Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Sebastian Böhle

3D-Objekterkennung mit Faltungsnetzwerken

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sebastian Böhle

3D-Objekterkennung mit Faltungsnetzwerken

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 24. Mai 2016

Sebastian Böhle

Thema der Arbeit
3D-Objekterkennung mit Faltungsnetzwerken

Stichworte
Faltungsnetzwerke, Support Vector Machine, 3D-Objekterkennung, HSI-Farbraum, Geometrische Merkmale, Caffe, OUR-CVFH, RGB-D

Kurzzusammenfassung
Für die Integration von Robotern in den Haushalt stellt die Objekterkennung eine zwingende Anforderung dar. In dieser Thesis wird ein neuartiger Ansatz vorgestellt, der die Objekterkennung mithilfe von Tiefeninformationen verbessert. Die Realisierung erfolgt durch die Kombination von zwei unterschiedlichen Verfahren zur Extraktion von Merkmalen eines Objekts. Zusätzlich wird eine Support Vector Machine zur Klassifizierung des Objekts eingesetzt. Aus den Tiefeninformationen werden Merkmale in Form von geometrischen Eigenschaften extrahiert. Die Merkmalsextraktion aus den Farbinformationen erfolgt mit einem Faltungsnetzwerk. Eine Gegenüberstellung mit vergleichbaren Arbeiten sowie eine Evaluierung des Konzepts werden durchgeführt und die Ergebnisse diskutiert.

Sebastian Böhle

Title of the paper
3D object recognition with convolutional neural networks

Keywords
Convolutional Neural Networks, Support Vector Machine, 3D object recognition, HSI color space, geometrical features, Caffe, OUR-CVFH, RGB-D

Abstract
Object recognition is a mandatory requirement for the integration of robots in our households. This thesis presents a novel approach that uses depth information to improve current object recognition methods. The concept uses a combination of two different techniques to extract features and classify the object with a support vector machine. The depth information is used to extract geometrical characteristics of the object. In addition, the extraction of the features from the color information takes place with a convolutional neural network. A comparison is made with similar works and an evaluation of the concept is discussed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einleitung

Roboter halten immer weiteren Einzug in unsere Gesellschaft. Sie werden schon seit längerer Zeit in der Industrie für Montagearbeiten eingesetzt und sind dort auch nicht mehr wegzudenken. Aber nicht nur an dieser Stelle, sondern auch im privaten Bereich schreitet die Integration von Robotern immer weiter voran. Laut Tsardoulias u. a. (2014) werden Serviceroboter den Menschen innerhalb der nächsten Jahre immer mehr bei der Bewältigung des Alltags unterstützen. Damit ein Serviceroboter in der Lage ist einen Menschen zu unterstützen, benötigt er verschiedene Fähigkeiten.

Um Roboter in den Haushalt integrieren zu können, stellt die Objekterkennung eine essenzielle Fähigkeit dar. Hierbei handelt es sich um eins der anspruchsvollsten Probleme im Bereich des Maschinensehens (vgl. Socher u. a. (2012)). Ein Serviceroboter benötigt diese Fähigkeit zum Beispiel, um dem Menschen einen Gegenstand überbringen zu können. Für die Durchführung dieser Aufgabe ist es notwendig, dass der Roboter den Gegenstand beziehungsweise das Objekt identifizieren kann.

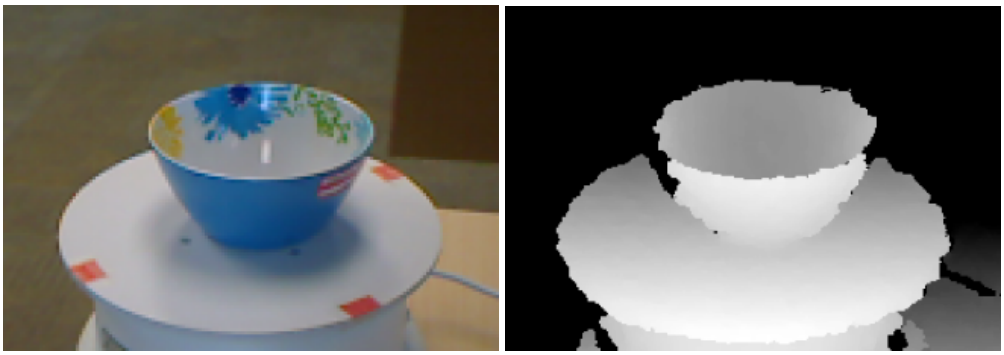
Eine detailliertere Beschreibung der Grundlagen und Herausforderungen der Objekterkennung befindet sich in Abschnitt 2.1. Die Objekterkennung stellt ein elementares Problem dar, das eine möglichst präzise sowie performante Lösung benötigt, um den Anforderungen aus der Robotik gerecht zu werden.

1.1. Motivation

Die Verbreitung von Kameras, die zusätzlich zum RGB-Bild auch Tiefeninformationen liefern, ist in den letzten Jahren stark angestiegen. Ein Beispiel für eine solche Kamera ist die Kinect One von Microsoft. Daten, die eine solche Kamera zur Verfügung stellt, sind eine Kombination aus Farbbild und Tiefeninformationen. Abbildung 1.1 zeigt ein solches Datum, das auch als RGB-D-Bild[1] bezeichnet wird. Neben anderen Anwendungsgebieten wird diese Art von Kamera vor allem im Bereich des Maschinensehens (*engl.* Robot Vision) eingesetzt, um zum Beispiel einem

[1] RGB-D-Daten stellen neben einem RGB-Bild auch die dazu passenden Tiefeninformationen (**D**epth) bereit.

Roboter eine bessere Wahrnehmung seiner Umgebung und der Objekte in ihr zu ermöglichen (vgl. Alexandre (2014)).



(a) (b)

Abbildung 1.1.: RGB-Bild (a) einer Tasse mit dem dazugehörigen Tiefenbild (b) aus dem Washington RGB-D-Datensatz (Lai u. a., 2011).

Damit ein Roboter ein Objekt identifizieren kann, müssen von dem Objekt zunächst Merkmale extrahiert werden. Bei diesen Merkmalen (*engl.* features) handelt es sich um Eigenschaften, die das Objekt charakterisieren und somit eine Wiedererkennung von diesem ermöglichen. Als Beispiel für Merkmale können Ecken und Kanten genannt werden. Traditionell werden für die Merkmalsextraktion RGB-Bilder verwendet, aber auch die Extraktion von Merkmalen aus RGB-D-Bildern ist möglich. Durch die Einbeziehung der Tiefeninformationen bei der Merkmalsextraktion können Objekte noch detaillierter beschrieben und somit leichter zusätzliche Merkmale zur Identifikation extrahieren werden.

Für das Extrahieren von Merkmalen in RGB-Bildern stellen Faltungsnetzwerke (*engl.* Convolutional Neural Networks (CNN)) die State-of-the-Art-Lösung dar. Aktuelle Arbeiten in diesem Bereich haben gezeigt, dass die Objekterkennung verbessert werden kann, wenn neben einem Faltungsnetzwerk für RGB-Daten noch ein weiteres Faltungsnetzwerk für Tiefeninformationen verwendet wird (vgl. Schmidhuber (2014), Gupta u. a. (2014), Socher u. a. (2012), Schwarz u. a. (2015)).

Die extrahierten Merkmale müssen für die Bestimmung des Objekts klassifiziert werden. In diesem Zusammenhang wurde des Weiteren bewiesen, dass eine Klassifizierung von Merkmalen aus einem Faltungsnetzwerk mithilfe einer Support Vector Machine zu einer Verbesserung gegenüber der ansonsten in diesem Bereich verwendeten Klassifikatoren führt (vgl. Huang und LeCun (2006), Tang (2013)).

Zusammen mit der größeren Verfügbarkeit von RGB-D-Kameras werden auch RGB-D-Datenmengen der breiten Masse zugänglich gemacht. Mithilfe von diesen Testdaten wird es zunehmend einfacher zusätzlich zum RGB-Bild auch die Tiefeninformationen mit einzubeziehen und so die Objekterkennung noch weiter zu verbessern.

1.2. Zielsetzung

Das Ziel dieser Thesis ist die Entwicklung eines neuartigen Ansatzes zur 3D-Objekterkennung für den Bereich Robotik. Um dieses Ziel zu erreichen, wird zunächst ein Konzept entwickelt, dass aus der Kombination eines Faltungsnetzwerks, einem Verfahren zur Ermittlung der Objektgeometrie sowie einer Support Vector Machine besteht. Abbildung 1.2 stellt die schematische Visualisierung dieses Konzepts dar.

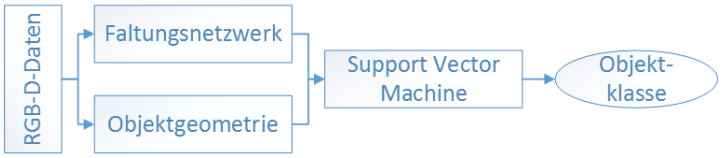


Abbildung 1.2.: Visualisierung des Konzepts, dass als Ziel erreicht werden soll.

Wie dort dargestellt werden als Eingangsdaten sogenannte RGB-D-Daten[1] verwendet. Diese setzen sich aus Farb- sowie Tiefeninformationen zusammen. Aus den Tiefeninformationen werden zunächst Merkmale in Form von geometrischen Eigenschaften extrahiert. Des Weiteren wird das Faltungsnetzwerk verwendet, um aus den Farbinformationen Merkmale zu extrahieren. Im Zuge dessen wird weiterhin evaluiert, ob der HSI-Farbraum einen Vorteil bei der Objekterkennung in unterschiedlichen Lichtverhältnissen im Gegensatz zum RGB-Farbraum bietet.

Die zwei gewonnenen Merkmalsarten aus der Merkmalsextraktion von Farb- und Tiefeninformationen werden anschließend von der Support Vector Machine für eine Klassifizierung verwendet. Das Ergebnis der Klassifizierung ergibt als Resultat die Objektklasse. Ein weiteres Ziel dieser Thesis ist die Ermittlung des Mehrwerts bei der Hinzunahme einer zweiten Merkmalsart am Beispiel der hier genutzten Tiefeninformationen. Weiterhin soll die Umsetzung des entwickelten Konzepts evaluiert werden.

1.3. Gliederung

Die vorliegende Arbeit ist in sieben Kapitel gegliedert. Die Inhalte der einzelnen Kapitel sind im Folgenden zusammengefasst.

Kapitel 1 gibt eine Einführung in das Thema und beschreibt die Motivation sowie die Zielsetzung dieser Arbeit.

Kapitel 2 stellt für das notwendige Verständnis dieser Arbeit die Grundlagen der Objekterkennung, von 3D-Daten sowie von Farbräumen bereit.

Kapitel 3 beschreibt die Grundlagen zum Verständnis von künstlichen neuronalen Netzen, Faltungsnetzwerken und Support Vector Machinen.

Kapitel 4 erläutert den Entwurf des Konzepts, der dazugehörigen Architektur sowie einzelner Komponenten von dieser.

Kapitel 5 schildert die Realisierung des entwickelten Konzepts. Des Weiteren werden die verwendeten Softwarekomponenten vorgestellt und Details zu Training und Test erläutert.

Kapitel 6 bewertet die Ergebnisse, die bei der Durchführung verschiedener Tests erzielt wurden. Zusätzlich werden die ermittelten Hyperparameter beschrieben und die Resultate einer Laufzeitmessung dargestellt.

Kapitel 7 fasst die Resultate dieser Arbeit zusammen und zeigt in einem Ausblick das Potenzial zur Weiterentwicklung.

2. Grundlagen

Im Folgenden werden die theoretischen Grundlagen beschrieben, auf denen diese Arbeit basiert. Zunächst wird die in diesem Zusammenhang verwendete Objekterkennung definiert. Im Zuge dessen werden auch zwei aktuelle Ansätze der 3D-Objekterkennung erläutert. Anschließend wird der Unterschied zwischen dem weiterverbreiteten RGB-Farbmodell und dem in dieser Arbeit verwendeten HSI-Farbmodell erläutert. Abschließend folgt eine Definition von 3D-Daten sowie eine Beschreibung der in dieser Arbeit verwendeten Datenquellen.

2.1. Objekterkennung

Für die Erkennung von Objekten in Bildern sind Verfahren notwendig, die in der Lage sind Merkmale von diesen zu extrahieren. Die Merkmale dienen dazu, dass Objekt charakteristisch zu beschreiben. Hierdurch wird eine Wiedererkennung ermöglicht. Abbildung 2.1 zeigt die wesentlichen Schritte, die notwendig sind, um eine Objekterkennung durchzuführen.

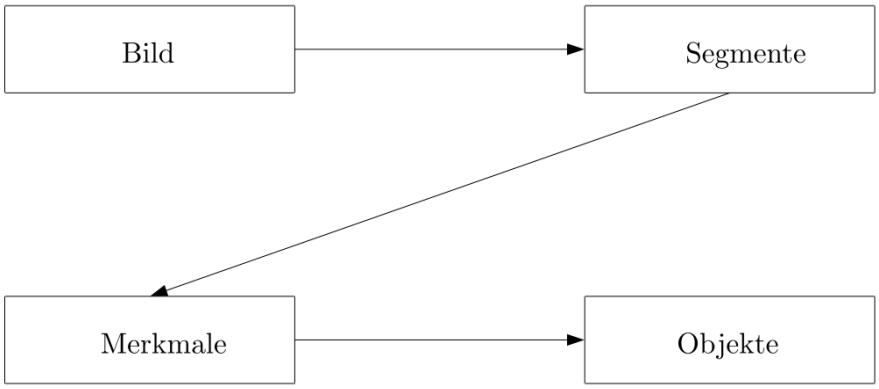


Abbildung 2.1.: Wesentliche Schritte der Objekterkennung (Steinmueller, 2008).

Zunächst wird das Bild in Segmente aufgeteilt, für die dann die Merkmale berechnet werden. Bei einer Aufnahme, die mehrere Objekte enthält, könnte ein Segment zum Beispiel aus dem

Bereich des Bildes bestehen, in dem sich ein einzelnes Objekt befindet. Mithilfe der Merkmale ist es anschließend möglich, das dazu passende Objekt zu identifizieren. Laut Steinmueller (2008) sind für eine Objekterkennung Merkmale erforderlich, die als Kriterium eine Invarianz der folgenden Eigenschaften erfüllen sollten:

- Verschiebung
- Drehung
- Skalierung
- Betrachterperspektive
- Beleuchtung
- Teilweise Verdeckung

Diese Anforderungen können allerdings nur näherungsweise erfüllt werden. Aus diesem Grund versuchen Algorithmen zur Merkmalsextraktion, die dadurch entstehenden Nachteile auszugleichen.

Verfahren

In den vergangenen Jahren wurden bereits viele Verfahren zur Erkennung von Objekten entwickelt. Der Großteil dieser Verfahren nutzt für die Merkmalsextraktion die RGB-Werte des Bildes. Durch die zunehmende Verbreitung von Tiefenbildkameras (siehe Abschnitt 2.3.1) stehen neben dem RGB-Bild allerdings auch Tiefeninformationen zur Verfügung. Aktuelle Forschungsarbeiten beschäftigen sich aus diesem Grund mit der Einbeziehung dieser 3D-Daten bei der Objekterkennung, um bessere Ergebnisse zu erzielen.

Im Folgenden werden zwei State-of-the-Art-Ansätze beschrieben, die sowohl Farb- als auch Tiefeninformationen für eine Objekterkennung einsetzten und Anregung für diese Thesis waren. Beide Arbeiten verwenden für die Umsetzung jeweils Faltungsnetzwerke (siehe Abschnitt 3.2) und Support Vector Machinen (siehe Abschnitt 3.3). Ein größerer Überblick, mit weiteren Verfahren sowie der Geschichte der Objekterkennung ist in Andreopoulos und Tsotsos (2013) zu finden.

Der erste Ansatz stammt von Gupta u. a. (2014). Die Autoren verwenden in dieser Arbeit als Grundlage das Faltungsnetzwerk „R-CNN“, das als State-of-the-Art-Lösung für 2D-Objekterkennung gilt (vgl. Girshick u. a. (2013)). Dieses Faltungsnetzwerk wurde ursprünglich von Krizhevsky u. a. (2012) entwickelt und von Girshick u. a. (2013) erweitert. Wie in Abbildung 2.2 dargestellt, besteht die entwickelte Architektur aus insgesamt zwei Faltungsnetzwerken. Neben einem unveränderten R-CNN für die RGB-Daten wird für die Verarbeitung der Tiefeninformationen ein zweites R-CNN ausgewählt. Als Eingangsdaten der Architektur werden

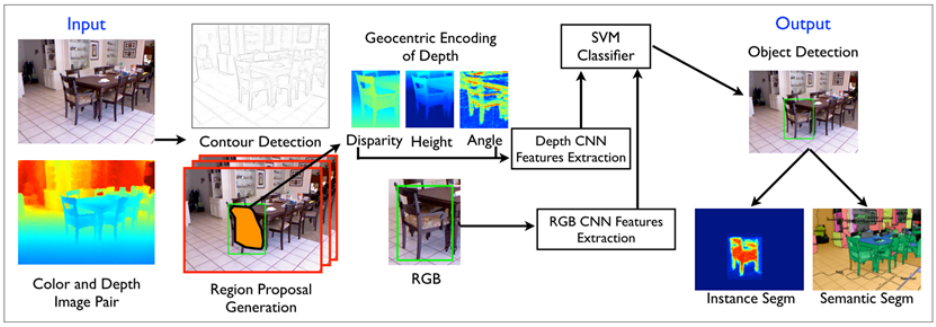


Abbildung 2.2.: Architektur der 3D-Objekterkennung von Gupta *u. a.* Gupta u. a. (2014).

das RGB-Bild sowie eine Aufnahme der gleichen Szene mit eingefärbten Tiefeninformationen verwendet.

Im ersten Schritt wird versucht alle Regionen im Bild zu identifizieren, die für eine Objekterkennung infrage kommen. Für die Durchführung dieses Schrittes wird das Bild zunächst so gefiltert, dass nur noch die Kontur (Ecken und Kanten) vorhanden ist. Mithilfe von dieser und dem Multiscale Combinatorial Grouping (MCG) Framework von Arbeláez u. a. (2014) werden anschließend Vorschläge für Regionen mit möglichen Objekten ermittelt. Diese vorgeschlagenen Bildregionen werden im Folgenden verwendetet, um mithilfe der zwei Faltungsnetzwerke signifikante Merkmale zur Identifikation zu extrahieren. Das unveränderte R-CNN nutzt hierfür die RGB-Bildinformationen. Bei dem um Tiefeninformationen erweiterten Faltungsnetzwerk wird die von den Autoren als „Geocentric Encodig of Depth“ betitelte Codierung verwendet. Dabei wird die Höhe des Objektes in Bezug zum Boden, sein Winkel in Bezug auf die Schwerkraft sowie die Disparität der Pixel mit einbezogen. Im Anschluss an die Merkmalsextraktion von beiden Faltungsnetzen werden diese zusammen mit einer Support Vector Machine für die Klassifizierung genutzt. Nach Beendigung dieses Schrittes liegt die Klasse vor zu der das Objekt gehört.

Der zweite Ansatz stammt von Schwarz u. a. (2015) und adaptiert ebenfalls das Faltungsnetzwerk R-CNN. Dieser Datensatz stellt zusätzlich zur Kategorie- und Instanzinformation des Objektes auch dessen Lageinformation bereit. Wie auch in der Arbeit von Gupta u. a. (2014), werden bei dieser Arbeit ebenfalls zwei separate Faltungsnetzwerke eingesetzt. Abbildung 2.3 zeigt die Architektur des Ansatzes grafisch.

Bei allen Eingangsbildern wird zunächst das Objekt vom Hintergrund nach Holz u. a. (2011) segmentiert, um möglichst nur die Merkmale vom Objekt selbst für die Erkennung zu verwenden. Anschließend werden die 3D-Daten im segmentierten Bereich zusätzlich auf fehlende

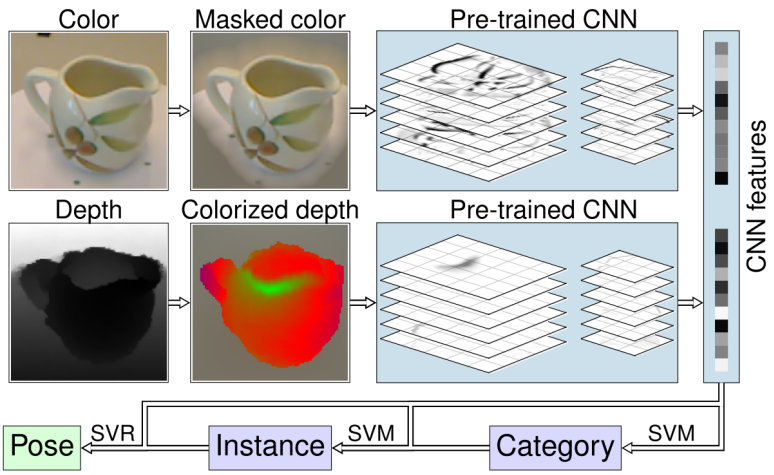


Abbildung 2.3.: Architektur der 3D-Objekterkennung von Schwarz *u. a.* Schwarz u. a. (2015).

Punkte in der Punktwolke geprüft und diese gegebenenfalls hinzugefügt. Aus den vollständigen 3D-Daten des Objektes wird anschließend ein Mesh[1] erzeugt, das als nächstes mithilfe eines Farbsystems eingefärbt wird. Hierfür wird zunächst der Mittelpunkt des Objektes bestimmt. Anschließend wird jedes Pixel des Objekts entsprechend seines Abstands zu dieser Mitte eingefärbt. Wie in Abbildung 2.4 dargestellt, verändert sich dabei die Farbe bei zunehmendem Abstand von Grün zu Rot über Blau zu Gelb. Das zweite Faltungsnetzwerk verwenden Schwarz u. a. (2015) ebenfalls, um Merkmale aus den RGB-Informationen zu extrahieren. Nach Extrakti-

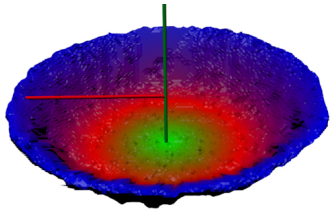


Abbildung 2.4.: Farbsystem das von Schwarz u. a. (2015) verwendet wird.

on der Merkmale mit beiden Faltungsnetzwerken werden diese mit Unterstützung einer SVM dazu genutzt, die Kategorie des Objektes zu bestimmen (siehe Abbildung 2.3). Anschließend wird das Ergebnis hieraus, sowie abermals die Merkmale, verwendet um auch die Instanz zu

[1] Als Mesh wird allgemein ein Polygonnetz zur Modellierung von 3D-Objekten bezeichnet. Es bildet als Folge die Grundstruktur eines 3D-Modells.

ermitteln. Auch dieser Schritt erfolgt mit einer SVM. Abschließend wird der Lagewinkel des Objektes ermittelt. Hierzu wird ein Support Vector Regressor[2] verwendet.

Beide Ansätze zeigen, dass durch das Hinzuziehen von Tiefeninformationen die Objekterkennung noch weiter verbessert werden kann. Deutlich wird zudem, dass die Tiefeninformationen bei beiden Arbeiten verwendet wurden, um einen Mehrwert aus ihnen zu schaffen. Bei Schwarz u. a. handelt es sich dabei um das Farbsystem und bei Gupta u. a. um die „Geocentric Encodig of Depth"-Codierung. Insgesamt erzielten beide Ansätze das Ergebnis, dass eine Kombination von Farb- und Tiefenbildinformationen jeweils die besten Erkennungserfolge liefert.

2.2. Farbraum

Der Farbraum beschreibt die Farbkombination, die innerhalb eines Farbmodells möglich ist. Angegeben werden die Farben in der Regel dreidimensional, woraus sich dann der Begriff Farbraum ergibt. Es folgt eine Beschreibung vom HSI- sowie des bekannteren RGB-Farbraums, um die Unterschiede (siehe Tabelle 2.3) sowie mögliche Vor- und Nachteile zu verdeutlichen.

2.2.1. RGB-Farbmodell

Das RGB-Farbmodell ist ein additives Farbmodell. Bei diesem werden die drei Primärfarben Rot, Grün und Blau jeweils in bestimmten Anteilen vermischt, um die gewünschte Farbe zu generieren. Alle drei Farben sind aufgrund dessen für die Mischung der zu entstehenden Farbe stark voneinander abhängig. Dies hat zur Folge, dass eine leichte Veränderung von beispielsweise der Belichtung eine zum Teil starke Veränderung aller RGB-Werte mit sich führt.

Tabelle 2.1 zeigt ein Beispiel hierfür. Dort wurde die Helligkeit der Farbe Blau erhöht. Wie zu sehen ändern sich alle drei Werte (R, G, B) des Farbmodells aufgrund der Additivität. Besonders bei dem Grün- und Rot-Anteil des Farbraums ist diese Änderung signifikant. Bei der Objekterkennung in der Robotik könnte dieser Effekt ein Nachteil sein, da sich die Belichtung oft schon durch eine kleine Bewegung des Roboters ändert.

[2] Ein Support Vector Regressor (SVR) wird nicht wie eine SVM zur Klassifizierung von Objekten eingesetzt, sondern für die Regression bzw. die Berechnung von einem realen Wert (hier: Lagewinkel des Objektes).

	Farbe		
R:	22	43	125
G:	41	70	142
B:	148	223	236

Tabelle 2.1.: RGB-Farbraum

	Farbe		
H:	154	154	154
S:	178	178	178
I:	80	125	170

Tabelle 2.2.: HSI-Farbraum

Tabelle 2.3.: Unterschied der Farbwerte bei den zwei Farbräumen RGB und HSI und variierender Helligkeit.

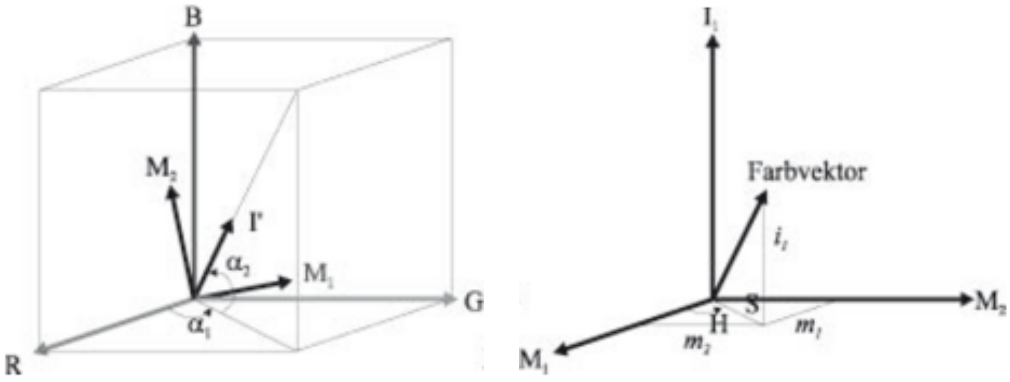
2.2.2. HSI-Farbmodell

Zur Gewährleistung einer besseren Erkennung von unterschiedlich belichteten Objekten wird der HSI-Farbraum als Alternative zum RGB-Farbraum in dieser Arbeit evaluiert. Dieser besteht ebenfalls aus drei Eigenschaften, allerdings handelt es dabei nicht um Farbanteile. Die Farbe wird stattdessen mithilfe des Farbtons (*engl.* **H**ue), der Sättigung (*engl.* **S**aturation) und der Lichtintensität (*engl.* **I**ntensity) beschrieben. Der HSI-Farbraum ist kein additiver Farbraum. Dies führt dazu, dass zum Beispiel eine Änderung der Helligkeit nur den Parameter für die Lichtintensität, allerdings nicht die zwei anderen Parameter beeinflusst.

