Technology
Arts Sciences
TH Köln

Technische Hochschule Köln
Fakultät für Informatik und Ingenieurwissenschaften

B A C H E L O R A R B E I T

Auswirkung von Farbnormalisierung der Trainingsdaten auf künstliche neuronale Netze

Vorgelegt an der TH Köln
Campus Gummersbach
im Studiengang
Medieninformatik

ausgearbeitet von:
Torben Krause
(Matrikelnummer: 11106885)

Erster Prüfer: Professor Dr. Martin Eisemann
Zweiter Prüfer: Professor Dr. Matthias Böhmer

Gummersbach, im Juli

Vorwort

Danksagung

Abstract

Abbildungsverzeichnis

Tabellenverzeichnis

Inhaltsverzeichnis

1 Einleitung

In keinem Zeitabschnitt waren digitale Technologien so stark im Fokus wie heutzutage. Gerade der Bereich selbstständig lernenden Computer werden in der Forschung untersucht und vorangetrieben. Auch bekannt unter dem Namen *Deep Learning* oder künstliche Intelligenz (KI), ist mit diesen Bezeichnungen meist der Bereich der künstlichen neuronalen Netze gemeint. Solche Netze können durch eingeführte Datensätze oder Regeln lernen. Dadurch werden sie in dem trainierten Bereich *intelligent*. Durch genügend Beispiele in den Trainingsdaten können sie Hypothesen aufstellen. Die Faktoren, welche beim Training wichtig sind, stellen zum einen die Menge, wie auch die Qualität der Trainingsdaten dar, zum anderen die Zeit, in welcher das neuronale Netz trainiert wird. Durch steigende Rechenleistung kann das Verfahren beschleunigt werden, da normalerweise viel Zeit für das Training benötigt wird. In der Vergangenheit konnten die Forschungen durch zu langsame Hardware nicht effektiv weiter geführt werden. Mit der Leistung heutige CPUs und GPUs ist dies ohne Probleme möglich.

1.1 Problemstellung und Ziele

Beim Erstellen eines neuronalen Netzes kann es auch zu Schwierigkeiten kommen. Probleme können beispielsweise durch schlechte Datensätze auftreten. Im Bereich der Objekterkennung können dadurch Probleme, wie fehlerhafte Zuordnungen auftreten. Datensätze bestehen üblicherweise aus mehreren Hunderttausend Bildern und bei Vortrainierten um die Tausend Bilder. Die richtige Ausleuchtung ist ein wichtiger Aspekte bei der Generierung der Trainingsbilder, da schon kleine Veränderungen der Lichtverhältnisse die Farben der Objekte verändern kann. Ein und dasselbe Objekt kann dadurch in vielen verschiedenen Farbvariationen auftreten. Gerade bei einem Datensatz mit wenig Bildern, kann das zu fehlerhaften Prognosen führen. Häufig kann eine konstante Ausleuchtung nicht gewährleistet werden, weil die Bilder in der freien Umwelt erstellt werden und eine Ausleuchtung zu umständlich wäre. Das bedeutet das dieses Problem hauptsächlich in der Nachbereitung der Bilder angegangen werden kann.

1.2 Erläuterung der These

Für solche Probleme bietet die Bildverarbeitung einige Lösungsansätze, welche von der Theorie aus, eine konstante Farbgebung und dadurch eine Erhöhung der Genauigkeit bringen könnten. Ob es in der praktischen Ausführung die erwarteten Ergebnisse erzielt, soll getestet werden. Deswegen sollen in dieser Arbeit verschiedene Verfahren der Farbnormalisierung von Bildern auf die Trainings- und Testdatensätze angewendet und auf

verschiedene künstliche neuronale Netze trainiert werden. Beim Vergleich der Trainingsergebnisse soll herausgestellt werden, ob das Normalisieren der Daten einen positiven Einfluss auf die Genauigkeit hat und welche Unterschiede die Normalisierung-Verfahren aufweisen.

1.3 Anforderungen

Bei dem geplanten Vorhaben, welches diese Arbeit thematisiert, ist es wichtig Anforderungen an das neuronale Netz, dem Datensatz und den verwendeten Algorithmen zu formulieren, um sicher zu stellen, das keine vermeidbaren Probleme auftreten. Zunächst werden Trainingsdaten benötigt, mit welchen die neuronalen Netze trainiert werden. Viele wissenschaftliche Arbeiten zeigen, dass eine große Menge an Daten benötigt wird, um ein gut funktionierendes neuronales Netz zu entwickeln. Im Schnitt werden 50.000 Trainingsbilder pro Klasse benötigt. Diese Menge an Daten kann im Anbetracht der verfügbaren Zeit nicht für drei verschiedene Datensätze generiert werden. Da die benötigten Ressourcen nicht zur Verfügung stehen, soll eine Möglichkeit gefunden werden, gute Ergebnisse mit einer geringeren Menge an Trainingsdaten zu realisieren. Um genügend Vergleichswerte untersuchen zu können, sollen mehrere Datensätze mit unterschiedlichen Klassen genutzt werden. Um sicher zu stellen, dass die Datensätze zum Vorhaben geeignet sind, werden diese für die Arbeit erstellt und ausgesucht. Da diese für die Arbeit geeignet sein müssen, ist es wichtig, eine hohe Qualität an Daten zu verwenden.

