine geringe Filteranzahl ($k < 5$) zu signifikant schlechteren Ergebnissen führt, während eine Verzehnfachung auf $k = 50$ keine nennenswerte Verbesserung bringt. Der gleiche Effekt kann im untersuchten 2-D-Netz beobachtet werden. Im Bereich $16 < k < 128$ unterscheiden sich die Ergebnisse je nach Filtergröße nur geringfügig. Eine weitere Senkung der Filteranzahl auf $k < 8$ führt zu einer Verschlechterung der Ergebnisse. Die Ergebnisse für nur zwei Filterkernel sind dennoch verhältnismäßig gut ($\overline{\text{KKR}} > 67\,\%$). Damit sind wenige große Filter im 2-D-Fall besser als viele kleine.
Eine weitere Vergrößerung der Filtermaske führt unter Berücksichtigung der Streuung zu keiner signifikanten Verbesserung. Die Ergebnisse sind jeweils vergleichbar. Wie im 1-D-Fall steigt die Anzahl der trainierbaren Parameter mit k und e, weshalb diese für den weiteren Verlauf zu $k = 16$ und $e = 11$ gewählt werden. Damit wird gegenüber $k = 8$ eine zusätzliche Kapazität erreicht, die für größere Datensätze notwendig ist.

Netzarchitektur für die Tiefen-Analyse Nach der Abstimmung des ersten Convolutional Layers wird die Netzarchitektur hinsichtlich Tiefe variiert. Buduma, Locascio [36, S. 100] behaup-

ten, dass durch das Stapeln mehrerer Convolutional Layer vor einem Pooling Layer gehaltvollere Repräsentationen erreicht werden. Verstraete et al. [142] bestätigen diese These in einer Diagnose-Anwendung mit Wälzlagern. Der vorherige Teil dieses Kapitels deutet ebenfalls darauf hin, dass zumindest im 1-D-Falle die konsekutive Anwendung von Convolutional Layern vorteilhaft ist. Für eine belastbarere Aussage wird der Einfluss der Block-Tiefe anhand des Baseline-Modells untersucht. Ein Block bezeichnet die Kombination aus einem bis mehreren Convolutional Layern mit einem Max-Pooling Layer.

Durch den Max-Pooling Layer halbiert sich die Dimension der Feature Maps. Um den Informationsverlust zu kompensieren, wird die Anzahl der Filter Kernel wie im VGGNet [71] nach jedem Block verdoppelt. Jeder Convolutional Layer eines Blocks besitzt die gleiche Anzahl von Filtern und verwendet Same-Padding. Die Größe der Filtermaske wird (abgesehen vom ersten Conv Layer) für 1-D- und 2-D-Netz zu $e = 3$ gewählt. Die Einflussuntersuchung der Filtergröße außerhalb des ersten Convolutional Layers wird für eine anschließende Arbeit vorgeschlagen. Wie bisher werden ReLU-Aktivierungsfunktionen und keine BN verwendet.

Netztiefe im 1-D-Fall Dong et al. [13] empfehlen, dass flache Strukturen durch tiefe ersetzt werden sollten, da Letztere komplexe Strukturen und eine hohe Kapazität besitzen. Zudem sind tiefe Netzwerke nach Janssens et al. [18, S. 340] robuster gegenüber Variation innerhalb der Daten. Es werden daher einereseits die Block-Tiefe und andererseits die Netz-Tiefe variiert, um Aussagen über deren Einflüsse zu gewinnen (Abbildung 4.6).

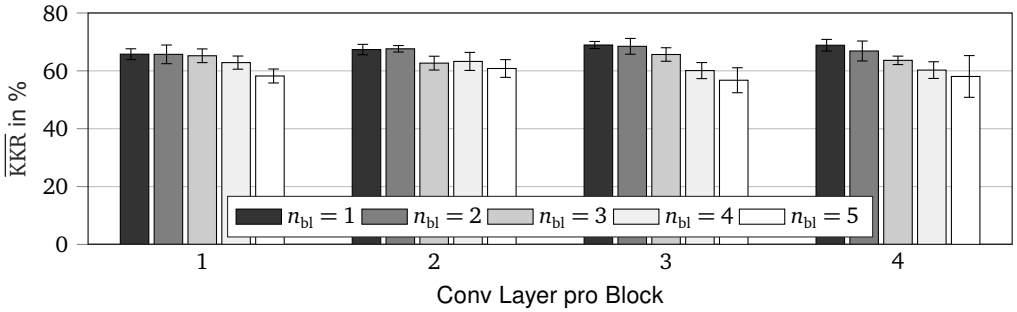


Abbildung 4.6: Ergebnisse der Tiefeneinfluss-Untersuchung anhand des 1-D-Baseline-Modells, fünffach kreuzvalidiert. Die Streuung bei der fünffachen Kreuzvalidierung ist als Fehlerbalken dargestellt. n_{bl} bezeichnet die Anzahl der Blöcke aus Convolution und Max-Pooling.

Es zeigt sich, dass die durchschnittliche Klassifikationsgenauigkeit beim 1-D-Baseline-Modell mit zunehmender Anzahl an Blöcken sinkt (Treppenstufen im Plot). Dies ist möglicherweise auf den Informationsverlust durch die vermehrte Anzahl an Max-Pooling Layern zurückzuführen. Das Hinzufügen von Convolutional Layern zu einem Block beeinflusst nur geringfügig die KKR (Abbildung A.9): für kleine Blockanzahlen $n_{\text{bl}} < 3$ zeigt sich ein leichter Aufwärtstrend bis zu einer Blocktiefe von drei Convolutional Layern. In allen Fällen streuen die Ergebnisse für unterschiedlich viele Convolutional Layer mit ca. $\pm 1,5\,\%$ um jeweils einen Wert für eine feste Blockanzahl. Diese Streuung ist in den meisten Fällen kleiner als die Standardabweichung der Kreuzvalidierung bei den Einzelversuchen. Daher kann keine robuste Aussage über einen positiven oder negativen Einfluss der Blocktiefe getroffen werden.

Netztiefe im 2-D-Fall Im 2-D-Fall zeigen sich ähnliche Tendenzen wie für den 1-D-Fall. Insbesondere für mehr als einen Convolutional Layer pro Block zeigt sich ein negativer Einfluss

steigender Blockanzahlen. Dies ist entweder auf die zunehmende Größenordnung der Nichtlinearität (mehr Aktivierungsfunktionen) oder die starke Dimensionsreduktion durch Max-Pooling zurückzuführen. Eine genaue Bestimmung der Ursache sollte in einer weiteren Arbeit erfolgen. Für $n_{\mathrm{bl}} = 5$ stellt sich bei drei und vier Conv Layern pro Block in mehreren Durchläufen der Kreuzvalidierung keine Konvergenz ein. Die Robustheit wird demnach mit mehr Blöcken negativ beeinflusst.

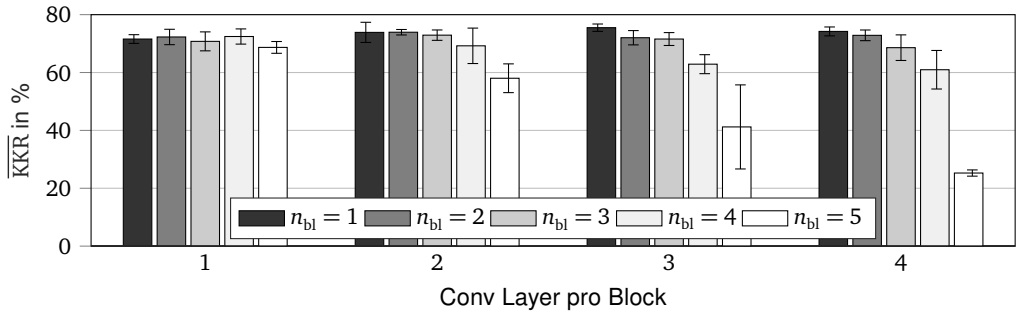


Abbildung 4.7: Ergebnisse der Tiefeneinfluss-Untersuchung anhand des 2-D-Baseline-Modells, fünffach kreuzvalidiert. n_{bl} bezeichnet die Anzahl der Blöcke aus Convolution und Max-Pooling. Die Streuung bei der fünffachen Kreuzvalidierung ist als Fehlerbalken dargestellt. Die Ergebnisse wurden mit $\epsilon = 0.0001$ bestimmt.

Das Stapeln von bis zu drei Convolutional Layern in den Blöcken wirkt sich bei $n_{\mathrm{bl}} < 4$ marginal positiv aus (Abbildung A.10). Für $n_{\mathrm{bl}} \geq 4$ zeigt sich dagegen ein Negativtrend mit zunehmender Anzahl an Convoltional Layern pro Block. Damit bestätigt sich das Verhalten, das bereits beim 1-D-Fall beobachtet wurde. Eine (flache) Architektur mit ein bis zwei Blöcken und jeweils ein bis drei Convolutional Layern zeigt in dieser Anwendung die besten Ergebnisse, während eine hohe Blockanzahl und mehr als drei Convolutional Layer pro Block zu einer verringerten KKR führen.

Mit dieser Aussage sind ebenfalls die Ergebnisse der Architektur-Übertragung aus anderen Anwendungen plausibel. Die tiefen Netze (SE-ResNeXt, SqueezeNet, TICNN) zeigen die schlechtesten Ergebnisse, während flache Netze besser abschneiden. Das gute Ergebnis des LeNet ist ebenfalls nachvollziehbar. Für 2-D-Daten kann ein positiver Einfluss großer Filtermasken im ersten Convolutional Layer festgestellt werden. Im DamNet werden die Erkenntnisse bereits teilweise umgesetzt (zwei Blöcke, zwei konsekutive Convolutions im ersten Block), was möglicherweise die Spitzenposition im Vergleich zu den übrigen Ergebnissen begründet.

Es wird ersichtlich, dass eine Mindestanzahl von Filtern gefordert werden sollte, da zu wenige Filter eine Verschlechterung der Ergebnisse nach sich ziehen. Eine Begrenzung nach oben scheint bei Anpassung der Hyperparameter nicht notwendig zu sein, ist insbesondere zur Reduzierung der Rechenzeit jedoch zu empfehlen. Bei Beachtung der Tatsache, dass das Eingangsbild im Verhältnis zur Filtergröße „nur“ 32×32 Pixel misst, ist eine Untersuchung des Padding-Verhaltens im ersten Convolutional Layer in Erwägung zu ziehen. In allen bisherigen Untersuchung wurde Same-Padding verwendet. Der Einfluss des Wechsels auf Valid-Padding beeinflusst möglicherweise die Wahl der Filtergröße, weil die Größe des vom Layer ausgegebenen Volumens verkleinert wird.

4.1.5 Variation der Datenmenge

Bei diesem Untersuchungsaspekt wird eine Prognose der benötigten Datenmenge aus dem Ist-Zustand abgeleitet. Davon ausgehend werden neue Daten eingefahren und die Vorhersage überprüft. Um die Hauptuntersuchungsaspekte abzuschließen, wird in diesem Abschnitt nur auf

die Überprüfung der Prognose eingegangen. Das Vorgehen zur Datenerhebung sowie weitere Untersuchungen folgen in Unterkapitel 4.3.

Ist-Zustand und Prognose Zur Abschätzung der benötigten Datenmenge für ein gefordertes Klassifikationsniveau wird die KKR des bestehenden Systems (DamNet mit FFT bzw. STFT) auf Basis verschiedener Datensatzgrößen untersucht. Die Anzahl der Trainingsbeispiele wird von 128 (Größe eines Mini-Batches, d. h. eine Epoche besteht aus einer einzigen Iteration) exponentiell mit Basis zwei bis 2048 variiert (Abbildung 4.8).

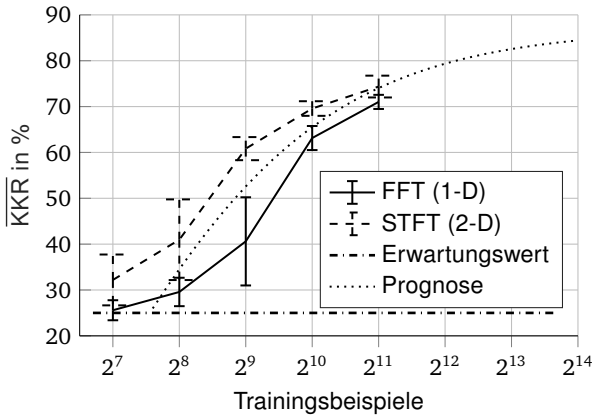


Abbildung 4.8: Untersuchung der Klassifikationsgenauigkeit in Abhängigkeit der Trainingsdatenmenge (DD-Datensatz). Die Ergebnisse sind zehnfach kreuzvalidiert. Jeder Schritt auf der horizontalen Achse bedeutet eine Verdoppelung der Datenmenge. Die maximal mögliche Anzahl beträgt bei einer Potenz von zwei $2^{11} = 2048$ Trainingsbeispiele, da der DD-Datensatz insgesamt mit einer Sequenzlänge von 256 Datenpunkten 3300 Datenbeispiele enthält, wovon 660 (20 %) als Testdaten verwendet werden.

Beide Datenrepräsentationen zeigen ein sehr ähnliches Verhalten bei Variation der Datenmenge. Mit zunehmender Anzahl der Trainingsbeispiele steigt die KKR, was für den positiven Effekt größerer Datenmengen spricht. Gleichzeitig verringert sich tendenziell die Streuung über die Durchläufe der 10-fachen Kreuzvalidierung. Bei sehr wenigen Trainingsbeispielen liegt die KKR dagegen nur knapp oberhalb des Erwartungswerts. Auf Basis der vorhandenen Datenpunkte wird eine $\tanh$-Funktion zur Extrapolation (Prognose) angepasst. Wird ein Niveau des FDI-Systems von über 80 % KKR gefordert, sind nach der Prognose mindestens 2^{13} Trainingsbeispiele erforderlich. Das bedeutet eine Vervierfachung des Datenbestands auf $\frac{5}{4}\frac{8192 \cdot 256}{50\text{Hz}} \approx 14,56\,\text{h}$ verwertbare Messdaten, wenn ein Anteil von 20 % Testdaten verwendet wird.

Überprüfung der Prognose Mit dem neuen Datensatz „DD2“ werden erneut Lernkurven in Abhängigkeit der Trainingsdatenmenge erstellt. Dadurch lässt sich überprüfen, ob mit einer ähnlich großen Datenmenge wie beim DD-Datensatz vergleichbare Ergebnisse erreicht werden. Gleichzeitig erlaubt der größere Datensatz die Überprüfung der Prognose (Abbildung 4.9).

Die Ergebnisse mit dem DD2-Datensatz sind durchgehend besser als mit dem DD-Datensatz und die Prognose wird erfüllt. Die KKR vom 1-D-DamNet mit DD2-Daten zeigt große Übereinstimmung mit den Ergebnissen des 2-D-Netzes beim DD-Datensatz. Das grundsätzlich bessere Ergebnis ist möglicherweise auf das Fehlen eines Streckenabschnitts im DD2-Datensatz zurückzuführen (Abschnitt 4.3.1). Hierfür wird eine tiefergehende Analyse empfohlen.

Weiterhin zeigt sich, dass das geforderte Niveau von über 80 % bei Vervierfachung des Datenbestands sowohl mit 1-D- als auch 2-D-Datenrepräsentationen erreicht wird. Mit der STFT werden

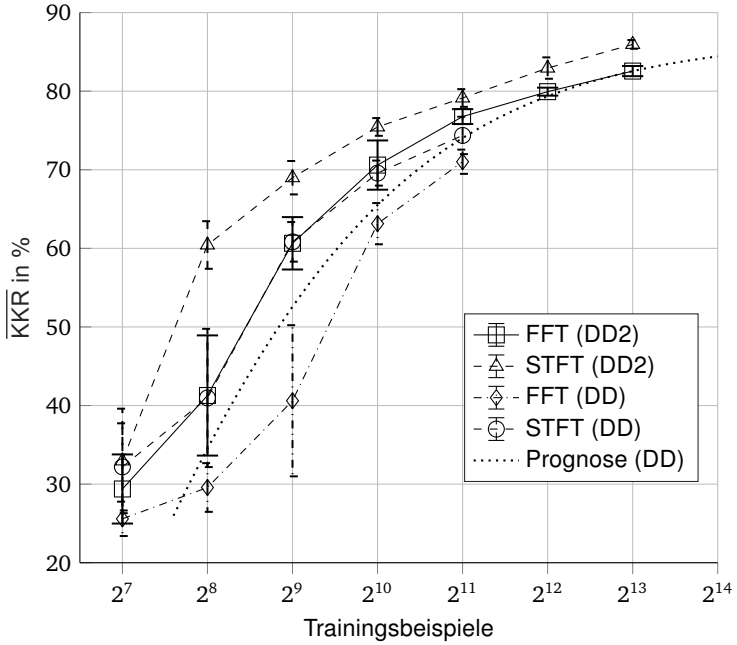


Abbildung 4.9: Untersuchung der Klassifikationsgenauigkeit in Abhängigkeit der Trainingsdatenmenge (DD- und DD2-Datensatz). Die Ergebnisse sind zehnfach kreuzvalidiert. Jeder Schritt auf der horizontalen Achse bedeutet eine Verdoppelung der Datenmenge.

wie zuvor durchgehend bessere Ergebnisse erzielt als mit der FFT. Die Erhaltung des zeitlichen Zusammenhangs, der bei der FFT verloren geht, ist hierfür möglicherweise ausschlaggebend. Allerdings ist der Rechenaufwand bei den 2-D-Netzen deutlich höher. Bei 2^{13} Trainingsbeispielen (DD2) beträgt die durchschnittliche Trainingsdauer auf einer Tesla P100 (16 GB) im 2-D-Fall $17,7\,\mathrm{min}$, während im 1-D-Fall nur $2,6\,\mathrm{min}$ benötigt werden. Mit der zunehmenden Datenmenge steigt die KKR und die Streuung sinkt. Auf Basis dieser Ergebnisse ist eine größere Datenmenge grundsätzlich zu bevorzugen.

4.2 Optimierungen

Nachdem in den Hauptuntersuchungsaspekten primär Daten- und Architektureinflüsse untersucht wurden, wird in diesem Abschnitt auf Optimierungen eingegangen. Diese beeinflussen das Netz oder das Framework des Netztrainings. Die einzelnen Optimierungen werden für das recheneffizientere 1-D-DamNet durchgeführt, was mehr Untersuchungen in kürzerer Zeit erlaubt. Anschließend werden die Einzeloptimierungen kombiniert. Die Übertragung der Optimierungen vom 1-D- auf den 2-D-Fall wird im letzten Schritt gesammelt vorgenommen. Die Untersuchung der einzelnen Optimierungen erfolgt jeweils ausgehend vom DamNet aus Abschnitt 3.5.4 mit den aktualisierten Standardparametern (Abschnitt 4.1.1).

4.2.1 Optimierungen im Netz

In der Literatur werden verschiedene Optimierungen wie unterschiedliche Aktivierungsfunktionen [155, 177–181] und Normalization Layer [89, 90, 182, 183] diskutiert. In diesem Abschnitt wird untersucht, ob solche Ansätze, die sich direkt auf das Netz auswirken, gewinnbringend in die bestehende Architektur eingebracht werden können.

Aktivierungsfunktion In den ersten KNN wurde eine logistische Sigmoid-Funktion als Aktivierung verwendet. Diese bringt wie der Tangens Hyperbolicus die Gefahr verschwindender Gradienten mit sich. Die ReLU-Funktion besitzt dieses Problem nicht. Daher existieren diverse Erweiterungen der ReLU-Funktion. Die ursprünglichen Funktionen werden ebenso wie einige Erweiterungen aus der ReLU-Familie im 1-D-DamNet getestet (Tabelle 4.6).

Tabelle 4.6: Ergebnisse bei unterschiedlichen Aktivierungsfunktionen im 1-D-DamNet, fünffach kreuzvalidiert. Die Aktivierungsfunktionen werden in allen Layern bis auf den SE-Block sowie den Output-Layer verwendet. Gerechnet auf einer NVIDIA Titan Xp (12 GB). Die Funktionen sind in Abbildung A.11 dargestellt.

