Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Tobias Hassenklöver

Klassifikation hochvarianter Muster mit Faltungsnetzwerken

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Tobias Hassenklöver

Klassifikation hochvarianter Muster mit Faltungsnetzwerken

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Abgegeben am 16. Januar 2012

Tobias Hassenklöver

Thema der Bachelorarbeit

Klassifikation hochvarianter Muster mit Faltungsnetzwerken

Stichworte

Faltungsnetzwerke, Gesichtserkennung, Objekterkennung, Faltung, Filter, Neuronale Netze

Kurzzusammenfassung

Die Klassifizierung von Objekten und besonders des Menschen durch Software stellt sich schon seit Jahren als schwer lösbares Problem heraus. Diese hochvarianten Formen werden optisch erfasst und bisher häufig mit Neuronalen Netzwerken in Kombination mit statistischen Verfahren erfasst. Eine neue Methode zur Erkennung von hochvarianten Mustern, wie Schriftzeichen, Gegenständen oder Menschen sind Faltungsnetzwerke. Faltungsnetzwerke sind eine Abart von Neuronalen Netzwerken, die ihre Entscheidungen auf Basis von Algorithmen aus der Bildverarbeitung treffen. In dieser Arbeit werden die Grenzen der Erkennung der Faltungsnetzwerke getestet. In Tests mit unterschiedlich modifizierten Eingabedaten wird die Auswirkung auf die Klassifizierungsgenauigkeit geprüft.

Tobias Hassenklöver

Title of the paper

High-variant pattern classification with Convolutional Neural Networks

Keywords

Convolutional Neural Networks, Facedetection, Objectdetection, Convolution, Filtering, Neural Networks

Abstract

The classification of objects and especially of humans by software is for years a great challenge. These highly variant forms are optically captured and often detected with the use of neural networks in combination with statistical methods. A new method for the detection of these highly variant patterns, such as characters, objects or people are Convolutional neural networks. Convolutional neural networks are a variety of Neural networks, which make their decisions based on algorithms used in image processing. In this thesis the limits of the detection of Convolutional neural networks are tested. In the tests the impact on the classification accuracy is checked with various modified input data.

Inhaltsverzeichnis

Abbildungsverzeichnis **6**

1. Einführung **8**
1.1. Motivation und Zielsetzung 8
1.2. Gliederung 11

2. Stand der Technik **12**
2.1. Faltungsnetzwerke 12
2.2. Ziel der Arbeit 12
2.3. Erweiterungsmöglichkeiten 14
2.3.1. Stereo-Eingabedaten 15
2.3.2. Erkennung von dreidimensionalen Eingabevektoren 15

3. Klassifikation von Mustern **17**
3.1. Neuronale Netze 17
3.1.1. Biologische Neuronale Netze 17
3.1.2. Künstliche Neuronale Netze 19
3.2. Hauptkomponentenanalyse 27
3.2.1. Analyse 27
3.2.2. Reduzierung der Dimensionen 29

4. Faltungsnetzwerke **31**
4.1. Funktionsweise und Aufbau 31
4.1.1. Faltung 32
4.1.2. Schichten 35
4.2. Lernvorgang 36

5. Realisierung und Test **37**
5.1. Testdaten 38
5.1.1. Erstellung von Trainings- und Testdaten 39
5.1.2. Verrauschung von Bilddaten 41
5.2. Testaufbau 43
5.2.1. Matlab LeNet-5 43

5.2.2. EBlearn++ Convolutional Neural Network 44
5.3. Ergebnisse 46
5.3.1. Allgemeine Gesichtserkennung 46
5.3.2. Erkennung von bestimmten Gesichtern 46

6. Diskussion **48**

Literaturverzeichnis **50**

A. Trainieren von Convolutional Neural Networks **53**
A.1. Training für *persnet* 53
A.2. Training für Matlab CNN 54

B. Starten der Persnet Anwendung **55**

C. Anwendung der Toolchain für Bild- und Trainingsdaten **57**
C.1. Generierung von Bildern aus Videostreams 57
C.2. Konvertierung von Bildern in das *mnist* Format 58
C.3. Hintergrundverrauschung von Gesichtsbildern 58

Abbildungsverzeichnis

1.1. Eine Zeichenkette handgeschrieben und als digitales Muster (links) und zwei hochvariante Formen von Stühlen [8] (rechts) 9
1.2. Stark variante Buchstaben, die sich in Rotation, Verzerrung, Größe, Verrauschung und Schärfe unterscheiden 9
1.3. Drei Autokennzeichen [9] [1] [2] in verschiedenen Perspektiven und Erkennbarkeiten 10

2.1. Skizzierter Aufbau eines Faltungsnetzwerkes mit Stereo-Eingabedaten . . . 14
2.2. Dreidimensionale Punktewolke einer Person [10] 15

3.1. Pyramidenzellen (grün hervorgehoben) in der menschlichen Großhirnrinde [31]. 18
3.2. Neuronales Netzwerk mit zwei Schichten von Neuronen und drei zu erkennenden Mustern 19
3.3. Innerer Aufbau eines Perzeptron-Neurons 20
3.4. Beispiele möglicher Aktivierungsfunktionen θ 21
3.5. Aufbau eines kleinen künstlichen Neuronalen Netzes. Eingabedaten $x_1 \ldots x_n$, verarbeitet durch Neuronen $e_1 \ldots e_n$ und $v_1 \ldots v_n$ und ausgegeben von Neuron a zu Ergebnis y 22
3.6. Ein Neuronales Netz mit mehreren Schichten. (a) Eingabeschicht, (b) versteckte Schicht, (c) Ausgabeschicht 23
3.7. Aufbau des Trainings eines Perzeptron-Neurons 24
3.8. Fehlergebirge eines Neurons mit zwei Eingabedaten und Gewichten 25
3.9. Der Gradientenabstieg zum Finden des kleinsten Fehlers mit dem *Backpropagation* Algorithmus. Jeder rote Pfeil stellt eine Iteration des Gradientenabstiegs dar 26
3.10. Anwenden des *Backpropagation* Algorithmus auf ein gesamtes Netzwerk . . 27
3.11. Punktewolke [21] mit ihrer ersten Hauptkomponente - einer Geraden, die alle Punkte approximiert 28
3.12. Punktewolke mit den ersten zwei Hauptkomponenten r und h berechnet (links). Die Hauptkomponenten reichen, um die Daten in einem neuen Koordinatensystem (rechts) anzuzeigen 29

4.1. Faltung eines zweidimensionen Vektors mit einer 3x3 Faltungsmaske) 32

4.2. Ungefiltertes Bild (links), mit 3x3 Laplace-Faltungsmaske gefiltertem Bild (mitte) und einem 3x3 zufällig initialisierter Faltungsmaske gefiltertem Bild (rechts) 33
4.3. Beispiel zur Lokalisierung ähnlicher Muster einer 1x3 Korrelationsmaske in einer Pixelreihe 34
4.4. Aufbau der Schichten eines Faltungsnetzwerkes 36

5.1. Aufbau einer Aufnahme der Bilderdatenbank *muct*[18]. Hier wird eine Bilderserie erstellt, in der fünf Bilder parallel aus unterschiedlichen Winkeln aufgenommen werden 38
5.2. Vier Beispiele aus der Bilderdatenbank *Faces in the wild* [30] 39
5.3. Einige Beispiele der Hintergrund-Klassifizierungsgruppe. Als Trainingsbilder wurden Objektausschnitte, verschieden farbige Hintergründe oder Kanten benutzt 39
5.4. Die einzelnen Verarbeitungsschritte der Werkzeugkette. Eine zufällige Verrauschung (links), die Verrauschung tiefengefiltert (mitte) und das eingefügte Gesicht in die Verrauschung (rechts) 42
5.5. Aufbau der Schichten des LeNet-5 Faltungsnetzwerkes 43

1. Einführung

1.1. Motivation und Zielsetzung

Die Erkennung von Formen, Objekten und des Menschen durch Software ist nach wie vor ein schwer zu lösendes Problem. Diese varianten Muster werden optisch erfasst und digital weiter verarbeitet. Über die letzten Jahre hat die Forschung viele Methoden entwickelt, diese Muster zu erkennen und zu klassifizieren. Die Muster können grob in zwei Kategorien unterteilt werden:

- zweidimensionale Muster, wie Schriftzeichen, Symbole oder Stanzteile. Ein Anwendungsbeispiel wäre die Erkennung von Handschriften.
- dreidimensionale Objekte, die in zweidimensionalen Bildern abgebildet sind. Eine praktische Anwendung hierfür wäre die Erkennung von Gesichtern oder Autokennzeichen in einem Bild oder Video. Wobei man bei den dreidimensionalen Objekten dynamische, lebende und starre unterscheidet.

Die Klassifizierung der Muster in den beiden Kategorien unterstehen bestimmten Varianzen. Diese erschweren die Erkennung. Bei zweidimensionalen Mustern, wie Schriftzeichen, können diese Varianzen Rotation, Skalierung, unterschiedliche Schreibweisen, Strichstärken oder Verzerrungen sein. Bei dreidimensionalen Mustern kommen zusätzliche Effekte wie Perspektive, Lichteinfluss, Schatten, Tiefe und Größe des Objektes hinzu. Ein Beispiel für die unterschiedliche Komplexität von zweidimensionalen und dreidimensionalen Objekten ist in Abbildung 1.1 gezeigt. Die Zeichenketten links im Bild haben eine vergleichsweise niedrige Varianz im Gegensatz zu den rechts im Bild zu sehenden Stühlen. Die linke Zeichenkette besteht aus vier Symbolen. Wenn die Zeichenkette klassifiziert wird, werden diese einzeln ausgewertet. Jedes Symbol birgt vergleichsweise wenig Rotation, farbliche Abweichung oder Verzerrung im Vergleich zu dem digitalen Schriftmuster daneben. Das Muster birgt also wenig Varianz. Sollen die Stühle optisch als diese klassifiziert werden, stellen sich mehrere Probleme. Es handelt sich bei einem Stuhl um ein dreidimensionales Objekt, dargestellt in einem zweidimensionalen Bild. Dieses dreidimensionale Objekt kann, wie beim rechten Stuhl zu sehen, rotationsbehaftet sein und von der Grundform stark abweichen. Trotz der

Unterschiede in Form und Rotation müssen beide Abbildungen als Stuhl klassifiziert werden. Auch Buchstaben und Ziffern können Rotation und Verzerrung ausgesetzt sein - was sie auch hochvariant macht.

Abbildung 1.1.: Eine Zeichenkette handgeschrieben und als digitales Muster (links) und zwei hochvariante Formen von Stühlen [8] (rechts)

Mit Hilfe der Kombination verschiedener Mustererkennungsverfahren können diese varianten Muster erkannt werden - jedoch versagen viele Verfahren zur Klassifikation bei starken Variationen dieser Muster. Die Klassifikation eines Musters basiert auf einer Beschreibung dessen, was erkannt werden soll. Je näher das zu erkennende Muster der Beschreibung entspricht, um so höher ist die Chance, dass das Verfahren das Muster richtig klassifiziert. Am Beispiel der Schrifterkennung soll dieses Problem in Abbildung 1.2 verdeutlicht werden. Hier sind die Buchstaben *i* und *a* in verschiedenen Varianzen abgebildet. In der Abbildung sind

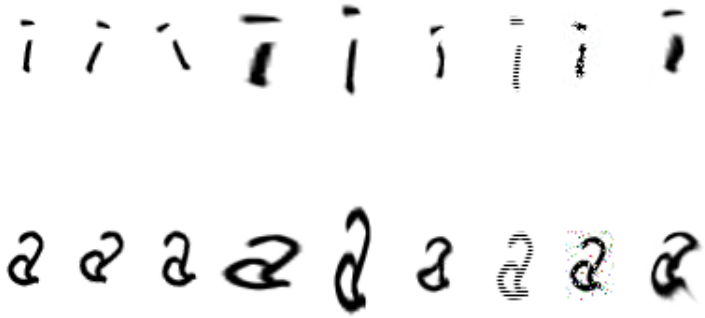


Abbildung 1.2.: Stark variante Buchstaben, die sich in Rotation, Verzerrung, Größe, Verrauschung und Schärfe unterscheiden

links die Formen der Buchstaben mit wenig Varianz und weiter rechts stärkere Formen der

Varianz abgebildet: Verzerrungen, leichte und starke Rotationen, Rauschen und Verschmierungen erschweren die Erkennung massiv. Wobei sich bei der unteren Zeile auch noch die Schwierigkeit bei der Erkennung gibt, ob der Buchstabe ein *a* oder *d* ist.
Viele Verfahren verbessern ihre Erkennung, indem sie bestimmte Formen der Varianz korrigieren. So kann Rauschen erkannt und unterdrückt werden, Verzerrung kann durch affine Transformation entgegen gewirkt werden oder schwache Strichstärke mit lokaler Kontrastanhebung prägnanter gemacht werden. Um dies zu tun, muss man sich dieser Varianzen auf dem jeweiligen Anwendungsgebiet aber gewahr sein. Ein Beispiel hierfür wäre die Erkennung von Autokennzeichen. In Abbildung 1.3 sind drei Kennzeichen zu sehen.

Abbildung 1.3.: Drei Autokennzeichen [9] [1] [2] in verschiedenen Perspektiven und Erkennbarkeiten

Bei dem linken Kennzeichen sind wenig Varianzen der Buchstaben und Ziffern zu erkennen. Nach der Bestimmung der Lage des Kennzeichens im Bild, können die Zeichen nacheinander ausgewertet werden.
Bei dem mittleren Kennzeichen gestaltet sich die Erkennung schwieriger. Hier sind Teile des Kennzeichens verschmutzt und die Auswertung könnte Ziffern falsch klassifizieren oder ganz überlesen. Wenn man bei der Konzeptionierung des Erkennungsverfahrens sich dieses Problems gewahr ist, könnte man mit der Kombination von Korrekturmaßnahmen diesem entgegenwirken.
Beim rechten Bild ist der Blickwinkel auf das Kennzeichen variant. Hier müsste man das Verfahren abermals anpassen, um den Winkel der Aufnahme und dessen Effekt auf die Sicht des Nummernschilds mit einer affinen Transformation zu korrigieren.
Einzelne Variationen sind bei der Konzeptionierung des Klassifizierungsverfahrens bekannt oder können zur Laufzeit erkannt und korrigiert werden. Da in vielen Anwendungsgebieten Muster nicht vorhersagbar variieren, müssen neue Verfahren entwickelt und bestehende kombiniert werden, um generell Muster trotz Variationen besser erkennen zu können.
In dieser Arbeit soll ein neuartiger Ansatz zur Erkennung von hochvarianten Mustern untersucht werden. Grundlage hierfür ist der Einsatz von Faltungsnetzwerken. Sie sind eine spezielle Form von Neuronalen Netzwerken, deren Verarbeitung auf Basis von grafischen Filtern basiert.

1.2. Gliederung

In Kapitel 2 wird zunächst ein Überblick über den Stand der Technik der Faltungsnetzwerke und die Ziele dieser Arbeit gegeben. In einer kurzen Auflistung soll klargestellt werden, welche Implementierungen nötig waren, um die Ziele dieser Arbeit zu realisieren. Weiter soll ein Ausblick über zukünftige Ansätze und Möglichkeiten mit der Technologie der Faltungsnetzwerke gegeben werden.

In Kapitel 3 werden zwei Verfahren zum Klassifizieren von Mustern vorgestellt. Eines dieser Verfahren ist die Verwendung von Neuronalen Netzen. Neuronale Netze stellen ein wichtiges Werkzeug für die digitale Klassifikation dar. Künstliche Neuronale Netzwerke sind von den biologischen Neuronalen Netzwerken inspiriert. Dies wird zunächst anhand des biologischen Vorbilds gezeigt. Als nächstes wird auf die Funktionsweise und das Training künstlicher Neuronaler Netzwerke eingegangen. Nach der Vorstellung der Neuronalen Netzwerke folgt die Erläuterung des zweiten Verfahrens, der Hauptkomponentenanalyse. Diese wird benötigt, um große Datenmengen auf ihre Erkennungsmerkmale zu untersuchen und unnötige Daten zu minimieren, wobei die massiv minimierten Datenmengen trotzdem die selben markanten Erkennungsmerkmale vorweisen. Diese minimierten Daten können dann mit anderen verglichen und wiedererkannt werden.

Beide Technologien finden in Klassifizierungswerkzeugen großen Einsatz und bilden eine wichtige Grundlage für die Funktionen von Faltungsnetzwerken. Kapitel 4 widmet sich ganz der Funktionsweise von Faltungsnetzwerken. Die Unterschiede zu herkömmlichen Neuronalen Netzwerken werden hier verdeutlicht und es wird auf die spezifischen Grundlagen der Faltungstechniken eingegangen.

In Kapitel 5 werden die unterschiedlichen Szenarien erläutert, in denen die Faltungsnetzwerke auf Ihre Erkennung von hochvarianten Mustern getestet werden. Zunächst wird in Sektion 5.1 genauer auf die Testdaten, also die hochvarianten Muster, eingegangen. Für manche Tests wurden Teile der Testdaten verrauscht, in der Hoffnung, Merkmale in den Mustern zu verdeutlichen und die Erkennungsrate der Netzwerke zu verändern. Die dafür benötigten Methoden werden an dieser Stelle genauer erklärt. In Sektion 5.2 werden im Detail der Aufbau der in den Tests benutzen Faltungsnetzwerke beleuchtet. Am Ende des Kapitels werden die Ergebnisse der Tests mit den verschiedenen Konfigurationen aufgelistet.

