SPERRVERMERK

*D*ie vorliegende Arbeit beinhaltet interne und vertrauliche Informationen der Firma dotSource GmbH. Die Weitergabe des Inhaltes der Arbeit und eventuell beiliegender Zeichnungen und Daten im Gesamten oder in Teilen ist grundsätzlich untersagt. Es dürfen keinerlei Kopien oder Abschriften - auch in digitaler Form - gefertigt werden. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma dotSource GmbH. Die Arbeit ist nur Mitgliedern des Prüfungsausschusses zugänglich zu machen.

ABSTRACT

Even today pattern classification is still a problem for machines. With the introduction of the term "deep learning" 2006 the branch of neural networks which has been almost declared dead has been rediscovered in the field of machine learning. Over the last five years major successes with convolutional neural networks have been achieved. Convolutional neural networks belong to the category "deep neural networks" and are based on digital signal processing techniques. In this thesis a number of convolutional network models will be developed and their abilities to classify and generalize will be tested. The data that has to be classified consists of pictures that needs to be labeled as productdetail and non productdetail sites. Resulting nets are based on algorithms and techniques that are used in network models that are considered to be state of the art.

ZUSAMMENFASSUNG

Die Klassifizierung von Mustern stellt für Maschinen immer noch ein Problem dar. Mit der Einführung des Begriffs "Deep Learning" 2006, wurde der schon für fast tot erklärte Zweig der Neuronalen Netze des Fachbereichs Maschinelles Lernen wieder entdeckt. Dabei wurden speziell in den letzten fünf Jahren große Erfolge mit Faltungsnetzen erzielt. Faltungsnetze zählen zu den "Deep neural networks" und basieren auf Techniken der digitalen Signalverarbeitung. In dieser Arbeit werden verschiedene Faltungsnetzwerke erstellt und auf ihre Fähigkeiten zu klassifizieren und zu generalisieren überprüft. Bei den zu klassifizierenden Daten handelt es sich um Bilder von Shop-Webseiten, diese sollen in die Kategorien Produktdetailseiten und Nicht-Produktdetailseiten eingeteilt werden. Dabei wird auf Algorithmen und Techniken zurückgegriffen, die in Modellen Verwendung finden die als State of the Art gelten.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ACRONYME

ILSVCR Imagenet Large Scale Visual Recognition Challenge

DNN Deep Neural Network

CNN Convolutional Neural Network

MLP Multilayer Perceptron

SGD Stochastic Gradient Descent

ReLU Rectified Linear Unit

DL4J Deeplearing4J

RNN Recurent Neural Network

1 EINFÜHRUNG

1.1 MOTIVATION

Lernen ist keine triviale Aufgabe. Was der Mensch spielerisch leicht vollbringt, ist keinesfalls einfach für die Maschinen. Die Erkennung von Mustern, Objekten und Formen durch Software ist nach wie vor ein Problem, das viel Zeit beansprucht. Damit Programme mit visuellen Reizen arbeiten können, müssen diese zunächst digitalisiert werden. Das kann auf unterschiedlichsten Wegen erfolgen, beispielsweise durch das einfache Aufnehmen eines Bildes oder eines Videostreams per Digitalkamera. Die dabei entstandenen Daten enthalten Muster, die grob in zwei Klassen eingeteilt werden können:

1. Muster mit zweidimensionalen Ausprägungen, darunter fallen vor allem Bilder von Schriftzeichen und Symbolen.

2. Muster mit dreidimensionalen Ausprägungen, zum Beispiel Bilder von Gesichtern, Personen oder Tieren.

Beide Musterklassen enthalten Varianzen, die sich in verschiedenen Formen äußern. Bei der ersten Klasse sind das zum Beispiel unterschiedliche Schriftstile, Strichdicke, Verzerrung, Skalierung, Rauscheffekt und Verunreinigung. In Abbildung 1 kann man Ziffern finden,

Abbildung 1: Handgeschriebene Ziffern

die selbst dem Menschen Probleme bereiten könnten. In der zweiten Klasse kommen noch weitere Varianzen hinzu, wie unterschiedliche Perspektiven oder die Stärke der Belichtung sowie die Tiefe der abgebildeten Muster [5]. Beispiele hierfür lassen sich in Abbildung 2 finden. Diese Varianzen erschweren die Klassifizierung der Muster erheblich. Zur Lösung des Problems wurden zahlreiche Ansätze verfolgt. Man unterscheidet heute drei prinzipielle Ansätze: den strukturellen, syntaktischen und statistischen Ansatz. In den letzten Jahren konnten einige große Erfolge in der Erkennung von Mustern mit neuronalen Netzen verzeichnet werden. Einzugliedern sind diese im

Bereich der statistischen Ansätze. Folgt man der jährlichen Imagenet Large Scale Visual Recognition Challenge (ILSVCR), kann man sehen wie durch die vielen eingereichten Modellen die Genauigkeit in der Mustererkennung ständig steigt. Dabei ist das erstplatzierte Modell von Krizhevsky, Sutskever und Hinton 2012 mit einer Verbesserung des Top-5 Fehlers von 10,9% im Vergleich zum zweitplatzierten zu nennen [9]. Top-5 Fehler beschreiben die Abweichung der berechneten und gewünschten Ausgabe der fünf wahrscheinlichsten Kategorien. Durch Verbesserungen mit solchen Ausmaß, eröffnen sich viele Anwendungsgebiete in der Praxis. Dafür gibt es zahlreiche Beispiele, so steht mit den neuesten Entwicklungen seitens Tesla die Automobilbranche kurz davor den Autopiloten zu realisieren. Ein weiteres Beispiel kommt aus der Medizin und berichtet von einem Deep Neural Network (DNN), das bei der Erkennung von Hautkrebs gleichauf mit Experten aus dem Fachgebiet Dermatologie ist [2]. Wie bei allen Innovationen muss sich jede Branche fragen, ob diese neue Technologie nicht auch in ihre Domäne erschließbar ist und für was für Umbrüche sie sorgen kann. So kommt es, dass auch im Bereich E-Commerce immer mehr Interesse wächst Erfahrungen mit neuronalen Netzen zu machen. Große Firmen spielen hier Vorbildrollen und zeigen eindrucksvoll den praktischen Nutzen in ihren Produkten. Ein Beispiel dafür ist Amazons Produktvorschlagssystem. Dahinter steckt ein DNN, dessen zugrunde liegende Software seit Mai 2016 frei verfügbar ist.

ILSVCR Quelle: www.image-net.org, letzter Besuch 28. Januar 2017

Die Bilder aus Abbildung 2 gehören urheberrechtlich: www.meinhusky.de, letzter Besuch 17. Januar 2017

Abbildung 2: Dreidimensionale Muster

1.2 ZIELSETZUNG

Es ist oftmals unklar, an welchen Merkmalen neuronale Netze lernen. Ein Beispiel liefert Googles Researchblog. Hier wird veranschaulicht,

nach was ein Netzwerk sucht, wenn es in einem Bild eine Hantel erkennen soll. In Abbildung 3 kann man sehen, wie ideale Bilder von Hanteln aussehen, um vom Netz als solche erkannt zu werden. Man kann vermuten, dass das Netz mit Bildern trainiert wurde, auf dem Hanteln häufig mit einer Hand oder einem Arm abgebildet wurden, weshalb es jetzt der Meinung ist, Hand bzw. Arm und Hantel gehören zusammen.

Abbildung 3: GoogLeNet Hantel

Quelle Abbildung 3: research.googleblog.com, letzter Besuch 27. Januar 2017

Im Mai 2016 gelingt es AlphaGo, einer Maschine basierend auf einem neuronalen Netz und entwickelt von Google, den Spitzenspieler Lee Sedol in einem Match Go von 5 Spielen 4:1 zu schlagen. Im Gespräch nach dem ersten Spiel sagt Lee, dass AlphaGo Züge gespielt hat, zu denen kein menschlicher Go-Spieler in der Lage gewesen wäre [3].

Erscheinungen wie diese lassen die Frage aufkommen, ob die Perspektive der Maschinen unsere eigene erweitern kann. Eine solche Perspektive könnte "deep dreaming" liefern. "Deep dreaming" wurde erstmals von Google 2015 benannt und erfreute sich sehr viel Aufmerksamkeit im Internet. Was auf den ersten Blick wirr und chaotisch wirkt, wie das Beispiel der Hanteln, ist für das erzeugende neuronale Netz das Ideal einer der Kategorien, in die es normalerweise zuordnen soll. Mit Ideal ist an dieser Stelle gemeint, wenn das Bild der Hantel von oben in ein Netz als Eingabe gegeben wird, welches Hanteln erkennen oder lokalisieren soll, ist die Zuversicht die Hantel erkannt zu haben nahe der 100%. Im Grunde funktioniert "deep dreaming" wie die stochastischen Gradientenabstiegsverfahren die in den Folgenden Kapiteln beschrieben werden. Es wird ein normales Bild rückwärts durch ein trainiertes Netz geleitet, doch statt Anpassungen im Netz vorzunehmen, wird hier das Bild angepasst. Mit einem Netz was versteht wie aktuelle Webseiten aussehen, wäre es möglich die Perspektive der Maschine im Gestaltungsprozess einer Webseite zu berücksichtigen. Dafür muss allerdings erst ein Netz entstehen, das in der Lage ist eine spezielle Seite zu erkennen. Da es für so ein Netz notwendig ist viele Beispieldaten zum Lernen zur Verfügung zu haben, soll im ersten Arbeitspaket zur Realisierung der Idee geprüft werden ob hierfür ein Netz eingesetzt werden kann, um den manuellen Aufwand bei der Datenerhebung zu minimieren.

Ziel dieser Arbeit ist es zu prüfen ob es möglich ist, Bilder von Shopseiten übergreifend zu kategorisieren. Um die Aufgabe zu ver-

einfachen, wird die Einteilung in nur zwei Kategorien vorgenommen: Produktdetailseiten und nicht-Produktdetailseiten. Für diese Arbeit soll geprüft werden, ob speziell ein Faltungsnetzwerk, bzw. ein DNN diese Aufgabe erfüllen kann.

1.3 GLIEDERUNG

Im Folgenden wird auf den Aufbau der Arbeit eingegangen. Kapitel 2 beschäftigt sich mit den Grundlagen neuronaler Netze. Dafür wird kurz auf das biologische Vorbild eingegangen, um dann den Sprung von der einzelnen, künstlichen Nervenzelle auf die daraus entstehenden neuronalen Netze zu schaffen. Am Ende des zweiten Kapitels wird erläutert wie diese Netze lernen.

Anschließend richtet sich in Kapitel 3 der Fokus auf das eigentliche Thema Faltungsnetzwerke. Vorweg wird geklärt, was die Probleme der Netze aus Kapitel 2 sind. Aufbauend werden die Lösungsansätze, die in Faltungsnetzen vereint werden, erklärt, angefangen bei der Faltung über die Unterabtastung (engl. Subsampling & Pooling) bis zu den geteilten Gewichten.

Kapitel 4 beschreibt das Vorgehen. Fragen wie

1. "Wie sehen die Daten aus und wie werden diese erhoben?" und
2. "Wie werden die Netze umgesetzt und trainiert?" sollen hier geklärt werden.

Dafür wird der Prozess über die Formulierung der Anforderungen, der Auswahl bis zur Benutzung der Werkzeuge beschrieben.

In Kapitel 5 werden die Testszenarien beschrieben und die Erwartungshaltung definiert. Gefolgt von den Ergebnissen und einer Auswertung.

Kapitel 6 soll rückblickend die Erkenntnisse der Arbeit zusammenfassen und einen Ausblick für Ansätze folgender Arbeiten schaffen.