Tabelle 2.2 zeigt dies beispielhaft. Für dieses Beispiel wurden die gleichen Farben wie in Beispieltabelle 2.1 des RGB-Farbraums gewählt. Ein Vergleich zwischen beiden Tabellen zeigt allerdings, dass anders als bei RGB, im HSI-Farbraum bei einer Erhöhung der Helligkeit sich lediglich die Lichtintensität ändert. Aufgrund dieser Eigenschaft wird eine Verwendung des HSI-Farbraums evaluiert mit dem Ziel bei unterschiedlichen Lichtverhältnissen eine robuste Objekterkennung zu erreichen.

2.2.3. Beziehung zwischen RGB und HSI

Da es sich bei RGB um einen additiven Farbraum handelt, und eine bestimmte Farbe somit eine Linearkombination der drei Primärfarben ist, wird er als dreidimensionales kartesisches Koordinatensystem dargestellt. Jeder Punkt in einem solchen Koordinatensystem entspricht einer Farbe mit einer bestimmten Helligkeit. Im Gegensatz dazu verwendet das HSI-Farbmodell eine Darstellung mit Zylinderkoordinaten. Abbildung 2.5 zeigt die Zusammenhänge zwischen dem RGB- und dem HSI-Farbmodell. Für eine Umrechnung von RGB-Daten in HSI-Daten sind zwei Schritte notwendig.



(a) (b)

Abbildung 2.5.: Zusammenhang zwischen dem RGB- (a) und HSI-Farbmodell (b) (Nischwitz u. a. (2011), S. 50).

Im ersten Schritt wird das Koordinatensystem gedreht, damit es mit der Raumdiagonale des HSI-Farbmodells zusammenfällt. Die Achse wird, wie in Abbildung 2.5.b zu sehen, mit I bezeichnet. Die zwei übrigen Achsen erhalten die Bezeichnung M1 und M2. Formel 2.1 zeigt die Gleichung, die für den ersten Schritt nötig ist (vgl. Nischwitz u. a. (2011), S. 45ff).

$$\begin{pmatrix} m_1 \\ m_2 \\ i_1 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.1}$$

Im zweiten Schritt werden anschließend die kartesischen Koordinaten m_1, m_2 und i_1 in das Zylinderkoordinatensystem des HSI-Farbmodells umgerechnet. Die Umrechnung ist in Formel 2.2 zu finden.

$$H = \arctan(\frac{m_1}{m_2}), S = \sqrt{m_1^2 + m_2^2}, I = \sqrt{3} \cdot i_1. \tag{2.2}$$

Ein Vektor aus dem RGB-Koordinatensystem entspricht demzufolge (siehe Abbildung 2.5) einem Vektor im gedrehten HSI-Koordinatensystem. Dort bestimmt die H-Komponente des Farbmodells dann den Winkel des Vektors mit der M_1-Achse und die S-Komponente den Betrag des Vektors. Die Komponente I kann anschließend abgelesen und mit der Formel aus 2.2 berechnet werden.

2.3. 3D-Daten

Bei den in dieser Arbeit verwendeten 3D-Daten handelt es sich um RGB-D-Daten. Als RGB-D-Daten werden von RGB-D-Datenquellen (Kameras) bereitgestellte Daten bezeichnet, die neben einem RGB-Bild auch die dazu passenden Tiefeninformationen (D) enthalten. Im Gegensatz zu reinen RGB-Datensätzen beinhalten RGB-D-Datensätze aufgrund des Aufwands beim Erstellen in den meisten Fällen eine wesentlich geringere Anzahl an Daten. Trotz des gesteigerten Aufwands existiert eine kleinere Auswahl von RGB-D-Datensätzen für unterschiedliche Zwecke. Des Weiteren können 3D-Kameras wie die Kinect One (siehe Abschnitt 2.3.1) von Microsoft verwendet werden, um RGB-D-Daten zu erzeugen.

Im Folgenden wird sowohl die Kinect One als auch der Washington RGB-D-Datensatz beschrieben. Beide werden im Rahmen dieser Arbeit verwendet.

2.3.1. Kinect One

Die Kinect One für Windows (Microsoft, 2014) ist eine von Microsoft entwickelte Kombination aus Farb- und Tiefenbildkamera. Es handelt sich bei der Kinect One um den Nachfolger der ebenfalls von Microsoft entwickelten Kinect. Aus diesem Grund wird die One auch als Kinect v2 bezeichnet. Daten, die aus einer Kombination von Farb- und Tiefenbild bestehen, werden auch als RGB-D-Daten bezeichnet (siehe Abschnitt 2.3). Wie in Abbildung 2.6 dargestellt, besitzt die Kinect One neben einer Farbkamera zusätzlich eine Infrarotkamera sowie einen Infrarot-Illuminator. Beide Infrarotsensoren liefern in Kombination die Tiefeninformationen.

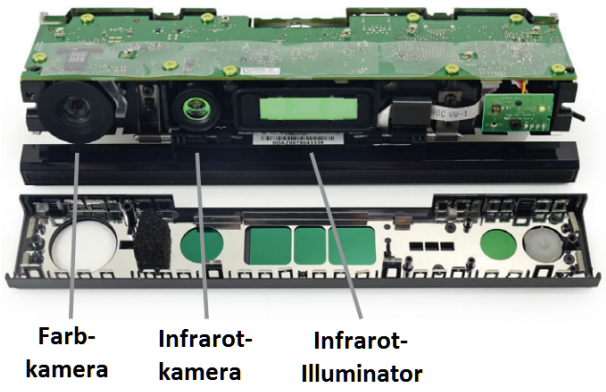


Abbildung 2.6.: Sensoren der Kinect One für Windows (vgl. Fankhauser u. a. (2015)).

Das System basiert dabei auf dem Time-of-Flight (ToF) Prinzip. Um Tiefeninformationen zu erhalten, sendet der Infrarot-Illuminator ein Muster aus, das von Hindernissen reflektiert wird. Die Infrarotkamera registriert anschließend die Laufzeit des ausgesendeten Lichts, bis zum Hindernis und zurück zur Kinect. Intern wird mithilfe von Wellenmodulation und Phasenverschiebung die Distanz jedes Pixels von der Kamera zum Hindernis berechnet. Die zur Verfügung stehende Auflösung für die Time-of-Flight-Erfassung beträgt 512 x 424 Pixel bei 30 Bildern pro Sekunde. Die Farbkamera verfügt über eine Auflösung von 1920 x 1080 bei einem Sichtfeld von 84.1° x 53.8°.

Durch den für eine solche Kamera günstigen Preis eignet sich die Hardware optimal für die Generierung von RGB-D-Daten zur Verwendung in dieser Arbeit. Da die Kinect One erst Ende 2014 veröffentlicht wurde, existiert zum jetzigen Zeitpunkt noch keine mit dem Gerät aufgenommene öffentliche RGB-D-Objektdatensammlung. Aus diesem Grund wird für diese Arbeit eine eigene Objektdatensammlung mithilfe der Kinect One angelegt.

2.3.2. Washington RGB-D-Objektdatensatz

Der RGB-D-Objektdatensatz (Lai u. a., 2011) ist ein Datensatz, der in Zusammenarbeit von der Universität Washington und dem Intel Lab in Seattle erzeugt wurde. Er beinhaltet 300 allgemein bekannte Objekte aus dem Haushalt, die in 51 Kategorien aufgeteilt sind. Ein Beispiel für aufgenommene Objekte ist in Abbildung 2.7 zu sehen.

Alle Bilder wurden mit einer Kinect ähnlichen 3D-Kamera mit 30 Hz und einer Auflösung von 640x480 aufgenommen. Jeder RGB-D-Frame besteht aus einem RGB-Bild und dem korrespondierenden Tiefenbild in Form einer Punktwolke. Die Tiefeninformationen werden als PCD-Dateien bereitgestellt. Dabei handelt es sich um das Punktwolkenformat der Point Cloud Library (siehe Abschnitt 5.1.1).

Die Aufnahme der Daten erfolgt auf einer Drehscheibe aus drei unterschiedlichen Polarwinkeln. Insgesamt wurden so 250.000 Frames aufgenommen. Abbildung 2.8 zeigt verschiedene Aufnahmen einer Kaffeetasse aus dem Datensatz. Insgesamt wurden von dieser Tasse 196 Aufnahmen aus unterschiedlichen Winkeln aufgezeichnet. Die Anzahl der Aufnahmen variiert von Objekt zu Objekt. Alle Objekte in diesem Datensatz sind nicht nur nach ihrer Kategorie gekennzeichnet, sondern auch mit speziellen Instanzeninformationen. Das heißt, dass zum Beispiel jedes Objekt der Kategorie Getränkedose zusätzlich noch eine spezielle Instanzeninformation enthält, um eine Erkennung verschiedener Objektinstanzen einer Objektkategorie zu ermöglichen.

Abbildung 2.7.: Auszug aus dem RGB-D-Objektdatensatz (Lai u. a., 2011).

Abbildung 2.8.: Aufnahme einer Kaffeetasse aus drei unterschiedlichen Winkeln aus dem Washington RGB-D-Objektdatensatz (Lai u. a., 2011).

3. Überwachtes Lernen

Das überwachte Lernen (*engl.* supervised learning) ist ein Teilgebiet des maschinellen Lernens. Bei dieser Form des Lernens versucht ein Lernalgorithmus eine Eingabe x, zu einer Ausgabe y zu assoziieren. Es handelt sich um überwachtes Lernen, da der Lernalgorithmus versucht diese Assoziation mithilfe von beispielhaften Trainingsdaten zu erzeugen. Für diesen Zweck existiert eine Vielzahl von unterschiedlichen Lernalgorithmen. Zwei weitverbreitete Methoden sind künstliche neuronale Netze, zusammen mit der Spezialform Faltungsnetzwerke, und Support Vector Machinen. Beide werden im Weiteren erläutert, da sie in dieser Arbeit verwendet werden.

3.1. Künstliche neuronale Netze

Künstliche neuronale Netze (*engl.* Artificial Neural Networks (ANN)) bestehen aus einer Struktur von künstlichen Neuronen, die miteinander vernetzt sind. Der Begriff „künstlich“, der im Weiteren entfällt, dient zur Abgrenzung zu biologischen neuronalen Netzen. Ein biologisches Neuron ist die Basisrecheneinheit des menschlichen Gehirns. Es gibt verschiedene Arten von biologischen und somit auch von künstlichen Neuronen. An dieser Stelle wird nur auf das Perzeptronneuron eingegangen. Das menschliche Gehirn besitzt ungefähr 86 Millionen dieser Neuronen, die mit circa 10^{14} - 10^{15} Synapsen zu einer Struktur vernetzt sind. Die Funktionsweise von künstlichen Neuronen ist den biologischen nachempfunden. Es existieren viele verschieden strukturierte Netze und das Finden einer neuen, verbesserten Struktur ist ein aktuelles Forschungsthema. Jedes dieser Netze besteht allerdings zumindest zum Teil aus mehreren vernetzten Neuronen.

Abbildung 3.1 zeigt das mathematische Modell eines einzelnen künstlichen Neurons im Vergleich zum biologischen nach Li und Karpathy (2015). Jedes Neuron empfängt Eingangssignale von seinen Dendriten und sendet Ausgangssignale über das Axon. Das Axon verzweigt sich schließlich und verbindet sich mittels Synapsen zu Dendriten anderer Neuronen. Im mathematischen Modell interagieren die Signale, die vom Axon (beispielsweise x_0) gesendet werden,

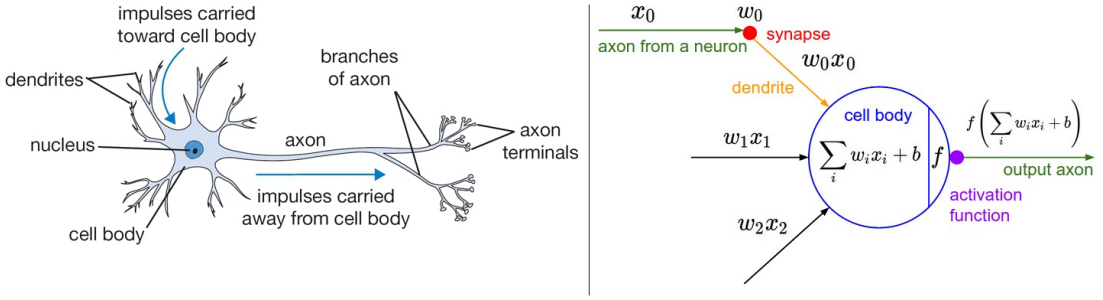


Abbildung 3.1.: Vergleich eines biologischen Neurons mit der mathematischen Umsetzung eines künstlichen Neurons (Li und Karpathy, 2015).

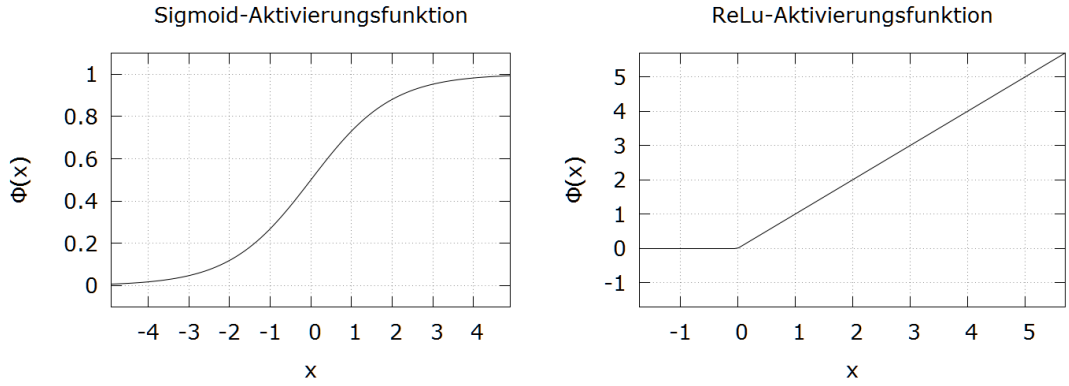
multiplikativ (beispielsweise w_0x_0) mit den Dendriten von anderen Neuronen, basierend auf der Synapsenstärke (beispielsweise w_0).

Die Idee hinter der Synapsenstärke (die Gewichtungen w) ist, dass diese erlernt wird und so den Einfluss, den ein Neuron auf ein anderes hat, bestimmen kann. Im biologischen Modell leiten die Dendriten anschließend Signale zum Zellkern weiter. Dort werden diese summiert und das Neuron ist in der Lage ein Signal über das Axon zu senden, sobald ein bestimmter Schwellenwert erreicht wird. Im mathematischen Modell wird der Zeitpunkt des Sendens vernachlässigt und es wird nur die Frequenz, in der das Signal gesendet wird, berücksichtigt. Dies wird mithilfe einer Aktivierungsfunktion modelliert.

Im mathematischen Beispiel aus Abbildung 3.1 handelt es sich um die Aktivierungsfunktion f. Mithilfe von f wird bestimmt, mit welcher Frequenz die Signale an das Axon weitergeleitet werden. Das Ergebnis der Aktivierungsfunktion f ist somit der Ausgangswert ϕ des Neurons. Die gesamte Berechnung innerhalb eines Neurons kann wie in Gleichung 3.1 dargestellt zusammengefasst werden.

$$\phi(x_i) = f\left(\sum_i w_ix_i + b\right) \tag{3.1}$$

Der Parameter x repräsentiert an dieser Stelle die Eingangsvariablen, während die Gewichtungen mit w bezeichnet werden. Die Anzahl von Eingangsvariablen und Gewichtungen muss stets gleich sein, da jede Variable mit ihrem respektiven Gewicht multipliziert wird. Zusätzlich wird der Parameter b addiert. Dieser wird als Bias bezeichnet. Er stellt eine Konstante dar und wirkt auf das Ausgabeverhalten eines Neurons. Die mithilfe von b erreichte Verzerrung ermöglicht erst das Abbilden von nichtlinearen Funktionen mit neuronalen Netzen.



(a) Sigmoid-Aktivierungsfunktion (b) ReLu-Aktivierungsfuntkion

Abbildung 3.2.: Beispiel für zwei unterschiedliche Aktivierungsfunktionen ϕ eines künstlichen Neurons.

Als Aktivierungsfunktion können verschiedene Funktionen verwendet werden. Abbildung 3.2.a zeigt die weit verbreitete Sigmoid $\phi(x) = \frac{1}{1+e^{-x}}$ Aktivierungsfunktion. In Abbildung 3.2.b ist die aufgrund aktueller Ergebnisse bevorzugte Rectified Linear Units (ReLu) $\phi(x) = max(0, x)$ Aktivierungsfunktion zu sehen.

Aktivierungsfunktionen weisen zum Teil sehr unterschiedliche Eigenschaften auf. Aus dem Vergleich der beiden Abbildungen 3.2 wird deutlich, dass bei der Sigmoid-Aktivierungsfunktion alle Werte positiv und im Bereich zwischen 0.0 und 1.0 bleiben. Die später eingeführt ReLu-Aktivierungsfunktion hingegeben liefert für alle negativen Eingaben den Wert 0 und arbeitet ansonsten linear. Eine Einschränkung in einem bestimmten Wertebereich findet also nicht statt.

3.1.1. Architektur

Die Struktur eines neuronalen Netzes wird auch als Netzarchitektur bezeichnet. Um eine solche Architektur zu bilden, werden mehrere Neuronen zu einem künstlichen neuronalen Netz zusammengeschlossen. Die Neuronen erfüllen dabei unterschiedliche Aufgaben. Abbildung 3.3 zeigt beispielhaft den Zusammenschluss zu einer solchen Netzarchitektur. Wie dort zu sehen, werden die Neuronen durch Knoten und die Verbindungen zwischen diesen als Kanten dargestellt. Die Neuronen werden bezüglich ihrer Aufgabe in Schichten (*engl.* Layer) angeordnet. Eine Verbindung zwischen Neuronen ist nur zwischen benachbarten Schichten erlaubt. Die

Beispielnetzarchitektur (siehe Abbildung 3.3) besteht aus einer Eingabeschicht (L_0), einer versteckten Schicht (L_1) sowie einer Ausgabeschicht (L_2).

Die L_0-Schicht nimmt die Eingabedaten $x_1, ..., x_n$ entgegen. Anschließend werden diese von der L_1-Schicht ($v_1, ..., v_n$) verarbeitet und an die Ausgabeschicht L_2 (α) weitergegeben, die wiederum das Ergebnis y ausgibt. Jede Architektur besitzt genau eine Eingabe- und eine Ausgabeschicht. Die Anzahl der versteckten Schichten ist jedoch variabel und unterscheidet sich von Architektur zu Architektur. Aktuelle Ansätze aus der Forschung bestehen zum Teil aus mehr als 152 gewichteten Schichten (He u. a., 2015).

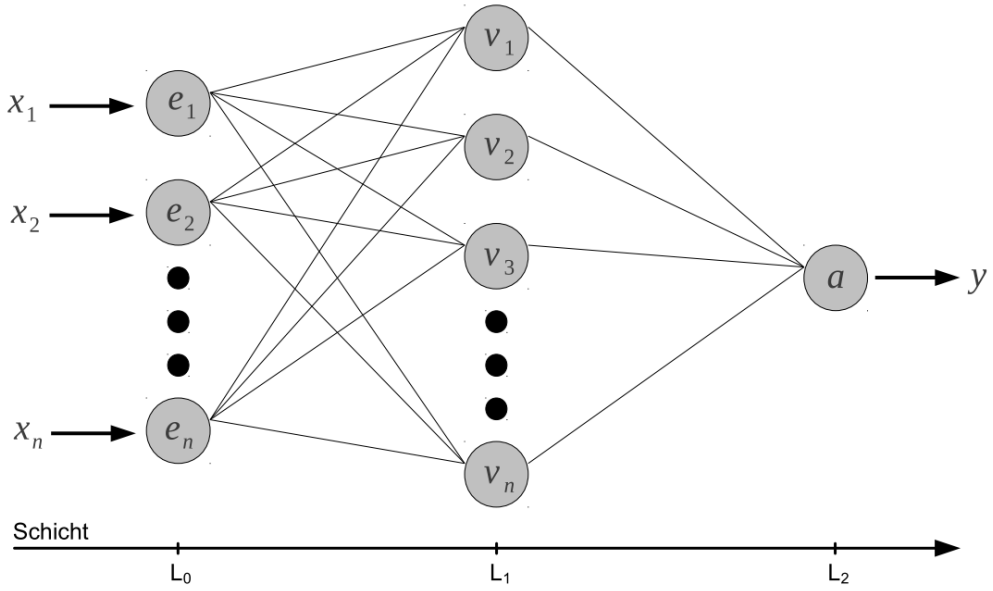


Abbildung 3.3.: Beispielhafter Aufbau der Netzarchitektur eines künstlichen neuronalen Netzes (vgl. Hassenkloever (2012)).

3.1.2. Lernprozess

Nach dem Vorbild des biologischen neuronalen Netzes im Gehirn muss auch das ANN trainiert werden, um etwas Neues zu erlernen. Das Training erfolgt mithilfe von Eingabedaten, zu denen bereits die passenden Ausgabedaten vorliegen. Konkret erfolgt das Training, indem ein Eingabedatum das Netz durchläuft und anschließend die Abweichung des Netzergebnisses von dem zu diesem Eingabedatum passenden Ausgabedatum berechnet wird. Die Funktion dieser Berechnung wird als Fehlerfunktion E (*engl.* loss function) bezeichnet. Ziel des Trainings ist die Minimierung des Ergebnisses dieser Fehlerfunktion. Um dieses Ziel zu erreichen, werden basie-

rend auf der Abweichung zwischen dem Resultat des Netzes und dem tatsächlichen Ergebnis anschließend Korrekturen an den Gewichtungen und dem Bias der Neuronen vorgenommen.

Für das Training, also die Minimierung des Fehlers, stehen verschiedene Optimierungsverfahren zur Verfügung. An dieser Stelle wird nur die für diese Arbeit interessante Methode des Gradientenabstiegs weiter erläutert. Der Gradientenabstieg verfolgt das Ziel einen möglichst geringen Fehler innerhalb eines Fehlergebirges zu finden. Umgesetzt wird dies mithilfe mehrerer Schritte, die immer in die Richtung des aktuell steilsten Abstieges gerichtet sind.

Abbildung 3.4 zeigt eine Visualisierung des Fehlergebirges E für die zwei Gewichtungen w_1, w_2 und das beschriebene Vorgehen. Dort ist der Prozess des Gradientenabstiegs dargestellt. Beim Start des Prozesses ist der Fehler groß und befindet sich aus diesem Grund auf dem Maximum des Fehlergebirges. Mit fortlaufendem Prozess und steigender Anzahl von Schritten (siehe Abschnitt 3.1.3) folgt eine Annäherung zum kleinsten Fehler im Tal des Fehlergebirges. Die Kombination aus dem Optimierungsverfahren des Gradientenabstiegs zusammen mit der

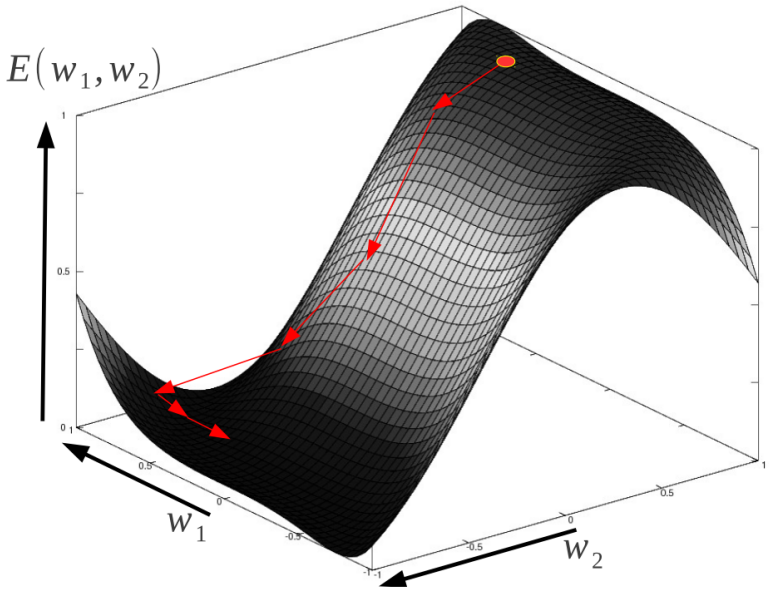


Abbildung 3.4.: Visualisierung der einzelnen Schritte des Gradientenabstiegs zum Finden des kleinsten Fehlers E (Hassenkloever, 2012).

gleichzeitigen Anpassung der Gewichtungen wird als Backpropagation bezeichnet.

Für das Training eines gesamten neuronalen Netzes mit Backpropagation muss jedes Neuron von der Ausgabe- bis zur Eingabeschicht angepasst werden. Abbildung 3.5 zeigt diesen Ablauf. Die Anpassung fängt bei der Ausgabeschicht an und arbeitet sich dann rückwärts durch das

gesamte Netz bis hin zur Eingabeschicht. Zur Ermittlung des Faktors für die Anpassung der Neuronengewichte wird ein Differenzvektor berechnet. Dieser ergibt sich aus dem Ergebnis, dass vom Netz ermittelt wurde und dem tatsächlichen Ergebnis. Der Zeitpunkt der Gewichtsaktualisierung ist variabel und wird mithilfe der Trainingsparameter (sieh Abschnitt 3.1.3) festgelegt.

