Fachbereich 2
Informatik und Ingenieurwissenschaften
Studiengang: Informatik (Bachelor of Science)

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science

Gestenerkennung im Augmented-Reality-Umfeld mittels neuronaler Netze

---

Autor: Tobias Rohrer
Matrikelnummer: 1089431
Hauptreferent: Prof. Dr. Thomas Gabel
Korreferent: Prof. Dr. Christian Baun
Abgabetermin: 23.01.2018

Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebenen Quellen, Hilfen und Hilfsmittel benutzt habe. Die Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht.

Ort, Datum Unterschrift

Danksagung

An dieser Stelle möchte ich mich bei meiner Familie und Freunden bedanken, die mich durch ein Korrekturlesen der Arbeit unterstützt haben.
Weiterhin bedanke ich mich bei Prof. Dr. Thomas Gabel für die Betreuung der Arbeit und vor allem für die hilfreichen Anregungen sowie für die konstruktive Kritik.

Zusammenfassung

Die vorliegende Bachelorarbeit untersucht neuronale Faltungsnetzwerke zur Erkennung von auf der Microsoft HoloLens durchgeführten Handgesten. Ein zweidimensionales Faltungsnetzwerk und eine dreidimensionale Erweiterung wurden entworfen, implementiert und im Anschluss in einer empirischen Analyse evaluiert. Da zu Beginn der Arbeit keine Basis an Daten zur Verfügung stand, wurden 600 Gestendaten aus vier verschiedenen Klassen durch reale Interaktion mit der Microsoft HoloLens erstellt und anschließend künstlich vermehrt. Das implementierte zweidimensionale Faltungsnetzwerk erreichte eine Klassifizierungsgenauigkeit von 100% auf die Testdaten (200 der 600 echten Gestendaten) und 95,68% auf Gestendaten, die von anderen Personen als dem Autor entstammen (Fremdgestendaten). Das implementierte dreidimensionale Faltungsnetzwerk erreichte eine Klassifizierungsgenauigkeit von 99% auf die Testdaten und 91,35% auf die Fremdgestendaten. In einem Versuch wurde das dreidimensionale Faltungsnetzwerk mit nur 10 Datensätzen pro Gestenklasse trainiert, wobei eine Klassifizierungsgenauigkeit von 92% auf die Testdaten und 86,49% auf die Fremdgestendaten erreicht werden konnte. Durch eine Visualisierung des Gelernten konnte anhand der zweidimensionalen Implementierung gezeigt werden, durch welche Gesteneigenschaften eine Klassifizierung vorgenommen wird. In einem abschließenden Experiment wurde das dreidimensionale Netzwerk anhand von Gestendaten untersucht, die um zeitliche Informationen ergänzt wurden. In diesem Experiment wurde eine Klassifizierungsgenauigkeit von 99,25% auf die Testdaten und 89,19% auf die Fremdgestendaten erreicht.

Abstract

This thesis analyses convolutional neural networks for recognizing hand gestures which were performed on the Microsoft HoloLens. A two dimensional convolutional network and a three dimensional extension were drafted, implemented and evaluated. In the beginning, there was no gesture data which could be used for this work. Therefore, 600 real gesture data from four different classes where created in real interaction with the Microsoft HoloLens. Data augmentation was applied to increase the data used for training and evaluating the network. The implemented two dimensional convolutional network achieved a classification accuracy of 100% on the test data (200 of the 600 real data) and 95,68% on gestures which were done by other persons (foreign gestures). The implemented three dimensional convolutional network was able to achieve a classification accuracy of 99% on the test data and 91,35% on the foreign gestures. In an experiment, the three dimensional implementation was trained by only 10% of the training data and achieved a classification accuracy of 92% and on the test data and 86,49% on the foreign gestures. Visualizations of the parameters, which were learned by the network could show gesture properties which were detected by the network. In a further experiment, the implemented three dimensional network was analyzed using data which was extended by time information. In this experiment the network archived a classification accuracy of 99,25% on the test data and 89,19% on the foreign gestures.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

1 Einleitung

Die Interaktion mit Systemen im Augmented-Reality-Umfeld kann anhand traditioneller Peripheriegeräte wie Maus und Tastatur nicht natürlich und intuitiv gestaltet werden. Gesten bieten dagegen die Möglichkeit einer Interaktion, wie wir es aus der realen Welt gewohnt sind. Jedoch ist die maschinelle Erkennung von Gesten aufgrund der Individualität jeder einzelnen Person und der Komplexität von menschlichen Bewegungsabläufen alles andere als trivial. Wenn 100 verschiedene Personen eine Geste in Form eines Kreises durchführen, hat dies 100 verschiedene Datensätze zur Folge. Die Kreise unterscheiden sich in ihrer Form, Ausführungsgeschwindigkeit, Richtung, Größe und anderen Faktoren, was die Erkennung beziehungsweise Klassifizierung dieser Gesten erschwert. Im Bereich der Gesten- und auch Bilderkennung haben sich künstliche neuronale Netze in den vergangen Jahren als effektive Technologie durchgesetzt und erzielen bahnbrechende Ergebnisse für diese komplexen Probleme [KSH12, NWTN15, MGKK15]. Diese Arbeit untersucht den Einsatz sogenannter neuronaler Faltungsnetzwerke zur Klassifizierung von auf der Microsoft HoloLens durchgeführten Handgesten.

1.1 Abgrenzung und Zielsetzung

Die Klassifizierung von Gesten mit Hilfe von Faltungsnetzwerken ist kein neues Thema und wurde bereits unter anderem in [MGKK15] sowie [NWTN15] untersucht. Im Gegensatz zu [NWTN15] und [MGKK15] stützt sich diese Arbeit aufgrund von technischen Restriktionen der Microsoft HoloLens [GBD+16] nicht auf das RGB-D Datenformat (Bild- oder Videodaten mit Farb- und Tiefeninformationen), sondern auf aufeinanderfolgenden Koordinaten der Handpositionen. Ebenso sind zu Beginn der Arbeit keine Gestendaten vorhanden, die für das Trainieren, Evaluieren oder Testen des neuronalen Netzwerks verwendet werden können. Im Folgenden werden die Ziele dieser Arbeit aufgelistet.

Entwurf eines Datenformats Ein Datenformat, mit dem auf der Microsoft HoloLens durchgeführte Gesten dargestellt werden können, soll definiert werden.

Erstellung einer Datenbasis Daten zum Trainieren, Evaluieren und Testen der implementierten Faltungsnetzwerk sollen erstellt und künstlich vermehrt werden.

Entwurf und Implementierung von Faltungsnetzwerken Es sollen Faltungsnetzwerke entworfen und implementiert werden, welche die Gestenklassifizierung durchführen.

Datenaustausch Eine Schnittstelle für den Datenaustausch zwischen den Faltungsnetzwerken und der HoloLens soll entworfen und implementiert werden.

Erweiterbarkeit Das Gesamtkonzept soll auf andere Technologien als auf die Microsoft HoloLens erweiterbar sein.

Generalisierungsfähigkeit auf andere Personen Gesten, die von anderen Personen als dem Autor durchgeführt werden, sollen mit einer vergleichbar guten Leistung klassifiziert werden können.

Evaluation Durch eine Evaluation sollen weitere Erkenntnisse über die implementierten Faltungsnetzwerke gesammelt werden.

1.2 Aufbau der Arbeit

Die Arbeit teilt sich in vier Teilbereiche. Basierend auf einer ausführlichen literarischen Recherche werden im *ersten Teil* die theoretischen Grundlagen künstlicher neuronaler Netze, Augmented Reality und Gestenerkennung erläutert.

Der *zweite Teil* stützt sich auf die gewonnenen Erkenntnisse des ersten Teils und beschreibt den Entwurf der künstlichen neuronalen Netze sowie die Verfahren zur Datenvermehrung und Vorverarbeitung. Ferner wird das Format,

in dem die Gestendaten abgespeichert werden, entworfen. Anschließend wird die Implementierung der entworfenen Konzepte im *dritten Teil* beschrieben.

Der *vierte Teil* beschreibt die Evaluation der entworfenen und implementierten neuronalen Netze anhand einer empirischen Analyse. Ein Test des Gestenerkennungssystems mit fünf Personen wird durchgeführt, um die Leistung der implementierten Netze im Hinblick auf die Generalisierungsfähigkeit auf andere Personen als dem Autor zu bewerten. Außerdem wird der Lernprozess des Netzwerks analysiert und das Gelernte anhand von Visualisierungen veranschaulicht. Darüber hinaus werden Experimente durchgeführt, um empirisch Informationen über Eigenschaften der implementierten neuronalen Netze zu gewinnen.

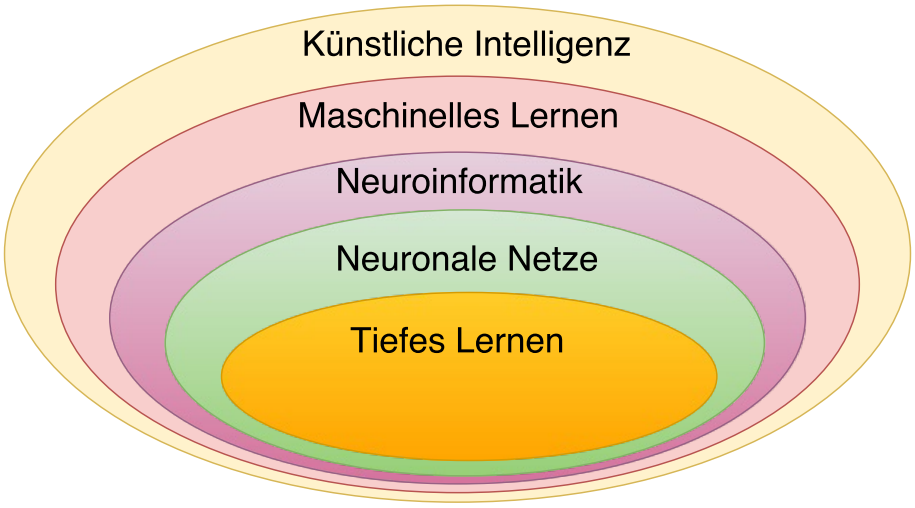

Abbildung 1: Tiefes Lernen und neuronale Netze werden innerhalb des Forschungsgebiets der künstlichen Intelligenz eingeordnet
Quelle: Angelehnt an [SCYE17]

2 Grundlagen

Dieses Kapitel vermittelt die notwendigen Grundlagen, die in der vorliegenden Bachelorarbeit von Bedeutung sind. Die hier aufgeführten Informationen werden in den folgenden Kapiteln als bekannt vorausgesetzt.

2.1 Neuronale Netze

Künstliche neuronale Netzwerke, auch künstliche neuronale Netze genannt, sind ein Modellierungsansatz, der im maschinellen Lernen, einem Teilgebiet der künstlichen Intelligenz, angewendet wird. Tiefes Lernen beschäftigt sich mit neuronalen Netzen mit mehreren Zwischenschichten [Ert16, S. 301]. Abbildung 1 ordnet tiefes Lernen und künstliche neuronale Netze in das Forschungsgebiet der künstlichen Intelligenz ein. In den folgenden Abschnitten werden die Grundlagen von künstlichen neuronalen Netzen erklärt.

2.1.1 Biologischer Hintergrund

> "Das menschliche Gehirn ist ein komplexes neuronales Netzwerk. Seiner Dynamik verdanken wir unsere Fähigkeiten zur Wahrnehmung, Bewegung, Emotion, Kognition und unser Bewußtsein." [Mai97, S. 8]

Kein anderes bekanntes biologisches System ist so leistungsvoll und komplex wie das menschliche Gehirn. Im Folgenden werden die Funktionsweise und Bestandteile eines neuronalen Netzwerks, wie es in unserem Gehirn zu finden ist, skizziert.

Wie in Abbildung 2 dargestellt, besteht eine einzelne Nervenzelle (Neuron) aus einem Zellkörper (Soma), einem Axon und mehreren Dendriten[1]. Der Zellkörper speichert eine interne elektrische Spannung. Elektrische Signale, die über die Dendriten zu dem Zellkörper gelangen, laden diesen auf bis er einen Schwellwert überschreitet und sich über das Axon entlädt. Das Axon wiederum ist über Synapsen mit den Dendriten anderer Neuronen verbunden und lädt diese auf. Durch diese Verbindungen entsteht ein komplexes Netzwerk aus Neuronen [Bra95, S. 33 bis 39] [Ert16, S. 265 bis 270] [Boe07, S.279 f.].

Die Synapsen spielen in der Funktionsweise eines neuronalen Netzes eine zentrale Rolle, denn sie steuern anhand ihrer Leitfähigkeit wie stark sich einzelne Neuronen gegenseitig beeinflussen. Die Leitfähigkeit der Synapsen steuert demnach den Informationsfluss im Netzwerk und definiert dadurch, wie es auf bestimmte Eingaben reagiert. Das Besondere an den Synapsen ist die Variabilität ihrer Leitfähigkeit. Durch Erhöhen oder Vermindern der Leitfähigkeit wird das neuronale Netzwerk lernfähig [Ert16, S. 268]. Der soeben skizzierte biologische Aufbau eines neuronalen Netzwerks ist vereinfacht dargestellt, da eine biologisch exakte Beschreibung den Rahmen dieser Arbeit sprengen würde.

[1] Es befinden sich etwa 86 Milliarden Neuronen in unserem Gehirn [SCYE17]

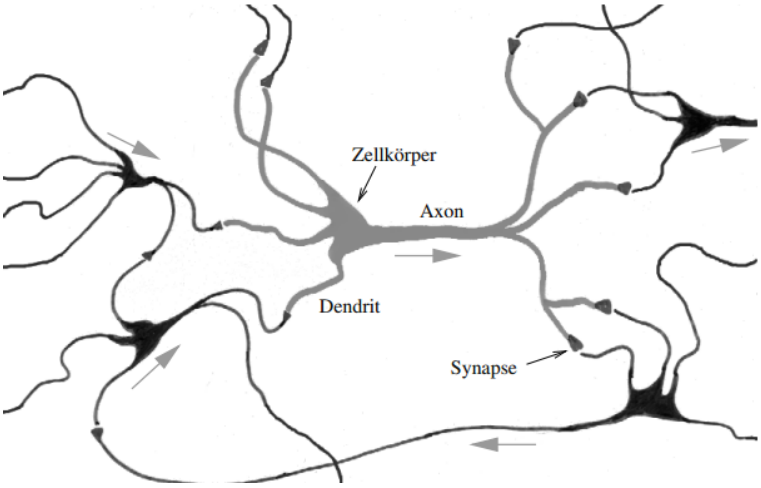

Abbildung 2: Biologisches Modell eines Neurons
Quelle: [Ert16, S. 267]

2.1.2 Formales Modell

Die oben beschriebene vereinfachte Funktionsweise eines neuronalen Netzwerks wurde 1943 erstmals von Warren S. McCulloch und Walter Pitts als Formalmodell festgehalten [MP43]. In diesem Modell wird ein Neuron als

> "[...]eine Art Addierer mit Schwellwert." [Bra95, S. 39]

bezeichnet. McCullochs und Pitts' Veröffentlichung gilt als Startschuss der Neuroinformatik [Ert16, S. 265]. In Abbildung 3 wird ein beispielhafter Aufbau eines Modell-Neurons beschrieben. Die interne Ladung beziehungsweise das Aktivierungspotential u eines Neurons wird durch die gewichtete Summe der Eingabewerte $x_1 \dots x_n$ dargestellt, die Gewichte werden als $w_1 \dots w_n$ bezeichnet. Auf das Ergebnis wird ein sogenannter Bias b addiert [Ert16, S. 269] [Boe07, S. 282 ff.].

$$u = \sum_{i=1}^{n} w_i x_i + b$$

Das Aktivierungspotential u wird durch die Aktivierungsfunktion f_{act} zu einem Aktivitätsniveau beziehungsweise einer Ausgabe α.

$$f_{act}(u) = \alpha$$

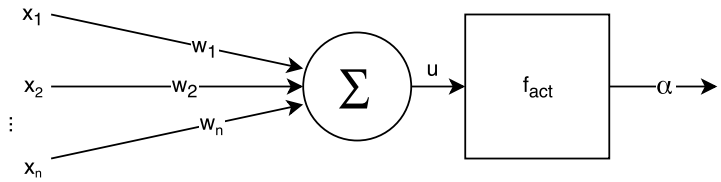

Abbildung 3: Formales Modell eines Neurons
Quelle: Angelehnt an [Boe07, GS03]

Das Aktivitätsniveau α dient wiederum als Eingabewert x_n für andere Neuronen im Netzwerk. In einigen Fällen wird auf das Aktivitätsniveau α noch eine Ausgabefunktion angewandt [Boe07, S. 283]. Diese wird zur Vereinfachung in dieser Erklärung vernachlässigt.

2.1.2.1 Aktivierungsfunktion Wie bereits eingangs erwähnt bringt die Aktivierungsfunktion die summierten Eingabewerte in eine Beziehung mit dem Aktivitätsniveau, beziehungsweise mit der Ausgabe eines Neurons [GDR17]. Beispiele für Aktivierungsfunktionen können Abbildung 4 entnommen werden.

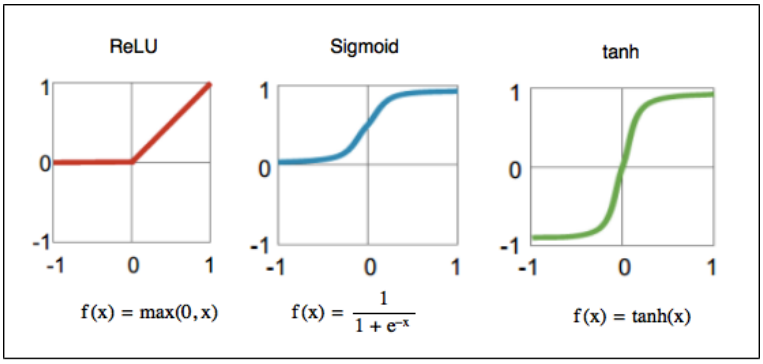

Abbildung 4: Nichtlineare Aktivierungsfunktionen
Quelle: Angelehnt an [SCYE17]

Um ein Problem mit nichtlinearem Zusammenhang zu lösen, wird eine nichtlineare Komponente im Modell benötigt. Die Aktivierungsfunktion bringt diese Nichtlinearität in das Netzwerk [Ada09, S. 1 f.].