Kapitel 6 beinhaltet eine Diskussion über die Tests, über die in Kapitel 5 dargestellten Ergebnisse sowie über das Training der Faltungsnetzwerke.

2. Stand der Technik

In diesem Kapitel soll ein Überlick über den Stand der Technik der Faltungsnetzwerke, die Ziele dieser Arbeit und die in dieser Arbeit verwendeten Technologien und Werkzeuge gegeben werden. Außerdem sollen Möglichkeiten vorgestellt werden, die Ergebnisse und Werkzeuge, die zu dieser Arbeit erstellt wurden, als Basis für weitere Tests und Erweiterungen zu nutzen.

2.1. Faltungsnetzwerke

Faltungsnetzwerke sind eine immer mehr in Erscheinung tretende Form der Neuronalen Netze. Trotz positiver Ergebnisse in der Ziffern- [13] und Objektklassifizierung [16] befindet sich diese Technologie erst am Anfang Ihrer breiten Verwendung in der Wirtschaft.
Insbesondere in der Erkennung von Gesichtern wurde die Technologie der Faltungsnetzwerke schon in kommerziellen Produkten verwendet. Als Beispiel wäre die Erkennung von Gesichtern in dem Bilderdatenbestand von *Google Street View* zu nennen. Hier mussten die Gesichter in den Bildern unkenntlich gemacht werden. Dazu wurde eine Programm-Pipeline entwickelt, die Gesichter im Bild erkennt und diese schwärzt. Ein Teil der Pipeline, basiert auf der Technologie der Faltungsnetzwerke [26]. Die Technologie der Faltungsnetzwerke befindet sich noch im Entwicklungszustand. Aus diesem Grund sind Werkzeuge, Dokumentationen und Open Source Initiativen in diesem Bereich, im Gegensatz zu anderen, sehr verbreiteten Technologien, so gut wie nicht vorhanden. Obwohl sich innerhalb der letzten Jahre viel in diesem Bereich getan hat, ist die Anzahl der Werkzeuge, auf die in dieser Arbeit aufgebaut werden kann, sehr gering.

2.2. Ziel der Arbeit

Das Ziel dieser Arbeit ist es, die Klassifizierungsergebnisse von *Osadchy, LeCun und Miller* beschrieben in [20] mit eigenen Testreihen nachzuvollziehen. In den Ergebnissen on *Osadchy, LeCun und Miller* wurden in Bildern die Gesichter mit Hilfe von Faltungsnetzwerken erkannt. Hierfür wurde ein Programm geschrieben, das jedes Bild in viele Teilbilder zerlegt.

Jedes Teilbild wurde an das trainierte Faltungsnetzwerk weitergegeben. Das Faltungsnetzwerk hat dann die Entscheidung getroffen, ob in dem Bildausschnitt ein Gesicht vorhanden ist oder nicht.
Diese Art der Klassifikation von Bildern mit Gesicht oder ohne Gesicht soll auch in dieser Arbeit mit Hilfe von zwei unterschiedlichen Implementierungen von Faltungsnetzwerken untersucht werden. Da die Technik der Faltungsnetzwerke noch jung ist, gibt es nur wenig unterstützende Werkzeuge. Aus diesem Grund mussten zusätzlich zu den schon implementierten zwei Faltungsnetzwerken noch einige Werkzeuge erstellt werden.
Alle in dieser Arbeit realisierten Teilaspekte sind im Folgenden aufgelistet:

- Es wurde ein Matlabprogramm, das ein Faltungsnetzwerk implementiert, analysiert und umgeschrieben. In Sektion 5.2.1 wird das Programm und die Änderungen genauer erläutert.
- Eine nicht dokumentierte Programmierschnittstelle der Arbeitsgruppe von LeCun, das *Computational and Biological Learning Lab* der Universität von New York, wurde analysiert und benutzt, um damit eigene Programme mit Faltungsnetzwerken und deren Training zu realisieren. Eine genauere Erläuterung der realisierten Programme und der Faltungsnetzwerke ist in Sektion 5.2.2 gegeben.
- Für die Konvertierung der Trainings- und Testdaten mussten folgende Programme geschrieben werden:
  - Programm für die Konvertierung von Bilderdaten in das MNIST-Format. Das MNIST-Format wird von dem Matlabprogramm als Eingabeformat vorgegeben. Ein genauer Aufbau des Dateiformates ist in Sektion 5.1.1 beschrieben.
  - Programm für die Verrauschung der Umgebung von Gesichtern in Bilderdaten. Für eine Reihe von Tests mit Faltungsnetzwerken sollte herausgefunden werden, ob durch die Verrauschung der Umgebung der Gesichter, die Erkennungsrate verbessert werden kann. Eine genaue Beschreibung der Bearbeitungsschritte des Programms wird in Sektion 5.1.2 gezeigt.
- Implementierung eines Skriptes, das mit Hilfe der Programme *Gimp* und *ffmpeg* Videostreams in Einzelbilder zerlegt. Die Einzelbilder wurden als 32x32 Graustufenbilder gespeichert. So konnten Trainings- und Testdaten aus erstellten Videos generiert werden.
- Training eines Faltungsnetzwerkes für die Generierung von Bilderdaten für Gesichtsumgebungen. Diese Bilderdaten wurden für das spätere Training von anderen Faltungsnetzwerken verwendet und stellten die Trainingsdaten dar für die Klassifizierungsgruppe "Bild enthält kein Gesicht". Eine genauere Beschreibung der Trainings- und Testdaten ist in Sektion 5.1 gezeigt.

- Durchführung von Trainings und Tests mit den zwei verschiedenen Faltungsnetzwerken mit dem Ziel, ähnliche Klassifizierungsergebnisse wie *Osadchy, LeCun und Miller* zu erzielen. Hierfür wurden für das Training der Faltungsnetzwerke verschiedene Mischungen von Klassifizierungsgruppen und Trainingsdaten verwendet. Eine genaue Auflistung der erzielten Ergebnisse ist in Sektion 5.3 gezeigt.

2.3. Erweiterungsmöglichkeiten

Im Folgenden soll ein kurzer Überblick gegeben werden, welche Möglichkeiten bestehen, auf den in dieser Arbeit erbrachten Stand weiter aufzubauen. In der Arbeitsgruppe *Computational and Biological Learning Lab* der Universität von New York [4] findet rege Entwicklung im Bereich der Faltungsnetzwerke statt. Die durch das Projekt EBlearn++ [23] zur Verfügung gestellten Ergebnisse und Werkzeuge können, zusätzlich zu den Erkenntnissen dieser Arbeit dazu beitragen, Faltungsnetzwerke weiter auf ihre Stärken und Schwächen zu testen.

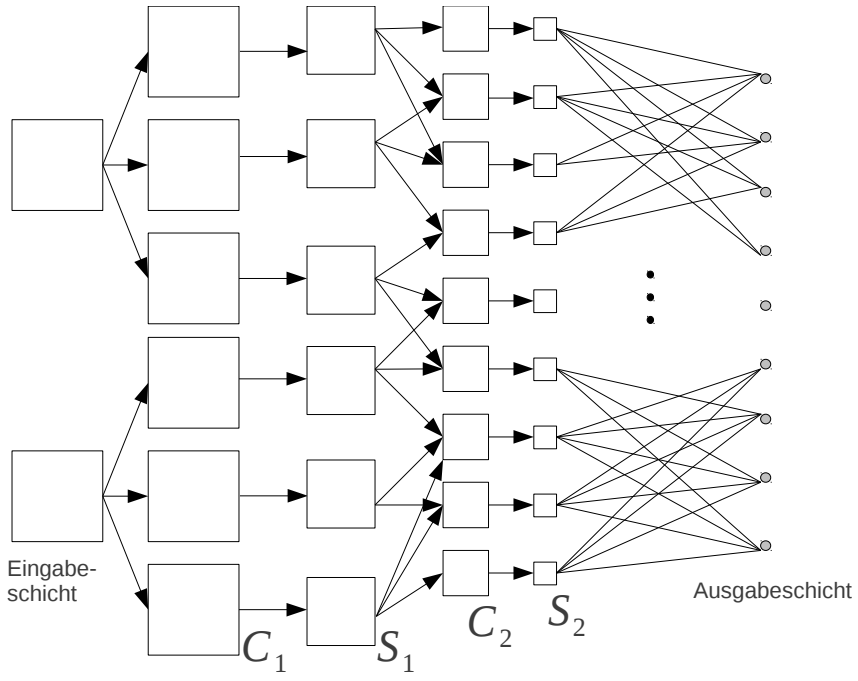


Abbildung 2.1.: Skizzierter Aufbau eines Faltungsnetzwerkes mit Stereo-Eingabedaten

2.3.1. Stereo-Eingabedaten

Wie Scherer, Müller und Behnke gezeigt haben [25], kann die Erkennung von varianten Mustern durch Stereoeingabe verbessert werden. Bei einem Faltungsnetzwerk mit Stereoeingabe hat die Eingabeschicht, anstatt wie in Sektion 4.1.2 beschrieben, zwei Eingabefelder. In Abbildung 2.1 ist ein Faltungsnetzwerk mit einem solchen Aufbau skizziert.
Die Eingabeschicht verteilt zwei Bilder in das Faltungsnetzwerk. Je nach Implementierung des Netzwerkes dauert es mehrere Convolution-Schichten, bis sich die Faltungen der Eingabebilder treffen.

2.3.2. Erkennung von dreidimensionalen Eingabevektoren

Die in dieser Arbeit vorgestellten Faltungsnetzwerke arbeiten auf zweidimensionalen Eingabevektoren. Da die in Sektion 4.1.1 beschriebene Faltung auch auf mehrdimensionalen Ebenen funktioniert, wäre ein Faltungsnetzwerk mit Eingabedaten von drei oder mehr Dimensionen möglich. Punktewolken von Objekten wären eine mögliche Art der Eingabedatenform. In

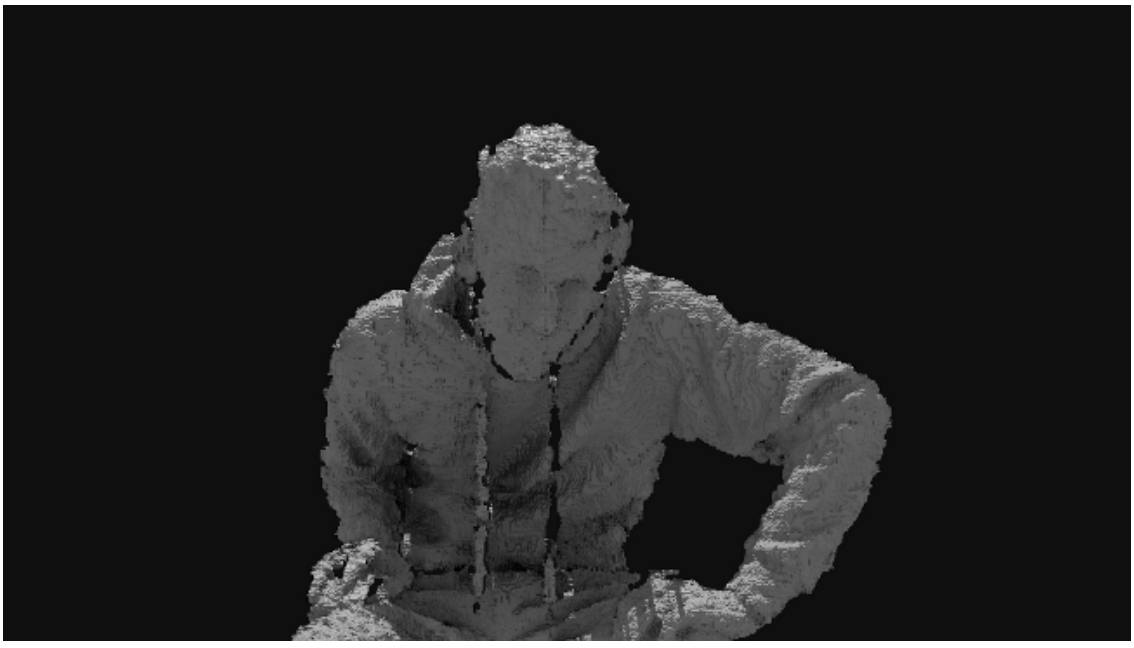


Abbildung 2.2.: Dreidimensionale Punktewolke einer Person [10]

Abbildung 2.2 ist eine Punktewolke von einer Person zu sehen. Diese Punktewolken können schnell mit moderner und immer kostengünstiger werdender Hardware, wie zum Beispiel die

Microsoft Kinect, erstellt werden [17]. Dies könnte die Generierung von Trainings- und Testdaten für weitere Tests enorm vereinfachen und beschleunigen. Mit genug dreidimensionalen Daten könnten die in dieser Arbeit vorgestellten Tests wiederholt werden. Wie *Lai und Fox* [12] gezeigt haben, ist es möglich, Objekte in Punktewolken zu erkennen und voneinander zu unterscheiden. Es stellt sich die Frage, ob Faltungsnetzwerke ein geeignetes Werkzeug darstellen könnten, um hochvariante Gesichter oder andere Objekte gleichen Typs voneinander zu unterscheiden oder wieder zu erkennen.

3. Klassifikation von Mustern

Mustererkennung ist die Fähigkeit, in einer Menge von Daten Regelmäßigkeiten, Wiederholungen, Ähnlichkeiten oder Gesetzmäßigkeiten zu erkennen [35]. Für diese Aufgaben gibt es in der Informatik viele verschiedene Verfahren. In dieser Arbeit liegt der Fokus auf der Erkennung von hochvarianten Mustern mit Faltungsnetzwerken. Da Faltungsnetzwerke auf Neuronalen Netzen basieren, werden in diesem Kapitel die Grundlagen für deren Aufbau und Funktionsweise erklärt.
Ein weiteres Verfahren im Kontext der Klassifizierung ist die Hauptkomponentenanalyse. Diese ermöglicht es große Datensätze auf markante Muster zu reduzieren. Die hervorgehobenen Merkmale eignen sich hervorragend, um Vergleiche und Erkennungen auszuführen. Die Hauptkomponentenanalyse ist ein statistisches Verfahren und wird oft in Kombination mit Neuronalen Netzen zur Klassifizierung von Mustern verwendet [32].

3.1. Neuronale Netze

Der Einsatz von Neuronalen Netzen in der Datenverarbeitung ist fast so alt wie der programmierbare Computer selbst [33]. Seither ist die Verwendung von künstlichen Neuronalen Netzen zum Approximieren von Funktionen ein häufig verwendetes Mittel in der Informatik. Die Idee hinter künstlichen Neuronalen Netzen ist die Nachahmung biologischer Prozesse im Gehirn. Da künstliche Neuronale Netze von den biologischen inspiriert sind, soll anhand des Vorbilds der Aufbau von Neuronen vertieft werden.

3.1.1. Biologische Neuronale Netze

Im Gehirn und Nervensystem von Menschen und Tieren sind Nervenzellen (Neuronen) hochgradig vernetzt [24]. Mit Hilfe dieser Netzstruktur (ca. 100 Millarden bis 1 Billion Neuronen) [29] werden Informationen im menschlichen Gehirn ausgewertet und verarbeitet. Diese Verarbeitung zwischen den Neuronen findet auf elektrochemischem Weg statt.

Die Abbildung 3.1 zeigt einen mikroskopischen Ausschnitt des Netzes mit mehreren vernetzten Neuronen. Hier ist (in der Mitte) ein Neuron farblich hervorgehoben. Ein Neuron hat

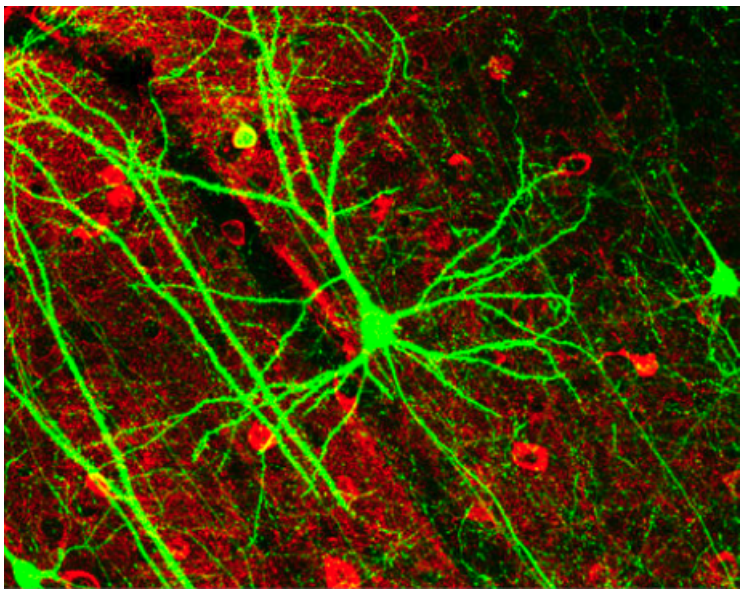


Abbildung 3.1.: Pyramidenzellen (grün hervorgehoben) in der menschlichen Großhirnrinde [31].

mehrere Eingangssignale und ein Ausgangssignal. Das Ausgangssignal wird durch mehrere Verbindungen an andere Neuronen als Eingangssignal weitergegeben. Je nach Eingangsreize eines Neurons kann ein Ausgangssignal erzeugt werden. Das Ausgangssignal wird dann an alle mit dem Ausgang verbundenen Neuronen weitergegeben. Auf diese Weise der Verarbeitung und Transportierung der Impulse werden im menschlichen Körper Entscheidungen getroffen, motorische Aktionen in Bewegung gesetzt und Reaktionen auf Ereignisse erzeugt.

