

Hochschule Furtwangen University

Ausarbeitung Präzisionsmesstechnik

KI in der Messtechnik

Autoren:

Manuel Caipo (286577)

M.Sc. Advanced Precision Engineering

Dozent: Prof. Dr.-Ing. Gunter Ketterer

20. Juni 2025 - 78056 Villingen-Schwenningen

Inhaltsverzeichnis

1	Einleitung	1
2	Theoretische Grundlagen	2
2.1	Grundlagen der Messtechnik	2
2.1.1	Definition	2
2.1.2	Aufgaben der Messtechnik:	3
2.1.3	Messgrößen und deren Klassifizierung	3
2.1.4	Messsignal	4
2.1.5	Messsysteme und Messketten	6
2.1.6	Messabweichungen und Messunsicherheiten	6
2.1.7	Herausforderungen moderner Messtechnik	9
2.2	Grundlagen der Künstliche Intelligenz	9
2.2.1	Künstliche Intelligenz (KI)	9
2.2.2	Maschinelles Lernen (ML)	10
2.2.3	Definitionen und Konzepte	10
2.2.4	Anwendungsfelder der KI	10
2.2.5	Klassifikation von ML-Modellen	10
2.2.6	Methoden des Maschinellen Lernens	12
2.2.7	Übergang zu Neuronalen Netzwerken	14
2.2.8	Neuronale Netzwerkarchitekturen: Perzeptron, MLP, CNN, RNN	16
2.2.9	Vergleich der Architekturtypen	20
2.2.10	Anwendungen von Künstlicher Intelligenz in messtechnischen Kontexten	20
2.3	Grundlagen der Bildvorverarbeitung	21
2.3.1	Grundlagen der digitalen Bildrepräsentation	21
2.3.2	Bildvorverarbeitungstechniken	22
2.3.3	Mathematische Grundlagen der Kantendetektion	23
2.3.4	Die Rolle der Kantendetektion	24
3	KI in der Messtechnik	26
3.1	Objektlokalisierung mittels Computer Vision	26
3.1.1	Assistive Technologien für sehbehinderte Menschen	27
3.1.2	Anwendung von Convolutional Neural Networks (CNNs) zur Positionsschätzung	28
3.1.3	Vorverarbeitungsschritte	29
3.1.4	Modelle für die Koordinatenregression	32
3.1.5	Robustheitsüberlegungen: Beleuchtung, Verdeckung, Echtzeit-Feedback	33
3.1.6	Beispielarchitekturen	34
3.1.7	Signale zur Anomalieerkennung	36
3.2	Algorithmen zur Kantenerkennung und KI-Verbesserungen	37
3.2.1	Grundlagen der Kantenerkennung	38
3.2.2	Arten von Kanten	38
3.2.3	Klassische Kantenerkennungsalgorithmen	39

3.2.4	Anwendungen in der Hochpräzisionsmesstechnik	41
3.2.5	Beschränkungen der klassischen Kantenerkennungsmethoden	44
3.2.6	Herausforderungen bei der Kalibrierung	46
3.2.7	AI-gestützte Verbesserungen bei der Kantenerkennung	47
3.2.8	Hybride Ansätze: Kombination von klassischen und AI-Methoden	50
3.2.9	Weitere wichtige Anwendungen	52
4	Netzwerkstrategien und ihre Bewertung	58
4.1	Architekturauswahl und Designkriterien	58
4.1.1	Aktivierungsfunktionen	61
4.1.2	Optimierungsmethoden: SGD, Adam	63
4.1.3	Regularisierung: Dropout und L2-Regularisierung	64
4.2	Leistungsbewertung von Netzwerkstrategien	66
4.2.1	Modellevaluierungsmetriken	67
4.2.2	Techniken zur Kreuzvalidierung und Überanpassungskontrolle	67
5	Neuronale Netze Programmierung in MATLAB	69
5.1	Regressions-Neuronales Netz	70
5.1.1	Datenerzeugung und Normalisierung	71
5.1.2	Netzwerkarchitektur und Initialisierung	72
5.1.3	Aktivierungsfunktion	73
5.1.4	Hyperparameter	74
5.1.5	Trainingsprozess (Forward und Backward Pass)	75
5.1.6	Test des Netzwerks mit einem Beispiel	77
5.1.7	Test des Netzwerks mit mehreren Beispielen	78
5.1.8	Vollständiger MATLAB-Code	81
5.2	Neuronales Netzwerk zur Klassifikation dreier Klassen	84
5.2.1	Datenerzeugung und Normalisierung	85
5.2.2	Zielwerte mit One-Hot-Encoding	86
5.2.3	Netzwerkarchitektur und Initialisierung	87
5.2.4	Hyperparameter	87
5.2.5	Trainingsprozess (Forward- und Backward-Pass)	88
5.2.6	Test mit Beispielen und Genauigkeit	90
5.2.7	Aktivierungsfunktionen	92
5.2.8	Vollständiger MATLAB-Code	93
5.3	Neuronales Netzwerk zur Erkennung handschriftlicher Ziffern	96
5.3.1	Datenladen und Visualisierung	97
5.3.2	Datenvorverarbeitung	98
5.3.3	Netzwerkarchitektur	99
5.3.4	Aktivierungsfunktionen	99
5.3.5	Trainingsprozess	100
5.3.6	Testen des Netzwerks	102
5.3.7	Visualisierung der Ergebnisse	103
5.3.8	Vollständiger MATLAB-Code	103
5.4	Neurales Netzwerk zur Klassifizierung von 3D-Spiralen	106
5.4.1	Datenladen	107
5.4.2	Datenvorverarbeitung	108
5.4.3	Netzwerkarchitektur	109
5.4.4	Aktivierungsfunktionen	110
5.4.5	Training	111
5.4.6	Auswertung	113
5.4.7	Ergebnisvisualisierung	114

5.4.8	Vollständiger MATLAB-Code	116
5.5	Automatische Durchmesserbestimmung mit Sobel-Kantendetektion	118
5.5.1	Bild laden und vorverarbeiten	119
5.5.2	Sobel-Filter definieren	120
5.5.3	Manuelle Gradientenberechnung	121
5.5.4	Gradientenberechnung mit conv2	122
5.5.5	Gradientenmagnitude und Kanten	123
5.5.6	Visueller Vergleich	123
5.5.7	Automatische Durchmesserbestimmung	124
5.5.8	Vollständiger MATLAB-Code	125
6	Fazit	127
6.1	Schlussfolgerungen	127

Abbildungsverzeichnis

2.1	Allgemeine Messkettenelemente	3
2.2	Allgemeine Messkettenelemente (detailliert)	6
2.3	Erfassen von nichtelektrischen physikalischen Größen mit einer Messkette	8
2.4	Messverstärker bestehend aus Spannungsfolger und invertierendem Verstärker	8
2.5	Grundprinzip der Abtast- und Halteschaltung (Sample-Hold)	9
2.6	Maschinelles Lernen im Überblick: Anwendungsbeispiele nach Arten	12
2.7	Maschinelles Lernen und seine Untergruppen innerhalb der Künstlichen Intelligenz	14
2.8	Rekurrente Verbindung von zwei der Neuronen	15
2.9	Signalflussdiagramm des Perzeptrons	17
2.10	Architektonischer Graph eines MLP mit zwei verborgenen Schichten	19
2.11	Ein CNN mit zwei Faltungslagen	19
3.1	Kernreaktionen auf Ziffern angewendet	30
3.2	Diese Abbildung veranschaulicht den DEKR-Prozess	32
3.3	YOLO-Architektur	35
3.4	Netzwerkarchitektur von YOLOv4.	35
3.5	Kanten Typen	38
4.1	Illustration von linearen, logistischen Sigmoid- und Tanh-Aktivierungsfunktionen, angepasst aus <i>Activation Functions in Deep Learning</i> " Typen	62
4.2	Beispielbilder aus den Datensätzen MNIST, CIFAR-10 und CIFAR-100. Jede Zeile zeigt Bilder einer bestimmten Kategorie. (Adaptiert aus „Towards Dropout Training for Convolutional Neural Networks“(2015)	65
5.1	Relu Aktivierung Funktion	81
5.2	Grafischer Vergleich der vorhergesagten und tatsächlichen Werte	81
5.3	Visuelle Klassifikation der Datenpunkte basierend auf den Netzwerkausgaben	85
5.4	Softmax-Wahrscheinlichkeitsverteilung für die Bildvorhersage	96
5.5	Vorhersageergebnisse für 12 Bilder	97
5.6	Neun synthetische Bilder in drei Spalten	106
5.7	Kantendetektion an Zahnrädern	118
5.8	Geschlossene Konturmessung mit Kantenerkennungsalgorithmus	119

Tabellenverzeichnis

2.1	Basiseinheiten des internationalen SI-Systems	2
2.2	Grundbegriffe der Messtechnik	3
2.3	Klassifizierung von Messsignalen bezüglich der Signalformen	5
2.4	Zusammenfassung der Lernarten im maschinellen Lernen	11
2.5	Vergleich der Architekturtypen	20
3.1	Vergleich verschiedener Ansätze zur Anomalieerkennung	57
4.1	Model parameters and configuration	59
5.1	Neural network prediction results	70
5.2	Genauigkeit des Netzes auf zuvor ungesesehenen Testbeispielen	84

Kapitel 1

Einleitung

Ziele der Arbeit

Die vorliegende Arbeit verfolgt mehrere zentrale Ziele, die sich aus der Untersuchung der Anwendung von Künstlicher Intelligenz (KI) in der Messtechnik ergeben:

1. **Theoretische Grundlagen der Messtechnik und KI erarbeiten:** Eine umfassende Darstellung der Grundlagen der Messtechnik, einschließlich Messgrößen, Messsysteme und Messunsicherheiten, sowie der relevanten KI-Konzepte wie Maschinelles Lernen (ML) und Neuronale Netze (NN).
2. **Anwendung von KI in der Messtechnik analysieren:** Untersuchung der Rolle von KI in modernen Messsystemen, insbesondere in den Bereichen Objektlokalisierung, Kantenerkennung und Signalverarbeitung.
3. **Praktische Implementierung von KI-Modellen in MATLAB:** Entwicklung und Evaluierung von neuronalen Netzen für verschiedene messtechnische Aufgaben, darunter Regression, Klassifikation und Bildverarbeitung.
4. **Verbesserung der Kantenerkennung durch KI:** Analyse klassischer Kantenerkennungsalgorithmen und deren Erweiterung durch KI-Methoden, um höhere Präzision und Robustheit zu erreichen.
5. **Leistungsbewertung von KI-Modellen:** Evaluierung der Leistungsfähigkeit verschiedener Netzwerkstrategien anhand von Metriken wie Genauigkeit, Robustheit und Echtzeitfähigkeit.

Überblick zum GitHub-Repository Das begleitende GitHub-Repository enthält MATLAB-Skripte für neuronale Netze und eine Dokumentation mit Ausführungsanleitung.



Kapitel 2

Theoretische Grundlagen

2.1 Grundlagen der Messtechnik

2.1.1 Definition

In der Fachliteratur existieren zahlreiche Definitionen des Begriffs „Messen“. Die umfassendste Beschreibung liefert dabei vermutlich die DIN 1319-1, in der es heißt:

„Messen ist die Ausführung von geplanten Tätigkeiten zum quantitativen Vergleich der Messgröße mit einer Maßeinheit.“

Die aktuelle Definition berücksichtigt, dass auch theoretische Überlegungen und Berechnungen für die Ausführung einer Messung erforderlich sein können. Aus dieser Definition folgt die Notwendigkeit, sich mit physikalischen Größen, für unsere Betrachtungen auch als Messgrößen zu interpretieren, und Maßeinheiten auseinander zu setzen. [1]

$$\text{Messwert} = \text{Maßzahl} \times \text{Einheit} \quad (2.1)$$

Dabei wird die Zahl der Einheiten entweder vom absoluten Nullpunkt oder von einem vereinbarten Bezugspunkt der Messgröße gerechnet. Das bekannteste Beispiel für eine unterschiedliche Zählung und Benennung ist die Temperatur mit den Einheiten °C (Grad Celsius) y K (Kelvin). Für eine Messung benötigt man nicht nur ein Messgerät, sondern auch die entsprechenden Einheiten. Heute setzt man ausschließlich das 1960 international vereinbarte „Système International d’Unités“, abgekürzt „SI“. Die SI-Einheiten beruhen auf sieben Basiseinheiten. [2]

Basisgröße	Name der Basiseinheit	Zeichen
Länge	Meter	m
Zeit	Sekunde	s
Masse	Kilogramm	kg
Elektrische Stromstärke	Ampere	A
Temperatur	Kelvin	K
Stoffmenge	Mol	mol
Lichtstärke	Candela	cd

Tabelle 2.1: Basiseinheiten des internationalen SI-Systems

Die Messtechnik ist ein Teilbereich der Ingenieurwissenschaften, der sich mit der Erfassung, Verarbeitung und Darstellung physikalischer Größen beschäftigt. Ziel ist es, physikalische, chemische oder biologische Größen quantitativ zu erfassen und reproduzierbar darzustellen. [3]

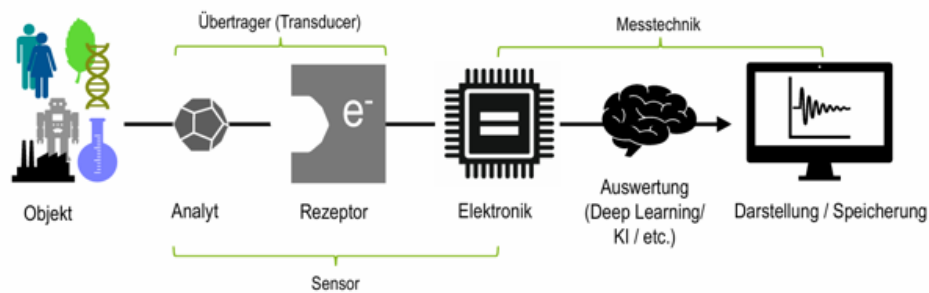


Abbildung 2.1: Allgemeine Messkettenelemente

Grundbegriffe der Messtechnik

- Wichtige Begriffe, die im Kontext von Messsystemen relevant sind[2]:

Begriff	Beschreibung
Messgröße	Die zu erfassende physikalische Größe
Messsignal	Elektrisches Signal, das die Messgröße repräsentiert
Messgerät	Technische Einrichtung zur Erfassung und Anzeige der Messgröße
Sensor	Bauteil, das die Messgröße in ein messbares Signal umwandelt
Messkette	Gesamtheit aller Komponenten vom Sensor bis zur Anzeige
Kalibrierung	Abgleich eines Messsystems mit einer bekannten Referenz

Tabelle 2.2: Grundbegriffe der Messtechnik

2.1.2 Aufgaben der Messtechnik:

- Erfassen physikalischer Größen (z. B. Temperatur, Druck, Spannung)
- Überwachen und Steuern von Prozessen
- Prüfen von Bauteilen und Systemen
- Qualitätssicherung und -kontrolle [3]

2.1.3 Messgrößen und deren Klassifizierung

Messgrößen lassen sich nach verschiedenen Kriterien klassifizieren:

- **Nach ihrer physikalischen Natur:**
 - Mechanische Größen (Kraft, Druck, Weg)
 - Thermische Größen (Temperatur, Wärmefluss)
 - Elektrische Größen (Spannung, Strom, Widerstand)
 - Optische Größen (Lichtstärke, Wellenlänge) [3]
- **Nach ihrem zeitlichen Verhalten:**
 - Statische Messgrößen (konstante Werte)
 - Dynamische Messgrößen (zeitabhängige Werte) [3]

2.1.4 Messsignal

Messwerte beinhalten die im Messprozess gesuchten Informationen über physikalische Größen. Die Übertragung dieser Informationen erfolgt in Form eines Signals. Im technischen Gebrauch, und hierbei speziell im Bereich der Messtechnik, wird ein Zeitverlauf einer physikalischen Größe als Signal bezeichnet. Bei Verwendung eines Signals in der Messtechnik sprechen wir auch konkret vom *Messsignal* [1].

Im Sinne dieser Definition ist ein Signal nicht an eine bestimmte physikalische Größe gebunden. Ein oder mehrere Parameter des Signals (die Informationsparameter) sind Träger des interessierenden Informationsgehalts, so dass meist nicht alle Kennzeichen der physikalischen Größe, die als Signalträger fungiert, ausgewertet werden müssen. Liegt beispielsweise ein Messsignal in Form einer sinusförmigen Spannung entsprechend der Gleichung

$$u(t) = \hat{u} \cdot \sin(\omega t + \varphi) \quad (2.2)$$

vor, ist deren zeitlicher Signalverlauf durch die Amplitude \hat{u} , die Kreisfrequenz ω und den Phasenwinkel φ gekennzeichnet. Die Angabe „*sin*“, Symbol für die Sinusfunktion, definiert eindeutig den Verlauf des Funktionswertes über die Zeit. Je nach messtechnischer Aufgabe kann sich die Auswertung auf die Amplitude, die Frequenz oder den Phasenwinkel beschränken [1].

Zur eindeutigen Beschreibung von Signalen ist eine Reihe von Merkmalen heranzuziehen. Man unterscheidet nach dem Wertevorrat des Informationsparameters (analog oder diskret), nach der zeitlichen Verfügbarkeit (kontinuierlich oder diskontinuierlich) und weiteren Merkmalsklassen, so nach der Art des Informationsparameters, der Modulationsart, der das Signal tragenden Größenart und der Erscheinungsform [4].

lfd. Nr.	Signalcharakteristik	Erläuterung	Vorteil	Nachteil
1.1	Analog	Informationsparameter kann theoretisch beliebig viele Werte innerhalb seines Wertebereichs annehmen	Proportionale Abbildung zwischen Messsignal und Informationsparameter	Einfach zu stören, z. B. durch externe Störungen, Rauschen, Temperaturdrift usw.
1.2	Diskret	Informationsparameter kann nur endlich viele Werte innerhalb seines Wertebereichs annehmen	Störeinflüsse machen sich erst nach Überschreiten von Grenzwerten bemerkbar	Bei der Abbildung analoger Messwerte auf einen diskreten Informationsparameter entsteht ein Informationsverlust
2.1	Kontinuierlich	Informationsparameter kann zu jedem beliebigen Zeitpunkt seinen Wert ändern	Jederzeit ist der zeitliche Verlauf von Messwerten verfolgbar	Störungen können jederzeit wirken, Informationsmenge ist oft unnötig groß
2.2	Diskontinuierlich	Informationsparameter kann nur zu diskreten Zeitpunkten seinen Wert ändern	Störungen zwischen den Zeitpunkten der Parameteränderungen können sich nicht auswirken	Informationen stehen nur zu diskreten Zeitpunkten zur Verfügung
3.1	Determiniert	Determiniertheit des Informationsparameters	Information mit einmaliger Messung gewinnbar	Information kann durch Störung unbrauchbar werden
3.2	Stochastisch	Informationsparameter repräsentiert eine stochastische Größe	Störungen machen sich entsprechend ihrer Verteilung nur stark reduziert bemerkbar, sie werden über die Messzeit integriert	Information ist erst mit mehrmaligen Messungen zu gewinnen, das erfordert einen größeren Zeitbedarf

Tabelle 2.3: Klassifizierung von Messsignalen bezüglich der Signalformen

Analoge und digitale Messgeräte In der praktischen Messtechnik unterscheidet man zwischen:

- **Analogen Messgeräten:** Sie sind Zeigerinstrumente, bei denen die Anzeige auf einer Skala durch einen Zeiger erfolgt.
- **Digitalen Messgeräten:** Sie geben das Messergebnis über eine mehrstellige 7-Segment-Anzeige aus.[2]

2.1.5 Messsysteme und Messketten

Ein Messsystem besteht aus einer Messkette, die folgende Elemente umfasst:

1. **Sensor / Messfühler:** Erfasst die physikalische Größe und wandelt sie in ein Primärsignal um.
2. **Signalaufbereitung:** Verstärkung, Filterung, Umwandlung des Signals (z. B. von analog nach digital).
3. **Signalverarbeitung:** Digitalisierung, Speicherung und mathematische Auswertung des Signals.
4. **Ausgabe / Anzeige:** Darstellung der Messwerte für den Anwender.[3]

Messkette (vereinfacht)

Messgröße → Sensor → Signalaufbereitung → Signalverarbeitung → Anzeige [5]

Am Anfang jeder elektrischen Messung von nicht elektrischen Größen stehen die Sensoren. Sensorelemente formen physikalische, chemische oder biologische Messgrößen in elektrische Messsignale um, die mit den Messwerten in eindeutigen, oft linearen Zusammenhängen stehen. [5]

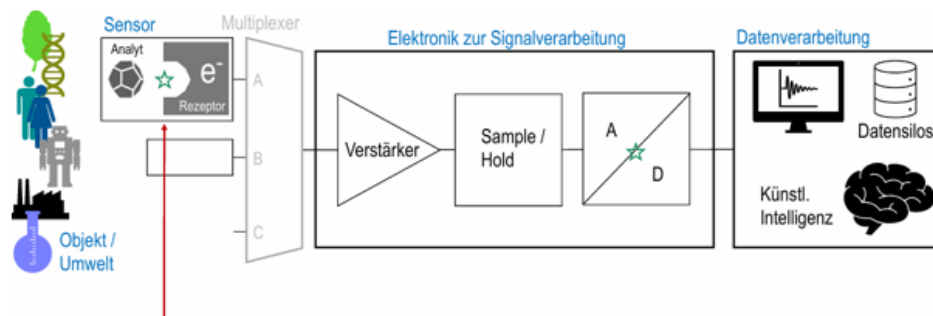


Abbildung 2.2: Allgemeine Messkettenelemente (detailliert)

2.1.6 Messabweichungen und Messunsicherheiten

Fehler- bzw. Abweichungsbegriff In der Literatur DIN 1319 findet man die Definition, dass ein „Fehler“ eine Grenzüberschreitung von „akzeptierten Abweichungen“ ist. Im Folgenden soll der Begriff „Fehler“ und „Abweichung“ (A) bzw. im engl. „Error“ (e) aber gleichermaßen verwendet werden.[5]

Anhand der gegebenen Nennkennlinie eines Messglieds oder seiner Nennfunktion lässt sich für jeden Betriebszustand und Zeitpunkt der theoretisch richtige (wahre) Wert x_w (Sollwert) eines Messsignals ermitteln. Messtechnisch lässt sich unter den gleichen Verhältnissen der tatsächlich ausgegebene Wert x_a (Istwert) desselben Signals ermitteln. [2]

Stimmen beide Werte überein, ist das Messglied im betrachteten Zustand fehlerfrei. In der Praxis hat man keine fehlerfreien Messungen. Selbst mit hohem Aufwand verbleibt bei jeder Messung eine Unsicherheit, da z. B. ein Messwert immer nur mit endlich vielen Dezimalstellen angegeben werden kann. Ein vollständiges Messergebnis besteht aus zwei Informationen, d. h. die eine betrifft den Wert des Ergebnisses, die andere dessen Fehler oder Unsicherheit. [2]

Als Messfehler (Abweichung) A wird die Differenz zwischen dem ausgegebenen (oder angezeigten) Wert x_a eines Messsignals und dem wahren Wert x_w desselben Signals bezeichnet:

$$A = x_a - x_w \quad (2.3)$$

Der relative Fehler A_r ist das Verhältnis des Messfehlers zu einem Bezugswert X [2]:

$$A_r = \frac{A}{X} \quad (2.4)$$

Kein Messsystem liefert absolut exakte Werte. Mögliche Ursachen:

- Systematische Abweichungen (konstante Fehlerquelle)
- Zufällige Abweichungen (Störgrößen, Rauschen)
- Große Abweichungen (Messfehler durch Defekte oder Bedienfehler).[5]

Systematische Fehler: Systematische Fehler sind Messfehler, die sich unter wiederholbaren Betriebsbedingungen der Messeinrichtung wiederholen lassen. Sie haben im jeweiligen Betriebszustand einen bestimmten Wert und ein bestimmtes Vorzeichen und diese entstehen durch Unvollkommenheiten der Messeinrichtung. Je größer die systematischen Fehler einer Messeinrichtung sind, desto geringer ist die Genauigkeit. Der Begriff „Genauigkeit“ ist nicht quantifizierbar und sollte nur für qualitative Aussagen verwendet werden [1]

Zufällige Fehler: Zufällige Fehler sind Messfehler, deren Ursachen nicht im Einzelnen erfassbar sind, und die auch bei gleichbleibenden Betriebsbedingungen nicht wiederholbar oder vorhersagbar sind. Zufällige Fehler führen innerhalb einer Messreihe zu Streuungen, und sie bewirken, dass vergleichbare Signalwerte unregelmäßig (stochastisch) voneinander abweichen. Bei der Messreihe kann es sich um eine endliche Zahl gleichartiger Messwerte verschiedener Messobjekte handeln, sie kann auch aus zeitlich aufeinanderfolgenden Augenblickswerten desselben Messsignals bestehen. Je stärker zufällige Fehler bei einer Messung in Erscheinung treten, desto geringer ist die Präzision der Messung. Zufällige Fehler lassen sich mit den Methoden der Statistik umso zuverlässiger erfassen, je mehr Einzelwerte zur Verfügung stehen. [1]

Kombinierte Standardunsicherheit: Behandlung von unbekannten/systematischen Abweichungen: Diese Abweichungen können in der Praxis natürlich auftreten und viele Messtechniker verzweifeln oft daran. Sichtbar werden diese, wenn unter Wiederholbedingungen der erkennbare Messfehler gleichermaßen in Erscheinung tritt, die Ursache dafür aber unbekannt bleibt. Eine mathematische Behandlung und Korrektur im Messergebnis sind nicht möglich. Daher können diese nur wie folgt behandelt werden:

- Weitere Optimierung des Messaufbaus und intensive Ursachenforschung
- Überprüfung, ob der Fehler toleriert werden kann und im Ergebnis mit ausgewiesen werden kann.[2]

Klassische Messverfahren Beispiele klassischer Methoden in der Messtechnik:

- Brückenmessschaltungen (z. B. Wheatstone-Brücke für Widerstandsmessung)
- Thermoelemente zur Temperaturmessung
- DMS (Dehnungsmessstreifen) zur Kraft- und Wegmessung
- Piezosensoren zur Druck- und Beschleunigungsmessung

Jede dieser Methoden hat eigene Einsatzbereiche, Vorteile und Einschränkungen. [2]

Signalverarbeitung in der Messtechnik Zur Verarbeitung der Messsignale werden verschiedene Verfahren eingesetzt.[1]

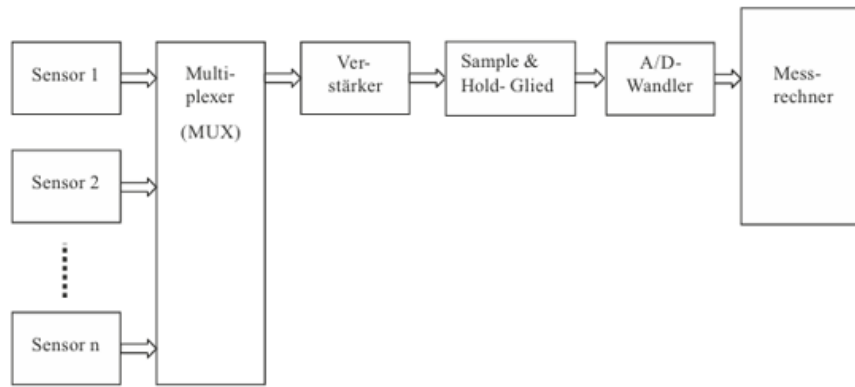


Abbildung 2.3: Erfassen von nichtelektrischen physikalischen Größen mit einer Messkette

- Multiplexen:** Bei einfachen Messaufgaben, wo nur eine Messstelle zu bewerten ist, ist das Multiplexen nicht erforderlich, i. Allg. sind aber mehrere Messstellen zu überwachen. In solchen Fällen bietet sich eine gemeinsame Nutzung der Messeinrichtung für das Aufnehmen der Messinformationen von mehreren Messstellen an. Hierzu dienen Multiplexer, die vorhandene Messstellen zeit-seriell mit der Messeinrichtung verbinden. [1]
- Verstärkung:** Dieser Teil der Messkette hat die Aufgabe, das Messsignal zu verstärken, d. h. den Pegel des Signals an den Eingangsbereich der nachfolgenden Baugruppe anzupassen. In der Messtechnik wird i. Allg. eine lineare Verstärkung gefordert. Hierfür stehen rauscharme und leistungsfähige Operationsverstärker (OPV) zur Verfügung, bei denen bei einfachster Außenbeschaltung mit zwei Widerständen die geforderte Verstärkung eingestellt werden kann. [5]

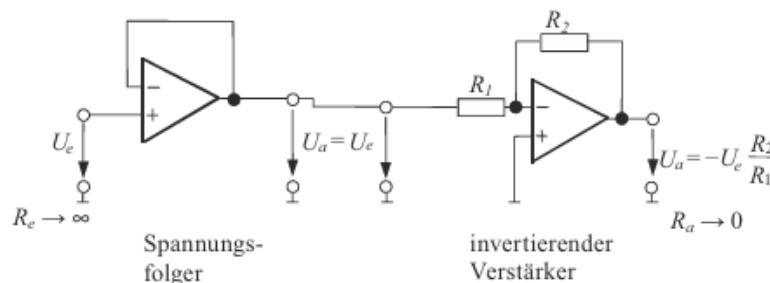


Abbildung 2.4: Messverstärker bestehend aus Spannungsfolger und invertierendem Verstärker

- Filterung (z. B. Tiefpass, Hochpass, Bandpass):** Die Filterung beinhaltet Maßnahmen zur Unterdrückung von Störsignalen mit Frequenzen außerhalb des Nutzsignalbereiches. Zur Filterung werden elektrische Filter, vorrangig 1. und 2. Ordnung, verwendet. Für höchste Ansprüche kommen auch Filter höherer Ordnung zur Anwendung, für deren Applikation aber große Erfahrungen erforderlich sind. Es müssen dann Schwankungen, d. h. Welligkeit der Ausgangsspannung des Filters im genutzten Frequenzbereich und eventuelle Schwingneigung der gesamten Baugruppe zur Messsignalverarbeitung beherrscht werden. [1]
- Abtast- und Halteschaltung:** Eine Abtast- und Halteschaltung, oft auch als Sample - Hold-Glied bezeichnet, hat die Aufgabe, zeitveränderliche Signale an ihrem Ausgang für eine kurze Zeit konstant zu halten. Man kann die Schaltung auch als Analogwertspeicher auffassen. Die Schaltung basiert auf dem Spannungsfolger, der mit einem Schalter S und dem Speicherkondensator C ergänzt wird. [1]

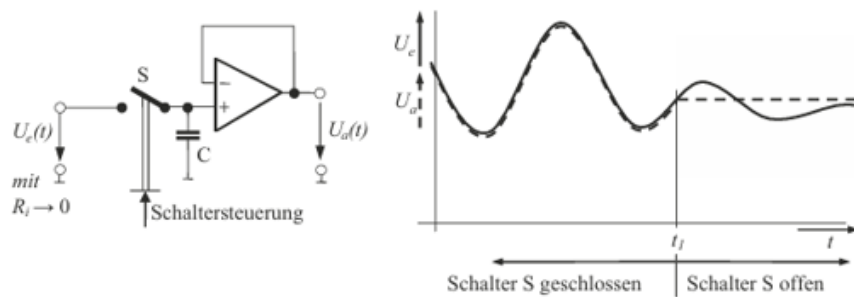


Abbildung 2.5: Grundprinzip der Abtast- und Halteschaltung (Sample-and-Hold)

- **Analog-Digital-Umwandlung (ADC):** Die Analog-Digital-Wandlung (AD-Wandlung) stellt ein zeit- und wertediskretes Signal zur Verfügung, das zur Ansteuerung einer Digitalanzeige oder zur Weiterverarbeitung in einem Rechner geeignet ist. Für den praktisch tätigen Messtechniker sind konkrete technische Prinzipien zur AD-Wandlung von Interesse, weil diese mit resultierenden Kenngrößen direkt korrespondieren und somit die Merkmale einer aufgebauten Messkette durch den verwendeten ADW bestimmend geprägt werden. [4]

Signalverarbeitung dient dazu, Messdaten für die Weiterverarbeitung zu optimieren, Störungen zu reduzieren und das Signal-Rausch-Verhältnis zu verbessern. [2]

2.1.7 Herausforderungen moderner Messtechnik

Moderne Messtechnik sieht sich mit folgenden zentralen Herausforderungen konfrontiert:

- **Hohe Datenraten und Datenmengen:** Die steigende Anzahl an Sensoren und die höhere Abtastfrequenz führen zu großen Datenvolumina, die verarbeitet und analysiert werden müssen.
- **Anforderungen an Echtzeitfähigkeit:** Viele industrielle Anwendungen erfordern die unmittelbare Verarbeitung und Auswertung von Messdaten ohne nennenswerte Verzögerungen.
- **Integration in vernetzte Systeme (IoT, Industrie 4.0):** Moderne Messsysteme müssen nahtlos mit vernetzten Umgebungen kommunizieren und Daten in übergreifenden Systemarchitekturen bereitstellen können.
- **Zunehmende Komplexität der Messaufgaben:** Die zu messenden Größen und Systeme werden immer komplexer, was anspruchsvolle Messverfahren und Auswertalgorithmen erfordert.
- **Bedarf an intelligenten, lernfähigen Messsystemen:** Traditionelle Messmethoden stoßen an Grenzen, wodurch adaptive und selbstlernende Systeme notwendig werden.

Diese Punkte begründen den zunehmenden Einsatz von Methoden der Künstlichen Intelligenz, insbesondere neuronalen Netzen, um komplexe, nichtlineare und dynamische Messaufgaben zu bewältigen [2].

2.2 Grundlagen der Künstliche Intelligenz

2.2.1 Künstliche Intelligenz (KI)

Die Künstliche Intelligenz (KI) bezieht sich auf die Simulation menschlicher Intelligenzprozesse durch Maschinen, insbesondere Computersysteme. Ziel ist es, Maschinen zu entwickeln, die Aufgaben ausführen können, die typischerweise menschliche Intelligenz erfordern, wie zum Beispiel logisches Denken, Lernen, Problemlösen, Wahrnehmung und Sprachverstehen [6].

2.2.2 Maschinelles Lernen (ML)

Maschinelles Lernen ist ein Teilbereich der KI und konzentriert sich auf Algorithmen und Modelle, die es Computern ermöglichen, auf Grundlage von Daten zu lernen und Entscheidungen zu treffen, ohne explizit für jede Aufgabe programmiert worden zu sein [7].

- KI ist das umfassendere Konzept der „intelligenten Maschinen“
- ML ist eine spezifische Methode innerhalb der KI, die Maschinen befähigt, selbstständig aus Daten zu lernen

2.2.3 Definitionen und Konzepte

Die Künstliche Intelligenz (KI, englisch: Artificial Intelligence, AI) ist ein Teilgebiet der Informatik, das sich hauptsächlich mit der Automatisierung intelligenten Verhaltens befasst. Eine kompakte Definition von Intelligenz ist:

$$\text{Intelligenz} = \text{Wahrnehmen} + \text{Analysieren} + \text{Reagieren} \quad (2.5)$$

In [6] definieren KI als „die Konstruktion von Agenten, die rational handeln“. In der heutigen Forschung umfasst KI viele Bereiche, darunter:

- Maschinelles Lernen (ML)
- Deep Learning
- Sprachverarbeitung
- Robotik
- Entscheidungsfindungssysteme[7]

2.2.4 Anwendungsfelder der KI

- Medizinische Diagnostik
- Autonome Transportsysteme
- Robotik
- Virtuelle Assistenten (z. B. Siri, Alexa)
- Finanzwesen (z. B. Betrugserkennung)[7]

2.2.5 Klassifikation von ML-Modellen

Das maschinelle Lernen lässt sich in drei Hauptkategorien einteilen:

Überwachtes Lernen (Supervised Learning)

- Trainiert mit Eingabe-Ausgabe-Paaren (gelabelte Daten)
- Ziel: Abbildungsfunktion für neue Eingaben lernen
- Anwendungen:
 - Klassifikation (Spam-Erkennung, Diagnosen)
 - Regression (Preisvorhersagen, Prognosen)

Mathematisch wird ein Verlustmaß (z. B. MSE) minimiert[7, 8]:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i; \theta))^2 \quad (2.6)$$

Unüberwachtes Lernen (Unsupervised Learning)

- Keine gelabelten Daten erforderlich
- Hauptaufgaben:
 - Clustering (Kundensegmentierung)
 - Dimensionsreduktion (PCA)
 - Anomalieerkennung[9]

Bestärkendes Lernen (Reinforcement Learning)

- Lernen durch Interaktion mit Umgebung
- Belohnungsmechanismen steuern den Lernprozess
- Anwendungen:
 - Spielintelligenz (AlphaGo)
 - Robotik (autonome Steuerung)
 - Dynamische Systeme[8]

Wie bereits erläutert [10]:

Typ	Beschreibung	Beispiele
Überwachtes Lernen	Lernen mit beschrifteten Daten; Vorhersage eines Outputs basierend auf Eingabe-Ausgabe-Paaren.	Spam-Erkennung, Kreditscoring
Unüberwachtes Lernen	Lernen ohne beschriftete Daten; Entdeckung versteckter Muster oder Strukturen.	Kundensegmentierung, Anomalieerkennung
Bestärkendes Lernen	Lernen durch Interaktion mit der Umgebung und Belohnungsmechanismen.	Spielstrategien (z. B. AlphaGo), Robotik

Tabelle 2.4: Zusammenfassung der Lernarten im maschinellen Lernen

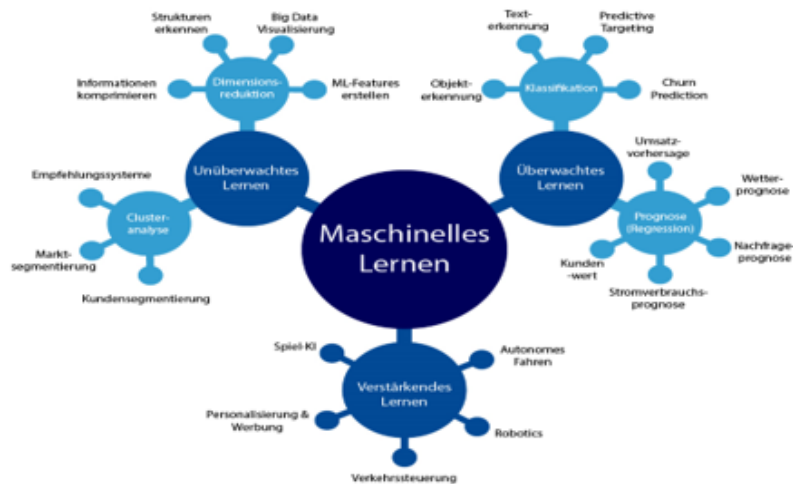


Abbildung 2.6: Maschinelles Lernen im Überblick: Anwendungsbeispiele nach Arten

2.2.6 Methoden des Maschinellen Lernens

In der Praxis des Maschinellen Lernens kommen verschiedene Modelle zum Einsatz, die auf mathematischen Grundlagen beruhen. Diese Modelle helfen dabei, Daten zu analysieren, Muster zu erkennen und Vorhersagen zu treffen. Zu den wichtigsten gehören [8]:

- Regression
- Klassifikation
- Hauptkomponentenanalyse (PCA)
- Support Vector Machines (SVM)
- Clustering (z. B. K-Means)

Regression

Die Regression ist ein überwachtes Lernverfahren, das verwendet wird, um den Zusammenhang zwischen einer abhängigen Zielvariable und einer oder mehreren unabhängigen Variablen (Prädiktoren) zu modellieren. Sie dient zur Vorhersage eines kontinuierlichen Wertes auf Basis von Eingabedaten, und zur Quantifizierung des Einflusses einzelner Merkmale auf die Zielvariable [11].

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n + \epsilon \quad (2.7)$$

Typen der Regression:

- **Lineare Regression:** Modelliert eine lineare Beziehung zwischen Eingangs- und Zielvariablen:

$$y = \mathbf{w}^T \mathbf{x} + b \quad (2.8)$$

- **Multiple Regression:** Berücksichtigt mehrere unabhängige Variablen gleichzeitig
- **Logistische Regression:** Modelliert Wahrscheinlichkeiten für Klassifikationsprobleme (z. B. Ja/Nein-Entscheidungen)[11]:

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}} \quad (2.9)$$

Klassifikation

Die Klassifikation ist ein weiteres überwachtes Lernverfahren, bei dem ein Modell aus Trainingsdaten lernt, neue Datenpunkte einer endlichen Menge diskreter Klassen zuzuordnen. Klassifikationsalgorithmen kommen typischerweise zum Einsatz, wenn die Zielvariable kategorial ist [12].

Das Ziel der Klassifikation ist es, eine Abbildung von Eingaben \mathbf{x} auf Ausgaben y zu lernen, wobei $y \in \{1, \dots, C\}$ ist, wobei C die Anzahl der Klassen ist.

- **Binäre Klassifikation:** $C = 2$ (in diesem Fall wird oft $y \in \{0, 1\}$ angenommen)
- **Multiklassen-Klassifikation:** $C > 2$
- **Multilabel-Klassifikation:** Nicht-exklusive Klassen (Vorhersage mehrerer zusammenhängender binärer Klassenlabels)

Ziffernerkennung Eine spezialisierte Form der Klassifikation ist die Ziffernerkennung (engl. digit recognition), bei der handgeschriebene oder gedruckte Ziffern in Bildform automatisch erkannt und zugeordnet werden. Ein bekanntes Standarddatenset ist das MNIST-Dataset (Modified National Institute of Standards), das über 70.000 handgeschriebene Ziffernbilder umfasst (60.000 Trainingsbilder und 10.000 Testbilder) [8].

Hauptkomponentenanalyse (PCA)

Eine Methode zur automatischen Dimensionsreduktion ist die Hauptkomponentenanalyse (engl. principal component analysis, PCA), welche im Vektorraum der Trainingsdaten die Richtungen höchster Varianz bestimmt und dann mittels einer linearen Abbildung die Daten auf den Unterraum der Hauptkomponenten projiziert [11]:

$$\mathbf{Z} = \mathbf{X}\mathbf{W} \quad (2.10)$$

wobei \mathbf{W} die Eigenvektoren der Kovarianzmatrix $\mathbf{X}^T\mathbf{X}$ enthält. Diese Methode transformiert eine große Menge korrelierter Variablen in eine kleinere Anzahl unkorrelierter Variablen, die als Hauptkomponenten bezeichnet werden.

Support Vector Machines (SVM)

Mit den Support Vector Machines (SVMs) wird ein neuer Ansatz für das maschinelle Lernen eingeführt; die Methode basiert mehr auf statistischen Grundlagen und weniger auf biologischer Plausibilität [9].

Die SVMs verwenden eine binäre Klassifizierung, d.h. die Eingabe wird immer in zwei Klassen aufgeteilt. Dieser Ansatz hat sich in einer Vielzahl von Anwendungen bewährt, die von Textklassifizierungen (z. B. „Ist dieser Artikel mit meiner Suchanfrage verwandt?“) bis zu komplexen Mustererkennungsaufgaben reichen.

Mathematisch wird eine maximale Margin-Hyperebene bestimmt:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad (2.11)$$

Die Support-Vektor-Maschinen gehören zu den überwachten Lernmodellen, die auf assoziativenernalgorithmen beruhen, die Daten analysieren und Muster erkennen; sie finden daher Anwendung in der Klassifizierungs- und Regressionsanalyse [9].

Clustering (K-Means)

Clustering ist eine Technik des unüberwachten Lernens. Wenn man in einer Suchmaschine nach dem Begriff „Decke“ sucht, so werden Treffer wie etwa „Microfaser-Decke“, „unter der Decke“, etc. angezeigt, genauso wie „Stück an der Decke“, „Wie streiche ich eine Decke“, etc. Bei der Menge der gefundenen Textdokumente handelt es sich um zwei deutlich voneinander unterscheidbare Cluster [11].

Die Besonderheit beim Clustering im Unterschied zum Lernen mit Lehrer ist die, dass die Trainingsdaten nicht klassifiziert sind. Immer dann, wenn die Zahl der Cluster schon im Voraus bekannt ist, kann das K-Means-Verfahren zum Einsatz kommen:

1. Initialisiere die Clusterzentren μ_1, \dots, μ_k zufällig.
2. **Wiederhole:**
 - (a) Weise jeden Datenpunkt dem nächstgelegenen Zentrum μ_j zu.
 - (b) Aktualisiere jedes μ_j als Mittelwert der zugewiesenen Punkte.
3. **Bis:** die Clusterzentren konvergieren oder die maximale Anzahl an Iterationen erreicht ist.

Beziehung zwischen KI, ML und Deep Learning (DL)

Die Beziehung zwischen den Begriffen kann als konzentrische Struktur dargestellt werden:

- **KI:** Umfasst alle Systeme, die intelligentes Verhalten zeigen
- **ML:** Spezielle Methode innerhalb der KI, die auf Lernen aus Daten basiert
- **DL:** Spezialisierte Technik innerhalb des ML, die tiefe neuronale Netze verwendet.[13]



Abbildung 2.7: Maschinelles Lernen und seine Untergruppen innerhalb der Künstlichen Intelligenz

2.2.7 Übergang zu Neuronalen Netzwerken

Motivation und historische Entwicklung

Die Grundlagen neuronaler Netzwerke wurden bereits 1943 von Warren McCulloch und Walter Pitts gelegt, die ein einfaches mathematisches Modell eines Neurons vorstellten. Donald Hebb erweiterte

dieses Modell 1949 mit seiner Theorie der synaptischen Plastizität – ein frühes Lernprinzip für neuronale Systeme [13].

Ein bedeutender Fortschritt war das Perzeptron, entwickelt von Frank Rosenblatt (1958), das einfache Klassifikationsaufgaben lösen konnte. Doch nach der Kritik von Marvin Minsky und Seymour Papert (1969), die zeigten, dass das Perzeptron nur linear separierbare Probleme lösen kann, stagnierte die Forschung – es folgte der sogenannte „AI Winter“ [6].

Erst in den 1980er Jahren kam es zu einem Wiederaufschwung, als David Rumelhart, Geoffrey Hinton und Ronald Williams (1986) den Backpropagation-Algorithmus einführten, mit dem mehrschichtige Netzwerke erfolgreich trainiert werden konnten. Dadurch wurde es möglich, komplexe, nicht-lineare Zusammenhänge zu modellieren [11].

In den 2000er- und 2010er-Jahren erlebten neuronale Netzwerke eine regelrechte Revolution. Dank großer Datenmengen (z.B. ImageNet), leistungsfähiger Grafikprozessoren (GPUs) und verbesserter Architekturen (z.B. Convolutional Neural Networks) gelang der praktische Durchbruch. Ein Meilenstein war AlexNet (2012), das bei der Bildklassifikation neue Maßstäbe setzte. Für sequenzielle Daten wurden rekurrente Netzwerke wie LSTMs (Long Short-Term Memory) entwickelt [11].

Die Arbeit an künstlichen neuronalen Netzen wurde seit ihren Anfängen durch die Erkenntnis motiviert, dass das menschliche Gehirn auf völlig andere Weise rechnet als ein herkömmlicher digitaler Computer. Das Gehirn ist ein hochkomplexes, nichtlineares und parallel arbeitendes System. Es besitzt die Fähigkeit, seine strukturellen Bestandteile – sogenannte Neuronen – so zu organisieren, dass es bestimmte Berechnungen (z.B. Mustererkennung) vielfach schneller durchführen kann als der schnellste digitale Rechner. Ein Beispiel dafür ist das menschliche Sehen: Das Gehirn bewältigt routinemäßig Wahrnehmungs- und Erkennungsaufgaben in etwa 100–200 Millisekunden [13].

Definition von neuronalen Netzen

Neuronale Netze sind Netzwerke aus Nervenzellen im Gehirn von Menschen. Nervenzellen besitzt das menschliche Gehirn. Der komplexen Verschaltung und der Adaptivität verdanken wir Menschen unsere Intelligenz und unsere Fähigkeit, verschiedenste motorische und intellektuelle Fähigkeiten zu lernen und uns an variable Umweltbedingungen anzupassen.

Um eine gute Leistung zu erzielen, setzen neuronale Netzwerke auf eine massive Vernetzung einfacher Recheneinheiten, die als „Neuronen“ oder „Verarbeitungseinheiten“ bezeichnet werden [11].

Erstaunlich ist dabei, dass nicht die Neuronen selbst, also die aktiven Einheiten, adaptiv sind, sondern vielmehr die Verbindungen zwischen ihnen – die Synapsen. Konkret bedeutet dies, dass diese Verbindungen ihre Leitfähigkeit verändern können. Es ist bekannt, dass eine Synapse dann verstärkt wird, wenn sie größere elektrische Ströme leiten muss. Eine „stärkere“ Synapse besitzt also eine höhere Leitfähigkeit. Synapsen, die häufig benutzt werden, erhalten ein zunehmend höheres Gewicht. Im Gegensatz dazu nimmt die Leitfähigkeit von selten genutzten oder inaktiven Synapsen kontinuierlich ab – bis hin zum vollständigen Abbau der Verbindung [13].

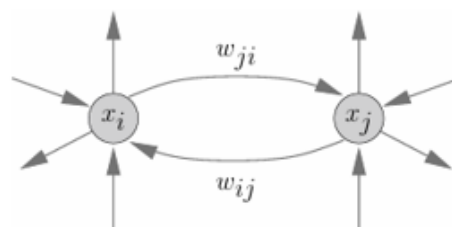


Abbildung 2.8: Rekurrente Verbindung von zwei der Neuronen

Wenn eine Verbindung zwischen Neuron j und Neuron i besteht und wiederholt Signale von j nach i gesendet werden, sodass beide Neuronen gleichzeitig aktiv sind, dann wird das Gewicht w_{ij} verstärkt. Eine mögliche Formel für die Gewichtsveränderung Δw_{ij} lautet:

$$\Delta w_{ij} = \eta x_i x_j \quad (2.12)$$

Dabei ist η die Lernrate, eine Konstante, die die Größe der einzelnen Lernschritte bestimmt [11].

Während klassische statistische Modelle (z.B. lineare Regression, SVM, PCA) in vielen Anwendungsbereichen sehr erfolgreich sind, stoßen sie bei komplexen Aufgaben wie etwa der Bilderkennung, Sprachverarbeitung oder automatischen Übersetzung an ihre Grenzen [13].