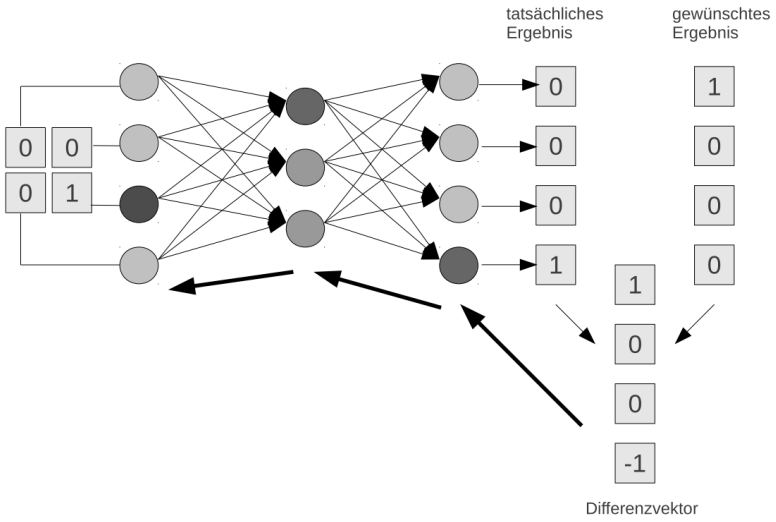


Abbildung 3.5.: Vorgehen von Backpropagation bei einem neuronalen Netz (Hassenkloever, 2012).

3.1.3. Hyperparameter

Als Hyperparameter werden, mit Ausnahme der Gewichtungen, die zahlreichen Parametereinstellungen des neuronalen Netzes bezeichnet. Dazu zählt zum Beispiel die Auswahl der Aktivierungsfunktionen, die Anzahl der versteckten Schichten, die Architektur des Netzes sowie die Einstellungen, die bei einem Trainingsverfahren vorgenommen werden können. Des Weiteren ergeben sich zusätzliche Parameter bei der Verwendung von Faltungsnetzwerken, wie zum Beispiel die Konfigurationsmöglichkeiten bei Faltungsoperationen oder Max-pooling (siehe Abschnitt 3.2).

Die Leistungsfähigkeit des Netzes hängt stark vom Finden der optimalen Hyperparameter ab. Dies hängt vor allem damit zusammen, dass jeder Datenbestand sich unterscheidet und infolge dessen auch jedes optimale neuronale Netz zum Erlernen eines Datenbestands anders ist. Aus diesem Grund stellt das Finden von diesen eine große Herausforderung dar. Im Folgenden

werden die Hyperparameter beschrieben, die in allen Netzen vorkommen und nicht nur in speziellen Netzformen eine Verwendung finden.

Trainingsparameter

Bei den drei häufigsten Trainingsparametern handelt es sich um Lernrate (*engl.* learning rate), Momentum und Chargengröße (*engl.* batch size). Wie bereits in Abschnitt 3.1.2 beschrieben wird die Richtung, in die das Lernverfahren sich im Fehlergebirge bewegt, mithilfe von Gradienten berechnet. Mit der Lernrate wird angegeben, wie groß die Schrittweite ist, mit der sich das Lernverfahren im Fehlergebirge bewegt. Wird dieser Wert zu groß gewählt, kann ein Minimum leicht übersehen werden, während ein zu kleiner Wert den Lernprozess stark verlangsamt. Zur Vermeidung dieser Nachteile wird die Lernrate während des Trainings an die Umgebung angepasst. Hierdurch wird erreicht, dass ein Minimum gefunden wird. Ob es sich dabei um ein globales oder lokales Minimum handelt, kann nicht festgestellt werden. Des Weiteren erfolgt eine generelle Verringerung der Lernrate mit fortschreitender Trainingsdauer. Hierdurch steigt die Wahrscheinlichkeit auch ein Minimum in einer Schlucht mit sehr schmalem Durchmesser zu finden.

Eine indirekte Anpassung der Lernrate wird zudem durch die Momentum-Eigenschaft vorgenommen. Diese Eigenschaft veranlasst das Lernverfahren die Lernrate bei Plateaus zu vergrößern und bei einer steilen Schlucht zu verkleinern. Abbildung 3.6 zeigt ein Beispiel für beide Fälle. Dies führt zu einer Problemminderung bei Plateaus und der Oszillation bei steilen Schluchten.

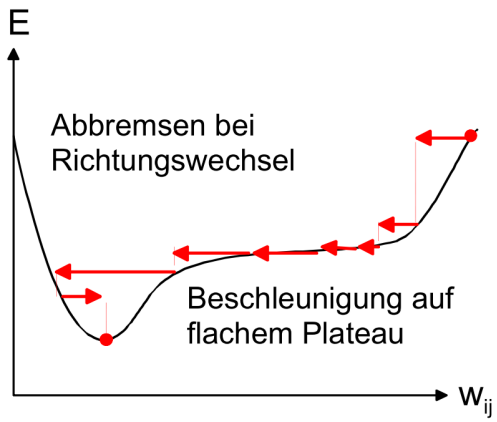


Abbildung 3.6.: Auswirkungen von Momentum zur Problemminderung bei Plateaus und Oszillation (Meisel, 2012).

Momentum und Lernrate stehen demzufolge in einer Beziehung zueinander. Aktuelle Ansätze verwenden zunächst eine große Lernrate in Kombination mit einem kleinen Momentum. Bei fortschreitendem Training wird anschließend die Trainingsrate immer weiter verkleinert, während das Momentum vergrößert wird (vgl. Heaton, 2015, S. 263).

Die Chargengröße gibt an wie viele Trainingsexemplare berechnet werden sollen bevor die Gewichtungen aktualisiert werden. Eine Chargengröße von 1 legt beispielsweise fest, dass die Gewichtungen nach jedem Trainingsexemplar angepasst werden. Laut Heaton (2015) sollte die Chargengröße am Anfang des Trainings etwa 10% der Anzahl an Trainingsexemplaren betragen.

Netzparameter

Neben den Trainingsparametern zählen auch die Eigenschaften zur Konfiguration des Netzes zu den Hyperparametern. Diese sind signifikant wichtiger als die Trainingsparameter, da sie sich direkt auf die Lernkapazität des Netzes auswirken.

Die Konfiguration der Neuronen stellt eines der wichtigsten Eigenschaften dar. Neben der Wahl einer Aktivierungsfunktion muss zusätzlich festgelegt werden aus wie vielen Schichten das Netz bestehen soll und wie viele Neuronen sich in jeder Schicht befinden. Die Anzahl der Schichten und Neuronen wirkt sich auf die Lernkapazität des Netzes aus. Ein einzelnes Neuron kann als Funktionsapproximator betrachtet werden, der durch die Gewichtungen parametrisiert wird. Durch Steigerung der Anzahl an Neuronen ist es möglich, die Kapazität des neuronalen Netzes zu steigern und so komplexere Funktionen zu approximieren.

Abbildung 3.7 zeigt den Unterschied der Klassifizierungsgenauigkeit für ein binäres Klassifizierungsproblem bei einer unterschiedlichen Anzahl von Neuronen. Wie dort zu sehen kann der zweidimensionale Raum mit drei Neuronen nur grob klassifiziert werden, sodass viele Punkte der Klasse „Rot“ falsch eingeordnet wurden. Eine Klassifizierung mit 20 Neuronen hingegen ermöglicht die Approximation einer Funktion, die wesentlich komplexer ist und die es ermöglicht alle roten und grünen Punkte richtig einzuordnen.

Eine Erhöhung der Neuronenanzahl verbessert die Funktion des neuronalen Netzes allerdings nicht in jedem Fall. Sobald eine bestimmte Grenze erreicht wurde, wird durch das Hinzufügen von Neuronen eine sogenannte Überanpassung erzeugt. Eine Überanpassung (*engl.* overfitting) führt dazu, dass das Netz nicht mehr in der Lage ist die eigentlich Beziehung zwischen den Klassen zu approximieren, sondern nur noch Sonderfälle approximiert werden und so die Generalisierung verloren geht. Genau wie bei den Trainingsparametern ist es also auch an

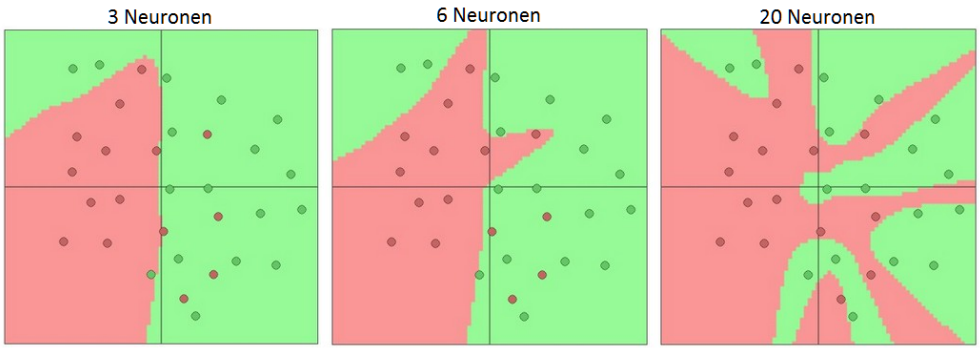


Abbildung 3.7.: Unterschied der Klassifizierungsgenauigkeit für ein binäres Klassifizierungsproblem bei einer unterschiedlichen Anzahl von Neuronen (Li und Karpathy, 2015).

dieser Stelle notwendig die optimalen Parameter für jeden neuen Datenbestand erneut zu ermitteln.

3.2. Faltungsnetzwerke

Faltungsnetzwerke (*engl.* Convolutional Neural Networks (CNN)) stellen eine Spezialform von künstlichen neuronalen Netzen dar. Das Originalkonzept der Faltungsnetzwerke wurde von Fukushima (1980) eingeführt und später von Lecun u. a. (1998) weiter verbessert. Aus dieser Forschungsarbeit ist das in Abbildung 3.8 dargestellte Faltungsnetzwerk „LeNET-5“ von Yann LeCun entstanden, mit dem es möglich ist handschriftliche Buchstaben zu erkennen (vgl. Heaton (2015)).

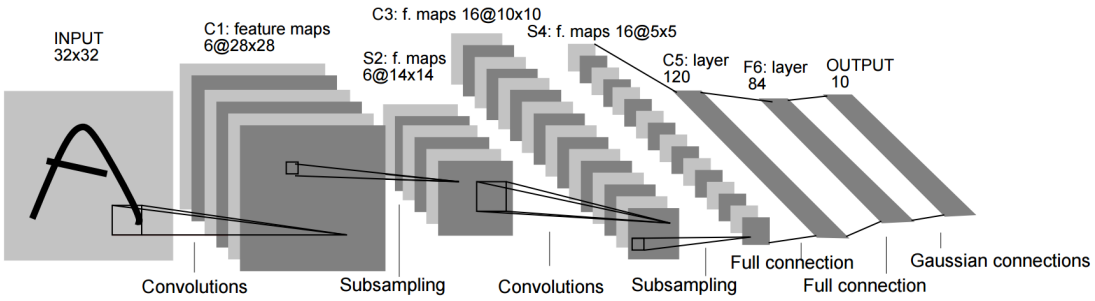


Abbildung 3.8.: LeNET-5 von Yann LeCun (Lecun u. a., 1998).

Neben den aus Abschnitt 3.1.1 bereits bekannten Schichten besitzen Faltungsnetzwerke zusätzliche Schichttypen mit speziellen Funktionen. Die drei bekanntesten wurden bereits im LeNET-5 genutzt. Wie in Abbildung 3.8 zu sehen handelt es sich dabei um die drei nachfolgenden Schichten, die unterschiedliche Aufgaben übernehmen.

- Convolutional-Schicht
- Subsampling-Schicht
- Fully-Connected-Schicht

Zusätzlich zu den drei erwähnten Schichten werden je nach Aufgabe noch weitere Schichten verwendet, die in dieser Arbeit nicht weiter betrachten werden sollen. Eine detaillierte Beschreibung der Zusatzschichten aus Abbildung 3.8 ist in Abschnitt 3.2.1 zu finden.

Ein weiterer Unterschied zu neuronalen Netzen ist die Form in der Daten durch das Netz transportiert werden. Bei ANN werden Daten in Form von Vektoren durch das Netz transportiert. Bei Faltungsnetzwerken sind die Daten in Form von 3D-Blöcken angeordnet. Die Höhe, Breite und Tiefe eines 3D-Blocks ist definierbar. Bei der Eingangsschicht hängt die Dimension von der Höhe und Breite des Eingangsbildes sowie die Tiefe von der Farbtiefe des Bildes ab. Bei einem RGB-Bild beispielsweise beträgt die Tiefe drei, bedingt durch die Rot-, Grün- und Blau-Kanäle des Bildes. Abbildung 3.9 zeigt zwei Arten von 3D-Blöcken. Während in Rot zunächst die Eingangsschicht dargestellt ist, zeigt der blaue 3D-Block beispielhaft die Anordnung der Gewichtungen der ersten Convolutional-Schicht. Die Dimension der Eingangsschicht beträgt in diesem Beispiel 32x32x3 (Höhe x Breite x (R+G+B)). Alle Neuronen aus dem blauen Block sind spatial mit einer kleinen Region des Bildes verbunden. Innerhalb dieser Region sind die Neuronen allerdings mit der gesamten Tiefe, also allen drei Farbwerten, vernetzt.

Diese spatialen Regionen werden auch als rezeptive Felder (*engl.* receptive fields) bezeichnet. Des Weiteren ist anzumerken das mehrere Neuronen (in diesem Fall fünf, aufgrund der Tiefe des Blocks) mit einem solchen Feld verbunden sind. Die Größe des Felds sowie die Tiefe des Blocks können als Hyperparameter definiert werden. Zur Reduzierung der Parameteranzahl werden die Gewichtungen in den 3D-Blöcken zwischen mehreren Neuronen geteilt.

Die Aufteilung geht dabei auf die Annahme zurück, dass Bilder viele räumliche Beziehungen zwischen benachbarten Pixeln besitzen, die nicht von ihrer Position innerhalb des Bildes abhängig sind (vgl. Zeiler (2014)). Ein gelerntes Merkmal an einer spatialen Position (x, y) kann demzufolge auch an einer anderen Position $(x2, y2)$ Verwendung finden. Aus diesem Grund werden die Gewichtungen der Neuronen innerhalb des 3D-Blocks anhand sogenannter Tiefenschnitte (*engl.* depth slices) geteilt. Am Beispiel aus Abbildung 3.9 bedeutet dies, dass fünf

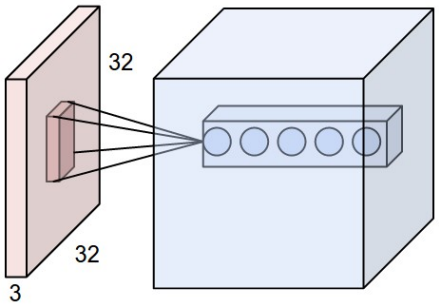


Abbildung 3.9.: Anordnung der Neuronen in einem 3D-Block innerhalb eines CNN (Li und Karpathy, 2015).

vertikale Schnitte vorgenommen werden, da die Tiefe aufgrund der fünf hintereinander angeordneten Neuronen fünf beträgt. Alle Neuronen innerhalb eines solchen Schnitts teilen sich dann die Gewichtungen. Dies führt dazu das Faltungsnetzwerke bis zu einem bestimmten Grad translations- und positionsinvariant sind.

3.2.1. Schichten

Faltungsnetzwerke nutzen, in Erweiterung zu einem neuronalen Netz, in mindestens einer Schicht eine mathematische Faltungsoperation anstelle einer Matrixmultiplikation (vgl. Goodfellow u. a., 2016, S. 334). Bei einer Faltung handelt es sich um eine Form der Bildfilterung.

Die Faltung ermöglicht es bestimmte Merkmale des Bildes hervorzuheben oder auszublenden. Für diese Operation wird ein sogenannter Faltungskern benötigt, der angibt, was für eine Faltungsoperation durchgeführt wird. Beispiele für einen solchen Kern sind das Schärfen des Bildes oder die Kantendetektion. Abbildung 3.10 zeigt die Faltung mit einer solchen Faltungsmaske. Wie dort visualisiert, und in Gleichung 3.2 formalisiert, wird die Faltungsmaske $h(m, n)$ zunächst an einer Stelle auf das Quellbild $s(x, y)$ gelegt. Anschließend werden die Farbwerte an dieser Stelle mit der Faltungsmaske multipliert und abschließend für das Zielbild $g(x, y)$ addiert.

$$g(x, y) = \sum_{m=-a}^{a} \sum_{n=-b}^{b} h(m, n) \cdot s(x - m, y - n) \tag{3.2}$$

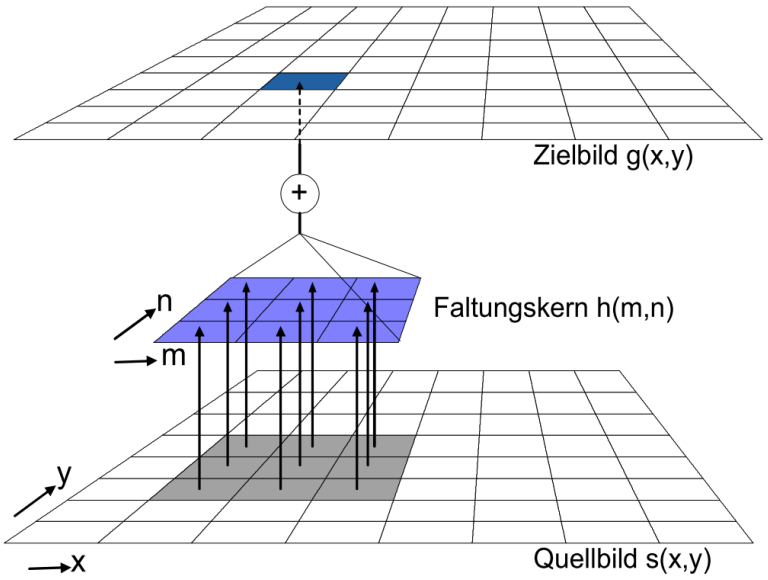


Abbildung 3.10.: Beispiel einer Faltungsoperation (Meisel, 2012).

Convolutional-Schicht

Die erste Schicht im LeNET-5 Beispiel aus Abbildung 3.8 ist eine Convolutional-Schicht. Die Hauptaufgabe einer solchen Schicht ist die Identifikation von visuellen Merkmalen, wie zum Beispiel Ecken oder Kanten. Hierfür werden die bereits in der Einleitung von Abschnitt 3.2.1 beschriebenen Faltungsmasken eingesetzt. Je mehr Filter an dieser Stelle verwendet werden, desto mehr unterschiedliche Merkmale können erkannt werden.

Tabelle 3.1 dient zur Verdeutlichung der Filteroperation bei einem Bild in der Convolutional-Schicht. Jede Zelle der Tabelle stellt dabei einen Pixel des Bildes dar. Der in grau markierte 2x2-Filter bewegt sich für die Faltungsoperation horizontal über das Bild, bis er das Ende erreicht. Das sogenannte Schritt-Hyperparameter (*engl.* stride) gibt die Anzahl an Positionen an, an denen der Filter hält. Im Beispiel beträgt die Schrittweite zwei. Jeder Schritt ist mit einer Farbverdunkelung visualisiert. Das bedeutet zudem, dass mit diesem Parameter definiert wird wie viele Neuronen innerhalb des 3D-Blocks gleichzeitig angesprochen werden. Wenn der Filter das rechte Ende des Bilds erreicht hat, bewegt er sich wieder horizontal nach links, um anschließend die definierte Anzahl an Schritten nach unten zu wandern und sich erneut horizontal nach rechts zu bewegen.

0	0	0	0	0	0
0	1	4	7	5	0
0	8	2	6	3	0
0	3	6	9	2	0
0	5	1	1	7	0
0	7	9	3	5	0
0	0	0	0	0	0

Tabelle 3.1.: Bewegung eines 2x2 Convolutional-Filter (grau) über ein Bild in einer Zeile mit einer Schrittweite von zwei. Jeder Schritt ist durch eine Farbverdunkelung visualisiert.

Diese Operation wird durchgeführt, bis das gesamte Bild vom Filter abgearbeitet wurde (vgl. Heaton, 2015, S. 200ff). Abbildung 3.8 zeigt sechs Filter in der ersten Convolutional-Schicht, die sich alle wie beschrieben über das Bild bewegen. Das Ergebnis dieser Filter wird in sogenannten Feature-Maps gespeichert. Diese Feature-Maps entsprechen den in Abschnitt 3.2 beschriebenen Tiefenschnitten innerhalb eines 3D-Blocks von Neuronen. Der 3D-Block respektive die Feature-Map-Sammlung „C1“ aus Abbildung 3.8 besitzt demzufolge eine Tiefe von sechs.

Die Anzahl der Schritte, die ein Filter benötigt um über ein Bild zu wandern, kann mithilfe von Gleichung 3.3 berechnet werden. Für die Berechnung wird neben der Breite (w) des Bildes, die Größe des Filters (f) und die Schrittweite (s) des Filters auch ein sogenannter Padding-Wert (p) benötigt. Padding wird zur Anpassung der Gleichung benötigt, da es sich bei der Anzahl der Schritte nur um einen Integer-Wert handeln darf. Das Padding wird in Tabelle 3.1 mithilfe der Nullen am Rand dargestellt.

$$Schritte = \frac{w - f + 2p}{s + 1} \tag{3.3}$$

Subsampling

Eine Subsampling-Schicht berechnet aus einem bestehenden 3D-Block einen neuen mit kleinerer Dimension. Die am weitesten verbreitete Methode hierfür ist das sogenannte Max-pooling. Auch für diesen Filter sind als Hyperparameter die Filtergröße sowie die Schrittweite festzulegen. Abbildung 3.11 zeigt das Max-pooling Verfahren beispielhaft für einen 2x2 Filter mit einer Schrittweite von zwei. Die Bewegung des Filters auf dem Bild ist dabei analog zu der Beschreibung der Convolutional-Schicht aus Abschnitt 3.2.1.

Bei Max-pooling wird vom Filter lediglich der größte numerisch Wert gewählt, der sich im aktuellen Filterbereich befindet, und als neuer Wert übernommen. Die verschiedenen Farben zeigen die einzelnen Schritte der Filterbewegung. Verkleinert werden bei dieser Operation nur die spatialen Abmessungen, die Tiefe des Blocks bleibt erhalten.

Die Anwendung von Max-pooling Schichten zwischen Convolutional-Schichten steigert die Merkmalsabstraktion, da die Verkleinerung es ermöglicht Merkmale höherer Ordnung zu erkennen.

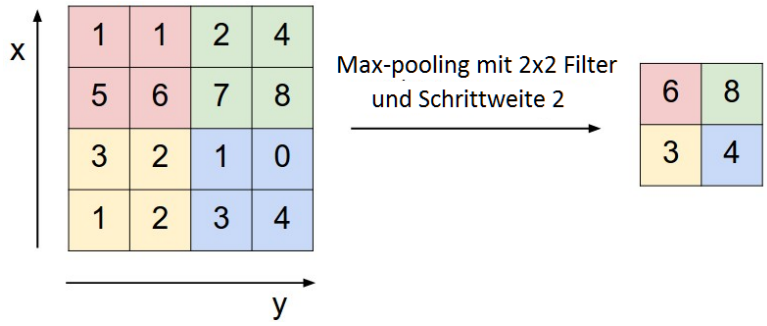


Abbildung 3.11.: Dimensionsreduktion mithilfe von Max-pooling (vgl. Li und Karpathy (2015)).

Fully-Connected

Die letzte Schicht aus dem LeNET-5 Beispiel ist eine Fully-Connected-Schicht. Dabei handelt es sich exakt um die gleiche Art von Schicht, die auch in einem künstlichen neuronalen Netz verwendet, und als versteckte Schicht bezeichnet, wird. Für die Weitergabe der Daten wird eine Verbindung zwischen jedem Neuron des 3D-Blocks der vorherigen Schicht mit jedem Neuron der Fully-Connected-Schicht hergestellt.

Dropout

Zusätzlich zu den bereits beschriebenen Schichten wird in aktuellen Faltungetzwerkarchitekturen mindestens eine Dropout-Schicht verwendet. Diese Art von Schicht dient zum Schutz vor einer Überanpassung (siehe Abschnitt 3.1.3). Erreicht wird dieses Ziel durch das periodische ausblenden von Neuronen während des Trainings. Die ausgeblendeten Neuronen werden während des Trainings allerdings nicht gelöscht, sondern stehen nach diesem wieder zur Verfügung. Durch das Ausblenden der Neuronen sind die übrigen Neuronen gezwungen ihre

Gewichtungen so anzupassen, dass auch ohne die Verfügbarkeit der ausgeblendeten Neuronen ein gutes Ergebnis erzielt wird. Dieser Vorgang verringert die Coadaption der Neuronen und erzeugt so ein Ergebnis mit einer geringeren Wahrscheinlichkeit für eine Überanpassung.

3.2.2. Lernprozess

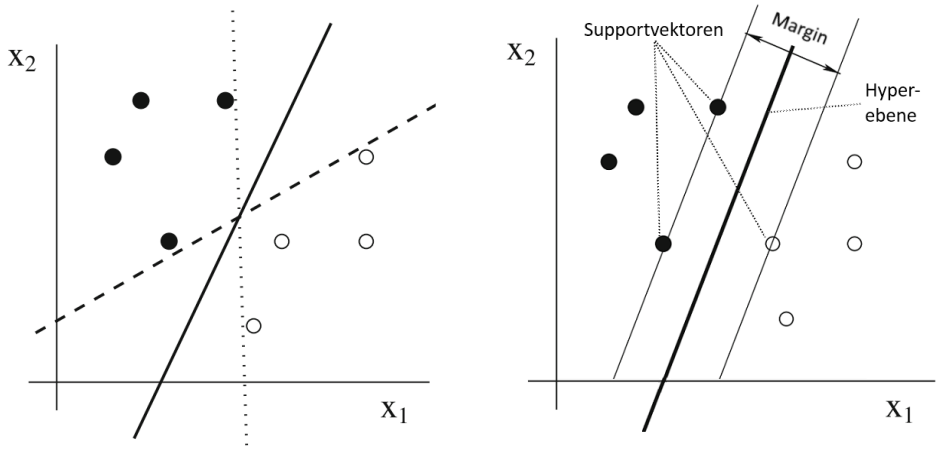
Der Lernprozess für ein künstliches neuronales Netz kann ebenso für ein Faltungsnetzwerk eingesetzt werden. Neben der Fully-Connected-Schicht, die der in Abschnitt 3.1.1 beschriebenen verstecken Schicht entspricht, besitzt auch die Convolutional-Schicht durch Training erlernbare Parameter. Im Gegensatz dazu besitzen Subsampling- und Pooling-Schicht weder Neuronen noch Bias und aus diesem Grund auch keine erlernbaren Parameter. Als Trainingsalgorithmus für den Lernprozess kann das in Abschnitt 3.1.2 bereits beschriebene Backpropagation-Verfahren genutzt werden.

