Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Martin Weschta

Neuronales Hybridsystem zur Klassifizierung dynamischer dreidimensionaler Gesten

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Martin Weschta

Neuronales Hybridsystem zur Klassifizierung dynamischer dreidimensionaler Gesten

Masterarbeit eingereicht im Rahmen der Masterprüfung

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

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

Eingereicht am: 8. Februar 2017

Martin Weschta

Thema der Arbeit
Neuronales Hybridsystem zur Klassifizierung dynamischer dreidimensionaler Gesten

Stichworte
Neuronale Recurrent Netze, Faltungsnetze, Hybridsysteme, Gestenerkennung, Gebärdensprache, CNTK, OUR-CVFH, Kinect2

Kurzzusammenfassung
Computer Vision bietet bereits unterschiedliche Ansätze zur Erkennung menschlicher Gesten. In dieser Thesis wird dazu eine neuartige Kombination von Ansätzen der Objekt- und Gestenerkennung, unter Verwendung von künstlichen neuronalen Netzen vorgestellt, um dynamische und detailreiche Gesten der Gebärdensprache zu klassifizieren. Um die Komplexität solcher Gesten erfassen zu können werden drei unterschiedliche Gesteninformationen genutzt. Die gewonnenen Daten dieser Teilsysteme werden mittels von einander unabhängigen neuronalen Netzen ausgewertet. Ein finales neuronales Netz fasst diese Auswertungen zusammen und klassifiziert die Bedeutung der Geste. Die Eigenschaften und Ergebnisse dieses Hybridsystems und seiner Teilsysteme werden untersucht und diskutiert.

Martin Weschta

Title of the paper
Neural hybrid system for dynamic threedimensional gesture classification

Keywords
Recurrent neural nets, Convolutional neural nets, Hybridsystems, Gesturerecognition, Sign language, CNTK, OUR-CVFH, Kinect2

Abstract
Computer vision already offers various approaches for recognizing human gestures. This thesis presents a new combination of approaches of object- and gesture-recognition with neural networks, for recognizing daynamic and detailed gestures of Sign Language. To acquire the complexity of such gestures, three different types of information of these gestures are exploited. The gathered data is evaluated in three independent neural networks. A final neural network pools these subsystems and classifies the gesture. The characteristics and results of this hybridsystem and its subsystems will be evalutated and discussed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einführung

Die heutige, fortgeschrittene Technik der Steuerung von Computern, Smartphones und anderen Endgeräten eröffnet viele Anwendungsmöglichkeiten. Besonders im Bereich "Smart-Home" findet die Gestenerkennung bereits große Beachtung. Die dort angewendeten Algorithmen und Ansätze zur detaillierten Erkennung von Hand- und Armbewegungen schreiten von Jahr zu Jahr voran. Solche Algorithmen sind hilfreich bei der Steuerung von Computern, der Umsetzung virtueller Spiele und der Erleichterung des Alltags.

All diese bis heute entwickelten Techniken können auch genutzt werden, um Menschen mit Behinderung von Nutzen zu sein. Gerade taubstumme Menschen sind nicht in der Lage die Funktionen der Spracheingabe oder Spracherkennung zu nutzen. Darüber hinaus gibt es immer noch Menschen mit einer Hör- und Sprachbehinderung, die des Lesens oder Schreibens nicht mächtig sind.

Ein System zur Erkennung der Gebärdensprache wäre für diese Menschen eine große Hilfe. Taubstummen würde ein solches System eine neue Ebene der Kommunikation mit Computern ermöglichen und somit zu einer Erleichterung des Alltags beitragen. Darüber hinaus wäre es durch die Erkennung solcher Gesten möglich, die Barriere der Verständigung mit Menschen, die der Gebärdensprache nicht mächtig sind, zu überwinden.

1.1. Motivation

Für eine erfolgreiche Umsetzung eines solchen Systems werden im Rahmen dieser Arbeit aktuelle Verfahren untersucht und implementiert.

In Ravikiran u. a. (2009) und Silanon und Suvonvorn (2014) wurden Ansätze präsentiert, die statische Gesten der Gebärdensprache untersuchen, um beispielsweise das Alphabet der American Sign Language (ASL) zu klassifizieren. Weiterhin wurden in Holden u. a. (1999) und Parker und Baumback (2009) unter anderem Ideen der allgemeinen Handgesten-Erkennung vorgestellt. Dort werden Gesten durch besondere Merkmale wie beispielsweise durch mathematische Modelle der Hand zu beschreiben.

Weiterhin wurden neben statischen auch dynamische Gesten untersucht. Alshekhali u. a. (2011) berechnet diese Dynamik durch Differenzen im optischen Fluss. Alternativen dazu sind in Gharasuie und Seyedarabi (2013) beziehungsweise Shen u. a. (2011) präsentiert worden, in denen Gesten durch Kantendetektion in Verbindung mit Hidden-Markov-Modellen erkannt werden.

Zwar verwenden alle genannten Ausarbeitungen für die Gestenerkennung keine Handschuhe oder ähnliche Hilfsmittel (Vgl. Davis und Shah (1994), Just und Marcel (2005)), jedoch findet die Gestenerkennung grundsätzlich durch zweidimensionale Bilder und einen einzelnen Merkmalsraum statt.

Dabei verschafft die Verwendung von Kameras und Sensoren mit Tiefeninformation Systemen zur Gestenerkennung einen Vorteil. Ein solches Kamerasystem ist beispielsweise die Kinect von Microsoft. Diese liefert Bilder in mehreren Modalitäten. Beipsielsweise stehen bereits erkannte Positionen fast aller Körperteile und Gelenke (*engl.* "Joints") zu Verfügung. Bereits Wang u. a. (2015) zeigt ein System zur Erkennung von Gesten der chinesischen Gebärdensprache, in dem die Kinect-Kamera für die die Extraktion von Tiefeninformation verwendet wird. Die Kombination von Distanzen zwischen verschiedenen Joints und den Tiefendaten der Hände zeigt ein hohes Potenzial. Die Klassifizierung der Gesten wird dabei mittels Hidden-Markov-Modell durchgeführt.

Zwar ist das Hidden-Markov-Modell in vielen erfolgreichen Systemen implementiert, wird aber nach und nach durch künstliche neuronale Netze und deren höhere Performanz (Waibel u. a. (1988)) abgelöst. Besonders Recurrent Neural Networks (RNN) zeigten dabei für die Erkennung von Trajektorien und Bewegtbildern gute Leistungen. Wu u. a. (2015) stellt ein hybrides neuronales System zur Klassifizierung menschlicher Aktionen in Videos vor. Die Kombination von solchen Recurrent-Netzen mit Convolutional Neural Networks (CNN) erzielt dabei robuste Ergebnisse.

Alle genannten Versuche zeigen interessante und erfolgreiche Ideen für unterschiedliche Anwendungsgebiete. Viele dieser Verfahren können zur Klassifizierung komplexer und dynamischer Gesten beitragen. Die meisten dieser Ansätze zeigen trotz ihrer eindimensionalen Merkmalsräume für den jeweiligen Zweck gute Ergebnisse auf. Eine Kombination solcher Verfahren ist daher für ein System zur Erkennung detaillierter und dynamischer Gesten vielversprechend.

1.2. Zielsetzung

In bereits durchgeführten Versuchen (Weschta (2016a), Weschta (2016b)) wurden die Grundlagen der Gestenerkennung mit Hilfe der Kinect 2-Kamera erarbeitet. Dabei stand vor allem die Untersuchung unterschiedlicher Typen künstlicher neuronaler Netze (NN) im Fokus. In diesen Projekten wurden sowohl statische, als auch dynamische dreidimensionale Gesten der Arme mit hoher Trefferrate klassifiziert. Dabei wurde jedoch die Handstellung nicht genauer betrachtet, was einen wesentlicher Faktor bei Gesten der Gebädensprache darstellt.

Ein System zur Erkennung der Handstellung bei komplexen Gesten setzt hohe Sensibilität für Details voraus. Gesten der Gebärdensprache eignen sich daher besonders, um die Fähigkeiten eines neuronalen Hybridsystems zur Klassifizierung detailreicher und dynamischer Gesten zu untersuchen und zu testen.

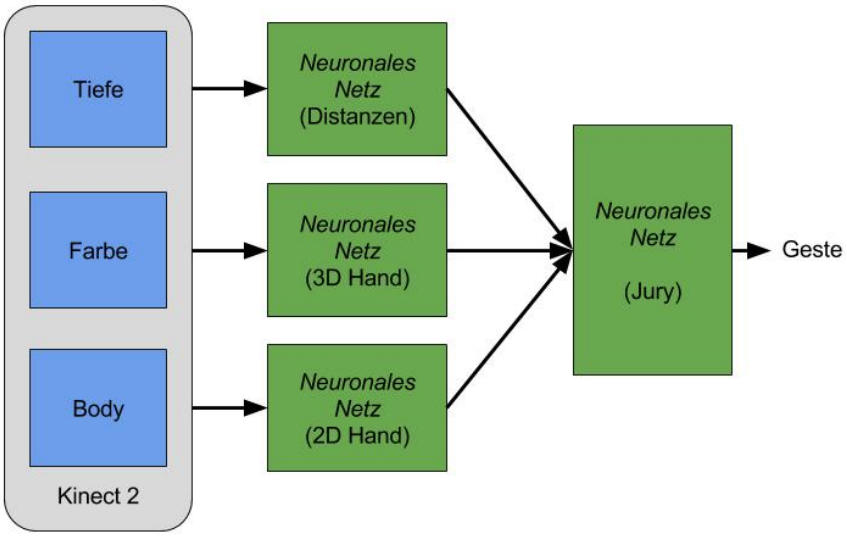


Abbildung 1.1.: Der hybride Ansatz: Drei unabhängige neuronale Netze für die jeweilige Teilaufgabe und ein neuronales Netz als Jury für vorher extrahierte Merkmale.

Angelehnt an Wu u. a. (2015) sollen in dieser Arbeit mehrere neuronale Netze Teilaufgaben erledigen. Die zu erkennenden Gesten werden in drei Bestandteile unterteilt. Diese sind Distanzen mehrerer Joints des Oberkörpers zueinander (Vgl. Wang u. a. (2015)), die dreidimensionale Oberflächenstruktur der Hände und zweidimensionale Merkmale der Hände im Farbbild. Eine detailreiche Geste der Gebärdensprache kann durch diese drei Teilbereiche vollständig identifiziert werden.

Jeder dieser Teilbereiche besitzt eine individuelle Vorverarbeitung. Anschließend extrahiert jeder Teilbereich die Merkmale durch ein eigenständiges neuronales Netz. Die so gesammelten Merkmale werden zusammengefügt und durch ein finales neuronales Netz klassifiziert, welches als Jury-System für die Informationen der Teilsysteme dient. Wie sehr ein solches Vorgehen für detaillierte Handgesten geeignet ist, soll anhand dieser Arbeit evaluiert werden.

Der in Abbildung 1.1 dargestellte hybride Ansatz zeigt, dass drei unterschiedliche Informationsquellen der Kinect für diese Teilsysteme genutzt werden. Diese sind die Bilder (*engl.* "Frames") der Farbkamera und des Tiefensensors, sowie Bilder des Body-Trackings. Darüber hinaus werden die Tiefen- und Bodyframes zusätzlich für Segmentierung der Hände vom Hintergrund verwendet.

Die Verarbeitung der Daten und die Klassifizierung erfolgt in drei Phasen:

- Zunächst werden Daten der Kinect gesammelt und für drei Teilsysteme individuell vorverarbeitet.
- Anschließend folgt die Klassifizierung dieser individuellen Merkmale im jeweiligen Teilsystemen. Die Ergebnisse der Klassifizierung aller Teilsysteme werden gesammelt.
- Abschließend erfolgt die Klassifizierung der in den Teilsystemen gesammelten Ergebnisse in einem finalen neuronalen Netz.

Die neuronalen Netze der Teilsysteme unterscheiden sich durch die Art des Netzes und der Anforderung der Vorverarbeitung. Aus der Kombination dieser unterschiedlichen Ansätze resultiert als gesamtes Konzept ein hybrides neuronales System. Dieses soll Schwächen in den einzelnen Teilsystemen ausgleichen. Nötige Vorverarbeitungsschritte, die Systeme selbst und deren Kombination werden in dieser Arbeit beschrieben. Weiterhin werden die Teilsysteme auf ihre einzelnen Leistungen geprüft und mit dem Resultat des Gesamtsystems verglichen. Die Ergebnisse sowie potentielle Probleme der einzelnen Verfahren werden untersucht und evaluiert.

1.3. Aufbau der Arbeit

Diese Arbeit ist in mehrere Kapitel gegliedert. Folgende Auflistung gibt einen Überblick über die Arbeit und die darin enthaltenen Kapitel.

Kapitel 2 führt zunächst in die Grundlagen des Systems ein. Das verwendete Framework für die Handhabung neuronaler Netze, die Kinect 2 Kamera und der Algorithmus zur

Vorverarbeitung der dreidimensionalen Daten sowie das dabei verwendete Framework werden hier genauer erklärt.

Kapitel 3 gibt einen Einblick in künstliche neuronale Netze und deren grundlegende Funktionsweise. Weiterhin wird auf Besonderheiten der Netze eingegangen, die in dieser Arbeit verwendet werden.

Kapitel 4 erläutert die in den Teilsystemen nötige Vorverarbeitung der Daten. Es wird kurz auf Gebärdensprache eingegangen, die als Grundlage für die getesteten Gesten dient. Besondere Verfahren zur Vorbereitung von Daten und Merkmalen für den hybriden Ansatz werden hier erklärt.

Kapitel 5 stellt das neuronale Hybridsystem detailliert vor. Neben der Erläuterung des Gesamtsystems, wird auch auf die Einzelkomponenten eingegangen. Deren Netzparameter und besonderen Eigenschaften sowie die Verarbeitung der Merkmale liegen hier im Fokus.

Kapitel 6 präsentiert die Tests für die Teilbereiche und das Gesamtsystem. Es wird sowohl auf das Trainings- und Testset, als auch auf die erzielten Resultate und das Trainingsverhalten der verwendeten neuronalen Netze eingegangen.

Kapitel 7 zeigt eine Diskussion über die erzielten Resultate und stellt einen Vergleich der Leistungen der Einzelkomponenten zueinander auf. Problemquellen werden aufgezeigt und die Performanz des Systems untersucht.

Kapitel 8 zieht abschließend ein Fazit und gibt einen Ausblick auf mögliche Weiterentwicklungen eines solchen neuronalen hybriden Ansatzes.

2. Grundlagen

Die Programmierwerkzeuge zur Erstellung und zum trainieren neuronaler Netze sind sehr unterschiedlich. Umfang und Handhabbarkeit spielen für eine erfolgreiche Klassifizierung eine große Rolle. Dieses Kapitel zeigt das für das Hybridsystem verwendete Framework “CNTK”.

Weiterhin setzt die Klassifizierung dreidimensionaler Gesten eine Kamera voraus, die Tiefenbilder bereitstellt. Die Verwertung dieser Tiefendaten verlangt zusätzlich eine entsprechende Konvertierung in ein für die Klassifizierung sinnvolles Format. Es wird daher in diesem Kapitel genauer auf das Kamerasystem Kinect und den Algorithmus für die Erstellung von Histogrammen zur Verarbeitung der Tiefendaten eingegangen.

2.1. CNTK

Für die Erstellung der Netze des Hybridsystems wurde das Microsoft Cognitive Toolkit (CNTK) von Microsoft Research verwendet (Microsoft (2016a) - vorher “Computational Network ToolKit”). CNTK bietet die Möglichkeit durch Skripte individuell neuronale Netze aller Art erstellen und konfigurieren zu können. Dazu wurde die Skriptsprache “BrainScript” entwickelt. Zur Veranschaulichung ist das BrainScript einer Netzkonfiguration in Anhang A.2 zu sehen.

CNTK bietet durch einfache Definition im BrainScript die Möglichkeit, das Training der neuronalen Netze auf dem Grafikprozessor durchzuführen. Dazu wird das Toolkit CUDA des Herstellers Nvidia eingebunden, um die GPU für Berechnungen zu verwenden. Bei Definition des Parameters “deviceID” (siehe Anhang A.2 - Zeile 3) kann die entsprechende GPU oder CPU ausgewählt werden. Ist dieser Parameter auf “auto” gesetzt, wird automatisch die GPU mit der besten Rechenleistung des Systems ausgewählt. Ist keine kompatible GPU vorhanden wird automatisch die CPU verwendet.

Zum Einlesen der Daten werden die in CNTK mitgelieferten “TextFormatReader” für textbasierte Daten und “ImageReader” für Bilddaten verwendet. Der TextFormatReader ist rein textbasiert und kann individuell konfiguriert werden. Das in dieser Arbeit verwendete Format beinhaltet Sequenznummern, Klassen-IDs und die zugehörigen Merkmale. Wie in folgendem

Beispiel zu sehen sind diese Daten durch die Separatoren “|L” und “|F” getrennt und indizieren nach der Sequenznummer das Label (L) und die Features (F). Diese Separatoren sind durch das BrainScript des jeweiligen Netzes definiert (siehe Anhang A.2).

```
000 |L 1 0 0 0 |F 0 123 0 22 44 44 290 22 22 22 33 0
000 |L 1 0 0 0 |F 0 0 0 0 0 0 1 0 0 0 0 0
001 |L 0 1 0 0 |F 34 56 0 0 0 45 1312 0 0 0 0 0
001 |L 0 1 0 0 |F 45 45 0 0 0 12 335 0 0 0 0 0
002 |L 0 0 1 0 |F 0 0 0 0 0 0 22 0 0 0 0 0
```

Der ImageReader ist für das Einlesen von Bilddaten zuständig. Hierfür werden im BrainScript die Höhe, Breite und Farbtiefe der Bilder und zusätzlich der Pfad zu einer Mapping-Datei angegeben. Diese Mapping-Datei enthält alle Bilder die dem Netz als Input zugeführt werden sollen.

In CNTK wird das Training auf der Konsole durchgeführt. Während des Trainings werden dort regelmäßig relevante Informationen ausgegeben. Beispielhaft ist ein Auszug einer solchen Ausgabe in Anhang A.3 angefügt. Für das Training und die Evaluation des Netzes wird lediglich die Netzkonfigurationsdatei, das heißt das BrainScript im Konsolen-Befehl angegeben.

2.2. Kinect 2

Für die Aufnahme der Bilder und Daten wurde die Kinect der zweiten Generation (Kinect 2) von Microsoft verwendet. Diese bietet durch ihre Fähigkeit gleichzeitig unterschiedliche Bild-Typen (*engl.* Frames) zu liefern mehrere Vorteile gegenüber anderen Kameras. Für die Bildaufnahme bietet sie

- eine Farbkamera für zweidimensionale Bilder im BGRA-Farbraum,
- einen Infrarot-Sensor für die Aufnahme licht-unabhängiger Bilder und
- einen Tiefensensor für dreidimensionale Daten.

Weiterhin liefert Microsoft mit der Kinect ein Software Developement Kit (SDK). Dieses ermöglicht, neben dem Zugriff auf die oben genannten Kameras und Sensoren, den Zugriff auf bereits erkannte Gliedmaßen und bietet Methoden zur Verarbeitung der Pixeldaten und Koordinaten. Die Frames der Farbkamera besitzen eine Auflösung von 1920x1080 Pixel, die des Tiefensensors 512x424 Pixel. Beide liefern eine Bildwiederholrate von 30 Frames pro Sekunde, was einer Zeit von 33 Millisekunden pro Frame entspricht.

Da die Kameras und Sensoren unterschiedliche Auflösungen besitzen und in unterschiedlichen Koordinatensystemen arbeiten, beinhaltet das Kinect-SDK einen “CoordinateMapper”. Dieser ermöglicht die Umwandlung der Koordinaten zwischen den unterschiedlichen Frametypen. So lassen sich beispielsweise Positionen der Joints in das Farbbild übertragen.

2.2.1. Tiefenbild

Die Kinect 2 verwendet für die Aufnahme der dreidimensionalen Tiefendaten eine “Time-of-Flight”-Kamera (TOF). Das entspricht einer Veränderung zur ersten Kinect, bei der die Tiefeninformation auf trigonometrischer Berechnung basiert. Eine TOF-Kamera sendet Lichtsignale aus und misst mit einem Sensor, wie lange das Licht braucht, um zurückzukehren. Mit solchen Messungen kann die Kinect 2 von Objekten und deren Umwelt reflektiertes Licht besser differenzieren und so die Umrisse dieser Objekte genauer erkennen (Microsoft (2016c)).

Abbildung 2.1.: Das Tiefenbild der Kinect 2, bereitgestellt von der Kinect SDK.

Das Tiefenbild liefert für jedes Pixel $p(x, y)$ einen Tiefenwert als ganzzahligen Wert ohne Vorzeichen, der die Entfernung des Pixels zur Kamera in Millimetern ausdrückt. Die maximale zuverlässige Tiefe der Kinect 2 sind acht Meter, was bedeutet, dass $p(x, y)$ im Wertebereich $[0; 8000]$ liegt. Zur Anzeige im Frame kann durch einen statischen Divisor $8000/256$ daraus ein Grauwert im Wertebereich $[0; 255]$ berechnet werden. Die Koordinaten beginnen sowohl bei den Farb- als auch bei den Tiefenbildern mit dem Nullpunkt oben links im Bild.

2.2.2. Bodyframe

Das Kinect-SDK stellt ein Body-Joint-Tracking zur Verfügung. Damit können die Koordinaten der in Abbildung 2.2 gezeigten Körperteile und Gelenke über das SDK ausgelesen werden. Für die Realisierung dieses Projekts ist dieses Feature essentiell. Vor allem die Joints der Schultern, Ellbogen und Hände sind von Nöten, da sie für spätere Berechnungen und Segmentierung verwendet werden.

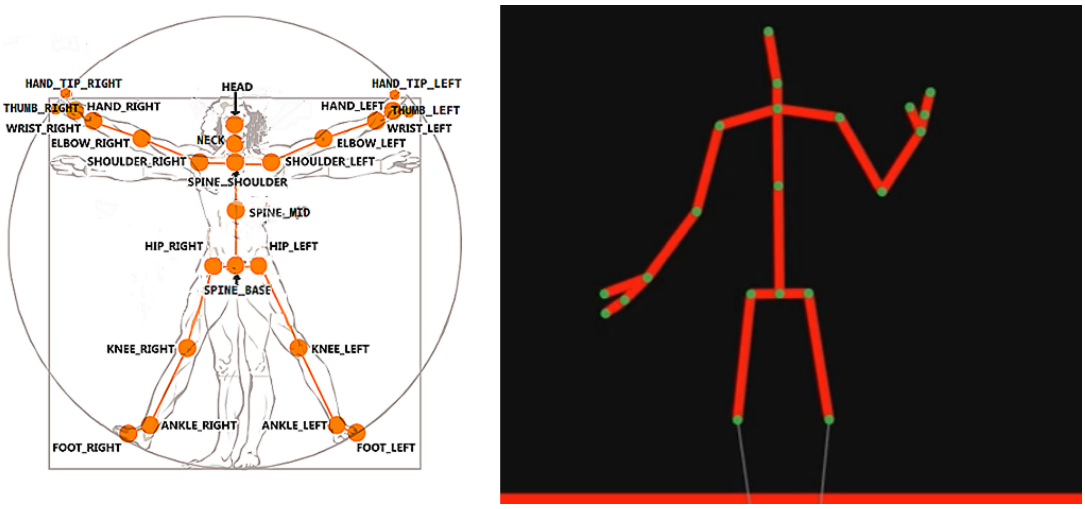


Abbildung 2.2.: Links: Die Joints des Körpers die das Kinect-SDK liefert Microsoft (2016b). Rechts: Ein Beispiel des SDK zur Veranschaulichung des Bodyframes.

Der Bodyframe wird durch das Tiefenbild gewonnen. In Shotton u. a. (2011) wird gezeigt, wie sogenannte “Bodyparts” aus dem Tiefenbild entstehen. Die Koordinaten der Joints werden durch die Berechnung von Differenzen der Tiefeninformation einzelner Pixel zu ihrer Umgebung gewonnen. Für die Entscheidung, ob es sich bei diesem Pixelbereich um einen Joint handelt kommen trainierte, heuristische Verfahren zur Anwendung (MacCormick (2016)). Nach dieser Entscheidung wird die dreidimensionale Koordinate des betrachteten Pixels als Joint bereitgestellt.

Das SDK stellt die Position aller Joints in drei Dimensionen zu Verfügung. Im Gegensatz zu den Farb- und Tiefenbildern liegt hier der Ursprung des Koordinatensystems direkt am Sensor der Kinect und somit im Zentrum des Bildes. Der Tiefenkoordinate z drückt die Entfernung zur Kamera aus. Alle Koordinaten sind dabei Fließkommazahlen und in Metern angegeben.

2.3. OUR-CVFH

Für die Weiterverarbeitung der Tiefendaten der Kinect werden diese in ein Histogramm konvertiert. Eine Möglichkeit dafür ist der Weg über Punktwolken (*engl.* "Point Clouds"). In dieser Arbeit kommt das in Aldoma u. a. (2012) vorgestellte "Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram" (OUR-CVFH) zur Anwendung.

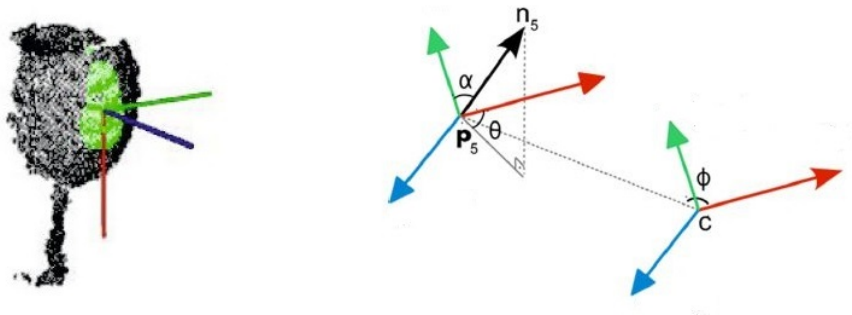


Abbildung 2.3.: Nach Aldoma u. a. (2012): Links die Darstellung einer Referenzfläche mit zugehörigem Koordinatensystem. Rechts die Beschreibung der Objektgeometrie durch die Winkel α, ϕ und θ (Böhle (2016))

.

Die Idee von OUR-CVFH besteht in der Definition einer Referenzfläche (*engl.* "reference frame"), die in Abbildung 2.3 beispielhaft grün gekennzeichnet ist. OUR-CVFH ist eine Weiterentwicklung des Verfahrens zu Erstellung von Viewpoint Feature Histograms (VFH), das in Rusu u. a. (2010) entwickelt wurde. Die Referenzfläche wird genutzt, um fünf identifizierende Eigenschaften des Objekts zu berechnen. Diese Eigenschaften sind

- drei Winkel zur Beschreibung der Objektgeometrie,
- der Blickwinkel auf das Objekt aus Sicht der aufnehmenden Kamera und
- die Verteilung der Wolkenpunkte auf die Oktanten eines Koordinatensystems.

Zunächst wird der Schwerpunkt c der Referenzfläche ermittelt, indem der Mittelwert der Normalen n aller Punkte dieser Fläche berechnet wird. Von jedem Punkt innerhalb der Referenzfläche werden drei verschiedene Winkel (α; ϕ; θ) in Bezug auf den Schwerpunkt c berechnet. Mithilfe dieser Winkel ist es möglich die Objektgeometrie zu beschreiben. Abbildung 2.3 zeigt beispielhaft die Beschreibung der Winkel zwischen einem Punkt $p5$ und dem Schwerpunkt c der Referenzfläche. Der Schwerpunkt c bildet den Ursprung des zugrunde liegenden Koordinatensystems. Die drei Winkel (α; ϕ; θ) berechnen sich durch die Winkeldifferenzen dieses Koordinatensystems und der Normalen des Punktes $p5$.

Für die Beschreibung des Blickwinkels wird ebenfalls eine Winkeldifferenz berechnet. Dies geschieht analog zum vorher beschriebenen Vorgehen. Allerdings wird hier lediglich der α-Winkel zwischen dem angegebenen Blickpunkt auf die Punktwolke und der Normalen berechnet.

Als abschließende Eigenschaft wird die räumliche Verteilung der Punkte angegeben. Ausgangspunkt ist dabei das Koordinatensystem des Referenzsystems. Hierfür werden alle Oktanten des Koordinatensystems des Schwerpunktes c genutzt.

