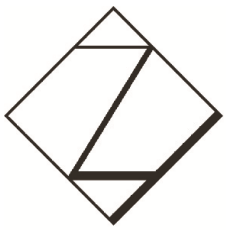


Westsächsische Hochschule Zwickau
University of Applied Sciences
Fakultät Elektrotechnik

Master Thesis

Automatisierte Fehlerdetektierung in der Halbleiter-Waferproduktion mittels maschinellen Lernens

zur Erlangung des

akademischen Grades

Master of Engineering

eingereicht von

Michaela Banert

am 27. März 2023

Hochschullehrer:	Prof. Dr. rer. nat. Mike Espig
Zweitgutachter:	Prof. Dr. rer. nat. Daniel Schondelmaier

Eidesstattliche Erklärung

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Chemnitz, am 27.03.2023 .....

(Unterschrift)

Kurzfassung

Die vorliegende wissenschaftliche Arbeit beschäftigt sich mit der Anwendung von maschinellem Lernen auf einen Waferdatensatz. Nach einer Einführung in Wafer und deren Defektbildung sowie einer Erklärung der Grundlagen des maschinellen Lernens und insbesondere des überwachten Lernens mit Hilfe von einem *Convolutional Neural Network* (*CNN*), wird der Datensatz analysiert, und es werden die verschiedenen Waferdefekte beschrieben. Der Datensatz wird für das maschinelle Lernen verwendet, und sowohl das *CNN* als auch das *Wavelet Scattering Transformation* (*WST*)-Modell erreichen eine hohe Genauigkeit von 96% bzw. 97%. Besonders hervorgehoben wird die höhere durchschnittliche Genauigkeit des *WST*-Modell im Vergleich zu den Ergebnissen des Papers, aus dem der Datensatz stammt.

Abstract

This work deals with the application of machine learning to a wafer dataset. After an introduction to wafers and their defect formation, and an explanation of the basics of machine learning and in particular supervised learning using a *CNN*, the dataset is analyzed and the different wafer defects are described. The dataset is used for machine learning and both the *CNN* and the *WST*-model achieve high accuracy of 96% and 97%, respectively. The higher average accuracy of the *WST*-model compared to the results of the paper from which the dataset was obtained is particularly emphasized.

Danksagung

Zuerst möchte ich mich bei den meinen Betreuern Herrn Prof. Dr. Mike Espig und Prof. Dr. Daniel Schondelmaier für die Betreuung und Unterstützung bei dieser Arbeit bedanken. Besonders danke ich ebenfalls Christian Walter, der mich bei dem Inhalt und der Entwicklung der Modelle unterstützt hat und mir bei jeder Problematik helfen konnte.

Ein weiterer Dank gilt meinen Arbeitskollegen der Data Science Research Group, die mir mit ihren Programmierfähigkeiten und fachlichem Wissen beistanden.

Gedankt sei auch meiner Familie. Besonders meinem Vater, der mich zu seinen Lebzeiten während meines Studium sehr unterstützt hat und meiner Mutter, die mich während des Schreibens dieser Arbeit motiviert hat.

Ich danke außerdem den Korrekturleserinnen Antje Holtz, Antje Schuschies und Helga Banert und dem Korrekturleser Christian Walter, die mir geholfen haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kurzzeichenverzeichnis

Symbol	Beschreibung	Einheit
$\mathcal{A}$	Lernalgorithmus	[ ]
argmin	Argument des Minimums	[ ]
$\mathbf{b}$	Bias-Parameter	[ ]
C	Kohlenstoff	[ ]
CO	Kohlenstoffmonoxid	[ ]
E	Näherungsfehler	[ ]
$\exp$	Exponentialfunktion	[ ]
f	Funktion	[ ]
F	Filtermatrix	[ ]
g	Hypothese mit kleinsten R_{emp} aus $\mathcal{H}$	[ ]
h	Hypothese aus $\mathcal{H}$	[ ]
$\mathcal{H}$	Hypothesenraum	[ ]
HCl	Chlorwasserstoff	[ ]
H_2	Wasserstoff	[ ]
L	Lossfunktion, Index	[ ]
l	Schicht	[ ]
N	Instanzen	[ ]
p	Größe eines Patches	[ ]
$p(x, r)$	Verteilung	[ ]
Q	Anzahl der Wavelets pro Oktave	[ ]
r^t	gewünschte Ausgabe	[ ]
R_{emp}	empirisches Risiko	[ ]
S	Samplemenge	[ ]
Si	Silicium	[ ]

Symbol	Beschreibung	Einheit
x	Eingabevariable	[ ]
$\mathcal{X}$	Merkmalsraum	[ ]
y	Zielvariable	[ ]
$\mathcal{Y}$	Ausgaberaum	[ ]
O	Sauerstoff	[ ]
$SiHCl_3$	Trichlorsilan	[ ]
σ	Aktivierungsfunktion	[ ]
$\tanh$	Tangens hyperbolicus	[ ]
θ	Parameter einer Zielverteilung	[ ]
ϕ	Tiefpassfilter	[ ]
ψ	Grundfunktion, Waveletfunktion	[ ]
$\forall$	für alle	[ ]

Abkürzungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Ziel

Die Produktion von Halbleiter-Wafern erfordert eine hohe Präzision und Qualitätssicherung, um diee optimale Funktionsfähigkeit der Bauteile zu gewährleisten. Die manuelle Fehlerdetektierung ist ein aufwändiger und fehleranfälliger Prozess, der die Produktionskosten erhöht und die Effizienz beeinträchtigt. In den letzten Jahren hat die Anwendung von maschinellem Lernen in der Halbleiterindustrie stark zugenommen und bietet eine vielversprechende Alternative zur manuellen Fehlererkennung. Durch die automatisierte Analyse von großen Datenmengen können Fehler in Echtzeit erkannt und präzise lokalisiert werden. In dieser Arbeit liegt der Fokus auf der Anwendung von maschinellen Lernverfahren wie *CNN* und *WST* zur automatisierten Fehlerdetektierung in der Halbleiter-Waferproduktion. Ziel ist die Entwicklung zweier Modelle, welche eine hohe Genauigkeit bei der Erkennung der verschiedenen Fehlertypen zu erreichen.

1.2 Aufgabenstellung

In der Halbleiterindustrie kommt es bei der Herstellung von Wafern zu Defekten. Heutzutage existieren spezielle Maschinen zur Messung von Wafern. Damit können Wafer zum Teil in unter 30 Sekunden vermessen werden. Dabei können Auffälligkeiten in der Rauheit und der Welligkeit von Wafern dargestellt werden. [1] Unbekannte Defekte können mit der „Golden Sample Methode“ gefunden und anhand von neuronalen Netzen klassifiziert werden [2]. Das „Golden Sample“ ist ein Muster, welches mit demselben Verfahren und Eigenschaften hergestellt wurde, wie die in der Massenproduktion hergestellte Produkte. Dieses Produkt dient dann als Vergleichsobjekt [3].

Die Aufgabe dieser Arbeit ist die Entwicklung eines Modells zur Klassifizierung von Waferdefekten mittels maschinellen Lernens.

Im Rahmen dieser Arbeit sind folgende Punkte zu erarbeiten:

- Literaturrecherche zu Wafern und deren Herstellung, sowie die Entstehung von Waferdefekten
- Recherche zur explorativen Modellbildung
- Durchführung der explorativen Modellbildung mit der Erläuterung des später verwendeten Datensatzes und der Hyperparameteroptimierung der Lernmaschine
- Erprobung des Prototyps anhand des Datensatzes
- Auswertung, Vergleich der Ergebnisse mit der Literatur und Ausblick

Kapitel 2

Grundlagen zur Waferherstellung

Im folgenden Kapitel werden Wafer definiert. Es wird die Herstellung von Wafern erläutert und beschrieben, wie Waferdefekte entstehen. Zum Schluss wird auf die Verwendung von Wafern eingegangen.

2.1 Wafer

In der Halbleiterindustrie werden hochprozessierte Scheiben als Wafer bezeichnet, die als Untergrund integrierter Schaltkreise, sogenannter Mikrochips, dienen. Die meisten Wafer bestehen aus Silicium, wobei auch beispielsweise Germanium, Siliciumcarbid und Galliumarsenid als Wafermaterial genutzt werden können. Alle diese Materialien besitzen Halbleitereigenschaften. Die Waferoberfläche wird als Substrat bezeichnet und darf für die weitere Verwendung Unebenheiten im Bereich weniger Nanometer aufweisen. Während der Produktion müssen die Wafer exakt ausgerichtet werden. Deshalb werden sie am Waferrand markiert. Bei einer Scheibengröße von maximal 150mm werden in der Regel Flats verwendet. Als Flat wird die gerade Kante an einer Waferseite bezeichnet, die zur Identifizierung und zur Bestimmung der Kristallorientierung des Wafers dient. Flats werden bei Wafergrößen von bis zu 6“ genutzt. Da jedoch durch Flats die nutzbare Fläche abnimmt, werden heutzutage Notches verwendet. Notches sind kleine Einkerbungen, welche anstelle des abgeflachten Waferrandes verwendet werden. Notches werden bei Wafergrößen ab 8“ zur Kennzeichnung der Wafer verwendet. [4, S. 48 f]. Durch eine Lasergravur wird jede Scheibe am Rand mit einem Barcode, einer Punktmatrix oder ähnlichem gekennzeichnet. [5]

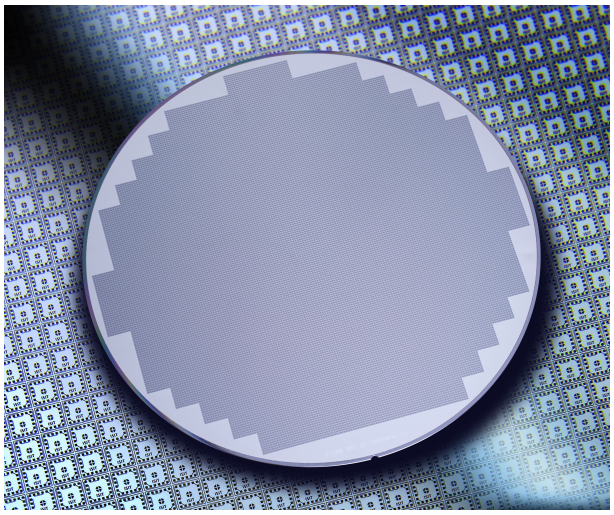


Abbildung 2.1: Beispielbild für einen Wafer [6]

Unter der Voraussetzung, dass ein Wafer in einkristalliner Form vorhanden ist, gibt es zwei Verfahren für dessen Herstellung, das Czochralski-Verfahren und das tiegelfreie Zonenziehen. Aktuell werden für die Fertigung von Chips oft Wafer verwendet, die einen Durchmesser von 150 bis 300mm aufweisen. [5]

Dass Wafer eine kreisrunde Form haben, ist durch das Herstellungsverfahren vorgegeben. In Kapitel 2.2.3 wird näher darauf eingegangen. Die runde Form von Wafern hat zwei wichtige Vorteile gegenüber rechteckigen Wafern. Runde Wafer haben keine Ecken, welche während des Prozesses oder beim Transportieren schnell Schaden erleiden können. Im Vergleich zu runden Wafern können rechteckige in einigen Verfahren, wie bei Spinprozessen, chemisch mechanischem Polieren oder Plasmaprozesse, nicht homogen bearbeitet werden. [5]

2.2 Waferherstellung

Die Herstellung von Wafern wird in drei Schritte eingeteilt, die im folgenden Unterkapitel beschrieben werden.

2.2.1 Herstellung von Rohsilicium

Das Element Silicium (Si) ist das Ausgangsmaterial und wird für die Fertigung von Wafern verwendet. Unter anderem kommt es in Quarzsand vor und kann daraus extrahiert werden. Dazu wird dem Quarz Sauerstoff entnommen, welcher schon bei Raumtemperatur mit Silicium in Verbindung geht. Dieser Vorgang findet bei einer Temperatur zwischen 1500 °C und 1650 °C statt (Schmelzpunkt von Silicium: 1414 °C). Durchgeführt wird der Prozess in Öfen. Dabei wird Kohlenstoff zum Abspalten des Sauerstoffs

genutzt. Der Sauerstoff des Siliciums reagiert mit Kohlenstoff (C) zu Kohlenmonoxid (CO) [7, S. 164]:

$$\mathrm{SiO_2} + 2\,\mathrm{C} \longrightarrow \mathrm{Si} + 2\,\mathrm{CO} \tag{2.1}$$

Damit das Silicium nicht mit dem Kohlenstoff zu Siliciumcarbid reagiert, wird Eisen während der Reaktion hinzugefügt. Kohlenmonoxid befindet sich bei den Temperaturen in den Öfen im gasförmigen Zustand und ist somit von dem Silicium trennbar. Das Rohsilicium, das bei diesem Prozess entsteht, besteht bis zu 5% aus Fremdstoffen, wie Eisen, Aluminium, Phosphor und Bor. Es wird von einer Verunreinigung des Rohsiliciums gesprochen. In späteren Prozessen werden die Fremdstoffe extrahiert. [8]

Zur Reinigung des Rohsiliciums wird unter anderem der Trichlorsilanprozess angewendet. Dabei werden Fremdstoffe durch Destillation herausgefiltert. Bei der Reaktion von Rohsilicium mit Chlorwasserstoff (HCl) bei einer Temperatur von 300 °C entstehen Trichlorsilan ($SiHCl_3$) und Wasserstoff (H_2) [7, S. 165]:

$$\mathrm{Si} + 3\,\mathrm{HCl} \longrightarrow \mathrm{SiHCl_3} + \mathrm{H_2} \tag{2.2}$$

Die fremden Partikel gehen mit dem Chlor eine Verbindung ein, jedoch verändert sich deren Aggregatzustand bei höheren Temperaturen zu gasförmig. Dadurch wird das Trichlorsilan abgespalten. Die Stoffe Kohlenstoff, Phosphor und Bor können durch dieses Verfahren nicht isoliert werden.[8]

Um das Silicium in polykristallinem Zustand herzustellen, wird der Trichlorsilanprozess umgekehrt. Dafür wird Wasserstoff in ein Quarzgefäß gegeben. In diesem Gefäß befinden sich dünne Siliciumstäbe (Siliciumseelen), und es herrscht eine Temperatur von 1100 °C. Es kommt zu folgenden Reaktionen. [7, S. 165]:

$$\mathrm{SiHCl_3} + \mathrm{H_2} \longrightarrow \mathrm{Si} + 3\,\mathrm{HCl} \tag{2.3}$$

$$4\,\mathrm{SiHCl_3} \longrightarrow \mathrm{Si} + 3\,\mathrm{SiCl_4} + 2\,\mathrm{H_2} \tag{2.4}$$

Es bildet sich ein Siliciumniederschlag, welcher sich auf den Siliciumstäben absetzt. Die Stäbe erreichen durch diesen Niederschlag eine Dicke von bis zu 200mm. [7, S. 165]

Zur Erhöhung des Reinheitsgrades von Silicium wird ein weiteres Verfahren angewandt, die Zonenreinigung. Dabei wird ein Draht um die Siliciumstäbe gewickelt, durch diese Spule fließt hochfrequenter Wechselstrom. Durch diese Konstruktion schmilzt das Silicium im Inneren der Stäbe, und es kommt zur Verringerung der Fremdkörperanzahl. Aufgrund der Oberflächenspannung des Siliciums tritt die Schmelze nicht aus

den Stäben heraus. Wird dieser Vorgang mehrfach wiederholt, erhöht sich der Reinheitsgrad stetig. Schließlich kann das Silicium zum Einkristall weiter verarbeitet werden. Damit Verunreinigungen von außen auf ein Minimum reduziert werden, findet der Prozess im Vakuum statt. [8]

Nach Abschluss dieser Prozesse hat das Silicium einen Reinheitsgrad von über 99%. [8]

2.2.2 Herstellung von Einkristallen

Für die Herstellung von Halbleitern werden Einkristalle benötigt [9]. Ein Einkristall besitzt Atome, welche in allen Richtungen den gleichen Abstand haben. Sie werden über ein dreidimensionales Gitter definiert, in dem Atome, Moleküle oder Ionen entlang eine Koordinatenachse angeordnet sind. Der Abstand zweier Atome wird Gitterabstand genannt [10, 21 f.]. In Abbildung 2.2 sind sechs Kristallorientierungen beispielhaft dargestellt [9].

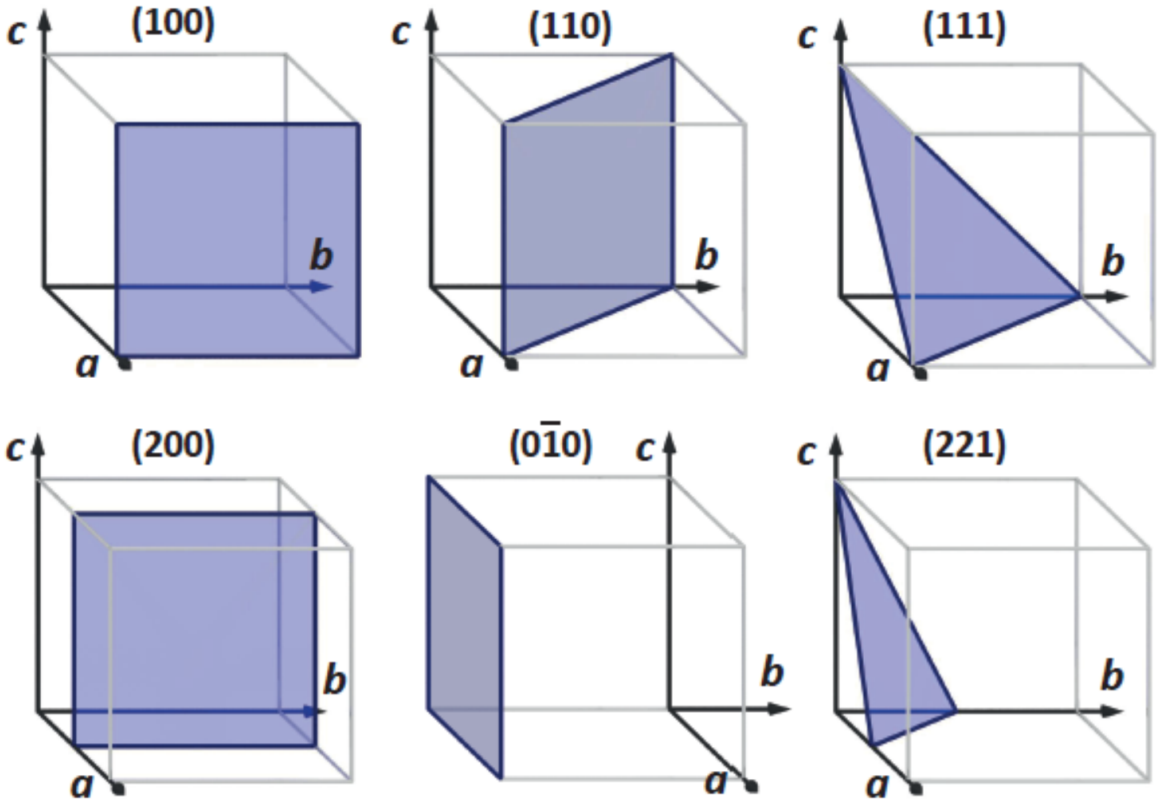


Abbildung 2.2: Verschiedene Kristallorientierungen [11, S. 3]

Die Kristallorientierung spielt eine große Rolle für die mikromechanischen Eigenschaften. Mikrokanäle mit senkrechten Wänden lassen sich auf (110)-Silicium herstellen, während sich auf (100)-Silicium Flankenwinkel von 54,74° ergeben. [9]

Zur Herstellung von Einkristallen gibt es zwei Verfahren, welche im Folgenden vorgestellt werden:

Bei dem Kristallziehverfahren nach Czochalski wird zur Keimbildung ein Einkristall an einem drehbaren Stab an die Oberfläche der Siliciumschmelze gebracht. Kommt der Keim in Berührung mit der Schmelze, findet eine Anlagerung des Siliciums an den Keim statt. Das Silicium erhält die Kristallstruktur des Keims. Aufgrund der Tatsache, dass die Tiegeltemperatur geringfügig höher ist als der Schmelzpunkt des Siliciums, kommt es zum Erstarren des Siliciums am Keimling. Es erfolgt ein Wachstum des Kristalls. [9] Der Keim wird rotierend in die Höhe gezogen. Dabei berührt er die Schmelze permanent. Entgegen der Drehrichtung vom Keim dreht sich zeitgleich der Tiegel. Damit der Keim homogen wächst, muss die Temperatur der Schmelze konstant gehalten werden. Die Ziehgeschwindigkeit des Keims variiert zwischen 3 und 20 cm/h. [4, S. 45] Sie hat Einfluss auf den Durchmesser des Kristalls. Es gilt, je höher die Geschwindigkeit, desto schmaler ist der Durchmesser des Kristalls. Zur Vermeidung einer Oxidation von Silicium findet der Prozess in einer Schutzgasatmosphäre statt. Nachteil dieses Verfahrens ist die Fremdstoffablagerung, die durch den Tiegel verursacht werden kann. Dadurch kommt es zu Änderungen im Kristallgitter.[7, S. 167] Vorteile des Verfahrens sind jedoch die geringen Kosten. [9]