Aktivierungsfunktion	Details	$\overline{\text{KKR}}$	Durchschnittliche Trainingsdauer
Linear	$\epsilon = 0{,}0003, \lambda = 0{,}003$	$68{,}17 \pm 1{,}36\,\%$	$34{,}6\,\text{s}$
Sigmoid	$\epsilon = 0{,}0006, \lambda = 0$; Geduld 75 Epochen	$71{,}62 \pm 0{,}65\,\%$	$49{,}4\,\text{s}$
$\tanh$	$\epsilon = 0{,}0003, \lambda = 0{,}05$	$71{,}61 \pm 1{,}50\,\%$	$32{,}6\,\text{s}$
ReLU	$\epsilon = 0{,}0003, \lambda = 0{,}003$	$71{,}95 \pm 2{,}48\,\%$	$57{,}6\,\text{s}$
Leaky ReLU	$\epsilon = 0{,}0003, \lambda = 0{,}01, \alpha = 0{,}2$	$71{,}45 \pm 1{,}41\,\%$	$44{,}2\,\text{s}$
cReLU	$\epsilon = 0{,}0003, \lambda = 0{,}01$; Tiefenverkettung	$71{,}28 \pm 1{,}18\,\%$	$60{,}8\,\text{s}$
ELU	$\epsilon = 0{,}0003, \lambda = 0{,}01$	$71{,}92 \pm 1{,}76\,\%$	$37{,}0\,\text{s}$
sELU	$\epsilon = 0{,}0003, \lambda = 0{,}03$	$71{,}27 \pm 0{,}79\,\%$	$42{,}2\,\text{s}$
ReLU6	$\epsilon = 0{,}0003, \lambda = 0{,}003$	$71{,}93 \pm 1{,}54\,\%$	$42{,}2\,\text{s}$

Da die erreichten Klassifikationsgenauigkeiten in allen Fällen bis auf die lineare Funktion in einem gemeinsamen Bereich liegen, spielt die exakte Wahl der Aktivierungsfunktion eine untergeordnete Rolle für die Güte des Ergebnisses. Krizhevsky et al. [20, S. 85] behaupten, dass CNN mit ReLU-Funktionen um ein vielfaches schneller lernen als mit $\tanh$-Neuronen. Dieser Sachverhalt kann in der vorliegenden Anwendung nicht bestätigt werden. Mit der $\tanh$-Funktion wird das Training bedeutend schneller beendet. Mit der Exponential Linear Unit (ELU)-Funktion kann ebenfalls eine Beschleunigung des Trainings erreicht werden. Beide Funktionen erlauben eine einfache Berechnung des Gradienten, was vermutlich ausschlaggebend für das Ergebnis ist. Weil das Netz flach und damit das Problem verschwindender Gradienten schwächer ausgeprägt ist, wird die $\tanh$-Aktivierung gewählt.

Padding Im vorherigen Teil der Arbeit wurde die Umstellung von Same- auf Valid-Padding als mögliche Untersuchung vorgeschlagen. Selbst wenn keine Verbesserung der Klassifikationsgenauigkeit erreicht wird, kann die Umstellung auf Valid-Padding eine Optimierung darstellen. Das ist damit zu begründen, dass das Ausgabevolumen des Convolutional Layers und damit das Eingangsvolumen des nachfolgenden Layers verkleinert wird. Dadurch verringert sich letztendlich die Anzahl aller trainierbaren Parameter und die Rechenzeit.

Diese Optimierung ist aufgrund des verwendeten DamCeption-Layers nicht vollständig umsetzbar. In diesem kommen verschieden große Filtermasken zum Einsatz, die bei Valid-Padding verschieden große Feature Maps produzieren würden. Die entstehenden Tensoren ließen sich nicht in der Tiefe aneinanderhängen. Daher kann nur im Convolutional Layer nach dem ersten Max-Pooling Layer der Padding-Modus verändert werden. Dadurch verringert sich die Parameteranzahl im 1-D-Fall um ca. 4.000 auf ingesamt ca. 187.000 (knapp 2 %). Die KKR bleibt zum ReLU-Fall aus Tabelle 4.6 mit $72{,}16 \pm 1{,}17\,\%$ nahezu unverändert. Damit stellt die Umstellung auf Valid-Padding im letzten Convolutional Layer eine Optimierung dar.

Batch Normalization und Batch Renormalization BN-Layer werden insbesondere für das Training tiefer Netze und bei problematischen Optimierungsproblemen vorgeschlagen [27, S. 413]. Zudem werden der regularisierende Effekt und die Beschleunigung des Trainings als Vorteile hervorgehoben. Daher werden BN-Layer sowie die deren Erweiterung zu BRN im 1-D-DamNet getestet (Tabelle 4.7). Sie werden am Eingang des DamCeption-Moduls sowie vor

Tabelle 4.7: Ergebnisse mit BN und BRN im 1-D-DamNet, fünffach kreuzvalidiert. Gerechnet auf einer NVIDIA Titan Xp (12 GB).

Setup	$\overline{\text{KKR}}$	Durchschnittliche Trainingsdauer
Ohne BN	$71,95 \pm 2,48\,\%$	$57,6\,\text{s}$
Mit BN	$68,43 \pm 1,50\,\%$	$47,6\,\text{s}$
Mit BRN	$67,24 \pm 1,81\,\%$	$59,0\,\text{s}$

jeder Nichtlinearität eingefügt. Es zeigt sich, dass mit BN und BRN keine Verbesserung erreicht wird. Die durchschnittliche Trainingsdauer erfährt dagegen bei Verwendung von BN-Layern eine Verringerung. Weil die Klassifikationsgenauigkeit ohne B(R)N wahrnehmbar geringer ist, werden BN- und BRN-Layer als Optimierungsoptionen fallen gelassen.

4.2.2 Optimierungen im Framework

Das Framework, in das bisher unterschiedliche Netze eingebettet wurden, bestimmt das Netztraining (Abschnitt 3.5.1). Dieses kann für eine Optimierung auf verschiedene Arten verändert werden.

Optimierer Es herrscht kein eindeutiger Konsens über die Wahl des Optimierungsalgorithmus'. Aus Untersuchungen zeichnen sich zwar Algorithmen mit adaptiver Lernrate als robust ab, die letztendliche Wahl scheint jedoch von der Vertrautheit des Entwicklers mit dem entsprechenden Algorithmen abzuhängen [27, S. 302]. Bisher wurde ausschließlich Adam [52] als Optimierer verwendet. Weit verbreitet und als Standard etabliert sind außerdem RMSProp [51] und ein SGD-Optimierer mit Momentum. Adam und RMSProp passen die Lernrate automatisch an. Bei „SGD+Momentum" kann zusätzlich eine Richtlinie zur Anpassung der Lernrate („learning schedule") vorgegeben werden [27, S. 413, 54, S. 292]. Die unterschiedlichen Optimierer werden im Framework implementiert und mit dem 1-D-DamNet getestet (Tabelle 4.8).

Tabelle 4.8: Ergebnisse bei unterschiedlichen Optimierern, fünffach kreuzvalidiert. Gerechnet auf einer NVIDIA Titan Xp (12 GB).

Optimierer	Details	$\overline{\text{KKR}}$	Durchschnittliche Trainingsdauer
Adam	$\epsilon = 0,0003, \lambda = 0,003$	$71,09 \pm 1,94\,\%$	$52,6\,\text{s}$
RMSProp	$\epsilon = 0,0003, \lambda = 0,003$, mit Nesterov-Momentum	$70,20 \pm 1,50\,\%$	$48,1\,\text{s}$
SGD + Momentum	ϵ linear abfallend, $\epsilon_{\text{init}} = 0,01$, $\epsilon_{\text{min}} = 0,00001$	$68,32 \pm 2,68\,\%$	$43,1\,\text{s}$
SGD + Momentum	ϵ exponentiell abfallend, $\epsilon_{\text{init}} = 0,01$	$68,81 \pm 2,54\,\%$	$40,8\,\text{s}$
SGD + Momentum	Halbierung von ϵ nach 20 Epochen ohne Verbesserung (analog Early Stopping), $\epsilon_{\text{init}} = 0,01$, $\epsilon_{\text{min}} = 0,00001$	$68,94 \pm 2,13\,\%$	$38,2\,\text{s}$

Unabhängig von der „learning schedule" wird keine Verbesserung der Klassifikationsgenauigkeit

erreicht. Das Training wird zwar eher beendet, es müssen jedoch zusätzliche Hyperparameter eingestellt werden, die das Konvergenzverhalten maßgeblich beeinflussen. Bei Adam und RMSProp ist dies nicht notwendig. Weil Adam eine Erweiterung von RMSProp darstellt, wird dieser Optimierer beibehalten.

Größe der Mini-Batches Die Anzahl der Trainingsbeispiele pro Mini-Batch b bestimmt die Genauigkeit des berechneten Gradienten. Daher ist für eine möglichst exakte Berechnung ein großes b erstrebenswert. Je nach Netzgröße muss die Größe der Mini-Batches allerdings durch den begrenzten GPU-Speicherplatz reduziert werden. Gleichzeitig können durch kleinere Mini-Batches Sattelpunkte in der Fehleroberfläche umgangen werden (Abschnitt 2.1.4). Zusätzlich steigt die Generalisierungsfähigkeit des Netzes, wenn kleinere Batches zur Berechnung der Approximation des Gradienten genutzt werden [139, S. 444]. Bisher wurde die Standard Mini-Batch-Größe von 128 [133, S. 194] verwendet. In diesem Schritt wird die Größe in beide Richtungen variiert (Tabelle 4.9).

Tabelle 4.9: Ergebnisse bei unterschiedlichen Mini-Batch-Größen, fünffach kreuzvalidiert. Gerechnet auf einer NVIDIA Titan Xp (12 GB).

Mini-Batch-Größe b	$\overline{\text{KKR}}$	Durchschnittliche Trainingsdauer
16	$70,30 \pm 1,37\,\%$	211,2 s
32	$70,73 \pm 1,82\,\%$	111,4 s
64	$71,38 \pm 1,52\,\%$	71,6 s
128	$70,90 \pm 0,98\,\%$	54,1 s
256	$69,41 \pm 2,11\,\%$	46,5 s
512	$69,53 \pm 1,56\,\%$	49,7 s
1024	$67,21 \pm 1,83\,\%$	53,2 s

Die Ergebnisse zeigen, dass die Wahl der Mini-Batch-Größe primär die Trainingsdauer beeinflusst. Wird b sehr klein gewählt, steigt die Dauer signifikant an. Bei $b > 256$ steigt die Trainingsdauer ebenfalls an, während gleichzeitig die KKR sinkt. Aufgrund des besseren Ergebnisses und zur Reduzierung des GPU-Speicherbedarfs wird $b = 128$ festgelegt bzw. beibehalten.

Regularisierung Die Regularisierung des Netzes erfolgt im DamNet mit mehreren Mechanismen. Daher bestehen verschiedene Optionen zur Variation der Regularisierungsstärke. Dazu zählen

- Dropout, anpassbar über die Dropout-Rate p_r,
- L2-Regularisierung bzw. „weight decay“, anpassbar über λ, und
- Early Stopping.

Der regularisierende Effekt von BN wird in dieser Arbeit nicht untersucht. Die L2-Regularisierung wurde für fast alle vorangegangenen Tests verwendet und z.T. angepasst. Dropout wurde durchgehend mit einem Standardwert von $p_r = 0,5$ eingesetzt. Diese beiden Mechanismen sind als Optimierungen im Netz zu werten, da sie direkt die trainierbaren Parameter beeinflussen. Durch die Änderung der Dropout-Rate wird keine zusätzliche Verbesserung erwartet, da bereits $50\,\%$ der Neuronen im FC-Layer zufällig deaktiviert werden. Bei Verringerung von p_r ist eine Verschlechterung der KKR zu erwarten, da sich einzelne Neuronen mehr auf die Trainingsbeispiele spezialisieren. Dies resultiert in einer Überanpassung des Netzes (Overfitting). Eine Verstärkung von λ kann dagegen die Lernkurven weiter annähern (Abbildung 3.3).

Early Stopping wird ebenfalls als Regularisierung kategorisiert. Das Verfahren wirkt sich nur indirekt auf die Parameter des Netzes aus, wobei neben der erforderlichen Schwelle (Early Stopping Delta) das Kriterium verändert werden kann. Bisher wurde ausschließlich die Fehlerquote anhand der Validierungsdaten als Kriterium verwendet. Stattdessen kann der Wert der Kostenfunktion bei den Validierungsdaten (Validierungskosten) eingesetzt werden (Tabelle 4.10).

Tabelle 4.10: Ergebnisse bei unterschiedlicher Regularisierung, fünffach kreuzvalidiert. Gerechnet auf einer Tesla P100 (16 GB). (Std) markiert den bisher betrachteten Fall.

Setup	$\overline{\text{KKR}}$	Durchschnittliche Trainingsdauer
Early Stopping anhand Validierungskosten, $\lambda = 0,003$	$71,80 \pm 1,27\,\%$	$73,7\,\text{s}$
Early Stopping anhand Validierungs-FQ, $\lambda = 0,003$ (Std)	$70,75 \pm 1,95\,\%$	$74,2\,\text{s}$
Early Stopping anhand Validierungs-FQ, $\lambda = 0,008$	$71,33 \pm 1,35\,\%$	$67,6\,\text{s}$
Early Stopping anhand Validierungs-FQ, $\lambda = 0,010$	$71,09 \pm 2,06\,\%$	$63,1\,\text{s}$
Early Stopping anhand Validierungs-FQ, $\lambda = 0,015$	$72,91 \pm 1,48\,\%$	$75,0\,\text{s}$
Early Stopping anhand Validierungs-FQ, $\lambda = 0,020$	$70,99 \pm 1,66\,\%$	$73,3\,\text{s}$
Early Stopping anhand Validierungs-FQ, $\lambda = 0,025$	$34,82 \pm 2,97\,\%$	$21,4\,\text{s}$

Die Verwendung der Validierungskosten bringt gegenüber der Validierungsfehlerquote keine Vorteile. Die Anpassung von λ ist nur bis zu einer Stärke $\lambda \leq 0,020$ möglich, ohne eine Divergenz im Training zu erzeugen. Bei einer Anpassung auf $\lambda = 0,01$ zeigt sich eine leichte Beschleunigung des Trainings, bei einer weiteren Erhöhung auf $\lambda = 0,015$ eine Verbesserung der durchschnittlichen KKR (erhöhte Trainingszeit). Da die zeitliche Differenz marginal zur Ausgangskonfiguration ($\lambda = 0,003$), die KKR jedoch höher ist, wird die L2-Regularisierungsstärke zu $\lambda = 0,015$ festgelegt.

4.2.3 Kombination der Einzeloptimierungen

In diesem Schritt wird getestet, ob durch die Kombination der einzelnen Optimierungen Synergien entstehen. Die Abweichungen zum bisherigen Setup können folgendermaßen zusammengefasst werden:

- $\tanh$-Aktivierungsfunktion
- $\lambda = 0,015$
- Valid-Padding vor dem zweiten Max-Pooling Layer

Die Ergebnisse zeigen eine Verbesserung der durchschnittlichen Trainingsdauer und eine geringfügig höhere KKR (Tabelle 4.11). Die reduzierte Rechenzeit ist inbesondere durch die Wahl der

Tabelle 4.11: Ergebnisse nach der Optimierung im 1-D-DamNet, fünffach kreuzvalidiert. Gerechnet auf einer Tesla P100 (16 GB). (Std) markiert den bisher betrachteten Fall, (Opt) die Konfiguration nach der 1-D-Optimierung.

Setup	$\overline{\text{KKR}}$	Durchschnittliche Trainingsdauer
ReLU, $\lambda = 0,003$, Same-Padding (Std)	$71,09 \pm 2,52\,\%$	$72,5\,\text{s}$
$\tanh$, $\lambda = 0,015$, Valid-Padding (Opt)	$72,51 \pm 1,50\,\%$	$49,2\,\text{s}$
$\tanh$, $\lambda = 0,025$, Valid-Padding	$73,03 \pm 0,96\,\%$	$65,8\,\text{s}$

Aktivierungsfunktion zu begründen und die gesteigerte KKR durch die höhere L2-Regularisierung. Die Lernkurven deuten jedoch darauf hin, dass das Netz durch die veränderte Nichtlinearität stärker zum Overfitting neigt. Daher wird eine weitere Erhöhung der L2-Regularisierung untersucht

(Tabelle 4.11). Das höhere λ trägt kaum zu einer verbesserten KKR bei, steigert jedoch die Trainingsdauer. Weil das Overfitting die Güte des Ergebnisses nur geringfügig zu beeinflussen scheint, wird die Kombination der Einzeloptimierungen als neuer Standard übernommen und die L2-Regularisierung nicht weiter erhöht.

Zusätzlich wird getestet, ob sich die Optimierungen aus dem 1-D-Netz auf den 2-D-Fall übertragen lassen (Tabelle 4.12). Die Ergebnisse zeigen, dass Netz-Optimierungen übertragen werden

Tabelle 4.12: Ergebnisse nach der Optimierung im 2-D-DamNet, fünffach kreuzvalidiert. Gerechnet auf einer Tesla P100 (16 GB). (Std) markiert den bisher betrachteten Fall, (Opt) die Konfiguration nach der 1-D-Optimierung.

Setup	$\overline{\text{KKR}}$	Durchschnittliche Trainingsdauer
ReLU, $\lambda = 0,003$, Same-Padding (Std)	$76,42 \pm 1,92\,\%$	$398,0\,\text{s}$
ReLU, $\lambda = 0,003$, Valid-Padding	$75,23 \pm 1,18\,\%$	$344,8\,\text{s}$
ReLU, $\lambda = 0,015$, Valid-Padding	$69,12 \pm 1,14\,\%$	$527,7\,\text{s}$
$\tanh$, $\lambda = 0,015$, Valid-Padding (Opt)	$72,68 \pm 0,75\,\%$	$378,1\,\text{s}$
$\tanh$, $\lambda = 0,003$, Valid-Padding	$74,65 \pm 2,00\,\%$	$247,1\,\text{s}$
ELU, $\lambda = 0,003$, Valid-Padding	$74,93 \pm 0,84\,\%$	$252,9\,\text{s}$

können, Framework-Optimierungen dagegen nicht. Durch Valid-Padding vor dem zweiten Max-Pooling Layer kann wie im 1-D-Fall Rechenzeit eingespart werden. Die starke L2-Regularisierung wirkt sich dagegen im 2-D-Fall negativ aus. Für die STFT-Datenrepräsentation wird die L2-Regularisierungsstärke daher auf $\lambda = 0,003$ zurückgesetzt. Die KKR mit $\tanh$-Aktivierung ist bei $\lambda = 0,003$ marginal geringer als mit ReLU-Funktion, die Trainingsdauer jedoch deutlich geringer. Weil mit der ELU-Aktivierungsfunktion im 1-D-Fall ebenfalls eine Reduzierung der Rechenzeit erreicht wurde und die Funktion zur standardmäßigen ReLU-Familie gehört, wird diese zusätzlich getestet. Dabei zeigt sich eine deutliche Verringerung der durchschnittlichen Trainingsdauer gegenüber der ReLU-Aktivierungsfunktion bei gleichzeitig geringer Streuung. Sollte das Netz in Zukunft erweitert werden, besteht bei der ELU-Funktion nicht die Gefahr verschwindender Gradienten. Daher wird im 2-D-Fall die ELU-Aktivierungsfunktion verwendet.