3.3. Support Vector Machinen

Eine Support Vector Machine (SVM) ist ein weiteres Verfahren des überwachten Lernens. Es wird eingesetzt um eine Menge von Objekten in zwei Klassen einzuordnen. Wie auch jedes andere Verfahren des überwachten Lernens werden auch hier Trainingsobjekte benötigt. Das Ziel des Trainings ist es eine lineare Trennebene zwischen zwei Klasse zu finden. Diese Trennebene wird auch als Hyperebene bezeichnet und verfolgt das Ziel die optimale Trennung zu erreichen. Die Güte der Trennung wird mithilfe des Abstands der Klassenobjekte zur Hyperebene, dem sogenannten Margin, definiert. Abbildung 3.12.a zeigt ein Beispiel für mehrere mögliche Hyperebenen.

Zur Findung der optimalen Hyperebene werden an beiden Seiten der Trennebenen sogenannte Supportvektoren festgelegt. Der Abstand der Hyperebene wird anschließend so lange erhöht, bis dieser die Supportvektoren berührt. Dargestellt wird dies in Abbildung 3.12.b. Die optimale Hyperebene wird gewählt anhand der maximalen Margin zu den beiden zu trennenden Klassen.

Anders als in den Abbildungen 3.12 können die Daten stärker verschränkt sein, sodass eine lineare Trennung nicht möglich ist. Da eine SVM aber nur in der Lage ist lineare Trennungen durchzuführen, kommt an dieser Stelle der sogenannte Kernel-Trick zum Einsatz. Dieser basiert auf der Annahme, dass in einem höher dimensionalen Raum eine lineare Trennung möglich ist. Abbildung 3.13 veranschaulicht diesen Effekt. Formal bedeutet dies, dass die Featurevektoren $x \in X$ mit einer nichtlinearen Abbildung



(a) Mögliche Hyperebenen (b) Optimale Hyperebene

Abbildung 3.12.: Beispiel für verschiedene Möglichkeiten von Hyperebenen (a) und der optimalen Auswahl zur Trennung (b) von zwei Klassen (Kubat, 2015, S. 84ff).

$$\begin{aligned} \Phi : X &\to Z \\ x &\to x := \Phi(x) \end{aligned}$$

in den sogenannten Featurespace Z transformiert werden. Diese Transformation ist sehr rechenintensiv. Aus diesem Grund wird anstatt der Berechnung des Skalarprodukts $(\Phi(x_i), \Phi(x))$ eine Kernelfunktion k verwendet, sodass Gleichung 3.4 gilt.

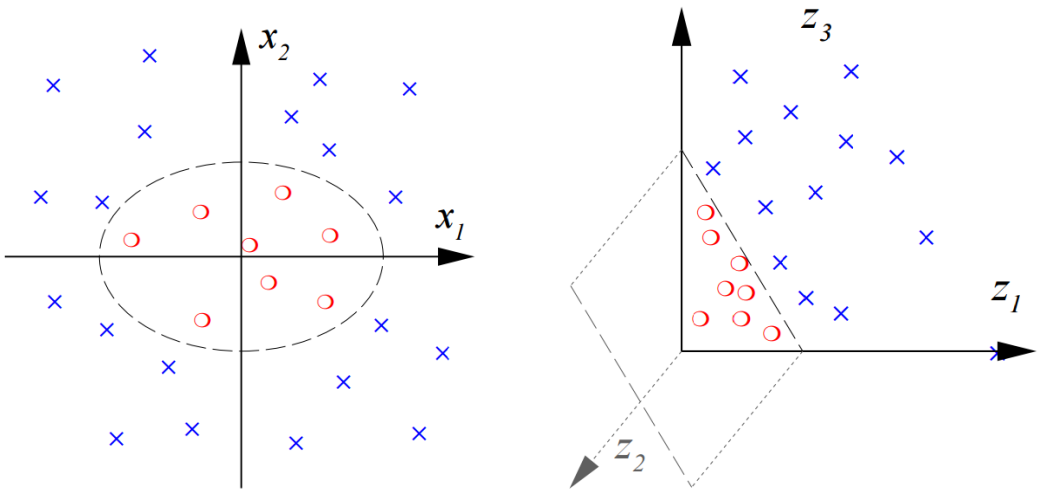
$$k(x_i, x) = (\Phi(x_i), \Phi(x)) \tag{3.4}$$

Für diesen Zweck existiert eine Vielzahl von Kernel-Funktionen. Die am häufigsten verwendeten werden im Folgenden in Gleichung 3.5 bis 3.8 aufgelistet.

- Linear-Kernel
$$k(x_i, x) = (\Phi(x_i), \Phi(x)) \tag{3.5}$$
- Polynomial-Kernel
$$k(x_i, x) = (\Phi(x_i), \Phi(x))^d, \quad d \in \mathbb{N} \tag{3.6}$$
- Radial-Basis-Funktion-Kernel
$$k(x_i, x) = exp\left(-\frac{||x_i - x||^2}{2\sigma^2}\right), \quad \sigma > 0 \tag{3.7}$$

- Sigmoid-Kernel

$$k(x_i, x) = tanh\left(k(x_i, x) + \theta\right), \qquad k > 0, \theta < 0 \tag{3.8}$$



(a) (b)

Abbildung 3.13.: Transformierung von Featurevektoren aus einem 2D-Raum (a) mithilfe des Kernel-Tricks in einen 3D-Raum (b), um eine lineare Trennung der Daten zu ermöglichen (Mueller u. a., 2001).

Da sich Support Vector Machinen nur zur Klassifizierung von zwei Klassen eigenen, müssen für eine Trennung von mehrklassigen Problemen weitere Verfahren hinzugezogen werden. Die verbreitetsten Verfahren sind One-Versus-One und One-Versus-Rest.

Bei One-Versus-One wird jede Klasse gegen jede andere Klasse trainiert. Hierdurch wird ein gerichteter, azyklischer Graph aufgebaut, der alle SVM-Paarungen enthält. Mithilfe von diesem kann anschließend entschieden werden, zu welcher Klasse ein Datum gehört. Zur Realisierung einer Klassifizierung von n Klassen werden für dieses Verfahren $\frac{(n-1)n}{2}$ Support Vector Machinen benötigt.

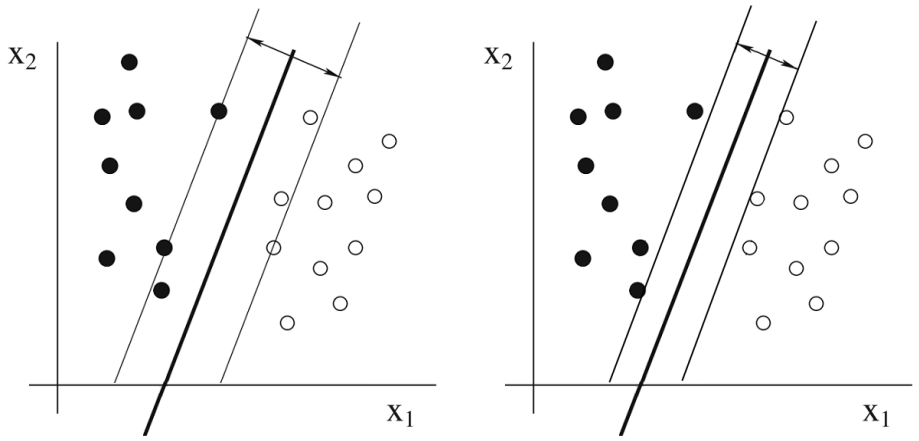
Das One-Versus-Rest-Verfahren trainiert für jede Klasse einen Klassifikator gegen die Kombination der Daten aller restlichen Klassen. Hieraus ergeben sich bei n Klassen als Folge n SVMs. Die Bestimmung der Klasse zu einem Datum wird anschließend mithilfe von Wahrscheinlichkeiten berechnet. Hierfür wird jedes neue Datum mit allen n SVMs klassifiziert und

anschließend wird die Klasse gewählt, dessen Ergebnis die höchste Wahrscheinlichkeit erzielt hat.

Hyperparameter

Neben den bereits in Abschnitt 3.1.3 beschriebenen Hyperparametern bei künstlichen neuronalen Netzen besitzen natürlich auch Support Vector Machinen diese Art der Parameter. Ein Beispiel ist der Kostenparameter (C). Er definiert die Toleranz, mit der falsch klassifizierte Daten bestraft werden. Abbildung 3.14 visualisiert den Effekt den der Parameter auf die Trennung der Daten besitzt.

Wird für den C-Parameter ein hoher Wert gewählt, so versucht die SVM alle Trainingsbeispiele korrekt zu klassifizieren (siehe Abbildung 3.14.b). Bei einem niedrigen Wert hingegeben erfolgt Gegenteiliges. Wie in Abbildung 3.14.a dargestellt werden dann für eine möglichst große Margin sogar Falschklassifizierungen in kauf genommen.



(a) Niedriger Wert des C-Parameters (b) Hoher Wert des C-Parameters

Abbildung 3.14.: Vergleich bei der Klassifizierung von Trainingsdaten mit verschiedenen Kostenparametern (C).

Je nach dem gewählten Kernel der SVM, ist es zusätzlich erforderlich noch weitere Hyperparameter zu ermitteln.

4. Entwurf

Im Folgenden wird zunächst das Konzept erläutert, das in dieser Arbeit umgesetzt wird. Anschließend wird die Architektur des entwickelten Konzepts beschrieben. Im Rahmen dieser Beschreibung wird detailliert auf den Aufbau der Architektur sowie das Verfahren zur Ermittlung der benötigten Hyperparameter und das Training eingegangen. Abschließend werden die verwendeten Softwarekomponenten zur Merkmalsextraktion, OUR-CVFH und VGGNet, erläutert.

4.1. Konzept

Die bereits beschriebenen State-of-the-Art-Verfahren der 3D-Objekterkennung (siehe Abschnitt 2.1) nutzen die zur Verfügung stehenden Tiefeninformationen nur codiert in ein Bild als Eingangsparameter für ein Faltungsnetzwerk und lassen eine automatische Merkmalsextraktion von diesem durchführen. Anders als bei diesen Ansätzen werden in dieser Arbeit die Tiefeninformationen genutzt, um aus ihnen geometrische Informationen des Objekts zu extrahieren. Da es sich dann bereits um Merkmale in Form von geometrischen Eigenschaften handelt, werden diese anschließend ohne die erneute Merkmalsextraktion durch ein Faltungsnetzwerk verwendet.

Des Weiteren wird in dieser Thesis ein Faltungsnetzwerk genutzt, um Merkmale aus dem Farbbild desselben Objekts zu extrahieren. Im Gegensatz zur üblichen Vorgehensweise wird das Faltungsnetzwerk allerdings nicht mit den verwendeten Daten trainiert. Stattdessen werden Gewichtungen eines bereits mit anderen Daten trainierten Faltungsnetzwerks verwendet. Für die Umsetzung wurde die VGGNet-Faltungsnetzwerkarchitektur gewählt. Es handelt sich dabei um ein Netz, dass 19 beziehungsweise 16 gewichtete Schichten besitzt und das für den ILSVRC-Wettbewerb 2014 konzipiert wurde. Die hier verwendeten Gewichtungen stammen vom Training des VGGNet mit Daten des ImageNet-Datensatzes (Deng u. a., 2009). Insgesamt besteht dieser Datensatz aus 1000 Objektklassen und umfasst allein für das Training circa 1.3 Millionen Bilder.

Diese Daten unterscheiden sich zwar von den Daten des in dieser Arbeit verwendeten Washington RGB-D-Objektdatensatzes, da das Faltungsnetzwerk allerdings nur verwendet wird, um mit diesem die Merkmale eines Objekts zu extrahieren und nicht zu klassifizieren, kann dies vernachlässigt werden.

Aktuelle Arbeiten (vgl. Zeiler und Fergus (2013); Simonyan und Zisserman (2014); Szegedy u. a. (2014); He u. a. (2015)) zeigen, dass die Objekterkennung durch eine steigende Anzahl an Schichten verbessert wird. Ein Grund zur Verwendung eines bereits trainierten Faltungsnetzwerks sind die für die Durchführung eines Trainings benötigten Ressourcen von Netzen solcher Art. Diese Netze benötigen in jedem Fall eine GPU[1] für die Ausführung, da eine oder mehrere CPUs die Verarbeitungszeit sowohl beim Training als auch bei der Erkennung auf mehrere Wochen steigern würden. In den meisten Fällen werden sogar mehrere GPUs benötigt. Das VGGNet bietet einen guten Kompromiss aus vielen Schichten und somit einer guten Erkennungsgenauigkeit, kann aber dennoch mit nur einer einzigen GPU ausgeführt werden. Zusätzlich sind durch das bereits durchgeführte Training alle Hyperparameter des Netzes bereits bekannt und liefern für Farbbilder gute Ergebnisse. Ein weiterer Grund zur Wahl eines trainierten Faltungsnetzwerks ist, dass somit in der entwickelten Architektur nur noch die SVM trainiert werden muss. Weitere Details zum VGGNet und dessen Netzarchitektur sind in Abschnitt 4.4 zu finden.

Das Ziel der Extraktion dieser unterschiedlichen Merkmalsarten (Farb- und Tiefeninformationen) ist es, so eine höherwertige Beschreibung des Objekts zu schaffen und eine robuste Objekterkennung bereitzustellen. Dieser Ansatz geht auf die Annahme zurück, dass mit einer Steigerung der Anzahl an Merkmalsarten eines Objekts auch die Robustheit und Erkennungsgenauigkeit zunimmt. Ein Ziel dieser Arbeit ist die Ermittlung der Funktionsweise einer solchen Kombination von Merkmalsarten anhand des Beispiels von Farb- und Tiefeninformationen.

Die Klassifizierung der extrahierten Merkmale wird mithilfe einer Support Vector Machine durchgeführt. Zusätzlich wird ermittelt, ob die Verwendung des HSI-Farbraums anstatt des RGB-Farbraums einen Vorteil bei unterschiedlich belichteten Objekten mit sich bringt. Als Trainingsdaten wird eine Auswahl aus dem Washington RGB-D-Datensatz verwendet. Die dort enthaltenen Objekte wurden mithilfe eines Drehtellers aufgenommen. Aus diesem Grund ist es nicht nötig jedes aufgenommene Bild zu verwenden, da sich die Bilder erst ab einem bestimmten Grad der Drehung maßgeblich unterscheiden. Abbildung 4.1 zeigt die ersten fünf Bilder der Objektinstanz „apple_1“ aus diesem Datensatz. Es wird deutlich, dass diese nahezu identisch sind und eine Hinzunahme jedes dieser Bilder zum Training keinen Vorteil bringen,

[1] Die GPU (*engl.* graphics processing unit) bezeichnet den Grafikprozessor einer Grafikkarte.

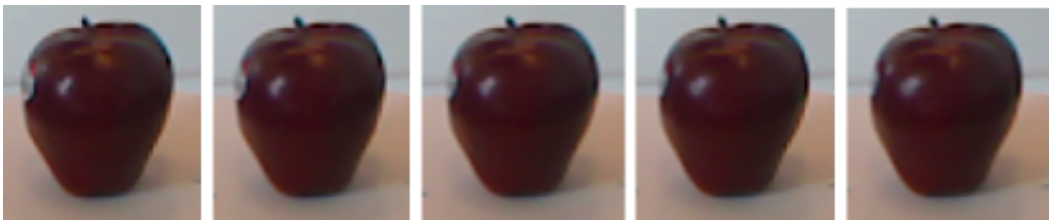


Abbildung 4.1.: Darstellung der ersten fünf Bilder von der Objektinstanz „apple_1“ aus dem Washington RGB-D-Objektdatensatz.

sondern lediglich eine erhöhte Trainingsdauer verursachen würde. Abbildung 4.2 hingegen zeigt das gleiche Objekt mit einem zeitlichen Abstand von jeweils fünfundzwanzig Bildern. Aus

Abbildung 4.2.: Darstellung jedes fünfundzwanzigsten Bildes von der Objektinstanz „apple_1“ aus dem Washington RGB-D-Objektdatensatz.

diesem Grund wird für das Training nur jedes fünfundzwanzigste Bild verwendet. Insgesamt ergeben sich für das Training damit 8278 RGB-D-Aufnahmen. Zusätzlich zum Training werden für die Evaluierung des Konzepts ebenfalls Daten aus dem Washington RGB-D-Objektdatensatz verwendet. Des Weiteren werden auch Daten mit der Kinect One für den gleichen Zweck erzeugt. Im Folgenden wird die Architektur beschrieben, mit der dieses Konzept umgesetzt wurde.

4.2. Architektur

Für die Umsetzung der Architektur werden verschiedene Softwarekomponenten zu einer Pipeline zusammengeschlossen. Die Pipeline wird sowohl für den Trainingsprozess als auch für die spätere Objekterkennung genutzt. Im Folgenden wird zunächst der Aufbau der Pipeline beschrieben. Anschließend erfolgt eine detaillierte Erläuterung ihrer einzelnen Komponenten.

Die Architektur der Pipeline ist in Abbildung 4.3 visualisiert. Insgesamt kann diese in mehrere Abschnitte unterteilt werden. Wie in Abbildung 4.3 dargestellt, werden im ersten Abschnitt die Eingangsdaten empfangen. Dabei handelt es sich um RGB-D-Daten aus einer

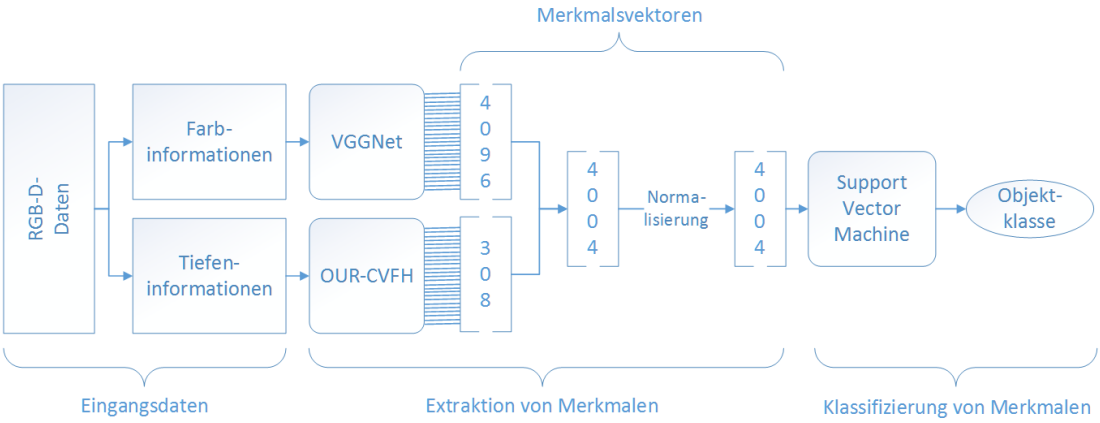


Abbildung 4.3.: Architektur der entworfenen Pipeline zur 3D-Objekterkennung.

beliebigen Datenquelle, die auf zwei unterschiedliche Pfade aufgeteilt und verarbeitet werden. Der obere Pfad dient der Extraktion von Merkmalen aus dem Farbanteil der RGB-D-Daten, während im unteren Pfad die Merkmale aus den Tiefeninformationen generiert werden. Für eine bessere Abgrenzung der einzelnen Aufgaben innerhalb der Pipeline sind die folgenden Abschnitte in die gleichnamigen Abschnitte der Pipeline aus Abbildung 4.3 unterteilt. Abschnitt 4.2.1 beschreibt zunächst den Prozess der Merkmalsextraktion. In diesem Abschnitt wird auch die Kombination der zwei entstehenden Merkmalsarten beschrieben. Anschließend erfolgt in Abschnitt 4.2.2 die Erläuterung der Merkmalsklassifizierung gefolgt vom Training und der Parameteroptimierung in Abschnitt 4.2.3.

4.2.1. Extraktion von Merkmalen

Im oberen Pfad der Pipeline werden die Farbinformationen gegebenenfalls je nach genutztem Farbraum zunächst in diesen konvertiert. Nach der Konvertierung werden die Daten an die Softwarekomponente „VGGNet“ übergeben. Das VGGNet ist ein Faltungsnetzwerk, dass an dieser Stelle genutzt wird, um Merkmale des Objekts aus den Farbinformationen zu extrahieren. Für diesen Zweck wird die original Netzwerkarchitektur verändert, sodass vom VGGNet keine Klassifizierung mehr durchgeführt wird. Um dieses Ziel zu erreichen, wird die letzte Schicht des Netzes entfernt. Demzufolge ist die neue letzte Schicht dann eine Fully-Connected-Schicht

aus dem die erkannten Merkmale entnommen werden können. Da diese Schicht beim VGGNet aus 4096 Neuronen besteht (siehe Abbildung 4.10, Schicht „fc8“) ergeben sich 4096 Merkmale, die wie in Abbildung 4.3 dargestellt, zu einem Merkmalsvektor zusammengestellt werden. Eine detaillierte Beschreibung der Architektur des VGGNet ist in Abschnitt 4.4 zu finden.

Im unteren Pfad der Pipeline werden die Tiefeninformationen genutzt, um mithilfe von OUR-CVFH aus diesen geometrische Eigenschaften des Objekts zu extrahieren. Es handelt sich dabei um ein Verfahren zur Erzeugung eines 3D-Deskriptors aus Tiefeninformationen. Die Tiefeninformationen müssen für diesen Zweck in Form einer Punktwolke vorliegen. OUR-CVFH wurde gewählt, da es sich dabei um einen semi-globalen Ansatz handelt, der die Nachteile von lokalen und globalen Ansätzen ausgleichen soll. Abschnitt 4.3 enthält hierzu Details und beschreibt, um was für geometrische Eigenschaften es sich handelt und wie diese von OUR-CVFH ermittelt werden. Das Ergebnis dieser Ermittlung, wie in Abbildung 4.3 zu sehen, ist ein 308-dimensionaler Merkmalsvektor.

Zur Zusammenführung der Ergebnisse des oberen und des unteren Pfades der Pipeline werden die beiden Merkmalsvektoren vereinigt. Wie in Abbildung 4.3 dargestellt, ergibt sich daraus ein insgesamt 4004-dimensionaler Merkmalsvektor.

Da sich die Wertebereiche der zusammengeführten Vektoren stark unterscheiden, ist eine Normalisierung der Daten notwendig. Während die Merkmale, die vom VGGNet produziert werden, in einem Wertebereich von 0 bis 14 liegen, bewegen sich die Werte von OUR-CVFH in einem Bereich von 0 bis zu mehreren Tausend. Ohne eine Normalisierung würden die Werte von OUR-CVFH bei der Objekterkennung dominieren und so die Einbeziehung beider Merkmalsarten zur Klassifizierung scheitern. Aus diesem Grund erfolgt eine Normalisierung in den Wertebereich $[-1, 1]$, der von Hsu u. a. (2010) für die Verwendung mit einer Support Vector Machine empfohlen wird.

4.2.2. Klassifizierung von Merkmalen

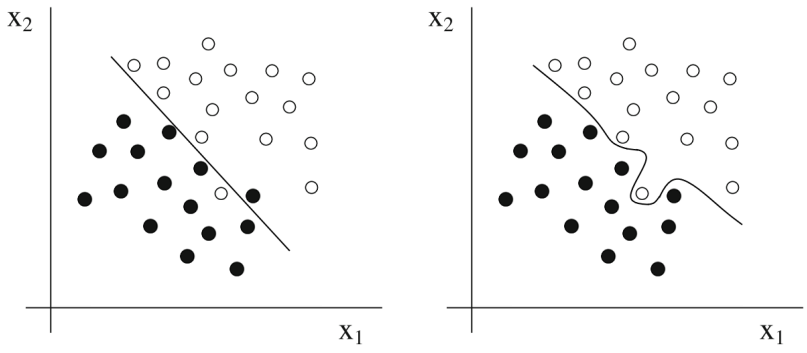
Für das Training und die spätere Klassifizierung der Objekte wird abschließend eine Support Vector Machine genutzt. Der Grund hierfür ist, dass in anderen Arbeiten bereits gezeigt wurde, dass die Klassifizierung von Merkmalen durch eine SVM zu besseren Ergebnissen führt, als die Klassifizierung durch ein Faltungsnetzwerk (vgl. Huang und LeCun (2006), Tang (2013)). An dieser Stelle in der Pipeline ergibt sich ein Unterschied zwischen dem Trainings- und dem späteren Erkennungsprozess (siehe Abschnitt 4.2.3). Abbildung 4.3 zeigt das weitere Vorgehen im Fall einer Objekterkennung. Die Daten werden in Form des Merkmalsvektors von der SVM

klassifiziert und die Ergebnisklasse wird ausgegeben. Das Vorgehen beim Trainingsprozess unterscheidet sich dadurch, dass an dieser Stelle zunächst nach passenden Hyperparametern für die SVM gesucht werden muss. Details zur Ermittlung der Hyperparameter werden in Abschnitt 4.2.3 beschrieben.

Sowohl für Erkennungs- als auch Trainingsprozess ist zunächst die Auswahl eines SVM-Kernels notwendig. Da es sich bei den Daten in dieser Arbeit um stark verschränkte Daten handelt, reicht ein linearer Kernel nicht aus. Laut Hsu u. a. (2010) eignet sich als nicht linearer Kernel in der Regel ein RBF-Kernel am besten, da dieser weniger Hyperparameter als der polynomiale Kernel besitzt und sich mit einer bestimmten Auswahl an Parametern wie ein Sigmoid-Kernel verhält. Aus diesen Gründen wird in dieser Arbeit ein RBF-Kernel verwendet.

Dieser Kernel besitzt den in Abschnitt 3.3 bereits beschriebene C-Parameter sowie als Zweites den Parameter γ. Diese müssen der Datenbasis entsprechend festgelegt werden. Der γ-Parameter bestimmt den Einfluss, den ein einzelnes Trainingsexemplar auf das Finden der Hyperebene hat. Ein Beispiel der Auswirkungen dieses Parameters ist in Abbildung 4.4 zu finden.

Wird ein niedriger Wert für γ gewählt, besitzen Punkte die weit entfernt sind von der Hyperebene stärkeren Einfluss, wodurch sich eine lineare Hyperebene ergibt (siehe Abbildung 4.4.a). Bei der Wahl eines höheren Wertes für γ besitzen Datenpunkte eine größere Gewichtung, umso näher sie sich an der Hyperebene befinden. In diesem Fall kann schon ein einziges Trainingsexemplar einen Ausschlag der Hyperebene hervorrufen, sodass diese einer Kurve ähnelt (siehe Abbildung 4.4.b).



(a) Niedriger Wert des γ-Parameters (b) Hoher Wert des γ-Parameters

Abbildung 4.4.: Vergleich der Trennung von Trainingsdaten mit verschiedenen γ-Parametern.

Im Allgemeinen führen höhere Werte für γ und C zu einer wenig generalisierenden SVM, und demzufolge zu einer Überanpassung. Die Auswahl dieser Parameter ist somit stark für die Leistung der SVM verantwortlich (siehe Abschnitt 3.3). Eine Ermittlung der Parameter ist bei jeder sich ändernden Datenbasis erneut vorzunehmen. Abschnitt 4.2.3 beschreibt das Vorgehen zur Auswahl der Kernel-Parameter.