Typische Herausforderungen, die traditionelle Modelle schwer bewältigen können:

- Nicht-lineare Beziehungen: Viele reale Probleme sind hochgradig nicht-linear und vielschichtig.
- Automatische Merkmalsextraktion: Klassische Modelle erfordern oft manuelles Feature Engineering.
- Skalierung auf große Datenmengen: Mit zunehmender Datenverfügbarkeit steigt der Bedarf an leistungsfähigeren Lernverfahren.

Neuronale Netzwerke bieten eine leistungsstarke Lösung, da sie in der Lage sind:

- komplexe, nicht-lineare Zusammenhänge automatisch zu modellieren,
- aus Rohdaten (z.B. Pixeln, Audiodaten) direkt relevante Merkmale zu extrahieren,
- große Mengen an Daten effizient zu verarbeiten und zu generalisieren.[11]

2.2.8 Neuronale Netzwerkarchitekturen: Perzeptron, MLP, CNN, RNN

Das Perzeptron

Das Perzeptron ist die einfachste Form eines neuronalen Netzes, das für die Klassifizierung von Mustern verwendet wird, die als linear trennbar gelten (d. h. Muster, die auf gegenüberliegenden Seiten einer Hyperebene liegen). Es besteht im Wesentlichen aus einem einzigen Neuron mit einstellbaren synaptischen Gewichten und Vorspannungen. Der Algorithmus zur Anpassung der freien Parameter dieses neuronalen Netzes tauchte erstmals in einem Lernverfahren auf, das von Rosenblatt (1958, 1962) für sein Perzeptron-Gehirnmodell entwickelt wurde [13].

Rosenblatt bewies nämlich, dass der Perzeptron-Algorithmus konvergiert und die Entscheidungsfläche in Form einer Hyperebene zwischen den beiden Klassen positioniert, wenn die zum Training des Perzeptrons verwendeten Muster (Vektoren) aus zwei linear trennbaren Klassen stammen. Der Nachweis der Konvergenz des Algorithmus ist als *Perceptron-Konvergenztheorem* bekannt. Das Perzeptron, das aus einem einzigen Neuron besteht, ist auf die Klassifizierung von Mustern mit nur zwei Klassen (Hypothesen) beschränkt. Erweitert man die Ausgabeschicht (Berechnungsschicht) des Perzeptrons um mehr als ein Neuron, kann man entsprechend eine Klassifikation mit mehr als zwei Klassen bilden. Allerdings müssen die Klassen linear trennbar sein, damit das Perzeptron richtig funktioniert.

Wichtig ist, dass wir für die grundlegende Theorie des Perzeptrons als Musterklassifikator nur den Fall eines einzelnen Neurons betrachten müssen. Die Ausweitung der Theorie auf den Fall von mehr als einem Neuron ist trivial [13].

Ein Perzeptron besteht aus Eingaben, Gewichten, einer Aktivierungsfunktion sowie einem Ausgabe-neuron. Das Lernverhalten wird durch mehrere Schlüsselparameter beeinflusst, darunter Epochen, Lernrate und Gewichtsaktualisierung [14].

Eingaben und Gewichtung Das Rosenblatt-Perzeptron, im Folgenden einfach als Perzeptron bezeichnet, basiert auf einem nichtlinearen Neuron, nämlich dem McCulloch-Pitts-Modell eines Neurons. Ein solches neuronales Modell besteht aus einem linearen Kombinator, gefolgt von einem harten Begrenzer (der die Signum-Funktion ausführt), wie in Abbildung dargestellt. Der Summierknoten des neuronalen Modells berechnet eine lineare Kombination der an seinen Synapsen anliegenden Eingänge und berücksichtigt auch eine extern angelegte Vorspannung.

Die resultierende Summe, d. h. das induzierte lokale Feld, wird auf einen harten Begrenzer angewendet. Dementsprechend erzeugt das Neuron eine Ausgabe von $+1$, wenn die Eingabe des harten Begrenzers positiv ist, und -1 , wenn sie negativ ist [13].

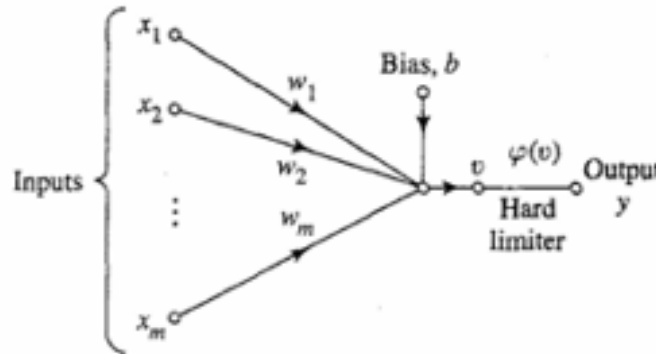


Abbildung 2.9: Signalflussdiagramm des Perzeptrons

In dem Signalflussgraphenmodell werden die synaptischen Gewichte des Perzeptrons mit w_1, w_2, \dots, w_m bezeichnet. Entsprechend werden die an das Perzeptron angelegten Eingänge mit x_1, x_2, \dots, x_m bezeichnet. Das extern angelegte Bias wird mit b bezeichnet. Aus dem Modell ergibt sich, dass der Begrenzereingang oder das induzierte lokale Feld des Neurons wie folgt lautet:

$$v = \sum_{j=1}^m w_j x_j + b \quad (2.13)$$

Ziel des Perzeptrons ist es, die Eingaben x_1, x_2, \dots, x_m korrekt in eine der beiden Klassen C_1 oder C_2 zu klassifizieren. Die Entscheidungsregel für die Klassifizierung besteht darin, den durch die Eingänge repräsentierten Punkt der Klasse C_1 zuzuordnen, wenn der Perzeptronausgang $y = +1$ ist, und der Klasse C_2 , wenn er $y = -1$ ist [13].

Aktivierungsfunktion Das „Aufladen“ des Aktivierungspotentials erfolgt durch Summation der gewichteten Ausgabewerte x_1, \dots, x_n aller eingehenden Verbindungen über die Formel:

$$\sum_{j=1}^n w_{ij} x_j \quad (2.5)$$

Diese gewichtete Summe wird bei den meisten neuronalen Modellen berechnet. Darauf wird dann eine Aktivierungsfunktion f angewendet und das Ergebnis

$$x_i = f \left(\sum_{j=1}^n w_{ij} x_j \right) \quad (2.6)$$

als Ausgabe über die synaptischen Gewichte an die Nachbarneuronen weitergegeben. [11]

Für die Aktivierungsfunktion existieren eine Reihe von Möglichkeiten. Die einfachste ist die Identität, das heißt $f(x) = x$. Das Neuron berechnet also nur die gewichtete Summe der Eingabewerte und gibt diese weiter. Dies führt aber häufig zu Konvergenzproblemen bei der neuronalen Dynamik, denn die Funktion $f(x)$ ist nicht beschränkt und die Funktionswerte können im Laufe der Zeit über alle Grenzen wachsen. [11]

Sehr wohl beschränkt hingegen ist die Schwellwertfunktion (Heavisidesche Stufenfunktion)

$$H_{\theta}(x) = \begin{cases} 0 & \text{if } x < \theta, \\ 1 & \text{else.} \end{cases} \quad (2.7)$$

Das ganze Neuron berechnet dann also seine Ausgabe als

$$x_i = \begin{cases} 0 & \text{if } \sum_{j=1}^n w_{ij}x_j < \theta, \\ 1 & \text{else.} \end{cases} \quad (2.8)$$

Die Stufenfunktion ist für binäre Neuronen durchaus sinnvoll, denn die Aktivierung eines Neurons kann ohnehin nur die Werte null oder eins annehmen. Für stetige Neuronen mit Aktivierungen zwischen 0 und 1 hingegen erzeugt die Stufenfunktion eine Unstetigkeit. Diese kann man aber glätten durch eine Sigmoid-Funktion wie zum Beispiel:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.9)$$

Neben der Sigmoid-Funktion wird häufig auch der Tangens Hyperbolicus (\tanh) verwendet, welcher das Intervall $[-\infty, \infty]$ streng monoton auf das Intervall $[-1, 1]$ abbildet. [11]

Sehr erfolgreich beim Deep Learning ist die ReLU:

$$\text{ReLU}(x) = \max(0, x) \quad (2.10)$$

Epochen Der Begriff der Epoche spielt eine zentrale Rolle im Training von künstlichen neuronalen Netzwerken. Eine Epoche bezeichnet im maschinellen Lernen einen vollständigen Durchlauf durch den gesamten Trainingsdatensatz. Dabei werden sämtliche Trainingsbeispiele dem Modell einmal präsentiert, wobei die Gewichte des Netzes anhand eines definierten Lernalgorithmus (z. B. Gradientenabstieg) aktualisiert werden. [7]

Wenn die Trainingsdatenmenge in kleinere Gruppen (Mini-Batches) unterteilt wird, ergibt sich folgende Beziehung:

$$\text{Anzahl der Iterationen pro Epoche} = \frac{\text{Anzahl der Trainingsbeispiele}}{\text{Batch-Größe}} \quad (2.11)$$

Die Anzahl der durchzuführenden Epochen ist ein Hyperparameter, der je nach Problemstellung gewählt werden muss. Zu wenige Epochen führen zu *Underfitting* – das Modell hat nicht genügend Muster gelernt. Zu viele Epochen hingegen können *Overfitting* verursachen – das Modell passt sich zu stark an den Trainingsdatensatz an und generalisiert schlecht auf unbekannte Daten. [11]

Lernrate Die Lernrate ist vielleicht der wichtigste Hyperparameter. Sie steuert die effektive Kapazität des Modells auf komplizierte Weise als anderer Hyperparameter – die effektive Kapazität des Modells ist am höchsten, wenn die Lernrate für das Optimierungsproblem richtig ist, nicht wenn die Lernrate besonders groß oder besonders klein ist. [7]

Eine hohe Lernrate kann dazu führen, dass das Modell über optimale Lösungen hinausgeschickt, während eine niedrige Rate zu einer langsamen Konvergenz oder Stagnation führen kann. [7]

Layer-Struktur Das einfache Perzeptron besteht aus nur einer Schicht (einlagig) mit einem einzigen Ausgabeneuron.

Das *Multilayer Perceptron (MLP)* besteht in der Regel aus einer Reihe von sensorischen Einheiten (Quellknoten), die die Eingabeschicht bilden, einer oder mehreren verborgenen Schichten (Hidden Layers) von Rechenknoten und einer Ausgabeschicht von Rechenknoten. Das Eingangssignal breitet sich schichtweise in Vorwärtsrichtung durch das Netz aus. Diese neuronalen Netze werden gemeinhin als Multilayer Perceptrons (MLPs) bezeichnet.

Mehrschichtige Perzeptrons wurden erfolgreich zur Lösung einiger schwieriger und vielfältiger Probleme eingesetzt, indem sie mit einem sehr beliebten Algorithmus, dem *Error-Backpropagation-Algorithmus*, unter Aufsicht trainiert wurden. Dieser Algorithmus basiert auf der Fehlerkorrektur-Lernregel [13].

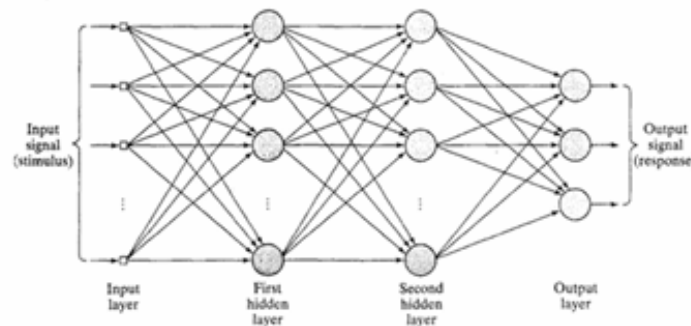


Abbildung 2.10: Architektonischer Graph eines MLP mit zwei verborgenen Schichten

Convolutional Neural Networks (CNNs) stellen eine Möglichkeit dar, die Dimension des Gewichtsraumes zu begrenzen und damit eine stabile Konvergenz des Trainings zu gewährleisten. Um die Zahl der Gewichte klein zu halten und damit Rechenzeit beim Training zu sparen, sind die Merkmalschichten nicht voll vernetzt, sondern jedes Merkmalsneuron erhält nur Eingaben von einigen wenigen Neuronen der darunter liegenden Schicht. Neben den Convolution-Schichten können auch, typischerweise im Wechsel, *Pooling-Schichten* verwendet werden [11].

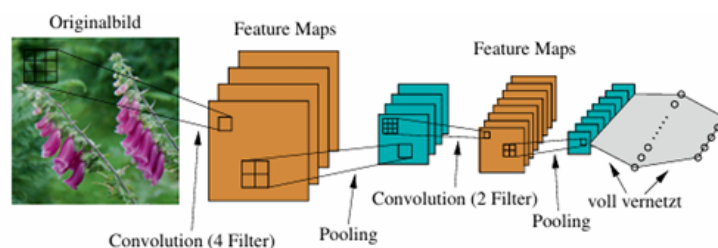


Abbildung 2.11: Ein CNN mit zwei Faltungslagen

Recurrent Neural Networks (RNNs) adressieren das Problem, dass standardmäßige Feedforward-Netze eine feste Anzahl von Eingängen und Ausgängen erwarten. Wird ein solches Netz zur Verarbeitung von Wortfolgen in natürlicher Sprache verwendet, tritt häufig das Problem auf, dass Wörter an unterschiedlichen Stellen dieselbe semantische Bedeutung tragen können, was ein festes Eingabeformat einschränkt [14].

Eine Lösung besteht in der Einführung einer expliziten versteckten Variablen z_n , die mit jedem Schritt n in der Sequenz verbunden ist. Das neuronale Netz nimmt als Eingabe sowohl das aktuelle Wort x_n als auch den aktuellen versteckten Zustand z_{n-1} und erzeugt ein Ausgabewort y_n sowie den nächsten Zustand z_n der versteckten Variablen. Kopien dieses Netzes können miteinander verkettet werden, wobei

die Gewichtungswerte über alle Zeitschritte hinweg geteilt werden. Die resultierende Architektur wird als rekurrentes neuronales Netz (RNN) bezeichnet [12].

2.2.9 Vergleich der Architekturtypen

Zusammenfassend lässt sich festhalten, dass sich neuronale Netzwerkarchitekturen in ihrer Struktur, Funktionalität und Anwendung deutlich unterscheiden. Während das einfache Perzeptron und das MLP universelle Klassifikations- und Regressionsprobleme adressieren, zeichnen sich CNNs durch ihre Effizienz und Genauigkeit bei bildbasierten Aufgaben aus. RNNs hingegen erschließen den Bereich der sequenziellen und zeitabhängigen Datenanalyse. Die Auswahl einer geeigneten Architektur ist somit entscheidend für den Erfolg eines KI-Systems und richtet sich stets nach der Natur der zugrundeliegenden Daten und der Zielstellung der Anwendung. [13].

Architekturtyp	Hauptmerkmale	Typische Anwendung
Perzeptron	Einfach, linear separierbare Probleme	Lineare Klassifikation
MLP	Tiefe Netzwerke, Backpropagation	Allgemeine Klassifikation und Regression
CNN	Lokale Merkmalsextraktion, Bildfokus	Bild- und Videoanalyse
RNN	Gedächtnis für Sequenzen	Sprachverarbeitung, Zeitreihen

Tabelle 2.5: Vergleich der Architekturtypen

2.2.10 Anwendungen von Künstlicher Intelligenz in messtechnischen Kontexten

Die Integration von Künstlicher Intelligenz (KI) in messtechnische Systeme hat in den letzten Jahren stark an Bedeutung gewonnen. Durch den Einsatz von Machine Learning (ML) und neuronalen Netzwerken können Messsysteme nicht nur effizienter, sondern auch intelligenter gestaltet werden. Insbesondere bei der Analyse großer Datenmengen, der Erkennung von Anomalien sowie der Vorhersage komplexer Messzusammenhänge zeigt sich der Nutzen KI-basierter Verfahren [15].

Automatisierte Datenauswertung und Mustererkennung Traditionelle Messsysteme liefern häufig hochdimensionale, kontinuierliche Datenströme, deren manuelle Auswertung zeitaufwändig und fehleranfällig ist. KI-gestützte Systeme – insbesondere auf Basis überwachter Lernverfahren – ermöglichen eine automatisierte Klassifikation von Messsignalen, z. B. zur Unterscheidung verschiedener Materialzustände in der zerstörungsfreien Prüfung oder zur Detektion von Defekten in Produktionsprozessen. Convolutional Neural Networks (CNNs) werden dabei erfolgreich zur Bildauswertung in optischen oder thermografischen Messsystemen eingesetzt, wo sie Strukturen erkennen, segmentieren oder klassifizieren können [15].

Anomalieerkennung und Qualitätssicherung Unüberwachte Lernverfahren wie Clustering oder Autoencoder-Netze kommen zunehmend in der Qualitätssicherung zum Einsatz. Sie können Unregelmäßigkeiten in Sensordaten erkennen, die auf Produktionsfehler, Verschleiß oder Systemausfälle hinweisen. Solche Anomalieerkennungsmodelle sind besonders in sicherheitskritischen Bereichen wie der Luftfahrt, im Energiesektor oder in der Medizintechnik von großer Bedeutung [16].

Sensorfusion und Kalibrierung Moderne Messsysteme beruhen häufig auf der Kombination mehrerer Sensorquellen (Sensorfusion), um präzisere oder robustere Ergebnisse zu erzielen. Hier leisten neuronale Netzwerke wertvolle Dienste, indem sie Korrelationen zwischen den Sensorsignalen lernen und nutzen. Auch bei der Kalibrierung von Messgeräten kommen KI-Modelle zum Einsatz, um Abweichungen vom

Referenzverhalten datengetrieben zu kompensieren – insbesondere in dynamischen oder schwer modellierbaren Umgebungen [16].

Zusammenfassung Die Anwendung von KI in messtechnischen Kontexten eröffnet ein breites Spektrum an Möglichkeiten zur Verbesserung der Datenanalyse, Systemüberwachung und Prozesssteuerung. Sie trägt dazu bei, die Genauigkeit, Effizienz und Autonomie moderner Messsysteme erheblich zu steigern. Der Einsatz intelligenter Algorithmen wird künftig eine zentrale Rolle in der digitalen Transformation industrieller und wissenschaftlicher Messverfahren spielen [16].

2.3 Grundlagen der Bildvorverarbeitung

Dieser Text erläutert die Grundlagen der Bildrepräsentation, wichtige Vorverarbeitungstechniken, die mathematischen Prinzipien der Kantenerkennung sowie deren Bedeutung für die Merkmalsextraktion in Convolutional Neural Networks (CNNs).

2.3.1 Grundlagen der digitalen Bildrepräsentation

Dieser Abschnitt erläutert die grundlegenden Konzepte der digitalen Bilddarstellung, einschließlich der Rolle von Pixeln, Bildkanälen und Farbräumen.

Pixel: Die Bausteine digitaler Bilder

Ein Pixel (Bildpunkt) ist die kleinste Einheit eines digitalen Bildes. Es repräsentiert die Helligkeit oder Farbe an einer bestimmten Position im Bildraaster. Digitale Bilder werden als zweidimensionale Matrizen von Pixelwerten gespeichert (bzw. als dreidimensionale Matrizen bei Farabbildungen) [17].

In einem Graustufenbild mit 8 Bit pro Pixel nimmt jeder Pixel einen Wert zwischen 0 (schwarz) und 255 (weiß) an. Dazwischen liegende Werte stehen für verschiedene Grautöne [17].

Kanäle: Farb- oder Intensitätsebenen

Ein Kanal beschreibt eine Schicht innerhalb eines mehrdimensionalen Bildes, die Intensitätswerte für eine bestimmte Komponente enthält. RGB-Bilder zum Beispiel bestehen aus drei Kanälen: Rot, Grün und Blau. Jeder Kanal ist eine separate Graustufenmatrix, die die jeweilige Farbintensität enthält. Durch das Kombinieren dieser drei Matrizen entsteht ein vollständiges Farbbild. Weitere Kanäle wie ein Alpha-Kanal (Transparenzinformation) oder spezielle Kanäle für Tiefen- oder Infrarotinformationen können ergänzt werden, je nach Anwendung [18].

Farbräume: Mathematische Modelle der Farbdarstellung

Ein Farbraum legt fest, wie Farben numerisch beschrieben und interpretiert werden. Die Wahl des Farbraums kann die Effizienz und Genauigkeit vieler Bildverarbeitungsaufgaben maßgeblich beeinflussen [18].

RGB (Rot, Grün, Blau) Die meisten Farbbilder sind mit jeweils einer Komponente für die primären Farben Rot, Grün und Blau (RGB) kodiert, typischerweise mit 8 Bits pro Komponente. Jeder Pixel eines solchen Farbbilds besteht daher aus $3 \times 8 = 24$ Bits und der Wertebereich jeder Farbkomponente ist wiederum $[0, 255]$ [19].

Ein additives Farbmodell, bei dem verschiedene Intensitäten von Rot, Grün und Blau kombiniert werden, um andere Farben zu erzeugen [19].

HSV (Farbton, Sättigung, Helligkeit) Da sich Farbton (Hue), Farbsättigung (Saturation) und Helligkeit (Value Colour Intensity) bei jeder Koordinatenbewegung gleichzeitig ändern, ist die manuelle Auswahl von Farben im RGB-Raum schwierig und wenig intuitiv. Alternative Farbräume, wie z. B. der HSV-Raum, erleichtern diese Aufgabe, indem subjektiv wichtige Farbeigenschaften explizit dargestellt werden [19].

Es orientiert sich stärker an der menschlichen Farbwahrnehmung und ist besonders nützlich für Anwendungen wie Farberkennung, Segmentierung oder Tracking bei wechselnden Lichtverhältnissen [18].

Graustufen Ein Bild kann als eine zweidimensionale Funktion $f(x, y)$ definiert werden, wobei x und y räumliche (ebene) Koordinaten sind und die Amplitude von f an einem beliebigen Koordinatenpaar (x, y) als Intensität oder Grauwert des Bildes an diesem Punkt bezeichnet wird. Wenn x , y und die Intensitätswerte von f alle endlichen, diskreten Größen sind, spricht man von einem digitalen Bild.

2.3.2 Bildvorverarbeitungstechniken

Die Bildvorverarbeitung ist ein zentraler Schritt in der digitalen Bildverarbeitung, da sie die Qualität und Nutzbarkeit von Bilddaten für nachfolgende Analyseverfahren verbessert.

Normalisierung

Die Normalisierung (auch Standardisierung genannt) bezeichnet die Umwandlung der Pixelwerte eines Bildes in einen festgelegten Wertebereich (z. B. von $[0, 255]$ auf $[0, 1]$) oder die Anpassung auf einen Mittelwert von 0 und eine Standardabweichung von 1. Ziel dieser Maßnahme ist es, eine einheitliche Darstellung der Daten zu erreichen, um die Leistung und Konvergenzgeschwindigkeit von Lernalgorithmen – insbesondere neuronalen Netzen – zu verbessern. Große Unterschiede in der Skalierung können zu schlechteren Ergebnissen beim Training führen[19].

Gängige Methoden

Min–Max–Normalisierung Diese Methode transformiert jeden Pixelwert x nach der Formel:

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \quad (2.2.1)$$

Sie skaliert die Daten auf einen festen Bereich – in der Regel $[0, 1]$. Diese Methode ist effizient, aber empfindlich gegenüber Ausreißern.

Z-Score–Normalisierung (Standardisierung) Hierbei werden die Daten so transformiert, dass sie einen Mittelwert von 0 und eine Standardabweichung von 1 aufweisen. Die Formel lautet:

$$z = \frac{x - \mu}{\sigma} \quad (2.2.2)$$

wobei μ der Mittelwert und σ die Standardabweichung ist. Diese Methode ist robuster gegenüber Ausreißern und eignet sich besonders, wenn Merkmale unterschiedliche Einheiten oder Skalen aufweisen.

Diese Normalisierungsverfahren sind grundlegende Schritte der Bildvorverarbeitung und spielen eine entscheidende Rolle bei der Verbesserung der Stabilität und Effizienz von Lernalgorithmen.

Filterung: Gauß- und Medianfilter

Die Filterung ist eine Technik zur Rauschunterdrückung oder Glättung von Bilddaten. Dabei werden die Pixelwerte durch die lokale Nachbarschaft ersetzt oder modifiziert[18].

Zwei besonders verbreitete Filtertypen sind:

- Gaußfilter
- Medianfilter

Gaußfilter Ein Gaußfilter wendet eine gewichtete Mittelung basierend auf der Gaußschen Verteilung an. Er ist ideal zur Reduktion von hochfrequentem Rauschen, da er weiche Übergänge erzeugt und gleichzeitig Bildkanten weitgehend erhält. Mathematisch entspricht der Filter einer Faltung mit einer zweidimensionalen Gaußfunktion:

$$H^{G,\sigma}(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2.2.3)$$

Ein zweidimensionales Gaußfilter kann als Hintereinander-Ausführung zweier eindimensionaler Gaußfilter formuliert werden, was eine effiziente Berechnung ermöglicht.

Medianfilter Der Medianfilter ist ein nicht-linearer Filter, der jeden Pixel durch den Medianwert seiner Nachbarschaft ersetzt:

$$I(u,v) \leftarrow \text{median}\{I(u+i, v+j) \mid (i,j) \in R\} \quad (2.2.4)$$

Er ist besonders effektiv zur Beseitigung von Impulsrauschen, ohne dabei Kanten zu stark zu glätten.

Histogramm Ausgleich (Histogram Equalization)

Der Histogramm Ausgleich ist eine Technik zur Kontrastverbesserung, bei der die Helligkeitsverteilung eines Bildes gestreckt oder angepasst wird, sodass möglichst viele Intensitätsstufen gleichmäßig genutzt werden. Dadurch werden Details in dunklen oder hellen Bildbereichen besser sichtbar. Das Verfahren basiert auf der kumulativen Verteilungsfunktion des Histogramms.[2]

Unter Berücksichtigung kontinuierlicher Intensitätswerte sei r die Variable für die Intensitäten eines zu verarbeitenden Bildes im Bereich $[0, L-1]$, wobei $r = 0$ Schwarz und Weiß darstellt.[2] In der Praxis wird das Histogramm des Bildes analysiert und durch Transformation der Graustufen neu verteilt:

$$s_k = (L-1) \sum_{j=0}^k p_r(r_j) \quad (2.2.5)$$

2.3.3 Mathematische Grundlagen der Kantendetektion

Die Kantendetektion ist ein grundlegender Bestandteil der Bildverarbeitung und Computer Vision, da sie relevante Informationen über die Struktur eines Bildes liefert. Die mathematische Basis umfasst die Faltung (Convolution), Gradientenoperatoren und den Laplace-Operator.

Korrelation und Faltung (Convolution)

Korrelation ist der Vorgang, bei dem eine Filtermaske über das Bild bewegt und die Summe der Produkte an jeder Stelle berechnet wird, genau wie im vorherigen Abschnitt erläutert. Die Mechanik der Faltung ist dieselbe, außer dass der Filter zunächst um 180° gedreht wird.[2]

Die Faltung ist eine zentrale mathematische Operation in der Bildverarbeitung. Sie verknüpft zwei Funktionen gleicher Dimensionalität, kontinuierlich oder diskret. Für diskrete, zweidimensionale Funktionen I und H ist die Faltungsoperation definiert als:

$$I'(u,v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) H(i,j) \quad (2.2.6)$$

Sie dient als Grundlage für viele Operationen, darunter Glättung, Rauschunterdrückung und insbesondere Kantendetektion[20].

Gradientenoperatoren: Prewitt und Sobel-Operator

Sie dient als Grundlage für viele Operationen, darunter Glättung, Rauschunterdrückung und insbesondere Kantendetektion. Zwei klassische Verfahren sind die Kanteneoperatoren von Prewitt und Sobel, die einander sehr ähnlich sind und sich nur durch geringfügig abweichende Gradientenfilter unterscheiden[20].

Prewitt-Operator Der Prewitt-Operator verwendet einfache 3×3 -Masken, um horizontale und vertikale Kanten zu erfassen:

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{und} \quad H_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad (2.2.7)$$

Sobel-Operator Die Filter des Sobel-Operators sind fast identisch, geben allerdings durch eine etwas andere Glättung mehr Gewicht auf die zentrale Zeile bzw. Spalte:

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{und} \quad H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (2.2.8)$$

Laplace-Operator

Für die Kantenschärfung im zweidimensionalen Fall verwendet man die zweiten Ableitungen in horizontaler und vertikaler Richtung kombiniert in Form des Laplace-Operators:

$$(\nabla^2 f)(x, y) = \frac{\partial^2 f}{\partial x^2}(x, y) + \frac{\partial^2 f}{\partial y^2}(x, y) \quad (2.2.9)$$

Typische Laplace-Masken (diskret) sind z. B.:

$$H = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (2.2.10)$$

2.3.4 Die Rolle der Kantendetektion**Bedeutung der Kantendetektion für die Merkmalsextraktion**

Kanten repräsentieren abrupte Änderungen in der Intensität eines Bildes und markieren damit Objektgrenzen, Texturen oder Formübergänge. Bei der Merkmalsextraktion geht es darum, aus Rohdaten (z. B. einem Bild) charakteristische Informationen zu extrahieren, die zur Erkennung oder Klassifikation verwendet werden können[18].

Kanten liefern:

- lokalisierte Strukturinformationen (z. B. Umrisse, Konturen),
- reduzierte Datenkomplexität durch Fokus auf relevante Regionen,
- Invarianz gegenüber Helligkeit und Farbe, da sie meist nur von Intensitätsänderungen abhängig sind[18].

Rolle in Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) lernen Merkmale in hierarchischen Schichten. In den frühen Schichten eines CNNs erkennen die Filter typischerweise einfache Muster – vor allem Kanten in verschiedenen Richtungen.

Diese Kantenmuster bilden die Grundlage für:

- die Lokalisation von Objekten,
- die Erkennung von Formen und Texturen in späteren Schichten,
- die Reduktion von Redundanz, da irrelevante Informationen verworfen werden[\[21\]](#).

Kapitel 3

KI in der Messtechnik

Künstliche Intelligenz hat die Messtechnik erheblich verändert, insbesondere durch Computer-Vision-Methoden wie Convolutional Neural Networks. Diese Netzwerke ermöglichen eine präzise Objektlokalisierung innerhalb von Bildern und erleichtern automatisierte Messungen in komplexen Umgebungen. Schlüsselaufgaben wie die Erstellung von Datensätzen, Bildbeschriftung und Datenaugmentation verbessern die Modellgenauigkeit und Widerstandsfähigkeit gegenüber Herausforderungen wie Beleuchtungsinkonsistenzen und Verdeckungen. Fortschrittliche Architekturen wie „You Only Look Once“ und „Residual Network“ mit Regressionsköpfen werden häufig zur Koordinatenschätzung und Schlüsselpunktdetektion eingesetzt.

Über die Objektlokalisierung hinaus verbessert Künstliche Intelligenz Echtzeit-Messsysteme durch Unterdrückung von Signalrauschen, Kalibrierung von Sensoren und Anomalieerkennung, wodurch eine verbesserte Zuverlässigkeit und Betriebseffizienz gewährleistet wird. Industrien nutzen Künstliche Intelligenz für vorausschauende Diagnosen, die Optimierung von Messprozessen und die Rationalisierung von Arbeitsabläufen.

Applications of Convolutional Neural Networks span across object localization using YOLO and Faster Region-Based Convolutional Neural Network (Faster R-CNN), as well as denoising, calibration, and anomaly detection. These technologies, once theoretical, are now integral to smart factories, precision agriculture, transportation, and infrastructure monitoring. [22]

Künstliche Intelligenz hat sich von einem theoretischen Feld zu einer transformativen Kraft in verschiedenen Bereichen entwickelt, angetrieben durch Maschinelles Lernen und dessen Unterfeld, Deep Learning. Neuronale Netze, inspiriert von der Gehirnstruktur, bilden die Grundlage moderner KI-Lösungen. Dieses Kapitel untersucht, wie Convolutional Neural Networks Messtechnologien revolutioniert haben und über konventionelle manuelle Kalibrierung und vordefinierte Modellierung hinausgehen.

Deep Learning ermöglicht es Modellen, umfangreiche Datenströme autonom zu verarbeiten, sich dynamisch anzupassen und mit größerer Fehlertoleranz zu arbeiten. Anwendungen von Convolutional Neural Networks umfassen die Objektlokalisierung mittels YOLO und Faster Region-Based Convolutional Neural Network (Faster R-CNN) sowie Entrauschung, Kalibrierung und Anomalieerkennung. Diese Technologien, einst theoretisch, sind heute integraler Bestandteil von Smart Factories, Präzisionslandwirtschaft, Transport und Infrastrukturüberwachung. [23]

3.1 Objektlokalisierung mittels Computer Vision

Die Objektlokalisierung spielt eine entscheidende Rolle in der Assistenztechnologie für sehbehinderte Personen (VIPs). Computer-Vision-basierte Systeme nutzen Tiefenkameras, RGB-Sensoren und Algorithmen des maschinellen Lernens, um Hindernisse zu erkennen und den Benutzern Echtzeit-Feedback zu geben.

Die Positionsschätzung ist eine grundlegende Aufgabe in der Messtechnik, da sie Anwendungen in der Robotik, autonomen Navigation, Augmented Reality und Assistenztechnologien für sehbehinderte Personen findet. Traditionelle Methoden basieren auf Sensorfusion und geometrischen Computer-Vision-

Techniken. Diese Ansätze haben jedoch oft Schwierigkeiten in dynamischen Umgebungen aufgrund von Rauschen, Verdeckungen und Rechenkomplexität.

Convolutional Neural Networks (CNNs) haben sich als leistungsstarke Alternative erwiesen, die ein End-to-End-Lernen räumlicher Beziehungen aus visuellen Daten ermöglicht. Im Gegensatz zu klassischen Methoden können CNNs Objektpositionen direkt aus Rohbildern vorhersagen, wodurch die Abhängigkeit von handgefertigten Merkmalen reduziert wird.

Dieser Abschnitt untersucht, wie CNNs für die Positionsschätzung angewendet werden, wobei der Schwerpunkt auf Architekturdesign, Trainingsmethoden und realen Implementierungen liegt. [24]

Mehrere Studien haben fortgeschrittene Lokalisierungstechniken untersucht:

- **Kinect-Tiefenkameras:** Diese Systeme analysieren Tiefenkarten, um Objektpositionen und -entfernungen zu bestimmen und die Genauigkeit der Hinderniserkennung zu verbessern.
- **Ultraschall- und RGB-Kamera-basierte Systeme:** Geräte integrieren Ultraschallsensoren mit RGB-Kameras, um die Lokalisierungspräzision zu verbessern und Feedback über Audio- oder haptische Signale zu geben.
- **Algorithmen des maschinellen Lernens:** Ansätze wie CNN, YOLO (You Only Look Once) und SVM (Support Vector Machine) wurden verwendet, um die Objekterkennung und das räumliche Bewusstsein zu verbessern. Diese Systeme erzeugen 3D-Darstellungen der Umgebung und verbessern so die Fähigkeit des Benutzers, sich sicher zu navigieren.
- **Tragbare Systeme:** Einige Assistenztechnologien integrieren Smart Glasses, mobile Apps oder am Kopf getragene Geräte mit Deep-Learning-Modellen, um Hindernisse zu erkennen und zu lokalisieren. Diese Systeme erreichen hohe Genauigkeitsraten und helfen VIPs, Kollisionen in dynamischen Umgebungen zu vermeiden.
- **Semantische Objekterkennung und Tiefenkarten:** Durch die Kombination von Objekterkennung mit Tiefenschätzung ermöglichen einige Lösungen Benutzern, zwischen verschiedenen Arten von Hindernissen zu unterscheiden und deren Nähe in Echtzeit zu beurteilen. [25]

3.1.1 Assistive Technologien für sehbehinderte Menschen

Assistive Technologien für sehbehinderte Menschen lassen sich in drei Hauptkategorien einteilen: Objekterkennungsgeräte, Navigationssysteme und Hybridgeräte, die beide Funktionen kombinieren. Diese Systeme vermitteln Informationen durch Ton oder Vibration und unterstützen Nutzer dabei, Hindernisse zu erkennen und sich sicher in ihrer Umgebung zu bewegen.

Objekterkennungsgeräte Diese Geräte nutzen Sensoren, Ultraschallsysteme oder Kameras, um Umgebungsdaten zu erfassen und dem Nutzer Feedback zu geben [26]:

- Kinect-Tiefenkameras und Ultraschallstöcke erkennen Hindernisse und berechnen deren Entfernung mithilfe adaptiver Schwellenwerte oder Schallwellen.
- Multisensorsysteme setzen maschinelles Lernen (z.B. SVM, CNN) zur Objekterkennung ein und liefern akustisches oder haptisches Feedback.
- Tragbare Lösungen wie Brillen oder Smartphone-basierte Deep-Learning-Modelle erreichen eine hohe Genauigkeit bei der Hinderniserkennung.

Navigationssysteme Navigationshilfen leiten VIPs mithilfe von GPS, RFID und AR-Technologien:

- Geräte mit RGB-D-Sensoren unterstützen Nutzer durch Tiefen- und Farbinformationen.
- Einige Wearables nutzen Sprachausgabe, haptische Signale oder Audiodaten für Echtzeitführung.

- Bildgestützte Systeme helfen beim Joggen, Laufen und Gehen, z.B. durch Handschuhe mit Bewegungserkennung [27].

Hybridsysteme Diese kombinieren Objekterkennung und Navigation für eine verbesserte Mobilität:

- Intelligente Blindenstöcke, kopfbasiertes Equipment und mobile Apps erkennen Hindernisse und unterstützen gleichzeitig die Wegführung.
- KI-gestützte Lösungen (z.B. YOLO-basierte Erkennung und Echtzeit-Segmentierung) identifizieren komplexe Hindernisse wie Wände oder Türen mit hoher Präzision.
- Einige Systeme nutzen elektrotaktilen Feedback, z.B. durch Zungenstimulation, um die Navigation zu erleichtern [28].

3.1.2 Anwendung von Convolutional Neural Networks (CNNs) zur Positionsschätzung

CNNs sind zu einem leistungsfähigen Werkzeug für die Positionsschätzung geworden, insbesondere in Anwendungen, die eine präzise Objektlokalisierung und -verfolgung erfordern. Ihre Fähigkeit, räumliche Merkmale aus Bildern und Sensordaten zu extrahieren, macht sie in der Robotik, autonomen Navigation und industriellen Automatisierung hochwirksam. [29]

Die Positionsschätzung beinhaltet die Bestimmung der räumlichen Koordinaten oder der Orientierung von Objekten in Bildern oder Sensordaten. CNNs werden für diese Aufgabe aufgrund ihrer Fähigkeit, hierarchische Merkmale aus Rohdaten zu lernen, weit verbreitet eingesetzt. Nachfolgend finden Sie eine Aufschlüsselung, wie CNNs in diesem Bereich angewendet werden, zusammen mit Schlüsselarchitekturen und Herausforderungen. [27]

Wie CNNs die Positionsschätzung ermöglichen

CNNs verarbeiten visuelle und Sensordaten, um die Position eines Objekts zu schätzen, indem sie:

- **Merkmalsextraktion:** Schlüsselmuster in Bildern oder Sensorinputs identifizieren.
- **Tiefen- & Raumwahrnehmung:** Mehrschichtige Architekturen zur Analyse von Objektabständen und -orientierungen nutzen.
- **Echtzeitanpassung:** Positionsschätzungen kontinuierlich auf der Grundlage von Umweltveränderungen verfeinern.

Anwendungen von CNNs in der Positionsschätzung

- **Robotik & Steuerung:** CNNs helfen selbstfahrenden Autos, Fahrspurpositionen, Hindernisse und Fußgänger zu erkennen. In der Robotik ermöglichen CNN-basierte Vision-Systeme eine präzise Objektmanipulation und Navigation. CNNs werden in der Robotik häufig für Lokalisierung und autonome Bewegung eingesetzt. Sie helfen bei der Schätzung der Position von Objekten für Roboterarme, was eine präzise Manipulation in der Industrieautomation und Gesundheitsrobotik ermöglicht.
- **Cybersicherheit:** CNNs können helfen, positionsbasiertes Verhalten in Cybersicherheitssystemen zu schätzen, indem sie Anomalien in der Netzwerkaktivität verfolgen und bösartige Zugangspunkte erkennen. Zum Beispiel analysieren Deep-Learning-Modelle Muster in Cyberangriffen, um deren Ursprung und Position innerhalb eines Systems zu bestimmen und so Intrusionen zu verhindern. Sie unterstützen die Lagerautomatisierung, indem sie Roboter an bestimmte Orte führen.

- **Medizinische Informationsverarbeitung:** CNNs verbessern die Positionsverfolgung in der Roboterchirurgie und gewährleisten Präzision. Sie helfen bei der MRT- und CT-Ausrichtung, verbessern die diagnostische Genauigkeit, indem sie die Position von Anomalien in Röntgen-, MRT- und CT-Bildern bestimmen. Sie unterstützen die Roboterchirurgie, indem sie chirurgische Instrumente mit hoher Präzision verfolgen.
- **Natural Language Processing (NLP):** CNNs können bei der Positionsschätzung von Schlüsselwörtern oder -phrasen innerhalb großer Datensätze helfen. Zum Beispiel helfen CNNs, ortsbezogene Informationen aus Textdaten zu extrahieren, was Anwendungen wie die geospatialen Analysen für die Smart-City-Entwicklung verbessert.

3.1.3 Vorverarbeitungsschritte

Die Vorverarbeitung ist eine kritische Phase bei der Entwicklung robuster Computer-Vision-Modelle, insbesondere für Aufgaben wie Bildklassifizierung, Objekterkennung oder Segmentierung. Dieser Abschnitt befasst sich mit drei wichtigen Vorverarbeitungsschritten: Datensatzerstellung, Bildbeschriftung und Datenaugmentation, die zusammen eine qualitativ hochwertige Eingabedaten für das Training von Deep-Learning-Modellen gewährleisten.

Datensatzerstellung

Die Datensatzerstellung umfasst das Sammeln und Organisieren von Bildern, die für das Problemfeld relevant sind. Wichtige Überlegungen sind:

- **Datenquellen:**
 - Öffentliche Datensätze (z. B. ImageNet, COCO, CIFAR-10).
 - Benutzerdefinierte Datensätze (z. B. Web-Scraping, manuelle Sammlung oder synthetische Datengenerierung). [30]
- **Diversität und Repräsentativität:**
 - Sicherstellen, dass der Datensatz Variationen in Beleuchtung, Winkeln, Hintergründen und Objektskalen abdeckt, um Verzerrungen zu vermeiden.
 - Für die medizinische Bildgebung müssen Datensätze seltene Fälle enthalten, um die Generalisierbarkeit des Modells zu verbessern. [31]
- **Strukturierte Speicherung:**
 - Bilder in Ordnern organisieren (z. B. klassenweise für Klassifizierungsaufgaben).
 - Standardisierte Formate (JPEG, PNG) und Auflösungen (z. B. 256x256 für Konsistenz) verwenden. [30]

Bildbeschriftung

Die Beschriftung weist Bildern sinnvolle Anmerkungen zu und ermöglicht uns ein überwachtes Lernen. In einer 2022 veröffentlichten Studie hat ein Forscherteam ein schwach überwachtes Convolutional Neural Network (CNN) zur Objektklassifizierung entwickelt, das während des Trainings nur Labels auf Bildebene (die das Vorhandensein oder Fehlen eines Objekts angeben) verwendet und dennoch in der Lage ist, aus überladenen Szenen mit mehreren Objekten zu lernen. Die Leistung des Modells wird an zwei Benchmark-Datensätzen bewertet: Pascal VOC 2012 (20 Objektklassen) und dem deutlich größeren Microsoft COCO (80 Objektklassen). Die Analyse zeigt, dass das Netzwerk Folgendes konnte:

- Erreicht hohe Genauigkeit bei der Vorhersage von Labels auf Bildebene.

- Lokalisiert Objekte ungefähr (aber nicht deren präzise Ausdehnung).
- Erreicht eine Leistung, die mit vollständig überwachten Gegenständen vergleichbar ist, die mit Bounding-Box-Annotationen trainiert wurden.

Bemerkenswert ist, dass das Netzwerk lernt, Objekte ohne jegliche räumliche Überwachung (z. B. Bounding Boxes oder Segmentierungsmasken) während des Trainings zu lokalisieren, indem es sich ausschließlich auf binäre Objekt-Präsenz-Labels stützt. Darüber hinaus werden Objekte mit typischerem Aussehen (z. B. Motorräder, wie in der linken Spalte gezeigt) im Trainingsprozess früher lokalisiert als solche mit größerer Variabilität innerhalb der Klasse. [32]

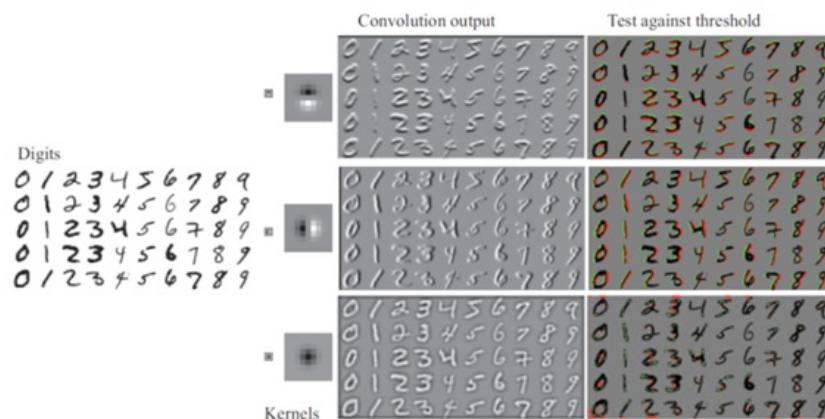


Abbildung 3.1: Kernreaktionen auf Ziffern angewendet

CNNs zeichnen sich bei der Bildklassifizierung durch ihre Fähigkeit aus, hierarchische Merkmalsdarstellungen direkt aus Pixeldaten zu lernen. Um zu verstehen, warum CNNs gut funktionieren, ist es lehrreich, Bilddatensätze wie MNIST zu untersuchen, einen Standard-Benchmark, der aus 70.000 handgeschriebenen Ziffern (0–9) besteht. Die Analyse dieses Datensatzes zeigt zwei Schlüsseleigenschaften von Bildern:

1. **Invarianz gegenüber lokalen Störungen:** Kleine Transformationen (z. B. Translation, Rotation, Helligkeitsänderungen) verändern die semantische Bedeutung eines Bildes nicht. Zum Beispiel bleibt eine Ziffer „8“ erkennbar, auch wenn sie leicht verschoben wird, was impliziert, dass genaue Pixelpositionen weniger kritisch sind als strukturelle Muster.
2. **Lokale und hierarchische Merkmalsbedeutung:** Diskriminierende Merkmale ergeben sich oft aus lokalisierten Mustern und ihren räumlichen Beziehungen.

Praktische Verbesserungen und Herausforderungen Obwohl CNNs ohne umfangreiche Abstimmung eine starke Leistung erzielen können, verbessern verschiedene Techniken die Robustheit weiter:

- **Datenaugmentation:** Die künstliche Erweiterung des Trainingsdatensatzes mit gestörten Kopien reduziert Overfitting und verbessert die Generalisierung. Testzeit-Augmentierung kann auch die Vorhersagestabilität verbessern.
- **Kontextuelles Lernen:** CNNs lernen implizit, diskriminierende Regionen zu priorisieren, selbst wenn Objekte nur einen Bruchteil der Pixel einnehmen. Kontextuelle Hinweise können die Erkennung unterstützen, obwohl ihr Nutzen von den Datenspezifika abhängt. [33]

Datenaugmentation

Datenaugmentation erweitert den Datensatz künstlich durch Anwenden von Transformationen auf vorhandene Bilder, wodurch die Modellrobustheit verbessert und Overfitting reduziert wird. Techniken umfassen:

Grundlegende Augmentierungen

- **Geometrische Transformationen:**
 - Rotation: Bilder um 10° – 30° drehen
 - Spiegelung: Horizontale/vertikale Spiegelungen
 - Zuschneiden/Translation: Zufällige Ausschnitte oder Verschiebungen
- **Photometrische Transformationen:**
 - Helligkeit/Kontrast: Beleuchtungsbedingungen anpassen
 - Farbjittering: RGB-Kanäle stören
 - Rauschinjektion: Gaußsches Rauschen hinzufügen, um Sensorartefakte zu simulieren

Fortgeschrittene Augmentierungen

- Zufälliges Löschen (Random Erasing): Zufällige Bereiche maskieren, um den Fokus auf ganze Objekte zu erzwingen
- Mixup: Zwei Bilder linear mischen
- **GAN-basierte Augmentierung:**
 - Generieren synthetischer Bilder
 - Stiltransfer
- **Meta-Learning Augmentierungen:**
 - Verwenden von Reinforcement Learning, um optimale Augmentierungsstrategien zu entdecken

Datenaugmentation im maschinellen Lernen Die synthetische Generierung von gelabelten Daten deutet darauf hin, dass dies durch die Nutzung inhärenter statistischer Symmetrien in den Daten geschieht. Viele Anwendungen des maschinellen Lernens umfassen Datensätze, die, zumindest annähernd, intrinsische Symmetrien und Invarianzen aufweisen. Zum Beispiel bleibt ein gedrehtes Bild einer handgeschriebenen Zahl als diese Zahl erkennbar, und eine an einem bestimmten Ort aufgenommene Temperaturmessung wird einer nur wenige Augenblicke später vorgenommenen Messung sehr ähnlich sein. Datenaugmentation nutzt solche Symmetrien und Invarianzen, um Rohdatensätze durch die Einführung zusätzlicher synthetischer Daten zu erweitern. [34]

Gegeben sei ein Rohdatensatz:

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\} \quad (3.1)$$

Der entsprechende augmentierte Datensatz wird bezeichnet als:

$$D' = \{(x_{11}, y_1), (x_{12}, y_1), \dots, (x_{mB}, y_m)\} \quad (3.2)$$

wobei $m' = B \times m$ die Gesamtgröße des augmentierten Datensatzes darstellt.