2

NEURONALE NETZE GRUNDLAGEN

2.1 DAS BIOLOGISCHE NEURON

Die Frage "Wie lösen biologische Systeme Probleme?"liefert die Motivation, als Einführung in das Thema, sich einen kurzen Überblick über natürliche neuronale Netze zu verschaffen. Ein Neuron ist eine

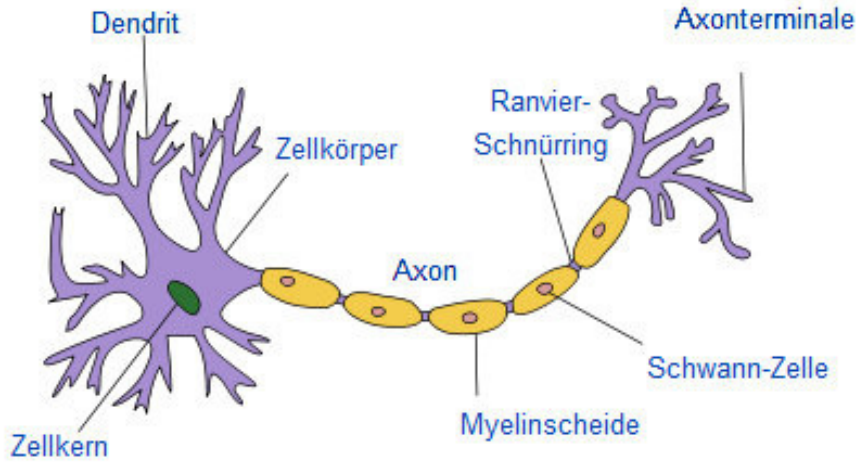

Abbildung 4: Vereinfachte Darstellung eines Neurons

Quelle Abbildung 4: scienceblogs.de, letzter Besuch 17. Januar 2017

Zelle, die sich auf die Aufnahme, Verarbeitung und Übertragung von Reizen spezialisiert hat. Normalerweise besteht es aus mehreren Dendriten, einem Zellkörper und einem Axon. Über die fein verzweigten Fortsätze, die Dendriten, erhält eine Nervenzelle von bis zu 200.000 anderen Nervenzellen Informationszufluss. Der Zellkörper ist die lebenswichtige Zentrale einer Nervenzelle und dient der Informationsverarbeitung. Über das Axon (griechisch oxon = Achse) leitet die Nervenzelle Impulse an andere Nervenzellen bzw. diverse Erfolgsorgane weiter. Dieser Nervenfortsatz ist weniger stark verzweigt und kann je nach Zelltyp bis über einen Meter lang sein. Die Reizweiterleitung erfolgt in einer Nervenzelle mittels schwacher elektrischer Ströme, dem sogenanntem Aktionspotential. Axon sind häufig von Markscheiden umhüllt, wobei diese eine Art Isolierung bilden und in regelmäßigen Abständen schmale Unterbrechungen (Schnürringe) aufweisen. Die Reizübertragung auf andere Nervenzellen oder Erfolgsorgane, zum Beispiel Muskeln oder Drüsen, erfolgt überwiegend chemisch an den Synapsen (synapsis = Verbindung). An diesen Kontaktstellen, die die Synapsen darstellen, bewirkt das eingehende elektrische Signal die Ausschüttung chemischer Substanzen, die als Überträger- oder Botenstoffe fungieren. Auf der gegenüberliegenden Seite des synapti-

schen Spalts besitzt die Oberfläche der Zelle spezielle Rezeptoren für die ausgeschütteten Überträgersubstanzen. Ist die Erregung dieser Rezeptoren groß genug, entsteht in der Zielzelle wieder ein elektrischer Impuls. Die Überträgersubstanzen werden im Zellkörper der Nervenzelle gebildet, im Axon an die Nervenenden transportiert und hier in kleinen Bläschen gespeichert. Bei eingehenden Nervenimpulsen werden diese Überträgersubstanzen dann in den synaptischen Spalt ausgestoßen, um schließlich durch Enzyme abgebaut oder zum Teil auch wieder in die Nervenenden aufgenommen [11].

2.2 DAS KÜNSTLICHE NEURON

Aufbauend auf den Grundlagen und der groben Funktionsweise des biologischen Vorbilds kann jetzt auf sehr einfache und rudimentäre Weise abstrahiert werden.

Auf die Idee eingehen, die Funktionsweise der einzelnen Nervenzelle zu übertragen, über Details Referenzen zu dem biologischen Vorbild knüpfen.

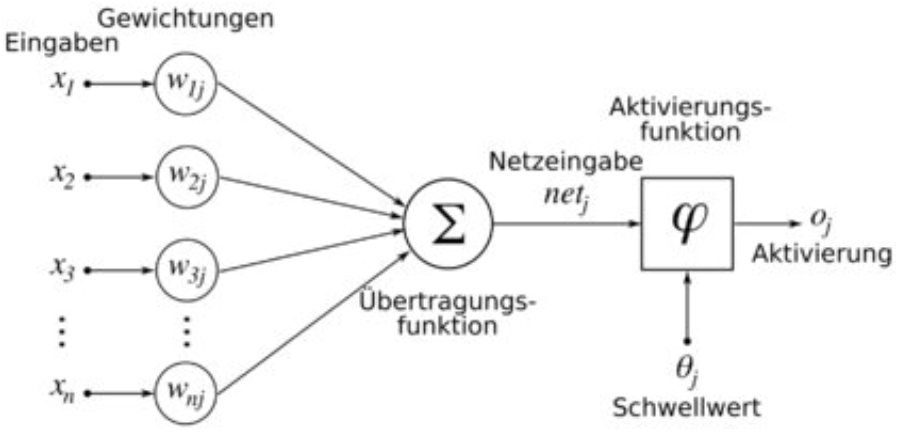

Abbildung 5: Aufbau der künstlichen Nervenzelle

Quelle Abbildung 5: wikipedia.org, letzter Besuch 17. Januar 2017

Das abgebildete Modell bezeichnet man gemeinhin als Perzeptron-Modell, welches bereits 1958 von Rosenblatt entwickelt wurde [14]. In seiner Grundform wird nur ein einzelnes Neuron (Simple Perceptron) simuliert. Dieses Perzeptron besteht aus vier großen Segmenten:

- eine Menge von Eingabewerten, gepaart mit jeweils einem Gewicht pro Eingabewert,
- eine Summenfunktion: $net_j = \sum(x_i w_i)$ mit x_i als Eingabewert und w_i als dazugehörige Gewichtung,
- einem Schwellwert Θ, dieser gibt das Niveau an, was erreicht werden muss, damit das Perzeptron "feuert"
- die Aktivierungsfunktion $\Phi(\sum(x_i w_i) + \Theta)$

Die gesamte Berechnung des Neurons kann also in einer Gleichung zusammengefasst werden:

$$\Phi(\sum(x_i w_i) + \Theta)$$

Dabei werden unterschiedliche Funktionen für $\Phi(x)$ eingesetzt. Häufig zur Anwendung kommende Funktionen sind die lineare $y = x$ und die sigmoide $y = \frac{1}{1+e^{-x}}$ Aktivierungsfunktion. Auf die in der Arbeit benutzte Funktion wird im Folgenden eingegangen.

2.3 KÜNSTLICHE NEURONALE NETZE

x_i muss kein Bestandteil eines direkten Eingabemusters sein, sondern kann auch die Ausgabe eines Neurons sein, was davor geschaltet und aktiviert wurde. Genau so muss ein Ausgangssignal nicht das Endgültige sein, es kann Eingangswerte für dahinter folgende weitere Neuronen liefern. Hängt man also mehrere Neuronen aneinander, erhält man ein neuronales Netzwerk. Es gibt viele verschiedene Modelle solcher Netzwerke die sich immer weiter in ihren Anwendungsgebieten entwickeln. Diese Modelle haben Schwächen und Stärken. Ein klassischer Vertreter ist das Multilayer Perceptron (MLP), was als Grundlage für alle vorwärtsgerichtete Modelle gilt. Da sich in dieser Kategorie auch Faltungsnetze befinden, bietet es sich an das MLP kurz näher zu beleuchten. MLP bestehen aus drei oder mehr Schichten. Einer Eingabeschicht, einer Ausgabeschicht und dazwischenliegend mindestens eine, aber oftmals mehrere versteckte Schichten. Dabei sind alle Neuronen der Eingabeschicht mit den Neuronen der ersten versteckten Schicht verbunden. Diese wiederum sind mit allen Neuronen der darunterliegenden Schicht, verbunden. Das wiederholt sich bis zur Ausgabeschicht. Die Anzahl der Neuronen auf der Eingabeschicht richtet sich nach den Ausprägungsmerkmalen des Eingabemusters. In der Ausgabeschicht sind es die Anzahl der unterschiedlichen Kategorien, in die das Netz einteilen soll. Die Anzahl der auf den versteckten Schichten benötigten Neuronen richtet sich nach der Komplexität der Eingabemuster. Dabei gilt folgende Daumenregel:

- Zu viele Neuronen auf einer Schicht und das Netzwerk hat genug Platz um alle unterschiedlichen Ausprägungsmuster zu speichern. In dem Fall ist das Netz zwar sehr genau im Erkennen bekannter Muster, verliert aber die Fähigkeit von bereits gelernten zu generalisieren. Dadurch wird das Ergebnis des Netzes schlechter auf Veränderung und Fehler reagieren. Dieses Problem wird auch “overfitting” genannt.

- Zu wenig Neuronen auf einer Schicht bedeuten wiederum, dass das Netzwerk nicht in der Lage ist, auf alle Merkmalsausprägungen der Eingabemuster zu lernen. In dem Fall wird das bereits Gelernte immer wieder mit dem Auftauchen neuer Ausprä-

gungen überschrieben und es ist in seiner Kapazität zu Lernen beschränkt.

Als Lernen wird im Bezug auf neuronale Netze das Anpassen der gewichteten Verbindungen und der Schwellwerte bezeichnet. Der Lernprozess bei einem MLP sowie bei Faltungsnetzen ist das überwachte Lernen. Das Grundprinzip dieses Lernverfahrens lässt sich mit dem Lernen mit hilfe eines Lehrers vergleichen. Ein Lehrer, der dem Lernenden etwas beibringen möchte, stellt eine Aufgabe, die der Lernende dann zu lösen hat. Hat dieser eine Lösung gefunden, wir diese mit der des Lehrenden verglichen und daraus Schlüsse gezogen. Wird ein MLP trainiert, so bekommt es Eingabedaten, die das Netzwerk durchlaufen sollen. Zu den jeweiligen Eingabedaten muss ein zweiter Satz dazugehöriger Ausgabedaten existieren, das erwartete Ergebnis. Die Eingabedaten laufen durch das Netz, es wird eine Ausgabe berechnet und dann mit der erwarteten Ausgabe verglichen. Tritt eine Abweichung zwischen den beiden Mustern auf, müssen die Stellschrauben des Netzwerks, also die Gewichtung der Verbindungen sowie die Schwellwerte, so angepasst werden, dass der Unterschied zwischen dem berechneten und erwarteten Ergebnis kleiner wird. Die berechnete Ausgabe wird hier vom erwarteten Ergebnis abgezogen, der daraus resultierende Fehler wird quadriert um ein Kompensieren von positiven und negativen Fehlern ausschließen zu können. Um den Fehler zu minimieren können verschiedene Ansätze verfolgt werden. Beliebt sind in solchen Situationen Brute-Force- und Gradienten-Abstiegsverfahren. Bei ersteren werden alle möglichen Werte für die Eingabe und die der Gewichtungen und Schwellwerte verglichen und die Konstellation mit dem kleinsten Fehler ausgewählt. Dieser Ansatz hat nur noch theoretischen Wert für neuronale Netze, praktisch sind aktuelle Netzwerke so komplex, dass diese Aufgabe unnötig viel Zeit beanspruchen würde. Stochastische Gradienten-Abstiegsverfahren werden immer dann eingesetzt, wenn man bei Optimierungsproblemen keine konventionelle Lösungsansätze mehr hat. Hierbei wird nach einer vorgegebenen Anzahl an Schritten versucht, einen relativ kleinen Fehler ausfindig zu machen. Dabei werden, anstatt alle Möglichkeiten durchzuiterieren, immer nur die steilsten Abstiege im Fehlergebirge verfolgt. Es gibt eine Vielzahl an Algorithmen für stochastische Gradienten-Abstiegsverfahren. Diese basieren alle auf der von Psychologen Hebb 1949 formulierten Hebb'schen Regel [6]. Der in der Lösung des Problems dieser Arbeit benutzte Algorithmus wird im Folgenden noch einmal detaillierter beleuchtet. Der wohl bekannteste Stochastic Gradient Descent (SGD), im Fachgebiet neuronale Netze, ist das Fehlerrückrechnungsverfahren, auch Backpropagation genannt. Eine Schwäche dieses Verfahrens ist, dass, wenn immer der steilste Abstieg genommen wird, es nicht zwangsläufig in das kleinste Fehlertal führen muss. Sogenannte "Hochtäler"können auftreten, aus der der Algorithmus nicht mehr

herausfindet. Dieses Problem wird zumindest auf theoretischer Seite angeführt, in der Praxis erwies es sich als weniger kritisch [10].