3.1.2. Künstliche Neuronale Netze

Künstliche Neuronale Netze (engl. Artificial Neural Networks) sind der Versuch, die Eigenschaften eines Neuronalen Netzes in einem Computer umzusetzen (ca. 100 bis 1000 Neuronen). Bei künstlichen Neuronalen Netzen macht man sich bei der Verarbeitung von Daten einige Ideen aus den biologischen Neuronalen Netzwerken zu nutze. In vielen Dingen unterscheiden sich künstliche Neuronale Netze von ihrem Vorbild erheblich, da der Aufbau und die elektrochemischen Vorgänge zu komplex wären, um diese effizient nachzubilden. Inspiriert von dem biologischen Vorbild werden auch Neuronen verwendet. Im Gegensatz zum biologischen Vorbild fließen die Informationen vom Eingang zum Ausgang durch das Netz.

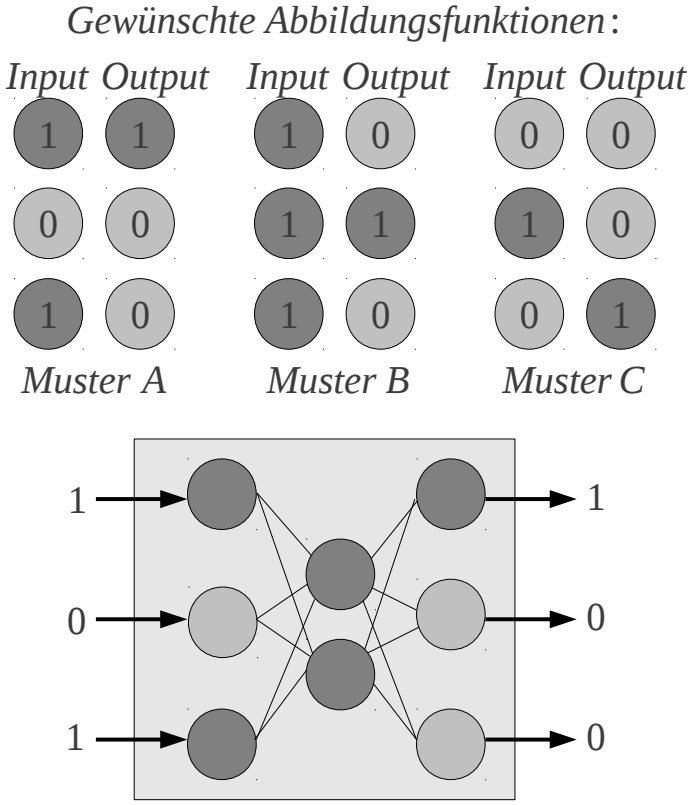


Abbildung 3.2.: Neuronales Netzwerk mit zwei Schichten von Neuronen und drei zu erkennenden Mustern

Grundidee ist es, eine komplexe Aufgabe anstatt mit Hilfe eines Algorithmus, durch ein Neuronales Netz zu bearbeiten. In einem Netz werden die Eingangsinformationen von Neuron zu Neuron weitergegeben und verändert. Jedes Neuron hat eine andere Parametrisierung die Eingangsinformationen zu bewerten. Ein künstliches Neuronales Netzwerk zeichnet sich

darin aus, dass es komplexe Aufgaben schnell durch die Berechnung seiner Neuronen ausführt. Jedes Neuron stellt seine Berechnung auf Basis von wenigen Additionen und Multiplikationen auf. Das Ergebnis eines Neurons wird dann durch eine Aktivierungsfunktion an andere Neuronen weitergegeben oder als Endergebnis bereitgestellt.
Auf diese Weise können Funktionen mit Neuronalen Netzen approximiert werden. In Abbildung 3.2 sind drei Muster gegeben. Mit der richtigen Parametrisierung der Neuronen könnte das gegebene Neuronale Netz diese Abbildungsfunktion realisieren. Die Informationen durchfließen das Netz der Neuronen, und anhand der Verbindungen zwischen den Neuronen und deren Parametrisierung wird eine Entscheidung getroffen.

Aufbau eines Neurons

In dem biologischen Vorbild gibt es verschiedene Arten von Neuronen [5]. Auch ein künstliches Neuron kann unterschiedlich aufgebaut sein. In dieser Arbeit wird aber nur auf den Aufbau des *Perzeptron-Neurons* eingegangen.

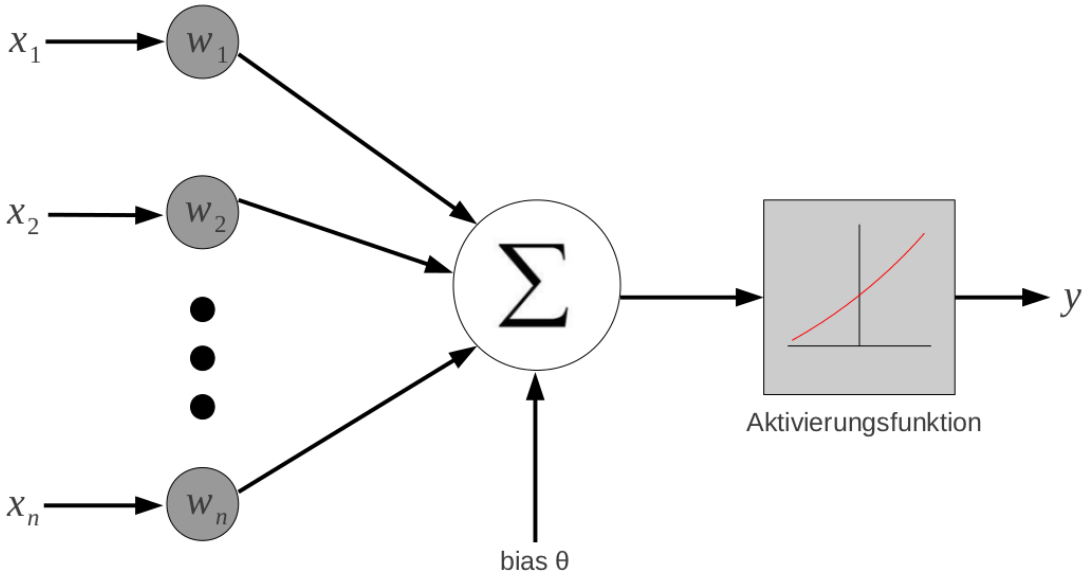


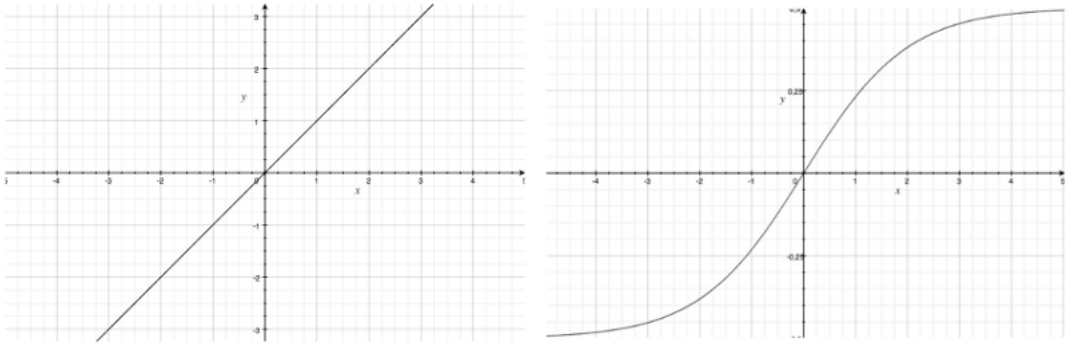
Abbildung 3.3.: Innerer Aufbau eines Perzeptron-Neurons

In Abbildung 3.3 ist der innere Aufbau eines Perzeptron-Neurons gezeigt. Jedes Perzeptron-Neuron hat eine bestimmte Anzahl von Eingängen. Jeder Eingang x_i eines Perzeptron-Neurons hat ein dazugehöriges Gewicht w_i. Dieses Gewicht ist ein Wert, der mit dem Eingangswert multipliziert wird. Nach der Multiplikation aller Eingänge mit den jeweiligen Gewichten werden diese summiert. Die Summe wird mit dem sogenannten Schwellenwert, den

bias θ, addiert und dann mit an die Aktivierungsfunktion ϕ übergeben. Das Ergebnis der Aktivierungsfunktion ist dann der Ausgangswert y des Neurons. Die gesamte Berechnung eines Neurons kann also auch als Gleichung 3.1 zusammengefasst werden.

$$y = \phi(\sum_{i=1}^{n} x_i w_i + \theta) \tag{3.1}$$

Als Aktivierungsfunktion können verschiedene Funktionen benutzt werden. In Abbildung 3.4 sind die häufig verwendeten lineare $y = x$ und sigmoide $y = \frac{1}{1+e^{-x}}$ Aktivierungsfunktionen gezeigt.



(a) linear (b) sigmoid

Abbildung 3.4.: Beispiele möglicher Aktivierungsfunktionen θ

Aufbau eines Neuronalen Netzes

In Abbildung 3.5 ist eine beispielhafte Darstellung eines künstlichen Neuronalen Netzes zu sehen. Ein Neuron hat ein Ausgangssignal. Dieses kann aber auch als Eingang für mehrere Neuronen dienen.
Die Schwierigkeit bei künstlichen Neuronalen Netzwerken besteht darin, die richtige Einstellung jedes Neurons zu finden. Das Ergebnis einer 'Berechnung' eines Neuronalen Netzwerkes hängt also von der Auswertung der Eingänge jedes einzelnen Neurons in einem Netzwerk ab. Mit der richtigen Einstellung jedes Neurons in einem Netzwerk, könnte man eine Aufgabe optimal approximieren. Für jede Aufgabe und Berechnung eines Neuronalen Netzes ist die Einstellung der Neuronen unterschiedlich. Da es sich sehr schwer gestaltet,

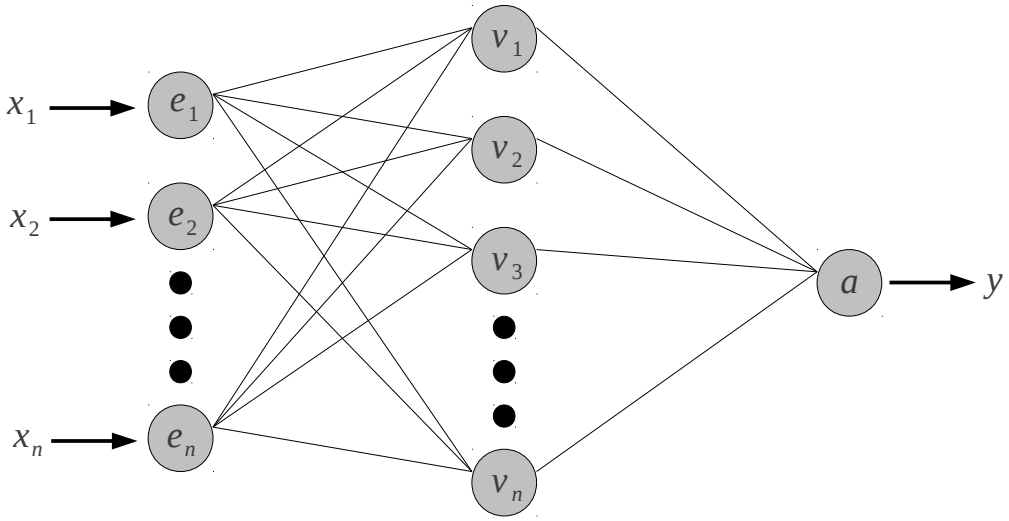


Abbildung 3.5.: Aufbau eines kleinen künstlichen Neuronalen Netzes. Eingabedaten $x_1 \ldots x_n$, verarbeitet durch Neuronen $e_1 \ldots e_n$ und $v_1 \ldots v_n$ und ausgegeben von Neuron a zu Ergebnis y

in einem Netz alle Neuronen per Hand zu setzen und es keinen Algorithmus gibt mit dem man ein Neuronales Netz bei endlicher Laufzeit mit den richtigen Werten initialisieren kann, muss eine alternative Vorgehensweise für das Suchen der idealen Einstellung verwendet werden. Diese Vorgehensweise wäre das dynamische Anpassen der Neuronenwerte durch trainieren des Netzwerkes mit verschiedenen Eingangsdaten. Der Aufbau von Neuronalen Netzwerken hängt von den verwendeten Neuronen ab. Dieses Kapitel beschränkt sich auf den Aufbau von Multilayer-Perzeptron-Netzwerken. Multilayer-Perzeptron-Netzwerke haben mindestens drei Schichten an Neuronen. In Abbildung 3.6 ist ein Beispiel eines solchen Netzwerkes zu sehen. Die erste Schicht ist die *Eingangsschicht*. Hier werden die Eingangsdaten des gesamten Netzes direkt mit den ersten Neuronen verbunden. Die zweite Schicht wird *versteckte Schicht* genannt. Ein Neuronales Netz kann beliebig viele versteckte Schichten besitzen. Wichtig ist nur, dass die Neuronen einer Schicht immer nur mit Neuronen der direkt benachbarten Schichten verbunden sind. Die dritte und letzte Schicht ist die *Ausgangsschicht*. Hier werden die letzten Berechnungen in den Neuronen durchgeführt, und

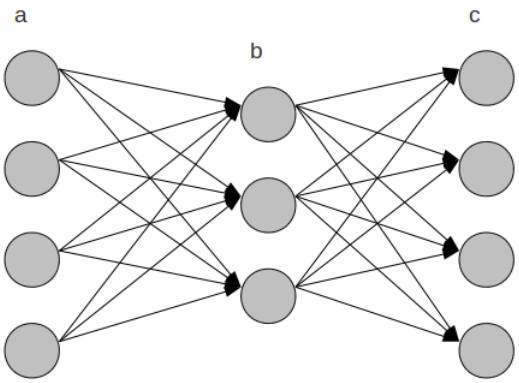


Abbildung 3.6.: Ein Neuronales Netz mit mehreren Schichten. (a) Eingabeschicht, (b) versteckte Schicht, (c) Ausgabeschicht

ihre Ergebnisse zeigen die Entscheidung des gesamten Neuronalen Netzes. Die Neuronen einer Schicht sind immer gleich aufgebaut. Die Neuronen der verschiedenen Schichten unterscheiden sich aber meistens in ihrer Aktivierungsfunktion. Eingangsschichten besitzen oft lineare Funktionen, wogegen die versteckten Schichten hauptsächlich sigmoide Funktionen verwenden. Die Aktivierungsfunktion der Ausgabeschicht hängt von der Anwendung des Neuronalen Netzes ab. Bei Funktionsapproximationen werden lineare - bei Klassifizierungen sigmoide Funktionen benutzt.

Lernvorgang eines Neuronalen Netzwerkes

Es gibt verschiedene Arten ein künstliches Neuronales Netzwerk zu trainieren. In dieser Arbeit ist der Fokus auf nur eine Art dieses Lernens gerichtet. Das Grundprinzip dieses Lernvorgangs kann man mit dem Lernen mit Hilfe eines Lehrers vergleichen. Ein Lehrer, der dem Lernenden eine Aufgabe lösen lässt und ihm dann, im Falle eines Fehlers, korrigiert. Dieses Vorgehen kann auch auf künstliche Neuronale Netzwerke zum lernen (engl. supervised learning) angewendet werden. Wird ein Neuronales Netz trainiert, so bekommt es Eingabedaten, die das Netz durchlaufen sollen. Zu den jeweiligen Eingabedaten muss ein Ergebnis vorliegen. Dann wird jedes Ergebnis des Netzes mit dem jeweiligen Ergebnis verglichen. Tritt eine Abweichung des Ergebnisses auf, so müssen alle Gewichte und der bias θ in den Neuronen angepasst werden. Eine schematische Darstellung des Vorgangs für ein Neuron wird in Abbildung 3.7 aufgezeigt. Die Ausgabe des Perzeptron-Neurons wird hier negiert und mit dem gewünschten Ergebnis summiert. Der daraus folgende Fehler wird dann, um eine gegenseitige Kompensation von positivem und negativem Fehler auszuschließen, quadriert.

Dies resultiert in dem Fehler E_p. Je kleiner der Fehler E_p, um so näher das Ergebnis des Neurons an dem vom Lehrer gesetzten Ziel.