Wenn empirische Risikominimierungen (ERM)-Methoden verwendet werden, tritt häufig Overfitting auf, wenn die Dimensionalität des Hypothesenraums die Anzahl der Trainingsbeispiele übersteigt. Regularisierung und Modellauswahl mildern dieses Problem, indem sie den Hypothesenraum einschränken. Ein alternativer Ansatz ist die Datenaugmentation, bei der der Trainingsdatensatz synthetisch erweitert wird, indem inhärente Datensymmetrien genutzt werden – zum Beispiel bleibt ein gedrehtes Bild einer Katze eine gültige Darstellung einer Katze. [35]

Datenaugmentation dient als implizite Regularisierungstechnik, wobei die Störverteilung $p(\epsilon)$ den Regularisierungseffekt bestimmt. Dieser Ansatz ermöglicht es, den Bias-Varianz-Kompromiss ohne explizite Strafterme auszugleichen, was ihn besonders in Szenarien mit wenig Daten wertvoll macht. [25]

3.1.4 Modelle für die Koordinatenregression

Die Koordinatenregression beinhaltet die Vorhersage kontinuierlicher numerischer Werte (Koordinaten) aus Eingabedaten. Die Koordinatenregression ist also der Prozess der Vorhersage präziser räumlicher Koordinaten aus einer gegebenen Eingabe, anstatt Objekte in Kategorien zu klassifizieren. Bei Computer-Vision-Aufgaben ist die Koordinatenregression für Anwendungen wie menschliche Haltungsschätzung, Objekterkennung und -verfolgung unerlässlich. Traditionelle CNN-Architekturen verlassen sich oft auf Heatmap-basierte Methoden für räumliche Aufgaben, aber DSNT führt einen direkteren Ansatz für die Koordinatenregression ein, während räumliche Generalisierbarkeit und End-to-End-Differenzierbarkeit erhalten bleiben. [26]

Problemdefinition Die Haltungsschätzung für mehrere Personen zielt darauf ab, anatomische Schlüsselpunkte für alle Personen in einem Bild zu identifizieren. Jede Haltung wird durch K 2D-Schlüsselpunkte dargestellt.

Entkoppelte Schlüsselpunktregression (DEKR) Das DEKR-Framework sagt Haltungskandidaten an jedem zentralen Pixel q voraus, indem es einen $2K$ -dimensionalen Offsetvektor o_q schätzt, der q auf die K Schlüsselpunkte abbildet. Offset-Karten O werden über eine Regressionsfunktion F generiert: $O = F(X)$, wobei X Backbone-Merkmale darstellt (HRNet in dieser Studie).

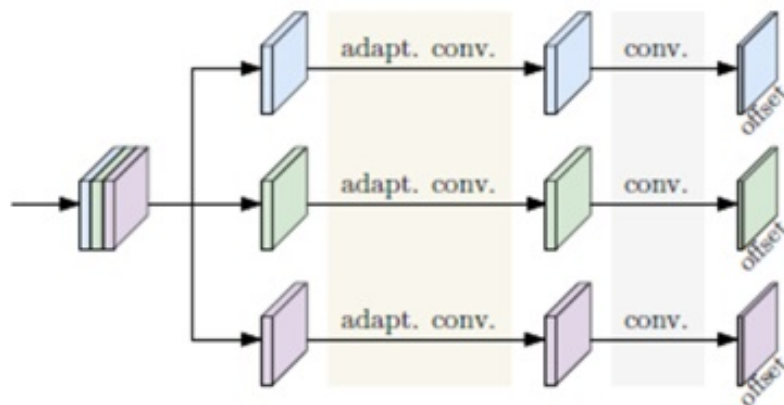


Abbildung 3.2: Diese Abbildung veranschaulicht den DEKR-Prozess

Schlüsselinnovationen

- **Adaptive Faltungen:** Im Gegensatz zu Standardfaltungen passen adaptive Faltungen die rezeptiven Felder dynamisch an, um sich auf Schlüsselpunktregionen zu konzentrieren. Für ein zentrales Pixel q wird die Aktivierung $y(q)$ berechnet als:

$$y(q) = \sum W_i x(q + g_{s_i} q) \quad (3.3)$$

wobei $g_{s_i} q$ gelernte Offsets sind, die aus einer affinen Transformation A_q und Translation t abgeleitet werden, was eine pixelweise räumliche Anpassung ermöglicht.

- **Mehrzweig-Regression:** DEKR verwendet K parallele Zweige, von denen jeder eine dedizierte Merkmalskarte X_k verarbeitet, um Offsets für den k -ten Schlüsselpunkt vorherzusagen:

$$O_k = F_k(X_k), \quad \text{für } k = 1, \dots, K \quad (3.4)$$

Dies entkoppelt das Merkmalenlernen und verbessert die Lokalisierungsgenauigkeit. [36]

Verlustfunktionen

- **Regressionsverlust:** Der Smooth-L1-Verlust bestraft Offset-Fehler, normalisiert durch die Instanzgröße Z_i :

$$\ell_p = \sum \left(\frac{1}{Z_i} \cdot \text{smooth}_{L1}(o_i - o_k) \right) \quad (3.5)$$

- **Heatmap-Verlust:** Ein separater Zweig sagt Schlüsselpunkt-Heatmaps H und Zentrum-Heatmaps C voraus, die mittels maskiertem L2-Verlust überwacht werden:

$$\ell_h = \|M_h \odot (H - H^*)\|_2^2 + \|M_c \odot (C - C^*)\|_2^2 \quad (3.6)$$

Hierbei gewichten M_h und M_c Nicht-Schlüsselpunktregionen herunter.

- **Gesamtverlust:** [37]

$$\ell = \ell_h + \lambda \ell_p, \quad \text{wobei } \lambda = 0,03 \quad (3.7)$$

Vorteile

- **Präzision:** Die Mehrzweig-Regression minimiert Interferenzen zwischen Schlüsselpunkten.
- **Flexibilität:** Adaptive Faltungen verbessern den Fokus auf Schlüsselpunktregionen. [36]

3.1.5 Robustheitsüberlegungen: Beleuchtung, Verdeckung, Echtzeit-Feedback

In einer Studie über CNN-Varianten untersuchten Forscher Beleuchtungsüberlegungen und betonten die Bedeutung der Beleuchtung in maschinellen Visionssystemen. Sie hoben hervor, dass „Beleuchtung das zu inspizierende Teil so beleuchtet, dass seine Merkmale hervorstechen und von der Kamera präzise gesehen werden können“, und diskutierten verschiedene Beleuchtungsaufbauten wie Front- und Hintergrundbeleuchtung zur Verbesserung der Objektsichtbarkeit [27]. Verschiedene Lichtquellen, darunter Glühlampen, Leuchtstofflampen, Laser, Röntgenröhren und Infrarotlichter, wurden als wesentlich für die Verbesserung der Bildqualität bei der industriellen Inspektion genannt [38].

Die Herausforderungen, denen Gesichtserkennungssysteme gegenüberstehen, betonen die Notwendigkeit, Variationen in der Beleuchtung, Haltungsänderungen und Gesichtsausdrücke zu berücksichtigen. Diese Faktoren beeinflussen die Modellleistung erheblich und erfordern fortschrittliche Techniken wie tiefe Convolutional Neural Networks, um die Robustheit zu verbessern. Studien haben Methoden wie Multitasking-kaskadierte CNNs und adaptive Vorverarbeitung vorgeschlagen, um die Erkennungsgenauigkeit unter verschiedenen Bedingungen zu verbessern [36].

Verdeckung

Zur Handhabung von Verdeckungen diskutierte die Studie Deep Learning (DL) und Convolutional Neural Networks (CNNs) als kritische Werkzeuge in modernen Computer-Vision-Techniken. Obwohl Verdeckungsprobleme nicht explizit detailliert wurden, verwies die Forschung auf fortgeschrittene KI-Modelle, die visuelle Daten unter verschiedenen Bedingungen verarbeiten und eine zuverlässige Bilderkennung ermöglichen, selbst wenn bestimmte Merkmale teilweise verdeckt sind [38].

In einer anderen Studie untersuchten Forscher die Verdeckungsbehandlung in Haltungsschätzungsmodellen. Sie stellten fest, dass „die Behandlung des Verdeckungsproblems“ eine kritische Herausforderung bei der Haltungsschätzung mehrerer Personen ist und verwiesen auf mehrere Ansätze zur Verbesserung der Leistung unter Bedingungen teilweiser Sichtbarkeit. Die Studie diskutierte Methoden wie Verbesserungen der Schlüsselpunktlokalisierung, Multi-Task-Lernarchitekturen und Gruppierungstechniken zur Assoziation von Schlüsselpunkten, die zu derselben Person gehören, was dazu beiträgt, die Auswirkungen der Verdeckung zu mindern [36].

Echtzeit-Feedback

Hinsichtlich des Echtzeit-Feedbacks untersuchte die Studie die Rolle von Machine Vision (MV)-Systemen, d.h. Systemen, die KI-gesteuerte Bildverarbeitung für die Echtzeit-Visualisierung verwenden, um eine sofortige und automatisierte Inspektion in Hochgeschwindigkeits-Produktionslinien zu gewährleisten. Es wurde festgestellt, dass „MV sich bei der quantitativen Messung einer strukturierten Szene aufgrund ihrer Geschwindigkeit, Genauigkeit und Wiederholbarkeit auszeichnet“, was eine effiziente Qualitätskontrolle in Fertigungsanwendungen ermöglicht [38].

3.1.6 Beispielarchitekturen

Convolutional Neural Networks (CNNs) sind ein entscheidendes KI-Werkzeug für Computer Vision (CV)-Aufgaben, die sich durch Objekterkennung, Klassifizierung und Segmentierung auszeichnen. Ihr Erfolg hängt von großen gelabelten Datensätzen ab, was sie in Anwendungen wie der Verkehrszeichenerkennung, medizinischen Bildgebung, Gesichtserkennung und Objekterkennung hochwirksam macht.

Schlüsselanwendungen in Computer Vision

- **Gesichtserkennung:** CNNs bewältigen Herausforderungen wie Beleuchtungsvariationen und Verdeckungen. Methoden wie tiefe CNNs und Multitask-kaskadierte CNNs haben die Genauigkeit verbessert.
- **Menschliche Haltungsschätzung:** CNN-Architekturen verfeinern Körper-Heatmaps mittels Regressionstechniken.
- **Aktionserkennung:** Traditionell auf Bewegungshistoriebilder und HMMs angewiesen, übertreffen 3D-CNNs in Kombination mit LSTMs nun frühere Ansätze.

Breitere KI-Anwendungen

CNNs erstrecken sich über Computer Vision hinaus in die Bereiche der natürlichen Sprachverarbeitung, Objekterkennung, Spracherkennung, Videoanalyse und mehrdimensionale Dateninterpretation, was ihre Vielseitigkeit in der modernen KI-Forschung demonstriert.

Ein Beispiel für eine Architektur ist YOLO (You Only Look Once), ein weit verbreitetes Objekterkennungsframework, das Bounding Boxes vorhersagt und Objekte in vordefinierte Kategorien klassifiziert. YOLO ist ein Echtzeit-Objekterkennungsmodell, das auf CNNs basiert. Es verarbeitet ein gesamtes Bild auf einmal und identifiziert mehrere Objekte effizient. Standard-YOLO fehlt jedoch die Fähigkeit,

Objektabstände von der Kamera zu schätzen. Um diese Einschränkung zu beheben, hat ein Forscherteam Dist-YOLO verwendet, eine verbesserte Version, die die Abstandsschätzung integriert, während die ursprünglichen YOLO-Erkennungsfähigkeiten beibehalten werden [32].

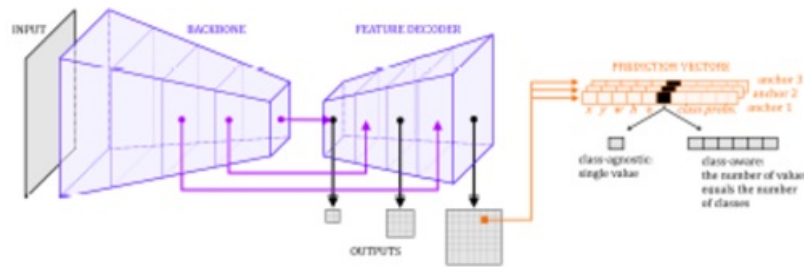


Abbildung 3.3: YOLO-Architektur

Der Ansatz erweitert die von den YOLO-Erkennungsköpfen generierten Vorhersagevektoren um Entfernungsinformationen und integriert eine spezielle Entfernungsverlustfunktion während des Trainings. Experimentelle Ergebnisse zeigen, dass Dist-YOLO nicht nur die ursprüngliche Kapazität des Backbone-Netzwerks beibehält, sondern auch die Genauigkeit der Bounding-Box-Erkennung im Vergleich zum YOLO-Basismodell verbessert. Darüber hinaus bietet Dist-YOLO in Verbindung mit einer monokularen Kamera eine präzise Abstandsschätzung für erkannte Objekte, was seine Anwendbarkeit in realen Szenarien erheblich erweitert [32].

Im Gegensatz zu herkömmlichen zweistufigen Detektoren wie R-CNN formuliert YOLO die Objekterkennung als einstufiges Regressionsproblem um, so dass das Modell die Bounding Boxes und Klassenwahrscheinlichkeiten direkt aus einem Bild in einem Durchgang vorhersagen kann. Diese Designentscheidung führt zu extrem schnellen Schlussfolgerungen, was es ideal für zeitkritische Anwendungen wie Videoüberwachung, autonomes Fahren und Robotik macht [39].

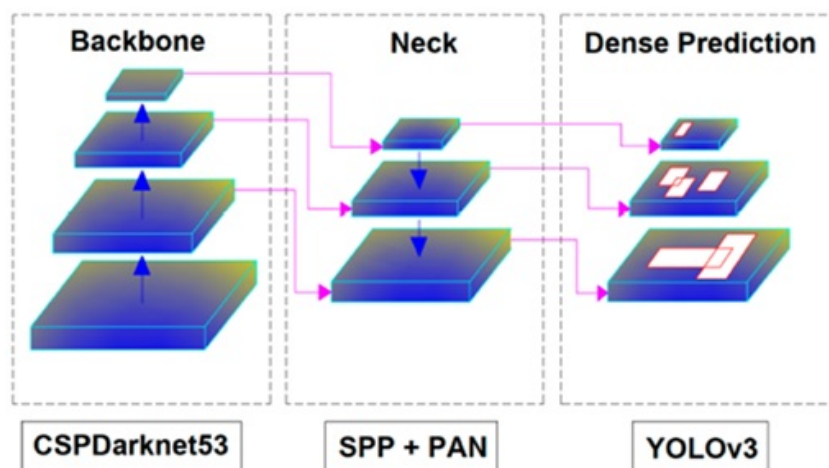


Abbildung 3.4: Netzwerkarchitektur von YOLOv4.

Key Improvements Across YOLO Versions

- **YOLOv1-v3:** Introduced multi-scale feature pyramids (FPN), an optimized Darknet53 backbone, and binary cross-entropy loss for classification.
- **YOLOv4:** Enhanced the framework with a three-component structure (backbone, neck, head) and advanced optimization techniques ("bag of freebies"). Achieved high accuracy (43.5% AP on MS COCO) with real-time performance (65 FPS on Tesla V100).

- **YOLOv4-tiny:** A lightweight version using CSPDarknet53-tiny, optimized for low-resource environments.
- **YOLOv5:** Improved network efficiency, streamlined architecture, and PyTorch-based implementation. It retained YOLOv4's accuracy while enhancing deployment with YOLOv5 offering enhanced accuracy and speed [40].

Eine andere Studie konzentrierte sich auf die Annäherung nichtlinearer Funktionen mit Hilfe fortschrittlicher neuronaler Architekturen, insbesondere residualer neuronaler Netze (ResNet), die für nichtlineare Regression und nicht für bildbasierte Aufgaben angepasst wurden. Der methodische Kern passt die ResNet-Prinzipien an die nichtlineare Regression an, indem er Faltungsschichten durch vollständig verbundene Blöcke ersetzt. Diese Anpassung geht auf die Dimensionsbeschränkungen traditioneller ResNet-Architekturen bei der Verarbeitung von Vektoreingaben anstelle von Bilddaten ein [41].

Einige der wichtigsten Änderungen sind:

1. **Architektonische Anpassung:** Umwandlung von Restblöcken in dichte Schichten unter Beibehaltung der Stapelnormalisierung zur Regularisierung.
2. **Dimensionale Handhabung:** Identitätsblöcke für dimensionserhaltende Flüsse und dichte Blöcke für dimensionsverschiebende Operationen.
3. **Rechnerische Optimierung:** Eliminierung lokaler Faltungsoperatoren zugunsten globaler dichter Verbindungen [42].

Die empirische Validierung nutzt synthetisch generierte Datensätze mit 10^7 Stichproben pro Funktionsklasse, aufgeteilt in:

- Trainingsdaten: 6.75×10^6
- Validierungsdaten: 7.5×10^5
- Testdaten: 2.5×10^6

Die optimale Konfiguration mit 28 Schichten und einer Breite von 16 erreicht eine außergewöhnliche Leistung (Testverluste zwischen 2.55×10^{-5} und 4.21×10^{-4}) und übertrifft deutlich:

- Lineare Methoden ($R^2 < 0.5$ in allen Fällen)
- Klassische nichtlineare Ansätze (z.B. SVR mit Trainingszeiten > 12 Stunden)
- Standard-Neuronale Netze (20–30% höhere Validierungsverluste)

3.1.7 Signale zur Anomalieerkennung

Im Zeitalter des Internet der Dinge (IoT) und der zunehmenden Komplexität von Netzwerken ist die Anomalieerkennung für die Gewährleistung der Datensicherheit und der Zuverlässigkeit von Systemen von entscheidender Bedeutung. Künstliche Intelligenz spielt eine zentrale Rolle bei der Identifizierung ungewöhnlicher Muster, die auf Bedrohungen, Fehler oder Fehlfunktionen hinweisen könnten. Dieser Abschnitt befasst sich mit der Anwendung von KI in der Anomalieerkennung und hebt bestimmte Herausforderungen und Ansätze hervor.

Herausforderungen in der KI-basierten Anomalieerkennung

Trotz der Fortschritte bei KI-basierten Methoden zur Anomalieerkennung bleiben erhebliche Herausforderungen bestehen, insbesondere in komplexen Systemen wie drahtlosen Sensornetzwerken (WSNs) und heterogenen Datensätzen.

- **Hohe Dimension und Datenvolumen:** Moderne Netzwerke generieren riesige Datenmengen mit vielen Merkmalen, was das Training von Anomalieerkennungsmodellen rechenintensiv und speicherintensiv macht.
- **Mangel an gelabelten Anomaliendaten:** Echte Anomalien sind selten und schwer zu labeln, was überwachte Lernansätze behindert. Dies erfordert den Einsatz von unüberwachten oder semi-überwachten Methoden.
- **Konzeptdrift:** Das Verhalten von Anomalien kann sich im Laufe der Zeit ändern, was dazu führt, dass trainierte Modelle veraltet werden und eine kontinuierliche Anpassung erfordern.
- **Rauschen und irrelevante Merkmale:** Daten enthalten oft Rauschen und irrelevante Merkmale, die die Leistung von Anomalieerkennungsalgorithmen mindern.
- **Rechenbeschränkungen am Edge:** In WSNs und IoT-Geräten mit begrenzten Rechenressourcen ist es schwierig, komplexe Anomalieerkennungsalgorithmen direkt am Sensor (Edge Processing) zu implementieren.

Eine 2023 veröffentlichte Studie, die verschiedene Methoden zur Anomalieerkennung in WSNs vergleicht, beleuchtet diese Herausforderungen [43]. Die Forscher kamen zu dem Schluss, dass die Neubewertung bestehender Modelle entscheidend ist, da veröffentlichte Ergebnisse oft aufgrund inkonsistenter Bewertungsmetriken oder Datensätze variieren. Sie fanden heraus, dass einfache Modelle wie Vanilla-Autoencoder (DAE) komplexere Architekturen übertreffen können, wenn sie auf herausfordernden Datensätzen wie CSE-CIC-IDS2018 trainiert werden, die aussagekräftigere Einblicke bieten als weit verbreitete, aber triviale Datensätze wie KDD. Darüber hinaus betonen sie die Notwendigkeit robuster Modelle zur Anomalieerkennung in WSNs angesichts von Bedrohungen wie Hardwarefehlern, Umweltveränderungen oder Cyberangriffen [43].

KI-basierte Anomalieerkennung in drahtlosen Sensornetzwerken (WSNs)

Drahtlose Sensornetzwerke (WSNs) spielen eine entscheidende Rolle im Internet der Dinge (IoT), aber Sensoranomalien – verursacht durch Hardwarefehler, Umweltveränderungen oder Cyberangriffe – bedrohen die Datenzuverlässigkeit. Traditionelle zentralisierte Anomalieerkennung ist aufgrund hoher Datenübertragungskosten ineffizient, während verteilte Methoden oft unter hoher Rechenkomplexität oder übermäßiger Kommunikation zwischen Sensoren leiden.

Autoencoder-basierte Anomalieerkennung für WSNs Ein dreischichtiger Autoencoder komprimiert und rekonstruiert Sensordaten, um Anomalien auf der Grundlage von Rekonstruktionsfehlern zu erkennen, wobei er ohne gelabelte Daten durch unüberwachtes Lernen arbeitet [42].

Verteilter Algorithmus-Ansatz

- **Sensor-Ebene (Edge Processing):** Jeder Sensor erkennt Anomalien lokal durch Vergleich von Eingabe- und rekonstruierten Werten, was keine Kommunikation zwischen Sensoren erfordert.
- **Cloud-Ebene-Verarbeitung:** Aggregiert Residuen und aktualisiert globale Statistiken, wobei der Autoencoder regelmäßig neu trainiert wird, um die Genauigkeit zu erhalten und gleichzeitig den Sensor-Overhead zu reduzieren. [34]

3.2 Algorithmen zur Kantenerkennung und KI-Verbesserungen

Die Kantenerkennung ist eine grundlegende Technik der Bildverarbeitung und des maschinellen Sehens und spielt eine entscheidende Rolle in einer Vielzahl von Anwendungen und Disziplinen. Sie dient häufig

als ein kritischer Vorverarbeitungsschritt bei der Bildsegmentierung, indem sie hilft, Bilder in sinnvolle Bereiche zu unterteilen, indem Objektgrenzen und -strukturen klar abgegrenzt werden. In der Objekterkennung und -klassifikation definiert die Kantenerkennung die Konturen von Objekten, was eine präzise Klassifizierung und weiterführende Analyse unterstützt. Besonders wichtig ist sie in der Robotik und in autonomen Systemen, da sie die visuelle Echtzeitverarbeitung für Hinderniserkennung, Navigation und Umweltinterpretation ermöglicht. In der Biometrie verbessert die Kantenerkennung die Erkennung komplexer Muster wie Fingerabdrücke und Netzhautscans und erhöht so die Genauigkeit der Identifikation. Auch bei der Dokumentenanalyse spielt sie eine Rolle, indem sie die Zeichenerkennung, Formularverarbeitung und Textextraktion erleichtert. Darüber hinaus unterstützt die Kantenerkennung bei der Verarbeitung von Satellitenbildern die Bewertung von Landschaftsformen, die Überwachung städtischer Entwicklungen und die Umweltanalyse. Diese vielfältigen Anwendungen unterstreichen die unverzichtbare Rolle der Kantenerkennung bei der Gewinnung kritischer visueller Informationen und bilden die Grundlage für fortschrittliche Bildanalysetechniken in zahlreichen Bereichen [44].

Die Fähigkeit, Kanten in einem Bild präzise zu identifizieren und zu extrahieren, hat direkten Einfluss auf die Leistungsfähigkeit nachgelagerter Anwendungen. Im Laufe der Jahrzehnte haben sich die Methoden der Kantenerkennung erheblich weiterentwickelt – von einfachen gradientenbasierten Operatoren bis hin zu hochentwickelten Deep-Learning-Modellen. Dieser Abschnitt bietet eine eingehende Untersuchung klassischer Kantenerkennungsalgorithmen, ihrer Anwendungen in präzisen Messtechnologien, ihrer inhärenten Einschränkungen sowie der Art und Weise, wie künstliche Intelligenz dieses Feld durch fortgeschrittene Modelle und hybride Ansätze revolutioniert hat.

Mit dem wachsenden Bedarf an hochpräziser und hocheffizienter Messtechnik in der Fertigung mechanischer Komponenten gewinnen robuste und genaue Kantenerkennungsalgorithmen zunehmend an Bedeutung. In dieser Studie wird eine neuartige Kantenerkennungsmethode auf Subpixel-Niveau basierend auf dem gaußschen Integrationsmodell vorgeschlagen, um diesem Bedarf gerecht zu werden [45].

3.2.1 Grundlagen der Kantenerkennung

Die Kantenerkennung bleibt ein Eckpfeiler des maschinellen Sehens und der Bildverarbeitung und spielt eine zentrale Rolle bei der Extraktion bedeutungsvoller struktureller Informationen aus digitalen Bildern. Durch die Identifikation signifikanter Intensitätsänderungen ermöglicht die Kantenerkennung die Abgrenzung von Objektgrenzen und unterstützt dadurch zentrale Aufgaben wie Objekterkennung, Bildsegmentierung und Merkmalsextraktion [44].

Im Laufe der Jahrzehnte wurden zahlreiche Methoden zur Kantenerkennung vorgeschlagen, die sowohl grundlegende Beiträge als auch aktuelle Fortschritte widerspiegeln. Zu den frühesten Techniken zählt der Kantenerkennungsalgorithmus basierend auf dem Roberts-Operator, der 1963 von Lawrence Roberts eingeführt wurde und einen Meilenstein in der Entwicklung des maschinellen Sehens darstellt [44].

3.2.2 Arten von Kanten

Das Verständnis dieser Kantentypen ist entscheidend, da sie wichtige Hinweise über die Objekte, Materialien und Lichtverhältnisse in einem Bild liefern. Die Erkennung des spezifischen Kantentyps verbessert die Genauigkeit der Interpretation und unterstützt die Entwicklung von Kantenerkennungstechniken, die auf unterschiedliche Kantenmerkmale abgestimmt sind.

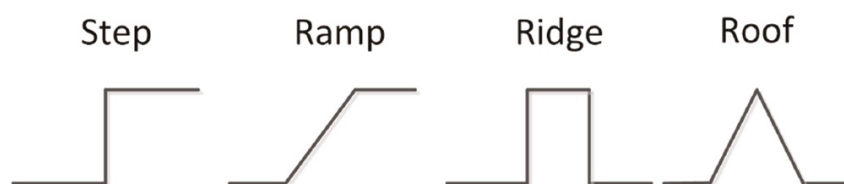


Abbildung 3.5: Kanten Typen

Kanten in Bildern weisen je nach Intensitäts- oder Farbänderung zwischen benachbarten Pixeln unterschiedliche Eigenschaften auf. Diese Unterschiede ermöglichen eine Klassifizierung der Kanten in mehrere unterschiedliche Typen, wie in Abbildung 3.5 dargestellt. Jeder Typ vermittelt einzigartige Informationen über die Struktur und den Inhalt des Bildes [45].

Stufenkanten (Step) Stufenkanten sind die grundlegendste und am häufigsten vorkommende Art von Kanten. Sie zeichnen sich durch einen abrupten und signifikanten Intensitätswechsel zwischen benachbarten Pixeln aus. Diese scharfe Übergänge entsprechen typischerweise Objektgrenzen oder plötzlichen Änderungen in der Oberflächenausrichtung, bei denen die Pixelwerte schnell von einem Helligkeits- oder Farbniveau zu einem anderen springen [46].

Rampenkanten (Ramp) Rampenkanten beschreiben einen allmählicheren Intensitätswechsel, der häufig einer gleichmäßigen, linearen Übergangsform über die Pixel hinweg ähnelt. Solche Kanten entstehen oft durch graduelle Beleuchtungsänderungen oder Materialübergänge, wie etwa beim sanften Wechsel von Schatten- zu beleuchteten Bereichen [46].

Gratkanten (Ridge) Gratkanten treten auf, wenn die Intensität zunächst allmählich abnimmt und anschließend stark ansteigt, wodurch ein gratartiges Profil im Bild entsteht. Diese Kanten sind häufig dort zu beobachten, wo helle Bereiche in Schattenzonen übergehen und so ein charakteristisches Intensitätsmuster erzeugen [46].

Firstkanten (Roof) Firstkanten zeigen eine allmähliche Zunahme der Intensität, gefolgt von einem abrupten Abfall. Sie treten häufig in Bereichen mit Reflexionen oder Übergängen von Schatten zu hellen Lichtreflexen auf [46].

3.2.3 Klassische Kantenerkennungsalgorithmen

Klassische Kantenerkennungsalgorithmen basieren hauptsächlich auf mathematischen Operationen zur Identifikation von Bildbereichen, in denen sich die Pixelintensitäten abrupt verändern. Solche Veränderungen entsprechen typischerweise den Grenzen von Objekten oder markanten Merkmalen innerhalb einer Szene. Die klassischen Ansätze lassen sich grob in Methoden erster und zweiter Ordnung unterteilen, die jeweils spezifische Vor- und Nachteile aufweisen. [47]

Die Kantenerkennung bleibt eine der grundlegendsten Aufgaben des maschinellen Sehens mit breit gefächerten Anwendungen, darunter Bildsegmentierung, Objekterkennung und Videoobjektsegmentierung. Ziel ist es, visuell auffällige Kanten und Objektgrenzen innerhalb eines Bildes präzise zu identifizieren. Trotz ihrer langen Geschichte stellt die Kantenerkennung weiterhin eine große Herausforderung dar – bedingt durch Faktoren wie komplexe Hintergrundszenen, Variationen im Erscheinungsbild von Objekten und Inkonsistenzen in von Menschen annotierten Ground-Truth-Daten. [47]

Klassische Kantenerkennungsalgorithmen haben das grundlegende Fundament für die moderne Computer Vision gelegt, indem sie die Extraktion von Objektgrenzen und Konturen aus digitalen Bildern ermöglichten. Zu den frühesten Methoden zählt der Roberts-Operator, der 1963 von Lawrence Roberts eingeführt wurde und einen richtungsweisenden, wenn auch rudimentären Schritt in der Kantenerkennung darstellte, indem er einfache Gradientenabschätzungen nutzte. Aufbauend darauf wurde später der Prewitt-Operator entwickelt, um besser mit verrauschten Bildern und schwachen Pixelintensitäten umzugehen und somit in kontrastarmen Umgebungen eine verbesserte Leistung zu erzielen. Der Sobel-Operator, eine populäre Weiterentwicklung von Prewitt, führte gewichtete Faltungskerne ein, die stärkere Kantenreaktionen und eine bessere Rauschunterdrückung ermöglichten und dadurch breite Anwendung fanden. Weitere Fortschritte wurden durch den Einsatz des Laplace-Operators erzielt, der auf Ableitungen zweiter Ordnung basierte, um insbesondere in Bereichen mit raschen Intensitätsänderungen eine verbesserte Kantenlokalisierung zu ermöglichen. Ein bedeutender Meilenstein war die Einführung des Canny-Kantendetektors, der die Kantenerkennung durch die Kombination von gaußscher Glättung, Gradientenanalyse, Nicht-Maximum-Unterdrückung und Hysterese-Schwellenwertsetzung in einem umfassenden, mehrstufigen Algorithmus revolutionierte. Canny setzte neue Maßstäbe in Bezug auf Präzision, Rauschrobustheit und Kantenkontinuität und zählt bis heute zu den einflussreichsten Verfahren des

Fachgebiets. Diese klassischen Algorithmen, auch wenn sie inzwischen als traditionell gelten, dienen aufgrund ihrer Einfachheit, Effizienz und Interpretierbarkeit weiterhin als Referenzmethoden und Vorverarbeitungsschritte in zahlreichen Anwendungen der Computer Vision. [48]

Sobel Operator

Der Sobel-Operator verwendet ein Paar von 3×3 -Faltungskernen, um für jedes Pixel die Gradientenstärke und -richtung in einem Bild zu berechnen. Die Gradientenstärke gibt die Intensität der Kante an, während die Gradientenrichtung die Ausrichtung der Kante beschreibt. Der Sobel-Operator ist eine der am weitesten verbreiteten Kantenerkennungstechniken erster Ordnung. Er arbeitet, indem das Bild mit zwei kleinen, separierbaren, ganzzahligen Filtern in horizontaler und vertikaler Richtung gefaltet wird. Diese Filter sind so konzipiert, dass sie den Gradienten der Bildintensitätsfunktion annähern. Der horizontale Kernel betont dabei Kanten in vertikaler Richtung, während der vertikale Kernel horizontale Kanten hervorhebt. Die Gradientenstärke wird anschließend durch Kombination der Ausgaben beider Kerne berechnet, und Kanten werden dort erkannt, wo diese Stärke einen bestimmten Schwellenwert überschreitet. [49]

Eine der zentralen Stärken des Sobel-Operators ist seine rechnerische Effizienz, wodurch er sich gut für Echtzeitanwendungen eignet. Darüber hinaus sorgt die Integration einer gaußschen Glättung innerhalb der Kerne für eine gewisse Rauschunempfindlichkeit. Allerdings erzeugt der Sobel-Operator tendenziell relativ breite Kanten, was problematisch für Anwendungen sein kann, die eine hohe Präzision erfordern. Zudem nimmt seine Leistungsfähigkeit bei starkem Rauschen oder komplexen Texturen deutlich ab, da ihm die Fähigkeit fehlt, höherstufige semantische Informationen zu berücksichtigen. [50]

Laplacian of Gaussian (LoG)

Der Laplacian of Gaussian (LoG) ist eine Kantenerkennungsmethode zweiter Ordnung, die einige der Einschränkungen von Verfahren erster Ordnung adressiert. Der LoG-Operator wendet zunächst eine gaußsche Unschärfe auf das Bild an, um Rauschen zu reduzieren, und berechnet anschließend den Laplace-Operator, also die zweite Ableitung des geglätteten Bildes. Die Nulldurchgänge des Laplace-Ergebnisses entsprechen dabei den Kanten im Bild. [51]

Der Schritt der gaußschen Glättung ist entscheidend, da er hilft, die Rauschempfindlichkeit, die für Ableitungen zweiter Ordnung typisch ist, abzumildern. Durch Variation der Standardabweichung des Gauß-Kernels kann der Operator so angepasst werden, dass Kanten in unterschiedlichen Maßstäben erkannt werden. Diese Multiskalenfähigkeit macht den LoG besonders nützlich für Anwendungen, bei denen Kanten unterschiedlicher Breite erkannt werden müssen. Allerdings ist der LoG-Operator rechnerisch aufwendig, da sowohl die Faltung mit einem großen gaußschen Kernel als auch die Berechnung der zweiten Ableitung notwendig sind. Zudem kann er in Bereichen mit allmählichen Intensitätsübergängen zu fehlerhaften Kanten führen, und seine Leistung hängt stark von der Wahl der Parameter des Gauß-Kernels ab. [52]

Canny Edge Detection

Der Canny-Kantendetektor, entwickelt von John Canny im Jahr 1986, zählt bis heute zu den beliebtesten und effektivsten Kantenerkennungsalgorithmen. Er wurde entwickelt, um drei zentrale Kriterien zu optimieren: gute Erkennung durch Minimierung von Fehlalarmen und Auslassungen, gute Lokalisierung, da die erkannten Kanten möglichst nah an den tatsächlichen Kanten liegen sollen, sowie minimale Reaktion, sodass pro Kante nur eine Antwort erfolgt. Der Canny-Operator ist ein hochentwickelter Kantenerkennungsalgorithmus, der eine präzise Kantenlokalisierung bei gleichzeitig geringer Rauschanfälligkeit gewährleistet. Er arbeitet in einer Reihe klar definierter Schritte:

1. **Gaußsche Glättung:** Das Bild wird zunächst mit einem Gauß-Filter geglättet, um das Rauschen zu reduzieren. [53]

2. **Berechnung des Gradienten:** Der Sobel-Operator (oder ein ähnliches Verfahren) wird verwendet, um die Größe und Richtung des Gradienten für jedes Pixel zu berechnen. [53]
3. **Nicht-Maximum-Unterdrückung:** In diesem Schritt werden die Kanten ausgedünnt, indem nur die lokalen Maxima in Gradientenrichtung beibehalten werden. [53]
4. **Hysteresese-Schwellenwertbildung:** Die Kanten werden mit zwei Schwellenwerten (hoch und niedrig) abgeschlossen, um starke Kanten von schwachen zu unterscheiden. Schwache Kanten werden nur dann beibehalten, wenn sie mit starken Kanten verbunden sind, wodurch die Fragmentierung verringert wird. [51]

Die Robustheit des Canny-Detektors gegenüber Rauschen und seine Fähigkeit, dünne, gut lokalisierte Kanten zu erzeugen, machen ihn für eine breite Palette von Anwendungen geeignet. Seine Leistung hängt jedoch in hohem Maße von der Wahl der Schwellenwerte ab, die häufig manuell eingestellt werden müssen. Da der Canny-Operator mehrere Stufen mit verschiedenen Parametern umfasst und seine Effektivität eher in diesem systematischen Ansatz als in einer einzigen Definitionsgleichung liegt, kann er bei strukturierten Hintergründen oder Bildern mit geringem Kontrast Probleme bereiten.

3.2.4 Anwendungen in der Hochpräzisionsmesstechnik

Kantenerkennungsalgorithmen spielen eine zentrale Rolle in hochpräzisen Messtechnologien, bei denen die genaue und zuverlässige Extraktion von Objektgrenzen unerlässlich ist. Diese Anwendungen erfordern häufig Subpixel-Genauigkeit und Robustheit gegenüber unterschiedlichen Umweltbedingungen. [46]

Die Kantenerkennung bleibt eine grundlegende Aufgabe der Bildverarbeitung, die die Identifikation von Objektgrenzen und relevanten Bildbereichen ermöglicht. Im Laufe der Jahre hat sich ihre Entwicklung erheblich weiterentwickelt – von klassischen Methoden wie Sobel, Prewitt und Canny bis hin zu fortgeschrittenen, KI-gestützten Verfahren. Forschende arbeiten kontinuierlich an der Verbesserung dieser Algorithmen, um domänenspezifische Herausforderungen in Bereichen wie der medizinischen Bildgebung, Unterwassersicht und Robotik zu bewältigen. [46]

Das Aufkommen von Convolutional Neural Networks (CNNs) hat die Kantenerkennung revolutioniert, da sie die Extraktion semantisch reichhaltiger Merkmale ermöglichen. Mit zunehmender Tiefe der CNN-Architekturen und wachsender Rezeptivfeldgröße erfassen sie schrittweise globale Strukturen und objektbezogene Informationen. Dies hat zu erheblichen Verbesserungen der Erkennungsgenauigkeit geführt. Ein wesentlicher Nachteil dieser tiefen Netzwerke besteht jedoch im Verlust feingranularer Details, insbesondere in den frühen Schichten, da die Merkmalskarten abstrakter werden und die räumliche Auflösung abnimmt. [45]

Um dieser Einschränkung entgegenzuwirken, wurden in jüngerer Zeit Methoden mit Multi-Level-Feature-Aggregation entwickelt, bei denen tiefe semantische Merkmale mit flachen, hochaufgelösten Merkmalen kombiniert werden. Obwohl diese Fusion dazu beiträgt, feinere strukturelle Informationen wiederherzustellen, bringt sie auch eine neue Herausforderung mit sich: Flache Merkmale erfassen meist nur lokale Intensitätsvariationen, ohne dabei sinnvolle semantische Kontexte zu berücksichtigen, was häufig zur Erkennung von falschen oder verrauschten Kanten führt.

Zur Überbrückung dieser Lücke wurden fortgeschrittene Modelle wie CNNs oder andere neue Architekturen vorgeschlagen. Sie könnten die Kantenerkennungsleistung verbessern, indem sie globalen Kontext und lokale Details effektiv integrieren. Dabei wird eine Dual-Cue-Architektur genutzt, um sicherzustellen, dass die erkannten Kanten sowohl semantisch bedeutsam als auch räumlich präzise sind. [45]

Berührungslose dimensionale Inspektion

In der industriellen Fertigung verlassen sich berührungslose dimensionale Prüfsysteme auf die Kantenerkennung, um die physischen Abmessungen von Objekten ohne physischen Kontakt zu messen. Die-

se Systeme sind entscheidend für die Sicherstellung von Produktqualität und -konsistenz, insbesondere in Branchen, in denen Präzision von höchster Bedeutung ist, wie etwa der Automobil-, Luft- und Raumfahrt-, Medizintechnik- und Elektronikindustrie. [47]

Beispielsweise müssen in der Automobil- oder Luftfahrtindustrie die Maße von bearbeiteten Bauteilen überprüft werden, um sicherzustellen, dass sie strengen Toleranzen entsprechen. Selbst geringfügige Abweichungen können zu Leistungsproblemen oder Sicherheitsrisiken führen, weshalb eine hochpräzise Inspektion unerlässlich ist. Klassische Methoden wie der Canny-Detektor werden aufgrund ihrer Genauigkeit häufig eingesetzt, können jedoch durch KI-basierte Verfahren ergänzt werden, um komplexe Geometrien oder reflektierende Oberflächen zu bewältigen. KI-gestützte Ansätze, wie auf Deep Learning basierende Kantenerkennung und durch neuronale Netze verbesserte Bildverarbeitung, ermöglichen eine adaptive Inspektion, die sich an Schwankungen in Beleuchtung, Materialeigenschaften und Oberflächenstrukturen anpassen kann. [47]

Darüber hinaus wurden fortschrittliche optische Systeme, einschließlich strukturiertem Lichtscanning und digitaler Holographie, in berührungslose Inspektionsprozesse integriert, um Submillimeter-Genauigkeit zu erreichen. Diese Techniken ermöglichen hochgeschwindige, echtzeitfähige Messungen und sind damit ideal für die Inline-Qualitätskontrolle in Produktionsumgebungen mit hohem Durchsatz. Der gPb-Algorithmus (global probability of boundary), der lokale und globale Hinweise kombiniert, hat sich in solchen Anwendungen als vielversprechend erwiesen, da er genauere und kontinuierlichere Kanten liefert, insbesondere bei Objekten mit komplexen oder unregelmäßigen Formen. [47]

Moderne berührungslose Inspektionssysteme integrieren zudem häufig Multi-Sensor-Fusion, bei der Daten von hochauflösenden Kameras, Lasertriangulation und Interferometrie kombiniert werden, um die Messzuverlässigkeit zu erhöhen. Dieser Ansatz minimiert Fehler, die durch Umwelteinflüsse entstehen, und verbessert die Robustheit bei der Erkennung subtiler Maßabweichungen. In Branchen wie der Herstellung medizinischer Geräte, in denen Präzision entscheidend für Funktionalität und Patientensicherheit ist, spielen diese Technologien eine zentrale Rolle bei der Einhaltung strenger Qualitätsstandards. [50]

Durch den Einsatz von KI, fortschrittlicher Optik und Multi-Sensor-Integration entwickelt sich die berührungslose dimensionale Prüfung stetig weiter und bietet Herstellern ein leistungsfähiges Werkzeug für hochpräzise, nichtinvasive Messungen bei gleichzeitiger Effizienzsteigerung und Reduktion von Produktionsfehlern. [50]

Mikro- und Nanostrukturen Detektion

Die Halbleiterindustrie erfordert die Erkennung äußerst feiner Kanten und Muster auf Siliziumwafern, oftmals im Nanometerbereich. Da Halbleiterbauelemente weiterhin schrumpfen, ist die präzise Erkennung von Strukturen entscheidend für die Aufrechterhaltung von Leistung und Zuverlässigkeit. Traditionelle Kantenerkennungsmethoden stoßen hierbei an ihre Grenzen, da sie aufgrund von Rauschen, geringem Kontrast oder Materialeigenschaften solche winzigen Merkmale nicht zuverlässig erfassen können. [52]

Tiefenlern-basierte Ansätze wie HED (Holistically-Nested Edge Detection) oder RCF (Richer Convolutional Features) werden zunehmend eingesetzt, da sie multiskalige Merkmale und kontextuelle Informationen nutzen können, um subtile Kanten zu erkennen. Diese Methoden werden auf hochauflösenden Bildern mit präzisen Annotationen trainiert, um das erforderliche Detailniveau zu erreichen. Zudem wird hyperspektrale Bildgebung in Kombination mit Deep Learning erforscht, um Defekte in Nanostrukturen mit hohem Aspektverhältnis zu erkennen und so die Genauigkeit bei der Identifizierung struktureller Anomalien zu verbessern. [52]

Neben der Kantenerkennung spielen fortschrittliche Messtechniken wie spektroskopische Ellipsometrie, Raman-Spektroskopie und Elektronenmikroskopie eine entscheidende Rolle bei der Charakterisierung von Halbleiter-Nanostrukturen. Diese Verfahren liefern Erkenntnisse über Materialzusammensetzung, Schichtdicke und Defektverteilung und ergänzen die KI-gestützte Bildanalyse. Die optische Scatterometrie, bei der durch Analyse der Fernfeld-Streuung optischer Signale Profilparameter von Nanostrukturen abgeleitet werden, hat sich ebenfalls als wertvolles Werkzeug für zerstörungsfreie Inspektionen etabliert. [50]

Darüber hinaus wurde die strukturlichtgestützte kohärente Fourier-Scatterometrie entwickelt, um die physikalische Parameterbestimmung von Nanostrukturen zu verbessern und eine Inline-Überwachung während der Halbleiterfertigung zu ermöglichen. Diese Technik erlaubt es den Herstellern, eine strenge Qualitätskontrolle aufrechtzuerhalten und gleichzeitig die Fertigungsprozesse zu optimieren. [50]

Durch die Integration von KI-basierter Kantenerkennung, hyperspektraler Bildgebung und fortschrittlichen Messtechniken gelingt es der Halbleiterindustrie, die Grenzen der Präzision bei der Erkennung von Mikro- und Nanostrukturen weiter zu verschieben und so die Zuverlässigkeit und Effizienz der nächsten Generation elektronischer Geräte sicherzustellen. [50]

Mikro- und Nanostrukturen Detektion

Die Halbleiterindustrie erfordert die Erkennung äußerst feiner Kanten und Muster auf Siliziumwafern, oftmals im Nanometerbereich. Da Halbleiterbauelemente weiterhin schrumpfen, ist die präzise Erkennung von Strukturen entscheidend für die Aufrechterhaltung von Leistung und Zuverlässigkeit. Traditionelle Kantenerkennungsmethoden stoßen hierbei an ihre Grenzen, da sie aufgrund von Rauschen, geringem Kontrast oder Materialeigenschaften solche winzigen Merkmale nicht zuverlässig erfassen können. [52]

Tiefenlern-basierte Ansätze wie HED (Holistically-Nested Edge Detection) oder RCF (Richer Convolutional Features) werden zunehmend eingesetzt, da sie multiskalige Merkmale und kontextuelle Informationen nutzen können, um subtile Kanten zu erkennen. Diese Methoden werden auf hochauflösenden Bildern mit präzisen Annotationen trainiert, um das erforderliche Detailniveau zu erreichen. Zudem wird hyperspektrale Bildgebung in Kombination mit Deep Learning erforscht, um Defekte in Nanostrukturen mit hohem Aspektverhältnis zu erkennen und so die Genauigkeit bei der Identifizierung struktureller Anomalien zu verbessern. [52]

Neben der Kantenerkennung spielen fortschrittliche Messtechniken wie spektroskopische Ellipsometrie, Raman-Spektroskopie und Elektronenmikroskopie eine entscheidende Rolle bei der Charakterisierung von Halbleiter-Nanostrukturen. Diese Verfahren liefern Erkenntnisse über Materialzusammensetzung, Schichtdicke und Defektverteilung und ergänzen die KI-gestützte Bildanalyse. Die optische Scatterometrie, bei der durch Analyse der Fernfeld-Streuung optischer Signale Profilparameter von Nanostrukturen abgeleitet werden, hat sich ebenfalls als wertvolles Werkzeug für zerstörungsfreie Inspektionen etabliert.

Darüber hinaus wurde die strukturlichtgestützte kohärente Fourier-Scatterometrie entwickelt, um die physikalische Parameterbestimmung von Nanostrukturen zu verbessern und eine Inline-Überwachung während der Halbleiterfertigung zu ermöglichen. Diese Technik erlaubt es den Herstellern, eine strenge Qualitätskontrolle aufrechtzuerhalten und gleichzeitig die Fertigungsprozesse zu optimieren. [50]

Durch die Integration von KI-basierter Kantenerkennung, hyperspektraler Bildgebung und fortschrittlichen Messtechniken gelingt es der Halbleiterindustrie, die Grenzen der Präzision bei der Erkennung von Mikro- und Nanostrukturen weiter zu verschieben und so die Zuverlässigkeit und Effizienz der nächsten Generation elektronischer Geräte sicherzustellen.

Geometrievalidierung in der industriellen Qualitätskontrolle

In der Qualitätskontrolle wird die Kantenerkennung eingesetzt, um die Geometrie hergestellter Komponenten mit ihren Konstruktionsvorgaben abzugleichen. Dieser Prozess ist in verschiedenen Branchen unerlässlich, darunter Automobilbau, Luft- und Raumfahrt, medizinische Bildgebung und Elektronik, in denen Präzision entscheidend für Funktionalität und Sicherheit ist. [50]

Beispielsweise müssen in der medizinischen Bildgebung die Konturen von Implantaten oder Prothesen exakt mit ihren vorgesehenen Designs übereinstimmen. Selbst geringfügige Abweichungen können zu Komplikationen beim Patienten führen, weshalb eine Validierung mit hoher Genauigkeit unverzichtbar ist. Strukturierte Kantenerkennungsverfahren wie SE (Structured Edges) nutzen maschinelles Lernen, um Kanten auf Basis lokaler Bildausschnitte vorherzusagen und bieten dadurch Robustheit gegenüber

Lichtveränderungen und Verdeckungen. Solche Verfahren werden häufig in automatisierte Inspektionssysteme integriert, um Konsistenz sicherzustellen und menschliche Fehler zu reduzieren. [50]

Über die strukturierte Kantenerkennung hinaus werden fortgeschrittene Techniken wie Deep-Learning-basierte Segmentierung und adaptive Schwellenwertverfahren eingesetzt, um die Genauigkeit bei komplexen Geometrien zu erhöhen. KI-gesteuerte Ansätze ermöglichen Echtzeitanpassungen an Umweltfaktoren und verbessern die Zuverlässigkeit unter dynamischen Fertigungsbedingungen. Darüber hinaus bieten optische Messtechniken wie Laserprofilometrie und strukturierte Lichtabtastung hochauflösende geometrische Validierung und ermöglichen eine Präzision im Submikrometerbereich bei der Bauteilprüfung. [47]

Ein weiterer wichtiger Fortschritt in der industriellen Qualitätskontrolle ist die Multisensorfusion, bei der Daten von hochauflösenden Kameras, Laserdreiecksvermessung und Interferometrie kombiniert werden, um die Messzuverlässigkeit zu verbessern. Dieser Ansatz minimiert Fehler, die durch Materialunregelmäßigkeiten entstehen, und erhöht die Robustheit bei der Erkennung feiner dimensionsbezogener Abweichungen. In Branchen wie der Halbleiterfertigung, in denen Präzision im Nanometerbereich erforderlich ist, spielen diese Technologien eine entscheidende Rolle bei der Einhaltung strenger Qualitätsstandards. [47]

3.2.5 Beschränkungen der klassischen Kantenerkennungsmethoden

Trotz ihrer weitverbreiteten Anwendung leiden klassische Kantenerkennungsverfahren unter mehreren inhärenten Einschränkungen, die ihre Leistungsfähigkeit in realen Szenarien beeinträchtigen können. Diese Techniken, obwohl sie eine Grundlage der Bildverarbeitung darstellen, haben häufig Schwierigkeiten mit Genauigkeit und Zuverlässigkeit aufgrund verschiedener Einflussfaktoren.

