TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik

Institut für Automatisierungstechnik

Diplomarbeit

zum Thema

Strukturbasierte Multi-View-Erkennung von 3D Objekten mit traditionellen und Deep-Learning Methoden

vorgelegt von Mo Li
im Studiengang AMR, Jg. 2011
geboren am 10.12.1986 in China

zur Erlangung des akademischen Grades eines

Diplomingenieurs
(Dipl.-Ing.)

Betreuer: Msc.-Ing. Fabio Bracci
Dr.-Ing. Zoltán-Csaba Márton
Verantwortlicher Hochschullehrer: Prof. Dr. techn. Klaus Janschek
Tag der Einreichung: 11.06.2018

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik

Institut für Automatisierungstechnik

Aufgabenstellung für die Diplomarbeit

Für: Frau Mo Li

Studiengang: Automatisierungs-, Mess- und Regelungstechnik

Thema: **Strukturbasierte Multi-View-Erkennung von 3D Objekten mit traditionellen und Deep-Learning Methoden**

Die traditionelle und Deep-Learning 3D-Objekterkennung verfolgt den Ansatz der Bestimmung des globalen Deskriptors und der darauf basierenden Klassifizierung. Die Idee für diese Diplomarbeit ist, die geometrischen Beziehungen der Deskriptoren durch die Erstellung eines Deskriptor-Baums zu kodieren, und eine spezifische Matching-Methode für solche strukturierten Daten anzuwenden.

Der Deskriptor-Baum stellt die hierarchische Zusammensetzung von glatten Objektteilen dar, so ähnlich wie die "geometric icons (geons)“ der Gestaltpsychologie-Theorie in Biedermanns „Recognition-by-components theory of human perception". Im Deskriptor-Baum können die Deskriptoren entweder traditionell (VFH) oder Deep-learning (Autoencoder) Deskriptoren sein. In der Arbeit wird eine Untermenge der SHREC 3D-Objektdatenbank zur Auswertung verwendet, und es werden mehrere 2.5D-Ansichten der Objekte betrachtet.

Alle Ergebnisse dieser Arbeit sind vollständig und nachvollziehbar zu dokumentieren. Ergebnisse aus Fremdquellen müssen nach den üblichen Zitierregeln eindeutig gekennzeichnet werden.

Im Rahmen der Arbeit sollen folgende Teilaufgaben bearbeitet werden:

1. Einarbeitung in die verwendete Software
2. Strukturierte Anforderungsdefinition
3. Überblick über den aufgaben-bezogenen Stand der Wissenschaft und Technik sowie Abgrenzung des eigenen Beitrags
4. Erfassung einer "Baseline" zur späteren Auswertung des eigenen Beitrags
5. Erfassung eine vorbereitende Analyse der Zerteilung eines 3D Objektes
6. Entwurf einer strukturbasierten Repräsentation eines 3D Objektes
7. Implementierung der in Teilaufgabe 6 entworfenen Deskriptor im Rahmen der PCL Library
8. Test und Auswertung der entwickelten Repräsentation mittels Daten von der SHREC 2010 Datensatz und die Baseline aus Teilaufgabe 4
9. Dokumentation der Ergebnisse

Betreuer: Fabio Bracci, DLR
Betreuer: Dr. Zoltán-Csaba Márton, DLR
Ausgehändigt: 01.01.2018 Einzureichen: 10.06.2018

Prof. Dr. techn. K. Janschek
Verantwortlicher Hochschullehrer

TECHNISCHE UNIVERSITÄT DRESDEN

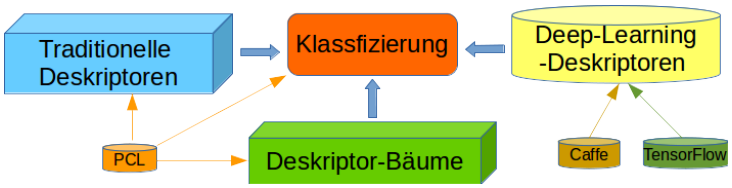
Fakultät Elektrotechnik und Informationstechnik

Institut für Automatisierungstechnik

Strukturbasierte Multi-View-Erkennung von 3D Objekten mit traditionellen und Deep-Learning Methoden

Diese Arbeit konzentriert sich auf 3D-Objekterkennung und Klassifizierung beim begrenzten Fall von 3D-Daten in Roboteranwendungen. Das Ziel dieser Arbeit war es, die Beschreibungsfähigkeit der Deskriptoren traditioneller Algorithmen und der Deep-Learning Methoden zu bewerten. Des Weiteren wurden ein Deskriptor-Tree umgesetzt, wodurch die geometrischen Beziehungen der Deskriptoren strukturiert abgelegt werden können. Zur Visualisierung wurden die traditionellen und Deep-Learning Deskriptoren mithilfe von Nächste-Nachbarn-Klassifikation geschätzt und klassifiziert. Beim Ansatz vom 3D Baum wurde die Klassifizierung durch Berechnung der Entfernung der Bäume eingeführt.

Nach dem Testen wird es so festgestellt, dass die Leistung der mit Deep-Learning, CaffeNet, trainierten Deskriptoren am Besten ist, während die Leistung der in dieser Arbeit implementierten Deskriptor-Bäume in der Mitte dazwischen liegt. Schließlich kommt der traditionelle Teil.



Betreuer:	Msc.-Ing. Fabio Bracci
	Dr.-Ing. Zoltán-Csaba Márton
Hochschullehrer:	Prof. Dr. techn. Klaus Janschek
Tag der Einreichung:	11.06.2018

DIPLOMARBEIT

Bearbeiter: Mo Li

TECHNISCHE UNIVERSITÄT DRESDEN

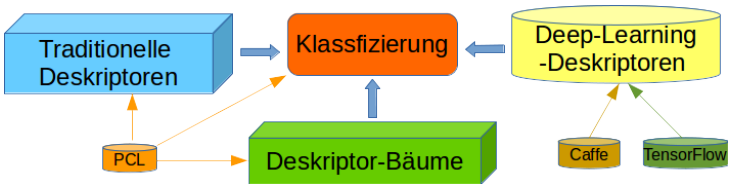
Fakultät Elektrotechnik und Informationstechnik

Institut für Automatisierungstechnik

Structure -based multi-view recognition of 3D objects using traditional and deep learning approches

This thesis focuses on 3D object recognition and classification in the limited case of 3D data in robotic applications. The aim of this work was to evaluate the descriptiveness of the descriptors of traditional algorithms and deep-learning methods. Furthermore, a descriptor tree was implemented, whereby the geometric relationships of the descriptors can be filed in a structured manner. For visualization, the traditional and deep-learning descriptors were estimated and classified using nearest-neighbor classification. In the approach of the 3D tree, the classification was introduced by calculating the distance of the trees.

After the testing it is determined that the performance of the deep-learning, CaffeNet, trained descriptors is best, while the performance of the descriptor trees lies in the middle. Finally comes the traditional part.



Tutor:	Msc.-Ing. Fabio Bracci
	Dr.-Ing. Zoltán-Csaba Márton
Supervisor:	Prof. Dr. techn. Klaus Janschek
Day of Submission:	11.06.2018

DIPLOMA THESIS

Author: Mo Li

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

1 Einleitung

Mit der steigenden Nachfrage nach automatischer und zuverlässiger Objekterkennung hat die Erfassung und Analyse von 3D-Sensordaten eine immer größere Bedeutung erlangt und wird immer weiter angewendet, beispielsweise zur Kollisionsvermeidung von Fahrzeugen, Flugzeugen und mobilen Robotern, sowie zur Optimierung der Annäherungsbewegung bei automatischen Andockmanövern.

In Vergleich zur 2D-Bildverarbeitung hat 3D-Objekterkennung mehrere Vorteile und rückt als Forschungsthema mehr in den Fokus, da bei der Auswertung von 3D-Daten die räumliche Objektgeometrie nicht rekonstruiert werden muss, sondern in den Daten unmittelbar vorliegt. Die Objekterkennung wird dadurch erleichtert, dass eine Segmentierung der 3D-Daten und eine Bewertung von Formmerkmalen unabhängig von den Beleuchtungsverhältnissen und Farbkontrasten möglich ist.

Kostengünstiger Bildsensoren, die Tiefendaten liefern, sind heute für 3D-Daten verfügbar. Mit Hilfe von 3D-Sensoren kann aus der Verbindung der gewonnenen 3D-Informationen eine sogenannte 3D Punktewolke generiert werden, welche die Basis für jegliche weiterführende Bildanalysen darstellt.

1.1 Motivation

Um 3D Objekte zu erkennen, gibt es 2 Methoden: die erste ist Matching durch lokale Deskriptoren; die zweite Methode ist das Matching durch globale Deskriptoren. Diese Arbeit behandelt hauptsächlich globale Deskriptoren. Globale Deskriptoren kodieren die Objektgeometrie. Sie werden nicht für einzelne Punkte berechnet, sondern für einen ganzen Cluster, der ein Objekt darstellt. Globale Deskriptoren werden zur Objekterkennung und Klassifizierung, zur geometrischen Analyse (Objekttyp, Form) und zur Positionsschätzung verwendet. Die traditionelle globale Deskriptoren können mit Hilfe Points Cloud Liabrary(*kurz.* PCL)[4] bearbeitet werden. Sie basieren auf 3D Punktwolken, daher werden sie sich auch als punktbasierte Deskriptoren bezeichnet.

Deep Learning ist aktuell sehr populär und erfolgreich. Ein wichtiger Grund ist, dass Deep Learning mit Rohdaten beginnen kann, da die Deskriptoren beim Lernen automatisch vom neuronalen Netzwerk erstellt werden und die

Zielfunktion aus den Daten gelernt werden kann. Deep Learning verschiebt die Last des Feature-Designs zu dem zugrundeliegenden Lernsystem und Klassifikationslernensystem, das typisch für ein mehrschichtiges neuronales Netzwerklernen ist. Im DLR beim RMC Institut werden Variation-Auto-Encoder (VAE)[3] als eine Deep-Learning Klassifiktorentrainingsmethode eingesetzt.

Basierend auf diesen Algorithmen macht es Sinn, die Beschreibungsfähigkeit der traditionellen punktbasierten Deskriptoren und die der vortrainierten oder trainierten (auf einem spezifischen Datensatz) Deep-Learning-Deskriptoren zur 3D-Objekterkennung zu untersuchen und zu vergleichen.

Außerdem bei den teil-basierte Deskriptoren werden ein Objekt mit mehreren unterteilten Teilen beschrieben werden. Anstelle eines traditionellen Objekts, das einem extrahierten Deskriptor entspricht, entspricht ein Objekt teil-basierte Deskriptoren, bei denen einer Deskriptoren-Liste, welche ungeordnet gespeichert werden. Daher ist es sinnvoll, eine neuen Ansatz vorzuschlagen, damit eine optimale Lösung gefunden werden kann, um das Problem der ungeordneten Anordnung von unterteilen Teilen in teil-basierten Algorithmen zu lösen.

1.2 Aufgabenstellung

Diese Arbeit ist hauptsächlich in zwei Aufgaben unterteilt. Erstens die Beschreibungsfähigkeit der Features traditioneller Algorithmen und der Deep-Learning Methoden zu vergleichen und zu bewerten. Das zweite Ziel dieser Arbeit ist die Umsetzung eines Deskriptor-Trees, wodurch die geometrischen Beziehungen der Deskriptoren strukturiert abgelegt werden können. Stand der Technik ist es, dass mithilfe von Nächste-Nachbarn-Klassifikation[13] die VFH, CVFH und OURCVFH Deskriptoren geschätzt und klassifiziert werden. Wird ein 3D Baum als Ansatz gewählt, so können die Deskriptoren durch Berechnung der Entfernung der Bäume klassifiziert werden.

Ein anderes Problem bei Deep-Learning ist, dass End-to-End Deep-Learning für 3D-Objekte möglich, aber kompliziert ist, da die Oberflächen mit Hilfe von Differentialgeometrie betrachtet werden müssen. Daher werden wird versucht, die Struktur anders zu betrachten.

1.3 Veröffentlichung

Die Ergebnissen aus dieser Arbeit werden teilweise für den Artikel „Applicability of Deep Learned vs Traditional Features for Depth Based Classification“[2] verwendet und wurden nach Peer-Review zum veröffentlichen im CompImage'18 Konferenz akzeptiert.

1.4 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 Aufgabenstellung dieser Arbeit.

Kapitel 2 erläutert die Anforderungsdefinition, die die Systemanforderungen betrachtet.

Kapitel 3 stellt die Grundlagen der Objekterkennung, der 3D-Daten sowie von Deep Learning vor.

Kapitel 4 erläutert den Entwurf der Systeme, die dazugehörigen Architekturen sowie die Komponenten.

Kapitel 5 dokumentiert die Umsetzung des in Kapitel 4 vorgestellten Lösungsvorschlages.

Kapitel 6 validiert die Implementierung und bewertet die Ergebnisse.

Kapitel 7 fasst den Inhalt der Arbeit zusammen.

2 Anforderungsdefinition

Die Anforderungsspezifikation steht am Anfang des Entwicklungsprozesses. Sie bildet die nötige Grundlage für die Entwicklung eines technischen Systems und ist von großer Bedeutung für den Erfolg dieser Entwicklung.

2.1 Nutzeranforderungen

2.1.1 Zielsetzung

Bei 3D Objekterkennung wird oft durch das Konzept der Punktwolken, einer Ansammlung von Punkten die Darstellung einer Oberfläche, realisiert.

Objekte können durch ihre geometrischen Eigenschaften unterschieden werden, die mittels mehrerer Deskriptoren kodiert werden. Neuerdings aus dem Bereich der Neuronalen Netze der sogenannten Deep Learning-Techniken haben an Dynamik gewonnen, von denen einige Deskriptoren lernen können, die für den gleichen Zweck geeignet sind.

In der Arbeit wird die Beschreibungsfähigkeit traditioneller punktbasierter Deskriptoren untersucht, ebenso die vortrainierten oder trainierten (auf einem spezifischen Datensatz) tief erlernten Deskriptoren zur 3D-Objekterkennung, abhängig von der Formkomplexität.

Es wird überprüft, ob die Funktion Sätze, die aus einem DNN extrahiert werden, sich am besten zur Beschreibung der 3D-Objektform eignet.

Zusätzlich wird ein Objekt für einige Algorithmen in einigen Teilen unterteilt und die extrahierten Deskriptoren werden ungeordnet gespeichert. Daher ist es sinnvoll, eine neue Methode vorzuschlagen, um eine optimale Lösung des Problems der ungeordneten Anordnung von Unterteilungen in einigen Algorithmen zu finden.

2.1.2 Softwareumgebung

Point Cloud Library(PCL)[4] ist ein 3D-Bildbearbeitung Tool und enthält zahlreiche Algorithmen zur Verarbeitung n-dimensionaler Punktwolken und 3D Geometrien. Diese Arbeit wird meisten in PCL implementiert, während der deep-Learning Teil in Deep-Learning Umgebung realisiert wird.

2.1.3 Funktionale Anforderungen

1 Kennenlernen aller Algorithmen: traditionelle Merkmalsextraktionsalgorithmen und Deep-Learning-Algorithmen

2 Analysiere das Konzept, eine Baum-Struktur zu erstellen, einen Schnittstandard zu etablieren, den Schnittalgorithmus zu verstehen,

3 Vertraut mit der Entwicklungsumgebung, der bereits benötigten Software und ihrer Bibliothek

4 Entwickeln des Klassifizierungsteiles und seine adaptiven Übertragungsports für Bewertung der unterschiedlichen Deskriptoren

5 Erstellen des Baumteiles auf PCL, realisieren des Schneidezyklus, die Zugehörigkeit des bereits zugewiesenen Endpunktes

6 Entwickeln eines Klassifikators für Deskriptor-Bäume, die Entwicklung erfolgt auf PCL.

2.2 Strukturierte Analyse

Die strukturierte Analyse definiert Vorgehensmodelle und Bausteine, um das komplexe Posenbestimmungssystem durch hierarchische Dekomposition in Subsysteme zu zerlegen. Die Struktur des Systems wird ermittelt, indem Eingaben auf Ausgaben abgebildet werden.

2.2.1 Kontextdiagramm

Ein Kontextdiagramm dient der Modellierung einer Systemumgebung in einer frühen Entwurfs- oder Analysephase.Das Kontextdiagramm stellt die oberste Hierarchieebene von Datenflussdiagrammen dar.Es handelt sich um ein abstraktes Datenflussdiagramm, mit dem die Schnittstellen des Systems zu dessen Umwelt abgebildet werden. Im folgenden Daten- und Steuerkontextdiagramm dieser Arbeit wird Eingänge und Ausgang der Schnittstelle von dem Projekt, siehe Abbildung 2.1. Tabelle 2.1 beschreibt die angeforderten Datenflüsse.

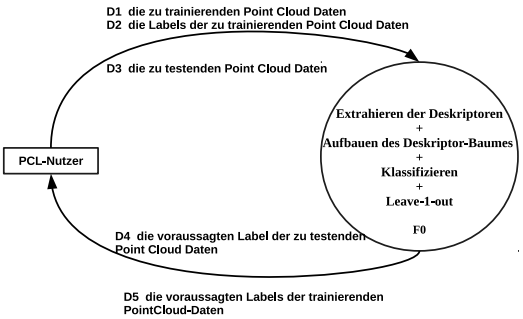


Abbildung 2.1: Kontextdiagramm

Tabelle 2.1: Kontextdiagramm - Daten- und Steuerflüsse

Datenfluss	**Beschreibung**
D1 die zu trainierenden Point Cloud Daten	die Trainingsdaten, PCD Form, um gute Klassifikatoren zu lernen
D2 die Labels der zu trainierenden Point Cloud Daten	die Labels für jedes Trainingsdaten
D3 die zu testenden Point Cloud Daten	die Testdaten in PCD Form, um die gelernten Klassfikatoren zu testen
D4 die Labels der getesten Point Cloud Daten	die Labels für die getesten Daten, die bei den gelernten Klassfikatoren vorausgesagten werden
D5 die Labels der trainierten Point Cloud Daten	die Labels der Trainingsdaten, die bei den gelernten Klassfikatoren vorausgesagten werden.

2.2.2 Datenflussdiagramm - Ebene 1

Die Aufgabe wird in mehrere Teilaufgaben zerlegt, die Abbildung 2.2 zeigt die Komposition auf Ebene 1. Die Tabellen 2.2 und 2.3 beschreiben die enthaltenen Funktionen und Datenflüsse.

2.2.3 Datenflussdiagramm - Ebene 2 - Funktion 1

Die im Abschnitt 2.2.2 dargestellten Funktion F1 wird weiter detailliert, um das Teilsystem deutlich zu machen.

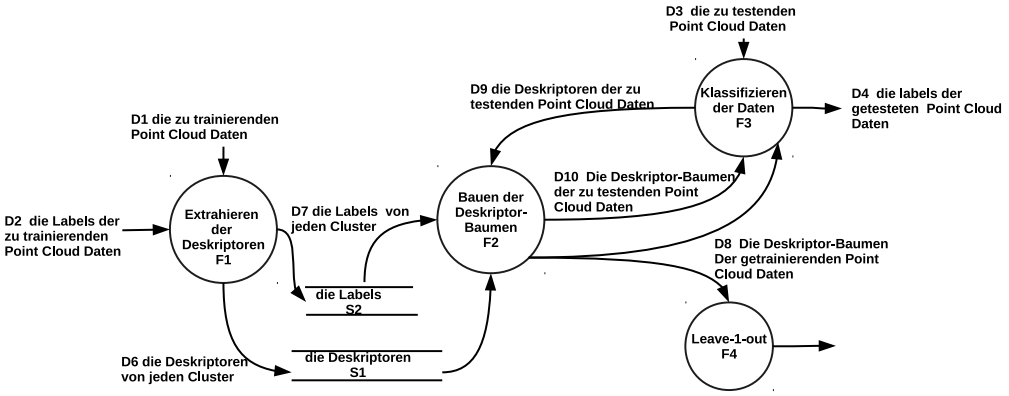


Abbildung 2.2: Datenflussdiagramm - Ebene 1

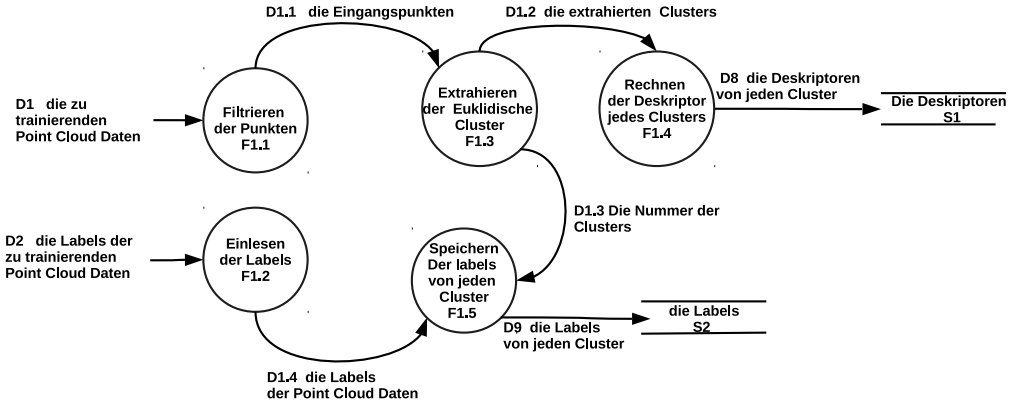


Abbildung 2.3: Datenflussdiagramm - Ebene 2 - Funktion 1

2.2.4 Datenflussdiagramm - Ebene 2 - Funktion 3

Abbildung 2.4 stellt die Dekomposition der Funktion: F2 dar. Die Tabellen 2.8 und 2.9 beschreiben die enthaltenen Subfunktionen und Datenflüsse.

2.2.5 Datenflussdiagramm - Ebene 2 - Funktion 4

Tabelle 2.2: Datenflussdiagramm - Ebene 1 - Funktionen

Funktion	**Beschreibung**
F1 Extrahieren der Deskriptoren	Die Objekt ist in mehrere Cluster unterteilt und die Deskriptoren von jedem Cluster werden berechnet.
F2 Erzeugen des Dekriptor-Baumes	Die Dekriptor-Baum wird erzeugt
F3 Klassfizieren der Daten	Die Testdaten werden nach den gelernten Klassfikatoren klassfiziert und bekommen die prognostizierten Labels
F4 Leave-1-out	die Trainingdaten werden nach dem gelernten Klassifkatoren klassifizieren und bekommen die prognostizieren Label

Tabelle 2.3: Datenflussdiagramm - Ebene 1 - Datenflüsse

Datenfluss	**Beschreibung**
D6 die Deskriptoren von jedem Cluster	die Deskriptoren für jedes unterteiltes Cluster
D7 die Labels von jeden Cluster	die Labels für jedes erzeugte Cluster
D8 die Deskriptor-Bäume der getrainierenden Point Cloud Daten	die Bäume von jeden Objekt
D9 die Deskriptoren der zu testenden Point Cloud Daten	die Labels für jedes erzeugte Cluster
D10 die Deskriptor-Bäume der zu testenden Point Cloud Daten	die Bäume von jedes Objekt

Tabelle 2.4: Datenflussdiagramm - Ebene 2 - F1 - Funktionen

Funktion	Beschreibung
F1.1 Filtrieren der Punkten	Die Normale mit höchsten Krümmung werden filtert
F1.2 Laden der Labels	Die Labels der Point Clouds werden heruntergeladen
F1.3 Extrahieren der Euklidische Cluster	Das Objekt wird mit bestimmtet Euklidischer Distanz unterteilt
F1.4 berechnen der Deskriptor jedes Clusters	Die Dekriptoren werden nach den unterteilten Cluster berechnet
F1.5 Speichern der Labels von jeden Cluster	Entsprechend der Cluster Nummer werden die Labels gepeichert.

Tabelle 2.5: Datenflussdiagramm - Ebene 2 - F1 - Datenflüsse

Daten- oder Steuerfluss	Beschreibung
D1.1 die Eingangspunkte	die Punkte in der PointCloud nach dem Filteriren
D1.2 die extrahierten Cluster	die unterteilten Clusters
D1.3 die Nummer der Cluster	Die Anzahl der Cluster
D1.4 die Labels der Point Cloud Daten	Die Labels von jeder PointCloud

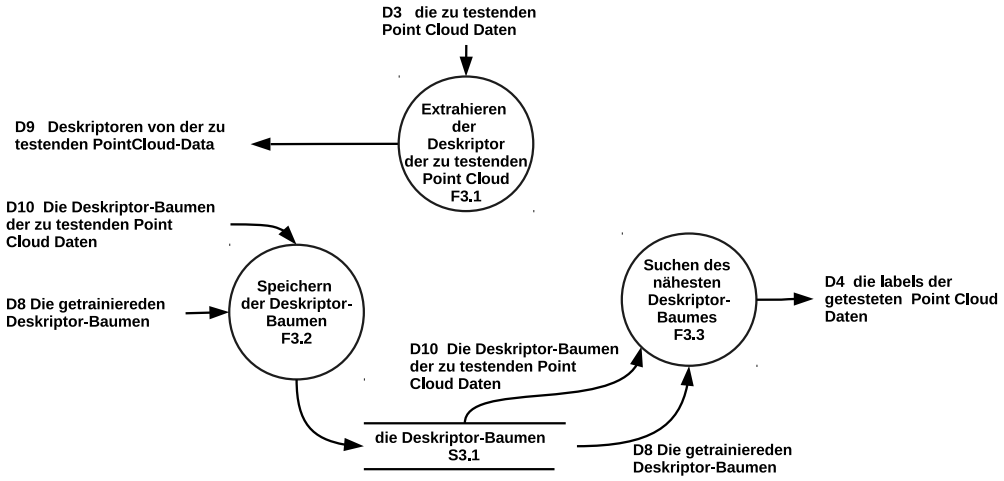


Abbildung 2.4: Datenflussdiagramm - Ebene 2 - Funktion 3

Tabelle 2.6: Datenflussdiagramm - Ebene 2 -F3 - Funktionen

Funktion	Beschreibung
F3.1 Extrahieren der Deskriptoren der zu testenden Point Cloud	Die Deskriptoren von der PointCloud, die als Testdaten vorhanden sind, werden berechnet.
F3.2 speichern der Deskriptor-Bäume	die gebauet Deskriptor-Baumen werden gespeichert.
F3.3 Suchen des nähesten Deskriptor-Baumes	Die nähesten Deskriptor-Baum in die gepeicherten getrainierten Bäume wird gesucht.

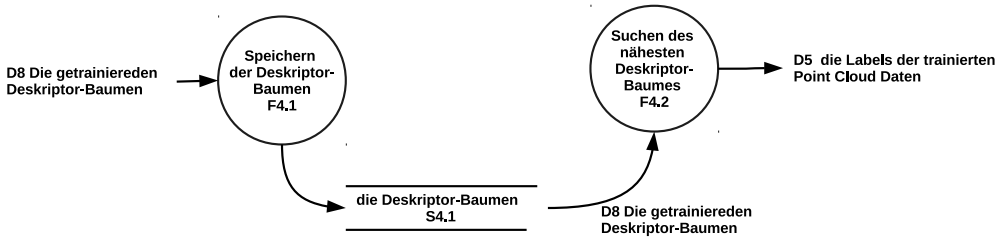


Abbildung 2.5: Datenflussdiagramm - Ebene 2 - Funktion 4

Tabelle 2.7: Datenflussdiagramm - Ebene 2 - F3 - Funktionen

Funktion	Beschreibung
F4.1 Speichern der Deskriptor-Baumen	die gebauet Deskriptor-Baumen werden gespeichert.
F4.2 Suchen des nähesten Deskriptor-Baumes	Die nähesten Deskriptor-Baum in die gepeicherten getrainierten Baum wird gesucht.

3 Grundlagen

Das Kapitel stellt die theoretischen Grundlagen, auf denen dieser Arbeit basiert.

3.1 Software

In diesem Abschnitt wird die in dem recherchierten Projekt verwendete Software kurz eingeführt:

- `Point Cloud Library[4]` ist ein 3D-Bildbearbeitung Tool und hat inzwischen einen ähnlichen Status wie OpenCV für die 2D-Bildverarbeitung erreicht. PCL enthält zahlreiche Algorithmen zur Verarbeitung n-dimensionaler Punktwolken und 3D Geometrien.
- `Caffe[14]` ist eine Programmbibliothek für Deep Learning. Caffe enthält zahlreiche Algorithmen und Deep-Learning-Architekturen für die Klassifikation und Clusteranalyse von Bilddaten. CNN, R-CNN (Rekurrentes convolutional neuronales Netz), LSTM (Long short-term memory) und vollständig verbundene neuronale Netze werden unterstützt
- `Tensorflow[15]` ist eine plattformunabhängige Open-Source Programmbibliothek für künstliche Intelligenz bzw. In der Forschung und im Produktivbetrieb wird sie derzeit von verschiedenen Teams in kommerziellen Google-Produkten wie Spracherkennung, Gmail, Google Fotos und der Google Suche verwendet[2]. So wird der Kartendienst Maps durch Analyse der von Street View aufgenommenen Fotos, die mit Hilfe einer auf TensorFlow basierenden KI analysiert werden, verbessert

3.2 3D-Objekterkennung

Um das Vorhandensein eines Objektes in einem Bild oder einem Bereich auch dessen Position zu bestimmen, beschreibt der Begriff Objekterkennung Verfahren zum Identifizieren eines bekannten Objekts innerhalb eines Objektraums. Wenn die zu identifizierenden Objekten sowie auch die Umgebung mithilfe von 3D Daten repräsentiert werden, heißt es 3D-Objekterkennung.

In der Bildverarbeitung dienen Objekterkennungsmethoden dazu, bestimmte Objekte bzw. Muster von anderen Objekten zu unterscheiden. Dazu muss das eigentliche Objekt zunächst mathematisch beschrieben werden.

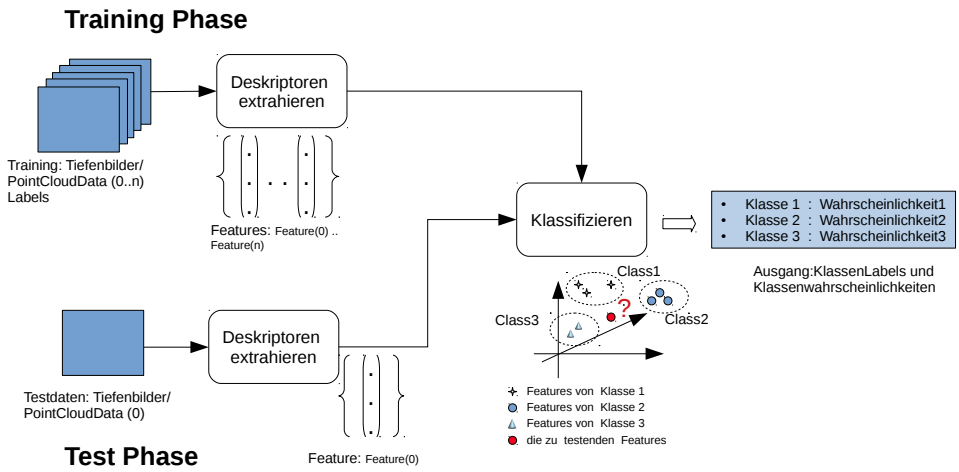


Abbildung 3.1: Traditionelles Objekterkennungssystem, das aus drei Teile besteht: aus Training-, Test-Phase und Klassifizierung

Bei 3D Daten, Punktwolken, oder normalen RGB-Bild werden nur Informationen jedes Punktes oder Pixels gesammelt, die die Position (x,y) oder auch Tiefeninformation (z) repräsentieren können. Deshalb ist es notwendig, mit besonderen Algorithmen das ganze Objekt zu beschreiben. Die Beschreibung des Objektes kann man als Feature oder Deskriptor annehmen. Die zusätzliche Segmentierung, Resampling, Registrierung und Oberflächenrekonstruktion, sowie Klassifikation und andere Verarbeitungsalgorithmen basieren alle auf der Feature Beschreibung. Normalerweise, je genauer die Beschreibung des Objektes möglich ist und je mehr auswertbare Informationen vorhanden sind, umso zuverlässiger arbeitet die Objekterkennung.

Das gesamte Objekterkennungssystem enthält insgesamt 3 Abschnitte:

Training-, Test-Phase, und Klassifikation, wie in Abbildung 3.1 gezeigt. In der Training-Phase werden Features extrahiert und in einer Datenbank gespeichert, die schließlich als eine Klassifikation-Datenbank zur Klassifizierung verwendet werden. Bei der Test-Phase handelt es sich um Test-Daten. Wird ein zu testendes Objekt eingegeben, können die Features des Testobjektes mit der Deskriptor-Datenbank verglichen werden, dadurch kann die passende Klasse des Testobjektes herausgefunden werden. Dies ist der letzte Schritt in der Objekterkennung: Klassifizieren.

Im Allgemeinen gibt zur Objekterkennung es einige wichtige Schritte, die im Folgenden zusammengefasst werden.

1. `Deskriptor Extraktion`: Auf einer Reihe von Trainingsbildern werden die lokalen Merkmale extrahiert.

2. `Segmentierung`: Die Erzeugung von inhaltlich zusammenhängenden Regionen durch Zusammenfassung benachbarter Pixel oder Voxel entsprechend einem bestimmten Homogenitätskriterium bezeichnet man als Segmentierung.

3. `Klassifikation` : eine planmäßige Sammlung von abstrakten Klassen (auch Konzepten, Typen oder Kategorien), die zur Abgrenzung und Ordnung verwendet werden.

Die obige Aussage kann als der traditionelle Identifikationsprozess betrachtet werden. Gegenwärtig wird eine sehr populäre Methode, der Deep-Learning, in 3D-Objekterkennungsbereich weitbreit verwendet. Deep Learning bezeichnet eine Klasse von Optimierungsmethoden künstlicher neuronaler Netze, die zahlreiche Zwischenschichten (engl. hidden layers) zwischen Eingabeschicht und Ausgabeschicht haben und dadurch eine umfangreiche innere Struktur aufweisen. Deep-Learning verwendet künstliche neuronale Netze, in denen sich zwischen den Input- und Output-Schichten viele verborgene Schichten befinden[16][17][18].