4.2.3. Training und Parameteroptimierung

Die Auswahl der Parameter für den in dieser Thesis verwendeten RBF-Kernel ist maßgeblich für die Leistung der SVM und somit der gesamten Pipeline. Das Ziel der Suche nach diesen ist die Identifizierung von passenden C und γ Parametern, die sowohl für trainierte als auch bei noch nicht trainierten Daten eine gute Klassifizierung ermöglichen. Zur Ermittlung der möglichen (C,γ)-Parameterkombinationen wird das *grid-search* Verfahren verwendet.

Bei grid-search handelt es sich um die erschöpfende Suche nach Parametern in einem vorher spezifizierten Suchraum. Das Verfahren beruht auf der Idee, in einem Gitter mit einer bestimmten Gitterweite nach dem Minimum beziehungsweise dem Maximum zu suchen. Sind zum Beispiel zwei Parameter zu ermitteln, erfolgt diese Suche in einem zweidimensionalen Gitter.

Im Folgenden wird zunächst die Festlegung der infrage kommenden Parameterkombinationen beschrieben. Als nächsten werden die notwendigen Schritte zum Test dieser Parameterkombinationen erläutert gefolgt vom Auswahlprozess der schlussendlichen Parameter.

Festlegen der Parameterkombinationen

Für die Suche mithilfe von grid-search nach den (C,γ)-Parametern eines RBF-Kernels empfehlen Hsu u. a. (2010) ein Gitter aus exponentiell wachsenden Sequenzen. Aus diesem Grund wird für die Ermittlung von C der Wertebereich $[2^{-5}, 2^{-3}, ..., 2^{15}]$ und für die Ermittlung von γ der Wertebereich $[2^{-15}, 2^{-13}, ..., 2^{3}]$ verwendet.

Test der Parameterkombinationen

Das Suchen und anschließende Testen der Parameter während des Trainings kann schnell zu einer Überanpassung (*engl.* overfitting; siehe Abschnitt 3.1.3) der Support Vector Machine führen. Das Ergebnis dieser Überanpassung ist eine SVM, die nicht generalisiert und so nur noch gute Ergebnisse mit bereits trainierten Daten liefert. Aus diesem Grund wird in dieser

Arbeit eine n-fach Kreuzvalidierung (*engl.* n-fold cross-validation) zum Test der ermittelten Parameterkombinationen verwendet. Bei diesem Verfahren werden die Trainingsdaten in n gleichgroße Datensätze aufgeteilt. Anschließend werden n Durchläufe der Parameterevaluierung durchgeführt, bei denen jeweils $n-1$ der Datensätze als Trainingsdaten dienen und der verbleibende Datensatz für die Evaluierung der gewählten Parameter verwendet wird.

Abbildung 4.5 zeigt diesen Vorgang beispielhaft anhand einer 5-fach Kreuzvalidierung. Wie dort dargestellt werden die Trainingsdaten zunächst in fünf gleichgroße Datensätze

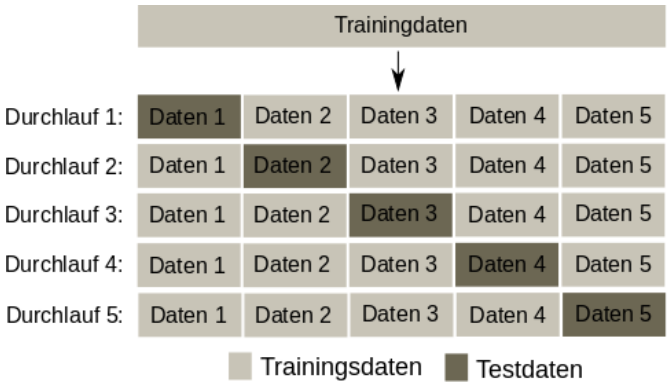


Abbildung 4.5.: Aufteilung von Trainings- und Testdaten bei einer 5-fach Kreuzvalidierung.

aufgeteilt. Anschließend wird in *Durchlauf 1* der Datensatz mit der Bezeichnung „Daten 1“ als Evaluierungsdatensatz ausgewählt. Die verbleibenden Datensätze („Daten 2“ - „Daten 5“) werden als Trainingsdaten gewählt. Die SVM mit den zu evaluierenden (C,γ)-Parametern wird mit den Trainingsdaten trainiert und der Test der Parameter erfolgt infolgedessen mit den ausgewählten Evaluierungsdaten. Dieser Vorgang wird für die verbleibenden Durchläufe wiederholt. Hierbei ändern sich wie in Abbildung 4.5 dargestellt jeweils die Evaluierungs- und Trainingsdaten, die (C,γ)-Parameter verbleiben allerdings, bis alle Durchläufe beendet wurden. Nach Vollendung aller Durchläufe werden die Ergebnisse der fünf Testdurchläufe gemittelt und so die Qualität der Parameter bestimmt. Diese Strategie der Parametersuche beugt einer Überanpassung vor, da die Parameter nicht nur an einen Trainings- und Evaluierungsdatensatz angepasst werden, sondern an mehrere sich ändernde Datensätze.

Auswahl der Parameter

Anders als andere Verfahren zur Parameteroptimierung lässt sich grid-search sehr gut parallelisieren und deckt zusätzlich einen großen Suchraum in unterschiedlichen Richtungen ab. Des

Weiteren beugt eine Kreuzvalidierung einer Überanpassung vor. Aus diesen Gründen erfolgt die Auswahl geeigneter (C,γ)-Parameter mit einer Kombination aus grid-search für die Auswahl der Parameter und einer 5-fach Kreuzvalidierung für das Testen der ausgewählten Parameter. Dementsprechend wird jede mit grid-search ermittelte Parameterkombination anschließend mithilfe der Kreuzvalidierung bewertet. Als Daten für die Kreuzvalidierung werden die in Abschnitt 4.1 bereits beschriebenen Trainingsdaten verwendet. Die Parameter, die am Ende dieses Vorgangs das beste Ergebnis der Kreuzvalidierung liefern, werden anschließend in dieser Arbeit verwendet.

4.3. OUR-CVFH

Das Akronym OUR-CVFH steht für „Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram“ (Aldoma u. a., 2012). Dabei handelt es sich um einen Algorithmus der 3D-Objekterkennung, der es mithilfe von Tiefeninformationen ermöglicht geometrische Eigenschaften eines Objekts zu ermitteln.

Algorithmen zur 3D-Objekterkennung sind zwischen lokalen und globalen Lösungen zu unterscheiden. Lokale Ansätze extrahieren einzelne Schlüsselpunkte der Objektoberfläche und beschreiben anschließend die lokale Geometrie in der Nachbarschaft von diesen in einem Deskriptor. Für eine Erkennung des Objekts werden anschließend ein oder mehrere dieser Deskriptoren zur Identifizierung genutzt. Bei globalen Ansätzen wird hingegen das Objekt in einem einzigen Deskriptor beschrieben, der das gesamte Objekt charakterisiert.

Bei OUR-CVFH handelt es sich um einen semi-globalen Ansatz, der die Nachteile von lokalen Ansätzen (zum Beispiel großer Speicherbedarf und Komplexität durch viele Deskriptoren) und globalen Ansätzen (zum Beispiel teilweise verdeckte Objekte werden nicht erkannt) ausgleichen soll.

Die Hauptidee von OUR-CVFH besteht in der Einführung eines semi-globalen Referenzbezugssystems (*engl.* reference frame). Dieses wird genutzt, um fünf identifizierende Eigenschaften des Objekts zu berechnen. Abbildung 4.6 zeigt die Punktewolke eines Weinglases, in dem ein solches Referenzsystem beispielhaft in Grün eingefärbt wurde. Nach der Bestimmung des Referenzsystems wird im nächsten Schritt der Schwerpunkt von diesem ermittelt. Hierzu wird der Mittelwert aller Punkte des Referenzsystems berechnet. Anschließend werden von jedem Punkt innerhalb des Referenzsystems drei verschiedene Winkel (α, θ, ϕ) in Bezug auf den Schwerpunkt berechnet. Diese stellen die ersten drei Eigenschaften des Objekts dar.

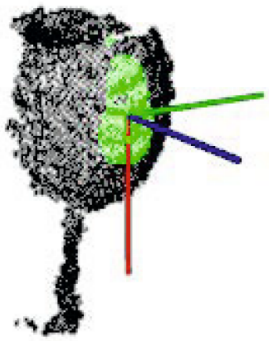


Abbildung 4.6.: Punktwolke eines Weinglases mit in Grün eingefärbtem Referenzbezugssystem (Aldoma u. a., 2012).

Mithilfe dieser Winkel ist es möglich die Objektgeometrie zu beschreiben. Abbildung 4.7 zeigt beispielhaft die Berechnung dieser Winkel zwischen dem Punkt p_5 und dem Schwerpunkt c. Grundlage für die Berechnungen ist ein Koordinatensystem, dass anhand des Schwerpunktes berechnet wird. Für die Berechnung des Koordinatensystems bildet dabei der Normalvektor n_c des Schwerpunktes eine Grundvoraussetzung. Mit diesem können die zwei anderen Achsen w und v berechnet werden. Das berechnete Koordinatensystem wird folgend vom Schwerpunkt c auf den Punkt p_5 übertragen, um anschließend die α-, θ- und ϕ-Winkel (siehe Abbildung 4.7) zu berechnet.

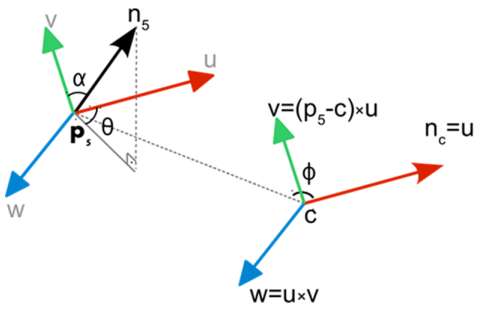


Abbildung 4.7.: Winkelberechnung zur Beschreibung der Objektgeometrie durch OUR-CVFH (Rusu u. a., 2010).

Für die Beschreibung des Blickwinkels, bei dem es sich um die vierte Eigenschaft des Objekts handelt, wird ebenfalls eine Winkeldifferenz berechnet. Ausgangspunkt für diesen Schritt ist allerdings nicht der Schwerpunkt, sondern der Standpunkt der Kamera. Ausgehend von diesem wird analog zum vorher beschriebenen Vorgehen auch hier zu jedem Punkt im Referenzsystem

ein Winkel berechnet. Der Unterschied besteht allerdings darin, dass an dieser Stelle nur der α-Winkel berechnet wird und die Winkel θ und ϕ entfallen.

Als abschließende, und fünfte Eigenschaft, wird zusätzlich die spatiale Verteilung der Punkte ausgehend vom Koordinatensystem des Referenzsystems angegeben. Hierfür werden Oktanten genutzt. Abbildung 4.8 zeigt beispielhaft die Aufteilung eines dreidimensionalen Koordinatensystems in Oktanten.

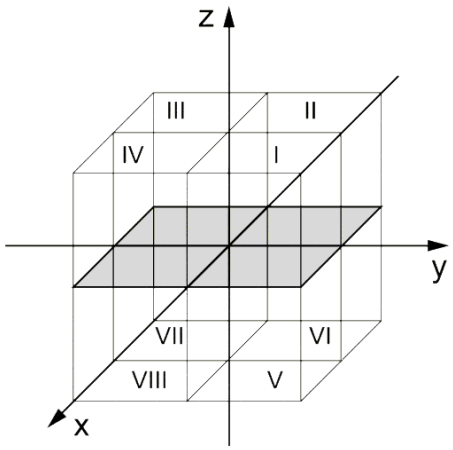


Abbildung 4.8.: Darstellung der räumlichen Aufteilung eines dreidimensionalen Koordinatensystems in Oktanten.

Alle ermittelten Eigenschaften werden als Ergebnis in einem wie in Abbildung 4.9 dargestelltem Histogramm codiert. Während für die drei Winkel α-, θ- und ϕ hierfür jeweils 45 Histogramm-Klassen zur Verfügung stehen, wird der Blickwinkel in 64 und die spatiale Verteilung in 104 Klasse codiert. Insgesamt besteht das Histogramm aus 303 Klassen. Beispielhaft für den α-Winkel bedeutet dies, dass ausgehend von 360° und 45 Klassen, jede Klasse einen Bereich von 8° (360 / 45 = 8) umfasst. In die einzelnen Klassen wird anschließend die Anzahl von Punkten der Punktwolke codiert, die sich in dem jeweiligen Bereich befindet.

4.4. VGGNet-Faltungsnetzwerk

Das VGGNet-Faltungsnetzwerk wurde von Karen Simonyan und Andrew Zisserman von der Visual Geometry Group (VGG) der Universität Oxford entwickelt. Die Entwicklung wurde im Rahmen des ILSVRC-Wettbewerbs 2014 durchgeführt. Alle Ergebnisse wurden in der Arbeit mit

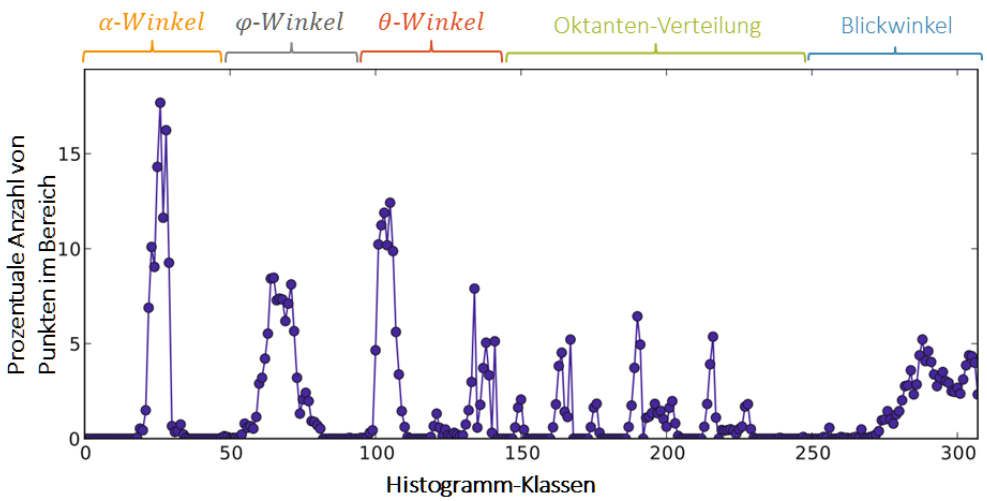


Abbildung 4.9.: Objektgeometrie, Objektlage sowie spatiale Punktverteilung in Form eines Histogramms als Ergebnis von OUR-CVFH (Aldoma u. a., 2012).

dem Titel „Very Deep Convolutional Networks for Large-Scale Image Recognition“ (Simonyan und Zisserman, 2014) veröffentlicht. In dieser Arbeit wird der Effekt auf die Erkennungsgenauigkeit bei steigender Anzahl von Schichten in einem Faltungsnetzwerk untersucht.

Die Autoren haben gezeigt, dass bei steigender Anzahl von Schichten eine signifikante Verbesserung der Objekterkennung erreicht werden kann. Dazu verwenden sie sehr kleine Convolution-Filter (3 x 3) und steigern die Tiefe des Faltungsnetzwerks auf bis zu 19 gewichtete Schichten. Für eine Auswertung wurden von den Autoren sechs Faltungsnetzwerke mit unterschiedlicher Tiefe (unterschiedliche Anzahl von Schichten) implementiert und getestet.

Das beste Ergebnis, und damit der zweite Platz des ILSVRC-Wettbewerbs 2014, wurde erzielt mit einem VGGNet-Faltungsnetzwerk, das 140 Millionen Parameter und 19 Schichten umfasst. Abbildung 4.10 zeigt die Originalarchitektur des VGGNet. Neben der Anordnung der Schichten wird dort zusätzlich die Dimension der Eingangs- und Ausgangsdaten bei jeder Schicht mit angegeben. Neben der Verbesserung der Objekterkennung und Steigerung der Netzwerktiefe haben die Autoren zusätzlich gezeigt, dass das VGGNet eine sehr gute Generalisierung aufweist. Diese Eigenschaft macht das Netzwerk ideal für den Prozess des Transferlernens oder das Extrahieren von Merkmalen anderer, nicht trainierter, Daten.

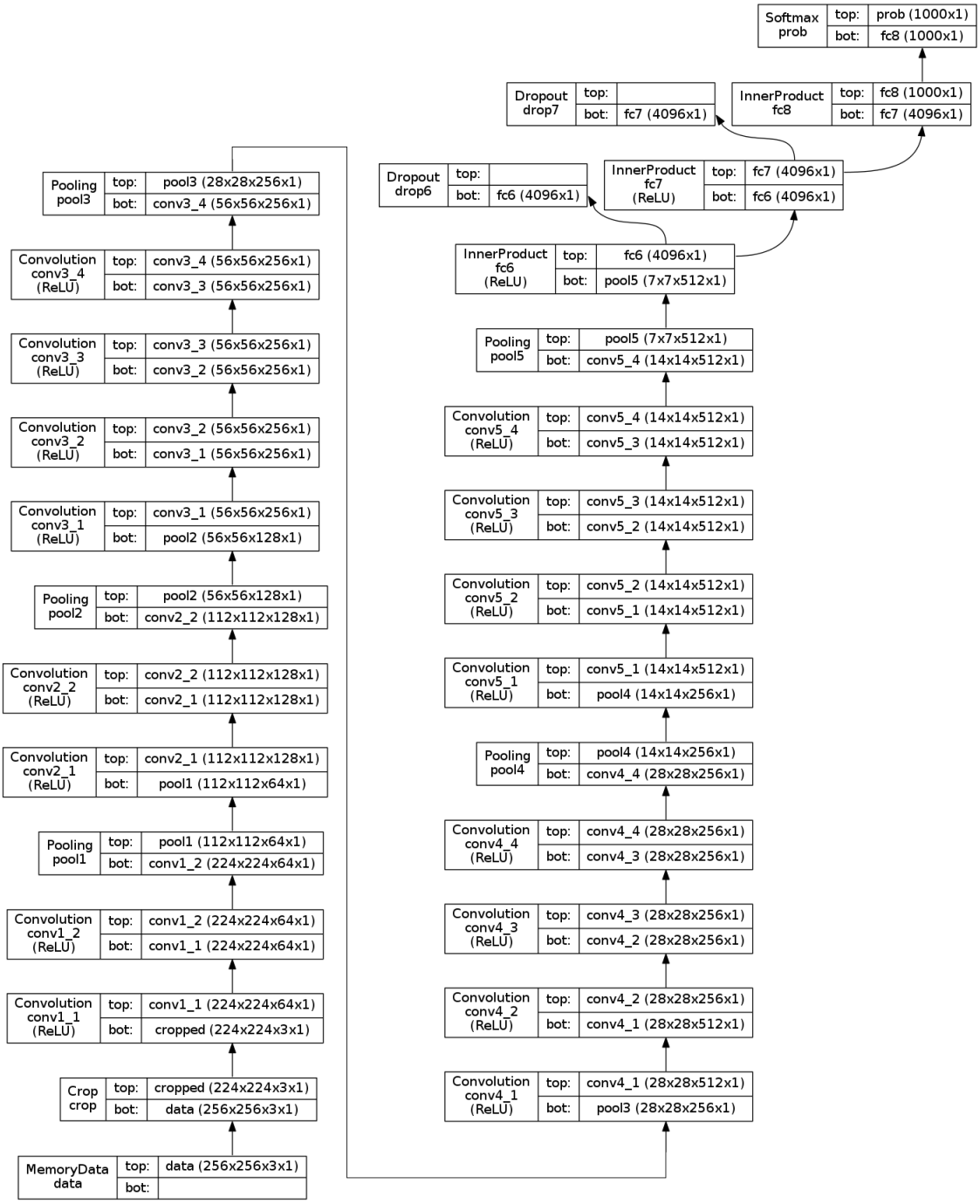


Abbildung 4.10.: Netzwerkarchitektur des VGGNet-Faltungsnetzwerks.

5. Realisierung

In diesem Kapitel wird die Umsetzung des entworfenen Konzepts beschrieben. Zunächst erfolgt eine Beschreibung der verwendeten Frameworks und Bibliotheken, mit denen verschiedene Teile dieser Arbeit praktisch umgesetzt wurden. Im Anschluss daran werden das umgesetzte System sowie dessen einzelne Komponenten detailliert behandelt. An dieser Stelle wird neben Implementierungsdetails auch die Umsetzung des Gesamtsystems in Form einer Objekterkennungs-Pipeline erläutert. Abschließend erfolgt eine Erläuterung von Details, die für das Training und den Test mit dem System zu beachten sind.

5.1. Verwendete Software

Für die Umsetzung der Softwarekomponenten wurden verschiedene Frameworks und Bibliotheken genutzt. Die Implementierung OUR-CVFH erfolgt mithilfe der Point Cloud Library. Außerdem wurde Caffe als Deep-Learning-Framework verwendet, um die Faltungsnetzwerachitektur des VGGNet zu implementieren und vortrainierte Gewichtungen zu laden. Abschließend wurde zur Realisierung der Support Vector Machine die Bibliothek LIBSVM genutzt. Im Folgenden werden die genutzten Frameworks und Bibliotheken erläutert.

5.1.1. Point Cloud Library

Die Point Cloud Library (PCL) (Rusu und Cousins, 2011) ist eine Bibliothek für die Verarbeitung von n-dimensionalen Punktwolken und dreidimensionalen Geometrien, die von Willow Garage entwickelt wurde. Dazu zählen unter anderem Verfahren zur Filterung, Merkmalsextraktion oder der Segmentierung von Objekten. Das Framework ist in C++ geschrieben und legt besonderen Wert auf hohe Geschwindigkeit. Um schnelle I/O-Operationen zu gewährleisten, wurde ein eigenes Format zur Speicherung von Punktwolken entwickelt. Es handelt sich dabei um das Point Cloud Data (PCD) Format. Zusätzlich werden aber auch alle anderen gängigen

Formate unterstützt. Des Weiteren stellt die PCL auch Möglichkeiten zur Visualisierung von Punktwolken bereit.

5.1.2. Caffe

Bei Caffe (Jia u. a., 2014) handelt es sich um ein Deep-Learning-Framework, dass vom Berkley Vision and Learning Center (BLVC) entwickelt wurde. Caffe ist in C++ geschrieben und beinhaltet unterstützende Python- und Matlab-Wrapper. Neben einer Unterstützung für die Ausführung und Berechnung entwickelter Architekturen auf der GPU ist es ebenfalls möglich alle Berechnungen von der CPU durchführen zu lassen. Der Wechsel zwischen GPU- und CPU-Berechnung erfolgt dabei simpel über eine Variable.

Für die Erstellung einer Faltungsnetzwerkarchitektur ist bei Caffe kein Programmieraufwand nötig. Die Konfiguration der zu erstellenden Netzwerkarchitektur erfolgt über eine Konfigurationsdatei. In dieser müssen die zu erstellenden Schichten sowie alle benötigten Hyperparameter angegeben werden. Die Konfigurationsdatei wird im Datenformat protobuf[1] gespeichert.

Eine Besonderheit von Caffe ist es, trainierte Netzwerke (das heißt Gewichtungen) als Modell zu speichern. Mit diesem Modell und der passenden protobuf-Konfigurationsdatei ist es anschließend möglich bereits trainierte Netze weiterzugeben und sogar mit diesen und neuen Daten das Training fortzusetzen. Die Caffe-Entwickler stellen auf einer Online-Plattform, dem sogenannten *Caffe Model Zoo*[2], beliebte Faltungsnetze sowie deren Gewichtungen und allem zum Betreiben des Netzwerks notwendigen Daten zum Herunterladen bereit.

5.1.3. LIBSVM

LIBSVM (Chang und Lin, 2011) ist eine Open-Source Bibliothek für maschinelles Lernen, die von der Nationaluniversität Taiwan entwickelt wurde. Die Bibliothek wurde in C++ geschrieben und stellt Algorithmen für Support Vector Machinen zum Zweck der Klassifizierung und der Regressionsberechnung bereit. Hierfür kann aus verschiedenen Kerneln gewählt werden. Für die Umsetzung einer SVM für mehr als zwei Klassen wird intern das One-versus-One-Verfahren genutzt, das bereits in Abschnitt 3.3 beschrieben wurde. Zusätzlich werden Werkzeuge zur Hyperparametersuche bereitgestellt. Die Entwickler legen dabei besonderen Wert darauf eine einfache Schnittstelle bereitzustellen, um es dem Benutzer zu ermöglichen LIBSVM in sein Projekt zu integrieren.

[1] Protocol Buffers (protobuf): Ein von Google entwickeltes Dateiformat zur Serialisierung.
[2] https://github.com/BVLC/caffe/wiki/Model-Zoo ; Zugriffsdatum: 02.05.2016

5.2. System

Zur Umsetzung der in Abschnitt 4.2 beschriebenen Architektur werden drei unterschiedliche Softwarekomponenten benötigt. Dabei handelt es sich neben den zwei Komponenten zur Extraktion von Merkmalen, VGGNet und OUR-CVFH, um die Support Vector Machine zur Klassifizierung der Merkmale. Zusätzlich wird eine Softwarekomponente entworfen, die diese zu einer Objekterkennungs-Pipeline vereint. Des Weiteren wurde für eine vereinfachte Bedienung der Pipeline ein Frontend entwickelt. Abbildung 5.1 zeigt ein Komponentendiagramm dieses Gesamtsystem.

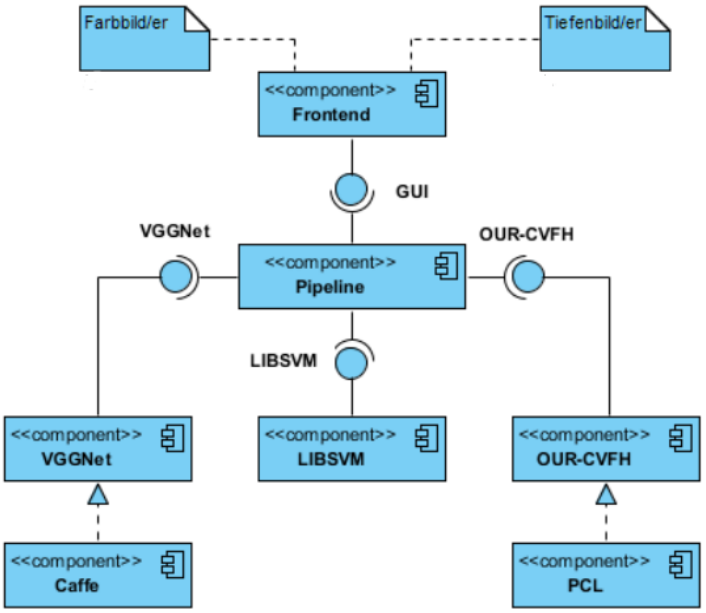


Abbildung 5.1.: Komponentendiagramm der eigenen Architektur.