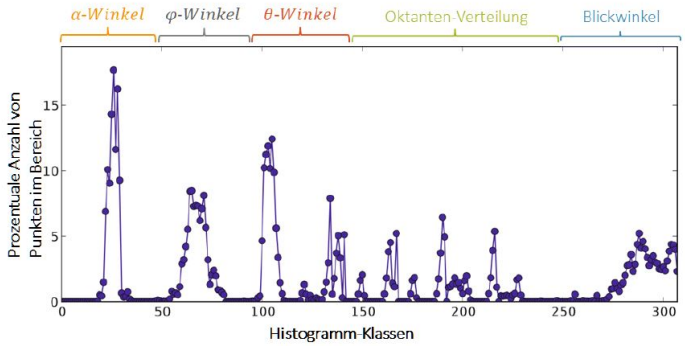


Abbildung 2.4.: Aus Böhle (2016): Beispiel des Histogramms mit Ausprägung der ermittelten Eigenschaften.

Alle ermittelten Eigenschaften werden als Ergebnis in einem Histogramm kodiert (siehe Abbildung 2.4). Die drei Winkel α-, ϕ- und θ nehmen jeweils 45 Histogramm-Klassen ein. Der Blickwinkel wird in 64 und die räumliche Verteilung in 13 Klassen je Oktant kodiert. Insgesamt besteht das OUR-CVFH daher aus 303 Klassen. Dies entspricht 5 Klassen weniger gegenüber dem Vorgänger-Verfahren VFH, bei dem das Histogramm 308 Klassen beinhaltet.

2.4. Point Cloud Library

Zur Implementierung des OUR-CVFH wurde die Point Cloud Library (PCL) verwendet. Diese ist ein in C++ umgesetztes Framework zur Verarbeitung von Bildern und Punktwolken. Darin ist eine Funktion zur Berechnung von Viewpoint Histogrammen enthalten. Die Konvertierung einer Punktwolke in ein OUR-CVFH ist in Anhang A.4 veranschaulicht.

Die in folgendem Kurzbeispiel gezeigte Instanz “ourcvfh” des Objekts “pcl::OURCVFHEstimation” bietet mehrere Funktionen. Unter anderem sind diese die Bestimmung der Punktwolke, der durch die PCL berechneten Normalen und der Berechnung des Histogramms selbst. Der Rück-

gabewert “descriptor” beinhaltet das Histogramm und ist vom Typ “pcl::VFHSignature308”. Die Größe des Histogramms ist dabei 308, anstatt der 303 Merkmale von OUR-CVFH. Dies dient zur Absicherung, sollten von OUR-CVFH keine Cluster in den Tiefeninformationen gefunden werden. In diesem Fall wird auf den VFH-Algorithmus zurückgegriffen, um in jedem Fall eine Beschreibung des Objekts zu erhalten.

```
1 pcl::OURCVFHEstimation<pcl::PointXYZ, pcl::Normal, pcl::VFHSignature308> ourcvfh;
2 ourcvfh.setInputCloud(object);
3 ourcvfh.setInputNormals(normals);
4 ourcvfh.compute(*descriptor);
```

3. Neuronale Netze

Diese Ausarbeitung befasst sich mit einem Zusammenschluss unterschiedlicher, künstlicher, neuronaler Netze (NN). Um diese grundlegend zu verstehen, wird in diesem Kapitel auf die Funktionsweise neuronaler Netze eingegangen. Weiterhin werden die in Kapitel 4 verwendeten Netztypen und deren Besonderheiten vorgestellt.

3.1. Multi-Layer Perzeptron

Künstliche neuronale Netze bestehen, nach dem Vorbild des menschlichen neuronalen Systems, aus einer Vielzahl untereinander vernetzten Neuronen. Ein künstliches neuronales Netz besteht aus einem ein- oder mehrschichtigen Gebilde von Neuronen. Jede Schicht (*engl.* "Layer") gibt die verarbeiteten Informationen an die nachfolgende weiter. Solche Gebilde ergeben ein Multi-Layer Perzeptron (MLP).

3.1.1. Neuronen

Ein einzelnes Neuron ist im wesentlichen eine Funktion, die sich Werte eines Eingabevektors zunutze macht, um eine vorher "antrainierte" Ausgabe zu erzeugen. Ein solches Neuron ist in der Regel mit jedem Neuron des vorherigen Layers verbunden und erhält als Eingabe die Werte aller Neuronen des vorherigen Layers. Ein solcher Verbund ist in Abbildung 3.1 dargestellt. Man unterscheidet bei den Schichten zwischen Eingabe-, Hidden- und Ausgabe-Layern.

Das Neuron besteht aus mehreren Parametern. Jedes Element des Eingabevektors wird mit einer Gewichtung multipliziert. Weiterhin wird ein Bias-Wert zum Ergebnis der Gewichtungen hinzu addiert. Zweck des Bias ist es, die Funktion des Neurons variabler zu gestalten, um dem gewünschten Output näher zu kommen und die Performanz des Netzes zu steigern. Sowohl die Gewichtungen als auch der Bias stellen die trainierbaren Elemente des Neurons dar.

Das Ergebnis aus Multiplikation der Gewichtungen W_n mit dem Input $i_1, \ldots, i_m$ und Addition des Bias $b$dient als Eingabewert für eine Aktivierungsfunktion f. Diese Funktion bestimmt,

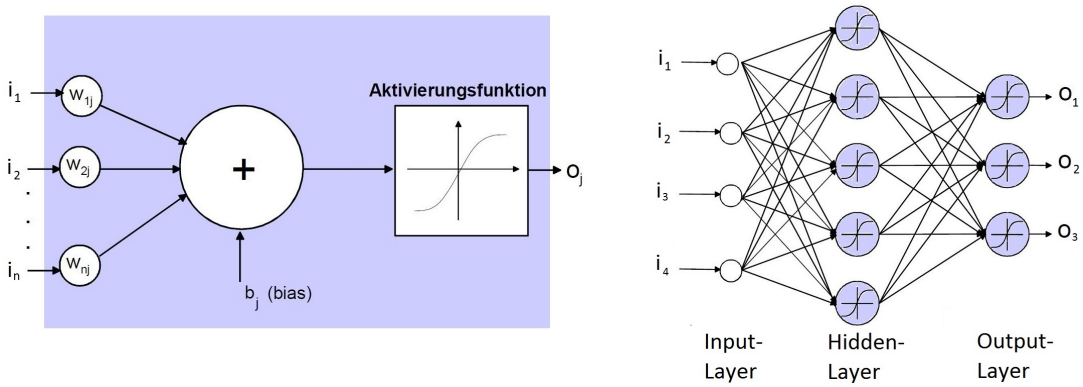


Abbildung 3.1.: Aus Meisel (2012). Links: Ein Neuron mit Eingangswerten $i_1, i_2, ...i_n$, der Gewichtungsmatrix W_{nj} und dem Bias b_j. Rechts der Verbund mehrerer Neuronen zu einem MLP mit einem Hidden-Layer.

wie sehr dieses Neuron Einfluss auf die Ausgabe des Layers hat. Der Output o eines Neurons errechnet sich daher durch die Summe aller gewichteten Input-Werte, der Addition des Bias und der Ausgabe durch die Aktivierungsfunktion f.

$$o = f(\sum_{n=1}^{m}(i_n * W_n) + b) \tag{3.1}$$

Da ein Neuron nur Teil eines Layers ist, wird in diesem eine Gewichtungsmatrix W_{nj} verwendet. Dabei entspricht n der Größe des Inputvektors und j der Anzahl der Neuronen des aktuellen Layers. Im BrainScript in CNTK wird ein solcher Layer durch die in Gleichung 3.2 gezeigte Formel ausgedrückt. Dabei steht $\circ$ für die elementweise Multiplikation der Gewichtungsmatrix mit dem Input. CNTK arbeitet grundsätzlich mit Vektoren und Matrizen, was eine solch vereinfachte Darstellung eines Layers ermöglicht.

$$o_j = f(i_n \circ W_{nj} + b_j) \tag{3.2}$$

Als Aktivierungsfunktion f wurden ursprünglich Sigmoid-Funktionen, wie beispielsweise die logistische Funktion verwendet. Eine Sigmoid-Funktion hat einen S-förmigen Verlauf und ergibt Werte im Bereich $[0; 1]$. Die logistische Funktion (dargestellt in Gleichung 3.3) ist eine spezielle Sigmoid-Funktion. Im wesentlichen ist sie eine skalierte und verschobene Version des Tangens Hyperbolicus und hat entsprechende Symmetrien. Daher wird als Erweiterung des

Werteraums der Aktivierungsfunktion oft der Tangens Hyperbolicus selbst verwendet. Damit kann das Neuron Werte im Bereich $[-1; 1]$ an die nächste Schicht weitergeben.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

Bei der Verwendung einer Sigmoid-Funktion als Aktivierungsfunktion müssen die Input-Werte normiert werden, um schneller zu einem erfolgreichen Ergebnis zu gelangen (LeCun u. a. (1998a)). Weiterhin haben Sigmoid-Funktionen den großen Nachteil, große Werte im positiven, als auch im negativen Bereich nicht mehr erfassen zu können. Immer häufiger findet daher die Funktion "Rectified Linear Unit" (ReLU) Anwendung als Aktivierungsfunktion.

$$f(x) = max(0, x) \tag{3.4}$$

Eine ReLU (Gleichung 3.4) bietet im Gegensatz zu Sigmoid-Funktionen die Möglichkeit, positive Werte auch über 1 hinaus abzubilden. Dies macht eine Normierung der Input-Werte eines MLPs überflüssig, solange sie im positiven Zahlenraum liegen. Die Gefahr, dass besonders hohe Werte wie durch eine Sigmoid-Funktion nicht greifbar sind, ist dadurch ebenfalls bereinigt. Ein Vergleich dieser Aktivierungsfunktionen ist in Abbildung 3.2 graphisch dargestellt.

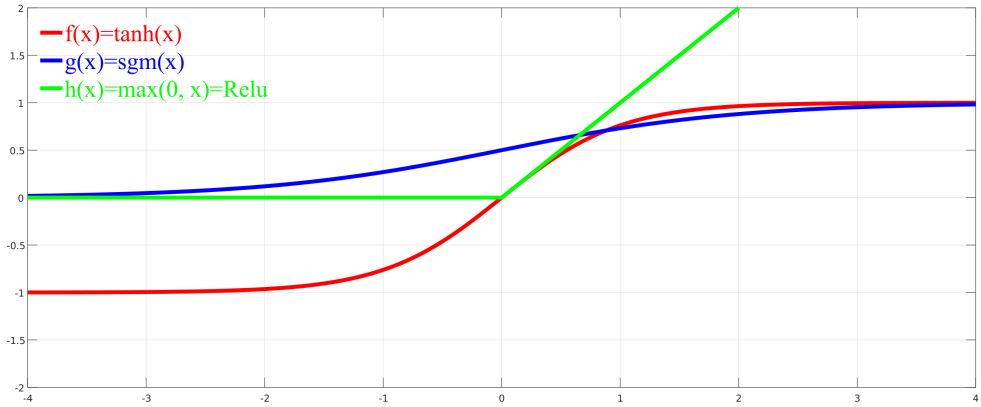


Abbildung 3.2.: Der Vergleich unterschiedlicher Aktivierungsfunktionen für neuronale Netze.

3.1.2. Netzaufbau

Für den Aufbau eines MLP spielt nicht nur die Anzahl der Neuronen eine Rolle, sondern auch auf die Anzahl der Layer muss geachtet werden. Der Aufbau eines Netzes kann allerdings

nicht verallgemeinert festgelegt werden, da sich die Anforderungen von System zu System unterscheiden.

Grundsätzlich gilt, dass eine höhere Anzahl an Neuronen auch Probleme mit steigender Komplexität lösen kann. Allerdings gibt es einen Grenzwert für diese Anzahl, ab der das Netz übertrainiert wird. Eine solche Überanpassung (*engl.* "overfitting") bewirkt, dass die zu erkennenden Klassen nicht mehr approximiert werden können. In einem solchen Fall werden nur noch Sonderfälle der Klassen antrainiert, was zu einem falschen Ergebnis führt. Die Zahl der Neuronen in den Hidden-Layern sollten die Merkmale des vorherigen Layers abdecken und genug Information an den nächsten Layer weitergeben können, ohne dass diese Anzahl zu overfitting führt. Es muss dabei ein Ausgleich zwischen ausreichender und nicht überladener Anzahl an Neuronen gefunden werden.

Weiterhin ist in Cybenko (1989) das "Universal-Approximation-Theorem" (UAT) festgelegt. Nach dem UAT gilt, dass ein MLP mit einem Hidden Layer grundsätzlich jede Funktion approximieren kann. Voraussetzung dafür ist allerdings eine ausreichende Anzahl an Neuronen im Hidden-Layer. Das UAT sagt jedoch nichts darüber aus, ob ein MLP mit einem Hidden-Layer besser lernt oder generalisiert, als andere. Oft werden mindestens zwei Hidden-Layer verwendet, um eine verbesserte Approximation gegenüber Netzen mit einem Hidden-Layer zu ermöglichen. Mehr Hidden-Layer führen jedoch auch nicht zwangsläufig zu besseren Resultaten oder beschleunigtem Lernverhalten.

3.1.3. Lernprozess

Wie auch das menschliche neuronale System wird das MLP "trainiert". Dazu wird beim Training bestimmt, bei welchem Eingabesignal eine bestimmte Ausgabe erzeugt werden soll. Dazu wird ein Trainingsset verwendet, in dem viele Paare von Merkmalen (*engl.* "Features") und zugehörigem, gewünschtem Output (*engl.* "Label") enthalten sind. Die Neuronen beziehungsweise die Layer des Netzes adaptieren ihre Gewichtungsmatrix und ihren Bias-Wert so, dass sich der erzeugte Output möglichst an die gewünschte Ausgabe (d.h. das Label) annähert. Die Anpassung der Gewichtswerte geschieht mit dem Backpropagation-Algorithmus.

Zunächst werden dabei alle Gewichtungen mit Zufallswerten belegt. Der anschließend angewendete Backpropagation-Algorithmus besteht aus folgenden Phasen:

1. Zunächst wird ein zufälliger Feature-Vektor aus dem Trainingsset gewählt und eine Ausgabe durch das Netz erzeugt.

2. Diese Ausgabe wird nun mit dem erwarteten bzw. gewünschten Label verglichen und der Fehler mit Fehlerquadratsumme (Gleichung 3.5) berechnet.
3. Dieser Fehlerwert wird durch das Netz zurück gegeben, d.h. zurück propagiert. Dabei werden Gewichtungen der Neuronen modifiziert.

Diese Schritte werden solange wiederholt, bis ein vorher festgelegter Fehlerwert unterschritten wird. Die Fehler-Funktion E ist in der oben genannten Phase die Fehlerquadratsumme:

$$E = \sum (g - o)^2 \tag{3.5}$$

Die Modifikation, bzw. Anpassung der Gewichtungen w wird durch einen Gradienten $\vec{\bigtriangledown} E$ ausgedrückt. Der Gradient ergibt sich durch die partielle Ableitung der Fehlerfunktion δE. Beispielsweise berechnet sich der Gradient $\vec{\bigtriangledown} E$ eines zweidimensionalen Vektors und dessen Gewichtungen w durch

$$\vec{\bigtriangledown} E(w_1, w_2) = \left(\frac{\delta E}{\delta w_1}, \frac{\delta E}{\delta w_2} \right)^T \tag{3.6}$$

Liegt das betrachtete Neuron in einem Hidden-Layer, so wird die Gewichtung abhängig vom berechneten Fehler der nachfolgenden Neuronen geändert. Da die Verkettung der Layer eine Verschachtelung von Funktionen darstellt, kann der Gradient über mehrere Layer hinweg durch die Kettenregel der Differentialrechnung bestimmt werden.

Die neuen Gewichtungen w_{t+1} ergeben sich, wie in Gleichung 3.7 zu sehen, durch Subtraktion des Gradienten $\vec{\bigtriangledown} E$ und einem Schrittfaktor μ mit der bisherigen Gewichtung w_t. Der vor dem Training festgelegte Schrittfaktor μ wird auch als Lernrate bezeichnet und bestimmt die Stärke mit der die Gewichtsanpassung durchgeführt wird.

$$w_{t+1} = w_t - \mu \vec{\bigtriangledown} E \tag{3.7}$$

Die Anpassung der Gewichtungen erwirkt bei einem erneuten Anlegen eines zufälligen Merkmals-Vektor aus dem Trainingsset in der Regel eine Annäherung an die gewünschte Ausgabe. Neben diesem inkrementellen Vorgehen existiert das Batch-Verfahren. Der Unterschied besteht dabei, dass das Batch-Verfahren zuerst den Fehler aller Trainingsmuster bestimmt. Erst danach wird die Modifikation der Gewichtungen vorgenommen.

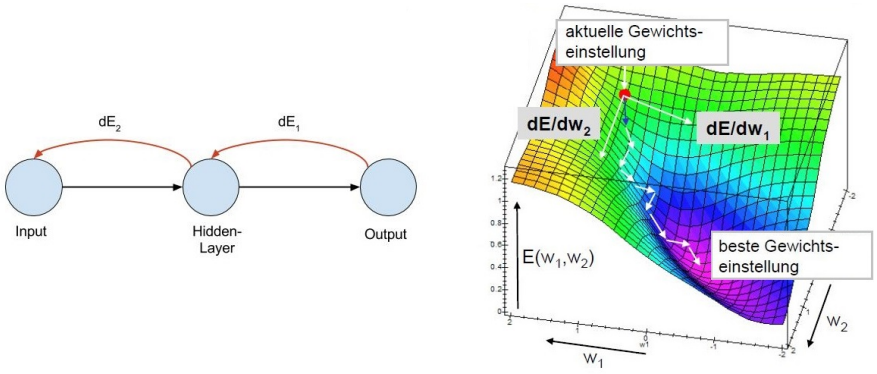


Abbildung 3.3.: Links: die Verkettung der Backpropagation. Rechts: Veranschaulichung des Fehlergebirges mit beispielhaftem Verlauf des Gradienten während des Trainings (Meisel (2012)).

Bei jeder Durchführung des Backpropagation-Algorithmus wird der Gradient dahingehend verändert, dass er sich im Optimalfall dem Minimum der Fehlerfunktion annähert. Die Fehlerfunktion wird oft auch als "Fehlergebirge" bezeichnet.

Dieses Vorgehen birgt einige Probleme. Beispielsweise kann der Gradient in Plateaus stagnieren oder bei steilen Abstiegen in Fehlertälern oszillieren. Solche Fälle führen häufig zu einem stark verlangsamten Lernprozess oder sogar verschlechterten Ergebnissen. Abhilfe schafft hier ein zusätzlicher Momentum-Term. Dieser berücksichtigt die vorherige Schrittweite mit einem zusätzlichen Faktor (siehe Gleichung 3.10). Der Vorteil des Momentums ist in Abbildung 3.4 graphisch veranschaulicht.

Gleichung 3.5 zeigt die Fehlerquadratsumme für die Durchführung der Backpropagation. Eine Alternative für eine solche Fehlerfunktion ist die kategorische Kreuzentropie (*engl.* "Cross-Entropy"). Der Fehlerwert CE berechnet sich hier durch den Fehler F zweier Wahrscheinlichkeitsverteilungen y_i und p_i, die im Kontext neuronaler Netze den erzeugten Output z und das Label des Trainingssets darstellen.

$$CE = F_y{-}log(p) = -\sum_i y_i * log(p_i), \quad \text{mit } p = Softmax(z) \tag{3.8}$$

Der in Gleichung 3.8 dargestellte Fehler beinhaltet die Kreuzentropie in Kombination mit der Softmax-Funktion. Die Softmax-Funktion wird auf die Inputwerte z angewendet und wird anschließend zur Berechnung der Kreuzentropie verwendet. Die Verwendung der Softmax-Funktion ermöglicht eine Berechnung der Kreuzentropie zweier Wahrscheinlichkeitsverteilun-

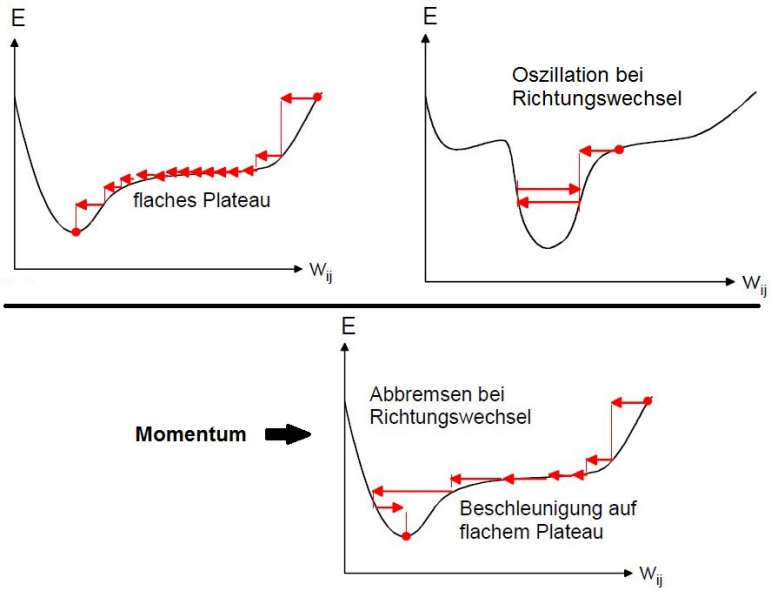


Abbildung 3.4.: Aus Meisel (2012): Beispielhafte Darstellung des Trainingsverlaufs bei Stagnation auf Plateaus, bzw. Oszillation in Tälern des Fehlergebirges. Abhilfe schafft hier die Einführung eines Momentum-Parameters.

gen, die nicht normiert sein müssen. Die Softmax-Funktion verändert die Werte eines Vektors, sodass die Summe der Elemente dieses Vektors 1 ergibt. Die Werte werden dabei so verändert, dass ihre Ausprägung weiterhin erhalten bleibt. Sie ist definiert durch

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad \text{mit } j = 1, \ldots, K. \tag{3.9}$$

Der Vorteil der Kreuzentropie gegenüber der Fehlerquadratsumme liegt im verbesserten Lernverhalten, wie in Abbildung 3.5 dargestellt. Liegen die initialen Gewichtungen und Bias-Werte weit entfernt vom optimalen Punkt im Fehlergebirge ist die Lernkurve gegenüber der Verwendung der Fehlerquadratsumme deutlich verbessert. Die Grafik zeigt, wie sich die Kreuzentropie bereits bei einem Neuron auswirkt bei dem Gewichtung und Bias weit vom Optimum entfernt liegen. Das Neuron hat in diesem Beispiel lediglich die Aufgabe, bei einem Input-Wert von 1 den Wert 0 auszugeben (Nielsen (2017)).

Die Parameter des Trainingsverfahrens und die Architektur des Netzes unterscheiden sich von den Parametern der Gewichtungen innerhalb des Netzes. Daher werden Parameter, die den Trainingsalgorithmus sowie den Aufbau des Netzes und dessen Layer beschreiben als Hyperparameter bezeichnet. Diese unterscheiden sich außerdem dadurch, dass sie vor Beginn

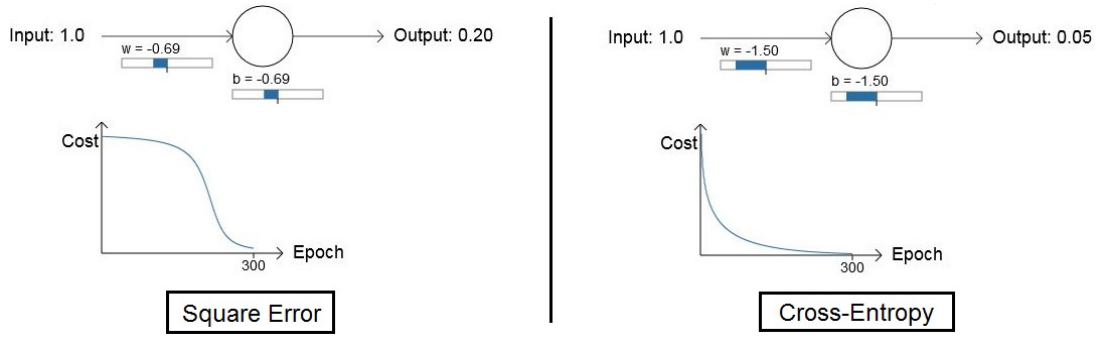


Abbildung 3.5.: Nielsen (2017) zeigt wie die Kreuzentropie die Lernkurve bereits bei einem Neuron verbessert.

des Trainings manuell definiert werden, während die Parameter der Gewichtung während des Trainings von der Backpropagation angepasst wird.

3.1.4. Stochastic Gradient Descent

Das Training der Netze wird im CNTK mittels "Stochastic Gradient Descent" (SGD) durchgeführt. Dies ist eine stochastische Methode, um bei iterativen, differenzierbaren Funktionen Minima und Maxima zu finden.

SGD ist ein Batch-Verfahren und verwendet im Gegensatz zum ursprünglichen Gradient Descent Algorithmus nur einen Teilbereich, d.h. eine "Mini-Batch" des Trainingssatzes. Die optimale Größe der Mini-Batch ist in jeder Architektur unterschiedlich. Daher kann die optimale Größe einer Mini-Batch vorher nicht bestimmt werden und muss für das Netz individuell festgestellt werden. Die Mini-Batch sollte die Größe des Training-Datensatzes nicht überschreiten.

In Gleichung 3.10 ist das Update J des Parameters θ durch das Trainingspaar $x^{(i)}, y^{(i)}$ dargestellt. α gibt hierbei die Lernrate des aktuellen Updates an.

$$\begin{aligned} v &= \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \\ \theta &= \theta - v \end{aligned} \tag{3.10}$$

Im Subterm γv der Gleichung 3.10 ist das Momentum $\gamma \in [0; 1]$ enthalten. Das Momentum wird nach UFLDL Computer Science Department (2016) in dieser Arbeit als Standardwert auf $\gamma = 0.9$ gesetzt.

3.2. Time Recurrent Networks

Ein Time Recurrent Neural Network (TRNN) unterscheidet sich vom Multi-Layer Perzeptron durch die Klassifizierung von Merkmalen, die sich über die Zeit verändern. Diese Netze haben den Vorteil, Zusammenhänge innerhalb von Sequenzen erkennen zu können. Es wird daher kein statischer Input verwendet, sondern eine Sequenz aus Inputvektoren, die nacheinander in das Netz gespeist werden. Wie diese Zusammenhänge durch das Netz erkannt werden können ist vom jeweiligen Netztyp abhängig.

3.2.1. Sliding-Window Verfahren

Eine Methode, um sequentielle Daten zu klassifizieren, ist eine besondere Form eines Multilayer Perzeptron. Die Klassifizierung dynamischer Abläufe wird durch das "Sliding-Window"-Verfahren (SW) realisiert. Dies bedeutet, dass sich der Inputvektor X zum Zeitpunkt t aus dem aktuellen Merkmalsvektor x_t und den Inputdaten vorheriger Merkmalsvektoren x_{t-i} zusammensetzt. Dadurch vervielfacht sich der Inputraum des Netzes und setzt mehr Neuronen im Hidden-Layer voraus, um den Input erfassen zu können.

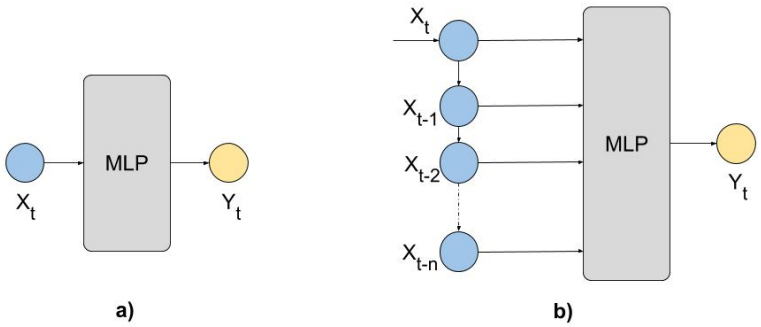


Abbildung 3.6.: Die schematische Darstellung des Sliding-Window-Prinzips. a) ein herkömmliches MLP für statische Daten. b) ein MLP für Sequenzen, dessen Input aus einem aktuellen und mehreren vorheriger Markmalsvektoren entsteht.