Ein wesentliches Problem ist das Bildrauschen, das zur Fragmentierung von Kanten führen kann, wobei zusammenhängende Kanten in unterbrochene Segmente zerfallen. Dieses Problem tritt besonders bei Bildern geringer Qualität oder hoher Kompression auf, bei denen das Rauschniveau hoch ist. Klassische Verfahren wie die Sobel-, Prewitt- und Canny-Detektoren basieren auf gradientenbasierten Ansätzen und sind daher anfällig für Rauschstörungen. Zwar kann eine Glättung mittels Gauß-Filter einige dieser Effekte mildern, jedoch kann sie auch feine Details verwischen und die Kantenschärfe reduzieren. [44]

Beleuchtungsunterschiede erschweren die Kantenerkennung zusätzlich, da sich durch veränderte Lichtverhältnisse das Erscheinungsbild von Kanten verzerren und die Konsistenz der Erkennung verringern kann. Klassische Methoden gehen häufig von gleichmäßiger Beleuchtung aus und sind daher weniger effektiv in Umgebungen mit Schatten, Reflexionen oder ungleichmäßiger Helligkeit. Adaptive Schwellenwertverfahren wurden eingeführt, um dieses Problem zu adressieren, erfordern jedoch eine sorgfältige Abstimmung, um die Robustheit in unterschiedlichen Szenarien aufrechtzuerhalten.

Eine weitere Einschränkung besteht in der Schwierigkeit, Kanten in komplexen oder texturierten Szenen präzise zu lokalisieren. Klassische Kantendetektoren verlassen sich primär auf Intensitätsgradienten, die möglicherweise nicht ausreichen, um Kanten in Bildern mit komplizierten Mustern oder überlappenden Texturen zu erkennen. Dies kann zu Fehlalarmen oder dem Übersehen kritischer Kanten führen, was sich negativ auf nachgelagerte Anwendungen wie Objekterkennung und Segmentierung auswirkt. [46]

Darüber hinaus erfordern klassische Methoden in der Regel eine sorgfältige Feinabstimmung der Parameter, um Sensitivität und Spezifität ins Gleichgewicht zu bringen. Die Wahl von Schwellenwerten, Kernelgrößen und Filtereinstellungen beeinflusst die Leistung erheblich, und die Ermittlung optimaler Konfigurationen kann zeitaufwendig und abhängig vom jeweiligen Anwendungsfall sein. Diese mangelnde Anpassungsfähigkeit macht klassische Kantenerkennung weniger geeignet für dynamische Umgebungen, in denen sich die Bedingungen häufig ändern.

Zur Überwindung dieser Einschränkungen integrieren moderne Ansätze Verfahren des maschinellen Lernens und Deep Learning, die kontextuelle Informationen und mehrskalige Merkmale nutzen, um die Genauigkeit der Kantenerkennung zu verbessern. Methoden wie Holistically-Nested Edge Detection (HED) und Richer Convolutional Features (RCF) haben durch das Erlernen von Kantenstrukturen aus großen Datensätzen eine überlegene Leistung gezeigt und verringern die Abhängigkeit von manueller Parametereinstellung. [46]

Lärmempfindlichkeit

Klassische Operatoren wie Sobel oder Prewitt sind äußerst empfindlich gegenüber Rauschen, was zu fragmentierten oder irreführenden Kanten führen kann. Diese Empfindlichkeit entsteht dadurch, dass diese Operatoren auf gradientenbasierten Methoden beruhen, welche Intensitätsänderungen verstärken und dadurch Rauschen als falsche Kanten interpretieren können.

Ein Beispiel hierfür ist die medizinische Bildgebung, bei der Rauschen von Bildsensoren falsche Kanten erzeugen kann, die echte anatomische Grenzen überdecken. Dieses Problem ist insbesondere bei Anwendungen wie der Tumorerkennung oder der Gefäßbildgebung kritisch, da eine präzise Lokalisierung der Kanten für eine genaue Diagnose unerlässlich ist. Auch in der industriellen Inspektion können Rauschartefakte zu fehlerhaften Messungen führen, was sich negativ auf Qualitätskontrollprozesse auswirkt. [49]

Zwar kann Gaußsche Glättung dieses Problem abschwächen, jedoch kann sie auch echte Kanten verwischen und dadurch die Lokalisierungsgenauigkeit verringern. Um diesen Zielkonflikt zu lösen, wurden adaptive Filtertechniken wie anisotrope Diffusion und bilaterale Filter eingeführt, die Rauschen selektiv glätten, während sie Kantendetails bewahren. Darüber hinaus haben sich wavelet-basierte Entrauschungsmethoden als vielversprechend erwiesen, da sie Bilder in mehrskalige Darstellungen zerlegen und eine gezielte Rauschunterdrückung ermöglichen. [46]

Jüngste Fortschritte im Deep Learning haben die Rauschresistenz in der Kantenerkennung weiter verbessert. Auf großen Datensätzen trainierte Convolutional Neural Networks (CNNs) können lernen, zwischen echten Kanten und Rauschen zu unterscheiden, was die Robustheit in anspruchsvollen Umgebungen deutlich erhöht. Hybride Ansätze, die klassische Kantenerkennung mit KI-gestützter Rauschunterdrückung kombinieren, finden zunehmend Anwendung in Bereichen wie medizinischer Bildgebung, Fernerkundung und automatisierter Inspektion.

Durch die Integration von adaptiven Filtertechniken, wavelet-basierter Entrauschung und KI-basierten Verfahren entwickeln sich moderne Kantenerkennungssysteme stetig weiter und bieten verbesserte Genauigkeit und Zuverlässigkeit unter rauschbehafteten Bedingungen. [46]

Einfluss der Lichtverhältnisse

Beleuchtungsvariationen stellen eine erhebliche Herausforderung für die Kantenerkennung dar. Schatten, Reflexionen oder ungleichmäßige Ausleuchtung können Intensitätsgradienten erzeugen, die fälschlicherweise als Kanten interpretiert werden. Im Bereich des autonomen Fahrens beispielsweise können Schatten, die von Bäumen oder Gebäuden geworfen werden, zu fehlerhaften Kantendetektionen führen und somit die Hinderniserkennung beeinträchtigen. Verfahren wie XDoG (eXtended Difference of Gaussians) versuchen, dieses Problem zu lösen, indem sie die Schwellenwerte dynamisch an den lokalen Kontrast anpassen. Dennoch stoßen sie unter extremen Bedingungen weiterhin an ihre Grenzen. [49]

Die Difference-of-Gaussians-Methode (DoG) ist ein grundlegendes Verfahren in der Kantenerkennung, das Kanten hervorhebt, indem zwei unterschiedlich stark geglättete Versionen eines Bildes voneinander subtrahiert werden. Dieser Vorgang wirkt wie ein Bandpassfilter, der spezifische räumliche Frequenzinformationen – insbesondere dort, wo sich die Intensität rasch ändert – betont, während Rauschen und irrelevante Details herausgefiltert werden. Trotz ihrer Einfachheit dient die DoG-Methode als Basis für zahlreiche weiterentwickelte Algorithmen. [53]

Im Jahr 2007 stellten Kang et al. die FDoG-Methode (Flow-based Difference of Gaussians) vor, um die Einschränkungen des klassischen DoG zu überwinden, das isotrope Filterung verwendet und nicht richtungssensitiv ist. FDoG nutzt richtungsabhängige Filterung durch Berücksichtigung der lokalen Kantensorientierung. Es wird ein sogenannter „Boundary Tangent Flow“ aus dem Eingabebild konstruiert, und die Differenzbildung der Gauss-Filter erfolgt gezielt in der Normalrichtung des Kantenverlaufs. Dieser selektive Ansatz ermöglicht es FDoG, glattere und kontinuierlichere Kanten zu erkennen, während gleichzeitig Rauschen reduziert und Fehlklassifikationen vermieden werden – insbesondere in Bereichen, in denen herkömmliche manuelle Merkmalsdefinitionen an ihre Grenzen stoßen. [53]

Eine weitere Entwicklung stellte im Jahr 2011 XDoG (eXtended DoG) von Winnemöller dar, das

auf der Fähigkeit von DoG aufbaut, stilisierte und saubere Konturen zu erzeugen. XDoG führte eine kontinuierliche Schwellwertfunktion sowie einen zusätzlichen Parameter zur Steuerung der Grenzintensität bei der Gauss-Filterung ein. Das Ergebnis ist eine gewichtete Kombination aus der ursprünglichen Gauß-Glättung und der DoG-Ausgabe, die wahrnehmungsrelevante Kantendetails betont und künstlerisch ansprechende Darstellungen ermöglicht. Im Gegensatz zu klassischen Kantenerkennungsverfahren eignet sich XDoG besonders für nicht-photorealistische Darstellungen und erzeugt ästhetisch ansprechende Kantenbilder mit minimalem Nachbearbeitungsaufwand.

Fehler bei der Kantenlokalisierung

Klassische Methoden erzeugen häufig Kanten, die mehrere Pixel breit sind, was die exakte Lokalisierung von Objektgrenzen erschwert. Diese Einschränkung ergibt sich daraus, dass traditionelle Kantenerkennungsverfahren auf gradientenbasierten Ansätzen beruhen, die von Natur aus eine gewisse Unschärfe in der Kantenposition mit sich bringen. In Anwendungen, die höchste Präzision erfordern – wie robotergestützte Chirurgie, Halbleiterfertigung oder hochauflösende Mikroskopie – können selbst geringfügige Lokalisierungsfehler gravierende Folgen haben. [47]

In der robotergestützten Chirurgie ist eine Subpixel-Präzision entscheidend, um eine genaue Positionierung der Instrumente zu gewährleisten und Gewebeschäden zu minimieren. Lokalisierungsfehler können die Genauigkeit von Navigationssystemen beeinträchtigen, was zu unbeabsichtigten Einschnitten oder einer Fehljustierung chirurgischer Werkzeuge führen kann. Zwar hilft die Nicht-Maximum-Unterdrückung im Canny-Detektor dabei, Kanten zu verschmälern, doch hängt das Endergebnis weiterhin von der anfänglichen Gradientenberechnung ab, die möglicherweise nicht präzise genug für sicherheitskritische Anwendungen ist. Zusätzlich können Beleuchtungsunterschiede, Texturen oder Materialeigenschaften die Genauigkeit der Kantenlokalisierung weiter verschlechtern.

Zur Überwindung dieser Herausforderungen wurden fortgeschrittene Verfahren wie aktive Konturmodelle (Snakes) und graphbasierte Segmentierung eingesetzt, um die Kantenlokalisierung zu verfeinern. Diese Methoden berücksichtigen kontextuelle Informationen und ermöglichen eine adaptive Verarbeitung zur Verbesserung der Grenzgenauigkeit. Darüber hinaus nutzen Deep-Learning-basierte Verfahren wie Holistically-Nested Edge Detection (HED) und Richer Convolutional Features (RCF) eine mehrskalige Merkmalsextraktion, um die Präzision gegenüber klassischen Methoden deutlich zu steigern.

In hochpräzisen industriellen Anwendungen wurden Verfahren wie Super-Resolution-Bildgebung und Subpixel-Interpolation in Kantenerkennungs-Workflows integriert, um eine feinere Lokalisierung zu ermöglichen. Diese Ansätze verbessern die Grenzdefinition, reduzieren die Messunsicherheit und steigern die Gesamtexaktheit. [47]

3.2.6 Herausforderungen bei der Kalibrierung

Die meisten klassischen Methoden erfordern eine manuelle Parametereinstellung, die zeitaufwändig und stark anwendungsspezifisch sein kann. Beispielsweise können die optimalen Schwellenwerte für den Canny-Detektor zwischen einer gut beleuchteten industriellen Umgebung und einer schwach beleuchteten Außenaufnahme erheblich variieren. Diese mangelnde Anpassungsfähigkeit begrenzt die Skalierbarkeit dieser Verfahren. [49]

Klassische Kantenerkennungstechniken, obwohl weit verbreitet, weisen mehrere bedeutende Einschränkungen auf, die ihre Wirksamkeit beeinträchtigen. Ein zentrales Problem ist die Kantenfragmentierung, verursacht durch Bildrauschen, welches durchgehende Kanten in getrennte Fragmente aufbricht und so eine weiterführende Analyse erschwert. Auch Variationen in der Beleuchtung stellen Herausforderungen dar, da Lichtänderungen das Kantenbild verzerren und die Detektionsgenauigkeit verringern können. Die präzise Lokalisierung von Kanten bleibt insbesondere bei Bildern mit komplexen Texturen oder subtilen Gradienten schwierig.

Darüber hinaus erfordern diese Methoden oft eine Feinabstimmung mehrerer Parameter wie Schwellenwerte und Filtergrößen, um die Leistung für spezifische Bilder oder Bedingungen zu optimieren – ein

Prozess, der sowohl zeitintensiv als auch anspruchsvoll sein kann. Kalibrierungsprobleme verschärfen die Situation zusätzlich, da Inkonsistenzen in Bildaufnahmegeräten oder Umgebungsfaktoren zu Ungenauigkeiten bei der Kantenlokalisierung und -messung führen können, insbesondere in Anwendungen mit hohen Präzisionsanforderungen.

Die Überwindung dieser Einschränkungen ist entscheidend, um die Robustheit und Zuverlässigkeit der Kantenerkennung zu verbessern. [49]

3.2.7 AI-gestützte Verbesserungen bei der Kantenerkennung

Der Aufstieg des Deep Learnings hat bahnbrechende Fortschritte in der Kantenerkennung ermöglicht, indem Modelle komplexe Merkmale erlernen und sich an vielfältige Szenarien anpassen können. Traditionelle Kantenerkennungsverfahren wie Sobel und Canny basieren auf vordefinierten Filtern und gradientenbasierten Ansätzen, die bei Rauschen, Beleuchtungsvariationen und komplexen Texturen an ihre Grenzen stoßen können. Im Gegensatz dazu nutzen Deep-Learning-Techniken große annotierte Datensätze, wie BSDS500, um hierarchische Repräsentationen von Kanten zu erlernen, was die Genauigkeit und Robustheit signifikant verbessert. [52]

Convolutional Neural Networks (CNNs) stehen im Zentrum dieses Fortschritts und zeigen aufgrund ihrer Fähigkeit, semantische und räumliche Merkmale hierarchisch zu extrahieren, starke Leistungen in visuellen Aufgaben. Im Unterschied zu klassischen Methoden können CNN-basierte Modelle sich an unterschiedliche visuelle Kontexte anpassen, was sie besonders effektiv in Anwendungen wie der medizinischen Bildgebung, autonomen Navigation und industrieller Inspektion macht. Neuere Deep-Learning-Methoden wie EDTER (Edge Detection with Transformer) und BiMLA (Bidirectional Multi-Level Attention) haben die Kantenlokalisierung weiter verbessert, indem sie mehrstufige Merkmale integrieren und Rauschinterferenzen minimieren. [45]

Über die Standard-Deep-Netzwerke hinaus wurden Optimierungsalgorithmen mit der Kantenerkennung kombiniert, um zusätzliche Verbesserungen zu erzielen. Einige Forscher haben hybride Algorithmen entwickelt, die geführte Bildfilterung mit Ameisen-Kolonien-Optimierung verbinden, wodurch glattere Kantenkarten und bessere Rauschunterdrückung im Vergleich zu herkömmlichen Ansätzen erreicht werden. Zudem haben sich selbstüberwachte Lernverfahren etabliert, die es Modellen ermöglichen, die Kantenerkennung ohne umfangreiche beschriftete Datensätze zu verfeinern. [46]

Transformer-basierte Architekturen gewinnen ebenfalls an Bedeutung in der Kantenerkennung, da sie besonders gut darin sind, Langzeitabhängigkeiten und globalen Kontext zu modellieren. Durch die Verarbeitung fein abgestufter Bildausschnitte und den Einsatz eines Merkmalsfusionsmoduls kombinieren diese Modelle lokaltexturale und globale semantische Informationen effektiv, was zu schärferen und präziseren Kantenkarten führt. Darüber hinaus ermöglichen mehrskalige Merkmalsextraktionstechniken Deep-Learning-Modellen, Kanten in verschiedenen Auflösungen zu erfassen und so die Leistung in komplexen Szenen zu verbessern.

Mit der fortschreitenden Entwicklung der KI wird erwartet, dass Kantenerkennungsmethoden noch präziser, anpassungsfähiger und effizienter werden, was Fortschritte in Bereichen wie der biomedizinischen Bildgebung, Robotik und hochauflösender Satellitenanalyse ermöglicht. [46]

Convolutional Neural Networks (CNNs) für die Kantenerkennung

Convolutional Neural Networks (CNNs) sind zur Grundlage moderner Kantenerkennung geworden, da sie hierarchische Merkmale direkt aus Daten lernen können. Im Gegensatz zu klassischen Kantenerkennungsmethoden, die auf vordefinierten Filtern und gradientenbasierten Verfahren basieren, ermöglichen CNNs die automatische Extraktion bedeutungsvoller Merkmale aus Bildern und sind somit hochgradig anpassungsfähig an unterschiedliche visuelle Kontexte. [45]

Bahnbrechende Arbeiten wie Holistically-Nested Edge Detection (HED) führten ein vollständig end-to-end trainierbares Netzwerk ein, das Kanten direkt aus Rohbildern vorhersagt und dabei multiskalige Merkmale mittels Deep Supervision nutzt. Dieser Ansatz verbesserte die Kantenerkennungsgenauigkeit

erheblich, indem tiefere Schichten Kantenvorhersagen verfeinerten und gleichzeitig feine Details erhalten blieben. [45]

Nachfolgende Modelle wie Richer Convolutional Features (RCF) und das Bi-Directional Cascade Network (BDCN) steigerten die Leistung weiter durch die Aggregation von Merkmalen aus allen Faltungsschichten, wodurch sowohl feine Details als auch hochstufige semantische Informationen erfasst werden. RCF verwendet eine Fusion von Merkmalen auf mehreren Ebenen zur Verbesserung der Kantelokalisierung, während BDCN bidirektionale Verfeinerungsmechanismen einführt, um Kantenkontinuität zu erhöhen und Rauschstörungen zu reduzieren.

Neben diesen Architekturen wurden jüngst auch transformerbasierte Modelle für die Kantenerkennung untersucht, welche durch globale Kontextmodellierung die Robustheit in komplexen Szenen verbessern. Zudem haben sich selbstüberwachte Lernverfahren etabliert, die CNNs ermöglichen, Kanten ohne umfangreiche beschriftete Datensätze zu verfeinern. [45]

Um die Genauigkeit der Kantenerkennung weiter zu erhöhen, integrierten Forscher Optimierungsalgorithmen wie geführte Bildfilterung und Ameisen-Kolonien-Optimierung, wodurch glattere Kantenkarten und eine bessere Rauschunterdrückung als bei konventionellen Verfahren erzielt werden. Diese hybriden Methoden kombinieren Deep Learning mit klassischen Bildverarbeitungstechniken, um Präzision und Anpassungsfähigkeit zu maximieren. [44]

Mit der fortschreitenden Entwicklung der künstlichen Intelligenz wird erwartet, dass CNN-basierte Kantenerkennungsmethoden noch präziser, effizienter und widerstandsfähiger gegenüber schwierigen Bildbedingungen werden und so Fortschritte in Bereichen wie medizinischer Bildgebung, autonomer Navigation und hochauflösender Satellitenanalyse ermöglichen. [44]

KI-gestütztes Training für die Kantendetektion

Convolutional Neural Networks (CNNs) sind zur Grundlage moderner Kantenerkennung geworden, da sie hierarchische Merkmale direkt aus Daten lernen können. Im Gegensatz zu klassischen Kantenerkennungsmethoden, die auf vordefinierten Filtern und gradientenbasierten Verfahren basieren, ermöglichen CNNs die automatische Extraktion bedeutungsvoller Merkmale aus Bildern und sind somit hochgradig anpassungsfähig an unterschiedliche visuelle Kontexte. [47]

Bahnbrechende Arbeiten wie Holistically-Nested Edge Detection (HED) führten ein vollständig end-to-end trainierbares Netzwerk ein, das Kanten direkt aus Rohbildern vorhersagt und dabei multiskalige Merkmale mittels Deep Supervision nutzt. Dieser Ansatz verbesserte die Kantenerkennungsgenauigkeit erheblich, indem tiefere Schichten Kantenvorhersagen verfeinerten und gleichzeitig feine Details erhalten blieben. [48]

Nachfolgende Modelle wie Richer Convolutional Features (RCF) und das Bi-Directional Cascade Network (BDCN) steigerten die Leistung weiter durch die Aggregation von Merkmalen aus allen Faltungsschichten, wodurch sowohl feine Details als auch hochstufige semantische Informationen erfasst werden. RCF verwendet eine Fusion von Merkmalen auf mehreren Ebenen zur Verbesserung der Kantelokalisierung, während BDCN bidirektionale Verfeinerungsmechanismen einführt, um Kantenkontinuität zu erhöhen und Rauschstörungen zu reduzieren.

Neben diesen Architekturen wurden jüngst auch transformerbasierte Modelle für die Kantenerkennung untersucht, welche durch globale Kontextmodellierung die Robustheit in komplexen Szenen verbessern. Zudem haben sich selbstüberwachte Lernverfahren etabliert, die CNNs ermöglichen, Kanten ohne umfangreiche beschriftete Datensätze zu verfeinern.

Um die Genauigkeit der Kantenerkennung weiter zu erhöhen, integrierten Forscher Optimierungsalgorithmen wie geführte Bildfilterung und Ameisen-Kolonien-Optimierung, wodurch glattere Kantenkarten und eine bessere Rauschunterdrückung als bei konventionellen Verfahren erzielt werden. Diese hybriden Methoden kombinieren Deep Learning mit klassischen Bildverarbeitungstechniken, um Präzision und Anpassungsfähigkeit zu maximieren. [48]

Mit der fortschreitenden Entwicklung der künstlichen Intelligenz wird erwartet, dass CNN-basierte Kantenerkennungsmethoden noch präziser, effizienter und widerstandsfähiger gegenüber schwierigen

Bildbedingungen werden und so Fortschritte in Bereichen wie medizinischer Bildgebung, autonomer Navigation und hochauflösender Satellitenanalyse ermöglichen.

Erweiterte AI-Modelle

Aktuelle Fortschritte umfassen transformerbasierte Modelle wie EDTER (Edge Detection Transformer), die Self-Attention-Mechanismen nutzen, um langfristige Abhängigkeiten und globalen Kontext zu erfassen. Die zweistufige Architektur von EDTER verarbeitet zunächst grobe Bildausschnitte zur Erfassung des globalen Kontexts und verfeinert anschließend die Kanten mit feinen Ausschnitten. Dadurch erzielt das Modell Spitzenleistungen auf Benchmarks wie BSDS500. [52]

Ein weiteres bemerkenswertes Modell ist PiDiNet (Pixel Difference Network), das durch den Einsatz leichter Faltungsschichten auf Effizienz optimiert ist und eine Echtzeit-Kantenerkennung ermöglicht, ohne die Genauigkeit zu beeinträchtigen.

Im Bereich der KI-gestützten Kantenerkennung existieren zahlreiche Modelle. Im Folgenden werden wir EDTER näher betrachten: Das erste vollständig transformerbasierte Modell für Kantenerkennung, das globale Kontextmodellierung in zwei Stufen nutzt, ein Feature-Fusion-Modul (FFM) einsetzt und herausragende Performance bietet:

EDTER (Edge Detection Transformer)

- Erstes transformerbasiertes Kantenerkennungsmodell, das globalen Kontext (Stufe I) und feinkörnige lokale Merkmale (Stufe II) in einem zweistufigen Framework kombiniert.
- Der Bi-directional Multi-Level Aggregation (BiMLA) Decoder verbessert kantenbewusste Features durch Kombination von Top-Down- und Bottom-Up-Pfaden und erhält dünne Kanten besser als herkömmliche CNNs.
- Übertrifft CNN-basierte Modelle wie HED und RCF auf Benchmarks wie BSDS500 und NYUDv2 mit höheren Precision-Recall-Kurven und zeigt eine robuste Leistung bei Rauschen und komplexen Hintergründen. [52]

Global Context Modeling (Stufe I)

- Zerlegt Bilder in grobkörnige Patches (16×16) und verarbeitet diese mit einem globalen Transformer-Encoder, um langfristige Abhängigkeiten (z. B. Objektformen) zu erfassen.
- Nutzt den innovativen BiMLA-Decoder zur Verschmelzung hierarchischer Merkmale aus mehreren Transformer-Schichten, was die Kontinuität der Kanten bewahrt. [52]

Lokale Verfeinerung (Stufe II)

- Verarbeitet feinkörnige Patches (8×8) mit einem lokalen Transformer, um dünne Kanten wiederherzustellen und gleichzeitig den hohen Rechenaufwand einer pixelgenauen Aufmerksamkeit zu vermeiden.
- Das Feature Fusion Module (FFM) kombiniert globale und lokale Features und verbessert so die Schärfe der Kanten (Abbildung 1 der Publikation zeigt klarere Kanten im Vergleich zu CNNs).

Leistung

- Übertrifft CNN-Modelle auf dem BSDS500-Datensatz mit einem ODS-F-Maß von 0,828 sowie auf NYUDv2 um 3–5 % und demonstriert eine überlegene Fähigkeit, semantische Grenzen und Rauschen zu handhaben. [52]

3.2.8 Hybride Ansätze: Kombination von klassischen und AI-Methoden

Hybridmethoden in der Kantenerkennung verbinden klassische und KI-basierte Techniken, um deren jeweilige Stärken zu nutzen und so die Robustheit, Anpassungsfähigkeit und Effizienz zu steigern. Während klassische Verfahren wie Sobel oder Canny auf Gradienten basieren und oft durch Rauschen, Beleuchtungsunterschiede und komplexe Texturen beeinträchtigt werden, können Deep-Learning-Methoden durch Extraktion von mehrskaligen Merkmalen und Kontextinformationen präzisere Kantenerkennung ermöglichen. [47]

Typische Hybridansätze sind:

- Kombination klassischer Detektoren mit Deep Learning: Beispielhaft sind Multi-Stage-Learning-Frameworks, die CNNs zur Merkmalsextraktion mit SVM-Klassifikatoren verbinden. Dadurch werden Kantengenauigkeit und Strukturtreue verbessert, und die Entkopplung von Merkmalsrepräsentation und Klassifikation erhöht die Interpretierbarkeit.
- Optimierungsalgorithmen zur Verbesserung klassischer Methoden: Geführte Bildfilterung zusammen mit Ameisen-Kolonien-Optimierung erzeugt glattere Kanten und unterdrückt Rauschen effizienter. Wavelet-basierte Hybridmodelle koppeln Laplace-of-Gaussian-Filter mit Wavelet-Transformationen, um Kantenkontinuität und visuelle Klarheit zu steigern.
- Anwendungen in der Bildsteganographie: Kombinationen aus Canny- und Sobel-Operatoren mit Medianfiltern verbessern die Kantenerkennung und bewahren gleichzeitig die Bildqualität beim Einbetten von Informationen. Edge-preserving Filter zusammen mit Type-1-Fuzzy-Logik erhöhen die Robustheit in verrauschten Umgebungen. [50]

Durch Multi-Sensor-Fusion, adaptive Filter und KI-gestützte Optimierungen entwickeln sich Hybrid-Kantendetektionssysteme stetig weiter und bieten verbesserte Genauigkeit und Stabilität in Anwendungsfeldern wie medizinischer Bildgebung, autonomer Navigation und industrieller Inspektion.

Hybrid edge detection: classic filtering and CNN refinement

Eine häufig angewandte hybride Strategie beginnt mit klassischen Methoden wie der Canny-Filterung, um initiale Kantenkarten zu erzeugen, die anschließend durch ein Convolutional Neural Network (CNN) verfeinert werden. Diese Kombination nutzt die Rauschresistenz und Interpretierbarkeit klassischer Filter sowie die adaptive Präzision moderner Deep-Learning-Methoden. Diese Strategie vereint die Stärken traditioneller und KI-basierter Techniken, wodurch sowohl Robustheit gegenüber Rauschen als auch höhere Kantenpräzision erreicht wird. [45] Beispielsweise nutzt DexiNed (Dense Extreme Inception Network for Edge Detection) diesen Ansatz, um dünne, präzise Kanten bei gleichzeitiger Beibehaltung der Rauschunterdrückung des Canny-Detektors zu generieren. Im Gegensatz zu rein CNN-basierten Modellen benötigt DexiNed keine vortrainierten Gewichte und wird direkt trainiert, was eine gute Generalisierbarkeit über verschiedene Datensätze hinweg ermöglicht¹. Die CNN-Komponente lernt, Lücken zu schließen, fehlerhafte Kanten zu entfernen und die Kantenkontinuität zu verbessern, was zu saubereren und zuverlässigeren Ergebnissen führt. [45]

Jenseits von DexiNed repräsentiert EDTER einen rein transformer-basierten Ansatz. Obwohl EDTER keine explizite CNN-Integration vorsieht, wurde in der Forschung vorgeschlagen, dass eine Kombination von CNN- und Transformer-Architekturen durch Feature-Fusion die Kantenlokalisierung weiter verbessern könnte. Transformer erfassen besonders effektiv langreichweitige Abhängigkeiten und globalen Kontext, während CNNs lokale Texturen und feine Details extrahieren – ihre Kombination eröffnet somit vielversprechende Forschungsrichtungen. [52]

Derartige Hybridstrategien sind besonders wertvoll für Anwendungen mit hohen Anforderungen an die Kantendetektion, wie medizinische Bildgebung, autonome Navigation oder industrielle Inspektion. Durch die Integration klassischer Filterverfahren mit Deep-Learning-Optimierungen entwickeln sich moderne Kantenerkennungssysteme stetig weiter und bieten verbesserte Genauigkeit, Anpassungsfähigkeit und Robustheit. [52]

AI-Guided Feature Enhancement

Methoden wie AMH-Net (Attention-Guided Multi-Scale Hybrid Network) integrieren Aufmerksamkeitsmechanismen, um relevante Kanten hervorzuheben und Rauschen zu unterdrücken. Diese Modelle nutzen Multi-Scale-Feature-Fusion, um die Kantendetektion dynamisch an verschiedene Bildbedingungen anzupassen und gleichzeitig Robustheit zu gewährleisten. Durch aufmerksamkeitsgesteuerte Verarbeitung verbessert AMH-Net die Kantenklarheit und reduziert Störungen durch Hintergrundtexturen und Beleuchtungsvariationen. [48]

Solche Modelle verwenden häufig klassische Operatoren als Vorverarbeitungsschritt, um die Rechenlast zu verringern, gefolgt von einer KI-basierten Nachbearbeitung für feinere Details. Beispielsweise können Canny- oder Sobel-Filter zunächst grobe Kantenkarten extrahieren, die anschließend durch Deep-Learning-basierte Merkmalsverfeinerung optimiert werden. Dieser hybride Ansatz ermöglicht eine effiziente Kantendetektion bei gleichbleibend hoher Präzision. [48]

Aufmerksamkeitsmechanismen – wie Self-Attention-Layer und kanalweise Feature-Gewichtung – helfen KI-Modellen, sich auf relevante Kantenstrukturen zu konzentrieren und Rauschen zu unterdrücken. Zudem wurden Transformer-Architekturen für die Kantendetektion untersucht, da sie ein verbessertes globales Kontextverständnis und die Erfassung langreichweitiger Abhängigkeiten ermöglichen.

Diese Kombination ist besonders nützlich für Echtzeitanwendungen wie Videoüberwachung, Augmented Reality und autonome Navigation, bei denen Kantendetektion sowohl präzise als auch rechenefizient sein muss. Durch die Integration von Multi-Scale-Feature-Fusion, aufmerksamkeitsgesteuerter Verfeinerung und hybriden KI-klassischen Verarbeitungsmethoden entwickeln sich moderne Kantendetektionssysteme stetig weiter und bieten verbesserte Präzision und Anpassungsfähigkeit in verschiedenen Bildumgebungen. [48]

Erreichen von Präzision im Nanometerbereich

In hochpräzisen Bereichen wie der Halbleiterinspektion integrieren hybride Modelle klassische Kantenerkennung nahtlos mit Deep Learning, um Subpixel-Genauigkeit zu erreichen. Traditionelle Verfahren wie die Laplacian-of-Gaussian-(LoG)-Filterung dienen als erster Schritt zur Hervorhebung potenzieller Kanten, die anschließend von CNNs, welche auf nanoskalige Muster trainiert sind, validiert und verfeinert werden. Dieser Ansatz minimiert Fehlalarme und gewährleistet gleichzeitig die Erkennung echter Merkmale – ein entscheidender Faktor in der Halbleiterfertigung, bei der bereits kleinste Defekte die Leistungsfähigkeit der Bauteile beeinträchtigen können. [48]

Zur weiteren Verbesserung der Präzision wurden Super-Resolution-Bildgebungstechniken erforscht, die auf wellenletzenbasierter Rauschunterdrückung und Multi-Skalen-Feature-Fusion basieren, um die Kantengenauigkeit zu verfeinern. Zusätzlich wurden Elektronenstrahlinspektion (EBI) und Rasterkraftmikroskopie (AFM) in die Workflows zur Fehlererkennung in Halbleitern integriert und bieten eine Nanometerauflösung zur Identifikation struktureller Anomalien. [47]

Trotz dieser Fortschritte bestehen noch einige Einschränkungen. So sind beispielsweise die von EDER erzeugten Kanten häufig mehrere Pixel breit und erreichen nicht die ideale Ein-Pixel-Präzision, die für die Detektion ultrafeiner Strukturen erforderlich ist. Zukünftige Forschungen könnten sich daher auf Nachbearbeitungstechniken wie Non-Maximum-Suppression oder hybride CNN-Transformer-Architekturen konzentrieren, um dünnere und präzisere Kanten zu erzielen. Darüber hinaus könnten selbstüberwachtes Lernen und adaptive Schwellenwertverfahren die Kantenglättung weiter verbessern, indem sie die Erkennungsparameter dynamisch an kontextuelle Bildmerkmale anpassen.

Da die Halbleiterfertigung zunehmend auf Sub-10-nm-Knoten voranschreitet, wird die Nachfrage nach hochpräziser Defektinspektion weitere Innovationen in KI-gestützter Kantenerkennung, multimodaler Sensorfusion und quantenunterstützten Bildgebungstechniken vorantreiben. Diese Entwicklungen werden höhere Ausbeuten, geringeren Materialverlust und eine verbesserte Zuverlässigkeit von Halbleiterbauteilen der nächsten Generation sicherstellen. [47]

Die Zukunft der Kantenerkennung: Mehr Präzision und Anpassungsfähigkeit

Der Übergang von klassischen Kantenerkennungsverfahren zu CNN-basierten Ansätzen markiert einen Paradigmenwechsel hin zu robusteren, präziseren und flexibleren Systemen. Innovationen wie Holistically-Nested Edge Detection (HED) und die Fusion von Multi-Skalen-Antworten haben die Kantengenauigkeit, Konsistenz und Rechenleistung erheblich verbessert. Diese Fortschritte ermöglichen es Modellen, komplexe Muster zu erlernen, sich an verschiedenste Bildgebungsbedingungen anzupassen und multimodale Datenquellen für eine bessere Leistung zu integrieren. [48]

Mit Blick auf die Zukunft werden selbstüberwachtes Lernen, hybride Intelligenzansätze und weiterentwickelte Architekturoptimierungen die Kantenerkennung kontinuierlich verfeinern. Die Kombination von Deep Learning mit Fuzzy-Logik, multimodaler Fusion und gerichteten azyklischen Graphen (DAG) wird die Präzision weiter steigern und Kantenerkennungssysteme widerstandsfähiger gegenüber Rauschen, Beleuchtungsschwankungen und komplexen Texturen machen.

Da KI-gestützte Kantenerkennung sich fortlaufend weiterentwickelt, werden ihre Anwendungen sich auf Bereiche wie Halbleiterfertigung, medizinische Bildgebung, autonome Navigation und industrielle Inspektion ausweiten. Durch die Verbindung klassischer Methoden mit modernsten KI-Techniken steht die Branche kurz davor, Nanometer-Präzision zu erreichen und so neue Möglichkeiten für die hochauflösende Merkmalserkennung unter anspruchsvollsten Bedingungen zu erschließen. [48]

3.2.9 Weitere wichtige Anwendungen

Moderne Messsysteme nutzen zunehmend Künstliche Intelligenz, um traditionelle Einschränkungen bei der Datenqualität, Sensorempfindlichkeit und Betriebsüberwachung zu überwinden. Über die grundlegenden Anwendungen in der Objektlokalisierung und Regressionsmodellierung hinaus bieten KI-gesteuerte Techniken nun transformative Verbesserungen in vielen kritischen Bereichen. Über die Objektlokalisierung

nung und Regressionsmodelle hinaus bieten KI-gesteuerte Messtechnologien robuste Verbesserungen bei der Datenverarbeitung, Systemgenauigkeit und prädiktiven Zuverlässigkeit. Diese Fortschritte betreffen die Signalintegrität, Sensorleistung und Anomalieerkennung und gewährleisten präzise und adaptive Reaktionen in verschiedenen Branchen. Von der Unterdrückung von Signalrauschen bis zur Sensorkalibrierung mittels Deep Learning verfeinert KI Messtechniken, indem sie komplexe Muster erkennt und Fehler vorhersagt, bevor diese die Systemleistung beeinträchtigen [54].

Praktische Anwendungen von CNN-basierter Lokalisierung umfassen Roboterarme, die Objekte auf Förderbändern erkennen und greifen [55], Drohnen, die Flugrouten basierend auf YOLO-abgeleiteten Orientierungspunkten anpassen, und intelligente Fabriken, die Teile und Defekte mit Hochgeschwindigkeitskameras und eingebetteten Bildverarbeitungsprozessoren verfolgen [55].

Signalrauschunterdrückung

Deep-Learning-Architekturen trennen effektiv sinnvolle Signale von Rauschen in Umgebungen mit niedrigen Signal-Rausch-Verhältnissen. Anwendungen mit hohem Rauschen umfassen die Vibrationsanalyse in der vorausschauenden Wartung, wo rekurrente neuronale Netze eine Rauschunterdrückung von >90% bei der Überwachung rotierender Maschinen erreichen.

Lernalgorithmen könnten auf vielfältige Weise zur Verbesserung der Genauigkeit und Zuverlässigkeit von IoT-Lokalisierungssystemen eingesetzt werden. Einige dieser Anwendungen sind wie folgt zusammengefasst:

- **Kalibrierung:** Lernalgorithmen können zur Kalibrierung von Sensoren und anderen IoT-Gerätekomponenten verwendet werden, um sicherzustellen, dass diese korrekt funktionieren und genaue Daten liefern. Dies kann dazu beitragen, die Gesamtgenauigkeit der Standortschätzungen zu verbessern [56].
- **Rauschunterdrückung:** Algorithmen des maschinellen Lernens können verwendet werden, um Rauschen und andere Fehlerquellen aus Daten von IoT-Geräten zu entfernen. Dies kann dazu beitragen, die Genauigkeit von Standortschätzungen zu verbessern, indem die Auswirkungen von Fehlern und anderen Faktoren, die die Daten verzerren können, reduziert werden. Algorithmen des maschinellen Lernens können verwendet werden, um die relevantesten Merkmale in von IoT-Geräten gesammelten Daten zu identifizieren, wodurch die Genauigkeit der Standortschätzung verbessert wird, indem sich auf die wichtigsten Faktoren konzentriert wird [54].
- **Modellauswahl:** Algorithmen des maschinellen Lernens können verwendet werden, um das beste Modell oder die beste Kombination von Modellen für eine bestimmte Anwendung zu identifizieren, wodurch die Genauigkeit der Standortschätzung verbessert wird, indem das beste Modell ausgewählt wird, das zu den Daten passt [57].
- **Verbesserung der Genauigkeit:** Große Datenmengen können von Algorithmen des maschinellen Lernens analysiert werden, um genauere Schätzungen des Gerätestandorts zu liefern [55].
- **Automatisierung des Lokalisierungsprozesses:** Maschinelles Lernen hat das Potenzial, den Prozess der Bestimmung des Gerätestandorts zu automatisieren und somit die Notwendigkeit manueller Eingaben zu eliminieren.
- **Anpassung an Umgebungsveränderungen:** Um genauere Standortschätzungen zu liefern, können sich Algorithmen des maschinellen Lernens an Umgebungsveränderungen anpassen, wie z.B. neue Hindernisse oder Änderungen der Signalstärke [37].

In einer Studie untersuchten Forscher die Rauschunterdrückung und Auflösungsverbesserung in der PET-Bildgebung. Die Positronen-Emissions-Tomographie (PET)-Bildgebung steht vor inhärenten Herausforderungen im Zusammenhang mit Rauschen und räumlicher Auflösung, die die quantitative Genauigkeit in der medizinischen Diagnostik beeinflussen. Traditionelle Techniken, wie die quadratische Straffung und die Gaußsche Filterung, wurden zur Minderung dieser Probleme eingesetzt. KI-basierte Me-

thoden, insbesondere Convolutional Neural Networks (CNNs), haben jedoch die PET-Bildverarbeitung transformiert, indem sie direkte Abbildungen zwischen verrauschten und sauberen Bildern lernen [58].

Moderne Architekturen umfassen:

- **3D-CNN-Modelle:** Bieten trotz hoher Rechenanforderungen überlegene Leistung.
- **Hybride 2D-3D-Netzwerke:** Integrieren dimensionale Merkmale für bessere Ergebnisse.
- **Unüberwachtes und selbstüberwachtes Lernen:** Nutzen Populationsmerkmale zur Verfeinerung verrauschter Daten [33].

Aufkommende Paradigmen, wie föderiertes Lernen für den Datenschutz in mehreren Zentren und Reinforcement Learning für die adaptive Verarbeitung, verbessern die Zuverlässigkeit der PET-Bildgebung weiter. Da KI-gesteuerte Techniken sich ständig weiterentwickeln, werden Transferlernen, Datenharmonisierung und erklärbare KI entscheidende Rollen bei der Erzielung klinisch signifikanter Verbesserungen spielen.

KI-gesteuerte Entrauschungstechniken in der PET-Bildgebung basieren überwiegend auf überwachtem Lernen, wobei 3D-U-Net-Architekturen weit verbreitet sind. Diese Modelle verwenden typischerweise einen L2-Verlust, oft kombiniert mit perzeptuellen oder adversariellen Zielen, um die Bildklarheit und diagnostische Genauigkeit zu verbessern [58].

Herausforderungen und zukünftige Richtungen Trotz ihres Erfolgs steht die KI-gesteuerte PET-Entrauschung vor Problemen der Generalisierbarkeit, hohen Rechenanforderungen und potenzieller GAN-bedingter Überglättung. Das Feld verschiebt sich hin zu hybriden Architekturen, die multimodale Daten und physikalische Einschränkungen für eine verbesserte klinische Anwendbarkeit integrieren [58].

Sensorkalibrierung

Da die Messtechnik ein integraler Bestandteil der Automatisierung, Robotik und industriellen Systeme wird, wächst der Bedarf an präzisen und adaptiven Sensordaten. Sensoren unterliegen jedoch Fehlern aufgrund von Umgebungsveränderungen, Alterung und Signalrauschen, wodurch traditionelle Kalibrierungsmethoden – die oft manuell und statisch sind – für Echtzeitanwendungen unzureichend werden.

Deep Learning transformiert die Sensorkalibrierung durch:

- Modellierung nichtlinearer Beziehungen zwischen Rohsensorausgaben und wahren Werten.
- Ermöglichung kontinuierlicher, In-situ-Kalibrierung ohne Betriebsunterbrechung.
- Verbesserung der Genauigkeit bei gleichzeitiger Reduzierung der Wartungskosten.

Die neuronale netzbasierte Kalibrierung ist besonders nützlich in der Industrieautomation, autonomen Fahrzeugen und Präzisionsfertigung und bietet skalierbare und adaptive Lösungen. Dieser Abschnitt wird Schlüsseltechniken, Architekturen und reale Anwendungen untersuchen, die KI für eine verbesserte Sensorkalibrierungsleistung unter dynamischen Bedingungen nutzen [59].

Drahtlose Sensornetzwerke (WSNs) Eine der wichtigsten Kalibrierungsmethoden sind Drahtlose Sensornetzwerke (WSNs), die für die Umweltüberwachung, intelligente Städte und Präzisionslandwirtschaft von entscheidender Bedeutung sind und Echtzeit-Raum- und Zeitmessungen ermöglichen. Sensor-Drift – eine allmähliche Abweichung von genauen Messwerten aufgrund von Umgebungsveränderungen oder Hardwareverschleiß – stellt jedoch eine große Herausforderung bei großflächigen Implementierungen dar, bei denen eine manuelle Neukalibrierung unpraktisch ist. Traditionelle Blind-Kalibrierungsmethoden basieren auf festen Annahmen wie linearen Signalsubräumen oder dichter Sensorplatzierung, was die Anpassungsfähigkeit einschränkt [28].

PRNet: KI-gesteuerte Sensorkalibrierung PRNet (Projection-Recovery Network) ist ein tiefes, vollständig faltendes Netzwerk, das entwickelt wurde, um die Blind-Kalibrierung in WSNs zu automatisieren und wichtige Herausforderungen zu bewältigen:

- **Merkmalsextraktion ohne vorherige Modelle:** Im Gegensatz zu unterraumgestützten Methoden, die Linearität annehmen, lernt PRNet räumlich-zeitliche Korrelationen direkt aus Sensordaten. Es verwendet eine Projektionsschicht, um Driftkomponenten zu isolieren, und Wiederherstellungsschichten, um saubere Signale zu rekonstruieren.
- **Dateneffizienz:** Um begrenzte reale Trainingsdaten zu überwinden, synthetisiert PRNet Trainingsbeispiele, indem es kurzfristige Messungen nach der Bereitstellung mit simulierten Driftmustern erweitert.

Methodik & Architektur

- **Projektionsschicht:** Bildet verdriftete Messungen in einen Merkmalsraum ab, in dem Drift und Signal trennbar sind.
- **Wiederherstellungsschichten:** Führen extrahierte Merkmale zusammen, um die Drift abzuschätzen, während Batch-Normalisierung für stabiles Training verwendet wird.
- **Sensormanipulation:** Optimiert die Eingabetensorstruktur durch Neuordnung benachbarter Sensoren, um lokale Korrelationen zu verbessern.
- **Verlustfunktionen & Augmentierung:** Kombiniert Projektionsverlust mit Wiederherstellungsverlust. Driftfreie Segmente werden mit synthetischen Rauschmodellen gestört, um diverse Trainingsbeispiele zu generieren [60].

Experimente & Ergebnisse

- **Testumgebung:** 6 Sensoreinheiten, jede mit 4 Sensoren, kalibriert gegen hochpräzise Thermometer.
- **Simulierte Daten:** Nichtlineares Sensorfeld mit kontrollierten Drifteinspeisungen.
- **Leistungshighlights:**
 - PRNet stellt doppelt so viele verdriftete Sensoren wieder her wie SPSR-TSBL (80% Wiederherstellungsrate).
 - Übertrifft SVR-KF (vorhersagebasierter Ansatz) unter rauschigen und nichtlinearen Bedingungen.
 - Generalisiert auf ungesehene Drift-Typen (z.B. Schrittdrift, variierende Rauschpegel) [27].

Anomalieerkennung in Messungen

Im Rahmen der KI in der Messtechnik spielt die Anomalieerkennung eine Schlüsselrolle bei der vorausschauenden Wartung, Qualitätssicherung und Fehlerdiagnose. Dieser Abschnitt untersucht, wie moderne KI-Modelle zur Verbesserung der Anomalieerkennung in sensorbasierten Systemen eingesetzt werden, welche Architekturen am häufigsten angewendet werden und wie diese Methoden die Zuverlässigkeit und Betriebseffizienz in einer Reihe von Branchen verbessern.

Anomalieerkennung ist in Bereichen wie Cybersicherheit, Gerätwartung, Betrugserkennung und Gesundheitsüberwachung von entscheidender Bedeutung, wo die Identifizierung von Abweichungen von erwarteten Mustern die Systemzuverlässigkeit gewährleistet. Trotz ihrer breiten Anwendbarkeit behindern methodische Inkonsistenzen in der Anomalieerkennungsforschung – wie z.B. unterschiedliche Datenpartitionierungsstrategien, Metrikwahl und Implementierungsdetails – die Reproduzierbarkeit und faire Bewertung [61].

Herausforderungen bei der Anomalieerkennung

- **Inkonsistente Datenteilung:** Unterschiedliche Partitionierungsmethoden beeinflussen die Modellleistung und Vergleichbarkeit.
- **Metrikvariabilität:** Metriken wie F1-Score, Präzision und Recall hängen von der Schwellenwertauswahl und dem Klassenungleichgewicht ab, was zu inkonsistenten Ergebnissen führt.
- **Implementierungslücken:** Unterschiede in der Datenvorverarbeitung wirken sich erheblich auf die gemeldete Genauigkeit aus.

Standardisierungsbemühungen und Ergebnisse Forscher plädieren für vereinheitlichte Protokolle, einschließlich des ausschließlichen Trainings auf normalen Daten, der Definition von Anomalien als positive Klasse und der Verwendung robuster Metriken wie AUPR für eine verbesserte Anomalieerkennungsempfindlichkeit. Empirische Studien zeigen:

- Neu bewertete Modelle unterscheiden sich oft von früheren veröffentlichten Ergebnissen aufgrund inkonsistenter Bewertung.
- Einfache Modelle wie Vanilla-Autoencoder (DAE) können komplexere Architekturen übertreffen.
- Herausfordernde Datensätze (z.B. CSE-CIC-IDS2018) liefern aussagekräftigere Erkenntnisse als weit verbreitete, aber triviale Datensätze (z.B. KDD).