4.3 Untersuchung eines weiteren Datensatzes

Um zu überprüfen, ob die Vergrößerung der Datenmenge eine Verbesserung für das System mit sich bringt, wurden neue Daten erhoben. In Abschnitt 4.1.5 bestätigten die Ergebnisse bereits den positiven Einfluss des gesteigerten Datenumfangs. Mit dem neu enstandenen Datensatz werden weitere Ansätze untersucht, um eine Verbesserung des FDI-Systems zu erreichen.
Aufgrund der kurzen Bearbeitungszeit werden einige grundsätzliche Untersuchungen angestellt, die als Basis für eine anschließende Arbeit dienen können. Nach einem Abschnitt zur Datenerhebung werden erneut Netze mit verschiedenen Architekturen trainiert. Damit wird untersucht, ob sich die gesteigerte Datenmenge auf verschiedene Netzarchitekturen gleich auswirkt. Anhand des besten Netzes wird einerseits die Übertragbarkeit und andererseits der Einfluss des Zusammenspiels aus Sequenzlänge und Abtastrate überprüft. Im letzten Teil erfolgt eine Validierung anhand von zusätzlichen Messdaten mit veränderter Fahrzeugbeladung.

4.3.1 Datenerhebung

Die Datenerhebung erfolgt analog zu [5, 6] (Unterkapitel 3.1) und die Umsetzung der Dämpferdefekte folgt der Umsetzung beim DD-Datensatz. Bei den Messfahrten befinden sich Fahrer und

Beifahrer im Auto, der Reifenluftdruck wird bei kalten Reifen auf $2,8$ bar eingestellt. Die Strecken sowie Geschwindigkeiten werden ebenfalls identisch zur Datenerhebung beim DD-Datensatz gewählt. Die Schlechtwegestrecke mit erhöhter Radanregung befindet sich in saniertem Zustand, weshalb der Streckenabschnitt nicht mehr als Schlechtwegestrecke zu verwenden ist. Es werden daher nur vier der fünf Streckenabschnitte gegenüber [5, 6] für Messzwecke befahren. Zusätzlich werden die Überführungsfahrten zwischen den einzelnen Streckenabschnitten aufgezeichnet und in den Datensatz aufgenommen.
Während der Messfahrten werden gegenüber [5, 6] zusätzlich die Daten vom FlexRay-Kommunikationsnetz des Fahrzeugs aufgezeichnet. Die FlexRay-Abtastrate liegt für a_X, a_Y und $\dot{\psi}$ bei 100 Hz und für die Raddrehzahlen bei 200 Hz. Zur Vereinheitlichung findet deshalb ein Resampling auf $f_{s_{\text{FlexRay}}} = 100\,\text{Hz} = 2 \cdot f_{s_{\text{CAN}}}$ statt. Zur besseren Unterscheidung wird der neue Datensatz „DD2“ nach Daten auf CAN- (DD2-C) und FlexRay-Basis (DD2-F) unterschieden. Bei Veränderung der Sequenzlänge gegenüber dem bisherigen Standardwert 256 wird diese ebenfalls angegeben, z. B. DD2-F-512 für FlexRay-Daten beim DD2-Datensatz mit einer Sequenzlänge von 512 Datenpunkten.

Um für eine ausgeglichene Klassenverteilung zu sorgen, werden von den Überführungsfahrten diejenigen mit Label „FL“ sowie einige mit Label „Alle“ ausgeschlossen. Mit der Daten-Vorauswahl (Enable-Bedingungen) und einer Sequenzlänge von 256 Datenpunkten ergibt sich ein balancierter Datensatz (Abbildung 4.10).

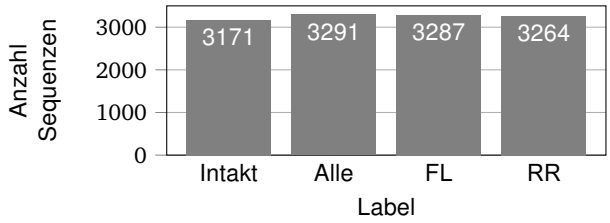


Abbildung 4.10: Histogramm des neuen Datensatzes „DD2-C(-256)“ bei einer Sequenzlänge von 256 Datenpunkten und CAN-Daten. Bei FlexRay-Daten verdoppelt sich die Anzahl der Beispiele ungefähr.

4.3.2 Untersuchung verschiedener Netze

In diesem Abschnitt werden verschiedene Netze mit dem neuen Datensatz trainiert. Auf Basis der Ergebnisse aus Abschnitt 4.1.4 und 4.2 werden ein angepasstes Baseline-Modell (Tabelle A.3), das optimierte DamNet, das Modell von Verstraete et al. [142] sowie das LeNet verwendet (Abbildung 4.11). Von den anderen Netzen aus der Variation der Netzwerkarchitektur wird keine signifikante Verbesserung für das FDI-System erwartet, da sie bereits in vorherigen Untersuchungen schlechtere Ergebnisse lieferten.

Generell zeigen alle Modelle durch die gesteigerte Datenmenge höhere KKR-Werte als mit dem DD-Datensatz. Abgesehen vom Verstraete-Modell im 2-D-Fall weisen sie durchgehend ein Konvergenzverhalten auf. Der Vergleich der Rechenzeiten zeigt, dass 1-D-Modelle wesentlich schneller trainiert werden als 2-D-Modelle. Letztere erreichen jedoch höhere Klassifikationsgenauigkeiten. Im Vergleich zeigt das (optimierte) DamNet die besten Ergebnisse bei geringer Streuung und kurzen (1-D) bis mäßigen (2-D) Rechenzeiten. Einzig das 2-D-Netz mit Depthwise Separable Convolution erreicht ebenfalls eine durchschnittliche KKR von über 86 %. Zur Zeiteinsparung kann bei 2-D-Daten alternativ das LeNet eingesetzt werden, das mit knapp 84 %

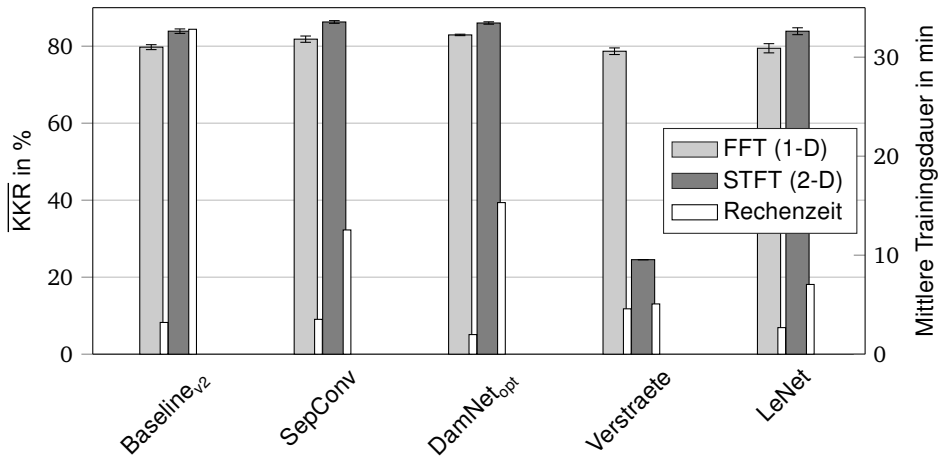


Abbildung 4.11: Ergebnisse der Architektur-Variation mit dem DD2-C-Datensatz, fünffach kreuzvalidiert. Die Streuung bei der fünffachen Kreuzvalidierung ist als Fehlerbalken dargestellt. „SepConv“ bezeichnet das Baseline-Modell mit Depthwise Separable Convolution wie es in Abschnitt 4.1.2 verwendet wurde. Gerechnet auf einer Tesla P100 (16 GB).

ein vergleichbares Ergebnis erreicht. Zur Maximierung der KKR wird in den nachfolgenden Untersuchungen jedoch weiterhin das DamNet eingesetzt.

4.3.3 Umstellung auf FlexRay-Daten

Mit den FlexRay-Daten steht durch die Abtastrate von 100 Hz eine höhere Auflösung zur Verfügung. Wird die Anzahl an Datenpunkten für ein Trainingsbeispiel beibehalten, verkürzt sich das betrachtete Zeitfenster bzw. die Sequenzdauer damit auf $2,56$ s. Gleichzeitig ergibt sich eine ungefähre Verdopplung der Datenbeispiele. Wird dagegen die Sequenzdauer beibehalten, ergibt sich eine Verdopplung der Datenpunkte pro Beispiel. Durch die Umstellung von CAN- auf FlexRay-Daten können mehrere Effekte untersucht werden:

- Die Auswirkung der Halbierung der Sequenzdauer bei gleichzeitiger Verdopplung der Anzahl der Datenbeispiele
- Die Auswirkung der Beibehaltung der Sequenzdauer bei gleichzeitiger Verdopplung der Datenpunkte pro Datenbeispiel

Bei Verwendung einer beträchtlichen Datensatzgröße ist die Modellkapazität zu berücksichtigen, da Modelle mit höherer Kapazität eine größere Datenmenge besser ausnutzen können als welche mit weniger [175, S. 7]. Das optimierte DamNet verfügt über ca. 187.000 Parameter und zeigt einen Hang zum Overfitting. Es besitzt somit mehr Kapazität als für den DD-Datensatz benötigt wird. Damit wird dieses Modell für das Training mit dem DD2-Datensatz verwendet. Es werden verschiedene Formen des DD- und DD2-Datensatzes zum Training des 1-D- und 2-D-DamNet verwendet (Tabelle 4.13). Die Ergebnisse deuten darauf hin, dass die Verwendung einer kürzeren Sequenzdauer bei gleichzeitiger Verdopplung der Datenmenge zu einer weiteren Verbesserung führt (von DD2-C-256 zu DD2-F-256). Insbesondere im 1-D-Fall ergibt sich mit ca. 4 % eine deutliche Steigerung der KKR beim Umstieg von CAN- auf FlexRay-Basis. Möglicherweise ist dieser Effekt auf die Vergrößerung des Datensatzes zurückzuführen, denn das Ergebnis passt gut zum Verlauf der Datenpunkte (Abbildung 4.9) aus Abschnitt 4.1.5. Zur Überprüfung wird eine Sequenzlänge von 128 Datenpunkten im DD-Datensatz verwendet (DD-128), was eine Sequenzdauer von $2,56$ s und eine Verdopplung der Beispiele gegenüber DD-256 bedeutet. Der positive Effekt bleibt dort allerdings aus.

Tabelle 4.13: Vergleich der Ergebnisse bei Umstellung auf FlexRay-Daten, fünffach kreuzvalidiert. Bei den mit * gekennzeichneten Werten gilt $\lambda = 0,003$, für den mit ° gekennzeichneten Fall $\lambda = 0,001$.

Setup	$\overline{\text{KKR}}$ **in %**				
Datensatz	DD-128	DD-256	DD2-C-256	DD2-F-256	DD2-F-512
Sequenzdauer	2,56 s	5,12 s	5,12 s	2,56 s	5,12 s
$\text{DamNet}_{\text{opt}}$ (1-D)	$64,43 \pm 1,21$	$72,51 \pm 1,50$	$82,92 \pm 0,19$	$86,94 \pm 0,26^*$	$90,36 \pm 0,63^*$
$\text{DamNet}_{\text{opt}}$ (2-D)	$73,06 \pm 1,16$	$74,93 \pm 0,84$	$86,02 \pm 0,31$	$87,61 \pm 0,25$	$91,44 \pm 0,67°$

Da die Verbesserung nicht auf die Verkürzung der Sequenzlänge zurückzuführen ist, scheint der Abtastrate eine bedeutendere Rolle zuzukommen. Zur Überprüfung wird testweise jedes zweite Beispiel des DD2-F-256-Datensatzes, d. h. die Hälfte aller Beispiele, aus dem Datensatz ausgeschlossen. Damit ist die Abtastrate gegenüber DD2-C-256 doppelt so hoch, die Anzahl der Beispiele jedoch in etwa identisch. Es ergibt sich eine KKR von $85,63 \pm 0,78\,\%$ (1-D) bzw. $86,72 \pm 0,57\,\%$ (2-D), was den positiven Einfluss der Abtastrate bestätigt. Die Ergebnisse liegen oberhalb derer des DD2-C-256-Datensatzes und sind schlechter als die bei Verwendung des gesamten DD2-F-256-Datensatzes. Sie sind somit konsistent zu der Hypothese, dass eine höhere Abtastrate und eine größere Datenmenge besser sind. Zusätzlich wird der Fall DD2-F-512 mit einer Sequenzdauer von $5,12\,\text{s}$ untersucht, bei dem durch das größere Zeitfenster tiefere Frequenzen erfasst werden können. In der angegebenen Konfiguration ergibt sich eine weitere Verbesserung auf über $90\,\%$ KKR. Als Ursache liegt die bessere Frequenzauflösung nahe, durch die u.a. Rad- und Aufbaufrequenzen genauer kategorisiert werden können. Von der Nutzung eines DD2-C-512-Datensatzes wird keine Verbesserung erwartet, da die relevanten Aufbau- und Radeigenfrequenzen (typischerweise $0,8-1,5\,\text{Hz}$ bzw. $10-15\,\text{Hz}$ [3, S.80]) bereits durch das ca. fünfsekündige Fenster erfasst werden können. Ein Dämpferdefekt erzeugt eine Verschiebung der Aufbaufrequenz in niedrigere Frequenzbereiche, da der Dämpfer weicher wird.

Auffällig ist zudem, dass die Ergebnisse von 1-D- und 2-D-Netz vergleichbar sind, weshalb aus Performance-Gründen die 1-D-Variante verwendet werden sollte. Das 1-D-DamNet besitzt ca. 190.000 Parameter weniger, das Preprocessing beansprucht weniger Zeit und das Training ist durchschnittlich fast $15\,\text{min}$ schneller.

4.3.4 Untersuchung von Übertragbarkeit und Robustheit

Zur Validierung der Güte des FDI-Systems werden in diesem Abschnitt Trainingsbeispiele aus Messfahrten klassifiziert, die dem System gänzlich unbekannt sind. Dadurch werden Übertragbarkeit bzw. Generalisierungsfähigkeit und Robustheit überprüft. Es wird folgendermaßen vorgegangen:

1. Einmaliges Training des 1-D-DamNet mit dem DD2-C-256-Datensatz (keine Kreuzvalidierung, Training:Validierung:Test 64:16:20 %).
2. Klassifikation von Messdaten
   (a) aus dem DD-Datensatz.
   (b) aus dem DD2-Datensatz, die zur Balancierung ausgeschlossen wurden.
   (c) mit veränderter Fahrzeugbeladung ($200\,\text{kg}$ im Kofferraum nahe der Rückbank).

Für den DD-Datensatz wurden keine Messdaten vom FlexRay aufgezeichnet. Daher wird für das Training der DD2-C-256-Datensatz gewählt, da nur so eine vergleichbare Klassifikation des DD-Datensatzes möglich ist. Im ersten Schritt nach dem Training werden der gesamte DD2-C-256-Datensatz und die 20 % zuvor verwendeten Testdaten klassifiziert. Damit wird sichergestellt, dass das Wiederherstellen des trainierten Netzes sowie die Klassifikation eines gegebenen Datensatzes korrekt ablaufen. Anschließend werden die unbekannten Datensätze klassifiziert (Tabelle 4.14).

Tabelle 4.14: Ergebnisse bei Übertragung des trainierten 1-D-DamNet auf andere Datensätze. Im trainierten Zustand werden anhand der 20 % ungesehenen Testdaten 82,67 % KKR erreicht.

Setup / **Datensatz**	**KKR**
DD2-C-256 Testdaten (20 % zufällig)	82,67 %
DD2-C-256 Komplett	84,78 %
DD-Datensatz	64,37 %
DD2-C-256 unbekannte Überführungsfahrten	68,16 %
DD2-C-256 Zusatzbeladung 200 kg (Massedatensatz)	67,14 %

Die Ergebnisse bestätigen die korrekte Wiederherstellung des trainierten Netzes. Bei Verwendung des kompletten DD2-C-Datensatzes wird eine leichte Verbesserung erreicht, da das Netz beim Training zum (leichten) Overfitting neigt und zusätzlich zu den unbekannten Testdaten Trainingsbeispiele wiedererkannt werden. Alle Klassifikationsergebnisse gänzlich unbekannter Messfahrten zeigen dagegen eine deutliche Abweichung zur KKR des DD2-C-256-Datensatzes. Das bedeutet, dass die Generalisierungsfähigkeit des trainierten CNN noch Verbesserungspotenzial besitzt. Sie ist offenbar nicht so hoch wie die zuvor erreichte KKR mit dem Testdatensatz suggeriert. Die Abweichung beim DD-Datensatz ist möglicherweise auf den Anteil von Messungen auf der Schlechtwegestrecke zurückzuführen, die im DD2-Datensatz nicht aufgenommen werden konnten. Die Überführungsfahrten, die nicht in den DD2-Datensatz aufgenommen wurden, werden mit einer ähnlichen KKR klassifiziert wie die anderen beiden unbekannten Datensätze. Das deutet darauf hin, dass Streckenprofile bei der Klassifikation eine Rolle spielen. Verändert sich die Beladung des Fahrzeug, hat dies Auswirkungen auf das gesamte Fahrzeugverhalten. Dieses Verhalten ist dem zuvor trainierten Netz unbekannt, weshalb sich die Klassifikationsgenauigkeit beim Massedatensatz verschlechtert. Dennoch können ca. zwei Drittel aller Datenbeispiele korrekt klassifiziert werden. Mit Aufnahme von genügend Daten unter veränderten Betriebsbedingungen in den Trainingsdatensatz wird eine Verbesserung des Ergebnisses erwartet. Diese Hypothese ist in weiteren Untersuchungen zu überprüfen.

Eine naheliegende Verbesserungmöglichkeit des FDI-Systems stellt auf Basis vorangegangener Ergebnisse die Verwendung einer höheren Abtastrate dar. Ob eine Verbesserung durch die Verwendung von FlexRay-Daten möglich ist, wird durch erneute Klassifikation der Überführungsfahrten und des Massedatensatzes überprüft (Tabelle 4.15), da hierbei zusätzlich zum CAN-Bus die Daten vom FlexRay-Netz aufgezeichnet wurden. Darüber hinaus wird der DD2-Datensatz um die Hälfte der Messungen des Massedatensatzes erweitert (Datensatz „DD2M0.5“). Anschließend wird ein Netz auf diesem kombinierten Datensatz trainiert und zur Klassifikation eingesetzt (Tabelle 4.15). Damit wird zur Überprüfung der obigen Hypothese angesetzt.

Die Ergebnisse zeigen, dass mit der Umstellung auf $f_s = 100$ Hz eine höhere Robustheit erreicht wird als mit Daten, die mit 50 Hz aufgezeichnet werden. Ohne Aufnahme der Messdaten mit zusätzlicher Beladung in den Datensatz werden fast 79 % des gesamten Massedatensatzes korrekt klassifiziert, was gegenüber den CAN-basierten Daten eine Steigerung von ca. 12 %

Tabelle 4.15: Zusatzuntersuchung zur Robustheit bei Veränderung der Fahrzeugmasse anhand des 1-D-DamNet. „DD2M0.5“ bezeichnet einen Datensatz, der aus DD2-Datensatz und der Hälfte der Messungen bei Variation der Fahrzeugmasse besteht.

Trainingsdatensatz	Testdatensatz	**KKR**
DD2-F-512	Testdaten (20 % zufällig)	90,23 %
DD2-F-512	unbekannte Überführungsfahrten	79,33 %
DD2-F-512	Zusatzbeladung 200 kg	78,84 %
DD2M0.5-F-512	Testdaten (20 % zufällig)	90,40 %
DD2M0.5-F-512	Zusatzbeladung 200 kg (komplett)	88,03 %
DD2M0.5-F-512	Zusatzbeladung 200 kg (unbekannter Teil)	79,09 %

bedeutet. Ein ähnlich gutes Ergebnis wird für die Überführungsfahrten erreicht. Wird ca. die Hälfte der Messfahrten mit Zusatzbeladung in den Datensatz aufgenommen, ergibt sich eine KKR des gesamten Masse-Datensatzes von ca. 88 %. Wird dagegen nur die unbekannte Hälfte klassifiziert, zeigt sich keine signifikante Verbesserung gegenüber dem Netz, das ohne Teile des Masse-Datensatzes trainiert wurde. Durch Vergleich der Konfusionsmatrizen kann jedoch festgestellt werden, dass die Verbesserung bei Klassifikation des gesamten Massedatensatzes nur durch Auswendiglernen der gesehenen Beispiele erreicht wird und nicht durch eine verbesserte Generalisierung (Tabelle 4.16).