Die Trainingsdaten sollen unter folgenden Anforderungen erstellt werden:

1. Ein qualitativ hochwertiger Datensatz sollte nicht unordentlich sein, da das Reinigen der von Daten lange dauern kann.

2. Der Datensatz sollte nicht zu große Daten enthalten, da sie den Trainingsfortschritt zurückhalten und mehr Ressourcen benötigen.

3. Beim Generieren der Trainingsbilder sollte darauf geachtet werden saubere und gut ausgeleuchtete Aufnahmen zu machen.

4. Das Ziel des Datensatzes sollte gut Definierte werden.

Damit herausgestellt werden kann, wie sich bestimmte Normalisierungsverfahren auf die Ergebnisse des Trainings auswirken sollen mehrere Verfahren ausgewählt werden. Diese Verfahren müssen, dafür sorgen gewisse Farbkonstanz zu erreichen.

1.4 Struktur der Arbeit

Diese Arbeit ist so aufgebaut, das zunächst alle notwendigen Informationen über künstliche neuronale Netze aufgeführt und erklärt werden. Dabei werden auch die unterschiedlichen Schichten welche durchlaufen werden aufgeführt und beschrieben. Im Weiteren wird auf die Bildverarbeitung eingegangen. Dabei soll zunächst vermittelt werden wie digitale Bilder entstehen und aus welchen Komponenten diese zusammengesetzt sind. Außerdem werden die Farbnormalisierungsverfahren, welche verwendet werden erklärt und beschrieben.

In der zweiten Hälfte der Arbeit wird es um die Durchführung des praktischen Teils der Arbeit gehen. In diesem wird zunächst die Grundlage aufgeführt, auf welcher das Training durchgeführt wurde. Die verwendeten Farbnormalisierungsverfahren werden zusätzlich in der technischen Umsetzung erläutert. Zum Schluss kommt es zur Durchführung des Trainings der verschiedenen Netze, welche daraufhin verglichen und ausgewertet werden. In einer Diskussion wird nun das Ergebnis der Auswertung anhand der angeführten These evaluiert und ein Fazit der Arbeit getroffen.

2 Theoretische Grundlagen

Das folgende Kapitel soll eine kurze und leicht verständliche Einführung sein. In den nächsten Abschnitten soll es um die Themen der künstlichen neuronalen Netze, der Aufbau digitaler Bilder, sowie die Bildverarbeitung gehen. Zunächst beginnt der kommende Abschnitt mit einigen Schwerpunkten der künstlichen neuronalen Netze und einer kurzen Einführung in die künstliche Intelligenz. Dabei werden hauptsächlich für die Arbeit relevante Grundlagen vermittelt und sind deswegen nicht vollständig.

2.1 Künstliche Intelligenz

Künstliche Intelligenz wird heute immer mehr von Firmen und Wissenschaft genutzt. Erst seit wenigen Jahren nutzen große Konzerne diese Technologie für sich, in Form von beispielsweise Vorhersagen der Kundenverhalten, Klassifikation von Bildern, die Spracherkennung in den heutigen Smartphones wird von derartigen Algorithmen gesteuert. Mit künstlicher Intelligenz beschreibt man solche Computer, welche gewisse Aufgaben erledigen, ohne ausdrücklich dafür programmiert worden zu sein. Dabei werden Informationen anhand von Vergleichsdaten dem System zugeführt, wodurch er sich Eigenschaften und Strukturen merkt und durch diese Rückschlüsse und Prognosen ziehen kann. Eine gute Beschreibung geben dabei die Autoren Rich und Knight, in dem sie schreiben:

Das Studium des Problems, Computer dazu zu bringen, Dinge zu tun, bei denen ihnen momentan der Mensch noch überlegen ist. - (Rich und Knight 1991)

Ein gutes Beispiel in welchem man das Potenzial von künstlicher Intelligenz sehen kann, ist die Spiele KI *AlphaGo*. Diese KI wurde von Google entwickelt und ist auf das Spielen des Brettspiels Go trainiert. So konnte *AlphaGo* in einem Match 2016, den damaligen Weltmeister in Go *Lee Sedol* schlagen [Dee16]. In vier von fünf spielen gewann der Computer. Solche Ergebnisse konnten bis zu diesem Zeitpunkt mit anderen Methoden nicht erzielt werden, da es für damalige Algorithmen zu viele mögliche Spielzüge gab. Durch Deep Learning konnte der Computer von allein, ohne feste Algorithmen, besser als der Weltmeister werden. Anhand der Definition von Rich kann man feststellen, dass *AlphaGo* im Bereich des Spieles Go als intelligent zählt, da sie dem besten Go-Spieler der Welt deutlich überlegen war.

2.2 Neuronale Netze

Um das Konzept hinter den künstlichen neuronalen Netzen besser verstehen zu können, betrachten wir zunächst wie der Lernprozess eines Menschen abläuft. Menschliches Lernen ist das Verändern der eigenen Verhaltensstrukturen und die Folgen individueller Erfah-

rungen [HE16]. Der Lernprozess, kann sowohl bewusst, so wie auch unbewusst unser Verhalten verändern. Menschen adaptieren also die eigenen als auch fremde Erfahrungen zu ihren Gewohnheiten. Diese Prozesse Finden im Gehirn statt, welches aus vielen Nervenzellen besteht. Diese Nervenzellen werden Neuronen genannt und sind untereinander verknüpft, weshalb auch von einem neuronalen Netz gesprochen wird. Maschinelles Lernen nimmt sich also die Funktionsweise des menschlichen Gehirns zum Vorbild [Ert13].