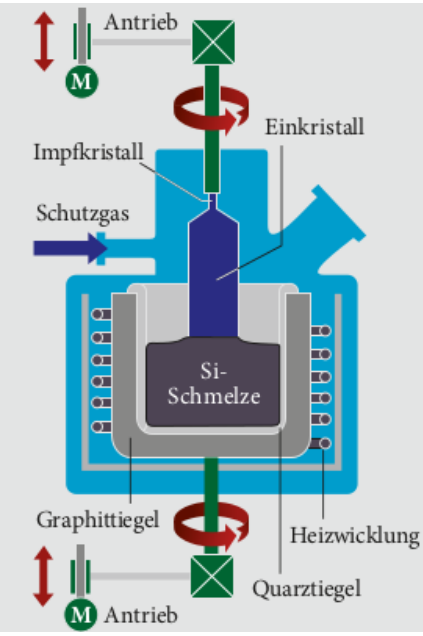


Abbildung 2.3: Czochralski-Verfahren [4, S. 45]

Das tiegelfreie Zonenziehen ist ein Verfahren, bei dem nur ein kleiner Teil des polykristallinen Siliciums (wenige Millimeter) geschmolzen wird. Genauso wie bei dem bereits geschilderten Kristallziehverfahren nach Czochralski wird ein Einkristall als Kristallstrukturorientierung genutzt. Der polykristalline Keim wird geschmolzen und erhält die Struktur des Einkristalls. Der ganze Prozess weist einen langsamen, schleichenden Verlauf auf. Aufgrund dessen, dass nur ein geringer Teil des Siliciums geschmolzen wird, kommt es zu geringeren Verunreinigungen als beim Kristallziehen. [9]

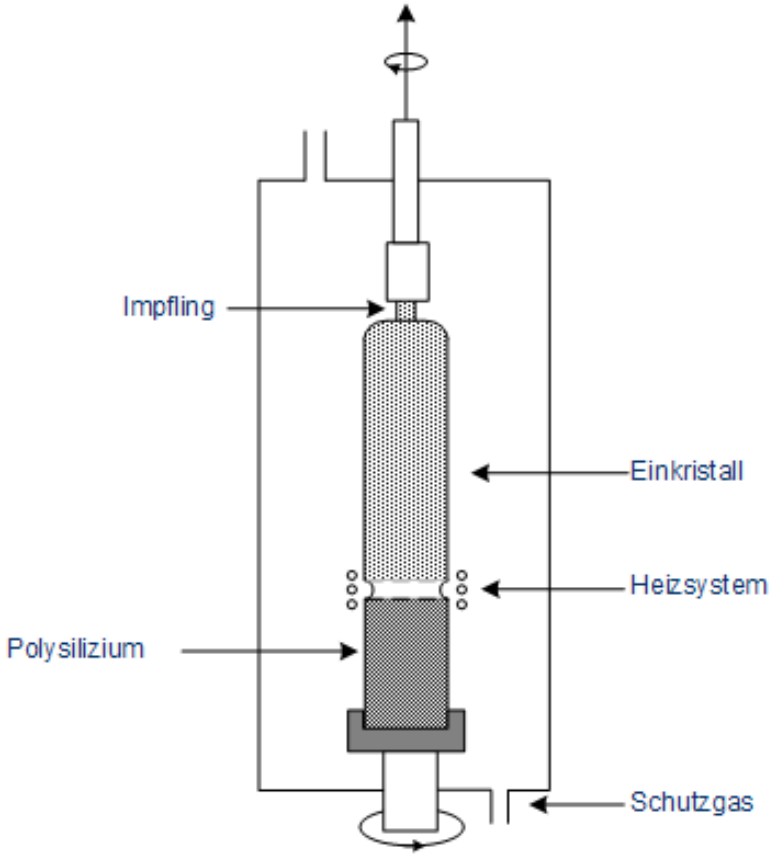


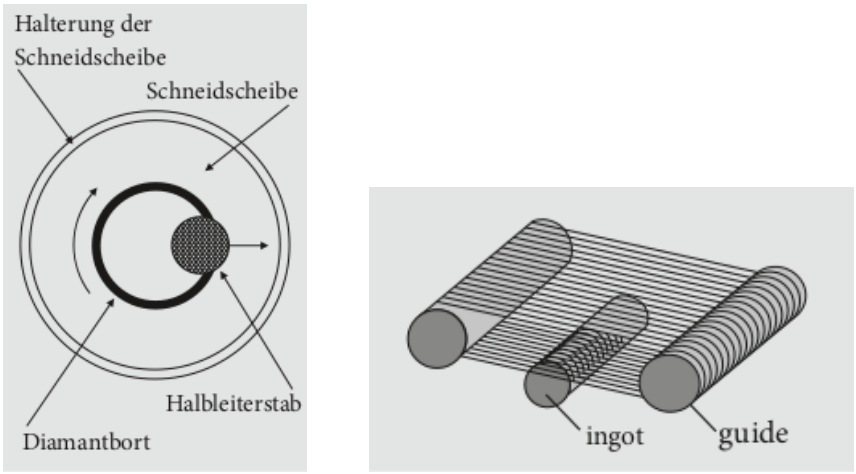
Abbildung 2.4: Tiegelfreies Zonenziehen [9]

2.2.3 Waferherstellung und Oberflächenveredlung

Der strukturierte Einkristall wird abgedreht und erhält, abhängig von der Orientierung des Kristalls, einen Flat. [4, S. 48 f]

Bei dem Sägeprozess wird der Einkristallstab mithilfe einer Innenlochsäge (siehe Abbildung 2.5 (a)) in schmale Scheiben gesägt. Es können zwei verschiedene Sägen verwendet werden. Vorteil der Innenlochsäge ist eine hohe Genauigkeit beim Sägen, ohne Ecken und Kanten zu verursachen. [12]. Bei der Verwendung von Drahtsägen (siehe Abbildung 2.5 (b)) können mehrere Wafer gleichzeitig aus einem Stab herge-

stellt werden. [4, S. 49] Dabei wird ein langer Draht über sich drehende Walzen geführt. Der Einkristallstab wird in das Gitter aus Draht abgelassen und dadurch in einzelne Waferscheiben zerteilt. Bei diesem Prozess bewegt sich der Draht in entgegengesetzte Richtung mit einer Geschwindigkeit von etwa 10m/s und hat im Normalfall eine Dicke von 0,1 bis 0,2mm. [12]



(a) (b)

Abbildung 2.5: Innenlochsäge und Drahtsäge [4, S. 49]

Während des Sägeprozesses entstehen aufgrund der mechanischen Beanspruchung des Materials Oberflächenrauheiten. Sie verursachen in der Folge Schäden im Kristallgitter. Die Veredlung der Oberfläche wird im Folgenden beschrieben.

Bei der Weiterverarbeitung der Wafer dürfen ihre Ränder nicht scharfkantig sein [4, S. 49]. Andernfalls besteht die Gefahr, dass aufgetragene Schichten abplatzen könnten [12]. Diesen Prozessabschnitt, bei dem die Ränder abgeschliffen werden, wird Grinding genannt.

Das Läppen ist ein Prozess, bei dem die Oberfläche der Scheibe um 50µm abgeschliffen wird. Das körnige Schleifmittel, ein Gemisch aus Glyzerin und Aluminiumoxid, wird dabei auf eine sich drehende Scheibe aufgebracht. [4, S. 49] Die Konzentration der Körner des Schleifmittels wird während des Prozesses stufenweise reduziert. Aufgrund der auch hier auftretenden mechanischen Beanspruchung, kommt es erneut zu Oberflächenrauheiten [12]. Die Ebenheit der Oberfläche nach dem Läppen beträgt 2µm. [4, S. 50]

Während des Ätzens werden nochmals 50µm der Waferschicht abgetragen. Zum Ätzen wird ein Flüssigkeitsgemisch aus Fluss-, Essig- und Salpetersäure verwendet [4, S. 50]. Da dies ein chemischer Prozess ist, kommt es zu keiner mechanischen Beanspruchung der Waferoberfläche [12].

Im Anschluss findet das Polieren statt. Nach Vollendung dieses Prozessschrittes weist die Oberfläche des Wafers eine Rauheit von unter 3nm auf. Zur Erreichung dieser Rauheit wird ein Stoffgemisch aus Natronlauge und Siliciumoxidkörnern verwendet. Es kann ein Materialabtrag von bis zu circa 5µm erreicht werden. [4, S. 50]

Am Ende des Prozesses werden die Wafer mit einem Laser gekennzeichnet. [4, S. 51]

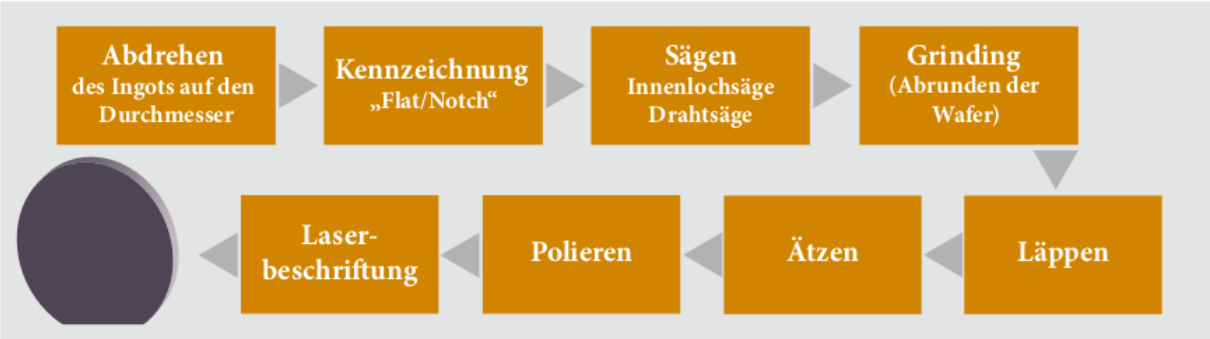


Abbildung 2.6: Übersicht der Prozessschritte bei der Waferherstellung [4, S. 48]

2.3 Defektentstehung bei der Herstellung von Wafern

Waferdefekte können verschiedene Ursachen haben. Kristallfehler, mechanische Fehler, der Mensch als Ursache und chemische Verunreinigungen werden im Folgenden erläutert.

Kristallfehler werden in Punkt-, Linien- und Flächendefekte unterteilt. Bei Punktdefekten kann es in den Kristallgittern zu Leerstellen oder Zwischengitterdefekten kommen. In den Leerstellen können sich Fremdatome einlagern. Diese Defekte sind in Abbildung 2.7 zu sehen. [4, S. 56]

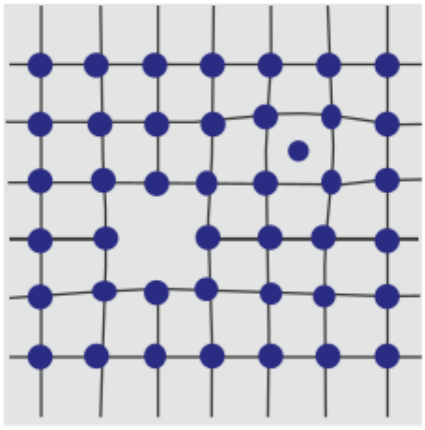
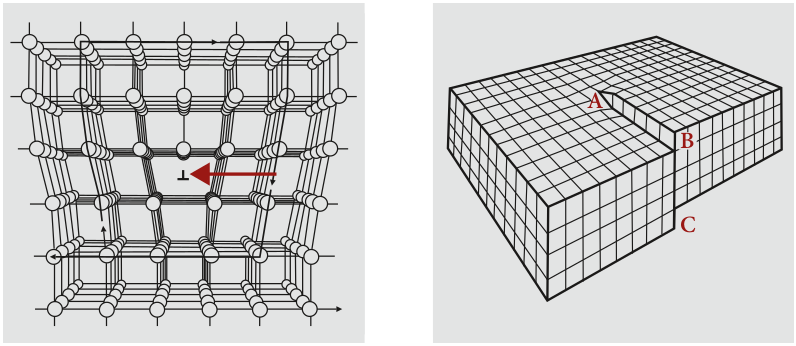


Abbildung 2.7: Es sind zwei Punktdefekte zu sehen. Links im Bild entsteht eine Leerstelle und rechts im Bild ist ein Atom zu viel im Gitter [4, S. 56]

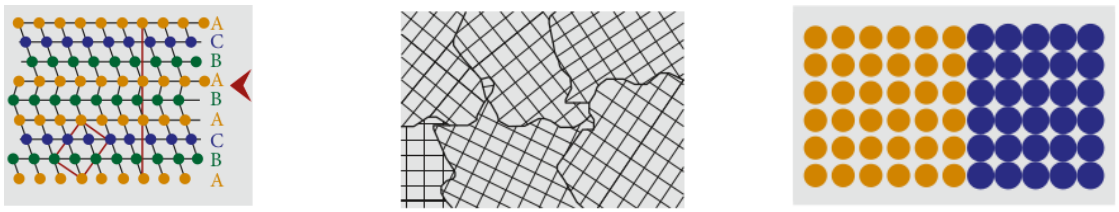
Als Liniendefekte werden eindimensionale Gitterfehler bezeichnet, welche eine Versetzung des Gittern hervorrufen können. Dabei wird zwischen Stufen-, Schrauben- und gemischten Versetzungen unterschieden. Die Stufenversetzung befindet sich am Ende einer Gitterebene innerhalb des Kristalls. Es kommt zu Verschiebung der benachbarten Atome. Es entsteht eine Stufe, wodurch Spannungen entstehen. Diese Spannungen können zu Fehlern bei der Fertigung führen. Bei der Schraubenversetzung wird eine horizontale Ebene verschoben, wodurch der Kristall schraubenförmig deformiert wird. Ein Beispiel für diese Defekte ist in der Abbildung 2.8 zu sehen. [4, S. 56 f]



(a) (b)

Abbildung 2.8: Stufenversetzung und Schraubenversetzung [4, S. 57]

Als Flächendefekte werden zweidimensionale Fehler im Gitter definiert. Es wird zwischen Stapelfehler, Korn- und Phasengrenzen unterschieden. Beim Stapelfehler kommt es zu Defekten in der Reihenfolge der Atomlagen. In Abbildung 2.9 a) fehlt beispielsweise die Ebene C. Korngrenzen entstehen bei dem Aufeinandertreffen zweier kristalliner Bereiche, deren Kristallorientierungen verschieden sind. Phasengrenzen treten bei dem Aufeinanderschichten unterschiedlicher Materialien auf. [4, S. 5 f]



a) Stapelfehler b) Korngrenzen c) Phasengrenzen

Abbildung 2.9: Flächendefekte [4, S. 7 f]

Abdrücke können bei mechanischem Kontakt mit Greifern und Fixiervorrichtungen entstehen. Dabei gelangen Partikel von den Halterungen an die Wafer und verursa-

chen Verunreinigungen. Ausbrüche, Risse und Kratzer entstehen nach Kontakt mit Maschinenteilen oder beim Transport der Wafer an oder auf der Kante von Wafern. Es kommt zu Spannungserhöhungen, welche Kratzer oder Risse verursachen, wenn bereits Defekte in der Kristallebene vorherrschen. Zur Prävention dieser Defekte werden die Kanten abgerundet. [13, S. 9 f]

Der Mensch ist zum großen Teil Ursache für Verunreinigung, da Partikel beim Ausatmen, Wimpern und oder abgestorbene Haut die Wafer verunreinigen können. Deshalb findet der Prozess der Waferherstellung in Reinräumen statt. Trotz dieser Vorkehrungen können Partikel bei Bewegungen entstehen und so an die Wafer gelangen. [13, S. 8 f]

Partikel treten vor allem an Kantendefekten auf und verursachen Kratzer oder Ausbrüche. In späteren Prozessschritten können sie zu Verunreinigungen führen. Es ist sinnvoll, zwischen mechanischen und rein produktionsbedingten Defekten zu unterscheiden. Ursache für die mechanischen Defekte sind zum Beispiel im Transport, beim Handling und der Rotation der Wafer zu erwarten, während die produktionsbedingten Defekte auf Fehler im chemisch-mechanischen Polieren oder Reiningungsvorgänge zurückzuführen sind. [13, S. 7 f]

Des Weiteren können sich Rückstände bei chemischen Prozessen und durch Trocknungsrückstände auf den Wafern bilden. [13, S. 9 f]

Kapitel 3

Theorie zur explorativen Modellbildung

3.1 Maschinelles Lernen

Als maschinelles Lernen wird das Entwickeln und Anwenden von Algorithmen bezeichnet zur Approximation von Zusammenhängen, in denen es keine analytischen Lösungen gibt. Anhand von Daten wird ein Modell entwickelt, mit welchem es möglich ist, komplexe Aufgaben zu lösen. [14, S. 15]

Ein Beispiel aus der heutigen Zeit für maschinelles Lernen ist der Spamfilter bei E-Mails. Der Spamschutz basiert auf einem maschinellen Lernmodell, das darauf trainiert ist, Spam-E-Mails von normalen E-Mails zu unterscheiden. Das Training wird anhand von Beispielen durchgeführt, die entweder vom Benutzer als Spam markiert wurden oder als gewöhnliche E-Mails erkannt wurden. Diese Beispiele bilden den Trainingsdatensatz, wobei jeder einzelne als Trainingsdatenpunkt bezeichnet wird. Das Ziel besteht darin, neue E-Mails als Spam zu identifizieren, basierend auf den Erfahrungen des Systems, die aus dem Trainingsdatensatz stammen. Um die Leistung des Systems zu bewerten, muss ein Leistungsmaß definiert werden, wie z.B. der Anteil der korrekt klassifizierten E-Mails. Dies wird als Genauigkeit bezeichnet und ist ein häufig verwendetes Leistungsmaß bei Klassifikationsaufgaben. [15, S. 4]

3.1.1 Arten des maschinellen Lernens

Es existieren verschiedene Arten des maschinellen Lernens, überwachtes, unüberwachtes und bestärkendes Lernen. Letzteres wird auch Reinforcement Learning genannt. [14, S. 15f]

Ziel des überwachten Lernen ist es, aus einer gegebenen Eingabe $\mathcal{X}$ und einer gegebenen Ausgabe $\mathcal{Y}$ ein Abbild der Eingabe auf die Ausgabe zu erlernen. Dafür werden die Datenpunkte gelabelt. [15, S. 9], [16, 10] Zu lösende Probleme des überwachten Lernens sind die Regression und die Klassifikation. Bei der Klassifikation werden die Eingaben in verschiedene Kategorien klassifiziert und die Zielmenge ist abzählbar. Der bereits erwähnte Spamfilter ist hierfür ein gutes Beispiel. Der Filter klassifiziert

neu eingegangene Mails in Spam-E-Mails und normale E-Mails. In Abbildung 3.1 sind die Klassen und die gelabelten Daten zu sehen. [15, S. 10] Die Regression hat das Ziel, eine überabzählbare Zielgröße vorherzusagen. Bei Regression und Klassifikation werden verschiedene Merkmale und Prädikatoren als Eingabedaten genommen und daraus die wahrscheinlichste Zielvariable berechnet. [16, S. 10] Die Eingabedaten können Bilder, Tabellen oder auch Videos und Audiodatein sein.