KI-basierte Anomalieerkennung in drahtlosen Sensornetzwerken (WSNs) WSNs spielen eine wichtige Rolle im Internet der Dinge (IoT), aber Sensoranomalien – verursacht durch Hardwarefehler, Umgebungsveränderungen oder Cyberangriffe – bedrohen die Datenzuverlässigkeit. Traditionelle zentralisierte Anomalieerkennung ist aufgrund hoher Datenübertragungskosten ineffizient, während verteilte Methoden oft unter hoher Rechenkomplexität oder übermäßiger Kommunikation zwischen Sensoren leiden.

Autoencoder-basierte Anomalieerkennung für WSNs Ein dreischichtiger Autoencoder komprimiert und rekonstruiert Sensordaten, um Anomalien auf der Grundlage von Rekonstruktionsfehlern zu erkennen, wobei er ohne gelabelte Daten durch unüberwachtes Lernen arbeitet.

Verteilter Algorithmus-Ansatz

- **Sensor-Ebene (Edge Processing):** Jeder Sensor erkennt Anomalien lokal, indem er Eingabe- und rekonstruierte Werte vergleicht, ohne dass eine Kommunikation zwischen den Sensoren erforderlich ist.
- **Cloud-Ebene-Verarbeitung:** Aggregiert Residuen und aktualisiert globale Statistiken, wobei der Autoencoder regelmäßig neu trainiert wird, um die Genauigkeit zu erhalten und gleichzeitig den Sensor-Overhead zu reduzieren.

Approach	Data Processing	Advantages	Limitations
Centralized	Sends all sensor data to a central server for analysis.	<ul style="list-style-type: none">- High accuracy with large-scale data.- Suitable for deep learning.	<ul style="list-style-type: none">- High bandwidth and latency.- Privacy concerns.
k-NN	Compares new data to nearest neighbors.	<ul style="list-style-type: none">- Simple and clear.- Good with structured data.	<ul style="list-style-type: none">- Costly for big datasets.- Sensitive to noise.
Autoencoder	Compresses and reconstructs data to detect anomalies.	<ul style="list-style-type: none">- Handles high-dimensional data.- Learns patterns automatically.- Low communication need.	<ul style="list-style-type: none">- Needs training data.- Hard to interpret.- Computationally heavy.

Tabelle 3.1: Vergleich verschiedener Ansätze zur Anomalieerkennung

Kapitel 4

Netzwerkstrategien und ihre Bewertung

Im dynamischen Bereich des Deep Learnings spielen das Design, die Optimierung und die Bewertung neuronaler Netze eine entscheidende Rolle beim Erreichen von Hochleistungsmodellen. Dieses Kapitel bietet einen umfassenden Überblick über wesentliche Strategien zum Aufbau und zur Bewertung tiefer Netze, wobei der Schwerpunkt auf Architekturauswahl, Aktivierungsfunktionen, Optimierungsmethoden, Regularisierungstechniken und Modellbewertungsmetriken liegt.

Wichtige Überlegungen beim Architekturdiseign, wie Schichtkonfiguration, Merkmalskalierbarkeit und rechnerische Effizienz, leiten die Wahl der Netzwerkstrukturen, während Fortschritte wie die Neuronale Architektursuche (NAS) den Prozess automatisieren. Die Auswahl der Aktivierungsfunktionen, wie ReLU, Sigmoid und Tanh, ist entscheidend für die Bestimmung der Lernfähigkeit eines Netzwerks, wobei Alternativen wie Leaky ReLU und Mish eine bessere Gradientenstabilität bieten. Optimierungsmethoden, insbesondere Stochastic Gradient Descent (SGD) und Adam, verfeinern den Lernprozess, wobei Hybridansätze wie SWATS die Konvergenzgeschwindigkeit und Generalisierung ausgleichen. Regularisierungstechniken wie Dropout und L2-Gewichtszerrfall sind entscheidend, um Overfitting zu kontrollieren und die Modellrobustheit zu verbessern. [62]

Zur Bewertung der Modellleistung liefern Metriken wie Mean Squared Error, R²-Score, Genauigkeit, Präzision und Recall Einblicke in die Vorhersagegenauigkeit, während Kreuzvalidierungstechniken und Early Stopping dazu beitragen, Overfitting zu mindern und die Generalisierung auf ungesehene Daten zu verbessern. Dieses Kapitel befasst sich mit diesen Strategien und bietet eine ganzheitliche Sicht darauf, wie Netzwerke entworfen, trainiert und bewertet werden, um eine optimale Leistung zu erzielen. [63]

4.1 Architekturauswahl und Designkriterien

Die Effektivität von Deep-Learning-Modellen wird maßgeblich durch ihr architektonisches Design bestimmt, das traditionell durch Expertenwissen und manuelle Abstimmung geleitet wird. Das Aufkommen der Neuronalen Architektursuche (NAS) hat diesen Prozess jedoch revolutioniert, indem es die Auswahl und Optimierung von Netzwerkstrukturen automatisiert. Bei der Bewertung oder dem Entwurf von Architekturen müssen mehrere Schlüsselkriterien berücksichtigt werden, um Effizienz, Skalierbarkeit und Generalisierung zu gewährleisten. [44]

Im Folgenden werden wir uns die neuronalen Netze (NN) genauer ansehen, einschließlich der Eingabe-, versteckten und Ausgabeschichten. Die Neuronen in aufeinanderfolgenden Schichten sind vollständig verbunden. Die Eingabedaten werden durch $\{x_1, x_2, \dots, x_n\}$ und die Ausgabedaten durch $\{y_1, y_2, \dots, y_m\}$ dargestellt. Die Ausgabedaten jeder Schicht werden als $\{h_1^{(l)}, h_2^{(l)}, \dots, h_{n_l}^{(l)}\}$ dargestellt, wobei das Gewicht und der Bias für jedes Neuron als $w_{ij}^{(l)}$ bzw. $b_{ij}^{(l)}$ bezeichnet werden. [33]

Die Sigmoid- und Swish-Aktivierungsfunktionen werden verwendet, um die Lernfähigkeit des Netzwerks zu verbessern. Zusätzlich wird Dropout angewendet, um Overfitting zu verhindern, und Batch-Normalisierung wird verwendet, um ein stabiles Training zu gewährleisten. Das Netzwerk wird mit dem Stochastic Gradient Descent (SGD)-Algorithmus trainiert. Schließlich werden die NN-Parameter, ein-

schließlich der Anzahl der Schichten, der Lernrate, der Dropout-Rate und anderer Einstellungen, in Tabelle 4.1 dargestellt.

Number	Parameters	Value
1	Full connection layer	3
2	Learning rate	0.001
3	Dropout	0.5
4	Epoch	10,000
5	Batch size	1024
6	Adam	Stochastic Gradient Descent

Tabelle 4.1: Model parameters and configuration

Diese Tabelle zeigt die wichtigsten Einstellungen, die für das Training des KI-Modells verwendet werden, wie ein Rezept für das Lernen. Die KI verwendet drei Schichten zur Verarbeitung von Informationen, lernt langsam (0,001), um Fehler zu vermeiden, und „vergisst“ während des Trainings zufällig die Hälfte ihres Wissens (0,5 Dropout), um flexibel zu bleiben. Es übt 10.000 Runden lang an jeweils 1.024 Beispielen und nutzt intelligente, selbstanpassende Techniken, um Aufgaben wie die Bilderkennung effizient zu meistern. Basierend auf der Forschung von „*Learning Activation Functions to Improve Deep Neural Networks*“ (2019). [64]

Suchraumgestaltung

Der Suchraum definiert den Bereich möglicher Architekturen, wobei Flexibilität und Handhabbarkeit ausbalanciert werden. Gängige Designentscheidungen umfassen:

- Kettenstrukturierte Netzwerke oder auch sequenzielle Schichten werden zur Vereinfachung und einfacheren Implementierung verwendet.
- Mehrzweig-Topologien zur Verbesserung des Gradientenflusses und der Merkmalswiederverwendung.
- Zellenbasierte Architekturen, bei denen modulare Bausteine gestapelt werden, um die Skalierbarkeit und Übertragbarkeit zu verbessern. [65]

Ein gut strukturierter Suchraum integriert vorhandenes Wissen, während er genügend Flexibilität beibehält, um Innovationen zu fördern.

Leistungsgesteuerte Auswahlstrategien

Die Auswahl einer optimalen Architektur erfordert eine effiziente Navigation im Suchraum unter Verwendung verschiedener Optimierungsstrategien:

- Reinforcement Learning (RL): Verwendet eine politikgesteuerte Architekturgenerierung, um die Validierungsgenauigkeit zu maximieren.
- Evolutionäre Algorithmen: Verwenden Mutations- und Selektionstechniken, um Architekturen zu verfeinern, wodurch oft kompakte, hochleistungsfähige Modelle entstehen.
- Gradientenbasierte Methoden: Nutzen kontinuierliche Relaxationen, um eine differenzierbare Architektursuche zu ermöglichen. [65]

Eine effektive Auswahlstrategie muss die Exploration und Exploitation (Verfeinerung bekannter hochleistungsfähiger Architekturen) ausbalancieren.

Effizienz und Ressourcenbeschränkungen

Die praktische Architekturauswahl beinhaltet die Bewertung von Kompromissen zwischen:

- Genauigkeit vs. Rechenkosten: Niedrige Wiedergabetreue-Schätzungen oder Gewichtsteilungstechniken beschleunigen die Bewertung, können aber Verzerrungen einführen.
- Skalierbarkeit: Hierarchische Suchräume verbessern die Anpassungsfähigkeit an verschiedene Aufgaben.
- Hardware-Kompatibilität: Architekturen müssen den Bereitstellungsbeschränkungen, wie Latenz- und Speicherbegrenzungen, entsprechen. [66]

Generalisierung und Robustheit

Über die aufgabenspezifische Leistung hinaus sollten ideale Architekturen:

- Effektiv auf neue Datensätze oder Domänen übertragen werden.
- Overfitting durch Modularität und Regularisierung widerstehen.
- Sich dynamisch an Multi-Task- oder Multi-Objective-Einstellungen anpassen, wobei Genauigkeit und Effizienz ausbalanciert werden. [66]

Neuronale Architektursuche (NAS)

Das traditionelle Design neuronaler Netze war lange Zeit von manuellem Fachwissen abhängig. Die Neuronale Architektursuche (NAS) automatisiert diesen Prozess:

- Reinforcement Learning (RL): Strategien generieren Architekturen basierend auf Validierungsgenauigkeit
- Evolutionäre Algorithmen: Architekturen entwickeln sich durch Mutation und Selektion
- Zufallssuche: Gut strukturierte Suchräume liefern effektive Ergebnisse [33]

NAS beschleunigt die Modellfindung und verbessert die Effizienz in Deep-Learning-Anwendungen.

Hybride NAS-Ansätze

Kombination von Automatisierung mit menschlichem Fachwissen [45]:

- Hierarchische Suchräume: Modulare Motive für skalierbare Architekturen
- Beschränkte Suche: Exploration auf plausible Topologien begrenzen
- Warm-Start NAS: Initialisierung mit handgefertigten Hochleistungsarchitekturen [63]

Diese Ansätze erhalten menschliche Intuition bei Automatisierungsvorteilen.

Industrielle Anwendungen

NAS-Erfolge in verschiedenen Branchen:

- Computer Vision:
 - Bildklassifizierung: Überlegene Performance auf CIFAR-10, ImageNet
 - Objekterkennung: Optimierung für Edge-Geräte
- Natural Language Processing: Verbesserte Transformer- und RNN-Modelle
- Hardware-Aware NAS: Anpassung für TPUs, GPUs und Embedded-Systeme [44]

4.1.1 Aktivierungsfunktionen

In künstlichen neuronalen Netzen spielen Aktivierungsfunktionen eine entscheidende Rolle bei der Gestaltung des Lernprozesses und der Beeinflussung der Gesamtleistung. Während viele Aktivierungsfunktionen als universelle Approximatoren dienen, variiert ihre Effizienz, was umfangreiche Forschungsarbeiten zur Verbesserung der Fähigkeiten neuronaler Netze anregt. Trotz ihrer Bedeutung stoßen konventionelle Aktivierungsfunktionen oft auf Herausforderungen wie Sättigung, das sterbende Neuron Problem und das Problem des explodierenden oder verschwindenden Gradienten.

Um diese Einschränkungen zu beheben, stellt diese Arbeit zwei neuartige Aktivierungsfunktionen vor – absolute lineare Einheiten und inverse polynomische lineare Einheiten – die jeweils einen einstellbaren Parameter enthalten, um die Steigung des Gradienten für eine verbesserte Optimierung zu optimieren. Diese Studie bietet auch eine detaillierte Taxonomie von Aktivierungsfunktionen, die deren Eigenschaften und Verhalten systematisch kategorisiert. Umfassende Experimente werden an verschiedenen tiefen neuronalen Architekturen durchgeführt, wobei Netzwerktyp und -tiefe berücksichtigt werden, um die Wirksamkeit dieser neuartigen Funktionen zu bewerten. Ihre Leistung wird bei Bild- und Textklassifizierungsaufgaben unter Verwendung mehrerer öffentlicher Benchmark-Datensätze rigoros getestet, um einen gründlichen Vergleich mit weit verbreiteten Aktivierungsfunktionen zu gewährleisten. Die Ergebnisse zeigen, dass die vorgeschlagenen Aktivierungsfunktionen viele konventionelle in mehreren Benchmarks übertreffen. Eine statistische Analyse über Klassifizierungskategorien hinweg bestätigt zusätzlich ihre Robustheit und überragende Genauigkeit und etabliert sie als vielversprechende Alternativen in Deep-Learning-Anwendungen. [63]

Grundlegende Aktivierungsfunktionen: Sigmoid, Tanh und ReLU

Aktivierungsfunktionen führen Nichtlinearität in neuronale Netze ein und ermöglichen es ihnen, komplexe Muster zu lernen. Zu den am weitesten verbreiteten gehören die Rectified Linear Unit (ReLU), die logistische Sigmoid-Funktion und die Hyperbolic Tangent (Tanh)-Funktion – jede bietet einzigartige Vorteile und Kompromisse. [67]

Traditionelle Nichtlinearitäten: Sigmoid und Tanh

Historisch waren Sigmoid- und Tanh-Funktionen grundlegend für die Einführung von Nichtlinearität.

- **Sigmoid-Funktion:** Die Sigmoid-Aktivierungsfunktion bildet Eingangswerte auf den Bereich $[0, 1]$ ab, was sie für probabilistische Interpretationen in Ausgabeschichten nützlich macht. Sie ist definiert als:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (4.1)$$

Obwohl historisch beliebt, leidet Sigmoid unter verschwindenden Gradienten bei extremen Eingangswerten, was das Training tiefer Netze aufgrund von Sättigung erheblich verlangsamt.

- **Hyperbolic Tangent (Tanh)-Funktion:** Die Tanh-Funktion transformiert Eingaben in den Bereich $[-1, 1]$, wodurch sie nullzentriert wird, was die Konvergenz im Vergleich zu Sigmoid unterstützen kann. Sie ist gegeben durch:

$$\text{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (4.2)$$

Tanh verbessert die Konvergenz gegenüber Sigmoid, leidet aber immer noch unter Gradientensättigung bei großen Eingabewerten, ähnlich der Sigmoid-Funktion.

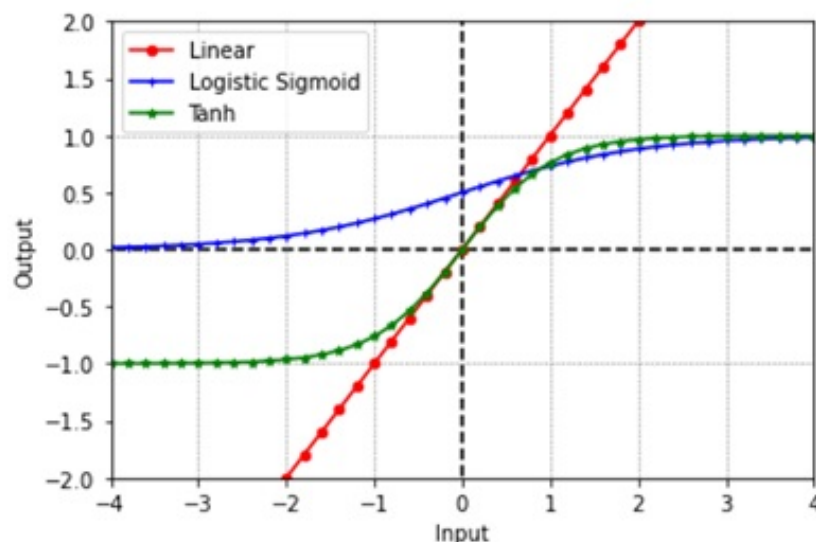


Abbildung 4.1: Illustration von linearen, logistischen Sigmoid- und Tanh-Aktivierungsfunktionen, angepasst aus *Activation Functions in Deep Learning* Typen

Der moderne Standard: Rectified Linear Unit (ReLU)

Die Rectified Linear Unit (ReLU) ist aufgrund ihrer Recheneffizienz und ihrer Fähigkeit, Sättigung für positive Eingaben zu vermeiden, weit verbreitet im Deep Learning. Sie ist definiert als:

$$\text{ReLU}(x) = \max(0, x) \quad (4.3)$$

Vorteile von ReLU:

- **Effiziente Berechnung:** ReLU beinhaltet keine Exponentialfunktion, wodurch die Berechnung wesentlich schneller erfolgt als bei Sigmoid oder Tanh.
- **Sparsity:** Durch die Ausgabe von Null für negative Eingaben führt ReLU Sparsity im Netzwerk ein, was potenziell die Aktivierungsredundanz reduziert und die Recheneffizienz verbessert. [68]

Einschränkungen von ReLU:

- **Sterbende ReLU (Dying ReLU):** Neuronen können dauerhaft inaktiv werden, wenn ihre Eingabe konstant in den negativen Bereich fällt, was zu einem Gradienten von Null führt. Dies bedeutet, dass das Neuron das Lernen vollständig einstellt.
- **Unbegrenzte Ausgabe:** Für positive Eingaben ist die Ausgabe von ReLU unbegrenzt, was manchmal zu Instabilität in sehr tiefen Netzen führen kann, indem es große Aktivierungswerte verursacht. [68]

Hybride und adaptive Varianten

Um die Mängel von ReLU zu mildern und die Einschränkungen traditioneller Aktivierungsfunktionen zu beheben, wurden mehrere modifizierte und adaptive Varianten eingeführt:

- **Leaky ReLU (LReLU):** Diese Variante behebt das Problem des "sterbenden ReLU", indem sie einen kleinen, nicht-null Gradienten für negative Eingaben zulässt. Sie ist definiert als:

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{falls } x > 0 \\ \alpha x & \text{falls } x \leq 0 \end{cases} \quad (4.4)$$

wobei α eine kleine positive Konstante ist. [63]

- **Parametric ReLU (PReLU):** Eine Erweiterung von Leaky ReLU, PReLU verwendet einen lernbaren Parameter α für negative Eingaben, wodurch das Netzwerk die Steigung während des Trainings anpassen kann.
- **Scaled Tanh & SiLU (Sigmoid Linear Unit):** Diese und andere Varianten zielen darauf ab, die Vorteile von Sigmoid/Tanh (z.B. begrenzte Ausgaben) zu kombinieren, während ein effizienter Gradientenfluss erhalten bleibt und Sättigungsprobleme behoben werden. [64]

Praktische Überlegungen

- **Prävalenz von ReLU:** ReLU wird aufgrund seiner Einfachheit und Effektivität im Allgemeinen für die meisten Deep-Learning-Anwendungen bevorzugt, da es ein gutes Gleichgewicht zwischen Rechenkosten und Leistung bietet.
- **Kontextspezifische Nutzung:** Sigmoid und Tanh bleiben in bestimmten Kontexten nützlich, wie z.B. bei der binären Klassifizierung (Sigmoid in der Ausgabeschicht) und rekurrenten neuronalen Netzen (Tanh), erfordern aber eine sorgfältige Initialisierung, um verschwindende Gradienten zu vermeiden.
- **Aufkommende Funktionen:** Die Entwicklung neuartiger Aktivierungsfunktionen, einschließlich der in dieser Arbeit vorgeschlagenen absoluten linearen Einheiten und inversen polynomischen linearen Einheiten, integriert weiterhin Eigenschaften bestehender Funktionen, um deren Einschränkungen zu überwinden. Während diese neuen Funktionen vielversprechende Ergebnisse zeigen, beeinflussen ihre Rechenkosten und die einfache Integration in bestehende Frameworks oft ihre weitreichende Akzeptanz. [64]

4.1.2 Optimierungsmethoden: SGD, Adam

Trotz ihrer starken Leistung während des frühen Trainings können adaptive Optimierungsmethoden wie Adam, Adagrad und RMSprop in den späteren Trainingsphasen oft nicht mit der Generalisierungsfähigkeit von Stochastic Gradient Descent (SGD) mithalten. Während diese Methoden anfänglich die Konvergenz beschleunigen, werden sie in Bezug auf die endgültige Modellleistung tendenziell von SGD übertroffen. Um dies zu beheben, wurde ein hybrider Ansatz untersucht – beginnend mit einer adaptiven Methode und dem Wechsel zu SGD im richtigen Moment. Eine solche Strategie, SWATS (Switching from Adam to SGD), führt einen einfachen Mechanismus ein, der von Adam zu SGD übergeht, basierend auf einer Bedingung, die aus der Ausrichtung der Optimierungsschritte mit dem Gradientenunterraum abgeleitet wird. Dieser Wechsel verursacht minimalen zusätzlichen Rechenaufwand und erfordert keine zusätzlichen Hyperparameter. [68]

Stochastic Gradient Descent (SGD)

SGD ist ein grundlegender Optimierungsalgorithmus, der zum Training tiefer neuronaler Netze verwendet wird, indem die Verlustfunktion $f(w)$ iterativ minimiert wird. Die Regel lautet:

$$w_k = w_{k-1} - \eta_{k-1} \nabla f(w_{k-1}) \quad (4.5)$$

wobei η die Lernrate ist und ∇f den stochastischen Gradienten darstellt. SGD wird aufgrund seiner Einfachheit und theoretischen Garantien, wie der Vermeidung von Sattelpunkten und seiner Bayes'schen Interpretation, weit verbreitet eingesetzt. Seine gleichmäßige Skalierung des Gradienten über alle Parameter kann jedoch bei schlecht konditionierten Problemen zu einer langsamen Konvergenz führen. [69]

Momentumbasiertes SGD (SGDM) Momentum verbessert SGD durch die Einbeziehung eines Geschwindigkeitsterms v_t , der hilft, die Gradientenaktualisierungen zu glätten:

$$w_k = w_{k-1} - \eta_{k-1} v_k \quad (4.6)$$

wobei $\beta \in [0, 1)$ der Momentumkoeffizient ist. Diese Technik beschleunigt die Konvergenz durch Reduzierung von Oszillationen in der Optimierungstrajektorie.

Adaptive Methoden: Adam

Adam (Adaptive Moment Estimation) verbessert SGD, indem es die Lernraten pro Parameter unter Verwendung von Schätzungen der Gradientenmomente anpasst. Die Aktualisierungsregeln sind:

$$w_k = w_{k-1} - \eta_{k-1} \frac{m_{k-1}}{\sqrt{v_{k-1} + \epsilon}} \quad (4.7)$$

wobei β_1 und β_2 die exponentiellen Abklingraten steuern und ϵ eine kleine Konstante ist, um eine Division durch Null zu verhindern. Adam ist besonders effektiv für spärliche Gradienten und schlecht skalierte Probleme, kann jedoch in einigen Fällen aufgrund der nicht-uniformen Gradientenskalierung schlechter generalisieren als SGD. [68]

Hybride Ansätze

Um Adams schnellen initialen Fortschritt mit SGD's überlegener Generalisierung zu kombinieren, wechselt SWATS (Switching from Adam to SGD) automatisch von Adam zu SGD:

- Schätzung der Lernrate: Projiziert Adams Schritt auf den Gradientenunterraum, um eine SGD-kompatible Rate abzuleiten.
- Wechselkriterium: Löst aus, wenn sich die bias-korrigierte Adam-Rate stabilisiert.

Empirische Studien zeigen, dass SWATS Adams Effizienz in der Frühphase beibehält und gleichzeitig SGD's Endleistung erreicht. [68]

Praktische Einblicke

- SGD/SGDM zeichnet sich bei generalisierungskritischen Aufgaben aus, erfordert jedoch eine sorgfältige Abstimmung der Lernrate.
- Adam ist robust gegenüber Hyperparametern und ideal für frühe Trainingsphasen.
- Adaptive Methoden bieten Kompromisse zwischen Konvergenzgeschwindigkeit und Generalisierung. [62]

4.1.3 Regularisierung: Dropout und L2-Regularisierung

Regularisierungstechniken sind entscheidend, um Overfitting in tiefen neuronalen Netzen zu mildern, insbesondere wenn die Trainingsdaten begrenzt sind. Sie verbessern die Modellgeneralisation, insbesondere beim Training auf komplexen und hochdimensionalen Daten. Dieser Abschnitt untersucht zwei weit verbreitete Methoden: Dropout und L2-Regularisierung, die beide erheblich zur Verbesserung der Generalisierungsfähigkeit neuronaler Netze beitragen und es ihnen ermöglichen, auf ungesehenen Daten gut zu funktionieren. [70]

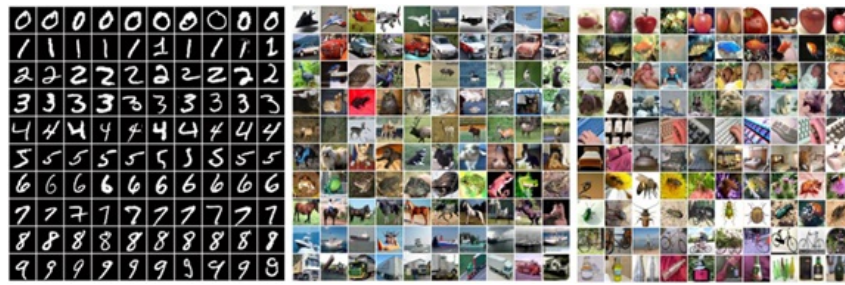


Abbildung 4.2: Beispielbilder aus den Datensätzen MNIST, CIFAR-10 und CIFAR-100. Jede Zeile zeigt Bilder einer bestimmten Kategorie. (Adaptiert aus „Towards Dropout Training for Convolutional Neural Networks“(2015)

Dropout: Stochastische Regularisierung

Dropout ist eine leistungsstarke Regularisierungstechnik, die Overfitting mindert, indem sie während des Trainings einen Bruchteil von Neuronen zufällig deaktiviert. Dies verhindert die Ko-Adaptation zwischen Merkmalen und trainiert effektiv ein Ensemble von Teilnetzwerken.

- **Mechanismus:** Während jeder Trainingsiteration setzt Dropout zufällig einen Bruchteil der Neuronenaktivierungen auf Null.
- **Effekt:** Diese stochastische Deaktivierung zwingt das Modell, robustere Merkmale zu lernen und verhindert, dass es sich zu stark auf ein einzelnes Neuron verlässt.
- **Anwendung:** Dropout ist besonders nützlich in großen neuronalen Netzen, wo das Risiko von Overfitting aufgrund einer übermäßigen Anzahl von Parametern hoch ist.
- **Trainingsphase:** Für jeden Mini-Batch werden Neuronen stochastisch "gedroppt", und die Aktivierungen der verbleibenden Neuronen werden mit $1/p$ (wobei p die Beibehaltungswahrscheinlichkeit ist) skaliert, um die erwarteten Größen der Netzwerkausgabe beizubehalten.
- **Inferenzphase:** Alle Neuronen bleiben aktiv, aber ihre Gewichte werden mit p (oder Aktivierungen mit $1/p$) skaliert, um die Modellmittelung zu approximieren, die charakteristisch für ein Ensemble ist. [70]

Varianten von Dropout

- **Max-Pooling Dropout:** Diese Variante wendet Dropout auf Pooling-Regionen an, was als Stichprobe aus einer multinomialen Verteilung interpretiert werden kann.
- **Probabilistic Weighted Pooling:** Diese Methode ersetzt deterministisches Max-Pooling zur Testzeit durch eine gewichtete Summe von Aktivierungen, was oft zu einer verbesserten Generalisierung führt.

Empirische Studien zeigen, dass Dropout Overfitting in vollständig verbundenen Schichten erheblich reduziert. Seine Wirksamkeit in Faltungsschichten kann jedoch aufgrund der inhärenten Parameterteilung und räumlichen Lokalität begrenzt sein. [33]

L2-Regularisierung: Gewichtszerfall

L2-Regularisierung, auch bekannt als Gewichtszerfall, ist eine weit verbreitete Regularisierungstechnik, die große Gewichte in einem neuronalen Netzwerk bestraft.

- **Mechanismus:** L2-Regularisierung fügt der Verlustfunktion einen Term hinzu, der auf der quadrierten Größe der Gewichte basiert. Die modifizierte Verlustfunktion ist gegeben durch:

$$\text{Loss}_{\text{new}} = \text{Loss}_{\text{original}} + \lambda \sum_i w_i^2 \quad (4.8)$$

wobei λ (Lambda) ein Hyperparameter ist, der die Stärke der Strafe steuert, und w_i die einzelnen Gewichte im Netzwerk darstellt.

- **Effekt:** Diese Strafe ermutigt das Modell, glattere, kleinere Gewichte zu lernen, wodurch die Modellkomplexität reduziert und die Generalisierung verbessert wird. Durch das Verhindern großer Gewichte verhindert die L2-Regularisierung, dass das Modell Rauschen in den Trainingsdaten anpasst, was zu stabileren Modellen führt. [33]

Vergleichende Analyse und Hybridansatz

Dropout und L2-Regularisierung sind komplementäre Techniken zur Bekämpfung von Overfitting, und ihre kombinierte Verwendung führt oft zu einer überlegenen Leistung.

- **Stärken von Dropout:** Dropout zeichnet sich in Hochkapazitätsnetzwerken aus und erzielt mit Benchmarks wie MNIST und CIFAR-10 modernste Ergebnisse, hauptsächlich durch die Verhinderung von Merkmalsko-Adaptation und die Schaffung eines Ensemble-Effekts.
- **Stärken der L2-Regularisierung:** L2-Regularisierung ist einfacher zu implementieren und recheneffizienter. Obwohl sie allein für sehr tiefe Netze weniger effektiv ist, bietet sie eine stabile und interpretierbare Strafe, indem sie die Gewichtsgrößen begrenzt.
- **Hybridansatz:** Die Kombination beider Methoden führt oft zu einer überlegenen Leistung. L2-Regularisierung begrenzt die Gewichtsgrößen, während Dropout die Merkmalsko-Adaptation verhindert, wodurch ein robustes und generalisiertes Modell entsteht. [63]

Praktische Überlegungen

Bei der Implementierung von Dropout und L2-Regularisierung sollten mehrere praktische Aspekte berücksichtigt werden:

- **Schichtspezifische Anwendung:** Dropout ist am effektivsten in vollständig verbundenen Schichten, während Faltungsschichten aufgrund ihrer inhärenten räumlichen Lokalität und Parameterteilung weniger profitieren.
- **Hyperparameter-Abstimmung:** Die Beibehaltungswahrscheinlichkeit p für Dropout und die Regularisierungsstärke λ für die L2-Regularisierung sind entscheidende Hyperparameter, die für die besten Ergebnisse durch Techniken wie Kreuzvalidierung sorgfältig optimiert werden müssen.
- **Berechnungsaufwand:** Dropout erhöht zwar die Trainingszeit aufgrund der stochastischen Natur seiner Funktionsweise, hat aber keinen signifikanten Einfluss auf die Inferenzkosten, sobald das Modell trainiert ist. [67]

4.2 Leistungsbewertung von Netzwerkstrategien

Die Leistungsfähigkeit künstlicher neuronaler Netze sowie anderer Modelle des maschinellen Lernens hängt nicht nur von der Qualität des Trainingsprozesses, sondern ebenso von der Art der Evaluation und Validierung ab. Eine sorgfältige Bewertungsstrategie liefert nicht nur ein objektives Urteil über die Leistung eines Modells, sondern hilft auch dabei, Schwächen wie Overfitting, Underfitting oder Verzerrungen frühzeitig zu erkennen. Zwei zentrale Aspekte sind die Auswahl geeigneter Metriken zur Modellevaluierung und der Einsatz robuster Verfahren zur Kreuzvalidierung und zur Kontrolle von Überanpassung.

4.2.1 Modellevaluierungsmetriken

Bewertungsmetriken dienen der quantitativen Einschätzung der Modellgüte. Je nach Art der Aufgabe (Regression oder Klassifikation) kommen unterschiedliche Metriken zum Einsatz. [71]

Mittlerer quadratischer Fehler (MSE)

Der mittlere quadratische Fehler (Mean Squared Error, MSE) ist eine gängige Verlustfunktion bei Regressionsproblemen. Er misst den Durchschnitt der quadrierten Abweichungen zwischen den vorhergesagten und den tatsächlichen Werten:

$$\text{MSE}_{\text{test}} = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i^{(\text{test})} - y_i^{(\text{test})})^2. \quad (4.9)$$

Ein niedriger MSE-Wert deutet auf eine hohe Genauigkeit des Modells hin.

Bestimmtheitsmaß (R^2)

Das Bestimmtheitsmaß R^2 (R-squared) gibt an, welcher Anteil der Varianz der Zielgröße durch das Modell erklärt wird. Ein Wert von 1 entspricht einer perfekten Modellanpassung, während ein Wert von 0 keine erklärte Varianz bedeutet:

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2} \quad (4.10)$$

Genauigkeit (Accuracy)

Genauigkeit ist eine Metrik für Klassifikationsaufgaben und gibt den Anteil korrekt klassifizierter Instanzen an. Sie ist leicht zu interpretieren, kann jedoch bei unausgewogenen Datensätzen irreführend sein.

Manchmal möchte man einen binären Classifier trainieren, der ein bestimmtes Ereignis erkennen soll. Zum Beispiel könnte man einen medizinischen Test für eine seltene Krankheit entwickeln. Wenn man annimmt, dass nur einer unter einer Million Menschen diese Krankheit hat: Wird ein Modell mit einer Genauigkeit von 99,9999 Prozent bei der Erkennung erreicht, indem wir den Klassifikator einfach so programmieren, dass er immer meldet, dass die Krankheit nicht da ist. Es liegt auf der Hand, dass sich die Leistung eines solchen Systems nur schlecht mit der Genauigkeit beschreiben lässt. [71]

Präzision und Sensitivität (Recall)

Eine Möglichkeit, dieses Problem zu lösen, besteht darin, stattdessen Präzision und Sensitivität zu messen. Die Präzision ist der Anteil der vom Modell gemeldeten Erkennungen, die korrekt waren, während Recall der Anteil der wahren Ergebnisse ist, die erkannt wurden. Ein Detektor, der sagt, dass jemand die Krankheit hat, würde eine perfekte Präzision erreichen, aber eine Sensitivität von null. Ein Detektor, der sagt, dass jeder die Krankheit hat, würde eine Sensitivität von 1 haben, aber eine Präzision, die dem Prozentsatz der Menschen entspricht, die die Krankheit haben (0,0001 Prozent in unserem Beispiel einer Krankheit, die nur einer von einer Million Menschen hat). [71]

4.2.2 Techniken zur Kreuzvalidierung und Überanpassungskontrolle

Ein zentrales Ziel des maschinellen Lernens ist die Entwicklung von Modellen, die auf neuen, bislang unbekannten Daten verlässlich funktionieren. Um dies zu gewährleisten, sind Verfahren zur Validierung und zur Kontrolle von Überanpassung (Overfitting) notwendig. [11]

Kreuzvalidierung

Bei der Kreuzvalidierung (engl. cross validation) versucht man, die Modellkomplexität so zu optimieren, dass der Klassifikations- oder Approximationsfehler auf einem beim Lernen unbekannten Testdatensatz minimal wird. Dazu muss die Modellkomplexität über einen Parameter v steuerbar und verlässlich bewertbar sein. [11]

- **K-fache Kreuzvalidierung (k-fold cross-validation):** Der Datensatz wird in k gleich große Teile unterteilt. In jedem Durchlauf wird einer dieser Teile als Validierungsmenge genutzt, während die restlichen $k - 1$ Teile zum Training dienen. Die mittlere Leistung über alle Durchläufe liefert eine verlässige Bewertung.
- **Leave-One-Out-Cross-Validation (LOOCV):** Spezialfall der k-fachen Kreuzvalidierung mit $k = n$ (Anzahl der Datenpunkte). Sehr präzise, aber rechenintensiv.

Die Kreuzvalidierung ist besonders nützlich bei kleinen Datensätzen und wenn eine möglichst objektive Bewertung erforderlich ist. [11]

Kontrolle von Überanpassung

Wenn ein Algorithmus für maschinelles Lernen verwendet wird, werden die Parameter natürlich nicht festgelegt, bevor beide Datensätze abgetastet werden. Der Trainingssatz wird abgetastet, dann werden die Parameter gewählt, um den Fehler des Trainingssatzes zu reduzieren, und dann wird der Testsatz abgetastet. Bei diesem Verfahren ist der erwartete Testfehler größer oder gleich dem erwarteten Wert des Trainingsfehlers. Dies bedeutet, dass, wenn es gut ein Algorithmus für maschinelles Lernen funktioniert, sind seine Fähigkeit:

1. Den Trainingsfehler klein zu halten.
2. Die Lücke zwischen Trainings- und Testfehler klein zu halten.

Diese beiden Faktoren entsprechen den beiden zentralen Herausforderungen des maschinellen Lernens: Underfitting und Overfitting. Underfitting tritt auf, wenn das Modell nicht in der Lage ist, einen ausreichenden Fehlerabfall in der Trainingsmenge zu erreichen. Overfitting hingegen tritt auf, wenn der Unterschied zwischen dem Trainingsfehler und dem Testfehler zu groß ist. [11]

Kapitel 5

Neuronale Netze Programmierung in MATLAB

- **Regressionsnetzwerk:** Ein Netz zur numerischen Vorhersage. In diesem Fall wird die nächste Zahl in einer Zahlenreihe vorhergesagt, basierend auf den fünf vorhergehenden Werten.
- **Klassifikationsnetzwerk:** Ein neuronales Netz, das kategorische Ausgaben erzeugt. Ziel ist es, Eingaben korrekt einer Klasse zuzuordnen.
- **Regressionsnetzwerk mit Bildanalyse:** Dieses Netz kombiniert klassische Regressionsaufgaben mit Bildverarbeitung, wobei synthetische Bilddaten verarbeitet werden. Die Vorverarbeitung umfasst eine eigenständig programmierte Kantenerkennung.
- **Neuronales Netz zur Ziffernerkennung:** Ein einfaches Netz, das visuelle Eingaben (z. B. handgeschriebene Ziffern) analysiert und klassifiziert.
- **Kantenerkennungsalgorithmus:** Zusätzlich wurde ein algorithmischer Ansatz zur *Kantenerkennung* integriert, der Bilder auf Kantenstrukturen untersucht. Obwohl dieser Algorithmus auch als Vorstufe eines Convolutional Neural Networks (CNN) dienen kann, wurde hier bewusst nur ein konventioneller Filteransatz umgesetzt und kein vollständiges CNN implementiert.

5.1 Regressions-Neuronales Netz

In diesem Code wird ein einfaches künstliches neuronales Netz implementiert, das die nächste Zahl in einer Sequenz vorhersagen soll. Ziel ist es, eine Regressionsaufgabe zu lösen: Aus fünf aufeinanderfolgenden Zahlen wird die sechste Zahl prognostiziert.

Dazu erzeugen wir zunächst ein künstliches Datenset mit Ganzzahlen von 1 bis 100. Jede Eingabe besteht aus fünf aufeinanderfolgenden Zahlen, z. B. [1, 2, 3, 4, 5], und das Ziel ist jeweils die darauffolgende Zahl (z. B. 6). Insgesamt entstehen dadurch 95 Trainingsbeispiele.

Die Daten werden anschließend mit der Min-Max-Normalisierung auf den Bereich [0, 1] skaliert. Diese Vorverarbeitung unterstützt insbesondere die ReLU-Aktivierungsfunktion, da diese nur positive Werte verarbeitet.

Das Netzwerk besteht aus:

- einer Eingangsschicht mit 5 Neuronen,
- einer versteckten Schicht mit 10 Neuronen (ReLU-Aktivierung),
- sowie einer linearen Ausgabeschicht mit einem Neuron (für die Vorhersage der sechsten Zahl).

Alle Gewichtsmatrizen und Bias-Vektoren werden manuell initialisiert. Der Trainingsprozess erfolgt mithilfe der klassischen Backpropagation und dem mittleren quadratischen Fehler (MSE) als Verlustfunktion. Die Parameter werden iterativ mit einer festen Lernrate aktualisiert.

Ein wesentliches Merkmal dieses Codes ist, dass **keine externen Machine-Learning-Bibliotheken** (wie TensorFlow, PyTorch oder ähnliche) verwendet werden. Stattdessen erfolgt die komplette Modellierung, Vorwärts- und Rückwärtsausbreitung sowie das Training ausschließlich mithilfe mathematischer Operationen und eigener Implementationen.

Am Ende wird das trainierte Modell auf neue Eingabedaten getestet. Die Vorhersagen werden zurücktransformiert (denormalisiert) und mit den tatsächlichen Zielwerten verglichen, um die Genauigkeit des Modells zu evaluieren. Die Resultate werden sowohl tabellarisch als auch grafisch dargestellt.

Test-Nr	Input-Werte	Erwartet	Vorhersage	Absoluter Fehler
1	[96 97 98 99 100]	101.00	101.06	0.06
2	[97 98 99 100 101]	102.00	102.06	0.06
3	[98 99 100 101 102]	103.00	103.06	0.06
4	[99 100 101 102 103]	104.00	104.06	0.06
5	[100 101 102 103 104]	105.00	105.06	0.06
6	[101 102 103 104 105]	106.00	106.07	0.07
7	[102 103 104 105 106]	107.00	107.07	0.07
8	[103 104 105 106 107]	108.00	108.07	0.07
9	[104 105 106 107 108]	109.00	109.07	0.07
10	[105 106 107 108 109]	110.00	110.07	0.07

Tabelle 5.1: Neural network prediction results

Die Tabelle 5.1 zeigt die Ergebnisse des trainierten Regressionsnetzwerks auf zuvor ungesehenen Testdaten. Dabei wurden jeweils fünf aufeinanderfolgende Zahlen als Eingabe verwendet, um die darauffolgende sechste Zahl vorherzusagen.

Die Vorhersagen stimmen sehr gut mit den tatsächlichen Zielwerten überein. Der absolute Fehler pro Testfall beträgt lediglich zwischen **0,06** und **0,07**, was auf eine präzise Modellanpassung hinweist. Dieses Ergebnis bestätigt, dass das Netzwerk erfolgreich gelernt hat, den linearen Zusammenhang in der Zahlenreihe zu erfassen und korrekt zu generalisieren – trotz der Tatsache, dass keine externen Machine-Learning-Bibliotheken verwendet wurden.

Besonders hervorzuheben ist, dass auch Werte außerhalb des ursprünglichen Trainingsbereichs (*Extrapolation*) mit vergleichbarer Genauigkeit vorhergesagt werden, was ein Indikator für die Robustheit des Modells ist.

5.1.1 Datenerzeugung und Normalisierung

Dieser Teil des Codes generiert die Trainingsdaten und normalisiert sie in den Bereich von 0 bis 1. Die Normalisierung verbessert die Stabilität und Konvergenzgeschwindigkeit des neuronalen Netzwerks während des Trainings.

Mathematische Beschreibung:

- Für jedes $i \in \{1, 2, \dots, 95\}$ wird ein Eingabevektor $X_i = [i, i + 1, i + 2, i + 3, i + 4]$ erstellt. Damit ergibt sich eine Eingabematrix $X \in \mathbb{R}^{95 \times 5}$:

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 95 & 96 & 97 & 98 & 99 \end{bmatrix} \quad (5.1)$$

- Der Zielwert Y_i entspricht dem nächsten Wert in der Sequenz, also $Y_i = i + 5$. Daraus ergibt sich ein Zielvektor $Y \in \mathbb{R}^{95 \times 1}$:

$$Y = \begin{bmatrix} 6 \\ 7 \\ \vdots \\ 100 \end{bmatrix} \quad (5.2)$$

- Die Normalisierung erfolgt nach der Min-Max-Methode, sodass alle Werte im Bereich $[0, 1]$ liegen. Für jedes Element x in X gilt:

$$x_{\text{norm}} = \frac{x - \min(X)}{\max(X) - \min(X)} \quad (5.3)$$

- Entsprechend wird auch der Zielvektor Y normalisiert:

$$y_{\text{norm}} = \frac{y - \min(Y)}{\max(Y) - \min(Y)} \quad (5.4)$$

Code-Ausschnitt:

```
1 clc; clear;
2
3 % --- DATENERZEUGUNG UND NORMALISIERUNG ---
4
5 X = zeros(95, 5);
6 Y = zeros(95, 1);
7
8 for i = 1:95
9     X(i, :) = i:i+4;
10    Y(i) = i + 5;
11 end
12
13 % --- NORMALISIERUNG DER DATEN ---
14
15 X_min = min(X(:));
```

```

16 X_max = max(X(:));
17 X = (X - X_min) / (X_max - X_min);
18
19 Y_min = min(Y);
20 Y_max = max(Y);
21 Y = (Y - Y_min) / (Y_max - Y_min);
    
```

Erläuterung:

- `clc; clear;` löscht das Command Window und entfernt alle Variablen aus dem Workspace.
- `X = zeros(95, 5);` und `Y = zeros(95, 1);` initialisieren Matrizen für die Eingabedaten (95 Beispiele mit 5 Merkmalen jeweils) und die Ausgabedaten (95 Zielwerte).
- Die `for`-Schleife generiert die Trainingsdaten, wobei jede Eingabe eine Sequenz von fünf aufeinanderfolgenden Zahlen ist und das Ziel die nächste Zahl in der Sequenz.
- Die folgenden Zeilen berechnen das Minimum und Maximum der Ein- und Ausgabedaten und normalisieren sie dann in den Bereich $[0, 1]$ mittels Min-Max-Normalisierung.

5.1.2 Netzwerkarchitektur und Initialisierung

Hier wird die Struktur des neuronalen Netzwerks definiert und die Gewichte sowie Bias-Werte initialisiert.

Mathematische Beschreibung:

- Die Architektur des neuronalen Netzwerks ist gegeben durch:

$$\text{layers} = [5, 10, 1]$$

Das bedeutet:

- Eingabeschicht: 5 Neuronen
- Eine versteckte Schicht: 10 Neuronen
- Ausgabeschicht: 1 Neuron
- Für jede Schichtverbindung i (von $i = 1$ bis $i = \text{num_layers} = 2$) werden die Gewichtsmatrizen $W^{(i)}$ und Bias-Vektoren $b^{(i)}$ wie folgt initialisiert:

$$W^{(i)} \sim \mathcal{N}(0, 0.1^2) \in \mathbb{R}^{n_i \times n_{i+1}} \quad (5.5)$$

$$b^{(i)} = \mathbf{0} \in \mathbb{R}^{1 \times n_{i+1}} \quad (5.6)$$

wobei n_i die Anzahl der Neuronen in der i -ten Schicht ist. Dies bedeutet konkret:

- Für $i = 1$: $W^{(1)} \in \mathbb{R}^{5 \times 10}$, $b^{(1)} \in \mathbb{R}^{1 \times 10}$
- Für $i = 2$: $W^{(2)} \in \mathbb{R}^{10 \times 1}$, $b^{(2)} \in \mathbb{R}^{1 \times 1}$

- Die Initialisierung mit kleinen Zufallswerten (normalverteilt mit $\mu = 0$ und $\sigma = 0,1$) unterstützt den Lernprozess und verhindert Symmetrieprobleme beim Backpropagation-Algorithmus.

Lineare Regression als Spezialfall eines neuronalen Netzwerks:

Eine einfache lineare Regression mit einem Eingabevektor $x \in \mathbb{R}^n$ und einem Skalarziel $y \in \mathbb{R}$ kann durch ein neuronales Netzwerk mit nur **einer Schicht ohne Aktivierungsfunktion** dargestellt werden:

$$y = f(x) = xW + b \quad (5.7)$$

Dabei gilt:

- $W \in \mathbb{R}^{n \times 1}$ ist ein Gewichtsvektor.
- $b \in \mathbb{R}$ ist der Bias-Term.
- Es wird **keine Aktivierungsfunktion** verwendet oder äquivalent die **lineare Aktivierung**:

$$\phi(z) = z \quad (5.8)$$

Somit entspricht eine lineare Regression exakt einer **einlagigen neuronalen Netzwerkschicht mit linearer Aktivierung**.

Fazit: Eine Regression ist ein Spezialfall eines neuronalen Netzwerks ohne nichtlineare Aktivierung. Erst durch zusätzliche Schichten oder Aktivierungen wie ReLU, Sigmoid oder Tanh erhält das Netzwerk die Fähigkeit, nichtlineare Zusammenhänge zu modellieren.

Code-Ausschnitt:

```

1 % --- NETZWERKARCHITEKTUR UND INITIALISIERUNG ---
2
3 layers = [5, 10, 1];
4 num_layers = length(layers) - 1;
5
6 W = cell(num_layers, 1);
7 b = cell(num_layers, 1);
8
9 for i = 1:num_layers
10     W{i} = randn(layers(i), layers(i+1)) * 0.1;
11     b{i} = zeros(1, layers(i+1));
12 end
    
```

Erläuterung:

- `layers = [5, 10, 1];` definiert die Anzahl der Neuronen in jeder Schicht: 5 in der Eingabeschicht, 10 in der versteckten Schicht und 1 in der Ausgabeschicht.
- `num_layers = length(layers) - 1;` berechnet die Anzahl der Gewichtsschichten.
- `W = cell(num_layers, 1);` und `b = cell(num_layers, 1);` erstellen Zell-Arrays zur Speicherung der Gewichtsmatrizen und Bias-Vektoren für jede Schicht.
- Die `for`-Schleife initialisiert die Gewichte zufällig mit einer kleinen Varianz (multipliziert mit 0.1) und setzt die Bias-Werte auf Null.

5.1.3 Aktivierungsfunktion

Dieser Abschnitt definiert die ReLU-Aktivierungsfunktion und ihre Ableitung und visualisiert die Funktion.