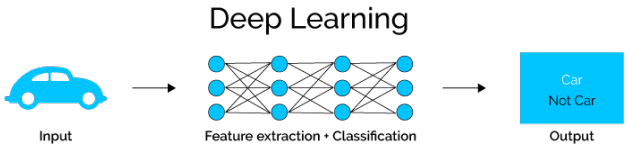


Abbildung 3.2: Deep-Learning System, ein „End-to-End-Lernvorgang“[19]

Bei der traditionellen Objekterkennung beginnt es damit, dass relevante Merkmale manuell aus Bildern extrahiert werden. Anhand der Merkmale wird dann ein Modell erstellt, das die Objekte im Bild kategorisiert. In einem Deep-Learning-Workflow werden relevante Merkmale automatisch aus Bildern extrahiert. Außerdem wird beim Deep Learning ein „End-to-End-Lernvorgang“ durchgeführt. Dabei erhält ein Netz Rohdaten und eine Aufgabe, etwa eine Klassifikation, und lernt, diese Aufgabe automatisch zu erledigen, wie in Abbildung 3.2 gezeigt.

3.3 3D Daten

Unter 3D-Daten versteht man Punkte, die in einem kartesischen 3D Koordinatensystem durch drei Koordinaten p = (x,y,z) identifiziert werden, die zusammengenommen einem Punkt im Raum relativ zu einem Ursprungspunkt entsprechen. X-, Y- und Z- Achsen erstrecken sich in zwei Richtungen und die Koordinaten identifizieren den Abstand des Punktes vom Schnittpunkt der Achsen.

In der Arbeit wird eine Untermenge der SHREC 2010 Datensatz, the Shape Retrieval Contest of Range Scans [10], zur Auswertung verwendet und es werden mehrere 2.5D-Ansichten der Objekte betrachtet. In Bezug auf die 2.5D-Ansicht kann verstanden werden, dass bei Datenmodellen die dritte Dimension nicht vollwertig bezogen auf die 2D-Lageinformation gespeichert ist, da nicht alle 3D-Formen modelliert werden können. Obwohl das 3D-Modell Positions- und Höheninformationen enthält, kann der 3D-Sensor beispielsweise keine Daten über die Rückseite des Objekts sammeln.

3.3.1 SHREC 2010 Datensatz

SHREC 2010 - Shape Retrieval Contest of Range Scans ist ein 3D-Objekt Datensatz . Die Datensatz enthält 800 vollständige 3D-Modelle, die in 40 Klassen eingeteilt sind. In jeder Klasse gibt es 20 Modelle. Die Entfernungsbilder werden mit einem Minolta Laser Scanner aufgenommen. Das Dateiformat ist im ASCII-Objektdateiformat. Die Aufnahme der Daten erfolgt von vier unterschiedlichen Beobachtungswinkel für jeden Objekte. Die vier Beobachtungswinkel enthalten Ansichten von oben, von hinten, von rechts und von links. Ein Beispiel für aufgenommene Objekte ist in Abbildung 3.3 gezeigt.

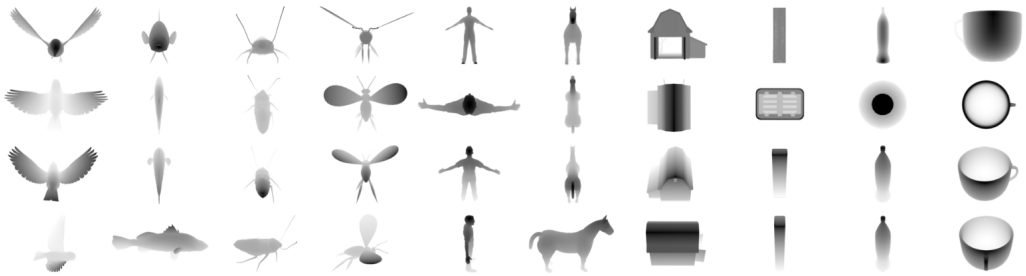


Abbildung 3.3: Beispiel Objektansichten, die aus dem SHREC 2010 Datensatz stammen. Die Zeilen in der Tabelle sind von oben nach unten in vier Ansichten: Ansicht von oben, Ansicht von hinten, Ansicht von rechts und Ansicht von links[1].

3.3.2 Point Cloud

Eine PointCloud ist eine Menge von Punkten eines Vektorraums, die Punkte im dreidimensionalen Koordinatensystem enthalten. Eine Punktwolke ist durch die enthaltenen Punkte beschrieben, die jeweils durch ihre Raumkoordinaten erfasst sind.

Abbildung 3.4: Die PointCloud für eine Tasse aus zwei unterschiedlichen Winkeln

Die Punktwolke ist eine sehr gute digitale Darstellung eines Objekts oder eines Raums. Es wird in Form einer sehr großen Anzahl von Punkten gespeichert, die die Oberflächen eines bestimmten Objekts repräsentieren. Punkte einer Punktwolke befinden sich auf den äußeren Oberflächen von sichtbaren Objekten. Die Punkte werden von einem Lichtstrahl des Scanners von einem Objekt reflektiert. 3D-Scanner erfassen Punktmessungen von realen Objekten oder Fotos für eine Punktwolke, die dann in ein 3D-Netz oder CAD-Modell übersetzt werden können. Wie in Abbildung 3.4 gezeigt sind Punktwolken aus zwei unterschiedlichen Winkeln für eine Tasse erforderlich .

Die PointCloud wird im PCD-Format gespeichert und kann mit dem

PCL-Tool einfach gelesen werden. PCL stellt eine Vielzahl von vordefinierten Punkttypen zur Verfügung, um Punkte mit verschiedenen Parametern darzustellen.Diese Typen sollten ausreichen, um die meisten der in PCL implementierten Algorithmen und Methoden zu unterstützen. Der einfachste und am häufigsten verwendete Datentyp ist *PointXYZ*, der XYZ-3D-Daten enthält. Der Datentyp *PointXYZRGB*, die die Elemente x, y, z und rgb enthalten, stellen RGB-Informationen dar. Ein in dieser Arbeit verwendeter allgemeiner Datentyp ist *PointNormal*, der XYZ-Daten mit Oberflächennormalen (dx, dy, dz) und Krümmung kombiniert.

3.4 3D Deskriptor

Lokale Merkmale und ihre Deskriptoren sind die Basis vieler Computer Vision Algorithmen. Es handelt sich am Bildregistrierung, Objekterkennung und Klassifizierung sowie Bewegungsschätzung. Merkmalsextraktion ist eine Art von Mehrdimensionalität-Reduktion, mit einer niedrigen Mehrdimensionalität die 3D Objekte beschreiben. Eine reduzierte Feature-Repräsentation stellt interessante Teile des Objektes als kompakten Merkmalsvektor dar. Sie ist erforderlich, um die Bilder schnell zu identifizieren und zu klassifizieren.

Die Feature Beschreibung und Extraktion von 3D Punktwolken, die der grundlegendste und wichtigste Teil der Verarbeitung von Punktwolkeninformationen ist, ist die Basic von Punktwolkenerkennung, Segmentierung, Resampling, Registrierung Oberflächenrekonstruktion und andere Verarbeitungsalgorithmen. Sie hängt stark von den Ergebnissen der Feature-Beschreibung und Extraktion ab.

3.4.1 Histogramm Deskriptor

Diese Arbeit verwendet eine Anzahl von 3D-Oberflächen Deskriptoren, die Histogramme verwenden, um die verschiedenen Eigenschaften der lokalen Oberfläche darzustellen. Sie beschreiben lokale Oberflächen, indem sie geometrische oder topologische Messungen im Histogrammen akkumulieren. Daher heißen sie Histogramm-basierte Deskriptoren mit geometrischen Attributen.

Ein Beispiel wird in Abbildung 3.5 gezeigt. Die rechte Punktwolke besteht aus 817909 Punkten, die durch (x, y, z) definiert sind und der auf der linken Seite extrahierte Histogramm-Deskriptor kann den Becher mit 308 Vektorwerten beschreiben.

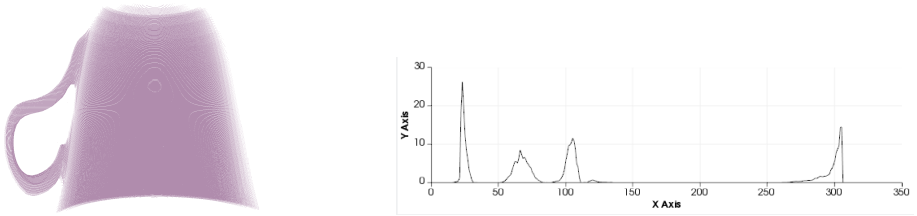


Abbildung 3.5: Histogramm Deskriptor: Links ist eine PointCloud für eine Tasse; Rechts ist der entsprechenden Histogramm-Deskriptor

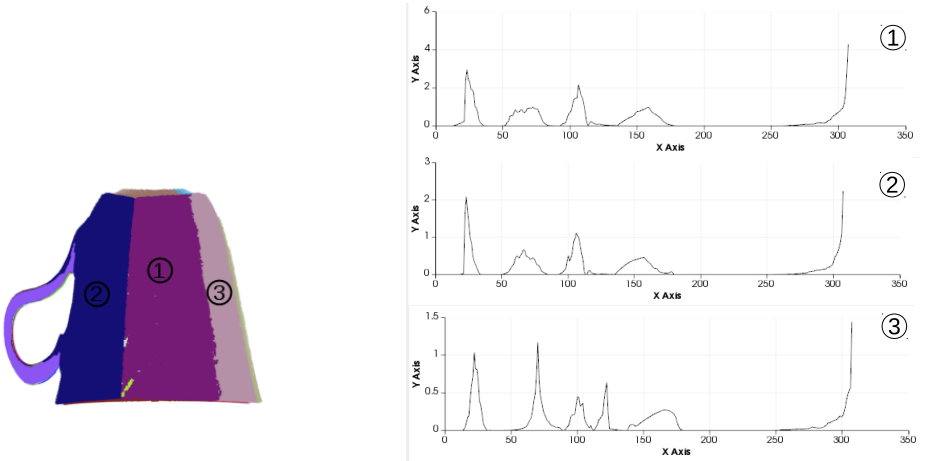


Abbildung 3.6: teil-basierter Histogramm Deskriptor: Links ist eine in 24 Teile segmentierte PointCloud für die in Abbildung 3.5 gezeigte Tasse; Rechts ist die entsprechende teil-basierte Histogramm-Deskriptor-List, die aus Teilen mit derselben Nummer extrahiert, wurde.

3.4.2 teil-basierter Deskriptor

Unter der Definition kann es so vertanden werden, dass ein Objekt mithilfe einzelner geschnittener Teile beschrieben wird. Anstelle eines traditionellen Objekts, das einem extrahierten Deskriptor entspricht, entspricht ein Objekt mehreren Deskriptoren, die jeweils einem unterteilten Teil entsprechen. Um einen teil-basierten Deskriptor zu extrahieren, sollte das Objekt zuerst unterteilt werden.

Wie in Abbildung 3.6 gezeigt, ist die Tasse aus Abbildung 3.5 in 24 Teile unterteilt, die mit dem PCl-Viewer visualisiert werden. Die Tasse wird mit Hilfe teil-basierter-Deskriptoren 24 unter-Deskriptoren extrahiert. Die drei Histogramme aus den größten drei Teilen sind rechts dargestellt, sie sind mit

Nummer markiert und die größten drei Teile werden rechts dargestellt. In der Arbeit werden drei unterschiedliche Histogramm-basierte Deskriptoren als traditionelle Deskriptoren zur Verfügung gestellt, während drei unterschiedliche Deskriptoren aus Deep-Learning zum Vergleich benutzt werden. Sie werden in Kapitel 4 im Detail erläutert.

3.5 Hierarchische Struktur

Um die Errichtung des Deskriptor-Baumes zu verstehen, wird zunächst das Konzept, „hierarchische Struktur“, erläutert. Als Hierarchie bezeichnet man ein System von Elementen, die einander über- bzw. untergeordnet sind. Im Sinne der Monohierarchie ist dabei jedem Element höchstens ein anderes Element unmittelbar übergeordnet, während bei einer Polhierarchie auch mehrere über- und untergeordnete Elemente möglich sind.

Hierarchien werden häufig zum Sortieren von Objekten verwendet. Grafisch gesehen werden Ebenen oft mit Pyramiden oder Familienstammbäumen verglichen. Die Elemente können übersichtlich angeordnet werden und jedes Element (bis oben) ist nur mit einem (einer Ebene) oder mehreren (Polyhierarchie) -Elementen der nächsthöheren Ebene verbunden. Mathematisch betrachtet bedarf eine Hierarchie einer Ordnungsrelation, die einen Baum oder gerichteten azyklischen Graphen definiert. Das Komplement ist die Hierarchie. Die Klassifizierung von Objekten in eine Hierarchie impliziert häufig eine Wertigkeit, die bereits in der Rangordnung, nach der die Objekte geordnet werden, enthalten ist. Grundsätzlich sind sie allerdings einfacher als komplexe Netzwerkstrukturen zu erfassen

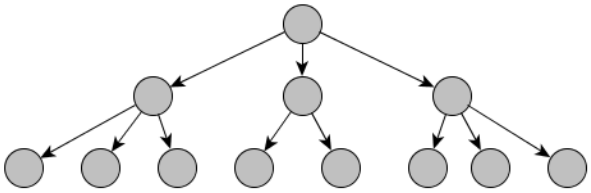


Abbildung 3.7: Hierarchische Struktur[30]

3.5.1 Oberflächennormalen Bestimmen

Um die geometrische Oberfläche zu beschreiben, werden die Oberflächen-Normalen benötigt. Zusätzlich ist der Vektor ein wichtiger Parameter für

Segmentierung. Da die Punktwolkendatensätze eine Menge von Punktproben auf der realen Oberfläche darstellen, gibt es zwei Möglichkeiten, um die Oberflächennormalen zu bestimmen:

1. die zugrunde liegende Oberfläche aus dem erfassten Punktwolken-Datensatz mithilfe von Oberflächenvernetzungstechniken zu ermitteln, und die Oberflächennormalen aus dem Netz zu berechnen.

2. Näherungen zu verwenden, um direkt aus dem Punktwolken-Datensatz auf die Oberflächennormalen zu schließen.

3.5.2 Segmentierung

Zum Erstellen der Hierarchischen Struktur ist Segmentierung notwendig. Der Begriff der Segmentierung befasst sich damit, inhaltlich zusammenhängende Regionen auf einem Bild zu erkennen. Dazu stehen unterschiedliche Verfahren zur Verfügung. Grundsätzlich werden sie oft in pixel-, kanten- und regionenorientierte Verfahren eingeteilt. Hierbei wird für jeden Pixel eines Bildes die Entscheidung getroffen, ob er zu einem Segment gehört oder nicht. Bei einem Grauwertbild wird für jedes Pixel geprüft, ob es über oder unter dem vorher festgelegten Schwellenwert liegt und je nach Fall wird der Pixel abgeändert. So können zum Beispiel ansonsten schwammig erkennbare Formen klar extrahiert werden. Die regionenorientierten Verfahren betrachten Punktmengen als Gesamtheit und versuchen dadurch zusammenhängende Objekte zu finden. Häufige Verwendung finden Verfahren wie Region Growing, Region-Splitting, Pyramid Linking und Split and Merge.

3.6 Deep Learning

Deep Learning ist eine spezialisierte Form von Machine Learning und auch ein neuer Trend im Machine Learning. Deep Learning hatte viele Erfolge in den Bereichen Computer-Vision, automatischer Spracherkennung und Verarbeitung natürlicher Sprache. Deep Learning bezieht sich auf eine Klasse von Methoden zur Optimierung künstlicher neuronaler Netze, die zahlreiche versteckte Schichten (engl. hidden layers) zwischen Eingabeschicht und Ausgabeschicht haben und daher eine komplexe in innere Struktur aufweisen. Das Deep-Learning-Modell wird daher häufig als tiefes neuronales Netzwerk bezeichnet.

Der Begriff „tief“ bezieht sich im Allgemeinen auf die Anzahl verborgener Schichten des neuronalen Netzes. Herkömmliche neuronale Netze enthalten

nur zwei bis drei verborgene Schichten, während tiefe Netze bis zu 150 Schichten enthalten. Eine der beliebtesten Arten tiefer neuronaler Netze wird als neuronales Faltungsnetzwerk (Convolutional Neural Networks, CNN oder ConvNet) bezeichnet. Im folgenden werden die zwei wichtigen Konzepte, künstlicher neuronaler Netz und Convolutional Neural Networks, beschrieben.

3.6.1 künstliche neuronale Netze

Künstliche neuronale Netze, kurz: KNN (engl. artificial neural network, ANN), die Grundrecheneinheit des Deep Learning, sind Netze aus künstlichen Neuronen, die von biologischen neuronalen Netzen inspiriert sind.

3.6.1.1 künstliches Neuron

Ein aus einem Zellkörper, Dendriten und einem Axon bestehendes biologisches Neuron ist die Grundrecheneinheit des menschlichen Gehirns.

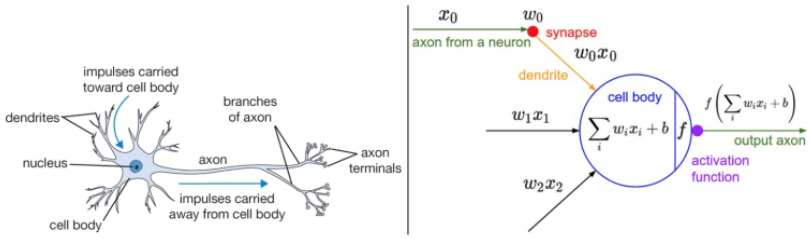


Abbildung 3.8: Links: Ein biologischenes Neuron und Rechts: die mathematischen Umsetzung eines künstlichen Neurons[20]

Ein grundlegendes Modell für die Funktionsweise der Neuronen lautet wie folgt: Jede Synapse hat eine Stärke(die Gewichtung w_i), die erlernbar ist und die Stärke des Einflusses eines Neurons auf ein anderes steuert. Die Dendriten tragen die Signale (die Eingabe x_i)zum Körper des Zielneurons, wo sie summiert werden. Wenn die letzte Summe $\sum_i w_i x_i + b$ über einer bestimmten Schwelle liegt, feuert das Neuron und sendet eine Spitze entlang seines Axons, im mathematischen wird der Sender mithilfe einer Aktivierungsfunktion modelliert.

Das oben erläuterte Modell ist in einer computergestützten Form zu formulieren. Im mathematischen Modell aus Abbildung 3.8 wird die Aktivierungsfunktion als f bezeichnet. Die gesamte Berechnung innerhalb eines Neurons

kann wie folgt dargetellt werden:

$$\phi(x) = f(\sum_i w_i x_i + b) \tag{3.1}$$

Ein künstliches Neuron hat eine endliche Anzahl von Eingängen, die der Parameter x repräsentiert. Der Parameter w bezeichnet die Gewichtungen und Parameter b ist ein Konstante, die auf das Ausgabeverhalten eines Neurones wirkt.

3.6.1.2 künstliche neuronale Netz

Künstliche Neuronen können auf vielfältige Weise zu einem künstlichen neuronalen Netz verbunden werden. Dabei werden Neuronen bei vielen Modellen in hintereinander liegenden Schichten (englisch layers) angeordnet; bei einem Netz mit nur einer trainierbaren Neuronenschicht spricht man von einem einschichtigen Netz.

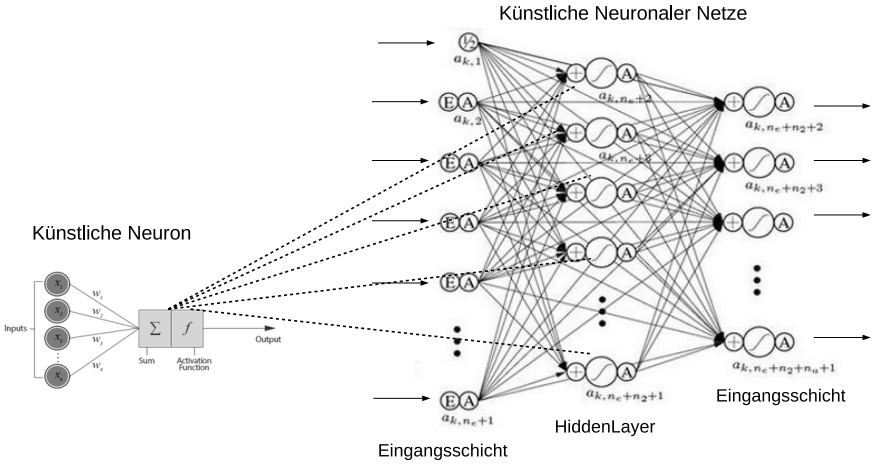


Abbildung 3.9: Neuronale Netze sind in Schichten aufgebaut, die aus einer Reihe von miteinander verbundenen Knotenpunkten bestehen. Netze können dutzende oder hunderte verborgener Schichten enthalten[28]

Künstliche Neuronen können auf vielfältige Weise zu einem künstlichen neuronalen Netz verbunden werden. Dabei werden Neuronen bei vielen Modellen in hintereinander liegenden Schichten (englisch layers) angeordnet; bei einem Netz mit nur einer trainierbaren Neuronenschicht spricht man von einem einschichtigen Netz.

Grundsätzlich können die Schichten zwischen Eingabeschicht, versteckter Schicht und Ausgabeschicht unterschieden werden. Der Eingabeschicht nimmt die Signalen von der Außenwelt, während sich die versteckte Schicht zwischen den Eingabe- und Ausgabe-Schicht die interne Informationen abbilden.Die Ausgabeschicht gibt die Informationen als Ergebnis weiter, wie in Abbildung 3.9 gezeigt. Die Anzahl der Schichten eines neuronalen Netze ist eine wichtige Information. Herkömmliche neuronale Netze enthalten nur zwei bis drei versteckten Schichten, aber theoretisch ist die Anzahl der Schichten unendlich .

3.6.2 Convolutional Neural Networks

Eine der beliebtesten Arten tiefer neuronaler Netze wird als neuronales Faltungsnetzwerk (Convolutional Neural Networks, CNN oder ConvNet) bezeichnet[33]. Zur Zeit hat sich CNN als einer der Forschungsscherpunkte in vielen wissenschaftlichen Bereichen entwickelt, insbesondere auf dem Gebiet der Objekt-Klassifizierung. Ein wichtiger Grund dafür ist, dass Convolutional Neural Networks die ursprünglichen Bilder direkt eingeben kann und die komplexe Vorverarbeitung des Bildes vermeiden.

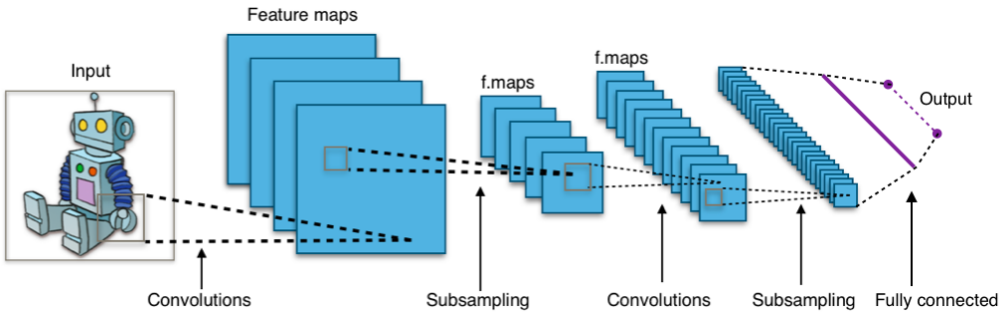


Abbildung 3.10: Struktur eines typischen CNNs. Subsampling entspricht Pooling. Dieses Netz besitzt pro Convolutional Layer mehrere Filterkernel, sodass Schichten an Feature Maps entstehen, die jeweils den gleichen Input bekommen, jedoch aufgrund unterschiedlicher Gewichtsmatrizen unterschiedliche Features extrahieren.[21]

Grundsätzlich besteht die Grundstruktur von CNN aus zwei Schichten: Merkmalsextraktionsschicht, und Feature-Mapping-Schicht. In der Merkmalsextraktonsschicht werden die Eingabe jedes Neurons mit der lokalen akzeptierten Domäne der vorherigen Schicht verbunden und werden die

lokalen Merkmale extrahiert. Nach dem Extrahieren wird die Positionsbeziehung zwischen der Features auch bestimmt. Die Gewichte aller Neuronen in der Ebene sind gleich. Die Merkmalabbildungsstruktur übernimmt die Sigmoid-Funktion, die den Kern der Funktion als Aktivierungsfunktion des Faltungsnetzwerks beeinflusst, so dass die Merkmalskarte eine Verschiebungsinvarianz aufweist. Da die Erkennungschicht von CNN durch Trainingsdaten lernt, vermeidet sie die Merknalsextraktoin anzuzeigen das CNN implizit aus den Trainingsdaten. Ein anderer Vorteil des Faltungntzwerkes im Vergleich zu anderen neuronalen Neztwerken ist, dass die Netze parallel lernen können, weil die Neuronen-Gewichte auf der gleichen Merkmals-Mapping-Schicht gleich sind.

Convolutional Neural Networks haben einzigartige Vorteile in Bezug auf Spracherkennung und Bildverarbeitung mit der speziellen Strukturen,bei der von lokale Gewichte geteilt werden. Das Layout ist ähnlicher zum tatsächlichen biologischen neuronalen Netzwerk. Die Eigenschaft der Gewichtsverteilung reduziert insbesondere die Komplexität des Netzwerks. Insbesondere kann das mehrdimensionale Eingabevektorbild direkt in das Netzwerk eingegeben werden, wodurch die Komplexität der Datenrekonstruktion während der Merkmalsextraktion und Klassifizierung vermieden wird.

Grundsätzlich bestehen Convolutional Neural Networks aus Filtern (Convolutional Layer) und Aggregations-Schichten (Pooling Layer), die sich abwechselnd wiederholen und am Ende aus einer oder mehreren Schichten von „normalen“ vollständig verbundenen Neuronen (Dense / Fully Connected Layer).

3.6.2.1 Convolutional Layer

In einem CNN Netzwerk besteht jede Faltungsschicht aus mehreren Faltungseinheiten. Intuitiv wird dabei schrittweise eine vergleichsweise kleine Faltungsmatrix über das Inputbild bewegt. Die Faltungsmatrix wird als Filterkernel bezeichnet, wie in Abbildung 3.11 gezeigt.

Der Begriff der Faltung befasst sich in der Funktionsanalyse mit einem mathematischen Operator, der eine dritte Funktion f*g für zwei Funktionen f und g bereitstellt. Die Faltung kann als ein Produkt von Funktionen verstanden werden. Beispielhaft ist $(f * g)(x)$ ein gewichteter Durchschnitt von f, wobei g das Gewicht ist. Ein einfaches Beispiel wird in Abbildung 3.11 angezeigt. Der 3×3-Filter wird verwendet, um das 5×5-Bild zum Falten zu verarbeiten, und schließlich wird das 3×3-Feature erhalten, wie es in der Abbildung gezeigt ist.

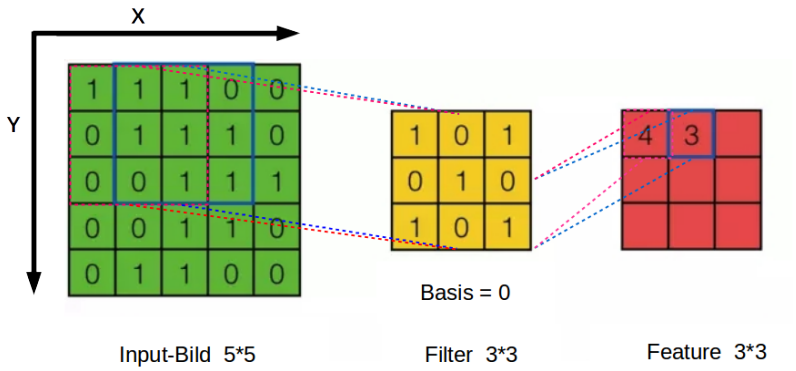


Abbildung 3.11: Falung mit einem 3×3-Filter und Schrittgröße = 1

Um den Faltungsberechnungsprozess zu beschreiben, wird zuerst jedes Pixel des Bildes nummeriert, dazu repräsentiert $X_{i,j}$ das Pixel aus der i-ten Zeilen- und j-ten Spaltenelemente des Bildes; Jede Anzahl von Filtern ist mit $W_{m,n}$ nummeriert, um das Gewicht der m-ten Zeile und der n-ten Spalte anzugeben. W_b repräsentiert den Offset des Filters. Jedes Element der Feature Map ist nummeriert, wobei $a_{i,j}$ die i-ten Zeilen- und j-ten Spaltenelemente der Feature Map darstellen. F ist die Größe des Filters, und die Aktivierungsfunktion wird mit f bezeichnet. Um die Faltung zu berechnen, wird die folgende Formel verwendet:

$$a_{i,j} = f(\sum_{m=0}^{F-1}\sum_{n=0}^{F-1} w_{n,m}x_{i+m,j+n} + w_b) \tag{3.2}$$

Die vorhergehende Formel ist ein Berechnungsverfahren für eine Faltungsschicht mit einer Tiefe 1. Normalerweise ist die Tiefe der Bilder 3, die R, G, B repräsentieren. Wenn die Bildtiefe vor der Faltung D ist, muss die Tiefe des entsprechenden Filters ebenfalls D sein. Um die Faltungsberechnungsformel mit einer Tiefe größer als 1 zu erhalten, wird die Formel erweitert:

$$a_{i,j} = f(\sum_{d=}^{D-1}\sum_{m=0}^{F-1}\sum_{n=0}^{F-1} w_{d,n,m}x_{d,i+m,j+n} + w_b) \tag{3.3}$$

Der Zweck der Faltungsoperation besteht darin, die verschiedenen Merkmale der Eingabe zu extrahieren. Die erste Faltungsschicht kann möglicherweise nur einige Merkmale auf niedriger Ebene wie Kanten, Linien und Winkel

extrahieren. Mithilfe mehreren Schichten des Netzwerks können komplexere Features aus Merkmalen auf niedriger Ebene iterativ extrahiert werden. Zusätzlich kann eine Mehrkern-Faltung eingeführt werden. Jeder Faltungskern wird verwendet, um verschiedene Merkmale zu lernen. Jeder Faltungskern lernt unterschiedliche Gewichte, damit mehr unterschiedliche Originalbildmerkmale extrahiert werden können.

3.6.2.2 Pooling Layer

Bei Pooling Layers werden die mit einer großen Dimension durch der Faltungsschicht bekommenden Features in mehreren Regionen ausgeschnitten, damit können neue Features mit kleineren Dimensionen bekommen. Der Zweck des Pooling Layer ist, überflüssige Informationen zu verwerfen, z.b die exakte Position einer Kante im Bild zu vernachlässigen, da die ungefähre Lokalisierung eines Features hinreichend ist.

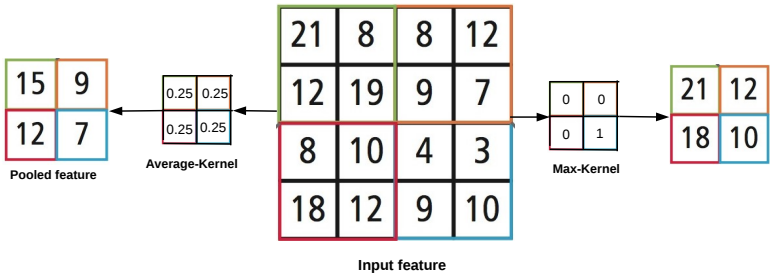


Abbildung 3.12: Pooling: Rechts: Max-pooling mit einem 2×2-Filter und Schrittgröße = 2; Links:Average-pooling mit einem 2×2-Filter und Schrittgröße = 2, [22]

Es gibt unterschiedliche Arten des Poolinges. Wie in Abbildung 3.12 gezeigt, sind zwei Methoden, max-pooling und mean-pooling:

1. Jede Gewichtung in dem Faltungskern von mean-pooling beträgt 0.25 und der Faltungskern bewegt sich über das Inputbild mit einem Gleitschritt von 2. Der Effekt der zweiten mittleren Unterabtastung entspricht

der Reduzierung des ursprünglichen Bildes auf die ursprünglichen Unschärfe.

2. In der max-pooling gibt es nur ein Faltungskernel mit Gewichtungswerte 1, während die anderen alle 0 sind. Die Position von 1 im Faltungskern entspricht der Position, an der der Eingabe-Vektor mit dem größten Teil vom Faltungskern abgedeckten Wertes gefaltet ist. Der Faltungskern hat einen Gleitschritt von 2 auf dem Eingabebild. Der Effekt der maximalen Unterabtastung besteht darin, das ursprüngliche Bild auf 1/4 zu reduzieren und die stärkste Eingabe für jede 2 × 2-Region beizubehalten.

3.6.2.3 Fully-Connected Layer

Nach einigen sich wiederholenden aus Faltung- und Pooling-Layer bestehenden Einheiten werden danach alle lokalen Features in einem fully-connected-Layer kombiniert, um das Endergebnis jeder Klasse zu berechnen. Zur Klassifizierung korrespondiert das Anzahl der Neurons in der letzten Schicht üblicherweise zu der Anzahl an Klassen.

4 Entwurf

Der Zweck dieser Arbeit besteht darin, zunächst traditionelle Deskriptoren und Deep Learning Deskriptoren durch Experimente zu bewerten. Auf der Grundlage dieser Daten wird der Begriff Hierarchie eingeführt, Deskriptor Tree zu ersetzen, damit die geometrische Beziehung der Deskriptoren strukturiert abgelegt wird.