Menschliches lernen

Wie bereits erwähnt, ist der Aufbau und Funktionsweise von künstlichen neuronalen Netzen, mit der des menschlichen Gehirns abgeleitet. Das Gehirn besteht aus ungefähr 10^{11} Nervenzellen, welche untereinander verknüpft sind [Ert13, 265ff.]. Für den Lernprozess sind demnach viele Neuronen nötig. In der Abbildung kann man den wesentlichen Aufbau eines Neurons sehen.

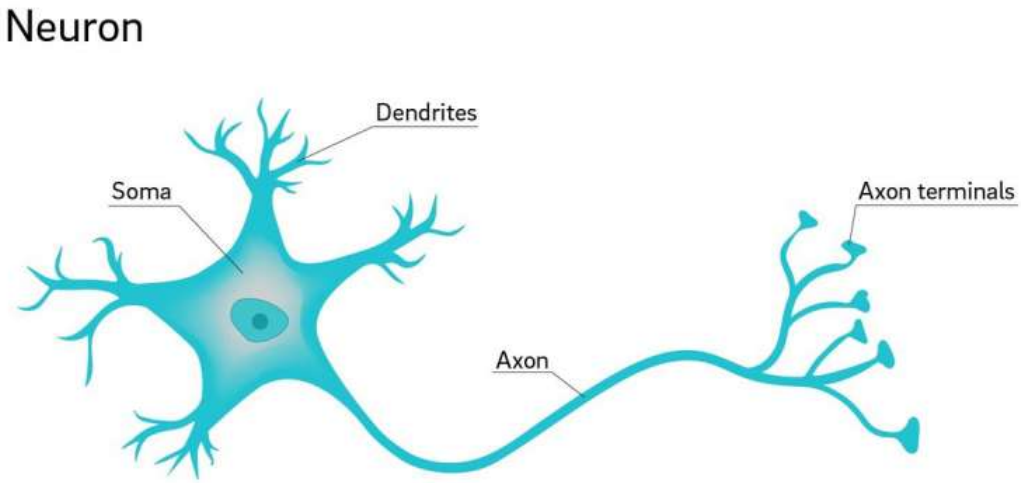

Abbildung 1: *Aufbau eines Neurons [CSD18]*

In diesem vereinfachten Modell eines Neurons lassen sich die wesentlichen Komponenten darstellen. Das Neuron selber besteht aus einem Zellkörper (Soma), einem Axon, den Dendriten auf der linken Seite und den Synapsen(Axon Terminals) auf der rechten Seite. Eine Dendrite stellt mit der Synapse eine Verbindung zu anderen Neuronen her. Diese senden elektrische Impulse an die Soma weiter, welche verarbeitet werden. Wenn die Spannung einen gewissen Schwellwert übersteigt, sendet die Soma einen elektrischen Impuls über das Axon. Jede Synapse sendet das Signal an die Dendriten des nächsten Neurons weiter. Die Stärke des Impulses und der Schwellwert, ist bei jedem Neuron unterschiedlich und verändern sich ständig. Der Schwellwert des Neurons ist niedriger, desto öfter es aktiv ist

und sinkt wenn es selten aktiv ist [ST13]. Die aufgeführte Funktionsweise beschreibt nur oberflächlich das Verhalten eines neuronalen Netzes und soll hauptsächlich der weiteren Verständlichkeit dienen.

Perzeptron

Die Grundlage für den mathematischen Ablauf künstlicher neuronaler Netze wurde von McCulloch und Pitts in ihrem Buch *A logical calculus oft he ideas immanent in nervous activity* erklärt. Auf diesen Grundlagen simulierte 1953 der Psychologe und Informatiker Frank Rosenblatt das erste künstliche Neuron, welches auch als Perzeptron bekannt wurde. In der Abbildung2 wird der Aufbau eines solchen Perzeptrons dargestellt.

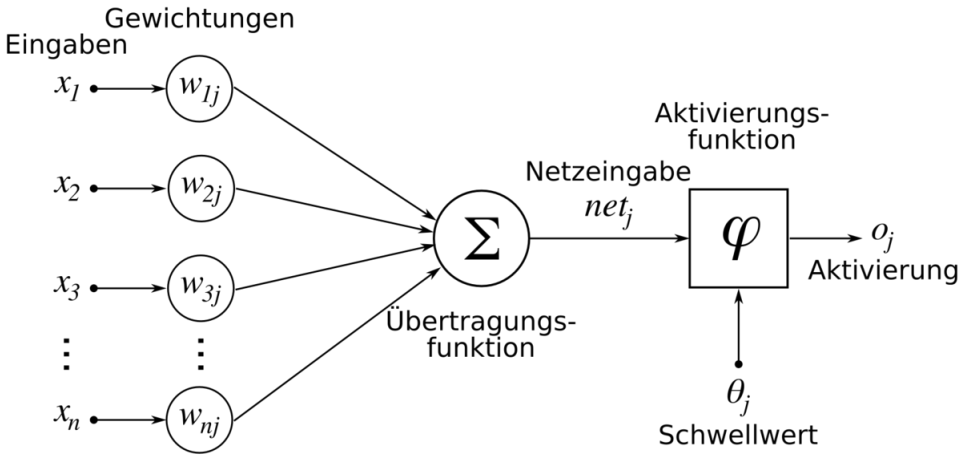

Abbildung 2: *Schematische Modellierung eines Perzeptrons nach Rosenblatt [Wik19]*