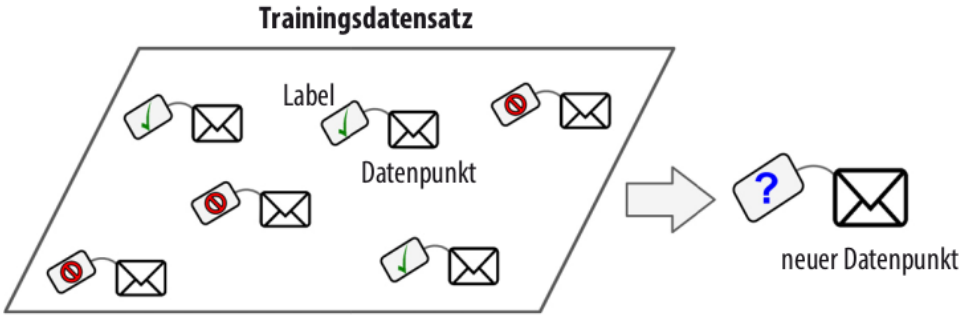


Abbildung 3.1: Labeln von E-Mails [16, S. 9]

Das unüberwachte Lernen hat die Eigenschaft, dass die Daten nicht gelabelt werden und nur die Eingabewerte verfügbar sind. Es sollen Strukturen in den Eingabedaten erkannt werden. Ein Beispiel für unüberwachtes Lernen ist das Besuchen eines Blogs im Internet. Angenommen, es liegen umfangreiche Daten über die Besucher eines Blogs vor und es besteht das Interesse, ähnliche Besuchergruppen mittels eines Clustering-Algorithmus zu identifizieren. Der Algorithmus wird ohne eine vorherige Kenntnis über die Gruppierung der Besucher eingesetzt und ermittelt eigenständig die Verbindungen zwischen ihnen. Dabei könnte der Algorithmus erkennen, dass 40% der Besucher Männer sind, die eine Vorliebe für Comics haben und den Blog am Abend frequentieren. Im Kontrast dazu sind 20% der Besucher junge Science-Fiction-Fans, die den Blog am Wochenende aufsuchen. Durch die Anwendung eines hierarchischen Clustering-Verfahrens können die ermittelten Gruppen weiter untergliedert werden. Dies kann von Nutzen sein, wenn zielgruppenspezifische Blog-Artikel verfasst werden sollen. Die Abbildung 3.2 zeigt solch ein Clustering beispielhaft. [15, S. 12]

Beim bestärkenden Lernen handelt es sich um einen Bereich des maschinellen Lernens, bei dem sich ein Software-Agent in einer Umgebung befindet und dort interagiert, um zu lernen, wie er eine kumulative Belohnung erhöht. Es werden Markov-Entscheidungsprozesse verwendet, um die Interaktion zwischen einem lernenden Agenten und seiner Umwelt in Form von Zuständen, Aktionen und Belohnungen zu definieren. Die Policy beschreibt dabei die Art und Weise, wie der Agent handelt. Die Belohnung ist das Ziel des Vorgehens. [17]. Die Belohnung kann dabei sowohl positiv, als auch negativ (Strafe) sein. Das System muss die beste Policy finden, um in einem ge-

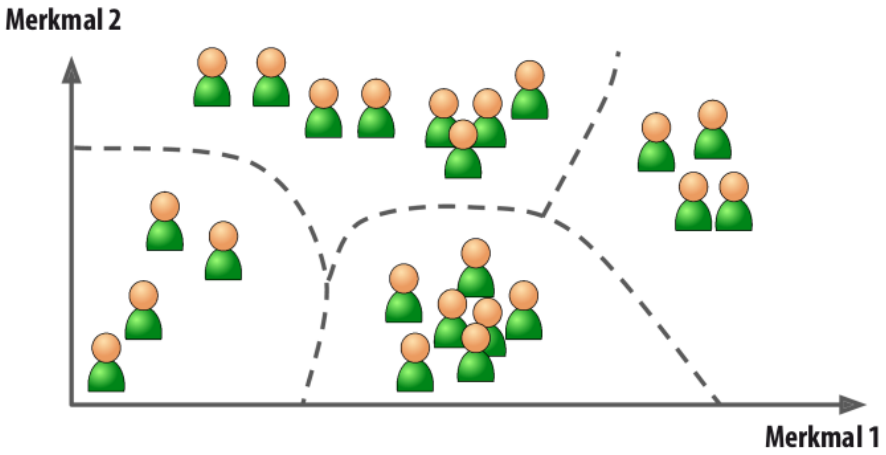


Abbildung 3.2: Beispiel für Clustering [15, S. 12]

wissen Zeitraum die maximale Belohnung zu erhalten [15, S. 15]. In Abbildung 3.3 ist ein Beispiel für diese Art des maschinellen Lernens abgebildet.

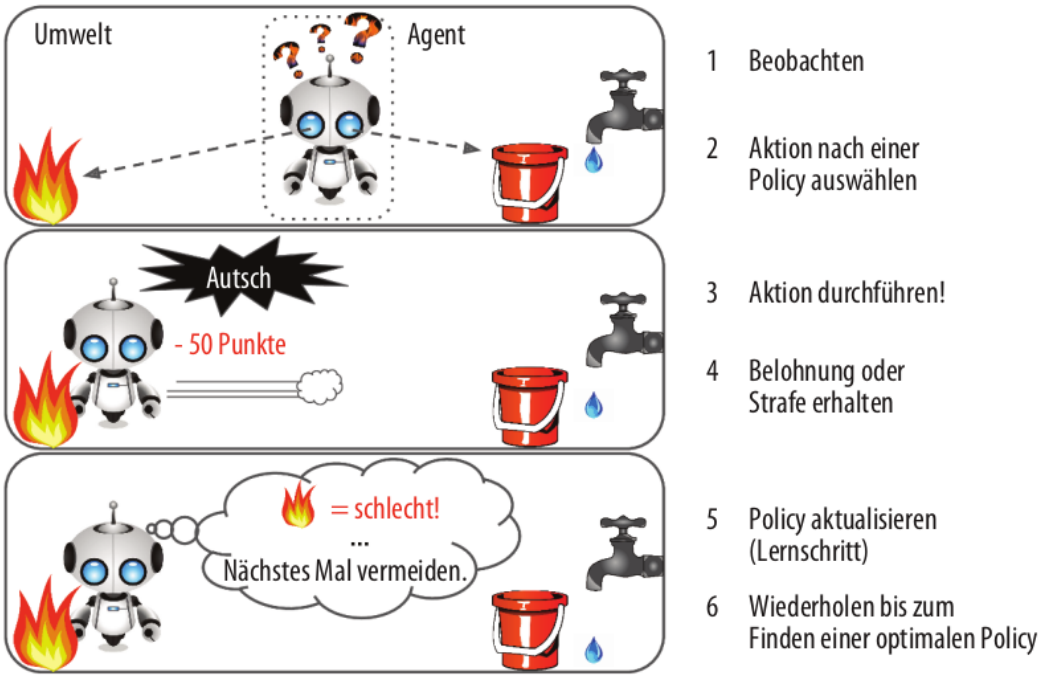


Abbildung 3.3: Beispiel für Reinforcement Learning [15, S. 15]

Die in Kapitel 3.1.2 aufgeführten Grundlagen gelten für das für alle Arten des maschinellen Lernen.

3.1.2 Grundlagen des maschinellen Lernens

Wie bereits im vergangenen Abschnitt erwähnt, existiert eine Eingabe x und die unbekannte Zielfunktion $f : \mathcal{X} \mapsto \mathcal{Y}$, wobei $\mathcal{X}$ der Eingaberaum und $\mathcal{Y}$ der Ausgaberaum ist. Dabei ist $\mathcal{X}$ die Menge aller Eingaben (Merkmalsraum) x und $\mathcal{Y}$ die Menge aller Ausgaben. Es existiert ein Datensatz S, der aus Eingabe- und Ausgabebeispielen $(x_1, y_1), \ldots, (x_N, y_N)$ besteht, wobei $y_n = f(x_n)$ für $n = 1, \ldots, N$ gilt. Diese Trainingsbeispiele werden auch als Datenpunkte (englisch: sample points) bezeichnet. [18, S. 3]

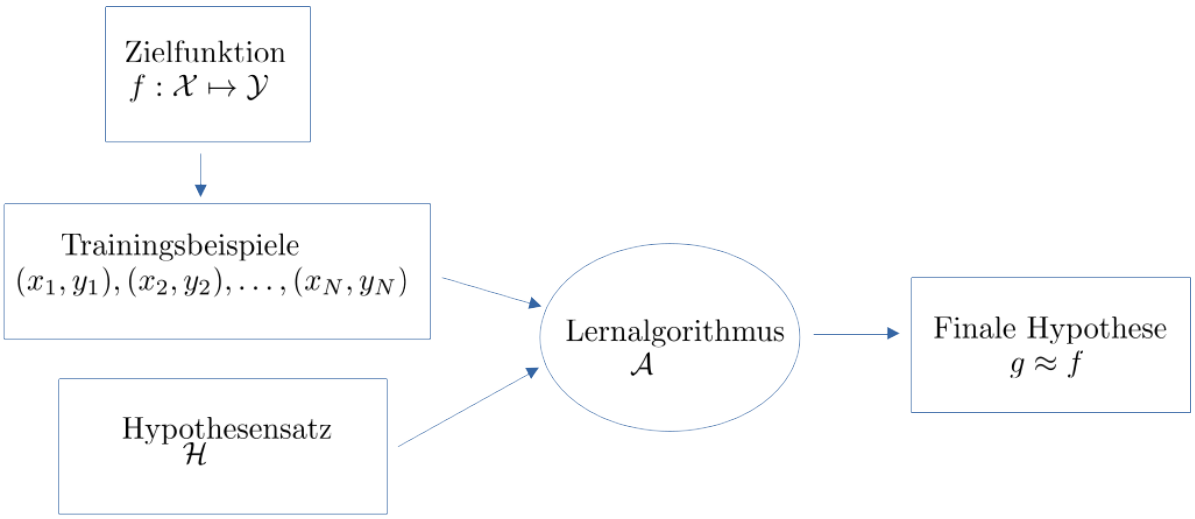


Abbildung 3.4: Grundlegender Aufbau des Lernproblems (in Anlehnung an Grafik aus [18, S. 4])

Der Lernalgorithmus $\mathcal{A}$, dargestellt in Abbildung 3.4, wählt bei gegebener Samplemenge S, die Hypothese $g : \mathcal{X} \mapsto \mathcal{Y}$ aus, die f mit dem geringsten Fehler approximiert. Der Lernalgorithmus wählt g aus einem Hypothesenraum $\mathcal{H}$, der für jedes Lernproblem angepasst sein muss. Der Hypothesenraum könnte z. B. die Menge aller linearen Formeln sein, aus der der Lernalgorithmus die lineare Anpassung wählt, die den geringsten Fehler bei den Trainingsdaten hat. [18, S. 3]

Ein Modell $g(x_n \mid \theta)$ muss gefunden werden. Die Parameter des Modells werden durch θ beschrieben.

Die Entscheidung des Loss (deutsch: Verlust), der Verlustfunktion $L(\cdot)$ wird getroffen, damit die Differenz zwischen der gewünschten Ausgabe y_n und der Approximation $g(x_n \mid \theta)$ unter Verwendung der Parameterwerte θ berechnet werden kann. Der Näherungsfehler ist definiert als die Summe der Verluste aller Instanzen und ist wie folgt definiert: [16, S. 46]

$$E(\theta \mid \mathcal{X}) = \sum_{n} L(y_n, g(x_n \mid \theta)). \tag{3.1}$$

Im Falle des Klassenlernens, bei dem die Ausgabewerte 0 oder 1 annehmen, wird die Gleichheit durch die Verlustfunktion $L(\cdot)$ überprüft. [16, S. 46]

Um sicherzustellen, dass dieser Ansatz effektiv ist, müssen die folgenden Bedingungen erfüllt sein: Die Hypothesenklasse von $g(\cdot)$ muss groß genug sein oder eine ausreichende Mächtigkeit besitzen, um die unbekannte Funktion zu erfassen, die die verrauschten Daten generiert hat, die durch y_n dargestellt werden. Des Weiteren muss es ausreichend Trainingsdaten geben, um die korrekte Hypothese (oder eine ausreichend gute) innerhalb der Hypothesenklasse zu identifizieren zu können. Zum Schluss wird eine geeignete Optimierungsmethode gebraucht, um die korrekte Hypothese zu finden. [16, S. 46]

3.2 Arbeitsschritte bei der explorativen Modellbildung

3.2.1 Datenexploration

Datenexploration hilft dabei, Hypothesen zu erstellen. Es ist möglich, aus der Visualisierung der Daten ein Verständnis für diese zu entwickeln und über weiterführende Schritte bei der Modellbildung zu entscheiden [19, S.1]. In diesem Prozess wird zunächst der Datensatz eingelesen und anschließend werden die Daten genauer untersucht. Dabei werden auch Bilddaten dargestellt und die Maxima und Minima bestimmt. Es erfolgt eine Erfassung der Anzahl der Daten und deren Label. Des Weiteren wird untersucht, ob es sogenannte „Missing Values“ (fehlende Werte) oder „Outlier“ (Ausreißer) in den Daten gibt.

3.2.2 Datenvorverarbeitung

Bei der Datenvorverarbeitung muss der Datensatz so angepasst werden, dass sich die Daten zum Trainieren eignen. Die Missing Values und die Ausreißer werden im Datensatz ersetzt oder aus diesem entfernt. Bilddateien werden auf eine einheitliche Größe skaliert.

Für die Anwendung werden die Samplepunkte in Trainings-, Test- und Validierungsdaten unterteilt. Die Trainingsdaten dienen dazu, dass das Modell trainiert wird. [20]

Mittels Testdaten wird bestimmt, ob das maschinelle Lernmodell zur Generalisierung geeignet ist. [16, S. 44] Die Generalisierung bezieht sich auf die Eignung eines Modells, das auf den Trainingsdaten trainiert wurde, genaue Vorhersagen für neue Instanzen zu

treffen. [16, S. 44] Es ist wichtig, dass die Testdaten sowohl zur Parameter- also auch zur Hyperparameteroptimierung nicht genutzt werden. [21, S. 22] Zur Hyperparameteroptimierung werden die Validierungsdaten genutzt.

In den meisten Fällen besteht der Testdatensatz aus 20% aller Datenpunkte, die zufällig gewählt werden. [15, S. 53] Es müssen Datenpunkte aus allen Klassen im Testdatensatz enthalten sein, die sowohl kein Teil der Test- und als auch der Validierungsmenge sind. [16, S. 42]

Die Klassenverteilung muss in allen drei Datenmengen, Training-, Validierungs-, und Testdaten, der Samplepunkte prozentual gleich sein.

3.3 Aufbau eines faltenden neuronalen Netzes

Das faltende neuronale Netz, auch *CNN* genannt, wird unter anderem zur Bilderkennung verwendet [15, S. 449]. Da in dieser Arbeit ein Datensatz mit Waferbildern verwendet wird, sollen *CNN*s im folgenden Abschnitt erläutert werden.

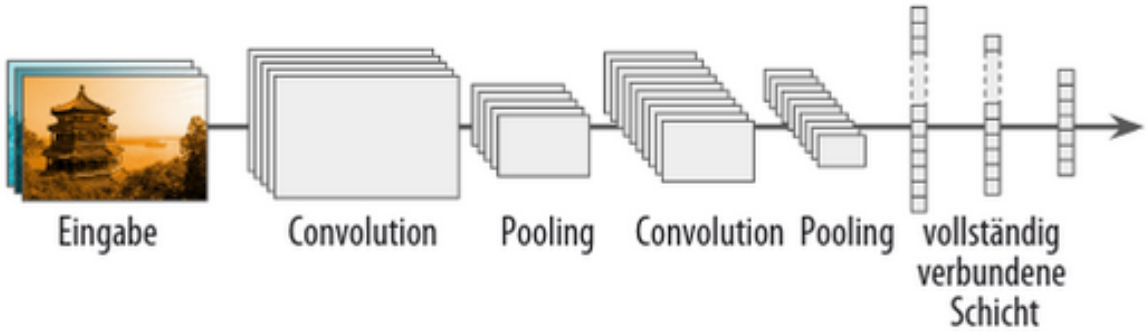


Abbildung 3.5: Beispiel für ein CNN [15, S. 464]

Ein einfaches neuronales Netz besteht aus künstlichen Neuronen. Ein künstliches Neuron verfügt über mindestens eine binäre Eingangsleitung sowie eine binäre Ausgangsleitung. Die Aktivierung des Neurons erfolgt nur dann, wenn eine festgelegte Anzahl von Eingangsleitungen aktiviert ist. [15, S. 285] In einem mehrschichtigen neuronalen Netz ist ein Neuron in einer vollständig verbundenen Schicht (englisch: Dense-Layer), wie in Abbildung 3.5 mit k Neuronen aus der vorherigen Schicht verbunden und wird durch eine Funktion $f : \mathbb{R}_k \to \mathbb{R}$ beschrieben. Diese Funktion besteht aus einer linearen Abbildung $q : \mathbb{R}_k \to \mathbb{R}$ und einer nichtlinearen Aktivierungsfunktion $\sigma : \mathbb{R} \to \mathbb{R}$. Konkret bedeutet dies

$$f(z_1, ..., z_k) = \sigma(q(z, ..., z_k)), \tag{3.2}$$

mit $z_1, z_2, ..., z_k$ als Ausgabe der Neuronen aus der vorherigen Schicht. Die lineare Abbildung wird durch

$$q(z_1, ..., z_k) = \sum_{j=1}^{k} w_j z_j + b \tag{3.3}$$

parametrisiert. Die reellen Zahlen $w_1, w_2, ..., w_k$ werden als Gewichte bezeichnet und repräsentieren die „Stärke“ der Verbindungen zu den Neuronen in der vorherigen Schicht. Die reelle Zahl b, der Bias, ist ein konstanter Offset. Während des Trainings des Netzwerks müssen Gewichte und Bias als Parameter optimiert werden. [21, S. 26 f]

Für die Mustererkennung in Bildern werden Faltungsschichten (englisch: Convolutional-Layer) in *CNN*s verwendet (siehe Abbildung 3.5). Ein Bild besteht aus mehreren aneinander gereihten Pixeln. Jedem Pixel ist eine Zahl oder eine Gruppe von Zahlen zugeordnet [22]. Mehrere Pixel können zu einem Patch zusammengefasst werden. Dabei handelt es sich dann um einen Ausschnitt des Bildes. Die *CNN*-Modelle sollen dabei in einem Patch ein bestimmtes Muster erkennen. In einem Bild soll das Modell beispielsweise die Umrisse eines Hauses, von denen eines Himmels unterscheiden können. [14, S. 95 f] Damit ein Muster erkannt wird, erlernt ein *CNN*-Modell die Parameter einer Matrix F (Filter-Matrix), welche die Dimension $p \times p$ besitzt. Dabei ist p die Größe des Patches. Zur Vereinfachung des Beispiels wurde hierbei ein Schwarz-Weiß-Bild verwendet. Der Wert 1 steht dabei für die Farbe Schwarz und 0 für weiß. Die Pixel des Bildes können also nur die Werte 0 und 1 annehmen. Die Matrix P_1 ist ein Beispiel für ein solches Patch:

$$P_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Die Matrix P_1 hat das Muster eines Kreuzes. Damit das Regressionsmodell genau dieses Muster erkennt, muss sie sich die Parametermatrix F mit der gleichen Dimension, 3×3, aneignen. Die Parameter an den Stellen im Eingabepatch, an denen sich Einsen befinden, sind positiv, während die Parameter an den Stellen, an denen sich Nullen befinden, nahe bei Null liegen. Der Wert, der durch die Faltung der Matrizen P und F berechnet wird, ist umso höher, je größer die Ähnlichkeit zwischen F und P ist. [14, S. 96] Als Beispiel einer Faltung wird folgende Matrix F gegeben:

$$F = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{bmatrix}$$

Die Faltung der zwei Matrizen P_1 und F wird wie folgt berechnet:

$$\overbrace{\begin{bmatrix}0 & 1 & 0\\ 1 & 1 & 1\\ 0 & 1 & 0\end{bmatrix}}^{P_1} \text{ Faltung } \overbrace{\begin{bmatrix}0 & 2 & 3\\ 2 & 4 & 1\\ 0 & 3 & 0\end{bmatrix}}^{F} \xrightarrow{\text{Überlagerung}} \begin{bmatrix}0\cdot 0 & 1\cdot 2 & 0\cdot 3\\ 1\cdot 2 & 1\cdot 4 & 1\cdot 1\\ 0\cdot 0 & 1\cdot 3 & 0\cdot 0\end{bmatrix} \xrightarrow{\text{Summe}} \begin{bmatrix}12\end{bmatrix}$$

Hat der Patch ein anderes Muster, zum Beispiel die Form eines Ls (P_2), besitzt die Faltung mit F den Wert 5, welcher niedriger ist als 12. [14, S. 97]

$$P_2 = \begin{bmatrix}1 & 0 & 0\\ 1 & 0 & 0\\ 1 & 1 & 1\end{bmatrix}$$

Es gilt, je größer die Ähnlichkeit zwischen Patch und Filter ist, desto höher ist der Faltungswert. Allen Filtern F ist ein Bias-Parameter b zugeteilt, welcher dem Faltungsergebnis angefügt wird. Dies geschieht, bevor die Aktivierungsfunktion σ verwendet wird. [14, S. 97]

Ein *CNN* beinhaltet mehrere Schichten, welche mehrere Faltungsfilter besitzen. Die Position des Filters wird von links nach rechts und von oben nach unten über die Eingangsabbildung verändert und „faltet“ den Patch. Dies geschieht, indem nach jeder Filterbewegung der Wert der Faltung bestimmt wird. Dieser Prozess wird in Abbildung 3.6 dargestellt. [14, S. 97]

Die Abbildung 3.6 stellt sechs Schritte einer Faltung mit einem Filter mit einer Schrittweite von 1 dar. Dabei bewegt sich der Filter also um eine Stelle nach rechts und respektive um eine Stelle nach unten. Eine Aktivierungsfunktion wird auf Faltungssumme und den Bias-Term angewandt. Im Normalfall findet in den verdeckten Schichten die *Rectified Linear Unit* (*ReLU*)-Aktivierungsfunktion Verwendung. Die Aktivierungsfunktion der letzten Schicht ist von der Aufgabenstellung abhängig. [14, S. 98]