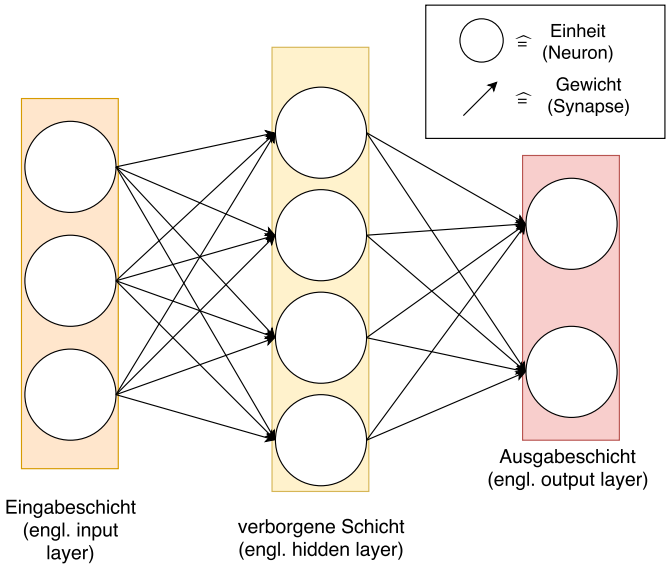

Abbildung 5: Ein Modell eines künstlichen neuronalen Vorwärtsvermittlungsnetzwerks (Feed-Forward-Netzwerk). Diese Klassifizierung ergibt sich aus der Eigenschaft, dass nur Verbindungen von einer Schicht zur nächst höheren Schicht existieren [Boe07, 284]. Es ist also frei von Zyklen [Bra95, S. 54].
Quelle: Angelehnt an [SCYE17]

2.1.3 Grundlegender Aufbau und Netztopologie

Als Ganzes betrachtet kann ein künstliches neuronales Netzwerk als Graph mit Ecken und gerichteten Kanten modelliert werden. Die Synapsen entsprechen den Kanten und die Ecken repräsentieren die Neuronen [Boe07, S. 281]. Im weiteren Verlauf der Arbeit wird das künstliche neuronale Netzwerk als neuronales Netzwerk, die Synapsen als Gewichte oder Verbindungen und die Neuronen als Einheiten bezeichnet. In Abbildung 5 ist ein beispielhafter Aufbau eines neuronalen Netzwerks, mit jeweils einer Eingabe-, Ausgabe- und verborgenen Schicht dargestellt. Einheiten, die den jeweiligen Schichten zugeordnet sind, werden auch Eingabe-, Ausgabe- und verborgene Einheiten genannt. Einheiten, die in Abbildung 5 übereinander abgebildet sind, befinden sich in der gleichen Schicht [Boe07, S. 282] [RW11, S. 15].

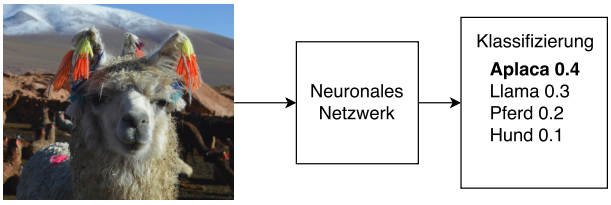

Abbildung 6: Beispiel für einen Klassifizierer
Quelle: Angelehnt an [SCYE17]

Im Folgenden wird die grobe Funktionsweise eines neuronalen Netzwerks und die Aufgaben der einzelnen Schichten anhand eines Beispiels eines Netzwerks zur Klassifizierung von Bildern erläutert. Die Eingabeschicht nimmt Informationen von der Außenwelt auf, was im Beispiel der numerischen Darstellung von Pixeln in einem Bild entspricht. Die Ausgabeschicht gibt eine Antwort auf die Informationen aus der Eingabeschicht [RW11, S. 14]. Auf das Beispiel übertragen ist die Ausgabe des Netzwerks eine Klassifizierung dieses Bildes. Diese Klassifizierung kann durch einen Vektor mit Punktzahlen für jede einzelne Klasse abgebildet werden. Jede Punktzahl in dem Ausgabevektor entspricht einer Wahrscheinlichkeit, nach der die Eingabe zu einer bestimmten Klasse gehört [SCYE17]. Das beschriebene Beispiel ist in Abbildung 6 skizziert. Im Gegensatz zu der Eingabeschicht l_0 und Ausgabeschicht l_n, die in der Regel jeweils nur einmal in der Netzwerkarchitektur vorkommen, können keine aber auch mehrere verborgene Schichten $l_1 \dots l_{n-1}$ vorhanden sein [RW11, S. 15]. Die eigentliche Informationsverarbeitung findet in den verborgenen Schichten und durch die dort vorhandenen verborgenen Einheiten statt [Boe07, S.282].

2.1.4 Lernen

Gewichtungen zwischen den Einheiten bringen zum Ausdruck wie stark sich die Einheiten gegenseitig erregen oder hemmen [RW11, S. 15]. Das Netzwerk lernt gewöhnlich, indem es während eines sogenannten Trainingsprozesses selbstorganisierend diese Gewichtungen und die Biase optimiert [SCYE17] [RW11, S. 15]. Grundsätzlich wird zwischen den folgenden Lernkonzepten unterschieden [RW11, S. 26]:

Überwachtes Lernen Während des Lernprozesses wird das Netzwerk mit Eingaben gespeist, deren korrekte Ausgaben (Zielwerte) bekannt sind. Sind die Zielwerte gleich der Ausgaben des Netzwerks, so lag das Netzwerk richtig. Der Netzwerkfehler beschreibt, wie sehr das Netzwerk falsch liegt. Ziel des Lernprozesses ist, diesen Wert zu minimieren [SCYE17]. Es gibt verschiedene Funktionen zur Bildung dieses Wertes. Eine anschauliche Erklärung zu verschiedenen Fehlerfunktionen ist in [LKJ] aufgeführt.

Reinforcement Learning Ähnlich wie beim überwachten Lernen wird dem Netzwerk während des Lernprozesses eine Rückmeldung zu einer Ausgabe gegeben. Im Gegensatz zum überwachten Lernen beschränkt sich diese jedoch auf einen Skalar, der als Belohnung oder als Kosten interpretiert werden kann.

Unüberwachtes Lernen Dem Netzwerk ist die richtige Ausgabe nicht bekannt. Das Ziel besteht darin, Strukturen oder Muster in den Daten zu erkennen und diese für das Lernen zu nutzen [SCYE17].

Direkter Entwurf Die Gewichte sind festgelegt. Es findet also kein Lernprozess im eigentlichen Sinne statt.

Nachdem das Netzwerk trainiert wurde, sollte es in der Lage sein, Eingaben zu bearbeiten, die es vorher noch nicht gesehen hat. Diese Eigenschaft eines neuronalen Netzwerks wird Generalisierungsfähigkeit genannt. Ist das Netzwerk zu sehr an die Daten gewöhnt, mit denen es trainiert wurde, spricht man von Überanpassung (engl. overfitting). Das Netzwerk hat in diesem Fall zu einem mehr oder weniger hohen Grad die Trainingsdaten auswendig gelernt und kann keine gute Ergebnisse für Daten erzielen, die es noch nicht zuvor gesehen hat. Das Ziel während des Lernprozesses ist es also, eine gute Generalisierungsfähigkeit zu erreichen und eine Überanpassung zu verhindern [Rei]. Die Anwendung eines trainierten Netzwerks wird als Inferenz beziehungsweise Anwendungsphase bezeichnet [SCYE17] [Boe07, S. 293]. Eine Lernregel definiert während des Lernprozesses, wie stark welche Gewichte angepasst

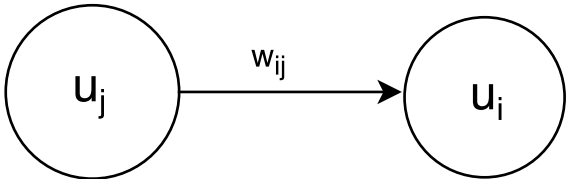

Abbildung 7: Veranschaulichung der Indizes für Einheiten und für die Gewichtungen zwischen den Einheiten

werden [RW11, S. 33 f.]. Im Folgenden wird näher auf zwei Lernregeln eingegangen.

2.1.4.1 Delta-Regel Die Delta-Regel, auch als Windroff-Hoff-Regel bekannt, basiert auf einem Vergleich aus dem gewünschten und tatsächlichen Ergebnis. Die Funktionsweise wird anhand der folgenden Gleichung veranschaulicht.

$$\Delta w_{ij} = \varepsilon \delta_i \alpha_j$$

Δw_{ij} steht für die Höhe der Gewichtsveränderung der Verbindung zwischen den Einheiten u_i und u_j. Die Lernrate ist durch ε definiert und dient als Konstante, die vor der Trainingsphase definiert wird und beeinflusst, wie stark die Gewichte angepasst werden sollen. δ_i ist die Differenz zwischen dem gewünschten und tatsächlichen Aktivitätsniveau α der Einheit u_i. Durch α_j wird das Aktivitätsniveau der sendenden Einheit u_j beschrieben. Ein großes α_j bewirkt also, dass die Verbindung zu der Einheit u_j stärker verändert wird [RW11, S. 37 f.] [GS03, S. 376 bis 378]. Eine Veranschaulichung der Indizes kann Abbildung 7 entnommen werden.

Um die Delta-Regel anwenden zu können, muss die korrekte Ausgabe der Einheit u_i bekannt sein. Nur so kann δ_i berechnet werden. Wie oben erwähnt, liegt diese korrekte Ausgabe im Falle des überwachten Lernens vor. Diese Eigenschaft beschränkt den Einsatz der Delta-Regel jedoch auf Netzwerke ohne verborgene Schichten.

2.1.4.2 Backpropagation

Mit Hilfe des Backpropagation Algorithmus kann, im Gegensatz zu anderen Verfahren, für die verborgenen Einheiten eine Gewichtsveränderung ermittelt werden [RW11, S. 51]. Mit Hilfe von Backpropagation können die Grundideen der Delta-Regel auf Netzwerke mit einer oder mehreren versteckten Schichten angewendet werden [GS03, S. 377 f.]. Die Fehler aus der Ausgabeschicht werden dabei abgeleitet und an die verborgenen Schichten zurückpropagiert. Gewichtungen unterhalb der Ausgabeschicht können somit, basierend auf der Delta-Regel, ebenfalls angepasst werden [Boe07, S. 294]. Das Verfahren teilt sich in die folgenden drei Schritte auf.

Forward-pass Die Ausgaben jeder einzelnen Einheit, von der Eingabe- bis hin zur Ausgabeschicht, werden ermittelt.

Fehlerbestimmung Die Fehler der Ausgabeeinheiten werden bestimmt, indem der Ausgabevektor des Netzes mit den Zielwerten verglichen wird.

Backward-pass In diesem Schritt werden die Gewichtungen der Verbindungen im Netzwerk angepasst. Die ermittelten Fehler breiten sich rückwärts von der Ausgabe- bis zur Eingabeschicht aus. Das sogenannte Gradientenabstiegsverfahren bestimmt dabei, in welchem Maße die Gewichte verändert werden.

Die oben skizzierten Schritte werden in den meisten Fällen wiederholt, bis der Fehler einen zufriedenstellenden Wert angenommen hat oder eine definierte Anzahl an Trainingsschritten erreicht wurde [RW11, S. 52]. Eine mathematische Definition und Erklärung des Gradientenabstiegsverfahrens kann im Umfang dieser Arbeit nicht behandelt werden. Für eine genaue Beschreibung wird auf [RW11, S. 39 - 54] sowie [Boe07, S. 293 - 310] verwiesen.

2.1.5 Neuronale Faltungsnetzwerke

Neuronale Faltungsnetzwerke (engl. convolutional neural networks, CNN), oder einfach Faltungsnetzwerke, sind eine spezielle Form von neuronalen

Netzwerken. Grundsätzlich besteht ein Faltungsnetzwerk aus einer oder mehreren Faltungsschichten, gefolgt von einer oder mehreren vollvernetzten Schichten, die gestapelt aufgebaut sind. Faltungsnetzwerke folgen dem Prinzip der Vorwärtsvermittlungsnetze. Somit baut eine Schicht l_i auf der Ausgabe der vorherigen Schicht l_{i-1} auf.

Jede Faltungsschicht beinhaltet Filter gleicher Größe, die in den meisten Fällen kleiner als die Eingabe der Faltungsschicht sind. Die Filter extrahieren Eigenschaften aus der Eingabe und halten die Ergebnisse dieses Verfahrens in sogenannten Merkmalskarten (engl. feature maps) fest. Jedem Filter ist eine Merkmalskarte zuzuordnen, in der zu sehen ist an welchen Stellen der Filter angeschlagen hat. Meist werden die Filter mit Zufallswerten initialisiert und lernen Parameter, nach denen Sie die Eingabe abtasten [CMM+11] [IG16, S. 327 - 330]. Eine detaillierte Beschreibung der Funktionsweise von Faltung würde den Rahmen der Arbeit sprengen und wird in [SCYE17] aufgeführt. Am Ende der Faltungsschichten folgen eine oder mehrere vollvernetzte Schichten und führen die Klassifizierung durch. Neben den Faltungs- und vollvernetzten Schichten werden weitere optionale Schichten in Architekturen von Faltungsnetzwerken eingesetzt.

Nichtlinearität Bereits in Abschnitt 2.1.2.1 wurden verschiedene Aktivierungsfunktionen und deren Nutzen beschrieben. Da es sich bei der Faltung um eine lineare Operation handelt, wird typischerweise eine Nichtlinearität auf die Ausgabe einer Faltungsschicht angewandt. Die Anwendung dieser Nichtlinearität findet in einer sogenannten Nichtlinearitätsschicht statt [LKJ]. In [LKJ] wird die Verwendung von ReLU (engl. rectified linear unit) Aktivierungsfunktionen für Faltungsnetzwerke empfohlen. In [KSH12] konnte durch den Einsatz von ReLUs eine Beschleunigung des Lernprozesses um den Faktor 6 im Vergleich zu einer klassischen $f(x) = tanh(x)$ Aktivierungsfunktion erreicht werden.

Pooling Das Pooling-Verfahren wird eingesetzt, um die Informationen einer Schicht zu komprimieren. Die Größe der zu verarbeitenden Daten kann

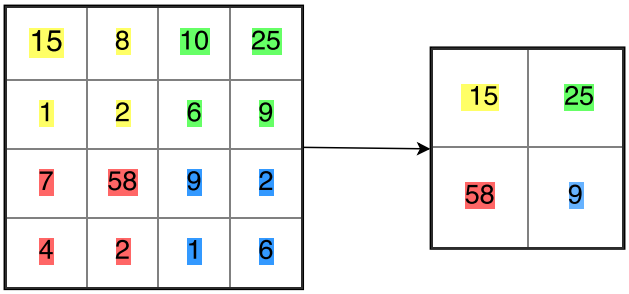

Abbildung 8: 2x2 Max Pooling mit einer Schrittgröße von 2
Quelle: Angelehnt an [LKJ]

mit Hilfe von Pooling mit der Tiefe eines Faltungsnetzwerks minimiert werden. Des Weiteren wird die Wahrscheinlichkeit einer Überanpassung des Netzwerks reduziert. Hierzu gibt es mehrere Ansätze, wie beispielsweise das Max Pooling, dass nur den höchsten Wert innerhalb eines definierten Bereichs übernimmt. Das Verfahren wird in Abbildung 8 veranschaulicht [IG16, S. 335 f.].

2.2 Augmented Reality

Schon im Jahre 1997 wurde Augmented Reality (AR) als

> "[...]next generation, reality-based interface[...]" [JEW+97]

bezeichnet. AR ergänzt die reale Welt mit virtuellen Objekten und Informationen, wobei sich die digitale und die reale Welt vermischen [vKP10]. AR-Technologien können dem Benutzer Informationen über die reale Welt in Echtzeit bereitstellen. Dies birgt interessante Einsatzmöglichkeiten, von denen einige bereits in einer Veröffentlichung von Ronalt T. Azuma [Azu97] im Jahre 1997 aufgezählt wurden. In der gleichen Veröffentlichung wird ein AR-System anhand von drei Eigenschaften definiert:

> "[...]this survey defines AR as systems that have the following three characteristics:
>
> 1. Combines real and virtual

2. Interactive in real time

3. Registered in 3-D"

.

2.2.1 Microsoft HoloLens

Die Microsoft HoloLens ist eine Augmented-Reality-Brille (engl. augmented reality smart glass). Der Begriff Augmented Reality Smart Glass ist in [RBR15] wie folgt definiert:

> "Augmented Reality Smart Glasses are defined as wearable Augmented Reality (AR) devices that are worn like regular glasses and merge virtual information with physical information in a user's view field."

Die HoloLens projiziert virtuelle Objekte (Hologramme) in das Sichtfeld des Benutzers, welche ihm Informationen bereitstellen. Der Benutzer kann anhand von Spracheingabe, Blickrichtung und Gesten mit den Hologrammen interagieren [Micc].

2.2.2 Entwicklung einer Augmented-Reality-Applikation

Die Entwicklung von AR-Anwendungen wird durch die Nutzung einer Spiele-Engine wie Unity erheblich vereinfacht. Unity bietet die Möglichkeit, dreidimensionale Animationen und Grafiken (Spielobjekte) zu erstellen und diese mittels sogenannter Skripte zu steuern. Ein komplettes Unity-Programm besteht aus einer oder mehreren Szenen. Eine Szene kann als ein Level in einem Spiel angesehen werden, dass anhand von Spielobjekten und Spiellogik zum Leben erweckt wird. Zu Testzwecken kann das Programm mit Hilfe des Vorschaumodus (engl. play mode) direkt in der Unity-Entwicklungsumgebung ausgeführt werden. Das Programm muss in diesem Fall also nicht auf die Zielplattform gebracht werden [Tec].