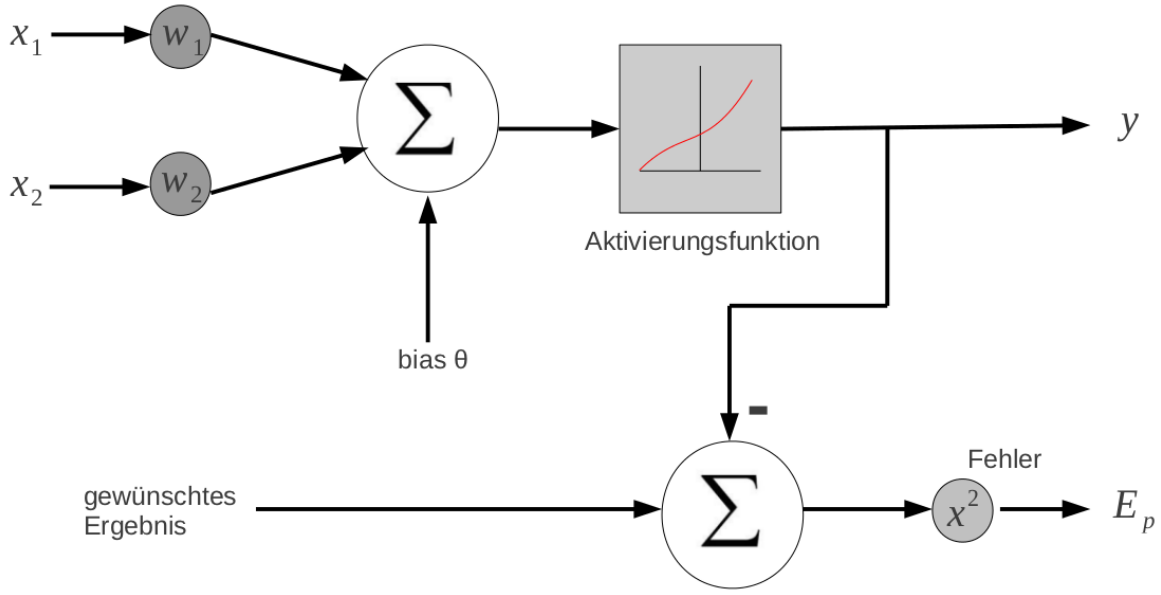


Abbildung 3.7.: Aufbau des Trainings eines Perzeptron-Neurons

Zum Finden des kleinsten Fehlers E_p für ein Neuron können verschiedene Ansätze verfolgt werden. An dem Beispiel in Abbildung 3.7 betrachtet, sollen hier zwei dieser Ansätze verglichen werden:

- Brute-force: Um den kleinsten Fehler E_p für die Gewichte w_1 und w_2 zu finden, werden alle Kombinationen von Gewichtwerten für die Eingabedaten x_1 und x_2 verglichen. In Abbildung 3.8 ist die Anzahl der möglichen Kombinationen in einem zweidimensionalen Raum zu sehen. Um den kleinsten Fehler in diesem Fehlergebirge zu finden, müssen alle Punkte durchlaufen und miteinander verglichen werden. Bei der Suche nach dem kleinsten Fehler E_p in dem Fehlergebirge würde pro iterativen Durchlauf pro Neuron $O(n^2)$ in Anspruch nehmen. In einem größeren Neuronalen Netz hat ein Neuron deutlich mehr Gewichte, die angepasst werden müssen. Dies würde zu einem vieldimensionalen Merkmalsraum führen und für einen Lernschritt ein $O(n^m)$, wobei m für die Anzahl der Gewichte der Neuronen steht, in Anspruch nehmen.
- Gradientenabstieg: Hier wird das Prinzip verfolgt, in einer vorgegebenen Anzahl von Schritten, einen relativ kleinen Fehler im Fehlergebirge zu finden. Dies geschieht nicht indem alle möglichen Gewichtskombinationen durchiteriert werden, sondern indem vom momentanen Fehler (im Fehlergebirge) ein Schritt in die Richtung getan wird,

in der der Abstieg am steilsten ist. Es gibt mehrere Algorithmen für den Gradientenabstieg. Einer ist der sogenannte *Backpropagation* Algorithmus. In dieser Arbeit wird dieser Ansatz für die Suche nach dem kleinsten Fehler erläutert, weil er sehr häufig in Perzeptron-Neuron basierten Neuronalen Netzwerken zum Lernen verwendet wird. Genauere Details darüber sind in Kapitel 4 beschrieben.

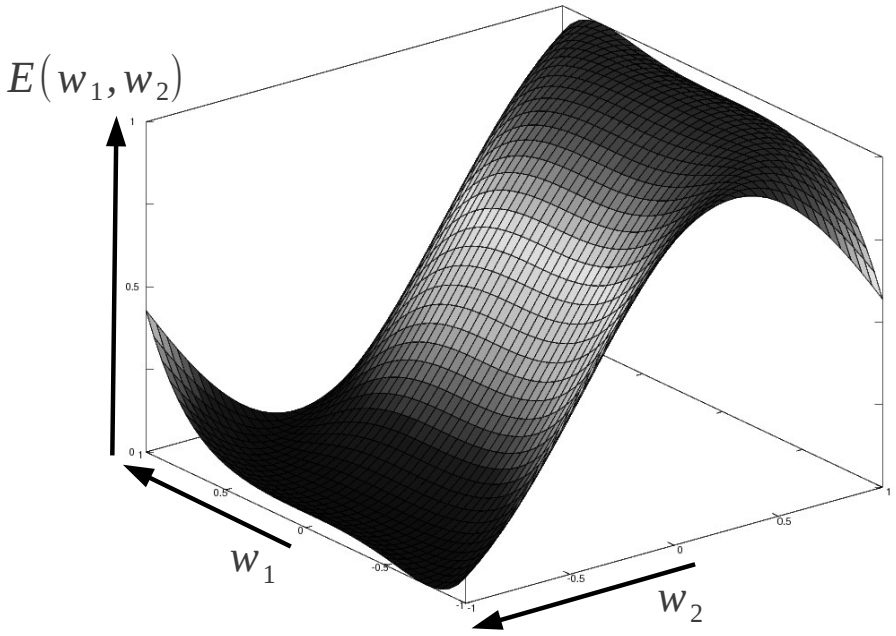


Abbildung 3.8.: Fehlergebirge eines Neurons mit zwei Eingabedaten und Gewichten

Wie in Abbildung 3.9 zu sehen, findet der *Backpropagation* Algorithmus innerhalb weniger Schritte einen sehr kleinen Fehler für die Gewichte. Dies kann für beliebig viele Dimensionen verwendet werden. Bei größeren Eingabedaten und Dimensionen können sehr viele Schritte notwendig sein, um einen kleinen Fehler für die Gewichte zu finden.

Um ein ganzes Neuronales Netz mit Hilfe des *Backpropagation* Algorithmus zu trainieren, muss wie in Abbildung 3.10 gezeigt, jedes Neuron von Ausgabeschicht zu Eingabeschicht angepasst werden. Hierzu wird die tatsächliche Ausgabe des Netzes mit der gewünschten Ausgabe verglichen und ein Differenzvektor gebildet. Dieser Differenzvektor wird für die Justierung aller Neuronen benötigt. Die Justierung der Gewichte fängt in der Ausgabeschicht an. Von hier aus arbeitet sich der *Backpropagation* Algorithmus rückwärts durch das Netz, bis in die Eingabeschicht vor und justiert die Gewichte aller Neuronen um einen Faktor. Der Faktor der Änderung hängt von dem Differenzvektor ab.

Das Training der ganzen Netzstruktur wird für jedes Trainingsset (bestehend aus Eingabedaten und dem gewünschten Ergebnis) vorgenommen. Nach dem Training sollte das Netz

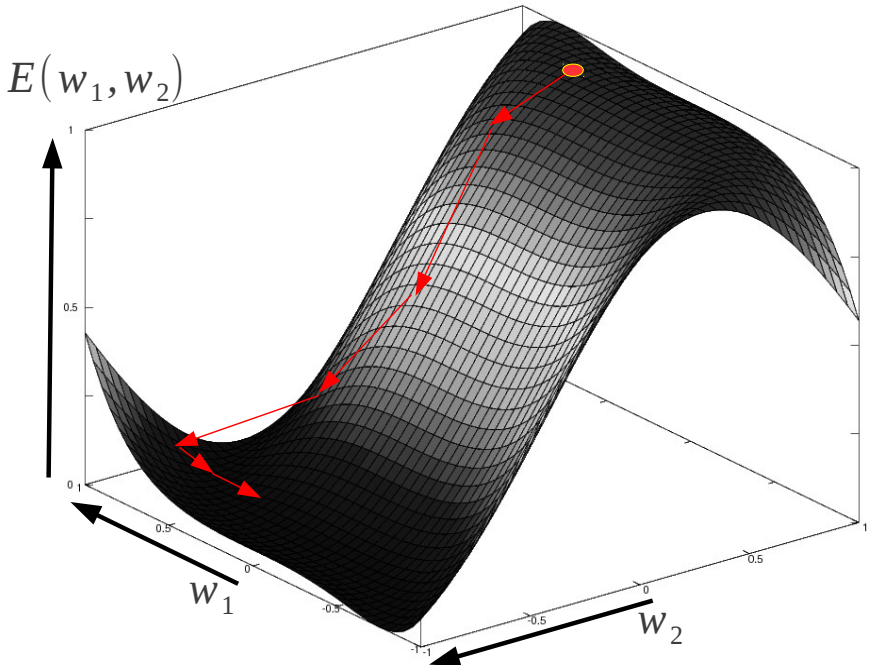


Abbildung 3.9.: Der Gradientenabstieg zum Finden des kleinsten Fehlers mit dem *Backpropagation* Algorithmus. Jeder rote Pfeil stellt eine Iteration des Gradientenabstiegs dar

mit Hilfe eines Testsets auf die Treffergenauigkeit überprüft werden. Wichtig hierbei ist, dass die Daten aus dem Testset nicht zum Trainieren des Netzes benutzt wurden. Um die Treffergenauigkeit des Netzes zu Testen, müssen die Testdaten dem Netz vorher nicht bekannt gewesen sein.

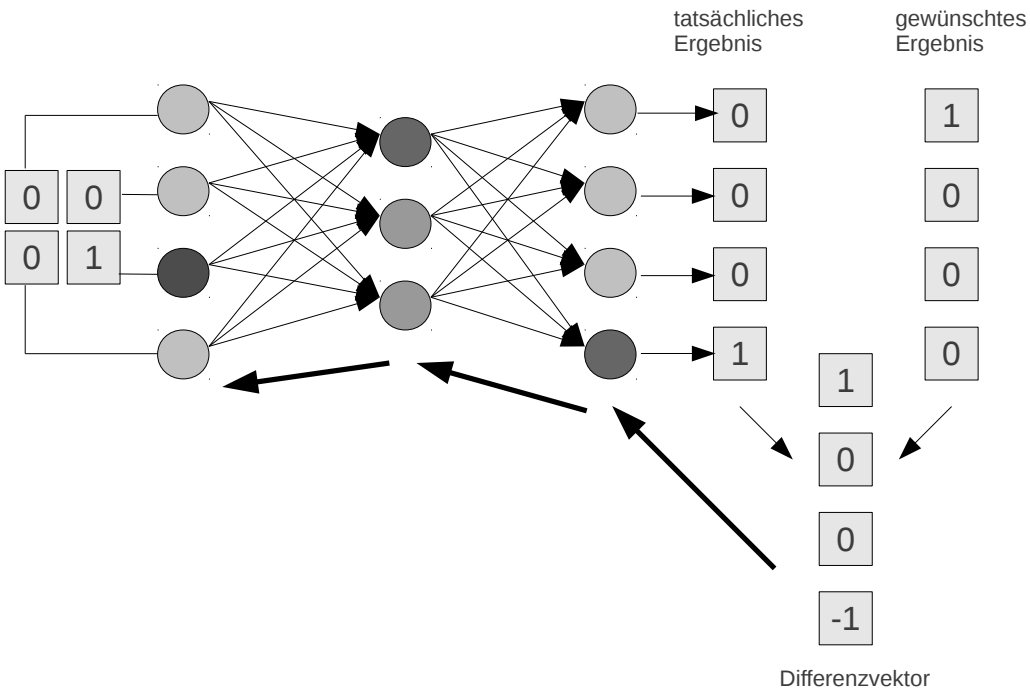


Abbildung 3.10.: Anwenden des *Backpropagation* Algorithmus auf ein gesamtes Netzwerk

3.2. Hauptkomponentenanalyse

Die Hauptkomponentenanalyse (engl. Principal Component Analysis) ist ein statistisches Verfahren, das dazu dient, umfangreiche Datensätze zu reduzieren und zu vereinfachen. Daten mit vielen Dimensionen können mit Hilfe der Hauptkomponentenanalyse komprimiert werden. Die Anzahl der Dimensionen der Daten wird dabei reduziert, ohne die markanten Muster in den Daten zu verlieren. Die reduzierten Teile der Daten sind für die Erkennung des Musters entbehrlich. Diese Technik stellt sich insbesondere in der Kombination mit Neuronalen Netzwerken als sehr effizient heraus, da dort möglichst kleine Eingabevektoren die Geschwindigkeit und Erkennungsrate steigern.

3.2.1. Analyse

Bei der Hauptkomponentenanalyse geht es darum, aus einem Satz von erfassten Daten, Merkmale zu finden. Diese Merkmale werden *Hauptkomponenten* oder auch *Eigenvektoren*

genannt. Gefundene *Hauptkomponenten*, die für den Informationsgehalt der Daten gering sind, könnten reduziert werden. Eine *Hauptkomponente* basiert auf den Daten zu unterschiedlichen Faktoren.
Hauptkomponenten werden iterativ aus Daten berechnet. In Abbildung 3.11 ist eine Punktewolke zu sehen, in der zehn Punkte mit jeweils drei Werten x, y, z Dreidimensional dargestellt sind. Die Hauptkomponenten werden ähnlich der Lösung einer Linearen Ausgleichsrechnung [32] ermittelt. Die Berechnung resultiert in einer Geraden, die alle Punkte möglichst gut approximiert.

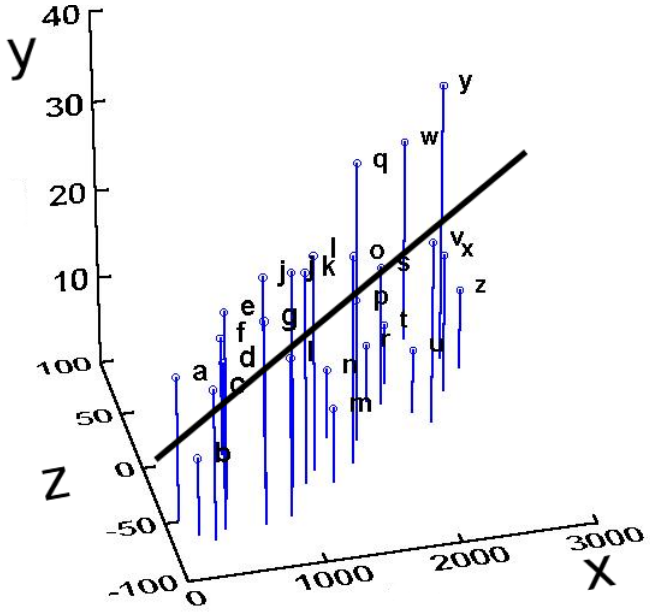


Abbildung 3.11.: Punktewolke [21] mit ihrer ersten Hauptkomponente - einer Geraden, die alle Punkte approximiert

Die nächsten Hauptkomponenten müssen orthogonal zur vorherigen sein und durch deren Mittelpunkt gehen. Es können nur so viele Hauptkomponenten berechnet werden, wie der Datensatz Dimensionen hat. Nachdem die Hauptkomponenten einer Datenmenge berechnet wurden, dienen diese als neues kartesisches Koordinatensystem, wie in Abbildung 3.12 dargestellt. Die neuen Achsen (die Hauptkomponenten), an denen sich die Daten orientieren bilden neue Wertebereiche ab, die sich aus den alten Wertebereichen der Achsen zu unterschiedlichen Gewichtungen zusammenstellen [28]. Dies kann an einem Beispiel verdeutlicht werden. Ein Punkt hat im ursprünglichen Koordinatensystem die Werte $y = 15, x = 120, z = -47$. In dem neuen Koordinatensystem, auf Basis der Hauptkomponenten, stellen sich die Werte aus $r = 0.465 * y$, $h = 0.623 * x$, $t = 0.153 * z$ zusammen.

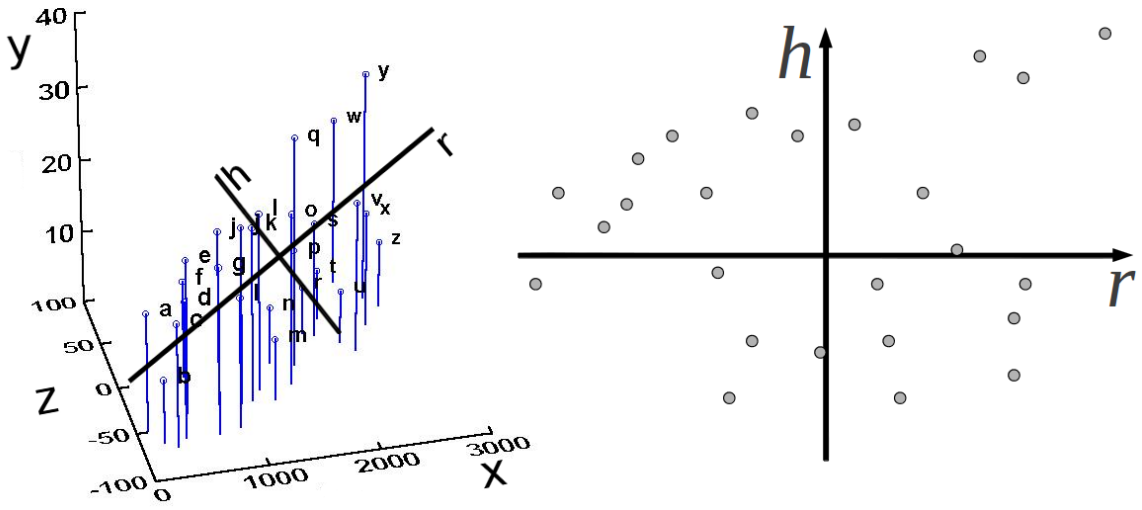


Abbildung 3.12.: Punktewolke mit den ersten zwei Hauptkomponenten *r* und *h* berechnet (links). Die Hauptkomponenten reichen, um die Daten in einem neuen Koordinatensystem (rechts) anzuzeigen

Die neuen Werte der Punkte sind eine Kombination der alten Werte. Mit Hilfe der genauen Kombination dieser Werte können Transitionen von einem Koordinatensystem zum anderen durchgeführt werden.