Tabelle 4.16: Konfusionsmatrizen bei der Klassifikation des gesamten ((a), (b)) und halben ((c),(d)) Datensatzes mit 200 kg Zusatzbeladung mit verschieden trainierten Netzen. Der Trainingsdatensatz ist jeweils im Titel angegeben.

Klassifikation des gesamten Masse-Datensatzes

(a) DD2-F-512-Datensatz, KKR 78,84 %

Tatsächlich \ Vorhergesagt	Intakt	Alle	FL	RR
Intakt	149	14	38	146
Alle	0	267	3	3
FL	6	6	285	12
RR	13	15	13	300

(b) DD2M0.5-F-512-Datensatz, KKR 88,03 %

Tatsächlich \ Vorhergesagt	Intakt	Alle	FL	RR
Intakt	245	4	39	59
Alle	0	266	4	3
FL	4	2	292	11
RR	13	6	7	315

Klassifikation des halben Masse-Datensatzes (nur ungesehene Beispiele)

(c) DD2-F-512-Datensatz, KKR 79,11 %

Tatsächlich \ Vorhergesagt	Intakt	Alle	FL	RR
Intakt	74	10	30	45
Alle	0	131	3	1
FL	4	5	146	8
RR	9	9	9	152

(d) DD2M0.5-F-512-Datensatz, KKR 79,09 %

Tatsächlich \ Vorhergesagt	Intakt	Alle	FL	RR
Intakt	69	4	38	48
Alle	0	129	3	3
FL	4	2	148	9
RR	10	6	6	157

Es fällt auf, dass insbesondere die Klasse „Intakt“ häufig falsch zugeordnet wird. Besonders viele dieser Beispiele werden fälschlicherweise einem Defekt hinten rechts zugeordnet. Eine naheliegende Begründung ist, dass das Zusatzgewicht primär auf der Hinterachse liegt, wodurch sich das Verhalten des Fahrzeugaufbaus im rückwärtigen Fahrzeugbereich stark verändert. Für eine eingehende Analyse werden mehr Daten benötigt, die derzeit nicht vorliegen.

5 Diskussion

Im vorherigen Kapitel wurden die Ergebnisse des FDI-Systementwurfs präsentiert. In diesem Kapitel werden diese rekapituliert und diskutiert. Zudem werden der Ansatz und die erreichten Ergebnisse kritisch hinterfragt.

5.1 Diskussion der Wahl des Ansatzes

DL-Verfahren stellen eine gute Möglichkeit dar, den Schritt der Merkmalsextraktion und -selektion überflüssig zu machen. Im Bereich des Deep Learning existieren neben CNN jedoch eine Vielzahl von möglichen Verfahren, die zur Diagnose eingesetzt werden können. Werk [39] untersucht bspw. das unüberwachte Lernen von Merkmalen mit Sparse Filtern für dieselbe Anwendung wie diese Arbeit. Veröffentlichungen aus anderen Bereichen der Defektdiagnose setzen u.a. DBN [118, 124] oder „stacked sparse autoencoder“ [120] ein. Die Wahl der in dieser Arbeit verwendeten Architektur – Convolutional Neural Networks – konnte jedoch durch eine ausführliche Literaturrecherche zur Defektdiagnose mechanischer Komponenten gefestigt werden. Zur Untersuchung der Verwendbarkeit von CNN und verschiedener Einflüsse auf die Ergebnisgüte wurde ein entsprechendes FDI-System implementiert.

Bei der Implementierung kamen reine CNN-Architekturen zum Einsatz. Andere Architekturen, die bspw. CNN und RNN kombinieren, konnten aufgrund des beschränkten Umfangs nicht untersucht werden. Diese zeigen jedoch in der Spracherkennung [184], im medizinischen Diagnosebereich [185] und in weiteren Anwendungen [186] gute Ergebnisse. Mit dem reinen CNN-Ansatz wurde eine KKR von über 90 % erreicht, was die Einsetzbarkeit von CNN für die Anwendung verifiziert.
Bei der Klassifikation von unbekannten Datensätzen offenbarten sich jedoch Schwächen des Systems, die nicht genauer untersucht wurden. Die Analyse von CNN wurde i.A. bisher nur dürftig behandelt. CNN stellen wie KNN Black-Box-Modelle dar. Nur wenige Veröffentlichungen beschäftigen sich damit, die Muster in diesen Modellen zu entdecken und zu analysieren [59]. Daher bleibt eine genauere Untersuchung des trainierten Netzes, z. B. durch „Deconvolution“ [187], offen. Die Analyse der erlernten Merkmale lässt möglicherweise Rückschlüsse für den Systementwurf wie bspw. die Datenkanalauswahl zu.

Weil das maschinelle Lernen und insbesondere das Deep Learning Ansätze mit vielen Hyperparametern sind, konnten nur ausgewählte (Hyper-)Parameterkonfigurationen getestet werden. Die in der Arbeit getroffenen Aussagen gelten daher generell nur für die untersuchten Konstellationen aus Daten, Modell und (Hyper-)Parametern. Die Allgemeingültigkeit der gewonnenen Ergebnisse ist in Frage zu stellen, da bereits das Verändern eines einzigen Hyperparameters zu signifikant unterschiedlichen Ergebnissen führen kann. Inwiefern eine Aussage für andere Parameterkonfigurationen gilt, ist im Zweifelsfall durch Parameterstudien zu überprüfen. In diesem Kapitel werden daher weitere Parameterkonfigurationen vorgeschlagen, die für all-

gemeingültigere Aussagen untersucht werden sollten. Diese Arbeit ist insgesamt als erste Machbarkeits-Untersuchung anzusehen, die für ein zukünftiges FDI-System richtungsweisend dienen soll.

5.2 Diskussion der Hauptuntersuchungsaspekte

Zum Entwurf eines CNN-basierten FDI-Systems für die Dämpferdefektdiagnose wurden auf Basis der Literaturrecherche verschiedene Hauptuntersuchungsaspekte herausgearbeitet. Die Reihenfolge der Hauptuntersuchungspakete wurde unter der Annahme festgelegt, dass sich die Untersuchungsergebnisse eines Aspekts gleichermaßen auf verschiedene Variationen der übrigen Untersuchungen auswirken.

Datenkanalauswahl Signale, die für die vorliegende Anwendung sinnvoll eingesetzt werden können, wurden bereits im Vorfeld durch Kenntnisse aus der Fahrdynamik festgelegt [6]. Der Anteil von Expertenwissen zur Merkmalsextraktion wird mit dem vorgeschlagenen Ansatz zwar reduziert, jedoch nicht vollständig eliminiert.
Im ersten Untersuchungspaket wurde aus den verfügbaren Datenkanälen diejenige Konstellation gewählt, welche die vielversprechendsten Ergebnisse lieferte. Unabhängig von der Datenrepräsentation lässt sich feststellen, dass mehr Signale zu einer Verbesserung der Klassifikationsgenauigkeit beitragen. Denn mit mehr Signalen steht eine reichhaltigere Repräsentation des zu erkennenden Defekts zur Verfügung.
Der Beitrag eines einzelnen Signals zur KKR des Systems kann durch dessen alleinige Verwendung abgeschätzt werden. Allerdings lässt sich nur einschätzen, ob ein Signal zur Verbesserung beiträgt oder nicht. Für die Abschätzung des Anteils einzelner Signale ist eine Metrik zu erarbeiten. Von den zur Verfügung stehenden Signalen wurde der Lenkwinkel für die weitere Verwendung im FDI-System ausgeschlossen. Die Analysen im 1-D-Baseline-Modell mit Daten im Zeit- und Frequenzbereich würden ebenfalls die Entscheidung unterstützen, die Längsbeschleunigung auszuschließen. Bei erneuten Tests mit dem in Abschnitt 4.3.4 verwendeten Setup konnte unter Ausschluss der Längsbeschleunigung eine KKR von ca. 90 % erreicht werden, was die vorangegangene Überlegung bestätigt. Der Ausschluss weiterer Signale sollte daher zur Effizienzsteigerung des Systems untersucht werden.

Datenfusionsebene Anschließend wurde untersucht, welche Fusionsart der einzelnen Sensorsignale der Klassifikationsgenauigkeit am zuträglichsten ist. Die gewöhnliche Convolution sowie die Depthwise Separable Convolution zeigten die besten Ergebnisse. Damit konnte das Ergebnis von Jing et al. [148] nicht bestätigt werden. Die besseren Ergebnisse sind möglicherweise auf die Erhaltung temporaler Zusammenhänge in den Feature Maps gegenüber der Fusion auf Rohdatenebene zurückzuführen. Eine Gruppierung von Signalen, die bspw. derselben Domäne entstammen, brachte bei der Depthwise Separable Convolution keine Verbesserung.
Die gewöhnliche Convolution stellt das üblicherweise gewählte Mittel dar und wurde deswegen als Fusionsart für das Baseline-Modell sowie die übrigen Untersuchungspakete festgelegt. Der Einfluss der Filteranzahl pro Sensor wurde kurz beleuchtet. Dabei zeigte sich ein leichter Aufwärtstrend der KKR mit zunehmender Anzahl. Auf Basis der Erkenntnisse durch die Architekturvariation wird jedoch durch eine weitere Steigerung der Filteranzahl keine Verbesserung erwartet.

Datenaufbereitung Ein End-to-End-System für die Dämpferdefektdiagnose, das ausschließlich Rohdaten verarbeitet, ist nach aktuellem Erkenntnisstand nicht umsetzbar. Stattdessen sollten die Daten im FDI-System in eine Frequenz-basierte Darstellung transformiert werden. Diese erreichte gegenüber den Zeit- und Phasen-basierten Darstellungen signifikant höhere Genauigkeiten. Die Amplitudeninformation spielt für die Dämpferdefektdiagnose eine untergeordnete Rolle, was sich anhand der Ergebnisse mit linearem Detrend sowie den Frequenz-basierten Darstellungen zeigt.
Mehrere Veröffentlichungen schlagen die Verwendung von Skalogrammen (WT-basiert) statt Spektrogrammen (STFT) vor [116, 142]. In dieser Arbeit wurde mit dem WPI ein WPT-basiertes Bild umgesetzt, um einen auf Wavelets basierenden Ansatz zu nutzen. Mit dem WPI wurden keine guten Ergebnisse erzielt, was möglicherweise auf das Entfernen von Energieanteilen durch Clipping zurückzuführen ist. Als weitere Frequenz-basierte 2-D-Datenrepräsentation wird daher die Implementierung von Skalogrammen zur Untersuchung anderer Wavelet-basierter Ansätze vorgeschlagen.
Die Parameter der STFT wurden nach der optischen Erscheinung des Transformationsergebnisses (Zeit-/Frequenzauflösung) und der resultierenden Bildgröße gewählt. Wird der 2-D-Ansatz weiter verfolgt, sollte der Einfluss unterschiedlicher STFT-Parameter untersucht werden. Eine Variation der Bildgröße wurde ebenfalls nicht genauer untersucht. Die eingehende Analyse dieses Aspekts wird vorgeschlagen.

Die 1-D-Netze (FFT) sind gegenüber den Netzen für 2-D-Datenrepräsentationen (STFT) wesentlich recheneffizienter, erreichten jedoch in den vorangegangenen Untersuchungen eine maximale KKR von ca. 70 %. Dagegen konnten mit der STFT durchschnittlich über 76 % erreicht werden. Somit besitzen beide Verfahren ihre Vor- und Nachteile, weshalb keines von beiden endgültig ausgeschlossen werden kann.

Netzarchitektur Im vierten Hauptuntersuchungspaket wurde die Netzarchitektur variiert. Für die Untersuchung von FFT- und STFT-basierten Daten wurden sowohl 1-D- als auch 2-D-Varianten unterschiedlicher Netze implementiert. Die Auswahl der untersuchten Netze erfolgte auf einer teilweise subjektiven Wertung. Eine erfolgreiche Übertragbarkeit der ausgewählten „Spezial-Netze“ aus anderen Anwendungen konnte nicht bestätigt werden. Möglicherweise lassen sich jedoch andere, nicht untersuchte Netz-Architekturen besser für die Dämpferdefektdiagnose einsetzen.
Von den Trends aus der Bildverarbeitung wurden Inception-Modul und SE-Block in das DamNet aufgenommen. Das Baseline-Modell sowie das als „SepConv“ bezeichnete Netz erreichten ähnlich gute Ergebnisse wie das DamNet, verzichten jedoch auf derartige Modell-Bausteine. Daher ist eine Untersuchung der Notwendigkeit solcher Blöcke zu empfehlen. Skip Connections wurden mit dem SE-ResNeXt eingeführt, das allerdings keine guten Ergebnisse erreichte. Möglicherweise können derartige Architektureinflüsse in simplere Architekturen gewinnbringend eingebracht werden, was nicht untersucht wurde. Von den übertragenen Architekturen zeigte einzig das LeNet für beide Datenrepräsentationen eine zum Baseline-Modell vergleichbare KKR. Auf Basis der Folgeuntersuchung zum ersten Convolutional Layer sowie der Netztiefe konnte dieses Ergebnis nachvollzogen werden. Flache Architekturen erzielten bessere Ergebnisse als tiefe. Zusätzlich konnte bei einer geringen Anzahl von Blöcken die KKR durch das Stapeln von zwei bis drei Convolutional Layern leicht gesteigert werden. Die Verschlechterung bei zunehmender Tiefe widerspricht teilweise der Literatur. Ob dieses Ergebnis durch die zunehmende Anzahl von Max-Pooling-Operationen oder die Größenordnung der Nichtlinearität zustandekommt, ist deswegen genauer zu untersuchen.

Bei der gewählten FFT-Darstellung der Daten spielte die Größe der Filtermaske eine untergeordnete Rolle. Bei der Darstellung als Bild sind dagegen größere Filtermasken im ersten Convolutional Layer zu bevorzugen. Die Anzahl der Filterkernel sollte eine Mindestanzahl übersteigen, eine weitere Erhöhung scheint jedoch nicht notwendig zu sein. Damit wird die Beobachtung von Lee et al. [149] bestätigt. Bei der Abstimmung des Baseline-Modells (Abschnitt 3.3.3) kann rückwirkend der gleiche Einfluss der Filteranzahl festgestellt werden. Das lässt vermuten, dass die Aussagen sowohl für Zeit- als auch Frequenz-basierte Datenrepräsentationen gelten.
Für die Untersuchung der Netzarchitektur wurde nur der erste Convolutional Layer variiert, während die Hyperparameter der anderen Layer weitestgehend unverändert blieben. Ob durch deren Anpassung eine weitere Verbesserung erreicht werden kann, wurde nicht weiter untersucht. In Anlehnung an VGGNet[71] ist die einheitliche Wahl der Netz-Hyperparameter jedoch zu bevorzugen, da sie den Entwurf einer Architektur wesentlich erleichtert.

Die besten Ergebnisse wurden mit der entworfenen Architektur „DamNet“ erreicht, weshalb diese für das FDI-System und den letzten Hauptuntersuchungsaspekt gewählt wurde.

Datenmenge Auf Basis unterschiedlich großer Teildatenmengen des DD-Datensatzes wurde im letzten Hauptuntersuchungspaket eine Prognose für die erforderliche Datenmenge erstellt, um ein gefordertes KKR-Niveau von $80\,\%$ zu erreichen. Ein neuer Datensatz wurde in der prognostizierten Größe eingefahren und analog zu [5, 6] vorausgewählt. Die Vervierfachung der Trainingsdatenmenge führte zu dem prognostizierten Ergebnis von über 80 % KKR. Allerdings lagen die Ergebnisse mit dem neu eingefahrenen Datensatz generell oberhalb derer mit dem DD-Datensatz, was möglicherweise auf das Fehlen des Streckenabschnitts mit hoher Radanregung zurückzuführen ist. Eine weitere Einflussmöglichkeit sind Unregelmäßigkeiten in den Messdaten, die bspw. durch eine Reifenunwucht oder einen Standplatten zustande kommen können. Diese Einflussfaktoren wurden vor der Aufnahme des DD2-Datensatzes explizit geprüft und eliminiert.

Die Ergebnisse mit dem DD2-Datensatz bestätigten den positiven Einfluss größerer Datenmengen auf die KKR und das Verfahren zur Abschätzung der benötigten Datenmenge bewies sich als haltbar. Des weiteren zeigten die Lernkurven zur Abschätzung der Datenmenge einen abflachenden Verlauf, d. h. es werden zunehmend mehr Daten benötigt, um ein höheres Niveau zu erreichen. Der Verlauf lässt vermuten, dass eine Klassifikationsgenauigkeit von über $95\,\%$ nicht erreichbar ist. Da die KKR mit dem DD2-F-512-Datensatz bereits oberhalb von $90\,\%$ liegt, wird behauptet, dass eine KKR von $95\,\%$ und mehr bei einer ausreichend großen und sauberen Datenmenge erreichbar ist. Für ein zukünftiges FDI-System sollte der Datensatz so groß wie möglich sein. Zusätzlich sollte erneut eine Prognose der benötigten Datenmenge mit den Erkenntnissen bezüglich Abtastrate erstellt werden.

5.3 Diskussion der Optimierungsstrategien

Weil im Rahmen der Hauptuntersuchungsaspekte ausschließlich Daten- und Architektureinflüsse untersucht wurden, folgte in Unterkapitel 4.2 eine Optimierung von Netz- und Implementierungseinflüssen.

Hinsichtlich Netzoptimierung konnte festgestellt werden, dass nach Ausschluss der linearen Aktivierungsfunktion unabhängig von der Wahl der Funktion eine KKR von ca. $70\,\%$ erreicht wurde. Einzig die durchschnittliche Trainingsdauer variierte wahrnehmbar. Die geringe Trainingsdauer

bei $\tanh$- und ELU-Aktivierung ist vermutlich auf die (computerseitig) einfach zu berechnende Ableitung zurückzuführen. Die Trainingsdauer wurde von Beginn der Optimierung bis nach der Wiederherstellung des Netzes durch Early Stopping (vor Evaluation des Test-Datensatzes) gemessen. Genauer wäre möglicherweise die Angabe der benötigten Rechenzeit pro Trainingsbeispiel. Die angegebenen Zeiten sind daher als Indikator der Geschwindigkeit zu sehen und nicht als exakte Messung. Längerfristig ist das Training ein einmaliger Prozess, weshalb die Minimierung der Trainingszeit nur bedingt hilfreich ist. Die Verringerung der Parameteranzahl dagegen beeinflusst die benötigte Rechenzeit bei der Evaluation des trainierten Netzes. Als weitere Netzoptimierung wurde daher die Umstellung von Same- auf Valid-Padding vorgeschlagen. Beim DamNet war eine Umsetzung nur bedingt möglich, sie zeigte jedoch einen leicht positiven Einfluss auf die Rechenzeit. Eine Untersuchung des Padding-Einflusses wird insbesondere für die 2-D-Darstellung bei großen Filtern im ersten Convolutional Layer vorgeschlagen, da möglicherweise enorme Parametereinsparungen möglich sind. Im Gegensatz zu vielen Veröffentlichungen, die BN nutzen, konnte durch das Hinzufügen von BN- oder BRN-Layern keine Verbesserung erreicht werden. Möglicherweise wirkt sich BN erst ab einer Netztiefe positiv aus, die im untersuchten Netz nicht gegeben war. Die Netz-Optimierungen ließen sich insgesamt gewinnbringend kombinieren und vom 1-D- auf den 2-D-Fall übertragen.