Als Eingabewert bekommt das Perzeptron ein Gewicht w, welches mit den Eingangswert x multipliziert wird. Das Ergebnis dieser beiden Werte, kann als lineare Funktion beschrieben werden.

$$f(x) \quad = \quad w * x \tag{1}$$

Das Ergebnis der beiden Werte wird an eine Aktivierungsfunktion übergeben. Die Aktivierungsfunktion oder auch Schwellwert genannt, gibt ein Signal weiter, sobald dieser Wert überschritten wird.

$$g(\sum_{i=0}^{n} x_i * w_i) \tag{2}$$

Als Aktivierungsfunktion wird häufig die Sigmoid-Funktion verwendet, ist aber auch mit anderen linearen und nicht linearen Funktionen möglich. Zusätzlich wird dem Ergebnis der Funktion der *Bias* addiert.

$$g((\sum_{i=0}^{n} x_i * w_i) + b \tag{3}$$

Der *Bias* ist für das Verringern von Konvergenz-Problemen, welche bei der Approximation der Funktion auftreten können. Der Bias hat ein Wert von 1 und hat ein Gewicht, welches unabhängig der Eingabewerte ist.

Maschinelles Lernen möchte unter anderem Maschinen in die Lage versetzten Objekte, Strukturen oder auch Abläufe anhand von vorher beigebrachten Beispielen eigenständig zu erkennen. Eine Methode um dieses Ziel erreichen zu können sind die künstlichen neuronalen Netze. Diese werden mit Datensätzen des jeweiligen Schwerpunktes trainiert. Es gibt vier grundlegende Methoden ein solches Netz zu trainieren.

Überwachtes lernen
Das überwachte Lernen funktioniert hauptsächlich mit Datensätzen, welche Eingabedaten sowie Ausgabedaten enthalten. Das künstliche neuronale Netz versucht bei Eingabe der Daten, eine Funktion aufzustellen, welche eine Hypothese darstellt. Diese beschreibt um, was es sich bei dem Datensatz handelt. Die Genauigkeit dieser Hypothese kann durch den Vergleich der enthaltenden Ausgabedaten ermittelt werden. Wenn die Möglichkeiten der Ausgaben endlich sind, handelt es sich um eine Klassifizierung (Bsp.: Milch, Orangensaft, Wasser, etc.). Sollten nur zwei Ausgabewerte möglich sein, ist es eine boolsche bzw. binäre Klassifizierung. Eine Zahl als Ausgabe wird als Regression bezeichnet.

Nicht überwachtes lernen
Die zweite Methode wie ein künstliches neuronales Netz trainiert werden kann, ist eine gegenteilige Herangehensweise. Hier werden lediglich die Eingabedaten übergeben. Das neuronale Netz versucht selbstständig auftretende Muster, in den Daten zu erkennen. Als Beispiel könnte man die sinnvolle Zuordnung von Bildern anhand ähnlicher Muster nehmen. Ähnliche Datensätze werden in Kategorien aufgeteilt, jedoch nicht benannt.

Halb überwachtes Lernen
Oft werden die beiden Methoden des überwachten und unüberwachten Lernen kombiniert, so hat man meist einen kleinen Datensatz mit Ein- und Ausgabedaten, welche aber nicht

explizit wahr sein müssen und einen Teil nur mit Eingabeinformationen. Diese Methode wird häufig bei Umfragen verwendet, wo nicht klar ist, ob der Proband wahrheitsgemäß antwortet. Dabei kann ein systematischer Fehler entstehen. Daraus ergibt sich eine Mischung aus überwachten lernen mit systematischen Fehler und unüberwachten lernen, bei welchem das Netz nach häufig auftretenden Mustern sucht.

Verstärktes lernen

Das verstärkte Lernen arbeitet auf einem Prinzip, mit welchem richtige Entscheidungen belohnt und falsche bestraft werden. Ein bekanntes Beispiel ist ein Netz, welches virtuell eine Rennstrecke ohne Fehler (nicht von der Strecke abkommen) beenden muss. Bei jedem Fehler muss von vorne begonnen werden, bis die Strecke beendet wird. Ein besonders bekanntes Beispiel ist die GO-KI *AlphaGo* [Dee16], welche nur durch die Informationen der Spielregeln trainiert wurde. Die KI wurde in diesem Spiel so gut, das sie den Weltmeister *Lee Sedol* 2016 in vier von fünf Partien GO besiegte.

Künstliche neuronale Netze

Nachdem etwas auf die Grundlagen die verschiedenen Lernverfahren eingegangen wurde, soll in diesem Abschnitt der Aufbau künstlicher neuronaler Netzes und die Logik fokussiert werden. Der mathematische Ablauf der Schichten wird in einem späteren Unterkapitel behandelt.