3.2.2. Reduzierung der Dimensionen

Die in 3.2.1 vorgestellten Hauptkomponenten können dazu benutzt werden, ein neues Koordinatensystem für die Daten zu bilden. In diesem neuen Koordinatensystem werden die Daten auf Basis der kombinierten alten Werte dargestellt. Häufig können die neuen Achsen (Hauptkomponenten) inhaltlich nicht interpretiert werden.
Mit Hilfe der sogenannten *totalen Varianz* kann festgestellt werden, welche Hauptkomponenten mehr Einfluss auf die Beschaffenheit der Daten haben als die anderen. Die *totale Varianz* kann als Maß für die Wichtigkeit der Hauptkomponenten dienen.
Die *Varianz* berechnet sich aus den Entfernungen der Punkte zur jeweiligen Hauptkomponente. Die aufsummierten Quadrate der Abstände in Richtung der Hauptkomponente bilden die *Varianz* der Daten. Je mehr Hauptkomponenten berechnet werden, um so geringer werden die Distanzen der Punkte zu den Hauptkomponenten. Die *Varianz* wird also bei der *i*-ten Hauptkomponente immer geringer. Die *totale Varianz* der Daten ist die Summe aller *Varianzen* der Hauptkomponenten.
Hauptkomponenten mit einer geringen *Varianz* können entfernt werden und die Daten somit

in einem Koordinatensystem mit weniger Dimensionen dargestellt werden. Die reduzierten Daten können in ihrem neuen Koordinatensystem so in weiterführenden Datenverarbeitungen benutzt werden.

4. Faltungsnetzwerke

Faltungsnetzwerke (engl. Convolutional Neural Networks) sind eine Abart der in Kapitel 3.1.2 vorgestellten Neuronalen Netze. Faltungsnetzwerke setzen nicht ausschließlich auf die Verwendung von den in Sektion 3.1.2 vorgestellten Perzeptron-Neuronen, sondern zusätzlich auf eine, aus der Signalverarbeitung stammende Technik zur Filterung von Daten. Aus dieser Technik heraus haben Faltungsnetzwerke ihre Bezeichnung gewonnen, denn die Technik aus der digitalen Signalverarbeitung wird *Faltung* genannt. Eine genauere Erklärung zum Aufbau von Faltungsnetzen ist in Sektion 4.1.2 zu finden.
Die Idee der Faltungsnetzwerke basiert auf dem in den 80er Jahren entwickelten Neocognitron [7]. Faltungsnetzwerke als selbstlernende Netze sind erstmals 1998 in Erscheinung getreten. Hier haben *Simard* [27] und *LeCun* [13] gezeigt, dass mit Faltungsnetzwerken Lernalgorithmen, wie der Backpropagation-Lernzyklus, benutzt werden können. Dies hat ermöglicht, dass Faltungsnetzwerke mit Bilderdaten trainiert werden können, ohne dass sonstige Einstellungen für das spätere Klassifizieren nötig sind. Mit den vorgestellten Faltungsnetzwerken [13] konnten hochvariante Ziffern in Bildern robuster klassifiziert werden, als dies mit herkömmlichen Perzeptron-Netzwerken möglich wäre. In den folgenden Jahren haben sich weitere Tests zur Klassifizierung von Menschen und Objekten mit Hilfe von Faltungsnetzwerken [16] in Bildern als sehr erfolgreich herausgestellt. Die Tatsache, das Faltungsnetzwerke in der Lage sind, komplexe Formen in Bilderdaten zu erkennen, macht sie zu einem kompetenten Nachfolger von Perzeptron-basierten Neuronalen Netzwerken im Bereich der Bilderklassifizierung.

4.1. Funktionsweise und Aufbau

Die Funktionsweise von Faltungsnetzwerken basiert auf der Filterung von Eingabedaten. Der Datenfluss innerhalb eines Faltungsnetzwerkes unterscheidet sich nicht erheblich von den in Sektion 3.1.2 vorgestellten Netzwerken. Die Eingangsdaten durchlaufen das Faltungsnetzwerk Schicht für Schicht. Der Unterschied besteht in der Verknüpfung der Schichten miteinander und der Verarbeitung der Daten in den jeweiligen Schichten. In den Schichten eines Faltungsnetzwerkes werden die Daten aus der vorherigen Schicht gefiltert. Um auf die Funktionsweise tiefer eingehen zu können, muss zuerst der Begriff Filterung bzw. Faltung von Bilderdaten weiter erklärt werden.

4.1.1. Faltung

In einer Faltung werden die Eingabedaten mit einer sogenannten Faltungsmaske bearbeitet, damit in dem Resultat bestimmte Merkmale hervorgehoben oder unterdrückt werden. Eine Faltung ist eine mögliche Form der Bildfilterung.

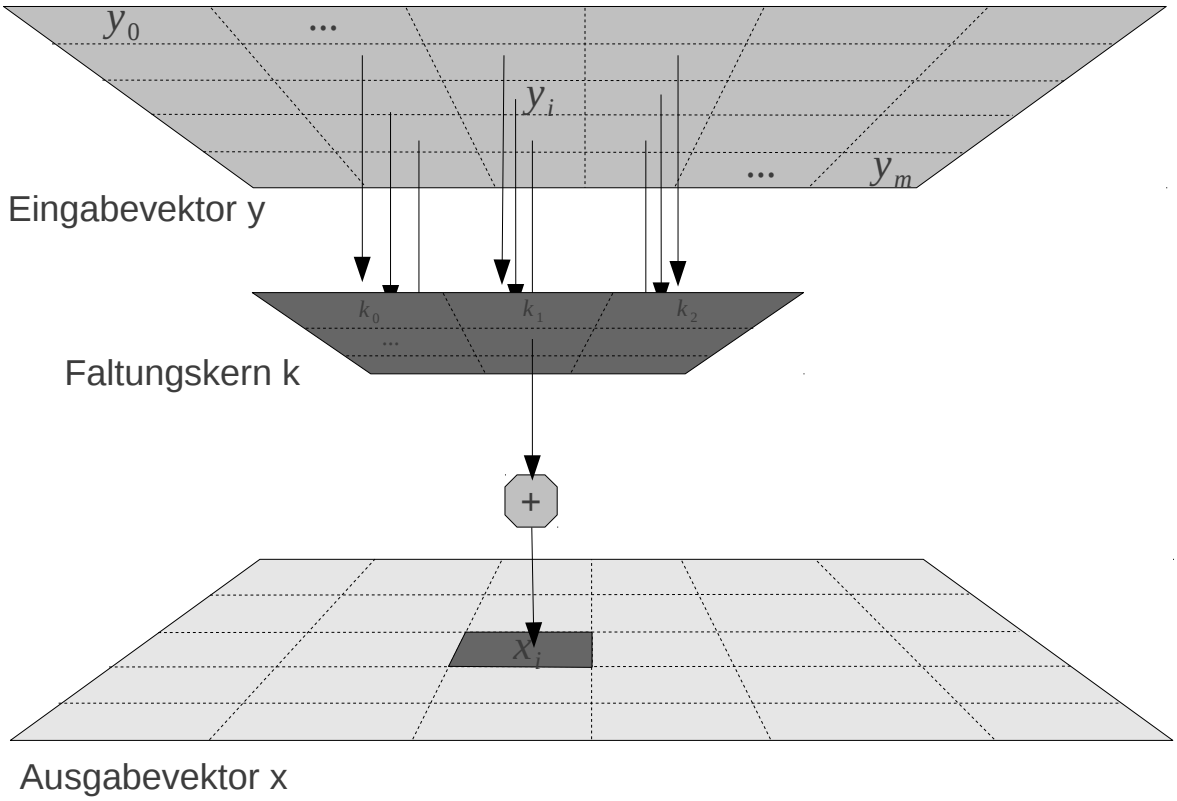


Abbildung 4.1.: Faltung eines zweidimensionen Vektors mit einer 3x3 Faltungsmaske)

Diese Filterung kann auf komplexer Ebene [19] statt finden oder auf der Basis von Faltungsmasken. In dieser Arbeit wird nur auf die Filterung auf Basis von Faltungsmasken weiter eingegangen, da Faltungsnetzwerke ihre Faltung auch auf diese Weise berechnen.
Bei einer Faltung mit Faltungsmasken wird ein Eingabevektor durchlaufen. Jedes Element y_i dieses Vektors wird mit der Faltungsmaske k multipliziert und summiert. Bei einer Multiplizerung der Faltungsmaske wird nicht nur y_i, sondern auch seine benachbarten Elemente $v_{i\pm1}...v_{i\pm n}$ in die Berechnung durch die Faltungsmaske mit einbezogen, wobei n von der Größe der Faltungsmaske k abhängt. Siehe Abbildung 4.1 für eine grafische Darstellung der Faltung.

Das Ergebnis kommt in den Ausgabevektor x. Die genaue Berechnung von x resultiert aus der in 4.1 abgebildeten Gleichung.

$$x_i = \sum_{i=0}^{n} v_i k_j \tag{4.1}$$

Mit der Filterung von Bildern können viele verschiedene Effekte erzielt werden. In Abbildung 4.2 ist ein Beispiel einer solchen Filterung zu sehen. Hier wurde das linke Graustufenbild mit der in 4.2 abgebildeten Laplace-Faltungsmaske gefiltert.

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{4.2}$$

Das mittlere Graustufenbild in Abbildung 4.2 ist das Resultat dieser Laplace-Faltung. Laplace Filterungen werden oft zur Bestimmung von Helligkeitswendepunkten verwendet. Initialisiert man die Faltungsmaske mit zufälligen Werten, so ist der resultierende Filterungseffekt auch ungewiss. Wie in Abbildung 4.2 rechts zu sehen, können bei einer solchen Filterung prägnannte Muster im Bild in Erscheinung treten. Die Eingabedaten eines Faltungsnetzwerkes werden auf die selbe Weise gefiltert. Die Faltungsmasken werden hier auch anfangs zufällig intialisiert, aber mit einem Trainingsprozess angepasst.

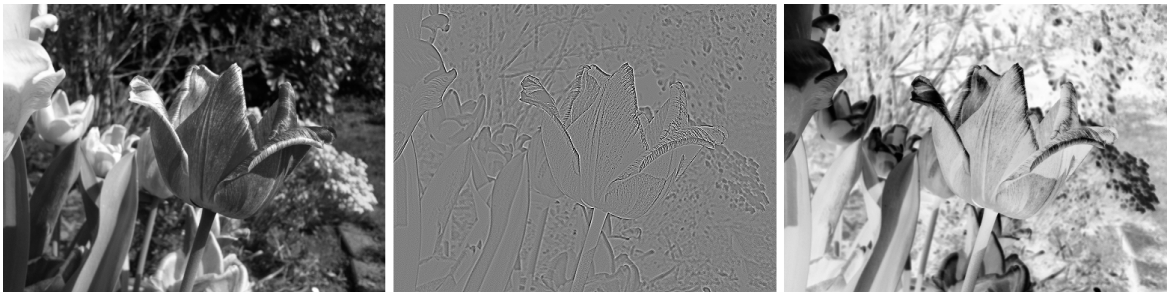


Abbildung 4.2.: Ungefiltertes Bild (links), mit 3x3 Laplace-Faltungsmaske gefiltertem Bild (mitte) und einem 3x3 zufällig initialisierter Faltungsmaske gefiltertem Bild (rechts)

Eine Faltung kann auch benutzt werden, um Daten mit anderen abzugleichen [11]. Diese Art der Faltung nennt man Korrelation. In der Bildverarbeitung nutzt man Korrelationsfunktionen unter anderem zur genauen Lokalisierung eines Musters in einem Bild [34]. Ein simples eindimensionales Bild ist als Beispiel in Abbildung 4.3 gegeben. Die Graustufenwerte der Pixelreihe sind als Zahlenwerte angegeben. Die Pixelwerte werden mit der Korrelationsmaske

gefaltet und der daraus resultierende Korrelationswert stellt ein Maß für die Wiedererkennung des Musters dar. In dem Beispiel ist der höchste Korrelationswert 85 und beziffert die Pixelfolge $\begin{pmatrix}3 & 8 & 4\end{pmatrix}$, die dem zu findenden Muster $\begin{pmatrix}3 & 7 & 5\end{pmatrix}$ von allen anderen Pixelfolgen im Bild am nächsten kommt. Auf diese Weise können auch zweidimensionale Bilder korreliert werden und so prägnante Muster lokalisiert werden. Diesen Effekt macht sich die Verarbeitung in Faltungsnetzwerken zu Nutze.
In einem Faltungsnetzwerk sind die Werte der Filtermasken den Gewichten der Perzeptron-Neuronen ähnlich. Je nach Einstellung der Gewichte werten Neuronen Ihre Eingabedaten anders aus. Dies passiert in Faltungsnetzwerken analog dazu, nur das hier Eingabebilder je nach Filtermaskeneinstellung anders gefaltet und neue Ausgabebilder generiert werden. In Faltungsnetzwerken können die Filtermasken - also die Gewichte - mit einem Lernzyklus, wie für Perzeptron-Neuronen in Sektion 3.1.2 vorgestellt, eingestellt werden. So entstehen durch das Training mit Lernalgorithmus und vielen Trainingsdaten angepasste Filtermasken. Die Eingabedaten werden dann durch das Faltungsnetzwerk je nach Beschaffenheit der verwendeten Trainingsdaten anders gefiltert und auswertet.
Durch jede Filterung eines Bildes verkleinert sich das Resultat um einen Pixelring, der so breit ist, wie die $\frac{Faltungsmaskenbreite+1}{2}$. Da die Faltungsmaske die äußersten Pixel eines Bildes mit seiner Maske nicht erfassen kann (die äußersten Pixel haben keine Nachbarn), werden diese bei der Berechnung ignoriert.

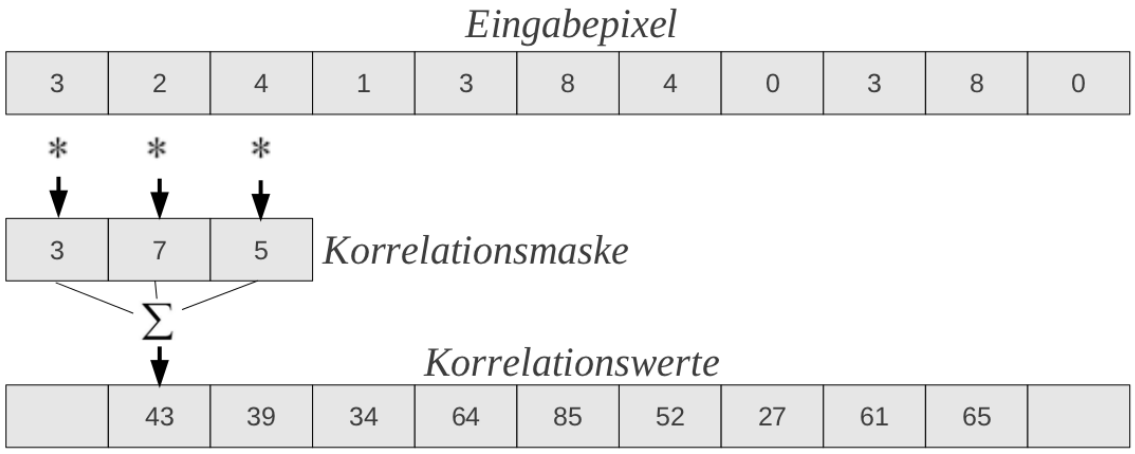


Abbildung 4.3.: Beispiel zur Lokalisierung ähnlicher Muster einer 1x3 Korrelationsmaske in einer Pixelreihe

4.1.2. Schichten

Neuronale Netzwerke sind in Schichten aufgebaut, wie in Sektion 3.1.2 erläutert. Faltungsnetzwerke bestehen aus Eingabeschicht, mehrere versteckte Schichten und einer Ausgabeschicht. Der wesentliche Unterschied zu herkömmlichen Multilayer-Perzeptron-Netzwerken besteht in der Art, wie die Daten verarbeitet und durch das Netz weitergereicht werden. Eine dieser anderen Verarbeitungsarten ist die in Sektion 4.1.1 erläuterte Faltung.
Die Schichten in einem Faltungsnetzwerk unterscheiden sich in ihrer Funktion. Im Gegensatz zu den Multilayer-Perzeptron-Netzwerken, in der jede Schicht aus den selben Neuronen aufgebaut ist, gibt es in Faltungsnetznetzwerken drei unterschiedliche Schichten, die sich in ihrer Funktion unterscheiden:

- Convolution-Schicht
- Subsampling-Schicht
- Perzeptron-Schicht

Die Convolution- und Subsampling-Schichten bestehen aus mehreren parallelen Faltungen, genannt *feature maps* [14]. Ähnlich mehreren Neuronen, die parallel in einer Schicht arbeiten, werden die *feature maps* in Faltungsnetzwerken parallel gefaltet. Jede Schicht hat dabei seine eigene Größe für die *feature maps* in Pixeln. Eine Schicht nimmt die Daten der vorherigen Schicht und faltet sie in jeder seiner *feature maps* mit einem Faltungskern. Wie in Abbildung 4.4 zu sehen, folgt auf jede Convolution-Schicht (C_n) eine Subsampling-Schicht (S_n). Diese Schichten arbeiten immer in Paaren.