In Abbildung 3.6 ist der Unterschied zwischen einem herkömmlichen MLP und solchen mit Sliding-Window-Verfahren dargestellt. Das Hinzufügen von Merkmalsvektoren vergangener Frames (x_{t-i}) ermöglicht die Klassifizierung von Trajektorien und Sequenzen.

Beispielsweise werden für das Sliding-Window Verfahren dieses Systems der aktuelle Input x_t und zusätzlich die vier vorherigen Inputvektoren $[x_{t-1}, \ldots, x_{t-4}]$ verwendet (siehe (3.11)). Durch den Sequenz-Zeitraum $|t| = 5$ und die Größe des Inputvektors $|x_t| = 10$ zum Zeitpunkt

t ergibt sich ein gesamter Inputraum $X_t = 5 * 10 = 50$. Der Output Y_t berechnet sich anschließend wie in einem MLP zur Klassifizierung statischer Daten durch die Gewichtungsmatrix W_x und den Biaswert b.

$$\begin{aligned} X_t &= concat(x_t, x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}) \\ Y_t &= f(W_x \circ X_t + b) \end{aligned} \tag{3.11}$$

3.2.2. Neuronales Recurrent-Netz

Ein Recurrent Neural Network (RNN) unterscheidet sich vom MLP durch Rückkopplung der Neuronen eines oder mehrerer Layer. In der Regel sind dies die Hidden-Layer. Die Rückkopplung ist dadurch gegeben, dass der Ausgabewert des Neurons nicht nur an die nächste Schicht, sondern auch an sich selbst übergeben wird. Dies bewirkt, dass der Zustand des Neurons im nächsten Zeitschritt der Sequenz weiter mit eingerechnet wird und sich das Neuron so an den Zustand "erinnert".

Abbildung 3.7 stellt das Prinzip eines Recurrent Netzes in mehreren Sichtweisen dar. In Bereich d) der Abbildung ist das RNN logisch über die gesamte Input-Sequenz X aufgespannt. Hier ist zu erkennen, dass zu jedem Zeitpunkt auch eine Ausgabe durch das Netz erfolgt und daher nicht nur der Input, sondern auch der Output eine Sequenz ist. Eine Betrachtung des Resultats der Klassifizierung ist somit häufig erst nach einigen Sequenz-Schritten, oder im schlechtesten Fall im letzten Schritt sinnvoll.

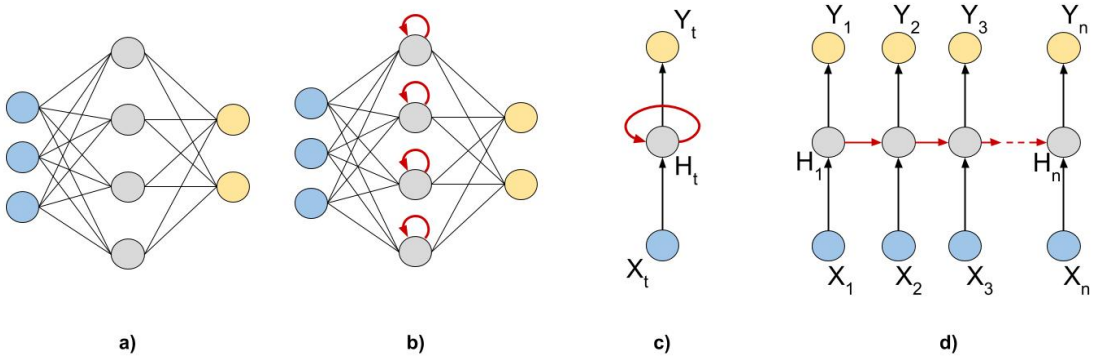


Abbildung 3.7.: Ein Recurrent Netz dargestellt in unterschiedlichen Betrachtungsformen: a) Ein MLP ohne Klassifizierung über Zeit. b) Ein Recurrent-Netz mit Rückkopplungen im Hidden-Layer. c) Schematische Veranschaulichung eines RNN. d) Ein RNN, aufgespannt über die gesamte Input-Sequenz.

Wie in Gleichung 3.12 gezeigt, berechnet sich der Wert der Recurrent-Neuronen zum Zeitpunkt t durch den aktuellen Input X und der Input-Werte des vorherigen Schrittes H_{t-1}. Beide besitzen dabei eigene Gewichtungsmatrizen.

$$H_t = f(W_x \circ X_t + W_h \circ H_{t-1} + b) \tag{3.12}$$

Da die Backpropagation bei Recurrent Netzen nicht nur durch die einzelnen Layer, sondern auch über die Sequenz hinweg durchgeführt werden muss, spricht man dabei von Backpropagation Through Time (BPTT). Diese ist im schematisch aufgespannten RNN (Vgl. Abbildung 3.7 d)) in Abbildung 3.8 dargestellt. Je größer das Zeitfenster der Sequenz, desto größer wird der Zeitraum und die Anzahl der Schichten, über die zurück propagiert werden muss. Bei der Verwendung von Sigmoid-Aktivierungsfunktionen können Gradienten die sich 0 oder 1 annähern bei voranschreitender BPTT verschwinden. Dieses Phänomen wird "vanishing gradients"-Problem genannt (Hochreiter und Schmidhuber (1997)).

Dieses Problem kann durch die ReLU als Aktivierungsfunktion gelöst werden, da diese Funktion positiv linear ist und im positiven Zahlenraum keine Sättigung findet, wie es bei Sigmoid-Funktionen der Fall ist. Voraussetzung dabei ist jedoch, dass die Input-Werte ausschließlich im Zahlenraum $\mathbb{R}_+$ liegen.

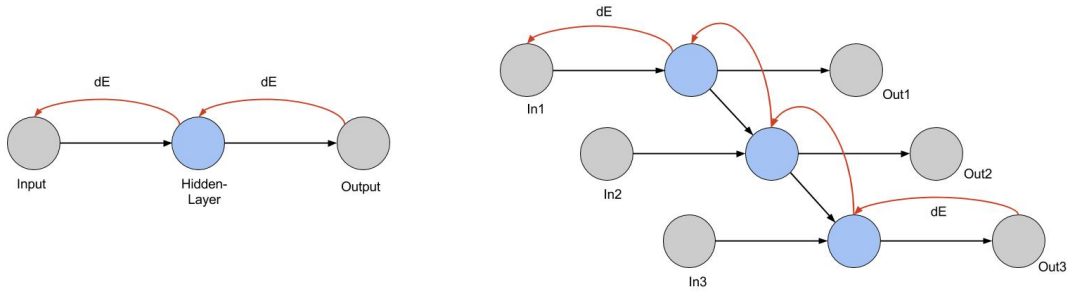


Abbildung 3.8.: Darstellung der BPTT. Links die Backpropagation eines MLP. Rechts die BPTT in einem über die Sequenz aufgespannten RNN.

3.3. Faltungsnetze

Mit Convolutional neural networks (CNN) kommt ein weiterer Typ neuronaler Netze in dieser Arbeit zur Anwendung. Solche Faltungsnetze eignen sich besonders, um Merkmale aus Bildern

zu extrahieren und Objekte zu erkennen. CNNs bestehen neben den in Abschnitt 3.1 erwähnten Layern aus mehreren zusätzlichen Schichten. Ein Faltungsnetz besteht aus

- Convolutional-Schichten,
- Subsampling-Schichten,
- Dropout und
- einem Fully-Connected Layer.

Faltungsnetze sind von Fukushima (1980) erstmals vorgestellt und von LeCun u. a. (1998b) unter dem Namen "LeNet-5" verbessert worden. Abbildung 3.9 zeigt ein Faltungsnetz eines "LeNet" Models. In diesen Modellen liegt sowohl Convolutional-, als auch Subsampling-Schicht in zweifacher alternierender Reihenfolge vor.

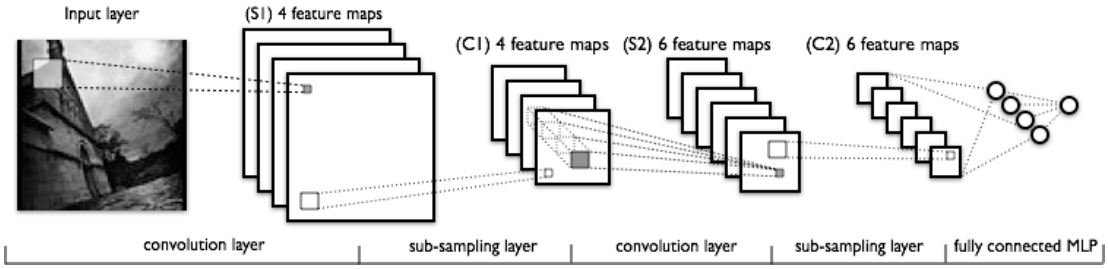


Abbildung 3.9.: Ein CNN[1] anhand des Modells "LeNet" (LeCun u. a. (1998b)). Zu erkennen sind die unterschiedlichen Layer für Faltung (*engl.* "Convolution") und Subsampling. Abschließend werden die daraus entstandenen Featuremaps mit einem Fully-Connected Layer zusammen gefasst.

Als Input dienen dem CNN Bilddaten in drei Dimensionen. Zu diesen Dimensionen gehören die Höhe und Breite des Bildes, sowie die Tiefe, in der das Farbspektrum angegeben wird. So ist beispielsweise die Tiefe bei Bildern des RGB-Farbraums 3, bei Grauwertbildern 1.

In jedem Schritt werden bei der Faltung Merkmale in den Pixeldaten des aktuellen Frames gesammelt und in "Featuremaps" abgelegt. Die Featuremaps werden durch ein Subsampling-Verfahren (beispielsweise "Max-Pooling") verkleinert.

3.3.1. Faltungsschicht

Bei einer Faltung handelt es sich um eine bestimmte Art der Filterung eines Bildes. Die Faltung ermöglicht es bestimmte Merkmale des Bildes hervorzuheben oder auszublenden. Solche

[1] Aus http://deeplearning.net/tutorial/lenet.html - Besucht am 8.2.2017

Merkmale sind beispielsweise Kanten. Für diese Operation wird ein sogenannter Faltungskern (*engl.* "Kernel") benötigt, der angibt, wie die Faltungsoperation durchgeführt werden soll. Solche Faltungsoperationen existieren auch in anderen Bereichen der Bildverarbeitung, wie beispielsweise der Kantenschärfung oder Kantendetektion.

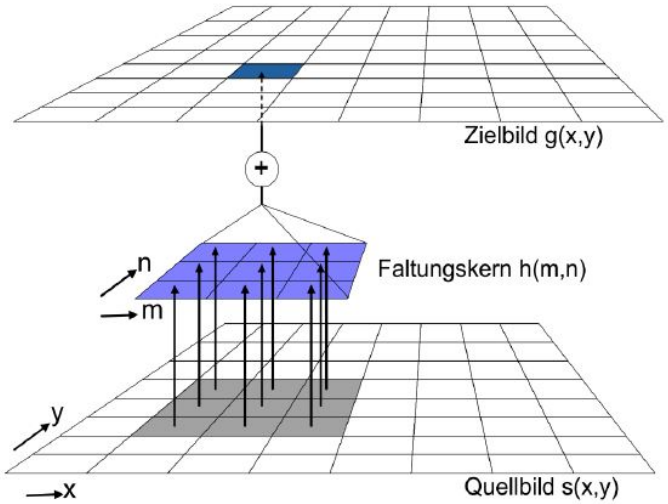


Abbildung 3.10.: Aus Meisel (2012): Eine Faltungsoperation mit einem 3×3-Kernel.

In Abbildung 3.10 ist die Faltung mit einem solchen Kernel abgebildet. Eine Faltung hat dabei mehrere Eigenschaften. Zunächst wird die Größe des Faltungskerns angegeben, beispielhaft in der Grafik als 3×3-Faltungskern dargestellt. Beim Faltungsvorgang wird der Kernel zunächst an einer Stelle auf das Quellbild gelegt. Anschließend werden die Grau- oder Farbwerte an dieser Stelle mit dem Faltungskern multipliziert und abschließend für das Zielbild addiert.

Dieser Vorgang wird über das gesamte Quellbild, beginnend von oben links horizontal wiederholt bis das letzte Pixel des Bildes abgedeckt ist. Zusätzlich kann als Parameter angegeben werden, ob der Kernel bündig oder überlappend laufen soll (*engl.* "padding"). Weiterhin wird für den Faltungskern eine Schrittweite (*engl.* "stride") angegeben, nach der jeweils eine Faltung durchgeführt wird.

3.3.2. Pooling-Schicht

Die Pooling-Schicht hat die Aufgabe die Dimension der Merkmale, die bei der Faltung in den Featuremaps gesammelt wurden, zu reduzieren und zu filtern. Dazu wird häufig der Algorithmus Max-Pooling verwendet. Die Parameter dieses Verfahrens sind analog zu denen

zur Faltungsschicht. Auch hier wird eine Kerngröße, sowie eine Schrittweite angegeben. Die Anwendung von Max-Pooling Schichten zwischen Faltungsschichten steigert die Abstraktion von Merkmalen, da die Verkleinerung es ermöglicht Merkmale höherer Ordnung zu erkennen.

Der Max-Pooling-Algorithmus ist weitaus weniger komplex als der zur Faltung. Hier wird lediglich der Maximalwert aus dem aktuellen Pooling-Kernel an die nächste Schicht übergeben. Dieser Vorgang ist in Abbildung 3.11 beispielhaft mit einem Filter der Größe 2×2 und der Schrittweite 2 abgebildet.

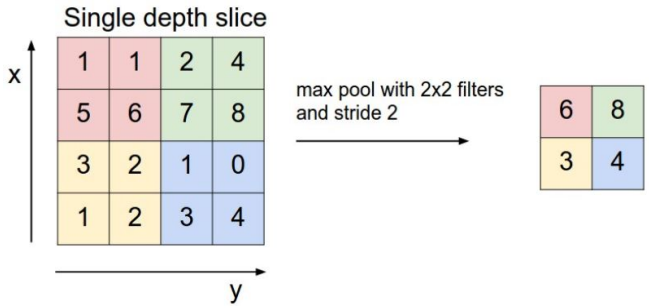


Abbildung 3.11.: Exemplarische Darstellung[2] des Max-Pooling für einen 2×2-Kernel mit Schrittweite 2.

3.3.3. Dropout

Viele CNN besitzen einen Dropout-Layer, der in der Regel nach dem letzten Pooling-Layer eingefügt wird. Diese Schicht verhindert das in Abschnitt 3.1.2 erwähnte Problem des overfittings. Erreicht wird dies durch das Ausblenden von bestimmten Neuronen während des Trainings, ohne sie zu löschen. Die Wertigkeit der Neuronen stehen nach dem Traininingsdurchlauf wieder zur Verfügung. Durch das Ausblenden sind die übrigen Neuronen gezwungen ihre Gewichtungen so anzupassen, dass auch ohne die Verfügbarkeit der ausgeblendeten Neuronen ein gutes Ergebnis erzielt wird. Dieser Vorgang verringert die Koadaption der Neuronen und erzeugt so ein Ergebnis mit einer geringeren Wahrscheinlichkeit für ein overfitting.

3.3.4. Fully-Connected Layer

Abgeschlossen wird das Netz durch einen Fully-Connected-Layer. Dieser entspricht dabei der Funktionsweise eines MLP. Dazu wird mindestens ein Hidden-Layer und ein Output-Layer

[2] Aus http://cs231n.github.io/convolutional-networks/ - Besucht am 8.2.2017

verwendet. Alle Neuronen der vorherigen Schicht sind dabei mit allen Neuronen dieses Hidden-Layers verbunden. Die Größe des Output-Layers entspricht wie bei den anderen Netztypen der Anzahl der Klassen.

4. Vorverarbeitung

Die Erkennung von Gesten ist besonders bei hohem Detailgrad ein komplexer Vorgang. Beispielsweise bei der Klassifizierung der Fingerstellung können schnell Probleme auftreten. Zusätzlich bedarf es bei detaillierten Gesten oft einer Vielzahl von Merkmalen, sodass der Umfang der nicht redundanten Informationen einer Geste schnell steigt. Es soll daher getestet werden, ob es möglich ist, diese Komplexität auf mehrere Ansätze zu verteilen. Dazu werden die Gesten in ihre Bestandteile aufgeteilt. Diese Bestandteile werden in verschiedenen Modalitäten untersucht und verarbeitet.

Aufbau der Gesten

Für das Testen dieses Konzepts wurden Gesten der deutschen Gebärdensprache als ausreichend Komplex erachtet. Diese Gesten bestehen aus folgenden nicht redundanten Merkmalen:

- Die Abstände der Hände und Arme zueinander und zum Körper und
- die Hand- und Fingerstellung.

Gesten der Gebärdensprache sind ein komplexer Zusammenschluss mehrerer optischer Faktoren. Primäre Objekte sind hierbei die Hände und Arme, die eine Geste und dadurch ein Wort formen. Entscheidend sind dabei die Stellung der Hände und Finger, sowie der Relation beider Hände zu einander und zum Körper. Der Kontext beider Hände muss also erhalten blieben, weshalb die Merkmale beider Hände grundsätzlich zusammengefasst werden. Weiterhin spielt die Bewegungsrichtung der Hände und Arme, deren Geschwindigkeit und Weitläufigkeit der Bewegung eine große Rolle. Wie in Abbildung 4.1 dargestellt zeigt bei der Geste “schreiben” eine flache Hand nach oben, während die andere mit Daumen und Zeigefinger zusammengedrückt kreisende Bewegungen vom Körper zur Seite weg macht.

Sekundären Einfluss auf die Geste hat das Gesicht. Der Mund formt dabei das gesprochene Wort nach, um die Geste zu verdeutlichen. Die Augen und Augenbrauen, sowie die gesamte Mimik des Gesichts bestimmen zusätzlich, ob es sich bei gebärdeten Sätzen um Fragestellungen, Befehlsformen oder anderen emotionale Ausdrücke handelt. Diese sekundären Faktoren werden

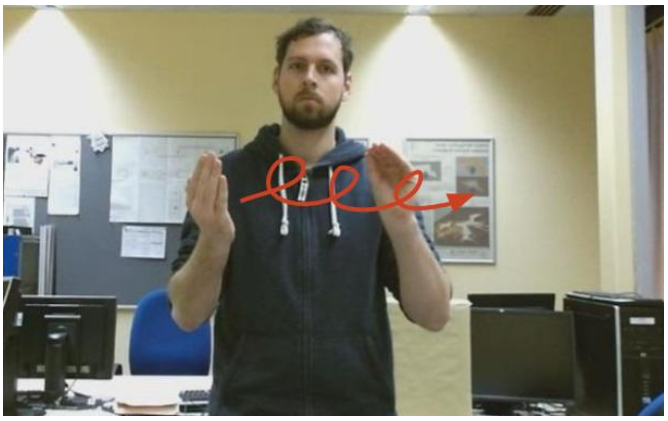


Abbildung 4.1.: Der Ablauf einer Gebärden-Geste anhand des Beispiels “schreiben”.

in dieser Arbeit nicht beachtet, da diese keinen Einfluss in den Gesten der Testdaten haben sollen.

Manche Gesten sind sich sehr ähnlich und unterscheiden sich lediglich anhand abweichender Fingerstellung, bei gleichbleibender Position der Hände. Andere sich ähnelnde Gesten wiederum haben dadurch unterschiedliche Bedeutung, dass sich bei gleichbleibender Handstellung die Position der Hände verändert.

Aufnahme der Daten

Eine Geste besteht in dieser Arbeit aus einer Sequenz über etwa fünf bis zehn Frames. Es wird jeder sechste Frame der Kinect verwendet, was bedeutet, dass alle 200 Millisekunden ein Frame aufgenommen wird. Die Aufnahme dieser sequenziellen Daten passiert in drei unterschiedlichen Modalitäten, welche mit der Kinect erlangt werden können. Dazu zählen die Frames

- des Teifen-Sensors,
- des Body-Trackings und
- der Farbkamera.

Aus diesen Quellen werden in drei Teilsystemen sequenzielle Informationen gesammelt, weiterverarbeitet und klassifiziert. Abbildung 4.2 zeigt die Verarbeitung der Kinect-Daten, welche anschließend in das hybride neuronale Netz gespeist werden. Zu erkennen ist, dass jedes Teilsystem eine eigene Verarbeitungskette besitzt.

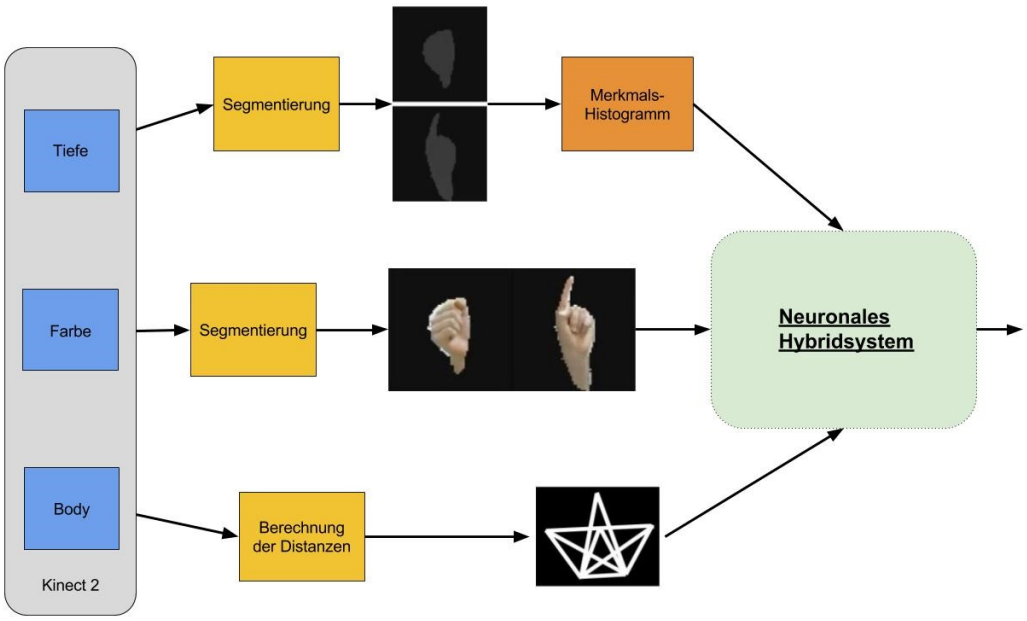


Abbildung 4.2.: Darstellung der Vorverarbeitungsketten der Teilsysteme, bevor sie als Input für das Hybridsystem verwendet werden.

Die Teilsysteme

Das Body-Tracking ermöglicht die Berechnung der Abstände der notwendigen Joints. Diese Distanzen dienen zur Klassifizierung der Orientierung der Arme. Zehn Abstände werden berechnet und der Sequenz für das Teilsystem der Distanzen angefügt.

Die Tiefendaten werden dazu genutzt, um dreidimensionale Merkmale beider Hände zu sammeln. Dazu wird lediglich die nähere Umgebung der jeweiligen Hand betrachtet und vom Hintergrund segmentiert. Anschließend wird ein Histogramm der einzelnen Hand-Tiefen-Frames erstellt und zur Sequenz für das Teilsystem dreidimensionaler Handmerkmale hinzugefügt.

Die Frames der Farbkamera dienen zur Sammlung zweidimensionaler Merkmale. Wie bereits bei den Tiefen-Daten wird hier nur die nähere Umgebung der Hände betrachtet und mithilfe des Tiefen-Bildes segmentiert. Die Vorverarbeitung der Merkmale besteht hierbei nur aus dem zusammenfügen beider Handframes zu einer Bitmap.

Bei keinem der Systeme wird Normierung durchgeführt, da als Aktivierungsfunktion ausschließlich die Rectified Linear Unit verwendet wird. Die Merkmale aller Systeme liegen im Werte-Bereich $\mathbb{R}_+$, was eine Verschiebung des Merkmalsraums ebenfalls hinfällig macht.

4.1. Orientierung der Arme

Die Orientierung der Arme kann durch die Abstände der Joints zueinander und zum Körper ausgedrückt werden. Als Distanz wird lediglich die Länge der jeweiligen Vektoren zwischen bestimmten Punkten berechnet. Die dreidimensionalen Positionen der Joints werden von der Kinect bereitgestellt. Hier betroffenen sind, wie in Abbildung 4.3 zu erkennen, jeweils beide Ellbogen, Handflächen, die Brust und der Kopf. Diese Idee wurde aus Wang u. a. (2015) übernommen und erweitert.

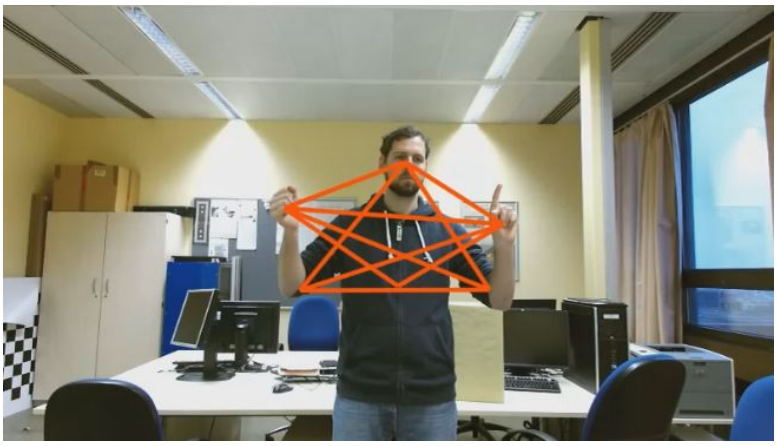


Abbildung 4.3.: Die berechneten Distanzen zwischen einzelnen Joints, die sich während einer Geste Verändern.

Alle Joints sind zwischen den Körperseiten miteinander verbunden. Eine Verbindung von Hand und Ellbogen einer Körperseite ist hinfällig, da diese Distanz stets gleich ist. Insgesamt werden somit zehn Distanzen pro Frame berechnet und aufgenommen.

Berechnet werden die Distanzen d als Vektorlängen zwischen zwei Punkten P_i und P_j:

$$d(P_i, P_j) = |\overrightarrow{P_i P_j}| = |\vec{p_j} - \vec{p_i}| \quad \text{, mit} \quad \vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4.1}$$

Durch die dreidimensionalen Joint-Koordinaten $\vec{p}$ der Kinect, sind die berechneten Vektoren und ihre Länge unabhängig von Rotation und Translation des Benutzers zur Kamera.

4.2. Beschreibung der Handstellung

Durch die hohe Komplexität der Hand und Fingerstellung werden die Merkmale der Handstellung auf zwei Arten gesammelt und verarbeitet:

- Dreidimensionale Merkmale über den Tiefensensor und
- Zweidimensionale Merkmale über die Farbkamera.

Bevor diese Merkmale gesammelt und verarbeitet werden, ist eine Segmentierung der Hände vom Hintergrund von Nöten, damit dieser keinen Einfluss auf die Klassifizierung hat. Weiterhin sind sowohl die drei- als auch die zweidimensionalen Daten der Handmerkmale jeweils pro Frame statisch. Für eine dynamische Klassifizierung müssen diese Frames in eine zusammenhängende Sequenz überführt werden.

Bildsegmentierung

Um die Hände vom Hintergrund und somit von groben Störfaktoren zu segmentieren, wird zunächst lediglich die nähre Umgebung der Hände in kleineren Bereichen ausgeschnitten. Dazu werden die aktuellen x- und y- Koordinaten des jeweiligen Handjoints benutzt und die Ausschnitte dahingehend zentriert. In Abbildung 4.4 ist dies dargestellt. Die pixelweise Berechnung der kleineren Frames erfolgt durch die Postion des Joints der Hand in folgendem Algorithmus:

```
const float MapDepthToGrayVal = 8000 / 256;
int frameHeight = frameWidth = 70;
for (int y = -frameHeight; y < frameHeight; y++)
{
   for (int x = -frameWidth; x < frameWidth; x++)
   {
      // Offset im Originalbild berechnen
      int offset = originalFrameWidth * ((int)p.Y + y) + ((int)p.X + x);
      // Schrieben des Tiefenwertes in neuen Bildausschnitt
      depthPixels[index++] = (byte)(frameData[offset] / MapDepthToGrayVal);
   }
}
```

Wie in Abbildung 4.4 zu sehen werden in den Frames mit den Handausschnitten (“Hand-Frames”) die Hände vom Hintergrund segmentiert. Dies wird realisiert durch eine Kombination zwischen Tiefen- und Joint-Frame. Der Joint Frame liefert die aktelle Postition der Hand p, d.h. des Handmittelpunktes. Durch den CoordinateMapper der Kinect wird dieser Punkt auf das Tiefenbild übertragen. Der Tiefenwert dieser Koordinate dient als Ausgangspunkt für die Segmentierung. Von links oben horizontal nach rechts unten werden nur die Pixel des ursprüng-

lichen Frames ausgewählt, die maximal 35 Pixel vertikal und horizontal vom Handmittelpunkt entfernt sind (siehe “frameWidth” bzw. “frameHeight” $= 70$). Diese Pixelpositionen werden in der Variable “offset” gespeichert und lesen den Tiefenwert des Pixels aus. Dieser Tiefenwert wird in den Grauwertbereich konvertiert (Vgl. Abschnitt 2.2.1) und in den Handausschnitt (“depthPixels”) geschrieben.