Abbildung 9: Milgrams und Kishinos *Mixed Reality Continuum*
Quelle: [PM95]

2.2.3 Abgrenzung zu Virtual Reality

Milgrams und Kishinos *Mixed Reality Continuum* [PM95] ordnet AR und Augmented Virtuality (AV) als Mixed Reality zwischen der realen und virtuellen Welt ein. Wie in Abbildung 9 zu erkennen ist, lehnt sich AR mehr an die reale und AV mehr an die virtuelle Welt an. Bei AV werden, im Gegensatz zu AR, echte Objekte in eine virtuelle Welt projiziert [Fur11]. In Virtual Reality (VR) taucht der Benutzer komplett in eine virtuelle Welt ein, wobei die reale Welt um den Benutzer herum nicht mehr wahrzunehmen ist [Azu97].

2.3 Gestenerkennung

Gesten sind ein natürliches Werkzeug, um auf einfache Art und Weise mit Computern zu interagieren [AW12]. Im Bereich der Mensch-Maschinen-Interaktion (MMI, engl. human computer interaction) werden Gesten wie folgt definiert:

> "Gestures are expressive, meaningful body motions involving physical movements of the fingers, hands, arms, head, face, or body with the intent of: 1) conveying meaningful information or 2) interacting with the environment." [MA07]

Die Gestenerkennung beschäftigt sich mit der Interpretation und Klassifizierung der von Menschen durchgeführten Gesten [MA07]. Der Prozess der Klassifizierung kann dabei wie folgt beschrieben werden. Ein unbekanntes Ereignis oder Datum ω wird einer endlichen Menge Ω von Ereignissen oder

Daten zugeordnet. Diese Menge wird im Folgenden auch als Gestenraum oder Vokabular bezeichnet [Kra06, S. 29].

$$\omega \in \Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$$

Der Prozess der Gestenerkennung kann in die in Abbildung 10 aufgeführten Schritte zerlegt werden. Nachdem die Geste durchgeführt und aufgenommen wurde, folgt der Schritt *Eigenschaften extrahieren*. In diesem Schritt werden die aufgenommenen Gestendaten in numerisch repräsentierte Eigenschaften, dem Eigenschaftsvektor, umgewandelt. Im folgenden Schritt wird die Geste anhand dieser Eigenschaften klassifiziert. Anhand der Klassifizierung kann dann eine bestimmte Aktion ausgelöst werden [Kra06, S. 7 f.]. Im Folgenden werden verschiedene Aspekte aufgeführt, die für den Entwurf eines Gestenerkennungssystems wichtig sind [Kra06, S. 8 ff.].

Vokabular beziehungsweise Gestenraum Ω definiert die Menge an Gesten, die von dem System klassifiziert werden können. Es kommt jedoch nicht nur auf die Anzahl der zu klassifizierenden Gesten an, sondern der Grad, in dem sich die einzelnen Gesten ähneln, ist entscheidend. Ferner ist es von Bedeutung, wie einfach oder schwer sich das Vokabular erweitern lässt.

Aufnahmebedingungen Grundsätzlich wird zwischen verschiedenen Formen der Aufnahme unterschieden. Im Bereich der Handgesten sind zwei der am häufigsten verwendeten Methoden visuell und anhand von Datenhandschuhen. Im visuellen Bereich haben vor allem Parameter wie der Bildhintergrund, die Distanz zur aufnehmenden Kamera oder die Lichtbedingungen einen Einfluss [MRH15].

Datengrundlage Die Datengrundlage bezeichnet das Format, in dem die Gestendaten repräsentiert werden. Ein verbreitetes Datenformat, dass auch in [MGKK15] sowie [NWTN15] verwendet wurde, ist das RGBD-Format, da es neben den Farb- zusätzlich Tiefeninformationen liefert [Lit07].

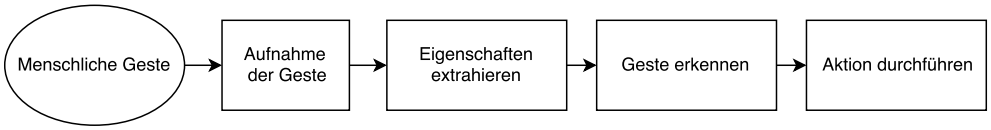

Abbildung 10: Schritte der Gestenerkennung
Quelle: Angelehnt an [AW12]

Leistung Die Leistung eines Gestenerkennungssystems kann anhand der Latenz und Genauigkeit gemessen werden. Die Latenz in diesem Kontext ist die Zeitdauer zwischen der Ausführung der Geste und der Ausgabe der Klassifizierung. Die Genauigkeit hingegen misst, wie oft das System richtig oder falsch klassifiziert. Im Zusammenhang mit AR und VR spielt das Thema Latenz eine wichtige Rolle. Nach [KBLH03] und [Bry96] sollte die Antwortzeit auf eine Benutzereingabe im VR-Umfeld nicht über 100 Millisekunden liegen [Kra06, S. 263].

Auf die Verfahren und Algorithmen, die über die Betrachtung neuronaler Netzwerke hinausgehen, die zur Erkennung und Klassifizierung von Handgesten eingesetzt werden, kann im Rahmen dieser Arbeit nicht weiter eingegangen werden. Weitere Verweise zu den klassischen Verfahren finden sich in Abschnitt 2.4 und in [Kra06].

2.4 Verwandte Arbeiten

Neben neuronalen Netzwerken werden auch andere Lösungsansätze zur Klassifizierung von Handgesten eingesetzt. Unter anderem sei auf das Hidden-Markov-Modell [Rab89, SWP98, SP95, YOI92] und auf den Einsatz von endlichen Automaten [HTH00, DS94] hingewiesen. Eine detaillierte Zusammenfassung dieser und weiterer Verfahren zur Gestenerkennung ist in [MA07] aufgeführt. Heute haben sich jedoch neuronale Netzwerke als effektives Werkzeug für den Einsatzzweck der Gestenerkennung etabliert. Die Gewinner des VIVA (engl. vision for intelligent vehicles and applications) Wettbewerbs [MGKK15] setzten, wie auch die Gewinner des ChaLearn 2014 (engl. looking at people challenge) Wettbewerbs [NWTN15] ein dreidimensionales Faltungsnetzwerk zur Gestenerkennung ein. Wie bereits in der Einleitung

erwähnt, setzen diese Konzepte auf eine Darstellung der Gesten im RGB-D Datenformat. Ohne Zusatzhardware kann die Microsoft HoloLens die Gestendaten nicht in diesem Format aufnehmen [GBD+16, FKM17].

Die in der vorliegenden Arbeit entworfenen und implementierten Faltungsnetzwerke stützen sich auf Kenntnisse, die in anderen Arbeiten gewonnen wurden. So wurden in dieser Arbeit sogenannte ReLU-Aktivierungsfunktionen [NH10] verwendet. Die Gewinner der ImageNet Large-Scale Visual Recognition Challange (ILSVRC) 2012 [KSH12] konnten dadurch den Lernprozess um den Faktor 6, im Vergleich zu einer klassischen $f(x) = tanh(x)$ Aktivierungsfunktion beschleunigen. Neben der ReLU-Aktivierungsfunktionen wurde, wie in [KSH12], das sogenannte Dropout-Verfahren [SHK+14] verwendet, um einer Überanpassung entgegen zu wirken. Durch die Verwendung des Adam-Verfahrens [KB14], einer Erweiterung des klassischen Gradientenabstiegsverfahrens, konnten die implementierten Faltungsnetzwerke um ein Vielfaches schneller trainiert werden. Aufgrund der geringen Menge an Daten wurden, wie auch in [KSH12, KTS+14, SZ14], Verfahren zur Datenvermehrung eingesetzt. Die Implementierung wurde anhand der Tensorflow-Bibliothek [AAB+16] durchgeführt.

Arbeiten wie [FKM17, PTJ17] untersuchen das Thema der Erkennung von benutzerdefinierten Gesten anhand der Microsoft HoloLens, konnten jedoch ohne die Verwendung von Zusatzhardware keine Lösung aufzeigen.

3 Entwurf

In Abschnitt 3.1 werden zunächst die Gesten definiert, die vom neuronalen Netzwerk klassifiziert werden sollen. Weiterhin wird in Abschnitt 3.2 der Aufbau einer Basis an Daten von diesen Gesten entworfen. In den Abschnitten 3.3 bis 3.6 werden die Verfahren zur Vorverarbeitung, Diskretisierung, sowie die Vermehrung und Aufteilung dieser Daten beschrieben. Anschließend wird das neuronale Netzwerk zur Klassifizierung in Abschnitt 3.7 entworfen. Der Entwurf richtet sich nach den in der Einleitung definierten Zielen.

3.1 Gesten

Für den Aufbau einer Datenbasis werden vier verschiedene Gesten entworfen, die von den neuronalen Netzwerken klassifiziert werden sollen. Die vier verschiedenen Gesten bilden den Gestenraum Ω, sodass gilt: $|\Omega| = 4$. Abbildung 11 skizziert jeweils ein Beispiel einer Geste aus jeder Gestenklasse. Zu beachten ist, dass eine Geste unabhängig von der Ausführungsgeschwindigkeit und der Ausrichtung zu der gleichen Klasse gehört.

3.2 Aufbau einer Datenbasis

Wie in der Einleitung beschrieben, ist keine Datenbasis vorhanden, um das neuronale Netzwerk zu trainieren, zu evaluieren oder zu testen. Die Menge an Daten, welche die Datenbasis bilden, wird im folgenden als A bezeichnet.

$\forall \omega \in A$ gilt: ω ist einem Element aus dem Gestenraum Ω zuzuordnen.

Es sollen insgesamt 600 echte Gestendaten erstellt werden, die A bilden. Dabei soll jede Gestenklasse aus Ω 150 Mal vorhanden sein. Da die zu klassifizierenden Handgesten auf der Microsoft HoloLens durchgeführt werden, wird die Datenbasis unter Verwendung jener Hardware aufgebaut. Im Folgenden wird aufgeführt, welche Daten von der HoloLens extrahiert werden können und wie sie dargestellt werden.

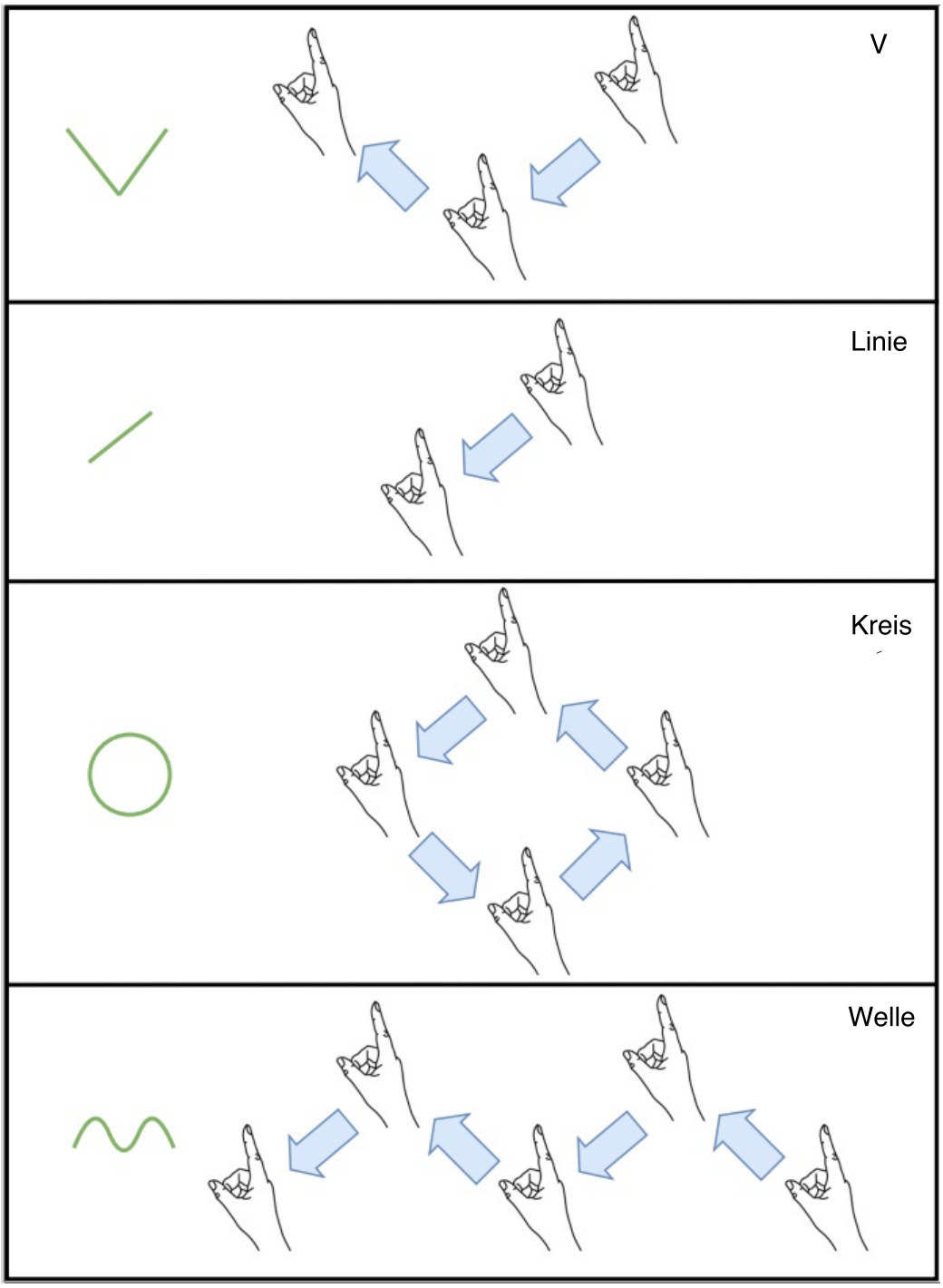

Abbildung 11: Beispiele von Gesten der vier Klassen
Quelle: Angelehnt an [Micb]

3.2.1 Daten der HoloLens

Die HoloLens stellt eine Schnittstelle bereit, mit der die aktuellen Koordinaten (x,y,z) einer Hand im Raum abgefragt werden können. Zu beachten ist jedoch, dass diese Koordinaten nur abrufbar sind, wenn sich die Hand in der sogenannten Gesture-Ready-Position befindet. Ist die Hand nicht in dieser Position, sind die Koordinaten nicht abrufbar. Eine Hand in der Gesture-Ready-Position ist in Abbildung 12 dargestellt.

Abbildung 12: Die Gesture-Ready-Position einer Hand
Quelle: [Micb]

Des Weiteren muss sich die Hand im sogenannten Gesture Frame befinden, welcher die Reichweite der Sensoren der HoloLens bezeichnet. Alles, was außerhalb der Sensorenreichweite liegt, kann nicht erfasst werden [Micb].

Die abrufbaren Koordinaten der HoloLens-Schnittstelle entsprechen den Handpositionen im Raum und beziehen sich nicht auf die Handposition innerhalb des Gesture Frames. Die Koordinaten beschreiben also eine absolute Position im Raum und keine Position, die relativ zur Kopfposition des Benutzers ist. Dies hat zur Folge, dass Bewegungen der Person im Raum erhebliche Auswirkungen auf die abzurufenden Koordinaten und dadurch auf die Gestendaten haben.

3.2.2 Darstellung der Gestendaten

Ein Datenmodell für die Darstellung der Gestendaten wird im Folgenden definiert. Ein einzelner Gestendatensatz ω besteht aus einer *Klassifizierung* und aus einer *Koordinatenliste*.

Die *Klassifizierung* wird durch einen Vektor dargestellt, der die Geste einer Klasse aus dem Gestenraum Ω zuordnet. Der Vektor hat die Eigenschaft, dass genau ein Wert 1 ist und alle anderen Werte 0 sind[2]. Der Index, an dem die 1 vorkommt, ordnet die Geste der entsprechenden Klasse in Ω zu.

Die *Koordinatenliste* repräsentiert die Geste anhand der extrahierten Handpositionen in Form von aufeinanderfolgenden dreidimensionalen Koordinaten. Mehrere Gestendaten werden in einer Gestenliste zusammengefasst und mit Hilfe des JSON-Formats[3] dargestellt. Im Folgenden ist eine exemplarische Darstellung der Gestendaten im JSON-Format zu sehen.

```
1 {
2   "gestureList": [
3     {
4       "gestureClass": [1, 0, 0, 0],
5       "gestureDescription": [{"x": 0.1, "y":0.9, "z":0.5}, ...]
6     }, ...
7   ]
8 }
```

3.3 Vorverarbeitung der Daten

Um möglichst optimale Ergebnisse zu erzielen, werden die Daten, bevor sie in das neuronale Netzwerk eingespeist werden, aufbereitet. Folgende Maßnahmen zur Datenvorverarbeitung wurden getroffen.

Manuelle Datensichtung Während der Aufnahme einer Geste kann es zu Fehlern kommen, was unbrauchbare Datensätze zur Folge hat. Daten

[2] In der Literatur wird hierfür auch der Terminus One-Hot-Kodierung gebraucht.
[3] `https://www.w3schools.com/js/js_json_intro.asp`

sollen visualisiert und auf diese Einträge untersucht werden. Die gesichteten fehlerhaften Gestendaten sollen manuell aus der Datenbasis gelöscht werden.

Reparatur Aufgrund von Fehlern während der Aufnahme einer Geste und anderer technischer Restriktionen ist es möglich, dass zwischen zwei aufeinanderfolgenden Koordinaten eine relativ große räumliche Distanz liegt. Dies hat eine Lücke in der Koordinatenliste zur Folge. Es soll eine Methode entwickelt werden, die diese Lücke durch Hinzufügen von künstlich erzeugten Koordinaten füllt.