Eine Subsampling-Schicht sorgt dafür, dass die Größe der Daten von der vorherigen Schicht um Faktor zwei verringert werden. Je nach Aufbau eines Netzes wechseln sich eine beliebige Anzahl von Convolution-Schicht- und Subsampling-Schicht-Paare bis zur Ausgabeschicht ab. Die Ausgabeschicht besteht aus Perzeptron-Neuronen, die mit der vorherigen Schicht vollvernetzt sind. Es ist auch möglich, wie in dem von *Yann LeCun* konstruierten LeNet-7 [23], mehrere vollvernetzte Perzeptron-Schichten vor die Ausgabe zu setzen.
Die Neuronen der Ausgabeschicht geben bei ihrer Berechnung einen Ausgabewert aus. Jedes Faltungsnetzwerk hat einen Schwellenwert, den sog. *Threshold*, der bestimmt, wann ein Ausgabewert ein Klassifizierungstreffer ist. Bei, zum Beispiel, einem *Threshold* von 0.8 gelten alle Neuronenberechnungen von 0.8 und höher als Treffer. Die Ausgabewerte sind von 0 bis 1.0 normalisiert.

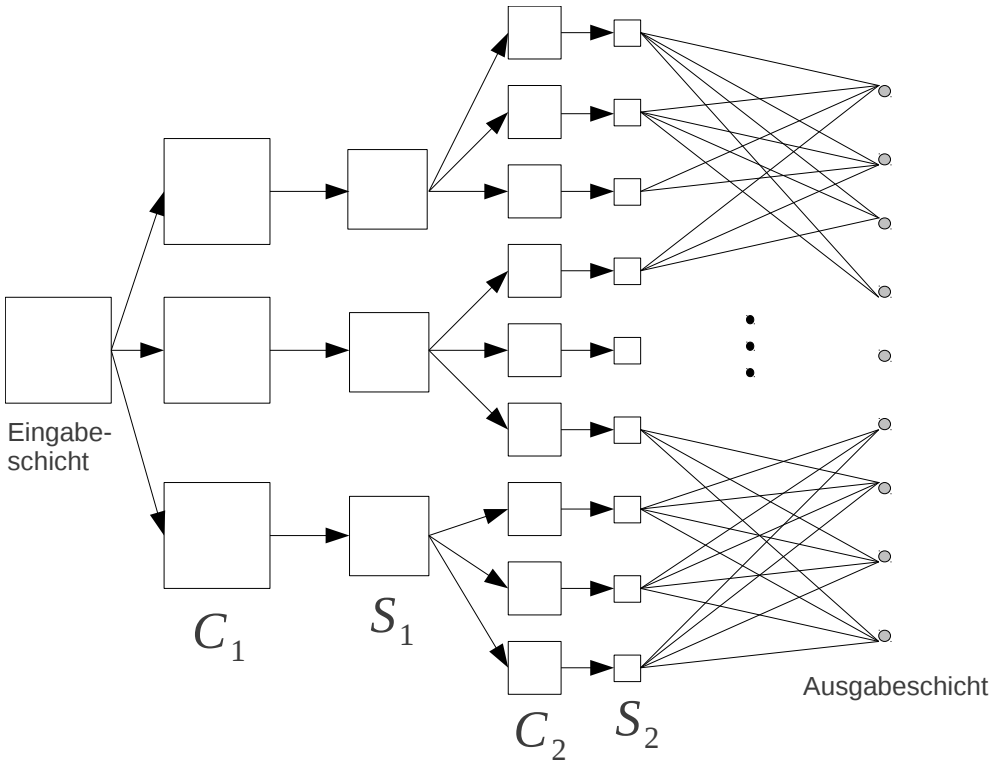


Abbildung 4.4.: Aufbau der Schichten eines Faltungsnetzwerkes

4.2. Lernvorgang

Das in Sektion 3.1.2 beschriebene Lernprinzip für das Multilayer-Perzeptron, kann auch für Faltungsnetzwerke angewendet werden, wobei die Gewichte in einem Faltungsnetzwerk, die Werte des Filterkerns sind. Diese gilt es, mit dem Lernvorgang für die jeweilige Aufgabe richtig einzustellen. Dies erfolgt über den Backpropagation-Lernalgorithmus, der, wie in 3.10 beschrieben von Ausgabeschicht zu Eingabeschicht alle Gewichte anpasst.
Bei dem Lernvorgang ist besonders herauszustellen, dass sich mehrere *feature maps* Gewichte - also Filterkerne - teilen [6] können. Hier ist wieder Convolution-Schicht und Subsampling-Schicht zu unterscheiden. In Subsampling-Schichten gibt es für jede Verbindung zur vorherigen Convolution-Schicht jeweils nur ein Gewicht, das eingestellt und somit trainiert werden kann. In Convolution-Schichten wiederum teilen sich die *feature maps* der jeweiligen Schicht einen Faltungskern.

5. Realisierung und Test

In dieser Arbeit wurden Faltungsnetzwerke auf Ihre Leistungsfähigkeit bei der Erkennung von hochvarianten Mustern getestet. Diese Testreihe wurde mit zwei unterschiedlichen Faltungsnetzwerken durchgeführt. Das erste Faltungsnetzwerk stützt sich auf die Arbeit von *Nikolay Chumerin*, der in Matlab ein Faltungsnetzwerk auf der Basis von *Yann LeCun's* LeNet-5 [3] umgesetzt hat. Dieses Matlabprogramm wurde mit dem Ziel entwickelt, die Erkennungsrate von Buchstaben und Ziffern von LeNet-5, erläutert in [14], zu erreichen. Das in dem Matlabprogramm verwendete Faltungsnetzwerk musste für die jeweiligen Testdaten angepasst werden.
Das zweite in den Tests verwendete Faltungsnetzwerk wurde dem Projekt EBlearn++ [23] entnommen. EBlearn++ ist ein OpenSource Projekt des Departements *Computational and Biological Learning Lab* der Universität New York. *Yann LeCun's* Forschungen basieren auf den Werkzeugen in diesem Projekt. EBlearn++ stellt viele Werkzeuge und Beispiele zum Testen von Faltungsnetzwerken bereit. Eines dieser Werkzeuge ermöglicht es, mit Hilfe von einer Konfigurationsdatei, ein Faltungsnetzwerk zu beschreiben, das dann in den nächsten Schritten trainiert und verwendet werden kann. Für die Testreihe in dieser Arbeit wurden Beispiele des EBLearn++ Projektes kleinen Veränderungen unterzogen und Konfigurationsdateien dieses Werkzeugs angepasst. Eine genauere Erläuterung bezüglich des Testaufbaus und der Netzwerke sind in Sektion 5.2 gegeben.

Die hochvarianten Muster, die in den Testreihen klassifiziert werden, entstammen verschiedenen Bilderdatenbanken. Da es bei einigen Tests notwendig war eigene Testdaten zu generieren oder die bestehenden Daten zu ergänzen, wurde dafür eine Reihe skriptgesteuerter Werkzeugketten entwickelt. Diese haben aus Videodaten Einzelbilder erstellt und, je nach Testreihe, diese in unterschiedliche Formate konvertiert und inhaltliche Veränderungen vorgenommen.
Da es sich bei den benutzten Faltungsnetzwerken um unterschiedliche Programme handelt und diese ihre Eingabedaten in spezifischen Formaten benötigen, musste eine Konvertierung der Testdaten für die jeweilige Plattform stattfinden. Auch hierfür wurde eine Reihe von skriptgesteuerten Werkzeugen entwickelt, die die Testdaten für das jeweilige Faltungsnetzwerk aufbereiten. Einen genaueren Überblick über die Testdaten und deren inhaltliche Veränderungen kann in Sektion 5.1 gefunden werden. Detaillierte Informationen über die Werkzeugketten und deren Benutzung, sind in Anhang C zu finden.

5.1. Testdaten

Zum Testen der Faltungsnetzwerke wurden in dieser Arbeit Gesichter als hochvariantes Muster gewählt. Ähnlich den Stühlen aus Kapitel 1.1 handelt es sich bei Gesichtern um dreidimensionale Muster, die auf einer zweidimensionalen Abbildung erkannt werden müssen.
Für die Tests wurden zwei Bilderdatenbanken verwendet. Die Bilderdatenbanken wurden je nach Test gemischt oder einzeln benutzt. Die erste verwendete Bilderdatenbank nennt sich *muct* [18]. Bei *muct* handelt es sich um 751 verschiedene Gesichter, die jeweils in fünf Posen aufgenommen und diese unterschiedlich beleuchtet wurden.

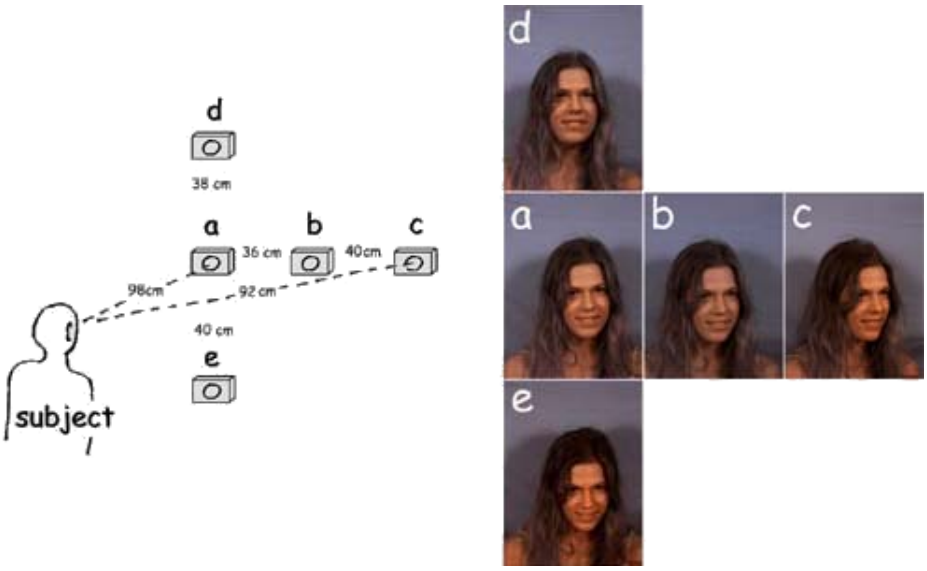


Abbildung 5.1.: Aufbau einer Aufnahme der Bilderdatenbank *muct*[18]. Hier wird eine Bilderserie erstellt, in der fünf Bilder parallel aus unterschiedlichen Winkeln aufgenommen werden

Die 751 unterschiedlichen Personen entstammen vielfältiger ethnischer Herkunft. Die Gesichter der Personen sind nicht immer zentriert im Bild sondern jeweils an unterschiedlichen Positionen. Auch der Aufnahmewinkel unterscheidet sich in jedem der fünf pro Person aufgenommenen Bilder. In Abbildung 5.1 ist der Aufbau einer Aufnahme zu sehen. Es wurden pro Person drei unterschiedliche Posen in unterschiedlichen Lichtverhältnissen aufgenommen, von denen jeweils fünf Bilder in den Winkeln a, b, c, d, e aufgenommen wurden. Alle Bilder haben eine Auflösung von 480x640 Pixel.
Als zweite Bilderdatenbank wurde *Faces in the wild* [30] verwendet. Hier handelt es sich um eine von der Universität von Massachusetts angelegte Bilderdatenbank die 5749 unterschiedliche Gesichter auf 13233 Bildern zeigt. Wobei 1680 dieser Personen mehrmals

abgebildet sind. Jedes Bild ist in der Auflösung 250x250 Pixel gegeben und zeigt ein Gesicht in jeweils unterschiedlicher Umgebung: von Stadt, Natur bis Menschenmengen sind verschiedene Farb- und Formvariantionen gegeben. Im Gegensatz zu den *muct* Bildern sind

Abbildung 5.2.: Vier Beispiele aus der Bilderdatenbank *Faces in the wild* [30]

die Gesichter der Personen in der Mitte der Bilder zentriert. Ein Ausschnitt dieser Bilder ist in Abbildung 5.2 zu sehen, die vier solcher Beispiele zeigt.
Für viele der Tests wurde eine zweite Klassifizierungsgruppe für die Hintergründe trainiert. Alle Bilder, die kein Gesicht enthalten, sollen als Hintergrund klassifiziert werden. In Abbildung 5.3 sind einige Beispiele der trainierten Hintergründe gezeigt. Zum Generieren der Trainingsdaten wurde ein Programm mit einem auf Gesichter spezialisierten Faltungsnetzwerk des EBlearn++ Projektes benutzt. Das Programm hat in Bildern Gesichter erkannt und diese ausgeschnitten. Die Treffer, die als Gesicht klassifiziert wurden aber keines enthielten, wurden als Hintergrund-Trainingsdaten verwendet. Auf diese Weise wurden 30.000 Hintergrundbilder aus der *Faces in the wild* Bilderdatenbank generiert.

Abbildung 5.3.: Einige Beispiele der Hintergrund-Klassifizierungsgruppe. Als Trainingsbilder wurden Objektausschnitte, verschieden farbige Hintergründe oder Kanten benutzt

5.1.1. Erstellung von Trainings- und Testdaten

Für die Durchführung von einigen Testreihen mussten die Testdaten mit zusätzlichen Bildern erweitert werden. Die in 5.1 erwähnten Bilderdatenbanken waren für manche Tests unvollständig. Für Tests, in denen bestimmte Personen von anderen wiedererkannt werden sollten, wurden mehr Trainingsdaten von einer Person benötigt. Ein Faltungsnetzwerk benötigt zum Lernen von Klassifikationsmerkmalen wie Gesichtern, je nach Test ca. 20.000

- 30.000 Datensätze [20]. Die Anzahl der Trainingsdaten hängt von der Komplexität der zu klassifizierenden Objekte ab. Bei starren Objekten mit wenig Varianz, die immer aus einer Pose erkannt werden sollen werden um gute Ergebnisse zu erzielen, nur ca. 4000 Bilder [22] benötigt.
Die verwendeten Bilderdatenbanken reichen von der Masse der Datensätze zum Training für einige Testfälle zwar aus, jedoch fehlen für das Training bestimmter Merkmale einige tausend Bilder. Deshalb musste eine Reihe von Werkzeugen kombiniert werden, um Testdaten zu generieren. Dies geschieht, indem ein Video von einer Person oder Objekt in Bewegung oder in verschiedenen Posen aufgenommen wird. Der Videostream wird dann in seine Einzelbilder zerlegt und auf das Format der jeweiligen Bilderdatenbank konvertiert. Je nach Test kann nun eine beliebige Anzahl der neuen Bilder in die Datenbestände der Bilderdatenbanken eingestreut werden. Eine genauere Erläuterung über die Anwendung der Werkzeugkette ist in Anhang C zu finden.
Alle Trainings- und Testdaten aus den kombinierten Bilderdatenbanken und den eigenen Trainingsdaten mussten in bestimmte Formate konvertiert werden. Diese Formate wurden von den beiden - in Sektion 5.2 beschriebenen - Faltungsnetzwerken vorgegeben. Bei der Implementierung des LeNet-5 Faltungsnetzwerkes in Matlab musste das Format der *MNIST-Datenbank für handgeschriebene Ziffern* für die Trainings- und Testdaten verwendet werden. Für das EBlearn++ basierte Faltungsnetzwerk wurde ein Werkzeug benutzt, das ebenfalls aus diesem Projekt stammt. Dieses Werkzeug hat rohe Bilddaten im *jpg*, *png* oder *bmp* Format aus einer Verzeichnisstruktur direkt in das benötigte Trainings- und Testdatenformat konvertiert.

MNIST

Die *MNIST-Datenbank für handgeschriebene Ziffern* enthält über 50.000 Trainings- und 10.000 Testdaten. *MNIST* wurde von *LeCun und Cortes* [15] entwickelt, um eine Datenbank für hochvariante, handgeschriebene Ziffern für Klassifizierungen und Tests statistischer Methoden zu verwenden. Ziel war es, eine aus der Realität entnommene dichte Menge von verschiedenen Handschriften in der Datenbank unterzubringen. Die über 60.000 Trainings- und Testdaten stammen von 250 verschiedenen Personen. Alle Ziffern in *MNIST* wurden in Größe normalisiert und zentriert. Die Auflösung aller Bilder ist gleich. Entwicklungsziel des Formates liegt in der Aufwandsminimierung des Formatierens und Auslesens der Datensätze. In der von *LeCun* vorgestellten Handschrifterkennung mit dem Faltungsnetzwerk LeNet-5 [13] wurde *MNIST* für Trainings- und Testdaten verwendet.
Ein Trainings- und Testdatensatz besteht jeweils aus zwei Dateien. Eine Datei enthält die Bilddaten, die andere die Bezeichnung (sog. *Label*), die die Bilder darstellen soll. Alle Dateien sind im Big-Endian-Format kodiert. In Tabelle 5.1 ist der Aufbau einer Labeldatei für einen Trainings- oder Testdatensatz dargestellt. Die Werte der *Label* können eine gewünschte Zahl

Tabelle 5.1.: Byteweiser Aufbau einer MNIST-Labeldatei

Bytenummer	Datentyp	Wert	Beschreibung
0000	4 Byte-Ganzzahl	2049	festgelegte Erkennungszahl
0004	4 Byte-Ganzzahl	0 - ca. 4,3 Milliarden	Anzahl der gesammten *Label* in der Datei
0008	vorzeichenloses Byte	0 - 255	gewünschtes *Label* des ersten Bildes
0009	vorzeichenloses Byte	0 - 255	gewünschtes *Label* des zweiten Bildes
...	...	...	...