2.4 BACKPROPAGATION ALS LERNVERFAHREN

Backpropagation ist ein Spezialfall des SGD und geht auf die Arbeit von Rumelhart, McClelland und PDP Research Group 1986 zurück [15]. Rumelhart, McClelland und PDP Research Group widerlegte mit seiner Arbeit, die fälschliche Annahmen von Minsky und Papert[12], komplexere Strukturen würden die selben Probleme wie das Perzeptrons erben.

Wie bei SGD benutzt das Verfahren das Prinzip der Fehlerrückführung.

$$E = \frac{1}{2} \sum_{P \epsilon S} (t_P - o_P)^2$$

, mit:

E	der quadratische Fehler der minimiert werden soll,
S	die Menge der Trainingsbeispiele,
$P \epsilon S$	das jeweilige Trainingsbeispiel aus S,
t_P	die gewünschte Ausgabe,
o_P	die berechnete Ausgabe.

Es wird ein Wertepaar benötigt, bestehend aus Eingabedaten für das Netz und die dazugehörige gewünschte Ausgabe des Netzes. Mithilfe der Eingabedaten wird eine Ausgabe berechnet. Die Differenz der berechneten Ausgabe und der gewünschten Ausgabe wird ein Fehler bestimmt. Zunächst wird die partielle Ableitung der Ausgabeneuronen gebildet, hier Ausgabe Neuron j:

$$\delta_j = t_j - o_j$$

Die Ableitungen der restlichen Neuronen werden durch Anwendung der Kettenregel gebildet:

$$\delta_j = g(a_j) \sum_k \delta_k w_{kj}$$

, mit k als Index für alle Neuronen der vorangehenden Schichten, die mit Neuron j verbunden sind und w_{kj} als deren gewichtete Verbindung zu j. Die tatsächliche Anpassung lässt sich dann wie folgt bestimmen:

$$\Delta w_{ij} = -n \frac{\delta E_P}{\delta w_{ij}} = n \delta_j x_i$$

, mit E_P als Fehler des jeweiligen Trainingsbeispiels, x_i als ursprünglicher Eingabewert und der Lernraten.

3

FALTUNGSNETZE GRUNDLAGEN

Die Fähigkeit von mehrschichtigen Netzwerken, mithilfe von SGD, selbst komplexe hoch-variante Muster zu lernen macht sie ideal für Objekt-Erkennung auf Bildern. Dennoch gibt es zwei allgemeine Probleme damit:

- Erstens, heutige Kameras nehmen nicht selten Bilder mit 20 Megapixel auf. Bilder in dieser Größenordnung direkt in ein mehrschichtiges Netz als Input zu laden, würde selbst aktuelle Technik überfordern. Eine voll verbundene Schicht auf der ersten Ebene, mit Hundert Neuronen, würde schon zu (20.000.000 Pixel * 3(RGB Ebenen) * 100 Neuronen =) 6 Milliarden trainierbaren Gewichtungen führen.
- Zweitens, die Topologie des Bildes wird komplett ignoriert. Das bedeutet, die Reihenfolge in der die Pixel-Informationen verarbeitet werden, hat keinen Einfluss auf das Trainingsverhalten. Mann könnte also, solange die Pixel Informationen die selben bleiben, sie in beliebiger Reihenfolge angeben. Dabei ist aber bekannt, dass benachbarte Pixel korrelieren und eine starke lokale Struktur aufweisen.

Auf diese Probleme wird mit Faltungsnetze reagiert. Faltungsnetze sind eine besondere Form von neuronalen Netzen. Zur Lösung der genannten Probleme, vereinen sie drei Kernideen: lokale rezeptive Felder (mittels Faltung), geteilte Gewichte und Unterabtastung (Sampling/Pooling). Auf die Techniken wird im Folgenden eingegangen. Erstmals wurden Faltungsnetze 1993 eingesetzt von Vaillant, Monrocq und LeCun [20] eingesetzt. Die Eingabe des Netzes war ein Graustufenbild mit fester Höhe und Breite. Als Ausgabe lieferte es dann einen Vektor kleiner Dimension, der die unterschiedlichen Muster die zu erkennen sind kodiert.

3.1 AUFBAU UND FUNKTIONSWEISE

Im Kern besteht ein Faltungsnetz aus vernetzten Faltungsoperationen, mit verschiedenen Filtermasken. In dem folgenden Abschnitt wird auf den Aufbau und die Funktionsweise eingegangen, um ein besseres Verständnis für die Experimente in Kapitel 5 zu erlangen. Als Beispiel soll das in [20] verwendete Netz zur Veranschaulichung dienen. Ziel des Netzes ist es, zu erkennen ob sich ein Gesicht auf dem Bild befindet. Der Übersicht halber werden die Schichten (eng. Layer), angefangen bei der Eingabeschicht bis zur Ausgabeschicht

von 1 bis 5 nummeriert. L1 entspricht der Eingabeschicht. Die Felder der Schicht L2 bis L4 stellen Zwischenergebnisse innerhalb des Netzes dar. L5 entspricht dem Ausgabevektor, der in diesem einfachen Beispiel nur aus einem Wert besteht. Entweder das Bild enthält ein Gesicht, dann liefert das Netz einen Wert größer als Null oder das Bild enthält kein Gesicht, dann liefert das Netz einen Wert kleiner oder gleich Null ist. Abbildung 6 zeigt den Aufbau des Faltungsnet-

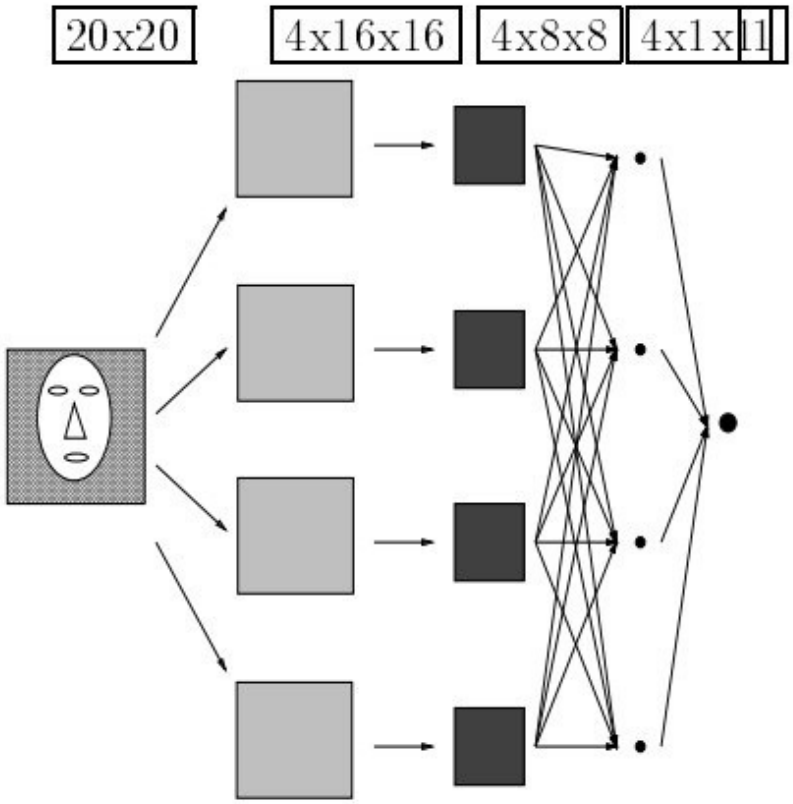

Abbildung 6: Aufbau Faltungsnetz nach Vaillant, Monrocq und Le Cun, Quelle:[20]

zes nach Vaillant, Monrocq und Le Cun. Im Folgenden wird auf die einzelnen Schritte eingegangen, die die Daten passieren müssen auf dem Weg durch das Netzwerk.

3.2 FALTUNG

Faltung stammt aus der Signalverarbeitung. In der ersten Schicht L1 werden parallel vier Faltungen mit unterschiedlichen Masken auf die Eingabe angewandt. Schichten auf die eine Faltungsoperation angewendet wird, werden Faltungsschichten genannt. Die Gleichung zeigt die Faltung eines Eingabebildes i mit einer Filtermaske h, $\otimes$ stellt hier den zweidimensionalen Faltungsoperator dar:

$$g(x,y) = i(x,y) \otimes h(x,y) = \sum_{m_x=-\infty}^{\infty} \sum_{m_y=-\infty}^{\infty} i(m_x, m_y) h(x - m_x, y - m_y)$$

Aufgrund der fehlenden Randbehandlung nimmt die Größe des Bildes zu L2 ab. Abbildung 7 veranschaulicht dabei den Vorgang, hier mit einem 7x7 Eingabebild und einem 3x3 großen Kernel.

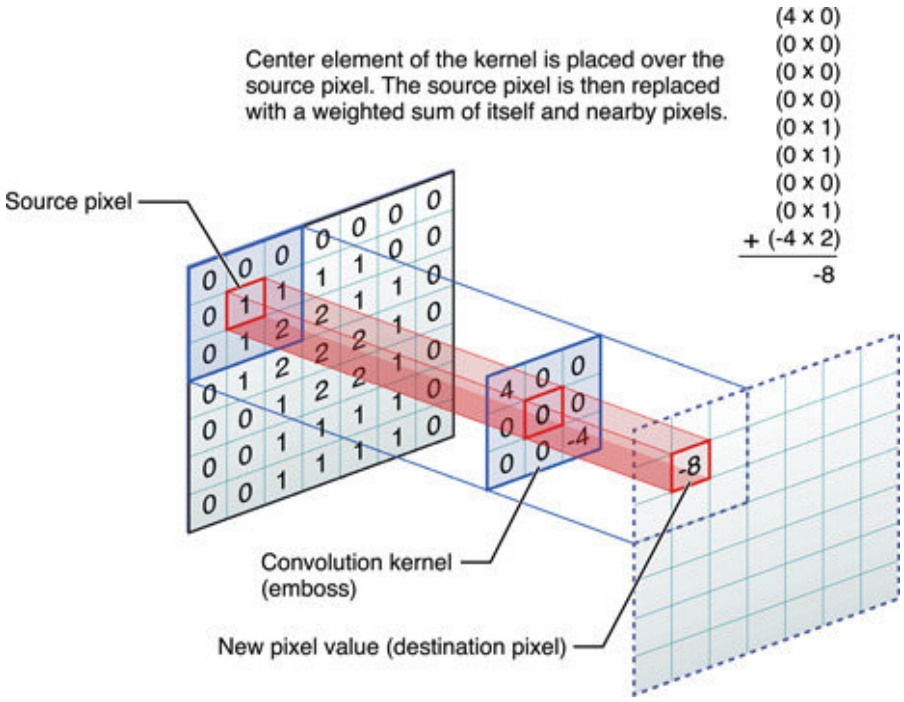

Abbildung 7: Faltung eines zweidimensionalen Vektors mit 3x3 Faltungsmaske oder Kernel

Quelle Abbildung 7: https://developer.apple.com, letzter Besuch 17. Januar 2017

Beispielhaft wird hier die Berechnung von g(x,y) durchgeführt. Anteile des Signals die für die Erkennung eines Gesichts wichtig sind, sollen durch Faltung verstärkt, bzw. irrelevante Anteile unterdrückt werden. Durch die Verwendung mehrere Masken können unterschiedliche Musterausprägungen betrachtet werden. In aktuellen Netzen werden die Filter mit zufälligen Werten initialisiert, der dadurch resultierende Filterungseffekt ist daher ungewiss. Ein solcher Filter kann beispielsweise wie K, hier 3x3, aussehen:

$$K = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Den Effekt der Anwendung von Filter K kann man in Abbildung 8 sehen. Eingabedaten eines Faltungsnetzwerks werden auf die selbe Weise gefiltert. Allerdings werden die Masken im Laufe des Trainings immer weiter angepasst.