In diesem Kapitel wird die Architektur der 3 unterschiedlichen Systeme, des traditionelle System, Deep-Learning System und des Deskriptor-Baum System, beschrieben. Im Rahmen dieser Beschreibung wird detailliert auf den Aufbau der Architektur sowie das Verfahren zur Ermittlung der benötigten Merkmalextraktion und das Training eingegangen. Abschließend werden die verwendeten Softwarekomponenten zur Klassifzierung, und Nächste-Nachbarn, Optimal-Matching erläutert.

4.1 Test Daten

In der Arbeit wird eine Untermenge der SHREC 3D-Objektdatenbank zur Auswertung verwendet. Die SHREC 2010 Datenbank, the Shape Retrieval Contest of Range Scans [12], ist schon in Kapitel 3.2 vorgestellt. Die Untermenge besteht aus 10 Objekten aus jeder von 11 Objektklassen, Vogel-, Fisch-, nicht-fliegendes-Insekt-, fliegendes-Insekt-, Zweifler-, Vierfüßler-, Apartment-, Hochhaus-, Einzel-Haus-, Flasche- und Tasse-Klasse. Für jedes Objekt werden vier Ansichten ausgewählt. Wie in Abbildung 3.1 dargestellt, dass die Vier Ansichten vorne Ansicht, oben Ansicht, Ansicht von oben nach hinten und vom unten nach oben zeigt. Insgesamt gibt es 440 gelabelte Tiefenbilder in Form einer Punktwolke als Test-Daten in der Arbeit.

Die Tiefeinformationen jedes Objektes werden als PCD-Dateien bereitgestellt. Die dazu gehörigen Labels werden in einer Text-Datei gespeichert. Beim Test werden die PCD-Dateien und die Label-Text-Datei im PCL-System eingegeben. Der obere Teil in Abbildung 4.1 sind die Labels der Objekte und der untere Teil sind Punktwolken jeder Klasse .

In der Arbeit werden die 440 Daten in zwei Untergruppen unterteilt. : distinct-Objekts: Biped, Bird, Quadruped, Fish, Mug, wobei Objekte durch wenige Details unterscheidbar sind, und similar-Objekts: Apartment House,

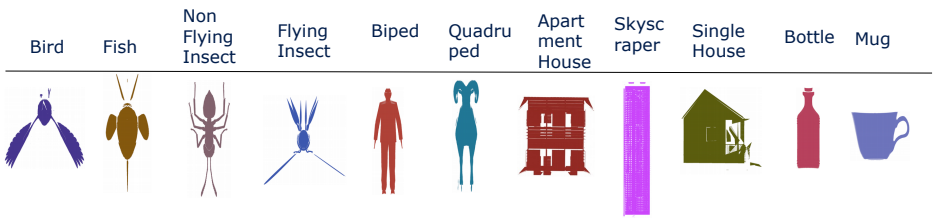


Abbildung 4.1: Test Daten, die oberen sind Labels jeder Klasse und die unteren sind die Punktwolke jeden Klasse

Flying, Insect, Non Flying Insect, Single House, Skyscraper , wobei Objekte mit einer größeren Anzahl von Details unterschieden werden können. Das Ziel besteht darin, die Eigenschaften der Deskriptoren in den zwei Situationen hervorzuheben.

4.2 Deep-Learning Deskriptoren

In diesem Teil werden 3 unterschiedliche Deskriptoren aus drei unterschiedlichen Methoden, CaffeNet, VAE und DLR-VAE getestet. Dabei werden die Deep-Learning Frameworks „ Caffe“ und „Tensorflow“ für den Trainingsprozess als auch für die Deskritptoren Extraktion, und PCL für den Klassifizierungs-Teil verwendet.

Die Architektur des Trainings und Klassifizierungs-Prozesses wird in Abbildung 4.2 visualisiert. Wie in der Abbildung dargestellt, wird die gesamte

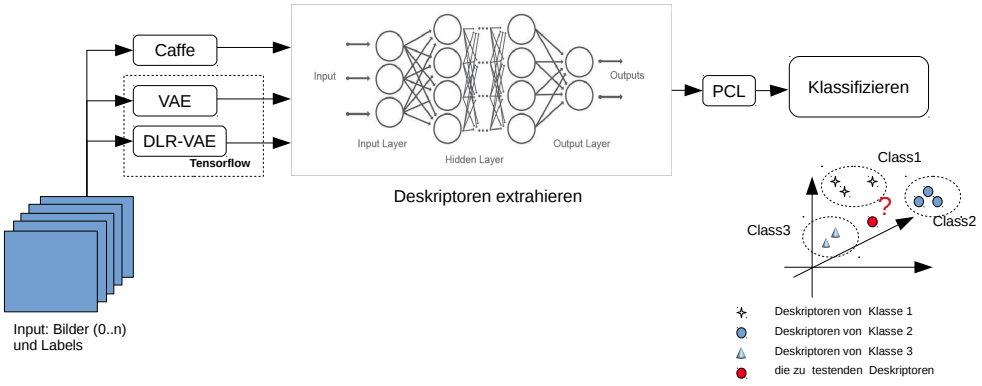


Abbildung 4.2: Architektur des Deep-Learning Teiles zur 3D-Objekterkennung

Architektur in mehreren Abschnitte unterteilt. Der ersten Abschnitt ist die Eingabe von Trainingsdaten. Dabei handelt es sich um RGB-Bilder und die zugehörigen Labels, die als Trainingsdaten auf drei unterschiedliche Pfade eingegeben werden können.

Danach kommt der wichtige Abschnitt, Merkmale zu extrahieren. Dabei wird der erste Teil mit AlexNet-Modell, das aus einem großen Deep Convolutional Neuronalen Netz besteht, trainiert. Abschnitt 4.2.1 beschreibt zunächst den Prozess . Die zweite und dritte Methode wird mit Hilfe des VAE-Modells, das aus Encoder und Decoder besteht, realisiert. Und die zugehörigen Prozesse werden von Abschnitt 4.2.2 und 4.2.3 erläutert.

4.2.1 CaffeNet

Caffe[10] ist eine Programmbibliothek für Deep Learning, die von Yangqing Jia entwickelt wurd. Zurzeit ist Caffe sehr populär und wird weit verbreitet zum Deep Learning verwendet, da sie zahlreiche Algorithmen und Deep-

Learning-Architekturen für die Klassifikation und Clusteranalyse von Bilddaten enthalten sind, und CNN, Rekrurentes neuronales Netz (kurz. R-CNN), Long short-term memoriy (kurz. LSTM) und vollständig verbundene neuronale Netze unterstützt. Außerdem ist Caffe mit Millionen von RGB-Bildern vor-trainiert.

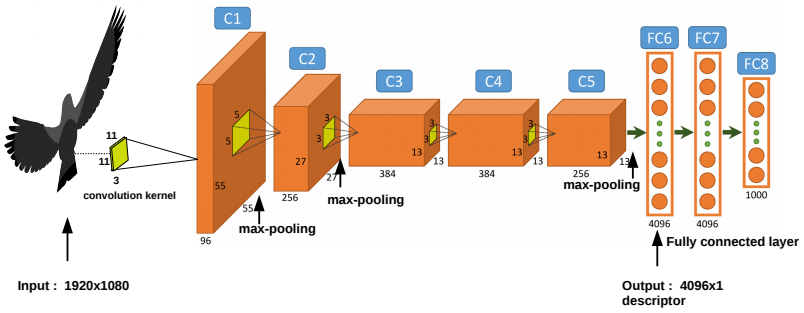


Abbildung 4.3: AlexNet Modell[23]

Der Betrieb von Caffe basiert auf einem Modell. Die am häufigsten verwendeten Modelle sind Lenet, AlexNet, VGG usw. Ein Modell besteht aus mehreren Layern. Es gibt viele Arten von Layern, z.b Datenschichten, Vision-Schichten und Aktivierungsschichten. Jede Schicht entspricht vielen Parametern, die in einer Konfigurationsdatei definiert sind. Dazu spielen Vision-Schichten eine wichtige Rolle. Visuelle Schichten enthalten Convolutions, Pooling, Local Respons Normalization (kurz. LRN) und so weiter, die bereits in Kapitel 3 vorgestellt werden.

Abbildung 4.3 zeigt das Modell von AlexNet, das in der Arbeit verwendet wird. AlexNet ist ein großes Deep Convolutional Neuronales Netz. Der Vorteil von AlexNet besteht darin, dass das Netz mit füf Faltungeschichten, drei full-connection-Layer und ein Softmax-Layer erweitert wird.

Wie in Abbildung 4.3 gezeigt enthält AlexNetz acht Schichten, die ersten fünf Schichten c1, c2, c3, c5 und c6 sind Faltungsschichten, während fc6, fc7 und fc8 Full-connected-Layers sind.

Die Faltungskerne auf der zweiten, vieten und fünften Faltungsschicht sind mit der vorherigen Faltungsschicht verbunden. Die normalisierte Response-Schicht folgt der ersten und der zweiten Faltungsschicht c1, c2, danach folgen die Max-pooling Schicht die normalisierten Response-Schichten und der fünften Faltungsschicht c5. Der Ausgang des letzten full-connection-Layeres wird zu einer SoftMax-Schicht mit 1000 Wegen gesendet, damit eine Verteilung erzeugt wird, die 1000 Etiketten abdeckt.

In der ersten Faltungsschicht werden 96 Kerne mit Größe 11×11×3 verwendet, um die Eingabebild 1920×1080×3 zu filter, und die Schrittgröße ist 4 Pixeln. Die zweite Faltungsschicht nimmt die Ausgabe der ersten Faltungsschicht als ihre eigene Eingabe auf, die schon nach der max-pooling-Layer normalisiert und gepoolt werden, und danach verwendet 256 Kerne mit Größe 5×5×48, die Daten weiter zu filtern. Die dritte, vierte und fünfte Faltungsschicht sind direkt miteinander verbunden. Dazwischen gibt es keine Pooling-und Normalisierungsschichten. Die dritte Falungsschicht hat 384 Faltungskerne mit Größe 3x3x256, und die vierte Schicht mit 384, 3x3x192 großen Kernen und fünfte Schicht mit 265 Kerne mit Größe 3x3x192. Die voll-connection-Schicht hat jeweils 4096 Neuronen. In der Arbeit sind die Ausgabe aus fc6 als Deskriptor zu testen.

4.2.2 Variational Autoencoder

Variational Autoencoder[9] ist ein künstliches neuronales Netz, das auf Autoencoder basiert. Es wird dazu genutzt, effiziente Codierungen zu lernen. Im Folgenden werden zuerst Autoencoder erläutert, danach Variational Autoencoder.

4.2.2.1 Autoencoder

Autoencoder befasst sich damit, deine komprimierte Repräsentation (Encoding) für einen Satz Daten zu lernen und somit auch wesentliche Merkmale zu extrahieren. Dadurch kann er zur Dimensionsreduktion genutzt werden. Autoencoder bestehen aus zwei Teilen: Codierer und Dekodierer, wie in Abbildung 4.4 gezeigt.

Der Codierer codiert die Eingabedaten(x), um den Ausdruck der versteckten Schicht zu erhalten. Der Decodierer decodiert den Ausdruck der verborgenen Schicht, um die Ausgabe($\tilde{x}$) mit gleichen Anzahl von Eingabe Knoten zu rekonstruieren, wie in Abbildung 4.4 gezeigt. Die Zielfunktion für die Modelloptimierung ist der Rekonstruktionsfehler zwischen x und $\tilde{x}$.

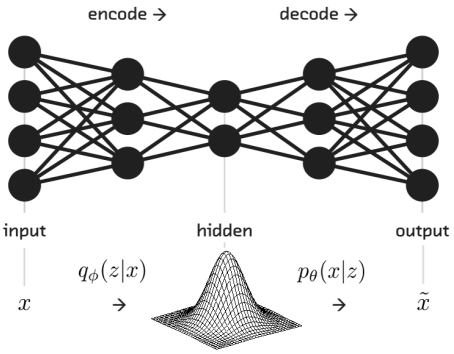


Abbildung 4.4: Autoencoder[24]

Der Codierer und der Decodierer können eine Struktur aus MLP, CNN und RNN sein. Beim Autoencodoer kann ein Funktionspaar (f,g) über einem d-dimensionalen Eingaberaum X definiert werden. Die Elemente des Eingaberaum sind Vektoren, deren Elemente Features entsprechen. Bei Anwendung der Funktion g auf das Ergebnis der Anwendung von f auf ein beliebiges Element x des Eingaberaums, optimalerweise genau des ursprüngliche Element $x = \tilde{x}$ enthält.

Die zwei Prozesse des Codierens und Decodierens können als zwei gegenseitige umgekehrte Funktionen betrachtet werden. Während des Kodierungsprozesse wird die Dimension reduziert, im Decodierungsprozess erhöht sich die Dimension. Beim Autoencodoer kann Faltungsnetzwerk verwendet werden, um die Features zu extrahieren. Man kann es so verstehen, dass in Kodierungsprozess ein aus mehreren Faltungs- und Pooling-Schichten bestehendes Deep-Convolutional-Neural-Netzwork zu Verfügung steht. In Decodierungsprozess werden mehrere Schichten aus Dekonvolution und Unpooling verwendet.

Unpooling

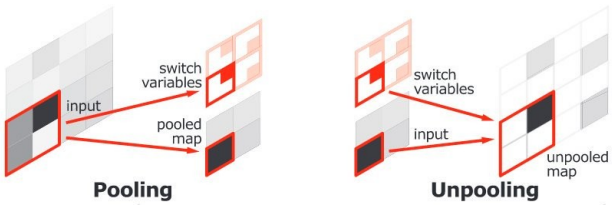


Abbildung 4.5: Unpooling[25]

Wie das in Abschnitt 3.4.2.2 erläuterte Pooling verwendet Unpooling auch ein Faltungskernel, um die Dimension zu erhöhen. Ein Bild enthält 32x32 Pixel und der Pool-Kernel enthält 2x2 Pixel, die Größe des Bildes ist nach dem Pooling-Prozess 16x16, dadurch, dass benachbarte 2x2 Pixel im Bild durch den Maximalwert in vier Punkten ersetzt werden. Der Unpooling-Prozess ändert das 16x16-Bild wieder in 32x32, damit dass, 2x2 Pixel mit den Maximalwert wieder ersetzt werden, wie in Abbildung 4.5 gezeigt.

Deconvolution

Mit Deconvolution bezeichnet man die Umkehrung der Faltungsoperation.

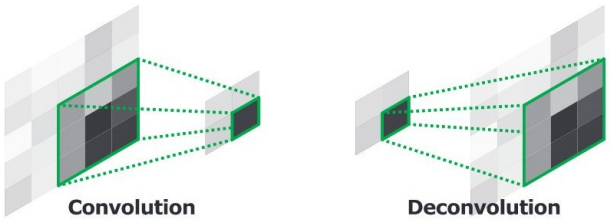


Abbildung 4.6: Deconvolution[25]

Allgemein können Autoencoder verwendet werden, um Daten zu reduzieren, Daten zu komprimieren, Daten zu entschärfen usw. Herkömmliche Deep-Learning Methoden verwenden auch Autoencoder, um Layer-by-Layer-Pre-Training von CNN-Netzwerken durchzuführen. Autoencoder lassen sich auch zum unüberwachtes Lernen verwenden, um die Eigenschaften der Datenexpression zu lernen. Autoencoder sind wie ein selbst-überwachtes Lernen.

4.2.2.2 Variational Autoencoder

Die Unterschied zwischen VAE und autoencoder ist, dass VAE nicht wie Autoencoder direkt die verdeckte Layer-Darstellung von Eingabedaten lernt, sondern das verborgene Variablenmodell der Eingabedaten. VAE enthalen hauptsächlich drei Module: Encoder, Sample und Decoder. Die Struktur von VAE wird in Abbildung 4.7 visualisiert.

Wie im letzten Abschnitt erklärt, wird das eingegebene Bild wird zuerst im Codierer in einen verborgenen Vektor kodiert, danach wird der Vektor bei Decodierer wieder zu einen Bild, das dem Originalbild entspricht, decodiert. Im Gegensatz zum autoencoder erzeugen VAE zuersten zwei Vektoren, Mittelwert-Vector und Standardabweichung-Vector und danach werden diese zwei Statistik Daten verwendet, um den verborgenen Vektor zu synthetisieren.

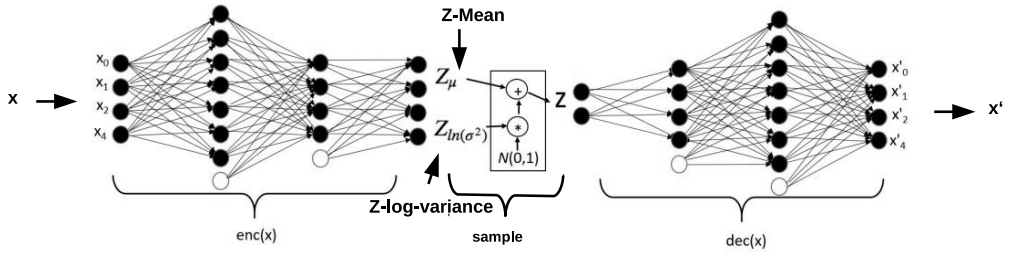


Abbildung 4.7: Variant Autoencoder[26]

Es wird angenommen, dass der Datensatz D der Eingabedaten durch einen Satz von versteckten Variablen z gesteuert wird. Die Verteilung des Datensatzes wird vollständig durch diesen Satz versteckter Variablen gesteuert und dieser Satz verborgener Variablen ist unabhängig und ist normal verteilt. Zum Training des versteckten-Variable-Modells von Lerneingabedaten bedeutet es, die Parameter der Gaußschen Wahrscheinlichkeitsverteilung dieser Menge versteckter Variablen zu lernen: Mittelwert und der Logarithmus der Varianz der Gaußverteilung der versteckten Variablen. Wie in Abbildung 4.7 gezeigt, repräsentiert *z-Mean* (Z_μ) den Mittelwert und *z-log-varianz* ($Z_{ln(\sigma^2)}$) den Logarithmus der Varianz der Gaußverteilung der versteckten Variablen. In der Arbeit werden z-Variablen, der vom z-Mean und z-log-varianz kombiniert werden, als Deskriptoren weiter zur Klassifizierung verwendet.

4.2.3 DLR-VAE

Um den Effekt von Trainingsklassifizieren oder Regressorren auf die Einbettung von VAE zu evaluieren, wurden DLR-VAE von Ingo Kossyk und Zoltan-Csaba Marton entwickelt. DLR-VAE ist ein Variation-Auto-Encoder, dem eine zusätzliche Klassifikation hinzugefügt wird. Dabei sind Encoder-, Decoder- und Klassifizierung-module mit zwei vollständig verbundenen Neuronalen Netzwerken realisiert.

4.2.3.1 Encoder und Decoder

Wie in Abbildung 4.8 dargestellt, besteht der Encoder aus drei Faltungsschichten und Decoder aus vier Faltungsschichten. Im Encoder folgen der Faltung zwei Full-Connected-Layer, um die Parameter eines Gaußschen Prior für die latente Mannigfaltigkeit z zu schätzen. Der Decoder codiert zuerst die latente Repräsentation in eine geeignete Dimensionalität mithilfe von einer

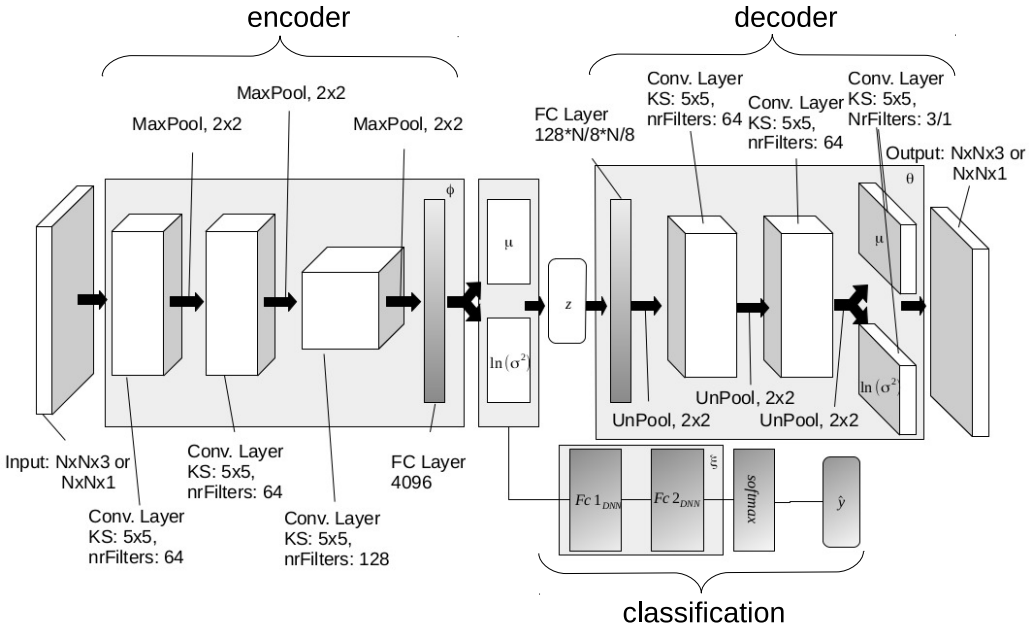


Abbildung 4.8: DLR-VAE Modell[3]

vollständig verbundenen Schicht, danach werden die Faltungsschichten mit 2 × 2 Unpooling-Layern kombiniert. Um die Parameter der Rekonstruktionswahrscheinlichkeitsverteilung am oberen Ende des Variational-Auto-Encoders zu schätzen, werden zwei Faltungsschichten verwendet.

4.2.3.2 Klassifikation

Der Encoder ist ein tiefes neuronales Netzwerk mit zwei vollständig verbundenen Schichten mit exponentiellen Linearen Einheiten(ELU) als Aktivierungsfunktionen in dem gesamten Netzwerk. Die letzten zwei Schichten des Encoders werden parallel mit dem Ausgang der vorhergehenden Schicht eingegeben. Zusätzlich zur Verwendung des geschätzten Mittelwerts der logarithmischen Varianz für die Reparametrisierung, um die stochastischen Repräsentation z zu bilden, werden sie auch verwendet, um die Klassifizierung zu trainieren. Der Decoder besteht aus zwei vollständig verbundenen Schichten mit der gleichen Größe wie die Encoder-Schichten. Die Aktivierungsfunktion der letzten Schicht besteht aus der softmax-Funktion, deren Ausgabe zur Berechnung der logarithmischen softmax-binären Crossentropie

als Loss von Klassifikation verwendet wird.

4.3 Traditionelle Deskriptoren

In diesem Abschnitt werden drei Methoden vorgestellt, die alle mit Hilfe der Point Cloud Library realisiert werden. Dabei sind zwei traditionelle Deskriptoren teil-basiert, Clustered Viewpoint Feature Histogram (CVHF)[7], Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram (OUR-CVFH)[8], die mithilfe von einem region-growing-Algorithmus in kleine Teile unterteilt werden, während der andere, Viewpoint Feature Histogram (VFH)[5], das ganze Objekt repräsentieren kann.

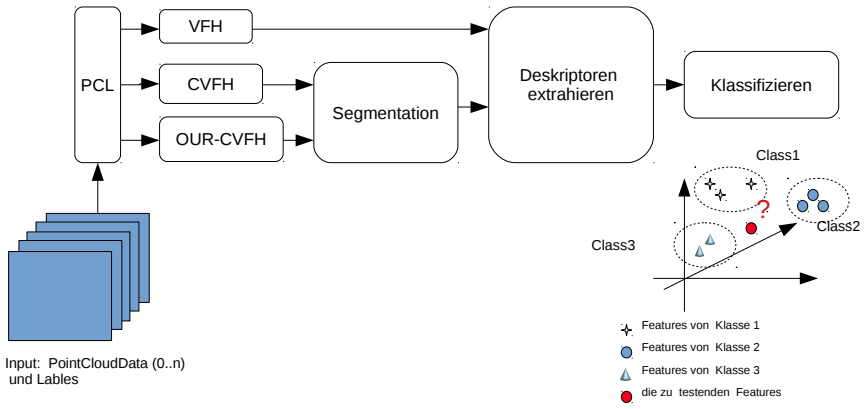


Abbildung 4.9: Architektur des traditionelle Teiles zur 3D-Objekterkennung

Die Architektur des Systems ist in Abbildung 4.9 gezeigt. Wie in Abbildung 4.9 dargestellt, besteht die Architektur aus mehreren Schichten. Bei dem ersten Abschnitt handelt es sich um das Empfangen der Eingebe. Die eingegebenen Tiefenbilder sind in Form einer Punktwolke, PCD-Dateien,

gespeichert und die zugehörigen Labels werden in einer Text-Datei gespeichert und die beiden können gleichzeitig in PCL heruntergeladen werden. Danach können die Tiefeninformationen bei drei Methoden behandelt werden, damit VFH-, CVFH- und OUR-CVFH-deskritporen extrahiert werden. Dabei werden die Tiefeninformationen bei CVFH und OUR-CVFH zuerst in kleine inhaltlich zusammenhängenden Regionen, dadurch benachbarte Pixel entsprechend Region-Growing-Algorithmus zusammengefasst werden, erzeugt, indem wird ein Deskriptor für jede Region extrahiert, damit entspricht ein Trainingsdatum mehrerer Deskriptoren, die in einer Liste gespeichert werden. Schließlich folgt die Klassifikations-Komponente, die in Abschnitt 4.5 erläutert wird. Im Folgenden werden die Prozess der Deskriptor-Extraktion detailliert beschrieben.

4.3.1 Viewpoint Feature Histogram

Viewpoint Feature Histograms (VFH)[6] ist ein globaler 3D-Deskriptor aus Tiefeninformationen, die in Form einer Punktwolke vorliegen und zum Zweck der Erkennung und Posenabschätzung für Punktecluster erstellt werden.

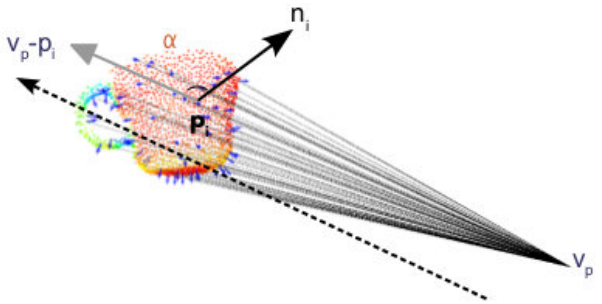


Abbildung 4.10: Punktwolke (schwarz) einer Tasse mit Blickrichtung v_p [6]

VFH besteht aus zwei Teilen: eine Oberflächenform-Komponente und eine Blickrichtungs-Komponente. Dabei ist der Zweck der Posenabschätzung von der zweiten Komponente, das Blickrichtungs-Komponente, abhängig. Die verschiedenen Posen von Objekten können unterschieden werden, indem die Variablen der Blickrichtungs-Komponente zusammengefügt sind. Im Teil der Oberflächenform wird das gesamte Punktwolken Objekt zum Berechnen und Schätzen verwendet. Er generalisiert die mittlere Krümmung um den Punkt unter Verwendung eines multidimensionalen Histogramms, und dadurch die k-Nachbarschafts-geometrischen Eigenschaften eines Punkts zu kodieren.

Die mathematische Erklärung ist wie folgt. Zuerst ist der Oberflächen-Teil, der aus innerhalb einer Referenzsystems in Bezug auf den Schwerpunkt berechneten drei Winkel (α, θ, ϕ) generiert. Um die relative Abweichung zwischen zwei Punkten Pi und Pj und ihren entsprechenden Normalen Ni und Nj zu berechnen, wird wie in Abbildung 4.11 ein festes lokales Koordinatensystem an einem der Punkte definiert. Mithilfe des u,v,w-Koordinatensystems kann die Abweichung zwischen den Normalen durch einen Winkel dargestellt werden:

$$\alpha = v \cdot n_t \tag{4.1}$$

$$\phi = u \times \frac{P_t - P_s}{d} \tag{4.2}$$

$$\theta = arctan(w \cdot n_t, u \cdot n_t) \tag{4.3}$$

Dabei ist d die Euklidische Entfernung zwischen zwei Punkten. Der linke Teil von Abbildung 4.11 zeigt beispielhaft die Berechnung dieser Winkel zwischen dem Punktp_s und p_t.

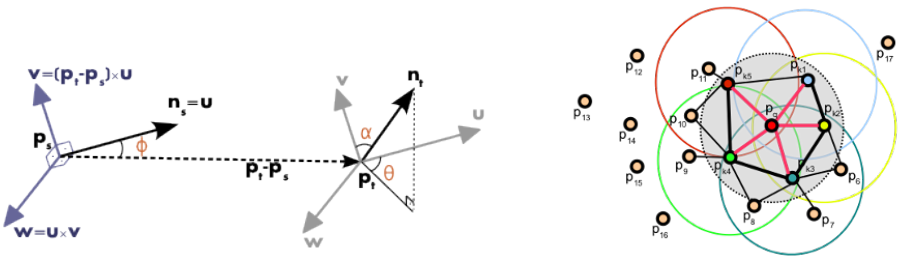


Abbildung 4.11: Links: Winkelberechnung zur Beschreibung der Objektgeometrie Rechts: Vereinfachen des Algorithmuses[27]

Theoretisch ist die Berechnungskomplexität für eine gegebene Punktwolke P mit n Punkten $O(nk^2)$, k ist die Anzahl der Nachbarn für jeden Punkt p in Punktwolke P. Deshalb ist es nötig, den Algothimus zu vereinfachen. Der Einfachheit wird zuerst eine Menge von (α, ϕ, θ) -Werten für jeden Abfragepunkt Pq berechnet. Diese Werte werden als Simplified-Point-Feature-Histogram (SPFH) bezeichnet. Danach werden die k-Nachbarn für jeden Punkt neu bestimmt, der SPFH-Wert wird dabei verwendet.

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{w_k} \cdot SPFH(p_k) \tag{4.4}$$

w_k repräsentiert die Entfernung zwischen zwei Punkten. Weight ist der Abstand zwischen dem Abfragepunkt und seinem benachbarten Punkt in einem bestimmten geometrischen Raum. Rechts in Abbildung 4.11 zeigt das Einflussbereichsdiagramm für einen k-Nachbarschaftssatz, der bei p_q zentriert ist.

Die Blickwinkel-Komponente wird durch eine Winkeldifferenz beschrieben. Indem das Histogramm des Winkels zwischen der Ansichtsrichtung und jeder Normalen gerechnet wird. Die Blickrichtung ist aus dem Standpunkt der Kamera und der Ausgangspunkt ist auch der Standpunkt der Kamera. Schließlich werden alle ermittelten Eigenschaften als Ergebnis in einem aus 303 Vektoren bestehenden Histogramm codiert.

4.3.2 Clustered Viewpoint Feature Histogram

Clustered Viewpoint Feature Histogram (CVFH)[7] ist eine Erweiterung von VFH und ist teil-basiert.

Bei CVFH und folgenden OUR-CVFH[8] werden die eingegebenen Tiefeninformationen, die PointCloud, in kleine inhaltlich zusammenhängenden Regionen unterteilt. Die Regionen werden durch die Zusammenfassung der benachbarten Pixel mithilfe eines Region-Growing-Algorithmus erzeugt. Danach wird ein VFH-Deskriptor für jede Region extrahiert, damit die Trainingsdaten mehrerer Deskriptoren in einer Liste gespeichert werden. Der Region-growing-Algorithmus wird in folgendem Abschnitt 4.4.1 erläutert. Abbildung 4.12 visualisiert die PointCloud für einen Vogel, der bei CVFH unterteilt wird.

4.3.3 Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram

Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram (kurz. OUR-CVFH)[8] basieren auf CVFH, ist auch ein teil-basierter globaler 3D-Deskriptor aus Tiefeninformationen. 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.

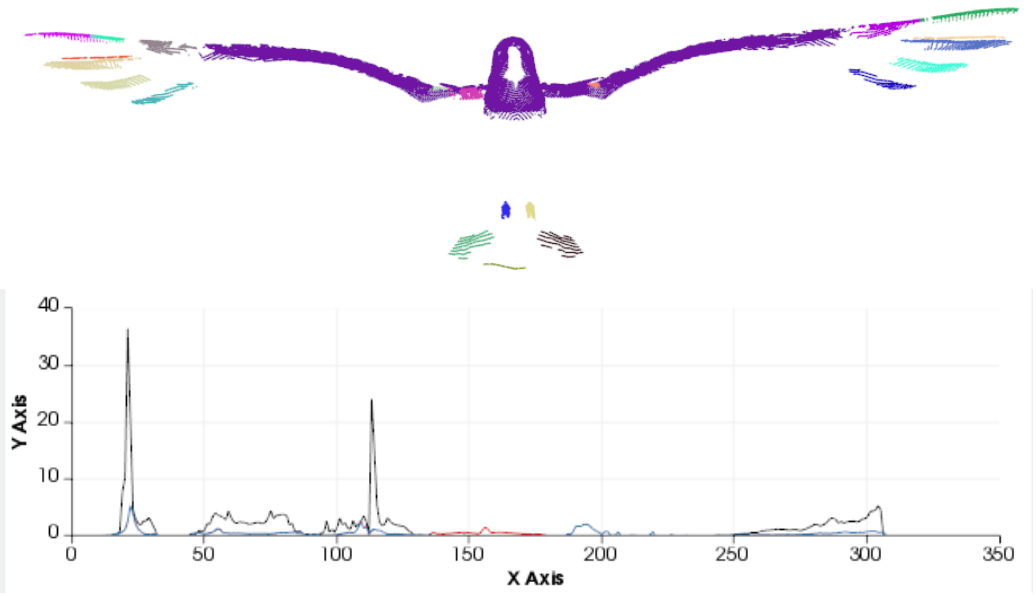


Abbildung 4.12: Oben: die durch CVFH segmentierter Punktwolke für einen Vogel; unter: die extrahierte Deskriptoren jeden aus VFH, CVFH und OUR-CVFH. Die schwarze Linie repräsentiert VFH-Deskrptor, blute Linie bezeichnet ein CVFH-Deskriptor, und rote Linie repräsentiert ein OUR-CVFH-Deskriptor

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 4.13 zeigt die Punktewolke eines Weinglases, in dem ein solches Referenzsystem beispielhaft in Grün eingezeichnet wurde. Nach der Bestimmung des Referenzsystems wird im nächsten Schritt der Schwerpunkt von diesem ermittelt. Anschließend wird ein Deskriptor mit Algorithmus VFH für jede Region extrahiert und alle werden nacheinander in einer Liste gespeichert, womit die OUR-CVFH Deskriptor Liste erzeugt wird.