Implementierungseinflüsse wie die Veränderung der Standards für Optimierer und Größe der Mini-Batches brachten keine Verbesserung. Einzig durch Anpassung der L2-Regularisierung λ konnte das Overfitting reduziert und die Generalisierungsfähigkeit verbessert werden. Optimierte Hyperparameter wie bspw. λ ließen sich nicht für die Anwendung mit einer anderen Datenrepräsentation anwenden. Die größten Einflüsse auf das Training konnten empirisch bei Lernrate ϵ und L2-Regularisierungsstärke λ festgestellt werden. Beim Entwurf eines Systems sollten daher vorrangig diese Parameter untersucht werden, während ansonsten Standards zu empfehlen sind [27, 54, 156].

Zusätzlich zu den Optimierungen aus Unterkapitel 4.2 können weitere Maßnahmen ergriffen werden. Verschiedene Initialisierungsstrategien [188] sowie die Reduktion der Netzparameter durch bspw. Pruning [189], d. h. das Aussortieren von ungenutzten Filtern, wurden nicht untersucht. Potenziell lässt sich mit einer Ensemble-Strategie eine zusätzliche Verbesserung erreichen [54, S. 381].
Insgesamt wurde jedoch durch die Ergebnisse bestätigt, dass signifikante Verbesserungen vorrangig durch Daten- und Architektureinflüsse entstehen und nur in geringem Maße durch Netz- oder Implementierungseinflüsse [36, S. 80, 156]. Daher sind derartige Optimierungen erst in späten Entwicklungsstadien zu empfehlen.

5.4 Diskussion von Übertragbarkeit und Robustheit

Die Übertragbarkeit kann aus mehreren Perspektiven diskutiert werden: Zum einen ist die Umsetzung der Dämpferdefekte zu berücksichtigen und zum anderen die Übertragungsfähigkeit eines trainierten Netzes auf neue Daten.

Defekte Dämpfer wurden wie in [5, 6, 32] durch Bestromung der Dämpferventile an den semiaktiven Dämpfern des Versuchsträgers nachgestellt. Die gewählte Bestromung resultiert für einen „Defekt“ in einer weicheren Dämpfercharakteristik. Inwiefern ein so simulierter Defekt mit der Realität übereinstimmt, wird in weiteren Arbeiten untersucht. Wird das Verfahren durch die experimentelle Überprüfung bestätigt, liefert der untersuchte Ansatz ein kostengünstiges und

zerstörungsfreies Verfahren zur Erhebung von Mess- bzw. Trainingsdaten. Zusätzlich ist zu überprüfen, ob Netze, die auf simulierten Defekten trainiert wurden, ebenfalls real verschlissene Dämpfer als defekt klassifizieren. Ein weiteres Problem des Ansatzes ist die Übertragung eines trainierten CNN auf andere Fahrzeugmodelle. Es ist zu erwarten, dass die Übertragung Schwächen des Ansatzes offenbart, da ein CNN Merkmale erlernt, die den Daten zugrunde liegen. Wird statt des hier verwendeten Gran Coupé bspw. ein VW Polo untersucht, verändern sich diese Merkmale. Es sind daher Daten über viele verschiedene Fahrzeugmodelle hinweg zu erheben, was nur durch Kooperation mit den Fahrzeugherstellern möglich sein wird. Eine Alternative stellen möglicherweise Flottensimulationen dar.

Eine Übertragung des trainierten Netzes auf unbekannte Streckenabschnitte und Betriebsbedingungen ist nach dem aktuellen FDI-Systementwurf nur bedingt gegeben. In Unterkapitel 4.3 wurde das DamNet mit dem neu eingefahrenen DD2-Datensatz trainiert und zur Klassifikation von unterschiedlichen Datensätzen eingesetzt. Gegenüber dem DD2-Testdatensatz zeigte sich eine deutliche Verschlechterung der Ergebnisse. Bei den Untersuchungen wurden $20\,\%$ der Datenbeispiele zufällig als Testdaten ausgewählt, was ein gängiges Verfahren darstellt. Weil bei der Datenerhebung jedoch die Streckenabschnitte mehrfach befahren wurden, besteht die Möglichkeit, dass Testdaten sehr große Übereinstimmung mit den Trainingsdaten zeigen. Dadurch entsteht möglicherweise eine Verzerrung der KKR nach oben, weil Overfitting nicht vollständig als solches erkannt wird. Die Verschlechterung der Ergebnisse deutet darauf hin, dass der DD2-Datensatz nicht genügend repräsentative Datenbeispiele besitzt, die für eine gute Generalisierungsfähigkeit notwendig sind. Für eine tiefgreifendere Untersuchung sollte der Testdatensatz aus Streckenabschnitten konstruiert werden, die nicht für das Training verwendet werden. Nur damit kann eine neutrale Bewertung unterschiedlicher FDI-Systeme garantiert werden.

Beim Vergleich zwischen DD- und DD2-Datensatz wurde festgestellt, dass die Abtastrate von $50\,\mathrm{Hz}$ (CAN) möglicherweise nicht ausreichend hoch für manche Zusammenhänge ist. Diese Vermutung wird durch die Verbesserung von Übertragbarkeit und Robustheit bei Umstellung auf eine Abtastrate von $100\,\mathrm{Hz}$ (FlexRay) unterstützt. Im Zuge der Umstellung wurde herausgefunden, dass die Beibehaltung der Sequenzdauer wichtiger ist als die der Sequenzlänge (Tabelle 4.13). Die mit der Umstellung erreichte Steigerung der KKR liegt bei über $10\,\%$. Eine weitere Untersuchung der Sequenzdauer wird daher vorgeschlagen. Ausschlaggebend ist womöglich die höhere Auflösung des Frequenzspektrums. Derzeit ist unklar, ob die Aufbau- oder Radfrequenzen wichtiger für die KKR sind. Zur Untersuchung sollten jeweils nur die Raddrehzahlen und die Beschleunigungen mit der Gierrate verwendet werden. Für beide Gruppen kann anschließend jeweils die Abtastrate bzw. Sequenzdauer variiert werden, was Rückschlüsse über die relevanten Frequenzen erlaubt. Für weitere Untersuchungen kann darüber hinaus ein Resampling auf $200\,\mathrm{Hz}$ erfolgen, da einige Signale bereits mit dieser Frequenz abgetastet werden. Problematisch ist möglicherweise die Interpolation von Signalen, die maximal mit $100\,\mathrm{Hz}$ abgetastet werden. Eine tiefergehende Analyse wurde zeitbedingt nicht durchgeführt, wird jedoch für zukünftige Untersuchungen empfohlen.

Die Aufnahme von Messdaten mit veränderter Masse in das Training brachte in den angestellten Versuchen keine Verbesserung. Möglicherweise ist der Anteil der aktuell verfügbaren Daten zu gering. Dies ist durch Einfahren weiterer Messdaten zu untersuchen.

6 Zusammenfassung und Ausblick

Merk [5] (auch [6]) untersucht die Diagnose von Schwingungsdämpfern im Fahrwerk mit einem SVM-basierten FDI-System. Dafür müssen Merkmale manuell extrahiert werden und das System basiert auf vom Entwickler getroffenen Annahmen, die die Qualität des Systems möglicherweise einschränken. Die Merkmalsextraktion kann durch „Feature Learning“ auf Basis von DL-Ansätzen überflüssig gemacht werden.

Die Literaturrecherche hat gezeigt, dass DL-Verfahren – insbesondere CNN – im industriellen Maschinenbau bereits erfolgreich zur Fehlerdiagnose eingesetzt werden. Dabei handelt es sich jedoch fast ausschließlich um Prüfstandsanwendungen. In der Automobilindustrie werden dagegen DL-Algorithmen für diverse Anwendungen auf Basis von Realdaten eingesetzt, jedoch nicht zur Defektdiagnose. In der vorliegenden Arbeit wurde der Einsatz der DL-Architektur CNN als FDI-System zur Dämpferdefektdetektion untersucht. Der Entwurf eines CNN-basierten FDI-Systems zur Diagnose von Fahrwerkskomponenten unter Verwendung von Messdaten aus normalen Fahrzyklen fand erstmalig in dieser Masterarbeit statt. Damit wird die Forschungslücke zwischen den beiden Feldern geschlossen und ein wichtiger Beitrag zur Zustandsüberwachung für höhere Autonomiestufen von Kraftfahrzeugen geleistet.

Es wurde ein FDI-System entworfen, das einerseits die Merkmalsextraktion und -selektion überflüssig macht und andererseits die Klassifikationsgenauigkeit steigert. Mit diesem wird auf zusätzliche Sensoren verzichtet und ausschließlich auf ABS- und ESP-Sensorik zurückgegriffen, wodurch keine zusätzlichen Kosten entstehen. Als Einflussfaktoren wurden Datenkanalauswahl, Datenfusionsebene, Datenvorverarbeitung, Netzarchitektur sowie Datenmenge untersucht. Auf Basis der Untersuchungsergebnisse wurde für die Dämpferdefektdiagnose folgendes Setup für ein CNN-basiertes FDI-System vorgeschlagen:

- Datenkanäle $n_{\mathrm{FL}}, n_{\mathrm{FR}}, n_{\mathrm{RL}}, n_{\mathrm{RR}}, a_{\mathrm{X}}, a_{\mathrm{Y}}, \dot{\psi}$
- Sequenzlänge 512 Datenpunkte mit Messdaten vom FlexRay bei $f_{\mathrm{s}} = 100\,\mathrm{Hz}$
- Entfernen linearer Trends und anschließende FFT als Datenvorverarbeitung
- Optimiertes 1-D-DamNet mit $\lambda_{\mathrm{FlexRay}} = 0,003$ als Architektur

Mit dem angegebenen Systementwurf konnte auf Basis neuer Daten (DD2-Datensatz) eine Korrektklassifizierungsrate (KKR) von über $90\,\%$ erreicht werden. Das FDI-System erzielte bei der Klassifikation von Daten „unbekannter“ Strecken sowie Daten unter zusätzlicher Beladung eine KKR von ca. $80\,\%$. Bei Verwendung des Datensatzes aus [5, 6] (DD-Datensatz) konnte eine leichte Verbesserung ($2-3\,\%$) gegenüber der SVM (ca. $73\,\%$) erreicht werden, ohne auf manuell extrahierte Merkmale angewiesen zu sein. Ähnliche Verbesserungen ($1-2\,\%$) können gegenüber [6] beobachtet werden, wenn die SVM zur Klassifikation des DD2-F-Datensatzes eingesetzt wird. Als größte Einflussfaktoren für die KKR zeichneten sich Datenmenge, Datenaufbereitung, Netzarchitektur sowie die Hyperparameter für Lernrate und Regularisierung ab. Optimierungen von Netz und Framework brachten keine signifikante Verbesserung.

Die Untersuchung neu eingefahrener Daten zeigte, dass hinsichtlich Übertragbarkeit und Robustheit des FDI-Systems weiterer Handlungsbedarf besteht. Insbesondere der Einfluss von Streckenprofilen sollte in nachfolgenden Arbeiten untersucht werden, da sowohl beim DD-Datensatz als auch beim Teil der „unbekannten“ Strecken des DD2-Datensatzes Leistungseinbußen zu verzeichnen waren. Für eine eingehende Untersuchung von Fehlklassifikationen wird ein System zur Rückverfolgung einzelner Datenbeispiele benötigt, das im Rahmen der Arbeit nicht implementiert wurde. Mit der Rückverfolgung von Trainingsbeispielen zu der entstammenden Messung kann eine zusätzliche Untersuchung bzgl. Ausreißern erfolgen. Bei Erweiterung des Systems um Global Positioning System (GPS)-Daten könnten zusätzlich Rückschlüsse auf schwer zu klassifizierende Fahrbahnabschnitte geschlossen werden. Auf Basis entsprechender Analysen könnten die gewonnen Erkenntnisse darüber hinaus zur Bereinigung des Datensatzes genutzt werden, was leistungssteigernd wirkt.

Die Datensätze enthalten nur gering dynamische Situationen. Die eigentliche Leistungsfähigkeit von DL-Systemen liegt jedoch insbesondere in der Klassifikation komplexer Zusammenhänge. In weiteren Untersuchungen sollte daher sukzessive der Datensatz um dynamische Fahrsituationen durch Schwächung der Enable-Bedingungen erweitert werden. Bei positiven Ergebnissen würde das FDI-System einen großen Schritt in Richtung Serienreife machen. Das Ausschließen gewisser Fahrsituationen (mindestens Stillstand) wird jedoch für eine Serienlösung nicht zu vermeiden sein. Längerfristig ist zudem eine Kombination von Dämpferdefekten mit weiteren Fahrwerksdefekten zu untersuchen, um ein einziges System zur Zustandsüberwachung des Fahrwerks zu entwickeln. Dafür sind jedoch große Datenmengen erforderlich. Das unüberwachte Lernen stellt hier eine Option dar, die in weiteren Arbeiten untersucht werden sollte. Mit Ausgabe der Klassifikationswahrscheinlichkeiten kann das System bei ausreichender Datenabdeckung für Predictive Maintenance erweitert werden.

In der vorliegenden Anwendung werden mit frequenzbasierten Darstellungen die besten Ergebnisse erreicht. Viele Implementierungen von Convolutional Layern verwenden eine FFT zur effizienten Berechnung der Faltung und transformieren das Ergebnis anschließend durch eine inverse FFT zurück. Es ist denkbar, Netz-intern auf eine Rücktransformation zu verzichten und somit ab der ersten Faltung innerhalb des CNN ausschließlich im Frequenzspektrum zu arbeiten [190]. Damit wäre die Umsetzung eines End-to-End-Systems möglich. Es sind allerdings Eingriffe in die als Bibliotheken zur Verfügung gestellten Algorithmen notwendig, was mit einem nicht unerheblichen Aufwand verbunden ist.

Ein Nachteil von CNN-basierten Methoden ist die große Menge an benötigten Mess- bzw. Trainingsdaten [22, S. 1309]. Während früher viele Forscher der Meinung waren, dass das Netztraining mit zufällig initialisierten Gewichten schlichtweg zu schwierig sei, weiß man heute, dass lediglich eine große Menge (beschrifteter) Daten und viel Rechenleistung benötigt werden, um die damals erträumten Ergebnisse zu erzielen [20, S. 84]. Die meisten Anwendungen mit (tiefen) CNN sind üblicherweise nur durch die Speicherkapazität der verwendeten GPU limitiert [36, S. 100]. Es wird daher behauptet, dass es ausreicht auf schnellere GPU und die Verfügbarkeit größerer Datensätze zu warten, um bessere Ergebnisse zu erreichen [20, S. 85].
Alternativ können Diagnose-Netze mit Simulationsdaten trainiert und anschließend in realen Applikationen eingesetzt werden [94]. In einer weiteren Arbeit sollte untersucht werden, inwiefern das am Lehrstuhl entwickelte Zweispurmodell für ein FDI-System benutzt werden kann. Insbesondere die Beurteilung der Verwendbarkeit der Simulationsdaten zum Training eines Diagnosenetzes (DL/CNN) sowie des mit Simulationsdaten trainierten Netzes für ein reales Fahrzeug (Realdaten) sind hier interessant.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Maurer, M. et al.: *Autonomes Fahren*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

[2] Johanning, V., Mildner, R.: *Car IT kompakt*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015.

[3] Heißing, B. et al., Hrsg.: *Fahrwerkhandbuch: Grundlagen · Fahrdynamik · Komponenten · Systeme · Mechatronik · Perspektiven*. 4., überarb. u. erg. Aufl. 2013. ATZ / MTZ-Fachbuch. Wiesbaden und s.l.: Springer Fachmedien Wiesbaden, 2013.

[4] Bedük, M. D. et al.: *Effects of damper failure on vehicle stability*. In: Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering 227, 7, S. 1024–1039, 2013.

[5] Merk, A.: *Dämpferdefektdetektion & -diagnose mittels Künstlicher Intelligenz*. Masterarbeit. München: Technische Universität München, 2018.

[6] Zehelein, T. et al.: *Damper diagnosis by artificial intelligence*. In: *9TH INTERNATIONAL MUNICH CHASSIS SYMPOSIUM 2018*. Hrsg. von Pfeffer, P. Proceedings. [S.l.]: MORGAN KAUFMANN, 2018, S. 461–482.

[7] Hardier, G.: *Recurrent RBF networks for suspension system modeling and wear diagnosis of a damper*. In: *The 1998 IEEE International Joint Conference on Neural Networks*. Piscataway, N.J: IEEE, 1998, S. 2441–2446.

[8] Liu, Y.-m. et al.: *Suspension system fault diagnosis method based on fuzzy mathematics*. In: *9th International Conference on Electronic Measurement & Instruments, 2009*. Hrsg. von Cui, J. Bd. 1. Piscataway, NJ: IEEE, 2009, S. 210–214.

[9] Sankavaram, C. et al.: *Data-driven fault diagnosis in a hybrid electric vehicle regenerative braking system*. In: *2012 IEEE Aerospace Conference*. Piscataway, NJ: IEEE, 2012, S. 1–11.

[10] González, J. P. N., Villanueva, P. P.: *Vehicle Lateral Dynamics Fault Diagnosis Using an Autoassociative Neural Network and a Fuzzy System*. In: *Advances in Computational Intelligence*. Hrsg. von Hutchison, D. et al. Bd. 7630. Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence. Berlin/Heidelberg: Springer Berlin Heidelberg, 2013, S. 236–246.

[11] Tang, D. et al.: *Feature selection and analysis of single lateral damper fault based on SVM-RFE with correlation bias reduction*. In: *Proceedings of the 35th Chinese Control Conference*. Hrsg. von Chen, J., Zhao, Q. Piscataway, NJ: IEEE, 2016, S. 3840–3845.

[12] Jing, L. et al.: *A convolutional neural network based feature learning and fault diagnosis method for the condition monitoring of gearbox*. In: Measurement 111, S. 1–10, 2017.

[13] Dong, S. et al.: *Design and application of unsupervised convolutional neural networks integrated with deep belief networks for mechanical fault diagnosis*. In: *2017 Prognostics and System Health Management Conference (PHM-Harbin)*. IEEE, 2017, S. 1–7.

[14] Hoang, D.-T., Kang, H.-J.: *Rolling element bearing fault diagnosis using convolutional neural network and vibration image*. In: Cognitive Systems Research 2018.

[15] Abadi, M. et al.: *TensorFlow: A system for large-scale machine learning*. 2016. url: http://arxiv.org/pdf/1605.08695v2.

[16] Collobert, R. et al.: *Torch7: A Matlab-like Environment for Machine Learning*. In: *BigLearn, NIPS Workshop*. 2011.

[17] The Theano Development Team et al.: *Theano: A Python framework for fast computation of mathematical expressions*. 2016. url: http://arxiv.org/pdf/1605.02688v1.

[18] Janssens, O. et al.: *Convolutional Neural Network Based Fault Detection for Rotating Machinery*. In: Journal of Sound and Vibration 377, S. 331–345, 2016.

[19] LeCun, Y. et al.: *Convolutional networks and applications in vision*. In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010, S. 253–256.

[20] Krizhevsky, A. et al.: *ImageNet Classification with Deep Convolutional Neural Networks*. In: *Advances in Neural Information Processing Systems 25*. Hrsg. von F. Pereira et al. Curran Associates, Inc, 2012, S. 1097–1105.

[21] Abdeljaber, O. et al.: *Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks*. In: Journal of Sound and Vibration 388, S. 154–170, 2017.

[22] Abdeljaber, O. et al.: *1-D CNNs for structural damage detection: Verification on a structural health monitoring benchmark data*. In: Neurocomputing 275, S. 1308–1317, 2018.

[23] Lienkamp, M.: *Dynamik der Straßenfahrzeuge*. Vorlesungsfolien. München: Technische Universität München, 2016.

[24] TÜV Süd: *Die häufigsten Mängel*. url: https://www.tuev-sued.de/auto-fahrzeuge/tools-services/tuev-report/die-haeufigsten-maengel, abgerufen am: 26.9.2018.

[25] Georgoulas, G., Nikolakopoulos, G.: *Bearing fault detection and diagnosis by fusing vibration data*. In: *Proceedings of the IECON2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society : Florence (Italy), October 24-27, 2016*. Piscataway, NJ: IEEE, 2016, S. 6955–6960.