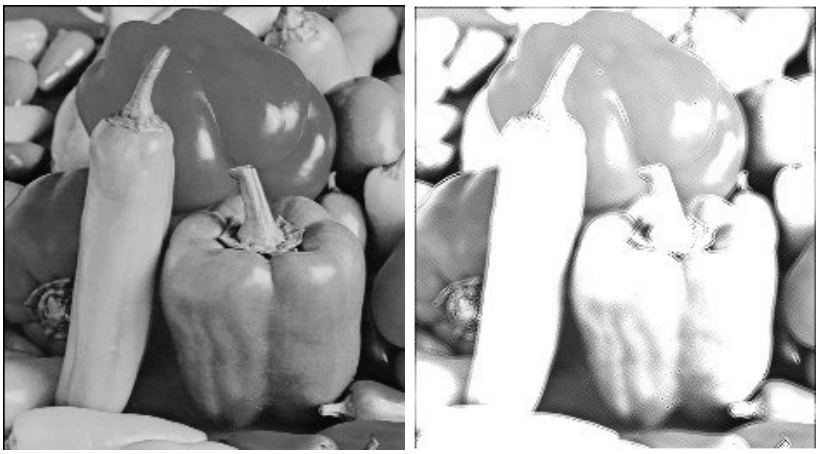

Abbildung 8: Faltung: Bild vor(links) und nach(rechts) Anwenden des Filters

In einem Faltungsnetzwerk sind die Werte der Masken, mit den Gewichte eines Perzeptrons zu vergleichen. Abhängig von den Gewichtungen der Neuronen verändert sich ihre Ausgabe, sollten diese angepasst werden. Faltungsnetze arbeiten analog dazu, nur das wenn ihre Masken angepasst werden, anders Gefalten wird. So entstehen in einem überwachten Lernprozess eine Sammlung an Masken die verstärkend auf die zu lernenden Merkmale korrelieren[8].

3.3 SUBSAMPLING UND POOLING

Von Vaillant, Monrocq und Le Cun in [20] vorgeschlagenen Netz folgt auf L1 dann eine Schicht die für die Reduzierung der Datenmenge bestimmt ist. Da im Allgemeinen die Ausgabeschicht weniger Werte besitzt als vergleichsweise die Eingabeschicht, wird Schicht für Schicht die Anzahl der Werte verringert und der Ausgabeschicht angenähert. Im Beispiel werden auf allen vier Pfaden von L1 zu L2 Subsampling-Operationen durchgeführt. L2 wird demnach als Subsampling-Schicht bezeichnet. Abbildung 9 veranschaulicht zwei häufig verwendete Operationen.

Ein weiterer großer Vorteil von Subsampling ist das es einen kleinen Grad an Varianz in verarbeiteten Daten bringt. An diesen Vorteil angelehnt ist auch die Arbeit von Graham [4]. Auf verschiedenen Convolutional Neural Network (CNN) Modellen hat Graham gezeigt wie sich „fractional max-pooling“, durch künstlich erzeugt Verzerrung positiv auf die Genauigkeit der Netze Auswirkt.

3.4 GETEILTE GEWICHTE

Unter geteilte Gewichte versteht man allgemein, dass zwei oder mehr Neuronen die selben Gewichtungen benutzen. CNN machen von die-

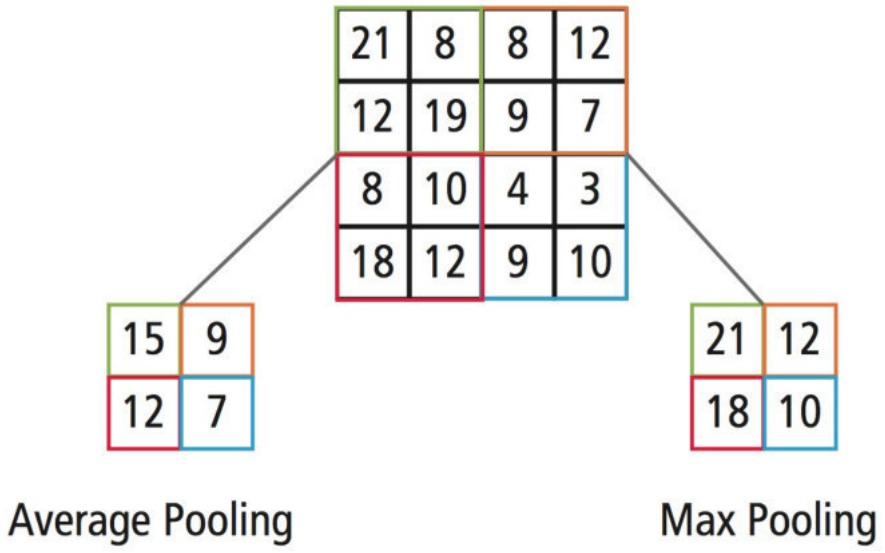

Abbildung 9: Unterabtastung mit average Pooling und max Pooling

Quelle Abbildung 9: www.embedded-vision.com

ser Idee Gebrauch. Ziel ist es, wie bei Einsatz von Subsamplingschichten, die Anzahl der trainierbaren Parametern zu verringern. Dafür muss der Backpropagations-Algorithmus aus Kapitel 2 angepasst werden. Die Besonderheit hier ist, dass die Gewichtung eines Neurons in einer Faltungsschicht nicht mehr unabhängig modifiziert werden kann. Jedes Wertepaar (i,j) soll zwei verbundene Neuronen darstellen. Jetzt wird für jedes Gewicht w_k eine Menge W_k definiert, bestehend aus einer Anzahl von Wertepaaren (i,j). Das Gewicht dieser Verbindung sei $u_{ij} = w_k$ mit $\forall(i,j)\epsilon W_k$. W_k adressiert alle Verbindungen die sich das Gewicht w_k teilen. Die Änderung der Gewichte entspricht nun:

$$\Delta w_{ij} = -n \sum_{(i,j)\epsilon W_k} \frac{\delta E_P}{\delta u_{ij}}$$

Die partielle Ableitung eines Parameters w_k ergibt sich durch Summieren der partiellen Ableitung bezüglich der einzelnen Werte für u_{ij}. Dazu sei erwähnt, in den Faltungsschichten werden Gewichtungen geteilt, in Subsamplingschichten nicht.

3.5 STATE-OF-THE-ART TECHNIKEN UND ALGORITHMEN

Im Folgenden werden einige essentielle Techniken und Algorithmen angerissen, um die Struktur der entwickelten Netzmodelle späterer Kapitel besser verstehen zu können.

3.5.1 *Momentum*

In der Praxis haben sich einige Techniken herauskristallisiert, die das Training beschleunigen oder sogar die Genauigkeit der Netze verbessern. Im Folgenden soll ein Einblick in aktuell verwendete Techniken stehen. Angefangen bei einer Optimierung des SGD, dem Momentum. Im Normalfall benötigt das Verfahren eine große Anzahl an Schritte um den Fehler zu einem akzeptablen Minimum zu führen. Nähert sich der Algorithmus einem lokalen Minimum, oszilliert der Graph stark bis er sein Ziel erreicht. Momentum hingegen verwendet "Schwung" aus dem letzten Update um den neuen Update Schritt zu bestimmen.

Quelle Abbildung 10: http:dsdeepdive.blogspot.com, letzter Aufruf 17. Januar 2017

Abbildung 10: Standart SGD vs. SGD mit Momentum

Abbildung 10 zeigt zwei Fehlergebirge, mit einem lokalen Minimum im jeweiligen Zentrum, rot eingezeichnet sind links die Schritte des SGD und rechts die Schritte des durch Momentum beschleunigten SGD. Der "Schwung" setzt sich aus dem Änderungsvektor addiert mit dem vorangegangenen Update multipliziert mit einer Momentum-Konstante $\eta < 1$. Wenn die Änderungsvektoren beide in dieselbe Richtung zeigen, werden so die Schritte vergrößert, im Fall dass der neue Änderungsektor die Richtung wechselt, wird die Änderung vermindert. Das führt Allgemein zu Beschleunigung und reduziert in der Nähe des lokalen Fehlerminimums das oszillieren des Änderungsgraphen.

3.5.2 *Dropout*

Dropout ist eine beliebte Möglichkeit zum Regularisieren geworden. Srivastava u. a. zeigen in [18], dass diese Technik eingesetzt werden kann um vorwärts gerichtete neuronal Netze zu verbessern und gleichzeitig "overfitting"-Effekte zu reduzieren. Dabei wird auf Schichten auf denen Dropout angewendet wird, zu einer Wahrscheinlichkeit p, als Hyperparameter, ein Neuron aktiv gehalten, anderenfalls auf Null gesetzt. p wird als Dropoutrate bezeichnet. Modelle die sich der Dropout-Technik bedienen erreichen state-of-the-art Leistungen auf den bekannten Trainingsdaten wie ImageNet, CIFAR-100 und MNIST. Die entscheidende Idee hinter Dropout ist, dass große Netzwerke schnell am "Overfitting"-Effekt leiden, durch Dropout werden diese großen Strukturen in viele kleine Netzwerke gebrochen. In Standart

SGD Modellen bilden sich über große Strukturen schnell Neuronen die nur zusammen "feuern", eine Art Koadaption, die "Overfitting" begünstigt.

Abbildung 11
Quelle: cs231n.github.io, letzter Besuch 17. Januar 2017

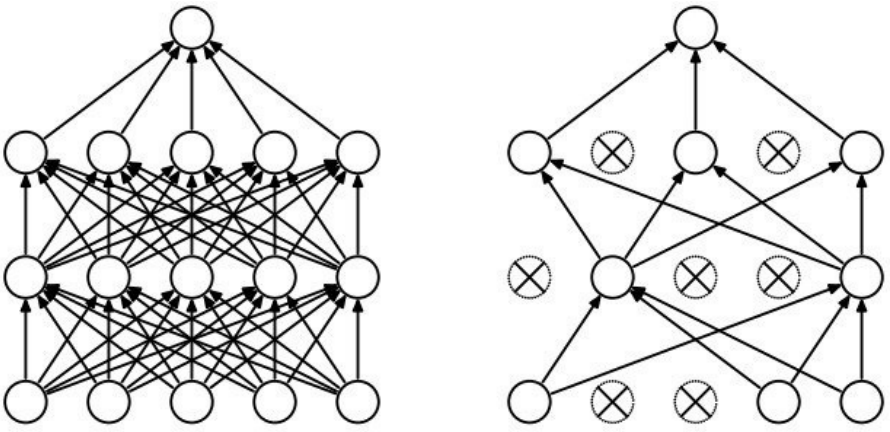

Abbildung 11: Normales KNN vs. Dropout KNN

Abbildung 11 zeigt ein normales dreischichtiges Modell links und rechts ein Dropout-Netz, auf dessen Eingabeschicht sogar schon Dropout angewandt wird. Kehrseite von Dropout ist ein erhöhter Zeitaufwand beim Trainieren des Netzwerks. Laut Srivastava u. a. beträgt die Dauer des Trainings schnell die doppelte bis dreifache Zeit was aber auch sich in der Qualität der gelernten Muster wiederspiegeln soll. In Kombination mit Momentum soll dieser nachteilige Effekt signifikant verbessert werden, es wird von Srivastava u. a. eine Momentum-Konstante von 0.95 - 0.99 vorgeschlagen [18].