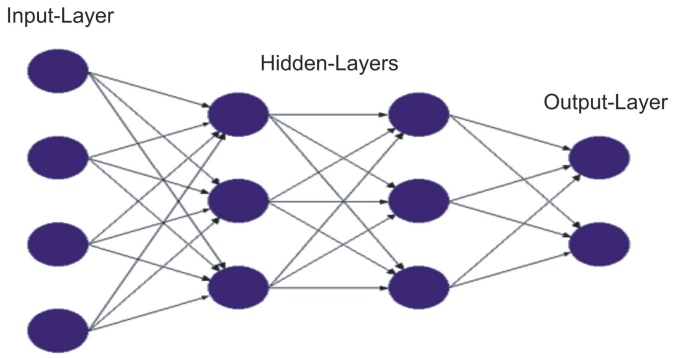

Abbildung 3: *Aufbau eines künstlichen neuronalen Netzes [Ang18]*

Der Aufbau von klassischen künstlichen neuronalen Netzen kann grundsätzlich in drei Bereiche unterteilt werden. Die erste Schicht stellt der *Input Layer* da. Dieser besteht wie alle folgenden Schichten aus mehreren Neuronen. In diese Schicht werden die Trainingsdaten eingegeben. Jeder wert eines Datensatzes wird mit einem Neuron verbunden. Die Neuronen besitzen eine Gewichtung und einen Schwellwert. Wenn der Schwellwert durch den Eingabewert erreicht wird, sendet das Neuron das Ergebnis seiner Berechnungen an das Neuron der nächsten Schicht weiter. Die folgenden Schichten werden *Hidden Layer*

genannt. Die Anzahl der *Hidden Layer* ist variabel und kann so viele Schichten enthalten wie benötigt. Diese Schichten werden auf mehrere verschiedene Merkmale der Eingabedaten trainiert. Dabei werden in frühen Schichten zunächst grobe Strukturen erkannt, welche nach jeder Schicht immer feiner werden. Die letzte Schicht des *Hidden Layer* übergibt seine Informationen an den *Output Layer*. Dieser stellt durch die erhaltenen Daten eine Hypothese auf, in welcher er beschreibt, wie er den Datensatz zuordnet. Dabei gibt das Netz an, zu welcher Wahrscheinlichkeit die Hypothese richtig sein sollte.

Faltende neuronale Netze

Convolutional neural Networks oder auch faltende neuronale Netze sind anders als das KNN, an das Konzept des menschlichen Sehens entworfen [SCL12] und für das Verarbeiten von matrixartigen Datensätzen konzipiert. Dazu zählen beispielsweise Bildaufnahmen [Goo+16]. Die erste Schicht besteht aus den *Convolutional Layer*. Bei dieser Schicht wird über die Pixel des Eingabebildes eine Schablone gelegt und vertikal wie auch horizontal verschoben. Diese Schablone hat eine ungerade quadratische Abmessung ($3x3$, $5x5$, $7x7$) und eigene Gewichtungen. Diese Schablone bildet ein Skalarprodukt der unterliegenden Pixel und der Gewichtung. Dabei werden die Strukturen hervorgehoben. Diese Schicht kommt mehrmals in einem Netz vor.

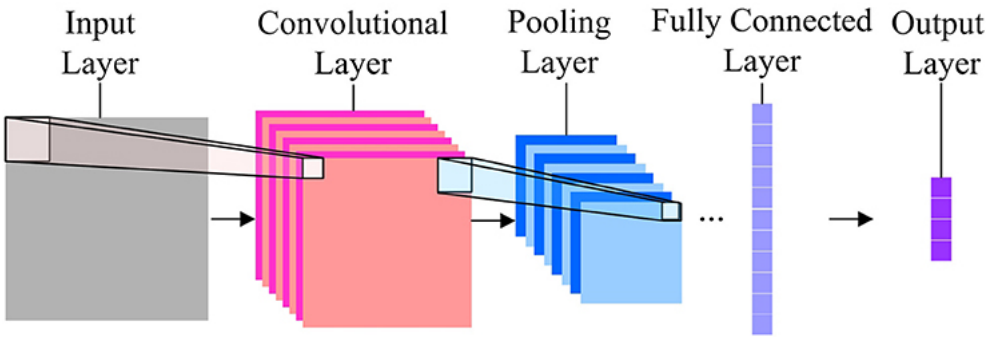

Abbildung 4: *Aufbau eines Convolutional neural Network [Pen+16]*

Auf den *Convolutional Layer* folgt immer der *Pooling Layer*. Dieser ist für die Komprimierung des Eingabebildes zuständig, damit im späteren Verlauf Rechenleistung gespart werden kann. Auch diese Schicht besteht aus einer Schablone welches über die Pixel des Bildes gelegt werden. Hierbei werden die Farbwerte der unterliegenden Pixel verarbeitet. Diese werden zu einem wert zusammengefasst und übergeben. Nachdem alle *Convolutional* und *Pooling Layer* durchlaufen wurden, werden die Daten an die letzte Schicht (*Fully Connected Layer*) übergeben. Der *Fully Connected Layer* besteht wie beim KNN beschrieben, aus mehreren Perzeptronen-Schichten und gibt die Informationen an den *Output Layer* weiter, welcher schlussendlich eine Hypothese mit zugehöriger Wahrscheinlichkeit ausgibt.