Wie dort dargestellt nutzt die Komponenten „Pipeline“ Schnittstellen, die von den Komponenten „VGGNet“ und „OUR-CVFH“ bereitgestellt werden. Beide Komponenten realisieren wiederum Funktionen, die von den Komponenten „Caffe“ und „PCL“ zur Verfügung gestellt werden. Außerdem verwendet die Pipeline zusätzlich Schnittstellen der Komponente „LIBSVM“ für die Umsetzung der SVM. Für eine vereinfachte Bedienung werden des Weiteren Schnittstellen von der Pipeline für ein Frontend bereitgestellt.

Die Komponenten „Frontend“ wurde unter Verwendung von Python und PyQt[3] entwickelt. Die Mindestvoraussetzung für die Nutzung der gesamten Pipeline ist ein vollständiges RGB-D-Datum in Form eines Farbbilds mit dazu passenden Tiefeninformationen als Punktwolke.

Im Folgenden werden die Kernschritte erläutert, die zur Umsetzung der einzelnen Komponenten und des Gesamtsystems in Form der Objekterkennungs-Pipeline notwendig waren.

5.2.1. VGGNet

Die Umsetzung des VGGNet wurde mithilfe des Deep-Learning-Frameworks Caffe (siehe Abschnitt 5.1.2) durchgeführt. Das Framework stellt eine Python-API bereit, die in dieser Arbeit verwendet wird. Für die Realisierung der VGGNet-Architektur ist diese zunächst in einem Modell in Form einer Caffe-Konfigurationsdatei nachzubilden. In dieser Konfigurationsdatei müssen alle Schichten der Architektur sowie die Eigenschaften der einzelnen Schichten angegeben werden. Listing 5.1 zeigt dies beispielhaft für die erste Convolutional-Schicht des VGGNet.

```
1  layers {
2    bottom: "conv1_1"
3    top: "conv1_2"
4    name: "conv1_2"
5    type: CONVOLUTION
6    convolution_param {
7      num_output: 64
8      pad: 1
9      kernel_size: 3
10   }
11 }
```

Listing 5.1: Protobuf-Definition der ersten Convolutional-Schicht des VGGNet

Wie in Zeile 1 dargestellt wird eine Schichtdefinition zunächst mit dem Schlüsselwort „layers“ eingeleitet. Anschließend werden in Zeile 2 die eingehenden Daten und in Zeile 3 die ausgehenden Daten dieser Schicht definiert. Nach der Angabe des Schichttyps in Zeile 5 erfolgt abschließend in Zeile 6-10 die Definition der Hyperparameter der gewählten Schicht. Nach diesem Muster wurden alle Schichten des VGGNet nach der Vorgabe der Originalarbeit von Simonyan und Zisserman (2014) definiert. Eine Ausnahme hiervon stellen die zwei letzten Schichten des

[3] PyQt ist eine Sprachanbindung von Python für die C++-Bibliothek Qt zur Erstellung von grafischen Oberflächen.

Netzes dar, die für die Klassifizierung verwendete werden. Diese werden in dieser Arbeit nicht benötigt und aus diesem Grund auch nicht definiert.

Wie in Abschnitt 4.1 bereits beschrieben werden für das VGGNet vortrainierte Gewichtungen geladen. Die trainierten Gewichtungen können aus dem Caffe Model Zoo bezogen werden. Eine Voraussetzung für den erfolgreichen Import der Gewichtungen in das Netz ist die Namensgebung der Schichten. Die Gewichtungen können der jeweiligen Schicht nur bei gleichem Namen zugeordnet werden. Aus diesem Grund entsprechen die Namen der Schichten exakt den Namen aus der Originalarbeit.

Die Extraktion der Merkmale mit dem Netzwerk erfolgt unter Verwendung von Python. Für diesen Vorgang muss zunächst das VGGNet definiert werden. Hierfür werden die Konfigurationsdatei sowie die Gewichtungen geladen und der Modus des Netzes bestimmt. Bei der Auswahl des Modus kann zwischen Trainings- und Testmodus gewählt werden. Der Trainingsmodus verwendet im Gegensatz zum Testmodus Dropout (siehe Abschnitt 3.2.1) und würde so bei dieser Art der Merkmalsextraktion gegebenenfalls unterschiedliche Ergebnisse bei gleichen Bildern liefern. Aus diesem Grund und weil kein weiteres Training durchzuführen ist, wird an dieser Stelle der Testmodus verwendet.

Listing 5.2 zeigt einen Auszug der Merkmalsextraktion mit Python. Die Definition des Netzes ist in Zeile 4 zu finden.

```
1 ...
2 model_def = '<Pfad zu>/vgg_architektur.prototxt'
3 model_weights = '<Pfad zu>/VGG_Gewichtungen.caffemodel'
4 vggNet = caffe.Net(model_def, model_weights, caffe.TEST)
5 ...
6 merkmale = vggNet.blobs['fc7'].data[0]
7 ...
```

Listing 5.2: Auszug der Merkmalsextraktion mit Python unter Caffe.

Die eigentliche Extraktion erfolgt anschließend, wie in Zeile 6 von Listing 5.2 definiert. Die Merkmale können unter Angabe der Schicht aus der sie entnommen werden sollen extrahiert werden. In dieser Arbeit handelt es sich um die Schicht „fc7“ (siehe Abbildung 4.10). Die Merkmale liegen dann in Form eines Arrays vor und können weiterverarbeitet werden. Zur späteren Verwendung der Merkmale mit der SVM werden die Daten an dieser Stelle noch dem SVM-Format angepasst (siehe Listing 5.4).

5.2.2. OUR-CVFH

Die Umsetzung von OUR-CVFH erfolgt unter Verwendung der Point Cloud Library (siehe Abschnitt 5.1.1). Die Implementierung erfolgt in C++ und es werden hierfür die von der Bibliothek bereitgestellten Funktionen genutzt. Da die Tiefeninformationen des Washington RGB-D-Objektdatensatzes bereits im PCL eigenen Format vorliegen, können diese ohne Probleme geladen werden.

Im ersten Schritt der Umsetzung von OUR-CVFH müssen zunächst die Oberflächen-Normalen der Punktewolke bestimmt werden. Für die Bestimmung von diesen sind zwei Parameter zu setzen. Dabei handelt es sich zum einen um den Radius, in dem Nachbarpunkte mit einbezogen werden sollen und zum anderen um die Methode mithilfe von der gesucht werden soll. In dieser Arbeit wurden hierfür die Empfehlungen der PCL-Bibliothek verwendet. Zeile 2-6 zeigen die Kernzeilen, die für die Ermittlung der Oberflächen-Normalen mithilfe der PCL notwendig sind.

Anschließend können mit den Oberflächen-Normalen die Merkmale von OUR-CVFH bestimmt werden. Auch diese Funktion wird von der PCL bereitgestellt, sodass ein Aufruf der passenden Funktion genügt, um als Rückgabewert einen Deskriptor mit den Ergebnissen von OUR-CVFH zu erhalten. Listing 5.3 zeigt in Zeile 8-13 einen Auszug des Codes zur Berechnung von OUR-CVFH.

```
1  ...
2  pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> normalEstimation;
3  normalEstimation.setInputCloud(object);
4  normalEstimation.setRadiusSearch(0.03);
5  normalEstimation.setSearchMethod(kdtree);
6  normalEstimation.compute(*normals);
7  ...
8  pcl::OURCVFHEstimation<pcl::PointXYZ, pcl::Normal,
9             pcl::VFHSignature308> ourcvfh;
10 ourcvfh.setInputCloud(object);
11 ourcvfh.setInputNormals(normals);
12 ourcvfh.setSearchMethod(kdtree);
13 ourcvfh.compute(*descriptor);
14 ...
```

Listing 5.3: Auszug der Merkmalsextraktion mit OUR-CVFH unter Verwendung der Point Cloud Library.

Als Rückgabetyp für den Deskriptor wird <pcl::VFHSignature308> verwendet. Der Grund hierfür ist, dass in einem Fall bei dem keine Cluster von OUR-CVFH in den Tiefeninformationen gefunden werden kann, stattdessen der VFH-Algorithmus verwendet wird, um in jedem Fall eine Beschreibung des Objekts zu erhalten. Auch die hier extrahierten Merkmale werden an dieser Stelle in das Format der SVM konvertiert (siehe Listing 5.4).

5.2.3. SVM

Die Support Vector Machine zur Klassifizierung der extrahierten Merkmale wird mithilfe der Bibliothek LIBSVM (siehe Abschnitt 5.1.3) realisiert. Die Bibliothek stellt Funktionen für das Training und die Klassifizierung mit einer SVM zur Verfügung. Des Weiteren werden auch Funktionen für die Normalisierung von Daten sowie für grid-search bereitgestellt. Die Daten müssen in einem SVM kompatiblen Format vorliegen, um ein Training oder eine Erkennung durchführen zu können. Listing 5.4 zeigt den Ausschnitt einer Datei in einem solchen Format. Zeile 1 zeigt den Anfang eines neuen Merkmals. An erster Stelle steht zunächst die Klasse des Merkmals. In diesem Beispiel handelt es sich dabei um Klasse 1, die wiederum dem Objekt „apple“ entspricht. Der Klassenbezeichnung folgt anschließend eine Kombination aus Merkmalsnummer sowie Merkmalswert. Im Beispiel von Zeile 3 des Listings wäre das für das erste Paar das Merkmal mit der Nummer 17 und dem Merkmalswert -0.658488.

```
1 1 1:-1 2:-0.718358 3:-1 4:-1 5:-1 6:-0.246187 7:-0.98715 8:-1
2 9:-1 10:-0.741852 11:-1 12:-1 13:-1 14:-0.812463 15:-1 16:-1
3 17:-0.658488 18:-1 19:-0.852026 20:-1 22:-0.823795 22:-0.907099 ...
```

Listing 5.4: Auszug einer Merkmalsbeschreibung im LIBSVM-Format.

Vor der Verarbeitung durch die SVM müssen die Merkmalswerte zunächst normalisiert werden. Für die Normalisierung erfolgt die Ermittlung der obersten (d_{og}) sowie der untersten Grenze (d_{ug}) pro Merkmal innerhalb des Wertebereichs aus den Trainingsdaten. Die Normalisierung erfolgt anschließend für jedes Datums x_i nach Gleichung 5.1 in einen Wertebereich mit einem minimalen Wert von -1 (w_{min}) und einem maximalen Wert von 1 (w_{max}).

$$w_{min} + \frac{(x_i - d_{ug})(w_{max} - w_{min})}{(d_{og} - d_{ug})} \tag{5.1}$$

Die Suche der (C,γ)-Hyperparameter der SVM erfolgt mithilfe des grid-search Verfahrens, für das LIBSVM eine Schnittstelle zur Verfügung stellt. Verwendet werden hierfür die in Abschnitt 4.2.3 bereits beschriebenen Wertebereiche.

5.2.4. Pipeline

Zur Verbindung der entwickelten Softwarekomponenten und Automatisierung des Trainings- und Erkennungsprozesses wurde eine Pipeline entwickelt. Die Implementierung wurde in Python vorgenommen. Zusätzlich wurde ein Frontend für eine benutzerfreundliche Bedienung der Pipeline entwickelt. Abbildung 5.2 zeigt dieses Frontend.

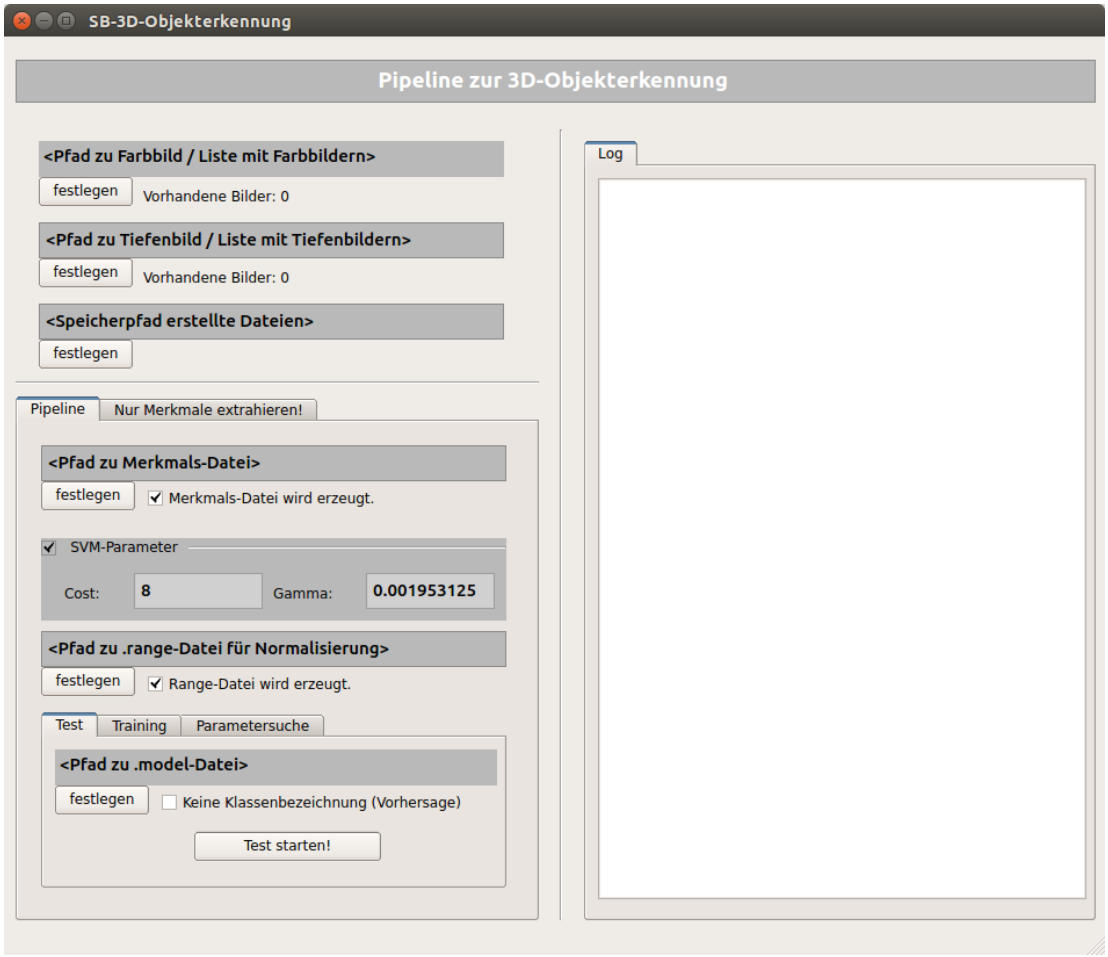


Abbildung 5.2.: Frontend der Pipeline.

Die Pipeline bildet das Gesamtsystem von der Merkmalsextraktion bis hin zur Klassifizierung ab. Grundvoraussetzung für die Nutzung des Gesamtsystems ist mindestens ein Farbbild mit dem dazugehörigen Tiefenbild in Form einer Punktwolke. Unabhängig von Menge der Daten und dem Modus der Pipeline müssen diese in Form von zwei getrennten Listen angegebenen werden. Das Format der zwei Listen unterscheidet sich. Während für die Tiefeninformationen

lediglich der Pfad zur Punktwolke benötigt wird, muss die Liste der Farbbilder im Format *<Pfad zu Objekt> <Name des Objekts> <Klasse des Objekts>* vorliegen. Listing 5.5 zeigt in Zeile 1 ein Beispiel für das Format der Tiefeninformationen und in Zeile 2 für das Format der Farbinformationen.

```
1 /<Pfad zu Objekt>/apple_1_1_122.pcd
2 /<Pfad zu Objekt>/ apple_1_1_122_.png 1
```

Listing 5.5: Beispiel des benötigten Formats der Daten-Listen zur Verwendung mit der Pipeline.

Des Weiteren ist es möglich die Komponenten einzeln zu verwenden, sodass Merkmalsextraktion und Klassifizierung getrennt voneinander durchgeführt werden können. Alle benötigten Hyperparameter können entweder vom Nutzer selbst definiert oder automatisch ermittelt werden. Zusätzlich erfolgt eine Trennung zwischen Trainings-, Test- und Erkennungsprozess.

Bei einem Testprozess liegen zu den Testdaten bereits die korrekten Klassenbezeichnungen vor. Dieser dient also dazu, die Qualität der produzierten Ergebnisse zu messen. Im Unterschied dazu sind bei einem Erkennungsprozess die Klassenbezeichnungen der Daten unbekannt und werden infolgedessen prognostiziert.

5.3. Training und Test

Das Training sollte mit mindestens 1000 Daten durchgeführt werden, um eine Tendenz der Leistungsfähigkeit deuten zu können. Zusätzlich ist darauf zu achten, dass sowohl Trainings- als auch Testdaten in den gleichen Bereich normalisiert werden. Hierfür wird von der Pipeline eine sogenannte *.range*-Datei bereitgestellt. Diese wird beim Training erzeugt und spiegelt, wie in Listing 5.6 zu sehen, für jedes Merkmal den maximal und den minimal angenommen Wert innerhalb der Trainingsdaten vor der Normalisierung wieder.

```
1 x
2 -1 1
3 1 0 10.2757492065
4 2 0 5.4382019043
5 3 0 5.13217735291
6 4 0 4.77635908127
7 ...
```

Listing 5.6: Auszug einer .range-Datei im LIBSVM-Format.

Ein Test mit Daten ohne Normalisierung auf den Wertebereich der .range-Datei führt zu einer signifikanten Verschlechterung der Ergebnisse. Aus diesem Grund ist eine Normalisierung für die Nutzung der Pipeline erforderlich.

Nach einem Training wird von der Pipeline eine *.model*-Datei erzeugt, die eine Repräsentation der trainierten SVM darstellt. In dieser Datei sind neben dem Typ der SVM, der Anzahl an erlernten Klassen und dem verwendeten Kernel unter anderem auch die Supportvektoren für jede einzelne Klasse festgehalten. Für einen Test der Klassifizierung beziehungsweise eine Objekterkennung ist lediglich diese Datei sowie eine Liste mit den zu testenden Objekten erforderlich.

6. Diskussion der Ergebnisse

In diesem Kapitel werden die experimentellen Ergebnisse der umgesetzten 3D-Objekterkennung vorgestellt und ausgewertet. Zur Auswertung werden zunächst die ermittelten Hyperparameter der Support Vektor Machine vorgestellt. Im Anschluss daran erfolgt die Darstellung und Diskussion der Ergebnisse der 3D-Objekterkennung. Hierfür werden verschiedene Testarten definiert, die unterschiedliche Aspekte des entwickelten Verfahrens untersuchen. Des Weiteren wird evaluiert, welchen Mehrwert die Kombination aus Tiefen- und Farbinformationen gegenüber reinen Farbinformationen bei der Objekterkennung darstellt. Weiterhin wird der Einfluss der Belichtung auf den HSI- und RGB-Farbraum untersucht. Abschließend erfolgt einen Gegenüberstellung mit vergleichbaren Arbeiten sowie eine Auswertung der Laufzeit des entwickelten Ansatzes.

6.1. SVM-Hyperparameter

Die Hyperparametersuche der SVM wurde, wie in Abschnitt 4.2.3 bereits beschrieben, mithilfe von grid-search durchgeführt. Der Wertebereich wurde für den Parameter C auf $[2^{-5}, 2^{-3}, ..., 2^{15}]$ und für den Parameter γ auf $[2^{-15}, 2^{-13}, ..., 2^{3}]$ festlegt. Der Test der Parameterkombinationen wurde mit einer 5-fach Kreuzvalidierung duchgeführt. Tabelle 6.1 zeigt die fünf besten Parameter sowie deren Ergebnisse der Kreuzvalidierung in Form der Erkennungsrate. Als Erkennungsrate wird an dieser Stelle die prozentuale Anzahl richtig klassifizierter Objekte bezeichnet. Für eine bessere Darstellung werden die Parameter als binäre Logarithmen angegeben.

Insgesamt wurden 110-Parameterpaare getestet. Das Training und dementsprechend die Kreuzvalidierung wurde mit 8278 Objekten aus dem Washington RGB-D-Objektdatensatz durchgeführt. Während die besten drei Ergebnisse als Resultat der Kreuzvalidierung alle eine Erkennungsrate von 99.6443% erreichen, erzielt das schlechteste Ergebnis einen Wert von 5.4452%. Dieser Unterschied verdeutlicht wie maßgeblich die Hyperparameter zur Leistung der SVM beitragen.

$log_2(C)$	$log_2(\gamma)$	Erkennungsrate (%)
3	-9	99.6443
5	-9	99.6443
7	-9	99.6443
5	-11	99.583
9	-11	99.583

Tabelle 6.1.: Die besten fünf Ergebnisse der (C,γ)-Hyperparametersuche mit 5-fach Kreuzvalidierung.

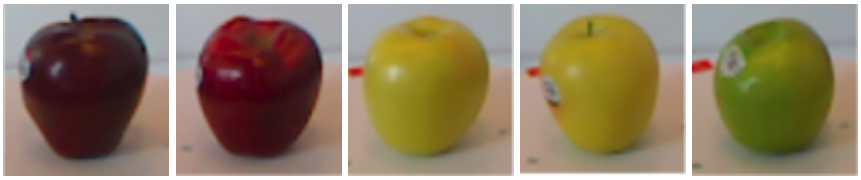
Auffällig bei den Ergebnissen ist vor allem, dass für den γ-Parameter ein Wert im letzten Drittel ($[2^{-9}, ..., 2^{-15}]$) des Wertebereichs ermittelt wurde. Zusätzlich scheint der γ-Parameter einen stärkeren Einfluss auf das Ergebnis zu besitzen. Die ausgewählten C-Parameter dagegen befinden sich eher im mittleren Teil des Wertebereichs ($[2^3, ..., 2^9]$). Ergebnisse, die sich noch in den besten zehn befinden, tendieren allerdings auch zu größeren C-Parametern wie $log_2(11)$ und $log_2(13)$ mit einem γ-Wert von $log_2(11)$.

Da es sich bei den ermittelten γ-Hyperparametern um niedrige Werte innerhalb des Wertebeichs handelt deutet dies auf eine gute Trennung der einzelnen Kategorie und eine lineare Hyperebene hin. Als Folge dessen ergibt sich mit den gefundenen Parametern eine gut generalisierende SVM. Für die weitere Verwendung in dieser Arbeit wird für das C-Parameter der Wert 8 (2^3) und für das γ-Parameter der Wert 0.001953125 (2^{-9}) verwendet.

6.2. Erzielte Ergebnisse

Für eine Bewertung des entwickelten Ansatzes wurden mehrere unterschiedliche Tests durchgeführt. Im Folgenden wird eine Objektklasse als Oberbegriff für eine Gruppe von Objekten des gleichen Typs sowie eine Objektinstanz als einzelne Instanz dieser Gruppe definiert. Abbildung 6.1 zeigt dies beispielhaft für die Objektgruppe „apple“ des Washington RGB-D-Datensatzes. Diese beinhaltet die fünf Objektinstanzen „apple_1“ bis „apple_5“. Jede dieser Objektinstanzen stellt einen anderen Apfel dar und enthält mehrere Aufnahmen aus verschiedenen Winkeln einer Rundumsicht der Instanz.

Zusätzlich zu Tests mit dem Washington RGB-D-Datensatz wird auch mit der Kinect One ein Test durchgeführt. Bei den durchgeführten Tests wird zwischen einer Instanzen- und einer Kategorieerkennung unterschieden. Folgend werden die Bedingungen jedes durchgeführten Tests definiert und die Ergebnisse vorgestellt.



(a) apple_1 (b) apple_2 (c) apple_3 (d) apple_4 (e) apple_5

Abbildung 6.1.: Darstellung einer Beispielaufnahme von allen fünf Objektinstanzen der Objektgruppe „apple“ aus dem Washington RGB-D-Objektdatensatz.

6.2.1. Objekterkennung

Für den Test mit dem Washington RGB-D-Objektdatensatz wurde ein Training mit 8278 RGB-D-Daten vorgenommen. Die Auswahl der Daten erfolgte, wie in Abschnitt 4.1 bereits beschrieben. Das Training umfasst demzufolge alle Instanzen der 51 Objektklassen. Für jede Instanz werden im Durchschnitt 23 RGB-D-Aufnahmen für das Training verwendet. Abhängig ist dieser Durchschnittswert von der Anzahl an insgesamt aufgenommenen RGB-D-Daten.

Die Testdaten umfassen insgesamt 9918 RGB-D-Daten. Hierfür wurde jede zwanzigste Aufnahme aus dem Datensatz verwendet und anschließend alle Aufnahmen entfernt, die bereits im Training enthalten waren. Zusätzlich wurde Wert darauf gelegt, dass es sich bei keinem Testdatum um eine Vorgänger- oder Nachfolgeraufnahme mit weniger als drei Aufnahmen Abstand zu einem Trainingsdatum handelt. Insgesamt werden im Durchschnitt 30 Aufnahmen jeder Objektinstanz für die Testdurchführung verwendet.

Instanzenerkennung

Für einen Test der Instanzenerkennung werden alle Objektinstanzen trainiert, um anschließend die Erkennung derselben Instanz aus einem anderen, nicht trainierten Winkel zu testen. Zur Umsetzung dieses Testszenarios wird zunächst das Training mit den 8278 RGB-Daten durchgeführt. Anschließend erfolgt die Klassifizierung der 9918 RGB-D-Testdaten.

Als Ergebnis der Klassifizierung wurden 9795 der 9918 Objekte korrekt erkannt. Das entspricht einer Erkennungsgenauigkeit von 98.76% beziehungsweise einer Fehlerrate von 1.24%.

Der Großteil der fehlerhaft eingeordneten Objekte stammt mit 115 falschen Klassifizierungen aus der Klasse „calculator“. Statt der korrekten Objektklasse wurden als Ergebnis an dieser Stelle beispielsweise die Klassen „cell_phone“ und „sponge“ ermittelt. Abbildung 6.2 zeigt einen Vergleich der beiden Objektinstanzen „calculator_4“ und „sponge_2“. Anhand der Abbildung

(a) (b)

Abbildung 6.2.: Vergleich der beiden Objektinstanzen „calculator_4“ (a) und „sponge_2“ (b).

wird deutlich, dass sich die Objekte sowohl in Farbe als auch ihren geometrischen Eigenschaften ähneln. Da die Klasse „calculator“ einer Quaderform entspricht, die ebenfalls auch viele andere Objekte des Datensatzes besitzen, kann die Hinzunahme der Tienfeninformationen hier keinen Vorteil herbeiführen. Zusätzlich besitzen die anderen Objektinstanzen der Klasse verbreitetere Farben, sodass die Klassifizierung anhand des Farbanteils ebenfalls eine Herausforderung darstellt. Aus diesen Gründen ist die starke Falschklassifizierung von insgesamt 115 Objekten der Klasse „calculator“ nachvollziehbar.