3.5.3 *Rectified Linear Unit*

Rectified Linear Unit (ReLU) bezeichnen Neuronen (bzw. Units) mit der Aktivierungsfunktion $f(x) = max(0, x)$. 2010 zeigten Nair und Hinton in [13] ihre Einfluss in der Anwendung auf dem Datenset Labled Faces in the Wild. ReLu sind dadurch in den letzten Jahren sehr beliebt geworden. Dieser Beliebtheit liegen zwei Vorteile zugrunde:

1. Relu beschleunigen die Konvergenz von SGD im Vergleich zu sigmoiden/tanh Aktivierungsfunktionen, Krizhevsky, Sutskever und Hinton [9] berichtet von der 6x Geschwindigkeit,

2. im Vergleich zu sigmoiden/tanh, die berechnungslastigere Operationen wie Exponenten zur Bestimmung brauchen, können ReLu einfach mit ihren Integer Werten übernommen werden.

Ein weiterer Grund warum ReLU den sigmoiden Funktionen vorgezogen wird, ist das Problem der verschwindenden Gradienten. Wenn

sich die x-Werte in diesen Funktionen immer weiter erhöhen oder verringern, verschwindet der Gradient. Der Gradient verschwindet nicht wenn die x-Werte erhöht werden bei der $max(x)$ Funktion.

4 VORGEHEN

In diesem Teil der Arbeit sollen notwendige Schritte angerissen werden, die zur Erfüllung des Ziels benötigt werden. Dafür wird kurz nochmal auf die Ziele eingegangen. Erst wird geprüft, ob Faltungsnetze in der Lage sind Online-Shop-Webseiten anhand von Bildern zu kategorisieren. Das zweite Ziel der Arbeit ist es, zu untersuchen, ob sich auf elementarer Ebene, auf der Faltungsnetze Informationen für Entscheidungen sammeln, Eigenschaften finden lassen, die Online-Shop übergreifend existieren. Aus den Grundlagen ergeben sich folgende Schritte zur Realisierung einer Lösung:

1. Werkzeugauswahl,
2. Datenerhebung ,
3. Netzemodell suchen,
4. Training und Optimierung.

Für das Trainieren eines Faltungsnetzwerks werden viele Daten benötigt, die sich mit den Daten zum Testen der Netze nicht überschneiden dürfen. Da es keine offiziellen Quellen solcher speziellen Daten gibt, werden diese selbst erhoben. Des Weiteren werden die Daten in eine Form überführt werden, in denen sie als Eingabe in ein Netz verwendet werden können. Zu jeder Eingabe wird als weiterer Wert das gewünschte Ausgabemuster benötigt. Die Anzahl der Daten kann nicht durch übliche Operationen künstlich erhöht werden. Darunter fallen zum Beispiel das Drehen und Spiegeln der Daten. Auch ist Verrauschen keine Option bei der Vorverarbeitung, denn Verrauschen könnte dazu führen, dass Muster systematisch eingefügt werden und das Netz anhand dieser Muster lernt. Die Daten müssen also nah am Original bleiben. Aus dem Grund werden mehr Daten benötigt.

4.1 WERKZEUGAUSWAHL

Für die Umsetzung des Netzes soll ein fertiges Werkzeug eingesetzt werden. Eine vorangegangene Onlinerecherche brachte eine 39 Einträge-umfassende Liste an Werkzeugen hervor. Darunter Applikationen, die sich grob in zwei Kategorien einteilen lassen:

1. Toolkits, bestehend aus Funktionsbibliotheken, mit deren Hilfe man programmatisch neuronale Netze erstellen kann,
2. Plattformen, diese bestehen meist aus einem abgeschlossenen System mit grafischer Benutzungsoberfläche, in denen es auf

die bereitgestellten Funktionen begrenzt ist, Netze zu konfigurieren.

Da Werkzeuge aus der Kategorie Plattformen in der Benutzung eingrenzende Funktionalität zur Verfügung stellen, werden diese vorab ausgeschlossen. Die Anzahl der Werkzeuge hat alle Erwartungen überstiegen, aus dem Grund mussten einschränkende Anforderungen definiert werden. Aus dem am Arbeitsplatz verwendeten Betriebssystem gingen keine weiteren einschränkenden Anforderungen hervor. Alle gängigen Betriebssysteme konnten über eine virtuelle Maschine simuliert werden. Aus der verwendeten Hardware ging hervor, dass das System Training über CPU und GPU unterstützen muss. GPU Unterstützung war vorerst weniger relevant, ist für zukünftige Arbeit mit dem System aber zwingend erforderlich. Diese Anforderung basiert auf zwei Gründen, die Anzahl der Daten wird in den folgenden Arbeiten sehr viel größer sein und mit dem Wachstum der Datenmenge wird auch die Zeit des Trainings, so wie die Zeit zur Evaluierung wachsen. In Kombination mit GPU Training ist Nvidia Cuda oder AMD OpenCL Unterstützung gewünscht. Das Werkzeug muss Recurent Neural Network (RNN) unterstützen, zu denen Faltungsnetze zählen. Das weitere Umfeld definierte Anforderungen an die Programmiersprache, die verhältnismäßig bekannt sein sollte, d.h. Java, C#, C, Python und JavaScript werden bevorzugt. Was hauptsächlich die Zeit zur Einarbeitung gering halten soll. Weitere Anforderungen werden an die Dokumentation und den Support gestellt. Wie vollständig ist die Dokumentation? Gibt es Support oder Ansprechpartner bei Problemen mit dem Werkzeug? Zusätzliche Funktionen, die helfen das Netzwerk zu konfigurieren und das Training zu überwachen, sind gewünscht.

Aus den aufgelisteten Toolkits hat Deeplearing4J (DL4J) den besten Eindruck gemacht.

Tabelle 1 gibt eine Überblick der Informationen zu DL4J. DL4J benötigt einige essentielle Programme (git, cmake, OpenMP, gcc und maven) und Systemvariablen (JAVA_HOME) bevor es gebaut werden kann. Je nach Rechnerarchitektur kommen für die Berechnung der im Netz anfallenden algebraischen Operationen folgende Bibliotheken zur Auswahl:

1. für Berechnungen per CPU: Intel MKL, OpenBLAS und ATLAS,
2. für Berechnungen per GPU: Nvidia CUDA.

Für die im Umfang der Arbeit anfallende Berechnungen per CPU wurde OpenBlas ausgewählt. Maven ermöglicht das Laden der DL4J Module, darunter Demos für RNN, aber auch speziell Faltungsnetze und DataVec. DataVec wird eingesetzt zum Vektorisieren der Daten, was für die Verwendung der Daten in der Eingabeschicht notwendig ist. Neben dem Vektorisieren liefert DataVec noch Funktionen zum

ANFORDERUNGEN	DEEPLEARNING4J
Unternehmen:	Skymind
Sprache:	Java, Scala
RNN Unterstützt:	ja
CPU Training Unterstützt:	ja
GPU Training Unterstützt:	ja
CUDA Unterstützt:	ja
Dokumentation:	JavaDocs
Aktive Entwicklung:	ja
Betriebssystem:	Linux,MS Windows, Mac
Zusätzliche Funktionen:	DataVec und Arbiter
Support:	Über Gitter

Tabelle 1: Deeplearning4J Übersicht

Vorverarbeiten der Bilder, wie zum Beispiel: Schneiden, Skallieren und Drehen. Ein weiteres wichtiges Werkzeug ist der Arbiter. Der Arbiter ermöglicht die Suche nach den sogenannten Hyperparametern, der Netzkonfiguration. Da es sich in der Arbeit um experimentelle Daten handelt, gibt es wenige Anhaltspunkte wie die Hyperparameter, wie zum Beispiel die Anzahl der Neuronen auf den einzelnen Schichten oder der Lernrate, zu setzen sind. Dafür muss eine grobe Modellstruktur, alle variablen Hyperparameter mit einem entsprechenden Wertebereich und einer Suchstrategie angegeben werden. Der Arbiter füllt dann die variablen Hyperparameter mit expliziten Daten, ausgewählt durch die entsprechende Suchstrategie. Dieses Modell wird dann Daten ausgesetzt und probeweise trainiert. Dafür müssen entsprechende Trainings- und Testdaten vorliegen. Nach dem Training ist es möglich die Konfiguration und den Zustand des Netzwerks, die Werte der gewichteten Verbindungen und Schwellwerte, zu speichern.

4.2 DATENERHEBUNG UND VORVERARBEITUNG

Die Beschaffung der Daten wurde mittels eines übersichtlichen Java-Programms realisiert. Dieses Programm bezieht URL aus einer UTF-8 kodierten Text- oder XML-Datei. Dabei liest das Programm die Datei zeilenweise ein. Die URL sind durch einen Zeilenumbruch getrennt. In einer Browserinstanz wird der Reihe nach jede URL aufgerufen und darauf gewartet, dass die Seite fertig ist alle HTML-Elemente zu laden. Wenn die Seite fertig geladen ist, wird ein Screenshot vom sichtbaren Teil der Seite gemacht und lokal unter dem im Programm gekennzeichneten Pfad als PNG-Datei gespeichert. Die Ausmaße der

NAME	PDS	QUELLE	NICHT-PDS
BigDataSample	otto.de	10000	10000
BigDataSample_reduced	otto.de	1000	1000
MixedBigDataSample	otto.de zalando.de	2000	2000
GeneralizationSample	tchibo.de	100	100

Tabelle 2: Datenpakete

Browserinstanz sind auf 1900 Pixel in der Breite und 1000 Pixel in Höhe festgelegt. Aus den Abmaßen der Browserinstanz ergeben sich die Werte der Bilder: 1887*929. Diese sind über alle, auf diese Art gesammelten Bilder gleich. Beim Sammeln von Daten auf diese Weise entsteht Last auf Servern der Shop-Betreiber. Um diese Last minimal zu halten, wurden allgemein gültige Regeln beachtet, so wird zum Beispiel nicht parallelisiert und zwischen den Aufrufen angemessene Pausen gesetzt.

Da eine große Menge an Daten benötigt wird, wurden aus der Liste der 100 umsatzstärksten Online-Shops 2015 drei geeignete Kandidaten ausgewählt, deren Erscheinungsbild sich ähnelt und die eine Sitemap zur Verfügung stellen.

Die 100 umsatzstärksten Online-Shops 2015, Quelle: www.ehi.org, letzter Besuch 28. Januar 2017

Sitemaps werden normalerweise dafür verwendet, um Webcrawlern der Suchmaschinen, zum Beispiel dem Googlebot von Google, einen Plan zu überreichen, wie sie am besten über die Website traversieren. Die ausgewählten Seiten sind:

1. Otto.de
2. Zalando.de
3. Tchibo.de

Die Sitemaps liefern keine konkrete Einteilung, durch den Aufbau der URL lassen sich aber Produktdetailseiten erkennen. Der Rest wird als Nicht-Produktdetailseite eingeteilt oder folgt ebenfalls einem Muster.

Die so entstandenen, verschiedenen Datenpakete sind mit Aufbau und Quelle in Tabelle 2 aufgelistet. Dabei ist das Paket BigDataSample_reduced ein Auschnitt vom Paket BigDataSample. MixedBigDataSample besteht aus einem Auschnitt aus BigDataSample und von Zalando bezogenen Daten. Das letzte, kleinere Datenpaket besteht wieder nur aus Daten einer Quelle. Trotz der eingehaltenen Regeln im Zusammenhang der Datenbeschaffung, wurde hin und wieder Wartungsseiten auf Otto.de und Zalando.de angezeigt. Die Bildschirmaufnahmen der Wartungsseiten sind die einzigen bekannten Probleme in den Datenpaketen.