Mathematische Beschreibung:

Die **ReLU-Funktion** (Rectified Linear Unit) ist eine der am häufigsten verwendeten Aktivierungsfunktionen in neuronalen Netzwerken. Sie ist definiert als:

$$\text{ReLU}(x) = \max(0, x) \quad (5.9)$$

Das bedeutet:

- Für $x \leq 0$ ist $\text{ReLU}(x) = 0$
- Für $x > 0$ ist $\text{ReLU}(x) = x$

Die Ableitung der ReLU-Funktion, welche im Backpropagation-Verfahren verwendet wird, lautet:

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 0 & \text{wenn } x \leq 0 \\ 1 & \text{wenn } x > 0 \end{cases} \quad (5.10)$$

Vorteile von ReLU:

- Einfach und effizient zu berechnen
- Vermeidet das Vanishing-Gradient-Problem im Vergleich zu Sigmoid- oder Tanh-Funktionen
- Einführung von Nichtlinearität, wodurch das Netzwerk komplexe Abbildungen lernen kann

Code-Ausschnitt:

```
1 % --- AKTIVIERUNGSFUNKTIONEN ---
2
3 act = @(x) max(0, x);
4 dact = @(x) double(x > 0);
5
6 x_vals = -5:0.01:5;
7 y_vals = act(x_vals);
8 figure;
9 plot(x_vals, y_vals, 'LineWidth', 2);
10 xlabel('x');
11 ylabel('ReLU(x)');
12 title('ReLU-Aktivierungsfunktion');
13 grid on;
```

Erläuterung:

- `act = @(x) max(0, x);` definiert die ReLU-Funktion als anonyme Funktion.
- `dact = @(x) double(x > 0);` definiert die Ableitung der ReLU-Funktion.
- Der folgende Code generiert Werte für x und $y = \text{ReLU}(x)$ und plottet die Funktion zur Visualisierung.

5.1.4 Hyperparameter

Hier werden die Hyperparameter für das Training des Netzwerks festgelegt.

Mathematische Beschreibung:

Hauptsächlich werden zwei zentrale Hyperparameter definiert:

- **Anzahl der Trainingszyklen (Epochen):**

$$\text{epochs} = 2000 \quad (5.11)$$

Das Netzwerk wird 2000-mal über den gesamten Trainingsdatensatz iterieren, um die Gewichte und Bias-Werte zu optimieren. Eine zu niedrige Anzahl von Epochen kann zu einem untertrainierten Modell führen, während eine zu hohe Anzahl zu Überanpassung (Overfitting) führen kann.

- **Lernrate (Learning Rate):**

$$\text{lr} = 0,01 \quad (5.12)$$

Die Lernrate steuert die Größe der Schritte, die das Netzwerk bei der Gewichtsaktualisierung macht. Diese Aktualisierung basiert auf dem Gradientenabstieg (Gradient Descent):

$$W^{(i)} \leftarrow W^{(i)} - \eta \cdot \nabla_{W^{(i)}} \mathcal{L} \quad (5.13)$$

$$b^{(i)} \leftarrow b^{(i)} - \eta \cdot \nabla_{b^{(i)}} \mathcal{L} \quad (5.14)$$

wobei:

- η die Lernrate ist ($\eta = \text{lr} = 0,01$)
- \mathcal{L} die Verlustfunktion (Loss Function)
- $\nabla_{W^{(i)}} \mathcal{L}$ der Gradient der Verlustfunktion in Bezug auf die Gewichte

Eine zu hohe Lernrate kann zu Divergenz führen, während eine zu niedrige Lernrate den Lernprozess extrem verlangsamt.

Code-Ausschnitt:

```
1 % --- HYPERPARAMETER ---
2
3 epochs = 2000;
4 lr = 0.01;
```

Erläuterung:

- `epochs = 2000;` setzt die Anzahl der Trainingsiterationen (Epochen) auf 2000.
- `lr = 0.01;` definiert die Lernrate als 0.01, die die Größe der Gewichtsadjustierungen während des Trainings bestimmt.

5.1.5 Trainingsprozess (Forward und Backward Pass)

Dieser Abschnitt beschreibt den mathematischen Hintergrund des Trainingsalgorithmus eines einfachen Feedforward-Neuronalen Netzwerks.

Forward Pass

Im **Forward Pass** werden die Voraktivierungen $Z^{(i)}$ und die Aktivierungen $A^{(i)}$ für jede Schicht berechnet. Für jede Schicht $i \in \{1, \dots, L\}$ gilt:

$$Z^{(i)} = A^{(i-1)} W^{(i)} + b^{(i)} \quad (5.15)$$

$$A^{(i)} = \begin{cases} \text{ReLU}(Z^{(i)}) & \text{für } i < L \\ Z^{(i)} & \text{für die Ausgangsschicht} \end{cases} \quad (5.16)$$

Dabei ist $A^{(0)} = X$ die Eingabematrix.

Fehlermessung

Der Fehler (Loss) wird als mittlerer quadratischer Fehler (MSE) zwischen der Vorhersage $A^{(L)}$ und dem Zielwert Y berechnet:

$$\mathcal{L} = \frac{1}{N} \sum_{j=1}^N (A_j^{(L)} - Y_j)^2 \quad (5.17)$$

Backward Pass

Der **Backward Pass** berechnet die Gradienten durch Rückpropagation. Beginnend mit der Ableitung des Fehlers:

$$\frac{\partial \mathcal{L}}{\partial A^{(L)}} = \frac{2}{N}(A^{(L)} - Y) \quad (5.18)$$

Für jede Schicht $i = L, L-1, \dots, 1$ gilt:

$$\delta^{(i)} = \begin{cases} \frac{\partial \mathcal{L}}{\partial A^{(L)}} & \text{wenn } i = L \\ (\delta^{(i+1)} W^{(i+1)T}) \circ \text{ReLU}'(Z^{(i)}) & \text{sonst} \end{cases} \quad (5.19)$$

Die Gradienten der Gewichtsmatrix und der Bias-Vektoren berechnen sich wie folgt:

$$\frac{\partial \mathcal{L}}{\partial W^{(i)}} = A^{(i-1)T} \delta^{(i)} \quad (5.20)$$

$$\frac{\partial \mathcal{L}}{\partial b^{(i)}} = \sum_{j=1}^N \delta_j^{(i)} \quad (5.21)$$

Parameteraktualisierung

Die Gewichte und Biases werden mit Lernrate η wie folgt aktualisiert:

$$W^{(i)} \leftarrow W^{(i)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(i)}} \quad (5.22)$$

$$b^{(i)} \leftarrow b^{(i)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(i)}} \quad (5.23)$$

Dies entspricht genau dem in MATLAB implementierten Trainingsloop mit ReLU-Aktivierung und MSE-Fehlermetrik.

Code-Ausschnitt:

```

1 % --- TRAININGSLoop ---
2 for epoch = 1:epochs
3     A = cell(num_layers+1,1);
4     Z = cell(num_layers,1);
5     A{1} = X;
6
7     for i = 1:num_layers
8         Z{i} = A{i} * W{i} + b{i};
9         if i < num_layers
10            A{i+1} = act(Z{i});
11        else
12            A{i+1} = Z{i};
13        end
14    end
15
16    loss = mean((A{end} - Y).^2);
17
18    dA = 2 * (A{end} - Y) / size(X,1);
19
20    for i = num_layers:-1:1
21        dZ = dA;
22        if i < num_layers
23            dZ = dZ .* dact(Z{i});
24        end
25        dW = A{i}' * dZ;
26        db = sum(dZ, 1);
    
```

```

27     dA = dZ * W{i}' ;
28
29     W{i} = W{i} - lr * dW;
30     b{i} = b{i} - lr * db;
31 end
32
33 if mod(epoch, 100) == 0
34     fprintf("Epoche %d - Fehler: %.6f\n", epoch, loss);
35 end
36 end
    
```

Erläuterung:

- Die äußere `for`-Schleife iteriert über die Anzahl der Epochen.
- Innerhalb jeder Epoche werden die Zell-Arrays `A` (für Aktivierungen) und `Z` (für gewichtete Summen) initialisiert. Die Eingabeschicht wird mit den Trainingsdaten `X` gefüllt.
- Die erste innere `for`-Schleife führt den Forward Pass durch, berechnet die gewichteten Summen und Aktivierungen für jede Schicht. In der Ausgabeschicht wird keine Aktivierungsfunktion verwendet.
- Der Fehler (mittlerer quadratischer Fehler, MSE) wird berechnet.
- Der Backward Pass (zweite innere `for`-Schleife) berechnet die Gradienten des Fehlers in Bezug auf die Gewichte und Biases und aktualisiert sie mittels Gradientenabstieg.
- Alle 100 Epochen wird der aktuelle Fehler ausgegeben.

5.1.6 Test des Netzwerks mit einem Beispiel

Nach dem Training wird das neuronale Netzwerk mit einem neuen Eingabebeispiel getestet. Es werden sowohl die Vorhersage im normalisierten Raum als auch die Rücktransformation in den ursprünglichen Wertebereich berechnet.

Normalisierung der Eingabedaten

Die Eingabedaten werden analog zur Trainingsphase normalisiert:

$$\hat{x} = \frac{x_{\text{test}} - X_{\min}}{X_{\max} - X_{\min}} \quad (5.24)$$

Dabei ist $x_{\text{test}} \in \mathbb{R}^n$ ein neues Eingabebeispiel, das auf den Bereich $[0, 1]$ skaliert wird.

Vorwärtspassage durch das Netzwerk

Der normalisierte Eingabewert \hat{x} wird durch das trainierte Netzwerk weitergeleitet. Für jede Schicht $i = 1, \dots, L - 1$:

$$\hat{x} \leftarrow \text{ReLU}(\hat{x} \cdot W^{(i)} + b^{(i)}) \quad (5.25)$$

Für die letzte Schicht $i = L$, die keine Aktivierungsfunktion verwendet:

$$\hat{y} = \hat{x} \cdot W^{(L)} + b^{(L)} \quad (5.26)$$

Denormalisierung der Ausgabe

Um die Vorhersage mit den realen Zielwerten zu vergleichen, wird die Ausgabe wieder in den Originalmaßstab zurücktransformiert:

$$y_{\text{pred}} = \hat{y} \cdot (Y_{\text{max}} - Y_{\text{min}}) + Y_{\text{min}} \quad (5.27)$$

Interpretation

Die Ausgabe \hat{y} ist die normalisierte Vorhersage, während y_{pred} der reale, interpretierbare Vorhersagewert ist. Durch Vergleich mit dem tatsächlichen Wert kann die Genauigkeit der Vorhersage überprüft werden.

Code-Ausschnitt:

```
1 % --- TESTEN DES NETZES MIT EINEM BEISPIEL ---
2
3 x_test = ([96 97 98 99 100] - X_min) / (X_max - X_min);
4
5 for i = 1:num_layers-1
6     x_test = act(x_test * W{i} + b{i});
7 end
8
9 y_pred = x_test * W{end} + b{end};
10 y_pred_orig = y_pred * (Y_max - Y_min) + Y_min;
11 fprintf('Vorhersage (normalisiert): %.4f   Tatsächlicher Wert: %.1f\n', y_pred,
        y_pred_orig);
```

Erläuterung:

- Ein Testbeispiel wird definiert und mit den gleichen Min-Max-Werten wie die Trainingsdaten normalisiert.
- Ein Forward Pass durch das Netzwerk wird durchgeführt, um die normalisierte Vorhersage zu erhalten.
- Die vorhergesagte Ausgabe wird denormalisiert, um den Wert in der ursprünglichen Skala zu erhalten.
- Die normalisierte und denormalisierte Vorhersage wird ausgegeben.

5.1.7 Test des Netzwerks mit mehreren Beispielen

Zur Beurteilung der Generalisierungsfähigkeit des Netzwerks wird dieses mit mehreren Testbeispielen geprüft. Dabei werden sowohl die Eingaben als auch die erwarteten Ausgaben systematisch erzeugt und mit den Vorhersagen verglichen.

Erstellung der Testdaten

Die Testeingaben bestehen aus gleitenden Fenstersegmenten der Form:

$$X_{\text{test}}^{(i)} = [x_i, x_{i+1}, x_{i+2}, x_{i+3}, x_{i+4}] \quad (5.28)$$

mit den zugehörigen Zielwerten:

$$Y_{\text{true}}^{(i)} = x_{i+5} \quad (5.29)$$

Für $i = 96, \dots, 105$ ergibt sich so ein Testdatensatz mit 10 Beispielen der Form:

$$X_{\text{test}} = \begin{bmatrix} 96 & 97 & 98 & 99 & 100 \\ 97 & 98 & 99 & 100 & 101 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 105 & 106 & 107 & 108 & 109 \end{bmatrix}, \quad Y_{\text{true}} = \begin{bmatrix} 101 \\ 102 \\ \vdots \\ 110 \end{bmatrix}$$

Normalisierung

Analog zum Training erfolgt eine Merkmalsnormalisierung der Eingabedaten:

$$\hat{X}_{\text{test}} = \frac{X_{\text{test}} - X_{\min}}{X_{\max} - X_{\min}} \quad (5.30)$$

Vorhersage durch das Netzwerk

Jede normalisierte Testzeile $\hat{x}^{(i)}$ wird durch das Netzwerk propagiert:

$$\text{für } i = 1, \dots, L - 1 : \quad \hat{x}^{(i+1)} = \text{ReLU}(\hat{x}^{(i)} \cdot W^{(i)} + b^{(i)}) \quad (5.31)$$

$$\text{für } i = L : \quad \hat{y}^{(i)} = \hat{x}^{(L)} \cdot W^{(L)} + b^{(L)} \quad (5.32)$$

Rücktransformation der Vorhersage

Die normalisierte Vorhersage wird anschließend zurück in den ursprünglichen Wertebereich transformiert:

$$y_{\text{pred}} = \hat{y} \cdot (Y_{\max} - Y_{\min}) + Y_{\min} \quad (5.33)$$

Fehlerberechnung

Für jede Vorhersage wird der absolute Fehler bestimmt:

$$\varepsilon_i = \left| y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)} \right| \quad (5.34)$$

Visualisierung

Zur qualitativen Analyse werden die tatsächlichen Zielwerte y_{true} und die Modellvorhersagen y_{pred} grafisch gegenübergestellt. So lässt sich visuell erkennen, wie gut das Modell generalisiert:

Code-Ausschnitt:

```
1 % --- TESTEN DES NETZES MIT MEHREREN BEISPIELEN ---
2
3 X_test = zeros(10, 5);
4 Y_true = zeros(10, 1);
5
6 for i = 96:105
7     idx = i - 95;
8     X_test(idx,:) = i:i+4;
9     Y_true(idx) = i + 5;
10 end
11
12 % Testdaten normalisieren
13 X_test_norm = (X_test - X_min) / (X_max - X_min);
14
15 Y_pred = zeros(10, 1);
16
17 for n = 1:size(X_test_norm, 1)
```

```

18     x = X_test_norm(n, :);
19     for i = 1:num_layers-1
20         x = act(x * W{i} + b{i});
21     end
22     Y_pred(n) = x * W{end} + b{end};
23 end
24
25 % Rücktransformation der Vorhersagen
26 Y_pred = Y_pred * (Y_max - Y_min) + Y_min;
27
28 % Fehleranzeige mit Input-Werten
29 disp('--- TESTERGEBNISSE ---');
30 fprintf('%6s %20s %12s %12s %12s\n', 'Test-Nr', 'Input-Werte', 'Erwartet', '
    Vorhersage', 'Absoluter Fehler');
31 for i = 1:10
32     fehler = abs(Y_true(i) - Y_pred(i));
33     input_str = sprintf('%d %d %d %d %d', X_test(i,1), X_test(i,2), X_test(i,3),
    X_test(i,4), X_test(i,5));
34     fprintf('%6d %20s %12.2f %12.2f %12.2f\n', i, input_str, Y_true(i), Y_pred(i)
    ), fehler);
35 end
36
37 % Visualisierung der Ergebnisse
38 figure;
39 plot(1:10, Y_true, '-o', 'LineWidth', 2);
40 hold on;
41 plot(1:10, Y_pred, '-x', 'LineWidth', 2);
42 legend('Erwartet (True)', 'Vorhersage (Predicted)');
43 xlabel('Test-Nr. ');
44 ylabel('Zielwert');
45 title('Modellvorhersage vs. Tatsächlicher Wert');
46 grid on;
    
```

Erläuterung:

- Test-Eingabedaten und die entsprechenden wahren Ausgabewerte für zehn aufeinanderfolgende Sequenzen werden generiert.
- Die Test-Eingabedaten werden normalisiert.
- Für jedes Testbeispiel wird ein Forward Pass durch das Netzwerk durchgeführt, um die vorhergesagte Ausgabe zu erhalten.
- Die normalisierten Vorhersagen werden denormalisiert.
- Die wahren und vorhergesagten Werte sowie der absolute Fehler werden tabellarisch ausgegeben.
- Schließlich werden die wahren und vorhergesagten Werte in einem Diagramm dargestellt, um die Leistung des Modells grafisch zu bewerten.

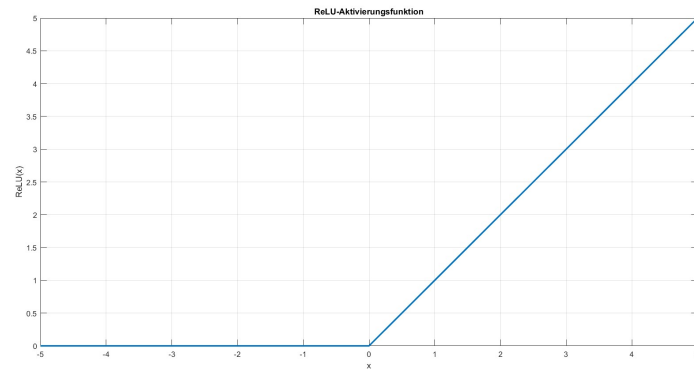


Abbildung 5.1: Relu Aktivierung Funktion

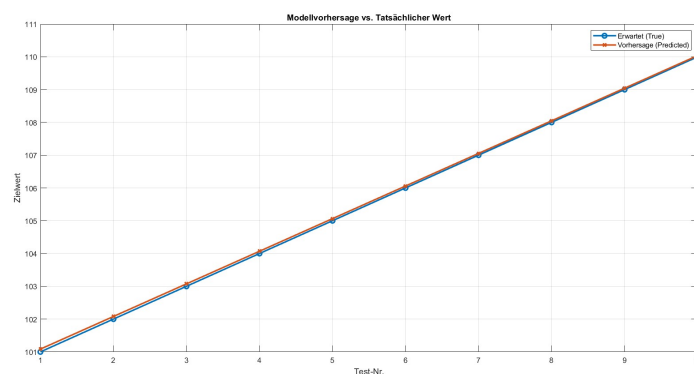


Abbildung 5.2: Grafischer Vergleich der vorhergesagten und tatsächlichen Werte

5.1.8 Vollständiger MATLAB-Code

Hier ist der vollständige MATLAB-Code, der das neuronale Netzwerk definiert, trainiert und testet:

```
1 clc; clear;
2
3 % --- DATENERZEUGUNG UND NORMALISIERUNG ---
4
5 X = zeros(95, 5);
6 Y = zeros(95, 1);
7
8 for i = 1:95
9     X(i,:) = i:i+4;
10    Y(i) = i + 5;
11 end
12
13 % --- NORMALISIERUNG DER DATEN ---
14
15 X_min = min(X(:));
16 X_max = max(X(:));
17 X = (X - X_min) / (X_max - X_min);
18
19 Y_min = min(Y);
20 Y_max = max(Y);
21 Y = (Y - Y_min) / (Y_max - Y_min);
22
23 % --- NETZWERKARCHITEKTUR UND INITIALISIERUNG ---
24
```

```
25 layers = [5, 10, 1];
26 num_layers = length(layers) - 1;
27
28 W = cell(num_layers,1);
29 b = cell(num_layers,1);
30
31 for i = 1:num_layers
32     W{i} = randn(layers(i), layers(i+1)) * 0.1;
33     b{i} = zeros(1, layers(i+1));
34 end
35
36 % --- AKTIVIERUNGSFUNKTIONEN ---
37
38 act = @(x) max(0, x);
39 dact = @(x) double(x > 0);
40
41 x_vals = -5:0.01:5;
42 y_vals = act(x_vals);
43 figure;
44 plot(x_vals, y_vals, 'LineWidth', 2);
45 xlabel('x');
46 ylabel('ReLU(x)');
47 title('ReLU-Aktivierungsfunktion');
48 grid on;
49
50 % --- HYPERPARAMETER ---
51
52 epochs = 2000;
53 lr = 0.01;
54
55 % --- TRAININGSLÖOP ---
56 for epoch = 1:epochs
57     A = cell(num_layers+1,1);
58     Z = cell(num_layers,1);
59     A{1} = X;
60
61     for i = 1:num_layers
62         Z{i} = A{i} * W{i} + b{i};
63         if i < num_layers
64             A{i+1} = act(Z{i});
65         else
66             A{i+1} = Z{i};
67         end
68     end
69
70     loss = mean((A{end} - Y).^2);
71
72     dA = 2 * (A{end} - Y) / size(X,1);
73
74     for i = num_layers:-1:1
75         dZ = dA;
76         if i < num_layers
77             dZ = dZ .* dact(Z{i});
78         end
79         dW = A{i}' * dZ;
80         db = sum(dZ, 1);
81         dA = dZ * W{i}';
82
83         W{i} = W{i} - lr * dW;
84         b{i} = b{i} - lr * db;
85     end
86
87     if mod(epoch, 100) == 0
```

```

88     fprintf("Epoche %d - Fehler: %.6f\n", epoch, loss);
89     end
90 end
91
92 % --- TESTEN DES NETZES MIT EINEM BEISPIEL ---
93
94 x_test = ([96 97 98 99 100] - X_min) / (X_max - X_min);
95
96 for i = 1:num_layers-1
97     x_test = act(x_test * W{i} + b{i});
98 end
99
100 y_pred = x_test * W{end} + b{end};
101 y_pred_orig = y_pred * (Y_max - Y_min) + Y_min;
102 fprintf("Vorhersage (normalisiert): %.4f  Tatsachlicher Wert: %.1f\n", y_pred,
        y_pred_orig);
103
104 % --- TESTEN DES NETZES MIT MEHREREN BEISPIELEN ---
105 X_test = zeros(10, 5);
106 Y_true = zeros(10, 1);
107
108 for i = 96:105
109     idx = i - 95;
110     X_test(idx,:) = i:i+4;
111     Y_true(idx) = i + 5;
112 end
113
114 X_test_norm = (X_test - X_min) / (X_max - X_min);
115 Y_pred = zeros(10, 1);
116
117 for n = 1:size(X_test_norm, 1)
118     x = X_test_norm(n, :);
119     for i = 1:num_layers-1
120         x = act(x * W{i} + b{i});
121     end
122     Y_pred(n) = x * W{end} + b{end};
123 end
124
125 Y_pred = Y_pred * (Y_max - Y_min) + Y_min;
126
127 disp('--- TESTERGEBNISSE ---');
128 fprintf('%6s %12s %12s %12s\n', 'Test-Nr', 'Erwartet', 'Vorhersage', 'Absoluter
        Fehler');
129 for i = 1:10
130     fehler = abs(Y_true(i) - Y_pred(i));
131     fprintf('%6d %12.2f %12.2f %12.2f\n', i, Y_true(i), Y_pred(i), fehler);
132 end
133
134 figure;
135 plot(1:10, Y_true, '-o', 'LineWidth', 2);
136 hold on;
137 plot(1:10, Y_pred, '-x', 'LineWidth', 2);
138 legend('Erwartet (True)', 'Vorhersage (Predicted)');
139 xlabel('Test-Nr. ');
140 ylabel('Zielwert');
141 title('Modellvorhersage vs. Tatsachlicher Wert');
142 grid on;
    
```

Listing 5.1: Neuronales Netzwerk: Datengenerierung, Training und Test

5.2 Neuronales Netzwerk zur Klassifikation dreier Klassen

In diesem Abschnitt wird ein vollständig in MATLAB implementiertes Feedforward-Neuronales Netz beschrieben, das zur Klassifikation von dreidimensionalen Datenpunkten in drei verschiedene Klassen trainiert wurde. Die Eingabedaten bestehen aus insgesamt 300 Punkten, die künstlich erzeugt wurden: **Klasse 1** enthält Werte im Bereich $[0, 100]$, **Klasse 2** im Bereich $[100, 200]$, und **Klasse 3** im Bereich $[200, 300]$. Diese Daten werden anschließend mittels *featureweiser Min-Max-Normalisierung* auf den Bereich $[0, 1]$ skaliert, um eine einheitliche Eingabeskalierung für das Netzwerk zu gewährleisten.

Zur Klassenzuordnung wird ein **One-Hot-Encoding** verwendet: Die Zielwerte werden jeweils als Vektor mit einem Eins-Eintrag an der Position der zugehörigen Klasse dargestellt. Die Netzarchitektur umfasst drei Schichten: eine Eingangsschicht mit drei Neuronen, zwei versteckte Schichten mit fünf Neuronen und eine Ausgangsschicht mit drei Neuronen. Die Aktivierungsfunktion für die versteckten Schichten ist die ReLU-Funktion, während in der Ausgangsschicht die Softmax-Funktion verwendet wird, um Wahrscheinlichkeiten für jede Klasse zu erzeugen.

Das Training erfolgt über 1000 Epochen mit einer Lernrate von 0,001 mittels Gradientenabstieg und Backpropagation unter Verwendung der Kreuzentropie als Verlustfunktion. Während des Trainings lernt das Netzwerk, die charakteristischen Merkmale jeder Klasse zu erfassen, basierend auf der relativen Position der Eingabedaten im ursprünglichen Wertebereich.

Nach dem Training wird die Klassifikationsfähigkeit der gelernten Gewichte sowohl an einzelnen als auch an mehreren neuen Beispielen getestet. Diese Testdaten decken denselben Wertebereich ab wie die Trainingsdaten (z. B. ein Punkt mit Werten um 120 gehört zur Klasse 2) und dienen der Überprüfung der Generalisierungsfähigkeit. Die Vorhersagen des Netzwerks zeigen eine hohe Genauigkeit, die durch Vergleich mit den erwarteten Klassen ausgewertet wird. Eine anschließende 3D-Visualisierung der ursprünglichen Daten verdeutlicht die räumliche Trennung der Klassen und unterstützt die Nachvollziehbarkeit des Lernprozesses.

Ziel dieses Modells ist es, zu erkennen, ob ein neuer Punkt eher im Bereich $[0, 100]$, $[100, 200]$ oder $[200, 300]$ liegt und dementsprechend korrekt klassifiziert wird. Die Netzausgabe stellt somit eine Form der automatisierten Bereichszuordnung dar.

Test-Nr	Input (Original)	Erwartet	Vorhergesagt	Überprüfung
1	[30 40 20]	1	1	ok
2	[80 70 60]	1	1	ok
3	[110 120 130]	2	2	ok
4	[150 160 140]	2	2	ok
5	[210 220 230]	3	3	ok
6	[270 280 260]	3	3	ok

Tabelle 5.2: Genauigkeit des Netzes auf zuvor ungesehenen Testbeispielen

Die in Tabelle 5.2 dargestellten Testergebnisse zeigen die Leistung des trainierten neuronalen Netzwerks bei der Klassifikation zuvor ungesehener Datenpunkte. Die Eingabedaten repräsentieren typische Werte für drei unterschiedliche Klassen, wobei die Zuordnung durch den Wertebereich erfolgt: Klasse 1 umfasst niedrigere Werte (z. B. $[30, 40, 20]$), Klasse 2 mittlere Werte (z. B. $[110, 120, 130]$) und Klasse 3 höhere Werte (z. B. $[210, 220, 230]$). Für alle sechs Testbeispiele konnte das Netzwerk die korrekte Klasse zuverlässig vorhersagen. Das Symbol 'ok' kennzeichnet dabei eine korrekte Klassifikation. Die vollständige Übereinstimmung zwischen erwarteter und vorhergesagter Klasse belegt eine hohe Generalisierungsfähigkeit des Modells bei strukturiert aufgebauten Daten mit klarer Klassenabgrenzung.

Die nachfolgende Abbildung zeigt eine visuelle Darstellung der durch das neuronale Netz erlernten Klassifikation. Die Datenpunkte wurden entsprechend ihrer zugehörigen Klasse eingefärbt, basierend auf den Ausgaben des Netzwerks. Diese Darstellung veranschaulicht deutlich, wie gut das Modell in der Lage ist, die verschiedenen Klassen im Merkmalsraum zu trennen und korrekt zuzuordnen.

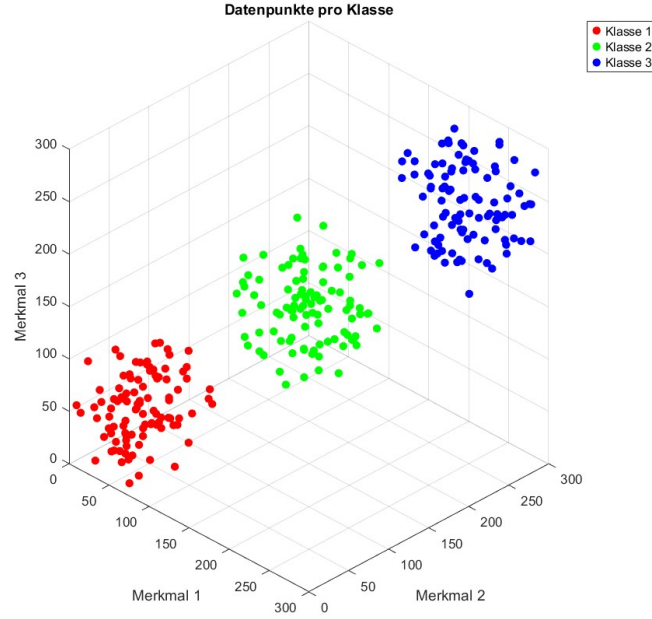


Abbildung 5.3: Visuelle Klassifikation der Datenpunkte basierend auf den Netzwerkausgaben

5.2.1 Datenerzeugung und Normalisierung

Zur Erzeugung von drei klar getrennten Klassen von Datenpunkten verwenden wir gleichverteilte Zufallszahlen. Mathematisch lässt sich das so schreiben:

$$X_1 = 100 U_1, \quad X_2 = 100 U_2 + 100, \quad X_3 = 100 U_3 + 200, \quad (5.35)$$

wobei $U_1, U_2, U_3 \in \mathbb{R}^{100 \times 3}$ Matrizen mit Einträgen aus $\mathcal{U}(0, 1)$ sind. Damit liegen die Punkte in den Bereichen

$$X_1 \in [0, 100]^{100 \times 3}, \quad X_2 \in [100, 200]^{100 \times 3}, \quad X_3 \in [200, 300]^{100 \times 3}.$$

Anschließend werden alle Daten zu einer großen Matrix zusammengefasst:

$$X_{\text{all}} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{300 \times 3}. \quad (5.36)$$

Wir bestimmen nun für jedes Merkmal (jede Spalte) das Minimum und Maximum:

$$X_{\min,j} = \min_{1 \leq i \leq 300} [X_{\text{all}}]_{ij}, \quad X_{\max,j} = \max_{1 \leq i \leq 300} [X_{\text{all}}]_{ij}, \quad (5.37)$$

und wenden dann die Feature-wise Min-Max-Normalisierung an:

$$[X_{\text{norm}}]_{ij} = \frac{[X_{\text{all}}]_{ij} - X_{\min,j}}{X_{\max,j} - X_{\min,j} + \varepsilon}, \quad (5.38)$$

wobei $\varepsilon = \text{eps}$ eine sehr kleine Zahl ist, um eine Division durch Null zu vermeiden.

Somit liegen alle normalisierten Merkmale in $[0, 1]$, was das Training von maschinellen Lernverfahren oft stabiler und effizienter macht.“

Code-Ausschnitt:

```

1 X1 = rand(100,3) * 100;
2 X2 = rand(100,3) * 100 + 100;
3 X3 = rand(100,3) * 100 + 200;
4
5 X_all = [X1; X2; X3];
6 X_min = min(X_all);
7 X_max = max(X_all);
8 X = (X_all - X_min) ./ (X_max - X_min + eps);
    
```

Erläuterung:

- Es werden drei Klassen simuliert: Werte in den Bereichen [0–100], [100–200] und [200–300].
- Die Daten werden merkmalsweise mit Min-Max-Normalisierung auf den Bereich [0,1] skaliert.

5.2.2 Zielwerte mit One-Hot-Encoding

Für die drei Klassen definieren wir die Einheitsvektoren

$$e_1 = (1, 0, 0), \quad e_2 = (0, 1, 0), \quad e_3 = (0, 0, 1).$$

Mathematisch entspricht das MATLAB-Statement

```

1 Y = [ repmat([1 0 0],100,1);
2       repmat([0 1 0],100,1);
3       repmat([0 0 1],100,1) ];
    
```

der folgenden Replikation und Stapelung:

$$Y = \underbrace{\begin{bmatrix} e_1 \\ e_1 \\ \vdots \\ e_1 \end{bmatrix}}_{100 \text{ Mal}} \mid \underbrace{\begin{bmatrix} e_2 \\ e_2 \\ \vdots \\ e_2 \end{bmatrix}}_{100 \text{ Mal}} \mid \underbrace{\begin{bmatrix} e_3 \\ e_3 \\ \vdots \\ e_3 \end{bmatrix}}_{100 \text{ Mal}} = \begin{bmatrix} \underbrace{1 \ 0 \ 0}_{100} \\ \vdots \\ \underbrace{1 \ 0 \ 0}_{100} \\ \underbrace{0 \ 1 \ 0}_{100} \\ \vdots \\ \underbrace{0 \ 1 \ 0}_{100} \\ \underbrace{0 \ 0 \ 1}_{100} \\ \vdots \\ \underbrace{0 \ 0 \ 1}_{100} \end{bmatrix} \in \{0, 1\}^{300 \times 3}. \quad (5.39)$$

Hierbei steht jede der drei Blöcke für 100 Zeilen des entsprechenden One-Hot-Vektors, sodass Y insgesamt 300×3 Einträge besitzt und jede Zeile genau eine Eins enthält.““

Code-Ausschnitt:

```

1 Y = [ repmat([1 0 0],100,1);
2       repmat([0 1 0],100,1);
3       repmat([0 0 1],100,1) ];
    
```

Erläuterung:

- One-Hot-Encoding wird verwendet: Klasse 1 $\rightarrow [1 \ 0 \ 0]$, Klasse 2 $\rightarrow [0 \ 1 \ 0]$, Klasse 3 $\rightarrow [0 \ 0 \ 1]$.

5.2.3 Netzwerkarchitektur und Initialisierung

Wir betrachten ein Feed-Forward-Netzwerk mit den Schichtgrößen

$$\text{layers} = (l_1, l_2, \dots, l_L), \quad L = \text{length}(\text{layers}). \quad (5.40)$$

Die Anzahl der Verbindungs-Schichten (Gewichtsmatrizen) ist dann

$$\text{num_layers} = L - 1. \quad (5.41)$$

Für jede Schicht $i = 1, \dots, \text{num_layers}$ wird die Gewichtsmatrix $W^{(i)} \in \mathbb{R}^{l_i \times l_{i+1}}$ und der Bias-Vektor $b^{(i)} \in \mathbb{R}^{1 \times l_{i+1}}$ folgendermaßen initialisiert:

$$W^{(i)} = \sqrt{\frac{2}{l_i + l_{i+1}}} Z^{(i)}, \quad Z_{jk}^{(i)} \sim \mathcal{N}(0, 1), \quad (5.42)$$

wodurch jede Komponente von $W^{(i)}$ eine Normalverteilung mit Varianz $\sigma_i^2 = \frac{2}{l_i + l_{i+1}}$ erhält. Dies gewährleistet, dass die Varianz der Aktivierungen über die Schichten hinweg annähernd konstant bleibt und verbessert so die Konvergenz beim Training.

Für die Bias-Vektoren wählen wir:

$$b^{(i)} = \mathbf{0}_{1 \times l_{i+1}}, \quad (5.43)$$

also einen Nullvektor, um zu Beginn keine systematischen Verschiebungen einzuführen.

Code-Ausschnitt:

```
1 layers = [3, 5, 3];
2 num_layers = length(layers) - 1;
3
4 for i = 1:num_layers
5     W{i} = randn(layers(i), layers(i+1)) * sqrt(2 / (layers(i) + layers(i+1)));
6     b{i} = zeros(1, layers(i+1));
7 end
```

Erläuterung:

- Architektur: 3 Eingänge, 5 Neuronen in einer versteckten Schicht, 3 Ausgänge.
- Gewichte werden mit He-Initialisierung gesetzt.

5.2.4 Hyperparameter

Die wichtigsten Hyperparameter für das Training sind die Lernrate η und die Anzahl der Epochen N_{ep} . Im MATLAB-Code sind diese definiert als

```
1 lr = 0.001;
2 epochs = 1000;
```

Mathematisch schreiben wir:

$$\eta = 0.001 \quad (5.44)$$

$$N_{\text{ep}} = 1000 \quad (5.45)$$

Die Lernrate η bestimmt die Schrittweite bei der Aktualisierung der Gewichte gemäß der Gradientenabstiegs-Formel

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} \mathcal{L}(\theta_t), \quad (5.46)$$

wobei θ alle Netzwerkparameter und \mathcal{L} die Verlustfunktion bezeichnet. Eine zu kleine Lernrate verlangsamt die Konvergenz, eine zu große kann zu Instabilitäten führen.

Die Anzahl der Epochen N_{ep} gibt an, wie oft das gesamte Trainingsdatenset während des Trainings durchlaufen wird. Insgesamt erfolgen also

$$\text{Anzahl der Updates} = N_{\text{ep}} \times N_{\text{Batches}} \quad (5.47)$$

Gradientenberechnungen und Parameteraktualisierungen.““

Erläuterung:

- Lernrate und Anzahl der Trainingszyklen.

5.2.5 Trainingsprozess (Forward- und Backward-Pass)

Im Folgenden sei m die Batch-Größe. Der Trainingsprozess gliedert sich in zwei Phasen:

- **Forward-Pass:** Eingaben werden schichtweise nach vorne propagiert.
- **Backward-Pass:** Gradienten werden rekursiv von der Ausgabeschicht zurück zur Eingabe berechnet.

Forward-Pass in den versteckten Schichten Für Schicht $i = 1, \dots, L-1$ mit Aktivierungen $A^{(i-1)} \in \mathbb{R}^{m \times n_{i-1}}$, Gewichten $W^{(i)} \in \mathbb{R}^{n_{i-1} \times n_i}$ und Bias $b^{(i)} \in \mathbb{R}^{1 \times n_i}$ gilt

$$Z^{(i)} = A^{(i-1)}W^{(i)} + \mathbf{1}_m b^{(i)}, \quad (5.48)$$

$$A^{(i)} = \text{ReLU}(Z^{(i)}) = \max(0, Z^{(i)}), \quad (5.49)$$

wobei $\mathbf{1}_m$ der Einheitsvektor in \mathbb{R}^m ist.

Forward-Pass in der Ausgabeschicht Für die letzte Schicht L erzeugt die Softmax-Funktion die Wahrscheinlichkeiten

$$A_{kj}^{(L)} = \text{softmax}(Z^{(L)})_{kj} = \frac{\exp(Z_{kj}^{(L)})}{\sum_{l=1}^{n_L} \exp(Z_{lj}^{(L)})}, \quad j = 1, \dots, m, \quad (5.50)$$

wobei k den Klassenindex und j den Batch-Beispielindex bezeichnet.

Verlustfunktion Wir verwenden die mittlere Kreuzentropie über das Batch:

$$\mathcal{L} = -\frac{1}{m} \sum_{j=1}^m \sum_{k=1}^{n_L} Y_{kj} \log(A_{kj}^{(L)} + \varepsilon), \quad (5.51)$$

wobei $Y \in \{0, 1\}^{n_L \times m}$ die One-Hot-Zielmatrix ist und $\varepsilon > 0$ numerische Stabilität gewährleistet.

Backward-Pass

1. **Ausgangsgradient:** Die Ableitung der Verlustfunktion nach $Z^{(L)}$ vereinfacht sich zu

$$dZ^{(L)} = A^{(L)} - Y,$$

da die Kombination von Softmax und Kreuzentropie zu dieser eleganten Form führt.

2. **Gradienten in Schicht i :** Für jede Schicht $i = L, \dots, 1$ berechnen wir

$$dW^{(i)} = \frac{1}{m} (A^{(i-1)})^T dZ^{(i)}, \quad (5.52)$$

$$db^{(i)} = \frac{1}{m} \sum_{j=1}^m dZ_j^{(i)} = \frac{1}{m} \mathbf{1}_m^T dZ^{(i)}, \quad (5.53)$$

$$dA^{(i-1)} = dZ^{(i)} (W^{(i)})^T. \quad (5.54)$$

3. **Gradient durch die Aktivierung (für $i > 1$):** Für $i > 1$ leiten wir weiter zurück bis zur vorherigen linearen Kombinationsgröße:

$$dZ^{(i-1)} = dA^{(i-1)} \odot \text{ReLU}'(Z^{(i-1)}),$$

mit

$$\text{ReLU}'(z) = \begin{cases} 1, & z > 0, \\ 0, & z \leq 0, \end{cases}$$

und \odot als Hadamard-Produkt.

```

1 for epoch = 1:epochs
2     A = cell(num_layers+1,1);
3     Z = cell(num_layers,1);
4     A{1} = X; % Erste Schicht = Input
5
6     for i = 1:num_layers
7         Z{i} = A{i} * W{i} + b{i};
8
9         if i < num_layers
10            A{i+1} = relu(Z{i});
11        else
12            A{i+1} = softmax(Z{i});
13        end
14    end
15
16    loss = -sum(sum(Y .* log(A{end} + eps))) / size(X,1);
17    % Ausgangsfehler (Differenz zwischen Vorhersage und echtem Wert)
18    dA = A{end} - Y;
19    dZ = A{end} - Y; % salida de softmax - etiquetas verdaderas
20
21    for i = num_layers:-1:1
22        dW = A{i}' * dZ;
23        db = sum(dZ, 1);
24
25        W{i} = W{i} - lr * dW;
26        b{i} = b{i} - lr * db;
27
28        if i > 1
29            dA_prev = dZ * W{i}';
30            dZ = dA_prev .* drelu(Z{i-1});
31        end
32    end
33
34    if mod(epoch,100) == 0
35        fprintf("Epoche %d - Verlust: %.4f\n", epoch, loss);
36    end
37 end
    
```

Erläuterung:

- ReLU in versteckten Schichten, Softmax in der Ausgabeschicht.

- Fehlerfunktion: Kreuzentropie.
- Rückwärtspropagation zur Gewichtsaktualisierung.

5.2.6 Test mit Beispielen und Genauigkeit

Im Folgenden wird die Testphase des Netzes mathematisch begründet. Dabei sei

$$X_{\min} = \min_{i,j} X_{ij}, \quad X_{\max} = \max_{i,j} X_{ij}, \quad \varepsilon = \text{kleinste darstellbare Gleitkommazahl.}$$

Normalisierung der Testdaten Jedes Test-Beispiel $x \in \mathbb{R}^d$ wird analog zum Training skaliert:

$$\tilde{x} = \frac{x - X_{\min}}{X_{\max} - X_{\min} + \varepsilon}, \quad (5.55)$$

wodurch $\tilde{x} \in [0, 1]^d$ liegt.

Vorwärtsthroughlauf Sei das Netz aus L Schichten aufgebaut, mit Gewichten $W^{(l)}$ und Bias $b^{(l)}$. Für $l = 1, \dots, L - 1$ berechnen wir die Aktivierungen in den verdeckten Schichten mittels ReLU:

$$a^{(0)} = \tilde{x}, \quad a^{(l)} = \text{ReLU}(a^{(l-1)}W^{(l)} + b^{(l)}), \quad (5.56)$$

wobei

$$\text{ReLU}(z) = \max(0, z) \quad (5.57)$$

elementweise angewendet wird.

Die Ausgabe-Schicht L verwendet Softmax:

$$z = a^{(L-1)}W^{(L)} + b^{(L)}, \quad y = \text{softmax}(z), \quad (5.58)$$

mit

$$\text{softmax}(z)_k = \frac{\exp(z_k - \max_j z_j)}{\sum_j \exp(z_j - \max_j z_j) + \varepsilon}. \quad (5.59)$$

Klassenvorhersage Die vorhergesagte Klasse \hat{k} erhält man durch

$$\hat{k} = \arg \max_k y_k. \quad (5.60)$$

Berechnung der Genauigkeit Für eine Testmenge mit N Beispielen $\{(x^{(i)}, y^{(i)}) \mid i = 1, \dots, N\}$ definiert man

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^N \mathbf{1}(\hat{k}^{(i)} = y^{(i)}) \times 100\%, \quad (5.61)$$

wobei $\mathbf{1}(\cdot)$ die Indikatorfunktion ist, die den Wert 1 annimmt, wenn die Bedingung wahr ist, und 0 sonst.

Visualisierung Zur Veranschaulichung der Datenverteilung pro Klasse in \mathbb{R}^3 werden die Punkte $\{X_1, X_2, X_3\}$ mit unterschiedlichen Farben geplottet:

$$\{X_c \mid c = 1, 2, 3\} \longrightarrow \text{scatter3}(X_c(:, 1), X_c(:, 2), X_c(:, 3)),$$

wobei jede Klasse c durch eine Farbe (rot, grün, blau) kodiert ist.

Damit ist der Testprozess formal begründet und die MATLAB-Implementierung entspricht direkt den obigen Formeln.““

```

1
2 x_test = [110 120 130];
3 x_test = (x_test - X_min) ./ (X_max - X_min + eps); % gleiche Normalisierung wie im
   Training
4
5 for i = 1:num_layers-1
6     x_test = relu(x_test * W{i} + b{i});
7 end
8 y_pred = softmax(x_test * W{end} + b{end});
9
10 [~, klasse] = max(y_pred);
11 fprintf("Vorhergesagte Klasse (Einzeltest): %d\n", klasse);
12
13 fprintf("\n--- Genauigkeit des Netzes auf Testbeispielen ---\n");
14 fprintf("Input (Original) | Erwartet | Vorhergesagt\n");
15 fprintf("-----\n");
16
17 X_test_neu = [
18     30 40 20; % Klasse 1 erwartet
19     80 70 60; % Klasse 1 erwartet
20     110 120 130; % Klasse 2 erwartet
21     150 160 140; % Klasse 2 erwartet
22     210 220 230; % Klasse 3 erwartet
23     270 280 260; % Klasse 3 erwartet
24 ];
25
26 Y_erwartet = [1; 1; 2; 2; 3; 3]; % Erwartete Klassen
27 korrekt = 0;
28
29 for i = 1:size(X_test_neu,1)
30     x = (X_test_neu(i,:) - X_min) ./ (X_max - X_min + eps); % Normalisierung
31     for j = 1:num_layers-1
32         x = relu(x * W{j} + b{j});
33     end
34     y_hat = softmax(x * W{end} + b{end});
35     [~, k] = max(y_hat);
36
37     fprintf("[%3d %3d %3d] | %d | %d", ...
38         X_test_neu(i,1), X_test_neu(i,2), X_test_neu(i,3), ...
39         Y_erwartet(i), k);
40
41     if k == Y_erwartet(i)
42         fprintf(" ok \n");
43         korrekt = korrekt + 1;
44     else
45         fprintf(" not ok \n");
46     end
47 end
48
49
50 genauigkeit = korrekt / length(Y_erwartet) * 100;
51
52 figure;
53 scatter3(X1(:,1), X1(:,2), X1(:,3), 36, 'r', 'filled'); hold on;
54 scatter3(X2(:,1), X2(:,2), X2(:,3), 36, 'g', 'filled');
55 scatter3(X3(:,1), X3(:,2), X3(:,3), 36, 'b', 'filled');
56
57 xlabel('Merkmal 1');
58 ylabel('Merkmal 2');
59 zlabel('Merkmal 3');
60 title('Datenpunkte pro Klasse');
61 legend('Klasse 1', 'Klasse 2', 'Klasse 3');
62 grid on;

```

```

63 view(45, 30);
64
65
66 function y = relu(x)
67     y = max(0, x);
68 end
69
70 function y = drelu(x)
71     y = double(x > 0);
72 end
73
74 function y = softmax(z)
75     z = z - max(z, [], 2);
76     e_z = exp(z);
77     y = e_z ./ (sum(e_z, 2) + eps);
78 end

```

5.2.7 Aktivierungsfunktionen

Im Folgenden werden die Formeln und die mathematische Begründung der drei im Code-Snippet implementierten Funktionen dargestellt.

1. ReLU-Funktion Die ReLU-Funktion (Rectified Linear Unit) ist definiert als

$$\text{ReLU}(x) = \max(0, x). \quad (5.62)$$

- Für $x > 0$ ist die Ausgabe x , was positive Informationen bewahrt.
- Für $x \leq 0$ ist die Ausgabe 0, was Nichtlinearität einführt und die Aktivierung von Neuronen mit negativer Eingabe verhindert.
- Sie ist kontinuierlich und hat eine einfache Ableitung, was die Berechnung des Gradienten erleichtert.

2. Ableitung von ReLU Die Ableitung von $\text{ReLU}(x)$ ist eine Indikatorfunktion:

$$\frac{d}{dx} \text{ReLU}(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0. \end{cases} \quad (5.63)$$

- Für $x > 0$ ist $\text{ReLU}(x) = x$ und daher ist die Ableitung 1.
- Für $x \leq 0$ ist $\text{ReLU}(x) = 0$ (konstant), sodass die Ableitung 0 ist.
- In der Praxis kann für den Bereich $x = 0$ eine beliebige Ableitung zwischen 0 und 1 angenommen werden; üblicherweise wird 0 gewählt.

3. Softmax-Funktion Für einen Eingabevektor $z = (z_1, \dots, z_K)$ ist die Softmax-Funktion definiert durch

$$\text{softmax}(z)_k = \frac{\exp(z_k - \max_j z_j)}{\sum_{j=1}^K \exp(z_j - \max_j z_j)}. \quad (5.64)$$

- Die Subtraktion von $\max_j z_j$ gewährleistet numerische Stabilität, indem sie ein Überlaufen von \exp verhindert.

- Der Nenner normalisiert die Komponenten so, dass sie sich zu 1 addieren, wodurch eine Wahrscheinlichkeitsverteilung entsteht: $\sum_{k=1}^K \text{softmax}(z)_k = 1$.
- Ein kleines ε wird in der Implementierung hinzugefügt, um eine Division durch Null in degenerierten Fällen zu vermeiden.