Tabelle 5.2.: Byteweiser Aufbau einer MNIST-Bilderdatei

Bytenummer	Datentyp	Wert	Beschreibung
0000	4 Byte-Ganzzahl	2051	festgelegte Erkennungszahl
0004	4 Byte-Ganzzahl	0 - ca. 4,3 Milliarden	Anzahl der gesammten Bilder in der Datei
0008	4 Byte-Ganzzahl	0 - ca. 4,3 Milliarden	Breite der Bilder in Pixel
0012	4 Byte-Ganzzahl	0 - ca. 4,3 Milliarden	Höhe der Bilder in Pixel
0016	vorzeichenloses Byte	0 - 255	erster Pixel in ersten Reihe des ersten Bildes
0017	vorzeichenloses Byte	0 - 255	zweiter Pixel in ersten Reihe des ersten Bildes
...	...	...	...

im Bereich 0 bis 255 sein. Je nach Test, können die Werte anders belegt werden. In Tabelle 5.2 ist der Aufbau einer Bilddatei für einen Trainings- oder Testdatensatz dargestellt. Die Bilddaten sind in Graustufen angegeben. Das bedeutet, dass die Pixel einen Wertebereich von 0 bis 255 haben, wobei 0 für Weiß und 255 für Schwarz steht. Die Bilder sind nacheinander kodiert, ohne Platzhalter oder Erkennungszahlen zwischen den Pixelreihen.

5.1.2. Verrauschung von Bilddaten

Aus einer falschen Klassifizierung eines Musters durch ein Faltungsnetzwerk, folgt eine falsche Interpretation. Wird zum Beispiel ein Muster als Gesicht klassifiziert, obwohl es kein

Gesicht darstellt, spricht man von einem sogenannten *false positive*. Beim Training von Neuronalen Netzen sollen die *false positives* möglichst minimiert werden. Ein in dieser Arbeit zu testender Ansatz zum Minimieren von *false positives*, ist die gezielte Verrauschung der Trainingsdaten.

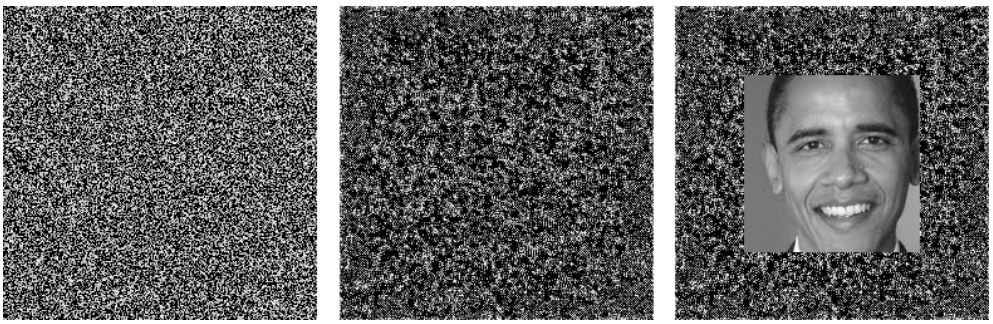


Abbildung 5.4.: Die einzelnen Verarbeitungsschritte der Werkzeugkette. Eine zufällige Verrauschung (links), die Verrauschung tiefengefiltert (mitte) und das eingefügte Gesicht in die Verrauschung (rechts)

Die Umgebung eines Gesichtes in einem Bild wird verrauscht. Somit wird der Fokus der zu trainierenden Charakteristika im Bild direkt auf das zu klassifizierende Gesicht konzentriert. Um nur die Umgebung und nicht das zu klassifizierende Gesicht zu Verrauschen, wurde eine entsprechende Werkzeugkette entworfen. Zur Erkennung der Gesichter in den verschiedenen Umgebungen in den Bildern wurde ein Programm verwendet, dass aus dem auf Gesichter spezialisierten Faltungsnetzwerk des EBlearn++ Projektes stammt. Das Programm schneidet nach Erkennung die Gesichter aus. In Abbildung 5.4 sind die drei Stufen der Verarbeitung zu sehen:

1. Ein zufällig verrauschtes Graustufenbild
2. Das aus dem vorherigen Schritt generierte, tiefengefilterte Graustufenbild
3. Das in die tiefengefilterte, verrauschte Szene kopierte erkannte Gesicht

In den Tests, genauer erläutert in Sektion 5.3, wurden nur die Trainingsdaten verrauscht. Die Faltungsnetzwerke wurden anschließend mit unverrauschten Bildern getestet, die Gesichter enthalten.

5.2. Testaufbau

Die in den Tests benutzten Faltungsnetzwerke bauen auf zwei unterschiedlichen Arbeiten auf. Bei dem ersten Faltungsnetzwerk handelt es sich um die Umsetzung des von *LeCun* vorgestellten LeNet-5 [14] Faltungsnetzwerks von *Chumerin* [3]. Die LeNet-5 Nachbildung ist ein Matlabprogramm.
Das zweite verwendete Faltungsnetzwerk ist ein, auf Beispielen des Projektes EBlearn++ [23] basiertes, C Programm. Zum Trainieren und Testen des Programms musste eine zum Teil selbst entwickelte Werkzeugkette verwendet werden. Es wurden die vorgenannten Arten von Faltungsnetzwerken für die Tests gewählt, damit Unterschiede zum Aufbau verglichen und somit mögliche Fehler in den Implementierungen ausgeschlossen werden können.

5.2.1. Matlab LeNet-5

Die Implementierung des LeNet-5 Faltungsnetzwerkes in Matlab hält sich sehr stark an das von *Yann LeCun* beschriebe [13] Vorbild. Die Implementierung in Matlab wurde von *Nikolay Chumerin* geschrieben. In Abbildung 5.5 ist der Aufbau des LeNet-5 Faltungsnetzwerkes gegeben.

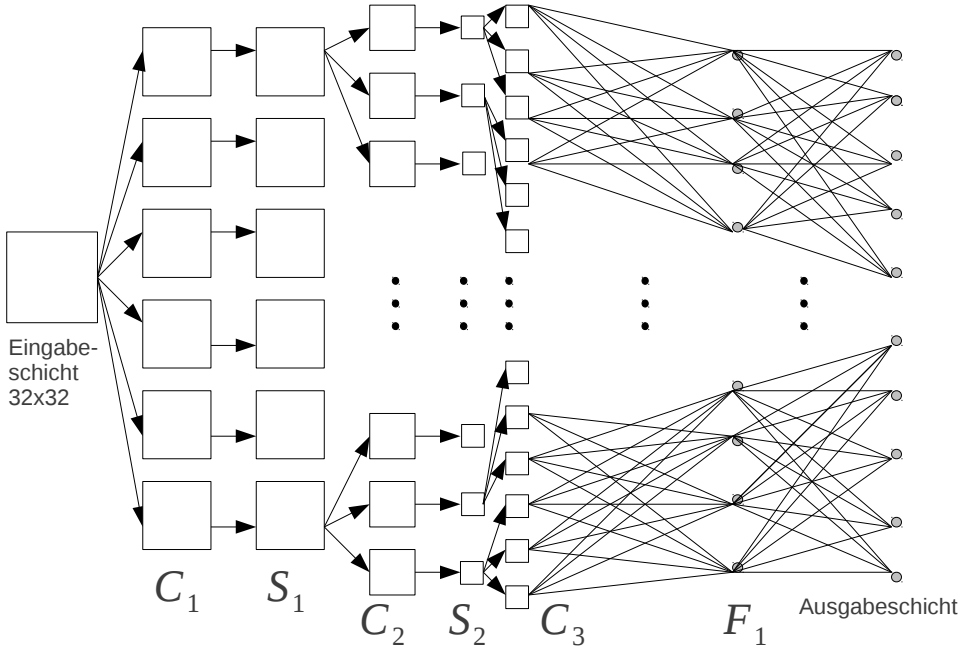


Abbildung 5.5.: Aufbau der Schichten des LeNet-5 Faltungsnetzwerkes

Im Vergleich zu dem in Abbildung 4.4 gezeigten Aufbau ist zu sehen, dass LeNet-5 zwei vollvernetzte Schichten von Perzeption Neuronen besitzt und eine zusätzliche Convolution-Schicht C_3. Die im Folgenden angegebenen Größen der *feature maps*, Schichten und Neuronen variieren je nach Faltungsnetzwerk. Die hier präsentierten Zahlen stammen aus der Implementierung des LeNet-5 Faltungsnetzwerks [13] von *Nikolay Chumerin* in Matlab.
Die Größe der Eingangsdaten betägt 32x32 Pixel. Die Convolution-Schicht C_1 umfasst sechs *feature maps* in der Größe 28x28 Pixel. Die darauf folgende Subsampling-Schicht S_1 reduziert die Daten um Faktor zwei zu 14x14 Pixel. Die nächste Convolution-Schicht C_2 verteilt die Daten mit 26 *feature maps* mit 5x5 auf jeweils 26 *feature maps* der Subsampling-Schicht S_2. Diese reduziert die Daten wieder auf Faktor zwei. Schicht F_1 bezeichnet eine vollvernetzte Schicht von Perzeptron-Neuronen, die sich zwischen Ausgabeschicht und C_3 befindet. Die letzte Convolution-Schicht C_3 gibt die Daten auf Einhundert vollvernetzte Perzeptron-Neuronen in Schicht F_1 weiter. Diese sind mit allen *feature maps* der Schicht C_3 und der Ausgangsschicht verbunden. Die Ausgangsschicht hat im Vorbild von *Yann LeCun* genau zehn Neuronen. Das Matlabprogramm und LeNet-5 sind auf das Trainieren und Erkennen von Ziffern ausgerichtet. Das bedeutet, LeNet-5 hat genau zehn Neuronen in der Ausgabeschicht. Für die jeweiligen Testreihen aus 5.3 wurde das Netz für die Tests angepasst. Die Anzahl der Neuronen der Ausgabeschicht wurde reduziert.
Das Einlesen der Bilddaten musste von Big-Endian auf Little-Endian umgestellt werden. Laut dem Format der MNIST-Datenbank sind alle Bilder der Trainings- und Testdaten in Big-Endian, wie in Sektion 5.1.1 weiter erläutert, kodiert. Dies wurde bei der Generierung der MNIST-Dateien und beim Einlesen geändert, da alle Tests auf Intel-basierten, Little-Endian-fähigen Prozessoren ausgeführt wurden.

5.2.2. EBlearn++ Convolutional Neural Network

Das Projekt EBlearn++ enthält neben einer Programmierschnittstelle einige Beispielprogramme, die Faltungsnetzwerke nutzen. Zwei dieser Beispiele mussten nur geringfügig angepasst werden, um gezielt Faltungsnetzwerke zu trainieren und zu testen. Die Faltungsnetzwerke beider Programme sind nicht statisch im Programmcode definiert, wie zum Beispiel in der in 5.2.1 beschriebenen Matlabumsetzung, sondern werden in einer Konfigurationsdatei definiert. Diese Konfigurationsdatei wird jeweils beim Start des Programms eingelesen. Zu jeder Konfigurationsdatei gehört eine Gewichtsdatei, die die Einstellungen der Gewichte des Faltungsnetzwerkes enthält. Für die Durchführung des Trainings musste eine Konfigurationsdatei mit dem Aufbau des Faltungsnetzwerkes und die Trainingsdaten an das Programm gegeben werden. Das Resultat des Trainings ist eine Gewichtsdatei. Zum Durchführen eines Tests muss die Konfigurationsdatei des Faltungsnetzwerkes mit der Gewichtsdatei und den Testbildern in das Programm gegeben werden.
Für alle Tests wurde folgende Konfiguration des Faltungsnetzwerks gewählt:

Die Eingangsschicht hat eine Größe von 32x32 Pixel. Ihr folgen zwei Paare Subsampling- und Convolution-Schichten. Vor der Ausgangsschicht ist eine versteckte Schicht von 100 vollvermaschter Perzeptron-Neuronen. Diese sind mit den Ausgangsneuronen verbunden. Die Anzahl der Ausgangsneuronen hängt von dem jeweiligen Test ab. Jedes mögliche Testergebnis bekommt ein Ausgangsneuron zugewiesen. Der *Threshold* des Faltungsnetzwerkes liegt bei 0.9.

5.3. Ergebnisse

Mit Hilfe der in Sektion 5.2 erläuterten Faltungsnetzwerke wurden zwei Testreihen durchgeführt. In der ersten Testreihe wurden die Faltungsnetzwerke auf die Erkennung von Gesichtern trainiert. Die Faltungsnetzwerke mussten unterscheiden, ob in einem Bild ein Gesicht abgebildet ist oder nicht. In der zweiten Testreihe wurde das Training auf die Unterscheidung von einem bestimmten Gesicht zu anderen Gesichtern ausgerichtet.
Jeder Test in beiden Testreihen hat eine unterschiedliche Anzahl an Trainingsdaten und Trainingsepochen. Eine Trainingsepoche ist die Anzahl der Trainingsdurchgänge über alle Trainingsdaten. Hat ein Test zwei Trainingsepochen, so werden alle Trainingsdaten zweimal durchlaufen. Je nach Test wurden, entweder die in Sektion 5.1 erwähnten Bilderdatenbank *Faces in the wild* oder *muct* als Trainingsdaten benutzt. Je nach verwendeten Trainingsdaten, wurden 500 der Bilder als Testdaten reserviert und nicht mit trainiert. Die Erkennungsrate des Trainings berechnet sich aus der Erkennung des Testdatensatzes.

5.3.1. Allgemeine Gesichtserkennung

Die Trainings und Tests dieser Testreihe hatten das Ziel, Bilder, die ein Gesicht enthalten, von Bildern ohne Gesicht zu unterscheiden. Es gab zwei Klassifizierungsklassen:

1. Bild enthält Gesicht
2. Bild enthält kein Gesicht

Die Ergebnisse dieser Tests sind in Tabelle 5.3 gezeigt. Alle in den Tests benutzten Faltungsnetzwerke wurden so konfiguriert, dass sie einen *Threshold* von 0.9 aufwiesen, um eine Klassifizierung zu bestätigen.

5.3.2. Erkennung von bestimmten Gesichtern

Die Trainings und Tests dieser Testreihe hatten das Ziel, ein bestimmtes Gesicht von anderen Gesichtern zu unterscheiden. Es gab zwei Klassifizierungsklassen:

1. Bild enthält Gesicht der bestimmten Person
2. Bild enthält Gesicht einer anderen Person

Tabelle 5.3.: Ergebnisse der Tests zur allgemeinen Gesichtserkennung

Bilderdatenbank	Anzahl: Bild enthält Gesicht	Anzahl: Bild enthält kein Gesicht	Epochen	Erkennungsrate in Prozent
Faces in the wild	12.000	17.000	5	96,40
Faces in the wild	12.000	17.000	3	94,20
Faces in the wild	12.000	17.000	2	87,40
Faces in the wild	12.000	8.000	2	82,80
Faces in the wild	10.000	2.000	3	62,80
Faces in the wild (Verrauscht)	12.000	17.000	2	81,20
Faces in the wild (Verrauscht)	12.000	8.000	2	60,20
muct	4.000	6.000	4	23,60
muct	4.000	6.000	2	45,80
muct	4.000	2.000	2	24,40

Tabelle 5.4.: Ergebnisse der Tests zur Erkennung von bestimmten Gesichtern

Bilderdatenbank	Anzahl: Bild enthält bestimmtes Gesicht	Anzahl: Bild enthält anderes Gesicht	Epochen	Erkennungsrate in Prozent
muct	9.000	6.000	2	48,20
Faces in the wild	9.000	13.000	2	53,80

Die Trainingsdaten einer bestimmten Person wurden mit den in Sektion 5.1.1 angesprochenen Werkzeugen generiert. In den hier vorgestellten Ergebnissen handelte es sich um 4.000 Bilder der bestimmten Person. Die Trainingsdaten der anderen Gesichter stammen aus den Bilderdatenbanken *Faces in the wild* und *muct*. Die Ergebnisse dieser Tests sind in Tabelle 5.4 gezeigt. Alle in den Tests benutzten Faltungsnetzwerke wurden mit einem *Threshold* von 0.9 konfiguriert, um eine Klassifizierung zu bestätigen.