Webcrawler - Politeness policy, Quelle: en.wikipedia.org, letzter Besuch: 28. Januar 2017

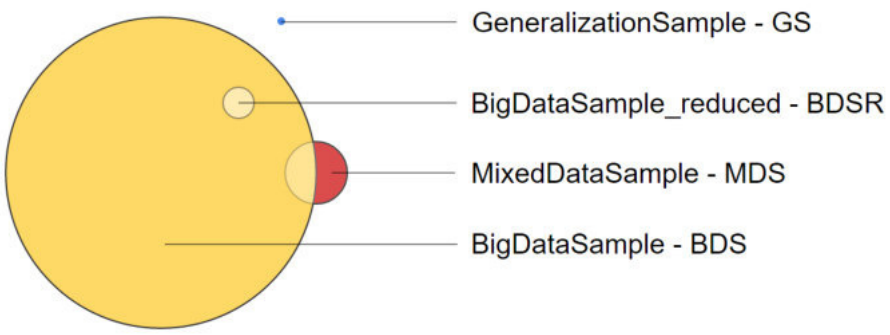

Abbildung 12: Aufbau der Datenpakete in Form eines Venn-Diagramms

Abbildung 12 veranschaulicht die relative Größe und soll einen intuitiven Überblick des Inhalts verschaffen. Für die Verwendung der Bilder müssen diese noch in ein Format überführt werden, in dem das neuronale Netz damit arbeiten kann. Dafür kommt eines der von DL4J gelieferten Werkzeuge zum Einsatz: DataVec. Wie der Name schon vermuten lässt, wird DataVec eingesetzt, um die Bilderdaten in Vektoren umzuwandeln. Zusätzlich lassen sich diese Vektoren weiter manipulieren. Das ist nötig, denn unverarbeitet würden die Bilder in der rohen, vektorisierten Version aus $1887 * 929 * 3 = 5259069$ Informationen bestehen. Um irrelevante Daten auszuschließen und damit die Masse der zu berechnenden Daten zu reduzieren, werden die Ränder links und rechts abgeschnitten. Mit den Daten in reduzierter und für die Netze zugeschnittener Form kann damit trainiert werden.

4.3 ARBITER

4.3.1 *Einstellungen*

Die grobe Netzkonfiguration, die dem Arbiter zum Suchen der idealen Hyperparameter als Ausgangspunkt gegeben wird, ist ein modifizierter, klassischer Aufbau eines Faltungsnetzes in seiner Grundform. AlexNet, VGGnet und auch GoogLeNet [9] [16] [19] folgen einem Muster, was den Aufbau der Netze angeht: Faltungsschicht gefolgt von Subsamplingschicht was n-mal wiederholt wird, gefolgt von einer oder mehrerer voll, verbundener Schichten. Der Aufbau des in dieser Arbeit genutzten Netzes ist an diese Idee angelehnt. Da sich aber mit wachsender Anzahl an Filtern Speicherprobleme ergeben haben, wurde auf den einfachsten Aufbau zurückgegriffen.

Die grobe Struktur beginnt mit einer Eingabeschicht, einer reinen Datenschicht, auf der die zuvor beschrieben Daten in der vorverarbeiteten, vektorisierten Form in das Netz geleitet werden. Der Eingabeschicht folgt eine Faltungsschicht mit einer Kernel-Größe von $4 * 4$ und einer Schrittgröße von "2*2", d.h. pro Schritt gibt es eine Überlappung von zwei Pixeln. Die Anzahl der unterschiedlichen Filter wird

dem Arbiter als Variable aus dem Wertebereich $(20; 28)$ übergeben. Statt einer Subsamplingschicht folgt nun eine weitere Faltungsschicht mit einem $1 * 1$ großen Kernel und einer Schrittgröße von $3 * 3$. Diese Umsetzung ist angelehnt an die Idee von Springenberg u. a. in [17].

Durch die kleinen Filter werden die Daten reduziert, ohne die lokalen Abhängigkeiten zwischen den benachbarten Pixel zu zerstören. Gleichzeitig kommt durch das Hinzufügen von weiteren Filtern mehr Varianz in das Modell, was für mehr Fehlertoleranz sorgt [18]. Auch hier wird die Anzahl der Filter durch einen Wertebereich von $(12; 48)$ an den Arbiter übergeben. In der zweiten Faltungsschicht wird zusätzlich noch Dropout angewendet, mit einer Dropoutrate von $p = 0,5$. Dadurch soll der Overfitting-Effekt unterdrückt werden. Die Daten sind im Bereich der zweidimensionalen Muster einzuordnen und mit heutigen Mitteln einfach zu erkennen, daher ist diese Vorsichtsmaßnahme essentiell für die Arbeit. Abschließend, auch an den klassischen Aufbau angelehnt, folgen zwei voll verbundene Schichten mit jeweils 500 Neuronen und der Ausgabeschicht mit zwei Neuronen für die jeweilige Einteilung in Produktdetailseiten und Nicht-Produktdetailseiten. Über alle Schichten hinweg wurde die Lernrate für die Gewichtungen der Verbindungen und die der Schwellwerte, in einer Variable dargestellt, die vom Arbiter aus einem Wertebereich von $(0,1; 0,0001)$ zu wählen ist. Bis auf Eingabe- und Ausgabeschicht, wird schichtübergreifend ein zusätzlicher Term bei der Berechnung des Fehlers hinzugefügt. Dieser Term basiert auf der L2-Regularisierung und soll der Generalisierung dienen, indem er hilft bei der Minimalisierung des Fehlers große Werte in Gewichtungen zu vermindern und "saubere" Kernel-Karten zu erstellen. Der für die L2-Regularisierung zugrunde liegende Hyperparameter wird mit 0.0001 in der groben Netzkonfiguration des Arbiters initialisiert. Für die Optimierung wird Momentum vom Typ Nesterov mit einer Momentum-Konstante von 0.95 verwendet. Die Verwendung erfolgt in Kombination mit Dropout wie es von Srivastava u. a. vorgeschlagen wurde in [18]. Um die Ausgabe des Netzes in die gewünschte Form zu überführen wird die Softmax-Funktion üblicherweise genutzt [9] [16] [19]. Diese überführt die reellen Werte eines K-Dimensionalen Vektors $\sigma(\mathbf{z})$ in Werte zwischen 0 und 1. Aufaddiert ergeben die Werte des Vektors 1.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

,mit $j = 1, \ldots, K$.

Als letzter Parameter fehlt nur noch die Initialisierung der Gewichtungen. Schwellwerte werden üblicherweise mit Null initialisiert. Es gibt eine Reihe an möglichen Entscheidungen für die Initialisierung. In dieser Arbeit werden die Gewichtungen mit kleinen, sich der Null nähernden, zufälligen Zahlen bestückt.

SCHICHT	AUFBAU
Faltungsschicht:	Kernelanzahl:17 Kernelgröße: $4 * 4$ Schrittgröße: $2 * 2$ Aktivierung: ReLU
Faltungsschicht:	Kernelanzahl: 29 Kernelgröße: $1 * 1$ Schrittgröße: $3 * 3$ Aktivierung: ReLU
Vollverbundene Schicht:	Neuronenanzahl: 500 Aktivierung: ReLU
Vollverbundene Schicht:	Neuronenanzahl: 500 Aktivierung: ReLU
Ausgangsschicht:	Neuronenanzahl: 2 Aktivierung: Softmax
Eigenschaften:	Optimierung: SGD Momentum: Nesterovs Momentumkonstante: 0.95 Lernrate: 0.0013668754917925854 L2Regulierung: 5.0E4

Tabelle 3: Netzmodell SmallConvNet

Bevor der Arbiter jetzt nach einer passenden Konfiguration, der variabel gehaltenen Hyperparameter suchen kann, muss die Suchstrategie festgelegt werden. Random-Search hat sich bei unbekannten Daten bewährt, in [1] wird das Thema von Bergstra und Bengio weiter beleuchtet.

4.3.2 *Resultate*

Bei der Suche nach passenden Konfigurationen kamen zwei Netzmodelle heraus, die mit nur fünf Durchläufen durch die Daten, auch Epochen genannt, gute Resultate lieferten. Die zwei Modelle werden in Tabelle 3 und Tabelle 4.

4.4 TRAINING MIT EARLYSTOPPING

Die Netzmodelle aus Tabelle 3 und Tabelle 4 müssen weiter trainiert werden. Das Problem, das sich regelmäßig beim Training von DNN mit Gradientenabstiegsverfahren zeigt, ist der Overfitting-Effekt. Intuitiv lässt sich dieser unerwünschte Effekt mit dem Auswendiglernen vergleichen. Der Effekt äußert sich beim Training durch die immer kleiner werdenden Werte der Fehlerfunktion, wobei die Werte

SCHICHT	AUFBAU
Faltungsschicht:	Kernelanzahl:20 Kernelgröße: $4 * 4$ Schrittgröße: $2 * 2$ Aktivierung: leaky ReLU
Faltungsschicht:	Kernelanzahl: 20 Kernelgröße: $1 * 1$ Schrittgröße: $3 * 3$ Aktivierung: leaky ReLU Dropoutrate: 0.5
Faltungsschicht:	Kernelanzahl:20 Kernelgröße: $4 * 4$ Schrittgröße: $2 * 2$ Aktivierung: leaky ReLU
Faltungsschicht:	Kernelanzahl: 20 Kernelgröße: $1 * 1$ Schrittgröße: $3 * 3$ Aktivierung: leaky ReLU Dropoutrate: 0.5
Vollverbundene Schicht:	Neuronenanzahl: 500 Aktivierung: leaky ReLU
Vollverbundene Schicht:	Neuronenanzahl: 500 Aktivierung: leaky ReLU
Ausgangsschicht:	Neuronenanzahl: 2 Aktivierung: Softmax
Eigenschaften:	Optimierung: SGD Momentum: Nesterovs Momentumkonstante: 0.95 Lernrate: 0.0002592720230415623 L2-Regulierung: $5.0\mathrm{E}-4$

Tabelle 4: Netzmodell DeepConvNet

der Fehlerfunktion auf der Testmenge gleich oder, im schlechtesten Fall, wieder größer werden.

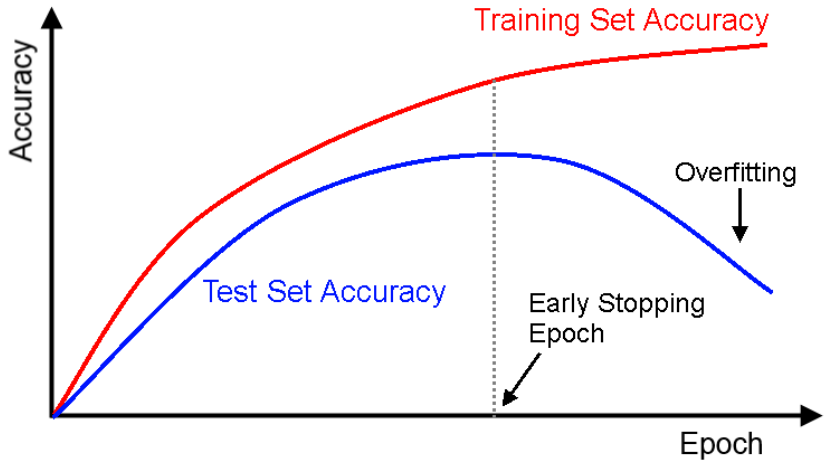

Abbildung 13: Earlystopping und Overfitting,

Abbildung 13
Quelle:
deeplearning4j.org/earlystopping,
letzter Besuch 28.
Januar 2017

Ist das der Fall, spricht man von Overfitting. Das Problem ist, die Anzahl der Trainingsdurchläufe kann von Netz zu Netz stark variieren. Es ist unklar, wann genau ein Netz genug trainiert wurde, um von gelerntem Wissen auf ähnliche Probleme abstrahieren zu können, aber gleichzeitig nicht so viel, als dass es anfängt die Muster zu speichern. In Abbildung 13 wird der Effekt veranschaulicht. Hier wird allerdings auf der vertikalen Achse die Genauigkeit anstelle des Netzfehlers verwendet. EarlyStopping ist eine einfache Möglichkeit dem Auftreten von Overfitting entgegenzuwirken. Nach jedem Durchlauf der Trainingsdaten wird der Fehler auf der Testmenge geprüft. Ist der Wert besser als der vorangegangene, wird eine Kopie des aktuellen Zustands gespeichert. Es ist so eine einfache, aber dennoch effektive Technik Overfitting-Effekte zu vermeiden, dass Geoff Hinton sie in einer seiner Vorlesungen [7] von 2015 “beautiful free lunch” nannte.