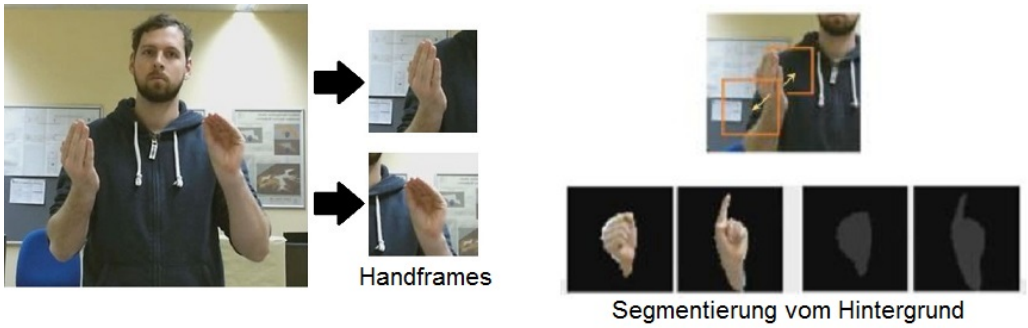


Abbildung 4.4.: Die Segmentierung der Hände im Tiefenbild. Links: Ausschnitt für die lokale Betrachtung der Hände in Handframes. Rechts: Die Segmentierung der Hände vom Hintergrund und die resultierenden segmentierten Handframes des Tiefen- und Farbbildes.

Für die Segmentierung der Hand vom Hintergrund werden die Tiefenwerte in den Hand-Frames geprüft. Gleichung 4.2 zeigt die Logik mit der Pixel des Hand-Frames im Tiefenbild selektiert werden. Liegt der Tiefenwert D eines Pixels p innerhalb eines Grenzbereichs c, so wird dieser Tiefenwert D_p gespeichert. Andernfalls wird er verworfen, was bedeutet, dass der Tiefenwert des Pixels $D_p = 0$ gesetzt wird. Der Grenzbereich erstreckt sich dabei, mit einem konstanten Schwellwert von $c = 60$ vor und hinter dem Initial-Wert, das heißt dem Handmittelpunkt D_i.

$$D_p = \begin{cases} D_p, & \text{wenn} \quad D_p < (D_i + c) \wedge D_p > (D_i - c), \\ 0, & \text{sonst.} \end{cases} \tag{4.2}$$

Die Segmentierung der Hand-Frames im Farbbild macht sich diese Pixel-Auswahl zu nutze. Beim segmentieren des Tiefenbildes wird ein Mapping erstellt, dass aussagt, welche Pixel selektiert und welche verworfen wurden. Die Anwendung dieses Mappings auf den Farb-Frame bedarf aufgrund der unterschiedlichen Auflösung zum Tiefen-Frame einer Konvertierung. Für die Handframes der Tiefendaten wurde die Bildgröße 70×70 Pixel, für die Handframes der Farbdaten 210×210 Pixel festgelegt, um die Auflösung aufeinander abzustimmen. Da das

Farbbild somit dreimal größer ist als das Tiefenbild, wird ein Pixel im Tiefen-Frame auf die Größe 3×3 im Farbframe konvertiert (siehe Abbildung 4.5).

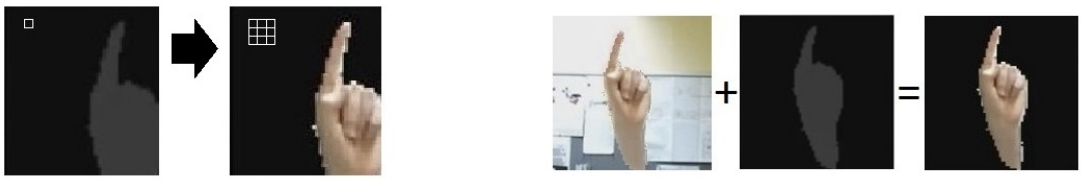


Abbildung 4.5.: Die Segmentierung der Hände im Farbframe. Links zu sehen ist die Adaption der Segmentierung des Tiefenbildes auf das Farbbild. Rechts das Resultat der Segmentierung der Farbframes.

Aus den Tiefenbildern können nun dreidimensionale Merkmale und aus den Farbbilder zweidimensionale Merkmale gewonnen werden, die sich nur auf die nähere Region der Hände bezieht.

Sequentielle Verarbeitung

Alle Daten der Gesten müssen als Sequenzen aufgenommen werden. Auf Grund der Vorgehensweise in dieser Arbeit und den verwendeten Algorithmen müssen die Frames bei der Vorverarbeitung der Handmerkmale einzeln betrachtet werden. Daher sind alle Dateien der aufgenommenen Tiefen- und Farbbilder in einem bestimmten Schema abgespeichert, welches wie folgt bestimmt ist:

$$..//[Datentyp][Handseite]\_[GestenID]\_[Sequenznummer]\_[Index].[dateiendung]$$

Dieses Schema bietet die Möglichkeit jedes Bild einer Hand, Geste und Sequenz für die Weiterverarbeitung zuordnen zu können. Beispielhaft lautet ein Eintrag:

$$..//pointCloudL\_031\_142\_00.pcd$$

Dieser Eintrag stellt das erste Bild (Index 0) der 142. Sequenz, der 32. Geste dar. Dabei handelt es sich um die Tiefendaten der linken Hand.

4.2.1. Dreidimensionale Merkmale

Für die Klassifizierung der dreidimensionalen Daten werden für jede Hand die segmentierten Hand-Frames der Tiefendaten verwendet. Diese Frames sind von der Größe 70×70 Pixel. Sie

bestehen daher aus 4900 Integer-Werten, die die Tiefe im Wertebereich $[0; 8000]$ ausdrücken. Aus diesen Tiefeninformationen soll ein Histogramm erstellt werden, um eine Klassifizierung effizient umsetzen zu können.

3D Feature Histogram

Als Feature-Histogramm wird OUR-CVFH verwendet. Für die Erstellung der dazu benötigten Punktwolke werden alle Werte des segmentierten Tiefen-Frames übernommen, die nicht 0 sind. Da die Daten des Tiefenframes der Kinect alle aus dem Sichtpunkt der Kamera sind, muss eine Umwandlung der Werte in eine Punktwolke erfolgen. Für diese Umwandlung existieren unterschiedliche Möglichkeiten, um eine Koordinate K_{xyz} der Kinect Tiefen-Daten in einen Punkt der Point Cloud umzurechnen. Eine Formel für diese Umrechnung ist folgende:

$$\begin{aligned} P_z &= 0.001 * K_z \\ P_x &= (K_x - CP_{cx}) * K_z / CP_{fx} \\ P_y &= (K_y - CP_{cy}) * K_z / CP_{fy} \end{aligned} \tag{4.3}$$

Bei den Werten CP handelt es sich um durch die Kamera definierte, konstante Parameter zum Ausgleich des Blickwinkels der Kamera. Dieser Ausgleich ist notwendig, um das zweidimensionale Tiefenbild, bestehend aus x ,y und Grauwert, in eine vom Blickwinkel unabhängige dreidimensionale Punktwolke umwandeln zu können (siehe Abbildung 4.6 [1]). Die in dieser Arbeit verwendeten Kamera-Parameter sind $CP_{cx} = 254.878$, $CP_{cy} = 205.395$ und $CP_{fx} = CP_{fy} = 365.456$.

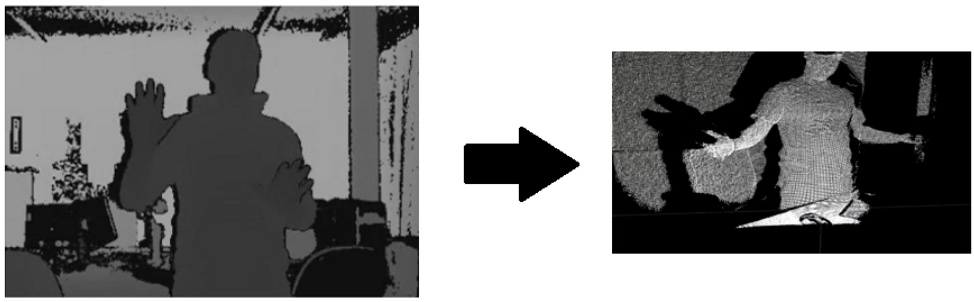


Abbildung 4.6.: Die Umwandlung des “zweidimensionalen” Tiefenframes zu einer dreidimensionalen Punktwolke.

[1] Die Darstellung der Punktwolke wurde freundlicherweise von Larry Li (https://larrylisky.com/) zu Verfügung gestellt.

Die berechneten Wolkenpunkte werden als dreidimensionale Koordinaten als Point Cloud Data (PCD) in einer Datei nach dem in Abschnitt 4.2 gezeigtem Schema abgelegt. Das Format einer PCD-Datei richtet sich nach der Point Cloud Library, um die Umrechnung in ein OUR-CVFH mit diesem Framework zu ermöglichen. Eine PCD-Datei enthält zusätzlich zu den Wolkenpunkten einen Header, der Informationen über die Punktwolke enthält. Dieser Header wird erst nach schreiben der Wolkenpunkte dynamisch erstellt und eingefügt. Dies ist dadurch begründet, dass manche Informationen wie beispielsweise die Anzahl der Punkte in der Wolke zuvor nicht vorhanden sind. Zusätzlich unterscheiden sich diese Informationen in jeder Punktwolke.

Eine aus diesem Zusammenschluss resultierende PCD-Datei ist in folgendem Listing beispielhaft dargestellt. Der Header legt mit Schlüsselwörtern beispielsweise die Anzahl der Punkte in der Punktwolke, die Anzahl der Koordinaten (x, y, z) und den Viewpoint der Kamera fest. Ab dem Schlüsselwort "DATA" beginnen die Punkte der Wolke in der Reihenfolge der angegebenen Koordinaten ("FIELDS").

```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z
SIZE 4 4 4
TYPE F F F
COUNT 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 2.2108
0.90805 0.35641 1.4358
0.81915 0.32 1.0241
...
0.97192 0.278 2.2122
```

Verarbeitungs-Pipeline

Während der Aufnahme der Gesten werden die berechneten Wolkenpunkte der Hand-Frames des Tiefenbildes in einer solchen PCD-Datei abgespeichert. Nach Aufnahme werden die Daten in einer Pipeline zu Cloud Training Data (CTD)-Dateien für die Speisung des Netzes umgewandelt. Diese Pipeline besteht aus folgenden Schritten:

1. Zunächst werden die PCD-Dateien mittels OUR-CVFH in Viewpoint Feature Histograms (VFH) konvertiert. In der VFH-Datei sind alle Merkmale des Histogramms größer 0 angegeben.

2. Als nächstes folgt die Umwandlung dieser VFH in Konvertierte Feature Histogramme (KFH). Diese ermöglichen die Lesbarkeit aller OUR-CVFH Features inklusive aller 0-Werte.
3. Abschließend wird eine CTD-Datei erstellt, in der alle KFH einer Sequenz in einem für CNTK lesbaren Format enthalten sind.

Die Umwandlung der VFH in KFH ist notwendig, da OUR-CVFH Nullwerte einzelner Klassen nicht in die VFH-Dateien schreibt, diese Nullwerte jedoch in die Merkmale mit einbezogen werden müssen. Abschließend werden CTD-Dateien beider Hände zusammengefügt, um den Kontext der Hände zueinander herzustellen.

Die Konfigurationsdatei der Berechnung des OUR-CVFH ist in Anhang A.4 angefügt. Dort sind auch die in dieser Arbeit gewählten Parameter für die Berechnung des OUR-CVFH zu entnehmen.

Ein wichtiger Parameter ist dabei der "EPSThreshold" für die Bestimmung der maximalen Abweichung der Winkel zwischen den Normalen. Hier wurde ein maximaler Winkel von $15°$ gewählt. Zwar entsteht dabei eine etwas größere Referenzfläche, jedoch wird weniger Zeit benötigt, um das Histogramm zu berechnen.

Bei der Ermittlung der OUR-CVFH-Parameter war es das Ziel die richtige Balance zwischen Detailgrad und Konvertierungsdauer zu finden. Eine Anpassung der Werte zu einem höheren Detailgrad hin könnte zwar zu einem genaueren Ergebnis führen, jedoch würde sich dadurch die Zeit deutlich erhöhen, die für die Umwandlung der Point Cloud in ein Histogramm erforderlich ist.

4.2.2. Zweidimensionale Merkmale

Neben den Tiefendaten werden auch die Farbbilder der Handframes aufgenommen. Wie auch bei den Tiefendaten werden diese Bilder für beide Hände einzeln abgespeichert. Bei den abgespeicherten Bildern handelt es sich im Vergleich zu den Tiefendaten jedoch um tatsächliche Bilddateien im "PNG"-Format. Auch diese werden im Sequenz-Schema nach Abschnitt 4.2 unter dem Datentyp "color" abgespeichert. Zusätzlich wird eine Map-Datei angelegt, welche die Dateinamen der Bild-Dateien enthält, um später beim Training und der Klassifizierung darauf zugreifen zu können.

Die Vorverarbeitung der zweidimensionalen Merkmale besteht aus zwei Schritten. Zunächst werden die Farb-Frames beider Hände zusammengefügt. Einzeln bestehen die Frames aus 210×210 Pixeln mit BGRA-Farbraum, was bedeutet, dass jedes Pixel vier Kanäle besitzt.

Nach der Fusion beträgt das Ergebnis-Bild eine Breite von 420 und einer Höhe von weiterhin 210 Pixeln. Der Farbraum bleibt dabei bestehen. Dieser Zusammenschluss der Hand-Frames in einem Frame ist notwendig, um den Kontext beider Hände zu erreichen. Dies geschieht bei der Verarbeitung der Tiefen-Daten erst nach der Konvertierung mit OUR-CVFH, beim Zusammenschluss der CTD-Dateien.

Der zweite Schritt besteht aus der Anpassung der Map-Datei. Hier werden die fusionierten Bild-Dateinamen eingefügt und die der einzelnen Hand-Frames entfernt. Bei den neuen Dateinamen bleibt das Namensschema vorhanden. Lediglich die Information der Handseite im Dateinamen wird entfernt (Vgl. siehe Abschnitt 4.2).

Bei der Vorverarbeitung der Farbbilder wurden die jeweils ersten drei Frames einer Sequenz entfernt, da diese oft irreführende Bilddaten beinhalten. Diese fehlerhaften Bilddaten werden in Abschnitt 7.2.1 näher erläutert.

Die weitere Verarbeitung zweidimensionaler Merkmale wird in einem Faltungsnetz durchgeführt, welches allerdings zum neuronalen Hybridsystem zählt. Wie diese Verarbeitung implementiert wird, zeigt das folgende Kapitel.

5. Das Hybridsystem

Der Kern dieser Arbeit besteht aus dem Zusammenschluss verschiedener neuronaler Netze. Einzelne Klassifizierungssysteme können die Komplexität, die mit detailreichen Gesten einher geht unter Umständen nicht erfassen. Möglicherweise werden unzureichende Trefferquoten bei der Klassifizierung erzielt, da der Umfang einer Geste nicht vollständig abgedeckt ist.

Die in Kapitel 4 beschriebenen Teilkomponenten sammeln unterschiedliche Merkmale solcher komplexer Gesten. Anschließend werden diese Teilkomponenten von eigenständigen neuronalen Netzen verarbeitet (siehe Abbildung 5.1). Abschließend werden die Merkmale dieser Teilsysteme in einem finalen Netz endgültig klassifiziert. So soll der Zusammenhang der Teilsysteme hergestellt und die Komplexität vollständig abgedeckt sein.

Der Aufbau, sowie die Konfiguration der Netze aller Teilbereiche und des finalen Netzes werden in diesem Kapitel genau erläutert.

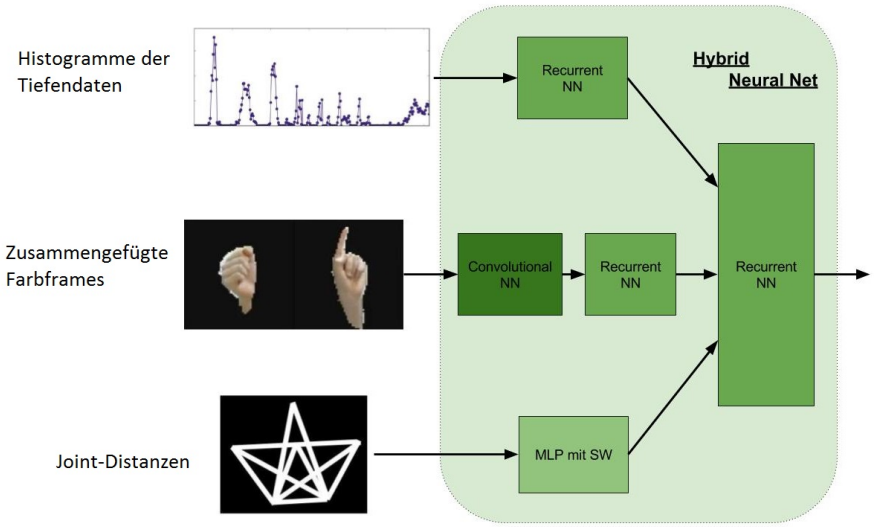


Abbildung 5.1.: Der neuronale hybride Ansatz: Unterschiedliche Netztypen in Teilsystemen mit Zusammenschluss in einem finalen Recurrent-Netz.

Für die Verarbeitung der Distanzen der Joints wird ein MLP mit Sliding-Window-Verfahren verwendet. Die Histogramme mit den dreidimensionalen Merkmalen werden sequentiell in einem RNN extrahiert. Bei der Erarbeitung der zweidimensionalen Features kommen zwei unterschiedliche neuronale Netze zur Anwendung. Die Farbbilder werden zunächst mit einem CNN klassifiziert und die dort erhaltenen Features in ein RNN zur sequentiellen Verarbeitung gespeist.

Die Klassifizierungen der einzelnen Systeme werden anschließend zusammengefasst und in einem Endsystem final klassifiziert. Dieses finale neuronale Netz stellt daher eine Art Jury dar, die die vorher gesammelten Merkmale in eine Gesten-Klasse umwandelt.

Allgemeine Netzeigenschaften

Zu klassifizieren sind 35 Gesten. Alle Netze des Systems besitzen daher 35 Neuronen im Output-Layer. Dies dient dazu, dass jedes Netz einzeln evaluiert werden kann und auch auf diesen Zweck hin trainiert wird. Wie in Abbildung 5.2 dargestellt, wird nicht der Output der Teilsysteme, sondern der letzte Hidden-Layer für die Weitergabe der Werte an das jeweils nächste Netz verwendet.

Die Verwendung des letzten Hidden-Layers anstatt des Output-Layers begründet sich durch zwei Faktoren. Der letzte Hidden-Layer stellt lediglich eine Approximation der Werte dar, anstatt die konkreten - gegebenenfalls falsch klassifizierten - Werte zu nutzen. Weiterhin besteht die Möglichkeit auf diese Art einen größeren Featurevektor unabhängiger Merkmale für die finale Klassifizierung zu erhalten. Dieses Prinzip wirde aus Wu u. a. (2015) übernommen. Der letzte Hidden-Layer ist dabei in keinem Teilsystem rekurrent und besitzt in jedem Netz der Teilsysteme genau 128 Neuronen. Diese Anzahl wurde so angepasst, dass er den zu Input-Größen aller Teilnetze passt. Es ist wichtig, dass dieser Hidden-Layer in den Teilsystemen gleich groß ist, da so eine gleichmäßiger Einfluss der Teilsysteme auf das finale Netz garantiert ist.

Die Input- und Output-Layer-Größen aller Netze ergeben sich aus der jeweiligen Anzahl der Input-Merkmale und der Anzahl der 35 zu klassifizierenden Gesten. Die Größe des ersten HL unterscheidet sich bei allen Netzen. In den Teilsystemen wurde im ersten Hidden-Layer stets eine größere Anzahl an Neuronen als im Input-Layer gewählt, um den Input-Raum abzudecken und die Komplexität entsprechend erfassen zu können. Es wurde außerdem darauf geachtet diese Anzahl nicht zu hoch zu setzen, um Overfitting und überflüssige Neuronen zu verhindern (Vgl. Abschnitt 3.1.2). Das finale Netz besitzt durch alle Layer eine fortlaufend

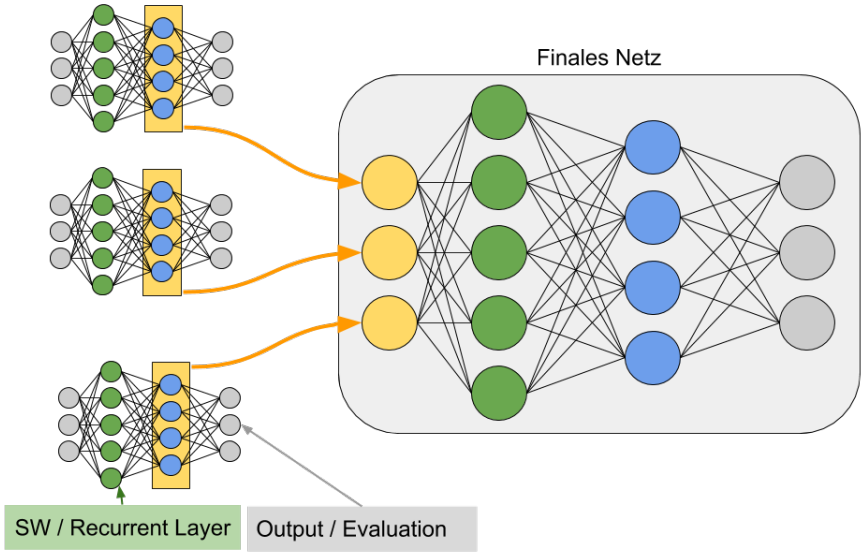


Abbildung 5.2.: Skizzierte Verwendung des Hidden-Layers in den Teilsystemen: Als Input des finalen Netzes werden nicht die Outputs, sondern die Features der jeweils zweiten Hidden-Layer der Teilsysteme verwendet.

sinkende Neuronenzahl, da hier die bereits in den Teilsystemen approximierten Merkmale lediglich zusammengefasst und auf 35 Klassen reduziert werden sollen.

Trainingsverfahren

Das Training der Netze wurde mit den folgenden Verfahren durchgeführt:

- Für den Fehlerwert (“Error-Prediction”) im Output-Layer wurde das Softmax-Verfahren und
- für die Backpropagation die kategorische Kreuzentropie mit Softmax verwendet.

CNTK empfiehlt als Fehler-Funktion für die Backpropagation grundsätzlich die kategorische Kreuzentropie mit Softmax.

Alle Teilsysteme verwenden als Aktivierungsfunktion die Rectified Linear Unit (ReLU). Dies soll den Verlauf und die benötigte Zeit des Trainings optimieren. Im folgenden werden die Details der einzelnen Netze betrachtet und die Trainings-Hyperparameter aufgelistet.

Die Ermittlung der SGD-Hyperparameter wurde in dieser Arbeit explorativ durchgeführt. Lediglich Lernrate und Minibatch-Größe wurden dabei für jedes Teilsystem angepasst. Das

Momentum wurde bei allen Netzen, außer dem CNN auf 0.9 gesetzt. Als maximale Anzahl der Epochen wurde festgelegt, dass ein Training abgebrochen wird, wenn sich die Error-Prediction über mehrere Epochen nicht verbessert oder einen Minimalwert 1.0^{-4} unterschritten hat.

5.1. Orientierung der Arme

Die Klassifizierung von Distanzen zwischen den Joints stellt das weniger komplexe Teilsystem dar. Daher kommt für die Erkennung der Orientierung der Arme das Sliding-Window Verfahren im MLP zum Einsatz. Ein MLP mit Sliding-Window (SW) ist einfacher zu implementieren und verbraucht durch Fehlen der Rekurrenz in CNTK weniger Speicherkapazität als Recurrent-Netze. Weiterhin kann ein SW bei geringer Komplexität eine schnellere Klassifizierung, sowie leicht verbesserte Ergebnisse gegenüber einem RNN erzielen (Weschta (2016b)). Dieses Subsystem erhält lediglich zehn Merkmale als Input.

Der gesamte Input für das Netz setzt sich aus den aktuellen zehn Merkmalen und den vier vorhergehenden Input-Vektoren zusammen. Insgesamt beträgt die Größe des Inputvektors daher 50. Das MLP besteht weiterhin aus zwei Hidden-Layern mit 256 Neuronen in der ersten und 128 Neuronen in der zweiten Schicht.

Netz-Parameter

Layer	Aktivierungsfunktion	# Neuronen
Input	-	50 [= 10 + 4*10]
Hidden 1	ReLU	256
Hidden 2	ReLU	128
Output	-	35

SGD-Parameter

Minibatch size	32
Momentum per MB	0.9
Epochs	1500
Learningrates per MB	0.002

Tabelle 5.1.: Hyperparameter des MLP mit Sliding-Window und den verwendeten SGD-Hyperparametern des Trainings.

Trainiert wurde das Netz mit einer Minibatch-Größe von 32 und einer Lernrate von 0.002 pro Minibatch. Um ein verwertbares Ergebnis zu erzielen, wurde in 1500 Epochen trainiert. Damit liegt die Anzahl der Epochen weitaus höher als die der restlichen Netze. Dies liegt daran,

dass dieses Verfahren grundsätzlich mehr Trainingsschritte benötigt, als Recurrent-Netze (vgl. Weschta (2016b)).

5.2. Dreidimensionale Handmerkmale

Für die Klassifizierung der OUR-CVFH zur Erkennung der dreidimensionalen Hand-Merkmale kommt ein neuronales Recurrent-Netz (RNN) zur Anwendung. Ein Multi-Layer Perzeptron mit Sliding-Window-Verfahren, würde bei diesem Teilsystem zur einer Überdimensionierung der Input-Daten führen. Dies liegt daran, dass mit den Histogrammen des CVFH pro Hand 308, d.h. 616 Merkmale insgesamt als Eingabe verwendet werden. Das Sliding-Window würde diese Zahl noch vervielfachen.

Das RNN besteht aus zwei Hidden-Layern: der erste Layer ist rekurrent und besteht aus 768 Neuronen, um den Inputraum vollständig abzudecken. Der zweite, lineare Layer besitzt analog zu den anderen Teilsystemen 128 Neuronen. Wie bereits erwähnt setzt sich der Eingabevektor aus $2 * 308$ Merkmalen zusammen, was bedeutet, dass der Input-Layer aus 616 Neuronen besteht. Der Output-Layer besitzt wie die der anderen Systeme 35 Neuronen, die die Gesten widerspiegeln und Auswertung dieses Einzelsystems ermöglicht.

Netz-Parameter

Layer	Aktivierungsfunktion	# Neuronen
# Input	-	616 (2*308)
# Recurrent	ReLU	768
# Hidden	ReLU	128
# Output	-	35

SGD-Parameter

Minibatch size	32
Momentum per MB	0.9
Epochs	300
Learningrates per MB	0.001

Tabelle 5.2.: Hyperparameter des RNN für die Erkennung dreidimensionaler Merkmale und den zugehörigen SGD-Hyperparametern des Trainings.

Die Hyperparameter des Trainings mit SGD unterscheiden sich nur minimal von denen des Teilsystems aus vorigem Abschnitt 5.1. Lediglich die Lernrate pro Mini-Batch ist reduziert.

Merklich geringer ist die Anzahl der Epochen, die wie in Weschta (2016b) gezeigt bei Recurrent-Netzen häufig deutlich geringer ist als bei Sliding-Window.