Besitzt das *CNN* mindestens zwei Faltungsschichten, wird die Ausgabe der ersten Schicht als Sammlung von Bildmatrizen der Größe $size_l$ betrachtet. Dies wird als Volume bezeichnet. Die Sammlungsgröße nennt sich Tiefe des Volumes. Der Filter der zweiten Schicht $l + 1$ faltet das Gesamtvolumen. Wird nun die Faltung eines Patches eines Volumes betrachtet, ist diese die Endsumme dieser Faltungen der Patches der verschiedenen Matrizen. [14, S. 98]

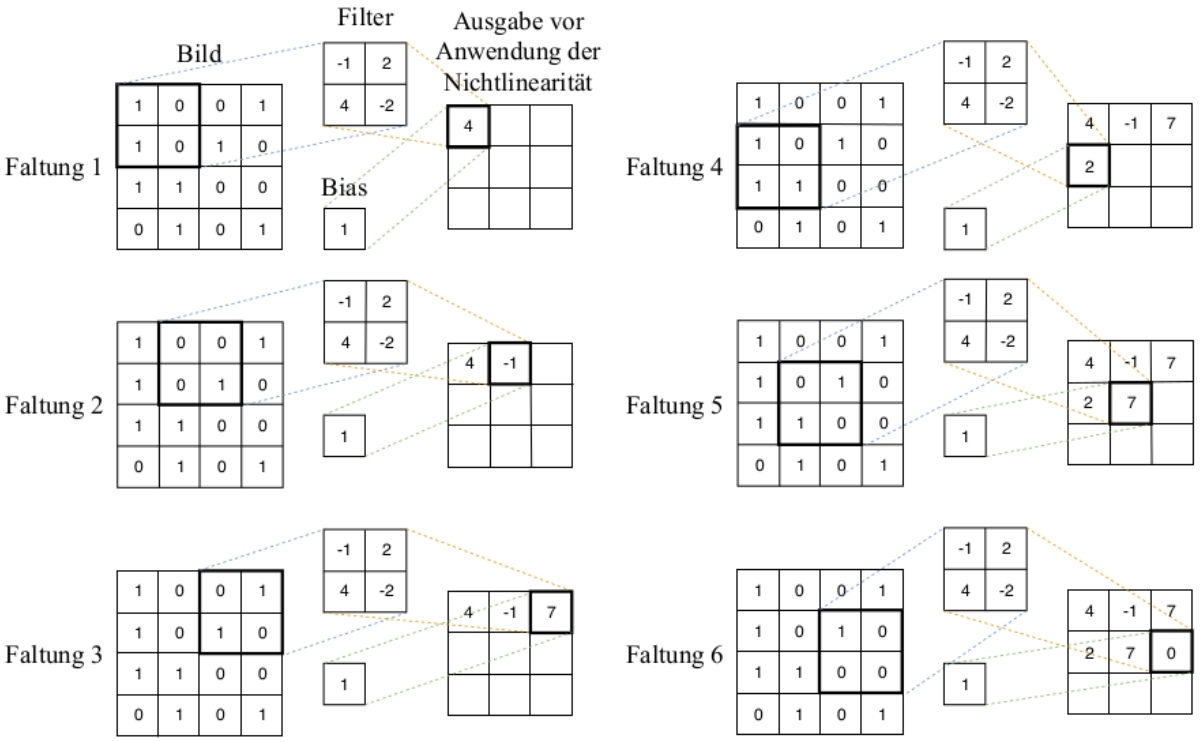


Abbildung 3.6: Faltung eines Bildes [14, S. 98]

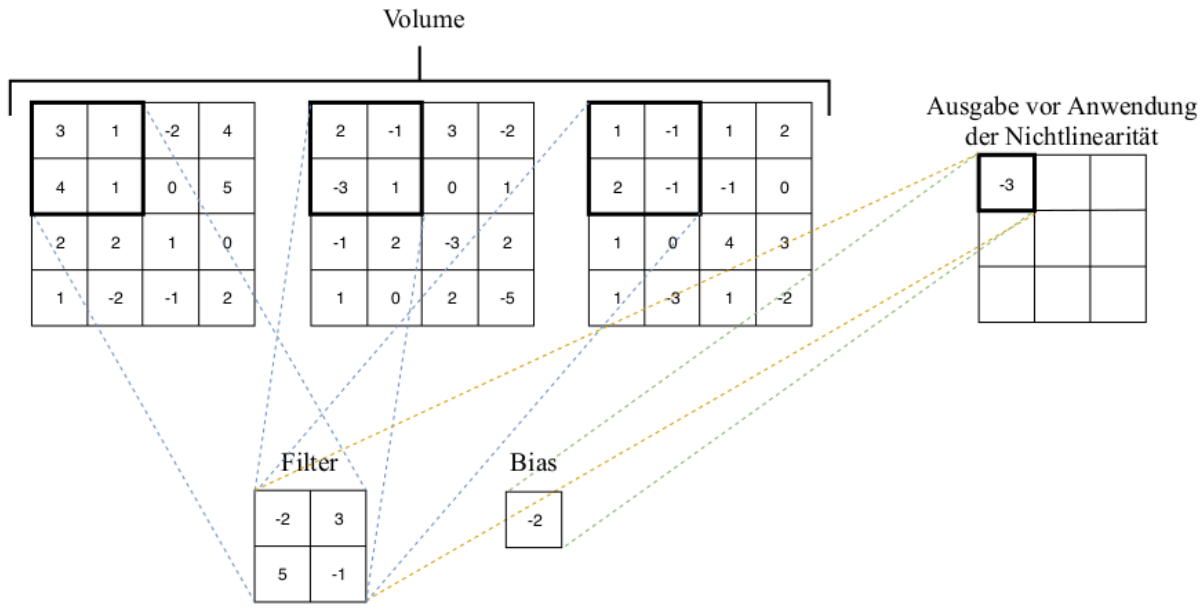


Abbildung 3.7: Faltung eines Volume, bestehend aus drei Matrizen [14, S. 99]

In Abbildung 3.7 ist ein Beispiel für Faltung eines Volumes, das aus drei Matrizen besteht, dargestellt. Der Faltungswert ergibt -3.

Anhand von Padding ist es möglich, größere Ausgabematrizen zu erlangen. Dadurch ist es möglich die Breite eines künstlichen Randes um das Bild zu definieren.

Diese Randzellen sind im Normalfall mit Nullen beschriftet [Quelle: Buch 3, S.101]. Die Abbildung 3.8 zeigt den mit Nullen beschrifteten Rand eines Bildes. [14, S. 99]

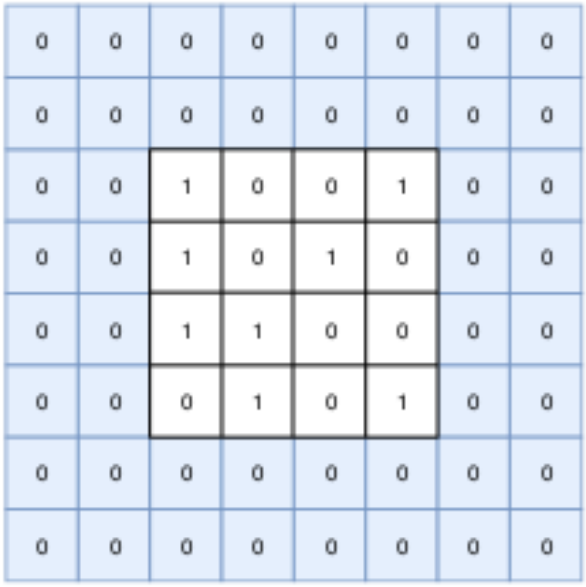


Abbildung 3.8: Padding eines Bildes mit einem Rand mit der Größe von zwei Zellen [14, S. 101]

Als Pooling wird ein Verfahren bezeichnet, das ähnlich wie die Faltung funktioniert. Die Schicht verwendet dabei den *max*- oder den *average*-Operator. Bei der Verwendung vom Pooling-Layer existieren Hyperparameter. Diese sind die Größe und die Schrittweite des Filters. [14, S. 101]

Im Normalfall schließt sich, wie in Abbildung 3.5 zu sehen, der Pooling-Layer dem Convolution-Layer an und beinhaltet das Ergebnis der Faltung als Eingabe. Wird Pooling bei Volume eingesetzt, werden die Matrizen separat behandelt. Folglich ergibt die Anwendung einer Pooling-Schicht auf ein Volume ein neues Volume mit derselben Tiefe wie die Eingabe. Im Pooling-Prozess werden ausschließlich Hyperparameter verwendet, und es sind keine zu erlernenden Parameter erforderlich. Im Normalfall trägt das Anwenden von Pooling dazu bei, die Klassifikationsgenauigkeit des Modells zu verbessern. [14, S. 101]

Die einfachste Form eines Lernmodells ist das Sequential-Modell bestehend aus einem Stack mit Schichten, welche fortlaufend verknüpft sind. [15, S. 300]

Der Flatten-Layer komprimiert den mehrdimensionalen Output in einen eindimensionalen Vektor. [23].

In jeder Dense-Schicht wird eine eigene Gewichtsmatrix erstellt, die sämtliche Verbindungsgewichte zwischen den Neuronen und ihrer Eigenwerte beinhaltet. Zusätzlich

wird ein Vektor mit Bias-Termen erstellt, wobei jedem Neuron ein Bias-Term zugewiesen wird. [15, S. 301]

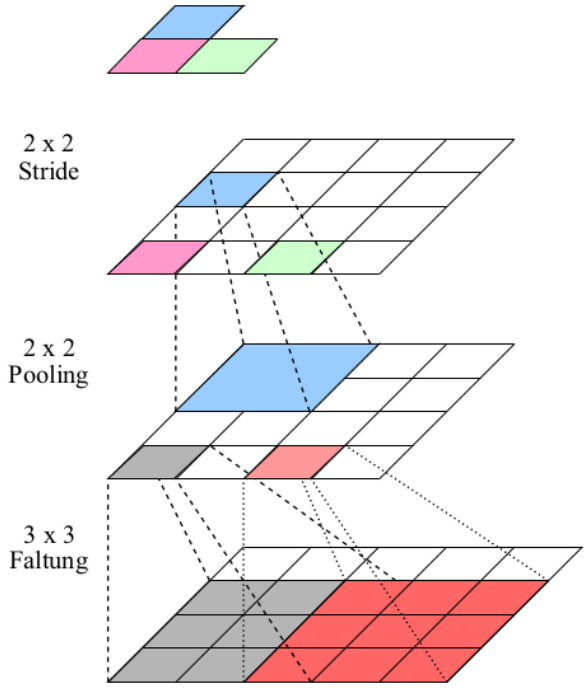


Abbildung 3.9: Grundlegende Schritte des CNN [16, 350]

Die Abbildung 3.9 zeigt eine 3x3-Faltungsschicht, die die Faltung an allen Punkten anwendet. Die Pooling-Schicht führt die Werte in einem 2 × 2-Patch zusammen, und ein Stride von 2 verringert die Auflösung um die Hälfte. [16, 350]

3.4 Methoden der Hyperparameteroptimierung

Bei der Hyperparameteroptimierung werden die Parameter, die nicht direkt aus den Daten gelernt werden können, sondern vom Entwickler des Modells ausgewählt werden müssen, optimiert. Die Einstellungen der Hyperparameter beeinflussen die Leistung von Modellen. [14, S. 86] Für die Hyperparameteroptimierung stehen verschiedene Verfahren zur Auswahl.

Eine der am häufigsten verwendeten Methoden ist die Grid Search. Bei dieser Methode werden verschiedene Kombinationen von Hyperparametern aus dem Hyperparametersuchraum ausgewählt und alle Modelle für jede Kombination von Hyperparametern trainiert und validiert. Diese Methode ist mit zunehmender Hyperparameteranzahl zeitaufwendig und rechenintensiv. [14, S. 87]

Eine weitere Methode ist die Random Search. Hierbei werden Hyperparameter zufällig ausgewählt, anstatt alle möglichen Kombinationen abzudecken. Diese Methode garantiert jedoch nicht, das Modell mit der besten Performance im Suchraum zu finden.

Der Bayes Search ist eine weitere gängige Methode zur Hyperparameteroptimierung. Bei dieser Methode wird ein probabilistisches Modell verwendet, um die Hyperparameter zu optimieren. Das Modell berücksichtigt die Ergebnisse der Modellvalidierungen, um die nächste Hyperparameterauswahl zu wählen. Diese Methode kann eine gute Balance zwischen Effizienz und Genauigkeit bieten. [14, S. 87]

In dieser Arbeit wird jedoch das Verfahren der Hyperparameteroptimierung *Adaptive Cross Search* (*ACS*) von der Data Science Research Group eingesetzt. Die *ACS*-Methode ermöglicht eine effiziente Optimierung der Hyperparameter, insbesondere bei hochdimensionalen Suchräumen reduziert sich der Aufwand zum Optimieren der Hyperparameter. Das *ACS* geht bei der Hyperparameterauswahl wie folgt vor. Ausgehend von einer Startkonfiguration werden die Hyperparameter in einer vorab gewählten Reihenfolge durchlaufen. In jeder Iteration werden alle Hyperparameter festgesetzt bis auf einen Hyperparameter, der mithilfe von Grid Search, Random Search oder Bayes Search nach dem Minimum für das Fehlermaß durchsucht wird. Dieser Prozess wird für eine vordefinierte Anzahl an Iterationen wiederholt. Durch diese schrittweise Optimierung der Hyperparameter konvergiert das Verfahren mit ausreichend vielen Iterationen zum Minimum für das Fehlermaß im Hyperparametersuchraum. [26]

3.5 Hyperparameter des CNNs

Hyperparameter sind ein wichtiger Aspekt in der Konstruktion von *CNN*. Sie beeinflussen maßgeblich die Leistung des Modells und müssen sorgfältig gewählt werden, um eine optimale Leistung zu erzielen.

Da die Größe des Faltungsfilters als einer der Hyperparameter bei der Auswahl berücksichtigt werden muss, muss eine Entscheidung bezüglich der optimalen Größe des Filters getroffen werden. Eine vergleichende Analyse zwischen einer kleineren und einer größeren Filtergröße ist notwendig, um die optimale Filtergröße zu ermitteln. Bei einer ungeraden Filtergröße würden alle Pixel der vorherigen Schicht symmetrisch um das Ausgabepixel angeordnet sein. Wenn jedoch ein Filter mit gerader Größe verwendet wird, muss die Möglichkeit von Verzerrungen über die Schichten hinweg berücksichtigt werden, weshalb Filter mit geradem Kern meist vermieden werden, um eine einfache Implementierung zu gewährleisten. Wenn sich die Faltung als eine Interpolation von den gegebenen Pixeln zu einem mittleren Pixel vorstellt wird, ist es mit einem Filter gerader Größe nicht möglich, zu einem mittleren Pixel zu interpolieren. [24]

Die Anzahl der Filter, die pro Schicht angewendet werden, kann ebenfalls variiert werden. Je mehr Filter verwendet werden, desto mehr Merkmale werden überprüft [16, S. 348]. Daraus folgt, dass bei einer größeren Filteranzahl mehr Rechenleistung notwendig ist.

Die Anzahl der Pixel, die der Filter bei jedem Schritt über das Eingangsbild bewegt wird, wird als Schrittweite bezeichnet. Eine größere Schrittweite führt zu einer Verringerung der Dimensionen des Ausgangsbildes. [25]

Als Padding-Typ wird die Art der Padding-Methode bezeichnet, die auf das Eingangsbild angewendet wird. Zero Padding fügt Nullen um das Eingangsbild herum hinzu, während Valid Padding keine zusätzlichen Pixel hinzufügt. [15, S. 457 f]

Die Aktivierungsfunktion wird auf die Ausgangswerte des Filters angewendet. Sie ist eine festgelegte und im Normalfall nicht lineare Funktion [14, S. 89f]. Die gängigsten Aktivierungsfunktionen sind *ReLU*, sigmoid und tanh. In Abbildung 3.10 sind die Step-, Sigmoid-, Tangens hyperbolicus (tanh)- und die *ReLU*-Funktionen und deren Ableitungen grafisch dargestellt.

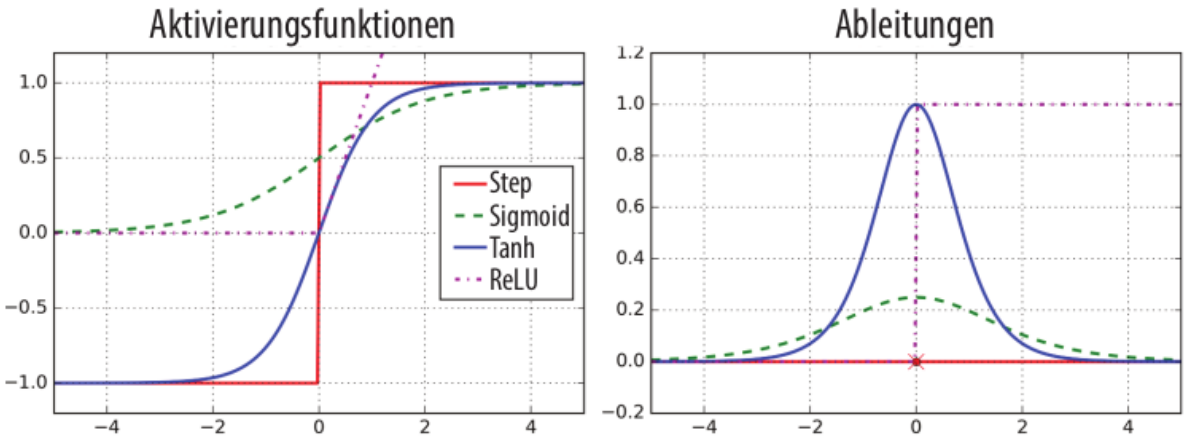


Abbildung 3.10: Aktivierungsfunktionen und ihre Ableitungen [15, S. 293]

Die *ReLU*-Funktion, die definiert ist als $ReLU(z) = max(0, z)$, ist in der Praxis weit verbreitet [15, S. 293]. Die Abkürzung „ReLU“ steht für rektifizierte lineare Einheit [21, S. 27]. Obwohl diese Funktion stetig ist, ist sie bei z = 0 nicht differenzierbar, da die Steigung sprunghaft ansteigt und die Gradientenmethode somit unvorhersehbar reagieren kann. Zudem ist die Ableitung der Funktion für z < 0 gleich null. Trotz dieser Einschränkungen hat sich die ReLU-Funktion aufgrund ihrer schnellen Berechenbarkeit als Standard etabliert. Ein weiterer Vorteil besteht darin, dass sie keinen maximalen Ausgabewert besitzt, was dazu beiträgt, Probleme im Zusammenhang mit dem Gradientenverfahren zu vermeiden. [15, S. 293]

Die Sigmoid-Funktion hat die Form [14, S. 44]:

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{3.4}$$

Die Aktivierungsfunktion Softmax-Funktion wird bei Klassfikationsproblemen eingesetzt. Sie ist eine Erweiterung der Sigmoid-Funktion auf Ausgaben mit mehreren Dimensionen und ist wie folgt definiert [14, S. 103]:

$$\sigma(z) := [\sigma^{(1)}, \ldots, \sigma^{D}], \quad \text{wobei } \sigma^{(j)} := \frac{\exp(z^{(j)}}{\sum_{j=1}^{D} \exp(z^{(k)}} \tag{3.5}$$

Dabei gilt $\sum_{j=1}^{D} = 1$ und $\sigma^{(j)} > 0, \forall j$. [14, S. 103]

Bei dem Variieren der Pooling-Größe wird die Größe des Fensters, das über das Ausgangsbild des Filters bewegt wird, verkleinert, um die Größe des Bildes zu reduzieren und die Anzahl der Parameter zu verringern. Ein Nebeneffekt hiervon ist die Senkung der Wahrscheinlichkeit des Overfittings. [15, S. 460] Overfitting wird im Abschnitt 3.6 definiert.