5 EXPERIMENTE

5.1 TESTAUFBAU

Die zwei entstandenen CNN Modelle werden mittels EarlyStopping auf die Datenpakete BigDataSample_reduced und MixedDataSample trainiert, getestet und anschließend in unterschiedlichen Experimenten auf Test- und Validierungsdatenpakete validiert. Für die Bewertung der Resultate werden Accuracy, Precision, Recall und F1Score verwendet.

5.1.1 *Kategorisierung*

Die erste Experimentreihe soll zeigen, ob es sowohl mit SmallConvNet als auch mit DeepConvNet möglich ist, in die vorgeschriebenen Kategorien Produktdetailseite und Nicht-Produktdetailseiten verlässlich zu unterteilen. Dafür werden beide auf BigDataSample_reduced trainierten Netzmodelle auf der separierten Testmenge des Pakets überprüft. Im Anschluss werden beide auf MixedDataSample trainierte Netzvarianten auf der zugehörigen Testmenge überprüft. Da mehrschichtige Netzwerke im Vergleich zu flachen Netzen mehr Daten und Epochen benötigen um ein gleiches Niveau an Erfolgschancen zu erreichen, wird erwartet, dass SmallConvNet auf beiden trainierten Datenpaketen mindestens gleich oder bessere F1Score liefert.

5.1.2 *Generalisierung*

Die zweite Reihe an Experimente wird in drei Stufen die Generalisierbarkeit von SmallConvNet und DeepConvNet, mit steigender Varianz in den Validierungsdatenmengen überprüfen. Unter der Generalisierbarkeit ab Stufe zwei wird die Fähigkeit verstanden, Eigenschaften Shop-übergreifend zu erkennen.

Auf Stufe Eins werden die auf BigDataSample_reduced und MixedDataSample trainierten Versionen von SmallConvNet und DeepConvNet validiert. Die Erwartungshaltung zum resultierenden F1-Score der Konstellationen ist ähnlich wie in der ersten Experimentreihe. DeepConvNet hat in beiden Fällen genau die selbe Menge an Daten zum Lernen bekommen wie SmallConvNet, daher wird erwartet, dass SmallConvNet in beiden trainierten Versionen leicht besser aber mindestens genau so gute Resultate liefert.

Auf Stufe Zwei werden die auf BigDataSet_reduced trainierten Versionen der Modelle mittels MixedDataSample validiert. An dieser

Stelle wird erwartet, dass bei beiden Modellen ein starker Fall des F1-Scores zu messen ist, da beide Modelle nur maximal auf 50% der Daten der Validierungsmenge von ihrem Training aus abstrahieren können.

Auf der letzten Stufe des Generalisierungsexperiments werden alle Versionen der trainierten Modelle auf eine völlig unbekannte Datenmenge validiert. Es wird erwartet, dass das Ergebnis verhältnismäßig schlecht ausfällt, aber die in DeepConvNet verwendeten Techniken zum Erhöhen der Fehlertoleranz eingesetzter Techniken messbar werden. Außerdem wird erwartet, dass die Modelle die auf dem Datenpaket MixedDataSet trainiert wurden, bessere Resultate erzielen, da in der dazugehörigen Trainingsmenge, an mehr Daten und eine höhere Anzahl varianter Muster gelernt wird.

5.2 ERGEBNISSE

In Tabelle 5 stehen die Ergebnisse der oben beschriebenen Testfälle. Die Konstellationen aus den jeweiligen Trainingsversionen beider Netzmodelle mit den dazugehörigen F1-Score wird in Abbildung 14 dargestellt.

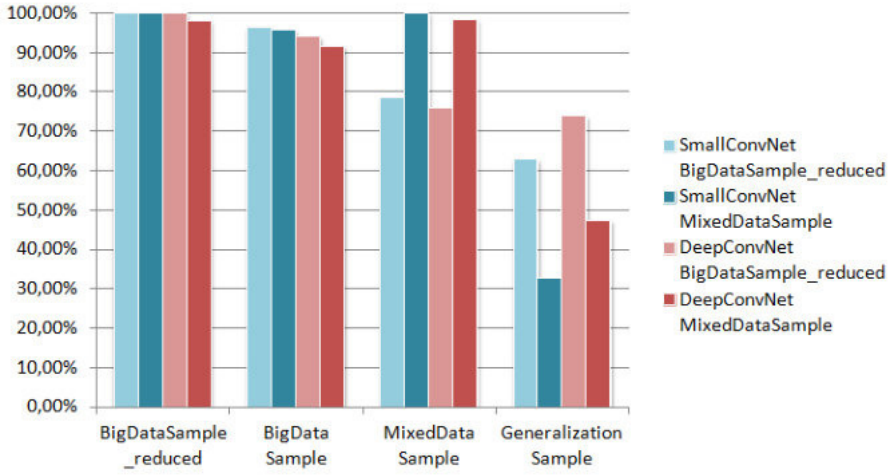

Abbildung 14: Vergleich der F1-Scores

5.3 AUSWERTUNG

Im Folgenden werden im Bezug auf die oben beschriebenen Testreihen die gesammelten Resultate diskutiert.

5.3.1 *Kategorisierung*

Die erste Reihe an Tests sollte primär zeigen, dass es mittels CNN möglich ist Bilder von Online-Shop-Webseiten in Produktdetailseiten

MODELL	BDSR	BDS	MDS	GS
SmallConvNet BigDataSample_reduced	100%	96,34%	78,42%	62,76%
SmallConvNet MixedDataSample	99,95%	95,63%	100%	32,43%
DeepConvNet BigDataSample_reduced	100%	93,83%	75,83%	73,72%
DeepConvNet MixedDataSample	98%	91,5%	98,27%	47,32%

Tabelle 5: Übersicht F1Score

Listing 1: DeepConvNetMixedDataSample auf MixedDataSample

```
Examples labeled as 0 classified by model as 0: 199 times
Examples labeled as 0 classified by model as 1: 1 times
Examples labeled as 1 classified by model as 0: 6 times
Examples labeled as 1 classified by model as 1: 194 times
```

und Nicht-Produktdetailseiten zu kategorisieren. Der erste Versuch aus der Reihe zeigt erwartungsgemäß gute Ergebnisse. Um die Daten nicht zu verfälschen, wurde nur auf einer kleinen Menge an Testdaten geprüft, hier 10% des BigDataSample_reduced, was insgesamt 200 Daten umfasst, 100 Bilder von Produktdetailseiten und 100 Nicht-Produktdetailseiten. Beide auf BigDataSample_reduced trainierte Netzmodelle kategorisierten die 200 Testdaten zu 100% richtig. Im zweiten Versuch wurden die auf MixedDataSample trainierten Netzmodelle auf den Testdaten geprüft. Wie zuvor wurden auf 90% der Daten des Pakets trainiert und auf 10% getestet, das heißt auf 400 Bilder des Pakets. Hier lassen sich lassen sich Unterschiede im beobachteten F1Score sehen. SmallConvNet verhält sich erwartungsgemäß und kategorisiert zu 100% richtig. Bei DeepConvNet mit einem vergleichbaren Wert an Epochen im Training, werden bereits hier geringe, fehlerhafte Abschätzungen erkennbar. Die exakten Abweichungen sind im Listing zu sehen:

Dieser Unterschied im daraus resultierende F1Score lässt sich auf die erhöhten Anforderungen mehrschichtiger Netzmodelle an das Training zurückführen. Mit mehr Daten oder Aufwand in der Optimierung kann man auch hier bessere Ergebnisse erzielen.

Abschließend lässt sich zur ersten Experimentreihe sagen, dass es auch mit größeren Bildern möglich ist sehr gute Resultate in der Kategorisierung mit absehbarem Aufwand zu erzielen.

5.3.2 *Generalisierung*

In der zweiten Reihe an Experimenten wurde der Fokus auf die Generalisierbarkeit der Netzmodelle gerichtet. Stufenweise wurden hier die direkt abstrahierbaren Merkmale reduziert. So waren es auf der ersten Stufe noch Bilder des Shops, auf denen die zwei Modelle trainiert wurden. Auf zweiter Stufe waren dann nur noch 50% der Daten direkt abstrahierbar aus dem Training davor, was bedeutet, dass auf den anderen 50% Muster messen lassen, die allgemeiner bzw. übergreifenden Ursprungs sind. Auf der letzten Stufe gab es keine direkt abstrahierbaren Muster mehr in der Trainingsmenge.

Auf der ersten Stufe wurden die zwei, auf BigDataSample_reduced und MixedDataSample trainierten Netzmodelle, auf das umfangreichste Datenpaket BigDataSample validiert. Alle vier trainierten Versionen der Modelle wurden auf 90% trainiert und jeweils auf 10% getestet. Das bedeutet, dass die zwei Versionen, die auf BigDataSample_reduced trainiert wurden, bereits 1800 Bilder aus BigDataSample aus dem Training kennen. Bei den zwei auf MixedDataSample trainierten Netzmodellen bedeutet das, dass nicht gesagt werden kann, wie anteilig Bilder der Otto.de und Zalando.de Shop-Webseite für die Tests benutzt wurden. Die Einteilung wird von DL4J vorgenommen und basiert auf einem Schlüssel. Vorteil ist, dass solange die Menge die selbe Größe hat, die Elemente durch den Schlüssel gleich ausgewählt werden . Daraus resultierender Nachteil ist, dass negative Effekte, die daraus entstehen können, demnach mit dem selben Schlüssel übertragen werden. Da überall derselbe Schlüssel verwendet wurde, sind die Vergleiche immer noch valide. Der Bereich, der schon aus dem Training bekannten Daten kann für die zwei auf MixedDataSample trainierten Modelle eingegrenzt werden. Da die Menge der Testdaten beim Training 400 war, bedeutet das, dass maximal 2000 Bilder und minimal 1600 Bilder bereits aus dem Training bekannt waren. Diese Daten werden im berechneten F1-Score nicht gesondert berücksichtigt. Aber daraus ergibt sich anteilig der schlechtere F1-Score der Modelle die auf MixedDataSample trainiert wurden. Neben dem übertragenen Fehler aus dem Training ist das Datenpaket nicht zu 100% sauber (siehe Kapitel 4), was weiter Erklärung für den fallenden F1-Score liefert. Das Verhalten ist immer noch im Rahmen der Erwartungen. Auf der zweiten Stufe des Experiments wurden die zwei auf BigDataSample_reduced trainierten Netzmodelle auf MixedDataSample validiert. Dabei liefert SmallConvNet mit 78,42%, im Vergleich zu DeepConvNet mit 75,83% den besseren F1-Score. Da die Testmenge aus dem Training nicht gleich ist, verglichen mit der, die hier zum Einsatz kommt, kann in ähnlicher Form wie in der ersten Stufe des Experiments von einer leichten Fehlerübertragung gesprochen werden. Da beide Netze im Test auf der Untermenge des BigDataSample 100% erreicht haben, kann davon ausgegangen werden, dass auch

Listing 2: DeepConvNetBigDataSample_reduced validiert auf GeneralizationSamplel

```
Examples labeled as 0 classified by model as 0: 100 times
Examples labeled as 1 classified by model as 0: 64 times
Examples labeled as 1 classified by model as 1: 36 times
=========================Scores========================
 Accuracy:  0.68
 Precision: 0.8049
 Recall:    0.68
 F1 Score:  0.7372
=======================================================
```

Listing 3: DeepConvNetMixedDataSample validiert auf GeneralizationSamplel

```
Examples labeled as 0 classified by model as 0: 30 times
Examples labeled as 0 classified by model as 1: 70 times
Examples labeled as 1 classified by model as 0: 35 times
Examples labeled as 1 classified by model as 1: 65 times
=========================Scores========================
 Accuracy:  0.475
 Precision: 0.4715
 Recall:    0.475
 F1 Score:  0.4732
=======================================================
```

hier der Erwartungswert von gegen 50% schon durch das Training erreicht wird. Zieht man diese Kenntnis in die Bewertung ein, so liefert SmallConvNet auf den unbekannten Validierungsdaten 56,84% und DeepConvNet 51,66%. In keinem der beiden Fälle liefern die Netzmodelle derart signifikante Ergebnisse, dass man hätte davon ausgehen können, sie würde übergreifendes Wissen angelernt haben.