5.3. Zweidimensionale Handmerkmale

Die Klassifizierung der zweidimensionalen Handstellung wird in zwei Schritten durchgeführt. Zunächst werden die einzelnen zusammengefügten Handframes mittels eines Faltungsnetzes klassifiziert. Die innerhalb des Netzes trainierten Gewichtungen des Fully-Connected Layers werden anschließend ausgenutzt, um Trainings- und Testdaten für das Recurrent-Netz zu erstellen. Diese Idee basiert auf dem System von Wu u. a. (2015). Wie in Abbildung 5.3 dargestellt ist das Recurrent-Netz dafür zuständig die Klassifizierung einzelner Farbbilder in einen dynamischen, d.h. sequentiellen Kontext zu setzen.

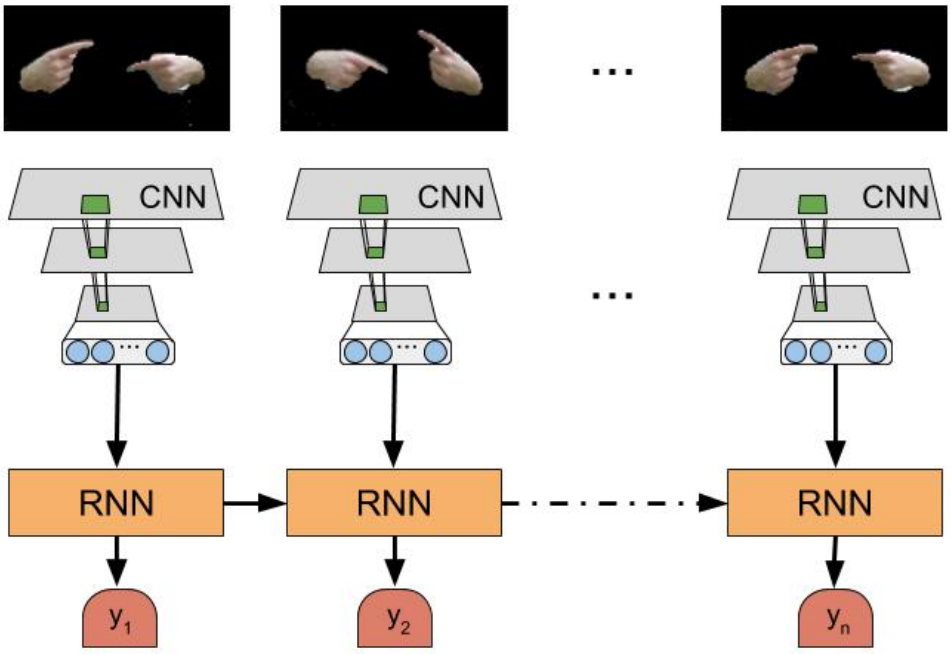


Abbildung 5.3.: Nach der Idee aus Wu u. a. (2015): Die Ergebnisse der Klassifizierung einzelner Bilder durch das CNN werden dafür genutzt, um für das RNN eine Sequenz zu erzeugen.

5.3.1. Convolutional-Netz

Das Faltungsnetz (CNN) nimmt als Input ein dreidimensionales Array an. Diese stellen die Breite, Höhe und Farbtiefe des zu klassifizierenden Bildes dar. Die Bilder, für dieses Teilsystem sind 420 Pixel breit, 210 Pixel hoch und haben eine Farbtiefe von vier Kanälen (BGRA). Die Höhe ist übernommen aus der Segmentierung aus Abschnitt 4.2.2. Die Breite ergibt sich aus der Fusion der Farbbilder der linken und rechten Hand von jeweils 210 Pixeln wie es in Abschnitt 4.2.2 beschrieben ist. Der Farbraum ändert sich während des Trainings und der Klassifizierung vom vier-kanaligen BGRA auf drei Kanäle (BGR). Dies liegt daran das der Alpha-Kanal im ImageReader in CNTK nicht beachtet wird und sogar Probleme beim Training des Netzes verursacht.

Netz-Parameter

Layer	Aktivierungsfunktion	Größe	Kernel [Stride]
Input	-	$420 \times 210 \times 3$ (Frame)	-
Convolution	ReLU	32 (Featuemaps)	5×5
Max-Pooling	-	32 (Featuemaps)	3×3 [2:2]
Convolution	ReLU	64 (Featuemaps)	3×3
Max-Pooling	-	64 (Featuemaps)	3×3 [2:2]
Dropout	-	-	-
Fully-Connected	ReLU	128 (Neuronen)	-
Output	-	35 (Neuronen)	-

SGD-Parameter

Minibatch size	16
Momentum TimeConstant	0.8
Dropout Rate	0.8
Epochs	30
Learningrates p. Sample	0.00001

Tabelle 5.3.: Hyperparameter des CNN mit dessen Layern in Reihenfolge und den zugehörigen SGD-Hyperparametern des Trainings.

Der Kern des Netzes beinhaltet zwei alternierende Layer mit je einem Convolutional- und einem Pooling Layer nach dem LeNet-Modell. Tabelle 5.3 zeigt die Layer in ihrer Reihenfolge. Der erste Convolutional Layer erzeugt 32 Feature-Maps, mit einem Faltungskernel der Größe 5×5. Anschließend folgt der erste Max-Pooling Layer. Dessen Kernel-Größe beträgt 3×3 mit der Schrittweite 2 des Kernels in jeweils horizontaler und vertikaler Richtung. Der zweite Convolutional Layer erzeugt mit 64 mehr Feature-Maps als der erste. Dafür ist die Kernel-

Größe hier auf 3×3 verringert. Kernel und Schrittweite des zweiten Max-Pooling Layer sind gegenüber dem ersten unverändert.

Die Kernel-Größen und Anzahl der Feature-Maps orientieren sich anhand von Beispielen des LeNet-5 und CNTK. Weitere Anpassungen wurden explorativ und anhand der Größe der Input-Bilder durchgeführt.

Neben den oben genannten Layern gibt es noch einen Input-Layer, ein Dropout, sowie einen Fully-Connected- und Output-Layer. Der Dropout wurde nach der letzten Max-Pooling-Schicht eingefügt. Nach dem Dropout folgt ein Fully-Connected Layer der Größe 128. Dieser wird später für die Erstellung der sequentiellen Merkmale für das RNN dieses Teilsystems genutzt. Der Output-Layer enthält, wie alle anderen Teilnetze 35 Neuronen zur Evaluation des Teilsystems.

Die SGD-Hyperparameter sind beim CNN teilweise von denen der Recurrent-Netze zu unterscheiden. Wie bereits bei den Hyperparametern der Netzarchitektur orientieren sich diese Werte an Beispielen und wurden explorativ angepasst. Hier wurden das Momentum anhand der Zeit ("TimeConstant") und die Lernrate pro Sample angegeben. Die Umrechnung zwischen Lernrate pro Minibatch und Lernrate pro Sample ist folgendermaßen definiert:

$$\text{lerningrate per sample } = \frac{\text{learning rate per MB}}{\text{minibatchSize}} \tag{5.1}$$

Diese Angaben sind nicht bindend, werden in CNTK jedoch besonders bei CNN empfohlen, da diese Parameter invariant zur Minibatch-Größe sind. Beispielsweise im Falle automatisierter Anpassungen der Minibatch-Größe ist dies von Bedeutung. Die Minibatch-Größe wurde auf 16 und die Lernrate (pro Sample) auf 0.00001 festgelegt. Im Gegensatz zu den anderen Netzen ist das Momentum 0.8. Auch die Dropout-Rate liegt bei 0.8. Mit 30 Epochen weist das CNN deutlich weniger Trainingsdurchläufe auf als die bisherigen Teilsysteme.

5.3.2. Recurrent-Netz

Um die Ergebnisse des CNN sequentiell verarbeiten zu können, werden die durch das CNN approximierten Merkmale der einzelnen, zueinander gehörenden Bilder in Sequenzen abgespeichert. Als Input für das RNN dienet der letzte Hidden-Layer des CNN, d.h. der Input-Layer des RNNs besteht aus 128 Neuronen. Der erste Hidden-Layer des RNN besitzt 192 Neuronen, um wie bei den dreidimensionalen Merkmalen den gesamten Merkmalsraum ausreichend

abzudecken. Dieser Layer ist der Recurrent-Layer des Netzes. Der zweite Hidden-Layer besteht wiederum aus 128 Neuronen. Auch in diesem Netz enthält der Output-Layer 35 Neuronen.

Netz-Parameter

Layer	Aktivierungsfunktion	# Neuronen
Input	-	128
Recurrent	ReLU	192
Hidden	ReLU	128
Output	-	35

SGD-Parameter

Minibatch size	32
Momentum per MB	0.9
Epochs	20
Learningrates per MB	0.01

Tabelle 5.4.: Hyperparameter des RNN für die Erkennung zweidimensionaler Merkmale und den zugehörigen SGD-Hyperparametern des Trainings.

Das Training dieses Netzes weist mit 20 Epochen die geringste Epochenzahl der Teilsysteme auf. Die Lernrate pro Mini-Batch liegt bei 0.01 und ist dadurch deutlich höher als die des RNN zur Erkennung dreidimensionaler Merkmale. Grund dafür ist die sehr effektive Vorverarbeitung durch das CNN, wie in Kapitel 6 gezeigt wird. Wie im Teilsystem in Abschnitt 5.2 beträgt auch hier die Minibatch-Größe 32.

5.4. Finale Klassifizierung

Wie bereits in Abbildung 5.2 dargestellt werden zur finalen Klassifizierung die Gewichtungen der jeweils letzten Hidden-Layer der bisher gezeigten Recurrent-Netze verwendet. Damit ergibt sich aus den drei Teilsystemen mit jeweils 128 Merkmalen ein Input-Vektor von 384 Werten.

Wie auch in den RNN der Teilsysteme ist hier nur der erste Hidden-Layer rekurrent und besitzt dabei 150 Neuronen. Der zweite Hidden-Layer ist nicht rekurrent und besteht aus 80 Neuronen. Der Output-Layer spiegelt mit seinen 35 Neuronen die endgültig klassifizierten Gesten wieder. Die Wahl der Anzahl der Neuronen pro Layer wurde hier so gewählt, dass eine annähernd gleichmäßige Reduktion der Merkmale bis zum Output stattfindet.

Jedes Teilsystem hat durch seine eigene Betrachtungsweise der Gesten sowohl Vorteile als auch Anfälligkeiten gegenüber den anderen. Die finale Klassifizierung hat den Zweck,

Netz-Parameter

Layer	Aktivierungsfunktion	# Neuronen
Input	-	384 (3*128)
Recurrent	ReLU	150
Hidden	ReLU	80
Output	-	35

SGD-Parameter

Minibatch size	64
Momentum per MB	0.9
Epochs	10
Learningrates per MB	0.05

Tabelle 5.5.: Hyperparameter des RNN für die finale Klassifizierung.

die Stärken der Teilsysteme zu sammeln und Schwächen von Teilsystemen durch andere ausgleichen zu können. Wie sich dieses finale Netz auf die Resultate der Teilsysteme auswirkt wird in den Tests im nächsten Kapitel dargestellt.

6. Tests

Um das erstellte Hybridsystem testen und evaluieren zu können, ist die Definition eines Test- und Trainingsets notwendig. Weiterhin wurde für die Aufnahme und Verarbeitung der Daten, sowie die Versuchsdurchführung ein User-Interface entwickelt. Dieses Kapitel gibt einen Überblick über solche Rahmenbedingungen und zeigt die Resultate der Versuche.

6.1. User-Interface

Die Aufnahme der Trainings- und Testdaten wurde mittels einem selbst erstellten User-Interface C# durchgeführt. Dieses wurd in der Programmiersprache C# geschrieben und bietet wie in Abbildung 6.1 zu sehen die Möglichkeit aktuelle Frames der jeweiligen Aufnahmekomponenten einzusehen und die Aufnahme zu steuern. Das Interface zeigt die gemessenen Distanzen der Joints und die Hand-Frames, sowohl für die Tiefendaten als auch für die Farbdaten. All diese Informationen werden in Echtzeit angezeigt. So kann jederzeit überprüft werden welche Daten und Frames aufgenommen werden.

Das Interface zeigt während der Aufnahme im Statusfenster (siehe Abbildung 6.1 unten rechts) welche Geste derzeit aufgenommen wird. Zusätzlich wird ein Zähler angezeigt, der aussagt wie viele Gesten dieses Typs noch aufzunehmen sind.

Neben dem Interface wurden weitere Tools in C# entwickelt. Diese vereinfachen die Zusammenfassung der Daten der Teilsysteme, die Verarbeitung der Test- und Trainingsdaten und die Evaluierung der Tests. Weiterhin wurde ein Tool entwickelt, welches die Konvertierung der Tiefendaten in OUR-CVFH umsetzt und den Status der Pipeline zur Erstellung von CTD-Dateien (Vgl. Abschnitt 4.2.1) anzeigt. Die Konvertierung erfolgt mittels einem C++ Wrapper, da die Point Cloud Library nur in C++ zu Verfügung steht.

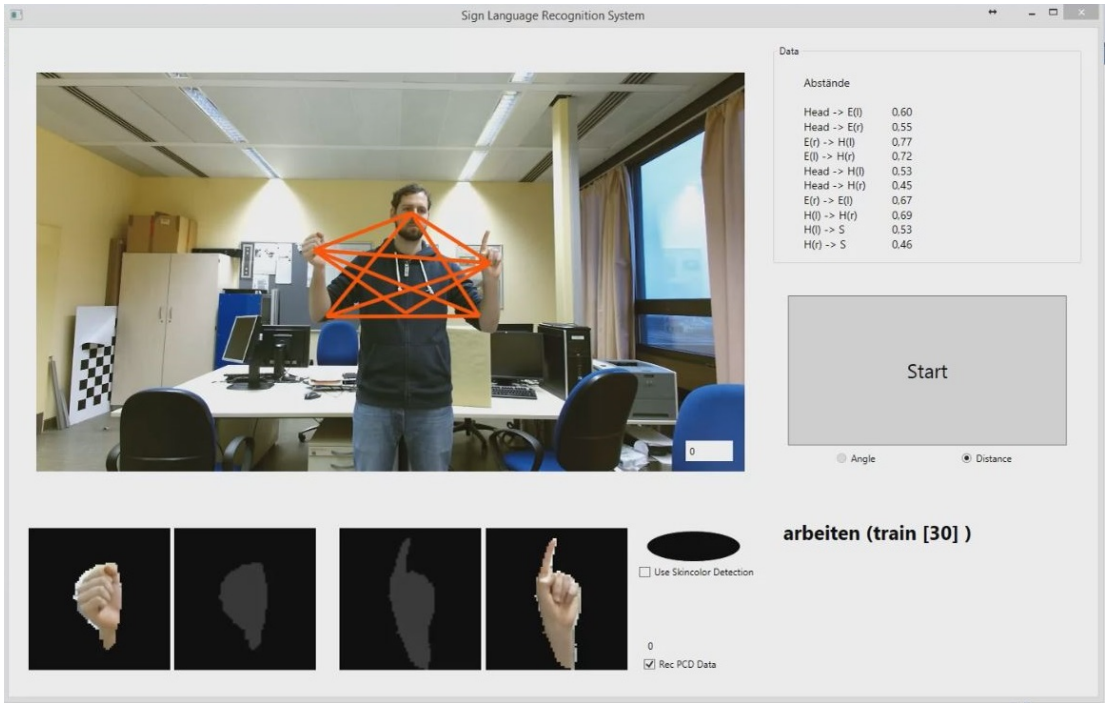


Abbildung 6.1.: Das User-Interface zur Aufnahme der Trainings- und Testdaten.

6.2. Trainings- und Testset

Das Ziel dieser Arbeit ist die Klassifizierung detailreicher Gesten. Dazu wurde ein Satz an Gesten der Gebärdensprache ausgewählt, der mehrere Anforderungen erfüllt. Diese Anforderungen umfassen Gesten, die

- viel Bewegung beinhalten,
- annähernd statisch sind,
- sich anderen Gesten sehr ähneln oder
- mit nur einer Hand gebärdet werden.

Diese Anforderungen bergen potentielle Probleme und sollen auf diese hin untersucht werden. Statische Gesten können beispielsweise bei Distanzen zu Verwechslungen führen. Gesten mit viel Bewegung wiederum bringen eine erhöhte Komplexität mit sich. Durch viele sich hierbei verändernde Parameter und gegebenenfalls hohe Geschwindigkeiten kann das System durchaus fehleranfällig werden. Der letzte Punkt wird betrachtet, da bei Rechtshändern

die linke Hand bei einhändigen Gesten grundsätzlich in der Grundposition verbleibt und sich dessen Daten über die komplette Sequenz nicht verändern.

arbeiten	alle	begleiten	besprechung	bruder
helfen	immer	was	schreiben	termin
verschieben	geschenk	besser*	danke*	idee*
sonne*	warum*	kinder*	ich*	du*
er/sie/es*	fahren	sehen*	film *	fussball
spielen	hoffen	du hast*	für dich*	mögen*
müssen*	wollen*	zusammen*	sagen*	zuhause

Tabelle 6.1.: Die Gesten der Testversuche im Überblick. Einhändige Gesten sind mit einem * gekennzeichnet.

Alle Gesten dieser Versuche sind in Tabelle 6.1 aufgeführt und in Anhang A.5 veranschaulicht. Die Grundstellung der Gesten sind nach unten gestreckte Arme, mit Abstand der Hand von der jeweiligen Hüfte von etwa zehn bis zwanzig Zentimeter. Das Starten und Anhalten der Aufnahme einer Sequenz wurde mittels selbstgebauter Fußtaste realisiert. Der Aufnahmestatus, sowie die aktuell aufzunehmende Geste war mit einer entsprechenden Anzeige im User-Interface gekennzeichnet. Bei der Aufnahme der Gesten betrug der Abstand der Kamera zur Person 160 Zentimeter. Die Kinect wurde in einer Höhe von 140 Zentimeter aufgestellt.

6.3. Resultate

Die Tests und Evaluation der Teilsysteme sowie des finalen Netzes wurden in drei Versuchen durchgeführt. Jeder dieser Versuche beinhaltete eine unterschiedliche Anzahl an Trainings- und Testdaten.

	# Trainingsdaten (# gesamt)	# Testdaten (# gesamt)
Versuch 1	50 (1750)	10 (350)
Versuch 2	100 (3500)	20 (700)
Versuch 3	150 (5250)	30 (1050)

Tabelle 6.2.: Übersicht über die Datenmengen der Versuche mit 35 zu klassifizierenden Gesten.

Überprüft wird in den unterschiedlichen Versuchen, wie sich die Anzahl der Trainingsdaten auf die Ergebnisse auswirkt. Dabei wird die Anzahl der richtig erkannten Gesten und die zuge-

hörigen Trefferquoten angegeben. Für die Tests wurde stets die gleiche Anzahl an Aufnahmen pro Geste verwendet, um eine Gleichverteilung für die Evaluation zu erhalten.

Da die ersten Frames einer Geste nach Start der Aufnahme nicht Teil der Geste selbst sind (d.h. Bewegung der Arme aus der Ausgangsposition zu Startposition der Geste) werden diese bei der finalen Klassifizierung aus der Zählung ausgelassen. Es wird angenommen, dass im Falle einer Echtzeit-Erkennung die Gesten direkt in einander übergehen und dabei die ersten drei Frames nicht zustande kämen. Dies hat zunächst den Vorteil, dass die aus irreführenden Gründen ausgelassenen Frames der zweidimensionalen Merkmale (siehe Abschnitt 4.2.2) keine Rolle mehr in der finalen Klassifizierung spielen. Weiterhin haben die Teilsysteme unterschiedliche Sequenzlängen, die durch das Auslassen der ersten Frames auf eine gemeinsame Zahl gebracht werden können.

Die gemessenen Latenzen, die in den Versuchen angegeben sind, beziehen sich darauf, wie viele Schritte in der Sequenz verstrichen sind, bis die Geste erstmals darin erkannt wird. Die Latenz berechnet sich durch die Anzahl der Frames. Jeder sechste Frame wird aufgenommen, was bei einer Framerate von 30 Frames pro Sekunde bedeutet, dass 200 Millisekunden zwischen den aufgenommenen Frames einer Sequenz liegen. Diese Zahl wurde so gewählt, dass eine gute Balance zwischen Performanz und einer möglichst geringen Verarbeitungszeit besteht. Das CNN besitzt keine Angaben über Latenzen, da hier jeder einzelne Frame klassifiziert wurde.

Als erkannte Klassifizierung wird eine Sequenz dann angesehen, wenn mindestens die letzte Ausgabe der Output-Sequenz des jeweiligen Netzes dem korrekten Label entspricht.

6.3.1. Versuch 1

Der erste Versuch wurde mit 50 Trainings- und 10 Testdaten durchgeführt. Eine solch geringe Menge an Daten macht es neuronalen Netzen bei komplexen Problemen oft schwer eine ausreichende Klassifikation zu erzielen. Dies spiegelt sich auch in den Ergebnissen dieses Versuchs wider.

Das stabilste Teilverfahren ist hier die Klassifizierung der Distanzen, welches gleichzeitig die geringste Komplexität unter den Teilsystemen aufweist. Trotz der geringen Menge an Trainingsdaten wurde dennoch eine Trefferquote von 84% erreicht. Die restlichen Teilsysteme weisen etwas geringere Trefferquoten auf.

In der finalen Klassifizierung wurde gegenüber den Teilsystemen keine Verbesserung erzielt. Im Vergleich zum Teilsystem für die Erkennung der Distanzen weist sie ein deutlich verschlechtertes Ergebnis auf.

Netztypen	Treffer (350)	Quote	∅ Latenz (min. # frames)
Distanz	294	84,00%	457,8 ms (3)
Tiefen	269	76,75%	401,4 ms (3)
Farbe (CNN)	772 / 1017	75,91%	-
Farbe (RNN)	255	73,06%	243,6 ms (2)
Final	263	75,36%	239,2 ms (2)

Tabelle 6.3.: Testergebnisse des ersten Versuchs mit 50 Trainingsdaten und 10 Testdaten pro Geste.

Nicht erkannt wurden bei den Distanzen vor allem die Gesten “danke”, “müssen” und “zuhause”. Bei der Klassifizierung der Tiefen-Merkmale wurden besonders die Worte “besser”, “ich”, “du”, “er”, “für”, “mögen” und “müssen” falsch klassifiziert. Fehlerbehaftet waren im Teilsystem der Farberkennung insbesondere die Gesten für “danke”, “warum”, “ich”, “du”, “wollen” und “zusammen”. Das finale System zeigte Schwächen bei der Erkennung von “besser”, “ich”, “du”, “für” und “müssen”. Bei all den genannten Gesten schlugen mindestens sechs der zehn Klassifizierungen fehl. Alle Klassifizierungen dieses Versuchs sind in den Konfusions-Matrizen in Abbildung A.3 dargestellt.

Die korrekt klassifizierten Sequenzen wurden in allen Teilsystemen spätestens nach dem dritten Frame richtig erkannt. Das finale System erreichte eine Erkennungszeit von durchschnittlich 239, 2 Millisekunden, was einer Klassifizierung bereits im zweiten Frame entspricht. Grundsätzlich ist zu erkennen, dass sich mit dem Voranschreiten der Verarbeitungskette die Latenz verringert.

6.3.2. Versuch 2

Im zweiten Versuch, mit erhöhter Anzahl an Trainings- und Testdaten, haben sich die Trefferquoten deutlich verbessert.

Die Performanz des Sliding-Window für die Erkennung der Distanzen hat sich gegenüber des ersten Versuchs nur unwesentlich gesteigert. Bei Betrachtung der anderen Teilsysteme zeichnete sich allerdings eine signifikante Verbesserung der Trefferquoten ab. Die Erfolgsrate steigerte sich bei der Klassifizierung der Tiefendaten um mehr als 10% gegenüber Versuch 1. Die Erkennung der Gesten mit den Farbdaten ist beim Recurrent Netz um über 20% gestiegen, bei einer Verdopplung der Trainingsdaten von 50 auf 100.

Netztypen	Treffer (700)	Quote	∅ Latenz (# frames)
Distanz	593	84,71%	470,4 ms (3)
Tiefen	610	87,14%	451,8 ms (3)
Farbe (CNN)	2798 / 3243	86,28%	-
Farbe (RNN)	666	95,14%	255,6 ms (2)
Final	677	96,71%	207,4 ms (2)

Tabelle 6.4.: Testergebnisse des zweiten Versuchs mit 100 Trainingsdaten und 20 Testdaten pro Geste.

Mit $95,14\%$ weist das Teilsystem der Erkennung zweidimensionaler Farbbilder eindeutig das beste Ergebnis auf. Im Gegensatz zum ersten Versuch besitzt die finale Klassifizierung das Resultat mit der höchsten Performanz. Verglichen mit dem besseren Teilsystemen erhöhte sich die Erfolgsrate von $96,71\%$ hier nur leicht. Erkannt wurden jedoch elf der insgesamt 700 Gesten mehr.

Die Klassifizierungen dieses Versuchs sind in den Konfusions-Matrizen in Abbildung A.4 dargestellt. Bei den Distanzen wurden besonders die Gesten “alle”, “spielen”, “müssen” und “sagen” nicht erkannt. Die Klassifizierung der Tiefen-Merkmale zeigte vor allem bei den Worten “er” und “für” falsche Werte. Im Teilsystem der Farberkennung war lediglich die Geste “zusammen” fehlerbehaftet. Das finale System hatte bei keiner der Gesten auffallende Schwächen, da die 23 nicht erkannten Sequenzen über mehrere Gesten verteilt waren. Bei allen genannten Gesten waren mindestens elf der 20 Klassifizierungen fehlgeschlagen.

Die Latenzen dieses Versuchs waren annähernd unverändert gegenüber des ersten Testversuchs. Über das gesamte System hinweg wurden hier ebenfalls mindestens drei Frames, bei der finalen Erkennung nur mindestens zwei Frames benötigt.

6.3.3. Versuch 3

Der letzte Versuch hat die besten Resultate erzielt. Die Teilsysteme schneiden hier allesamt mit hoher Trefferquote ab.

Gegenüber dem vorhergehenden Versuch steigert sich die Trefferquote des Sliding-Window Verfahrens um 13% auf $97,90\%$. Auch das neuronale Recurrent-Netz für die Tiefendaten erhöht die Erfolgsrate auf $95,81\%$. Die Quote des Teilsystems für Klassifizierung der Farb-Bilder steigt um einen knappen Prozentpunkt ($96,10\%$).

Netztypen	Treffer (1050)	Quote	∅ Latenz (min. # frames)
Distanz	1028	97,90%	397,8 ms (≥ 2)
Tiefen	1006	95,81%	383,4 ms (≥ 2)
Farbe (CNN)	3962 / 4239	93,46%	-
Farbe	1009	96,10%	215,4 ms (≥ 2)
Final	1040	99,05%	207,2 ms (≥ 2)

Tabelle 6.5.: Testergebnisse des dritten Versuchs mit 150 Trainingsdaten und 30 Testdaten pro Geste.

Das finale System erwirkt in diesem Versuch eine signifikante Erhöhung der Performanz. Auch wenn die Ergebnisse der Teilsysteme bereits sehr robuste Resultate erzielen, steigt die Klassifizierung des Gesamtsystems auf $99,05\%$. Damit wird die Trefferquote des besten Teilsystems (Distanzen) nochmals um $1,15\%$ angehoben. Die Erkennung durch das Gesamtsystem liegt damit nur bei zehn der 1050 Gesten falsch.

Die Erkennung der Distanzen zeigt nur wenige Schwächen. Die häufigsten Fehler traten bei den Gesten “du” und “spielen” auf. Probleme bei der Klassifizierung der Tiefen-Merkmale waren die Gesten “besser”, “mögen” und “müssen” auf. Teilweise Fehlerbehaftet waren im Teilsystem der Farberkennung die Gesten “ich”, “du”, “sehen” und “zusammen”. Abbildung A.5 zeigt, dass das finale System mit insgesamt nur zehn falsch erkannten Gesten kaum Fehler aufweist. Lediglich die Worte “besser” (1 nicht erkannt), “ich” (3 nicht erkannt), “müssen” (1 nicht erkannt) und “sagen” (5 nicht erkannt) wurden falsch klassifiziert.

Die Latenzen der Gestenerkennung haben sich hier bei den Teilsystemen für Distanzen und Tiefen-Merkmalen verbessert. Es benötigte im Durchschnitt nur noch mindestens zwei Frames für die Erkennung. Bei der Klassifizierung der Farb-Merkmale ist dieser Wert gleich geblieben. Das finale Netz benötigte weiterhin mindestens zwei Frames für die Erkennung, ist jedoch mit 207,2 Millisekunden durchschnittlicher Erkennungszeit das schnellste Verfahren aller Testversuche.