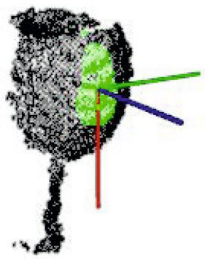


Abbildung 4.13: Punktwolke (schwarz) eines Weinglases mit dazugehörigem Ci (grün) und der SGURF Referenzrahmen[8]

4.4 Deskriptor-Baum

In diesem Abschnitt wird der wichtigste Teil dieses Themas erläutert, den Deskriptor-Baum zu erstellen. Dieser Teil basiert auf all den theoretischen Grundlagen, die im vorherigen Abschnitt vorgestellt wurde. Durch die Analyse der Vor- und Nachteile der vorher genannten Algorithmen wird eine neue Idee vorgeschlagen. Es befasst sich hauptsächlich damit, eine Optimal-Lösung zu finden, um das Problem der ungeordneten Anordnung von Schneidteilen in teil-basierten Algorithmen zu lösen. Es besteht die Hoffnung, dass mithilfe des in Kapitel 3 vorgestellten Begriffes, Hierarchische Struktur, wo die Ordnungsbeziehungen eines Baumes definiert werden, damit die Logik- und Geometrie-Beziehung der Teile herauszufinden.

Um diese Idee zu realisieren, wird eine Architektur entwickelt, die in Abbildung 4.14 gezeigt wird. Diese erfolgt in Point Cloud Liabrary und wird insgesamt in einige Abschnitte unterteilt. Bei dem ersten Abschnitt handelt es sich um das Empfangen der Eingabe bei PCL. Dazu enthält die Eingabe zwei Teile: Tiefinformationen, die in PCD-Datein gespeichert sind, und die entsprechenden Labels.

Bei dem ersten Abschnitt handelt es sich um das Empfangen der Eingabe. Dazu enthält die Eingabe zwei Teile: Tiefinformationen, die in PCD-Datein gespeichert sind, und die entsprechenden Labels. Danach der Segmentierung-Abschnitt, der mit mehreren Malen Segmentierung erreicht wird.

Ähnliche Pixelindizes werden in einem Cluster gespeichert und die Clusters werden über Segmentierung-Teil ausgegeben. Danach kann man sie in zwei Pfade aufteilen. Die unterteilten Cluster können in der PCDs zur Visualisierung gespeichert werden, d.h. in Segmentbaum-Pfade. Die anderen Pfade werden der Extraktion von Deskriptoren aus der Clusters verwendet und generiert werden, bei der es sich um den Deskriptoren-extrahieren-Abschnitt handelt.

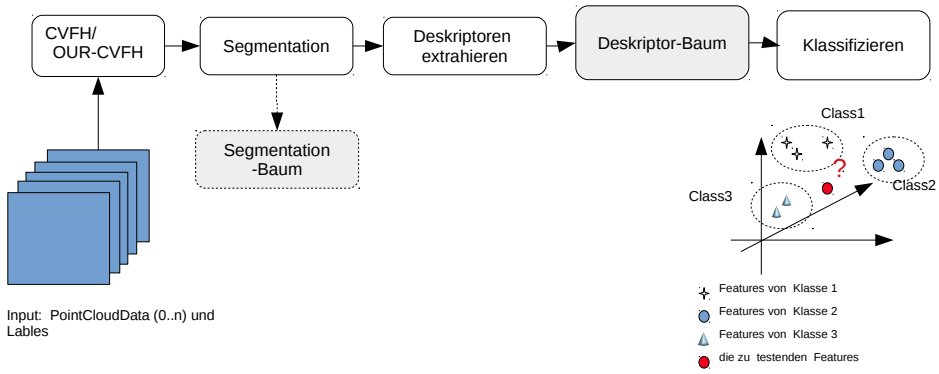


Abbildung 4.14: Architektur des Deskriptor-Baum Teiles zur 3D-Objekterkennung

Anschließend können alle extrahierten Deskriptoren in einem Deskriptor-Baum gespeichert und schließlich zur Klassifizierung verwendet werden. Das sind die letzten zwei Abschnitte: der Deskriptor-Baum Abschnitt, und der Klassifikations Abschnitt. Nachfolgend werden die Prozesse im Detail beschrieben.

4.4.1 Region-growing Segmentierung

In Algorithmus CVFH und OUR-CVFH wird der Segment Algorithmus, Region-growing, benutzt. So wie im Deskriptor-Baum, steht es weiter zur Verfügung. Die Grundidee des Region-Growing-Algorithmuses ist, ähnliche Pixel in einem Gebiet zusammenzufassen. Erstens werden die Punkte nach den Krümmungswerten der Punkte gespeichert. Der Region-Growing-Algorithmus geht von dem Punkt, an dem die Krümmung am kleinsten ist, aus. Dieser Punkt ist der anfängliche Startpunkt und der Bereich, in dem sich der anfängliche Startpunkt befindet, ist der glatteste Bereich. Wachsen aus dem glattesten Bereich reduziert die Gesamtzahl segmentierter Segmente und verbessert die Effizienz.

Nach dem Festlegen einer leeren Startpunktsequenz und eines leeren Cluster Bereiches wird der anfängliche Startwert zur Startpunktsequenz hinzugefügt, um anschließend Nachbarschaftspunkte zu suchen. Für jeden Nachbarschaftspunkt wird der Winkel zwischen der Normalen des Nachbarschaftspunktes und der Normalen des aktuellen Startpunktes, verglichen und geprüft. Wenn der Winkel kleiner ist als die Glättungsschwelle, wird der aktuelle Punkt zum aktuellen Bereich hinzugefügt. Dann wird der Krümmungswert jedes Nachbarschaftspunkts erfasst und der kleine Krümmungsschwellenwert wird zu der Seed-Sequenz hinzugefügt. Danach den aktuellen Startpunkt zu löschen und die obigen Schritte zu wiederholen, bis die Startsequenz leer ist.

In der Segmentierung spielen die Oberflächennormalen eine wichtigen Rolle. Ein Beispiel darüber wird in Abbildung 4.15 visualisiert. Das visualisierte Bild wird nachfolgend mit dem region-growing-Algorithmus in 23 Teilen unterteilt. Dazu wird jede 60-te Punktoberflächennormale als Linie in Abbildung 4.15 gezeigt. Die weißer Linien bezeichnen die Normalen. Um die Normalen im Detail beobachten zu können, werden folgend zwei vergrößerte Bilder gezeigt, von denen eines aus dem Kopf des Vogels ist, der andere Teil aus den Flügeln des Vogels.

Der Algorithmus kann wie folgt zusammengefasst werden:

1. Eingabe Punktwolke;

2. Bild KdTree;
3. Schätzen der Oberflächennormalen
4. Regionales Wachstum
   a) k-Nachbarschaftspunkt von Startpunkt suchen
   b) prüfen, ob die Winkel zwischen Normalen des Nachbarschaftes und der Startpunkt kleiner als Glättungsschwelle
   c) wenn es a, b zutrifft, kann dieser Punkt in dem glatten Gebiet zusammengefasst werden
   d) wenn es nur a zutrifft, wird der Punkt weiter klassifiziert
   e) Fängt von einem anderen Punkt an, wiederholen a, b und c
   f) die Größe der Cluster kann nicht kleiner sein. (z.b > 50 Punkte)

4.4.2 Baum

Ein Baum als ein spezieller Typ von Graph in der Graphentheorie kann eine Monohierarchie modellieren, da er zusammenhängend ist und keine geschlossenen Pfade enthält. Graphentheoretische Bäume lassen sich in ungerichtete Bäume und gewurzelte Bäume unterteilen, und für gewurzelte Bäume in Out-Trees, bei denen die Kanten von der Wurzel ausgehen, und In-Trees, bei denen Kanten in Richtung Wurzel zeigen. Diese Arbeit wird einen Out-Tree erzeugen, wodurch die geometrische Beziehung der Deskriptoren strukturiert abgelegt wird. Ein gewurzelter Baum ist ein gerichteter von einem Knoten aus stark zusammenhängender kreisfreier Graph, der als Wurzel genannt wird.

In der Arbeit wird der Deskriptor für das gesamte Objekt extrahiert und wird als Wurzel des Baumes auf der ersten Ebene eingesetzt. Beim Erzeugen der zweiten Ebene, wird die Wurzel als der Eltern-knoten genommen und der ausgeschnittene Kind-Knoten wird als Knoten auf der zweiten Ebene eingesetzt. Danach werden die Knoten auf die zweite Ebene als Eltern-knoten weiter Kind-Knoten ausgeschnitten, die als Knoten der dritten Ebene in dem Baum hinzuzufügen sind. Dann werden die Knoten geschnitten, um dadurch einen Deskriptor-Baum zu bekommen. Bei jedem Schnitt muss man neue Eingänge und Ausgänge geben. Die neuen Eingänge bestehen aus allen vorherigen Knoten und die Ausgänge sind Knoten der neuen Ebenen. Zur gleichen

Zeit werden die Eltern-Kind-Beziehungen jedes Knotens aufgezeichnet. Damit erhöht sich die Komplexität des Schneidens.

Nachfolgend sind die wichtigen Prinzipien des Deskriptor-Baumes zusammengefasst:

1. der Deskriptor für das ganze, nicht geschnitte Objekt wird extrahiert, als Wurzel des Deskriptor-Baumes in der ersten Ebene
2. die vorherigen Knoten als Eingänge werden weiter unterteilt, die Ausgänge als Kind-Knoten in den neuen Ebenen einzusetzen
3. die Eltern-Kind-Beziehung jedes Knotens werden aufgezeichnet.
4. die Punkten-Anzahl jedes knotens werden notiert.

Einige wichtige Parameter sollten nach Test entscheiden.

1. Winkel
2. Radius
3. Min Punktanzahl

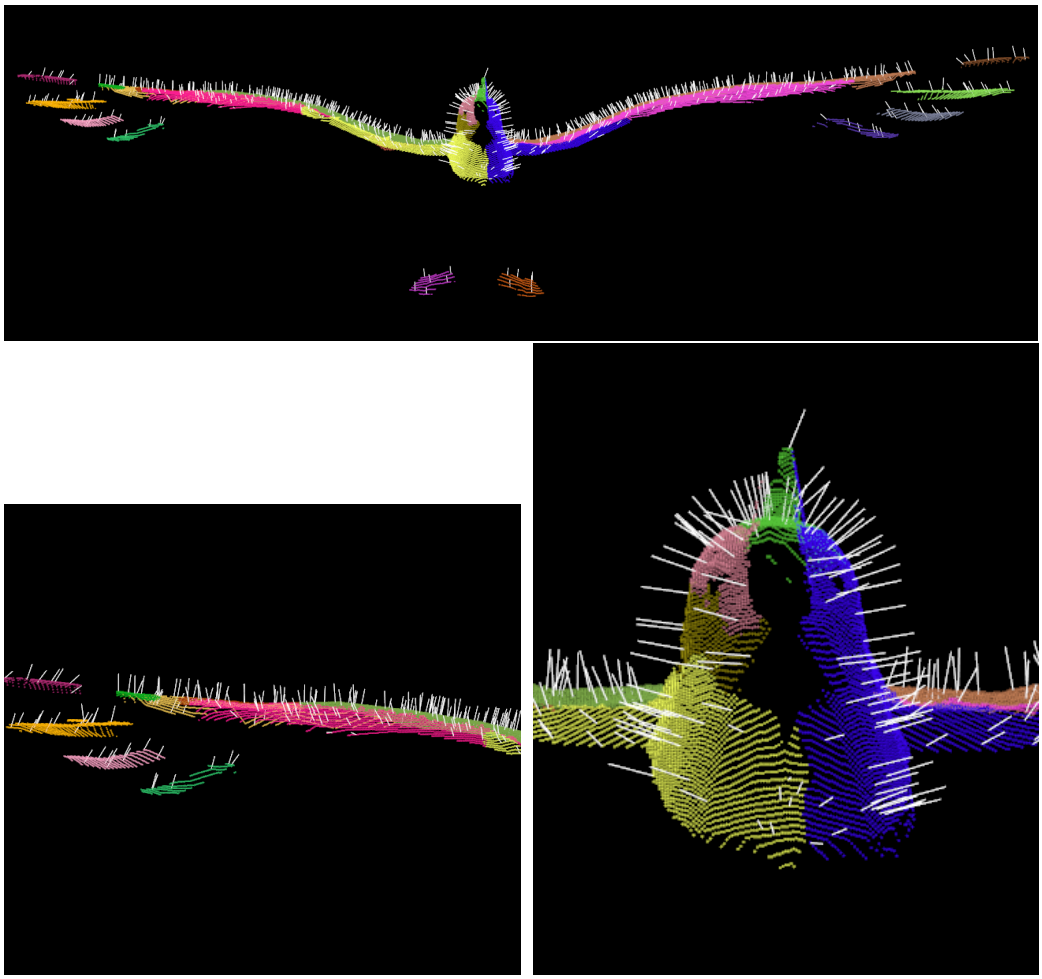


Abbildung 4.15: Point Cloud für einen Vogel und jede 60-te Punkt Oberflächennormale mit weißen Linen bezeichnet werden

4.5 Klassifzieren

In den vorherigen Abschnitten werden drei Deskriptoren aus den traditionellen Algorithmen und drei Deskriptoren, die durch neuronale Netzwerke trainiert wurden und den darauf aufgebauten Deskriptor-Baum eingeführt. Jetzt ist der letzte Schritt, die unterschiedlichen Deskriptoren, sowie Deskriptor-Bäume zu trainieren und zu klassifizieren.

Diese Arbeit ist häuptsächlich in zwei Aufgaben unterteilt. Als erstes ist es, die Beschreibungsfähigkeit der Features des traditionellen Algorithmus und des DeepLearning Methoden zu vergleichen, zu analysen und zu bewerten; zweites ist das in der Arbeit vorgestellte Optimierungsschema, Deskriptor-Baum, zu realisieren und den übereinstimmenden Baum Algorithmus zu verwenden, um die Leistung des Baumes zu bewerten.

Für den Vergleich der sechs Algorithmen wird die einfache Nächste-Nachbarn-Klassifikation verwendet. Der Grund hierfür ist, dass sich der Nächste-Nachbarn nur durch Überprüfung der Entfernung des Merkmals erreichen lässt. Aufgrund der Einfachheit sind die Testergebnisse stärker abhängig von der Leistung der Desriptoren, sodass die traditionellen und neuen Algorithmen realistischer vergleichbar sind. Für die zweite Aufgabe, um die in diesem Dokument beschriebene Baumstruktur zu ermitteln ist es notwendig einen Algorithmus anzuwenden, der spezifisch die Abstände der Bäume vergleicht. Daher wird der optimale Übereinstimmungsalgorithmus, Optimal-Matching, für Klassifizierung der Deskriptor-Bäume ausgewählt. Im Detail wird Letzteres erläutert.

4.5.1 Metrik

Abstandsmaße sind ein wichtiger Faktor, der die Klassifizierung beeinflusst. In der Arbeit lassen sich die folgenden Metriken testen.

4.5.1.1 Euklidischer Abstand

Euklidischer Abstand ist der kürzeste Abstand zwischen zwei Punkten im n-dimensionalen Raum. Er wird z.B. in der Ebene berechnet, indem der Pythagoras für ein rechtwinkliges Dreieck von zwei Punkten aus gebildet wird. Im n-dimensionalen euklidischen Raumes $\mathbf{R}^n$ wird für zwei Punkte p und q durch die euklidische Norm $\|q - p\|_2$ des Differenzvektors zwischen den beiden Punkten so definiert:

$$d(p,q) = \|q-p\|_2 = \sqrt{(q_1-p_1)^2 + ... + (q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \tag{4.5}$$

4.5.1.2 Manhattan Abstand

Die Manhattan-Metrik ist eine Metrik, in der die Distanz zwischen zwei Punkten als die Summe der absoluten Differenzen ihrer Einzelkoordinaten definiert wird:

$$d(p,q) = \sum_i |q_i - p_i| \tag{4.6}$$

4.5.1.3 Hellinger Abstand

Der Hellingerabstand ist eine Metrik für Wahrscheinlichkeitsmaße, die sich durch Wahrscheinlichkeitsdichten darstellen. Gegeben seien zwei Wahrscheinlichkeitsmaße P_1 und P_2 auf dem Ereignisraum (X, $\mathcal{A}$), die beide absolut stetig bezüglich eines σ-endlichen Maßes μ sind und somit die Dichtefunktionen f_1 und f_2 bezüglich des Maßes μ haben. Der Hellingerabstand ist dann definiert als

$$H(P_1,P_2) = (\frac{1}{2}\int_x |\sqrt{f_1} - \sqrt{f_2}|^2 d\mu)^{\frac{1}{2}} \tag{4.7}$$

4.5.1.4 Kosinus Abstand

Der in der Arbeit verwendete Kosinus Abstand basiert auf der Kosinus-Ähnlichkeit, da der Deskriptor-Abstand größer als Null sein sollte, wird die Kosinus-Ähnlichkeit erweitert, um den Kosinus Abstand zu erfüllen.

Der Begriff, Kosinus-Ähnlichkeit, verfasst sich um eine Maß für die Ähnlichkeit zweier Vektoren. Dabei wird der Kosinus des Winkels zwischen beiden Vektoren bestimmt. Der Kosinus des eingeschlossenen Winkels Null ist eins; für jeden anderen Winkel ist der Kosinus des eingeschlossenen Winkels kleiner als eins. Er ist daher ein Maß dafür, ob zwei Vektoren ungefähr in die gleiche

Richtung zeigen. Die Kosinus-Ähnlichkeit zweier Vektoren **a** und **b** ist der Kosinus des eingeschlossenen Winkels θ

$$KosinusÄhnlichkeit = cos(\theta) = \frac{a \cdot b}{\|a\|_2 \|b\|_2} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n}(a_i)^2} \cdot \sqrt{\sum_{i=1}^{n}(b_i)^2}} \tag{4.8}$$

Die Kosinus-Ähnlichkeit reicht daher von -1, genau entgegengerichtet, bis 1, genau gleichgerichtet. Ein Wert von 0 bedeutet üblicherweise Unabhängigkeit. Der Kosinus Abstand wird weiter wir Folgend abgeleitet:

$$Kosinusabstand = 1 - (KosinusÄhnlichkeit) = 1 - cos(\theta) \tag{4.9}$$

Damit verteilt der kosinus Abstand zwischen 1 und 2.

4.5.2 Nächste-Nachbarn-Klassifikation

Für das Training und die Klassifizierung im ersten und zweiten System in Abschnitt 4.1 und 4.2 wird 1-Nächste-Nachbarn-Klassifikation genutzt. Der Grund hierfür ist, dass der erste Teil der Arbeit sich auf die Deskriptoren Konzentriert, statt auf die Abstimmung verschiedener Klassifizierungssysteme. Nächste-Nachbar basiert auf dem mit einen bestimmten Metrik Feature-Vektor, bei dem eine Klassenzuordnung unter Berücksichtigung seiner ersten nächsten Nachbarn vorgenommen wird, Deshalb ist 1-Nächste-Nachbarn-Klassifikation ist empfindlich für Klassenüberlappungen, daher zeigt es auch die Feature-Schwächen, um die echte Testleistung des Deskriptores anzuzeigen.

Um die Entscheidungsmenge der zu Klassifikation k-nächsten Nachbarn zu betrachten, wird k-Nächste-Nachbarn-Algorithmus genommen. Abbildung 4.16 zeigt ein Beispeil über K-Nächste-Nachbarn in einer zweidimensionalen Punktmenge mit k=3 und k=5. In dem Beispiel gibt es zwei Klassen, die rote Klasse und die blaue Klasse, und ein unbekanntes Objekt. Mit verschiedenen Radien der Kreise werden die drei Nächsten Nachbarn und die fünf Nächsten Nachbarn betrachtet, bei dem die Zuordnung der Klasse das unbekannten Objekt ist abhängig von der größten Anzahl der Objekte dieser k Nachbarn, deshalb bei k= 3 ist der unbekannte zur roten Klasse, und bei k= 5 ist zur blauen Klasse. Bei die Klassifikation sind viele Abstandsmaße denkbar, z.b Euklidischer Abstand, Manhattan-Metrik usw.

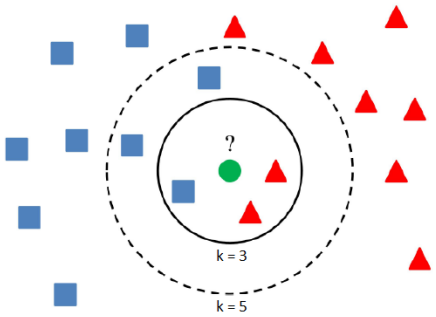


Abbildung 4.16: K-Nächste-Nachbarn in einer zweidimensionalen Punktmenge mit k=3 und k=5. Der Radius der Kreise ist nicht fest[13].

4.5.3 Optimal-Matching

Für das Training und die Klassifizierung der Deskriptor-Bäume, sowie CVFH und OUR-CVFH Deskriptoren kann Optimal-Matching verwendet werden. Optimal-Matching wird mit Hilfe Jonker-Volgenant Algorithmus[ ] realisiert, ein Algorithmus ist definiert zum Lösen gewichteter Zuordnungsprobleme auf bipartiten Graphen. Diese Problemklasse kann als Spezialfall der Linearen Optimierung formuliert werden.

4.5.3.1 Zuordnungsproblem

Zuordnungsprobleme gehören zu den speziellen Transportproblemen. Der Unterschied zum klassischen Transportproblem liegt darin, dass hier nicht Mengen mögichst kostenminimal von einem zum anderen Ort transportiert werden sollen, sondern es geht um die kostenminimale Zurodnung von Sachen, Personen oder Betriebsmittel auf bestimmte Orte, Stellen oder Aufgaben. Dabei sind alle Angebots- und Bedarfsmengen gleich 1. Mit der (zu bestimmenden) Variablen x_{ij}:

$$x_{ij} = \begin{cases} 1, & \text{falls Arbeiter i die Tätigkeit j ausführen soll} \\ 0, & \text{sonst} \end{cases} \tag{4.10}$$

und den Ausführungskosten c_{ij} ergibt sich das folgende mathematische Modell:

$$z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \rightarrow min! \tag{4.11}$$

unter den Nebenbedingungen:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1, \cdots, n) \tag{4.12}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad (j = 1, \cdots, n) \tag{4.13}$$

4.5.3.2 Jonker-Volgenant Algorithmus

Es stellt sich heraus, dass es viele lineare Optimierungsalgorithmen für allgemeine Zwecke gibt, angefangen von der Simplex-Methode bis hin zu sehr ausgereiften Lösungen. Dazu sind Hungarian-Algorithm[32] und Jonker-Volgenant-Algorithmus[31] sehr bekannt. Die Ungarische Methode wurde 1955 von Harold W. Kuhn unter Einbeziehung vorheriger Ideen der ungarischen Mathematiker Dénes Kőnig und Jenő Egerváry entwickelt und von James Munkres 1957, einer Analyse der Laufzeit folgend, verbessert. Es ist einfach zu verstehen und zu implementieren, also die beliebte Wahl in vielen Projekten. Leider funktioniert es bei größeren Problemen nur langsam.

Der Jonker-Volgenant-Algorithmus ist ein verbesserter Ansatz, der 1987 entwickelt wurde. Sein Kern ist immer noch der kürzeste argumentierende Pfad und seine Komplexität ist immer noch kubisch, aber er verwendet einige intelligente Heuristiken und Tricks, um die Rechenlast drastisch zu reduzieren. In der Arbeit wird diese Methode verwendet.

Die Anwendung des Jonker-Volgenant-Algorithmus[31] auf den Baumvergleich kann wie folgt verstanden werden. Baum A besteht aus n Knoten während Baum B m Knoten enthält. Das Problem ist, die kleinsten Knoten-Paare aus A und B herauszufinden. Wenn die Abstände zwischen Knoten aus Baum-A und Knoten aus Baum-B als Kosten bezeichnet werden, verfasst das Problem sich damit, den kleinsten Kosten zwischen A und B zu finden. Wenn Baum A weniger Knoten als Baum B hat, nämlich n<m, werden n Paare kleiner Distanzen gesucht, gegenseitig, nämlich n>m, werden m Paare zurückgegeben.

Wie in Abbildung 4.17, oben rechts sind die zwei zu vergleichenden Bäume, von denen der Baum A aus Knoten(0,1,$\cdots$, 8),und der Baum B aus Knoten(0,1, $\cdots$, 11) besteht. Unten links ist die Distanz-Matrix dazwischen, von dem die vertikale Achse Alle Knoten-Indizes aus Baum A repräsentiert, und die horizontale Achse zeigt die Knoten-Indizes aus Baum B. Und das Element in dem Matrix $m_{i,j}$ repräsentiert die Distanz zwischen Knoten-i

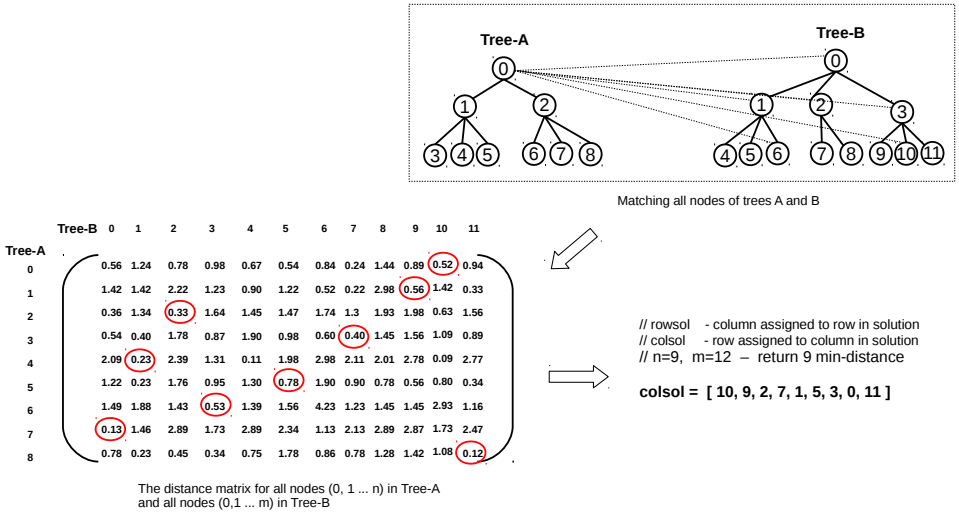


Abbildung 4.17: Optimal Matching von zwei Beispiel Bäume

von A und Knoten-j von B. Weil n kleiner ist als m,(n=9, m =13), sollten insgesamt 9 Elemente in dem Matrix herausgefunden werden, die mit Rot in der Abbildung gezeichnet werden. Die Koordinaten der vertikalen Achse, auf die sich die kleinste Entfernung befindet, werden in „ rowsol“ gespeichert, und die entsprechenden Abszisse werden in „colsol“ gespeichert.

4.5.4 Ähnlichkeitsmaß der 3D-Form

Um der Bäumen zu vergleichen, wird es durch den Vergleich von Baumähnlichkeit erreicht. Zunächst wird ein Baum als T = (V,E) mit Eckpunkt V und Kanten E definiert. Zur Erhalten der Ähnlichkeit zweier Bäume kann durch die Optimale Zuordnung der Eckpunkten erreichen, indem direkt sich die Eckpunkten der Bäume anpassen lassen, ohne die Kanten zu berücksichtigen. Die maximale übereinstimmende Gesamtähnlichkeit kann mit Hilfe Ähnlichkeit zwischen Eckpunkt V1 und V2, die als $\varphi \subset V_1 \times V_2$ bezeichnet, wie folgt berechnet:

$$W(\varphi) = \sum_{(u,v)\in\varphi} s(u,v) \tag{4.14}$$

Wobei φ eine Eins-zu-Eins-Vertex-Zuordnung zwischen zwei BäumenV_1, V_2

darstellt und s(u, v) repräsentiert die Ähnlichkeit zwischen Knoten u und v.

In der Arbeit wird Optimal-Matching benutzt, um die entsprechenden Teile der zwei Bäume zu suchen, um die Baumübereinstimmung zu berechnen. Durch der Optimale Zuordnung wird die optimale Abbildung von φ_{12} gefunden, und somit wird die durch φ_{12} akkumulierte Gesamtähnlichkeit $W(\varphi_{12})$ maximiert. Um den Verzerrungseffekt wegen der Baumgrößen zu vermeiden, sollte die Gesamtähnlichkeit $W(\varphi_{12})$ noch relativ zu den ursprünglichen Baumgrößen gewichtet werden. Dazu werden vier Metriken[29] wie folgt gelistet:

$$d_1(T_1,T_2) = max(|V_1|, |V_2|) - W(\varphi_{12}) \tag{4.15}$$

$$d_2(T_1,T_2) = |V_1| + |V_2| - 2W(\varphi_{12}) \tag{4.16}$$

$$d_3(T_1,T_2) = W(\varphi_{12})/max(|V_1|, |V_2|) \tag{4.17}$$

$$d_4(T_1,T_2) = W(\varphi_{12})/(|V_1| + |V_2| - W(\varphi_{12})) \tag{4.18}$$

4.5.5 Leave-one-out

Leve-One-Out ist ein in diesem Arbeit benutztes Konzept. In dem Klassifizierungsalgorithmus müssen die Daten in Trainingsdaten und Testdaten unterteilt werden, wobei die Trainingsdaten als eine Bibliothek verwendet werden, und die getesteten Daten werden damit verglichen, um eine Vorhersage-klasse der Testdaten zu erhalten. In diesem Konzept wird die Aufteilung von Trainingsdaten und Testdaten abgeschwächt. Alle Daten werden trainiert, dann wird eins der Daten im Vergleich zu anderen Daten einzeln extrahiert und getestet, um ihren Klasse vorherzusagen.

Kurz zusammengefasst wie folgt: Nachdem alle Daten als Trainingsdaten trainiert wurden, wird eins nacheinander extrahiert und mit die anderen getestet und getestet, bis alle Daten getestet sind. Dieses Konzept basiert auf den begrenzten Trainingsdaten und maximiert die Datenverarbeitung.

5 Implementierung

In diesem Kapitel wird die Umsetzung der Systeme beschrieben. Die Implementierung basiert hauptsächlich auf dem Klassifizierung und ist in zwei Hauptteile unterteilt. Dazu konzentriert sich der erste Teil auf die Nächste-Nachbarn-Klassifikation, um die sechs Deskriptoren zu testen. Der zweite Teil verwendet die Optimal-Matching-Klassifikation, um den Klassifizierungstest der Deskriptor-Bäume zu realisieren. Im folgenden werden die umgesetzten Systeme und somit dessen einzelne Komponenten detailliert behandelt.

5.1 Deskriptoren mit Nächste-Nachbarn System

In diesem Schritt werden die ersten wichtigen Teile implementiert, eine Klassifikation und sechs Methoden zur Extraktion von Deskriptor, drei traditionelle Methoden, VFH, CVFH und OUR-CVFH, und drei deep-learning Methoden, CaffeNet, VAE und DLR-VAE. Dabei sind CVFH und OUR-CVFH teilbasiert. Sie werden mit einer Methode, der Nächsten-Nachbarn-Klassifikation, klassifiziert.

Theoretisch werden Sie als 5 separate Systeme betrachtet: Deskriptor Extraktion mit Caffe und Klassifizierung mit Nächste-Nachbarn Methode, Deskriptor Extraktion mit Tensorflow und Klassifizierung mit Nächsten Nachbarn, Deskriptor Extraktion mit VFH und Klassifizierung mit Nächsten Nachbarn, Deskritpr Extraktion mit CVFH und Klassifizierung mit Nächsten Nachbarn und Extraktion mit OUR-CVFH und Klassifizierung mit Nächsten Nachbarn. Da sie sich aber einen gleichen Klassifikator teilen, werden sie zu einer Grafik kombiniert, um die fünf Systeme darzustellen. Abbildung 5.1 zeigt ein Komponentendiagramm dieses Gesamtsystems.

Zur Umsetzung des Gesamtsystems werden drei unterschiedliche Softwarekomponente benötigt. Dabei handelt es sich neben den zwei Komponenten zur Extraktion der Deskriptoren, CaffeNet und VAE (sowie DLR-VAE). Und die Beiden Komponenten realisieren wiederum Funktionen, die von den Komponenten „Caffe“ und „Tensorflow“ zur Verfügung gestellt werden. Zusätzlich wird mit Hilfe von „PCL“ eine wichtige Softwarekomponente entworfen.

Wie dort dargestellt stellt die Komponente „NN-Classification“ die Schnittstelle, die von den Komponeten „CaffenNet“ und „VAE/DLR-VAE“, sowie

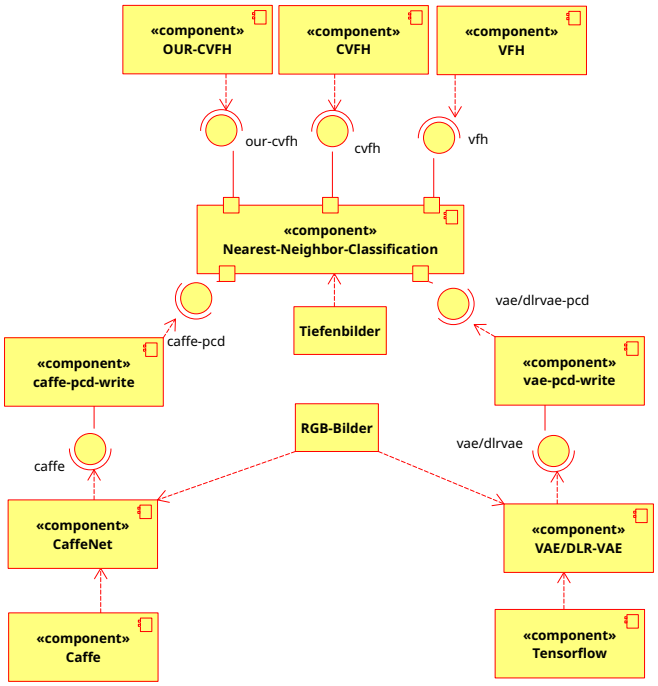


Abbildung 5.1: Komponentendiagramm des Systemes

auch die „VFH“, „CVFH“ und „OUR-CVFH“-Komponente verwendet werden, bereit, und ist der wichtigste Teil. Während „CaffeNet“ und „VAE“ unter Verwendung von Python entwickelt, werden die anderen Teile mit C++ realisiert. Im Folgenden werden die Kernschritte erläutert, die zur Umsetzung der einzelnen Komponenten und das Gesamtsystems benötigt werden.