3.4 Diskretisierung der Daten

> "Discretization is the process of putting values into buckets so that there are a limited number of possible states." [Mica]

Nach der Aufnahme liegen die Gestendaten als Liste von Koordinaten vor. Eine Diskretisierung der Daten muss implementiert werden, um die kontinuierliche Liste von Koordinaten auf einer Ebene mit Kacheln fester Größe abzubilden. Durch die Diskretisierung entsteht eine Datenstruktur, die durch ein Faltungsnetzwerk ausgewertet werden kann. Zunächst wurde eine Methode zur Diskretisierung von zweidimensionalen Koordinaten entworfen, welche die Tiefeninformationen (z-Werte) der Koordinaten ausblendet. Dieser Entwurf wurde anschließend auf eine Methode zur Diskretisierung von dreidimensionalen Koordinaten erweitert.

Die Kanten der Ebene werden in 30 gleich große Einheiten eingeteilt, wodurch eine quadratische Ebene mit 900 gleich großen quadratischen Kacheln entsteht. Die Diskretisierungsmethode bildet jede Koordinate auf eine dieser Kacheln ab. Jede Kachel auf der diskreten Ebene wird dadurch zu einem Abbild eines Teilbereichs der Koordinatenwerte einer Geste. Die entstehende Abbildung weist weder surjektive noch injektive Eigenschaften[4] auf.

[4]Mehrere Koordinaten können auf die gleiche Kachel abgebildet werden. Nicht jede Kachel in der Ebene muss ein Abbild von Koordinaten sein.

Die entworfene Datendiskretisierung kann auf eine dreidimensionale Methode erweitert werden. Die Koordinaten werden dann auf einen Raum mit Würfeln anstatt auf einer Ebene mit Kacheln abgebildet.

3.5 Aufteilung der Daten

Bei der Bewertung der Leistung eines neuronalen Netzwerks sollte auf die Generalisierungsfähigkeit und auf die Überanpassung geachtet werden (Abschnitt 2.1.4). Um dies zu erreichen, wird die Datenbasis in drei Kategorien unterteilt.

- Trainingsdaten werden, wie der Name schon andeutet, für das Training des neuronalen Netzwerks verwendet.
- Evaluierungsdaten werden für die Überwachung des Trainingsprozesses eingesetzt.
- Testdaten werden verwendet, um die Leistung eines Netzwerks nach dem Training zu messen. Es handelt sich um Daten, die das Netzwerk noch nicht zuvor gesehen hat.

Die Datenbasis wird zunächst in ein Drittel Testdaten und zwei Drittel Trainings- und Evaluierungsdaten aufgeteilt. Nach der künstlichen Datenvermehrung (Abschnitt 3.6) werden 600 der vermehrten Datensätze als Evaluierungsdaten definiert. Es wird sichergestellt, dass jede der vier Gestenklassen gleich oft in den 600 Evaluierungsdatensätzen vorkommt. Eine zweite Menge an Testdaten mit sogenannten *Fremdgestendaten* beeinhaltet 185 weitere Datensätze. *Fremdgestendaten* bezeichnen Daten von Gesten, die von anderen Personen als dem Autor durchgeführt wurden. Somit wird sichergestellt, dass die Leistung des Netzwerks auch im Hinblick auf die Generalisierungsfähigkeit auf andere Personen bewertet werden kann.

3.6 Künstliche Datenvermehrung

Die Effektivität eines neuronalen Netzwerks steigt mit der Anzahl an Daten, die zur Verfügung stehen [JW17]. Wie zuvor beschrieben, gibt es im Rahmen dieser Arbeit keine initiale Datengrundlage, was zur Folge hat, dass die Daten manuell erzeugt werden müssen. Es ist jedoch nicht praktikabel, eine ausreichend große Anzahl an Daten zum Trainieren und Bewerten des Netzwerks in realer Interaktion mit der verwendeten Augmented-Reality-Brille selbst zu erzeugen. Wie in Abschnitt 3.2 erwähnt wurden auf diesem Weg insgesamt nur 600 Datensätze erzeugt, wovon 200 für das Testen des Netzwerks verwendet werden. Die 400 restlichen Datensätze sind Trainings- und Evaluierungsdaten. Aus diesem Grund ist es unabdingbar, Methoden zur Verzerrung, Verschiebung, Spiegelung und zufallsbasierten Verrauschung der Gestendaten zu entwickeln, um die 400 Trainings- und Evaluierungdaten zu vermehren.

3.7 Topologie des neuronalen Netzwerks

Zunächst werden die Begriffe Topologie, Architektur und Hyperparameter für diese Arbeit erklärt und abgegrenzt. Der Begriff Architektur verweist auf den groben Aufbau, also auf die Grundstruktur des Netzwerks. Gemeint ist damit die Art und Reihenfolge der einzelnen Schichten im Faltungsnetzwerk. Hyperparameter hingegen beschreiben Werte, mit denen die Funktion des Netzwerks optimiert und geändert werden kann, ohne dass die Grundstruktur verändert wird. Die Größe der einzelnen Schichten ist beispielsweise ein Hyperparamter und nicht Bestandteil der Architektur. In der Literatur und in Veröffentlichungen konnte keine einheitliche Definition der beiden Begriffe im Kontext neuronaler Netzwerke gefunden werden. Der Begriff Topologie definiert die Architektur unter Berücksichtigung der Hyperparameter. Das Hauptaugenmerk der Architektur des entworfenen Faltungsnetzwerks ist auf Einfachheit, Verständlichkeit und Nachvollziehbarkeit gerichtet.

Aus Gründen der Einfachheit wird zunächst ein Netzwerk mit zweidimensionaler Faltung entworfen. Das erarbeitete Konzept wird dann auf ein Netzwerk

mit dreidimensionaler Faltung erweitert. Im Folgenden wird der zweidimensionale Entwurf und die dreidimensionale Erweiterung im Hinblick auf die in der Einleitung definierten Ziele entworfen.

3.7.1 Art des Netzwerks

Wie in Abschnitt 2.4 aufgeführt, haben sich Faltungsnetzwerke im Bereich der Bild- und Gestenklassifizierung durchgesetzt. Ein Hauptgrund stellt die Größe der Eingabedaten in diesen Bereichen dar. In einem durchgängig vollvernetzten Netzwerk wächst die Anzahl der Parameter mit großen Eingabedaten rasch an, was lange Trainings- und Berechnungszeiten sowie eine potentielle Überanpassung zur Folge hat [LKJ]. Der Zusammenhang zwischen der Größe der Eingabedaten und der Anzahl an Parametern in einem vollvernetzten Netzwerk wird in [LKJ] weiter ausgeführt. Weitere Gründe für den Einsatz eines Faltungsnetzwerks werden in den in Abschnitt 2.4 zitierten Arbeiten genannt.

3.7.2 Architektur

Das zu implementierende Faltungsnetzwerk besteht aus elf Schichten, deren Anordnung in Abbildung 13 veranschaulicht wird. Die diskretisierten Datensätze werden von der Eingabeschicht entgegengenommen und von den verborgenen Schichten verarbeitet. Die Klassifizierung der Datensätze kann der Ausgabeschicht in Form von Vektoren der Länge $|\Omega|$ entnommen werden.

Die verborgenen Schichten bestehen neben einer sogenannten Dropout- (Abschnitt 3.7.3), ReLU-[5] und vollvernetzten Schicht aus zwei sogenannten Faltungsstapeln. Ein Faltungsstapel beschreibt die Abfolge von jeweils einer Faltungs-, Pooling- und ReLU-Schicht. Der beschriebene Aufbau des zu implementierenden Faltungsnetzwerks richtet sich nach dem in [LKJ] aufgeführten Entwurfsmuster für Schichten in Faltungsnetzwerken.

[5] Die Anwendung der ReLU-Aktivierungsfunktion wird, wie in [LKJ], als Schicht aufgeführt.

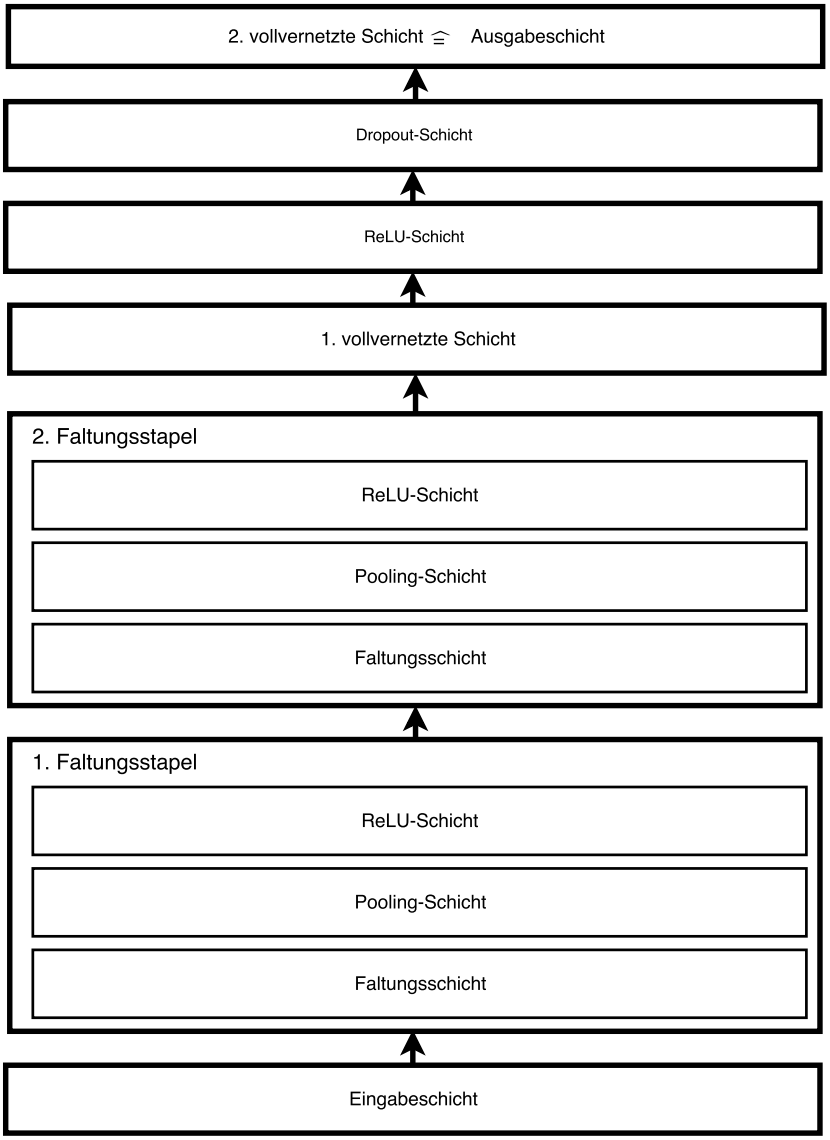

Abbildung 13: Veranschaulichung der Architektur des zu implementierenden Faltungsnetzwerks

3.7.3 Hyperparameter

In Abschnitt 3.7.2 wurde der Begriff Hyperparameter bereits beschrieben. Bei der Implementierung sollte darauf geachtet werden, dass diese mit wenig Aufwand geändert werden können. Nachfolgend werden die Hyperparamter aufgelistet und entworfen.

Größe der Eingabeschicht Die Größe der Eingabesschicht beziehungsweise die Anzahl der Eingabeeinheiten richtet sich nach der Größe des diskretisierten Datensatzes. Jedem Feld in der diskretisierten Geste wird eine Einheit der Eingabeschicht zugeordnet. Durch das Ändern der Größe der Eingabeschicht und des diskretisierten Datensatzes lässt sich die Schärfe der Gestendaten bestimmen. Einem diskreten Eingabedatensatz wird ein Rand von 5 Feldern an jeder Seite hinzugefügt, weshalb das zweidimensionale Faltungsnetzwerk mit 1600 (40^2) Einheiten in der Eingabeschicht entworfen wird. Die dreidimensionale Erweiterung des Faltungsnetzwerks wird entsprechend mit 64000 (40^3) Eingabeeinheiten entworfen. Die Größe der Ausgabeschicht ist hingegen fest definiert und richtet sich nach der Größe des Gestenraums $|\Omega|$.

Größe der Filter Die Filter in dem Faltungsnetzwerk werden mit einer Kantenlänge von 5 entworfen. In dem zweidimensionalen Faltungsnetzwerk bestehen die Filter dementsprechend aus 25 (5^2) und in der dreidimensionalen Erweiterung aus 125 (5^3) Einheiten.

Schrittgröße der Filter Die Schrittgröße (engl. stride) der Faltungsfilter definiert den Abstand, in dem sich die Filter über die Eingabe einer Faltungsschicht bewegen. Abbildung 14 veranschaulicht eine Schrittgröße von eins, die auch für das zu implementierende Faltungsnetzwerk gewählt wurde.

Anzahl der Filter Diese beeinflusst wie viele und welche Eigenschaften der Eingabedaten gelernt werden (siehe Abschnitt 2.1.5). Mehr Filter ermöglichen es dem Netzwerk, mehr Eigenschaften zu lernen und zu erkennen. Die Gefahr bei der Verwendung zu vieler Filter besteht darin, dass das Netzwerk zu spezielle Eigenschaften erlernen könnte, was

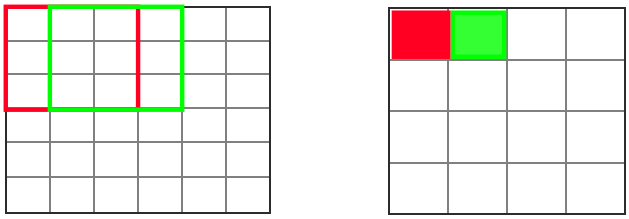

Abbildung 14: Veranschaulichung der Schrittgröße eins. Rot repräsentiert die erste Filterposition, Grün die Zweite.
Quelle: Angelehnt an [Des]

eine Überanpassung zur Folge haben kann. Zu wenige Filter hingegen können eine schlechte Leistung des Netzwerks im Bezug auf die Klassifizierungsgenauigkeit zur Folge haben. Die Anzahl der Filter in der ersten Faltungsschicht wird mit 16 und in der zweiten Faltungsschicht mit 32 definiert.

Größe der vollvernetzten Schichten Wie die Anzahl der Filter hat auch die Anzahl der Einheiten in den vollvernetzten Schichten direkten Einfluss auf die Netzwerkleistung und auf eine potenzielle Überanpassung. Die vollvernetzte Schicht, die auf den zweiten Faltungsstapel folgt, wird mit 32 Einheiten entworfen. Die zweite vollvernetzte Schicht ist die Ausgabeschicht des Netzwerks und enthält, wie bereits oben aufgeführt, vier Einheiten.

Pooling Das zu implementierende Faltungsnetzwerk greift auf das 2×2 Max-Pooling-Verfahren (Abschnitt 2.1.5) mit einer Schrittgröße von zwei zurück. Die Größe der zu verarbeitenden Daten kann dadurch in jeder Pooling-Schicht um 75% vermindert werden. Bei der Erweiterung auf das dreidimensionale Faltungsverfahren wird das $2 \times 2 \times 2$ Max-Pooling-Verfahren angewendet. Die Größe der zu verarbeiteten Daten vermindert sich dadurch um 87,5% mit jedem Pooling-Durchlauf.

Initialisierung der Gewichte Die Gewichte zwischen den Einheiten in neuronalen Netzwerken werden üblicherweise mit kleinen Zufallszahlen initialisiert [LKJ]. Alle Gewichte werden anhand einer Normalverteilung mit einer Standardabweichung von 0.1 initialisiert. Die Biase werden

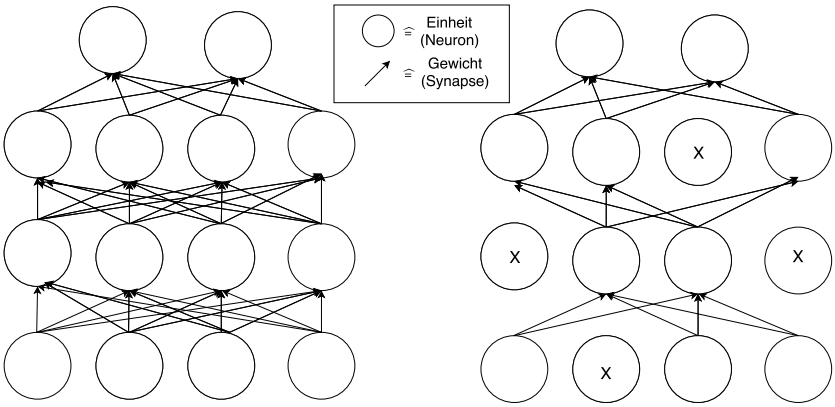

Abbildung 15: Veranschaulichung des Dropout-Verfahrens. Links ist ein vollvernetztes neuronales Netzwerk mit zwei verborgenen Schichten zu sehen. Rechts ist ein Netzwerk nach Anwendung des Dropout-Verfahrens zu sehen. Teile des Netzwerks wurden deaktiviert.
Quelle: Angelehnt an [SHK+14]

mit dem Wert 0.1 initialisiert.

Dropout In [SHK+14] wurde das Dropout-Verfahren beschrieben. Teile des Netzwerks werden mit einer Wahrscheinlichkeit p während einzelner Trainingsphasen deaktiviert. Dadurch wird die Generalisierungsfähigkeit des Netzwerks gefördert und eine Überanpassung gehemmt. Der Hyperparameter p wird auf den Wert 0.5 festgelegt.

Größe der Trainingsstapel Die Größe der Trainingsstapel (engl. batches) gibt die Anzahl an Traingsdatensätzen an, die während einer sogenannten Trainingsiteration an das Netzwerk gegeben werden. Eine Trainingsiteration bezeichnet die Abarbeitung eines Trainingsstapels und die daraus resultierenden Anpassungen der Gewichte im Netzwerk. Ein einzelner Trainingsstapel wird im weiteren Verlauf als B_i bezeichnet. Die Menge an vermehrten Gestendaten C, die für das Trainieren des Netzwerks verwendet werden, lässt sich in n gleich große Trainingsstapel aufteilen.

$$B_i \subset C \text{ und } B_i \cap B_j = \emptyset \text{ für } i \neq j \text{ und } i, j \in \{0, ..., n\}$$

Netzwerke, die während des Trainings auf das Gradientenabstiegsverfahren zurückgreifen, verwenden meist sogenannte Ministapel. Die Größe eines Ministapels $|B_i|$ ist dabei meist sehr viel kleiner als die Anzahl an Ministapeln. Es ist üblich die Stapelgröße als Zweierpotenz zu wählen [NH10], da dies Vorteile in der rechentechnischen Umsetzung und Parallelisierbarkeit des Gradientenabstiegsverfahrens mit sich bringt.

$$|B_i| \in \{32, 64, 128, ..., 512\}$$

Die Anzahl an Trainingsdatensätzen pro Trainingsstapel wird für das zu implementierende Faltungsnetzwerk mit 32 definiert.