Code-Ausschnitt:

```

1 function y = relu(x)
2     y = max(0, x);
3 end
4
5 function y = drelu(x)
6     y = double(x > 0);
7 end
8 function y = softmax(z)
9     z = z - max(z, [], 2); % numerische Stabilität
10    e_z = exp(z);
11    y = e_z ./ (sum(e_z, 2) + eps); % Vermeidung von Division durch Null
12 end
    
```

Erläuterung:

- `relu(x)`: Gibt für jede Eingabe nur positive Werte oder null zurück – wichtig für tiefere Netzwerke zur Vermeidung des Vanishing Gradient Problems.
- `drelu(x)`: Die Ableitung der ReLU-Funktion, notwendig für Backpropagation.
- `softmax(z)`: Wandelt die Ausgaben der letzten Schicht in Wahrscheinlichkeiten um, sodass jede Zeile der Ausgabe auf 1 normiert ist. Die Zeile `z = z - max(z, [], 2)` vermeidet numerisches Overflow durch Stabilisierung.

5.2.8 Vollständiger MATLAB-Code

Code-Ausschnitt:

```

1 clc
2 X1 = rand(100,3) * 100;
3 X2 = rand(100,3) * 100 + 100;
4 X3 = rand(100,3) * 100 + 200;
5 X_all = [X1; X2; X3];
6 X_min = min(X_all);
7 X_max = max(X_all);
8 X = (X_all - X_min) ./ (X_max - X_min + eps);
9 Y = [repmat([1 0 0],100,1); repmat([0 1 0],100,1); repmat([0 0 1],100,1)];
10 layers = [3, 5, 3];
11 num_layers = length(layers) - 1;
12 W = cell(num_layers,1);
13 b = cell(num_layers,1);
14 for i = 1:num_layers
15     W{i} = randn(layers(i), layers(i+1)) * sqrt(2 / (layers(i) + layers(i+1)));
16     b{i} = zeros(1, layers(i+1));
17 end
18 lr = 0.001;
19 epochs = 1000;
20 for epoch = 1:epochs
21     A = cell(num_layers+1,1);
22     Z = cell(num_layers,1);
23     A{1} = X;
24     for i = 1:num_layers
25         Z{i} = A{i} * W{i} + b{i};
26         if i < num_layers
27             A{i+1} = relu(Z{i});
28         end
29     end
30 end
    
```

```

28         else
29             A{i+1} = softmax(Z{i});
30         end
31     end
32     loss = -sum(sum(Y .* log(A{end} + eps))) / size(X,1);
33     dA = A{end} - Y;
34     dZ = A{end} - Y;
35     for i = num_layers:-1:1
36         dW = A{i}' * dZ;
37         db = sum(dZ, 1);
38         W{i} = W{i} - lr * dW;
39         b{i} = b{i} - lr * db;
40         if i > 1
41             dA_prev = dZ * W{i}';
42             dZ = dA_prev .* drelu(Z{i-1});
43         end
44     end
45     if mod(epoch,100) == 0
46         fprintf("Epoch %d - Loss: %.4f\n", epoch, loss);
47     end
48 end
49 x_test = [110 120 130];
50 x_test = (x_test - X_min) ./ (X_max - X_min + eps);
51 for i = 1:num_layers-1
52     x_test = relu(x_test * W{i} + b{i});
53 end
54 y_pred = softmax(x_test * W{end} + b{end});
55 [~, predicted_class] = max(y_pred);
56 fprintf("Predicted Class (Single Test): %d\n", predicted_class);
57 fprintf("\n--- Network Accuracy on Test Examples ---\n");
58 fprintf("Input (Original) | Expected | Predicted\n");
59 fprintf("-----\n");
60 X_test_new = [
61     30 40 20;
62     80 70 60;
63     110 120 130;
64     150 160 140;
65     210 220 230;
66     270 280 260;
67 ];
68 Y_expected = [1; 1; 2; 2; 3; 3];
69 correct_predictions = 0;
70 for i = 1:size(X_test_new,1)
71     x = (X_test_new(i,:) - X_min) ./ (X_max - X_min + eps);
72     for j = 1:num_layers-1
73         x = relu(x * W{j} + b{j});
74     end
75     y_hat = softmax(x * W{end} + b{end});
76     [~, k] = max(y_hat);
77     fprintf("[%3d %3d %3d] | %d | %d", ...
78         X_test_new(i,1), X_test_new(i,2), X_test_new(i,3), ...
79         Y_expected(i), k);
80     if k == Y_expected(i)
81         fprintf(" ok");
82         correct_predictions = correct_predictions + 1;
83     else
84         fprintf(" not ok");
85     end
86 end
87 accuracy = correct_predictions / length(Y_expected) * 100;
88 fprintf("\nAccuracy on test data: %.2f %%\n", accuracy);
89 figure;
90 scatter3(X1(:,1), X1(:,2), X1(:,3), 36, 'r', 'filled'); hold on;

```

```
91 scatter3(X2(:,1), X2(:,2), X2(:,3), 36, 'g', 'filled');
92 scatter3(X3(:,1), X3(:,2), X3(:,3), 36, 'b', 'filled');
93 xlabel('Feature 1');
94 ylabel('Feature 2');
95 zlabel('Feature 3');
96 title('Data Points per Class');
97 legend('Class 1', 'Class 2', 'Class 3');
98 grid on;
99 view(45, 30);
100 function y = relu(x)
101     y = max(0, x);
102 end
103 function y = drelu(x)
104     y = double(x > 0);
105 end
106 function y = softmax(z)
107     z = z - max(z, [], 2);
108     e_z = exp(z);
109     y = e_z ./ (sum(e_z, 2) + eps);
110 end
```

5.3 Neuronales Netzwerk zur Erkennung handschriftlicher Ziffern

Dieser Code implementiert ein zweischichtiges neuronales Netz zur Erkennung handgeschriebener Ziffern mithilfe des MNIST-Datensatzes.

Zunächst werden die Trainings- und Testbilder geladen. Anschließend werden 12 Beispiele aus dem Trainingsdatensatz zusammen mit ihren Labels visualisiert. Die Daten werden vorverarbeitet, indem die Bilder in Vektoren umgewandelt und die Labels in eine One-Hot-Kodierung überführt werden.

Das neuronale Netz ist mit einer versteckten Schicht von 64 Neuronen und einer Ausgabeschicht von 10 Neuronen (eine für jede Ziffer von 0 bis 9) definiert. Als Aktivierungsfunktionen werden die Sigmoid-Funktion in der versteckten Schicht und die Softmax-Funktion in der Ausgabeschicht verwendet, um Wahrscheinlichkeiten zu erhalten. Das Training erfolgt mittels Backpropagation, wobei Gewichte und Bias mit einer Lernrate von 0,1 über 5 Epochen angepasst werden.

Nach dem Training wird das Modell mit dem Testsatz evaluiert, wobei die Genauigkeit berechnet und die Vorhersagen für 12 Testbilder zusammen mit ihren tatsächlichen Labels angezeigt werden. Darüber hinaus enthält der Code wichtige Visualisierungen: die in der versteckten Schicht verwendete Sigmoid-Funktion und ein Beispiel einer von Softmax generierten Wahrscheinlichkeitsverteilung für 10 Klassen (Ziffern von 0 bis 9). Letzteres wird in einem Balkendiagramm mit unterschiedlichen Farben pro Klasse und numerischen Werten über jedem Balken dargestellt. Diese Abbildung verdeutlicht, wie Softmax die Netzwerkausgaben zu Wahrscheinlichkeiten normiert, die sich zu 1 summieren.

Zusammenfassend umfasst das Projekt das Laden und Vorverarbeiten von Daten, das Training und die Evaluierung eines Ziffernerkennungsmodells, ergänzt durch didaktische Visualisierungen der Aktivierungsfunktionen (Sigmoid) und der Wahrscheinlichkeitsverteilung (Softmax), sowie konkrete Beispiele von Vorhersagen für Testbilder. All dies ermöglicht ein Verständnis sowohl der internen Funktionsweise des Netzes als auch seiner praktischen Leistung bei der Klassifizierungsaufgabe.

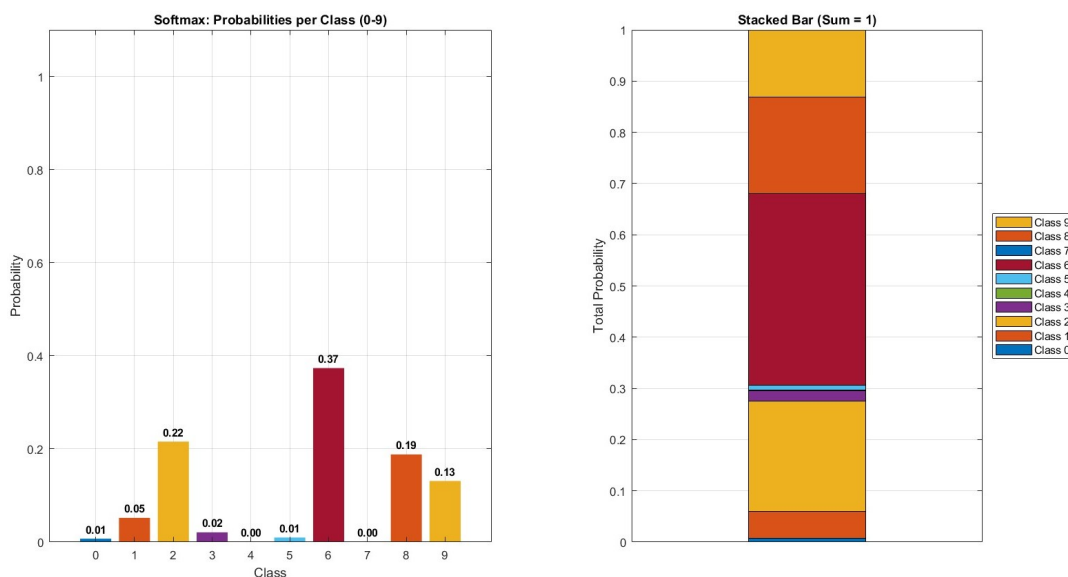


Abbildung 5.4: Softmax-Wahrscheinlichkeitsverteilung für die Bildvorhersage

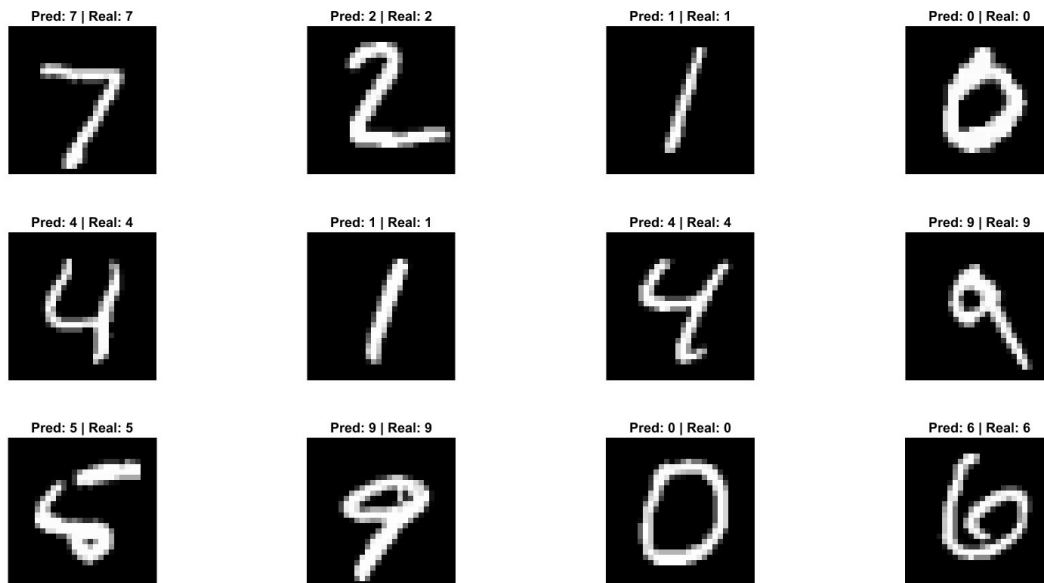


Abbildung 5.5: Vorhersageergebnisse für 12 Bilder

Ergebnisanalyse Die Abbildung zeigt sechzehn zufällig ausgewählte handschriftliche Ziffern aus dem Testdatensatz zusammen mit den vorhergesagten (Pred) und den tatsächlichen (Real) Klassenlabels. In allen dargestellten Fällen stimmen die Vorhersagen genau mit den realen Labels überein, was auf eine nahezu fehlerfreie Klassifikation dieser Beispiele hinweist. Besonders bemerkenswert ist, dass selbst ähnlich aussehende Ziffern wie „1“ und „7“ oder „4“ und „9“ korrekt unterschieden werden, was die hohe Genauigkeit und Robustheit des trainierten Netzwerks unterstreicht.

5.3.1 Datenladen und Visualisierung

Dieser Teil lädt die MNIST-Daten (handgeschriebene Ziffern) und zeigt Beispiele der Trainingsdaten an.

Code-Ausschnitt:

```

1 clc; clear;
2
3 % === DATA LOADING ===
4 trainImages = MNISTLoader.loadImages('train-images.idx3-ubyte');
5 trainLabels = MNISTLoader.loadLabels('train-labels.idx1-ubyte');
6 testImages = MNISTLoader.loadImages('t10k-images.idx3-ubyte');
7 testLabels = MNISTLoader.loadLabels('t10k-labels.idx1-ubyte');
8
9 % === VISUALIZATION OF 12 TRAINING SET IMAGES ===
10 figure('Name','Training Set Examples');
11 for i = 1:12
12     subplot(3,4,i);
13     imshow(trainImages(:,:,i));
14     title(['Label: ', num2str(trainLabels(i))]);
15 end

```

Erläuterung:

- `clc; clear;` löscht die Konsole und den Workspace
- Die MNIST-Daten werden geladen (Trainings- und Testdaten)
- 12 Beispiele aus dem Trainingsset werden mit ihren Labels visualisiert

5.3.2 Datenvorverarbeitung

Die Vorbereitung der Eingabedaten erfolgt in zwei Schritten: Aplanierung der Bilder und One-Hot-Kodierung der Labels.

Aplanierung der Bilder Sei $\{I^{(i)}\}_{i=1}^N$ eine Menge von N Grauwertbildern der Größe 28×28 . Wir definieren den Vektor

$$\text{vec}(I^{(i)}) = (I_{1,1}^{(i)}, I_{1,2}^{(i)}, \dots, I_{1,28}^{(i)}, I_{2,1}^{(i)}, \dots, I_{28,28}^{(i)})^\top \in \mathbb{R}^{784}.$$

Dann erhält man die Datenmatrix

$$X = \begin{bmatrix} \text{vec}(I^{(1)})^\top \\ \text{vec}(I^{(2)})^\top \\ \vdots \\ \text{vec}(I^{(N)})^\top \end{bmatrix} \in \mathbb{R}^{N \times 784}. \quad (5.65)$$

In MATLAB entspricht dies dem Befehl

```
1 X = reshape(trainImages, 28*28, [])'; % [N x 784]
```

One-Hot-Kodierung der Labels Sei $\{l_i\}_{i=1}^N$ die Menge der Labels, wobei $l_i \in \{0, 1, \dots, 9\}$. Die One-Hot-Kodierung liefert eine Matrix

$$Y = [y^{(1)}, y^{(2)}, \dots, y^{(N)}]^\top \in \{0, 1\}^{N \times 10},$$

wobei jedes $y^{(i)} \in \{0, 1\}^{10}$ durch

$$y_k^{(i)} = \begin{cases} 1, & k = l_i, \\ 0, & k \neq l_i, \end{cases} \quad k = 1, \dots, 10 \quad (5.66)$$

definiert ist. In MATLAB wird dies durch

```
1 Y = oneHot(trainLabels); % [N x 10]
```

realisiert.

Damit liegen die Daten in der Form

$$X \in \mathbb{R}^{N \times 784}, \quad Y \in \{0, 1\}^{N \times 10}$$

vor und können direkt für das Training eines neuronalen Netzes verwendet werden.

Code-Ausschnitt:

```
1 % === PREPROCESSING ===
2 X = reshape(trainImages, 28*28, [])'; % [numSamples x 784]
3 Y = oneHot(trainLabels); % [numSamples x 10]
4
5 disp("Example of one-hot encoding for the first 5 labels:");
6 disp(trainLabels(1:5));
7 disp(Y(1:5, :));
```

Erläuterung:

- Die 28x28 Pixel Bilder werden in Vektoren der Länge 784 umgewandelt
- Die Labels werden one-hot-kodiert (z.B. wird '3' zu [0 0 0 1 0 0 0 0 0 0])
- Ein Beispiel der Kodierung wird angezeigt

5.3.3 Netzwerkarchitektur

Definition der Netzwerkstruktur und Initialisierung der Gewichte.

Definition der Layer-Größen Wir legen fest:

$$\text{inputSize} = 784, \quad \text{hiddenSize} = 64, \quad \text{outputSize} = 10.$$

Dies entspricht einem zweischichtigen Feedforward-Netzwerk mit:



Dimensionierung der Gewichts- und Bias-Matrizen Die Gewichtsmatrizen und Bias-Vektoren haben folgende Dimensionen:

$$W_1 \in \mathbb{R}^{\text{inputSize} \times \text{hiddenSize}}, \quad b_1 \in \mathbb{R}^{1 \times \text{hiddenSize}} \quad (5.67)$$

$$W_2 \in \mathbb{R}^{\text{hiddenSize} \times \text{outputSize}}, \quad b_2 \in \mathbb{R}^{1 \times \text{outputSize}}. \quad (5.68)$$

Zufällige Initialisierung der Gewichte Um symmetriebrechend zu initialisieren, wählt man für die Einträge von W_1 und W_2 eine Normalverteilung mit Mittelwert 0 und kleiner Standardabweichung $\sigma = 0.01$:

$$(W_1)_{ij} \sim \mathcal{N}(0, 0.01^2), \quad (W_2)_{jk} \sim \mathcal{N}(0, 0.01^2), \quad (5.69)$$

wobei $i = 1, \dots, 784, j = 1, \dots, 64, k = 1, \dots, 10$. Die Bias-Vektoren werden initial auf null gesetzt:

$$b_1 = \mathbf{0} \in \mathbb{R}^{1 \times 64}, \quad b_2 = \mathbf{0} \in \mathbb{R}^{1 \times 10}. \quad (5.70)$$

Code-Ausschnitt:

```
1 % === NETWORK ARCHITECTURE DEFINITION ===
2 inputSize = 784;
3 hiddenSize = 64;
4 outputSize = 10;
5
6 % === WEIGHTS AND BIASES INITIALIZATION ===
7 W1 = randn(inputSize, hiddenSize) * 0.01;
8 b1 = zeros(1, hiddenSize);
9 W2 = randn(hiddenSize, outputSize) * 0.01;
10 b2 = zeros(1, outputSize);
```

Erläuterung:

- Das Netzwerk hat:
 - Eingabeschicht: 784 Neuronen (28×28 Pixel)
 - Versteckte Schicht: 64 Neuronen
 - Ausgabeschicht: 10 Neuronen (eine pro Ziffer 0-9)
- Die Gewichte werden mit kleinen Zufallswerten initialisiert
- Die Bias-Werte werden auf Null gesetzt

5.3.4 Aktivierungsfunktionen

In diesem Abschnitt werden die verwendeten nichtlinearen Funktionen formal beschrieben.

Sigmoid-Funktion Die Sigmoid-Funktion $\sigma : \mathbb{R} \rightarrow (0, 1)$ ist definiert durch

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \quad (5.71)$$

Diese Funktion transformiert beliebige reelle Eingaben in den Intervall $(0, 1)$ und wird komponentenweise auf Vektoren angewendet.

Ableitung der Sigmoid-Funktion Für das Backpropagation-Verfahren benötigen wir die Ableitung $\sigma'(x)$. Man zeigt durch Ableiten von $\sigma(x)$:

$$\sigma'(x) = \frac{d}{dx} \sigma(x) = \sigma(x) (1 - \sigma(x)). \quad (5.72)$$

Diese Formulierung erlaubt eine effiziente Berechnung, da der Wert von $\sigma(x)$ bereits bekannt ist.

Softmax-Funktion Die Softmax-Funktion $\text{softmax} : \mathbb{R}^K \rightarrow (0, 1)^K$ wandelt einen Eingabevektor $x = (x_1, \dots, x_K)$ in eine Wahrscheinlichkeitsverteilung um:

$$\text{softmax}(x)_i = \frac{\exp(x_i - \max_j x_j)}{\sum_{k=1}^K \exp(x_k - \max_j x_j)} \quad \text{für } i = 1, \dots, K. \quad (5.73)$$

Die Subtraktion von $\max_j x_j$ stabilisiert die Berechnung numerisch.

Code-Ausschnitt:

```

1 % Sigmoid activation function
2 function y = sigmoid(x)
3     y = 1 ./ (1 + exp(-x));
4 end
5
6 % Derivative of the sigmoid function
7 function y = sigmoid_derivative(x)
8     s = sigmoid(x);
9     y = s .* (1 - s);
10 end
11
12 % Softmax to convert output to probabilities
13 function y = softmax(x)
14     expX = exp(x - max(x));
15     y = expX / sum(expX);
16 end
    
```

Erläuterung:

- `sigmoid`: Nichtlineare Aktivierungsfunktion für die versteckte Schicht
- `sigmoid_derivative`: Wird für die Backpropagation benötigt
- `softmax`: Wandelt die Ausgabe in Wahrscheinlichkeiten um (für die Klassifizierung)

5.3.5 Trainingsprozess

Der Trainingsprozess besteht aus einem Forward- und einem Backward-Pass über alle Trainingsdaten, gefolgt von der Gewichtsaktualisierung.

Definition der Verlustfunktion Für eine Trainingsprobe (x, y) mit Vorhersage a_2 verwenden wir die Kreuzentropie:

$$\mathcal{L}(a_2, y) = - \sum_{k=1}^{10} y_k \log(a_{2,k}). \quad (5.74)$$

Forward-Pass Gegeben $x \in \mathbb{R}^{1 \times 784}$ und Parameter W_1, b_1, W_2, b_2 berechnen wir:

$$z_1 = x W_1 + b_1, \quad a_1 = \sigma(z_1), \quad (5.75)$$

$$z_2 = a_1 W_2 + b_2, \quad a_2 = \text{softmax}(z_2). \quad (5.76)$$

Backward-Pass Die Gradienten der Verlustfunktion ergeben sich zu:

$$\delta^{(2)} = a_2 - y, \quad (5.77)$$

$$\frac{\partial \mathcal{L}}{\partial W_2} = a_1^\top \delta^{(2)}, \quad \frac{\partial \mathcal{L}}{\partial b_2} = \delta^{(2)}, \quad (5.78)$$

$$\delta^{(1)} = (\delta^{(2)} W_2^\top) \odot \sigma'(z_1), \quad (5.79)$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = x^\top \delta^{(1)}, \quad \frac{\partial \mathcal{L}}{\partial b_1} = \delta^{(1)}. \quad (5.80)$$

Gewichtsaktualisierung Mit Lernrate η erfolgt das Gradientenabstiegs-Update:

$$W_2 \leftarrow W_2 - \eta \frac{\partial \mathcal{L}}{\partial W_2}, \quad b_2 \leftarrow b_2 - \eta \frac{\partial \mathcal{L}}{\partial b_2}, \quad (5.81)$$

$$W_1 \leftarrow W_1 - \eta \frac{\partial \mathcal{L}}{\partial W_1}, \quad b_1 \leftarrow b_1 - \eta \frac{\partial \mathcal{L}}{\partial b_1}. \quad (5.82)$$

Code-Ausschnitt:

```
1 epochs = 5;
2 lr = 0.1;
3
4 for epoch = 1:epochs
5     for i = 1:numSamples
6         % FORWARD PASS
7         x = X(i, :);
8         y = Y(i, :);
9         z1 = x * W1 + b1;
10        a1 = sigmoid(z1);
11        z2 = a1 * W2 + b2;
12        a2 = softmax(z2);
13
14        % BACKWARD PASS
15        dz2 = a2 - y;
16        dW2 = a1' * dz2;
17        db2 = dz2;
18        dz1 = (dz2 * W2') .* sigmoid_derivative(z1);
19        dW1 = x' * dz1;
20        db1 = dz1;
21
22        % UPDATE WEIGHTS
23        W2 = W2 - lr * dW2;
24        b2 = b2 - lr * db2;
25        W1 = W1 - lr * dW1;
26        b1 = b1 - lr * db1;
27    end
28    fprintf('Epoch %d completed\n', epoch);
29 end
```

Erläuterung:

- Das Netzwerk wird für 5 Epochen trainiert

- Lernrate (lr) ist 0.1
- Für jedes Beispiel:
 - Forward Pass berechnet die Vorhersage
 - Backward Pass berechnet die Gradienten
 - Gewichte werden mittels Gradientenabstieg aktualisiert

5.3.6 Testen des Netzwerks

Evaluation der Performance auf den Testdaten.

Aplanierung der Testdaten Sei $\{I_{\text{test}}^{(i)}\}_{i=1}^{N_{\text{test}}}$ die Testbilder. Wir definieren die Test-Datenmatrix

$$X_{\text{test}} = \begin{bmatrix} \text{vec}(I_{\text{test}}^{(1)})^\top \\ \text{vec}(I_{\text{test}}^{(2)})^\top \\ \vdots \\ \text{vec}(I_{\text{test}}^{(N_{\text{test}})})^\top \end{bmatrix} \in \mathbb{R}^{N_{\text{test}} \times 784}, \quad (5.83)$$

wobei $\text{vec}(I)$ wie zuvor alle Pixel spaltenweise zu einem Vektor der Länge 784 anordnet.

Forward-Pass und Vorhersage Für jedes Testbeispiel i mit Eingabe $x^{(i)} \in \mathbb{R}^{1 \times 784}$ und Parameterpaar $(W_1, b_1), (W_2, b_2)$ wird gerechnet:

$$z_1^{(i)} = x^{(i)} W_1 + b_1, \quad a_1^{(i)} = \sigma(z_1^{(i)}), \quad (5.84)$$

$$z_2^{(i)} = a_1^{(i)} W_2 + b_2, \quad a_2^{(i)} = \text{softmax}(z_2^{(i)}). \quad (5.85)$$

Die vorhergesagte Klasse $\hat{y}^{(i)}$ ergibt sich als Index des maximalen Softmax-Werts:

$$\hat{y}^{(i)} = \arg \max_{1 \leq k \leq 10} (a_{2,k}^{(i)}) - 1. \quad (5.86)$$

Berechnung der Genauigkeit (Accuracy) Seien $\{l_{\text{test}}^{(i)}\}_{i=1}^{N_{\text{test}}}$ die wahren Labels. Die Test-Accuracy ist definiert durch

$$\text{Accuracy} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbf{1}(\hat{y}^{(i)} = l_{\text{test}}^{(i)}), \quad (5.87)$$

wobei $\mathbf{1}(\cdot)$ die Indikatorfunktion ist.

Code-Ausschnitt:

```
1 % === TESTING ===
2 X_test = reshape(testImages, [], size(testImages,3));
3 predictions = zeros(num_test,1);
4
5 for i = 1:num_test
6     x = reshape(X_test(:,i), 1, []);
7     z1 = x * W1 + b1;
8     a1 = sigmoid(z1);
9     z2 = a1 * W2 + b2;
10    a2 = softmax(z2);
11    [~, pred] = max(a2);
12    predictions(i) = pred - 1;
13 end
14
15 accuracy = sum(predictions == double(testLabels)) / num_test;
16 fprintf('Test Accuracy: %.2f%%\n', accuracy * 100);
```

Erläuterung:

- Die Testbilder werden vorbereitet
- Das Netzwerk macht Vorhersagen für jedes Testbild
- Die Genauigkeit wird berechnet und ausgegeben

5.3.7 Visualisierung der Ergebnisse

Darstellung der Vorhersagen für Beispielbilder.

Code-Ausschnitt:

```

1 figure('Name','Predictions on Test Set');
2 for i = 1:12
3     subplot(3,4,i);
4     imshow(testImages(:,:,i));
5     title(['Pred: ', num2str(predictions(i)), ...
6           ' | Real: ', num2str(testLabels(i))]);
7 end

```

Erläuterung:

- 12 Testbilder werden mit den vorhergesagten und tatsächlichen Labels angezeigt

5.3.8 Vollständiger MATLAB-Code

```

1
2 ```matlab
3 clc;
4 clear;
5 trainImages = MNISTLoader.loadImages('train-images.idx3-ubyte');
6 trainLabels = MNISTLoader.loadLabels('train-labels.idx1-ubyte');
7 testImages = MNISTLoader.loadImages('t10k-images.idx3-ubyte');
8 testLabels = MNISTLoader.loadLabels('t10k-labels.idx1-ubyte');
9 figure('Name','Training Set Examples');
10 for i = 1:12
11     subplot(3,4,i);
12     imshow(trainImages(:,:,i));
13     title(['Label: ', num2str(trainLabels(i))]);
14 end
15 X = reshape(trainImages, 28*28, []).';
16 Y = oneHot(trainLabels);
17 disp("Example of one-hot encoding for the first 5 labels:");
18 disp(trainLabels(1:5));
19 disp(Y(1:5,:));
20 inputSize = 784;
21 hiddenSize = 64;
22 outputSize = 10;
23 numSamples = size(X,1);
24 epochs = 5;
25 lr = 0.1;
26 W1 = randn(inputSize, hiddenSize) * 0.01;
27 b1 = zeros(1, hiddenSize);
28 W2 = randn(hiddenSize, outputSize) * 0.01;
29 b2 = zeros(1, outputSize);
30 for epoch = 1:epochs
31     for i = 1:numSamples
32         x = X(i,:);
33         y = Y(i,:);
34         z1 = x * W1 + b1;
35         a1 = sigmoid(z1);

```

```

36     z2 = a1 * W2 + b2;
37     a2 = softmax(z2);
38     dz2 = a2 - y;
39     dW2 = a1' * dz2;
40     db2 = dz2;
41     dz1 = (dz2 * W2') .* sigmoid_derivative(z1);
42     dW1 = x' * dz1;
43     db1 = dz1;
44     W2 = W2 - lr * dW2;
45     b2 = b2 - lr * db2;
46     W1 = W1 - lr * dW1;
47     b1 = b1 - lr * db1;
48 end
49 fprintf('Epoch %d completed\n', epoch);
50 end
51 X_test = reshape(testImages, [], size(testImages,3));
52 num_test = size(X_test, 2);
53 predictions = zeros(num_test,1);
54 for i = 1:num_test
55     x = reshape(X_test(:,i), 1, []);
56     if i == 1
57         disp("Debug info:");
58         disp("x = "); disp(size(x));
59         disp("W1 = "); disp(size(W1));
60     end
61     z1 = x * W1 + b1;
62     a1 = sigmoid(z1);
63     z2 = a1 * W2 + b2;
64     a2 = softmax(z2);
65     [~, pred] = max(a2);
66     predictions(i) = pred - 1;
67 end
68 accuracy = sum(predictions == double(testLabels)) / num_test;
69 fprintf('Test Accuracy: %.2f%%\n', accuracy * 100);
70 disp("Dimensions:");
71 disp("x ="); disp(size(x));
72 disp("W1 ="); disp(size(W1));
73 figure('Name','Predictions on Test Set');
74 for i = 1:12
75     subplot(3,4,i);
76     imshow(testImages(:,:,i));
77     title(['Pred: ', num2str(predictions(i)), ...
78         ' | Real: ', num2str(testLabels(i))]);
79 end
80 function y = sigmoid(x)
81     y = 1 ./ (1 + exp(-x));
82     if isequal(size(x), [1,1]) && x == 0
83         figure('Name','Sigmoid Function');
84         x_vals = -10:0.1:10;
85         y_vals = 1 ./ (1 + exp(-x_vals));
86         plot(x_vals, y_vals, 'LineWidth', 2);
87         grid on;
88         title('Activation Function: Sigmoid');
89         xlabel('x'); ylabel('sigmoid(x)');
90     end
91 end
92 function y = sigmoid_derivative(x)
93     s = sigmoid(x);
94     y = s .* (1 - s);
95 end
96 function y = softmax(x)
97     expX = exp(x - max(x));
98     y = expX / sum(expX);

```

```

99 end
100 function Y = oneHot(labels)
101     numLabels = length(labels);
102     Y = zeros(numLabels,10);
103     for i = 1:numLabels
104         Y(i, labels(i)+1) = 1;
105     end
106 end
107 disp('--- Softmax example with 10 elements (classes 0-9) ---');
108 valores = randn(1,10) * 2;
109 probabilidades = softmax(valores);
110 disp('Input values (logits):');
111 disp(valores);
112 disp('Probability distribution after softmax:');
113 disp(probabilidades);
114 fprintf('Sum of probabilities: %.4f\n', sum(probabilidades));
115 figure('Name','Softmax Distribution - Colors by Class');
116 subplot(1,2,1);
117 colores = lines(10);
118 x_vals = 0:9;
119 for i = 1:10
120     bar(x_vals(i), probabilidades(i), 'FaceColor', colores(i,:), 'EdgeColor','none')
121     ;
122     hold on;
123     text(x_vals(i), probabilidades(i) + 0.02, ...
124         sprintf('%.2f', probabilidades(i)), ...
125         'HorizontalAlignment','center', ...
126         'FontSize', 9, 'FontWeight', 'bold');
127 end
128 title('Softmax: Probabilities per Class (0-9)');
129 xlabel('Class');
130 ylabel('Probability');
131 xticks(0:9);
132 ylim([0, 1.1]);
133 grid on;
134 subplot(1,2,2);
135 bar(1, probabilidades, 'stacked');
136 colormap(lines(10));
137 title('Stacked Bar (Sum = 1)');
138 ylabel('Total Probability');
139 xticks([]);
140 ylim([0, 1]);
141 grid on;
142 legend(arrayfun(@(x) sprintf('Class %d', x), 0:9, 'UniformOutput', false), ...
143         'Location','eastoutside');
144 ```

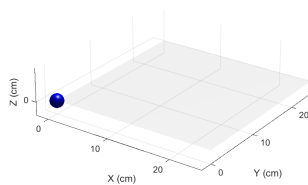
```

Listing 5.2: Neuronales Netzwerk zur Ziffernerkennung

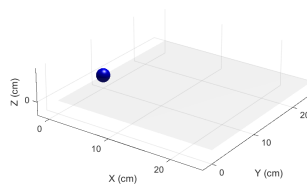
5.4 Neutrales Netzwerk zur Klassifizierung von 3D-Spiralen

Im vorliegenden MATLAB-Skript wird zunächst ein synthetisches Spiralbild-Datenset mit nur 18 Beispielbildern und zugehörigen Labels geladen. Die Bilder werden auf 128×128 Pixel skaliert, in einen 4D-Array umgewandelt und von uint8 in Gleitkommazahlen im Bereich $[0,1]$ konvertiert. Anschließend werden die räumlichen Positionen (x,y,z) und binären Farblabels (Rot/Blau) extrahiert. Zur Veranschaulichung werden sechs zufällige Bilder mit ihren echten Positionen und Farbetiketten in einer 2×3 -Subplot-Figur dargestellt. Danach werden die Bilder zu Vektoren mit 49152 Merkmalen umgeformt und zusammen mit den Labels in Trainings- (80

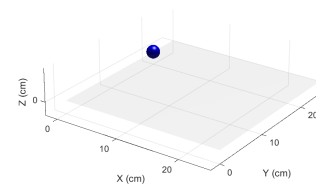
Die Netzwerkstruktur ist ein einfacher, voll vernetzter Feedforward-Netzwerk (Multilayer-Perzeptron) mit einer Eingabeschicht von 49152 Neuronen, einer verdeckten Schicht mit 64 Neuronen und einer Ausgabeschicht mit vier Neuronen (x,y,z,Farbe) . Die Gewichte werden zufällig initialisiert und mit der ReLU- bzw. Sigmoid-Aktivierungsfunktion trainiert (1000 Epochen, Lernrate 0,01) unter Verwendung des mittleren quadratischen Fehlers. Nach dem Training erfolgen Vorhersage und Auswertung auf dem Testset: Die mittlere Positionsabweichung und die Farbklassifikationsgenauigkeit werden berechnet und ausgegeben. Obwohl hier nur 18 synthetische Bilder zur Demonstration verwendet wurden, soll dies eine industrielle Lösung simulieren, die in realen Anwendungen mit Tausenden von Aufnahmen skaliert würde.



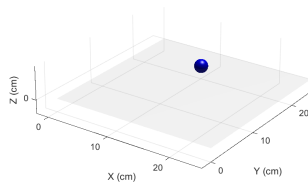
Synthetisches Bild 1



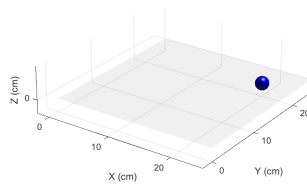
Synthetisches Bild 2



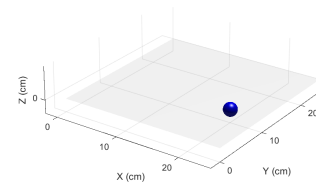
Synthetisches Bild 3



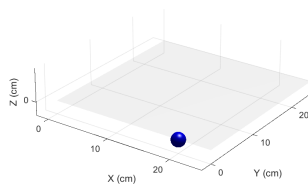
Synthetisches Bild 4



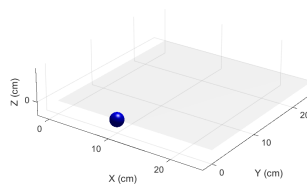
Synthetisches Bild 5



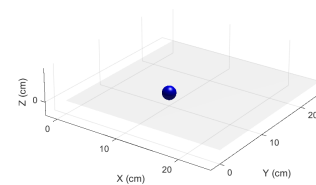
Synthetisches Bild 6



Synthetisches Bild 7



Synthetisches Bild 8



Synthetisches Bild 9

Abbildung 5.6: Neun synthetische Bilder in drei Spalten

5.4.1 Datenladen

In diesem Abschnitt erläutern wir die mathematischen Grundlagen des vorgestellten Codefragments zum Laden und Vorverarbeiten der Bilddaten.

Laden der Bild- und Label-Daten Wir betrachten einen Datensatz mit n Bildern der Größe 128×128 Pixel. Die CSV-Datei enthält zu jedem Bild zwei Informationstypen:

$$\begin{aligned} \text{Positionen: } Y_{\text{pos}} &= [(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)]^\top \in \mathbb{R}^{n \times 2}, \\ \text{Farben: } Y_{\text{color}} &= [c_1, c_2, \dots, c_n]^\top, \quad c_i \in \{\text{rot, blau, grün, ...}\}. \end{aligned}$$

Hier initialisieren wir einen Loader, der die Abmessungen (128×128) vorgibt.

Konstruktion des Bild-Tensors Der Loader erzeugt einen Tensor

$$X_{\text{uint8}} \in \{0, 1, \dots, 255\}^{n \times 128 \times 128 \times 3}, \quad (5.88)$$

wobei die letzte Dimension für die RGB-Kanäle steht. Durch

```
1 X = loader.getImageArray();
```

wird dieser Tensor in MATLAB geladen.

Normalisierung der Pixelwerte Um die Pixelwerte auf den Bereich $[0, 1]$ zu transformieren, wenden wir

$$X = \text{im2double}(X_{\text{uint8}})$$

an. Mathematisch gilt für jedes Pixel $p_{ijkl} \in \{0, \dots, 255\}$:

$$\tilde{p}_{ijkl} = \frac{p_{ijkl}}{255}, \quad (5.89)$$

wodurch $\tilde{p}_{ijkl} \in [0, 1]$ entsteht.

Definition der Zielgrößen Die Positionen Y_{pos} werden als reelle Zahlenpaare gespeichert:

```
1 Ypos = loader.Positions;
```

$$Y_{\text{pos}} \in \mathbb{R}^{n \times 2}.$$

Die Farben werden in eine kategoriale Variable überführt:

```
1 Ycolor = categorical(loader.Colors);
```

$$Y_{\text{color}} = [c_1, \dots, c_n]^\top, \quad c_i \in \{1, 2, \dots, K\}, \quad (5.90)$$

wobei K die Anzahl der eindeutigen Farbklassen im Datensatz bezeichnet.

Durch die obigen Schritte erhalten wir:

$$X \in [0, 1]^{n \times 128 \times 128 \times 3}, \quad Y_{\text{pos}} \in \mathbb{R}^{n \times 2}, \quad Y_{\text{color}} \in \{1, \dots, K\}^n. \quad (5.91)$$

Damit sind die Daten in einer geeigneten Form für nachfolgende Lernalgorithmen aufbereitet.

Code-Ausschnitt:

```
1 clc; clear;
2
3 imageFolder = 'synthetic_spiral_images';
4 csvPath = fullfile(imageFolder, 'labels.csv');
5 loader = SpiralDataLoader(csvPath, imageFolder, [128 128]);
6
7 X = loader.getImageArray();
8 X = im2double(X);
9 Ypos = loader.Positions;
10 Ycolor = categorical(loader.Colors);
```

Erklärung:

- `clc; clear;` leert Konsole und Arbeitsspeicher
- Lädt synthetische Spiralbilder und zugehörige Labels
- Konvertiert die Bilder in den Wertebereich $[0, 1]$ zur Normalisierung

5.4.2 Datenvorverarbeitung

In diesem Abschnitt begründen wir die einzelnen Schritte der Code-Ausschnitte zur Vorbereitung der Eingabedaten X und der Zielgrößen Y sowie deren zufälliges Aufteilen in Trainings- und Testmengen.

Reshaping der Eingabedaten Ursprünglich liegen die Bilder als Tensor

$$X_{\text{orig}} \in \mathbb{R}^{H \times W \times C \times N} \quad (5.92)$$

vor, mit $H = W = 128$, $C = 3$ Farbkanälen und N Stichproben. Durch

```
1 X = reshape(X, [], size(X, 4));
```

wird dieser Tensor in eine Matrix

$$X = \text{reshape}(X_{\text{orig}}, d, N) \quad \text{mit} \quad d = H \cdot W \cdot C = 128 \cdot 128 \cdot 3, \quad (5.93)$$

umgeformt. Damit gilt

$$X \in \mathbb{R}^{d \times N}, \quad (5.94)$$

wobei jede Spalte ein linearisiertes Bild darstellt.

Transponieren und Kodieren der Zielgrößen Die Positionen liegen als Matrix

$$Y_{\text{pos,orig}} \in \mathbb{R}^{N \times 2} \quad (5.95)$$

vor. Mit

```
1 Ypos = Ypos';
```

erhalten wir

$$Y_{\text{pos}} = Y_{\text{pos,orig}}^{\top} \in \mathbb{R}^{2 \times N}. \quad (5.96)$$

Die Farbklasse Y_{color} wird von einer kategorischen Variable in einen numerischen Vektor überführt:

```
1 Ycolor = double(Ycolor)';
```

$$Y_{\text{color}} \in \{1, \dots, K\}^{1 \times N}. \quad (5.97)$$

Durch

```
1 Y = [Ypos; Ycolor];
```

erhalten wir schließlich

$$Y = \begin{pmatrix} Y_{\text{pos}} \\ Y_{\text{color}} \end{pmatrix} \in \mathbb{R}^{(2+1) \times N} = \mathbb{R}^{3 \times N}. \quad (5.98)$$

Zufällige Permutation und Split in Training/Test Sei

$$N = \text{size}(X, 2) \quad (5.99)$$

die Gesamtzahl der Datenpunkte. Mit

```
1 idx = randperm(N);
2 train_count = round(0.8 * N);
```

wird eine Zufallspermutation $\pi = (i_1, \dots, i_N)$ der Indizes und die Anzahl der Trainingsbeispiele

$$N_{\text{train}} = \lceil 0.8 N \rceil \quad (5.100)$$

bestimmt. Der Datensatz wird dann wie folgt aufgeteilt:

$$\begin{aligned} X_{\text{train}} &= X(:, \pi(1 : N_{\text{train}})), & Y_{\text{train}} &= Y(:, \pi(1 : N_{\text{train}})), \\ X_{\text{test}} &= X(:, \pi(N_{\text{train}} + 1 : N)), & Y_{\text{test}} &= Y(:, \pi(N_{\text{train}} + 1 : N)). \end{aligned} \quad (5.101)$$

Damit liegen die Trainings- und Testmengen in zufällig gemischter Form und im passenden Format für das Netzwerktraining vor.“

Code-Ausschnitt:

```
1 X = reshape(X, [], size(X, 4));
2
3 Ypos = Ypos';
4 Ycolor = double(Ycolor)';
5 Y = [Ypos; Ycolor];
6
7 N = size(X, 2);
8 idx = randperm(N);
9 train_count = round(0.8 * N);
10
11 Xtrain = X(:, idx(1:train_count));
12 Ytrain = Y(:, idx(1:train_count));
13 Xtest = X(:, idx(train_count+1:end));
14 Ytest = Y(:, idx(train_count+1:end));
```

Erklärung:

- Bilder werden zu Vektoren der Länge $128 \times 128 \times 3 = 49152$ abgeflacht
- Labels werden in Positions- und Farbkomponenten organisiert
- Aufteilung in Trainingsdaten (80 %) und Testdaten (20 %)

5.4.3 Netzwerkarchitektur

In diesem Abschnitt erläutern wir die Wahl der Dimensionen und die Initialisierung der Gewichte und Biases des zweischichtigen, vollständig verbundenen Netzes.

Dimensionierung der Gewichte und Biases Die Größen der Matrizen und Vektoren ergeben sich direkt aus den definierten Layer-Dimensionen:

$$\begin{aligned} \text{Input-Dimension: } d &= \text{input_size}, \\ \text{Hidden-Dimension: } H &= \text{hidden_size}, \\ \text{Output-Dimension: } K &= \text{output_size}. \end{aligned}$$

Daraus folgen die Formen der Parameter:

$$\begin{aligned} W^{(1)} &\in \mathbb{R}^{H \times d}, & b^{(1)} &\in \mathbb{R}^H, \\ W^{(2)} &\in \mathbb{R}^{K \times H}, & b^{(2)} &\in \mathbb{R}^K. \end{aligned} \quad (5.102)$$

Zufällige Initialisierung der Gewichte Um die Symmetrie zwischen den Neuronen zu brechen und zu verhindern, dass alle Einheiten identische Gradienten erhalten, initialisieren wir die Gewichte mit unabhängigen, normalverteilten Zufallswerten mit kleinem Varianzfaktor σ^2 :

$$\begin{aligned} W_{ij}^{(1)} &\sim \mathcal{N}(0, \sigma^2), & \sigma = 0,01, \\ W_{kl}^{(2)} &\sim \mathcal{N}(0, \sigma^2), & \sigma = 0,01. \end{aligned} \quad (5.103)$$

Initialisierung der Bias-Vektoren Die Bias-Vektoren werden mit Null initialisiert, um das Netz ohne systematischen Versatz starten zu lassen:

$$b_i^{(1)} = 0, \quad \forall i = 1, \dots, H, \quad (5.104)$$

$$b_k^{(2)} = 0, \quad \forall k = 1, \dots, K. \quad (5.105)$$

Mit dieser Konfiguration erhält man ein zweischichtiges Perzeptron mit Parameterraum

$$\Theta = \{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\} \quad (5.106)$$

in den Dimensionen

$$W^{(1)} \in \mathbb{R}^{H \times d}, \quad b^{(1)} \in \mathbb{R}^H, \quad W^{(2)} \in \mathbb{R}^{K \times H}, \quad b^{(2)} \in \mathbb{R}^K. \quad (5.107)$$

das für das Training mittels Gradientenverfahren bereitsteht.

Code-Ausschnitt:

```
1 input_size = size(Xtrain, 1);
2 hidden_size = 64;
3 output_size = 4;
4
5 W1 = randn(hidden_size, input_size) * 0.01;
6 b1 = zeros(hidden_size, 1);
7 W2 = randn(output_size, hidden_size) * 0.01;
8 b2 = zeros(output_size, 1);
```

Erklärung:

- Vollständig verbundenes Netz mit
 - Eingabeschicht: 49152 Neuronen
 - Verborgene Schicht: 64 Neuronen
 - Ausgabeschicht: 4 Neuronen (3 für x, y, z , 1 für Farbe)
- Gewichte werden mit kleinen Zufallswerten initialisiert

5.4.4 Aktivierungsfunktionen

In diesem Abschnitt erläutern wir die mathematischen Eigenschaften und Ableitungen der verwendeten Aktivierungsfunktionen ReLU und Sigmoid.

ReLU (Rectified Linear Unit) Die ReLU-Funktion wird definiert als

$$y = \text{ReLU}(x) = \max(0, x). \quad (5.108)$$

Diese Funktion besitzt folgende Ableitung:

$$\frac{d}{dx} \text{ReLU}(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0. \end{cases} \quad (5.109)$$

Im Code wird dies umgesetzt durch

```

1 y = max(0, x);
2 dy = double(x > 0);

```

Die diskrete Indikatorfunktion $\mathbf{1}_{\{x>0\}}$ sorgt dafür, dass nur positive Eingaben die Identität weitergeben, negative Eingaben werden abgeschnitten. Dies führt zu sparsamen Aktivierungen und vermindert das Vanishing-Gradient-Problem für $x > 0$.

Sigmoid Die Sigmoid-Funktion ist gegeben durch

$$y = \sigma(x) = \frac{1}{1 + e^{-x}}. \quad (5.110)$$

Ihre Ableitung kann aus der Funktion selbst ausgedrückt werden:

$$\frac{d}{dx} \sigma(x) = \sigma(x) (1 - \sigma(x)). \quad (5.111)$$

Im Code wird dies realisiert durch

```

1 y = 1 ./ (1 + exp(-x));
2 s = sigmoid_func(x);
3 dy = s .* (1 - s);

```

Die Sigmoid-Funktion komprimiert beliebige reelle Eingaben in das Intervall $(0, 1)$ und eignet sich insbesondere für binäre Klassifikationsaufgaben, da ihr Wertebereich der Interpretation als Wahrscheinlichkeiten entspricht.

Beide Aktivierungsfunktionen sind differenzierbar (bis auf eine Unstetigkeitsstelle bei ReLU) und erlauben eine einfache, effiziente Implementierung ihrer Gradienten. ReLU beschleunigt das Training durch sparsames Aktivierungsverhalten, Sigmoid bietet eine glatte Wahrscheinlichkeitsinterpretation.

Code-Ausschnitt:

```

1 function y = relu_func(x)
2     y = max(0, x);
3 end
4
5 function dy = drelu_func(x)
6     dy = double(x > 0);
7 end
8
9 function y = sigmoid_func(x)
10    y = 1 ./ (1 + exp(-x));
11 end
12
13 function dy = dsigmoid_func(x)
14    s = sigmoid_func(x);
15    dy = s .* (1 - s);
16 end

```

Erklärung:

- `relu_func`: ReLU für verborgene Schichten
- `sigmoid_func`: Sigmoid für binäre Farbausgabe
- Ableitungen für Backpropagation werden bereitgestellt

5.4.5 Training

In diesem Abschnitt erläutern wir die mathematischen Grundlagen der Forward- und Backward-Propagation sowie des Parameter-Updates.