5.1.1 CaffeNet

Um CaffeNet-Komponente umzusetzen, wird eine Deep-Learning-Frameworks, Caffe, verwendet. Dieses Paket wird bereits vom DLR zur Verfügung gestellt, und erfolgt in Python. In der Arbeit wird es benutzt, um die 3D-Daten(siehe Abschnitt 4.1) zu trainieren, und die Deskriptoren aus Layer 6 werden zum weiteren Test genutzt. In dem Abschnitt wird kurz der Zusammenbau der zu trainierenden sechs Schichten erläutert, sowie die damit trainierten Deskriptoren.

Wie in Abschnitt 4.2.1 beschrieben verwendet CaffeNet ein achtschichtiges Modell, welchem die ersten fünf Schichten aus Faltungschichten bestehen, und letzten 3 Schichten full-connection-Layers sind. CaffeNet ist vor-trainiert,

damit müssen zuerst vor-trainierte Gewichtungen geladen werden, die aus dem Caffe Model bezogen werden. In Listing 5.1 wird das Herunterladen der Konfigurationsdatei sowie die Gewichtung und der Modus des Netzes mit Zeile 3 beschrieben. Bei full-connection-Schicht „fc6“ werden die Features mit 4096 Neuronen kombiniert, damit ein 4096 Vektor- Deskriptor extrahiert werden kann, wie Zeile 10 von listing 5.1 definiert. Sie werden in einer Text-Datei gespeichert, und werden weiter in PCD-Datein umgewandelt, und zur Klassifizierung in PCL verwendet.

Listing 5.1: Auzug aus der CaffeNet Datei, sie beschreibt die Featuresextraktion aus „fc6“ in Zeile 10 und die Definition des Netzes in Zeile 3

```
1 ...
2 caffe.set_mode_cpu()
3 net = caffe.Net(args.deploy_file, args.model_file, caffe.
    TEST)...
4 ...
5 print("conv1...")
6 filters = net.params['conv1'][0].data
7 feat = net.blobs['conv1'].data[0]
8 ...
9 print("fc6...")
10 feat = net.blobs['fc6'].data[0]
11 print(feat)
12 plt.title("fc6 output")
13 _ = plt.hist(feat.flat, bins=100)
14 plt.title("fc6 histogram")
15 ...
```

5.1.2 VAE/DLR-VAE

Dieser Abschnitt befasst sich mit der Umsetzung der VAE-Komponente. Die Komponenten von VAE und DLR-VAE wurden mit Hilfe des Deep-Learning-Frameworks, Tennsorflow(sieh Abschnitt 3.1) realisiert. Die Implementierung wird schon vom DLR bereitgestellt und erfolgt in Python. In dem Abschnitt wird der Zusammenbau der zu trainierenden Modelle sowie die damit trainierten Deskriptoren erläutert.

Wie in Abschnitt 4.2 vorgestellt, besteht VAE aus Encoder und Decoder, und DLR-VAE aus Encoder, Decoder und Klassifizierung. Dabei sind die Encoder und Decoder von VAE und DLR-VAE gleich.

Beim Encoder folgen der Faltung zwei Full-Connected-Layer, um die Para-

meter eines Gaußschen Prior für die Latente Mannigfaltigkeit z zu schätzen. Für das vollständig verbundene Modell werden die Zwei full-connecte-layer mit jeweils 1000 Neuronen in der Arbeit eingestellt, und die längste Dimension von Latenter Mannigfaltigkeit z werden zu 100 gewählt. Listing 5.2 zeigt die Definition des Netzes, sowie den Input des Netzes. Wie in Zeile 12 dargestellt ist die Definition von z, der als Deskriptor schließlich zur Verfügung steht. Während die ersten und zweiten Layer von Encoder in Zeile 5 und 6 definiert werden, wird die Definition vom ersten und zweiter Layer von Decoder in Zeile 7 und 8 gezeigt.

Die 100-dimensionale Feature aus z wird in einer Textdatei geschrieben und wird weiterhin in der Arbeit verwendet, so dass der 100-dimensionale Deskriptor in eine PCD-Datei in Form eines Histogramms konvertiert wird, das dann weiter in die PCL eingefügt und schließlich klassifiziert werden kann.

Listing 5.2: Auszug aus der VAE Datei, sie beshcreibt die Definition des Netzes in Zeile 12-21 und die Featuresextraktion aus n_z in Zeile 20

```
1  ...
2  if __name__ == '__main__':
3      ...
4      network_architecture = \
5          dict(n_hidden_encoder_1=1000,
6               n_hidden_encoder_2=1000,
7               n_hidden_decoder_1=1000,
8               n_hidden_decoder_2=1000,
9               n_hidden_mlp_1=10,
10              decoder_dist='bernoulli',
11              n_feat=40000,
12              n_z=100,
13              n_classes=11)
14
15     vae = train(network_architecture, X, X_test, y, y_test,
           learning_rate=1e-3, training_epochs=1000)
16     ...
```

5.1.3 PCD-Write

Diese Komponente ist mit Hilfe Point Cloud Liabrary realisiert, und erfolgt in C++.

Die von CaffeNet, VAE sowie DLR-VAE extrahierten Deskriptoren werden

in der Text-Datei gespeichert, wie in Abschnitt 5.1.1 und 5.1.2 erläutert, die nicht von PCL lesbar sind. Das Ziel der Komponente besteht darin, die Information der Tex-Datei im PCD umzuwandeln.

Da VAE- und DLR-VAE-Deskiptoren 100-Vektoren lang sind, und Caffe-Deskriptoren 4096 Vektoren haben, gibt es keinen passenden Datentyp in PCL, deshalb ist es zuerst nötig, eine neue Datentype zu definieren, welche *pcl::Histogram<4096>* für CaffeNet definiert wird, und *pcl::Histogram<100>* für VAE und DLR-VAE ist.

Nach dem Schreiben aller Daten können sie mithilfe IO-Paket für PCL in einigen PCD gespeichert werden. Listing 5.3 zeigt die Definition des PCDs in Zeile 2, sowie den Daten-Schreib-Vorgang in Zeile 10-14 und Speichern des PCDs in Zeile 15.

Listing 5.3: Auszug aus der caffe-pcd-write Datei, sie beschreibt die Definition des Datentyps für Caffe-Deskriptor in Zeile 2 und den Schreiben-Vorgang aus in Zeile 10-14 und das endgültige Speichern in Zeile 15

```
1  ...
2  POINT_CLOUD_REGISTER_POINT_STRUCT(pcl::Histogram<4096>,
3                                    (float[4096], histogram,
                                        histogram)
4  )
5
6  int
7  main (int argc, char* argv[])
8  {
9      ...
10     typedef pcl::Histogram<4096> pclHist;
11     pcl::PointCloud<pclHist> cloud;
12     ...
13     for (int i=0; i<cloud.points.size(); ++i)
14       cloud.points[i] = Histogram_points[i];
15     pcl::io::savePCDFileASCII ("caffe.pcd", cloud);
16     ...
17 }
```

5.1.4 CVFH

Die Umsetzung von CVFH erfolgt unter Verwendung der Point Cloud Libarary (sieh Abschnitt 3.1). Es wird auf der von der Bibliothek bereitgestellen Funktion erweitert und erfolgt in C++. In der Arbeit wird eine neue Schnittstelle hinzugefügt, um zusätzliche Informationen zu transportieren, die die

Punkte jedes Clusters beschreiben.

Die Eingabe für den Deskriptor-Extraktion-Algorithmus ist in Datentyp pcl::PointCloud<pcl::PointXYZ>, und die Ausgabetyp für den Deskriptor ist pcl::PointCloud<pcl::VFHSignature308>, die beide Typen sind von PCL definiert, und können in *pcd* Form speichert werden. Die erweiterte Schnittstelle wird in Quellcode *cvfh.hpp* hinzugefügt. Ein Beispiel ist das Interface für die Punktanzahl der Clusters, wie Listing 5.4 in Ziele 3 die Ausgabe des CVFHs gezeigt, dabei *output* die CVFH-Deskriptor in Form pcl::VFHSignature308 speichert und *cluster_size* die Clutersinformationen über die Punkte in Form std::vector<int> transportiert.

Listing 5.4: Auszug aus cvfh.hpp

```
1 ...
2 template<typename PointInT, typename PointNT, typename
    PointOutT> void
3 pcl::CVFHEstimation<PointInT, PointNT, PointOutT>::
    computeRegion (PointCloudOut &output, std::vector<int> &
    clusters_size)
4 {
5   ...
6 }
```

Beim Aufrufen des Code *cvfh.hpp*, um die Tiefeninformationen einzugeben und CVFH Deskriptoren zu erzeugen, wird ein zusätzliches Schnittstellenprogramm erforderlich, wie in Listing 5.5 gezeigt. Im erste Schritt muss zunächst die KdTree ersetzt werden. Dabei handelt es sich um das spätere Nachbarpunkte-Suchen. Der zweite Schritt ist die Bestimmung der Oberflächen-Normale der Punktewolke. Die beide Schritte sind mithilfe der PCL-Bibliothek realisiert. In Listing 5.5 zeigen die Zeilen 8-10 die Code zur Ermittlung der Oberflächen-Normale und Zeilen 3-6 sind für die Bestimmung der KTdree. Anschließend können die Deskriptoren von CVFH mithilfe der KDTree und Oberflächen-Normalen bestimmt werden. Hier wird das in der Arbeit implementierte Interface, *computRegion(PointCloudOut &output, std::vector<int> &cluters_sizs)*, verwendet, wie in Zeile 18 gezeigt.

Listing 5.5: CVFH.hpp

```
1 ...
2
3 typename pcl::search::KdTree<PointT>::Ptr kdtree (new pcl::
      search::KdTree<PointT>());
4 NormalEstimation<PointT, Normal> ne;
5 ne.setInputCloud (cloud);
6 ne.setSearchMethod (kdtree);
7
8 PointCloud<Normal>::Ptr normals (new PointCloud<Normal>());
9 ne.setRadiusSearch (radius);
10 ne.compute (*normals);
11
12 CVFHEstimation<PointT, Normal, VFHSignature308> cvfh;
13 cvfh.setInputCloud (cloud);
14 cvfh.setInputNormals (normals);
15 cvfh.setSearchMethod (kdtree);
16
17 PointCloud<VFHSignature308>::Ptr cvfhs (new PointCloud<
      VFHSignature308>);
18 cvfh.computeRegion(*cvfhs, parents, points_nummer );
19
20 ...
```

Da die Implementierung von OUR-CVFH und VFH sehr ähnlich wie CVFH ist, können sich die Implementierungsschritte von VFH und OUR-CVFH auf die obigen Schritte beziehen.

5.1.5 NN-Classification

Die NN-Classification-Komponente ist die Kernkomponente in dem ganzen System, und auch in der Arbeit implementierte wichtige Teil. Sie realsiert den Kernalgorithmus zur Klassifizierung der extrahierten Deskriptoren, die in den letzten Schritten erläutert sind. Gleichzeitig werden Übertragungsschnittstellen für die Extraktion der Deskriptoren bereitgestellt. Außerdem werden die Schnittstellen zum Eingeben der tiefen Testdatenbilder in den Deskriptoren-Extraktion-Algorithmus von der Komponente gebildet.

Die hierarchische Struktur der Komponente, die in Abbildung 5.2 angezeigt wird, besteht aus der Basisklasse, *nn_classification.h*, und drei abgeleitete Klassen, *vfh_nn_classifier.h*, *caffe_nn_classifier.h* und *vae_nn_classifier*. Dabei beschreibt die Basisklasse, *nn_classification.h*, allgemeine Eingenschaf-

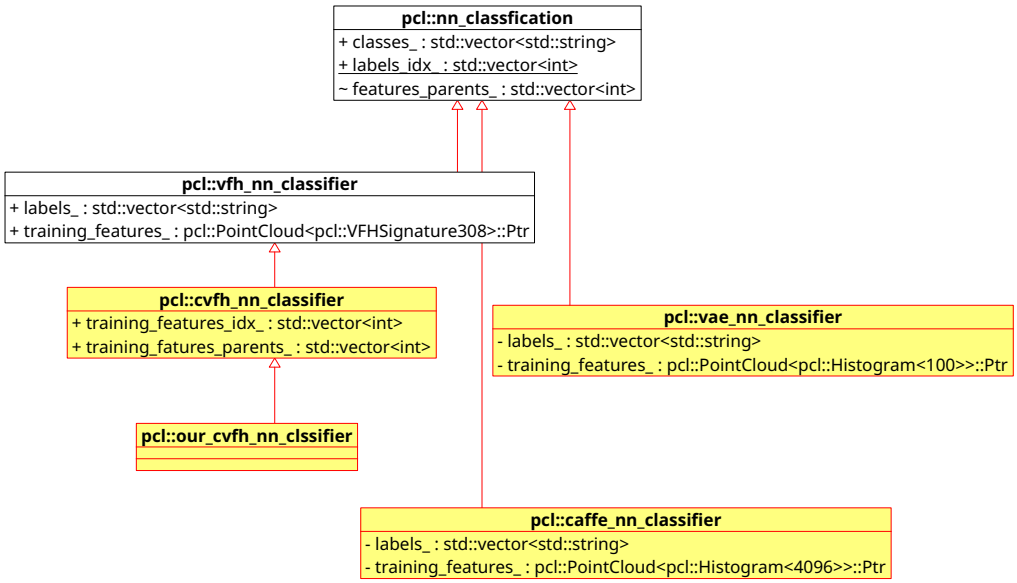


Abbildung 5.2: Die hierarchischen Struktur der NN-Classification-Komponente, der gelde Teil wird in der Arbeit implementiert, währen der weiße Teil erweitert wird.

ten und Methoden, die von anderen drei abgeleiteten Klassen übernommen werden.

Da CVFH und OUR-CVFH ihre erzeugte Deskriptoren in Datentypen, pcl::PointCloud<pcl::VFHSignature308>, speichern, die gleich wie VFH sind, erben die Klasse *cvhf_nn_classifier.h* die Eigenschaften und Methoden von *nn_classification.h*, als abgeleitete Klassen und werden entsprechend umgesetzt. Auch aufgrund der sehr ähnlichen Eingenschaften zwischen CVFH und OUR-CVFH, wurde *out_cvfh_nn_classfication.h* als abgeleitete Klasse von *cvfh_nn_classifiaction.h* implementiert.

Die sechs Klassen zur Klassifizierung der Deskriptoren erfolgt unter Verwendung der Point Cloud Library, und ist in C++ realisiert. Dabei werden Basisklasse *nn_classification.h* und eine abgeleitete Klasse, *vfh_nn_classifier.h* von PCL-Bibliothek schon bereitgestellt, Zur Realisierung einiger Methoden werden sie in Arbeit erweitert, z.b Leav-one-out, während alle anderen Klassen in der Arbeit implementiert werden. Die in der Arbeit erweiterten Funktionen von *nn_classification.h* und *vfh_nn_classifier.h* werden in Rot markiert.

5.1.5.1 nn_classfication.h

nn_classification.h als die Basisklasse beschreibt die allgemeinen Eigenschaften und Methoden, dabei werden die Kernalgorithmus, Nächste-Nachbarn-Klassifizierung(sieh Abschnitt 4.5.1) und Leave-one-out-Algorithmus in der Komponente umgesetzt. Die wichtigen Eigenschaften und Methoden von *nn_classification.h* sind schon in Abbildung 5.3 angezeigt. Die Implementierung in diesem Abschnitt erfolgt durch die Erweiterung bestehender Paket, und die in der Arbeit erweiterten Methoden und Eingenschaften werden in Rot bezeichnet und werden wie folgt detailliert beschrieben.

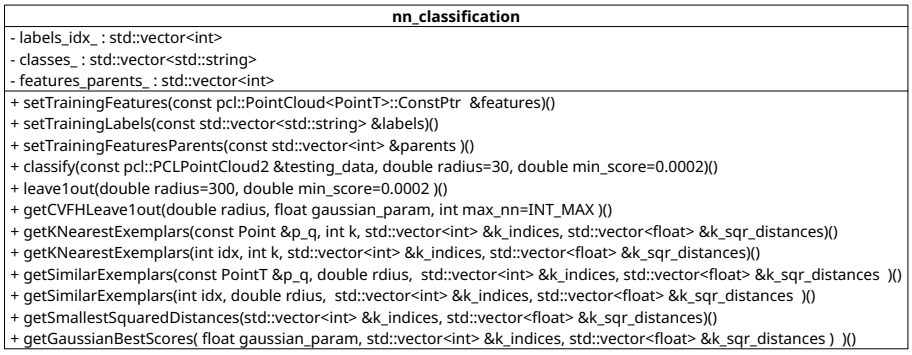


Abbildung 5.3: Klassendiagramm von nn_classification

Eigenschaften

1. `labels_idx_` : der Index in der Klassen-Labels-List für alle Trainingsdaten; im Datentyp: `std::vector<int>`

2. `classe_` : Liste der Klassen-Labels; im Datentyp: std::vector<std::string>

3. **features_parents_** : der Index der zu schneidenden TrainingDaten, die jedem unterteilten Kind-Cluster entsprechen; für CVFH und OUR-CVFh; im Datentype: `std::vector<int>` ; implementiert

Methoden

1. `setTrainingFeatures()` : Die zu trainierenden Deskriptoren werden in kd-tree eingegeben, um sie später zum Suchen der Nächsten Nachbarschaft zu verwenden

2. `setTrainingLabels()` : Die Labels von der zu trainierenden Deskriptoren werden in *labels_idx_* gespeichert, und werden klassifiziert, in *classe_* gespeichert.

3. **setTrainingFeaturesParents()** : Der Eltern Index für jeden Segmentierungsteil der Trainingsdaten werden in *features_parents_* gespeichert.

4. `classify()` : Die Testdaten werden mithilfe der Nachbarschaft klassifiziert. Input: die zu testenden Features; Output: ein Paar von Labels und Möglichkeit-Prozent für jede Trainingsklasse, dadurch dass mit Hilfe der Nachbarschaft geprüft werden kann.

5. **leave1out()** : Jede Trainingsdatei wird automatisch mit alle anderen verglichen und dadurch klassifiziert. Output: ein Paar von Labels und Möglichkeit-Prozent für jede Trainingsklasse

6. **getCVFHLeave1out()** : Diese Methode ist besonders für teil-basierte Deskriptoren, CVFH und OUR-CVFH. Alle segmentierten Deskriptoren weden als Trainingsdaten automatisch mit alle segmentierten Deskriptoren aus anderen Objekten verglichen. Output: ein Paar von Labels und Möglichkeit-Prozent für jede Trainingsklasse

7. `getKNearestExemplars ()` : k-nächste Nachbarn für den angegebenen Abfragepunkt werden gesucht. Input: der angegebene Abfragepunkt und die Anzahl der zu suchenden Nachbarn. Output: die resultierenden Indizien der benachbarten Punkte und die zugehörigen quantisierten Abstände zu den benachbarten Punkten

8. `getSimilarExemplars()` : Alle Nachbarn des Abfragepunkts werden in einem bestimmten Radius gesucht.

9. `getGaussianBestScores()` : Eine Punktzahl, die exponentiell mit der Entfernung für jede Klasse in einer Umgebung abnimmt, wird berechnet

5.1.5.2 vfh_nn_classfier.h

vfh_nn_classifier.h ist eine abgeleitete Klasse von *nn_classifier.h*, wie in Abbildung 5.4 dargestellt. Es handelt sich um eine spezielle Komponente zum Pfropfen zwischen Nächste-Nachbarn-Klassifizierung-Algorithmusdateien und VFH-Merkmalsextraktionsdateien. Die Komponente bietet Schnittstelle, über denen die zu trainierenden PCD-Dateien in PCL heruntergeladen

werden und sich die Informationen aller Punkte bei *vfh-classfier.h* lesen und bearbeiten lassen. Danach können die extrahierten Deskriptoren wieder von *vfh_nn_classifier.h* herunterladen, und anschließend werden sie bei *nn_classifier.h* verarbeitet. Dieser Abschnitt ist durch die Erweiterung im bestehenden Paket realisiert. Im Folgeenden werden sie detailliert beschrieben und die in der Arbeit erweiterten Methoden werden in Rot bezeichnet.

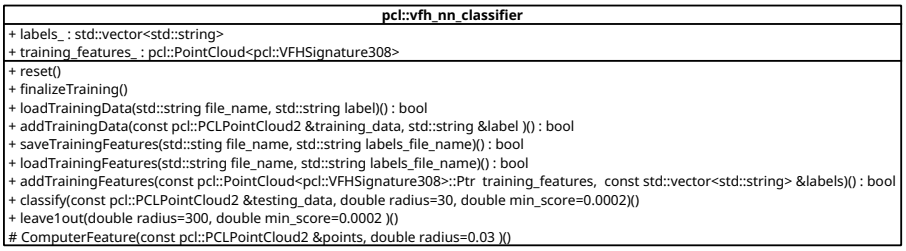


Abbildung 5.4: Klassendiagramm von vfh_nn_classifier

Eigenschaften

1. `labels_` : Liste der Klassen-Labels für alle Trainingsdaten; im Datentyp: std::vector<std::string>

2. `training_features_` : Punktwolke mit den zu trainierenden VFH-Deskriptoren; im Datentyp: pcl::PointCloud<pcl::VFHSignature308>::Ptr

Methoden

1. `reset()` : *training_features* und *labels_* werden neu gesetzt.

2. `loadTrainingData()` : die Punktwolken an der angegebenen Stelle werden eingegeben, die zur Deskriptor-Extraktion verwendet werden.

3. `addTrainingData()` : Die aus der Punktwolke extrahierte Deskriptoren werden als Trainingsdaten mit den angegebenen Labels hinzugefügt.

4. `saveTrainingFeatures()` : Die Liste der zu trainierenden Deskriptoren wird in PCD-Datei gespeichert. Die Liste der entsprechenden Labels wird in einer Text-Datei gespeichert.

5. `loadTrainingFeatures()` : Die zu trainierenden Deskriptoren und die entsprechenden Labels werden in PCL heruntergeladen.

6. `addTrainingFeatures()` : Die zu trainierenden Deskriptoren werden in *training_features_* gespeichert und die entsprechenden Labels werden in *labels_* gespeichert.

7. `finalizeTaining()` : Der NN-Klassifikator wird mit den aktuellen Trainingsdeskriptoren und Labels konfiguriert. Die gespeicherten *training_features_* und *labels_* werden in *nn_classification.h* weitergeleitet.

8. `classify()` : Die zu testenden Punktwolken werden bearbeitet, um VFH-Deskriptoren zu extrahieren. Und *classify* in *nn_classification.h* wird angerufen, um die Testdaten zu klassifizieren.

9. **leave1out()** : Die Funktion *leave1out* in *nn_classification.h* wird angerufen, um alle Trainingsdaten zu klassifizieren.

10. `computeFeature ()` : das Interface zwischen *vfh.h* und *vfh_nn_classfier.h*, um die VFH-Deskriptoren mithilfe *vfh.h* zu extrahieren

5.1.5.3 cvfh_nn_classfier.h

cvfh_nn_classifier.h ist eine abgeleitete Klasse von *vfh_nn_classifier.h*, weil CVFH-Methode auf VFH basiert, und der Datentyp von den extrahierten vfh-Deskriptoren und cvfh-Deskriptoren in einem gleichen Datentyp, pcl::PointCloud<pcl::VFHSignature308>, geschrieben werden. Ein weiteres Hauptmerkmal von CVFH besteht jedoch darin, dass das eingegebene Objekt in kleine Teilen unterteilt wird, die Eingabedaten mehrere Deskriptoren entsprechen. Daher wird die Komplexität des Pakets stark vergrößert. Ein Klassendiagramm wird in Abbildung 5.5 dargestellt.

vfh_nn_classifier.h als ein Pfropfen zwischen Nächste-Nachbarn Algorithmus-Dateien und CVFH-Merkmals-Extraktion-Dateien bietet auch eine bidirektionale Schnittstelle, über welche die zu trainierenden PCD-Dateien in PCL heruntergeladen werden und weiter alle Punkte bei *cvfh-classfier.h* gelesen und bearbeitet werden, und die extrahierten Deskriptoren wieder von *cvfh_nn_classifier.h* bearbeitet werden. Da jedes Objekt mehrere Deskriptoren entspricht, müssen sie in der übergeordneten Beziehung aufgezeichnet werden, damit sie weiter klassifiziert werden können. Was in `training_features_points_` gespeichert ist, ist die Anzahl, wie viele segmentierte Teile jede Trainingsdate hat. Mit Hilfe von diesen Daten werden alle Deskriptor-Indizes in einer übergeordneten Beziehung anschließend umwandelt und gespeichert.

Im Teil der Klassifizierung und Leave-one-out kann jeder Deskriptor ein Testergebnis bekommen. Das Ergebnis enthält aller Klasse und die entsprechende Wahrscheinlichkeit jeder Klasse, Dann kann die Wahrscheinlichkeit jeder Klasse, die jedem Teil entspricht, gewichtet, gemittelt und zu einem Ergebnis kombiniert werden, die wird bei Methode `getAverageClusterScores()` realisiert. Oder der Klassentyp des Objektes kann über den optimalen Teil bestimmen, der bei Methode `getBestClusterScores()` realisiert wird. Im Folgenden werden sie detailliert erläutert.

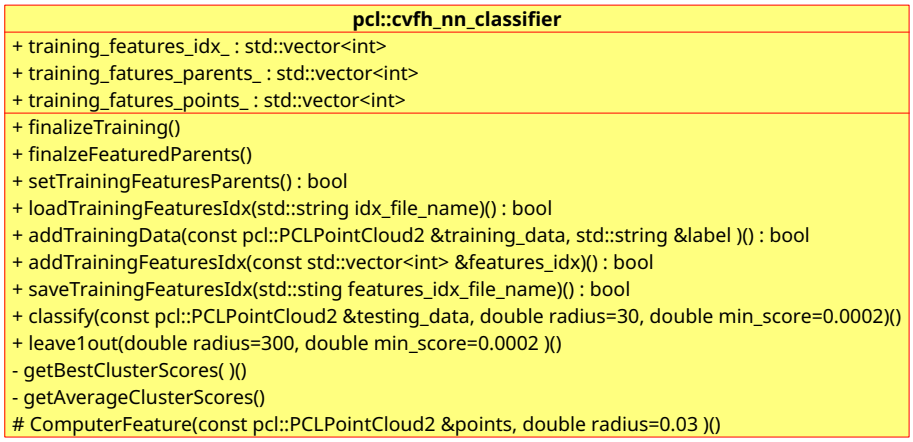


Abbildung 5.5: Klassendiagramm von cvfh_nn_classifier

Eigenschaften

1. `training_features_idx_` : Liste der Anzahl, wie viele segmentierte Teile jede Trainingseinheit hat; im Datentyp: std::vector<int>

2. `training_features_parents_` : Liste der Eltern, die mit segmentierten Kindsdeskriptoren verknüpft sind. im Datentyp: std::vector<int>

3. `training_features_points_` : Liste der Anzahl, wie viele Punkte jeder segmentierte Teil hat im Datentyp: std::vector<int>

Methoden

1. `finalizeTaining()` : Der NN-Klassifikator wird mit den aktuellen Trainingsdeskriptoren und Labels und der enprechenden Eltern-Indizes

konfiguriert. Die gespeicherten *training_features_* und *labels_* werden in *nn_classification.h* weitergeleitet.

2. `finalizeTainingParents()` : Der NN-Klassifikator wird im enprechenden Eltern-Index konfiguriert. Die gespeicherten *training_features_* und *labels_* werden in *nn_classification.h* weitergeleitet.

3. `setTrainingFeaturesParents()` : Die Eltern, die mit segmentierten Kindsdeskriptoren verknüpft sind, werden in *training_features_parents_* gespeichert.

4. `loadTrainingFeaturesIdx()` : Die Liste der Anzahl, wie viele segmentierte Teile die zu trainierenden Deskriptoren haben, wird in PCL heruntergeladen.

5. `addTrainingData()` : Die aus der Punktwolke extrahierten Deskriptoren werden als Trainingsdaten hinzugefügt, und die angegebenen Labels werden entsprechend jedem segmentierten Deskriptor in eine Liste geschrieben.

6. `addTrainingFeaturesIdx()` : Die Liste der Anzahl, wie viel segmentierte Teilen die zu trainierenden Deskriptoren haben, wird in *training_features_idx_* beschrieben.

7. `addTrainingFeaturesPoints()` : Die Liste der Anzahl, wie viele Punkte jeder segmentierter Teil hat, wird in *training_features_points_* beschrieben.

8. `saveTrainingFeaturesIdx()` : Die Liste der Indizes der zu trainierenden Deskriptoren wird in einer Text-Datei gespeichert.

9. `saveTrainingFeaturesPoints()` : Die Liste der Punkte-Anzahl der zu trainierenden Deskriptoren wird in einer Text-Datei gespeichert.

10. `classify()` : Zuerst werden die Deskriptoren von zu testenden Punktwolken extrahiert. Da eine zu testende Punktwolke mehrere VFH-Deskriptoren generiert, wird jeder der segmentierten Deskriptoren mithilfe *classify* in *nn_classification.h* klassifiziert. Danach wird für jede der segmentierten Deskriptoren ein Ergebniss-Vektor mit einem Paar von Labels und Möglichkeit-Prozent für jede Trainingsklasse zurückkommen. Entprechend der segmentierten-Teil-Liste werden die Ergebniss-Vektoren in einer List gespeichert. Anschießend werden alle

Klassifizierungs-Ergebnisse-Vektor-Liste als Eingabe von *getAverage-ClusterScores()* oder *getBestClusterScores()* weiter bearbeitet, damit ein endgültiger Ergebniss-Vektor von jeder Test-Datei erhältlich ist.

11. `leave1out()` : Die Funktion *getCVFHLeave1out* in *nn_classification.h* wird aufgerufen, um alle segmentierten Trainingsdeskriptoren zu klassifizieren, danach werden alle Leave1out-Ergebnisse folglich der *training_features_idx_* bearbeitet. Damit wird Leave1out-Vektoren-List entsprechend der Liste der eingegebenen Punktwolke gesichert, und werden sie wieder als Eingabe von *getAverageClusterScores()* oder *getBestClusterScores()* bearbeitet, damit ein endliches Leave1out-Ergebniss zustande kommt.

12. `getBestClusterScores()` : Input: ein Liste von Vektoren, jeder Vektor ist mit einem Paar von Labels und Möglichkeit-Prozenten für jede Trainingsklasse beschrieben. Mithilfe von Möglichkeit-Prozenten für jede Trainingsklasse wird das Höchste Prozent herausgefunden, das entsprechend Paar von Labels und Prozenten wird als klassifizierte Klasse der zu testenden Daten zurückgegeben.

13. `getAverageClusterScores()` : Input: ein Liste von Vektoren, jeder Vektor ist mit einem Paar von Labels und Möglichkeit-Prozenten für jede Trainingsklasse beschrieben. Jeder Vektor wird kombiniert, die Möglichkeit-Prozente für die gleiche Klasse werden summiert und schließlich ermittelt. Ein Paar von Labels und ermittelten Prozente als Output wird zurückgegeben

14. `computeFeature ()` : die Interface zwischen *cvfh.h* und *vfh_nn_classfier .h*, um die CVFH-Deskriptoren mithilfe *cvfh.h* zu extrahieren

5.1.5.4 caffe_nn_classfier.h

caffe_nn_classifier.h ist eine abgeleitete Klasse von *nn_classification.h*. Es handelt sich um eine Komponente zum Pfropfen zwischen Nächste-Nachbarn-Klassifizierung-Algorithmusdateien und CaffeNet-Deskriptor. Die schon in PCD-Datein speicherte und bei CaffeNet trainierte Deskriptoren lassen sich über die bei der Komponente bereitetet Schnittstelle in PCL herunterladen, und können weiter mit Nächst-Nachbarn klassifiziert werden. Die CaffeNet-Deskriptoren werden in einem Datentype pcl::PointCloud<

pcl::Histogram<4096» gespeichert. Die wichtige Eingenschaften und Methoden von *caffe_nn_classifier.h* sind schon in Abbildung 5.6 angezeigt. Im Folgenden werden sie detailliert beschrieben.

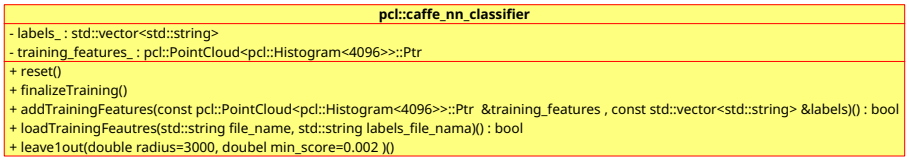


Abbildung 5.6: Klassendiagramm von caffe_nn_classifier

Eigenschaften

1. `labels_` : Liste der Klassen-Labels für alle Trainingsdaten; im Datentyp: std::vector<std::string>

2. `training_features_` : Punktwolke mit den zu trainierenden VFH-Deskriptoren; im Datentyp: pcl::PointCloud< pcl::Histogram<4096> >::Ptr

5.1.5.5 vae_nn_classfier.h

vae_nn_classifier.h ist eine abgeleitete Klasse von *nn_classification.h*. Diese Komponente fungiert als eine Rolle, die die Algorithmus-Datei des Nächsten-Nachbarn mit VAE-Deskriptoren kombiniert.