Anzahl an Epochen Eine Epoche definiert im Rahmen dieser Arbeit den Zeitraum bis das Netzwerk einmal anhand aller Trainingsdaten trainiert wurde. Die Anzahl der Epochen soll solange erhöht werden, bis sich der Fehlerwert der Evaluierungsdaten des Netzwerks nicht mehr vermindert. Mit der steigenden Anzahl an Epochen steigt auch die Gefahr einer Überanpassung. Es sollen Maßnahmen implementiert werden, die den Trainingsprozess und eine mögliche Überanpassung überwachen.

Lernrate Die Lernrate wird mit dem Wert 0.0001 entworfen.

Fehlerfunktion Das Kreuzentropie-Verfahren wird für die Ermittlung des Fehlerwertes verwendet. Eine Beschreibung von Kreuzentropie im Zusammenhang mit neuronalen Netzwerken kann [LKJ] entnommen werden.

Die Wahl der Hyperparameter und deren Erklärungen sind an [LKJ] und [SCYE17] angelehnt. In komplexeren Netzwerken ist die Suche nach den perfekten Hyperparametern sehr zeitintensiv, weshalb Verfahren wie die Zufalls- und Rastersuche für Hyperparameter in der Praxis Anwendung finden. Diese Verfahren können in [BB12] nachgeschlagen werden.

3.8 Schnittstelle zwischen HoloLens und Faltungsnetzwerk

Um Gesten die auf der Microsoft HoloLens durchgeführt werden, anhand des Faltungsnetzwerks klassifizieren zu können, wird eine Schnittstelle implementiert. Da das Faltungsnetzwerk nicht auf der Hardware der Microsoft HoloLens betrieben wird, wird diese Schnittstelle als Webservice implementiert. Der Datenaustausch soll über das JSON-Format (Abschnitt 3.2.2) erfolgen, in dem die Gestendaten vorliegen. Während der Trainingsphase des Netzwerks wird die Webservice-Schnittstelle zwischen HoloLens und Faltungsnetzwerk nicht verwendet. Die Trainingsdaten sind nach der Erstellung durch die verwendete Augmented-Reality-Brille in JSON-Dateien auf der Festplatte gespeichert und werden in das neuronale Netzwerk importiert.

4 Implementierung

Das nachfolgende Kapitel beschreibt die Implementierung der in Kapitel 3 entworfenen Konzepte. In Abschnitt 4.1 wird der grobe Aufbau und das Zusammenspiel der einzelnen Module des Gesamtsystems zur Gestenklassifizierung veranschaulicht. In den Abschnitten 4.2 bis 4.5 wird auf die Implementierung dieser Module eingegangen. Die einzelnen Module wurden zunächst für die Arbeit mit zweidimensionalen Informationen (x- und y-Werte) implementiert und anschließend für die Arbeit mit dreidimensionalen Informationen (x-, y- und z-Werte) erweitert. Implementierungsaspekte, die speziell für die dreidimensionale Erweiterung gelten, sind in Abschnitt 4.6 aufgeführt.

4.1 Softwarearchitektur

Das System wurde in vier Teilsysteme (Module) aufgeteilt. Die Teilsysteme und deren Zusammenspiel sind in Abbildung 16 skizziert. Die Softwarearchitektur sieht eine Trennung in der Erstellung und Anwendung des Faltungsnetzwerks vor. Die Erstellung implementiert die Topologie des entworfenen Netzes, sowie den Trainings- und Evaluierungsprozess. Die Anwendung umfasst die Ausführung und die Bereitstellung des Faltungsnetzwerks über einen Webservice. In den folgenden Kapiteln wird die Implementierung der vier Module aus Abbildung 16 beschrieben.

4.2 Gesten-Extrahierer

Aufbauend auf den technischen Rahmenbedingungen und dem Entwurf aus Abschnitt 3.2.1 wurde ein Programm entwickelt, um eine Basis an Gestendaten anhand der Microsoft HoloLens zu erstellen. Im Rahmen dieser Arbeit wird dieses Softwaremodul *Gesten-Extrahierer* genannt.

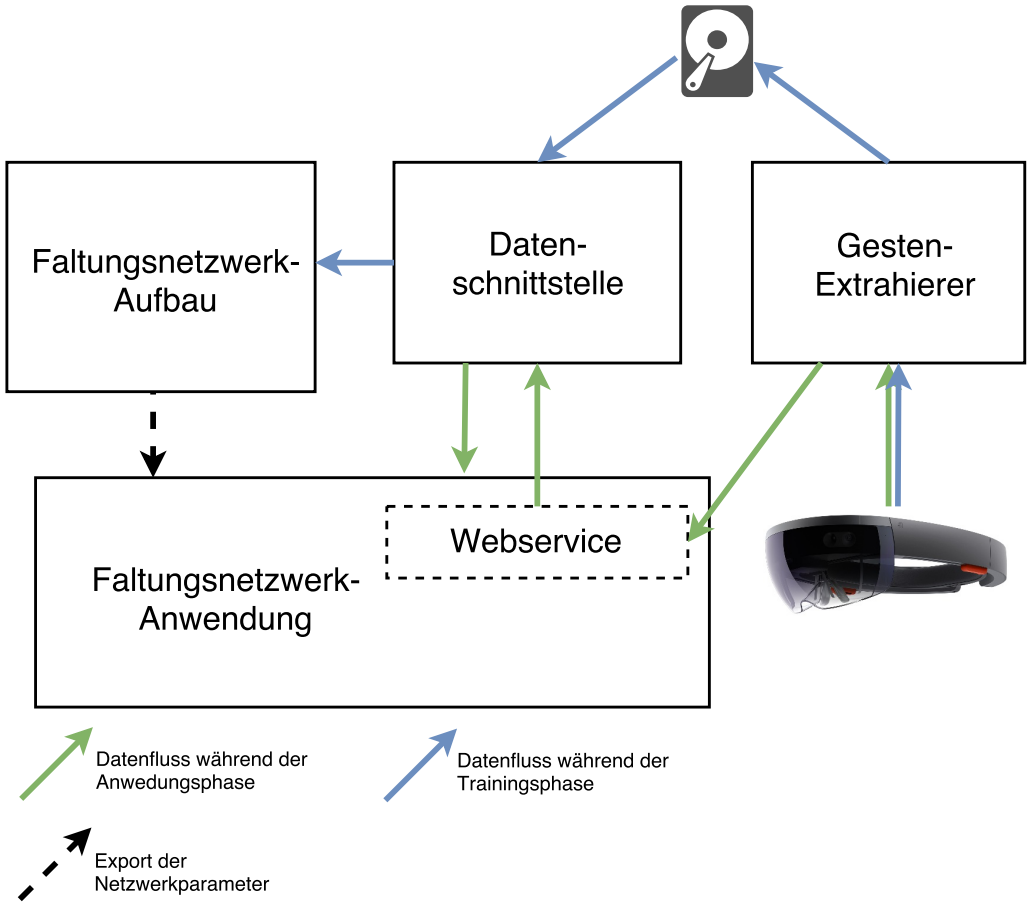

Abbildung 16: Softwarearchitektur (vereinfacht)
Quelle: HoloLens Image: [Micc]

4.2.1 Entwicklungsumgebung

Um die Entwicklung zu beschleunigen und zu vereinfachen, wurde der *Gesten-Extrahierer* mit Hilfe von Unity (siehe Abschnitt 2.2.2) in der Version 2017.1 implementiert. Unity bietet Programmierschnittstellen (engl. application programming interface, API) für den Umgang mit der Microsoft HoloLens an, die mit Hilfe von Skripten angesprochen werden können. Als Entwicklungsumgebung für die Skripte wurde Visual Studio 2017 gewählt. Die Skripte wurden in der Programmiersprache C# erstellt.

4.2.2 Funktionsweise

Der *Gesten-Extrahierer* wurde in einer Unity-Szene implementiert, welche nur ein Spielobjekt in Form einer Sphäre enthält. Diese dient zur Orientierung sowie als Rückmeldung für den Benutzer, dass die Szene gestartet wurde und funktioniert. Für die Kernfunktionalität ist das Spielobjekt nicht notwendig.

Die Logik für die Aufnahme der Gestendaten ist in einem C#-Skript implementiert. Das Skript verwendet die *Unity*-API mit dem Namensraum *UnityEngine.XR.WSA.Input*, um die Koordinaten der aktuellen Handposition zu extrahieren. Befindet sich die Hand in der Gesture-Ready-Position sowie im Gesture Frame (Abschnitt 3.2.1), kann die Aufnahme der Geste gestartet werden. Während der Aufnahme darf die Hand den Gesture-Ready-Status sowie den Gesture Frame nicht verlassen.

Die Aufnahme einer Geste kann anhand der Air-Tap-Geste der HoloLens (Abbildung 17) gestartet und gestoppt werden. Beim Start oder Stopp einer Geste wird dem Benutzer eine Rückmeldung in Form eines Klick-Geräusches gegeben. Mehrere Gesten, die während eines Programmdurchlaufs hintereinander durchgeführt wurden, werden in einer Liste von Gesten zusammengefasst.

Beim Beenden des Programms werden die erfassten Gesten in eine JSON-Datei exportiert. Der Export im JSON-Format wurde durch die *JsonUtility*-Klasse der *Unity*-API im Namensraum *UnityEngine* realisiert. Die Datenstruktur dieses Exports wurde bereits in Abschnitt 3.2.2 beschrieben.

4.3 Datenschnittstelle

Dieses Kapitel beschreibt die Implementierung der Verfahren zur Vorverarbeitung, Diskretisierung und Vermehrung der Gestendaten. Das Softwaremodul *Datenschnittstelle* fasst diese Verfahren zusammen.

Abbildung 17: Die Air-Tap-Geste
Quelle: [Micb]

4.3.1 Entwicklungsumgebung

Die *Datenschnittstelle* wurde in Python 3.6.3, unterstützt durch das *NumPy*-Paket, implementiert. *NumPy* ist ein Erweiterungspaket für Python um wissenschaftliche Berechnungen durchzuführen [dev]. Als Entwicklungsumgebung wurde hauptsächlich Eclipse mit der Pydev-Erweiterung genutzt. Für die Datenvisualisierung, die für die Vorverarbeitung der Daten notwendig ist, wurde ein sogenanntes Jupyter Notebook[6] als Entwicklungsumgebung verwendet. Die Implementierung innerhalb eines Jupyter Notebooks hat vor allem den Vorteil, dass einzelne Programmteile separat voneinander ausgeführt werden können.

4.3.2 Softwarearchitektur

Die einzelnen Softwarekomponenten, aus denen das Modul *Datenschnittstelle* besteht sowie deren Zusammenspiel können der Abbildung 18 entnommen werden. Jede der im Sequenzdiagramm aufgeführten Softwarekomponenten wurde in einem eigenen Python-Skript implementiert. Die Aufgabe der künstlichen Datenvermehrung wurde auf die sogenannte Rohdatenvermehrung und die diskrete Datenvermehrung aufgeteilt. Rohdaten sind Gestendaten vor der Diskretisierung.

[6] http://jupyter.org/

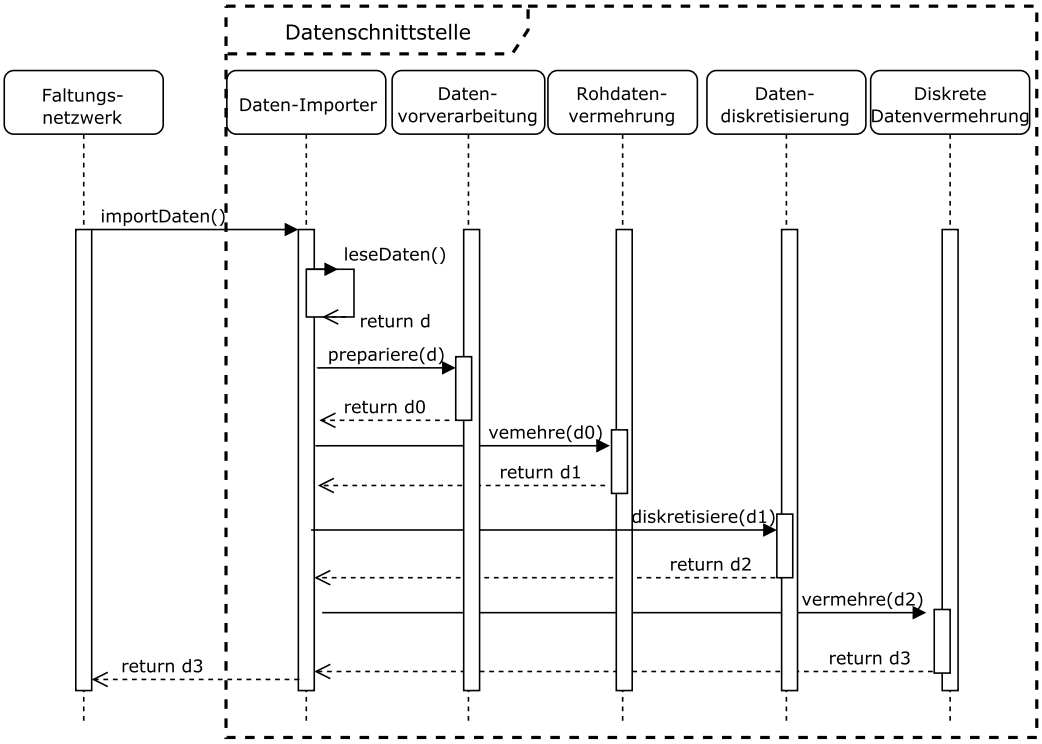

Abbildung 18: Veranschaulichung des Zusammenspiels der Softwarekomponenten der *Datenschnittstelle*. Die Abläufe sind von oben nach unten zeitlich geordnet.

4.3.3 Daten-Importer

Der *Daten-Importer* implementiert die Funktion zum Einlesen der Gestendaten, die zuvor vom *Gesten-Extrahierer* im JSON-Format erstellt wurden. Mit Hilfe der Python *JSON*-API konnten diese JSON-Dateien als Python Dictionary-Objekt eingelesen werden, was weitere Bearbeitungen an den Gestendaten ermöglicht. Die Aufgaben als zentrale Steuerungseinheit und als Schnittstelle zum Faltungsnetzwerk werden im Sequenzdiagramm (Abbildung 18) veranschaulicht.

4.3.4 Datenvorverarbeitung

Bereits in Abschnitt 3.3 wurde auf die Problematik fehlerhafter Gestendaten hingewiesen. Um diesem Problem entgegenzuwirken wurde ein Jupyter Notebook zur Datenvisualisierung entwickelt. Mit Hilfe dieses Notebooks wurden anhand der Python *matplotlib*-Bibliothek zweidimensionale Plots aller Gestendaten erstellt und auf fehlerhafte Datensätze untersucht, um diese manuell aus der Datenbasis zu entfernen. Um das Verhältnis zwischen den verschiedenen Gestenklassen beizubehalten, wurden gelöschte Gestendaten im Anschluss neu erstellt.

Neben dieser manuellen Datensichtung und -bereinigung wurden zwei weitere Maßnahmen zur Datenvorverarbeitung implementiert. Zum Einen wurde eine Funktion entwickelt, die Datensätze mit mehr als 200 oder weniger als 50 Koordinaten löscht. Sind diese Grenzen über- beziehungsweise unterschritten, ist davon auszugehen, dass ein Fehler während der Aufnahme der Geste aufgetreten ist.

Außerdem wurde eine Funktion implementiert, die eine Datenreparatur durchführt. Der Wert γ approximiert die durchschnittliche Länge der Intervalle $[x_{min}, x_{max}]$, $[y_{min}, y_{max}]$ und $[z_{min}, z_{max}]$, wobei die Grenzen dieser Intervalle den Extremwerten aller Koordinaten von allen Gesten aus der Datenbasis entsprechen. γ entspricht in etwa dem Wert 0.2. Ist der Abstand zwischen zwei aufeinanderfolgenden Koordinaten größer als $\frac{\gamma}{20}$, wird eine Koordinate in der Mitte hinzugefügt. Dieses Verfahren wird so lange wiederholt, bis keine Koordinate mehr hinzugefügt werden muss. In Abbildung 19 wird das Ergebnis einer Datenreparatur veranschaulicht.

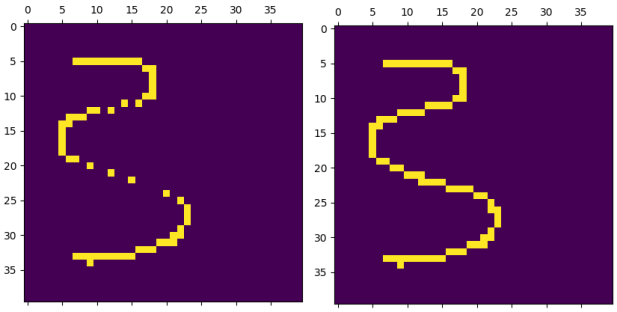

Abbildung 19: Ein diskretisierter Datensatz vor (links) und nach (rechts) der Datenreparatur

4.3.5 Rohdatenvermehrung

Es wurden zwei Verfahren zur Rohdatenvermehrung implementiert. Eines der Verfahren führt eine Verrauschung der Gestendaten durch (Abbildung 20), bei der alle Koordinaten in der *Koordinatenliste* einer Geste um den sogenannten *Verrauschungswert* θ auf der x- und y-Achse verschoben werden. θ wird für alle x- und y-Werte neu gewählt und entstammt aus einer uniformen Wahrscheinlichkeitsverteilung, für die gilt:

$$\{-a \leq \theta \leq a | a \approx \tfrac{\gamma}{80}\}.$$

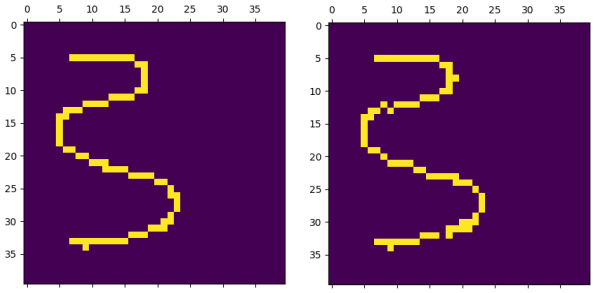

Abbildung 20: Gegenüberstellung eines Datensatzes vor (links) und nach (rechts) der Verrauschung

Des Weiteren wurde ein Verfahren zur Streckung der Geste um den Faktor 0.65 anhand der x- und y-Achse implementiert (Abbildung 21).