6. Diskussion

Primäres Ziel dieser Arbeit war es herauszufinden, ob eine Gesichtserkennung mit Hilfe von Faltungsnetzwerken realisiert werden kann. Wie *Osadchy, LeCun und Miller* [20] schon bewiesen haben, ist die allgemeine Gesichtserkennung mit Faltungsnetzwerken möglich. Die in Sektion 5.3 beschriebenen Ergebnisse zeigen ähnlich gute Resultate.
Das beste Trainingsergebnis ergab eine Erkennungsrate von 96% (siehe Tabelle 5.3). Bei den Trainingssessions konnten pro Test mehrere Variablen angepasst werden:

1. Anzahl der Trainingsepochen
2. Anzahl der Trainingsbilder in jeder Klassifizierungsgruppe
3. Verhältnis der Trainingsbilder der Klassifizierungsgruppen

Die besten Ergebnisse wurden erzielt, wenn die Anzahl der Trainingsbilder der Klassifizierungsgruppe ohne Gesichter im Verhältnis zu der Gruppe mit Gesichtern 3:2 steht. Bei Tests mit weniger Trainingsbildern in der Klassifizierungsgruppe öhne Gesichter", wurden stets schlechtere Erkennungsraten erzielt.
Die Anzahl der Epochen eines Trainings kann eine Erkennungsrate bis zu einem bestimmten Punkt steigern. Ab diesem Punkt wird das Netzwerk übertrainiert und kann neben einer marginalen Verbesserung sogar eine Verschlechterung nach sich ziehen. Obwohl das Verhältnis der Trainingsdaten "mit Gesicht"den Trainingsdaten öhne Gesicht"bei den Tests mit den Bilder der Datenbank *muct* auch 2:3 entsprach, sind die Erkennungsraten deutlich schlechter als die der anderen Tests. Die Anzahl der Trainingsbilder hat einen signifikanten Einfluss auf die Erkennungsraten.
Das in Sektion 5.1.2 erläuterte Verrauschen der Bilderdatenbank *Faces in the wild* hat gute Klassifizierungsergebnisse erzielt. Im Vergleich zu den nicht verrauschten Tests mit den Bildern *Faces in the wild*, (gleiche Konfiguration), schneiden die verrauschten Tests aber schlechter ab. Das Verrauschen der Hintergründe vermindert also nicht die Anzahl der *falsch positiven* Klassifizierungen. Die Verrauschung kann von Vorteil sein, wenn man Trainingsdaten in nicht varianten Umgebungen hat und diese varianter machen möchte. Trainingsdaten, die in gut beleuchteten, hellen Umgebungen aufgenommen wurden, können so für ein Training verwendet werden, um die Objekte in varianten Umgebungen zu erkennen.
Zur gezielten Gesichtserkennung wurden zwei Tests durchgeführt. Hier wurden einerseits die Bilderdaten *Faces in the wild* und *muct* jeweils als Klassifizierungsgruppe für Gesichter und

andererseits ein bestimmtes Gesicht mit 9.000 Trainingsdaten als weitere Klassifizierungsgruppe verwendet. Bei beiden Tests lag die Erkennungsrate bei ca. 50%. Dies beweist, dass Faltungsnetzwerke mit einer Eingangsgröße von 32x32 Pixel nicht ausreichen, um bestimmte Gesichter von anderen zu unterscheiden. In weiterführenden Arbeiten könnte getestet werden, ob größere Eingangsvektoren der Faltungsnetzwerke dieses Problem beheben.
Bei dem Training und der Verwendung von Faltungsnetzwerken spielen diese oben genannten drei Faktoren (Anzahl Epochen, Anzahl Bilder und Verhältnis der Bilder) eine wichtige Rolle. Um die geeigneten Einstellungen für das Training eines Faltungsnetzwerkes herauszufinden, können keine Algorithmen benutzt werden. Vielmehr muss iterativ durch Veränderung der drei Faktoren vorgegeangen werden, um immer bessere Erkennungsraten zu erzielen. Dieses Vorgehen muss für jeden Klassifikator neu gestartet und durchgeführt werden, wobei auf Erfahrungen dieser Arbeit zurückgegriffen werden kann.

Literaturverzeichnis

[1] 4.bp.blogspot.com. Url: *http://4.bp.blogspot.com/_6sdWUTBboAQ/S8L4GReuJTI/AAAAAAAAADc/zEKzFkgv6AU/s1600/image-upload-56-797955.jpg*, September 2011.

[2] bild.de. Url: *http://bilder.bild.de/fotos-skaliert/13-der-winter-ist-da-wie -hat-der-schnee-sie-erwischt-mbqf-14841460/2,h=343.bild.jpg*, September 2011.

[3] Nikolay Chumerin. Url: *http://www.mathworks.com/matlabcentral/fileexchange/25247*, September 2009.

[4] Computational and Biological Learning Lab New York University. Url: *http://eblearn.svn.sourceforge.net/viewvc/eblearn/trunk/*, May 2011.

[5] Josef Dudel, Randolf Menzel, and Robert F. Schmidt. *Neurowissenschaft: Vom Molekül zur Kognition*. Springer-Verlag, 2001.

[6] Stefan Duffner. Face image analysis with convolutional neural networks. Dissertation, Fakultät für Angewandte Wissenschaften an der Albert-Ludwigs-Universität Freiburg, 2007.

[7] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36[4]:193–202, April 1980.

[8] Loungeclub Gmbh. Url: *http://www.loungeclub.ch/shop/bilder/137_stuhl2_1.jpg http://www.loungeclub.ch/shop/bilder/3536_Schwingstuhl_Croco_Schwarz_01.jpg*, September 2011.

[9] h3.abload.de. Url: *http://h3.abload.de/img/img00048-20100516-20072esr.jpg*, September 2011.

[10] http://kaicent.wordpress.com. Url: *http://kaicent.files.wordpress.com/2011/04/kinected3.png*, September 2011.

[11] David Jacobs. Correlation and convolution class notes for cmsc 426. *University of Maryland*, 2005.

[12] K. Lai and D. Fox. Object recognition in 3d point clouds using web data and domain adaptation. *Department of Computer Science and Engineering University of Washington, Seattle, WA*, April 2010.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[15] Yann LeCun and Corinna Cortes. Url: *http://yann.lecun.com/exdb/mnist/*, September 2011.

[16] Yann LeCun, Fu-Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press, 2004.

[17] David Leonard. Url: *http://davidleonardtv.wordpress.com/2011/02/02/turn-kinect-into-3d-scanner-explained-full-tutorial-with-code*, 2011.

[18] S. Milborrow, J. Morkel, and F. Nicolls. The muct landmarked face database url: *http://www.milbo.org/muct*, 2010.

[19] Alfred Nischwitz, Max W. Fischer, and Peter Haberäcker. *Computergrafik und Bildverarbeitung*. Vieweg+Teubner, 2007.

[20] M. Osadchy, Y. LeCun, and M. Miller. Synergistic face detection and pose estimation with energy-based models. *Journal of Machine Learning Research*, 8:1197–1215, May 2007.

[21] Michael W. Palmer. Url: *http://ordination.okstate.edu/PCA.htm*, September 2011.

[22] Peemen, Maurice, Mesman, Bart, Corporaal, and Henk. Speed sign detection and recognition by convolutional neural networks. may 2011.

[23] EBlearn++ projekt. Url: *http://eblearn.sourceforge.net/tutorials/libeblearn/index.shtml/*, May 2011.

[24] Gerhard Roth and Wolfgang Prinz. *Kopf-Arbeit: Gehirnfunktionen und kognitive Leistungen*. Spektrum Akademischer Verlag GmbH, 1996.

[25] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. *20th International Conference on Artificial Neural Networks (ICANN), Thessaloniki, Greece*, sep 2010.

[26] H. Schulz and S. Behnke. Object-class segmentation using deep convolutional neural networks. *DAGM Workshop on New Challenges in Neural Computation*, August 2011.

[27] P. Y. Simard, Y. LeCun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propagation. *International Journal of Imaging Systems and Technology*, 11(3), 2001.

[28] Lindsay I Smith. A tutorial on principal components analysis. feb 2002.

[29] Richard F. Thompson. *Das Gehirn*. Spektrum Akademischer Verlag GmbH, 2001.

[30] Gary Huang University of Massachusetts. Url: *http://vis-www.cs.umass.edu/lfw*, May 2011.

[31] Hayden Huang Wei-Chung Allen Lee, Guoping Feng, Joshua R. Sanes, Emery N. Brown, Peter T. So, and Elly Nedivi. Url: *http://upload.wikimedia.org/wikipedia/commons/0/0d/GFPneuron.png*, May 2011.

[32] Wikipedia. Url: *http://de.wikipedia.org/wiki/Hauptkomponentenanalyse*, May 2011.

[33] Wikipedia. Url: *http://de.wikipedia.org/wiki/K%C3%BCnstliches_neuronales_Netz*, May 2011.

[34] Wikipedia. Url: *http://de.wikipedia.org/wiki/Korrelation*, May 2011.

[35] Wikipedia. Url: *http://de.wikipedia.org/wiki/Mustererkennung*, May 2011.

A. Trainieren von Convolutional Neural Networks

Um ein Convolutional Neural Network zu trainieren werden Trainings- und Testdaten benötigt. Um diese komfortabel zu konvertieren und generieren, siehe Anhang C.

A.1. Training für *persnet*

Das Training eines Convolutional Neural Network resultiert in eine Gewichtsdatei. Diese beschreibt alle Gewichte und Bias eines trainierten Netzes. Für jede Trainingsiteration wird eine solche Gewichtsdatei beim Trainingsvorgang angelegt.
Ähnlich dem Starten der *persnet* Anwendung (siehe Anhang B), benötigt das Trainingsprogramm *train_persnet* eine Konfigurationsdatei. Diese kann identisch mit der von *persnet* verwendeten Datei sein. Nicht benötigte Einträge werden ignoriert. Die Anwendung *train_persnet* ist eine Abänderung des *train* programms[4].

Die wichtigsten Einträge der Konfigurationsdatei training_data/sample.conf sind:

```
root=training_data/sample
```

Setzt das Verzeichnis, in welchem die Training- und Testdaten liegen. Siehe C für die Erstellung der Datensets.

```
net_type=cscsc
```

Konfiguriert den Aufbau des Convolutional Neural Networks. Jedem **c** muss ein **s** folgen, wobei die Buchstaben für die *Convolution*- und *Sampling*-Schicht stehen (siehe 4.1.2).

```
net_ih=32
net_iw=32
```

Legt die Größe der Eingabedaten des Netzes fest.

```
net_c1h=5
net_c1w=5
net_s1h=2
net_s1w=2
net_c2h=5
net_c2w=5
net_s2h=2
net_s2w=2
net_full=100
```

Die Höhe und Breite der jeweiligen *Convolution*- und *Sampling*-Schichten werden hier in Pixeln definiert. Die Anzahl der benötigten Einträge hängt hier von dem **net_type** Parameter ab. Der **net_full** Parameter legt die Anzahl der Perzeptron-Neuronen fest, die die letzte Schicht vor der Ausgabeschicht bilden. Diese werden mit der letzten *Sampling*-Schicht vollvermascht.

```
iterations=50
```

Legt die Anzahl der Iterationen über das gesamte Trainingset fest.

Das Trainingsprogramm *train_persnet* wird dann mit der Konfiguration wie folgt gestartet:

```
src/train_persnet training_data/sample.conf
```

A.2. Training für Matlab CNN

Nachdem die Trainings- und Testdaten in den Matlab-Workspace als **training.byte** und **test.byte** abgelegt wurden, kann in der Matlab-Konsole

```
cnet_train
```

eingegeben werden. Dies startet das GUI und den Trainingsvorgang. Sollen kleine Veränderungen am Trainingsvorgang, ähnlich dem A.1 gemacht werden, so muss die Datei **cnet_train.m** entsprechend angepasst werden. Ist das Training beendet, muss die Variable *sinet* als sinet.m im Matlab-Workspace gespeichert werden. Nun kann das GUI in der Matlab-Konsole mit

```
cnet_tool
```

aufgerufen werden. Hier kann mit dem Suchen-Button ein Datensatz ausgewählt und vom trainierten Netz bearbeitet werden. Mit der Autoerkennung und dem Pfeil-Button kann der Datensatz durchsucht werden.

B. Starten der Persnet Anwendung

Persnet ist eine Anwendung um trainierte Convolutional Neural Networks zu testen. Die Anwendung ist eine Abwandlung von dem Beispielobjekterkennungsprogramm *detect* [4]. Die Hauptaufgabe des Programms besteht in der Erkennung von Personen in Bildern.
Um die Anwendung zu starten, muss zuerst eine Konfigurationsdatei vorliegen. In dieser Konfiguration ist der Aufbau des Convolutional Neural Networks, mit der Anzahl der Schichten und deren Größe angegeben. Außerdem wird dort die Datei der Gewichtsmatizen und zu erkennenden Label angegeben. Wichtig zu beachten: die Größe und Aufbau des Convolutional Neural Networks muss identisch sein mit dem des trainierten Netzwerkes (siehe A für weitere Informationen, wie man ein Netz trainiert).

Die wichtigsten Einträge der Konfigurationsdatei sample/sample.conf sind:

```
root=sample
```

hier wird das Unterverzeichnis angegeben, in dem alle anderen zugehörigen Dateien zu der Konfiguration liegen.

```
weights=${root}/gewichtsdatei.mat
classes=${root}/label_classes.mat
```

Mit den beiden Einträgen werden die Dateien für die Gewichte und die zu erkennenden Objekte angegeben. Die Datei für die Gewichte wird beim Training des Netzwerks erstellt, die Labeldatei, bei der Erstellung des Trainingsets.

```
input_height=32
input_width=32
```

Diese Parameter geben die Größe der Eingabedaten an. Die Eingabebilder werden auf diese Größe in Pixel transformiert und anschließend an das Convolutional Neural Network übergeben. Wird als Höhe und Breite jedoch jeweils **-1** angegeben, werden die Eingabebilder nicht transformiert, sondern nach *Region of Interest* untersucht. Diese werden erst transformiert und dann an das Convolutional Neural Network übergeben.

Um nun ein trainiertes Netz auf Eingabebildern laufen zu lassen, muss zunächst ein Verzeichnis mit den Bildern angegeben werden. Ein Beispiel wäre das Verzeichnis **input**, dann würde ein Aufruf von *persnet* wie folgt aussehen:

```
src/persnet sample/sample.conf input
```

C. Anwendung der Toolchain für Bild- und Trainingsdaten

Die in dieser Arbeit verwendeten Trainingsdaten stammen aus den Bilderdatenbanken *muct* [18] und *Faces in the wild* [30]. Diese mussten für die Verwendung mit den Faltungsnetzwerken, beschrieben in Sektion 5.2, in verschiedene Formate konvertiert werden. Für die Tests der gezielten Gesichtserkennung wurden tausende Bilder eines bestimmten Gesichts erstellt. Für diese Probleme wurde eine Toolchain entwickelt, auf die im Folgenden näher eingegangen wird.

C.1. Generierung von Bildern aus Videostreams

Um in kurzer Zeit eine Bilderdatenbank von Gesichtern einer bestimmten Person zu erstellen wurde ein Skript entwickelt, das eine Sammlung von Werkzeugen benutzt, um aus Videos Einzelbilder zu erstellen. So muss zum Beispiel anstatt hunderte einzelner digitaler Fotografien nur ein ca. zwei Minuten langes Video erstellt werden. Aus dem Video werden dann in einem bestimmten Intervall die Bilder generiert. So entstehen aus mehreren Videos schnell tausende variante Aufnahmen eines Gesichtes. Die Werkzeugkette basiert auf den Programmen *Gimp* und *ffmpeg*. Die Anwendung des Skriptes ist dann wie folgt:

```
sh genPicFromStream.sh input_dir output_dir 20
```

Das Skript nimmt drei Parameter:

1. Eingabeverzeichnis - dieses Verzeichnis enthält eine Anzahl .mpeg Videodateien, die in einem gegebenem Intervall in Millisekunden in Bildern zerlegt werden
2. Ausgabeverzeichnis - Gibt das Verzeichnis an, in dem die generierten Bilder hinterlegt werden
3. Intervall - gibt das Intervall der Streamabtastung in Millisekunden an

In diesem Beispiel werden alle .mpeg Videos im Verzeichnis *input_dir* in einem Intervall von 20 Millisekunden abgetastet und das Verzeichnis *output_dir* gespeichert. Das Skript geht davon aus, dass die Videos in dem Seitenverhältnis 4:3 hinterlegt sind. Mit dem Programm *Gimp* werden die gespeicherten Bilder so gedreht, dass sie dem Format der Bilderdatenbank *muct* entsprechen.

C.2. Konvertierung von Bildern in das *mnist* Format

Die Bilder der Datenbanken *muct* und *Faces in the wild* sind in einer Verzeichnisstruktur abgelegt. Um diese Bilder in das in Sektion 5.1.1 beschriebene Format *mnist* zu bringen musste ein Programm geschrieben werden das diese Konvertierung vornimmt. Das geschriebene Programm, genannt *block_converter*, verwendet eine variable Anzahl von .bmp Dateien und komprimiert diese in eine *mnist*-Label- und eine *mnist*-Bilderdatei. Im Folgenden ist ein Beispiel für einen solchen Aufruf gegeben:

```
./block_converter ExampleSet example1.bmp example2.bmp example3.bmp
```

Der erste Parameter gibt den Prefix der generierten *mnist*-Dateien an. Alle folgenden .bmp Dateien werden als Eingabe für die Komprimierung verwendet. Das oben gezeigte Beispiel generiert die Dateien *ExampleSet-label.idx2-ubyte* und *ExampleSet-image.idx2-ubyte*. Um die Bilderdaten aus verschiedenen Formaten in das 8-Bit .bmp Format zu bringen kann das Unixprogramm *convert* benutzt werden.

C.3. Hintergrundverrauschung von Gesichtsbildern

Um den in Sektion 5.1.2 beschriebenen Verrauschungseffekt zu erzielen, musste eine Werkzeugkette geschrieben werden, die zuerst alle Gesichter einer Bilderdatenbank ausschneidet und dann deren Hintergrund verrauscht. Um die Gesichter Bild für Bild auszuschneiden, wurde das in Anhang A und B erwähnte Programm *persnet* mit einer, auf Gesichter trainierten Konfiguration von dem *Departements Computational and Biological Learning Lab* der Universität New York verwendet. Für die Verrauschung des Hintergrunds musste ein Programm, genannt *noise_maker*, geschrieben werden. Das Programm nimmt ein .png Bild und vergibt einen Namen für das resultierende, verrauschte Bild. Im Folgenden ist ein Beispiel für einen solchen Aufruf gegeben:

```
./noise_maker face.png noised_face.bmp
```

Der erste Parameter ist der Dateiname einer .png Datei, die ein Gesicht enthält. Der zweite Parameter gibt den Dateinamen des erstellten Bildes an.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 16. Januar 2012	
Ort, Datum	Unterschrift