[26] Schuck, S.: *Anomaliedetektion mittels Machine Learning in der Dämpferdiagnose*. Masterarbeit (unveröffentlicht). München: Technische Universität München, 2019.

[27] Goodfellow, I. et al.: *Deep learning (Adaptive Computation and Machine Learning)*. Cambridge, Massachusetts und London, England: MIT Press, 2016.

[28] Bedford, T. et al.: *A Comparison of Data-Driven and Model-Based Approaches to Quantifying Railway Risk*. In: *Probabilistic Safety Assessment and Management*. Hrsg. von Spitzer, C. et al. London und s.l.: Springer London, 2004, S. 2765–2771.

[29] Sankavaram, C. et al.: *Model-based and data-driven prognosis of automotive and electronic systems*. In: *IEEE International Conference on Automation Science and Engineering, 2009*. Piscataway, NJ: IEEE, 2009, S. 96–101.

[30] Medjaher, K., Zerhouni, N.: *Hybrid prognostic method applied to mechatronic systems*. In: The International Journal of Advanced Manufacturing Technology 69, 1-4, S. 823–834, 2013.

[31] Baraldi, P. et al.: *Model-based and data-driven prognostics under different available information*. In: Probabilistic Engineering Mechanics 32, S. 66–79, 2013.

[32] Jautze, M.: *Ein signalmodellbasiertes Verfahren zum Erkennen von Dämpferschäden bei Kraftfahrzeugen: Zugl.: Erlangen-Nürnberg, Univ., Diss., 2002*. Als Ms. gedr. Bd. 498. Fortschritt-Berichte VDI Reihe 12, Verkehrstechnik/Fahrzeugtechnik. Düsseldorf: VDI-Verl., 2002.

[33] Borner, M. et al.: *Comparison of different fault detection algorithms for active body control components: automotive suspension system*. In: *Proceedings of the 2001 American Control Conference*. Bd. 1. Evanston, Ill und Piscataway, N.J: American Automatic Control Council, 2000, S. 476–481.

[34] Moncada, H. B., Marin, J. A.: *Fault Detection for a Bus Suspension Model Using an Utkin Observer*. In: *IEEE Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2011*. Piscataway, NJ: IEEE, 2011, S. 246–251.

[35] Hernandez-Alcantara, D. et al.: *Fault estimation methods for semi-active suspension systems*. In: *2015 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*. Piscataway, NJ: IEEE, 2015, S. 1–5.

[36] Buduma, N., Locascio, N.: *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms*. 1. Aufl. Sebastopol, CA: O'Reilly Media, 2017.

[37] Schmidhuber, J.: *Deep learning in neural networks: an overview*. In: Neural networks : the official journal of the International Neural Network Society 61, S. 85–117, 2015.

[38] Chen, Z. et al.: *Gearbox Fault Identification and Classification with Convolutional Neural Networks*. In: Shock and Vibration 2015, 2, S. 1–10, 2015.

[39] Werk, P.: *Dämpferdefektdiagnose mittels Deep Learning basierend auf unüberwachtem Lernen von Merkmalen*. Masterarbeit. München: Technische Universität München, 2018.

[40] Rosenblatt, F.: *The perceptron: A probabilistic model for information storage and organization in the brain*. In: Psychological Review 65, 6, S. 386–408, 1958.

[41] Habibi Aghdam, H., Jahani Heravi, E.: *Guide to Convolutional Neural Networks*. Cham: Springer International Publishing, 2017.

[42] Kramer, O.: *Computational Intelligence: Eine Einführung*. Informatik im Fokus. Berlin: Springer, 2009.

[43] Conradt, J. et al.: *Computational Intelligence*. Unterlagen zur Vorlesung. München: Technische Universität München, 2017.

[44] Hornik, K. et al.: *Multilayer feedforward networks are universal approximators*. In: Neural Networks 2, 5, S. 359–366, 1989.

[45] Allison, J.: *Simplified Gradient Descent Optimization - File Exchange - MATLAB Central*. url: https://de.mathworks.com/matlabcentral/fileexchange/35535-simplified-gradient-descent-optimization, abgerufen am: 20.6.2018.

[46] LeCun, Y. A. et al.: *Efficient BackProp*. In: *Neural Networks: Tricks of the Trade*. Hrsg. von Montavon, G. et al. Bd. 7700. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 9–48.

[47] Hagan, M. T. et al.: *Neural network design*. 2. Aufl. Wrocław: Amazon Fulfillment Poland Sp. z o.o, 2014.

[48] Polyak, B. T.: *Some methods of speeding up the convergence of iteration methods*. In: USSR Computational Mathematics and Mathematical Physics 4, 5, S. 1–17, 1964.

[49] Sutskever, I. et al.: *On the importance of initialization and momentum in deep learning*. In: *Proceedings of the 30th International Conference on Machine Learning*. Hrsg. von Sanjoy Dasgupta, David McAllester. Proceedings of Machine Learning Research. Atlanta, Georgia, USA: PMLR, 2013, S. 1139–1147.

[50] Duchi, J. et al.: *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. In: Journal of Machine Learning Research 12, S. 2121–2159, 2011.

[51] Hinton, G. et al.: *Neural Networks for Machine Learning: Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. Vorlesungsunterlagen. Toronto: University of Toronto, 2012.

[52] Kingma, D. P., Ba, J.: *Adam: A Method for Stochastic Optimization*. 2014. url: http://arxiv.org/pdf/1412.6980v9.

[53] Raschka, S., Mirjalili, V.: *Machine Learning mit Python und Scikit-Learn und TensorFlow : Das Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning*. Frechen, GERMANY: MITP, 2017.

[54] Géron, A.: *Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. First edition. Sebastopol, CA: O'Reilly Media, 2017.

[55] Srivastava, N. et al.: *Dropout: a simple way to prevent neural networks from overfitting*. In: Journal of Machine Learning Research 15, 1, S. 1929–1958, 2014.

[56] Thoma, M.: *Analysis and Optimization of Convolutional Neural Network Architectures*. Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2017.

[57] Santos, P. et al.: *Identifying maximum imbalance in datasets for fault diagnosis of gearboxes*. In: Journal of Intelligent Manufacturing 29, 2, S. 333–351, 2018.

[58] He, H., Garcia, E. A.: *Learning from Imbalanced Data*. In: IEEE Transactions on Knowledge and Data Engineering 21, 9, S. 1263–1284, 2009.

[59] Jia, F. et al.: *Deep normalized convolutional neural network for imbalanced fault classification of machinery and its understanding via visualization*. In: Mechanical Systems and Signal Processing 110, S. 349–367, 2018.

[60] Krummenacher, G. et al.: *Wheel Defect Detection With Machine Learning*. In: IEEE Transactions on Intelligent Transportation Systems 19, 4, S. 1176–1187, 2018.

[61] LeCun, Y. et al.: *Handwritten Digit Recognition with a Back-Propagation Network*. In: 1990, S. 396–404.

[62] Lecun, Y. et al.: *Gradient-based learning applied to document recognition*. In: Proceedings of the IEEE 86, 11, S. 2278–2324, 1998.

[63] Lawrence, S. et al.: *Face recognition: a convolutional neural-network approach*. In: IEEE transactions on neural networks 8, 1, S. 98–113, 1997.

[64] Girshick, R. et al.: *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. url: http://arxiv.org/pdf/1311.2524v5.

[65] Girshick, R.: *Fast R-CNN*. 2015. url: http://arxiv.org/pdf/1504.08083v2.

[66] Ren, S. et al.: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. url: http://arxiv.org/pdf/1506.01497v3.

[67] Goodfellow, I. J. et al.: *Generative Adversarial Networks*. 2014. url: http://arxiv.org/pdf/1406.2661v1.

[68] Karpathy, A., Fei-Fei, L.: *Deep Visual-Semantic Alignments for Generating Image Descriptions*. 2014. url: http://arxiv.org/pdf/1412.2306v2.

[69] Jaderberg, M. et al.: *Spatial Transformer Networks*. 2015. url: http://arxiv.org/pdf/1506.02025v3.

[70] Szegedy, C. et al.: *Going Deeper with Convolutions*. 2015. url: http://arxiv.org/pdf/1409.4842v1.

[71] Simonyan, K., Zisserman, A.: *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. url: http://arxiv.org/pdf/1409.1556v6.

[72] He, K. et al.: *Deep Residual Learning for Image Recognition*. 2015. url: http://arxiv.org/pdf/1512.03385v1.

[73] Szegedy, C. et al.: *Rethinking the Inception Architecture for Computer Vision*. 2015. url: http://arxiv.org/pdf/1512.00567v3.

[74] Xie, S. et al.: *Aggregated Residual Transformations for Deep Neural Networks*. 2016. url: http://arxiv.org/pdf/1611.05431v2.

[75] Szegedy, C. et al.: *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. url: http://arxiv.org/pdf/1602.07261v2.

[76] He, K. et al.: *Identity Mappings in Deep Residual Networks*. 2016. url: http://arxiv.org/pdf/1603.05027v3.

[77] Huang, G. et al.: *Densely Connected Convolutional Networks*. 2016. url: http://arxiv.org/pdf/1608.06993v5.

[78] Huang, G. et al.: *Deep Networks with Stochastic Depth*. 2016. url: http://arxiv.org/pdf/1603.09382v3.

[79] Iandola, F. N. et al.: *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*. 2016. url: http://arxiv.org/pdf/1602.07360v4.

[80] Howard, A. G. et al.: *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. url: http://arxiv.org/pdf/1704.04861v1.

[81] Hu, J. et al.: *Squeeze-and-Excitation Networks*. 2017. url: http://arxiv.org/pdf/1709.01507v2.

[82] Hubel, D. H.: *Single unit activity in striate cortex of unrestrained cats*. In: The Journal of Physiology 147, 2, S. 226–238.2, 1959.

[83] Hubel, D. H., Wiesel, T. N.: *Receptive fields of single neurones in the cat's striate cortex*. In: The Journal of Physiology 148, 3, S. 574–591, 1959.

[84] Hubel, D. H., Wiesel, T. N.: *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*. In: The Journal of Physiology 160, 1, S. 106–154.2, 1962.

[85] Russakovsky, O. et al.: *ImageNet Large Scale Visual Recognition Challenge*. 2014. url: http://arxiv.org/pdf/1409.0575v3.

[86] Palaz, D. et al.: *Estimating Phoneme Class Conditional Probabilities from Raw Speech Signal using Convolutional Neural Networks*. 2013. url: http://arxiv.org/pdf/1304.1018v2.

[87] Cook, A.: *Global Average Pooling Layers for Object Localization*. url: https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/, abgerufen am: 22.8.2018.

[88] Lin, M. et al.: *Network In Network*. 2013. url: http://arxiv.org/pdf/1312.4400v3.

[89] Ioffe, S., Szegedy, C.: *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. url: http://arxiv.org/pdf/1502.03167v3.

[90] Ioffe, S.: *Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models*. 2017. url: http://arxiv.org/pdf/1702.03275v2.

[91] LeCun, Y. et al.: *LeNet-5, convolutional neural networks*. In: URL: http://yann. lecun. com/exdb/lenet S. 20, 2015.

[92] Glorot, X., Bengio, Y.: *Understanding the difficulty of training deep feedforward neural networks*. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Hrsg. von Yee Whye Teh, Mike Titterington. Bd. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, S. 249–256.

[93] Salim, G. et al.: *Vibro-acoustic fault detection and diagnosis in hybrid electric vehicle*. In: *Fourth International Conference on Power Engineering, Energy and Electrical Drives (POWERENG), 2013*. Piscataway, NJ: IEEE, 2013, S. 309–313.

[94] Murphey, Y. L. et al.: *Model-based fault diagnosis in electric drives using machine learning*. In: IEEE/ASME Transactions on Mechatronics 11, 3, S. 290–303, 2006.

[95] Kang, M., Kim, J.-M.: *Reliable Fault Diagnosis of Multiple Induction Motor Defects Using a 2-D Representation of Shannon Wavelets*. In: IEEE Transactions on Magnetics 50, 10, S. 1–13, 2014.

[96] Gonzalez, J. P. N. et al.: *Vehicle Fault Detection and Diagnosis combining AANN and ANFIS*. In: IFAC Proceedings Volumes 42, 8, S. 1079–1084, 2009.

[97] Guo, H. et al.: *Automotive signal diagnostics using wavelets and machine learning*. In: IEEE Transactions on Vehicular Technology 49, 5, S. 1650–1662, 2000.

[98] Hu, C. et al.: *Vehicle Color Recognition With Spatial Pyramid Deep Learning*. In: IEEE Transactions on Intelligent Transportation Systems 16, 5, S. 2925–2934, 2015.

[99] Hu, Q. et al.: *Deep CNNs With Spatially Weighted Pooling for Fine-Grained Car Recognition*. In: IEEE Transactions on Intelligent Transportation Systems 18, 11, S. 3147–3156, 2017.

[100] Zeng, Y. et al.: *Traffic Sign Recognition Using Kernel Extreme Learning Machines With Deep Perceptual Features*. In: IEEE Transactions on Intelligent Transportation Systems S. 1–7, 2016.

[101] Schlosser, J. et al.: *Fusing LIDAR and images for pedestrian detection using convolutional neural networks*. In: *2016 IEEE International Conference on Robotics and Automation, Stockholm, Sweden, May 16th-21st*. Hrsg. von Okamura, A., Menciassi, A. Piscataway, NJ: IEEE, 2016, S. 2198–2205.

[102] Gaisser, F., Jonker, P. P.: *Road user detection with convolutional neural networks: An application to the autonomous shuttle WEpod*. In: *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*. IEEE, 2017, S. 101–104.

[103] Bojarski, M. et al.: *End to End Learning for Self-Driving Cars*. 2016. url: http://arxiv.org/pdf/1604.07316v1.

[104] Fayjie, A. R. et al.: *Driverless Car: Autonomous Driving Using Deep Reinforcement Learning in Urban Environment*. In: *2018 15th International Conference on Ubiquitous Robots (UR)*. Piscataway, NJ: IEEE, 2018, S. 896–901.

[105] Koh, J. J. et al.: *Autonomous Road Potholes Detection on Video*. In: *Computational science and technology*. Hrsg. von Alfred, R. et al. Bd. 481. Lecture Notes in Electrical Engineering. Singapore: Springer, 2019, S. 137–143.

[106] Kumar, A. K. T. R. et al.: *Indoor localization of vehicles using Deep Learning*. In: *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2016, S. 1–6.

[107] Dwivedi, K. et al.: *Drowsy driver detection using representation learning*. In: *2014 IEEE International Advance Computing Conference (IACC)*. IEEE, 2014, S. 995–999.

[108] Le, T. H. N. et al.: *DeepSafeDrive: A grammar-aware driver parsing approach to Driver Behavioral Situational Awareness (DB-SAW)*. In: Pattern Recognition 66, S. 229–238, 2017.

[109] Jia, Y. et al.: *Traffic speed prediction using deep learning method*. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, S. 1217–1222.

[110] Polson, N. G., Sokolov, V. O.: *Deep learning for short-term traffic flow prediction*. In: Transportation Research Part C: Emerging Technologies 79, S. 1–17, 2017.

[111] Rekabdar, B.: *Anticipating Maneuvers with Dilated Convolutions*. In: *HRI'18*. Hrsg. von Kanda, T. et al. New York, New York: Association for Computing Machinery, 2018, S. 213–214.

[112] Ma, R. et al.: *Data-driven proton exchange membrane fuel cell degradation predication through deep learning method*. In: Applied Energy 231, S. 102–115, 2018.

[113] Giering, M. et al.: *Multi-modal sensor registration for vehicle perception via deep neural networks*. In: *2015 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2015, S. 1–6.

[114] Yin, J., Zhao, W.: *Fault diagnosis network design for vehicle on-board equipments of high-speed railway: A deep learning approach*. In: Engineering Applications of Artificial Intelligence 56, S. 250–259, 2016.

[115] Kanarachos, S. et al.: *Detecting anomalies in time series data via a deep learning algorithm combining wavelets, neural networks and Hilbert transform*. In: Expert Systems with Applications 85, S. 292–304, 2017.

[116] Liao, Y. et al.: *Wavelet transform based convolutional neural network for gearbox fault classification*. In: *2017 Prognostics and System Health Management Conference (PHM-Harbin)*. IEEE, 2017, S. 1–6.

[117] van Tran, T. et al.: *An approach to fault diagnosis of reciprocating compressor valves using Teager–Kaiser energy operator and deep belief networks*. In: Expert Systems with Applications 41, 9, S. 4113–4122, 2014.

[118] Shao, H. et al.: *Rolling bearing fault diagnosis using an optimization deep belief network*. In: Measurement Science and Technology 26, 11, S. 115002, 2015.

[119] Jia, F. et al.: *Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data*. In: Mechanical Systems and Signal Processing 72-73, S. 303–315, 2016.

[120] Liu, H. et al.: *Rolling Bearing Fault Diagnosis Based on STFT-Deep Learning and Sound Signals*. In: Shock and Vibration 2016, 2, S. 1–12, 2016.

[121] Lei, Y. et al.: *An Intelligent Fault Diagnosis Method Using Unsupervised Feature Learning Towards Mechanical Big Data*. In: IEEE Transactions on Industrial Electronics 63, 5, S. 3137–3147, 2016.

[122] Sun, W. et al.: *A sparse auto-encoder-based deep neural network approach for induction motor faults classification*. In: Measurement 89, S. 171–178, 2016.

[123] Chen, Z. et al.: *Machine fault classification using deep belief network*. In: *2016 IEEE International Instrumentation and Measurement Technology Conference Proceedings*. IEEE, 2016, S. 1–6.

[124] Liu, J. et al.: *An integrated multi-sensor fusion-based deep feature learning approach for rotating machinery diagnosis*. In: Measurement Science and Technology 29, 5, S. 1–12, 2018.

[125] Zhang, W. et al.: *Deep convolutional neural networks for multi-modality isointense infant brain image segmentation*. In: NeuroImage 108, S. 214–224, 2015.

[126] Moeskops, P. et al.: *Automatic Segmentation of MR Brain Images With a Convolutional Neural Network*. In: IEEE transactions on medical imaging 35, 5, S. 1252–1261, 2016.

[127] Zahia, S. et al.: *Tissue classification and segmentation of pressure injuries using convolutional neural networks*. In: Computer methods and programs in biomedicine 159, S. 51–58, 2018.

[128] Pang, S. et al.: *A novel biomedical image indexing and retrieval system via deep preference learning*. In: Computer methods and programs in biomedicine 158, S. 53–69, 2018.

[129] Zhao, X. et al.: *Agile convolutional neural network for pulmonary nodule classification using CT images*. In: International journal of computer assisted radiology and surgery 13, 4, S. 585–595, 2018.

[130] Li, X. et al.: *Remaining useful life estimation in prognostics using deep convolution neural networks*. In: Reliability Engineering & System Safety 172, S. 1–11, 2018.

[131] Sateesh Babu, G. et al.: *Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life*. In: *Database Systems for Advanced Applications*. Hrsg. von Navathe, S. B. et al. Bd. 9642. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, S. 214–228.

[132] Zhao, R. et al.: *Learning to Monitor Machine Health with Convolutional Bi-Directional LSTM Networks*. In: Sensors (Basel, Switzerland) 17, 2, 2017.

[133] Appana, D. K. et al.: *Speed Invariant Bearing Fault Characterization Using Convolutional Neural Networks*. In: *Multi-disciplinary Trends in Artificial Intelligence*. Hrsg. von Phon-Amnuaisuk, S. et al. Bd. 10607. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, S. 189–198.

[134] Eren, L.: *Bearing Fault Detection by One-Dimensional Convolutional Neural Networks*. In: Mathematical Problems in Engineering 2017, S. 1–9, 2017.

[135] Ince, T. et al.: *Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks*. In: IEEE Transactions on Industrial Electronics 63, 11, S. 7067–7075, 2016.