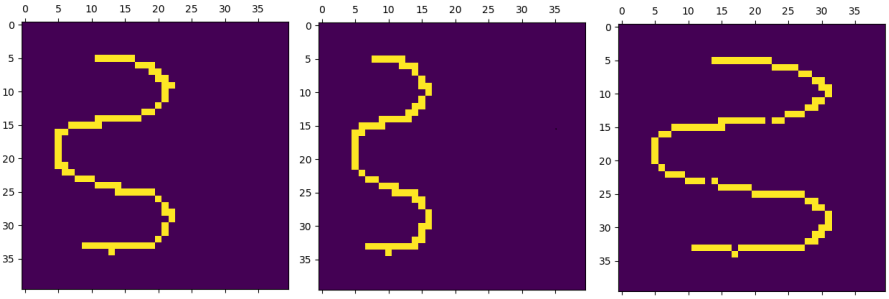

Abbildung 21: Ein diskretisierter Datensatz vor (links) und nach (Mitte und rechts) der Streckung

Durch die Anwendung der Rohdatenvermehrung konnten die Daten in der zweidimensionalen Implementierung um den Faktor 6 vermehrt werden. Implementierungsaspekte der dreidimensionalen Erweiterung der Rohdatenvermehrung werden in Abschnitt 4.6.2 erläutert.

4.3.6 Datendiskretisierung

Pseudocode 1 (Algorithm 1 Datendiskretisierung) veranschaulicht die Implementierung der Datendiskretisierung (Abschnitt 3.4) im zweidimensionalen Bereich.

Durch die Datendiskretisierung wurde jede Koordinate einer Geste auf eine Kachel auf der Ebene abgebildet. Die Ebene wurde als Matrix der Größe 30^2 implementiert, wobei eine Kachel einem der 30^2 Felder in der Matrix entspricht. Jedes dieser Felder wird während der Initialisierung der Matrix auf den Wert 0 gesetzt. Das Abbild einer Koordinate wurde realisiert, indem das Feld in der Matrix, welches die abbildende Kachel repräsentiert, auf den Wert 1 gesetzt wird.

Algorithmus 1 Datendiskretisierung

```
 1: procedure DISKRETISIERE2D(koordinatenliste)
 2:     x_min, y_min, x_max, y_max ← Extremwerte der koordinatenliste
 3:
 4:     intervall_x ← x_max − x_min
 5:     intervall_y ← y_max − y_min
 6:
 7:     teilIntervall ← max(intervall_x, intervall_y)
 8:
 9:     teilIntervall_l ← teilIntervall/30
10:
11:     ebene ← initialisiere Matrix der Größe (30,30) mit 0
12:
13:     for all koord ∈ koordinatenliste do
14:         x_0 ← koord_x − x_min
15:         y_0 ← koord_y − y_min
16:
17:         i_x ← ⌊x_0/teilIntervall_l⌋
18:         i_y ← ⌊y_0/teilIntervall_l⌋
19:
20:         ebene[i_x, i_y] ← 1
21:     return ebene
```

4.3.7 Diskrete Datenvermehrung

Die folgenden Operationen wurden implementiert, um die diskretisierten Gestendaten zu vermehren.

Verschiebung Die Daten in der Gestenmatrix werden um 5 Einheiten nach rechts, unten und rechts-unten verschoben. Bei der Realisierung der Verschiebung wurden Funktionen aus dem *NumPy*-Paket verwendet. Um zu verhindern, dass Gestendaten über die Ränder des Raums oder der Ebene hinweg geschoben werden, wurde der diskretisierten Ebene ein Rand von 5 Einheiten hinzugefügt. Durch diesen Rand vergrößert sich die Matrix, die die Gestendaten enthält, auf 40^2 Felder.

Spiegelung und Transposition *NumPy*-Funktionen für die horizontale und vertikale Spiegelung sowie zur Matrix-Transposition wurden ver-

wendet, um die Daten weiter zu vermehren. Abbildung 22 veranschaulicht die Datenvermehrung anhand einer Transposition und von Spiegelungen eines Gestendatums.

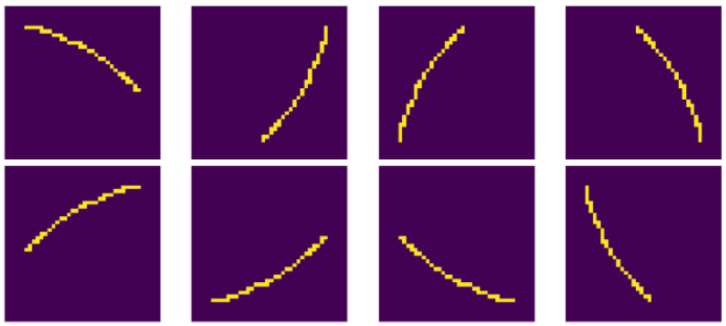

Abbildung 22: Veranschaulichung der Datenvermehrung durch Spiegelungen und Transposition

Durch die Anwendung der diskreten Datenvermehrung im zweidimensionalen Bereich konnten die Daten um den Faktor 32 vermehrt werden. Kombiniert mit der Rohdatenvermehrung ergibt sich im zweidimensionalen Bereich ein Vermehrungsfaktor von 192.

4.4 Aufbau des Faltungsnetzwerks

Der Aufbau und die Anwendung des Faltungsnetzwerks wurden bereits in Abschnitt 4.1 voneinander abgegrenzt. Im Anschluss an den Aufbau werden alle Parameter exportiert, um sie im anwendbaren Faltungsnetzwerk zu importieren. Analog zum Entwurf aus Abschnitt 3.7 wurde zunächst ein Netzwerk mit zweidimensionaler Faltung implementiert und anschließend zu einem Netzwerk mit dreidimensionaler Faltung erweitert. Implementierungsaspekte der dreidimensionalen Erweiterung sind in Abschnitt 4.6.4 beschrieben.

4.4.1 Entwicklungsumgebung

TensorFlow, eine Bibliothek für maschinelles Lernen, wurde für die Implementierung des Faltungsnetzwerks verwendet. Das Netzwerk wurde in einem

Jupyter Notebook implementiert. Um die Leistung während der Entwicklung des Netzwerks zu erhöhen, wurde *TensorFlow* mit GPU-Unterstützung verwendet. In dem Entwicklungs-PC steht eine NVIDIA GeForce GTX 860M Grafikkarte zur Verfügung. Um diese im Zusammenhanag mit *TensorFlow* nutzen zu können, musste zunächst das CUDA-Toolkit, eine Platform für GPU-gestützte parallele Berechnungen [Cor] auf dem Entwicklungs-PC installiert werden. Wie die *Datenschnittstelle* wurde auch das neuronale Netzwerk in Python 3.6.3, unterstützt durch *NumPy* entwickelt. Für das Training der Netzwerke wurde ein Server der Frankfurt University of Applied Sciences mit zwei NVIDIA GeForce GTX 1080 Grafikkarten hinzugezogen. Weitere Systemparameter können der Tabelle 1 entnommen werden.

	Entwicklungs-PC	Server
Betriebssystem	Windows 10 Enterprise	Ubuntu 16.10
Arbeitsspeicher	16 GB	64 GB
Prozessor	i7-4710HQ	i7-6800K
Grafikkarte	GeForce GTX 860M	2 × GeForce GTX 1080

Tabelle 1: Systemparameter

4.4.2 TensorFlow

Neuronale Netzwerke, die anhand der *TensorFlow*-Bibliothek implementiert wurden, lassen sich in die Erstellung und in die Ausführung eines sogenannten Berechnungsgraphen unterteilen. Ein Berechnungsgraph definiert eine Aneinanderreihung von *TensorFlow*-Operationen, die in einer *Session* ausgeführt werden können. Der Tensor ist die zentrale Datenstruktur von *TensorFlow* und gibt der Bibliothek ihren Namen. Ein Tensor ist ein Array mit beliebig vielen Dimensionen [Ten].

Auch der Aufbau dieses Faltungsnetzwerks lässt sich in die Erstellung und die Ausführung des Berechnungsgraphen unterteilen. Die Ausführung des Berechnungsgraphen verweist in diesem Abschnitt auf das Training des Netzwerks und sollte nicht mit der Anwendung des Faltungsnetzwerks (Abschnitt 4.5) verwechselt werden.

4.4.3 Erstellung des Berechnungsgraphen

Durch die Erstellung des Berechnungsgraphen wurde die entworfene Netzwerktopologie (Abschnitt 3.7) implementiert. Die Dropout-, Max Pooling-, ReLU- und Faltungsschichten wurden anhand von Funktionen aus der *TensorFlow*-Bibliothek und die vollvernetzten Schichten durch eine Matrix-Multiplikation implementiert.

Die Gewichte aus den vollvernetzten Schichten, alle Biase sowie die Faltungsfilter wurden als *TensorFlow*-Variablen implementiert, welche während des Trainingsprozesses automatisch im Hinblick auf den Netzwerkfehler optimiert werden. Auch die Initialisierung der Variablen anhand einer Normalverteilung mit der Standardabweichung von 0.1 wurde durch eine *TensorFlow*-Funktion implementiert.

Die Eingabeschicht des Faltungsnetzwerks wurde als *TensorFlow*-Platzhalter definiert. Unter Angabe dieses Platzhalters in Form von einem oder mehreren Gestendaten kann das Netzwerk in einer *Session* ausgeführt werden.

Ferner beeinhaltet der Berechnungsgraph die Definition eines Trainingsschrittes, der während des Trainingsprozesses von einem sogenannten Optimierer ausgeführt wird. Der Optimierer passt, basierend auf der Lernrate und des Netzwerkfehlers, die oben beschriebenen *TensorFlow*-Variablen selbständig an. Für die Ermittlung des Netzwerkfehlers wurde eine *TensorFlow*-Funktion verwendet, die das Kreuzentropieverfahren implementiert.

TensorFlow stellt verschiedene Optimierer bereit. In [LKJ] wird die Verwendung des Adam-Optimierers [KB14] vorgeschlagen, der auch in dieser Implementierung verwendet wurde. Im Gegensatz zu einem Optimierer, der mit dem klassischen Gradientenabstiegsverfahren arbeitet, konnte das Netzwerk anhand des Adam-Optimierers schneller generalisieren. Nach einer Epoche konnte das Netzwerk unter Verwendung des Adam-Optimierers 87,735% der Testdaten richtig klassifizieren. Mit dem Gradientenabstiegs-Optimierer konnten nach einer Epoche unter Verwendung der gleichen Lernrate nur 0,269% der Testdaten richtig klassifiziert werden.

4.4.4 Ausführung des Berechnungsgraphen

Die Ausführung des Berechnungsgraphen bezeichnet die Implementierung des Trainingsprozesses, welcher in Pseudocode 2 (Algorithm 2 Netzwerktraining) veranschaulicht wird.

Algorithmus 2 Netzwerktraining

```
1:  procedure TRAINING
2:      bs ← 32                                  ▷ bs definiert die Trainingsstapelgröße
3:      ec ← 20                                  ▷ ec definiert die Anzahl an Epochen
4:      daten ← importiere Daten von Datenschnittstelle
5:      trainDaten ← alle bis auf 600 Datensätze von daten
6:      evalDaten ← 600 Datensätze von daten
7:      stapel[ ] ← unterteile trainDaten in Stapel der Größe bs
8:      j, i ← 0
9:      starte TensorFlow session
10:     while j < ec do                                              ▷ Epochen
11:         mische(trainDaten)
12:         while i < len(stapel) do                          ▷ Trainingsiterationen
13:             trainingsschritt(stapel[i].gestenDaten, stapel[i].zielwerte)
14:             i ← i + +
15:             if i mod 100 = 0 then
16:                 Reporte Klassifizierungsgenauigkeit anhand von evalDaten
17:         j ← j + +
```

Der Trainingsschritt aus Zeile 13 verweist auf den in Abschnitt 4.4.3 erklärten Trainingsschritt. Unter der Angabe der Zielwerte kann der Optimierer den Fehlerwert des Netzwerks ermitteln.

4.5 Anwendbares Faltungsnetzwerk

Alle Parameter werden nach dem Training des Faltungsnetzwerks exportiert. Diese Parameter werden von dem anwendbaren Faltungsnetzwerk importiert, sodass ein Training des anwendbaren Netzwerks nicht notwendig ist. Das anwendbare Faltungsnetzwerk wurde ebenfalls in Python mit Hilfe der *TensorFlow*-Bibliothek implementiert. Anders als der Faltungsnetzwerk-Aufbau wurde es in Eclipse als ausführbare Python Datei implementiert.

Die Schnittstelle zwischen Microsoft HoloLens und dem anwendbaren Faltungsnetzwerk in Form eines Webservice wurde anhand der Python *Flask*-Bibliothek erstellt. Der Webservice nimmt Anfragen mit Gestendaten im JSON-Format entgegen und führt daraufhin eine Klassifizierung mit Hilfe des ausführbaren Faltungsnetzwerks durch.

4.6 Erweiterung auf 3D

Der folgende Abschnitt beschreibt Implementierungsaspekte der dreidimensionalen Erweiterungen der *Datenschnittstelle* und des Faltungsnetzwerks. Die dritte Dimension beeinhaltet Tiefeninformationen (z-Werte) zu den Koordinaten der Gestendaten.

4.6.1 Datendiskretisierung

Die Koordinaten werden unter Berücksichtigung der Tiefeninformationen auf einen dreidimensionalen Tensor der Form (30,30,30), statt auf eine zweidimensionale Matrix der Form (30,30) abgebildet. Wie der Matrix wird auch dem Tensor ein Rand von 5 Einheiten hinzugefügt, wodurch der dreidimensionale Tensor auf die Größe (40,40,40) erweitert wird.

4.6.2 Datenvermehrung

Durch zusätzliches Strecken der Rohdaten auf der z-Achse konnten die Daten um den Faktor 8 (anstatt dem Faktor 6) vermehrt werden. Die diskretisierten Daten wurden zusätzlich anhand der Tiefeninformationen gespiegelt und konnten dadurch um den Faktor 128 (anstatt 32) vermehrt werden. Insgesamt wurden die Daten durch die dreidimensionale Erweiterung der Datenvermehrung um den Faktor 1024 (anstatt 192) vermehrt.

4.6.3 Daten-Importer

Durch das Hinzufügen der Tiefeninformationen wuchs die Größe eines diskretisierten Datensatzes um den Faktor 40 an. Anstatt $\sim$ 13 Kilobyte nimmt ein Datensatz nun $\sim$ 520 Kilobyte des Arbeitsspeichers in Anspruch. Durch die Vermehrung der 400 Trainings- und Evaluierungsdatensätze um den Faktor 1024 würden $\sim$ 210 Gigabyte Arbeitsspeicher benötigt werden, um die Daten in das Netzwerk zu importieren. Um den Arbeitsspeicherverbrauch zu reduzieren wurde eine Methode zur Komprimierung und Dekomprimierung der diskretisierten Gestendaten implementiert. Der *Daten-Importer* liefert die Daten in einem komprimierten Format an das Faltungsnetzwerk aus. Wenn das Netzwerk einen Teil dieser Daten zum Trainieren oder Evaluieren verwendet, werden diese vorher dekomprimiert. Die implementierte Komprimierungsmethode konnte den Bedarf an Arbeitsspeicher im dreidimensionalen Bereich von $\sim$ 210 Gigabyte auf $\sim 0,4$ Gigabyte reduzieren.

Der Komprimierungsalgorithmus nutzt die Struktur der diskreten Gestendatensätze aus. Die meisten Felder in dem Tensor, der die diskreten dreidimensionalen Daten enthält, besitzen den Wert 0 und nur etwa jedes 500. der insgesamt 64000 Felder besitzt den Wert 1. Die komprimierte Datenstruktur speichert die Indizes der Felder, in denen sich der Wert 1 in dem dreidimensionalen Tensor befindet. Durch dieses Verfahren lassen sich die dreidimensionalen diskreten Daten um den Faktor $\sim$ 500 komprimieren.

4.6.4 Faltungsnetzwerk

Um die dreidimensionalen diskreten Daten anhand des Faltungsnetzwerks auswerten zu können, musste eine dreidimensionale Erweiterung des Faltungsverfahrens verwendet werden. Auch die dreidimensionale Faltung wurde mit Hilfe der *TensorFlow*-Bibliothek implementiert. Die Ausprägungen und Dimensionierungen der Tensoren, welche die Gestendaten und Gewichte enthalten, wurden entsprechend erweitert.

5 Evaluation

Dieses Kapitel befasst sich mit der Auswertung und Untersuchung des implementierten Faltungsnetzwerks. Ziel der Evaluation ist die Gewinnung weiterer Erkenntnisse über diese Netzwerke.