Loss Funktion
Bei jedem Trainingsschritt versucht das neuronale Netz, eine Hypothese aufzustellen. Damit das Netz Fortschritte im Lernen macht, ist das sogenannte, Backpropagation zuständig, welche auf der Basis des Loss Graphen arbeitet. Die Loss Funktion berechnet den Fehler, welcher das Netz in seinem aktuellen Zustand in der Berechnung begeht, also die Differenz zwischen dem Ist- und Sollwert der Ausgabe. Beispiel einer solchen Funktion ist die euklidische Loss-Funktion

$$E_{Euclid} = \frac{1}{2N}\sum_{n=1}^{N} ||\hat{f}_n - f_n|| \binom{2}{2} \tag{4}$$

dabei stellt

$\hat{f}E[-\infty, +\infty]$ den berechneten Ist-Wert und
f den Sollwert dar

Das Ergebnis der euklidischen Funktion ist das Maß für den Grad der Abweichung von ist und Sollwert.

Rückpropagierung
Die Rückpropagierung oder auch *Backpropagation* [Ert13] ist unter anderem Teil des überwachten Lernens, und beinhaltet das Lernen aus Fehlern. Dadurch das bei einer Klassifizierung Datensätze mit einem Zielwert verwendet werden. Kann das Netz bei einer Zuweisung überprüfen, ob das Ergebnis zu dem Zielwert passt. Dabei wird aus der Ausgabe und dem Zielwert ein Fehler (Loss) berechnet. Sollte das Ergebnis zu weit vom Zielwert abweichen, wird überprüft, welches Perzeptron den größten Einfluss auf die Fehlzuweisung verursacht hat. Dieses Perzeptron wird in seiner Gewichtung angepasst, um den Fehlerwert zu verringern [Goo+16].

2.2.1 Funktionsweise künstlicher und Faltender neuronaler Netze

Die oben aufgeführten Netzarten haben einen unterschiedlichen Aufbau und bestimmte Einsatzgebiete. Das Convolutional neural Network ist beispielsweise auf das Verarbeiten von Bilddaten optimiert und Künstliche neuronale Netze eher für serielle Daten. In der Struktur sind die Netzarten ähnlich aufgebaut. Das CNN hat neben den üblichen Neuronenschichten zwei zusätzliche Schichten, welche beim KNN nicht vorhanden sind. In den folgenden Abschnitten soll die Funktionsweise beschrieben werden.

Faltungsschicht
Es existieren eine menge von Filtern n welche eine Höhe von f_h, einer Breite von f_w und

einer tiefe von f_d haben. Dieser Filter wird über das Eingabebild mit der Höhe e_h, der breite e_w und der Tiefe von e_d gelegt und in ein Pixel schritten bewegt. Die Ergebnisse stellen eine Liste von Merkmalen da welche auf der Höhe m_h, breite m_w und tiefe m_d liegen. Formell ausgedrückt:

$$m_h = (e_h - f_h) + 1 \tag{5}$$
$$m_w = (e_w - f_w) + 1 \tag{6}$$

Um die Verringerung von Informationen nach der Faltung zu verhindern, werden an den Rändern der Bilder Polsterungen, auch *padding* genannt, angehangen [Goo+16, S. 343]. Diese Polsterungen bestehen aus Pixeln mit einem Wert von *0*. Die Anzahl der zusätzlichen Pixel ergibt sich aus der Höhe und Breite des Eingabebildes, um wie viele Pixel das Bild vergrößert wird. Formell ausgedrückt:

$$p_h = f_h - 1 \tag{7}$$
$$p_w = f_w - 1 \tag{8}$$

Die Faltungsschicht hat eigene Gewichte, welche beim Vorgang mit denen des Bildes zusammengerechnet wird [Goo+16, 331ff.]. Für die gleiche Schicht wird derselbe Filter verwendet. Die Weise der Berechnung wird in der folgenden Grafik beschrieben.

Pooling Schicht

Die Pooling Schicht oder auch *Pooling Layer* kommt nach jeder Faltungsschicht [Goo+16, 336f.]. Die Aufgabe dieser Schicht ist es das Eingabebild in der Auflösung und den zugehörigen Rechenaufwand für die folgenden Schichten zu reduzieren. Üblicherweise gibt es zwei Verfahren, welche beim Pooling eingesetzt werden. Zum einen das Average Pooling und das am häufigsten verwendete Max Pooling. Auch diese Schicht besteht aus einem quadratischen Filter, welcher wie bei der Faltungsschicht über die Bildpunkte gelegt wird. Dabei wird beim *Max Pooling* der höchste Wert in diesem Bereich ermittelt und übernommen. Bei einem standardmäßigem Kernel von 2x2, wird das Bild um den Faktor 2 Verkleinert. Die Abmessungen sind, anders wie bei der Faltungsschicht, gerade.