6.4. Trainingsverlauf

Das Training der einzelnen Teilnetze, sowie des finalen Netzes verlief durchweg zufriedenstellend. Grundsätzlich zeigt sich, je mehr Vorverarbeitung geleistet wurde, desto schneller und effizienter war der Trainingsverlauf (siehe Abbildung 6.2). Diese Vorverarbeitung bezieht sich nicht nur auf die in Kapitel 4 gezeigten Vorgehensweisen, sondern auch auf die Vorarbeit

durch die Teilsysteme. Die Trainingszeiten in diesem Abschnitt beziehen sich auf den in Abschnitt 6.3.3 dargelegten dritten Versuch. Dieser wird als repräsentativer Versuch verwendet, da dort eine ausreichende Anzahl an Trainingsdaten für die Evaluierung des Hybridsystems vorhanden ist.

Das Training wurde beendet, wenn der ErrorPrediction-Wert auf einem Wert stagnierte, oder unter $0,0001$ fiel. Die Zeiten die das Training benötigte sind unabhängig von den Epochen. Beispielsweise beansprucht das Faltungsnetz trotz der geringen Epochenanzahl von 30 dennoch die meiste Trainingszeit auf der GPU, wie in Tabelle 6.6 zu sehen.

Um die Bedeutung des Trainings mit der GPU zu untersuchen, wurden die Netze zunächst mit der CPU und anschließend auf dem Grafikprozessor trainiert. Der Rechner, auf dem die Trainings statt fanden war ausgestattet mit einer Nvidia GTX 960 GPU, einer Intel i7-3770 CPU mit vier Kernen mit je 3,40 GHz und 8 Gigabyte RAM. Als Betriebssystem wurde Windows 8.1 in der 64-Bit Version verwendet, da die Kinect 2 dies voraussetzt.

Das Training mit GPU und CPU unterscheidet sich beim Sliding-Window Verfahren lediglich um ein paar Sekunden. Erst bei Recurrent-Netzen wird der Nutzen des Trainings mittels GPU sichtbar. Trainieren des RNN für Tiefendaten mit der CPU benötigt mehr als das doppelte der Zeit. Besonders kritisch ist der Unterschied beim Faltungsnetz, bei dem das Training mit der CPU mehr als das 17-fache beträgt. Je "tiefer" das Netz wird, desto mehr lohnt sich daher das Training auf der GPU. Bei MLP geringer Größe ist von der Verwendung der GPU jedoch abzuraten. Hier muss mehr Zeit für den Transfer der Daten zwischen CPU und GPU aufgewendet werden, als für das Training selbst. Oft wird dabei sogar mehr Zeit verwendet, als beim Training direkt mit der CPU.

Trainingsepochen und -Zeiten

	Epochen	Trainingszeit GPU	Trainingszeit CPU
SW / Distance	1500	4684,96 s (≈ 78 min)	4678,55 s (≈ 78 min)
RNN / 3D	300	2643,29 s (≈ 44 min)	5462,88 s (≈ 91 min)
CNN / 2D	30	6693,88 s (≈ 112 min)	124741,8 s (≈ 34 h 39 min)
RNN / 2D	20	39,73 s	39,53 s
Finalsystem	10	8,29 s	11,43 s

Tabelle 6.6.: Übersicht über Trainingszeiten und Epochen der Teilsysteme des dritten Versuchs.

Wie bereits in Abschnitt 5.1 gezeigt, wurden beim trainieren des Netzes der Distanzen mit 1500 die meisten Epochen benötigt. Trotz der durchgehend absteigenden Error-Prediction und Kreuzentropie sind die Werte nur sehr langsam gesunken. Schneller und effizienter verlief das

Training des Netzes zur Erkennung der Tiefen-Daten. Die Anzahl der Epochen betrug hierbei lediglich 300. Bereits nach 29 Epochen sank hier die Error-Prediction unter 0.1. Das effizienteste Teilsystem im Bezug auf das Training war die Klassifizierung der Farb-Frames. Aufgrund des bereits guten Trainingsverlaufs des CNNs zeigte auch das Training des RNNs für Farbmerkmale nach wenigen Epochen sehr niedrige Werte der Error-Prediction. Das Training des Recurrent-Netzes der Farbmerkmale wurde bereits nach 20 Epochen beendet. Kreuz-Entropie, sowie Error-Prediction sanken bereits nach wenigen Trainings-Durchläufen auf sehr niedrige Werte.

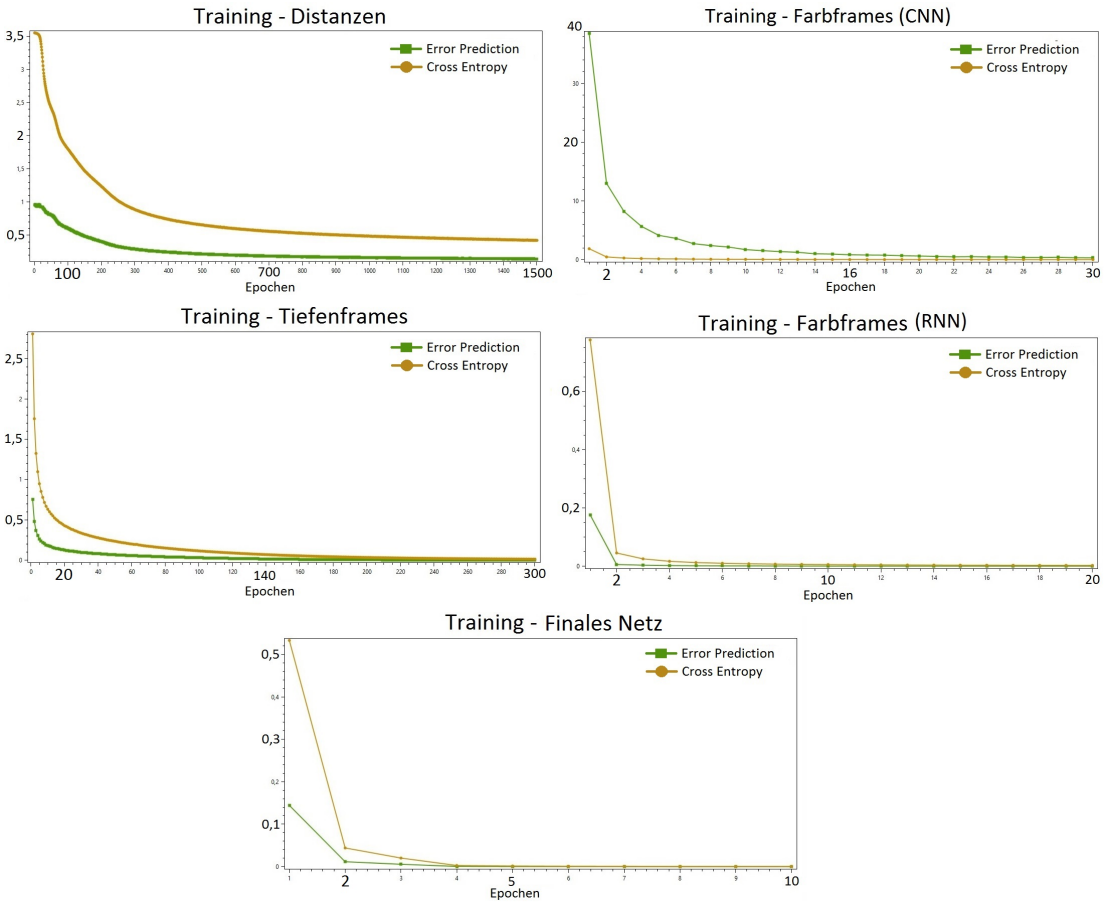


Abbildung 6.2.: Trainingsverläufe der einzelnen Systeme.

Die Error-Prediction des CNN ist deutlich höher als die der anderen Netze, da hier keine Softmax-Funktion verwendet wurde. Dies ermöglicht durch uneingeschränkte Fehlerwerte eine bessere Approximation für die Weiterverarbeitung durch das RNN.

In allen Versuchen wies das finale Netz die geringste Trainingszeit auf. Wie in Abbildung 6.2 zu sehen sank die Error-Prediction im finalen System bereits nach der ersten Epoche unter 0.2 und nach der dritten Epoche sogar unter 0.1.

7. Diskussion

Die Evaluation der Versuche ergibt, dass der Zusammenschluss der Einzelsysteme durch das finale Netz das Resultat verbessern kann. Die Annahme, ein hybrides System aus neuronalen Netzen ermöglicht eine Verbesserung der Ergebnisse gegenüber einzelner Komponenten, hat sich daher bei ausreichender Anzahl an Trainingsdaten für dieses System bestätigt. Besonders die Ergebnisse des letzten Versuchs weisen eine robuste Klassifizierung auf. $99,05\%$ sind eine beachtliche Erkennungsrate bei solch detailreichen Gesten. Zwar werden nur 35 unterschiedliche Gesten klassifiziert, dennoch zeigt besonders der letzte Versuch das Potential dieses hybriden Ansatzes.

Grundsätzlich lässt sich sagen, dass die finale Klassifizierung nur zuverlässig sein kann, wenn es die Teilsysteme selbst sind. Allerdings ist in Kapitel 6 und den Konfusions-Matrizen in Anhang A.6 zu erkennen, dass Schwächen der Teilsysteme durch eine finale Klassifizierung großteils verteilt und somit minimiert werden können. Zwar wird im ersten Versuch keine Verbesserung erzielt, jedoch werden im zweiten und dritten Versuch die Ergebnisse der Teilsysteme jeweils um mehr als einen Prozentpunkt übertroffen. Die polarisierten Schwächen bei der Erkennung einzelner Gesten wird großteils aufgehoben.

Mit $99,05\%$ zeigt das finale Klassifizierungssystem im letzten Versuch eine enorme Trefferquote. Auch die Teilsysteme zeigen eine sehr hohe Erkennungsrate. Bei diesen lagen die gravierenden Probleme häufig bei bestimmten Gesten, die sich sehr ähnlich sind. In der finalen Klassifizierung waren die wenigen Fehler über mehrere Gesten gleichmäßig verteilt.

7.1. Vergleich der Teilsysteme

In Abbildung 7.1 ist zu erkennen, dass die Erfolgsquoten der Teilsysteme bei den Versuchen mit wachsender Anzahl an Trainingsdaten sehr unterschiedlich ansteigen. Beispielsweise schneidet die Klassifizierung der Distanzen zunächst als bestes Teilsystem und sogar als bestes System insgesamt ab. Im zweiten Versuch zeigt es kaum Verbesserung und liegt im Ergebnis deutlich hinter den anderen Verfahren.

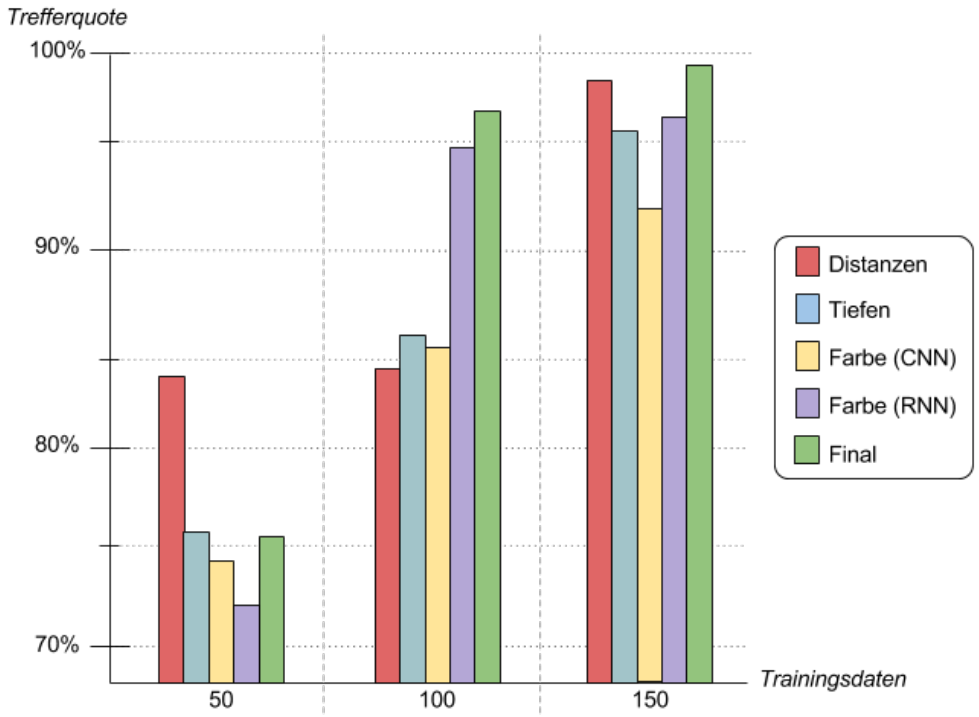


Abbildung 7.1.: Übersicht über die Trefferquoten der Versuche. Die Verbindungen stellen die Steigerung der Trefferquote der unterschiedlichen Anzahl Trainingsdaten dar.

In Weschta (2016b) war das Sliding-Window Verfahren das stabilsten Netz. In den Versuchen dieser Arbeit weist das Verfahren allerdings Mängel auf. Die Distanzen sind wichtig für eine Erkennung von Gebärdensprache (siehe Kapitel 4). Alleinstehend jedoch kommt es bei Gesten wie beispielsweise “danke” und “sprechen” zur Verwechslung. Dies ist der hohen Ähnlichkeit der Bewegungsabläufe dieser beiden Gesten geschuldet.

Das Verfahren zur Erkennung der Handoberfläche durch Punktwolken zeigt eine hohe Trefferquote bei ausreichender Anzahl der Trainingsdaten. Die Neutralität gegenüber Translation und Rotation der Person zur Kamera ist hierbei ein Vorteil. Allerdings ist die in Abschnitt 7.4 gezeigte, hohe Dauer der Vorverarbeitung und der Konvertierung der Tiefendaten in OUR-CVFH ein enormer Nachteil.

Die Erkennung von sequentiellen Farb-Frames beansprucht deutlich weniger Zeit für die Vorverarbeitung. Die Tests zeigen sehr erfolgreiche Trefferquoten. Bereits im zweiten Versuch mit nur 100 Trainingsdaten pro Geste liegt das Ergebnis nur einen Prozentpunkt unter der finalen Klassifizierung.

Das finale System zeigte sich im ersten Versuch schlechter als seine Teilsysteme. Grund dafür sind die unzureichenden Trainingsdaten, was sich auch in den Teilsystemen selbst widerspiegelt. Jedoch wurden die Ergebnisse immer robuster und die Verteilung der wenigen Fehler immer größer, je mehr Trainingsdaten verwendet wurden. Diese Fehler sind auf andere Ursachen zurückzuführen.

7.2. Fehlerquellen

Grundsätzliche Probleme bei der Gestenerkennung sind die Rotation und Translation der Person zur Kamera. Durch die dreidimensionale Berechnung der Distanzen und die Beschreibung der Tiefen-Merkmale durch das OUR-CVFH, sind diese Teil-Systeme davon nicht betroffen. Die Erkennung der Farb-Frames jedoch ist diesbezüglich durch die Zweidimensionalität weiter ein Problem. Es lässt sich jedoch sagen, dass die Unterscheide in den aufgenommenen Bildern sehr gering sind, da die Person in der Regel zentral vor der Kinect steht und die Rotation, sowie Translation dabei verschwindend gering ist. Darüber hinaus ließen sich diese Unterschiede durch eine erhöhte Anzahl an Trainingsdaten durch das CNN kompensieren.

Weiterhin birgt das Teilsystem für die Klassifizierung der Farb-Merkmale das Problem der Hautfarbe. Auch dies lässt sich aber durch zusätzliches Trainingsmaterial im Faltungsnetz dieses Teilsystems antrainieren. Alternativ dazu wäre es möglich, die zweidimensionalen Merkmale in Grauwert-Bilder zu speichern oder in einen anderen Farbraum umzuwandeln, um die Hautfarbe annähernd zu egalisieren. Inwiefern sich der Wechsel oder die Reduktion des Farbraumes (beispielsweise Grau-Werte) auf die Resultate auswirkt, müsste in weiteren Tests evaluiert werden. Das Problem der Hautfarbe tritt in den anderen Teilsystemen nicht auf, da hier ausschließlich mit Tiefen-Daten gearbeitet wird.

Trotz der robusten Erkennungsrate des letzten Versuchs sind im Voraus und auch während der Versuche einige potentielle Fehlerquellen identifiziert worden. Diese Fehlerquellen sind nicht nur auf technische Faktoren der Kinect, sondern auch auf optische Faktoren der Gesten, sowie deren Verarbeitung zurückzuführen.

7.2.1. Fehlerquelle Kinect

Deutlich war bei der Aufnahme der Gesten zu erkennen, dass ein Grund der fehlerhaften Klassifizierungen auf Seiten der Kinect lag. Wie in Abbildung 7.2 dargestellt, hatte dies mehrere Gründe.

Zunächst hat die Kinect Probleme schnelle Bewegungen in guter Qualität aufzunehmen. Dabei verschwimmen nicht nur die Farbbilder, sondern auch die Synchronisation mit dem Bodytracking und somit auch der Tiefendaten ist dann oft fehlerhaft.

Weiterhin traten beim Bodytracking besonders bei der Aufnahme bestimmter Gesten Probleme auf. Bei der Geste “für” beispielsweise bilden Daumen und Zeigefinger einen Ring. Die Koordinate des Hand-Joint liegt dann aufgrund des heuristischen Algorithmus (siehe Abschnitt 2.2.2) häufig in genau diesem Ring, was zu einer falschen Tiefeninformation führt. Bei Ausrichtung der Handkante zur Kamera lag die Koordinate des Hand-Joints ebenfalls nicht exakt auf der Hand, was ähnlich fehlerhafte Tiefen-Werte lieferte. Diese falschen Tiefen-Daten verursachten eine völlig veränderte und falsche Segmentierung der Hände und dadurch auch fehlerbehafteten Input für die Teilsysteme.

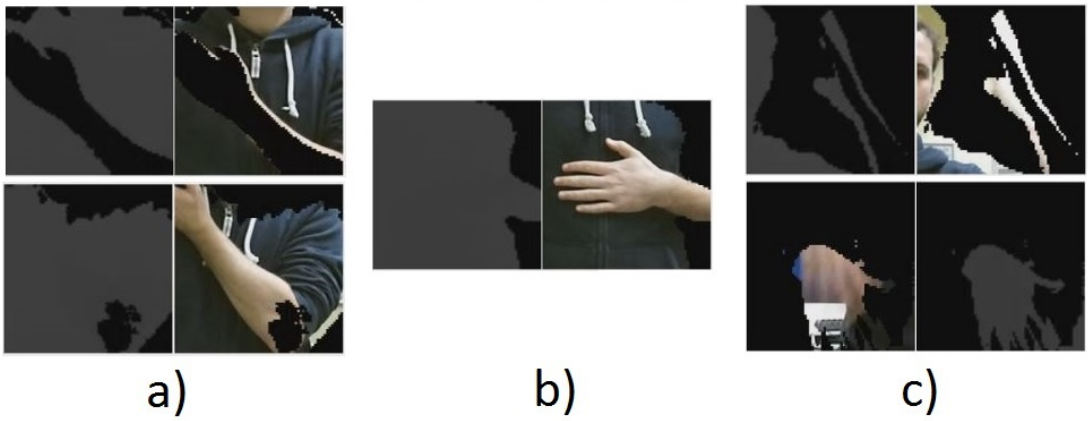


Abbildung 7.2.: Veranschaulichung der Fehlerquellen bei der Aufnahme der Gesten mit der Kinect. a) zeigt die Aufnahme eines Fehlers beim Body-Tracking. b) stellt dar, wie durch die Segmentierung der Körper mitaufgenommen wird. c) zeigt die Aufnahme von Gesten mit hoher Geschwindigkeit.

Ein weiteres potentielles Problem war die Aufnahme der Frames bei Gesten direkt am Körper. Beispielsweise wird bei der Geste “mögen” die Hand über die Brust gestrichen. Oft wurde hier der Hand-Joint von der Kinect nicht korrekt erkannt und lieferte falsche Koordinaten.

Vor allem bei der Aufnahme der Daten für den ersten Versuch traten solche Fehler sehr häufig auf. Daher wurde in den Versuchen 2 und 3 darauf geachtet, diese Fehler für die Trainingsdaten durch langsamere und kontrollierte Bewegungen die Aufnahmen fehlerfrei zu gestalten. Da die Trainingsdaten der Versuche aufeinander aufbauen, also nicht immer wieder neu aufgenommen wurden, sind diese “fehlerhaften” Trainingsdaten im letzten Versuch weiterhin miteinbezogen worden. Trotz dieser Tatsache wurden dabei sehr gute Ergebnisse erzielt.

Das Weglassen der ersten Frames beim Zusammenschluss der Teilsysteme (Vgl. Abschnitt 6.3) ist vernachlässigbar. Eine Geste dieser Versuche sagt durch die Grundstellung in diesen ersten Frames nichts aus, das heißt es kann daraus jede Geste entstehen. Daher folgt sowohl beim Weglassen der drei ersten Frames zur Klassifizierung der Farbbilder als auch beim Zusammenschluss der Teilsysteme kein Nachteil in Bezug auf Latenz oder Erkennungszeit.

7.2.2. Fehlerquelle Gesten

Bei der Auswahl der Gesten wurde für Evaluationszwecke darauf geachtet, Gesten auszusuchen die anderen sehr ähnlich sind. Diese Ähnlichkeit bezieht sich sowohl auf die Distanzen, als auch die Handstellung in zwei und dreidimensionalen Merkmalen. Die Versuche betrachteten die Teilsysteme und deren Resultate bei Gesten mit den potentiellen Problemen. Im Gegensatz zur Kinect stellen diese Probleme eine natürliche Fehlerquelle dar. Es wurde untersucht, ob ein Zusammenschluss dieser Teilsysteme in einem finalen neuronalen Netz solche potentiellen Problemquellen egalisieren kann.

Distanzen

Betrachtet man lediglich die Veränderung der Distanzen, wiederholen sich Abläufe bei Gesten der Gebärdensprache oft. Bei Distanzen ähneln sich besonders die gebärdeten Worte in den Gruppen

- “du”, “zusammen” und “du hast”,
- “was” und “zuhause”, sowie
- “danke”, “sagen” und “sehen”.

Die Klassifizierung der Distanzen verursachte in den Versuchen bei mehreren Gesten Probleme. Bei Versuch 1 wurden die Geste “danke” erwartungsgemäß sehr oft als “sagen”, manchmal auch als “sehen” falsch erkannt. “Müssen” wurde mit unterschiedlichen Gesten verwechselt. “zuhause” wurde oft nur wegen der letzten Sequenzelemente fehlerhaft klassifiziert, was dadurch geschehen kann, dass am Ende der Aufnahme einer Geste die Hände einer dieser Geste fremde Bewegung machen.

In Versuch 2 wurde offensichtlich, dass die Anzahl der Trainingsdaten weiterhin unzureichend ist. So wurden die Gesten “alle” oft mit “bruder” verwechselt und “müssen” als “kinder” klassifiziert. Grund dafür ist, dass diese Gesten alle in der gleichen Position enden. “Sagen” wur-

de hier meist mit “danke” vertauscht, was die gegenteilige Verwechslung zum ersten Versuch darstellt.

Im letzten Versuch war deutlich zu sehen, dass es seltener zu Verwechslungen kam. Lediglich bei den Gesten “du” (klassifiziert als “zusammen”) und “spielen” (klassifiziert als “immer”) kam es zu Fehlentscheidungen. Dies ist nachvollziehbar, da diese Gesten sich auch nur gering unterscheiden und aufgrund der minimalen Bewegungen während der Gesten schwer auseinander zu halten sind.

Tiefen-Frames

Die Gebärdensprache legt das Hauptaugenmerk auf die Handstellung und Fingerpositionen. Bei der Betrachtung dieser Merkmale fielen folgende sich ähnelnde Gestenpaare auf:

- “für und “besser”,
- “was” und “geschenk”, sowie
- “ich”, “du”, “er”, “müssen” und “sagen”.

Bei diesem Ansätzen sind Fehler vor allem auf Seiten der Kinect aufgetreten. Daher sind gerade in Versuch 1 und Versuch 2 Fehlklassifizierungen aufgetreten, die nicht mit Ähnlichkeit zwischen Gesten zu erklären sind.

Der Unterschied der oben genannten Gesten-Paare liegt lediglich in Abweichungen der Orientierung oder Rotation der Hand. Bei der Erkennung der Tiefendaten traten in Versuch 1 große Schwierigkeiten auf. Die Gesten “müssen”, “du”, “ich und “er” wurden aufgrund ihrer Ähnlichkeit untereinander häufig verwechselt. Bei solch geringer Anzahl an Trainingsdaten wurden auch Gesten wie beispielsweise “für” und “sonne” verwechselt, obwohl diese nur wenig gemeinsame Eigenschaften haben. Versuch 2 zeigt verbesserte Ergebnisse, jedoch treten bei der Erkennung von “er” oder “für” weiterhin Verwechslungen mit den unterschiedlichsten Gesten auf. Dies weist besonders auf die Empfindlichkeit des Teilsystems zur Erkennung der Tiefendaten hin. Im letzten Versuch sind erneut Verwechslungen der Gesten “du”, “er” und “müssen” am häufigsten aufgetreten.

Farb-Frames

Die Abweichungen in Rotation und Ausrichtung der Handstellung wurden durch das CNN erkannt und in den Versuchen mit ansteigender Anzahl der Trainingsdaten auch zuverlässiger klassifiziert. Bereits das CNN erreichte in Versuch 3 eine Trefferquote von 93,46%. Dennoch

gab es Gesten, bei denen im Teilsystem für die Erkennung der Farb-Frames Probleme auftraten. Mitunter sind diese Probleme durch die Fehler der Kinect entstanden. Darüber hinaus traten Fehler auch dadurch auf, dass sich bei manchen Gesten die einzelnen, zu klassifizierenden Farb-Bilder innerhalb einer Sequenz stark unterscheiden. Beispiele dafür sind Gesten wie “alle”, “müssen” oder “zusammen”.

Vor allem die Gesten “ich” und “du” wurden in Versuch 1 häufig miteinander verwechselt. Diese Gesten sind aufgrund der Zweidimensionalität nur schwer zu unterscheiden. Oft kam es zur einer falschen Klassifizierung von “sehen” anstatt “danke”, da deren unterschiedliche Details im ersten Versuch nicht erfasst wurden. Der zweite Versuch brachte deutlich bessere Ergebnisse hervor. Gegenüber dem ersten Versuch wurden Details etwas besser erkannt, jedoch trat hier Konfusion bei Erkennung der Geste “zusammen” auf. Die in vorigem Abschnitt erwähnen, erwarteten Verwechslungen innerhalb der Gestenpaare traten kaum auf. In Versuch 3 wurden die Gesten “ich” und “du” oftmals mit “zusammen” verwechselt. Auch hier, sowie bei der Konfusion der Geste “sehen” und “danke” waren erneut die Details ausschlaggebend, die das System nicht trennen konnte.

7.2.3. Segmentierung

Bei der Segmentierung wurden vor allem bei Gesten, die dem Körper nah sind, die Brust oder der Kopf mit aufgenommen (siehe Abbildung 7.2). Die dadurch erwarteten Fehler blieben größtenteils aus und stellten trotz deren Mitaufnahme wenig Probleme dar. Diese Oberflächen sind bei der Verarbeitung der Tiefendaten mit OUR-CVFH als Teil der Oberfläche mit eingeflossen. Die Erkennung der Farb-Frames waren gegenüber diesem Problem ebenfalls wenig empfindlich, da die aufgenommenen Teile des Körpers auch hier als Input für das CNN-Teilsystem dienten. Die Aufgabe der Segmentierung, den Hintergrund auszublenden war also ausreichend.

Als Segmentierungs-Distanz wurde 60 als Schwellwert gesetzt. Dieser Wert wurde dahingehend gewählt, dass bei einer Ausrichtung der Finger zur Kamera diese noch innerhalb der Segmentierung liegen, ohne zu viel von der Umgebung mit aufzunehmen.