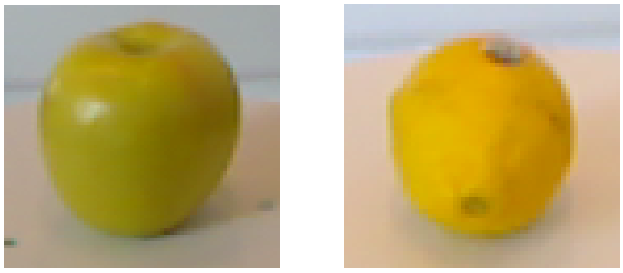
Kategorieerkennung

Im Unterschied zur Instanzenerkennung wird der Test einer Kategorieerkennung durchgeführt, indem alle bis auf eine der Objektinstanzen einer Objektklasse für ein Training genutzt werden. Anschließend wird die nicht trainierte Objektinstanz verwendet, um zu testen, ob auch diese als Teil der Objektklasse erkannt wird. Der Zweck dieses Tests ist die Evaluierung der Erkennungsrate von nicht trainierten, also fremden, Objekten. Als fremdes Objekt wird in diesem Fall ein Objekt bezeichnet, dessen Objektinstanz nicht trainiert wurde, das aber zu einer trainierten Objektklasse gehört. Am Beispiel der Abbildung 6.1 würden für dieses Testszenario nur die Objektinstanzen „apple_1“ bis „apple_4“ trainiert werden. Anschließend würde ein Test mit der Objektinstanz „apple_5“ zeigen, ob mit dem Training eine Generalisierung für die Objektklasse „apple“ erreicht wurde.

Zur Evaluierung der Kategorieerkennung wird für ein aussagekräftiges Ergebnis eine 10-fach Kreuzvalidierung genutzt. Die Kreuzvalidierung ist notwendig, da gewisse Objektinstanzen besser als andere erkannt werden. Aus diesem Grund werden zehn Testläufe durchgeführt, bei denen aus jeder der 51 Kategorien jeweils andere Objektinstanzen für das Training entfernt werden. Als Datenbasis werden die gleichen 9918 Objekte wie bei der Instanzenerkennung genutzt. Natürlich verringert sich diese Anzahl entsprechen der entfernten Objektinstanzen. Ein Protokoll welche Objektinstanzen in den einzelnen Testdurchläufen entfernt wurden liegt in Anhang A.2 bei.

Das Ergebnis der Kreuzvalidierung ergibt einen Mittelwert von 85.044% mit einer Standardabweichung von 2.2413%. Eine Auswertung der einzelnen Testdurchläufe zeigt, dass es sich in jedem Fall um nachvollziehbar falsche Klassifizierungen handelt. Die verwechselten Objekte ähneln sich immer stark in Form und Farbe. Ein Beispiel hierfür ist die Objektklasse „apple“ mit der Objektklasse „lemon“. Abbildung 6.3 visualisiert die verwechselten Klassen im Vergleich.

Des Weitere wurden einzelne Objektinstanzen in Testläufen der Kreuzvalidierung zu 100% erkannt. Betrachtet auf die Gesamtheit aller Testläufe hingegeben, konnte jedoch keine Objektklasse zu 100% erkannt werden.



(a) (b)

Abbildung 6.3.: Die Objektklasse „apple“ (a) und „lemon“ (b) wurden bei der Klassifizierung der Objektkategorie mehrfach verwechselt.

Kinect One

Für einen weiteren Test der Kategorieerkennung wurden RGB-D-Aufnahmen mit der Kinect One erstellt. Hierfür wurde neben dem Microsoft Kinect for Windows SDK Version 2.0[1] auch die Software MeshLab[2] für die Nachbearbeitung verwendet.

Der Prozess zur Erstellung der passenden Daten gestaltete sich dabei sehr zeitaufwendig. Neben fehlenden Werkzeugen zur Segmentierung konnten keine Aufnahmen mit den Eigenschaften des Washington RGB-D-Objektdatensatz erzeugt werden. Des Weiteren war eine manuelle Nachbearbeitung jeder erstellten Punktwolke notwendig. Abbildung 6.4 zeigt das Resultat in Form einer mit der Kinect One aufgenommenen Punktwolke einer Tasse, nach der manuellen Nachbearbeitung.

[1] https://www.microsoft.com/en-us/download/details.aspx?id=44561 ; Zugriffsdatum: 03.05.2016
[2] http://meshlab.sourceforge.net/ ; Zugriffsdatum: 03.05.2016

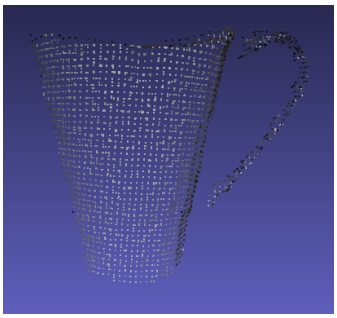


Abbildung 6.4.: Mit der Kinect One aufgenommene Punktwolke einer Tasse nach einer manuellen Nachbearbeitung.

Beim Test der Kategorieerkennung mithilfe der Kinect One konnte keine der Objektklasse korrekt klassifiziert werden. Da selbst der Test mit einer großen Anzahl an RGB-D-Daten des gleichen Datensatzes nur zu einem 85.044% Erfolg führte, steht die Menge der verschiedenen getesteten Objekteinstanzen mit großer Wahrscheinlichkeit im Zusammenhang zur Erfolgsquote. Ein Test mit einer mehrfach Kreuzvalidierung kann allerdings aufgrund der geringen Anzahl an Daten nicht durchgeführt werden.

Die falsche Klassifizierung ist unter Umständen auch auf die Qualitätsunterschiede zwischen den zwei verwendeten RGB-D-Kameras zurückzuführen. Die Aufnahmen des Washington RGB-D-Objektdatensatzes wurde mit einer ASUS Xtion Pro Live aufgenommen. Im Gegensatz zur Kinect One handelt es sich dabei um eine ältere Kamera mit einer geringeren Auflösung. Da das Training mit Aufnahmen dieser Kamera erfolgte, ist zu vermuten, dass aufgrund dessen keine korrekte Klassifizierung erfolgen kann. Der Unterschied der aufgenommenen Daten ist zu signifikant, sodass keines der Testobjekte korrekt erkannt wurde. Die zeitaufwendige Erstellung passender RGB-D-Daten macht es zum aktuellen Zeitpunkt allerdings unmöglich einen gesamten Trainingsdatensatz in der geforderten Größe mit der Kinect One zu erzeugen.

6.2.2. Mehrwert der Tiefeninformationen

Das VGGNet stellt einen sehr guten Merkmalsextraktor dar und liefert aus diesem Grund bereits sehr viele hochwertige Merkmale. Der Mehrwert der Tiefeninformationen ergibt sich infolgedessen bei Objekten, die sich in ihren Farbinformationen sehr ähnlich sind, aber sich in ihren geometrischen Eigenschaften unterscheiden.

Um diese Eigenschaften ebenfalls zu testen, wurde zusätzlich eine Klassifizierung nur anhand des Farbanteils der RGB-D-Daten vorgenommen. Im Folgenden wurden Ergebnisse

dieser Klassifizierung mit den Ergebnissen der bereits durchgeführten Instanzenerkennung gegenübergestellt. Für das Training sowie den Test wurden dieselben Objekte wie bei der Instanzenerkennung genutzt.

Das Ergebnis der Klassifizierung ohne Tiefeninformationen liegt bei einer Fehlerrate von 1.49%. Für eine bessere Übersicht wurden die Fehlerraten ohne sowie mit Tiefeninformationen in Tabelle 6.2 visualisiert.

Datenbasis	Fehlerrate
RGB	1.49%
RGB-D	1.24%

Tabelle 6.2.: Vergleich der Fehlerrate bei Verwendung von RGB- oder RGB-D-Daten.

Die Ergebnisse zeigen, dass durch die zusätzliche Verwendung von Tiefeninformationen eine relative Verbesserung von 20% erzielt wird. Die Fehlerrate ist an dieser Stelle definiert als der prozentuale Anteil von Objekten, die zwar als Objektklasse trainiert, aber vom System als eine andere Objektklasse klassifiziert wurden.

Ein Beispiel hierfür stellt die Objektklasse „cereal_box“ dar. Bei einer Durchführung der Objekterkennung, die nur auf RGB-Daten basiert, wurde diese Klasse 24-mal fälschlicherweise als Klasse „food_box“ klassifiziert. Durch die Hinzunahme der Tiefeninformationen wurden allerdings alle fehlerhaften Klassifizierungen dieser Objektklasse behoben. Abbildung 6.5 zeigt, wie ähnlich sich die Objektinstanzen der Klassen „food_box“ und „cereal_box“ sind. Durch die Visualisierung wird deutlich, dass selbst ein menschliches Auge Probleme hätte, nur anhand der dargestellten Bilder beider Objekte diese auseinanderzuhalten. Bedingt ist dies durch die Aufnahme, mit der die Unterschiede der Objekteigenschaften nicht deutlich werden. Die Hinzunahme der Tiefeninformationen ermöglicht es allerdings, die geometrischen Eigenschaften zu extrahieren und so eine korrekte Klassifizierung zu erreichen.

6.2.3. Mehrwert des HSI-Farbraums

Zur Überprüfung des Mehrwerts einer Verwendung des HSI- gegenüber des RGB-Farbraums wurden zwei Pipelines mit dem jeweiligen Farbraum sowie Tiefeninformationen unter Verwendung einer geringeren Anzahl an Daten trainiert. Beide Pipelines werden mit dem gleichen Datenbestand trainiert. Während die Daten für die HSI-Pipeline zunächst in diesen Farbraum

(a) (b)

Abbildung 6.5.: Vergleich der beiden Objektinstanzen „cereal_box_5“ (a) und „food_box_2“ (b).

konvertiert werden müssen, können die Daten der RGB-Pipeline ohne Vorverarbeitung verwendet werden. Die Konvertierung wurde mithilfe von OpenCV[3] implementiert.

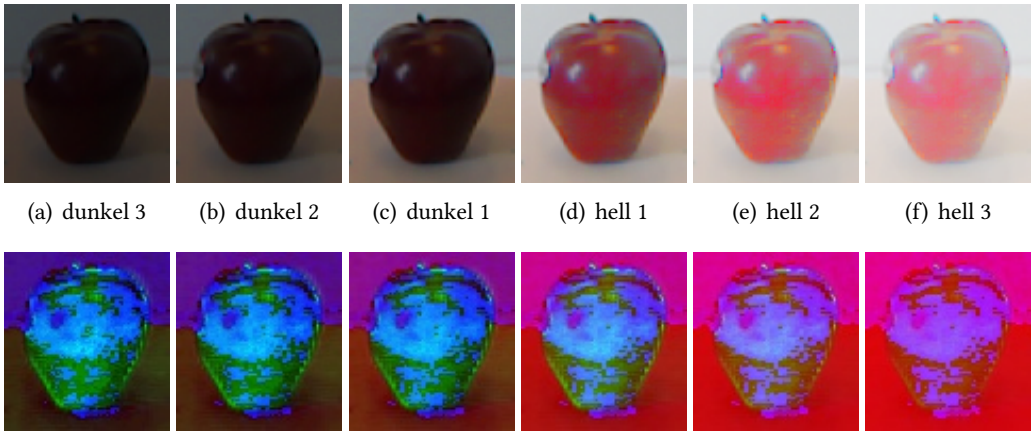
Für die Zusammenstellung des Datenbestands wird jeweils eine Objektinstanz aus allen Objektkategorien gewählt. Innerhalb dieser Objektinstanz werden anschließend 20 Aufnahmen gewählt. Insgesamt ergibt sich aufgrund dieser Auswahl ein Trainingsdatenbestand mit 1020 Objektaufnahmen.

Für den Test nach dem Trainingsprozess wird eine Instanzenerkennung verwendet. Da der Mehrwert des HSI-Farbraums gegenüber des RGB-Farbraums bei unterschiedlichen Lichtverhältnissen ermittelt werden soll, sind für diesen Zweck zunächst Daten zu erzeugen, die diese Eigenschaft besitzen. Abbildung 6.6 zeigt beispielhaft die Objektinstanz „apple_1“ bei unterschiedlichen Lichtverhältnissen. Die angepassten Lichtverhältnisse wurden für diesen Zweck in die Belichtungsklassen „hell“ und „dunkel“ eingeteilt.

Für einen Test wurden die Lichterverhältnisse des Testobjekts zunächst so lange sowohl positiv (hell) als auch negativ (dunkel) angepasst, bis mit der RGB-Pipeline keine erfolgreiche Klassifizierung mehr möglich war. Anschließend wurden die Testbilder in den HSI-Farbraum konvertiert und dasselbe Testszenario mit der HSI-Pipeline durchlaufen. Tabelle 6.3 fasst die Ergebnisse dieses Tests zusammen.

Aus den Ergebnissen geht klar hervor, dass der RGB-Farbraum eine bessere Wahl für unterschiedliche Lichtverhältnisse darstellt. Während der HSI-Farbraum nur in der Lage ist ein Objekt bei Lichtverhältnissen bis zu einer Belichtungsklasse von „hell 1“ korrekt zu klassifizieren, ist dies mit dem RGB-Farbraum bis zur Klasse „hell 4“ möglich. Des Weiteren ist der

[3] Bei OpenCV handelt es sich um eine Bibliothek, die Algorithmen für die Bildverarbeitung und das maschinelle Sehen sowie Lernen bereitstellt.



(a) dunkel 3 (b) dunkel 2 (c) dunkel 1 (d) hell 1 (e) hell 2 (f) hell 3

Abbildung 6.6.: Abbildung eines Objekts bei unterschiedlichen Lichtverhältnissen im RGB- (obere Reihe) und HSI-Farbraum (untere Reihe).

Belichtungs-klasse	Klassifizierung korrekt?	
	RGB-Farbraum	HSI-Farbraum
hell 5	nein	nein
hell 4	ja	nein
hell 3	ja	nein
hell 2	ja	nein
hell 1	ja	ja
normal	ja	ja
dunkel 1	ja	ja
dunkel 2	ja	ja
dunkel 3	ja	ja
dunkel 4	ja	nein
..	..	..
dunkel 9	ja	nein
dunkel 10	nein	nein

Tabelle 6.3.: Vergleich der korrekten Klassifizierung bei unterschiedlichen Belichtungsklassen zwischen dem RGB- und dem HSI-Farbraum.

RGB-Farbraum in der Lage bei sehr schwachen Lichtverhältnissen bis zur Klasse „dunkel 9„ eine korrekte Klassifizierung zu erreichen, während HSI dies nur bis zur Klasse „dunkel 3“ gelingt. Mithilfe des RGB-Farbraums ist es der Pipeline sogar möglich Objekte zu klassifizieren, die selbst durch einen Menschen nur schwer einzuordnen wären. Abbildung 6.7 zeigt als Beispiel hierfür eine Aufnahme der Objektinstanz „apple_1“ mit einer Belichtungsklasse von „dunkel 9“, die korrekt als solche von der Pipeline klassifiziert wurde.

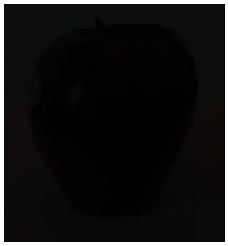
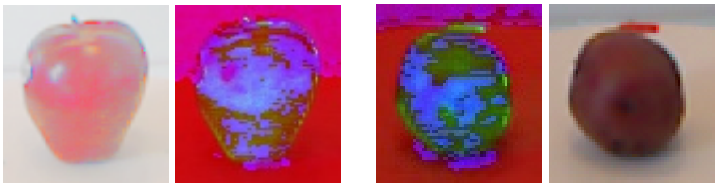


Abbildung 6.7.: Aufnahme der Objektinstanz „apple_1“ mit der Belichtungsklasse „dunkel 9“.

Der Grund für die signifikant schlechteren Ergebnisse von HSI gegenüber RGB könnte die Ähnlichkeit der Farbinformationen zu anderen Objektklassen einer anderen Belichtungsklasse sein. Während des Testszenarios wurde beispielsweise die Aufnahme der Objektklasse „apple“ fälschlicherweise als ein Objekt der Klasse „potato“ klassifiziert. Ein Vergleich der beiden Aufnahmen zeigt, dass gerade durch die Anpassung der Belichtungsklasse der Klasse „apple“ zu einer größeren Übereinstimmung mit der Belichtungsklasse „normal“ der Objektklasse „potato“ führt. Abbildung 6.8 zeigt die Visualisierung dieses Beispiels. Wie dort zu sehen weisen sowohl Farbton als auch die Farbverteilung innerhalb des Objekts im HSI-Farbraum, siehe Abbildung 6.8.b und 6.8.c, eine große Ähnlichkeit auf. Anzumerken ist an dieser Stelle, dass nur die Belichtung der Objektklasse „apple“ auf „hell 3“angepasst wurde. Im Gegensatz dazu zeigen die gleichen Aufnahmen im RGB-Farbraum, siehe Abbildung 6.8.a und 6.8.d, keine derartigen Übereinstimmungen.

Als Ergebnis der Evaluierung des HSI-Farbraums für die Objekterkennung ist festzuhalten, dass sich dieser für die gegebenen Umstände nicht eignet. Durch die Verwendung gegenüber des RGB-Farbraums ergeben sich keine Vorteile. Entgegen den Erwartungen würde eine Verwendung sogar zu signifikanten Nachteilen führen.



(a) (b) (c) (d)

Abbildung 6.8.: Beispiel der zwei Objektklassen „apple“ (a, b) und „potato“ (c, d) bei unterschiedlichen Belichtungsklassen im RGB- (a, d) und HSI-Farbraum (b, c).

6.2.4. Vergleichbare Arbeiten

In Abschnitt 2.1 wurden bereits zwei Arbeiten beschrieben, die neben den Farb- auch die Tiefeninformationen für die Objekterkennung einsetzen. Die Arbeit von Schwarz u. a. (2015) setzt hierfür ebenfalls den Washington RGB-D-Objektdatensatz ein. Des Weiteren wurde auch von den Erstellern des Datensatzes (Lai u. a., 2011) ein Konzept für die Nutzung von RGB-D-Daten zur Objekterkennung entwickelt. Um die Ergebnisse dieser Thesis mit anderen Resultaten des gleichen Datensatzes zu vergleichen, erfolgt eine Gegenüberstellung und Diskussion der Ergebnisse dieser Arbeiten mit den Ergebnissen des eigenen Ansatzes.

Wie in Abschnitt 2.1 bereits beschrieben, verwenden Schwarz u. a. (2015) zwei Faltungsnetzwerke für die Merkmalsextraktion und mehrere SVMs für die Objektklassifizierung. Im Gegensatz dazu wird in der Arbeit von Lai u. a. (2011) eine Kombination aus verschiedenen lokalen Deskriptoren zur Merkmalsextraktion eingesetzt, um diese anschließend ebenfalls von einer SVM klassifizieren zu lassen.

Beide Arbeiten verwenden genau wie diese Thesis die Testarten der Instanzen- und Kategorieerkennung. Anders als beim eigenen Ansatz wird für einen Test der Instanzenerkennung jedoch jede fünfte RGB-D-Aufnahme verwendet. Aus diesem Grund steigert sich die Anzahl an Trainingsdaten auf circa 205000 und die Anzahl der Testdaten auf circa 45000 RGB-D-Aufnahmen. Im Gegensatz dazu wurde der eigene Ansatz mit 8278 Aufnahmen trainiert und anschließend mit 9918 Aufnahmen getestet. Ein Vergleich der Ergebnisse wird trotzdem durchgeführt, um eine Tendenz zu ermitteln. Für den Test der Kategorieerkennung werden beim eigenen Ansatz ebenfalls 8278 RGB-D-Aufnahmen abzüglich der jeweiligen Objektinstanz für ein Training verwendet. Die zwei anderen Arbeiten nutzen für den Kategorietest zwar die gleichen Objektinstanzen, trainieren allerdings mit allen übrigen Daten.

	Erkennungsrate (%)	
Methode	Kategorie	Instanz
Lai u. a. (2011)	81.9±2.8	73.9
Schwarz u. a. (2015)	**89.4±1.3**	94.1
Eigener Ansatz	85.0±2.2	**98.8**

Tabelle 6.4.: Gegenüberstellung der Ergebnisse des eigenen Ansatz und der Ergebnisse zweier vergleichbarer Arbeiten.

Tabelle 6.4 zeigt die Ergebnisse aller drei Ansätze. Wie dort dargestellt besitzt der Ansatz von Schwarz u. a. (2015) einen Vorsprung von 4.4% bei der Kategorieerkennung. Der Ansatz von Lai u. a. (2011) wird allerdings mit 4.1% fast mit einem genauso großen Vorsprung vom Ansatz dieser Arbeit geschlagen.

Anders fällt das Ergebnis bei der Instanzenerkennung aus. Dort liegt der in dieser Thesis entwickelte Ansatz mit einem Ergebnis von 98.8% vor dem Ergebnis von Schwarz u. a. (2015) mit 94.1% und deutlich vor dem Ergebnis von Lai u. a. (2011) mit 73.9%. Der große Vorsprung vom eigenen und dem Ansatz von Schwarz u. a. (2015) gegenüber der Arbeit von Lai u. a. (2011) deutet auf die Vorteile bei der Verwendung von Faltungsnetzwerken zur Merkmalsextraktion hin. Da in dieser Arbeit ein „tieferes“ (mehr Schichten) Faltungsnetzwerk als bei Schwarz u. a. (2015) verwendet wird, könnte dies ebenfalls ein ausschlaggebender Punkt für den 4.7 prozentigen Vorsprung bei der Instanzenerkennung sein.

6.3. Laufzeit

In diesem Abschnitt wird die Dauer ermittelt, die von der Pipeline für das Training und die Klassifizierung von RGB-D-Daten benötigt wird. Die Zeitmessungen wurden mit einem System durchgeführt, dass folgende Soft- und Hardware-Ressourcen besitzt:

- **CPU:** Intel(R) Core(TM) i7-4712MQ CPU @ 2.30GHz
- **Arbeitspeicher:** 8 GB
- **Grafikkarte:** NVIDIA GeForce GT 740M
- **Betriebssystem:** Ubuntu 14.04 LTS 64-bit

Die Zeitmessung erfolgt für die einzelnen Komponenten der Pipeline. Ausgenommen hiervon ist das grid-search Verfahren zur Hyperparametersuche, da dies nicht zum Trainings- bezie-

hungsweise Testprozess gehört. Die Messungen werden während der Verarbeitung von einem RGB-D-Datum innerhalb der Pipeline durchgeführt. Insgesamt werden für den Trainingsprozess eines einzelnen RGB-D-Datums 2.391 Sekunden benötigt. Da für ein Training mehr als ein RGB-D-Datum benötigt wird, werden die Trainingsdaten aus Abschnitt 6.2.1 genutzt und anschließend durch die Anzahl der trainierten Objekte dividiert. Die reine Erkennung eines Objekts ohne Training benötigt eine Dauer von 2.402 Sekunden. Die Zeit, die für den Startprozess des VGGNet benötigt wird, wurde an dieser Stelle nicht berücksichtigt, da dieser Vorgang unabhängig von der Datenmenge nur einmal durchgeführt werden muss und hier nur die Dauer der Verarbeitung eines RGB-D-Datums evaluiert werden soll. Tabelle 6.3 ordnet die Ergebnisse der Messungen den einzelnen Komponenten der Pipeline zu und fasst diese anschließend zusammen. Alle dargestellten Ergebnisse beziehen sich auf die Ausführung mit einem einzelnen RGB-D-Datum.

Anzumerken ist an dieser Stelle, dass die Berechnungen des Faltungsnetzwerks auf der GPU durchgeführt wurden. Eine Berechnung dieser Operationen auf der CPU würde die Dauer um einen Faktor von zehn erhöhen. Da der Testrechner nur eine einzige GPU besitzt, konnten die Berechnungen des Faltungsnetzwerks aus diesem Grund nicht parallelisiert werden. Die Verarbeitung der Tiefeninformationen hingegen lässt sich ohne Probleme beliebig parallelisieren.

Des Weiteren wurde eine Zeitmessung für die Suche der Hyperparameter durchgeführt. Für das grid-search Verfahren mit 5-fach Kreuzvalidierung und 8278 RGB-D-Objekten wurde eine Gesamtdauer von 77 Stunden 5 Minuten und 3 Sekunden gemessen. Die Rahmenbedingungen und Ergebnisse dieser Messung wurden in Abschnitt 6.1 bereits beschriebenen.

	Dauer (s)	
Pipeline-Stufe	Training	Test
VGGNet Start[1]	4.036	4.036
VGGNet Extraktion	1.789	1.789
OUR-CVFH	0.573	0.573
Normalisierung	0.005	0.005
Training	0.024	-
Test	-	0.035
Gesamtdauer	2.391	2.402

[1] Wird bei der Berechnung der Gesamtdauer nicht berücksichtigt (siehe Abschnitt 6.3).

Tabelle 6.5.: Ergebnisse der Zeitmessungen für unterschiedliche Komponenten der Pipeline sowie die Gesamtdauer von Training- und Testprozess.

7. Schluss

In diesem Kapitel wird ein Fazit über die Resultate dieser Thesis gezogen. Im Zuge dessen erfolgt außerdem eine Zusammenfassung aller Arbeitsschritte sowie der im Rahmen dessen gewonnenen Ergebnisse. Abschließend wird im Ausblick dargelegt, welche Weiterentwicklungsmöglichkeiten die vorhandenen Ergebnisse bieten.

7.1. Fazit

Im Rahmen dieser Arbeit wurde ein Konzept entwickelt, welches die Merkmale aus Farb- und Tiefeninformationen kombiniert und so die Objekterkennung verbessert. Für den Entwurf dieses Konzepts wurden ein Faltungsnetzwerk, ein Verfahren zur Ermittlung der Objektgeometrie sowie eine Support Vector Machine verwendet. Das entworfene Konzept basiert auf einzelnen Komponenten, die für sich genommen bereits gute Ergebnisse liefern. Zusätzlich wurde der Mehrwert, den eine Kombination mit Tiefeninformationen im Gegensatz zu reinen Farbinformationen liefert ausgewertet und der Einfluss der Lichtverhältnisse auf die Wahl des Farbraums evaluiert.

Bei der Umsetzung des Konzepts wurden Softwarekomponenten zu einer Pipeline für 3D-Objekterkennung kombiniert und weitere Komponenten hinzugefügt. Des Weiteren wurde für eine vereinfachte Bedienung dieser Pipeline ein Frontend entwickelt. Die entwickelte Pipeline bildet den gesamten Prozess der Objekterkennung ab. Sie kann sowohl für das Training als auch den Test und die Hyperparametersuche genutzt werden.