Aufgrund der Tatsache, dass Dense-Layer viele Parameter besitzen, haben tiefere Netzwerke eine hohe Genauigkeit, benötigen aber auch mehr Trainingsdaten und haben höhere Rechenkosten. Bei einer begrenzten Anzahl an Trainingsdaten birgt dies auch das Risiko des Overfittings. [15, S. 302]

Die Wahl der Lernrate ist ein wichtiger Hyperparameter bei der Modellierung. In der Regel liegt die optimale Lernrate bei der Hälfte der maximalen Lernrate, über der der Trainingsalgorithmus divergiert. Eine gängige Methode zur Bestimmung einer geeigneten Lernrate besteht darin, das Modell für mehrere hundert Iterationen zu trainieren, beginnend mit einer sehr niedrigen Lernrate (zum Beispiel 10^{-5}) und diese schrittweise bis zu einem sehr hohen Wert zu erhöhen (zum Beispiel 10), indem die Lernrate bei jeder Iteration um einen konstanten Faktor multipliziert wird (z.B. durch Anwendung von $\exp(log(10^6)/500)$ für 500 Schritte von 10^{-5} bis 10). Der Verlust sollte als Funktion der Lernrate aufgetragen werden (mit einer logarithmischen Skala für die Lernrate), um zu beobachten, wie er anfangs abfällt. Nach einer gewissen Zeit wird er jedoch zu groß sein und der Verlust wird schnell steigen. Die optimale Lernrate wird etwas unterhalb des Punktes liegen, an dem der Verlust wieder zu wachsen beginnt (meist etwa zehnmal niedriger als der Wendepunkt). Das Modell kann dann mit der optimalen Lernrate neu initialisiert und trainiert werden. [15, S. 328]

Die Batch-Größe beeinflusst die Leistung und Trainingsdauer des Modells erheblich. Der wichtigste Vorteil einer großen Batch-Größe ist, dass sie effizient von Hardwarebeschleunigern wie *Graphics Processing Unit* (*GPU*) verarbeitet werden kann, was dazu führt, dass der Trainingsalgorithmus mehr Instanzen pro Sekunde verarbeiten kann. Die Empfehlung von vielen Forschern und Praktikern ist, die größtmögliche

Batch-Größe zu wählen, die in den Speicher der *GPU* passt. Trotzdem können große Batch-Größen in der Praxis zu Beginn des Trainings zu Instabilitäten führen, und das daraus resultierende Modell kann möglicherweise nicht so gut generalisieren wie ein Modell mit einer geringeren Batch-Größe. [15, S. 238f]

3.6 Auffinden der richtigen Modellkomplexität: Bias-Varianz-Dilemma

Neben den in Kapitel 3.5 erwähnten Hyperparametern und deren Optimierungsansätzen gibt es noch die Problematik des Over- beziehungsweise Underfittings.

Overfitting, auf deutsch Überanpassung, ist ein Problem, dass auftritt, wenn die Komplexität eines Modells aufgrund eines hohen Rauschens zu groß ist. Potenzielle Lösungsansätze könnten darin bestehen, das Modell zu simplifizieren, indem es durch ein Modell mit weniger Parametern ersetzt wird, wie beispielsweise ein lineares Modell anstelle eines polynomiellen Modells höherer Ordnung. Alternativ kann die Anzahl der Merkmale im Trainingsdatensatz reduziert, oder dem Modell können Einschränkungen auferlegt werden. Eine weitere Option wäre das Sammeln zusätzlicher Trainingsdaten oder die Reduzierung von Rauschen in den vorhandenen Trainingsdaten, zum Beispiel durch Beheben von Datenfehlern oder Entfernen von Ausreißern. In der Regularisierung wird das Modell eingeschränkt, um es zu vereinfachen und das Risiko von Overfitting zu reduzieren. In Abbildung 3.11 ist ein Beispiel für ein polynomes Modell dargestellt, das die Zufriedenheit über das Bruttoinlandsprodukt pro Kopf zeigt. Das Modell überfittet. Ein weiteres Beispiel ist das lineare Modell. Es hat die zwei Parameter θ_0 und θ_1. Durch diese beiden Freiheitsgrade kann das Modell an die Trainingsdaten angepasst werden, indem sowohl die Höhe (θ_0) als auch die Steigung (θ_1) der Geraden verändert werden. Wenn θ_1 auf den Wert Null festgelegt wird, hat das Modell nur noch einen Freiheitsgrad und es wird schwieriger, die Daten gut anzupassen. Die Gerade kann sich nur noch in der Höhe bewegen, um möglichst nah an den Trainingsdatenpunkten zu liegen, und landet daher in der Nähe des Mittelwerts. Das resultierende Modell ist sehr einfach. Wenn jedoch ein kleiner Wert für θ_1 erlaubt wird, hat das Modell zwischen einem und zwei Freiheitsgrade. Das entstehende Modell ist einfacher als das mit zwei Freiheitsgraden, aber komplexer als das mit nur einem Freiheitsgrad. Es ist wichtig, das optimale Gleichgewicht zwischen dem perfekten Fitting der Daten und einem möglichst einfachen Modell zu finden, das gut verallgemeinert. [15, S. 28 f] Dies wird im maschinellen Lernen auch als Bias-Varianz-Dilemma bezeichnet und ist einer der wichtigsten Axiome beim erstellen von Modellen.

Die Überanpassung eines Modells wird somit in der Regel durch dessen Komplexität verursacht. Eine effektive Methode, um Überanpassung zu vermeiden, besteht darin, die Komplexität des Modells zu kontrollieren, wofür die Regularisierung zuständig ist. Die Regularisierung reguliert die Modellkomplexität, indem sie höhere Terme im Modell bestraft. Wenn ein Regularisierungsterm dem Modell ergänzt wird, versucht das Modell, sowohl den Loss als auch die Modellkomplexität zu minimieren. Die L1- und die L2-Regularisierungen werden sehr häufig genutzt. Dabei reguliert L1 die Gesamtzahl der Merkmale und L2 die Gewichtung der Merkmale. Die L1-Regularisierung bewirkt, dass die Gewichte der informationslosen Merkmale auf Null gesetzt werden, indem bei jeder Iteration ein kleiner Betrag vom Gewicht subtrahiert wird, bis das Gewicht schließlich Null erreicht. Die L2-Regularisierung veranlasst die Reduzierung der Gewichte in Richtung Null, aber setzt sie nicht exakt auf Null. Die L2-Regularisierung wirkt wie eine Kraft, die bei jeder Iteration einen kleinen Prozentsatz der Gewichte entfernt. Aus diesem Grund werden die Gewichte niemals exakt auf Null gesetzt. [27]

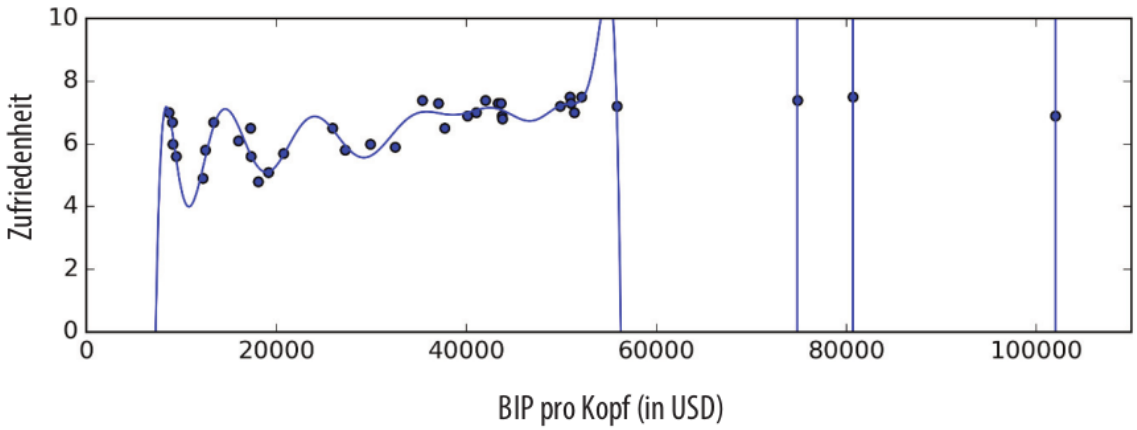


Abbildung 3.11: Ein Beispiel für ein überfittetes Modell [15, S. 28]

Ein Beispiel für ein anfälliges Modell für Underfitting ist ein lineares Modell der Zufriedenheit. Die Realität ist komplexer als das Modell, was bedeutet, dass Vorhersagen auf den Trainingsdaten an Genauigkeit verlieren. Um diese Problematik zu lösen, kann ein leistungsstärkeres Modell mit einer höheren Anzahl an Parametern verwendet werden. Die Bereitstellung von verbesserten Merkmalen, kann das Underfitting ebenfalls reduzieren. Zu dem können die Modellbeschränkungen reduziert werden, zum Beispiel in dem Hyperparameter zur Regulierung verringert werden. [15, S. 30]

3.7 Konfusionmatrix zur Evaluierung des Klassifizierers

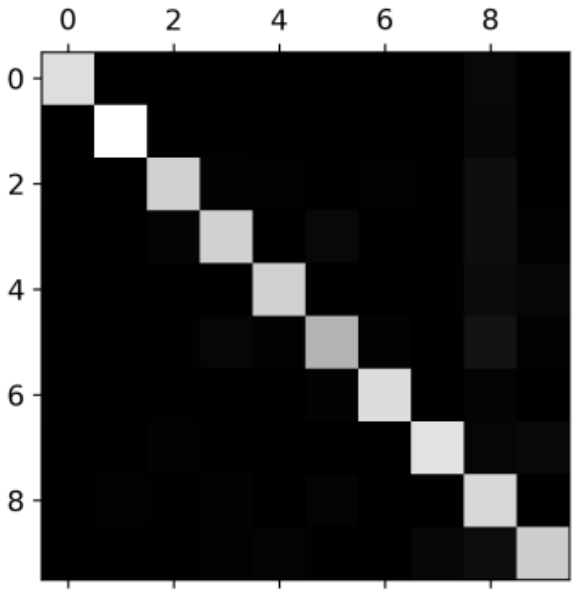


Abbildung 3.12: Beispiel für eine Konfusionmatrix [15, S. 107]

Eine Möglichkeit, das Training der Daten zu beschleunigen und zu verbessern, ist das Einsetzen von Optimierern. [15, S. 354] Der Adam-Optimierer kommt in dieser Arbeit zum Einsatz.

Die Konfusionsmatrix (siehe Abbildung 3.12) dient der anschaulichen Darstellung der vorhergesagten Ergebnisse und der Auswertung eines Klassifizierers. Diese Matrix gibt an, wie oft der Klassifikator Datenpunkte fälschlicherweise einer anderen Klasse zuordnet hat. Alle tatsächlichen Kategorien werden als Zeile und jede vorhergesagte Kategorie als Spalte dargestellt. Die Hauptdiagonale der Matrix gibt die Anzahl der korrekt klassifizierten Datenpunkte an. Als Relevanz (engl. Precision) des Klassifikators wird die Auskunft über die Genauigkeit des Klassifikators bezeichnet. Sie wird anhand folgender Formel berechnet:

$$\text{Relevanz} = \frac{RP}{RP + FP}. \tag{3.6}$$

Dabei wird RP als Anzahl richtig Positiver und FP als Anzahl falsch Negativer bezeichnet. Die Sensitivität (engl. Recall), welche auch Richtig-positiv-Rate genannt wird, ist die Menge positiver Datenpunkte, welche vom Klassifikator erfasst wurden. [15, S. 94 f]

$$\text{Sensitivität} = \frac{RP}{RP + FN}, \tag{3.7}$$

wobei FN der Anzahl falsch Negativer entspricht.

Der F_1-Score wird berechnet, indem der harmonische Mittelwert von Relevanz und Sensitivität bestimmt wird. Bei dem gewöhnlichen Mittelwert werden alle Werte als gleichwertig betrachtet. Im Gegensatz dazu werden bei dem harmonischen Mittelwert niedrigen Werten ein höheres Gewicht zugeordnet. Aus folgender Formel lässt sich ableiten, dass der F_1-Score hoch ist, wenn Relevanz und Sensitivität groß sind:

$$F_1 = \frac{2}{\frac{1}{\text{Relevanz}} + \frac{1}{\text{Sensitivität}}} = 2 \cdot \frac{\text{Relevanz} \cdot \text{Sensitivität}}{\text{Relevanz} + \text{Sensitivität}} = \frac{RP}{RP + \frac{FN+FP}{2}} \tag{3.8}$$

Der F_1-Score bevorzugt Klassifikatoren, bei denen Relevanz und Sensitivität ähnlich sind. Dies ist jedoch nicht zweckdienlich. In einigen Anwendungen ist die Relevanz wichtiger als die Sensitivität, während in anderen die Sensitivität von entscheidender Bedeutung ist. [15, S. 96]

Um sich einen schnellen Überblick über die Daten zu verschaffen, kann für jedes Merkmal ein Histogramm erstellt werden. Ein Histogramm kann für jedes Merkmal erstellt werden. [15, S. 51]

3.8 Wavelet Scattering Transformation

Scattering-Transformationen werden als Faltungsnetze implementiert, deren Filter nicht erlernt, sondern als Wavelet-Filter festgelegt werden. [28]

Die Einbindung von Kymatio (eine Bibliothek aus der Pythonumgebung) in Deep-Learning-Frameworks ermöglicht es, den Gradienten von Wavelet-Streukoeffizienten rückwärts laufen zu lassen und sie so in eine durchgängig trainierbare Leitung, beispielsweise ein tiefes neuronales Netz, zu integrieren. [28]

Die *WST* besteht aus einer Reihe von Wavelet-Transformationen und Modulus-Nichtlinearitäten. Um eine Wavelet-Streuungstransformation eines Eingangssignals x zu erzeugen, sind drei aufeinander folgende Operationen erforderlich: Faltung, Nichtlinearität und Mittelwertbildung, wie in Abbildung 3.13 beschrieben. Die WST-Koeffizienten werden durch Mittelung der Wavelet-Modulkoeffizienten mittels eines Tiefpassfilters Φ gewonnen. [29, S. 3]

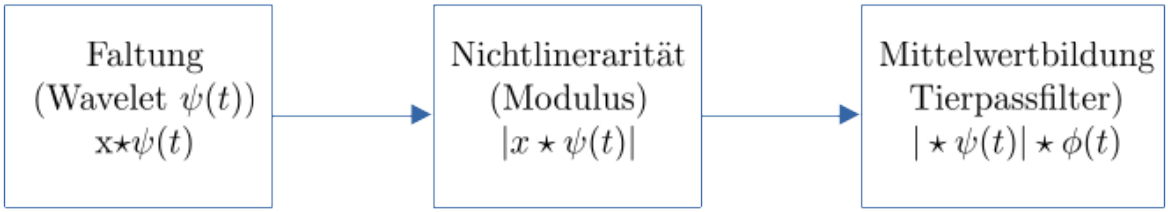


Abbildung 3.13: Prozess der Wavelet Scattering Transformation, in Anlehnung an [29, S. 3]

Angenommen, es existiert ein Bandpassfilter $\psi(t)$ mit einer normierten Zentralfrequenz von 1. Des Weiteren sei $\psi\lambda(t)$ eine Wavelet-Filterbank, die durch Dilatation des genannten Wavelets erzeugt wird. Dabei ist $\psi_\lambda(t)$ wie folgt definiert [29, S. 3]:

$$\psi_\lambda(t) = \lambda\psi(\lambda t), \tag{3.9}$$

wobei $\lambda = 2^{\frac{j}{Q}}, \forall j \in \mathbb{Z}$, und Q ist die Anzahl der Wavelets pro Oktave ist [29, S. 3]. Als Oktave wird das Intervall, bei dem die Frequenz des tieferen zu der des höheren Tons im Verhältnis 1:2 steht, bezeichnet [30].

Das Wavelet $\psi(t)$ weist eine Bandbreite von der Ordnung $\frac{1}{Q}$ auf, was dazu führt, dass die Filterbank aus einer Gruppe von Bandpassfiltern besteht, die im Frequenzbereich zentriert sind und eine Frequenzbandbreite von $\frac{\lambda}{Q}$ besitzen. Bei der nullten Ordnung ergibt sich ein einzelner Koeffizient durch $S_0x(t) = x \star \phi(t)$, wobei $\star$ den Faltungsoperator darstellt. [29, S. 3]

$$S_1x(t, \lambda_1) = |x \star \psi\lambda_1| \star \phi(t). \tag{3.10}$$

Die Koeffizienten der zweiten Ordnung sind in der Lage, die Amplitudenmodulationen im Hochfrequenzbereich zu erfassen, welche in jedem Frequenzband der ersten Schicht auftreten. Diese Koeffizienten können durch folgende Formel berechnet werden: [29, S. 4]

$$S_2x(t, \lambda_1, \lambda_2) = ||x \star \psi\lambda_1| \star \psi\lambda_2| \star \phi(t). \tag{3.11}$$

Die Frequenzauflösung der Wavelets $\psi\lambda_2$ kann sich von Q_1 unterscheiden. Die Darstellung des Signals erfolgt spärlich, was bedeutet, dass die Signalinformationen auf so wenige Wavelet-Koeffizienten wie möglich konzentriert werden. Diese Koeffizienten werden durch den Tiefpassfilter ψ gemittelt, welcher wie bei den Koeffizienten erster Ordnung eine lokale Invarianz gegenüber Zeitverschiebungen gewährleistet. [29, S. 4]

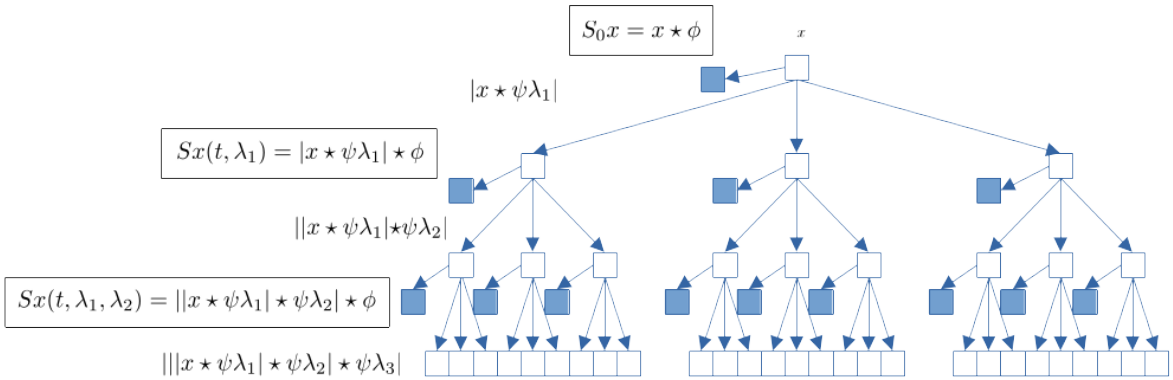


Abbildung 3.14: Hierarchische Darstellung von WSTW auf mehreren Schichten, in Anlehnung an [29, S. 4]

Die Hierarchie der Wavelet-Streuungskoeffizienten wird in Abbildung 3.14 dargestellt. Diese Struktur ähnelt tiefen neuronalen Netzen, aber im Gegensatz dazu liefert jede Schicht des Wavelet-Streuungsverfahrens einen Teil der Ausgabe, während bei den meisten tiefen neuronalen Netzen nur die letzte Schicht eine Ausgabe liefert. Die Wavelet-Streuungskoeffizienten erster und zweiter Ordnung werden auf das Signal im Zeitbereich angewendet. Die Merkmale der zweiten Ordnung werden durch die Merkmale der ersten Ordnung normalisiert, um sicherzustellen, dass die höhere Ordnung der Streuung von der Amplitudenmodulationskomponente des Sprachsignals abhängt. Die WST-Koeffizienten erster und zweiter Ordnung werden zu einem Streu-Merkmalsvektor für jeden Signalrahmen zusammengefügt. [29, S. 4]

Kapitel 4

Explorative Modellbildung

Das folgende Kapitel gibt einen Abriss zum maschinellen Lernen. Es liefert Aufschluss über die Frage, wie induktives Lernen vonstatten geht, wie Methoden des maschinellen Lernens eingeordnet werden und stellt *CNN* als Basis für den Klassifikator vor.

4.1 Datenvorbetrachtung

Aufgrund der Tatsache, dass die Daten bereits vorverarbeitet vorliegen, werden die Ergebnisse aus dem Paper [31] in diesem Kapitel zusammengefasst.

Der komplette Datensatz besteht aus circa 38.000 Waferkarten, wobei dieser 38 Fehlertypen beinhaltet. Im folgenden werden jedoch nur die Typen „Normal“ und acht einzelnen Fehlertypen betrachtet.

In den Listen 1 und 2 ist zu sehen, wie die Daten ausgelesen und ein Bild des Wafers, der mit „Normal“ gelabelt wurde. Die Ausgabe des Bildes ist in Abbildung 4.1(a) zu sehen.