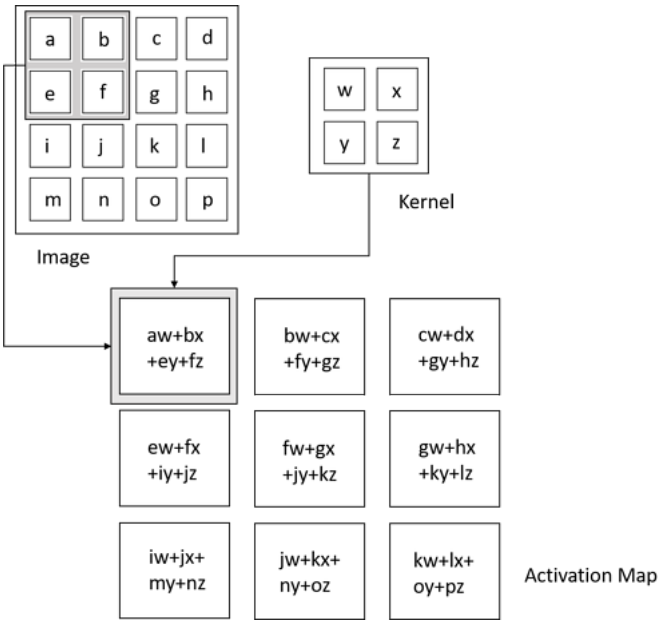

Abbildung 5: *Funktionsweise der Faltungsschicht [Mis19]*

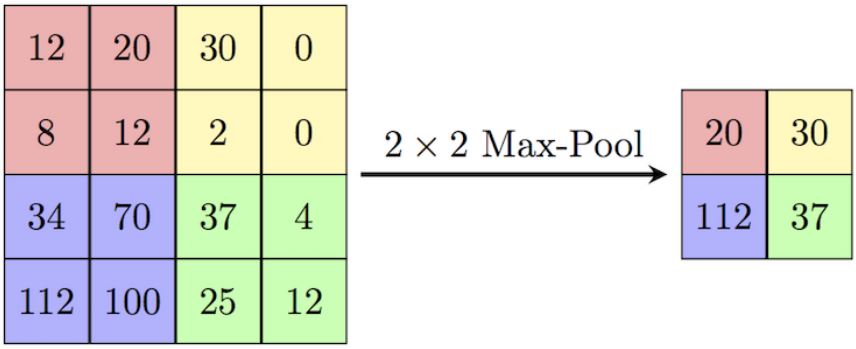

Abbildung 6: *Funktionsweise des Pooling Layers mit der Max Pooling variante [Wik18]*

Fully Connected Layer nachdem das Eingabebild durch alle Faltungs- und Pooling Schichten verarbeitet wurde, werden die Daten an den Fully connected Layer übergeben [SCL12, S. 14]. Dieser verwendet, wie bei einem normalen KNN, Perzeproten um die Daten zu verarbeiten. Die vorherigen Schichten wurden durchlaufen, damit diese rechenintensive Schicht nicht zu viel Informationen verarbeiten muss.

2.2.2 Entstehung von Trainingsdaten

Damit ein künstliches neuronales Netz Objekte erkennen und zuordnen kann, benötigt es zunächst eine Reihe von Beispieldaten. Anhand dieser Beispieldaten lernt das neuronale Netz, welche Eigenschaften das zu erlernende Objekt hat und wie es aufgebaut ist. Für diesen Vorgang werden üblicherweise mehrere Hunderttausend Daten benötigt, um ein akzeptables Ergebnis zu erzielen. Da für ein solches Training nicht die benötigten Res-

sourcen zur Verfügung stehen, sollte eine Möglichkeit gefunden werden, wie das Training auch mit weniger Trainingsdaten durchgeführt werden kann. Eine Möglichkeit, das Training mit weniger Trainingsdaten durchzuführen, ist das Verwenden vortrainierter Netze. Diese Netze wurden mit rund 300.000 Bildern über mehrere Tage trainiert und als Open Source zur Verfügung gestellt. Beim Verwenden dieser Netze werden lediglich die obersten Schichten, welche für die Zuweisung verantwortlich sind, durch den neuen Datensatz verändert. Durch das vortrainieren werden nur noch durchschnittlich 150 Bilder pro Objektklasse und eine geringere Trainingszeit benötigt. Die Bilder der eigenen Daten werden mit einer normalen 12 Megapixel Kamera aufgenommen. Die verwendeten Objekte, müssen von allen möglichen Seiten fotografiert werden, dabei müssen, für das Training, pro klasse ungefähr 150 Bilder vorhanden sein. Da die Bilder mit einer 12 Megapixel Kamera mehrere 100 MB groß werden, müssen diese auf 100-200 KB komprimiert werden.