Die Deskriptoren, die in der PCD-Datei gespeichert und von den VAE trainiert wurden, können über die in der Komponente vorbereitete Schnittstelle in den PCL heruntergeladen werden und können weiter mit dem nächsten Nachbarn klassifiziert werden. Der VAE-Deskriptor wird im Datentyp pcl :: PointCloud <pcl :: Histogram <100 » gespeichert. Bei Bearbeitung von DLR-VAE funktioniert es gleich, da DLR-VAE von dlr-vae-modell trainiert, danach werden sie im Datenty pcl :: PointCloud <pcl :: Histogram <100 » geschrieben, danach können die dlrvae-Deskriptor beim *vae_nn_classifier.h* bearbeitet werden.

Um den DLR-VAE-Deskriptor zu bearbeiten, funktioniert diese Komponente auf die gleiche Art. Nachdem die Testdaten von dlr-vae-modell trainiert wurden, wird der extrahierte Deskriptor in den Datentyp pcl :: PointCloud <pcl :: histogram <100 » geschrieben, anschießend kann der dlrvae-Deskriptor von *vae_nn_classifier.h* verarbeitet werden. Die wichtigen Eigenschaften und

Methoden von emph vae_nn_classifier.h sind in Abbildung 5.7 dargestellt. Diese werden im Folgenden detailliert beschrieben.

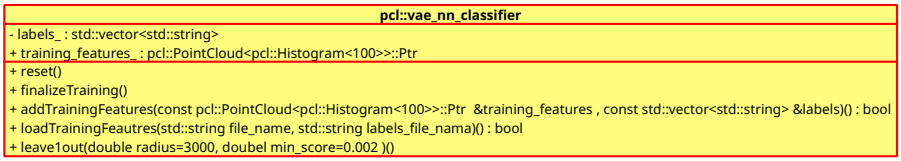


Abbildung 5.7: Klassendiagramm von vae_nn_classifier

Eigenschaften

1. `labels_` : Liste der Klassen-Labels für alle Trainingsdaten; im Datentyp: std::vector<std::string>

2. `training_features_` : Punktwolke mit den zu trainierenden VFH-Deskriptoren; im Datentyp: pcl::PointCloud< pcl::Histogram<100> >::Ptr

5.2 Deskriptor-Bäume mit Optimal-Matching System

Um die in Abschnitt 4.4 beschriebene Architektur werden drei wichtige Komponente benötigt. Dabei handelt es sich neben den zwei Komponenten zur Extraktion von Deskriptor-Baum, CVFH-Tree und OUR-CVFH-Tree. Zusätzlich wird eine Softwarekomponente entworfen, die zur Klassifizierung der Bäume dient. die CVFH-Deskriptoren und OUR-CVFH-Deskriptoren können auch mit Optimal-Matching Methoden getestet werden, die zugehörigen Komponenten werden schon in System 1 erläutert.

Wie in Abbildung 5.8 dargestellt werden Schnittstellen von OA-Komponente bereitgestellt, die von „CVFH-Tree“, „OUR-CVFH-Tree“, „CVFH“ und sowie „OUR-CVFH“ benutzt werden, um die Deskriptoren zur OA-Klassifikation weiterzuleiten. Alle Komponente wurden unter Verwendung von C++ im PCL entwickelt. Im Folgenden werden die Details des Systemes in Schritte erläutert.

5.2.1 CVFH-Tree

Dieser Abschnitt berfasst sich mit der Umsetzung der CVFH-Tree-Komponente. Das CVFH-Tree Teil wird unter Verwendung Point Cloud Library, basie-

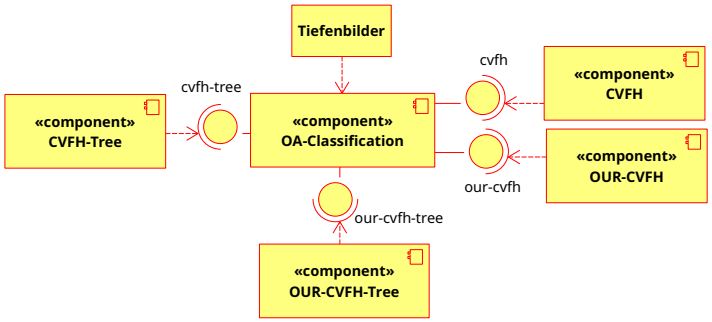


Abbildung 5.8: Komponentendiagramm des Systemes

rend auf bereitgestellte VFH-Funktion implementiert, und er erweitert die region-growing-Segmentierung Algorithmus. Diese Komponente wird dadurch realisiert, dass die Eingabedaten Tiefeninformationen mit Form PointCloud Daten sind, Ausgabe ein Baum ist, deren Knoten von VFH-Dekriptoren, und der entsprechenden Eltenknoten Information repräsentiert werden.

Der Kern der Komponente besteht aus der Segmentierung und dem Aufbau-Vorgang des Baumes. Um es klar zu erklären wird der Vorgang in einem Aktivitätsdiagramm dargestellt. Wie Abbildung 5.9 zeigt, wird durch den ersten Schritt der Umsetzung von CVFH-Tree-Komponente zunächst die Oberflächen-Normale der Punktewolke bestimmen. Dadurch wird jeder Punkt der PointCloud mit einem Parameter (x,y,z,dx,dy,dz) repräsentiert, von denen (dx,dy,dz) die Normalen-Informationen sind. Vor dem Segmentierungsprozess wird zuerst die Wurzel des Deskriptor-Baumes erzeugt, mit Hilfe aller Punkten einen VFH-Deskriptor zu extrahieren. Anschließend kommt der Segmentierungprozess.

Der Segmentierungsprozess ist periodisch, wobei die Anzahl der Zyklen den Segmentierungsparameter entsprechen. Um ein mit 6 Ebenen Baum zu bauen, sollte man zuerst den fünften Segmentierungswinkel einstellen, sowie auch das Radius-Parameter, der zum Untersuchen der Nachbarpunkte verwendet wird. Danach wird es automatisch nach jedem eingestellten Winkel unterteilt. Bei jedem Schnitt müssen einigen Merkmale beachtet werden: Erstens werden die Informationen aller Punkte in dem Cluster-Bereich, der beidem letzten Schnitt erzeugt sind, gespeichert. zweitens wird jeder geschnittene Bereich als Eingabe verwendet und zum Schnitt durchgeführt. Drittens wird der Elternknoten jedes Clusters aufgezeichnet. Als Rückgabetyp für den Deskriptor-Baum stellt <pcl:VFHSignatur308> zur Verfügung. Alle Knoten werden nacheinander in einer Liste gespeichert und die Knotenvater-Information wird in einer

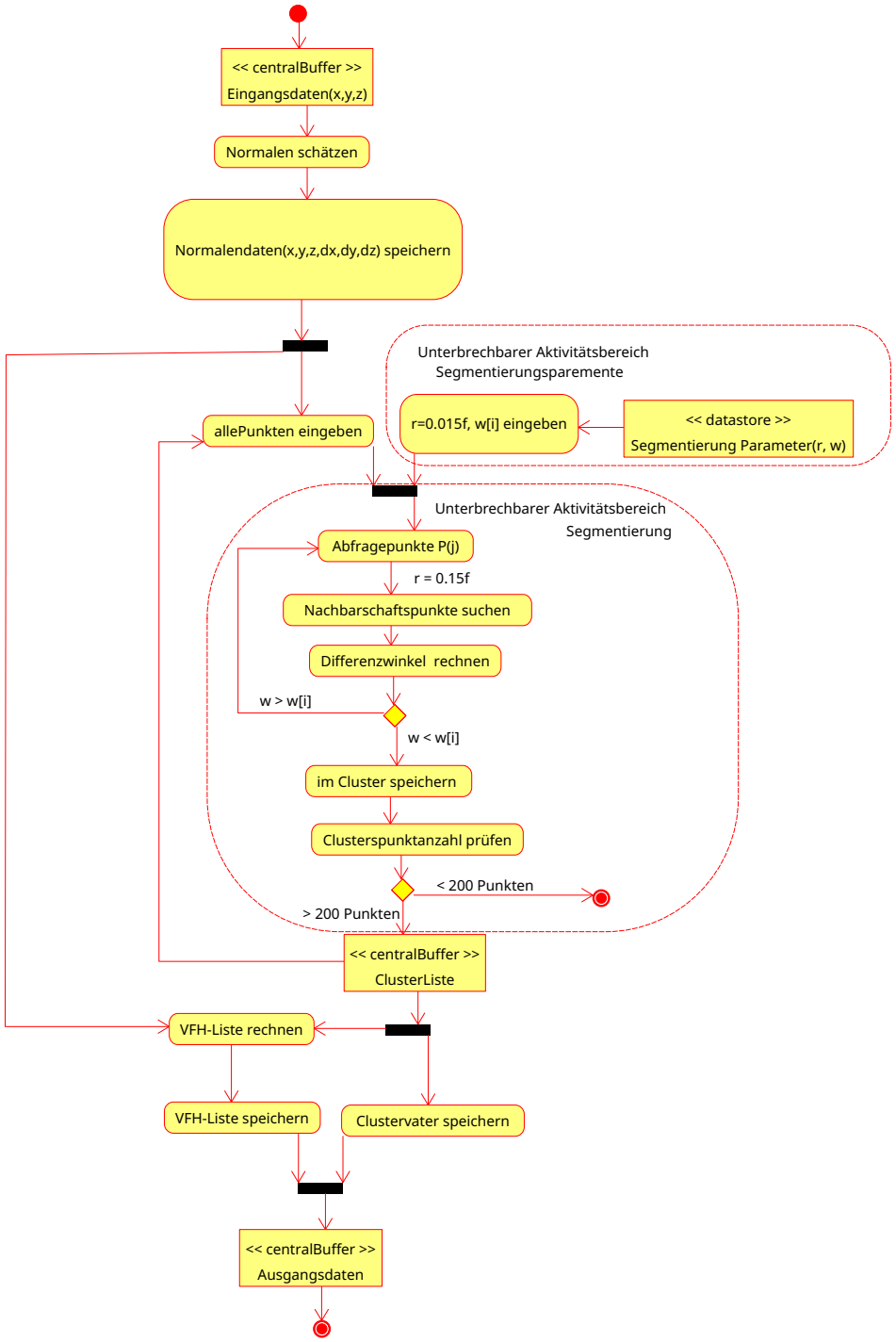


Abbildung 5.9: Aktivitätsdiagramm für CVFH-Tree

Text-Datei geschrieben.

5.2.2 OA-Classification

Die OA-Classification-Komponente realisiert die Optimal-Matching-Algorithmus sowie auch die Übertragungsschnittstellen für extrahierte Deskriptoren und Deskriptor-Bäume, und sie wird als eine Kernkomponente in diesem Abschnitt erläutert.

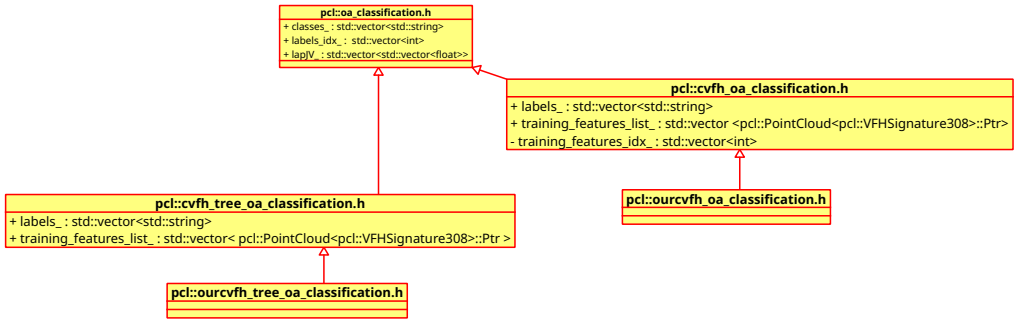


Abbildung 5.10: Die hierarchische Struktur der OA-Classification-Komponente

Die Optimal-Matching-Klassifikation lässt sich für Deskriptor-Bäume und auch teil-basierte Deskriptoren aus CVFH und OUR-CVFH verwenden. Da bei teil-basierten Deskriptoren eine zu testende Punktwolke segmentiert wird und damit mehreren Deskriptoren generiert, sowie auch ein Deskriptor-Baum Hunderte von Knoten besteht, wird eine spezielle Speicher- und Datenübertragungsmethode in dieser Komponente verwendet, anstatt die Deskriptoren in NN-Komponenten in einer Liste zu speichern. Zusätzlich bietet die Komponente auch bidirektionale Schnittstellen, über denen die zu trainierenden und testenden Tiefenbilder, Punktwolke, in PCL heruntergeladen werden und weiter nach z.b *cvfh-tree.h* leiten können, sowie die extrahierten Deskriptoren oder Deskriptor-Bäume können weiter zum Klassifikator transportiert werden.

Die OA-Komponente besteht aus der Basisklasse, *oa_classification.h*, und zwei abgeleitete Klassen, *cvfh_tree_oa_classifier.h* und *cvfh_oa_classifier.h*. Dabei beschreibt die Basisklasse, *oa_classification.h*, allgemeine Eigenschaften und Methoden, die von anderen abgeleiteten Klassen geerbt werden. Wie die Abbildung 5.10 zeigt, wird *out_cvfh_tree_oa_classfication.h* als abgleitete Klasse von *cvfh_tree_oa_classifiaction.h* implementiert. Der Grund dafür ist, dass die Knoten von CVFH-Tree und OUR-CVFH-Tree in einem gleichen Datentyp, pcl::PointCloud<pcl::VFHSignature308>, gespeichert werden. Die Datenübertragung kann ganz ähnlich sein. Aus dem gleichen Grund

erben die Klassen *cvhf_oa_classifier.h* die Eigenschaften und Methoden von *our_cvfh_oa_classification.h*, als abgeleitete Klasse.

Die Umsetzung der Komponente erfolgt unter Verwendung der Point Cloud Library, und ist in C++ realisiert.

5.2.2.1 oa_classfication.h

oa_classification.h als die Basisklasse beschreibt die allgemeinen Eigenschaften und Methoden, dabei werden die Kernalgorithmus, Optimal-Matching(sieh Abschnitt 4.5.2) und Leave1out-Algorithmus in der Komponente umgesetzt. Die wichtigen Eigenschaften und Methoden von *oa_classification.h* sind schon in Abbildung 5.11 angezeigt. Im Folgenden werden sie detailliert beschrieben.

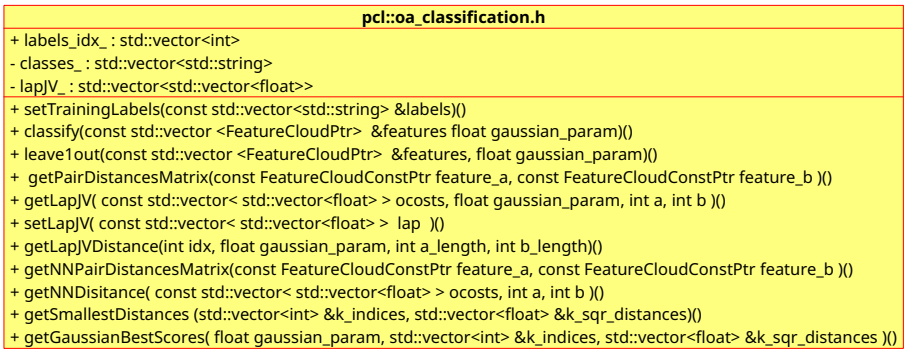


Abbildung 5.11: Klassendiagramm von oa_classification

Eigenschaften

1. `labels_idx_` : der Index in der Klassen-Labels-Liste für alle Trainingsdaten; im Datentyp: `std::vector<int>`

2. `classe_` : Liste der Klassen-Labels; im Datentyp: std::vector<std::string>

3. `lapJV_` : die Entfernung-Vektoren zwischen einem Testbaum und allen anderen Testbäumen; im Datentype: `std::vector< std::vector<float> >` ;

Methoden

1. `setTrainingLabels()` : Die Labels von den zu trainierenden Deskriptor-Bäume werden in *labels_idx_* gespeichert, und und werden klassifiziert, in *classe_* gespeichert.

2. `classify()` : Der zu testende Baum wird klassifiziert. Input: der zu testende Deskriptor-Baum; Output: ein Paar von Labels und Möglichkeit-Prozente für jede Trainingsklasse.

3. `leave1out()` : Jeder zu trainierender Deskriptor-Baum wird automatisch mit aller anderen Bäumen verglichen und mithilfe des nächsten Baumes klassifiziert. Output: ein Paar von Labels und Möglichkeit-Prozent für jede Trainingsklasse

4. `getPairDistancesMatrix()` : die Entfernung aller Knoten von zwei Bäumen wird berechnet und in einem Matrix gespeichert.Das Abstandsmatrix stellt für Optimal-Matching zur Verfügung. Output: Abstandsmatrix

5. `getLapJV ()` : Die Abstandsmatrix von zwei Bäumen wird eingegeben, und alle nächsten Punkte werden übereinstimmen. Damit wird der Abstand zwischen den zwei Bäumen berechnet. Output: die Abstände zwischen zwei Bäumen

6. `setLapJV()` : die Entfernung-Vektoren zwischen einem Testbaum und allen anderen Testbäumen werden in *lapJV_* geschrieben.

7. `getLapJVDistance()` : das Abstand zwischen den zwei Bäumen wird berechnet.

8. `getNNPairDistancesMatrix()` : die Entfernung aller Knoten von zwei Bäumen wird berechnet und in einem Matrix gespeichert. Das Abstandsmatrix wird besonders für Nächste-Nachbarn verwendet. Output: Abstandsmatrix

9. `getNNDisitance()` : der Abstand zwischen den zwei Bäumen wird berechnet, indem alle nächsten Entfernungspunkte herausgefunden werden.

10. `getSimilarExemplars()` : Alle Nachbarn des Abfragepunkts werden in einem bestimmten Radius gesucht.

11. `getGaussianBestScores()` : Eine Punktzahl, die exponentiell mit der Entfernung für jede Klasse in einer Umgebung abnimmt, wird berechnet

5.2.2.2 cvfh_tree_oa_classfier.h

cvfh_tree_oa_classifier.h ist eine abgeleitete Klasse von *oa_classifier.h* als die Basisklasse beschreibt die allgemeine Eigenschaften und Methoden, dabei werden die Kernalgorithmus, Optimal-Matching-Klassifizierung und Leave-one-out-Algorithmus in den Komponenten umgesetzt. Im Folgenden werden sie detailliert beschrieben.

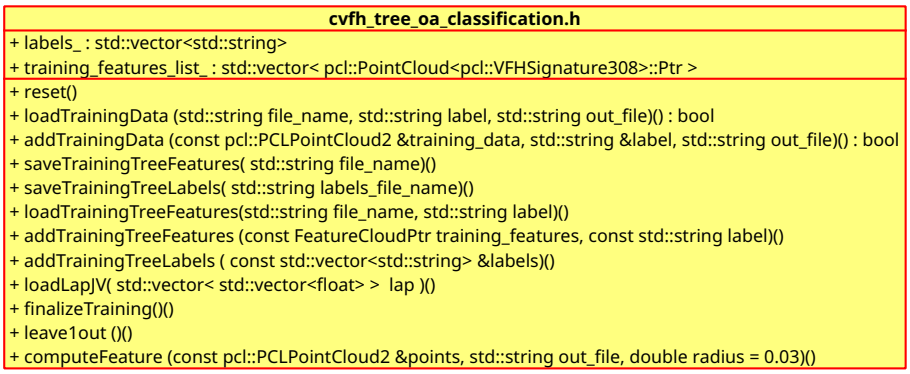


Abbildung 5.12: Klassendiagramm von cvfh_tree_oa_classifier

Eigenschaften

1. `labels_` : Liste der Klassen-Labels für alle Trainingsdaten; im Datentyp: std::vector<std::string>

2. `training_features_list_` : Liste der zu trainierenden Deskriptor-Bäume; im Datentyp:

   std::vector< pcl::PointCloud <pcl::VFHSignature308>::Ptr >

Methoden

1. `reset()` : *training_features_list_* und *labels_* werden neu gesetzt.

2. `loadTrainingData()` : die Punktwolke an der angegebenen Stelle wird eingegeben, die zur Extraktion der Deskriptor-Bäumen verwendet werden.

3. `addTrainingData()` : Die angegebenen Punktwolken wird bearbeitet, um Deskriptor-Bäume zu extrahieren, und sie werden als Trainingsdaten mit den angegebenen Labels hinzugefügt.

4. `saveTrainingTreeFeatures()` : Der trainierte Deskriptor-Baum wird in PCD-Datei geschrieben. Die Liste der entsprechenden Labels wird in einer Text-Datei gespeichert.

5. `saveTrainingTreeLabels()` : die Liste der Labels der trainierten Deskriptor-Bäume wird in einer Text-Datei gespeichert.

6. `loadTrainingTreeFeatures()` : Die zu trainierenden Deskriptor-Bäume und die entsprechenden Labels werden in PCL heruntergeladen.

7. `addTrainingTreeFeatures()` : Die zu trainierenden Deskriptor-Bäume werden in *training_features_list_* gespeichert und die entsprechenden Labels werden in *labels_* gespeichert.

8. `addTrainingTreeLabels()` : Die Labels der zu trainierenden Deskriptor-Bäume in *labels_* gespeichert.

9. `loadLapJV()` : die mithilfe von Optimal-Matching gerechneten Entfernung-Vektoren zwischen einem Testbaum und allen anderen Testbäumen werden in PCL heruntergeladen.

10. `finalizeTaining()` : *labels_* werden in *oa_classification.h* weitergeleitet.

11. `leave1out()` : Die gespeicherten *training_features_list_* werden in weitergeleitet. Die Funktion *leave1out* in *oa_classification.h* wird angerufen, um alle Trainingsdaten zu klassifizieren.

12. `computeFeature ()` : Das Interface zwischen *cvfh_tree.h* und *cvfh_tree_oa_classfier.h*, um die VFH-Deskriptor-Bäume mithilfe *vfh.h* zu extrahieren

5.2.2.3 cvfh_oa_classfier.h

cvfh_oa_classifier.h ist eine abgeleitete Klasse von *oa_classifier.h*. Die Basisklasse beschreibt die allgemeinen Eingenschaften und Methoden, dabei werden die Kernalgorithmus, Optimal-Matching-Klassifizierung(sieh Abschnitt 4.5.2) und Leave1out-Algorithmus in der Komponente umgesetzt. Im Folgenden werden sie detailliert beschrieben.

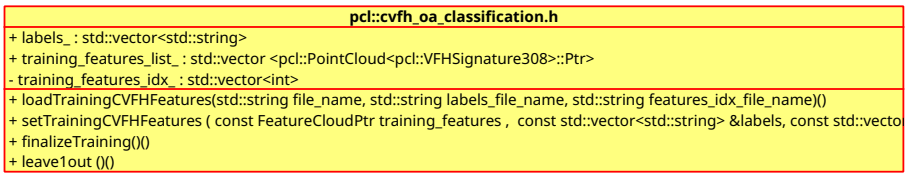


Abbildung 5.13: Klassendiagramm von cvfh_oa_classifier

Eigenschaften

1. `labels_` : Liste der Klassen-Labels für alle Trainingsdaten; im Datentyp: std::vector<std::string>

2. `training_features_list_` : Liste der zu trainierenden Deskriptor-Bäume; im Datentyp:

   std::vector< pcl::PointCloud <pcl::VFHSignature308>::Ptr >

3. `training_features_idx_` : Liste der Anzahl, wie viele segmentierte Teile jede Trainingsdate hat; im Datentyp: std::vector<int>

Methoden

1. `finalizeTaining()` : Der NN-Klassifikator wird mit den aktuellen Trainingsdeskriptoren und Labels und der enprechenden Eltern-Indizes konfiguriert. Die gespeicherten *training_features_* und *labels_* werden in *nn_classification.h* weitergeleitet.

2. `loadTrainingCVFHFeatures()` : Die zu trainierenden CVFH Deskriptoren, die entsprechenden Labels und Liste der Anzahl, wie viele segmentierte Teilen jede Trainingsdaten, werden in PCL heruntergeladen.

3. `setTrainingCVFHFeatures()` : die Index werden in *training_features_idx_* geschrieben, Die Liste der zu trainierenden Deskriptoren nachfolgend *training_features_idx_* werden zerlegt, und nacheinander in *training_features_list_* gespeichert und die entsprechenden Labels werden in *labels_* gespeichert.

4. `leave1out()` : Die Funktion *getCVFHLeave1out* in *nn_classification.h* wird aufgerufen, um alle segmentierten Trainingsdeskriptoren zu klassifizieren, danach werden alle Leave1out-Ergebnisse folgend der *training_features_idx_* bearbeitet. Damit wird Leave1out-Vektoren-List

entsprechend der List der eingegebenen Punktwolke gespeichert, und werden sie wieder als Eingabe von *getAverageClusterScores()* oder *getBestClusterScores()* bearbeitet, damit ein endliches Leave1out-Ergebnisse erreicht wird.

6 Validierung

Zur Bewertung der Deskriptoren und der implementierten Deskriptor-Bäume wird nachfolgend eine Validierung des Programms durchgeführt.

6.1 Validierungskriterien

Zum Validieren der Experimente werden 440 3D-Objektdaten aus einer Untermenge SHREC 2010 Datenbank (siehe Abschnitt 4.1) verwendet. Um die beschriebene Leistung der unterschiedlichen Deskriptoren für Datensätze mit unterschiedlichen Eigenschaften besser hervorzuheben, werden die 440 Daten in zwei Untergruppen unterteilt. : distinct-objects {Biped, Bird, Quadruped, Fish, Mug}, wobei Objekte durch wenige Details unterscheidbar sind und similar-objects{Apartment House, Flying, Insect, Non Flying Insect, Single House, Skyscraper }, wobei Objekte mit einer größeren Anzahl von Details unterschieden werden können. Sie werden wie folgt klar zusammengefasst:

1. `distinct-objects` : Biped, Bird, Quadruped, Fish, Mug
2. `similar-objects` : ApartmentHouse, FlyingInsect, NonFlyingInsect, SingleHouse, Skyscraper
3. `all-objects` : Biped, Bird, Quadruped, Fish, Mug, ApartmentHouse, FlyingInsect, NonFlyingInsect, SingleHouse, Skyscraper

Zum Validieren wird es in drei Teile unterteilt: einer dient zum Testen der traditionellen und deep-learning Deskriptoren, der zweite dient zur Visualisierung des Deskriptor-Baums, und der letzte Teil besteht darin, den Deskriptor-Baum unter Verwendung von Optimal-Matching zu klassifizieren, um die Leistung des Deskriptor-Baums zu bewerten.

1. Deskriptoren Validierung
   a) distinct-objects Validierung
   b) similar-objects Validierung
   c) all-objects Validierung

2. Deskriptor-Bäume Validierung
3. Optimal-Matching Validierung
   a) distinct-objects Validierung
   b) similar-objects Validierung

Die Endergebnisse werden mit einer korrekten Rate bewertet. Die korrekte Rate wird dadurch erzeugt, dass durch Erfassen die Korrektheit aller Daten berücksichtigt wird. Wenn die Deskriptoren eine sehr gute Formbeschreibungsfähigkeit haben, könnte die Genauigkeit der Klassifizierung innerhalb von 90% verteilt sein. Umgekehrt wird die Genauigkeit niedriger sein

6.2 Deskriptoren Validierung

Dieser Abschnitt befasst sich mit der Auswertung und dem Vergleichen der traditionellen Punkt-basierten Deskriptoren und vortrainierten oder trainierten deep-learned Features für 3D-Objekterkennung.

Zur Auswertung bestehen die Testdaten aus den 440 Daten, die im letzten Abschnitt erläutert wurden und sie werden in drei Gruppe unterteilt, um die Deskriptoren zu testen. Zur Klassifizierung der Deskriptoren wird der einfachste 1NN-Klassifikator bevorzugt, damit die Leistung der Deskriptoren besser hervorgehoben wird, und die Feature-Formbeschreibungsfähigkeit gut bewertet werden kann. Zweitens besteht die Traditionelle Methode aus VFH, CVFH und OUR-CVFH und CaffeNet und wird als trainieirte Deep-Learning Methode verwendet. Schließlich werden VAE und DLR-VAE als vor-trainierte Deep-Learning Methode eingesetzt.

Drittens, da Distanz-Metriken ein wichtiger Faktor für Klassifizierung sind, werden eine Reihe von Metriken zur Validierung betrachtet, die Manhattan(L1), Euklidisch(L2), Hellinger,χ^2 und Kosinus Distanz enthält (siehe Abschnitt 4.5.3). Für DeepLearning Deskriptoren werden nur L1, L2 und Kosinus Distanz berücksichtigt, da Hellinger und X^2 für Histogramm-basierte Deskriptoren verwendet werden. Anschließend werden die korrekte Rate von den zurückgegebenen Ergebnissen der Testdaten erzeugt, die als abschließende Bewertung zu betrachten sind.

Eine Übersicht über die besten Klassifikationsgenauigkeiten wird in Abbildung 6.1 gezeigt und die Ergebnisse für die Klassifikation aller Objekte sind in Tabelle 6.3 angegeben, während Tabelle 6.2 und 6.1 die Ergebnisse für Klassifikation von similar-Klasse und distinct-Klasse zeigen. Bildlich werden

die Verwirrungsmatrizen für die 1NN - Klassifikation der berücksichtigten Merkmale für alle Klassen, wobei die Metrik die beste Genauigkeit liefert, in Abbildung 6.4 gezeigt und Abbildung 6.2 für die distinct- Klassen und Abbildung 6.3 für die similar- Klassen.

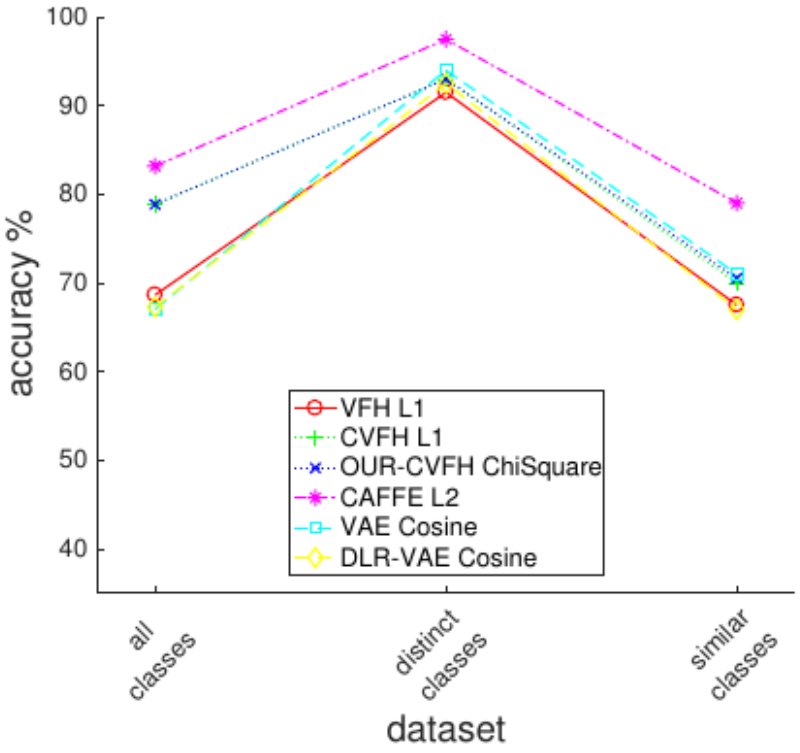


Abbildung 6.1: Klassifizierungsergebnisse. Die vertikale Achse zeigt die Klassifikationsgenauigkeit, die horizontale Achse zeigt die Klassifizierungs-Testgruppe. [2]

Bewertung

Eine Übersicht über die besten Klassifikationsgenauigkeiten wird in Abbildung 6.1 visualisiert. Der Linke Teil zeigt die endlichen Klassifizierungsergebnisse bildlich. Durch Vergleichen der drei Gruppen, all-objects, similar-objects und distinct-objects, werden die besten Metriken gefunden, die den sechs Methoden entsprechen, und die besten Ergebnisse, die auf den Bildern angezeigt

werden, werden zusammengefasst. Bildlich zeigt die vertikale Achse die Klassifikationsgenauigkeit, die horizontale Achse die Klassifizierungs-Testgruppe. Es ist deutlich zu sehen, dass CaffeNet mit L2-Metrik immer am Besten ist, gefolgt von teilbasierten Teilen, CVF mit L1 und OUR-CVFH mit χ^2 und die Leistung ist sehr ähnlich. Anschießlich kommen VFH und der VAE-Teil.

6.2.1 distinct-objects Validierung

In diesem Abschnitt stellen distinct-objects{Biped, Bird, Quadruped, Fish, Mug}, wobei Objekte durch wenige Details unterscheidbar sind, zur Verfügung, um die Deskriptoren zu bewerten.

	VFH	CVFH	OUR-CVFH	CaffeNet	VAE	DLR-VAE
L1	**91.5%**	**93%**	**93%**	**98.5%**	89.5%	82.5%
L2	86%	91%	90%	97.5%	90.5%	90%
χ^2	90%	92.5%	**93%**	-	-	-
Hellinger	90%	92%	92.5%	-	-	-
Cosine	-	-	-	-	**94%**	**92.5%**

Tabelle 6.1: Der Prozentsatz der Klassifizierungsgenauigkeit aller distinct-objects. Die Höchste Genauigkeit der Deskriptoren sind fett gedruckt.

Wie Abbildung 6.2 gezeigt, sind die Gesamtgenauigkeiten für diese Gruppe viel höher, die zwischen 90% bis 97% verteilt sind. Dabei schneiden die durch vor-trainierten CaffeNet trainierten Deep-Learning-Deskriptoren mit einer korrekten Rate von 97.50% am besten ab. Die vor VAE trainierenden und Deskriptoren sind auch ganz gut, die mit 94% Genauigkeit auf Platz 2 liegen. Danach folgen die teil-basierten Deskriptoren, CVFH und OUR-CVFH mit einer Korrektheitsrate von 93% und DLR-VAE-Deskriptoren mit 92.5%.