Forward-Propagation Für jede Epoche berechnen wir zunächst die Vorwärtsdurchläufe:

$$\begin{aligned} Z^{(1)} &= W^{(1)} X_{\text{train}} + b^{(1)}, \\ A^{(1)} &= \text{ReLU}(Z^{(1)}), \\ Z^{(2)} &= W^{(2)} A^{(1)} + b^{(2)}, \\ A^{(2)} &= \begin{pmatrix} Z_{1:K-1, :}^{(2)} \\ \sigma(Z_{K, :}^{(2)}) \end{pmatrix}, \end{aligned} \quad (5.112)$$

wobei für die letzten K -ten Ausgabeneuronen Sigmoid aktiviert wird.

Mittelwert-Quadrat-Fehler (MSE) Die Verlustfunktion ist der mittlere quadratische Fehler über alle Ausgaben und Beispiele:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^K (A_{j,i}^{(2)} - Y_{\text{train},j,i})^2. \quad (5.113)$$

Backward-Propagation Wir bestimmen die Gradienten folgendermaßen:

$$D^{(2)} = \frac{\partial \mathcal{L}}{\partial Z^{(2)}} = \frac{2}{N} (A^{(2)} - Y_{\text{train}}), \quad (5.114)$$

$$D_{K, :}^{(2)} = D_{K, :}^{(2)} \cdot \sigma'(Z_{K, :}^{(2)}), \quad (5.115)$$

Gradientes de la capa de salida:

$$\nabla_{W^{(2)}} \mathcal{L} = D^{(2)} (A^{(1)})^\top, \quad \nabla_{b^{(2)}} \mathcal{L} = \sum_{i=1}^N D_{:, i}^{(2)}, \quad (5.116)$$

Propagación hacia atrás:

$$D^{(1)} = (W^{(2)})^\top D^{(2)}, \quad (5.117)$$

$$D^{(1)} = D^{(1)} \cdot \text{ReLU}'(Z^{(1)}), \quad (5.118)$$

Gradientes de la capa oculta:

$$\nabla_{W^{(1)}} \mathcal{L} = D^{(1)} X_{\text{train}}^\top, \quad \nabla_{b^{(1)}} \mathcal{L} = \sum_{i=1}^N D_{:, i}^{(1)}. \quad (5.119)$$

Gradientenabstieg Die Parameter werden mit Lernrate η (im Code: `lr`) aktualisiert:

$$W^{(1)} \leftarrow W^{(1)} - \eta \nabla_{W^{(1)}} \mathcal{L}, \quad (5.120)$$

$$b^{(1)} \leftarrow b^{(1)} - \eta \nabla_{b^{(1)}} \mathcal{L}, \quad (5.121)$$

$$W^{(2)} \leftarrow W^{(2)} - \eta \nabla_{W^{(2)}} \mathcal{L}, \quad (5.122)$$

$$b^{(2)} \leftarrow b^{(2)} - \eta \nabla_{b^{(2)}} \mathcal{L}. \quad (5.123)$$

Damit wird in jedem Trainingsschritt der Parameterraum in Richtung des negativen Gradienten bewegt, um den Verlust zu minimieren.

Code-Ausschnitt:

```

1 epochs = 1000;
2 lr      = 0.01;
3
4 for epoch = 1:epochs
5     Z1 = W1 * Xtrain + b1;
6     A1 = relu_func(Z1);
7     Z2 = W2 * A1 + b2;
8     A2 = Z2;
9     A2(end, :) = sigmoid_func(Z2(end, :));
10
11     loss = mean((A2 - Ytrain).^2, 'all');
12
13     dZ2 = 2 * (A2 - Ytrain);
14     dZ2(end, :) = dZ2(end, :) .* dsigmoid_func(Z2(end, :));
15     dW2 = dZ2 * A1' / size(Xtrain, 2);
16     db2 = sum(dZ2, 2) / size(Xtrain, 2);
17
18     dA1 = W2' * dZ2;
19     dZ1 = dA1 .* drelu_func(Z1);
20     dW1 = dZ1 * Xtrain' / size(Xtrain, 2);
21     db1 = sum(dZ1, 2) / size(Xtrain, 2);
22
23     W1 = W1 - lr * dW1;
24     b1 = b1 - lr * db1;
25     W2 = W2 - lr * dW2;
26     b2 = b2 - lr * db2;
27
28     if mod(epoch, 100) == 0
29         fprintf('Epoch %d, Loss = %.4f\n', epoch, loss);
30     end
31 end
    
```

Erklärung:

- 1000 Epochen, Lernrate 0,01
- Forward-Pass: Vorhersagen
- Backward-Pass: Gradienten
- Gewichtsaktualisierung via Gradientenabstieg
- Verlust: mittlerer quadratischer Fehler

5.4.6 Auswertung

In diesem Abschnitt erklären wir die Berechnung der Modellvorhersagen und der Evaluationsmetriken auf dem Testdatensatz.

Forward-Propagation im Test

$$Z_{\text{test}}^{(1)} = W^{(1)} X_{\text{test}} + b^{(1)}, \quad (5.124)$$

$$A_{\text{test}}^{(1)} = \text{ReLU}(Z_{\text{test}}^{(1)}), \quad (5.125)$$

$$Z_{\text{test}}^{(2)} = W^{(2)} A_{\text{test}}^{(1)} + b^{(2)}, \quad (5.126)$$

$$Z_{K,\text{test}}^{(2)} = \sigma(Z_{K,\text{test}}^{(2)}), \quad (5.127)$$

wobei nur die letzte Ausgabedimension K (Farbklassifikation) durch die Sigmoid-Funktion transformiert wird.

Vorhersage von Position und Farbe Die Positionen werden direkt aus den ersten drei Ausgabekanälen entnommen:

$$\hat{Y}_{\text{pos}} = Z_{1:3,\text{test}}^{(2)}, \quad (5.128)$$

die Farbvorhersage erfolgt durch Thresholding bei 0,5:

$$\hat{Y}_{\text{color},i} = \begin{cases} 1, & Z_{4,i}^{(2)} > 0,5, \\ 0, & \text{sonst.} \end{cases} \quad (5.129)$$

Berechnung des Positionsfehlers Der mittlere euklidische Fehler über alle N_{test} Testbeispiele ist gegeben durch

$$\text{pos_error} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \|\hat{Y}_{\text{pos},i} - Y_{\text{test,pos},i}\|_2. \quad (5.130)$$

Berechnung der Farbgenauigkeit Die Farbgenauigkeit als Anteil korrekt klassifizierter Beispiele ist

$$\text{acc_color} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbf{1}(\hat{Y}_{\text{color},i} = Y_{\text{test,color},i}), \quad (5.131)$$

wobei $\mathbf{1}(\cdot)$ die Indikatorfunktion ist.

Damit lassen sich die Testmetriken Positionsfehler und Farbgenauigkeit interpretierbar ausgeben.“

Code-Ausschnitt:

```
1 Z1_test = W1 * Xtest + b1;
2 A1_test = relu_func(Z1_test);
3 Z2_test = W2 * A1_test + b2;
4 Z2_test(end, :) = sigmoid_func(Z2_test(end, :));
5
6 Ypred_pos = Z2_test(1:3, :);
7 Ypred_color = Z2_test(4, :) > 0.5;
8
9 pos_error = mean(vecnorm(Ypred_pos - Ytest(1:3, :)));
10 acc_color = mean(Ypred_color == Ytest(4, :));
11
12 fprintf('Positionsfehler: %.2f\n', pos_error);
13 fprintf('Farbgenauigkeit: %.2f%%\n', acc_color * 100);
```

Erklärung:

- Positionsfehler: mittlere euklidische Distanz
- Farbgenauigkeit: Anteil korrekter Klassifikationen

5.4.7 Ergebnisvisualisierung

In diesem Abschnitt erklären wir die Schritte zur Umformung der Testdaten in Bilddarstellungen, die zufällige Auswahl von Beispielen und die Zuordnung von realen und vorhergesagten Werten.

Rücktransformation der Bildvektoren Die Testdaten liegen als Matrix

$$X_{\text{test}} \in \mathbb{R}^{d \times N_{\text{test}}}$$

vor, mit $d = 128 \cdot 128 \cdot 3$ und N_{test} Testbeispielen. Durch

```
1 Xtest_imgs = reshape(Xtest, [128, 128, 3, size(Xtest, 2)]);
```

wird diese Matrix in einen Tensor umgeformt:

$$X_{\text{test_imgs}} = \text{reshape}(X_{\text{test}}, 128, 128, 3, N_{\text{test}}) \in \mathbb{R}^{128 \times 128 \times 3 \times N_{\text{test}}}, \quad (5.132)$$

wobei jeder Vektor in der Spalte zu einem RGB-Bild der Größe 128×128 rekonstruiert wird.

Zufällige Auswahl von Testbeispielen Mit

```
1 idx_show = randperm(size(Xtest, 2), 3);
```

wird eine Zufallspermutation π über die Indizes $\{1, \dots, N_{\text{test}}\}$ erzeugt und die ersten drei Indizes ausgewählt:

$$\{\ell_1, \ell_2, \ell_3\} = \{\pi(1), \pi(2), \pi(3)\}, \quad \ell_i \in \{1, \dots, N_{\text{test}}\}. \quad (5.133)$$

Extraktion und Zuordnung von realen und vorhergesagten Werten Für jedes ausgewählte Beispiel ℓ_i gilt:

$$\text{Bild: } X_{\text{img},i} = X_{\text{test_imgs}}(:, :, :, \ell_i). \quad (5.134)$$

$$\text{Reale Position: } Y_{\text{real,pos},i} = Y_{\text{test}}(1 : 3, \ell_i), \quad (5.135)$$

$$\text{Reale Farbe: } Y_{\text{real,color},i} = Y_{\text{test}}(4, \ell_i), \quad (5.136)$$

$$\text{Vorhergesagte Position: } Y_{\text{pred,pos},i} = Y_{\text{pred_pos}}(:, \ell_i), \quad (5.137)$$

$$\text{Vorhergesagte Farbe: } Y_{\text{pred,color},i} = Y_{\text{pred_color}}(\ell_i). \quad (5.138)$$

Darstellung Die Funktion `imshow` zeigt das rekonstruierte Bild $X_{\text{img},i}$, und die Titel werden mit den Vektoren $[Y_{\text{real,pos},i}, Y_{\text{real,color},i}]$ sowie $[Y_{\text{pred,pos},i}, Y_{\text{pred,color},i}]$ beschriftet. Damit werden Vorhersage und Ground-Truth anschaulich verglichen.

Code-Ausschnitt:

```
1 Xtest_imgs = reshape(Xtest, [128, 128, 3, size(Xtest, 2)]);
2 idx_show = randperm(size(Xtest, 2), 3);
3
4 figure('Name', 'Vorhersagen vs. Echt (Test)');
5 for i = 1:3
6     subplot(1, 3, i);
7     imshow(Xtest_imgs(:, :, :, idx_show(i)));
8
9     pos_real = Ytest(1:3, idx_show(i));
10    color_real = Ytest(4, idx_show(i));
11    pos_pred = Ypred_pos(:, idx_show(i));
12    color_pred = Ypred_color(idx_show(i));
13
14    title({
15        sprintf('Echt: [%0f,%0f,%0f] - %s', ...
16            pos_real(1), pos_real(2), pos_real(3), colorText(color_real)),
17        sprintf('Vorh.: [%0f,%0f,%0f] - %s', ...
18            pos_pred(1), pos_pred(2), pos_pred(3), colorText(color_pred))
19    });
20 end
```

Erklärung:

- Zeigt 3 zufällige Testbilder
- Vergleich von echten und vorhergesagten Positionen und Farben
- Intuitive Darstellung der Modellleistung

5.4.8 Vollständiger MATLAB-Code

```
1
2 clc; clear;
3
4 imageFolder = 'synthetic_spiral_images';
5 csvPath = fullfile(imageFolder, 'labels.csv');
6 loader = SpiralDataLoader(csvPath, imageFolder, [128 128]);
7
8 X = loader.getImageArray();
9 X = im2double(X);
10 Ypos = loader.Positions;
11 Ycolor = categorical(loader.Colors);
12
13 X = reshape(X, [], size(X, 4));
14
15 Ypos = Ypos';
16 Ycolor = double(Ycolor)';
17 Y = [Ypos; Ycolor];
18
19 N = size(X, 2);
20 idx = randperm(N);
21 train_count = round(0.8 * N);
22
23 Xtrain = X(:, idx(1:train_count));
24 Ytrain = Y(:, idx(1:train_count));
25 Xtest = X(:, idx(train_count+1:end));
26 Ytest = Y(:, idx(train_count+1:end));
27
28 input_size = size(Xtrain, 1);
29 hidden_size = 64;
30 output_size = 4;
31
32 W1 = randn(hidden_size, input_size) * 0.01;
33 b1 = zeros(hidden_size, 1);
34 W2 = randn(output_size, hidden_size) * 0.01;
35 b2 = zeros(output_size, 1);
36
37 epochs = 1000;
38 lr = 0.01;
39
40 for epoch = 1:epochs
41     Z1 = W1 * Xtrain + b1;
42     A1 = relu_func(Z1);
43     Z2 = W2 * A1 + b2;
44     A2 = Z2;
45     A2(end, :) = sigmoid_func(Z2(end, :));
46
47     loss = mean((A2 - Ytrain).^2, 'all');
48
49     dZ2 = 2 * (A2 - Ytrain);
50     dZ2(end, :) = dZ2(end, :) .* dsigmoid_func(Z2(end, :));
51     dW2 = dZ2 * A1' / size(Xtrain, 2);
52     db2 = sum(dZ2, 2) / size(Xtrain, 2);
53
54     dA1 = W2' * dZ2;
55     dZ1 = dA1 .* drelu_func(Z1);
56     dW1 = dZ1 * Xtrain' / size(Xtrain, 2);
57     db1 = sum(dZ1, 2) / size(Xtrain, 2);
58
59     W1 = W1 - lr * dW1;
60     b1 = b1 - lr * db1;
61     W2 = W2 - lr * dW2;
```

```

62     b2 = b2 - lr * db2;
63
64     if mod(epoch, 100) == 0
65         fprintf('Epoch %d, Loss = %.4f\n', epoch, loss);
66     end
67 end
68
69 Z1_test = W1 * Xtest + b1;
70 A1_test = relu_func(Z1_test);
71 Z2_test = W2 * A1_test + b2;
72 Z2_test(end, :) = sigmoid_func(Z2_test(end, :));
73
74 Ypred_pos = Z2_test(1:3, :);
75 Ypred_color = Z2_test(4, :) > 0.5;
76
77 pos_error = mean(vecnorm(Ypred_pos - Ytest(1:3, :)));
78 acc_color = mean(Ypred_color == Ytest(4, :));
79
80 fprintf('Position error: %.2f\n', pos_error);
81 fprintf('Color accuracy: %.2f%%\n', acc_color * 100);
82
83 Xtest_imgs = reshape(Xtest, [128, 128, 3, size(Xtest, 2)]);
84 idx_show = randperm(size(Xtest, 2), 3);
85
86 figure('Name', 'Predictions vs Actual (Test)');
87 for i = 1:3
88     subplot(1, 3, i);
89     imshow(Xtest_imgs(:, :, :, idx_show(i)));
90
91     pos_real = Ytest(1:3, idx_show(i));
92     color_real = Ytest(4, idx_show(i));
93     pos_pred = Ypred_pos(:, idx_show(i));
94     color_pred = Ypred_color(idx_show(i));
95
96     title({
97         sprintf('Real: [%0f,%0f,%0f] - %s', ...
98             pos_real(1), pos_real(2), pos_real(3), colorText(color_real)),
99         sprintf('Pred: [%0f,%0f,%0f] - %s', ...
100             pos_pred(1), pos_pred(2), pos_pred(3), colorText(color_pred))
101     });
102 end
103
104 function y = relu_func(x)
105     y = max(0, x);
106 end
107
108 function dy = drelu_func(x)
109     dy = double(x > 0);
110 end
111
112 function y = sigmoid_func(x)
113     y = 1 ./ (1 + exp(-x));
114 end
115
116 function dy = dsigmoid_func(x)
117     s = sigmoid_func(x);
118     dy = s .* (1 - s);
119 end

```

Listing 5.3: Neutrales Netzwerk zur Klassifizierung von 3D-Spiralen

5.5 Automatische Durchmesserbestimmung mit Sobel-Kantendetektion

In diesem Code wird die Kantendetektion anhand des Sobel-Operators demonstriert, sowohl manuell als auch mit der `conv2`-Funktion. Zuerst wird ein Graustufenbild geladen und normalisiert. Anschließend werden die horizontalen und vertikalen Gradienten (G_x , G_y) berechnet, indem eine 3×3 -Nachbarschaft pixelweise mit den Sobel-Filtern gefaltet wird. Die Magnitude des Gradienten ergibt sich aus der euklidischen Norm der Einzelgradienten ($\sqrt{G_x^2 + G_y^2}$), und durch Schwellenwertbildung (`threshold = 0.2`) werden die Kanten segmentiert. Die automatische Durchmesserbestimmung nutzt morphologische Operationen (`imfill`, `bwareaopen`), um Rauschen zu entfernen, und berechnet den äquivalenten Durchmesser über `regionprops`, skaliert mit einer realen Auflösung (0,25 mm/Pixel). Die Ergebnisse werden visuell dargestellt, einschließlich der gemessenen Abmessungen.

Der Fokus liegt auf der mathematischen Anwendung der Faltung für die Gradientenberechnung und der Kombination von partiellen Ableitungen zur Kantenerkennung. Die manuelle Implementierung zeigt dabei explizit die diskrete Faltungsoperation, während `conv2` dies optimiert umsetzt. Die Schwellenwertbildung und Nachbearbeitung illustrieren den Übergang vom Gradienten zur binären Kantenkarte, die für metrische Analysen genutzt wird.

Anmerkung zur praktischen Anwendung:

Dieser Algorithmus simuliert, wie in der industriellen Bildverarbeitung reale Messungen durchgeführt werden können. Durch die Kalibrierung mit einer bekannten Skalierung (hier 0,25 mm/Pixel) lassen sich präzise Maße wie Durchmesser oder Abstände in technischen Komponenten automatisiert erfassen. Solche Methoden kommen beispielsweise in der Qualitätskontrolle oder Robotik zum Einsatz, wo pixelgenaue Kantendetektion in metrische Daten umgewandelt werden muss.



Abbildung 5.7: Kantendetektion an Zahnrädern

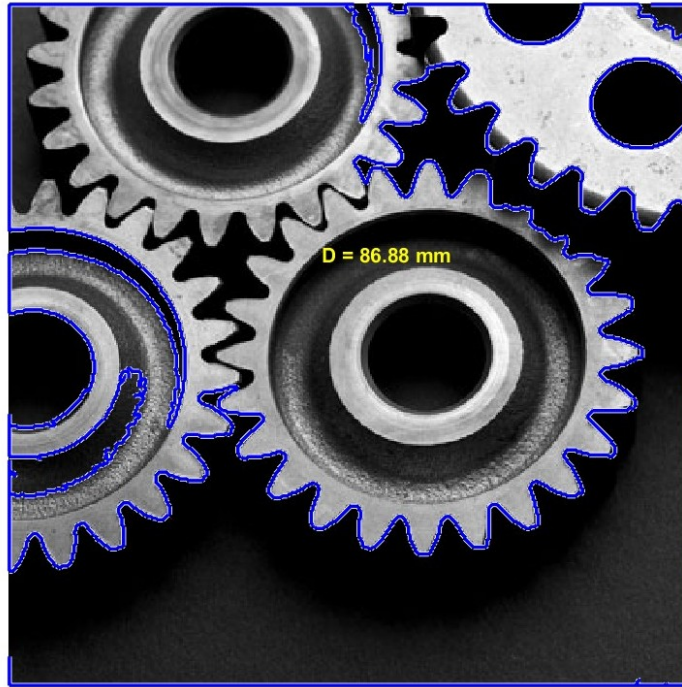


Abbildung 5.8: Geschlossene Konturmessung mit Kantenerkennungsalgorithmus

5.5.1 Bild laden und vorverarbeiten

Laden und Normalisierung Durch

```
1 img = imread(img_url);
2 img = im2double(rgb2gray(img));
```

wird zunächst das Farbbild geladen und in Graustufen umgewandelt. Die Grauwertbildung erfolgt pixelweise nach

$$I_{\text{gray},ij} = 0,2989 R_{ij} + 0,5870 G_{ij} + 0,1140 B_{ij}, \quad (5.139)$$

wobei $R_{ij}, G_{ij}, B_{ij} \in \{0, \dots, 255\}$ die Farbkanäle des ursprünglichen Bildes sind. Anschließend werden die Werte in den Bereich $[0, 1]$ skaliert:

$$I_{ij} = \frac{I_{\text{gray},ij}}{255} \in [0, 1]. \quad (5.140)$$

Bestimmung der Bilddimensionen Mit

```
1 [h, w] = size(img);
```

werden die Höhe und Breite des Graustufenbildes ermittelt:

$$[h, w] = (\# \text{Zeilen}(I), \# \text{Spalten}(I)), \quad (5.141)$$

wobei $I \in [0, 1]^{h \times w}$ das normalisierte Graustufenbild darstellt.

Code-Ausschnitt:

```
1 % === 1. Load image and normalize ===
2 img_url = 'https://irp.cdn-website.com/1fec50a0/MOBILE/jpg/16395.jpg';
3 img = imread(img_url);
4 img = im2double(rgb2gray(img));
5 [h, w] = size(img);
```

Erklärung:

- Bild wird von URL geladen
- Konvertierung zu Graustufen und Normalisierung auf [0,1]
- Speichern der Bildabmessungen in h (Höhe) und w (Breite)

5.5.2 Sobel-Filter definieren

In diesem Abschnitt erklären wir, wie die Sobel-Kerne zur Kantendetektion als diskrete Approximation der Bildgradienten dienen.

Finite-Differenzen-Approximation der Ableitungen Der Gradient einer kontinuierlichen Bildfunktion $I(x, y)$ lässt sich durch partielle Ableitungen in x - und y -Richtung beschreiben. Diskret approximieren wir diese Ableitungen durch endliche Differenzen:

$$\frac{\partial I}{\partial x}(i, j) \approx I(i+1, j) - I(i-1, j), \quad (5.142)$$

$$\frac{\partial I}{\partial y}(i, j) \approx I(i, j+1) - I(i, j-1). \quad (5.143)$$

Kombination von Glättung und Differenzbildung Der Sobel-Filter erweitert die reine Differenzbildung um eine eindimensionale Glättung in orthogonaler Richtung, um Rauschen zu unterdrücken. Die beiden Filterkerne lauten:

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad (5.144)$$

$$G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}. \quad (5.145)$$

Hierbei berechnet G_x eine geglättete horizontale Ableitung und G_y eine geglättete vertikale Ableitung. Die mittleren Zeilen-/Spaltengewichte von ± 2 verstärken den Einfluss der unmittelbaren Nachbarn.

Faltung mit dem Bild Durch diskrete Faltung erhält man die Gradientenbilder:

$$I_x = I * G_x, \quad I_y = I * G_y, \quad (5.146)$$

wobei $*$ die zweidimensionale Faltungsoperation bezeichnet.

Kantendetektion Die Kantstärke im Punkt (i, j) ergibt sich dann aus der Magnitude des Gradientenvektors:

$$\|\nabla I(i, j)\| = \sqrt{I_x(i, j)^2 + I_y(i, j)^2}. \quad (5.147)$$

Somit liefern die Sobel-Filter robuste Approximationen der Bildkanten durch Kombination von Glättung und Differenzbildung.““

Code-Ausschnitt:

```
1 % === 2. Sobel filters ===
2 Gx = [-1 0 1; -2 0 2; -1 0 1]; % Horizontal derivative
3 Gy = [-1 -2 -1; 0 0 0; 1 2 1]; % Vertical derivative
```

Erklärung:

- Gx: Detektiert vertikale Kanten
- Gy: Detektiert horizontale Kanten
- Beide Filter approximieren den Bildgradienten

5.5.3 Manuelle Gradientenberechnung

In diesem Abschnitt leiten wir her, wie die Gradienten mit diskreter, manueller Faltung berechnet werden.

Initialisierung Wir definieren zwei Matrizen für die Gradienten in x - und y -Richtung:

$$\text{grad_x_manual} = 0_{h \times w}, \quad \text{grad_y_manual} = 0_{h \times w}. \quad (5.148)$$

Lokale Regionsextraktion Für jedes Pixel (i, j) im Inneren des Bildes (ohne Rand) betrachten wir die 3×3 -Region

$$\text{region}_{i,j} = I[i-1:i+1, j-1:j+1] \in \mathbb{R}^{3 \times 3}. \quad (5.149)$$

Diskrete Faltung Die Gradienten werden durch Summation der gewichteten Nachbarn berechnet:

$$\text{grad_x_manual}(i, j) = \sum_{u=1}^3 \sum_{v=1}^3 \text{region}_{i,j}(u, v) \cdot G_x(u, v), \quad (5.150)$$

$$\text{grad_y_manual}(i, j) = \sum_{u=1}^3 \sum_{v=1}^3 \text{region}_{i,j}(u, v) \cdot G_y(u, v). \quad (5.151)$$

Hierbei entsprechen G_x und G_y den Sobel-Kernen für horizontale bzw. vertikale Ableitungen.

```
1 for i = 2:h-1
2     for j = 2:w-1
3         region = img(i-1:i+1, j-1:j+1);
4         grad_x_manual(i, j) = sum(sum(region .* Gx));
5         grad_y_manual(i, j) = sum(sum(region .* Gy));
6     end
7 end
```

Durch diese doppelte Schleife wird für jeden inneren Bildpunkt die Faltung mit dem jeweiligen Filterkern direkt berechnet.

Code-Ausschnitt:

```

1 grad_x_manual = zeros(h, w);
2 grad_y_manual = zeros(h, w);
3 for i = 2:h-1
4     for j = 2:w-1
5         region = img(i-1:i+1, j-1:j+1);
6         grad_x_manual(i, j) = sum(sum(region .* Gx));
7         grad_y_manual(i, j) = sum(sum(region .* Gy));
8     end
9 end

```

Erklärung:

- Schleife über alle inneren Pixel
- Anwendung der Sobel-Filter durch punktweise Multiplikation
- Speichert horizontale und vertikale Gradienten

5.5.4 Gradientenberechnung mit conv2

Diskrete Faltung als Kompaktoperator Die Funktion `conv2` berechnet die zweidimensionale diskrete Faltung des Bildes I mit dem Filterkern G . Für den Kern G_x gilt

$$(I * G_x)[i, j] = \sum_{u=-1}^1 \sum_{v=-1}^1 G_x(u+2, v+2) I(i-u, j-v), \quad (5.152)$$

wobei die Indizes so verschoben sind, dass der Kernel-Mittelpunkt auf das Pixel (i, j) gelegt wird.

„Same“-Option und Randausgleich Mit der Option `'same'` wird das Ergebnis auf die gleiche Größe wie das Eingangsbild gebracht, indem am Rand Nullen (oder eine andere standardmäßige Randbedingung) hinzugefügt werden:

$$\text{size}(I * G_x, 'same') = \text{size}(I). \quad (5.153)$$

Effiziente Berechnung Anstelle der manuellen Doppelschleifen erfolgt die Faltung hier als optimierte Matrixoperation:

$$\text{grad_x_conv2} = \text{conv2}(I, G_x, 'same'), \quad \text{grad_y_conv2} = \text{conv2}(I, G_y, 'same'). \quad (5.154)$$

Äquivalenz zur manuellen Faltung Mathematisch ist diese Implementierung äquivalent zur manuellen Faltung:

$$\text{grad_x_manual}(i, j) = \sum_{u=1}^3 \sum_{v=1}^3 I(i-2+u, j-2+v) \cdot G_x(u, v), \quad (5.155)$$

denn beide Verfahren berechnen dieselbe diskrete Faltungsoperation.

Code-Ausschnitt:

```

1 % === 4. Gradient with conv2 ===
2 grad_x_conv2 = conv2(img, Gx, 'same');
3 grad_y_conv2 = conv2(img, Gy, 'same');

```

Erklärung:

- `conv2` führt die Faltung durch
- `'same'` behält Originalbildgröße bei
- Schneller als manuelle Implementierung

5.5.5 Gradientenmagnitude und Kanten

In diesem Abschnitt erläutern wir die Berechnung der Gradientenmagnitude und die Erzeugung der Binar-kante durch Schwellwertsetzung.

Berechnung der Gradientenmagnitude Die Gradientenmagnitude an jedem Pixel (i, j) ist die euklidische Norm des Gradientenvektors:

$$\text{grad_mag}(i, j) = \sqrt{(\text{grad_x}(i, j))^2 + (\text{grad_y}(i, j))^2}. \quad (5.156)$$

Im Code wird dies für die manuelle und die `conv2`-Variante umgesetzt durch

```
1 grad_mag_manual = sqrt(grad_x_manual.^2 + grad_y_manual.^2);
2 grad_mag_conv2  = sqrt(grad_x_conv2.^2 + grad_y_conv2.^2);
```

Schwellwertanwendung zur Kantenextraktion Zur Erzeugung eines Binärbildes, in dem Kanten durch `true` und Nicht-Kanten durch `false` repräsentiert werden, wird ein globaler Schwellwert τ verwendet. Formal:

$$\text{edge}(i, j) = \begin{cases} 1, & \text{wenn } \text{grad_mag}(i, j) > \tau, \\ 0, & \text{sonst.} \end{cases} \quad (5.157)$$

Im Code entspricht dies

```
1 threshold = 0.2;
2 edges_manual = grad_mag_manual > threshold;
3 edges_conv2  = grad_mag_conv2 > threshold;
```

Interpretation Der Schwellwert $\tau = 0,2$ legt fest, ab welcher Gradientenstärke ein Pixel als Kante gilt. Höhere Werte von τ führen zu dünneren, strengeren Kanten, niedrigere zu dichteren, verrauschten Kanten. Durch Vergleich von `edges_manual` und `edges_conv2` lässt sich die äquivalente Funktionalität beider Implementierungen prüfen.

Code-Ausschnitt:

```
1 % === 5. Gradient magnitude and edges ===
2 grad_mag_manual = sqrt(grad_x_manual.^2 + grad_y_manual.^2);
3 grad_mag_conv2  = sqrt(grad_x_conv2.^2 + grad_y_conv2.^2);
4
5 threshold = 0.2;
6 edges_manual = grad_mag_manual > threshold;
7 edges_conv2  = grad_mag_conv2 > threshold;
```

Erklärung:

- Magnitude = $\sqrt{G_x^2 + G_y^2}$
- Schwellwert erzeugt binäres Kantenbild
- Vergleich von manueller und `conv2`-Methode

5.5.6 Visueller Vergleich

Darstellung der Zwischenergebnisse.

Code-Ausschnitt:

```
1 % === 6. Show visual comparison ===
2 figure;
3 subplot(2,3,1); imshow(img); title('Original');
4 subplot(2,3,2); imshow(grad_mag_manual); title('Grad (manual)');
5 subplot(2,3,3); imshow(edges_manual); title('Edges (manual)');
6 subplot(2,3,5); imshow(grad_mag_conv2); title('Grad (conv2)');
7 subplot(2,3,6); imshow(edges_conv2); title('Edges (conv2)');
```

Erklärung:

- 2x3 Gitter mit Originalbild, Gradienten und Kanten
- Vergleich zwischen beiden Methoden

5.5.7 Automatische Durchmessermessung

Morphologische Operationen Ausgehend vom binären Kantenbild $B = \text{edges_conv2}$ werden zunächst alle eingekapselten Löcher gefüllt:

$$B_{\text{filled}} = \text{imfill}(B, \text{'holes'}). \quad (5.158)$$

Anschließend werden alle zusammenhängenden Komponenten $C \subseteq B_{\text{filled}}$ mit Fläche $|C| < A_{\min}$ entfernt:

$$B_{\text{clean}} = \bigcup_{\substack{C \subseteq B_{\text{filled}} \\ |C| \geq A_{\min}}} C, \quad (5.159)$$

wobei $A_{\min} = 500$ Pixel ist.

Regionseigenschaften Für jede zusammenhängende Komponente $C \subseteq B_{\text{clean}}$ werden folgende Eigenschaften berechnet:

$$A = |C|, \quad (\bar{x}, \bar{y}) = \text{Centroid}(C), \quad (5.160)$$

und der Äquivalentdurchmesser D_{eq} , definiert als Durchmesser eines Kreises mit gleicher Fläche:

$$D_{\text{eq}} = \sqrt{\frac{4A}{\pi}}. \quad (5.161)$$

Pixel-zu-Maßstabskonversion Gegeben die Kalibrierung

$$s = 0,25 \text{ mm/px}, \quad (5.162)$$

wird der Durchmesser in Millimetern berechnet durch:

$$D_{\text{mm}} = D_{\text{eq}} \times s. \quad (5.163)$$

Auswahl der größten Region Die Komponente mit der maximalen Fläche $|C^*| = \max_C |C|$ wird ausgewählt:

$$C^* = \arg \max_{C \subseteq B_{\text{clean}}} |C|. \quad (5.164)$$

Sie liefert D_{eq}^* und den Schwerpunkt (\bar{x}^*, \bar{y}^*) .

Visualisierung und Ausgabe Der gefundene Schwerpunkt wird in das Bild gezeichnet und der geschätzte Durchmesser D_{mm}^* als Text ausgegeben:

$$D = D_{\text{mm}}^* \text{ mm}, \quad \text{Scaling} = s \text{ mm/px}. \quad (5.165)$$

Code-Ausschnitt:

```
1 % === 7. Automatic detection and measurement ===
2 % Fill holes and remove small noise
3 BW = imfill(edges_conv2, 'holes');
4 BW = bwareaopen(BW, 500);
5
6 % Calculate equivalent diameter
7 stats = regionprops(BW, 'EquivDiameter', 'Centroid', 'Area');
```

```

8
9 % Real scale: 100 pixels = 25 mm => 1 px = 0.25 mm
10 pixel_scale_mm = 0.25;
11
12 % Select largest region
13 [~, idx] = max([stats.Area]);
14 equiv_diam_px = stats(idx).EquivDiameter;
15 equiv_diam_mm = equiv_diam_px * pixel_scale_mm;
16
17 % Show on image
18 figure; imshow(img); hold on;
19 visboundaries(BW, 'Color', 'b');
20 centro = stats(idx).Centroid;
21 text(centro(1), centro(2), ...
22      sprintf('D = %.2f mm', equiv_diam_mm), ...
23      'Color', 'yellow', 'FontSize', 12, 'FontWeight', 'bold');
24
25 fprintf('\n==== Automatic diameter measurement ==== \n');
26 fprintf('Equivalent diameter in pixels: %.2f px \n', equiv_diam_px);
27 fprintf('Estimated outer diameter: %.2f mm \n', equiv_diam_mm);
28 fprintf('Scaling: 1 pixel = %.3f mm \n', pixel_scale_mm);
    
```

Erklärung:

- Kanten werden gefüllt und kleine Objekte entfernt
- Äquivalentdurchmesser wird berechnet
- Skalierung von Pixeln zu Millimetern
- Visualisierung mit Durchmesserangabe
- Konsolenausgabe der Messergebnisse

5.5.8 Vollständiger MATLAB-Code

Der vollständige Code ist im vorherigen Abschnitt dargestellt und kann direkt in MATLAB ausgeführt werden.

```

1 clc;
2 clear;
3 img_url = 'https://irp.cdn-website.com/1fec50a0/MOBILE/jpg/16395.jpg';
4 img = imread(img_url);
5 img = im2double(rgb2gray(img));
6 [h, w] = size(img);
7 Gx = [-1 0 1; -2 0 2; -1 0 1];
8 Gy = [-1 -2 -1; 0 0 0; 1 2 1];
9 grad_x_manual = zeros(h, w);
10 grad_y_manual = zeros(h, w);
11 for i = 2:h-1
12     for j = 2:w-1
13         region = img(i-1:i+1, j-1:j+1);
14         grad_x_manual(i, j) = sum(sum(region .* Gx));
15         grad_y_manual(i, j) = sum(sum(region .* Gy));
16     end
17 end
18 grad_x_conv2 = conv2(img, Gx, 'same');
19 grad_y_conv2 = conv2(img, Gy, 'same');
20 grad_mag_manual = sqrt(grad_x_manual.^2 + grad_y_manual.^2);
21 grad_mag_conv2 = sqrt(grad_x_conv2.^2 + grad_y_conv2.^2);
22 threshold = 0.2;
23 edges_manual = grad_mag_manual > threshold;
24 edges_conv2 = grad_mag_conv2 > threshold;
    
```

```
25 figure;
26 subplot(2,3,1); imshow(img); title('Original');
27 subplot(2,3,2); imshow(grad_mag_manual); title('Grad (manual)');
28 subplot(2,3,3); imshow(edges_manual); title('Edges (manual)');
29 subplot(2,3,5); imshow(grad_mag_conv2); title('Grad (conv2)');
30 subplot(2,3,6); imshow(edges_conv2); title('Edges (conv2)');
31 BW = imfill(edges_conv2, 'holes');
32 BW = bwareaopen(BW, 500);
33 stats = regionprops(BW, 'EquivDiameter', 'Centroid', 'Area');
34 pixel_scale_mm = 0.25;
35 [~, idx] = max([stats.Area]);
36 equiv_diam_px = stats(idx).EquivDiameter;
37 equiv_diam_mm = equiv_diam_px * pixel_scale_mm;
38 figure; imshow(img); hold on;
39 visboundaries(BW, 'Color', 'b');
40 centro = stats(idx).Centroid;
41 text(centro(1), centro(2), ...
42     sprintf('D = %.2f mm', equiv_diam_mm), ...
43     'Color', 'yellow', 'FontSize', 12, 'FontWeight', 'bold');
44 fprintf('\n=== Automatic Diameter Measurement ===\n');
45 fprintf('Equivalent Diameter in Pixels: %.2f px\n', equiv_diam_px);
46 fprintf('Estimated Outer Diameter: %.2f mm\n', equiv_diam_mm);
47 fprintf('Scaling: 1 Pixel = %.3f mm\n', pixel_scale_mm);
```

Kapitel 6

Fazit

6.1 Schlussfolgerungen

Die Arbeit hat gezeigt, dass Künstliche Intelligenz (KI) die Messtechnik in vielfältiger Weise revolutioniert. Die folgenden Schlussfolgerungen lassen sich aus den Ergebnissen ableiten:

1. **Theoretische Grundlagen:** Die Kombination aus Messtechnik und KI bildet eine solide Basis für moderne Messsysteme. Neuronale Netze, insbesondere Convolutional Neural Networks (CNNs), eignen sich hervorragend für die Verarbeitung komplexer Messdaten.
2. **Anwendungen in der Messtechnik:** KI ermöglicht präzisere und robustere Lösungen für Aufgaben wie Objektlokalisierung und Kantenerkennung. Methoden wie YOLO und CNNs haben sich in der Praxis bewährt.
3. **Praktische Implementierung:** Die in MATLAB umgesetzten neuronalen Netze demonstrieren die Machbarkeit und Effektivität von KI in der Messtechnik. Die Modelle zeigten hohe Genauigkeit in den Bereichen Regression und Klassifikation.
4. **Kantenerkennung:** Klassische Algorithmen wie der Sobel-Operator und der Canny-Detektor wurden durch KI-basierte Ansätze signifikant verbessert. Hybride Methoden kombinieren die Stärken beider Welten.
5. **Leistungsbewertung:** Die Evaluierung der Modelle bestätigte, dass KI-gestützte Messtechnik höhere Präzision, bessere Anpassungsfähigkeit und verbesserte Echtzeitfähigkeit bietet. Dies ist besonders in industriellen Anwendungen von Vorteil.

Zusammenfassend lässt sich festhalten, dass KI die Messtechnik nachhaltig verändert und neue Möglichkeiten für präzise, adaptive und effiziente Messsysteme eröffnet. Die Ergebnisse dieser Arbeit unterstreichen das Potenzial von KI in diesem Bereich und zeigen Wege für zukünftige Forschung auf.

Literaturverzeichnis

- [1] Rainer Parthier. *Messtechnik SI-Einheitensystem – Messergebnisse bewerten – Elektrische Messtechnik anwenden*. Springer Fachmedien Wiesbaden, 2022.
- [2] Herbert Bernstein. *Messelektronik und Sensoren: Grundlagen der Messtechnik, Sensoren, analoge und digitale Signalverarbeitung*. Springer Vieweg, 2013.
- [3] Prof. Dr. Massimo Kubon. Messtechnik skript. Fakultät „Mechanical and Medical Engineering". Skriptum.
- [4] Jörg Hoffmann. *Taschenbuch der Messtechnik*. Carl Hanser Verlag GmbH & Co. KG, 2015.
- [5] H.-R. Tränkler and L. M. (Hrsg.) Reindl. *Sensortechnik – Handbuch für Praxis und Wissenschaft*. Springer Berlin Heidelberg, 2013.
- [6] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition, 2016.
- [7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [8] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.
- [9] K. R. Chowdhary. *Fundamentals of Artificial Intelligence*. Springer, 2019.
- [10] Datasolut. Was ist machine learning? <https://datasolut.com/was-ist-machine-learning/>. Online; abgerufen am [Datum].
- [11] W. Ertel. *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung*. Springer Vieweg, 2024.
- [12] Christopher M. Bishop and Hugh Bishop. *Deep Learning: Foundations and Concepts*. Springer, 2023.
- [13] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Pearson Education, 2nd edition, 2005.
- [14] S. Haykin. *Neural Networks and Learning Machines*. Pearson, 3rd edition, 2009.
- [15] Wasif Shafaet Chowdhury and Yong Yan. Applications of artificial intelligence to instrumentation systems for monitoring complex industrial processes. *Cybernetics and Intelligence Journal*, 2023. In press as of December 2023.
- [16] Rolf Isermann. *Mechatronic Systems Fundamentals*. Springer London, 2005.
- [17] Bernd Jähne. *Digitale Bildverarbeitung: und Bildgewinnung*. Springer Vieweg, Berlin, Heidelberg, 8 edition, 2024.
- [18] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson Prentice Hall, Upper Saddle River, NJ, 3 edition, 2008.

- [19] S. Gopal Krishna Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *CoRR*, abs/1503.06462, 2015.
- [20] Wilhelm Burger and Mark James Burge. *Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java*. X.media.press. Springer Vieweg, Berlin, Heidelberg, 3 edition, 2015.
- [21] David J. Fleet, Tomáš Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors. *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part II*, volume 8690 of *Lecture Notes in Computer Science*, Cham, 2014. Springer.
- [22] AIME Planning Team. Artificial intelligence measurement and evaluation at the national institute of standards and technology. Technical report, NIST, 2021. Draft, 2021-06-14.
- [23] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. Unpublished Manuscript. IIIT Allahabad, Techno India University, Indian Statistical Institute, 2023.
- [24] S. Sharma, S. Sharma, and A. Athaiya. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 4(12):310–316, 2020.
- [25] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Springer*, 2021.
- [26] Y. Wang, A. Yang, X. Chen, P. Wang, Y. Wang, and H. Yang. A deep learning approach for blind drift calibration of sensor networks. *IEEE Transactions (preprint)*, 2020.
- [27] M. Chui, E. Hazan, R. Roberts, A. Singla, K. Smaje, A. Sukharevsky, L. Yee, and R. Zemmel. The economic potential of generative ai: The next productivity frontier. Technical report, McKinsey & Company, 2023.
- [28] T. Luo and S. G. Nagarajan. Distributed anomaly detection using autoencoder neural networks in wsn for iot. In *IEEE ICC*, 2018.
- [29] Q. Wang, Y. Ma, K. Zhao, and Y. Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9:187–212, 2022.
- [30] N. Saini. Research paper on artificial intelligence & its applications. *International Journal for Research Trends and Innovation*, 8(4):356–360, 2023.
- [31] T. Szandała. Review and comparison of commonly used activation functions for deep neural networks. *Proceedings of COMPSTAT'2010*, 2020.
- [32] F. Ashiq, M. Asif, M. B. Ahmad, S. Zafar, K. Masood, T. Mahmood, M. T. Mahmood, and I. H. Lee. Cnn-based object recognition and tracking system to assist visually impaired people. *IEEE Access*, 10, 2022.
- [33] S. Haripriya and L. C. Manikandan. A study on artificial intelligence technologies and its applications. *Musaliar College of Engineering & Technology, Kerala*, 2022.
- [34] S. Yu, J. Ma, and W. Wang. Deep learning for denoising. *arXiv preprint arXiv:1810.11614*, 2019.
- [35] D. Bhatt, C. Patel, H. Talsania, J. Patel, R. Vaghela, S. Pandya, K. Modi, and H. Ghayvat. Cnn variants for computer vision: History, architecture, application, challenges and future scope. *Electronics*, 10(24):2470, 2021.
- [36] W. Wu and Q. Li. Machine vision inspection of electrical connectors based on improved yolo v3. *IEEE Access*, 8, 2020.

- [37] S. S. Sumit, D. R. Awang Rambli, S. Mirjalili, M. M. Ejaz, and M. S. U. Miah. Restinet: On improving the performance of tiny-yolo-based cnn architecture for applications in human detection. *Applied Sciences*, 12(18):9331, 2022.
- [38] Z. Geng, K. Sun, B. Xiao, Z. Zhang, and J. Wang. Bottom-up human pose estimation via disentangled keypoint regression. Unpublished Manuscript, 2022.
- [39] K. Janocha and W. M. Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.
- [40] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 2010.
- [41] H. Su, V. Jampani, D. Sun, O. Gallo, E. Learned-Miller, and J. Kautz. Pixel-adaptive convolutional neural networks. *arXiv preprint arXiv:1904.05373*, 2019.
- [42] J. Liu, M. Malekzadeh, N. Mirian, T. Song, C. Liu, and J. Dutta. Artificial intelligence-based image enhancement in pet imaging: Noise reduction and resolution enhancement. *PET Clinics*, 16(4):553–576, 2021.
- [43] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [44] Z. Duan, N. Wang, J. Fu, W. Zhao, B. Duan, and J. Zhao. High precision edge detection algorithm for mechanical parts. *Measurement Science Review*, 18(2):65–71, 2018.
- [45] K. Masaoka. Accuracy and precision of edge-based modulation transfer function measurement for sampled imaging systems. In *IEEE International Conference on Imaging Systems and Techniques*, 2018.
- [46] R. Sun, T. Lei, Q. Chen, Z. Wang, X. Du, W. Zhao, and A. K. Nandi. Survey of image edge detection. *Frontiers in Signal Processing*, 2024.
- [47] A. H. Abdel-Gawad, L. A. Said, and A. G. Radwan. Optimised edge detection technique for brain tumor detection in mr images. *IEEE Access*, 8:136243–136259, 2020.
- [48] X. Li, H. Jiao, and Y. Wang. Edge detection algorithm of cancer image based on deep learning. *Bioengineered*, 11(1):591–600, 2020.
- [49] M. Pu, Y. Huang, Y. Liu, Q. Guan, and H. Ling. Edter: Edge detection with transformer. *IEEE Transactions on Image Processing*, 2023. Beijing Key Laboratory of Traffic Data Analysis and Mining, Beijing Jiaotong University.
- [50] Anonymous. Advancements in edge detection techniques for image enhancement: A comprehensive review. *International Journal of Artificial Intelligence & Robotics (IJAIR)*, 6(1):29–39, 2024.
- [51] C. Liu and J. Sun. Ai in measurement science. *Annual Review of Analytical Chemistry*, 2023.
- [52] S. Xie and Z. Tu. Holistically-nested edge detection. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [53] A. BenHajyoussef and A. Saidani. Recent advances on image edge detection. In *Digital Image Processing – Latest Advances and Applications*. Springer, 2019.
- [54] R. Shahbazian, G. Macrina, E. Scalzo, and F. Guerriero. Machine learning assists iot localization: A review of current challenges and future trends. *Sensors*, 23(7):3551, 2023.

- [55] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, Stockholm, Sweden, 2018.
- [56] F. M. Shiri, T. Perumal, N. Mustapha, and R. Mohamed. A comprehensive overview and comparative analysis on deep learning models. *Journal of Artificial Intelligence*, page 054314, 2024.
- [57] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Is object localization for free? weakly-supervised learning with convolutional neural networks. Unpublished Manuscript. INRIA and Facebook AI Research, 2015.
- [58] Y. Xu, Y. S. Shmaliy, C. K. Ahn, T. Shen, and Y. Zhuang. Tightly-coupled integration of ins and uwb using fixed-lag extended uir smoothing for quadrotor localization. *IEEE Internet of Things Journal*, 1, 2021.
- [59] S. Mokhtari, A. Abbaspour, K. K. Yen, and A. Sargolzaei. A machine learning approach for anomaly detection in industrial control systems based on measurement data. *Electronics*, 10(4):407, 2021.
- [60] C. Pavlatos, E. Makris, G. Fotis, V. Vita, and V. Mladenov. Utilization of artificial neural networks for precise electrical load prediction. *Energies*, 15(6):2022, 2022.
- [61] A. Ettalibi, A. Elouadi, and A. Mansour. Ai and computer vision-based real-time quality control: A review of industrial applications. *Procedia Computer Science*, 231:212–220, 2024.
- [62] M. Moreira and E. Fiesler. Neural networks with adaptive learning rate and momentum terms. *Institut Dalle Molle d'Intelligence Artificielle Perceptive*, 1995.
- [63] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2020.
- [64] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.
- [65] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. In *Accepted as a workshop contribution at ICLR 2015*, 2015.
- [66] N. S. Keskar and R. Socher. Improving generalization performance by switching from adam to sg. *arXiv preprint arXiv:1712.07628*, 2017.
- [67] J. Byrd and Z. C. Lipton. What is the effect of importance weighting in deep learning? *arXiv preprint arXiv:1812.06205*, 2019.
- [68] J. Jepkoech, D. M. Mugo, B. K. Kenduiywo, and E. C. Too. The effect of adaptive learning rate on the accuracy of neural networks. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 12(8):736–742, 2021.
- [69] S. Cai, Y. Shu, W. Wang, G. Chen, B. C. Ooi, and M. Zhang. Efficient and effective dropout for deep convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
- [70] M. Yang, M. K. Lim, Y. Qu, X. Li, and D. Ni. Deep neural networks with l1 and l2 regularization for high dimensional corporate credit risk prediction. *Expert Systems with Applications*, 213:118873, 2023.
- [71] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.