Die Erstellung der Waferkarten erfolgte während der Testung der Wafer. Mithilfe zweier Sonden wurde jeder Stempel (Pixel) der Waferkarte abgetastet und an dieser Stelle die elektrischen Eigenschaften dokumentiert. Die Waferkarte ist eine Matrix m_i welche 3 Zahlenwerte besitzt. Die Variable x_{ij} in der Matrix beinhaltet das Ergebnis des Wafertests eines Stempels bei Zeile i und Spalte j. Der Wert 0 tritt an den Stellen x_{ij} auf, an denen keine Stempel (Pixel) vorhanden sind. Die Stellen x_{ij}, die fehlerfrei funktionieren, erhalten den Index 1. 2 bedeutet, dass an dieser Stelle ein Funktionsfehler vorliegt. Die Variable x_{ij} ist somit aus der Menge $\{0, 1, 2\}$ wählbar. Die Werte 0,1,2 entsprechen in den Bildern der Waferkarte jeweils einer Farbe. Die Wafermaps sind von den Erstellern des Datensatzes skaliert worden und haben eine Größe von $52 \times 52 Pixeln$. [31, S. 5]

Die Wafer des Datensatzes beinhalten Wafermaps ohne Fehler (Normal) und acht Fehlertypen. Die neun verschiedenen Wafermaps sind in Abbildung 4.1 beispielhaft dargestellt. Dabei sind die Fehlerpunkte gelb. Abbildung 4.1(a) zeigt die Karte ohne Fehler. Diese Art von Bild wird als „Normal“ bezeichnet. Der erste Fehlertyp wird mit

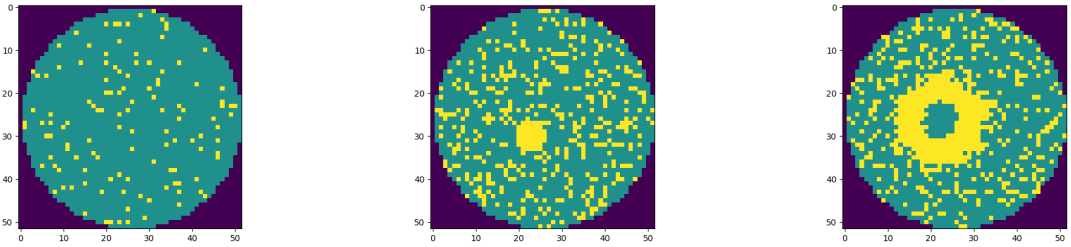
„Center “ betitelt, da in der Mitte der Waferkarte eine ein Defekt vorliegt. Der Fehler, welcher als „Donut“ bezeichnet wird, ist in Abbildung 4.1(c) zu sehen. Dieser Fehler hat die Form eines Rettungsrings oder Donuts und befindet sich ebenfalls in der Mitte der Wafermap. Der dritte Fehlertyp wird als „Edge_Loc“ bezeichnet und ist dadurch gekennzeichnet, dass es einen Defekt am Rand der Wafermap gibt. Der Fehler „Edge_Ring“ weist ebenfalls einen Defekt am Rand der Map auf. Jedoch umschließt der Fehler den kompletten Rand der Karte in Form eines Rings. Die Abbildung 4.1(f) zeigt den Fehlertyp „Loc“. Dies ist ein Fehler ohne erkennbare Form, welcher nicht am Rand oder in der Mitte der Waferkante auftritt. Der Fehlertyp „Near_Full“ beschreibt einen Defekt, welcher sich über die gesamte Fläche der Waferkarte verteilt, sodass die Fläche an defektfreiem Material geringer ist als die fehlerhafte Fläche. Die Abbildung 4.1(h) zeigt den Fehlertyp „Scratch“, welcher die Form eines Kratzers auf der Karte hat. Die letzte Art von Fehlern wird als „Random“ bezeichnet und ist dadurch charakterisiert, dass die Fehlerpunkte auf der Wafermap zufällig verteilt sind. [31, S. 3]

Aus [32] sind die Bezeichnungen der jeweiligen Defekte, das Label, die Anzahl und der Indexbereich zu entnehmen. Diese Informationen sind in Tabelle 4.1 aufgelistet.

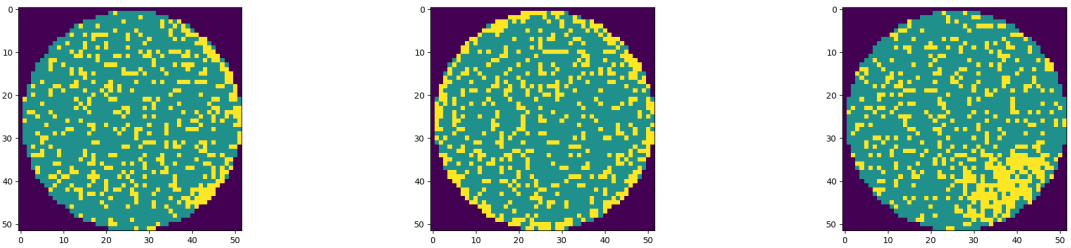
Name	Label	Anzahl	Indexbereich
Normal	[0 0 0 0 0 0 0 0]	1000	33866-34865
Center	[1 0 0 0 0 0 0 0]	1000	12000-12999
Donut	[0 1 0 0 0 0 0 0]	1000	24000-24999
Edge_Loc	[0 0 1 0 0 0 0 0]	1000	25000-25999
Edge_Ring	[0 0 0 1 0 0 0 0]	1000	26000-26999
Loc	[0 0 0 0 1 0 0 0]	1000	32000-32999
Near_Full	[0 0 0 0 0 1 0 0]	866	33000-33865
Scratch	[0 0 0 0 0 0 1 0]	1000	37015-38014
Random	[0 0 0 0 0 0 0 1]	149	34866-35014

Tabelle 4.1: acht Einzelwaferdefekttypen

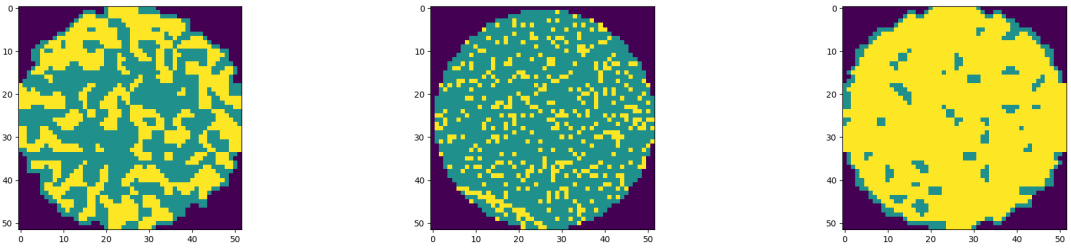
Bei der Betrachtung der Daten fällt auf, dass die Wafermaps der Defekte „Near_Full“ und „Random“ einen Maximalwert von $x_{ij,max} = 3$ haben, wohingegen die anderen Wafermaps einen Maximalwert von $x_{ij,max} = 2$ haben, wie in Abbildung Tabelle 4.2 zu sehen ist. Aus diesen maximalen Werten lässt sich die Anzahl der Farben der Bilder ableiten. Das heißt, ein Bild mit dem Maximalwert 2 hat drei Farben (Beginn Indexzählung bei 0) und ein Bild mit dem Maximalwert 3 besitzt 4 Farben. Im Paper wird, wie oben erwähnt, beschrieben, dass die Variable x_{ij} aus einer Menge mit drei Werten besteht. Existiert bei einer Waferkarte ein Maximalwert $x_{ij,max} = 3$, bedeutet das, dass eine Menge aus vier Werten (0,1,2,3) existiert. Die entsprechenden Wafermapbilder bestehen dann aus vier Farben.



(a) Normal (b) Center (c) Donut



(d) Edge_Loc (e) Edge_Ring (f) Loc



(g) Near_Full (h) Scratch i) Random

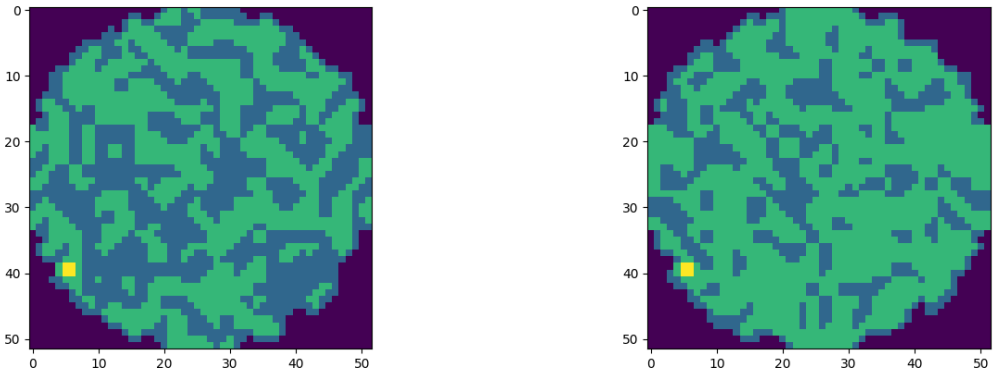
Abbildung 4.1: Beispielbilder aller Fehlertypen

Name	Maximalwerte
Normal	2
Center	2
Donut	2
Edge_Loc	2
Edge_Ring	2
Loc	2
Near_Full	3
Scratch	2
Random	3

Tabelle 4.2: Maximalwerte der Klasse

Diese Vierfarbbilder treten bei den Fehlertypen „Near_Full“, zu sehen in Abbildung 4.2(a), und „Random“, zu sehen in Abbildung 4.2(b), auf. Die beiden Abbildungen ha-

ben gemeinsam, dass es Vierfarbbilder sind und dass jeweils genau ein Pixel gelb ist. Die anderen Fehlerpunkte, die bei den dreifarbigen Bildern gelb sind, sind in diesem beiden Bildern grün. Es ist anzunehmen, dass es bei der Aufnahme der Bilder zu Fehlern kam. Die Anzahl der vierfarbigen Waferkarten vom Typ „NearFull“ beträgt 97 und die von „Random“ 8.



(a) 4-Farbbild-„Near_Full“-Index:33003 (b) 4-Farbbild-„Random“-Index:34867

Abbildung 4.2: Beispielbilder für vierfarbige Bilder

Da dies zu veränderten Ergebnissen führen kann, wird ein neuer Datensatz ohne die vierfarbigen Bilder erstellt.

Zusammenfassend lässt sich Folgendes daraus schließen.

Der Datensatz, der in dieser Arbeit verwendet wird, enthält Bilder von verschiedenen Waferdefekten. Fehler müssen in verschiedene Klassen sortiert und gelabelt werden. Es herrscht ein Multiklassenproblem vor, bei dem die Fehler der Wafer klassifiziert werden. Aufgrund der Einteilung der Arten des maschinellen Lernens in Kapitel 3.1.1 lässt sich daraus schließen, dass überwachtes Lernen hierfür am besten geeignet ist. Ein *CNN* und eine *WST* werden auf die Daten angewendet.

4.2 CNN

Wie im vorherigen Kapitel erwähnt, existiert für diesen Datensatz ein Multiklassenproblem. Aus diesem Grund wird als Aktivierungsfunktion eine Softmaxfunktion benötigt. Matrizen in Form von Bildern dienen als Eingabe. Als Netz bietet sich ein *CNN* an.

Als erstes wird der Grundaufbau eines *CNN* verwendet, damit ein Netz existiert, das trainiert werden kann. Die Daten werden in Train-, Test- und Validierungsdaten eingeteilt. In Listing 5 im Anhang ist der entsprechende Code dafür zu sehen.

In Listing im Anhang 4 ist der typische Code eines *CNN* zu sehen, welcher zum Antrainieren des Modells genutzt wurde. Das Schema des Modell ist in Abbildung 4.3 zu sehen

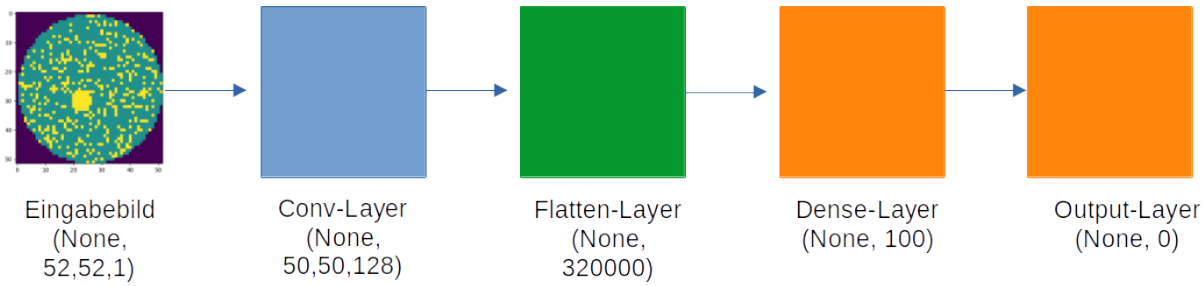


Abbildung 4.3: Schema des CNNs

Dieses Modell besteht aus allen Schichten, die nacheinander definiert wurden.

Im Anschluss folgt das Einfügen einer Eingabeschicht in der Form (52,52,1). Diese Eingabeschicht besteht aus den Waferbildern, welche aus 52×52 Pixeln besteht. Die 1 steht dafür, dass der Farbkanal eine Größe von 1 hat. Dies wird bei Graustufenbildern verwendet. [15, S. 300]

Es wird ein Vektor mit neun verschiedenen Wahrscheinlichkeiten erzeugt. Die Zahl 9 steht für die verschiedenen Fehlertypen, die detektiert werden sollen.

Mithilfe einer implementierten Datenaugmentation werden vorhandene Bilder variiert, und dadurch werden neue Bilder durch verschiedene Operationen, wie Drehen oder Spiegeln, erzeugt. Ziel dieser Codezeile ist die verbesserte Ausführung des Modells. [14, S. 142 f]

Als nächstes wird eine Convolutional-Schicht (in Abbildung 4.3 hellblau) eingeführt. Dabei werden die Dimensionsanzahl, die Filtergröße und die Aktivierungsfunktion als Parameter festgelegt.

Im Anschluss folgt ein Flatten-Layer (in Abbildung 4.3 grün). Dieser formt die vorherige faltende Schicht in einen einen eindimensionalen Array um. [15, S.300]

Im Folgenden wird eine Dense-Schicht (in Abbildung 4.3 orange) hinzugefügt. Diese besteht aus zwölf Neuronen und der Aktivierungsfunktion Softmax. [15, S. 301]

Die Ausgabe ergibt sich wie folgt: Es wird ein weiterer Dense-Layer (in Abbildung 4.3 orange) eingefügt, welcher neun Ausgänge hat. Auch hier ist eine Softmax-Funktion als Aktivierungsfunktion festgelegt worden. Aus diesen beiden Festlegungen lässt sich schließen, dass die neun Outputs als Wahrscheinlichkeiten dargestellt werden. Des Weiteren wird zur Vermeidung des Overfittings eine L2-Regulierungs eingeführt.

Zum Schluss erfolgt die Definition des Optimierers. Dabei wird das Modell mit dem Adam-Optimierer der Lossfunktion und der Metrik erfasst.

4.2.1 Der Prototyp

In Tabelle 4.3 im Anhang werden die Paramater definiert, die zum Antrainieren des Modells genutzt werden können. Die Lernrate *lr* beträgt 10^{-4}. Die Parameter des Adam-Operators werden ebenfalls festgelegt. Der Convolutional-Layer hat eine Schichtanzahl von 128 und zwei Filter mit einer Größe von 3. Als Aktivierungsfunktion wird wieder die *ReLU*-Funktion verwendet. Die Anzahl der Neuronen in der Dense-Schicht beträgt 100. Für den Regularisierungsterm ist der Standardwert 0,0 angegeben worden. Mit der 'batch_size' wird angegeben, dass 64 Trainingsbeispiele in einem Schritt bearbeitet werden, bevor es zur Aktualisierung der Parameter des Modells kommt.

Hyperparameter	Output
'lr'	1e-4
'beta1'	8.121504e-01
'beta2'	9.610024e-01
'conv_filter_num'	128
'conv_filter_size1'	3
'conv_filter_size2'	3
'conv_acti'	'relu'
'dense1_units'	100
'dense1_activation'	'relu'
'l2_reg'	0.0
'batch_size'	64

Tabelle 4.3: Anfängliche Hyperparameter

In Tabelle 4.4 ist der Aufbau des Modells zu sehen, welcher in dem Listing 4 und Tabelle 4.3 festgelegt wurde. Es wird die Schichttyp (Layer (type)), die Form der Ausgabe (Output Shape) und die Parameteranzahl (Param #) dargestellt.

Die Sequenzial-Schicht, bestehend aus einer Abfolge von Sequenzen, ist durch den in Listing 4 beschriebenen Befehl *data_augmentation* definiert worden.

Die faltende Schicht, die als nächstes folgt, besitzt 128 Filter, die eine Filtergröße von 3×3 aufweisen. Aus der Form dieser Schicht (None, 50, 50, 128) lässt sich schließen, dass alle Eingabebilder auf eine Größe von von 50×50 Pixel abstrahiert werden. Es werden 128 Merkmale entnommen.

In der Flatten-Schicht wird die faltende Schicht auf die Form (None, 3200000) geändert. Die Merkmale werden in einen Vektor umgewandelt.

Es folgt ein Dense-Layer mit 100 Neuronen. Dabei besitzen alle Neuronen eine festgelegte Gewichtung aus der Flatten-Schicht.

Zuletzt wird wieder eine Dense-Schicht beschrieben, welche neun Neuronen beinhaltet. Diese Neuronen dienen der Berechnung der neun Klassen.

Die Tabelle 4.4 gibt zusätzlich alle trainierbaren und alle nicht trainierbaren Parameter an. Alle Parameter sind trainierbar.

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 52, 52, 1)	0
conv2d (Conv2D)	(None, 50, 50, 128)	1280
flatten (Flatten)	(None, 320000)	0
dense (Dense)	(None, 100)	32000100
dense_1 (Dense)	(None, 9)	909
Total params: 32,002,289		
Trainable params: 32,002,289		
Non-trainable params: 0		

Tabelle 4.4: Zusammenfassung Basis Modell

Im Anschluss erfolgt das Aufteilen des Datensatzes in Traings-, Test- und Validierungsdaten. Der entsprechende Code ist im Anhang List 5 zu sehen. Es folgt das Trainieren, indem das Modell für 10 Epochen trainiert wird. Währenddessen wird die Batch-Größe, die in *config_start* festgelegt wurde, genutzt. Im Anschluss werden die Validierungsdaten verwendet, um zu testen, wie die Leistung des Modells ist. In diesem Codeblock wird eine Early-Stopping-Methode angewendet. Dieser wird durch das *tf.keras.callbacks.EarlyStopping*-Objekt mit den Parametern *monitor='loss', patience= 20* und min_delta=1e-5 bestimmt. Daraus lässt sich ableiten, dass das Training unterbrochen wird, wenn der Loss auf den Validierungsdaten über 20 Epochen gleich bleibt. Der Befehl *min_deleta* legt den minimalen Unterschied im Loss fest, welcher als Verbesserung angesehen wird. Zum Schluss werden die Ergebnisse des Training in der Variable *value* gespeichert.

4.2.2 Praktische Umsetzung der Hyperparameteroptimierung

Für die Hyperparameteroptimierung wird die Funktion *fit_model* definiert. In Listing 6 im Anhang ist der entsprechende Code für diese Definition zu sehen. Diese Funktion führt das Training mit den Trainingsdaten und wahlweise mit den Validierungsdaten des übergebenen Modells aus. Es wird die Batch-Größe beschrieben. Die Early-Stopping-Methode wird auch hier verwendet. Das Training wird wieder unterbrochen, wenn der Loss über 20 Episoden gleich bleibt. Im Fall, dass keine Validierungsdaten übergeben werden (*(val_data is None* oder *val_data[0] is None*, erfolgt das Training nur mit Trainingsdaten. Sonst erfolgt das Testen mit den Validierungsdaten, damit die Leistung des Modells auf den unbekannten Daten eingeschätzt werden kann.

Die Trainingsergebnisse werden in der Variable *value* gesichtet und wiedergegeben. In dieser Variable ist der Verlauf des Trainings für die verschiedenen Epochen enthal-

ten. Weil der Parameter *verbose=0* ist, wird der Trainingsfortschritt im Terminal nicht angezeigt. Für andere Argumente gibt es die Möglichkeit, die Funktion an *fit_model* zu übergeben.

Die Optimierung der Hyperparameter wird im Codeabschnitt Listing 6 durchgeführt. Dabei wird der Suchraum beschrieben, welcher für die Hyperparameteroptimierng verwendet wird. Die Hyperparameter werden als Mode-Objekte konkretisiert. Jedes Objekt dient der Beschreibung der Eigenschaften der Hyperparameter, wie die Lernrate, Anzahl der Filter oder die Aktivierungsfunktion.

Der Code wird in Listing 8 im Anhang dargestellt und zeigt, dass die Durchführung mit dem Agenten-Objekt (*agent*) erfolgt . Dieser beinhaltet als Eingabe die Trainingsdaten, Validierungsdaten, die Fit-Funktion und das vorher definierte *CNN* als Parameter, damit der optimierte Algorithmus und die Zielmetrik angegeben werden können.

Der Parameterraum (*parameter_space*) ist eine mit Parametern beschriebene Liste, die den potentiellen Bereich und Optimierungsalgorithmus zur Algorithmusoptimierung aufführt.

Der Startpunkt zur Optimierung des Algorithmus ist *config*. Im Code wird der beste Konfigurationspunkt aus den vergangenen Trainingsdurchläufen zur Optimierung genommen. Zum Antrainieren des Modells, wird zunächst die Codezeile *config = config_start* verwendet. Danach wird der Code erneut mit der besten Konfiguration durchgeführt.

Mit dem Befehl *iterations* wird die Anzahl an Durchläufen festgelegt, die das System durchlaufen soll. In diesem Fall sind es drei Durchläufe.

Der Code *n_trainings* gibt die Anzahl an Trainingsdurchläufen an. Hier wird ein Trainingsdurchlauf während der Hyperparameteroptimierung durchgeführt.

Der Befehl *target* gibt die Metrik an, die optimiert werden soll, in diesem Fall die Validierungslossfunktion.

Mit *verbose=1* wird der Verfahrensfortschritt angegeben.