Bewertung

In diesen Fall ist die Gesamtgenauigkeit sehr viel höher und die Streuung in den Verwirungsmatrizen geringer. Die Gesamtverteilung der korrekten Rate lag zwischen 91% und 97%. Dabei zeigt der Deep-Learning Teil, Caffe und VAE, einige Vorteile, danach folgen teil-basierte Methoden, CVFH und OUR-CVFH. Wenn man die Abbildung 6.2 sieht, werden bei CVFH 12% Biped als Quadruped betrachtet, bei OUR-CVFH werden 10% Quardruped als Biped betrachtet, damit kann man herausfinden, dass bei

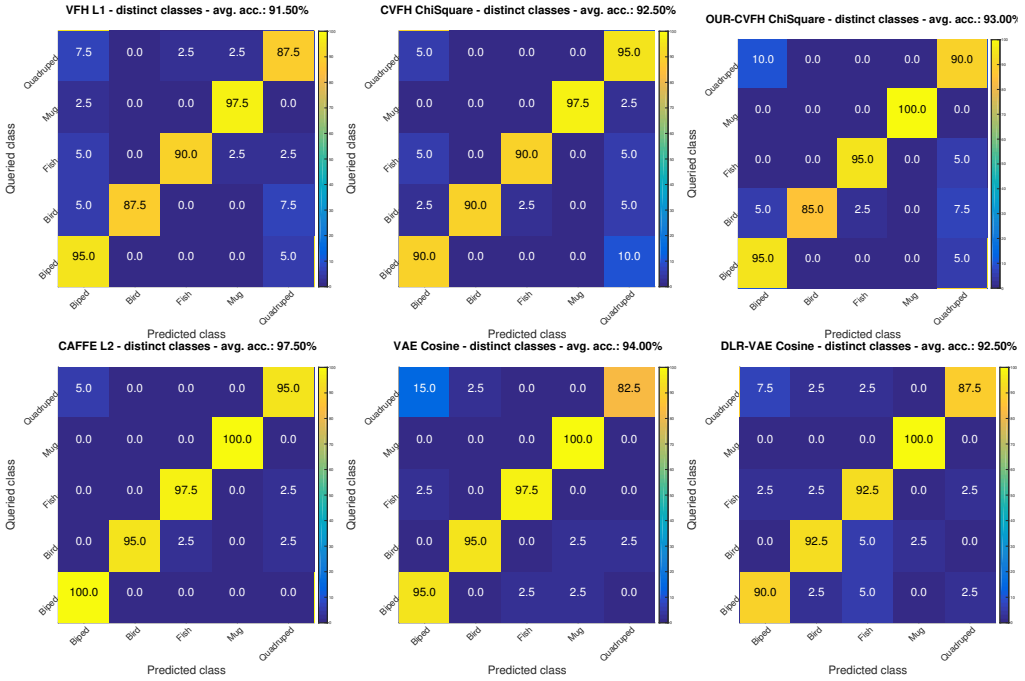


Abbildung 6.2: Verwirrungsmatrizen für distinct-objects: die vertikale Achse repräsentiert die befragten Klassen, und die die horizontale Achse zeigt die zugehörigen vorhergesagten Klassen. Die erwartete Wahrscheinlichkeit wird in die Tabelle eingetragen und farblich markiert, wobei helleres Gelb die höchste Wahrscheinlichkeit und dunkleres Blau die niedrigste Wahrscheinlichkeit zeigt[2]

distinct-objects Biped-Objekte für Quadruped-Objekte und die Quadruped-Objekte für Biped-Objekte sind. Darüber hinaus zeigte L1 für traditionelle Punkbasierte-Deskriptoren, VFH,CVFH und OUR-CVFH, in der Distanz-Metriken Vorteile, sowie auch für CaffeNet, während VAE und DLR-VAE für einen anderen Metrik, Kosinus-Distanz geeigneter ist.

6.2.2 similar-objects Validierung

In diesem Abschnitt werden Similar-objects {Apartment House, Flying, Insect, Non Flying Insect, Single House, Skyscraper}, wobei Objekte mit einer größeren Anzahl von Details unterschieden werden können, als Testdaten verwendet, um die Deskriptoren zu bewerten.

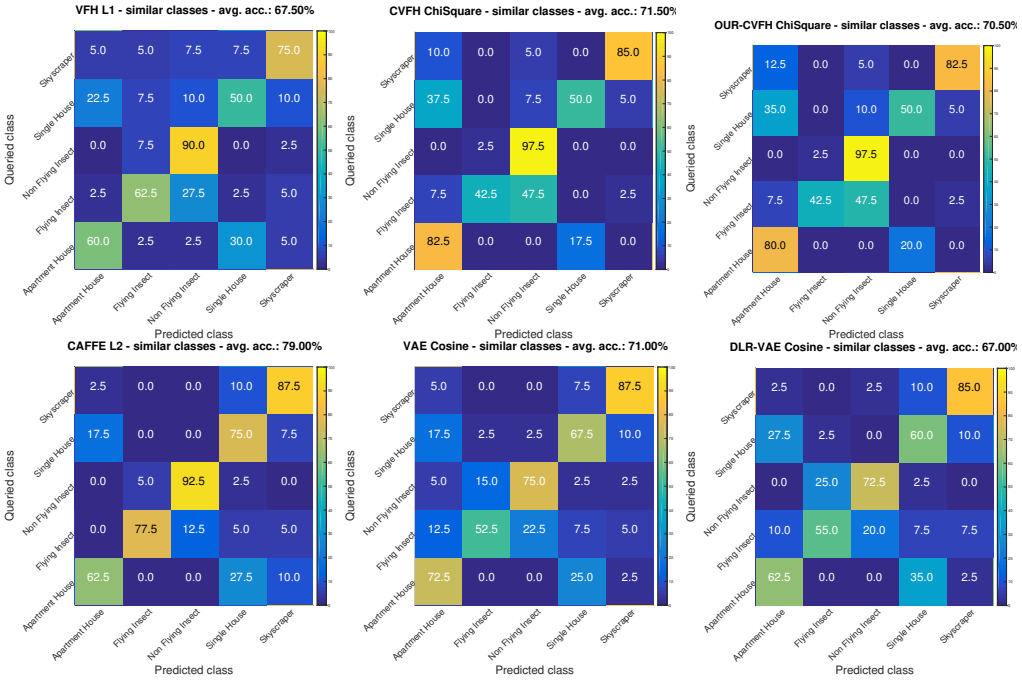


Abbildung 6.3: Verwirrungsmatrizen für similar-objects: die vertikale Achse repräsentiert die befragten Klassen, und die die horizontale Achse zeigt die zugehörige vorhergesagte Klasse. Die erwartete Wahrscheinlichkeit wird in die Tabelle eingetragen und farblich markiert, wobei helleres Gelb die höchste Wahrscheinlichkeit und dunkleres Blau die niedrigste Wahrscheinlichkeit zeigt[2]

Die zusammengefassten Ergebnisse werden bildlich in Tabelle 6.2 dargestellt, damit können wir feststellen, dass die Genauigkeit dieser Gruppe sehr gering

	VFH	CVFH	OUR-CVFH	CaffeNet	VAE	DLR-VAE
L1	**67.5%**	70%	69%	**79%**	60%	52%
L2	58.5%	70%	69%	79%	62.5%	59.5%
χ^2	67%	**71.5%**	**70.5%**	-	-	-
Hellinger	67%	69%	69.5%	-	-	-
Cosine	-	-	-	-	**71%**	**67%**

Tabelle 6.2: Der Prozentsatz der Klassifizierungsgenauigkeit aller similar-ojects. Die Höchste Genauigkeit der Deskriptoren sind fett gedruckt.

ist. Die höchste Genauigkeitsrate für jede Methode liegt zwischen 67.5% und 79%. Für diesen Testdatensatz ist L2 immer noch am besten für VFH geeignet, die mit einer 67.5% korrekten Rate die Beste für VFH liegt. Bei CVFH und OUR-CVFH zeigte L2 für diese Gruppe jedoch eine gute Leistung, das sind 71.5% für CVFH bzw. 70.5% für OUR-CVFH, die für das beste Ergebnisse am meisten eingesetzt werden. VAE und DLR-VAE haben einen Test mit Kosinus-Distanz gemacht und die besten Ergebnisse erzielt. Sie sind 67.05% für VAE und 67.27% für DLR-VAE. Detailliertere Testergebnisse für jede Testklasse sind in der Abbildung 6.3 gezeigt.

Bewertung

In diesen Fall ist die Gesamtgenauigkeit niedriger und die Streuung in den Verwirungsmatrizen höher. Die Gesamtverteilung der korrete Rate lag zwischen 67.5% und 79%. Für Objekte mit mehreren Details ist CaffeNet immer noch sehr gut und hervorragend, danach kommen immer noch teil-basiert Features aus CVFH und OUR-CVFH sowie auch VAE Teil. VFH ist relativ rückständig mit eingeschränkter Identifizierungsfähigkeit.

Bei Beobachten der detailliertere Testergebnisse für jede Testklasse in der Abbildung 6.3, kann man herausfinden, dass sich für diese Gruppe die Arten der Verwirrung hauptsächlich auf ApartmentHouse mit SingleHouse und FlyingInsect mit NonFlyingInsect konzentrieren. Die Wahrscheinlichkeit, dass sie verwirrt sind, ist sehr hoch, von denen 47% der FlyingInsect in CVFH und OUR-CVFH als NonFlyingInsect betrachtet werden, aber im Gegensatz werden sehr weniger NonFlyingInsect als FlyingInsect, falsch identifiziert. Ungefähr 36% der SingleHouse sind mit ApartmentHouse verwirrt und sie sind miteinander vermischt und stören sich gegenseitig bei der Identifizierung

Darüber hinaus zeigte L1 für traditionelle Punkbasierte-Deskriptoren,

VFH,CVFH und OUR-CVFH, in der Distanz-Metriken Vorteile, sowie auch für CaffeNet, während VAE und DLR-VAE für eine andere Metrik, Kosinus-Distanz geeigneter ist.

6.2.3 all-objects Validierung

Der Abschnitt befasst sich mit der Auswertung und dem Vergleichen der traditionellen Punkt-basierten Deskriptoren und vortrainierte oder trainierte deep-learned Features mit alle Objekten aus Klasse { Biped, Bird, Quadruped, Fish, Mug, ApartmentHouse, FlyingInsect, NonFlyingInsect, SingleHouse, Skyscraper };

	VFH	CVFH	OUR-CVFH	CaffeNet	VAE	DLR-VAE
L1	**68.64%**	**78.86%**	77.73%	82.95%	52.05%	58.64%
L2	59.82%	74.09%	78.18%	**83.18%**	57.73%	60.00%
χ^2	66.82%	78.41%	**78.86%**	-	-	-
Hellinger	67.05%	76.36%	**78.86%**	-	-	-
Cosine	-	-	-	-	**67.05%**	**67.27%**

Tabelle 6.3: Der Prozentsatz der Klassifizierungsgenauigkeit aller Daten. Die Höhesten Genauigkeit der Deskriptoren sind fett gedruckt.

Die zusammengefassten Ergebnisse werden bildlich in Tabelle 6.3 dargestellt, damit können wir feststellen, dass die Genauigkeit dieser Gruppe sehr gering ist. Die höchste Genauigkeitsrate für jede Methode liegt zwischen 67.5% und 79%. Für diesen Testdatensatz ist L2 immer noch am besten für VFH geeignet, die mit einer 67.5% korrekten Rate die Beste für VFH ist. Bei CVFH und OUR-CVFH zeigte L2 für diese Gruppe jedoch eine gute Leistung, sie sind 71.5% für CVFH bzw. 70.5% für OUR-CVFH, die als das beste Ergebnisse am höchsten einzuschätzen ist. VAE und DLR-VAE hat einen Test mit Kosinus-Distanz gemacht und die besten Ergebnisse erzielt. Sie sind 67.05% für VAE und 67.27% für DLR-VAE. Detailliertere Testergebnisse für jede Testklasse sind in dem Abbildung 6.3 gezeigt.

Bewertung

Bei der Klassifikation aller Klassen sehen wir, dass die in den beiden Teilmengen beobachtete Verwirrung auf die Gesamtklassifikation übertragen wird. Etwas interessante Verwirrung lässt sich hinzufügen, z.b Verwirrung zwischen

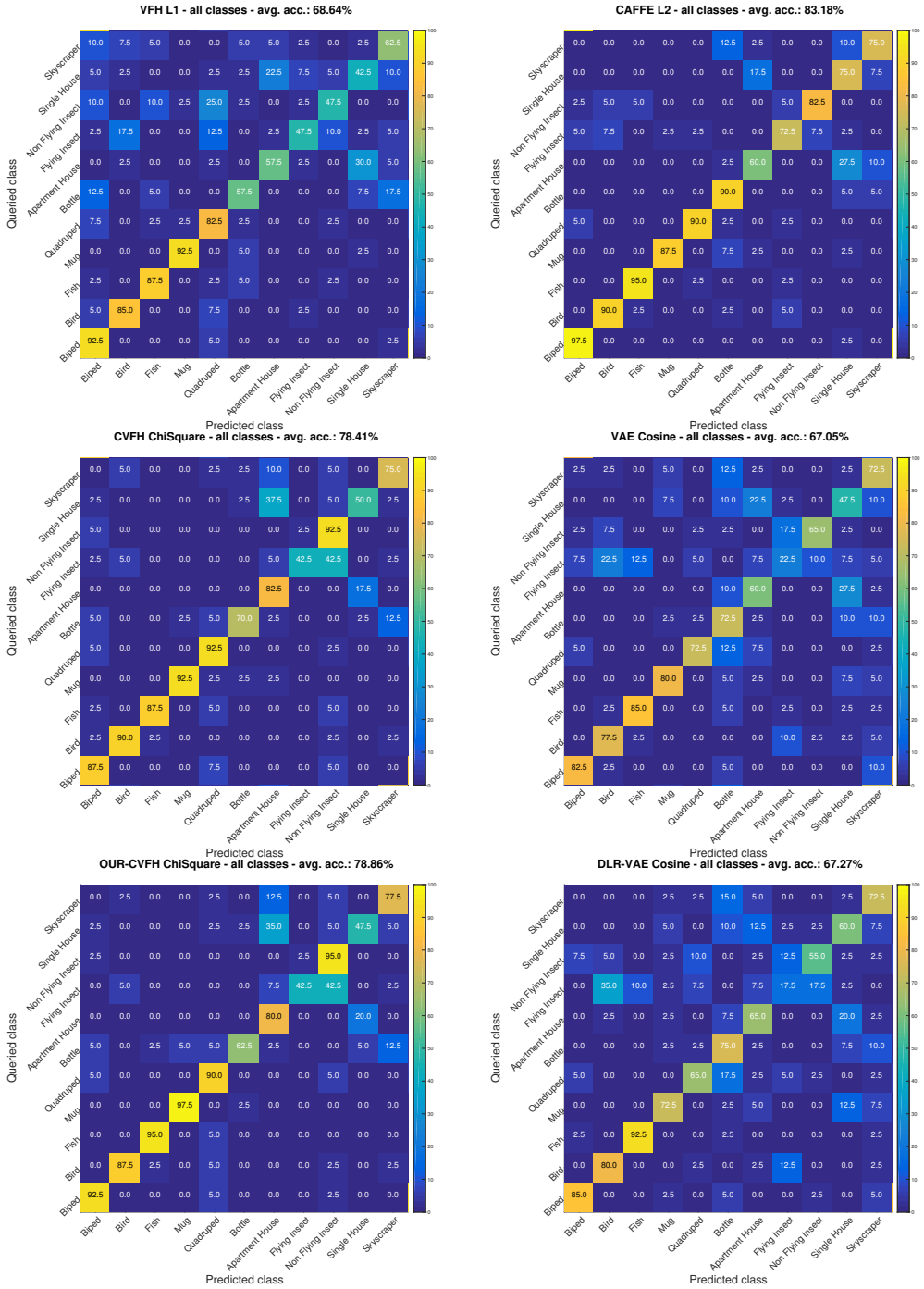


Abbildung 6.4: Verwirrungsmatrizen für all-objects: die vertikale Achse repräsentiert die befragte Klasse, und die horizontale Achse zeigt die zugehörige vorhergesagte Klasse. Die erwartete Wahrscheinlichkeit wird in die Tabelle eingetragen und farblich markiert, wobei helleres Gelb die höchste Wahrscheinlichkeit und dunkleres Blau die niedrigste Wahrscheinlichkeit zeigt. Bild aus [2].

Skyscraper und Bottle. Bei Klassifizierung der unterschiedlichen Arten extrahierter Deskriptoren zeigt es einen Unterschied im Vergleich, CaffeNet mit einem Genauigkeitsrat 83% liegt immer noch auf der besten Seite, während VAE, DLR-VAE und VFH nur etwa 68% beträgen. teil-basierte Deskriporen sind immer noch in der Mitte, die ungefähr mit einer Genauigkeit bei 78% sind. ApartmentHouse und SingleHouse, FlyingInsect und NonFlyingInsect sind immer noch hauptsächlich Objekte, die verwirrend identifiziert werden. Aber bei Identifizierung der Verwirrung-Paare, Biped und Quadruped aus der similar-Objekt Gruppe, wird es verbessert bei Klassifizierung aller Klassen.

Bei teil-basierte Dekriptoren ist die Verwirrung-Möglichkeit für ApartmentHouse und SingleHouse, FlyingInsect und NonFlyingInsect am höchsten, dabei werden ungefähr 45% der FlyingInsect als NonFlyingInsect bei CVFH und OUR-CVFH identifiziert und klassifiziert, während es nur 10% bei VFH und 7.5% bei CaffeNet beträgt. Die gleiche Situation kommt auch für ApartmentHouse und SingleHouse. Gliedmaßen und Körper von fliegenden Insekt und nicht fliegenden Insekten sehen sehr ähnlich aus und die Details der Apartment-Hause und einziger Hause sind sehr ähnlich, da beide aus Fenstern, Wänden und Dächern bestehen. Wenn sie in Regionen eingeteilt werden, dann übereinstimmen, lässt sich die Verwirrung erhöhen. Daher muss im nachfolgenden Konstruktionsprozess der Segmentierung von Objekten mit ähnlichen Details mehr Aufmerksamkeit geschenkt werden.

6.3 Deskriptor-Baum Validierung

Der Abschnitt befasst sich damit, das Schneiden und Etablieren vom Deskriptor-Baum zu validieren. Der Schlüsselpunkt, um die Bäume zu prüfen, hängt von der Langsamkeit und Rationalität jeder Schicht ab. Es lässt sich beobachten, ob es eine Änderung in jeder Schicht des Ausschnitts gibt und in der letzten Schicht, ob die minimalen Schneidblöcke auch eine Formbeschreibungsfähigkeit und Identifizierbarkeit haben. Also die Validierung dieses Abschnittes wird durch die primitivste Methode, Visualisierung des Segmentierung-Baumes, realisiert.

Nach den kontinuierlichen Tests wurde so entschieden, dass der Deskriptor-Baum mit sechs Schichten generiert wird. Die Winkelschwellen jeder Schicht sind 180°, 90°, 45°, 22° und 11°. Der Suchradius ist 0.015f. Die erste Ebene besteht darin, einen VFH-Deskriptor für das gesamte Objekt ohne Beschneidung zu extrahieren, der als der Wurzelknoten des Baums eingefügt wird. Die zweite Ebene verwendet alle Punkte in der ersten Ebene als Eingabepunkt,

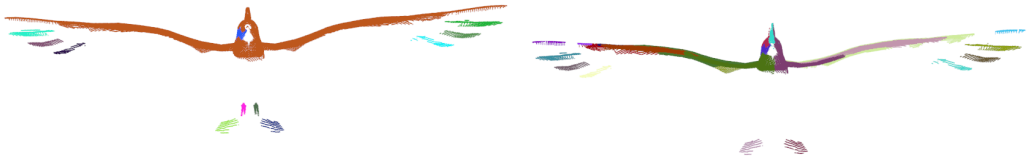
mit einem Radius f = 0.015f die Nachbarschaft vom Abfragepunkt zu finden, den Winkel zwischen allen Punkten zu schätzen und alle Punkte mit einem kleiner als 180 ° Winkeldifferenzen in einem Gebiet zu teilen. Die dritte Ebene besteht aus 90 Grad und verwendet auch alle Punkte aus zweiter Schicht und die vierte Schicht aus 45°, fünfte Ebene aus 22.5°, und sechste Ebene aus 11 °.

Um die Referenz-Ergebnisse zu erhalten, wurde eine Datei aus einigen vorgestellten Klassen gewählt, den entsprechenden Segmentierung-Baum zu erzeugen, um ein Beispiel der Klasse zu visualisieren. Sie sind die Visualisierung des Segmentierung-Baumes aus der Vogel-Klasse, die in Abbildung 6.5 gezeigt wird, die in Abbildung 6.6 gezeigte Visualisierung von Segmentierung-Baum aus Klasse-NonFlyingInsect und die Abbildung 6.7 zeigt die Ansicht des Segmentierung-Baumes aus Klasse-Biped. Das Beispiel „ Segmentierung-Baum “ aus Klasse-Quadruped wird in Abbildung 6.8 visualisert, und ein Segmentierung-Baum aus der Skyscraper-Klasse in Abbildung 6.9, ein Beispiel von Klasse SingleHousein Abbildung 6.10 und endlich visualisierte Abbildung 6.11 ein Beispiel aus der Klasse-Mug

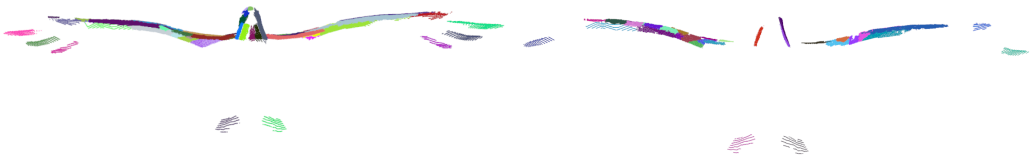
Bewertung

Beim Beobachten der Segmentierung Abbildungen, kann man es finden, dass aus Sicht der menschlichen visuellen Analyse der Segmentierungseffekt der Bäume gut ist. Die Bäume ändern sich Schicht für Schicht und die Formbeschreibungsfähigkeit und Identifizierbarkeit haben die minimalen Schneidblöcke auch. Der spezifische Klassifizierungseffekt erfordert noch den folgenden Test.

(a) Tree: Level0 - Wurzel - **(b)** Tree: Level1 - 180 -

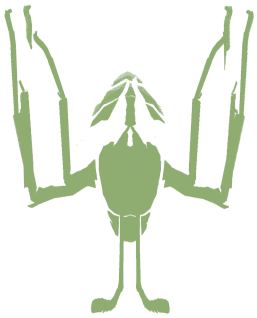


(c) Tree: Level2 - 90 - **(d)** Tree: Level3 - 45 -

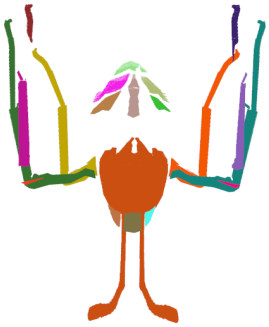


(e) Tree: Level4 - 22.5 - **(f)** Tree: Level5 - 11 -

Abbildung 6.5: Segmentierung-Baum aus {Bird}-Klasse: (a) die Wurzle des Baumes, der nicht unterteilt wird; (b) die zweite Ebene, in der unter 180° unterteilt wird; (c) die dritte Ebene, in der unter 90 unterteilt wird;(d) die vierte Ebene, in der unter 45° untetreilt wird;(e) die fünfte Ebene, in der unter 22.5° unterteilt wird; (f) die sechste Ebene, in der unter 11° unterteilt wird;

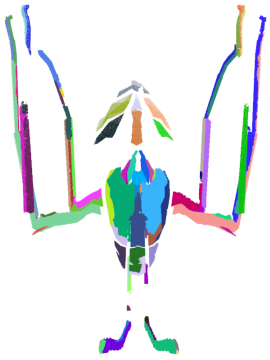


(a) Tree: Level0 - Wurzel -



(b) Tree: Level1 - 180 -

(c) Tree: Level2 - 90 -

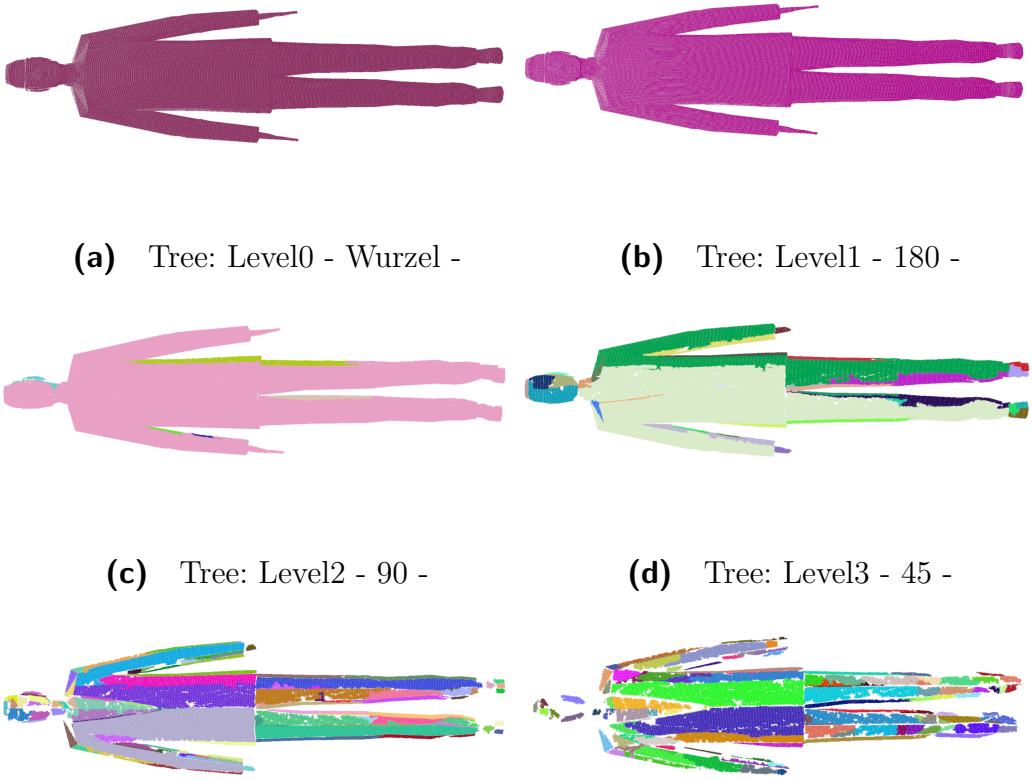


(d) Tree: Level3 - 45 -

(e) Tree: Level4 - 22.5 -

(f) Tree: Level5 - 11 -

Abbildung 6.6: Segmentierung-Baum aus {NonFlyingInsect}-Klasse: (a) die Wurzle des Baumes, der nicht unterteilt wird; (b) die zweite Ebene, in der unter 180° unterteilt wird; (c) die dritte Ebene, in der unter 90 unterteilt wird;(d) die vierte Ebene, in der unter 45° untetreilt wird;(e) die fünfte Ebene, in der unter 22.5° unterteilt wird; (f) die sechste Ebene, in der unter 11° unterteilt wird;



(a) Tree: Level0 - Wurzel - **(b)** Tree: Level1 - 180 -

(c) Tree: Level2 - 90 - **(d)** Tree: Level3 - 45 -

(e) Tree: Level4 - 22.5 - **(f)** Tree: Level5 - 11 -

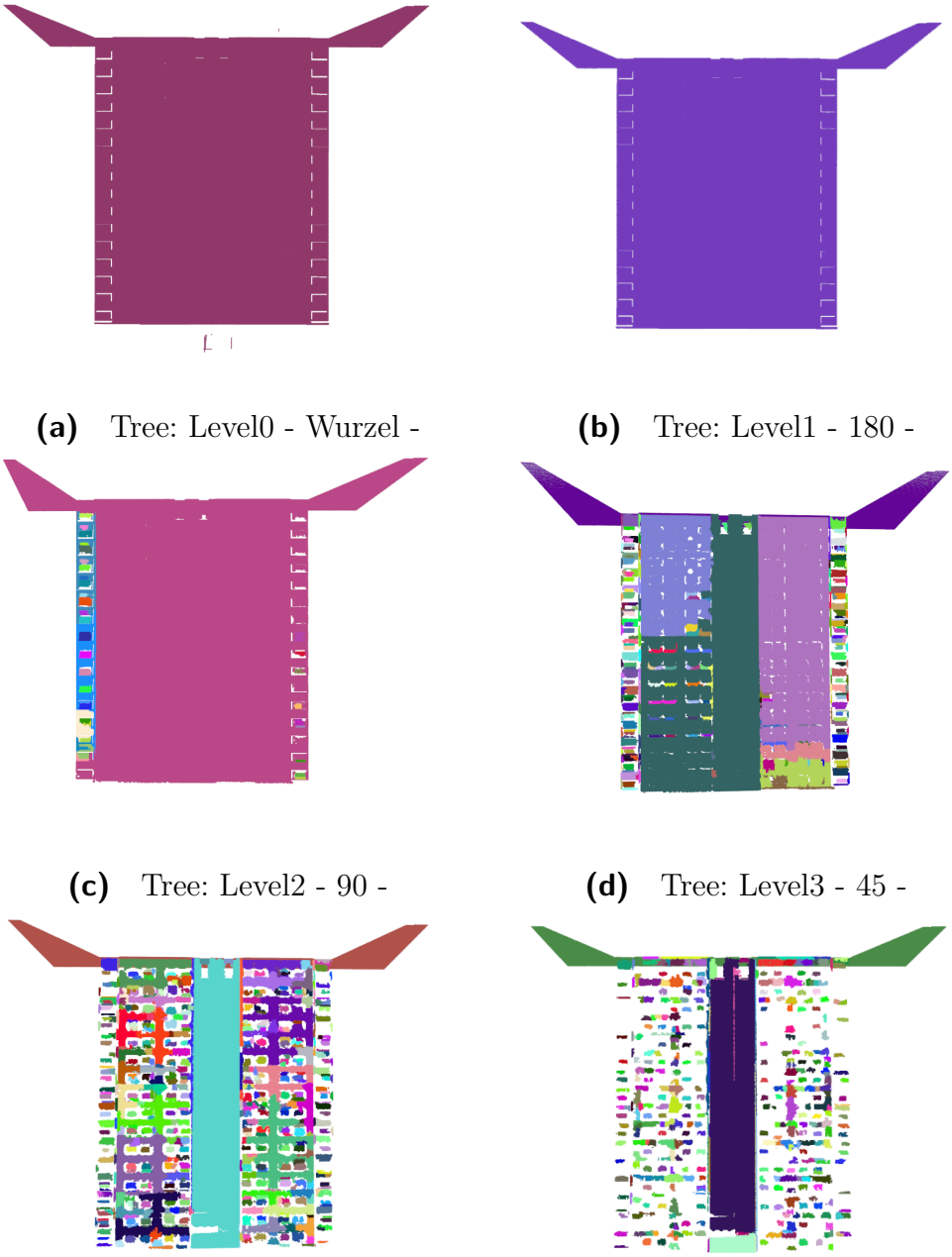
Abbildung 6.7: Segmentierung-Baum aus {Biped}-Klasse: (a) die Wurzle des Baumes, der nicht unterteilt wird; (b) die zweite Ebene, in der unter 180° unterteilt wird; (c) die dritte Ebene, in der unter 90 unterteilt wird;(d) die vierte Ebene, in der unter 45° untetreilt wird;(e) die fünfte Ebene, in der unter 22.5° unterteilt wird; (f) die sechste Ebene, in der unter 11° unterteilt wird;

(a) Tree: Level0 - Wurzel - **(b)** Tree: Level1 - 180 -

(c) Tree: Level2 - 90 - **(d)** Tree: Level3 - 45 -

(e) Tree: Level4 - 22.5 - **(f)** Tree: Level5 - 11 -

Abbildung 6.8: Segmentierung-Baum aus {Quadruped}-Klasse: (a) die Wurzle des Baumes, der nicht unterteilt wird; (b) die zweite Ebene, in der unter 180 unterteilt wird; (c) die dritte Ebene, in der unter 90° unterteilt wird;(d) die vierte Ebene, in der unter 45° untetreilt wird;(e) die fünfte Ebene, in der unter 22.5° unterteilt wird; (f) die sechste Ebene, in der unter 11° unterteilt wird;

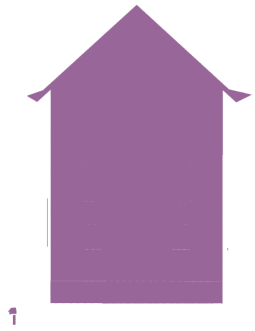


(a) Tree: Level0 - Wurzel - **(b)** Tree: Level1 - 180 -

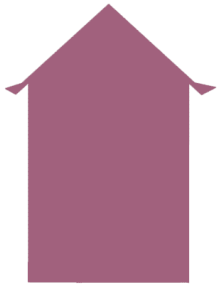
(c) Tree: Level2 - 90 - **(d)** Tree: Level3 - 45 -

(e) Tree: Level4 - 22.5 - **(f)** Tree: Level5 - 11 -

Abbildung 6.9: Segmentierung-Baum aus {Skyscraper}-Klasse: (a) die Wurzle des Baumes, der nicht unterteilt wird; (b) die zweite Ebene, in der unter 180° unterteilt wird; (c) die dritte Ebene, in der unter 90° unterteilt wird;(d) die vierte Ebene, in der unter 45° untetreilt wird;(e) die fünfte Ebene, in der unter 22.5° unterteilt wird; (f) die sechste Ebene, in der unter 11° unterteilt wird;

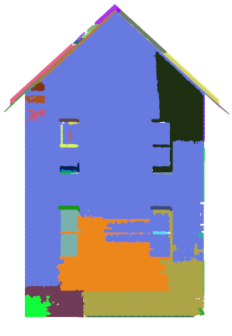


(a) Tree: Level0 - Wurzel -



(b) Tree: Level1 - 180 -

(c) Tree: Level2 - 90 -

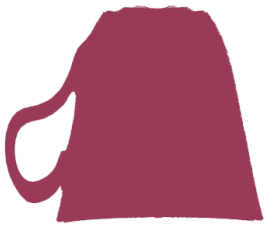


(d) Tree: Level3 - 45 -

(e) Tree: Level4 - 22.5 -

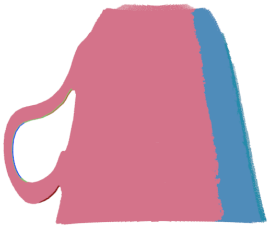
(f) Tree: Level5 - 11 -

Abbildung 6.10: Segmentierung-Baum aus {SingleHouse}-Klasse: (a) die Wurzle des Baumes, der nicht unterteilt wird; (b) die zweite Ebene, in der unter 180° unterteilt wird; (c) die dritte Ebene, in der unter 90° unterteilt wird;(d) die vierte Ebene, in der unter 45° untetreilt wird;(e) die fünfte Ebene, in der unter 22.5° unterteilt wird; (f) die sechste Ebene, in der unter 11° unterteilt wird;

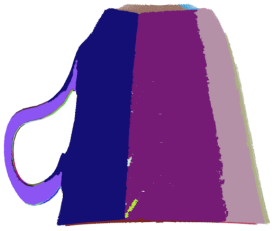


(a) Tree: Level0 - Wurzel -

(b) Tree: Level1 - 180 -



(c) Tree: Level2 - 90 -



(d) Tree: Level3 - 45 -

(e) Tree: Level4 - 22.5 -

(f) Tree: Level5 - 11 -

Abbildung 6.11: Segmentierung-Baum aus {Mug}-Klasse: (a) die Wurzle des Baumes, der nicht unterteilt wird; (b) die zweite Ebene, in der unter 180° unterteilt wird; (c) die dritte Ebene, in der unter 90 unterteilt wird;(d) die vierte Ebene, in der unter 45° untetreilt wird;(e) die fünfte Ebene, in der unter 22.5° unterteilt wird; (f) die sechste Ebene, in der unter 11° unterteilt wird;

6.4 Optimal-Matching Validierung

In diesem Abschnitt werden die Deskriptor-Bäume mit Optimal-Matching-Klassifikationsalgorithmus validiert. Zur Auswertung bestehen zuerst die Testdaten aus den 440 Daten, die in zwei Gruppen unterteilt werden(siehe Abbschnitt 6.1). Die Testdaten werden zuerst zur Deskriptor-Bäume trainiert. Anschließend werden die Bäume mit Hilfe Optimal-Matching klassifiziert.