Für die Umsetzung des Faltungsnetzwerks wurde das VGGNet gewählt. Dies stellt einen guten Kompromiss zwischen der Ressourcenauslastung und Erkennungsrate dar. Das Netz wurde genutzt, um Merkmale aus Farbinformationen zu extrahieren. Zu diesem Zweck wurde das Netz nicht wie üblich zunächst mit Daten trainiert, sondern die Gewichtungen eines bereits trainierten VGGNet-Faltungsnetzwerks importiert. Ein weiteres Kriterium für die Wahl dieses Netzes war, dass es sich sehr gut für den Prozess des Transferlernens sowie die Extraktion von Merkmalen eignet.

Bei der Auswahl des Verfahrens zur Ermittlung der Objektgeometrie wurde das OUR-CVFH Verfahren gewählt. Die Selektion erfolgte aufgrund der semi-globalen Eigenschaften dieses Verfahrens, die Nachteile von lokalen und globalen Deskriptoren ausgleichen.

Die Evaluierung der Ergebnisse verdeutlicht, dass sich durch die Kombination der Merkmalsarten ein Vorteil gegenüber anderen Verfahren erzielen lässt. Zurückzuführen ist dies vor allem auf die Kombination von VGGNet und OUR-CVFH. Das VGGNet extrahiert im Großteil aller Fälle bereits sehr eindeutige Merkmale. Bei Objekten, die sich im Farbbild kaum unterscheiden, werden von OUR-CVFH zusätzlich die entscheidenden Merkmale bereitgestellt, sodass auch diese durch eine Kombination der beiden Merkmalsarten korrekt klassifiziert werden können. Des Weiteren wurde durch die Ergebnisse verdeutlicht, dass sich der HSI-Farbraum nicht für eine robustere Erkennung von Objekten in unterschiedlichen Lichtverhältnissen eignet. Entgegen den Erwartungen kann für diesen Zweck der RGB-Farbraum einen Vorteil verschaffen.

Die Erstellung eigener Tiefeninformationen mit der Kinect One stellte sich als problematisch dar. Der Grund hierfür war vor allem die begrenzte Funktionalität des Microsoft Kinect for Windows SDK Version 2.0, aber auch die unterschiedlichen Eigenschaften der Aufnahmen verschiedener RGB-D-Datenquellen. Weiterhin fehlen im Washington RGB-D-Objektdatensatz teilweise Punktwolken zu einem vorhandenen Farbbild. Zusätzlich waren einige Punktwolken fehlerhaft, sodass von diesen keine Extraktion mithilfe von OUR-CVFH möglich war. Insgesamt war die zur Verfügung stehende Auswahl an Trainingsdaten sehr begrenzt, was in direktem Zusammenhang mit dem Arbeitsaufwand bei der Erstellung der Daten steht.

Eine Auswertung der Laufzeit hat ergeben, dass der Ansatz dieser Thesis zu diesem Zeitpunkt nicht echtzeitfähig ist. Bedingt ist dies vor allem durch die Verarbeitungszeit, die vom VGGNet benötigt wird.

Insgesamt wurde die Kombination aus Farb- und Tiefeninformationen erfolgreich umgesetzt. Bei der Zusammenführung der Merkmalsarten ist eine Normalisierung notwendig, da sonst nicht beide Merkmalsarten in gleichem Maße berücksichtigt werden. Da von OUR-CVFH wesentlich höhere Werte produziert werden, würde diese Merkmale ohne eine Normalisierung die Klassifizierung dominieren. Zusätzlich ist es notwendig das Faltungsnetzwerk für die Merkmalsextraktion im Test und nicht im Trainingsmodus zu betreiben. Die Ausführung des Netzes im Trainingsmodus beinhaltet die Dropout-Schichten. Diese sollten bei der Merkmalsextraktion nicht verwendet werden, da sonst für gleiche RGB-Bilder unterschiedliche Merkmale extrahiert werden.

Eine Auswertung der eignen Tests sowie die Gegenüberstellung mit vergleichbaren Arbeiten zeigt zudem, dass durch das Hinzunehmen der Tiefeninformationen ein Mehrwert geschaffen wird. So konnte mithilfe der Tiefeninformationen die Fehlerrate um 20% gesenkt werden.

7.2. Ausblick

Die Ergebnisse dieser Arbeit können als Grundlage verwendet werden, um die Objekterkennung weiter zu verbessern. Die Hypothese, dass die Hinzunahme einer weiteren Merkmalsart bei anspruchsvoll zu klassifizierenden Objekten einen Vorteil mit sich bringt, wurde bestätigt. In dieser Arbeit wurden hierfür Tiefeninformationen als Zusatz zu Farbinformationen gewählt. Dieses Konzept lässt sich aber auch auf weitere Merkmalsarten ausweiten. Eine Hinzunahme von Audiosignalen könnte beispielsweise in gewissen Situationen einen Vorteil verschaffen. Im Allgemeinen könnte eine detailliertere Beschreibung des Objekts, durch eine gesteigerte Anzahl an Merkmalsarten, demzufolge zu einer besseren Erkennungsrate und Robustheit bei der Objekterkennung führen.

Des Weiteren könnten die Ergebnisse dazu verwendet werden weitere Tests mit noch tieferen Faltungsnetzwerken durchzuführen. Außerdem könnte durch eine Verteilung der Berechnung auf mehrere GPUs diese Arbeit als Grundlage für einen Ansatz mit geringerer Laufzeit dienen. Eine Voraussetzung für einen Test mit noch tieferen Faltungsnetzwerken sowie der Verteilung der Berechnung auf mehrere GPUs ist allerdings eine größere Verfügbarkeit von Hardware-Ressourcen. Eine Steigerung der Performanz würde den Ansatz für den Bereich der Robotik praktikabler machen.

Weiteres Entwicklungspotenzial bietet der fehlerbehaftete und aufwendige Prozess der Punktwolkenerstellung mit der Kinect One. Eine Verbesserung in diesem Bereich ermöglicht die Erzeugung von eigenen Daten in gefordertem Umfang und infolge dessen auch neue Möglichkeiten für Training und Test der Pipeline.

A. Anhang

A.1. Inhalt DVD

Abbildung A.1 zeigt die Verzeichnistruktur der beigefügten DVD. Im Wurzelverzeichnis befinden sich neben einer PDF-Version dieser Thesis die zwei Ordner „Code“ sowie „Daten“. Code beinhaltet den programmierten Anteil dieser Thesis. Hierzu zählt neben allen Komponenten der Pipeline auch Bash-Skripte für die Datenvorverarbeitung und die Caffe-Konfigurationsdatei. Im Ordner „Daten“ befinden sich außerdem alle Ergebnisse die Erzeugt wurden. Dies beinhaltet die unterschiedlichen .model-Dateien der verschiedenen SVMs, aber auch die normalisierten Merkmale sowie die Resultate der in Kapitel 6 diskutierten Ergebnisse.

A.2. Testobjekte der Kategorieerkennung

In den Tabellen A.1 und A.2 sind die Testobjekte aufgelistet, die für die Kategorieerkennung verwendet wurden. Details hierzu sind in Abschnitt 6.2.1 zu finden.

A.3. Klassenzuordnung des Washington RGB-D-Objektdatensatzes

Tabelle A.3 zeigt die Objektklassen des Washington RGB-D-Objektdatensatzes sowie die in dieser Arbeit verwendete Zuordnung von diesen zu einer Klassennummer.

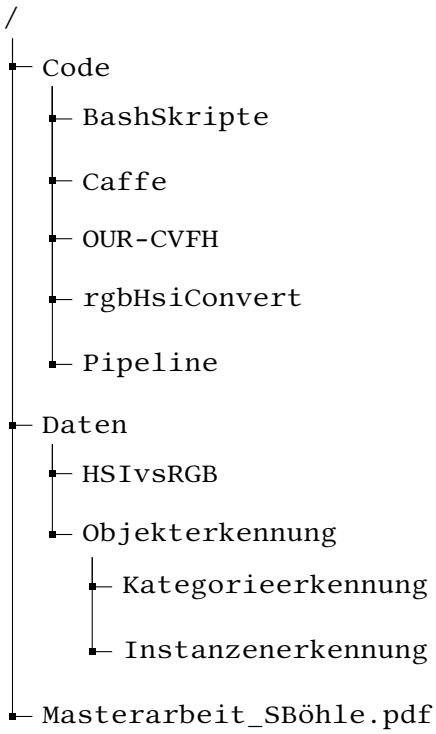


Abbildung A.1.: Verzeichnisstruktur der DVD

Durchlauf				
1	2	3	4	5
apple_3	apple_3	apple_2	apple_5	apple_4
ball_1	ball_1	ball_4	ball_6	ball_2
banana_4	banana_4	banana_4	banana_1	banana_4
bell_pepper_6	bell_pepper_3	bell_pepper_1	bell_pepper_2	bell_pepper_1
binder_3	binder_3	binder_3	binder_1	binder_2
bowl_5	bowl_6	bowl_1	bowl_6	bowl_5
calculator_1	calculator_2	calculator_4	calculator_1	calculator_2
camera_2	camera_2	camera_2	camera_3	camera_1
cap_1	cap_4	cap_4	cap_3	cap_3
cell_phone_2	cell_phone_2	cell_phone_3	cell_phone_5	cell_phone_5
cereal_box_3	cereal_box_3	cereal_box_5	cereal_box_3	cereal_box_3
coffee_mug_2	coffee_mug_8	coffee_mug_8	coffee_mug_7	coffee_mug_6
comb_1	comb_5	comb_5	comb_2	comb_3
dry_battery_3	dry_battery_4	dry_battery_5	dry_battery_6	dry_battery_6
flashlight_3	flashlight_5	flashlight_1	flashlight_4	flashlight_5
food_bag_1	food_bag_2	food_bag_2	food_bag_7	food_bag_6
food_box_5	food_box_9	food_box_5	food_box_4	food_box_1
food_can_11	food_can_7	food_can_11	food_can_8	food_can_12
food_cup_2	food_cup_3	food_cup_5	food_cup_1	food_cup_3
food_jar_4	food_jar_4	food_jar_5	food_jar_2	food_jar_4
garlic_2	garlic_2	garlic_2	garlic_5	garlic_1
glue_stick_5	glue_stick_1	glue_stick_4	glue_stick_4	glue_stick_1
greens_2	greens_1	greens_2	greens_3	greens_2
hand_towel_2	hand_towel_2	hand_towel_4	hand_towel_4	hand_towel_4
instant_noodles_3	instant_noodles_5	instant_noodles_3	instant_noodles_4	instant_noodles_4
keyboard_1	keyboard_1	keyboard_5	keyboard_3	keyboard_3
kleenex_4	kleenex_4	kleenex_2	kleenex_5	kleenex_3
lemon_4	lemon_3	lemon_2	lemon_3	lemon_6
lightbulb_3	lightbulb_2	lightbulb_3	lightbulb_1	lightbulb_1
lime_3	lime_4	lime_1	lime_4	lime_2
marker_1	marker_4	marker_9	marker_2	marker_1
mushroom_2	mushroom_1	mushroom_2	mushroom_3	mushroom_3
notebook_3	notebook_1	notebook_2	notebook_1	notebook_2
onion_4	onion_2	onion_4	onion_6	onion_5
orange_4	orange_1	orange_4	orange_1	orange_3
peach_3	peach_2	peach_2	peach_1	peach_3
pear_8	pear_1	pear_2	pear_2	pear_2
pitcher_3	pitcher_2	pitcher_1	pitcher_1	pitcher_1
plate_4	plate_6	plate_2	plate_7	plate_1
pliers_3	pliers_2	pliers_3	pliers_6	pliers_2
potato_1	potato_5	potato_5	potato_1	potato_4
rubber_eraser_3	rubber_eraser_4	rubber_eraser_1	rubber_eraser_1	rubber_eraser_2
scissors_4	scissors_4	scissors_1	scissors_3	scissors_1
shampoo_3	shampoo_4	shampoo_2	shampoo_6	shampoo_4
soda_can_2	soda_can_3	soda_can_4	soda_can_6	soda_can_4
sponge_5	sponge_11	sponge_4	sponge_10	sponge_7
stapler_6	stapler_5	stapler_7	stapler_1	stapler_5
tomato_8	tomato_5	tomato_8	tomato_5	tomato_1
toothbrush_3	toothbrush_3	toothbrush_1	toothbrush_1	toothbrush_2
toothpaste_5	toothpaste_4	toothpaste_1	toothpaste_1	toothpaste_3
water_bottle_2	water_bottle_9	water_bottle_2	water_bottle_3	water_bottle_10

Tabelle A.1.: Testobjekte der Kategorieerkennung für Durchlauf 1 - 5.

Durchlauf				
6	7	8	9	10
apple_3	apple_2	apple_5	apple_3	apple_3
ball_3	ball_2	ball_3	ball_1	ball_1
banana_3	banana_3	banana_4	banana_2	banana_1
bell_pepper_1	bell_pepper_3	bell_pepper_1	bell_pepper_5	bell_pepper_4
binder_1	binder_1	binder_2	binder_1	binder_2
bowl_6	bowl_4	bowl_2	bowl_6	bowl_5
calculator_3	calculator_5	calculator_3	calculator_4	calculator_5
camera_3	camera_1	camera_2	camera_3	camera_3
cap_4	cap_2	cap_1	cap_4	cap_1
cell_phone_1	cell_phone_4	cell_phone_3	cell_phone_3	cell_phone_1
cereal_box_3	cereal_box_1	cereal_box_1	cereal_box_5	cereal_box_1
coffee_mug_7	coffee_mug_3	coffee_mug_5	coffee_mug_5	coffee_mug_7
comb_2	comb_3	comb_2	comb_1	comb_3
dry_battery_4	dry_battery_5	dry_battery_3	dry_battery_6	dry_battery_4
flashlight_3	flashlight_2	flashlight_4	flashlight_3	flashlight_4
food_bag_6	food_bag_3	food_bag_8	food_bag_4	food_bag_3
food_box_8	food_box_5	food_box_5	food_box_6	food_box_6
food_can_4	food_can_9	food_can_13	food_can_3	food_can_4
food_cup_1	food_cup_2	food_cup_1	food_cup_3	food_cup_3
food_jar_1	food_jar_1	food_jar_4	food_jar_2	food_jar_4
garlic_4	garlic_4	garlic_4	garlic_5	garlic_3
glue_stick_2	glue_stick_3	glue_stick_1	glue_stick_1	glue_stick_1
greens_2	greens_3	greens_1	greens_1	greens_4
hand_towel_5	hand_towel_3	hand_towel_3	hand_towel_3	hand_towel_3
instant_noodles_6	instant_noodles_1	instant_noodles_6	instant_noodles_7	instant_noodles_4
keyboard_1	keyboard_5	keyboard_2	keyboard_1	keyboard_4
kleenex_3	kleenex_2	kleenex_5	kleenex_1	kleenex_4
lemon_3	lemon_6	lemon_2	lemon_2	lemon_4
lightbulb_3	lightbulb_2	lightbulb_1	lightbulb_4	lightbulb_1
lime_4	lime_3	lime_2	lime_2	lime_3
marker_3	marker_4	marker_2	marker_5	marker_4
mushroom_1	mushroom_1	mushroom_2	mushroom_1	mushroom_2
notebook_2	notebook_3	notebook_1	notebook_4	notebook_3
onion_3	onion_4	onion_3	onion_6	onion_4
orange_2	orange_2	orange_3	orange_3	orange_2
peach_2	peach_2	peach_3	peach_2	peach_2
pear_3	pear_4	pear_8	pear_4	pear_2
pitcher_2	pitcher_2	pitcher_3	pitcher_1	pitcher_2
plate_1	plate_3	plate_7	plate_7	plate_2
pliers_2	pliers_6	pliers_6	pliers_2	pliers_6
potato_6	potato_3	potato_6	potato_4	potato_6
rubber_eraser_3	rubber_eraser_4	rubber_eraser_3	rubber_eraser_4	rubber_eraser_1
scissors_3	scissors_4	scissors_2	scissors_3	scissors_1
shampoo_2	shampoo_5	shampoo_3	shampoo_3	shampoo_3
soda_can_6	soda_can_2	soda_can_2	soda_can_2	soda_can_2
sponge_3	sponge_1	sponge_5	sponge_11	sponge_11
stapler_8	stapler_7	stapler_4	stapler_3	stapler_1
tomato_1	tomato_2	tomato_2	tomato_5	tomato_4
toothbrush_2	toothbrush_4	toothbrush_1	toothbrush_3	toothbrush_5
toothpaste_3	toothpaste_5	toothpaste_5	toothpaste_2	toothpaste_2
water_bottle_1	water_bottle_1	water_bottle_3	water_bottle_7	water_bottle_5

Tabelle A.2.: Testobjekte der Kategorieerkennung für Durchlauf 6 - 10.

Klassennr.	Objektklasse	Klassennr.	Objektklasse
1	apple	27	kleenex
2	ball	28	lemon
3	banana	29	lightbulb
4	bell_pepper	30	lime
5	binder	31	marker
6	bowl	32	mushroom
7	calculator	33	notebook
8	camera	34	onion
9	cap	35	orange
10	cell_phone	36	peach
11	cereal_box	37	pear
12	coffee_mug	38	pitcher
13	comb	39	plate
14	dry_battery	40	pliers
15	flashlight	41	potato
16	food_bag	42	rubber_eraser
17	food_box	43	scissors
18	food_can	44	shampoo
19	food_cup	45	soda_can
20	food_jar	46	sponge
21	garlic	47	stapler
22	glue_stick	48	tomato
23	greens	49	toothbrush
24	hand_towel	50	toothpaste
25	instant_noodles	51	water_bottle
26	keyboard		

Tabelle A.3.: Darstellung aller Objektklassen des Washington RGB-D-Objektdatensatzes sowie die Zuordnung zur jeweiligen internen Klassennummer dieser Arbeit.

Literaturverzeichnis

[Aldoma u. a. 2012] Aldoma, Aitor ; Tombari, Federico ; Rusu, Radu B. ; Vincze, Markus: OUR-CVFH–Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram for Object Recognition and 6DOF Pose Estimation. In: *Pattern Recognition*. Springer Berlin Heidelberg, 2012, S. 113–122

[Alexandre 2014] Alexandre, Luís A.: 3D Object Recognition using Convolutional Neural Networks with Transfer Learning between Input Channels. In: *13th International Conference on Intelligent Autonomous Systems* Bd. 301. Padova, Italy : Springer, July 2014

[Andreopoulos und Tsotsos 2013] Andreopoulos, Alexander ; Tsotsos, John K.: 50 Years of object recognition: Directions forward. In: *Computer Vision and Image Understanding* 117 (2013), Nr. 8, S. 827 – 891. – URL http://www.sciencedirect.com/science/article/pii/S107731421300091X. – ISSN 1077-3142

[Arbeláez u. a. 2014] Arbeláez, P. ; Pont-Tuset, J. ; Barron, J. ; Marques, F. ; Malik, J.: Multiscale Combinatorial Grouping. In: *Computer Vision and Pattern Recognition*, 2014

[Chang und Lin 2011] Chang, Chih-Chung ; Lin, Chih-Jen: LIBSVM: A library for support vector machines. In: *ACM Transactions on Intelligent Systems and Technology* 2 (2011), S. 27:1–27:27. – Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[Deng u. a. 2009] Deng, J. ; Dong, W. ; Socher, R. ; Li, L.-J. ; Li, K. ; Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*, 2009

[Fankhauser u. a. 2015] Fankhauser, P. ; Bloesch, M. ; Rodriguez, D. ; Kaestner, R. ; Hutter, M. ; Siegwart, R.: Kinect v2 for mobile robot navigation: Evaluation and modeling. In: *Advanced Robotics (ICAR), 2015 International Conference on*, July 2015, S. 388–394

[Fukushima 1980] Fukushima, Kunihiko: Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. In: *Biological Cybernetics* 36 (1980), S. 193–202

[Girshick u. a. 2013] Girshick, Ross B. ; Donahue, Jeff ; Darrell, Trevor ; Malik, Jitendra: Rich feature hierarchies for accurate object detection and semantic segmentation. In: *CoRR* abs/1311.2524 (2013). – URL http://arxiv.org/abs/1311.2524

[Goodfellow u. a. 2016] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. 2016. – URL http://goodfeli.github.io/dlbook/. – Book in preparation for MIT Press

[Gupta u. a. 2014] Gupta, Saurabh ; Girshick, Ross ; Arbelaez, Pablo ; Malik, Jitendra: Learning Rich Features from RGB-D Images for Object Detection and Segmentation. 2014

[Hassenkloever 2012] Hassenkloever, Tobias: *Klassifikation hochvarianter Muster mit Faltungsnetzwerken*, Hochschule fÃ¼r Angewandte Wissenschaften Hamburg, Diplomarbeit, 2012

[He u. a. 2015] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. In: *CoRR* abs/1512.03385 (2015). – URL http://arxiv.org/abs/1512.03385

[Heaton 2015] Heaton, Jeff: *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks*. CreateSpace Independent Publishing Platform, Dezember 2015. – ISBN 978-1505714340

[Holz u. a. 2011] Holz, Dirk ; Holzer, Stefan ; Rusu, Radu B. ; Behnke, Sven: Real-Time Plane Segmentation using RGB-D Cameras. In: *Proceedings of the 15th RoboCup International Symposium* Bd. 7416. Istanbul, Turkey : Springer, July 2011, S. 307–317

[Hsu u. a. 2010] Hsu, Chih-Wei ; Chang, Chih-Chung ; Lin, Chih jen: *A practical guide to support vector classification*. 2010

[Huang und LeCun 2006] Huang, Fu J. ; LeCun, Y.: Large-scale Learning with SVM and Convolutional for Generic Object Categorization. In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on* Bd. 1, June 2006, S. 284–291. – ISSN 1063-6919

[Jia u. a. 2014] Jia, Yangqing ; Shelhamer, Evan ; Donahue, Jeff ; Karayev, Sergey ; Long, Jonathan ; Girshick, Ross ; Guadarrama, Sergio ; Darrell, Trevor: Caffe: Convolutional Architecture for Fast Feature Embedding. In: *arXiv preprint arXiv:1408.5093* (2014)

[Krizhevsky u. a. 2012] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F. (Hrsg.) ; Burges, C.J.C. (Hrsg.) ; Bottou, L. (Hrsg.) ; Weinberger, K.Q. (Hrsg.): *Advances in Neural Information*

Processing Systems 25. Curran Associates, Inc., 2012, S. 1097–1105. – URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[Kubat 2015] Kubat, M.: *An Introduction to Machine Learning*. Springer International Publishing, 2015. – URL https://books.google.de/books?id=2EAwCgAAQBAJ. – ISBN 9783319200101

[Lai u. a. 2011] Lai, K. ; Bo, Liefeng ; Ren, Xiaofeng ; Fox, D.: A large-scale hierarchical multi-view RGB-D object dataset. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, S. 1817–1824. – ISSN 1050-4729

[Lecun u. a. 1998] Lecun, Y. ; Bottou, L. ; Bengio, Y. ; Haffner, P.: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86 (1998), Nov, Nr. 11, S. 2278–2324. – ISSN 0018-9219

[Li und Karpathy 2015] Li, Fei F. ; Karpathy, Andrej: *CS231n: Convolutional Neural Networks for Visual Recognition*. University Lecture. 2015. – URL http://cs231n.stanford.edu/. – Zugriffsdatum: 11.03.2016

[Meisel 2012] Meisel, Andreas: *Robot Vision*. University Lecture. 2012

[Microsoft 2014] Microsoft: *Kinect v2 fuer Windows*. URL https://dev.windows.com/de-de/kinect. 2014. – Zugriffsdatum: 11.03.2016

[Mueller u. a. 2001] Mueller, Klaus-Robert ; Mika, Sebastian ; Raetsch, Gunnar ; Tsuda, Koji ; Schoelkopf, Bernhard: An introduction to kernel-based learning algorithms. In: *IEEE TRANSACTIONS ON NEURAL NETWORKS* 12 (2001), Nr. 2, S. 181–201

[Nischwitz u. a. 2011] Nischwitz, A. ; Fischer, M. ; Haberaecker, P. ; Socher, G.: *Computergrafik und Bildverarbeitung: Band II: Bildverarbeitung*. Bd. Band II: Bildverarbeitung. Vieweg+Teubner Verlag, 2011. – ISBN 9783834883001

[Rusu und Cousins 2011] Rusu, Radu B. ; Cousins, Steve: 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 9-13 2011

[Rusu u. a. 2010] Rusu, R.B. ; Bradski, G. ; Thibaux, R. ; Hsu, J.: Fast 3D recognition and pose using the Viewpoint Feature Histogram. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Oct 2010, S. 2155–2162. – ISSN 2153-0858

[Schmidhuber 2014] Schmidhuber, Jürgen: Deep Learning in Neural Networks: An Overview. In: *CoRR* abs/1404.7828 (2014). – URL http://arxiv.org/abs/1404.7828

[Schwarz u. a. 2015] Schwarz, Max ; Schulz, Hannes ; Behnke, Sven: RGB-D Object Recognition and Pose Estimation based on Pre-trained Convolutional Neural Network Features. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2015), May

[Simonyan und Zisserman 2014] Simonyan, K. ; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *CoRR* abs/1409.1556 (2014)

[Socher u. a. 2012] Socher, Richard ; Huval, Brody ; Bhat, Bharath ; Manning, Christopher D. ; Ng, Andrew Y.: Convolutional-Recursive Deep Learning for 3D Object Classification. In: *Advances in Neural Information Processing Systems 25*. 2012

[Steinmueller 2008] Steinmueller, J.: *Bildanalyse*. Springer Berlin Heidelberg, 2008 (EXamen. press Series). – ISBN 9783540797432

[Szegedy u. a. 2014] Szegedy, Christian ; Liu, Wei ; Jia, Yangqing ; Sermanet, Pierre ; Reed, Scott E. ; Anguelov, Dragomir ; Erhan, Dumitru ; Vanhoucke, Vincent ; Rabinovich, Andrew: Going Deeper with Convolutions. In: *CoRR* abs/1409.4842 (2014). – URL http://arxiv.org/abs/1409.4842

[Tang 2013] Tang, Yichuan: Deep Learning using Support Vector Machines. In: *CoRR* abs/1306.0239 (2013). – URL http://arxiv.org/abs/1306.0239

[Tsardoulias u. a. 2014] Tsardoulias, E. ; Psomopoulos, F. ; Trochidis, I. ; Gkiokas, A. ; Arabatzis, S. ; Szynkiewicz, W. ; Prunet, V. ; Daney, D. ; Kasprzak, W.: *RAPP - State-of-the-art report [CERTH]*. May 2014

[Zeiler 2014] Zeiler, Matthew: *Hierarchical Convolutional Deep Learning in Computer Vision*, New York University, Dissertation, 2014

[Zeiler und Fergus 2013] Zeiler, Matthew D. ; Fergus, Rob: Visualizing and Understanding Convolutional Networks. In: *CoRR* abs/1311.2901 (2013). – URL http://arxiv.org/abs/1311.2901

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24. Mai 2016 Sebastian Böhle