Zunächst wird in Abschnitt 5.1 die Leistung der implementierten Faltungsnetzwerke bewertet. Um die Effektivität des Lernprozesses bewerten zu können, wurde in Abschnitt 5.2 die Trainingsphase des zwei- und dreidimensionalen Faltungsnetzwerks im Hinblick auf die Klassifizierungsgenauigkeit und den Netzwerkfehler untersucht. Weiterhin wird in Abschnitt 5.3 eine Visualisierung der erlernten Parameter der ersten Faltungsschicht vorgenommen, um zu zeigen welche Eigenschaften der Gestendaten von dem Netzwerk gelernt wurden. Das *Verrauschungsexperiment* in Abschnitt 5.4 soll zeigen, wie sich das Netzwerk gegenüber unkenntlicher Gestendaten verhält. Ferner wird im *Datenexperiment* aus Abschnitt 5.5 das Netzwerk anhand von nur 10 Datensätzen pro Gestenklasse trainiert, um anschließend die Klassifizierungsgenauigkeit auf die Test- und *Fremdgestendaten* auszuwerten. Abschließend werden in Abschnitt 5.6 die Gestendaten um zeitliche Informationen ergänzt, um das Netzwerk mit dreidimensionaler Faltung anhand dieser Daten zu untersuchen.

Die soeben beschriebenen Untersuchungen werden anhand der in Abschnitt 3.7 entworfenen Netzwerktopologie, der in Abschnitt 4.4 beschriebenen Implementierung, sowie anhand der in Abschnitt 3.5 beschriebenen Datenaufteilung durchgeführt. Auf Hyperparameter oder Implementierungsaspekte, die davon abweichen, wird in den einzelnen Abschnitten hingewiesen. Die implementierten Faltungsnetzwerke wurden auf dem in Abschnitt 4.4.1 beschriebenen Server trainiert und anschließend auf dem Entwicklungs-PC (Abschnitt 4.4.1) ausgewertet.

5.1 Bewertung der Netzwerkleistung

Wie bereits in Abschnitt 2.3 erläutert, kann die Leistung eines Gestenerkennungssystems unter anderem anhand der Klassifizierungsgenauigkeit gemessen werden. Diese wird jeweils für die 200 Testdaten und 185 *Fremdgestendaten* (Abschnitt 3.5) ermittelt. Die *Fremdgestendaten* wurden durch Anleitung des Autors von fünf Testpersonen durch reale Interaktion mit der Microsoft HoloLens erstellt. Den Testpersonen wurde die Ausrichtung und Geschwindigkeit, in der die Gesten durchgeführt wurden, selbst überlassen. Sie wurden einzig über die Funktionsweise des *Gesten-Extrahierers* sowie über die Form der durchzuführenden Gesten unterrichtet. Die Gruppe der Testpersonen beinhaltete sowohl Links- als auch Rechtshänder/-innen.

Tabelle 2 listet die Klassifizierungsgenauigkeit des zwei- und dreidimensionalen Faltungsnetzwerks nach Abschluss der Trainingsphase auf. Das zweidimensionale Faltungsnetzwerk wurde während der Trainingsphase mit insgesamt 20 Epochen und das dreidimensionale mit 10 Epochen trainiert. Die unterschiedliche Epochenanzahl ergibt sich aus der unterschiedlichen Menge an vermehrten Trainingsdaten (Abschnitt 4.6.2). Eine Auswertung der Leistung eines Netzwerks, dass nur mit 10% der Testdaten trainiert wurde, wird durch das *Datenexperiment* (Abschnitt 5.5) durchgeführt.

	Testdaten	*Fremdgestendaten*
2D (20 Epochen)	100%	95,68%
3D (10 Epochen)	99%	91,35%

Tabelle 2: Klassifizierungsgenauigkeit der trainierten Faltungsnetzwerke

5.2 Untersuchung der Trainingsphase

Im Folgenden wird der Verlauf des Trainingsprozesses des zwei- und dreidimensionalen Faltungsnetzwerks veranschaulicht. Das dreidimensionale Faltungsnetzwerk konnte anhand von 404.904 Gestendatensätzen[7], das zweidimensionale hingegen mit nur 75.432 pro Epoche trainiert werden (Abschnitt

[7]Die meisten Gestendatensätze wurden durch die künstliche Datenvermehrung erzeugt.

4.6.2). Für eine Epoche entspricht das 2.357 Iterationen mit je 32 Trainingsdatensätzen in der zweidimensionalen Implementierung und 12.653 Iterationen mit je 32 Trainingsdatensätzen im dreidimensionalen Bereich. Die Definitionen für die Begriffe Epoche und Iterationen können in Abschnitt 3.7.3 nachgeschlagen werden. Das dreidimensionale Netzwerk konnte, unter Verwendung der in Abschnitt 4.4.1 beschriebenen Hardware, innerhalb von dreieinhalb Stunden und das zweidimensionale Netzwerk innerhalb von drei Minuten trainiert werden.

Die Graphen in den Abbildungen 23 bis 26 zeigen die Klassifizierungsgenauigkeit und den durchschnittlichen Netzwerkfehler (Abschnitt 3.7.3) anhand der Evaluierungsdaten (Abschnitt 3.5) in Abhängigkeit mit der Anzahl an durchgeführten Trainingsiterationen und Epochen.

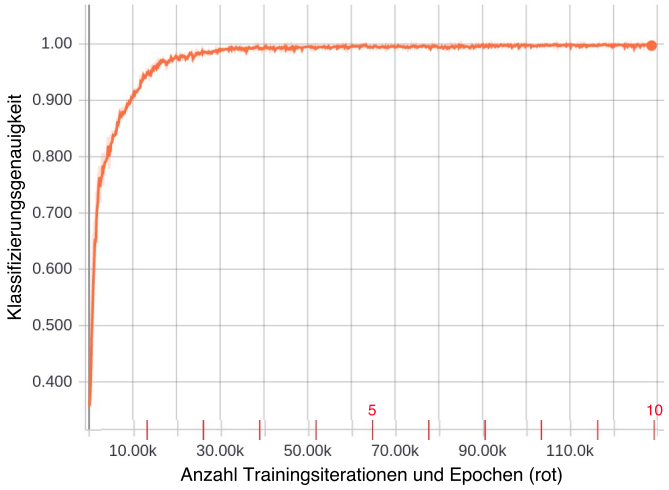

Abbildung 23: Klassifizierungsgenauigkeit der dreidimensionalen Implementierung während der 10 Trainingsepochen. Der Wert auf der y-Achse zeigt die Klassifizierungsgenauigkeit auf die Evaluierungsdaten in Abhängigkeit mit der Anzahl an durchgeführten Trainingsiterationen (x-Achse) und Epochen (x-Achse in rot).

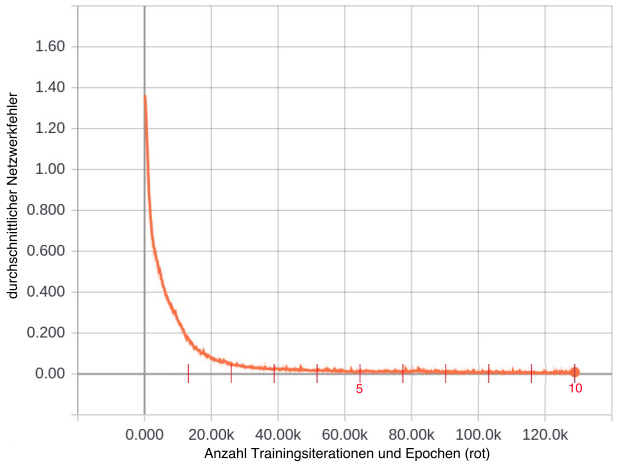

Abbildung 24: Netzwerkfehler der dreidimensionalen Implementierung während der 10 Trainingsepochen. Der Wert auf der y-Achse zeigt den durchschnittlichen Netzwerkfehler der Evaluierungsdaten in Abhängigkeit mit der Anzahl an durchgeführten Trainingsiterationen (x-Achse) und Epochen (x-Achse in rot).

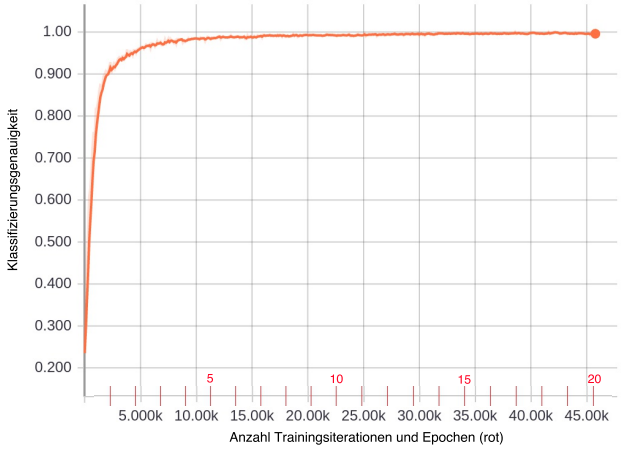

Abbildung 25: Klassifizierungsgenauigkeit der zweidimensionalen Implementierung während der 20 Trainingsepochen. Der Wert auf der y-Achse zeigt die Klassifizierungsgenauigkeit auf die Evaluierungsdaten in Abhängigkeit mit der Anzahl an durchgeführten Trainingsiterationen (x-Achse) und Epochen (x-Achse in rot).

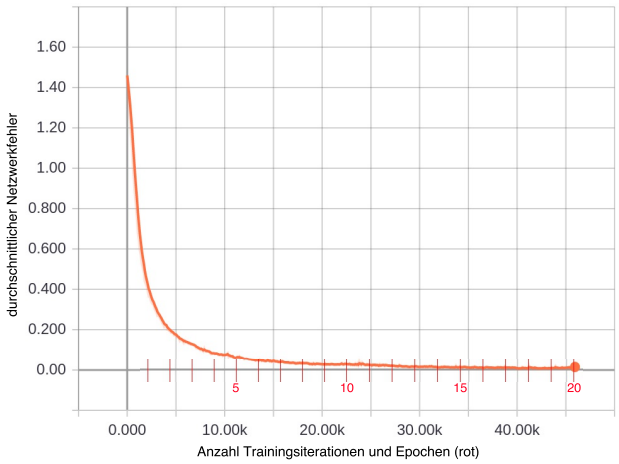

Abbildung 26: Netzwerkfehler der zweidimensionalen Implementierung während der 20 Trainingsepochen. Der Wert auf der y-Achse zeigt den durchschnittlichen Netzwerkfehler der Evaluierungsdaten in Abhängigkeit mit der Anzahl an durchgeführten Trainingsiterationen (x-Achse) und Epochen (x-Achse in rot).

Wie in den Graphen zu sehen ist, generalisieren die zwei- und dreidimensionalen Netzwerke unterschiedlich schnell auf die Evaluierungsdaten. Abbildung 23 kann entnommen werden, dass das dreidimensionale Netzwerk in etwa nach drei Epochen beziehungsweise 40.000 Trainingsiterationen eine Klassifizierungsgenauigkeit von fast 100% erreicht. Abbildung 25 zeigt, dass das zweidimensionale Netzwerk nach etwa sieben Epochen beziehungsweise 17.000 Trainingsiterationen eine Klassifizierungsgenauigkeit von fast 100% erreicht. Im Hinblick auf die Anzahl an Trainingsdaten generalisiert das zweidimensionale Faltungsnetzwerk also schneller als das Dreidimensionale, jedoch langsamer betreffend der Anzahl an Epochen.

5.3 Visualisierung des Gelernten

Wie in Abschnitt 2.1.5 beschrieben extrahieren die Faltungsfilter Eigenschaften aus den Gestendaten. Während des Trainingsprozesses lernen diese Faltungsfilter, welche Eigenschaften aus den Gestendaten zu extrahieren sind. Abbildung 27 stellt die 16 Filter der ersten Faltungsschicht der zweidimen-

sionalen Implementierung nach Abschluss der Trainingsphase (20 Epochen mit allen Trainingsdaten) dar.

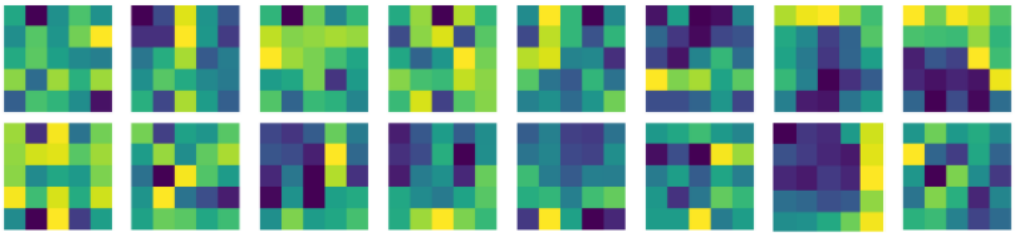

Abbildung 27: Filter der ersten Faltungsschicht nach Abschluss von 20 Trainingsepochen. Positive Gewichtungen sind gelb, negative blau und neutrale grün dargestellt.

Die Ergebnisse der Faltung anhand der erlernten Filter werden in Merkmalskarten festgehalten (Abschnitt 2.1.5). Abbildung 28 veranschaulicht das Faltungsverfahren anhand eines Filters aus der ersten Faltungsschicht und der daraus resultierenden Merkmalskarte. Diese speichert Informationen über Eigenschaften der Eingabedaten, die durch den entsprechenden Filter extrahiert wurden. Gelbe Felder in der Merkmalskarte zeigen, wo der Filter die entsprechende Eigenschaft in den Eingabedaten finden konnte.

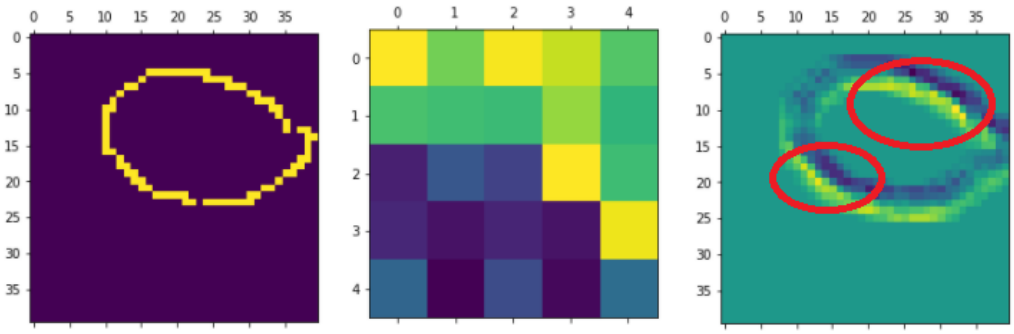

Abbildung 28: Exemplarische Darstellung des Faltungsverfahrens. Die Merkmalskarte (rechts) speichert Informationen über Eigenschaften eines Eingabedatensatzes (links), die durch den Filter (Mitte) extrahiert wurden.

Durch die Visualisierung des Gelernten konnte gezeigt werden, dass das Netzwerk Eigenschaften der Gestendaten erlernt. Des Weiteren konnte das Auftre-

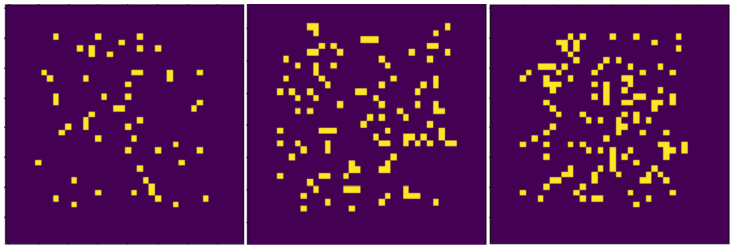

Abbildung 29: Beispiele für maximal verrauschte Gestendaten

ten sogenannter toter Filter durch die Visualisierung ausgeschlossen werden. Tote Filter verursachen Merkmalskarten, die unabhängig von der Eingabe immer Null sind. Sie können bei der Verwendung von ReLU-Aktivierungsfunktionen in Kombination mit einer zu hoch definierten Lernrate auftreten [LKJ].

5.4 Verrauschungsexperiment

In einem weiteren Experiment wurde die Netzwerkausgabe des zweidimensionalen Faltungsnetzwerks nach Abschluss der Trainingsphase (20 Epochen mit allen Trainingsdaten) anhand von sogenannten *maximal verrauschten* Eingabedaten ausgewertet. Durch die *maximale Verrauschung* wird jede Koordinate eines Gestendatensatzes zufällig auf eine beliebige Position im Koordinatenbereich jenes Gestendatensatzes abgebildet. Beispiele von *maximal verrauschten* Gestendaten können Abbildung 29 entnommen werden.

Ziel des Experiments ist es zu zeigen, wie sich das Netzwerk gegenüber Daten verhält, die keiner der vier Gestenklassen (Abschnitt 3.1) zuzuordnen sind. Diese Daten werden durch die *maximale Verrauschung* der 200 Testdaten simuliert. Die einzige Eigenschaft, welche die Daten auch nach einer *maximalen Verrauschung* beibehalten, ist die Anzahl an Koordinaten.

Auf den Graphen in Abbildung 30 ist zu sehen, wie die 200 *maximal verrauschten* Testdatensätze mit zunehmenden Trainingsiterationen von dem zweidimensionalen Faltungsnetzwerk klassifiziert wurden.

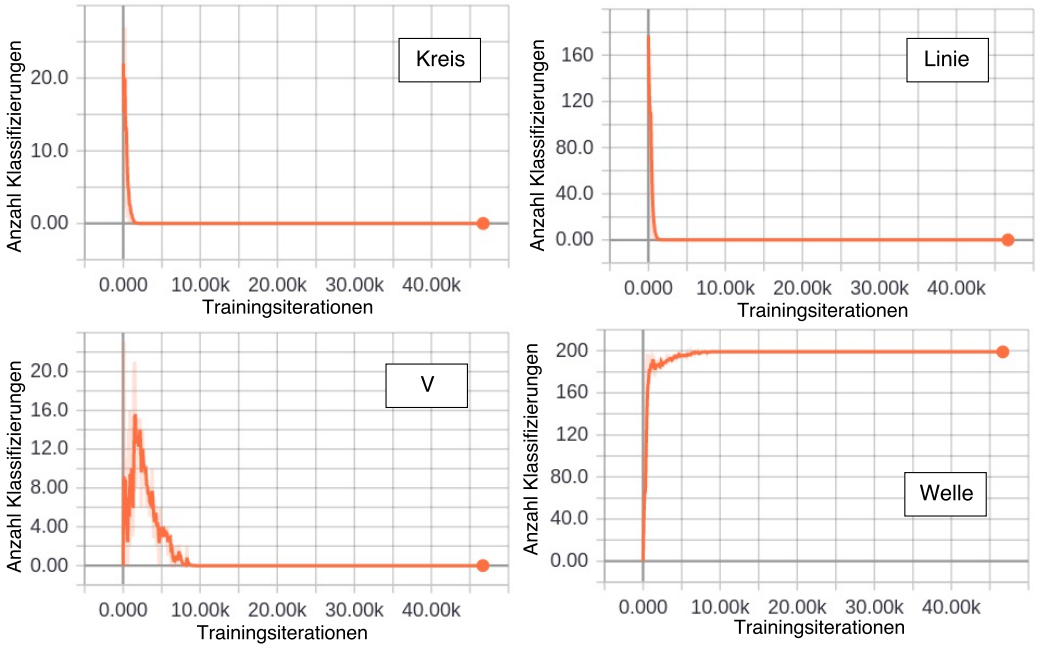

Abbildung 30: Verlauf der Klassifizierungsergebnisse während der 20 Trainingsepochen. Der Wert auf der y-Achse zeigt die Anzahl an Klassifizierungen pro Gestenklasse in Abhängigkeit mit der Anzahl an durchgeführten Trainingsiterationen (x-Achse).