[136] Pan, J. et al.: *LiftingNet: A Novel Deep Learning Network With Layerwise Feature Learning From Noisy Mechanical Data for Fault Classification*. In: IEEE Transactions on Industrial Electronics 65, 6, S. 4973–4982, 2018.

[137] Zhang, W. et al.: *A New Deep Learning Model for Fault Diagnosis with Good Anti-Noise and Domain Adaptation Ability on Raw Vibration Signals*. In: Sensors (Basel, Switzerland) 17, 2, 2017.

[138] Zhang, W. et al.: *Rolling Element Bearings Fault Intelligent Diagnosis Based on Convolutional Neural Networks Using Raw Sensing Signal*. In: *Advances in Intelligent Information Hiding and Multimedia Signal Processing*. Hrsg. von Pan, J.-S. et al. Bd. 64. Smart Innovation, Systems and Technologies. Cham: Springer International Publishing, 2017, S. 77–84.

[139] Zhang, W. et al.: *A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load*. In: Mechanical Systems and Signal Processing 100, S. 439–453, 2018.

[140] Guo, X. et al.: *Hierarchical adaptive deep convolution neural network and its application to bearing fault diagnosis*. In: Measurement 93, S. 490–502, 2016.

[141] Hoang, D.-T., Kang, H.-J.: *Convolutional Neural Network Based Bearing Fault Diagnosis*. In: *Intelligent Computing Theories and Application*. Hrsg. von Huang, D.-S. et al. Bd. 10362. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, S. 105–111.

[142] Verstraete, D. et al.: *Deep Learning Enabled Fault Diagnosis Using Time-Frequency Image Analysis of Rolling Element Bearings*. In: Shock and Vibration 2017, S. 1–17, 2017.

[143] Zhang, W. et al.: *Bearings Fault Diagnosis Based on Convolutional Neural Networks with 2-D Representation of Vibration Signals as Input*. In: MATEC Web of Conferences 95, 8, S. 13001, 2017.

[144] Kiranyaz, S. et al.: *Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks*. In: IEEE transactions on bio-medical engineering 63, 3, S. 664–675, 2016.

[145] Avci, O. et al.: *Structural Damage Detection in Real Time: Implementation of 1D Convolutional Neural Networks for SHM Applications*. In: *Structural Health Monitoring & Damage Detection, Volume 7*. Hrsg. von Niezrecki, C. Bd. 2015. Conference Proceedings of the Society for Experimental Mechanics Series. Cham: Springer International Publishing, 2017, S. 49–54.

[146] Ji, S. et al.: *3D convolutional neural networks for human action recognition*. In: IEEE transactions on pattern analysis and machine intelligence 35, 1, S. 221–231, 2013.

[147] Kamnitsas, K. et al.: *Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation*. In: Medical image analysis 36, S. 61–78, 2017.

[148] Jing, L. et al.: *An Adaptive Multi-Sensor Data Fusion Method Based on Deep Convolutional Neural Networks for Fault Diagnosis of Planetary Gearbox*. In: Sensors (Basel, Switzerland) 17, 2, 2017.

[149] Lee, D. et al.: *Convolutional neural net and bearing fault analysis*. In: *Proceedings of the International Conference on Data Mining (DMIN)*. 2016, S. 194–200.

[150] Wen, L. et al.: *A New Convolutional Neural Network-Based Data-Driven Fault Diagnosis Method*. In: IEEE Transactions on Industrial Electronics 65, 7, S. 5990–5998, 2018.

[151] Wang, Z., Oates, T.: *Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks*. In: *Trajectory-Based Behavior Analytics*. Technical report / Association for the Advancement of Artificial Intelligence WS. Palo Alto, California: AAAI Press, 2015, S. 40–46.

[152] Smith, J. O.: *Mathematics of the discrete Fourier transform (DFT): With audio applications*. 2. Aufl. North Charleston: BookSurge, 2008.

[153] Mertins, A.: *Kurzzeit-Fourier-Transformation*. In: *Signaltheorie*. Hrsg. von Mertins, A. Wiesbaden: Springer Fachmedien Wiesbaden, 2013, S. 285–300.

[154] Ding, X., He, Q.: *Energy-Fluctuated Multiscale Feature Learning With Deep ConvNet for Intelligent Spindle Bearing Fault Diagnosis*. In: IEEE Transactions on Instrumentation and Measurement 66, 8, S. 1926–1935, 2017.

[155] He, K. et al.: *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. url: http://arxiv.org/pdf/1502.01852v1.

[156] Mishkin, D. et al.: *Systematic evaluation of convolution neural network advances on the Imagenet*. In: Computer Vision and Image Understanding 161, S. 11–19, 2017.

[157] Chollet, F.: *Xception: Deep Learning with Depthwise Separable Convolutions*. 2016. url: http://arxiv.org/pdf/1610.02357v3.

[158] Cooley, J. W., Tukey, J. W.: *An Algorithm for the Machine Calculation of Complex Fourier Series*. In: Mathematics of Computation 19, 90, S. 297, 1965.

[159] Liu, Z.: *Signalverarbeitungsverfahren und virtuelle Instrumente zur Messung von elektrischen Signalen und zur Fehlerdiagnose an Maschinen*. Dissertation. Siegen: Universität Siegen, 2006.

[160] Xiang, L., Hu, A.: *Comparison of Methods for Different Time-frequency Analysis of Vibration Signal*. In: JSW 7, S. 68–74, 2012.

[161] Sifuzzaman, M. et al.: *Application of wavelet transform and its advantages compared to Fourier transform*. In: Journal of Physical Science; Band 13 [2009] S. 121–134, 2009.

[162] Sengur, A. et al.: *Wavelet packet neural networks for texture classification*. In: Expert Systems with Applications 32, 2, S. 527–533, 2007.

[163] Han, J.-G. et al.: *Wavelet packet based damage identification of beam structures*. In: International Journal of Solids and Structures 42, 26, S. 6610–6627, 2005.

[164] Algernon, D., Wiggenhauser, H.: *Anwendung der Hilbert-Huang-Transformation zur Auswertung von Impact-Echo-Datensätzen*. In: *DGZfP-Berichtsband*. Hrsg. von Bundesanstalt für Materialforschung. Bd. 94-CD. Berichtsband / Deutsche Gesellschaft für Zerstörungsfreie Prüfung e.V. Berlin: Deutsche Gesellschaft für Zerstörungsfreie Prüfung (DGZfP), 2005, S. 1–13.

[165] Hatami, N. et al.: *Classification of Time-Series Images Using Deep Convolutional Neural Networks*. 2017. url: http://arxiv.org/pdf/1710.00886v2.

[166] Längkvist, M. et al.: *A review of unsupervised feature learning and deep learning for time-series modeling*. In: Pattern Recognition Letters 42, S. 11–24, 2014.

[167] Möser, M.: *Digitale Signalverarbeitung*. Vorlesungsfolien. Berlin: Technische Universität Berlin, 2011. (Besucht am 19. 09. 2018).

[168] Wang, Z., Oates, T.: *Imaging Time-Series to Improve Classification and Imputation*. 2015. url: http://arxiv.org/pdf/1506.00327v1.

[169] Faouzi, J.: *pyts: a Python package for time series transformation and classification*. 2018.

[170] Marwan, N. et al.: *Recurrence plots for the analysis of complex systems*. In: Physics Reports 438, 5, S. 237–329, 2007.

[171] Daubechies, I.: *The wavelet transform, time-frequency localization and signal analysis*. In: IEEE Transactions on Information Theory 36, 5, S. 961–1005, 1990.

[172] van Fleet, P. J.: *Discrete wavelet transformations: An elementary approach with applications*. Hoboken, NJ: Wiley, 2008.

[173] Banko, M., Brill, E.: *Scaling to very very large corpora for natural language disambiguation*. In: *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics - ACL '01*. Hrsg. von Webber, B. L. Morristown, NJ, USA: Association for Computational Linguistics, 2001, S. 26–33.

[174] Halevy, A. et al.: *The Unreasonable Effectiveness of Data*. In: IEEE Intelligent Systems 24, 2, S. 8–12, 2009.

[175] Sun, C. et al.: *Revisiting Unreasonable Effectiveness of Data in Deep Learning Era*. 2017. url: http://arxiv.org/pdf/1707.02968v2.

[176] The TensorFlow Authors: *TensorFlow API 1.9 Dokumentation tf.losses: Sparse Softmax Cross Entropy*. url: https://www.tensorflow.org/versions/r1.9/api_docs/python/tf/losses/sparse_softmax_cross_entropy, abgerufen am: 1.10.2018.

[177] Maas, A. L. et al.: *Rectifier nonlinearities improve neural network acoustic models*. In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[178] Shang, W. et al.: *Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units*. 2016. url: http://arxiv.org/pdf/1603.05201v2.

[179] Clevert, D.-A. et al.: *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2015. url: http://arxiv.org/pdf/1511.07289v5.

[180] Klambauer, G. et al.: *Self-Normalizing Neural Networks* 2017.

[181] Krizhevsky, A.: *Convolutional Deep Belief Networks on CIFAR-10*. url: http://www.cs.utoronto.ca/~kriz/conv-cifar10-aug2010.pdf..

[182] Li, Y. et al.: *Revisiting Batch Normalization For Practical Domain Adaptation*. 2016. url: http://arxiv.org/pdf/1603.04779v4.

[183] van Laarhoven, T.: *L2 Regularization versus Batch and Weight Normalization*. 2017. url: http://arxiv.org/pdf/1706.05350v1.

[184] Choi, K. et al.: *Convolutional Recurrent Neural Networks for Music Classification*. 2016. url: http://arxiv.org/pdf/1609.04243v3.

[185] Zihlmann, M. et al.: *Convolutional Recurrent Neural Networks for Electrocardiogram Classification*. 2017. url: http://arxiv.org/pdf/1710.06122v2.

[186] Yao, S. et al.: *DeepSense: A Unified Deep Learning Framework for Time-Series Mobile Sensing Data Processing*. In: *Proceedings of the 26th International Conference on World Wide Web*. Hrsg. von Barrett, R. New York: Association for Computing Machinery, 2017, S. 351–360.

[187] Zeiler, M. D., Fergus, R.: *Visualizing and Understanding Convolutional Networks*. In: *Computer Vision – ECCV 2014*. Hrsg. von Fleet, D. et al. Bd. 8689. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, S. 818–833.

[188] Mishkin, D., Matas, J.: *All you need is a good init*. 2015. url: http://arxiv.org/pdf/1511.06422v7.

[189] Li, H. et al.: *Pruning Filters for Efficient ConvNets*. 2016. url: http://arxiv.org/pdf/1608.08710v3.

[190] Rippel, O. et al.: *Spectral Representations for Convolutional Neural Networks*. 2015. url: http://arxiv.org/pdf/1506.03767v1.

[191] BSIC: *The Automotive Industry Revolution - BSIC | Bocconi Students Investment Club*. url: http://www.bsic.it/automotive-industry-revolution/, abgerufen am: 29.10.2018.

[192] Broad, T.: *Visualisation of Convolutional Neural Networks Topology*. url: http://terencebroad.com/convnetvis/vis.html, abgerufen am: 29.10.2018.

Anhang

A Anhang

Das Titelbild wurde auf Basis von [191, 192] erstellt.

A.1 Anhang zu Kapitel 3

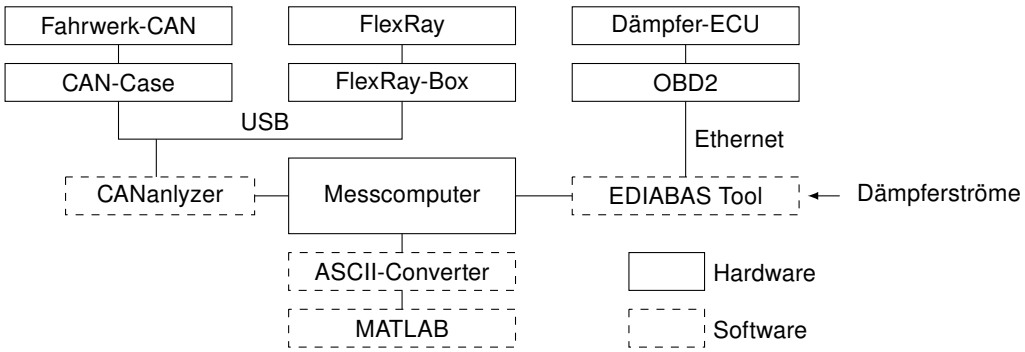


Abbildung A.1: Messkette zur Datengewinnung nach [5, S. 30]

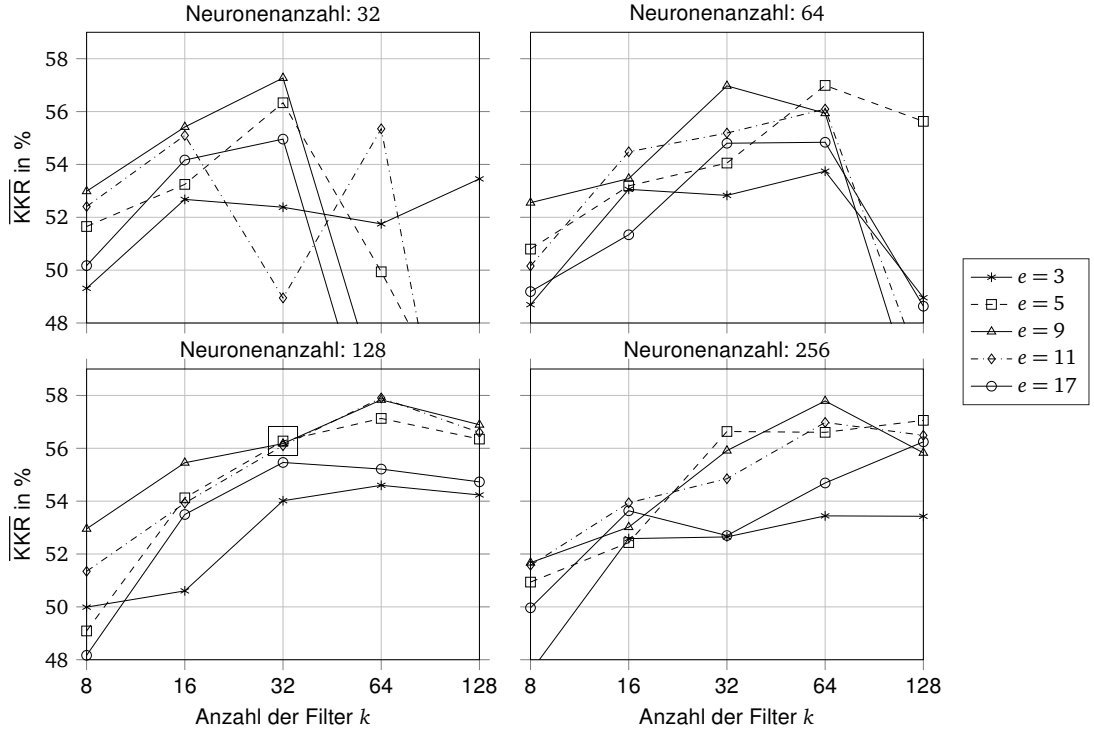


Abbildung A.2: Fünffach kreuzvalidierte Ergebnisse der Rastersuche für die Filteranzahl bei variierender Filtergröße im Baseline-Modell (Ergänzende Darstellung zu Abbildung 3.4). Der große Marker markiert die Erstkonfiguration des Modells aus Tabelle 3.3.

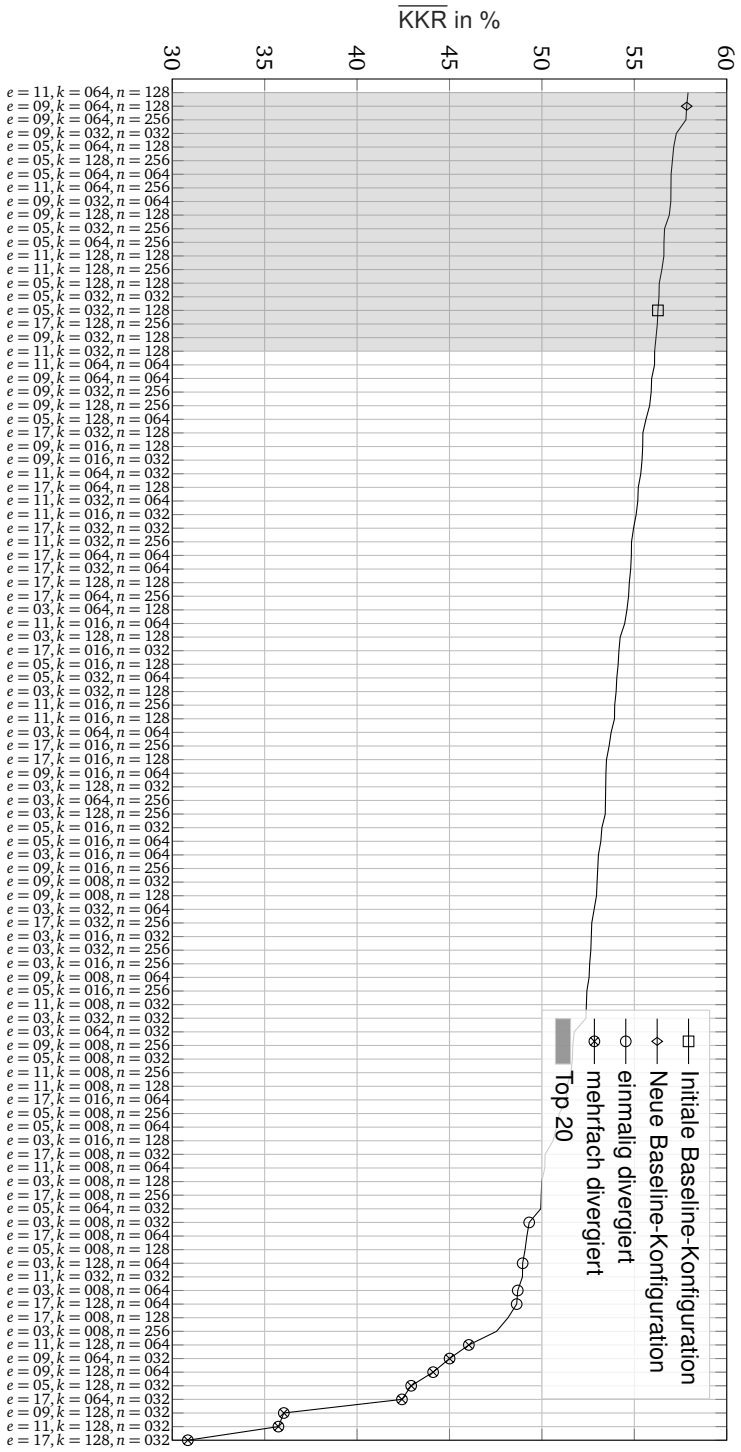


Abbildung A.3: Fünffach kreuzvalidierte Ergebnisse der Rastersuche am Baseline-Modell absteigend sortiert nach Korrektklassifizierungsrate

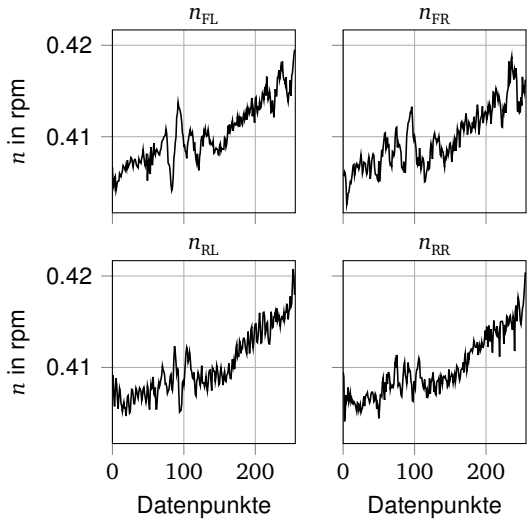


Abbildung A.4: Datenbeispiel (skaliert über den Datensatz) mit Label „FL“ (defekt), Segmentlänge 256 Datenpunkte

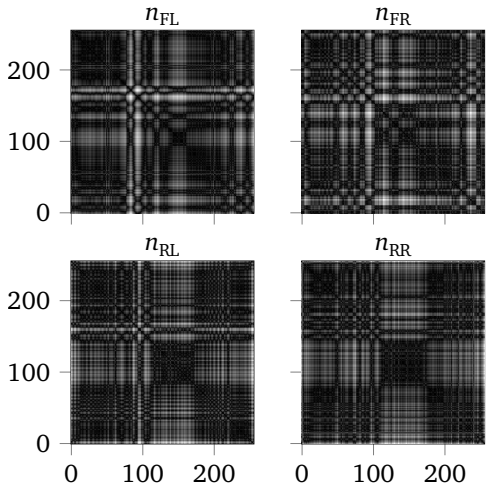


Abbildung A.5: Rekurrenzplot des Datenbeispiels mit Label „FL“ (defekt), Segmentlänge 256 Datenpunkte

Tabelle A.1: Übersicht der implementierten LeNet-Architektur. In allen Layern mit Aktivierungsfunktionen (außer Output) werden ReLU-Funktionen eingesetzt.