Zusammenfassend kann gesagt werden, dass der gesamte Codeblock ein automatisches Hyperparameter-Tuning durchführt. Es werden unterschiedliche Parameterkombinationen getestet, damit die Modellleistung auf den Validierungsdaten verbessert wird.

Nach dem ersten Durchlauf der Hyperparameteroptimierung wird das Objekt *user_data* ausgegeben. Diese Ausgabe ist in Tabelle 4.5 zu sehen. Es existiert ein Modell, welches durch die optimalste Optimierung der Hyperparameter-Konfiguration erstellt wurde. Des Weiteren sind aus der Tabelle folgende Parameter herauszulesen: der finale Trainingsverlust des Modells, der Validierungsloss, die besten Hyperparameter-Konfigurationen, die Anzahl der Trainingsdurchgänge und die Gesamtzeit der Hyperparameter-Optimierung.

Hyperparameter	Output
'scaler'	None
'loss'	0.0808454304933548
'val_loss '	0.14046894013881683
'best_config '	
'lr'	0.0009416291838060672
'beta1'	0.36420644243785105
'beta2'	0.9607096116178175
'l2_reg'	10.192293256353413
'conv_filter_num '	128
'conv_filter_size1 '	3
'conv_filter_size2 '	5
'conv_acti'	'relu'
'dense1_units'	100
'dense1_activation '	'elu'
'batch_size '	64
'fit_call_count'	83
'time_elapsed'	datetime.timedelta(seconds=37684)

Tabelle 4.5: Ausgabe angepasster Hyperparameter

Mit der Codezeile *from sklearn.metrics import balanced_accuracy_score* wird die ausgewogene Genauigkeit eines Klassifikators, welcher auf dem Datensatz, dessen Klassen ungleich verteilt sind, berechnet. Es ist die durchschnittliche Genauigkeit pro Klasse. Anhand dieses Wertes lässt sich die Wahrscheinlichkeit, dass der Klassifikator die Klassen im Verhältnis zu ihrer Verteilung im Datensatz vorhersagt, bestimmen.

Der Code *y_pred = np.argmax(user_data['model'].predict(X_train), axis=-1)* verwendet das Modell (*user_data['model']*), um Vorhersagen aus den Eingabedaten abzuleiten. Damit der Index des höchsten Wertes jeder Ausgabe gefunden wird, wird die Funktion *np.argmax()* verwendet. Die *axis=-1*-Argument gibt an, dass die Suche nach dem höchsten Wert entlang der letzten Achse (in diesem Fall die Klassen) durchgeführt werden soll.

Die Funktion *balanced_accuracy_score(y_train, y_pred)* dient der Berechnung der ausgewogenen Genauigkeit zwischen dem vorhergesagtem Layer *y_pred* und dem tatsächlichen Layer *y_train*.

Die Codezeile *user_data['model'].evaluate(X_test,y_test)* führt die Auswertung des Netzes durch. Es wird der Loss und die Genauigkeit des Modells zurückgegeben. Das Ergebnis des Trainings wird in den Kapiteln 5 und 6 erläutert.

In den nächsten Codeblocks wird eine Konfusionsmatrix erstellt und geplottet. Die Darstellung und die Auswertung wird ebenfalls in den Kapiteln 5 und 6 erläutert.

Im Anschluss wird mit Hilfe des Befehls *print(sklearn.metrics.classification_report(y_test, basis_pred))* eine Zusammenfassung der Ergebnisse des Multiklassenproblems ausgegeben. Die Tabelle ist im Kapitel 5 dargestellt, wo auch die Ergebnisse ausgewertet werden.

Anschließend wird ein Histogramm erstellt, das die Eigenschaften der wahren Vorhersagen, welche aus den Testdaten erstellt wurden beinhaltet.

4.3 Wavelet Scattering Transformation Model

Anhand der Funktion *build_wstCNN* wird ebenfalls ein neuronales Netz für die Klassifikation von Bilddaten erstellt. Das Netz, dessen Code in der Liste 13 im Anhang zu sehen ist, nutzt die Scattering-Transformation für das Extrahieren der Eigenschaften. Die Abbildung 4.4 zeigt das Schema eines *WST*s. Der Scattering2D-Schicht (lila) folgen ein Flatten-Layer (grün), ein Dense-Layer (orange) und eine Ausgabeschicht (orange) mit neun Neuronen. Durch die Konfigurationsoperationen können die Anzahl der Einheiten in der dichten Schicht, die Aktivierungsfunktion, die L2-Regularisierung und die Lernrate des Optimierers angepasst werden. Das Netz wird mit dem Adam-Optimierer und der Verlustfunktion *sparse_categorical_crossentropy* kompiliert und mit der Metrik *sparse_categorical_accuracy* ausgewertet.

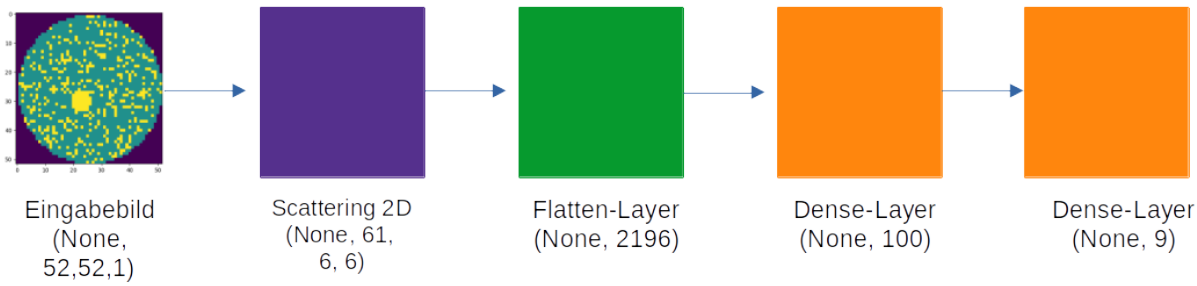


Abbildung 4.4: Schema des WSTs

Die Variable *basic_model_wst* wird anhand der Funktion *build_wstCNN* gebildet. Als Eingabe wird die Konfiguration *config* verwendet. Das Modell besteht aus verschiedenen Schichten. Die Eingabeschicht hat die Form (52,52). Das ist die Dimension der Eingabebilder. Auf diese Eingabedaten wird die WST2D-Transformation erstellt. Anschließend erfolgt das Flatten der Transformation. Danach wird eine Dense-Schicht mit Neuronen und der Aktivierungsfunktion hinzugefügt. Zum Schluss folgt die Ausgabeschicht mit der Softmax-Aktivierungsfunktion und die L2-Regularisierung. Durch die Zuweisung der Variable *config_start* an die Funktion build_wstCNN wird das Modell *basic_model_wst* entsprechend der vorgegebenen Konfiguration erstellt.

Die Tabelle 4.6 Hat den gleichen Aufbau wie die Tabelle 4.4. Es werden die vier verwendeten Schichten Scattering2D-Layer, Flatten-Layer, und zwei Dense-Layern in der Übersicht betrachtet. Die Scattering2D-Schicht hat die Ausgabeform (None, 61, 6, 6). Dabei steht „None“ für eine veränderbare Batch-Größe. Der Flatten-Layer formt die Ausgabe der Scattering2D-Schicht in einen Vektor der Länge 2196 um. Die erste Dense-Schicht beinhaltet 100 Neuronen und die *ReLU*-Funktion. Die zweite Dense-Schicht beinhaltet neun Neuronen und die Softmax-Funktion als Aktivierungsfunktion. Diese Schicht gibt die Klassenvorhersage des Modells aus. Alle 220609 Parameter, die das Modell besitzt, sind trainierbar.

Layer (type)	Output Shape	Param #
scattering2d_12 (Scattering 2D)	(None, 61, 6, 6)	0
flatten_341 (Flatten)	(None, 2196)	0
dense_682 (Dense)	(None, 100)	219700
dense_683 (Dense)	(None, 9)	909
Total params: 220,609		
Trainable params: 220,609		
Non-trainable params: 0		

Tabelle 4.6: Zusammenfassung Basis Modell WST

Mit der Codezeile *basic_model_wst.evaluate(X_train, y_train)* wird das trainierte Modell aus Trainingsdaten ausgewertet. Die Trainingsdaten werden als Eingabe *X_train* übergeben und die entsprechenden Labels als *y_train*. Die Ausgabe gibt die berechneten Metriken als Liste zurück, wobei die erste Zahl den Loss und die zweite Zahl die Genauigkeit darstellt.

Es folgen die Codeblocks für die Ausgabe der Genauigkeiten und Losses für die Validierungs- und Testdaten.

Zum Schluss wird wieder der Klassifikationsbericht der Vorhersagen des Modells in Form einer Tabelle ausgegeben. Diese Tabelle ist Kapitel 5 abgebildet und die Ergebnisse werden dort diskutiert.

Kapitel 5

Ergebnisse

5.1 Das CNN

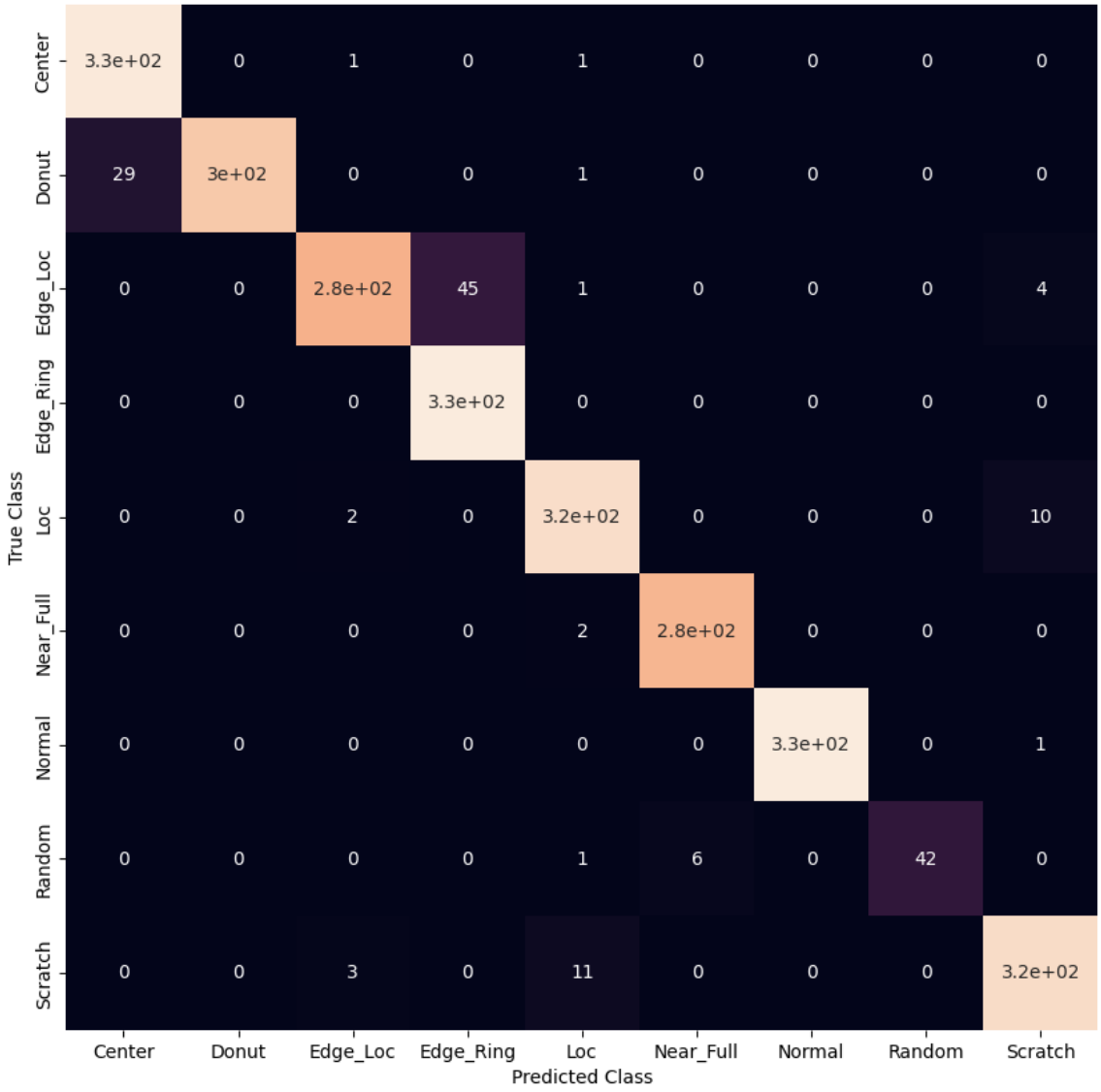


Abbildung 5.1: Konfusionsmatrix zur Analyse der richtigen und falschen Klassifizierungen

Die Abbildung 5.1 zeigt die Konfusionsmatrix des *CNN*s. Aus dieser Matrix lässt sich die Anzahl der richtigen und falschen Klassifizierungen der Klassen auslesen.

Listing 12 gibt folgendes aus, dass der Losswert etwa 0.207 und die Genauigkeit der Trainingsdaten des Netzes etwa 95,5% beträgt.

Die Tabelle 5.1 beinhaltet den Zusammenhang der Ergebnisse des Multiklassenproblems. Jede Zeile steht dabei für eine der insgesamt neun Klassen (Zählbeginn bei Null). Jede Spalte gibt Informationen zu den einzelnen Metriken an. Es fällt auf, dass die Präzision bei Klasse 4 („Edge_Loc“) unter 0,88 beträgt, alle anderen Klassen habe mindestens eine Präzision von 0,92. Der Recallwert ist bei Klasse 2 („Donut“) mit 0,85 am geringsten. Der F_1-Score beträgt bei allen Klassen mindestens 0,91. Die Spalte Support gibt die Anzahl der Beispiele pro Klasse an. Auffällig ist, dass bei Klasse 7 („Scratch“) sehr wenig Beispiele vorhanden sind. Die Zeile accuracy gibt die Genauigkeit, also die Anzahl der korrekt vorhergesagten Beispiele geteilt durch die Gesamtheit der Beispiele, an. Dieser Bericht ist in Abbildung 5.2 in Form eines Balkendiagrammes grafisch dargestellt. In diesem Fall liegt eine Genauigkeit von 0,96 vor. Für den Fall, dass das Modell verbessert werden soll, ist es sinnvoll sich auf eine Klasse zu konzentrieren, die häufig vertreten ist und darauf den größten Fehler anzuwenden. In diesem Fall ist es sinnvoll sich die Klasse 2 genauer anzuschauen.

	precision	recall	f1-score	support
0	0.92	0.99	0.95	330
1	1.00	0.91	0.95	330
2	0.98	0.85	0.91	330
3	0.88	1.00	0.94	330
4	0.95	0.96	0.96	330
5	0.98	0.99	0.99	286
6	1.00	1.00	1.00	330
7	1.00	0.86	0.92	49
8	0.95	0.96	0.96	330
accuracy			0.96	2645
macro avg	0.96	0.95	0.95	2645
weighted avg	0.96	0.96	0.96	2645

Tabelle 5.1: Klassifikationsbericht CNN

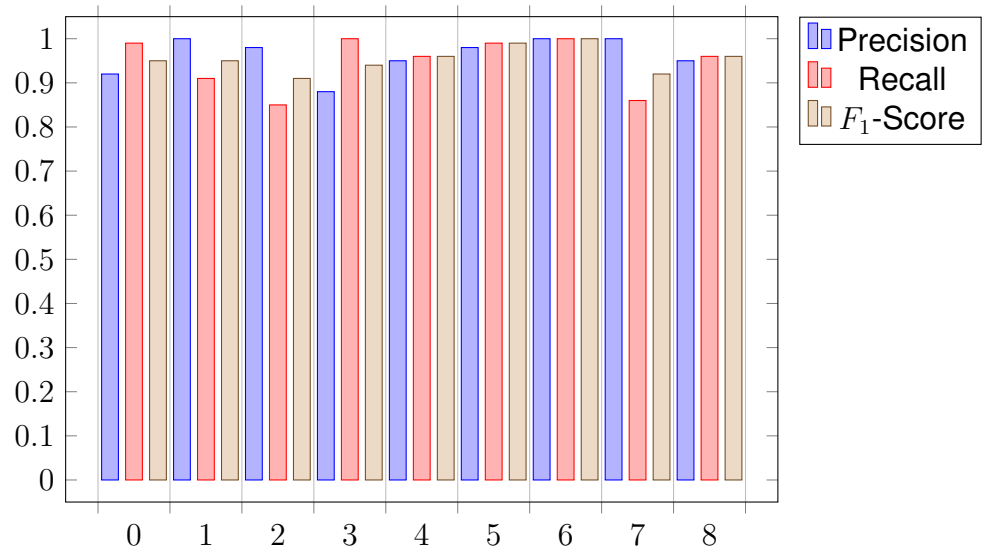


Abbildung 5.2: Grafische Darstellung des Klassifikationsberichtes des *CNN*

Das Histogramm in Abbildung 5.3 beinhaltet die Eigenschaften von wahren Vorhersagen. Diese wurden aus den Testdaten des Modells erstellt. Es sind 20 Balken, die auf der horizontalen Achse gleichmäßig verteilt sind.

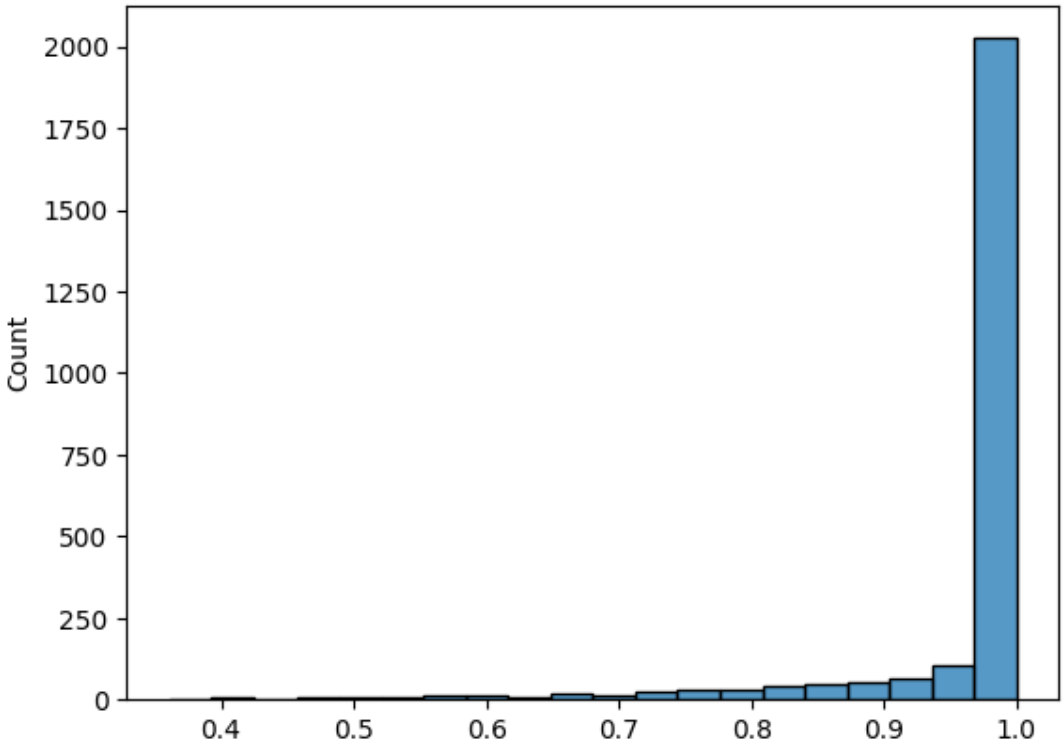


Abbildung 5.3: Histogramm der Eigenschaften von wahren Vorhersagen, die aus den Testdaten des Modells generiert wurden

5.2 WST

Die Listings 14 bis 16 geben folgendes aus:

	Loss	Genauigkeit
Trainingsdaten	0.11843835562467575	0.9684151411056519
Validierungsdaten	0.10665207356214523	0.9667297005653381
Testdaten	0.10665207356214523	0.9667297005653381

Tabelle 5.2: Loss und Genauigkeiten des WSTst CNN

Dabei ist wieder, wie in Tabelle 5.2 zu sehen, der linke Wert der Losswert und der rechte die Genauigkeit der Trainings-, Validierungs- und Testdaten des Netzes.