Stufe drei der zweiten Experimentreihe liefert erstaunliche Daten. Weder die auf BigDataSample_reduced noch die auf dem MixedDataSet trainierten Modelle können direkt von Mustern abstrahieren. Die intuitive Erwartungshaltung wäre, dass die Modelle die auf dem MixedDataSample-Paket trainiert wurden, bessere Resultate liefern. An dieser Stelle der Arbeit kann nur noch spekuliert werden, was in dem letzten Experiment wirklich passiert. Eine detaillierte Analyse der Situation könnte in einem Folgeexperiment Aufschlus ergeben. Im Folgenden sind die detaillierten Ergebnisse aufgelistet, dabei sind die als 0 gekennzeichneten Bilder, Bilder von Nicht-Produktdetailseiten. Bilder die als 1 gekennzeichnet sind, sind somit Produktdetailseiten.

Einer der oben angerissenen, negativen Effekte, die sich aus der zufällig gewählten Testmenge aus MixedDataSample ergeben können, ist ein Problem im Zusammenhang mit EarlyStopping. Geht man von

Listing 4: SmallConvNetBigDataSample_reduced validiert auf GeneralizationSamplel

```
Examples labeled as 0 classified by model as 0: 100 times
Examples labeled as 1 classified by model as 0: 93 times
Examples labeled as 1 classified by model as 1: 7 times
=========================Scores========================
 Accuracy:  0.535
 Precision: 0.7591
 Recall:    0.535
 F1 Score:  0.6276
=======================================================
```

Listing 5: SmallConvNetMixedDataSample validiert auf GeneralizationSamplel

```
Examples labeled as 0 classified by model as 0: 96 times
Examples labeled as 0 classified by model as 1: 4 times
Examples labeled as 1 classified by model as 0: 100 times
=========================Scores========================
 Accuracy:  0.48
 Precision: 0.2449
 Recall:    0.48
 F1 Score:  0.3243
=======================================================
```

den unwahrscheinlichen Ereignissen aus, dass sich in der Testmenge nur Daten des einen oder des anderen Shops befinden, so kann es beim Training dazu führen, dass EarlyStopping große Overfitting-Effekte nicht erkennt. Somit wird ein Vorteil der größeren Mustervielfalt zu einem speicherblockierenden Nachteil. Auf das Problem aus Stufe drei angewandt könnte es sein, dass beispielsweise der Tchibo.de Anteil der Muster direkt gespeichert. Das würde dazu führen, dass eine ausreichende Fähigkeit zu Generalisieren auf seiten des Otto.de Anteils der Muster entsteht, aber im Ganzen die die Fähigkeit verliert auf neue Muster zuversichtlich Entscheidungen zu treffen. Dazu sei hinzuzufügen, dass es sehr unwahrscheinlich ist, dass in der Testmenge nur Bilder von einem Shop vertreten sind.

6 ZUSAMMENFASSUNG UND AUSBLICK

6.1 ZUSAMMENFASSUNG

Primäres Ziel dieser Arbeit war es, mittels DNN auf Bildern von Onlineshop Webseiten eine Einteilung in Produktdetailseiten und Nicht-Produktdetailseiten vorzunehmen. Die erste Experimentreihe lief sehr erfolgreich. So erreichte das fünfschichtige Netzmodell SmallConvNet mit nur wenigen Trainingsepochen und ohne weitere Optimierung einen Netzfehler, der gegen Null geht. Erwartungsgemäß erreichte das siebenschichtige Netzmodell DeepConvNet mit mehr Aufwand in Epochen und Optimierung einen vergleichbaren, aber immer noch größeren Netzfehler. Innerhalb dieser Arbeit lässt sich nur vermuten, dass das große Potential tieferer Netzmodelle erst mit zunehmender Datenmenge und Trainingsepochen stärker in Erscheinung tritt. Sekundäres Ziel war es, eine Einteilung in Produktdetailseiten und Nicht-Produktdetailseiten Shop-übergreifend zu untersuchen. Abgesehen von einer, gegen die intuitive Haltung, auftretenden Erscheinung in der letzten Stufe der zweiten Experimentreihe, konnten keine nennenswerten Erfolge erzielt werden.

Trotz der schlechten Ergebnisse der Generalisierungsexperimente sind Faltungsnetze ein mächtiges Werkzeug, das mit der richtigen Netzkonfiguration, genug Daten und Zeit für Training und Optimierung sehr gute Resultate in der Mustererkennung liefert.

6.2 AUSBLICK

Auf Basis der Ergebnisse dieser Arbeit können folgende Empfehlungen ausgesprochen werden. Da keine nennenswerte Generalisierung Shopseiten-übergreifend beobachtet werden konnte, kommen einige mögliche Wege für zukünftige Arbeiten infrage:

1. Erstens, es wird weiter nach einem Netzmodell gesucht, was die Fähigkeit besitzt, übergreifend zu generalisieren. Dafür sollte man dem üblichen Trend folgen und mehr Daten und Schichten verwenden. Es wird außerdem der Einsatz eines Schwellwert-Neurons als Ausgabe-Neuron empfohlen, wie es Vaillant, Monrocq und Le Cun vorschlagen haben in [20]. Dadurch ergeben sich weitere Vorteile. Für die Klasse der Nicht-Produktdetailseiten können übliche Techniken verwendet werden, um die Datenmenge künstlich zu vergrößern. So kann der Fokus auf das Erheben von Produktdetailseiten gerichtet werden. Außerdem kann die berechnete Einteilung durch die Angabe des Schwell-

werts geregelt werden. Das kann sich als sehr nützliche Stellschraube erweisen, um bei dem Einsatz des Netzes später, die Qualität der Ergebnisse zu verbessern oder eine größere Toleranz zuzulassen.

2. Zweitens, wäre es denkbar die Einteilung in Produktdetailseite und Nicht-Produktdetailseite beispielsweise umzuformulieren in die Fragestellung "Wird ein Checkout-Button im Bild erkannt?" So könnte der Fokus auf ein spezielles Dialogelement gerichtet werden, was eine Produktdetailseite implizieren würde.

3. Drittens, wäre es möglich, wenn zuvor genannte Ansätze sich als Sackgasse erweisen, ein Werkzeug zu entwickeln, was die Datenerhebung per Hand vereinfacht. Dann wäre es auch denkbar Modelle aus dieser Arbeit wiederzuverwenden. In dem Zusammenhang wäre es auch ratsam zu überprüfen wie viele Daten notwendig sind, um ein vergleichbares Ergebnis zu dem vorliegenden Kategorisierungsexperiment zu erhalten.

Des Weiteren wird bei Folgearbeiten mit DL4J empfohlen, die Erscheinung aus der zweiten Experimentreihe, dritter Stufe weiter zu analysieren. Es ist denkbar, dass es Toolkit-intern Probleme gab, die zu dem Problem führten.

Abschließend ist zu sagen, dass durch das gewaltige Interesse der Welt an dem Thema neuronale Netze und speziell Faltungsnetze, sehr viele Arbeiten in dem Bereich entstehen, die sich in Zukunft auch für die Idee dieser vorliegenden Arbeit als nützlich erweisen können. Eine Arbeit seitens Google beispielsweise [21] beschreibt die Entwicklung eines neuronalen Netzes, das die Suche nach dem idealen Netzmodell übernimmt. Rückblickend auf diese Arbeit war die Suche nach einem lernfähigen Netzmodell die zeitintensivste Aufgaben. Der Ansatz von Google kann dieses Problem lösen und ermöglicht es, mehr Aufwand in saubere Daten und die zugehörigen Ausgabe investieren zu können.

[1] James Bergstra und Yoshua Bengio. "Random Search for Hyper-Parameter Optimization." In: *Journal of Machine Learning Research* 13 (2012), S. 281–305. URL: http://dblp.uni-trier.de/db/journals/jmlr/jmlr13.html#BergstraB12.

[2] Andre Esteva, Brett Kuprel, Roberto Novoa, Justin Ko, Susan Swetter, Helen Blau und Sebastian Thrun. "Dermatologist-level classification of skin cancer with deep neural networks". In: (2017). URL: http://www.nature.com/nature/journal/vaop/ncurrent/full/nature21056.html#tables.

[3] *Google's AI beats world Go champion in first of five matches*. http://www.bbc.com/news/technology-35761246. Accessed: 2017-01-17.

[4] Benjamin Graham. "Fractional Max-Pooling". In: *CoRR* (2014). URL: http://arxiv.org/abs/1412.6071.

[5] Tobias Hassenklöver. *Klassifikation hochvarianter Muster mit Faltungsnetzwerken*. 2012.

[6] Donald O. Hebb. *The organization of behavior*. 1949.

[7] Geoff Hinton, Yoshua Bengio und Yann LeCun. *Deep Learning NIPS 2015 Tutorial*. http://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf. Accessed: 2017-01-28.

[8] David. Jacobs. *Correlation and Convolution*. 2015. URL: https://arxiv.org/pdf/1412.6071v4.pdf.

[9] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Hrsg. von F. Pereira, C. J. C. Burges, L. Bottou und K. Q. Weinberger. Curran Associates, Inc., 2012, S. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[10] U. Lämmel und J. Cleve. *Künstliche Intelligenz*. Hanser, 2008. ISBN: 9783446413986. URL: https://books.google.de/books?id=vPmLVla95LEC.

[11] Carlo Michaelis. *Biologische Psychologie*. 2011. URL: http://uni.carlo-michaelis.de/lib/exe/fetch.php/biologische-psychologie.pdf.

[12] Marvin Minsky und Seymour Papert. *Perceptrons : an introduction to computational geometry*. Cambridge (Mass.), London, 1969.

[13] Vinod Nair und Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines.” In: *ICML*. Omnipress, 2010, S. 807–814.

[14] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain”. In: *Psychological Review* (1958), S. 65–386.

[15] David E. Rumelhart, James L. McClelland und CORPORATE PDP Research Group, Hrsg. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986. ISBN: 0-262-68053-X.

[16] Karen Simonyan und Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[17] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox und Martin A. Riedmiller. “Striving for Simplicity: The All Convolutional Net”. In: *CoRR* abs/1412.6806 (2014). URL: http://arxiv.org/abs/1412.6806.

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever und Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *J. Mach. Learn. Res.* 15.1 (2014), S. 1929–1958. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2627435.2670313.

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke und Andrew Rabinovich. “Going Deeper with Convolutions”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2015. URL: http://arxiv.org/abs/1409.4842.

[20] R. Vaillant, C. Monrocq und Y. Le Cun. *Original approach for the localization of objects in images*. Aug. 1994.

[21] Barret Zoph und Quoc V. Le. “Neural Architecture Search with Reinforcement Learning”. In: *CoRR* abs/1611.01578 (2016). URL: http://arxiv.org/abs/1611.01578.

VERSICHERUNG ÜBER SELBSTSTÄNDIGKEIT

Diese Arbeit wurde von mir selbständig verfasst und in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt. Ich habe keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet.

Jena, Januar 2017

Sebastian Ebert