Alternative Segmentierungs-Algorithmen, wie beispielsweise das in Tompson u. a. (2014) vorgestellte Verfahren, wären für eine Stabilisierung des in dieser Arbeit vorgestellten Ansatzes besser geeignet. Jedoch ist bei solchen Verfahren wesentlich mehr Rechenaufwand nötig. Darüber hinaus müssen weitere Frameworks und Algorithmen implementiert werden, die die Verarbeitungszeit zwischen einzelnen Frames verlängert. Die in diesem Verfahren

implementierte Segmentierung wurde für die Untersuchung eines neuronalen Hybridsystems daher als ausreichend erachtet.

7.3. Trainingszeiten

Die in Tabelle 6.6 aufgelisteten Trainingszeiten zeigen eindeutig, dass eine Vorverarbeitung von Daten und Merkmalen eine enorme Reduktion der Trainingszeit der nachfolgenden Netze bewirkt. Besonders deutlich wird dies bei der Betrachtung der Trainingsdauer des finalen Systems. Die Vorverarbeitung erwirkt eine Vereinfachung der Merkmale, sodass lediglich zehn Epochen in $8,29$ Sekunden trainiert werden mussten.

Auffällig ist, dass besonders das Training des CNN von einer Verwendung der GPU profitiert, während das MLP mit Sliding-Window keinerlei Vorteile daraus ziehen kann. Auch die Netze die bereits durch die Teilsysteme verarbeitete Merkmale als Input erhalten (2D-RNN und finales Netz) zeigen keine Verbesserung. Dies ist durch die geringe Epochenzahl des jeweiligen Trainings begründet.

Die Verwendung der Rectified Linear Unit (ReLU) als Aktivierungsfunktion brachte mehrere Vorteile mit sich. Die dadurch hinfällige Normierung vereinfachte die Interpretation von aufgenommenen Daten und Werten erheblich. Darüber hinaus waren die Trainingszeiten gegenüber den Werten aus Weschta (2016b) deutlich verbessert. Die dort verwendete Normierung machte herausstechende Werte in den Daten und dadurch auch minimale Veränderungen der Distanzen zunichte. Dies wurde in diesem System durch ReLU umgangen, was zu einer merklichen Beschleunigung des Trainings führte.

7.4. Laufzeiten

Die Latenzen sind durchaus beachtenswert, da nur zwei bis drei Frames zur Erkennung in allen Teilsystemen und dem Gesamtsystem benötigt werden. Die Aufnahme jedes sechsten Bildes ist sehr viel in einem System zur Echtzeit-Erkennung, da dies 200 Millisekunden pro Frame entspricht. Mit einer Latenz von drei Frames führt dies schnell bis zu 600 Millisekunden Verzögerung pro Geste. Jedoch betrug die Aufnahmedauer einer Geste durchschnittlich eine Sekunde, womit die Aufnahme jedes sechsten Frames ausreichend ist. Mehr Frames pro Sekunde führen möglicherweise zu einer verringerten Erkennungszeit, jedoch muss dabei darauf geachtet werden, dass die benötigte Zeit der Verarbeitung der Daten die Zeit der Framerate nicht überschreitet.

Die benötigte Zeit der Verarbeitung, das heißt von der Eingabe der Merkmale in das System bis zur Ausgabe der Klassifizierung in den einzelnen Systemen ergibt sich aus mehreren Anteilen. Dabei spielt die benötigte Zeit der Vorverarbeitung, sowie die Laufzeit der Klassifizierung eine Rolle. Die in Tabelle 7.1 dargestellten Gesamtlaufzeiten ergeben sich durch die Zeit der Vorverarbeitung einer Geste, summiert mit deren Klassifizierungsdauer. Die Zeit die es benötigt in CNTK das Netz zur Klassifizierung zu laden wurden bei der Gesamtdauer mit einbezogen, bei der Dauer pro Geste nicht.

Die Sequenz einer Geste besteht etwa aus sieben bis acht Frames. Diese Anzahl variiert mit der Ausführungsgeschwindigkeit der Geste sowie der Ausführungsdauer der Geste selbst. Es wurden für diese Laufzeitberechnungen die Testdaten des dritten Versuchs verwendet. Es wurden daher 1050 Gesten einberechnet, wobei die Anzahl der zusammengefügten Bilder für das CNN 4242 betrug.

Bei der in Tabelle 7.1 gezeigten Klassifizierungsdauer der Farbbilder durch das CNN wurde angenommen, dass eine Geste aus 8 Frames besteht. Da die ersten drei Frames ausgelassen werden wird mit 5 Frames pro Sequenz gerechnet was eine Klassifizierungsdauer von 4,7 Millisekunden pro Bild ergibt.

Laufzeiten Vorverarbeitung

	Distanzen	OUR-CVFH (pro PCD)	Farb-Frames	Final
Zeit Gesamt	-	**12075 s**	33,9 s	1,8 s
Zeit / Geste	-	**13750 ms** (859 ms)	32 ms	1,7 ms

Laufzeiten Klassifizierung

	Distanzen	Tiefe	Farbe CNN	Farbe RNN	Final
Benötigte Zeit Gesamt	2,5 s	3,6 s	23,6 s	2,3 s	2,1 s
Davon laden des Netzes	0,5 s	0,5 s	3,5 s	0,4 s	0,3 s
Zeit / Geste (ohne laden)	1,9 ms	2,9 ms	23,5 ms	1,8 ms	1,7 ms

Laufzeiten Gesamt

	Distanzen	Tiefe	Farbe	Final	$\sum$	(ohne Tiefe)
Zeit Gesamt	2,5 s	12079 s	59,8 s	3,9 s	**202,42 min**	(64,5 s)
Zeit / Geste	1,9 ms	13753 ms	57,3 ms	3,4 ms	**13,816 s**	(62,6 ms)

Tabelle 7.1.: Laufzeit der Verarbeitung bis zur Ausgabe Klassifizierung.

Eindeutig zeigt sich die Verarbeitung der Tiefendaten als Flaschenhals des Systems. Die Dauer der Umwandlung der Point-Cloud Daten in OUR-CVFH nimmt mit Abstand die meiste Zeit in Anspruch. Für die Umwandlung der 1050 Test-Gesten wurden 3 Stunden, 21 Minuten

und 15 Sekunden benötigt. Dies entspricht durchschnittlich 13,75 Sekunden pro Geste. Zwar benötigte OUR-CVFH für die Umwandlung einer Punktwolke im Durchschnitt 859 Millisekunden, jedoch müssen für eine Geste etwa acht Punktwolken von jeweils beiden Händen konvertiert werden. Selbst ein Weglassen der ersten drei Frames (Vgl. Abschnitt 6.3) würde hier eine Bearbeitungszeit von 8,59 Sekunden in Anspruch nehmen. Miteinbezogen sind in diesen Angaben auch die Zeit die es benötigt, um die konvertierten Daten in CTD-Dateien zusammenzufassen, wobei dieser Anteil verschwindend gering ist.

Die Laufzeit der Klassifizierung mit dem Faltungsnetz ist in Tabelle 7.1 in Zeit pro Bild angegeben. Hierbei werden die einzelnen Frames der Sequenz einer Geste klassifiziert und nicht die Geste selbst. Die Vorverarbeitung der Color-Frames besteht lediglich aus der Zusammenlegung der einzelnen Hand-Farb-Bilder und dem Update der Map-Datei. Die Gesamt-Zeit der Color-Frames wird durch die Addition der Zeiten deren Vorverarbeitung, der Approximation der Merkmale durch das CNN und der Klassifizierung des RNN berechnet.

Die gesamte Laufzeit von der Vorverarbeitung bis zur finalen Klassifizierung einer Geste beträgt über 13 Sekunden. Dies ist für eine Übersetzung in Echtzeit-Systeme unzureichend. Ohne die Betrachtung der Tiefe liegt die Laufzeit bei nur 63 Millisekunden pro Geste. Dies entspräche bei einer Latenz von 600 Millisekunden bis zur korrekten Erkennung der Geste lediglich einem Zehntel der Zeit. Diese Laufzeit würde einer Implementierung in einem Echtzeit-System gerecht werden.

7.5. SGD Lernparameter

Die Ermittlung der Hyperparameter für das Training mit Stochastic Gradient Descent erwies sich in diesen Versuchen als unkompliziert. Es wurden explorativ verschiedene Kombinationen der Mini-Batch-Größe und der Lernrate ausprobiert. Die in Kapitel 5 angegebenen Werte erwiesen sich als die am besten geeigneten Werte für diese Hyperparameter.

Beim Training mit unterschiedlicher Anzahl an Trainingsdaten war keine Änderung von Nöten. Anpassungen der Hyperparameter führten nur zu einer Verschlechterung der Ergebnisse und Performanz.

Es hat sich gezeigt, dass die Rectified Linear Unit gegenüber der Verwendung des Tangens Hyperbolicus (Weschta (2016b)) das Trainingsverhalten verbessern kann. Durch die Verwendung der Originalwerte ohne Normierung wurde der SGD-Algorithmus und dadurch das Training gegenüber Weschta (2016b) erheblich beschleunigt.

8. Schluss

Die Annahme, ein Hybridsystem kann einzelne Systeme verbessern und zu einem optimierten Ergebnis beitragen hat sich in dieser Ausarbeitung bestätigt. Die Teilsysteme selbst sind zwar bereits sehr zuverlässig, dennoch zeigten diese Anfälligkeiten bei bestimmten Gesten. Im finalen System waren solche Schwächen nicht auf einzelne Gesten zurückzuführen, sondern sehr verteilt. Wie bereits der Ansatz aus Wu u. a. (2015) kann auch in dieser Arbeit ein Vorteil aus der Kombination unterschiedlicher Verfahren gezogen werden.

8.1. Fazit

Das Hybridsystem eignet sich sehr gut für die Kombination der hier angewendeten Teilverfahren. Die Betrachtung der Abstände verschiedener Joints für deren räumliche Ausrichtung ist ein wichtiger Bestandteil dieser Gesten, der jedoch alleinstehend nicht ausreicht, um Gesten der Gebärdensprache zu erkennen. Die Implementierung dieses Teilsystems war durch die Anwendung ohne Vorverarbeitung der Daten mit wenig Aufwand verbunden.

Für die Erkennung der Handstellung war erheblich mehr Aufwand nötig. Die Untersuchung und Anwendung zweier Ansätze zu detaillierten Gestenerkennung betrachteten zum einen zweidimensionale und zum anderen dreidimensionale Merkmale.

Die Klassifizierung von Tiefendaten erzielte zwar gute Ergebnisse, stellte jedoch den Flaschenhals bezüglich der Verarbeitungszeit dar. Für eine Echtzeit-Erkennung wäre das Teilsystem mit der Implementierung von OUR-CVFH ungeeignet. Die Konvertierungsdauer einer Geste in Histogramme von über 13 Sekunden liegt dabei weit außerhalb des Rahmens. Die Anpassung der Parameter für das OUR-CVFH führen zwar zu einer Beschleunigung der Umwandlung, jedoch gehen dabei unter anderem der Detailgrad und die Genauigkeit der Oberflächenberechnung verloren. Eine Änderung der Parameter der Berechnung des Histogramms wiederum kann zwar zu einer Verbesserung der Ergebnisse führen, jedoch steigt dann die Verarbeitungsdauer weiter an.

Bei Betrachtung der Ergebnisse in Tabelle 6.5 ist zu erkennen, dass das Teilsystem der Erkennung der Tiefendaten etwas schlechter abschneidet als das Teilsystem zu Klassifizierung der Farb-Bilder. Eine Kombination aus Distanzen und der Vereinigung von CNN und RNN für zweidimensionale Farbbilder würde möglicherweise ausreichen, um ein ähnlich gutes Gesamtergebnis zu erzielen. Vorteil dabei wäre, dass eine Verarbeitungs- und Klassifizierungszeit von insgesamt 62,6 Millisekunden deutlich besser für ein System zur Echtzeit-Klassifizierung geeignet wäre.

Das CNN selbst liefert bereits gute Resultate, obwohl hier einzelne Frames klassifiziert werden, die innerhalb einer Sequenz einer Geste sehr unterschiedlich sein können. Das sind beispielsweise Wörter wie “zusammen” oder “müssen”, bei denen sich nicht nur die Orientierung oder auch die Stellung einzelner Finger über die Zeit ändert. Dies kann bei wachsender Anzahl an zu klassifizierenden Gesten durchaus problemanfällig werden. Jedoch ist es mit dem anschließenden Kontextbezug durch das finale RNN wieder möglich diese Fehler zu beheben, da Gesten so wieder unterschieden werden können.

Weiterhin zeigen die Resultate, dass bereits wenige dieser Farb-Frames ausreichen, um ein gutes Ergebnis zu erzielen. Die Aufnahme und Klassifizierung von drei bis vier Frames hat eine hohe Trefferquote in Kombination mit dem RNN vorzuweisen, ohne auf weitere Algorithmen wie beispielsweise eine Low Rank Approximation aus Wang u. a. (2015) oder ähnliches zurückzugreifen.

Einzig der Trainingsaufwand sticht bei der Verwendung einer Kombination von CNN und RNN negativ hervor. Dieser ist beim CNN deutlich höher als bei den anderen Netzen, was jedoch nicht in die Erkennungszeit einspielt. Auch die zur Klassifizierung benötigte Zeit ist hier gegenüber den anderen Netzen nur unwesentlich erhöht. Die Kombination von CNN und RNN schneidet bei ausreichender Anzahl an Trainingsdaten grundsätzlich besser ab als die Klassifizierung von OUR-CVFH.

Neben den Aspekten der Netze bringt die Kinect-Kamera grundsätzliche Vorteile mit sich. Diese sind beispielsweise das Tracking der Joints und dem Coordinate-Mapper zur Umwandlung von Koordinaten zwischen Farb-, Tiefen- und Bodyframes. In der Praxis jedoch stellte sie einen großen Schwachpunkt in diesem System dar. Vor allem die Koordinaten und Tiefeninformation, die bei der Aufnahme bestimmter Gesten falsch bereitgestellt wurden, erhöhten die Fehlerrate. Weiterhin wäre die Kinect-Kamera aufgrund ihrer Probleme mit der Geschwindigkeit ungeeignet für den Einsatz zu Erkennung von Gesten der Gebärdensprache im Alltag. Für gewöhnlich wird viel schneller gebärdet als es bei diesen Aufnahmen der Fall war.

8.2. Ausblick

Um den Aufwand für die Klassifizierung der zweidimensionalen Farbbilder zu reduzieren, könnte das Faltungsnetz mit dem RNN kombiniert werden. Solche Ansätze wurden bereits in Donahue u. a. (2015) und Shi u. a. (2015) präsentiert. In dieser Arbeit wurde ein solches Vorgehen bewusst getrennt, um den Einfluss des CNN und RNN einzeln betrachten und evaluieren zu können.

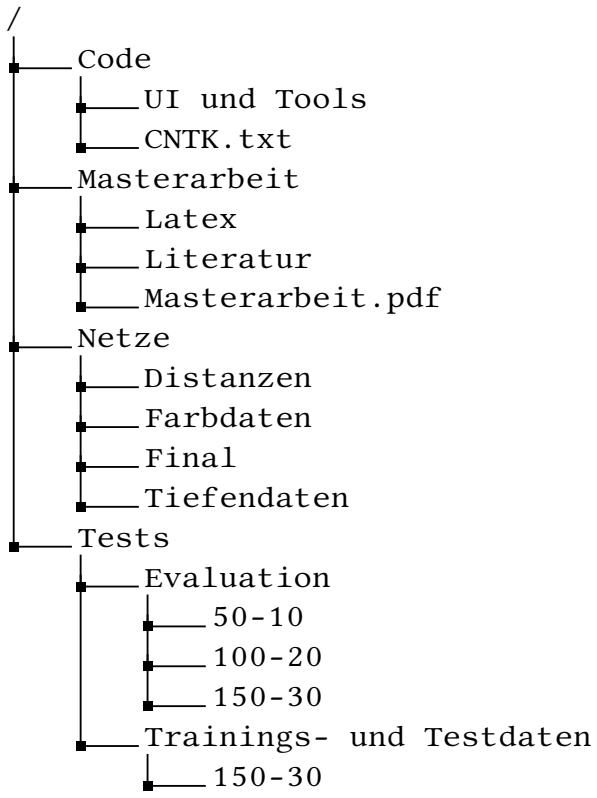
Weitere Tests sollten mit entsprechend mehr zu klassifizierenden Gesten durchgeführt werden. Da der Wortschatz der deutschen Gebärde enorm groß ist, bleibt hier nur zu erahnen, wie viel Aufwand mehr betrieben werden muss. Eine Erhöhung des Vokabulars dieses Systems hat erheblich mehr Trainingsaufwand, Vorverarbeitungszeit und nötige Tests zur Folge. Der Hidden-Layer zur Weitergabe der approximierten Merkmale der Teilsysteme sollte entsprechend vergrößert werden. Da es sich bei diesem System um einen neuartigen Ansatz handelt, müssen darüber hinaus die Daten für dieses spezielle Hybridsystem neu gesammelt werden. Dies stellte in dieser Arbeit den zeitlich größten Aufwand in den Versuchen dar. Die Entwicklung eines öffentlichen Programms zur Aufnahme solcher Daten wäre notwendig, damit zur Beschleunigung dieses Prozesses Gesten von mehreren Personen eingespielt werden können.

Mit ansteigendem Vokabular und erweitertem Umfang steigt auch die Komplexität. Es kommt in der Gebärde vor, dass Wörter erst aus dem Kontext eines Satzes genau identifiziert werden können. Diese Eigenschaft kann auch nützlich sein, um der steigenden Komplexität entgegen zu wirken. Analog zu Spracherkennung können auf höherer Ebene die einzelnen Worte als Satz klassifiziert werden. Dazu könnte beispielsweise das finale RNN durch ein LSTM-Netz, welches in Hochreiter und Schmidhuber (1997) präsentiert wird, ersetzt werden. Durch die dort erwiesene bessere Eignung bei der Erkennung größerer Zeitfenster wäre es damit möglich, auch längere Wortkombinationen und ganze Sätze zu klassifizieren.

A. Anhang

A.1. Inhalt der DVD

Der Inhalt der DVD gliedert sich in mehrere Bereiche. Zunächst wurde das User-Interface für die Aufnahme und Verwaltung der Daten, sowie Informationen über die verwendete CNTK-Umgebung angefügt. Weiterhin sind die in dieser Arbeit verwendeten Netzkonfigurationsdateien, sowie Test- und Evaluationsdateien aller Versuche beigelegt. Die Trainings- und Testdaten beinhalten die Daten aller Versuche, da diese inkrementell aufgenommen wurden.



A.2. CNTK -Konfigurationsdatei eines RNN

Reduzierte Darstellung einer CNTK-Konfigurationsdatei für ein RNN. Aufgeteilt ist die Konfiguration in Training (ab Zeile 8) und Test (ab Zeile 75). Der “BrainscriptNetworkBuilder” deklariert ab Zeile 11 den Aufbau des Netzes. Enthalten sind hier die Definition aller Layer und deren Dimensionen (Zeile 13-16) mit deren Gewichtungsmatrizen (Zeile 25-28) und Bias-Vektoren (Zeile 22-24). Die Rekurrenz wird durch den in Zeile 20 angegebenen Parameter “PastValue” realisiert. Die Zeilen 30-32 implementieren die Layer des Netzes. In der Testkonfiguration wird das trainierte Model (“rnn.dnn”, angegeben in Zeile 6) geladen und die Indizes der Ergebnisse in einen entsprechenden String gemapped.

```
1  # set "deviceId" to -1 (Choosing the CPU),  >=0 (Choosing specified GPU)
2  # or "auto" (choosing the best GPU, or CPU if no GPU available)
3  deviceId = auto
4  command = train:test  # Chooses definitions to run
5
6  modelPath = "Models/rnn.dnn"  # Save net and Checkpoint-Files to this
7
8  train = [
9    action = "train"  # Definition of the net and its training
10
11   BrainScriptNetworkBuilder = [
12
13     FDim = 128    # feature dimension (input layer)
14     HDim1 = 192   # first hidden-layer dimension
15     HDim2 = 128     # second hidden-layer dimension
16     LDim = 35   # number of classes (output-layer)
17
18     features  = Input(FDim)
19     labels    = Input(LDim)
20     Rd1     = PastValue(HDim1, R1, timeStep=1)
21
22     b1 = Parameter (HDim1, 1, init="fixedValue", value=0.0, initOnCPUOnly=true);
23     b2 = Parameter (HDim2, 1, init="fixedValue", value=0.0, initOnCPUOnly=true);
24     by = Parameter (LDim, 1, init="fixedValue", value=0.0, initOnCPUOnly=true);
25     Wx1 = Parameter (HDim1, FDim, init="uniform", initValueScale=1, initOnCPUOnly=true);
26     Wx2 = Parameter (HDim2, HDim1, init="uniform", initValueScale=1, initOnCPUOnly=true);
27     Wr1 = Parameter (HDim1, HDim1, init="uniform", initValueScale=1, initOnCPUOnly=true);
28     Wy  = Parameter (LDim, HDim2, init="uniform", initValueScale=1, initOnCPUOnly=true);
29
30     R1 = ReLU((Wx1 * features)  + (Wr1 * Rd1) + b1) # Recurrent Layer
31     L2 = ReLU(Wx2 * R1 + b2)
32     y = Wy * L2 + by
33
34     out     = Softmax(y)
35     ce    = CrossEntropyWithSoftmax(labels, y)
36     err   = ErrorPrediction(labels, y)
37
38     # Root Nodes
39     featureNodes = (features)
40     labelNodes = (labels)
41     criterionNodes = (ce)
42     evaluationNodes = (err)
43     outputNodes = (out:L2)  # Output last Hidden-Layer and Output-Layer in files
44   ]
45
46
47   SGD = [     #stochastic gradient descent
48     epochSize = 0
49     minibatchSize = 32
```

```
50      learningRatesPerMB = 0.01
51      numMBsToShowResult = 10000
52      momentumPerMB = 0.9
53      maxEpochs = 20
54      keepCheckPointFiles = false
55    ]
56
57    reader = [
58      readerType = "CNTKTextFormatReader"
59      file = "rnnTrainData.txt"
60      randomize = true
61
62      input = [
63        labels = [
64          alias = "L"
65          dim = 35
66        ]
67        features = [
68          alias = "F"
69          dim = 128
70        ]
71      ]
72    ]
73  ]
74
75  test=[
76    action = "write"  # Definition of tests for the trained net
77
78    reader = [
79      readerType = "CNTKTextFormatReader"
80      file = "rnnTestData.txt"
81      randomize = false
82
83      input = [
84        labels = [
85          alias = "L"
86          dim = 35
87        ]
88        features = [
89          alias = "F"
90          dim = 128
91        ]
92      ]
93    ]
94
95    outputPath = "Output" # Write Results to this file (outputNode-names will be appended)
96    format = [
97      type = "category"       # Finds the index of the highest-scoring entry.
98      labelMappingFile = "../Mappings.txt"  # Translates the index into a mapped string.
99    ]
100 ]
```

A.3. Ausschnitt aus der CNTK Konsole

Ein Ausschnitt der Ausgabe eines Trainings mit CNTK. Aufgerufen wird die Konfiguration des Netzes und Trainings durch den in Zeile 1 dargestellten Konsolenbefehl. Angegeben werden bei Beginn Informationen über die verwendete CNTK-Version und das System. Während des Trainings werden in jeder Epoche unter Anderem die Werte der Kreuzentropie (“ce”), des

Prediction-Errors (“err”) oder die für die Epoche benötigte Zeit angegeben (siehe Zeile 26 und 27).

```
1  > c:\CNTK\tests\hands_test>cntk configfile=hands.cntk
2  CNTK 2.0.beta5.0+ (master 8a22b3, Dec  1 2016 19:13:41) on TIMB39 at 2016/12/13 21:03:05
3
4  cntk  configfile=hands.cntk
5  ----------------------------------------------------
6  Build info:
7
8  [...]
9
10 ----------------------------------------------
11 GPU info:
12 Device[0]: cores = 1536;
13 computeCapability = 5.2;
14 type = "GeForce GTX 960";
15 memory = 4096 MB
16 ----------------------------------------------
17
18 [...]
19
20 Starting Epoch 7:
21 learning rate per sample = 0.000031
22 effective momentum = 0.900000  momentum as time constant = 303.7 samples
23
24 Starting minibatch loop.
25 Finished Epoch[ 7 of 300]: [Training]
26 ce = 0.77850293 * 32887; err = 0.22057348 * 32887; totalSamplesSeen = 230209;
27 learningRatePerSample = 3.1250001e-005; epochTime=9.27851s
28
29 SGD: Saving checkpoint model 'Models/hands.dnn.7'
30
31 [...]
32
33 Action "train" complete.
```

A.4. Berechnung des OUR-CVFH der 3D-Merkmale

Die verwendete Konfiguration zur Konvertierung der Punktwolken in OUR-CVFH. Nach Anlegen der Objekte der Point Cloud Library (Zeile 3-12) werden die Normalen der Referenzfläche berechnet (Zeile 20-24). Dazu wird das KdTree-Verfahren verwendet, dessen Radius der Knotensuche gesetzt wird (Zeile 22). Bevor die Histogrammwerte in Zeile 39 berechnet und gegebenenfalls normalisiert werden (Zeile 36), sind in den Zeilen 31-36 Parameter für die Konvertierung festgelegt. Der Parameter “EPSAngleThreshold” (Zeile 32) gibt an, in welchem Winkel die Normalen zwischen zwei Punkten von einander abweichen dürfen, um einem Cluster anzugehören. Der “CurvatureThreshold” (Zeile 34) bestimmt, innerhalb welches Winkels die Normalen gelöscht werden können. In Zeile 36 wird das minimale Achsenverhältnis zwischen Referenzflächen angegeben. Die Ausgabe der berechneten Histogrammwerte erfolgt in eine “.VFH”-Datei (Zeile 43-50).

```
int evaluatePCD(const char* filename, bool print, const char* exportFile, int offset, bool plot)
{
 // Cloud for storing the object.
 pcl::PointCloud<pcl::PointXYZ>::Ptr object(new pcl::PointCloud<pcl::PointXYZ>);
 // Object for storing the normals.
 pcl::PointCloud<pcl::Normal>::Ptr normals(new pcl::PointCloud<pcl::Normal>);
 // Object for storing the OUR-CVFH descriptors.
 pcl::PointCloud<pcl::VFHSignature308>::Ptr descriptors(new pcl::PointCloud<pcl::VFHSignature308>);
 // KDTree as SearchMethod for Normalestimation
 pcl::search::KdTree<pcl::PointXYZ>::Ptr kdtree(new pcl::search::KdTree<pcl::PointXYZ>);
 // Object for estimating the normals
 pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> normalEstimation;

 // Read a PCD file from disk.
 if (pcl::io::loadPCDFile<pcl::PointXYZ>(filename, *object) != 0) {
  std::cout << "Unable to open or read file : " << filename << std::endl;
  return -1;
 }

 // Estimate the normals.
 normalEstimation.setInputCloud(object);
 normalEstimation.setRadiusSearch(0.05);
 normalEstimation.setSearchMethod(kdtree);
 normalEstimation.compute(*normals);

 // OUR-CVFH estimation object.
 pcl::OURCVFHEstimation<pcl::PointXYZ, pcl::Normal, pcl::VFHSignature308> ourcvfh;
 ourcvfh.setInputCloud(object);
 ourcvfh.setInputNormals(normals);
 ourcvfh.setSearchMethod(kdtree);
 // 15.0 degrees. Sets max. deviation of the normals between two points
 ourcvfh.setEPSAngleThreshold(15.0 / 180.0 * M_PI);
 // Sets curvature threshold for removing normals.
 ourcvfh.setCurvatureThreshold(3.0);
 // Set the minimum axis ratio between the axes.
 ourcvfh.setAxisRatio(1.0);
 // Sets wether the signatures should be normalized or not.
 ourcvfh.setNormalizeBins(true);
 ourcvfh.compute(*descriptors);

 int histo_bins = 308;  // Standard in CVFH
 int histogram_std_idx = 0;

 //=== Write histogram values to VFH file ===
 std::ofstream output_file;
 output_file.open(exportFile, ios::out);

 for (int binIdx = 0; binIdx < histo_bins; ++binIdx){
  output_file << (binIdx + offset) << " "
     << descriptors->points[histogram_std_idx].histogram[binIdx] << "\n";
 }
 output_file.close();

 return 0;
}
```

A.5. Übersicht über die Gesten der Tests

Abbildung A.1 und Abbildung A.2 zeigen die 35 Gesten für das Trainings- und Testset. Diese Gesten [1] unterscheiden sich möglicherweise von denen anderer Dialekte und sind daher unter Vorbehalt gewählt worden.