Die Tabelle 5.3 beinhaltet einen Klassifikationsbericht der Vorhersagen des *WST*-Modells. Die Präzision bei Klasse 3 ist auch bei diesem Modell geringer, als bei den anderen Klassen. Das gleiche trifft auf den Recall mit Klasse 2 zu. Ebenfalls wie in der Tabelle 5.1 hat der F_1-Score bei allen Klassen mindestens einen Wert von 0,91. Es ist ebenfalls auffällig, dass die Anzahl der Instanzen jeder Klasse bei Klasse 7 wieder deutlich geringer ist, als bei den anderen Klassen. Dieser Bericht ist in Abbildung 5.2 ebenfalls in Form eines Balkendiagrammes grafisch dargestellt. Hier herrscht eine Genauigkeit von 0,97 vor.

	precision	recall	f1-score	support
0	0.99	1.00	1.00	330
1	1.00	0.99	1.00	330
2	0.95	0.83	0.88	330
3	0.87	0.96	0.91	330
4	0.98	0.98	0.98	330
5	0.99	0.99	0.99	286
6	1.00	1.00	1.00	330
7	0.96	0.96	0.96	49
8	0.96	0.99	0.98	330
accuracy			0.97	2645
macro avg	0.97	0.97	0.97	2645
weighted avg	0.97	0.97	0.97	2645

Tabelle 5.3: Klassifikationsbericht WST

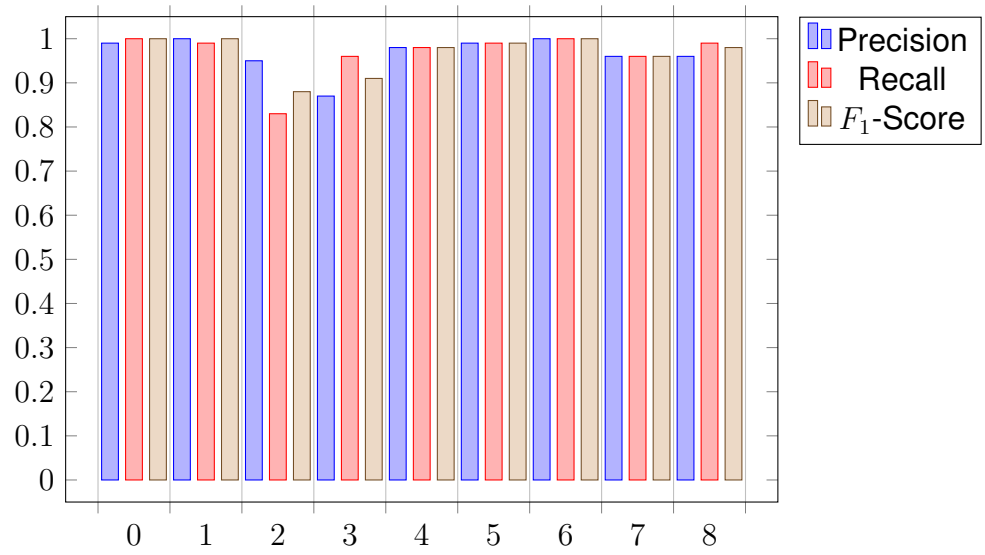


Abbildung 5.4: Grafische Darstellung des Klassifikationsberichtes des *WST*

Kapitel 6

Auswertung und Diskussion

Das Covolutional neural network

Die Konfusionmatrix ist in Abbildung 5.1 zu sehen. Die vertikale Achse zeigt die tatsächlichen Klassen an, während die horizontale Achse die Vorhersagen des Algorithmus angibt. Die Hauptdiagonalen von links oben nach rechts unten repräsentieren die korrekten Vorhersagen des Algorithmus, während die diagonalen Linien von rechts oben nach links unten die falschen Vorhersagen anzeigen. Die Anzahl der Vorhersagen jeder Klasse kann auf der rechten Seite der Matrix angezeigt werden. Es ist zu sehen, dass die meisten Bilder auf der Hauptdiagonalen liegen. Daraus lässt sich ableiten, dass diese Bilder korrekt zugeordnet wurden. Die Klasse „Random“ ist auf der Hauptdiagonale dunkler dargestellt als die anderen Klassen. Auch die Anzahl der richtig zugeordneten Bilder beträgt lediglich 42. Demnach wurden nur wenige Bilder dieser Klasse korrekt zugeordnet. Die niedrige Anzahl ist daher begründet, dass in der Klasse „Random“ im Vergleich zu den anderen Klassen sehr wenig Bilddaten vorhanden waren.

Es konnte eine Genauigkeit der Trainingsdaten des *CNN* von circa 95,5% erreicht werden.

Die Gesamtgenauigkeit des *CNN* beträgt 96%.

Das Histogramm zeigt die Klassenverteilungen in dem *CNN*, das für Vorhersagen eingesetzt wird. Die x-Achse des Histogramms ist skaliert durch die Eigenschaften der wahren Vorhersagen. Es wird gezeigt, dass das Modell mit einer Wahrscheinlichkeit von über 90% 2000 Testdaten richtig zuordnet.

Wavelet-Scattering-Transformation-Modell

Es konnte eine Genauigkeit der Trainings-, Validierungs- und Testdaten des *WST* von jeweils circa 96,8% erreicht werden.

Die Gesamtgenauigkeit des *WST* beträgt 97%.

Vergleich der Genauigkeiten

Die im Paper [31] beschriebene, erreichte Gesamtgenauigkeit beträgt 96,8%. Somit wurde mit dem *CNN* die gleiche Genauigkeit und mit der *WST* eine höhere Genauigkeit erlangt.

Kapitel 7

Zusammenfassung und Ausblick

Zusammenfassung

An einem Waferdefektdatensatz des Papers [31] ist maschinelles Lernen angewendet worden. Nach der Erläuterung von Wafern und deren Defektentstehung folgte ein Einblick in das maschinelle Lernen und deren Arten. Das *CNN* als Modell des überwachten Lernens ist dabei genauer betrachtet worden. Es wurde der Aufbau des *CNN* und der des *WST* zum besseren Verständnis erläutert.

Im Anschluss wurde der Datensatz analysiert, die verschiedenen Waferdefekte beschrieben und eine Datenexploration durchgeführt. Dabei wurde festgestellt, dass die Fehlertypen „Near_Full“ und „Random“ auch vierfarbige Bilder enthalten. Die anderen Wafermaps sind dreifarbig. Zur Durchführung des maschinellen Lernens ist der gesamte Datensatz verwendet worden, um die Ergebnisse mit denen des Papers vergleichen zu können. Beim *CNN* ist eine Genauigkeit von 96% und beim *WST* von 97% erreicht worden. Mit dem *WST* wurde eine höhere durchschnittliche Genauigkeit erreicht, als die, die im Paper angegeben wurde.

Ausblick

Während des Bearbeitens dieser wissenschaftlichen Arbeit ist ein Datensatz ohne vierfarbige Fehler erstellt worden, der einen Bias-Fehler hatte und deshalb für die Anwendung auf das *CNN* und das *WST* nicht genutzt werden konnte. Der Bias-Fehler kann in den Trainingsdaten auftreten, wenn diese nicht ausreichend vielfältig sind oder wenn ungleiche Gewichtungen zwischen verschiedenen Merkmalen bestehen. Das kann dazu führen, dass das Modell nicht ausgewogen und repräsentativ lernt und in der Anwendung ungenaue oder diskriminierende Entscheidungen trifft. Eine Durchführung des maschinellen Lernens mit so einem Datensatz sollte in Zukunft untersucht werden, um zu beurteilen, ob die vierfarbigen Bilder Einfluss auf den Klassifikator haben.

In Zukunft könnten Fehlertypen, die eine große Ähnlichkeit besitzen, vermieden werden, damit der Klassifikator eine höhere Genauigkeit erreichen kann.

Für ein effektiveres Training ist es von Vorteil, wenn alle Fehlertypen die gleiche Anzahl an Datenpunkten besitzen, damit das Modell die einzelnen Klassen gleichmäßig trainieren kann. Die Verzerrung, die bei einer ungleichen Anzahl an Datenpunkten pro Klasse auftritt, zeigt sich in der Konfusionsmatrix und könnte dazu führen, dass in diesem Fall diese Klassen falsch zugeordnet werden.

Der gesamte Datensatz könnte ebenfalls auf ein *Circular Convolutional Neural Network* angewendet werden, das auf runde Bilder spezialisiert ist. Diese haben zum Beispiel den Vorteil, dass sie eine geringere Anzahl von Parametern benötigen als *CNN*s und dass die auf Bilder verschiedener Größe anwendbar sind.

Literaturverzeichnis

[1] Optosurf GmbH. Wafermaschine wafermaster, . URL https://www.optosurf.de/produkte/wafermaster/. Zugriffsdatum 19.03.2023.

[2] confovis GmbH. Confovis: 100. messsystem ausgeliefert. URL https://www.confovis.com/unternehmen/aktuelles/100-messsystem-ausgeliefert/. Zugriffsdatum 19.03.2023.

[3] sofeast. Golden sample: Why you need it before mass production starts. URL https://www.sofeast.com/glossary/golden-sample/. Zugriffsdatum 19.03.2023.

[4] Sabine Globisch et al. *Lehrbuch Mikrotechnologie*. Hanser Buchverlag, 2011.

[5] Philipp Laube. Halbleitertechnologie von a bis z, . URL https://www.halbleiter.org/lexikon/W/Wafer/. Zugriffsdatum 17.03.2023.

[6] Fraunhofer. Silizium wafer. URL https://www.isit.fraunhofer.de/de/mikro-fertigungsverfahren/module-services/testwafer-substrate/silizium-wafer.html. Zugriffsdatum 28.02.2023.

[7] Frederic Adler Norbert Schwesinger, Carolin Dehne. *Lehrbuch Mikrosystemtechnik*. Oldenbourg Buchverlag, 2009.

[8] Philipp Laube. Waferherstellung: Herstellung von rohsilicium, . URL https://www.halbleiter.org/waferherstellung/rohsilicium/. Zugriffsdatum 17.03.2023.

[9] Philipp Laube. Waferherstellung: Herstellung des einkristalls, . URL https://www.halbleiter.org/waferherstellung/einkristall/. Zugriffsdatum 17.03.2023.

[10] Ingolf Ruge. *Der ideale Einkristall*. Springer Berlin Heidelberg, 1984. URL https://doi.org/10.1007/978-3-642-96782-5_2.

[11] Rudolf Gross. Physik der kondensierten materie 1. URL https://www.wmi.badw.de/fileadmin/WMI/Lecturenotes/Condensed_Matter_Physics/2-KM-Expert-1_VS_05Nov2020.pdf. Zugriffsdatum 17.03.2023.

[12] Philipp Laube. Waferherstellung: Der wafer, . URL https://www.halbleiter.org/waferherstellung/wafer/. Zugriffsdatum 17.03.2023.

[13] Felix Heß. *Generierung von Defekten auf der Waferkante mittels Nanosekunden-Laserablation für die Analyse von Messsystemen*. dissertation, Universität Paderborn, 2012.

[14] Andriy Burkov. *Machine Learning kompakt - alles, was Sie wissen müssen*. mitp, 2019.

[15] Aurélien Géron. *Praxiseinstieg Machine Learning - mit Scikit-Learn, Keras und TensorFlow: Konzepte, Tools und Techniken für intelligente Systeme*. O'reilly, 2020.

[16] Ethan Alpaydin. *Maschinelles Lernen*. De Gruyter, 2021.

[17] Mike Espig. Vorlesung data science sose 2021, SoSe 2021.

[18] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012.

[19] Daniel A. Keim. Datenvisualisierung und data mining. URL `https://www.dbis.prakinf.tu-ilmenau.de/papers/dbspektrum/dbs-02-30.pdf`. Zugriffsdatum 17.03.2023.

[20] Anke Stoll. Supervised machine learning mit nutzergenerierten inhalten: Oversampling für nicht balancierte trainingsdaten. 2020.

[21] Mark H. Fischer Titus Neupert Kenny Choo, Eliska Greplova. *Machine Learning kompakt - Ein Einstieg für Studierende der Naturwissenschaften*. Springer Spektrum, 2020.

[22] meltwater. Bilderkennung: Was ist image recognition und wie funktioniert sie? URL `https://www.meltwater.com/de/blog/bilderkennung-image-recognition`. Zugriffsdatum 12.03.2023.

[23] Julia Fischer. Neuronale netze. URL `https://user.phil.hhu.de/~petersen/SoSe17_Teamprojekt/AR/neuronalenetze.html#flatten-und-dense`. Zugriffsdatum 21.03.2023.

[24] Sabyasachi Sahoo. Deciding optimal kernel size for cnn. URL `https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363`. Zugriffsdatum 13.03.2023.

[25] Rachel Gardner an others. Convolutional neural networks (cnns / convnets). URL `https://cs231n.github.io/convolutional-networks/`. Zugriffsdatum 14.03.2023.

[26] Mike Espig. Vorlesung data science ws 202, WS 2020.

[27] Soner Yıldırım. L1 and l2 regularization — explained. URL https://towardsdatascience.com/l1-and-l2-regularization-explained-874c3b03f668. Zugriffsdatum 21.03.2023.

[28] Kymatio. Kymatio: Wavelet scattering in python - v0.3.0 “erdre”. URL https://www.kymat.io/. Zugriffsdatum 15.03.2023.

[29] Olivier Lézora Wajdi Ghezaiel, Luc Brun. Wavelet scattering transform and cnn for closed set speaker identification. 10 2020.

[30] PREFORM GmbH. Oktaven und terzen, . URL https://www.preform.de/oktaven-und-terzen/. Zugriffsdatum 16.03.2023.

[31] Junliang Wang et al. Deformable convolutional networks for efficient mixed-type wafer defect pattern recognition. 11 2020.

[32] Junliang Wang et al. Mixed-type wafer map defect dataset. 2020.

Anhang

Convolutional neuronal Network

```
# when you opened it in the git folder
data_0 = np.load('/datasets/natural_science/mixed_wm38/arr_0.npy',
    mmap_mode=None, allow_pickle=False, fix_imports=True, encoding='ASCII
    ')
type(data_0), np.shape(data_0), data_0.dtype
```

Listing 1: Einlesen der Daten

```
# Exapmle Iluustration of Pattern named "Normal"
# Amount: 1000
# Index: 33866-34865
# Label: [0 0 0 0 0 0 0 0]
#fig = plt.figure(figsize=(52,52))
plt.imshow(data_0[33866,:,:], interpolation='nearest')
plt.show()
```

Listing 2: Ausgabe des Pattern „Normal“

```
# see if pictures have same max-value
for klasse in list(np.unique(y)):
print(klasse," hat den MaxWert:", wafers[np.where(y==klasse)[0]].max())
```

Listing 3: Bestimmung der Maximalwerte der verschiedenen Klassen

```
def build_CNN(config):

        model = Sequential()
        model.add(Input(shape=(52, 52, 1)))

        output_size    = 9

        model.add(data_augmentation)

        model.add(Conv2D(config['conv_filter_num'],
                (config['conv_filter_size1'],
                config['conv_filter_size2']),
                activation=config['conv_acti']))
```

```
        #model.add(Conv2D(64, (3, 3), activation='relu'))

        #model.add(Conv2D(64, (3, 3), activation='relu'))


        # Flatten Layer
        model.add(Flatten())

        model.add(Dense(config['dense1_units'],
                config['dense1_activation']))

        # Output layer
        model.add(Dense(output_size, activation='softmax',
              kernel_regularizer= l2(config['l2_reg'])) )

        optimizer =tf.keras.optimizers.Adam(
                  learning_rate=config['lr'],
                  beta_1=config['beta1'],
                            beta_2=config['beta2'],
                                        )


        model.compile(optimizer=optimizer,
                loss='sparse_categorical_crossentropy',
                metrics=["sparse_categorical_accuracy"])

        return model
```

Listing 4: Aufbau des CNN

```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='loss', patience
   =20, min_delta=1e-5)
value = basic_model.fit(X_train, y_train, validation_data=(X_val, y_val)
   , callbacks=early_stop, epochs=10, verbose=1, batch_size=config_start
   ['batch_size'])
```

Listing 5: Code für die Einteilung des Datensatzes in Trainings-, Test- und Validierungsdaten

```
def fit_model(model, config, train_data, val_data, **kwargs):
    early_stop = tf.keras.callbacks.EarlyStopping(monitor='loss',
          patience=20, min_delta=1e-5)
    if val_data is None or val_data[0] is None:
                value = model.fit(train_data[0], train_data[1],
                callbacks=early_stop, epochs=1000, verbose=0,
                batch_size=config['batch_size'])
```

```
    else:
            value = model.fit(train_data[0], train_data[1],
            validation_data=val_data, callbacks=early_stop,
            epochs=1000, verbose=0, batch_size=config['batch_size'])
    return value
```

Listing 6: Code für Definition der Funktion *fit_model*

```
# greate biva agent
agent = Agent(X_train, y_train,
              validation_data=(X_val, y_val),
              preprocessing=None,
              fit_func=fit_model,
              build_model=build_CNN,
              random_seed=None)

# setup bayessearch
init_points = 10
n_trials    = 3

param_space = [
    Mode('lr',              values = [1e-6, 1e-2],
                mode_optimizer='bayessearch',
                init_points=init_points,
                n_trials=n_trials,
                target='loss'),
    Mode('beta1',           values = [0.1, 0.999] ,
                mode_optimizer='bayessearch',
                init_points=init_points,
                n_trials=n_trials,
                target='loss'),
    Mode('beta2',           values = [0.9, 0.999] ,
                mode_optimizer='bayessearch',
                init_points=init_points,
                n_trials=n_trials,
                target='loss'),
    Mode('l2_reg',            values = [0, 100] ,
                mode_optimizer='bayessearch',
                init_points=init_points,
                n_trials=n_trials),
    Mode('conv_filter_num',   values = [5, 10, 20, 30, 50, 100,
                128, 150],
                mode_optimizer="gridsearch"),
    Mode('conv_filter_size1', values = [3, 5, 7],
                mode_optimizer="gridsearch"),
    Mode('conv_filter_size2', values = [3, 5, 7],
                mode_optimizer="gridsearch"),
```

```
    Mode('conv_acti',      values = ['relu','tanh','elu',
                  'sigmoid','linear'],
   mode_optimizer='gridsearch'),
    Mode('dense1_units',       values = [20, 30, 50, 60, 80,
                  100, 120],
                  mode_optimizer="gridsearch"),
    Mode('dense1_activation', values = ['relu','tanh','elu',
                  'sigmoid','linear'],
   mode_optimizer='gridsearch'),
          ]
```

Listing 7: Bevorzugter Biva-Agent

```
# HP-tuning model
user_data = agent.scan(optimizer='acs',
                       parameter_space = param_space,
                       #order           = order,
                       #config          = config_start,
                       config          = user_data['best_config'],
                       iterations      = 3,
                       n_trainings     = 1,
                       target          = 'val_loss',
                       verbose         = 1,
                      )
```

Listing 8: Hyperparameter-Tuning Model

```
from sklearn.metrics import balanced_accuracy_score
```

Listing 9: Funktion für die Berechnung der ausgewogenen Genauigkeit

```
y_pred = np.argmax(user_data['model'].predict(X_train), axis=-1)
```

Listing 10: Funktion für die Erstellung von Vorhersagen aus denn Eingabedaten

```
balanced_accuracy_score(y_train, y_pred)
```

Listing 11: Funktion für die Berechnung der ausgewogenen Genauigkeit zwischen den vorhergesagten und den tatsächlichen Labeln

```
user_data['model'].evaluate(X_test,y_test)
```

Listing 12: Funktion für Auswertung des Testdaten des neuronalen Netzes

Wavelet Scattering Transformation Modell

```
def build_wstCNN(config):

        model = Sequential()
        model.add(Input(shape=(52, 52)))

        output_size    = 9

        #model.add(data_augmentation)

        model.add(Scattering2D(J=3, L=4))


        # Flatten Layer
        model.add(Flatten())

        model.add(Dense(config['dense1_units'],
        config['dense1_activation']))

        # Output layer
        model.add(Dense(output_size, activation='softmax',
        kernel_regularizer= l2(config['l2_reg'])) )

        optimizer = tf.keras.optimizers.Adam(learning_rate=config['lr'],
         beta_1=config['beta1'],
         beta_2=config['beta2'],
                        )


        model.compile(optimizer=optimizer,
        loss='sparse_categorical_crossentropy',
        metrics=["sparse_categorical_accuracy"])

        return modeldef build_wstCNN(config):

        model = Sequential()
        model.add(Input(shape=(52, 52)))

        output_size    = 9

        #model.add(data_augmentation)

        model.add(Scattering2D(J=3, L=4))
```

```
        # Flatten Layer
        model.add(Flatten())

        model.add(Dense(config['dense1_units'],
        config['dense1_activation']))

        # Output layer
        model.add(Dense(output_size, activation='softmax',
        kernel_regularizer= l2(config['l2_reg'])) )

        optimizer = tf.keras.optimizers.Adam(learning_rate=config['lr'],
        beta_1=config['beta1'],
        beta_2=config['beta2'],
                        )


        model.compile(optimizer=optimizer,
        loss='sparse_categorical_crossentropy',
        metrics=["sparse_categorical_accuracy"])

        return model
```

Listing 13: Ausfbau wst

```
basic_model_wst.evaluate(X_train, y_train)
```

Listing 14: Funktion für Auswertung der Trainingsdaten des Wavelet Scattering Modells

```
basic_model_wst.evaluate(X_val, y_val)
```

Listing 15: Funktion für Auswertung deder Validierungsdaten des Wavelet Scattering Modells

```
user_data['model'].evaluate(X_test,y_test)
```

Listing 16: Funktion für Auswertung der Testdaten des Wavelet Scattering Modells