Layer	k @ e / s	Padding
Input		
Convolutional	6 @ $5 \times 5 \times d_{\mathrm{in}}$ / 1	same
MaxPool	2×2 / 2	valid
Convolutional	16 @ $5 \times 5 \times 6$ / 1	valid
MaxPool	2×2 / 2	valid
FC	120 Neuronen	
Dropout		
FC	84 Neuronen	
Dropout		
Output (FC)	4 Neuronen	

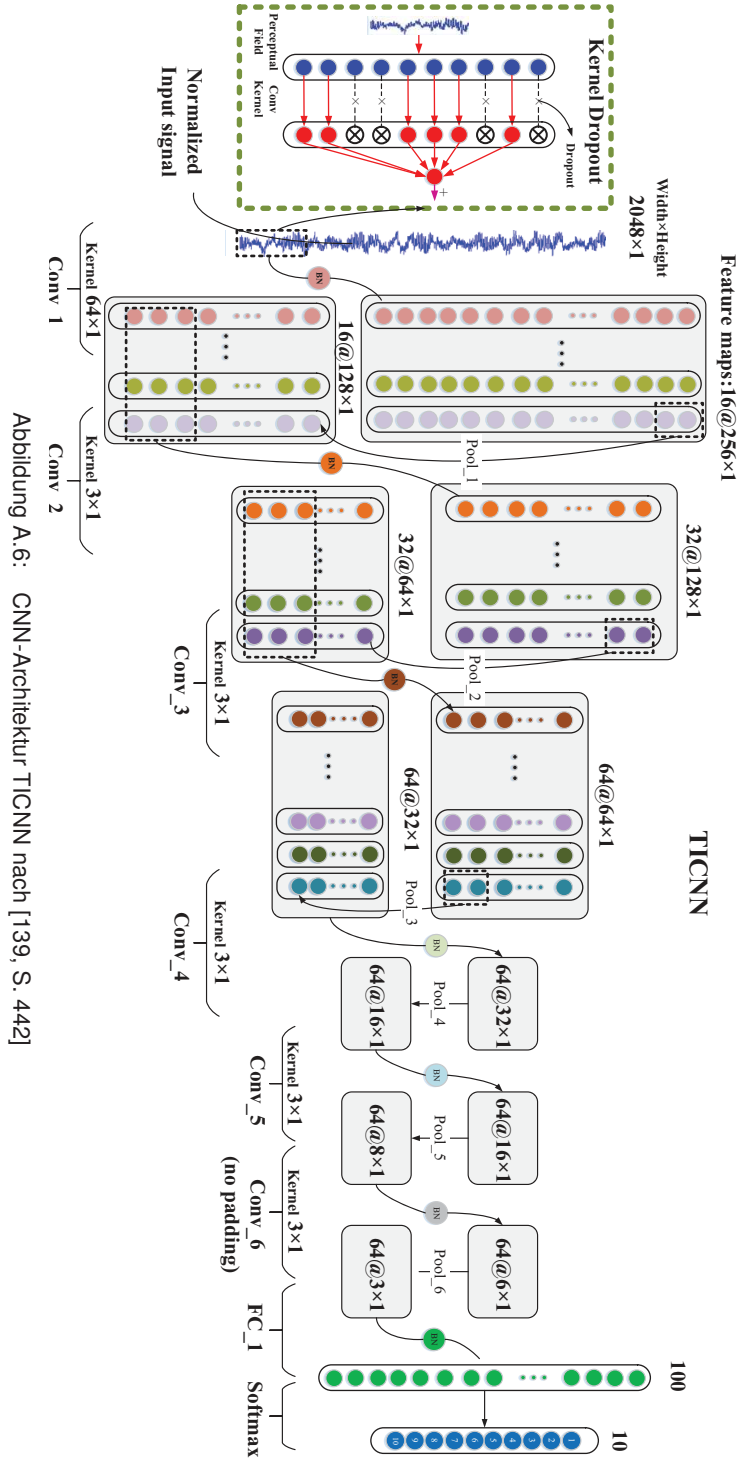


Abbildung A.6: CNN-Architektur TICNN nach [139, S. 442]

A.2 Anhang zu Kapitel 4

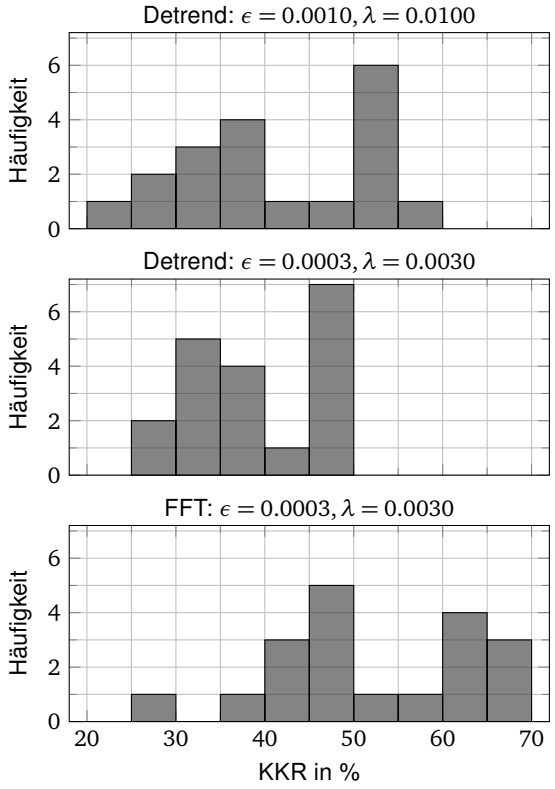


Abbildung A.7: Histogramme der Ergebnisse der Datenkanalauswahl

Tabelle A.2: Ergebnisse der Architektur-Variation, fünffach kreuzvalidiert. „SepConv“ bezeichnet das Baseline-Modell mit Depthwise Separable Convolution wie es in Abschnitt 4.1.2 verwendet wurde.

Netz	$\overline{\textbf{KKR}}$ **in %**		**Kommentar**
	1-D / FFT	2-D / STFT	
Baseline	$64,74 \pm 2,08$	$72,03 \pm 2,07$	
SepConv	$68,15 \pm 1,43$	$70,91 \pm 1,61$	
DamNet	$70,11 \pm 2,35$	$76,42 \pm 1,92$	
Verstraete	$69,68 \pm 1,84$	$59,21 \pm 6,29$	2-D mit $\lambda = 0$, da sonst keine Konvergenz
TICNN	$59,50 \pm 1,83$	$25,30 \pm 1,10$	$62,41 \pm 1,89$ mit $b = 16$ (FFT), 2-D konvergiert bei Trainingsdaten
SqueezeNet	$41,52 \pm 4,65$	$48,13 \pm 2,54$	2-D mit $\epsilon = 0.0001$, da sonst keine Konvergenz
LeNet	$60,83 \pm 3,35$	$73,34 \pm 1,97$	$67,52 \pm 4,60$ mit Mean-Pooling (STFT)
SE-ResNeXt	36.06 ± 1.66	$26,80 \pm 2,31$	1-D nur drei Durchläufe, Overfitting bei Trainingsdaten

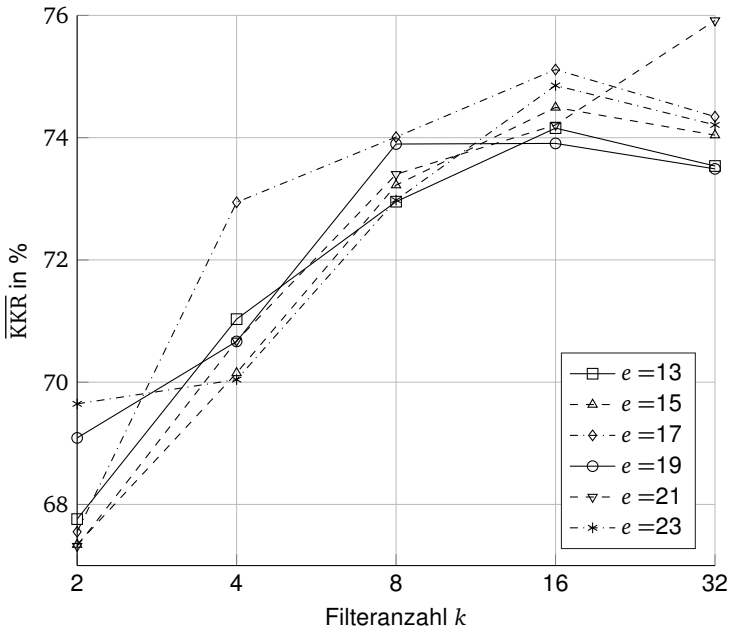


Abbildung A.8: Ergebnisse der Zusatzuntersuchung architektonischer Einflüsse anhand des 2-D-Baseline-Modells, fünffach kreuzvalidiert.

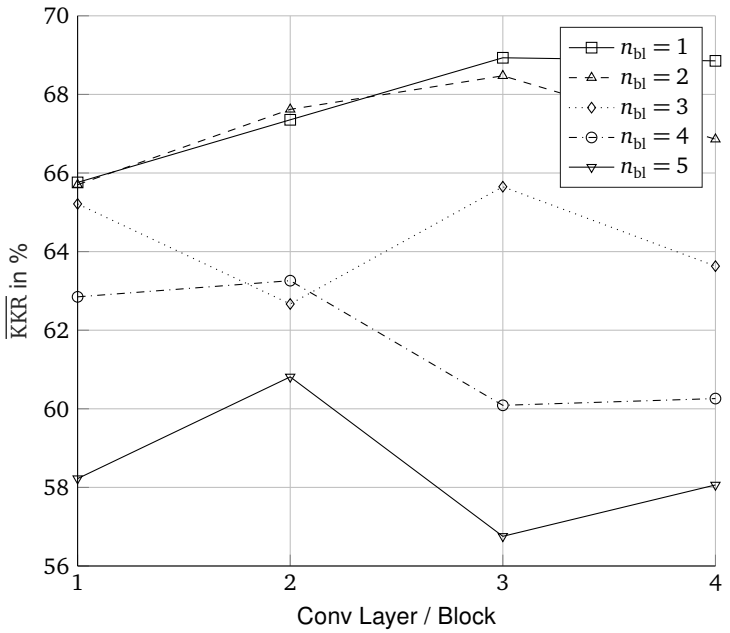


Abbildung A.9: Zusätzliche Darstellung der Ergebnisse zur Tiefeneinfluss-Untersuchung anhand des 1-D-Baseline-Modells, fünffach kreuzvalidiert. n_{bl} bezeichnet die Anzahl der Blöcke aus Convolution und Max-Pooling.

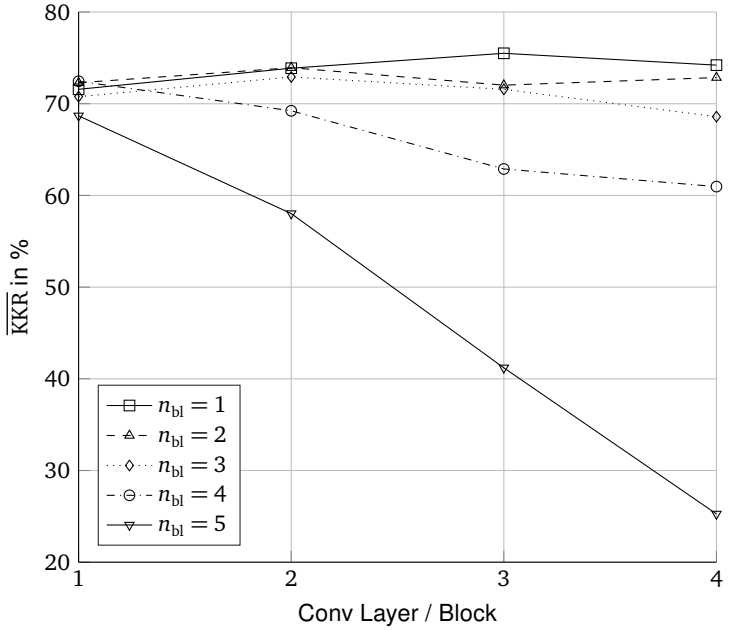


Abbildung A.10: Zusätzliche Darstellung der Ergebnisse zur Tiefeneinfluss-Untersuchung anhand des 2-D-Baseline-Modells, fünffach kreuzvalidiert. n_{bl} bezeichnet die Anzahl der Blöcke aus Convolution und Max-Pooling. Die Ergebnisse wurden mit $\epsilon = 0.0001$ bestimmt.

Tabelle A.3: Architektur des aktualisierten Baseline-Modells. In allen Layern mit Gewichten (außer Output) werden ReLU-Aktivierungsfunktionen verwendet.

Layer	**1-D**		**2-D**	
	k @ $e \times f_{\mathrm{d}}$ / s	Padding	k @ $e \times f_{\mathrm{d}}$ / s	Padding
Input				
Convolutional	16 @ 3 × 7 / 1	same	16 @ 11 × 7 / 1	same
Convolutional	16 @ 3 × 16 / 1	same	16 @ 3 × 16 / 1	same
MaxPool	2 × 1 / 2	valid	2 × 1 / 2	valid
Convolutional	32 @ 3 × 16 / 1	same	32 @ 3 × 16 / 1	same
Convolutional	32 @ 3 × 32 / 1	same	32 @ 3 × 32 / 1	same
MaxPool	2 × 1 / 2	valid	2 × 1 / 2	valid
Fully Connected (FC)	128 Neuronen	-	128 Neuronen	-
Dropout				
Output (FC)	4 Neuronen	-	4 Neuronen	-

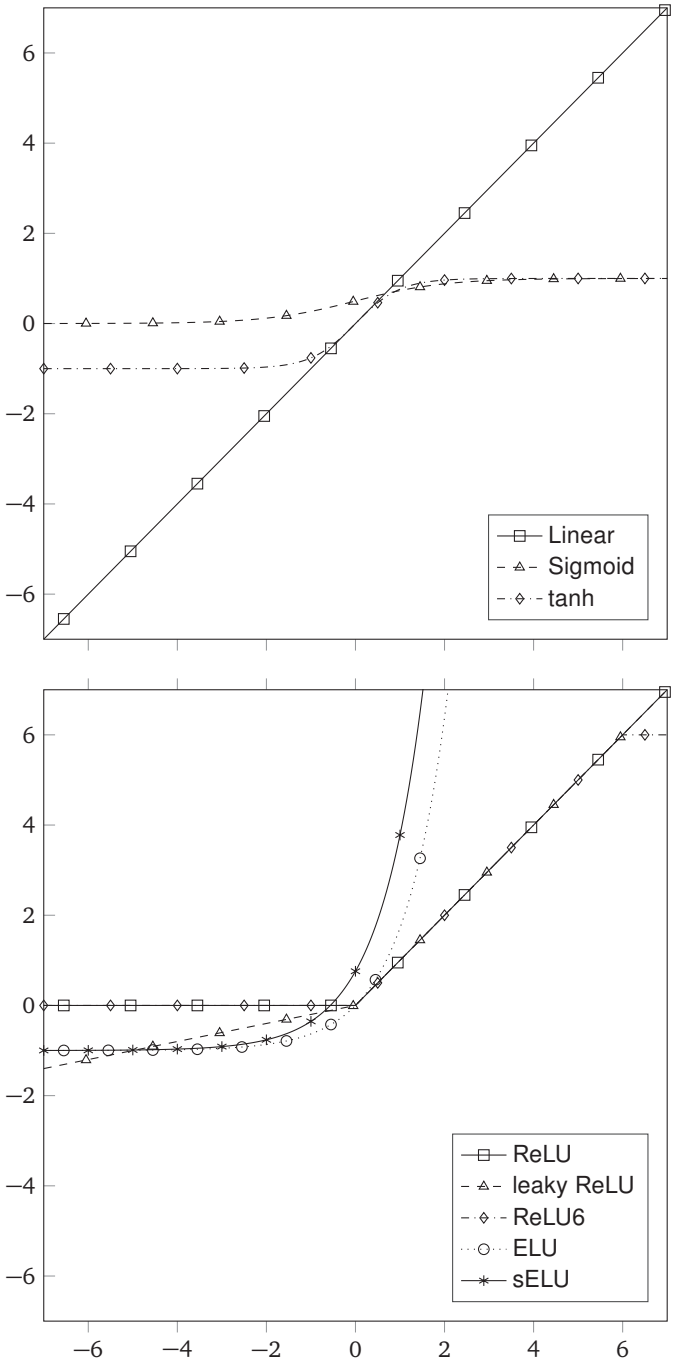


Abbildung A.11: Darstellung unterschiedlicher Aktivierungsfunktionen im Intervall $[-7, 7]$ als Ergänzung zu Tabelle 4.6.

B Code

B.1 Requirements.txt

Code B.1: Übersicht aller verwendeten Python-Module

```
absl-py==0.3.0
astor==0.7.1
backcall==0.1.0
bleach==2.1.3
certifi==2018.4.16
chardet==3.0.4
cloudpickle==0.5.3
colorama==0.3.9
cycler==0.10.0
dask==0.18.2
decorator==4.3.0
entrypoints==0.2.3
future==0.16.0
gast==0.2.0
grpcio==1.13.0
html5lib==1.0.1
idna==2.7
ipykernel==4.8.2
ipython==6.5.0
ipython-genutils==0.2.0
ipywidgets==7.3.1
jedi==0.12.1
Jinja2==2.10
jsonschema==2.6.0
jupyter==1.0.0
jupyter-client==5.2.3
jupyter-console==5.2.0
jupyter-core==4.4.0
kiwisolver==1.0.1
Markdown==2.6.11
MarkupSafe==1.0
matplotlib==2.2.2
matplotlib2tikz==0.6.17
mistune==0.8.3
nbconvert==5.3.1
nbformat==4.4.0
networkx==2.1
notebook==5.6.0
numpy==1.15.0
pandas==0.23.3
pandocfilters==1.4.2
parso==0.3.1
pickleshare==0.7.4
Pillow==5.2.0
```

```
plotly==3.1.0
prometheus-client==0.3.1
prompt-toolkit==1.0.15
protobuf==3.6.0
Pygments==2.2.0
pyparsing==2.2.0
python-dateutil==2.7.3
pyts==0.7.1
pytz==2018.5
PyWavelets==0.5.2
pywinpty==0.5.4
pyzmq==17.1.0
qtconsole==4.3.1
requests==2.19.1
retrying==1.3.3
scikit-image==0.14.0
scikit-learn==0.19.2
scipy==1.1.0
Send2Trash==1.5.0
simplegeneric==0.8.1
six==1.11.0
tensorboard==1.9.0
tensorflow-gpu==1.9.0
termcolor==1.1.0
terminado==0.8.1
testpath==0.3.1
toolz==0.9.0
tornado==5.1
traitlets==4.3.2
urllib3==1.23
wcwidth==0.1.7
webencodings==0.5.1
Werkzeug==0.14.1
widgetsnbextension==3.3.1
```