A.6. Konfusions-Matrizen

Dieser Abschnitt zeigt die Konfusions-Matrizen der rekurrenten Teilsysteme aller Versuchsreihen. Vertikal angeordnet sind die zu klassifizierenden Labels. Horizontal gekennzeichnet ist die tatsächliche Klassifizierung. Die Treffer wurden farblich kodiert, um die Gewichtung der Abweichungen von den Labels zu verdeutlichen. Die Kodierung ist den jeweiligen Abbildungen beigefügt.

Versuch 1

Abbildung A.3 zeigt die Konfusions-Matrizen der rekurrenten Teilsysteme des ersten Versuchs mit 50 Trainings- und 10 Testdaten.

Versuch 2

Abbildung A.4 zeigt die Konfusions-Matrizen der rekurrenten Teilsysteme des zweiten Versuchs mit 100 Trainings- und 20 Testdaten.

Versuch 3

Abbildung A.5 zeigt die Konfusions-Matrizen der rekurrenten Teilsysteme des dritten Versuchs mit 150 Trainings- und 30 Testdaten.

[1] Die Gesten wurden unter Zuhilfenahme der Internetseiten https://www.youtube.com/watch?v=BAB4n84F1og und http://www.gebaerdenlernen.de/ festgelegt.

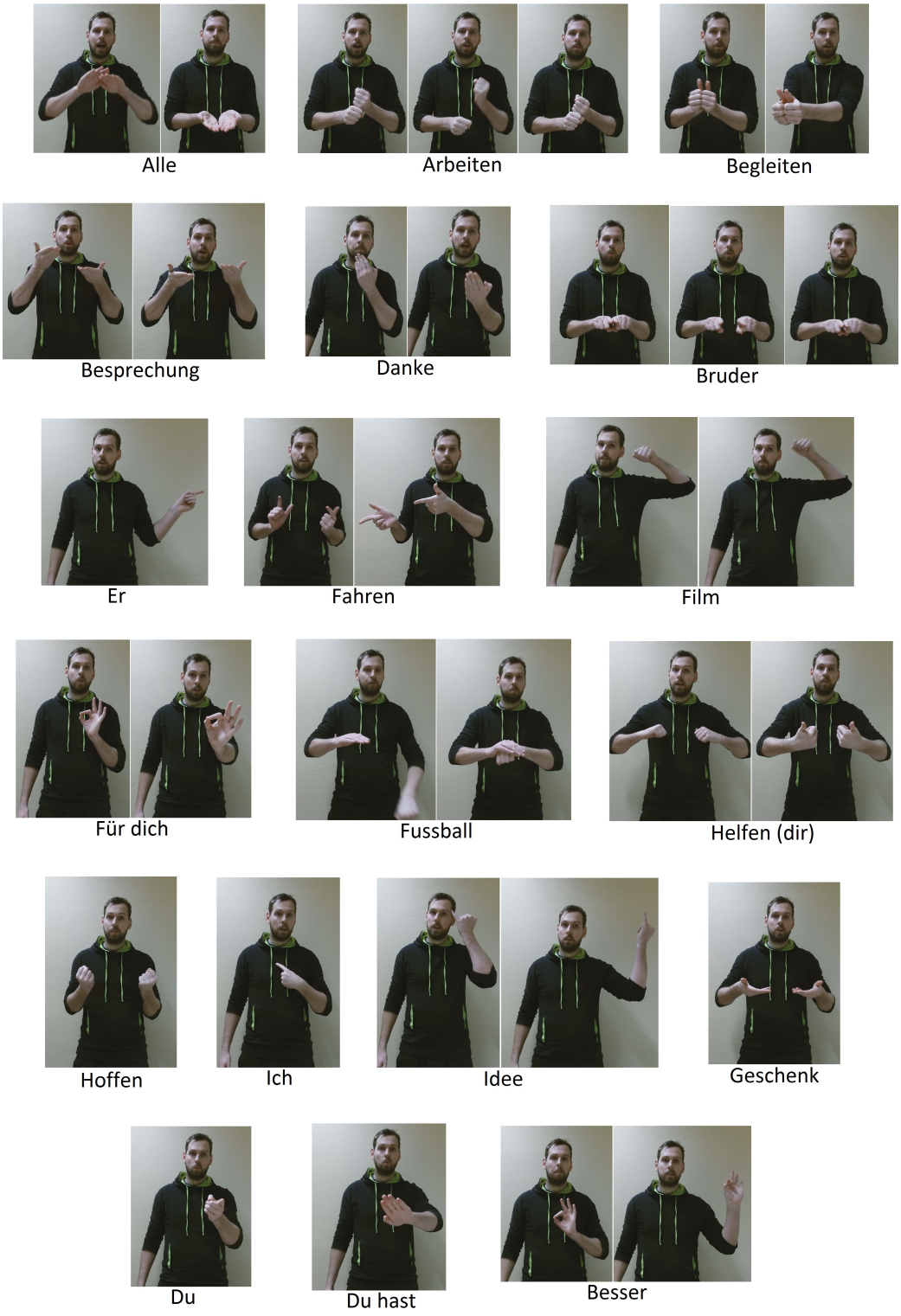


Abbildung A.1.: Die Gesten der Versuche (Teil 1).

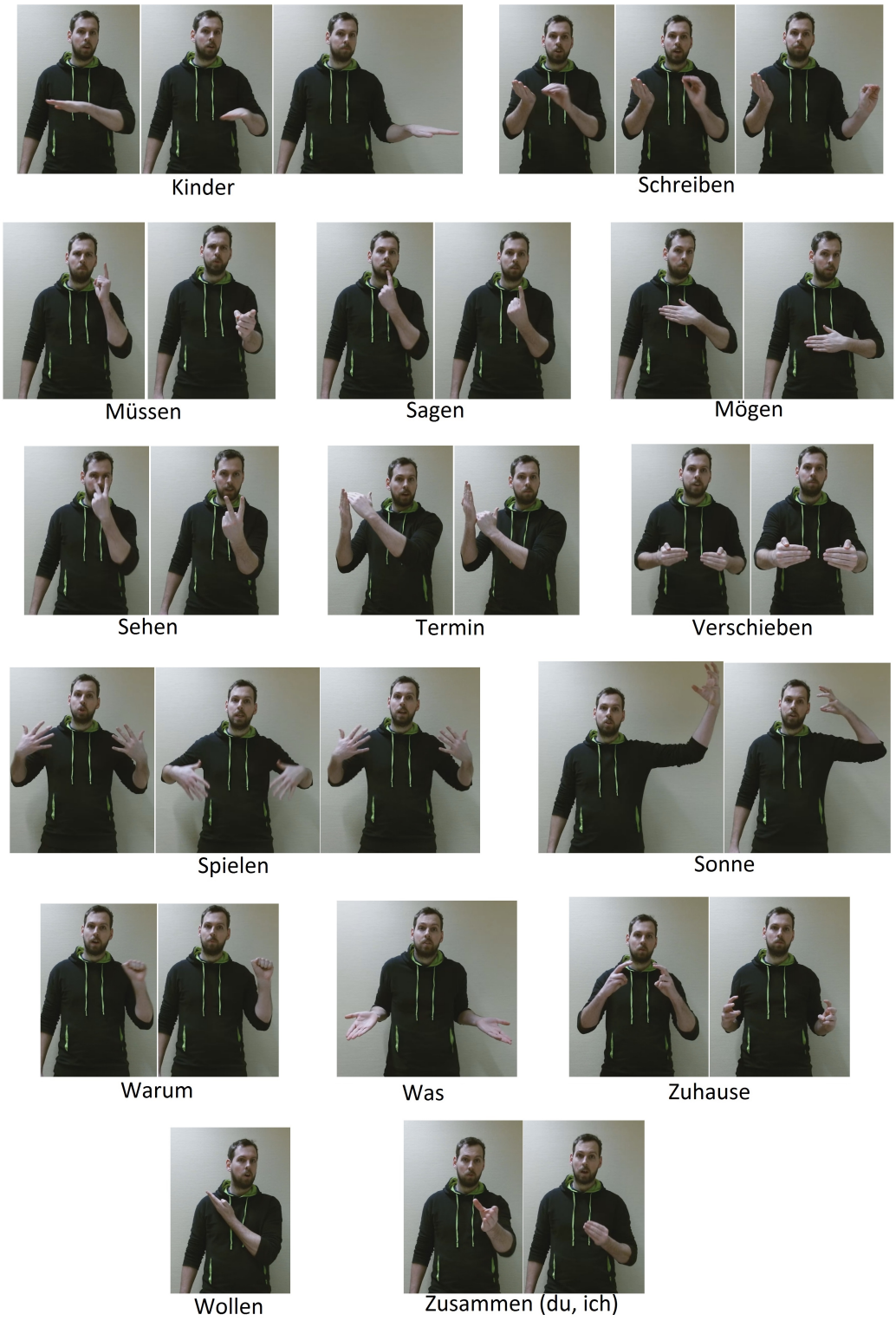


Abbildung A.2.: Die Gesten der Versuche (Teil 2).

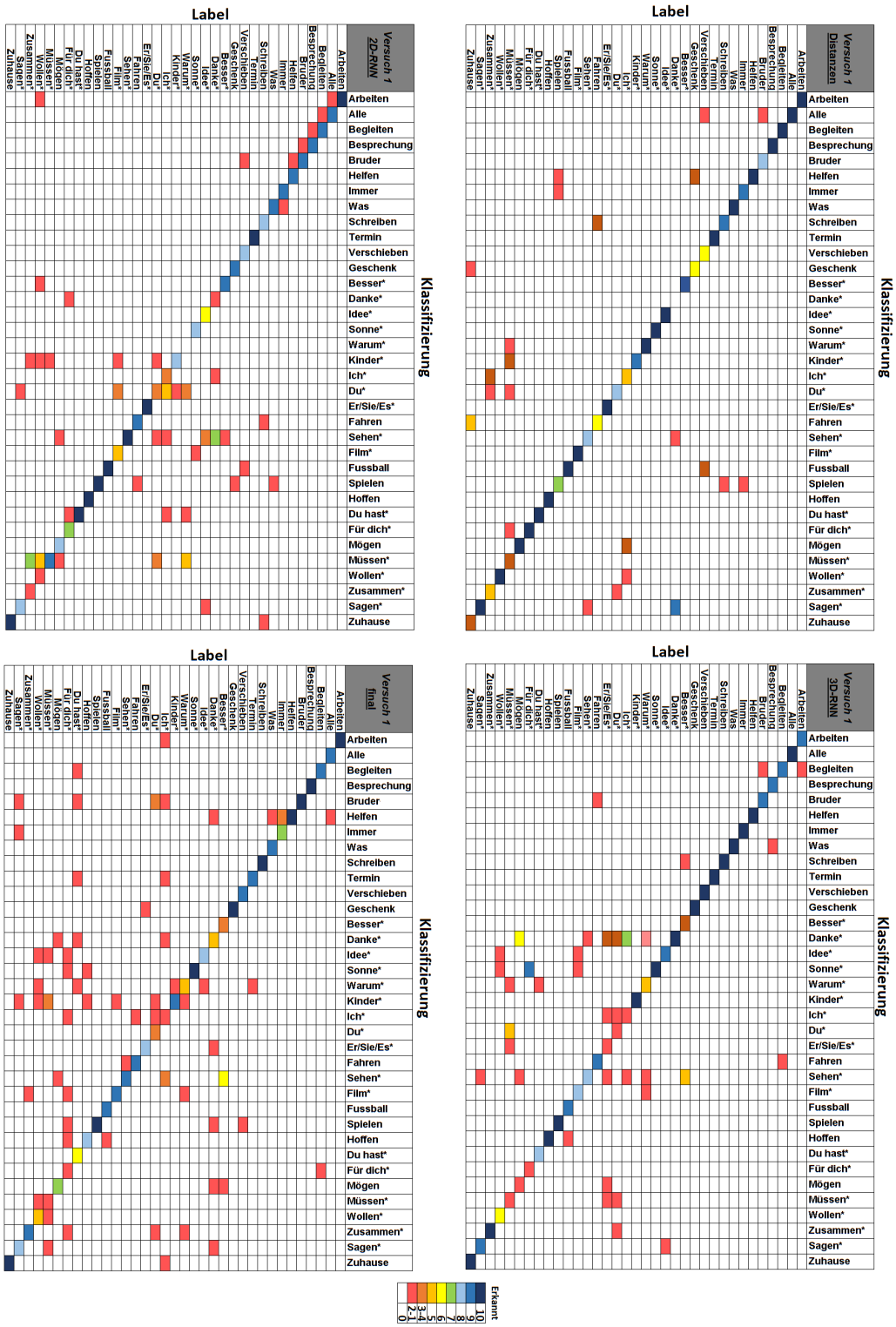


Abbildung A.3.: Die Konfusions-Matrizen des ersten Versuchs.

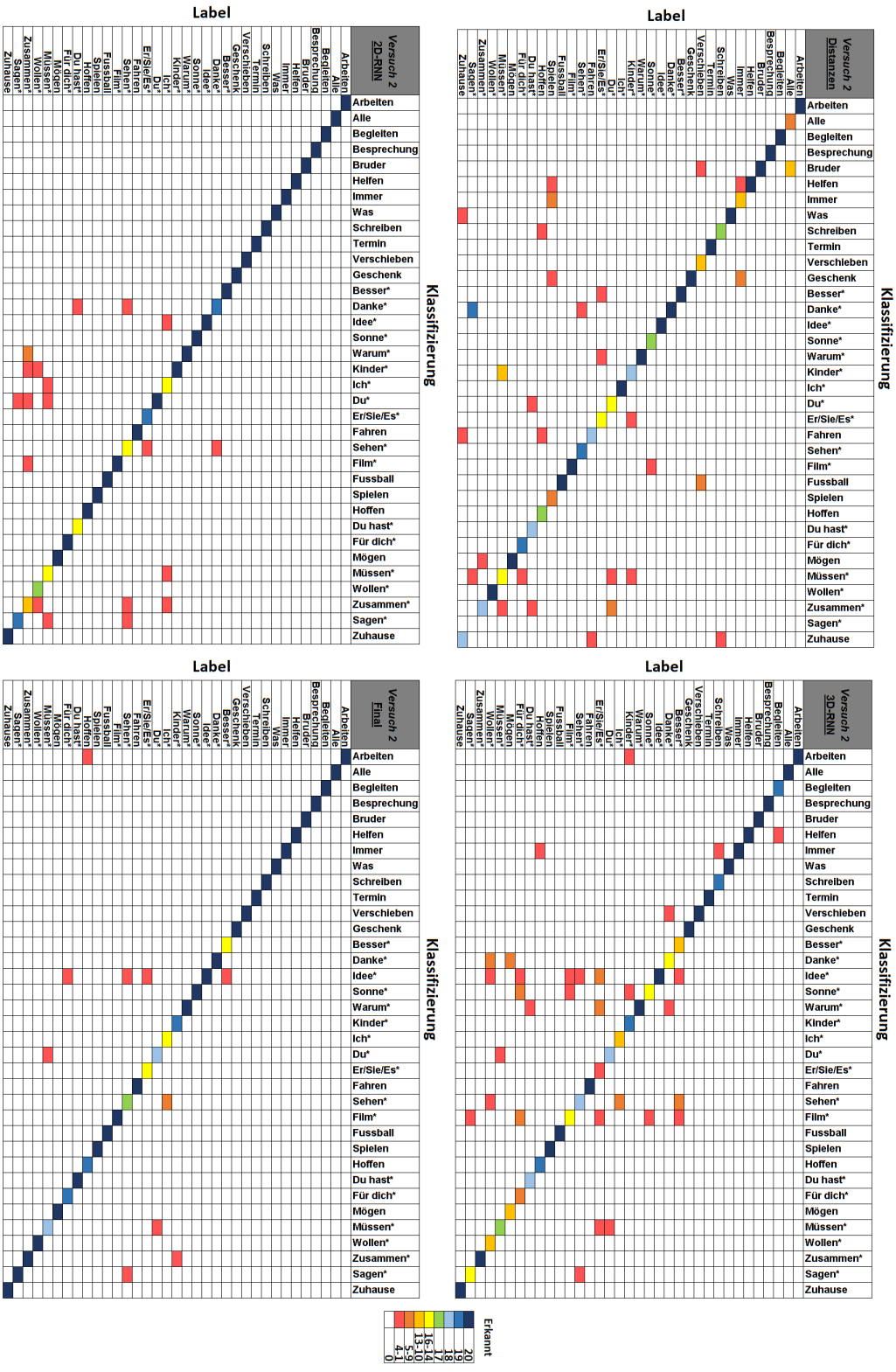


Abbildung A.4.: Die Konfusions-Matrizen des zweiten Versuchs.

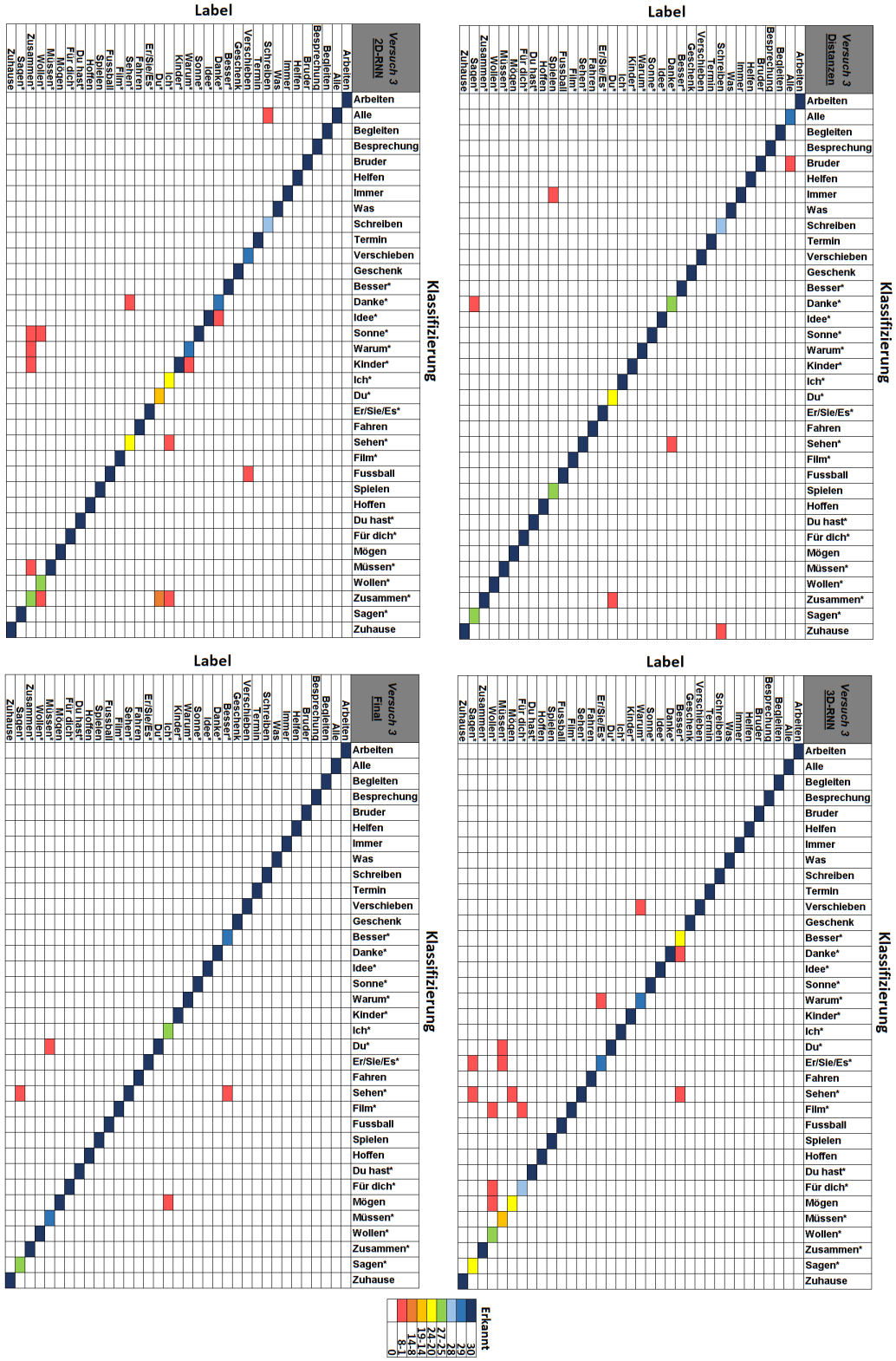


Abbildung A.5.: Die Konfusions-Matrizen des dritten Versuchs.

Literaturverzeichnis

[Aldoma u. a. 2012] Aldoma, Aitor ; Tombari, Federico ; Rusu, Radu B. ; Vincze, Markus: OUR-CVFH – Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram for Object Recognition and 6DOF Pose Estimation / Vision4Robotics Group, ACIN, Vienna University of Technology; Computer Vision Lab., DEIS - ARCES, University of Bologna; Open Perception Inc. 2012. – Forschungsbericht

[Alshekhali u. a. 2011] Alshekhali, M. u. a.: Hand Gesture Recognition System / Compter Engineering Department, The Islamic University of Gaza. 2011. – Forschungsbericht

[Böhle 2016] Böhle, Sebastian: *3D-Objekterkennung mit Faltungsnetzwerken*, Hochschule für angewandte Wissenschaften Hamburg, Dept. Informatik, Diplomarbeit, 2016

[Cybenko 1989] Cybenko, G.: Approximation by Superpositions of a Sigmoidal Function. In: *Math. Control Signals Systems*, 1989, S. 303–314

[Davis und Shah 1994] Davis, J. ; Shah, Mubarak: Recognizing Hand Gestures / Computer Vision Laboratory, University of Central Florida, Orlando FL 32816, USA. Mai 1994. – Forschungsbericht

[Donahue u. a. 2015] Donahue, Jeff ; Hendricks, Lisa A. ; Guadarrama, Sergio ; Rohrbach, Marcus ; Venugopalan, Subhashini ; Saenk, Kate ; Rohrbach, Marcus ; Venugopalan, Subhashini ; Saenko, Kate ; Darrell, Trevor: Long-term Recurrent Convolutional Networks for Visual Recognition and Description. In: *CVPR*, 2015

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

[Gharasuie und Seyedarabi 2013] Gharasuie, M.M. ; Seyedarabi, H.: Realtime dynamic hand gesture recognition using hidden markov models. In: *8th Iranian Conference on Machine Vision and Image Processing (MVIP), 2013*, 9 2013, S. 194–199

[Hochreiter und Schmidhuber 1997] Hochreiter, Sepp ; Schmidhuber, Jürgen: LONG SHORT-TERM MEMORY. In: *NEURAL COMPUTATION*, 1997 (9), S. 1735 – 1780

[Holden u. a. 1999] Holden, E. ; Owens, R. ; Roy, G.G.: Visual Sign Language Recognition / Department of Computer Science, University of Western Australia; School of Engnieering, Murdoc University. 1999. – Forschungsbericht

[Just und Marcel 2005] Just, A. ; Marcel, S.: Two-Handed Gesture Recognition / IDIAP Research Instsitute, CH-1920 Martigny, Switzerland. Mai 2005. – Forschungsbericht

[LeCun u. a. 1998a] LeCun, Y. ; Bottou, L. ; Orr, G. ; Muller, K.: Efficient BackProp. In: Orr, G. (Hrsg.) ; K., Muller (Hrsg.): *Neural Networks: Tricks of the trade*, Springer, 1998

[LeCun u. a. 1998b] LeCun, Yann ; Bottou, Léon ; Bengio, Yoshua ; PatrickHaffner: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE 86*, November 1998, S. 2278–2324

[MacCormick 2016] MacCormick, John: *How does the kinect work?* Presentation: https://users.dickinson.edu/~jmac/selected-talks/kinect.pdf. 2016. – "[ONLINE, Besucht: 12. Januar 2017]"

[Meisel 2012] Meisel, Prof. Dr. A.: *University Lecture: Robot-Vision.* 2012

[Microsoft 2016a] Microsoft: *CNTK - Computation Network Toolkit.* https://github.com/Microsoft/CNTK/wiki. 2016. – "[ONLINE, Besucht: 6. Juni 2016]"

[Microsoft 2016b] Microsoft, DeveloperNetwork: *JointType Enumeration.* https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx. 2016. – "[ONLINE, Besucht: 21. November 2016]"

[Microsoft 2016c] Microsoft, TechNet: *Collaboration, expertise produce enhanced sensing in Xbox One.* https://blogs.technet.microsoft.com/microsoft_blog/2013/10/02/collaboration-expertise-produce-enhanced-sensing-in-xbox-one/. 2016. – "[ONLINE, Besucht: 21. November 2016]"

[Nielsen 2017] Nielsen, Michael: *Neural Networks and Deep Learning.* Online Book - "http://neuralnetworksanddeeplearning.com/chap3.html#the_cross-entropy_cost_function". 2017. – "[ONLINE, Besucht: 3. Januar 2017]"

[Parker und Baumback 2009] Parker, J.R. ; Baumback, M.: Finger recognition for hand pose determination. In: *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, Okt 2009, S. 2492–2497

[Ravikiran u. a. 2009] Ravikiran, J. u. a.: Finger Detection for Sign Language Recognition. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol I*, März 2009

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

[Shen u. a. 2011] Shen, Xiaohui ; Hua, Gang ; Williams, L. ; Wu, Ying: Motion divergence fields for dynamic hand gesture recognition. In: *Automatic Face Gesture Recognition and Workshops (FG 2011), 2011 IEEE*, 2011

[Shi u. a. 2015] Shi, Baoguang ; Bai, Xiang ; Yao, Cong: An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. In: *CoRR* abs/1507.05717 (2015)

[Shotton u. a. 2011] Shotton, Jamie ; Fitzgibbon, Andrew ; Cook, Mat ; Sharp, Toby ; Finocchio, Mark ; Moore, Richard ; Kipman, Alex ; Blake, Andrew: Real-Time Human Pose Recognition in Parts from Single Depth Images / Microsoft Research Cambridge and Xbox Incubation. 2011. – Forschungsbericht

[Silanon und Suvonvorn 2014] Silanon, Kittasil ; Suvonvorn, Nikom: Finger-spelling recognition system using fuzzy finger shape and hand appearance features. In: *Digital Information and Communication Technology and it's Applications (DICTAP), 2014 Fourth International Conference on*, Mai 2014, S. 419–424

[Tompson u. a. 2014] Tompson, Jonatan ; Stein, Murphy ; LeCun, Yann ; Perlin, Ken: Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks. In: *ACM Transaction on Graphics* (2014)

[UFLDL Computer Science Department 2016] UFLDL Computer Science Department, Stanford U.: *Deep Learning Tutorial - Optimization: Stochastic Gradient Descent.* http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/. 2016. – "[ONLINE, Besucht: 21. Juni 2016]"

[Waibel u. a. 1988] Waibel, A. ; Hanazawa, T. ; Hinton, G. ; Shikano, K. ; Lang, K.: Phoneme recognition: Neural networks vs. hidden Markov models. In: *Conference Paper in Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference*, Mai 1988

[Wang u. a. 2015] Wang, Hanjie ; Chai, Xiujuan ; Zhou, Yu ; Chen, Xilin: Fast Sign Language Recognition Benefited From Low Rank Approximation. In: *The 11th IEEE International Conference on Automatic Face and Gesture Recognition (FG2015), 2015, Slovenia.*, 2015

[Weschta 2016a] Weschta, Martin: Grundlagen zur Erkennung statischer Gesten im dreidimensionalen Raum / Hochschule für angewandte Wissenschaften Hamburg, Dept. Informatik, Deutschland. 4 2016. – Forschungsbericht

[Weschta 2016b] Weschta, Martin: Neuronale Recurrent-Netze zur Klassifizierung dynamischer Gesten / Hochschule für angewandte Wissenschaften Hamburg, Dept. Informatik, Deutschland. 8 2016. – Forschungsbericht

[Wu u. a. 2015] Wu, Zuxuan ; Wang, Xi ; Jiang, Yu-Gang ; Ye, Hao ; Xue, Xiangyang: Modeling Spatial-Temporal Clues in a Hybrid Deep Learning Framework for Video Classification. In: *Proceedings of the 23rd ACM international conference on Multimedia (MM '15). ACM, New York, NY, USA, 461-470.*, 2015

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

Hamburg, 8. Februar 2017 Martin Weschta