In Abschnitt 6.3 wird ein Deskriptor-Baum mit sechs Schichten vorgeschlagen, der mit der Winkelschwellen jeder Schicht sind 180°, 90°, 45°, 22° und 11° generiert ist.

Zur Überprüfung der Leistung der Schichten werden in dieser Arbeit verschiedene Bäume separat getestet: Baum T1 mit Winkelschwellen 180°, Baum T2 mit Winkelschwellen 180°, 90°, Baum T3 mit Winkelschwellen 180°, 90°, 45°, Baum T4 mit mit Winkelschwellen 180°, 90°, 45°, 22°, und endlich Baum T5 mit mit Winkelschwellen 180°, 90°, 45°, 22° und 11°. Dazu besteht die erste Ebene immer darin, einen VFH-Deskriptor für das gesamte Objekt ohne Beschneidung zu extrahieren, der als der Wurzelknoten des Baums eingefügt wird.

Im Abschnitt 6.2 wurden verschiedene Metriken getestet, es zeigt scih, dass die Manhattan(L1) für VFH die besten sind. Daher wird in diesem Teil Manhattan verwendet. Anschließend werden die Korrektheit-Rate von den zurückgegebenen Ergebnissen der Testdaten erzeugt, die als abschließende Bewertung zu betrachten sind. Wie in Abschnitt 4.5.4 vorgestellt hat, nach dem die Gesamtähnlichkeit von Optimal-Matching gefunden wurden, sollten noch die Gesamtähnlichkeit $W(\varphi_{12})$ noch relativ zu den ursprünglichen Baumgrößen gewichtet werden[29]. In Teil werden die Folgenden zwei Metriken getestet werden.

$$d_3(T_1,T_2) = W(\varphi_{12})/max(|V_1|,|V_2|) \tag{6.1}$$

$$d_4(T_1,T_2) = W(\varphi_{12})/(|V_1| + |V_2| - W(\varphi_{12})) \tag{6.2}$$

Bewertung

Eine Übersicht über die besten Klassifikationsgenauigkeiten wird in Abbildung 6.12 visualisiert. Um die Leistung des Baumes objektiver zu bewerten, wird es mit den anderen sechs Deskriptoren, VFH, CVFH, OUR-CVFH, CAFFE,

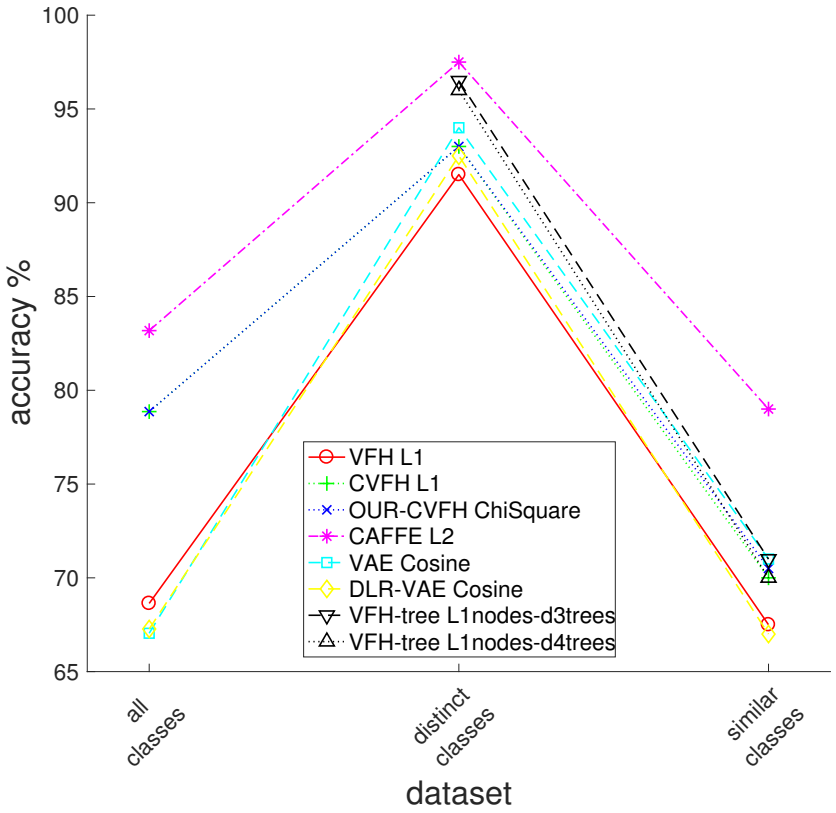


Abbildung 6.12: Klassifizierungsergebnisse. die vertikale Achse zeigt die Klassifikationsgenauigkeit, die horizontale Achse zeigt das Klassifizierungs-Testgruppe.

VAE und DLR-VAE in der Abbildung zusammen gezeigt. Aufgrund von Zeitbeschränkungen und Einschränkungen der Testgeschwindigkeit des Baums testete diese Arbeit jedoch die zwei Gruppen, similar-objects und distinct-objects, sondern fehlt es für all-Objekts Teil. Die vertikale Achse zeigt die Klassifikationsgenauigkeit und die horizontale Achse zeigt die Klassifizierungs-Testgruppe. Es ist deutlich zu sehen, dass Deskriptor-Baum im Vergleich teil-basierten Teilen verbessert ist. CaffeNet mit L2-Metrik ist immer am Besten, danach kommt Deskriptor-Baum-Teil, der auf Platz 2 liegt. Danach folgt von teilbasierten Teilen, CVF mit L1 und OUR-CVFH mit χ^2 und die Leistung ist sehr ähnlich. Anschließend kommen VFH und VAE-Teil.

6.4.1 distinct-objects Validierung

In diesem Abschnitt stehen distinct-objects{Biped, Bird, Quadruped, Fish, Mug}, wobei Objekte durch wenige Details unterscheidbar sind, zur Verfügung, um die Deskriptor-Baum zu bewerten.

Tabelle 6.4: Klassifizierungsergebnisse. es zeigt die Klassifikationsgenauigkeit von Deskriptor-Bäume mit unterschiedlichen Layers: Baum mit Winkelschwellen 180°; Baum mit Winkelschwellen 180°, 90°; Baum mit Winkelschwellen 180°, 90°, 45°; Baum mit mit Winkelschwellen 180°, 90°, 45°, 22.5° ; und Baum mit Winkelschwellen 180°, 90°, 45°, 22.5° und 11°

	180°	180°-90°	180°-90°-45°	180°-90°-45°-22.5°	180°-90°-45°-22.5°-11°
d_3	83.5%	84.5%	88.5%	90.5%	**96.5%**
d_4	**85.5%**	**87.5%**	**91.5%**	**93%**	96%

In diesem Teil werden verschiedene Bäume separat getestet: Baum T1 mit Winkelschwellen 180°, Baum T2 mit Winkelschwellen 180°, 90°, Baum T3 mit Winkelschwellen 180°, 90°, 45°, Baum T4 mit mit Winkelschwellen 180°, 90°, 45°, 22° und endlich Baum T5 mit mit Winkelschwellen 180°, 90°, 45°, 22° und 11°. Um die relative Distanz zwischen Bäumen zu berechnen, werden 6.1 und 6.2 als Metriken getestet werden. Die zusammengefassten Ergebnisse werden bildlich in Tabelle 6.4 dargestellt. Es kann herausgefunden werden, dass wenn die Schichten des Baums zunehmen, die Übereinstimmungsgenauigkeit des Baums höher wird. Der Baum mit einer separaten Ebene hat 83% Genauigkeit, während sich die Genauigkeit des Baumes mit fünf Schichten auf 96% erhöhert. Die Metrik d4 ist durchschnittlich ein bisschen besser als Metrik d3.

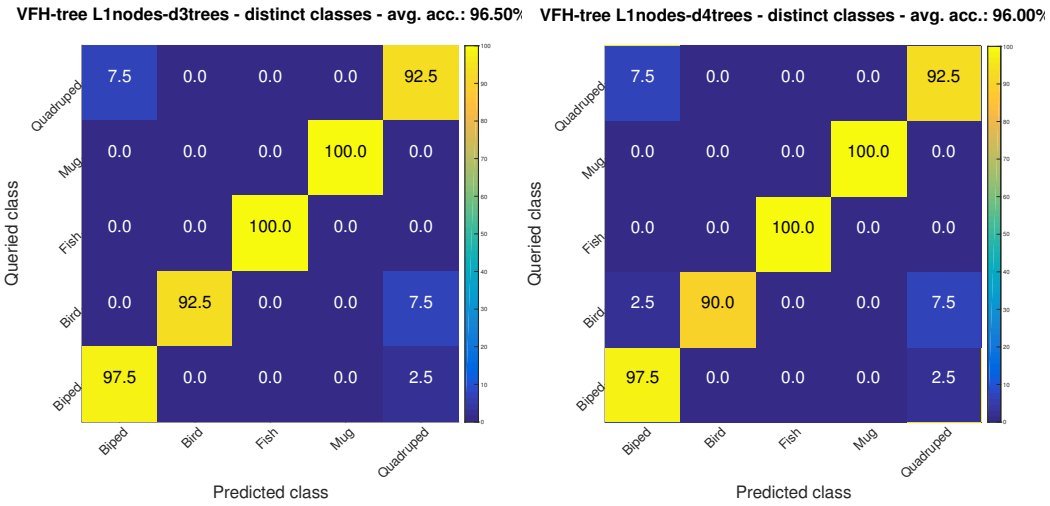


Abbildung 6.13: Verwirrungsmatrizen für distinct-objects: die vertikale Achse repräsentiert die befragte Klasse, und die horizontale Achse zeigt die zugehörige vorhergesagte Klasse. Die erwartete Wahrscheinlichkeit wird in die Tabelle eingetragen und farblich markiert, wobei helleres Gelb die höchste Wahrscheinlichkeit und dunkleres Blau die niedrigste Wahrscheinlichkeit zeigt.

Um die Detailliertere Testergebnisse für jede Testklasse zu zeigen, werden die Verwirrungsmatrizen für das besten Ergebnissen in Abbildung 6.13 zusammengefasst. Sie entsprechen den optimalen Ergebnissen aus Metrik d3 und d4.

Bewertung

Mit dem Ansteigen der Anzahl der Baumschichten erhöht sich die Genauigkeit des Baums. Der Baum mit einer separaten Ebenen hat 83% Genauigkeit, während sich die Genauigkeit des Baumes mit fünf Schichten auf 96% erhöht. Das Baum mit fünf-unterteilen Schichten mit 96% ist ein bisschen schlechter als Caffe mit 68%, aber verbessert als teil-basierte Teilen mit 93% verbessert. In den Verwirrungsmatrizen ist die Streuung viel geringer, und Mug-Klasse und Fish-Klasse betragen 100%, die in Abbildung 6.13 gezeigt werden. Die ähnlichen Objekte, Quadruped und Biped, sind immer noch verwirrend. Die Metriken d4 ist durchschnittlich ein bisschen besser als Metriken d3.

6.4.2 similar-objects Validierung

In diesem Abschnitt werden Similar-objects {Apartment House, Flying, Insect, Non Flying Insect, Single House, Skyscraper}, wobei Objekte mit einer größeren Anzahl von Details unterschieden werden können, als Testdaten verwendet, um den Deskriptor-Baum zu bewerten.

Tabelle 6.5: Klassifizierungsergebnisse. Es zeigt die Klassifikationsgenauigkeit von Deskriptor-Bäume mit unterschiedlichen Layers: Baum mit Winkelschwellen 180°; Baum mit Winkelschwellen 180°, 90°; Baum mit Winkelschwellen 180°, 90°, 45°; Baum mit mit Winkelschwellen 180°, 90°, 45°, 22° ; und Baum mit Winkelschwellen 180°, 90°, 45°, 22° und 11°

	180°	180°-90°	180°-90°-45°	180°-90°-45°-22.5°	180°-90°-45°-22.5°-11°
d_3	66%	68.5%	**70.5%**	69.5%	71%
d_4	66%	**69%**	70%	**71.5%**	**71.5%**

In diesem Teil werden auch verschiedene Bäume separat getestet: Baum T1 mit Winkelschwellen 180°, Baum T2 mit Winkelschwellen 180°, 90°, Baum T3 mit Winkelschwellen 180°, 90°, 45°, Baum T4 mit mit Winkelschwellen 180°, 90°, 45°, 22°, und endlich Baum T5 mit mit Winkelschwellen 180°, 90°, 45°, 22° und 11°. Zur Berechnung der relativ Distanz zwischen Bäumen werden Metriken 6.1 und 6.2 getestet werden. Die zusammengefassten Ergebnisse werden bildlich in Tabelle 6.5 dargestellt.

Um die Detailliertere Testergebnisse für jede Testklasse zu zeigen, werden die Verwirrungsmatrizen für das besten Ergebnissen in Abbildung 6.14 zusammengefasst. Sie entsprechen den optiamlen Ergebissen aus Metrik d3 und d4.

Bewertung

Mit dem Ansteigen der Anzahl der Baumschichten erhöht sich die Genauigkeit des Baums langsam, aber nicht so stark. Die Genauigkeit vom Baum mit fünf-unterteilen Schichten beträgt 71%, gleichbleibend wie teil-basierte Teilen. In diesem Abschnitt hat sich die Leistung des Deskriptor-Baums nicht wie erwartet verbessert.

Beim Beobachten der detailliertere Testergebnisse für jede Testklasse in Abbildung 6.14, kann man herausfinden, dass die Streuung in den Verwirungsmatrizen viel höher ist. Für diese Gruppe konzentrieren sich die Arten der

Abbildung 6.14: Verwirrungsmatrizen für similar-objects: die vertikale Achse repräsentiert die befragte Klasse, und die horizontale Achse zeigt die zugehörige vorhergesagte Klasse. Die erwartete Wahrscheinlichkeit wird in die Tabelle eingetragen und farblich markiert, wobei helleres Gelb die höchste Wahrscheinlichkeit und dunkleres Blau die niedrigste Wahrscheinlichkeit zeigt.

Verwirrung hauptsächlich auf ApartmentHouse mit SingleHouse und FlyingInsect mit NonFlyingInsect. Die Wahrscheinlichkeit, dass sie verwirrt sind, ist sehr hoch, von denen ungefähr 45% der FlyingInsect als NonFlyingInsect betrachtet werden, aber im Gegensatz werden sehr weniger NonFlyingInsect als FlyingInsect falsch identifiziert. Ungefähr 25% der SingleHouse sind mit ApartmentHouse verwirrt und sie sind miteinander vermischt und stören sich gegenseitig bei der Identifizierung

7 Zusammenfassung

Diese Arbeit konzentriert sich auf 3D-Objekterkennung und Klassifizierung, bei denen es sich um einen begrenzten Fall von 3D-Daten in Roboteranwendungen handelt. Durch das Testen mehrerer Deskriptoren können wir feststellen, dass die Leistung des CaffeNet-Features am Besten ist, während VAE am Schlechtesten ist. Die 2D-Projektion der Merkmalsdaten mit der t-SNE-Methode, wie in Abbildung C.2 gezeigt, zeigt qualitativ, dass sich die CaffeNet-Merkmale in getrennte Cluster gruppieren, während die VFH-Werte mit niedrigerer Qualität zu mehr Streuverteilung mit vielen überlappenden Bereichen zwischen verschiedenen Klassen führen. Die Leistung des Deskriptor-Baumes, der in dieser Arbeit implementiert wurde, liegt zwischen CaffeNet und dem teil-basierten Teil.

Zweitens sind ähnliche Objekte schwieriger zu unterscheiden. Bei unterschiedlichen Objektarten mit vielen Details und ähnlichen Details, wie Hochhäuser und Einfamilienhäuser, die aus Fenstern und Dächern bestehen, erhöht sich die Schwierigkeit des Erkennens, wenn sie in kleine Teile aufgeteilt werden. Gleichzeitig ist dies auch ein Nachteil für die Konstruktion von Bäumen.

Für den Teil des Baumes können wir feststellen, dass für weniger detaillierte Objekte die Aufspaltung des Baumes zu einer großen Verbesserung des Klassifikationsergebnisses führt. Für Objekte mit mehr Details, wie Insekten, hohe Gebäude und die Errichtung von Baumstrukturen, gab es jedoch keine signifikante Verbesserung. Aufgrund von Zeitbeschränkungen und begrenzter Geschwindigkeit sind Parameter, die beim Erstellen von Bäumen verwendet werden, wie Layering-Winkel, Suchradius und Parameter, die sich bereits im Klassifizierer befinden, möglicherweise nicht optimal. Wenn eine große Anzahl von Tests durchgeführt wird, werden die Parameter angepasst, und das Ergebnis sollte besser sein. Außerdem wurde in der Arbeit Optimal-Matching verwendet, um die Bäume übereinzustimmen, was nicht die beste Methode ist. Die Tree-Matching-Theorie von Torsellor[29] sollte eigentlich besser geeignet sein. In der Zukunft könnte die Methode von Torsellor zur Klassifizierung der Bäume noch optimaler getestet werden.

Anhang

A Sonstiges

CD-Inhalt

1. Tex:
   a) das Dokument PDF des Diplomarbeit
   b) das Dokument PDF der Kurzfassung
2. Bilder: alle Bildern im Dokument
3. Programm:
   a) pcl_apps_nearest_neigbors: Der Sourcecoder für Nearest-Neigbor Klassifizierung
   b) pcl_apps_oa_classification: Der Sourcecode für Optimal-Matching und Deskriptor-Baum

B Abkürzungsverzeichnis

VFH Viewpoint Feature Histogram

CVFH Clustered Viewpoint Feature Histogram

PCL Point Cloud Library

PCD Point Cloud Data

VAE Variant Autoencoder

KNN Künstliche neuronale Netze

CNN Convolutional Neural Networks

LRN Local Respons Normalization

ELU exponentiellen Linearen Einheiten

SPFH Simplified-Point-Feature-Histogram

DLR-VAE Variant Autoencoder bei DLR

OURCVFH Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram

C Abbildung

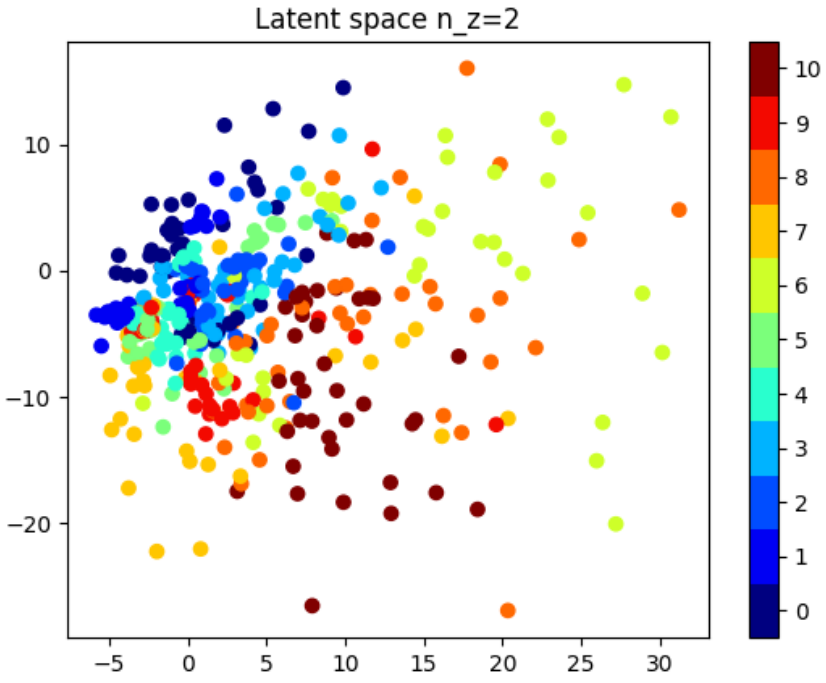


Abbildung C.1: Visualisierung der DLR-VAE Deskriptoren in 2D Laten Space. Die 11-Klassen sind mit verschiedenen Farben bezeichnet[2]

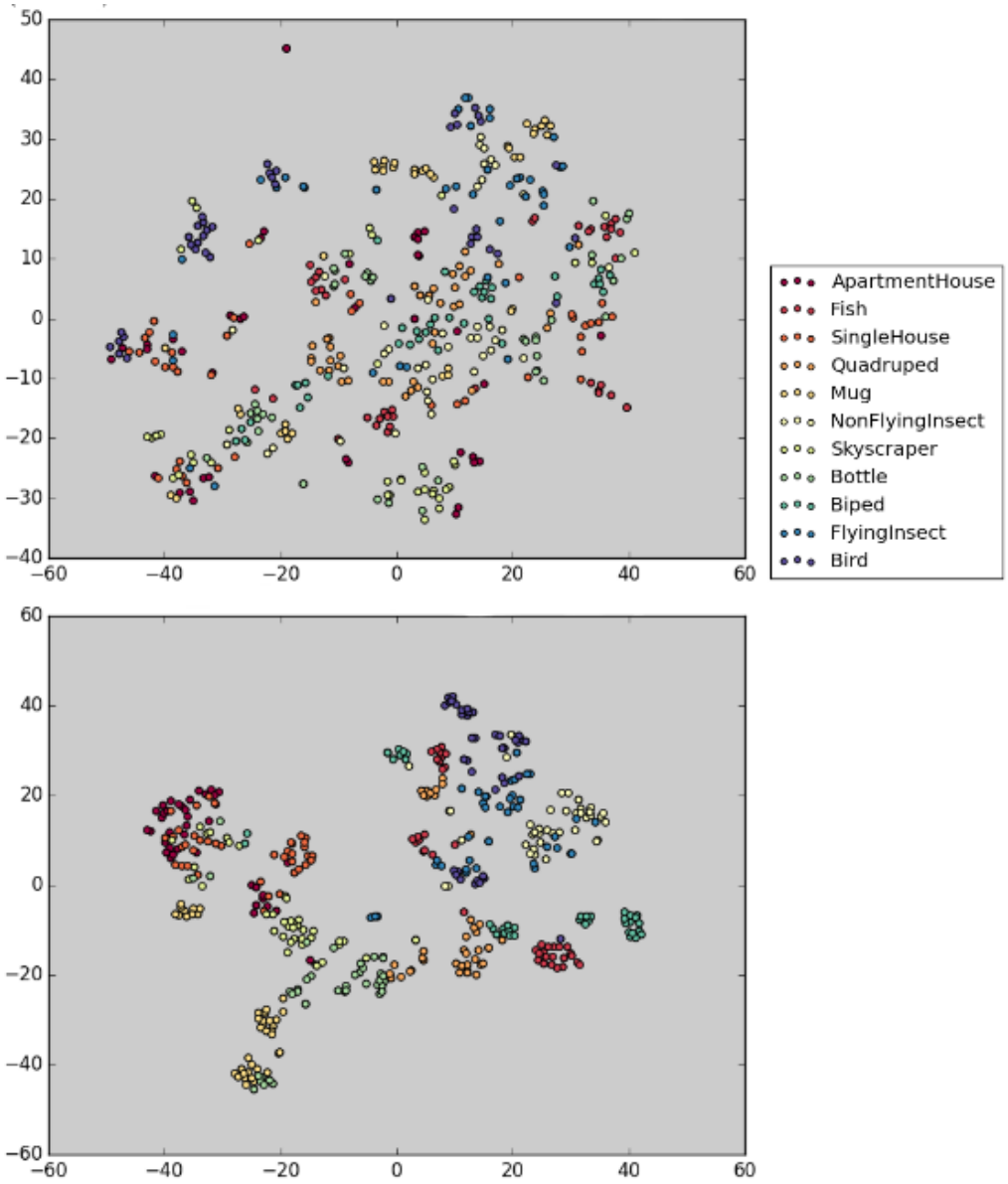


Abbildung C.2: Visualisierung der mit t-SNE der VFH-Features (oben) und der CaffeNet-Features(unter) in 2D Laten Space. Die 11-Klassen sind mit verschiedenen Farben bezeichnet[2]

Literaturverzeichnis

1. Bracci, F., Hillenbrand, U., Marton, Z.C., et al.: On the Use of the Tree Structure of Depth Levels for Comparing 3D Object Views. In: Felsberg, M., Heyden, A., Krüger, N. (eds.) Computer Analysis of Images and Patterns, Lecture Notes in Computer Science, vol. 10424, pp. 251-263. Springer International Publishing(2017), http://dx.doi.org/10.1007/978-3-319-64689-3_21

2. Fabio Bracci, Mo Li, Zoltan-Csaba Marton. Applicability of Deep Learned vs Traditional Features for Depth Based Classification. In: R. Barneva et al. (Eds.) Computational Modeling of Objects Presented in Images. Fundamentals, Methods and Applications. Lecture Notes in Computer Science, Springer, Heidelberg (to appear)

3. Kossyk, I., Marton, Z.S.: Discriminative regularization of the latent mani-fold of variational auto-encoders for semi-supervised recognition. Online (2017), `https://tinyurl.com/y8p3tjle`

4. Rusu, R.B., Cousins, S.: 3D is here: Point Cloud Library (PCL). In: 2011 IEEE International Conference on Robotics and Automation. pp. 1-4. IEEE (May 2011), http://dx.doi.org/10.1109/icra.2011.5980567

5. Aldoma, A., Marton, Z.C., Tombari, F., et al.: Tutorial: Point Cloud Library: Three-Dimensional Object Recognition and 6 DOF Pose Estimation. IEEE Robotics & Automation Magazine 19(3), 80-91 (2012), http://dx.doi.org/10.1109/mra.2012.2206675

6. Rusu, R.B., Bradski, G., Thibaux, R., et al.: Fast 3D recognition and pose using the Viewpoint Feature Histogram. In: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on. pp. 2155-2162. IEEE (Oct 2010), http://dx.doi.org/10.1109/iros.2010.5651280

7. Aldoma, A., Vincze, M., Blodow, N., et al.: CAD-model recognition and 6DOF pose estimation using 3D cues. In: 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops). pp. 585-592. IEEE (Nov 2011), http://dx.doi.org/10.1109/iccvw.2011.6130296

8. Aldoma, A., Tombari, F., Rusu, R., et al.: OUR-CVFH - Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram for Object Recognition and 6DOF Pose Estimation. In: Pinz, A., Pock, T., Bischof, H., et al. (eds.) Pattern Recognition, Lecture Notes in Computer Science, vol. 7476, pp. 113-122. Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-32717-9_12

9. Kingma, D.P.: Variational inference & deep learning: A new synthesis. Intelligent Sensory Information Systems (IVI, FNWI) (2017),

   http://dare.uva.nl/personal/pure/en/publications/variational-inference-deep-learning(8e55e07f-e4be-458f-a929-2f9bc2d169e8) .html

10. Jia, Y., Shelhamer, E., Donahue, J., et al.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22Nd ACM International Conference on Multimedia. pp. 675-678.MM '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2647868.2654889

11. Krizhevsky,A., Sutskever, I., Hinton, G.E.: Imagenet classiffication with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. vol. 25 (2012),

    http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.299.205

12. Pratikakis,I.,Spagnuolo,M.,Theoharis,T.,etal.:SHREC 2010-Shape Retrieval Contest of Range Scans

    http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.361.8068

13. Cover, Thomas, and Peter Hart. "Nearest neighbor pattern classification.İnformation Theory, IEEE Transactions on13.1 (1967): 21-27.

14. Caffe, Deep Learning Framework, [Online]: http://caffe.berkeleyvision.org/

15. TensorFlow, Deep Learning Framework, [Online]: https://www.tensorflow.org/

16. Deng, L.; Yu, D. Deep Learning: Methods and Applications. Foundations and Trends in Signal Processing. (2014)

17. Bengio, Yoshua. Learning Deep Architectures for AI. Foundations and Trends in Machine Learning. 2009, 2 (1): 1–127.

18. Bengio, Y.; Courville, A.; Vincent, P. Representation Learning: A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2013, 35 (8): 1798–1828.

19. Certified Nerd. AI : I'll tell you why Deep Learning is so popular and in demand [Online] https://medium.com/swlh/ill-tell-you-why-deep-learning-is-so-popular-and-in-demand-5aca72628780

20. Li, Fei F. ; Karpathy, Andrej: CS231n: Convolutional Neural Networks for Visual Recognition. University Lecture. 2015. – URL

    http://cs231n.stanford.edu/ . –Zugriffsdatum: 11.03.2016

21. [Online]

    `https://de.wikipedia.org/wiki/Convolutional_Neural_Network`

22. Convolutional Neural Network – Long – Medium- Pooling-Layer,[Online] https://medium.com/@Aj.Cheng/convolutional-neural-network-d9f69e473feb

23. Online: Caffe What restaurant would your computer like to go to? https://www.linkedin.com/pulse/what-restaurant-would-your-computer-like-go-alexander-rakhlin

24. Miriam. ; Under the Hood of the Variational Autoencoder (in Prose and Code): [Online] http://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in.html

25. Volodymyr, T., Eric, C., Artur, L.: A Deep Convolutional Auto-Encoder with Pooling - Unpooling Layers in Caffe. Pin ArXiv published, (2016); [Online]

    https://www.semanticscholar.org/paper/A-Deep-Convolutional-Auto-Encoder-with-Pooling-in-Turchenko-Chalmers/

26. Online: Manifold Learning and Deep Autoencoders in Science;

    https://cloud4scieng.org/manifold-learning-and-deep-autoencoders-in-science/

27. Rusu, R.B., Blodow, N., Beetz, M.: Fast Point Feature Histograms (FPFH) for 3D Registration. In: The IEEE International Conference on Robotics and Automation (ICRA). Kobe, Japan (2009),

    http://files.rbrusu.com/publications/Rusu09ICRA.pdf

28. [Online]

    `http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/technologien-methoden/KI-und-Softcomputing/Neuronales-Netz`

29. Torsello, A., Hidovic, D., Pelillo, M.: Four metrics for efficiently comparing at- tributed trees. In: Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on. vol. 2, pp. 467–470 Vol.2. IEEE (Aug 2004)

30. Online

    http://easyrequirement.blogspot.com/2015/02/struktur-gibt-es-nicht-ohne-die-zeit.html

31. Malkoff, B.: Ëvaluation of the Jonker-Volgenant-Castanon (JVC) assignment algorithm for track association", Proc. SPIE 3068, Signal Processing, Sensor Fusion, and Target Recognition VI, (28 July 1997); doi: 10.1117/12.280801; https://doi.org/10.1117/12.280801

32. Mordecai J. Golin, Bipartite Matching and the Hungarian Method, Course Notes, Hong Kong University of Science and Technology.

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

Selbstständigkeitserklärung

Hiermit versichere ich, Mo Li, geboren am 10.12.1986 in China, dass ich die vorliegende Diplomarbeit zum Thema

Strukturbasierte Multi-View-Erkennung von 3D Objekten mit traditionellen und Deep-Learning Methoden

ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistungen von folgenden Personen erhalten:

Msc.-Ing. Fabio Bracci, Dr.-Ing. Zoltán-Csaba Márton

Weitere Personen waren an der geistigen Herstellung der vorliegenden Diplomarbeit nicht beteiligt. Mir ist bekannt, dass die Nichteinhaltung dieser Erklärung zum nachträglichen Entzug des Diplomabschlusses (Masterabschlusses) führen kann.

Dresden, den 11.06.2018 .....

Unterschrift