In den Graphen ist zu sehen, dass das fertig trainierte zweidimensionale Faltungsnetzwerk Datensätze, die nicht einer Gestenklasse zuzuordnen sind, als Wellengeste klassifiziert. Durch das *Verrauschungsexperiment* konnte also ausgeschlossen werden, dass das Netzwerk eine Klassifizierung der Gesten anhand der Anzahl an Koordinaten einer Geste vornimmt.

5.5 Datenexperiment

In einem weiteren Experiment wurde das dreidimensionale Faltungsnetzwerk mit nur 10 Datensätzen pro Gestenklasse trainiert. Um das Netzwerk trotz der geringen Datenmenge trainieren und evaluieren zu können, wurde der Datenvermehrungsfaktor durch zusätzliche Transpositionen der Gestendaten von 1024 auf 3072 sowie die Anzahl der Trainingsepochen auf 40 erhöht. Das Faltungsnetzwerk konnte demnach anhand von 122.280 Gestendatensätzen

trainiert und evaluiert werden. Für die Evaluierung, also die Überwachung des Trainingsprozesses, wurden 600 der vermehrten Daten verwendet (150 aus jeder Gestenklasse). Für die Bewertung der Leistung des experimentellen Faltungsnetzwerks wurde auf die 200 Test- und 185 *Fremdgestendaten* zurückgegriffen. Auf die Testdaten konnte eine Klassifizierungsgenauigkeit von 92% und auf die 185 *Fremdgestendaten* eine Genauigkeit von 86,49% erreicht werden. Abbildung 31 veranschaulicht die 40 Gestendatensätze, die für das Training des Faltungsnetzwerks verwendet wurden.

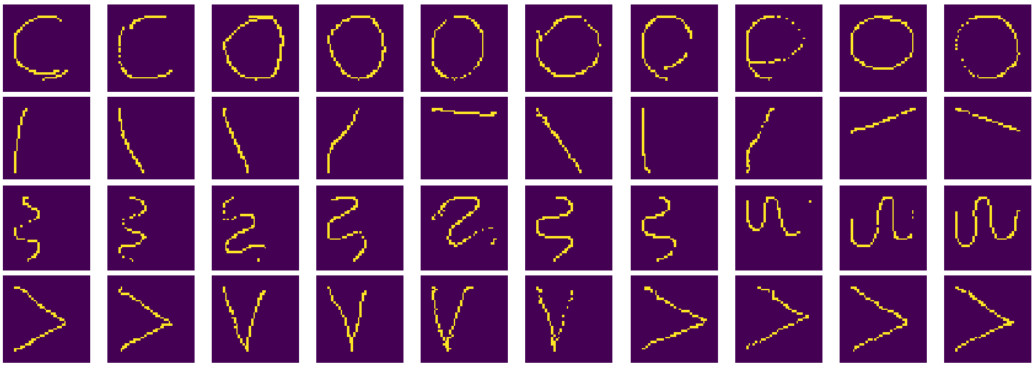

Abbildung 31: Veranschaulichung der Trainingsdaten für das *Datenexperiment*

5.6 Hinzufügen von zeitlichen Informationen

Abschließend wurde das implementierte Netzwerk mit dreidimensionaler Faltung anhand von Gestendaten untersucht, die um zeitliche Informationen ergänzt wurden. Den Gestendaten wurden Informationen über die zeitliche Abfolge der durchgeführten Handgesten hinzugefügt, was durch eine Anpassung der Diskretisierungsmethode (Abschnitt 3.4 und 4.3.6) realisiert wurde. Die Felder beziehungsweise Würfel eines diskretisierten Datensatzes enthalten durch diese Erweiterung die Information, zu welchem *Zeitpunkt* sich die Hand während der Gestenaufnahme an dem abgebildeten Punkt befand. Der *Zeitpunkt* entspricht dabei der Positionierung einer Koordinate in der *Koordinatenliste* (Abschnitt 3.2.2) einer Geste.

Um verifizieren zu können, dass das Netzwerk die zeitlichen Informationen bei der Klassifizierung in Betracht zieht, wurden künstlich sogenannte *langsame Gesten* erstellt. Diese wurden aus den Gesten der Datenbasis (Abschnitt 3.2) erzeugt und haben die spezielle Eigenschaft, dass die Gestenausführung durch Hinzufügen von Koordinaten in der Mitte des Datensatzes künstlich pausiert wurde. Jeder *langsamen Geste* wurden 300 Koordinaten hinzugefügt, was einer Pausierung der Ausführung von fünf Sekunden entspricht. Aus jedem einzelnen Datensatz aus der Datenbasis wurde eine *langsame Geste* erzeugt. Diese wurden dabei einer eigenen Klasse (Abschnitt 3.2.2) zugeordnet, wodurch der Gestenraum Ω (Abschnitt 3.1) von vier auf acht erweitert wurde.

Die Topologie des Netzwerks mit dreidimensionaler Faltung wurde für die Arbeit mit dem neuen Datenformat optimiert indem die Anzahl der Einheiten in den vollvernetzten Schichten von 32 auf 256 erhöht wurde (Abschnitt 3.7). Die Erweiterung des dreidimensionalen Faltungsnetzwerks wurde anhand von 810.608 Trainingsdatensätzen aus acht Gestenklassen in 10 Epochen trainiert. Nach Abschluss des Trainings konnte eine Klassifizierungsgenauigkeit von 99,25% auf die Testdaten und 89,19% auf die Fremdgestendaten erreicht werden. Zu beachten ist, dass sich die Anzahl an Fremd- und Testdaten durch die Erstellung der *langsamen Gestendaten* jeweils verdoppelt hat.

6 Fazit

Momentan wird die Definition und Erkennung von benutzerdefinierten Handgesten auf der Microsoft HoloLens durch den restriktiven Zugriff auf die Sensordaten eingeschränkt. Dies hat zur Folge, dass wichtige Informationen, wie die Positionierung der einzelnen Finger bei der Erkennung einer Handgeste nicht in Betracht gezogen werden können. Arbeiten wie [GBD+16] und [FKM17] verwenden sogar Zusatzhardware, um bei der Verwendung der HoloLens auf mehr Umgebungsinformationen zugreifen zu können. In dieser Arbeit konnte der Entwurf und die Implementierung eines zwei- und dreidimensionalen Faltungsnetzwerks zur Erkennung von Handgesten vorgestellt werden, wobei die Daten über offizielle Schnittstellen der HoloLens bezogen wurden (Abschnitt 3.2.1). Alle Daten, die für das Training, die Evaluierung und die Bewertung des neuronalen Netzwerks verwendet wurden, wurden in realer Interaktion mit der Microsoft HoloLens erstellt und im Anschluss künstlich vermehrt.

Zu Beginn der Arbeit wurden die theoretischen Grundlagen über neuronale Netzwerke, Gestenerkennung und Augmented Reality vermittelt. In Abschnitt 2.4 wurden alternative Lösungsansätze sowie Vorarbeiten durch verwandte Arbeiten erläutert.

Im Zuge des Entwurfs wurde zunächst ein Datenmodell zur Darstellung der auf der Microsoft HoloLens durchgeführten Gesten definiert. Im Anschluss wurden Konzepte zur Erstellung und künstlichen Vermehrung von Gestendatensätzen nach diesem Modell erstellt. Ferner wurde die Topologie des zweidimensionalen Faltungsnetzwerk und der dreidimensionale Erweiterung erarbeitet.

Nach der Implementierung der erarbeiteten Konzepte wurde eine abschließende Evaluierung durchgeführt. Durch das *Datenexperiment* konnte auch mit einer geringen Anzahl an Daten (10 pro Klasse) eine Klassifizierungsgenauigkeit von 91,1% auf die Testdaten erreicht werden. Durch die Visualisie-

rung der Filter der ersten Faltungsschicht konnten interessante Informationen über das Erlernte sowie über die Funktionsweise von Faltungsnetzwerken gewonnen werden. Die Ergebnisse aus der Evaluierung zeigen, dass ein zweidimensionales Faltungsnetzwerk und eine dreidimensionale Erweiterung zur Erkennung von auf der Microsoft HoloLens durchgeführten Handgesten erfolgreich entworfen und implementiert wurde.

Folgende Aspekte konnten aufgrund der begrenzten Bearbeitungszeit nicht im Rahmen dieser Arbeit behandelt werden und bleiben für zukünftige Weiterentwicklungen offen.

Erweiterung des Gestenraums Die Faltungsnetzwerke in dieser Arbeit wurden anhand von vier verschiedenen Gestenklassen trainiert und bewertet. Für einen produktiven Einsatz bietet sich die Erweiterung des Gestenraums an.

Erweiterung des Datenexperiments Eine Untersuchung der Netzwerkleistung bei Verwendung von weniger als 10% der Trainingsdaten kann weitere Erkenntnisse liefern. Die Netzwerktopologie sollte dabei entsprechend angepasst werden, um einer Überanpassung auf die geringe Datenmenge vorzubeugen.

Visualisierung der dreidimensionalen Erweiterung Im Rahmen dieser Arbeit wurden Visualisierungen ausschließlich im zweidimensionalen Bereich betrachtet. Eine Erweiterung der Visualisierungen auf den dreidimensionalen Bereich kann weitere Klarheit über die erlernten Parameter liefern.

Literatur

[AAB+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *CoRR*, abs/1603.04467, 2016.

[Ada09] Jürgen Adamy. Grundlagen nichtlinearer Systeme. In *Nichtlineare Regelungen*. Springer, Januar 2009.

[AW12] Barinder Pal Singh Ahluwalia and Ritika Wason. Gestural Interface Interaction; A Methodical Review. *International Journal of Computer Applications*, 60(1), 2012.

[Azu97] Ronald T. Azuma. A Survey of Augmented Reality. *Presence: Teleoper. Virtual Environ.*, 6(4):355–385, August 1997.

[BB12] James Bergstra and Yoshua Bengio. Random Search for Hyper-parameter Optimization. *J. Mach. Learn. Res.*, 13:281–305, Februar 2012.

[Boe07] Ingo Boersch. *Eine Einfuehrung in die Kuenstliche Intelligenz fuer Informatiker und Ingenieure*. Elsevier, 2007.

[Bra95] Rüdiger Brause. Neuronale Netze: eine Einführung in die Neuroinformatik-2. überarb. und erw. Aufl. *Stuttgart: Teubner*, 1995.

[Bry96] Steve Bryson. Virtual Reality in Scientific Visualization. *Commun. ACM*, 39(5):62–71, Mai 1996.

[CMM+11] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, High Performance Convolutional Neural Networks for Image Classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1237–1242. AAAI Press, 2011.

[Cor] NVIDIA Corporation. CUDA Toolkit Documentation. `http://docs.nvidia.com/cuda/` (aufgerufen am 10.12.2017).

[Des] Adit Deshpande. A Beginner's Guide To Understanding Convolutional Neural Networks . `https://adeshpande3.github.io/` (aufgerufen am 29.11.2017).

[dev] NumPy developers. NumPy. `http://www.numpy.org/` (aufgerufen am 06.12.2017).

[DS94] J. Davis and M. Shah. Visual gesture recognition. *IEE Proceedings - Vision, Image and Signal Processing*, 141(2):101–106, April 1994.

[Ert16] Wolfgang Ertel. Neuronale Netze. *Grundkurs Künstliche Intelligenz*, Januar 2016.

[FKM17] Markus Funk, Mareike Kritzler, and Florian Michahelles. HoloLens is more than Air Tap: Natural and Intuitive Interaction with Holograms. 2017.

[Fur11] Borko Furht. *Handbook of Augmented Reality*. Springer Publishing Company, Incorporated, 2011.

[GBD+16] Mathieu Garon, Pierre-Olivier Boulet, Jean-Philippe Doironz, Luc Beaulieu, and Jean-François Lalonde. Real-Time High Resolution 3D Data on the HoloLens. In *2016 IEEE International*

Symposium on Mixed and Augmented Reality, ISMAR 2016 Adjunct, Merida, Yucatan, Mexico, September 19-23, 2016, pages 189–191, 2016.

[GDR17] Fabian Beck Günter Daniel Rey. Neuronale Netze eine Einführung. *Fabian Beck*, 2017. `http://www.neuronalesnetz.de/aktivitaet.html` (abgerufen am 11.12.2017).

[GS03] Günther Görz and Josef Schneeberger. *Handbuch der künstlichen Intelligenz*. Walter de Gruyter, 2003.

[HTH00] Pengyu Hong, M. Turk, and T. S. Huang. Gesture modeling and recognition using finite state machines. In *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, pages 410–415, 2000.

[IG16] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org` (abgerufen am 12.12.2017).

[JEW+97] Tony Jebara, Cyrus Eyster, Joshua Weaver, Thad Starner, and Alex Pentland. Stochasticks: Augmenting the Billiards Experience with Probabilistic Vision and Wearable Computers. In *First International Symposium on Wearable Computers (ISWC 1997), Cambridge, Massachusetts, USA, 13-14 October 1997, Proceedings.*, pages 138–145, 1997.

[JW17] Luiz Perez Jason Wang. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. 2017.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[KBLH03] Oliver Kreylos, E. Wes Bethel, Terry J. Ligocki, and Bernd Hamann. *Virtual-Reality Based Interactive Exploration of Multiresolution Data*, pages 205–224. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Kra06] Karl-Friedrich Kraiss. *Advanced man-machine interaction*. Springer, 2006.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[KTS+14] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale Video Classification with Convolutional Neural Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Juni 2014.

[Lit07] Krystof Litomisky. Consumer RGB-D Cameras and their Applications. *University of California, Riverside*, 2007.

[LKJ] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition 2016.

[MA07] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS - PART C*, 37(3):311–324, 2007.

[Mai97] Klaus Mainzer. Evolution des Gehirns. In *Gehirn, Computer, Komplexität*, pages 8–14. Springer, 1997.

[MGKK15] Pavlo Molchanov, Shalini Gupta, Kihwan Kim, and Jan Kautz. Hand Gesture Recognition With 3D Convolutional Neural Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Juni 2015.

[Mica] Microsoft. Discretization Methods (Data Mining). `https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/discretization-methods-data-mining` (aufgerufen am 11.01.2018).

[Micb] Microsoft. Gestures and motion controllers in Unity. `https://developer.microsoft.com/en-us/windows/mixed-reality/gestures_and_motion_controllers_in_unity` (aufgerufen am 06.12.2017).

[Micc] Microsoft. Microsoft HoloLens. `https://www.microsoft.com/de-de/hololens` (aufgerufen am 26.11.2017).

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115, Dezember 1943.

[MRH15] Abhijit Nale Ankush Barkade Milind R. Hegade, Neha Jamdar. Hand Gesture Recognition: A Survey. *International Journal of Advanced Research in Computer Science Technology*, 3, 2015.

[NH10] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010.

[NWTN15] Natalia Neverova, Christian Wolf, Graham W. Taylor, and Florian Nebout. *Multi-scale Deep Learning for Gesture Detection and Localization*, pages 474–490. Springer International Publishing, Cham, 2015.

[PM95] Akira Utsumi Fumio Kishino Paul Milgram, Haruo Takemura. Augmented reality: a class of displays on the reality-virtuality continuum. *Proc.SPIE*, 2351:2351 – 2351 – 11, 1995.

[PTJ17] Tran Pham, Anthony Tang, and Christian Jacob. User-Defined Gestures for Holographic Medical Analytics. *Proceedings of the Graphics Interface, Edmonton, Alberta*, pages 16–19, 2017.

[Rab89] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Februar 1989.

[RBR15] Philipp Rauschnabel, Alexander Brem, and Young Ro. Augmented Reality Smart Glasses: Definition, Conceptual Insights, and Managerial Importance. Juli 2015.

[Rei] Zuzana Reitermanová. Data Splitting. *WDS*, 10:31 – 36.

[RW11] Gunter Daniel Rey and Karl F Wender. Neuronale Netze: Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung. *Auflage, Huber, Bern*, 2011.

[SCYE17] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *CoRR*, abs/1703.09039, 2017.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[SP95] Thad E. Starner and Alex Pentland. Visual Recognition of American Sign Language Using Hidden Markov Models, 1995.

[SWP98] T. Starner, J. Weaver, and A. Pentland. Real-time American sign language recognition using desk and wearable computer based video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1371–1375, Dezember 1998.

[SZ14] Karen Simonyan and Andrew Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. *CoRR*, abs/1406.2199, 2014.

[Tec] Unity Technologies. Unity Documentation. `https://docs.unity3d.com/Manual/UnityManual.html` (aufgerufen am 06.12.2017).

[Ten] Tensorflow. Getting Started. `https://www.tensorflow.org/get_started/` (aufgerufen am 10.12.2017).

[vKP10] D. W. F. (Rick) van Krevelen and Ronald Poelman. A Survey of Augmented Reality Technologies, Applications and Limitations. *International Journal of Virtual Reality*, 9(2):1–20, Juni 2010.

[YOI92] J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden Markov model. In *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 379–385, Juni 1992.