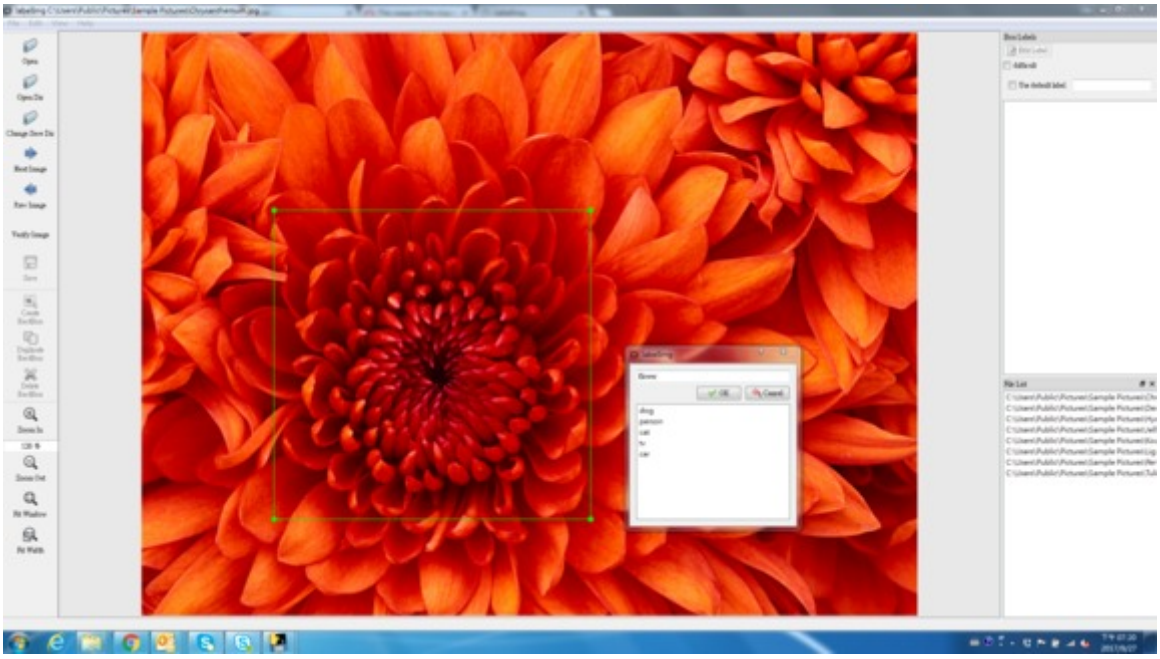

Abbildung 7: *Vorbereitung der Datensätzen mit dem Labelimg Programm [tzu19]*

Eine Klassifizierung gehört zum überwachten Lernen und benötigt deswegen gelabelte Datensätze. Das Labeln wurde mit dem Open Source Programm *LabelIMG* [tzu19] durchgeführt. Dabei wird das zu Trainierende Objekt mit einem rechteckigen Kasten umschlossen und einer Klasse auf den Bildern markiert. Diese Daten werden zunächst in einer XML Datei abgespeichert.

2.3 Mean average Precision

Oft können neuronale Netze nicht nur ein Objekt gleichzeitig erkennen, sondern geben als Ergebnis mehrere Hypothesen aus. Dadurch entsteht das Problem, das nicht mehr nur zwischen richtig und falsch unterschieden werden kann, sondern eine Genauigkeit benötigt wird, welche alle möglichen Objekte einbezieht. Eine Metrik welche auf diese Informationen eingeht, ist die (mean) average Precision (AP/mAP). Für die Berechnung wird die Präzision (Precision) und der Rückgabewert eines Ergebnisses benötigt. Die Präzision P beschreibt den Anteil der richtig erkannten Objekte unter allen erkannten Objekten. Der Rückgabewert R ist der korrekte Anteil von allen möglichen Objekten. Formell ausgedrückt:

$$P = \frac{richtig\,erkannte}{richtig\,erkannte + falsch\,erkannte} \tag{9}$$

$$R = \frac{richtig\,erkannte}{richtig\,erkannte + falsche\,nicht\,erkannte} \tag{10}$$

Häufig wird ein Schwellwert für die Pseudowahrscheinlichkeit genutzt, mit welchem entschieden wird, wann ein erkanntes Objekt der Rückgabeliste hinzugefügt wird oder nicht. Wird der Schwellwert niedrig angesetzt, werden meist mehr Objekte, als erkannt eingestuft. Dadurch steigt der Rückgabewert mit richtig erkannten Objekten. Gleichzeitig sinkt aber auch die Präzision da natürlicherweise auch mehr falsche Objekte erkannt werden. Objekterkennungsverfahren mit einer hohen Qualität zeigen aber weiterhin eine gute Präzision.

2.4 Digitale Bilder

Damit der Ansatz der These, dass die Farbnormalisierung von Trainingsdaten das Trainingsverhalten und die spätere Genauigkeit künstlicher neuronaler Netze verbessern kann, nachvollzogen werden kann, soll in diesem Unterkapitel, auf die Aufnahme digitaler Bilder, mit deren Eigenschaften und Aufbau eingegangen werden. Außerdem sollen die Schwächen herausgearbeitet und mögliche Lösungsansätze erläutert werden. Zunächst folgt eine kurze Einführung.

Noch vor einiger Zeit war die Aufnahme und Verarbeitung von Bildern nur einer kleinen Gruppe von Spezialisten, mit kostspieliger Ausrüstung möglich. Spätestens seid der Entwicklung von digitalen Aufnahmegeräten wie Digitalkameras, Scannern und Multimedia-PCs ist die Aufnahme und Verarbeitung von Bildern, nicht mehr einer kleinen Gruppe von Spezialisten vorbehalten, sondern für jeden erschwinglich. Mittlerweile gibt es eine riesige Community von Menschen, bei denen das Arbeiten mit digitalen Bildern schon längst

zum Alltag geworden sind. Durch heutige Bildverarbeitungsprogramme wird ein tiefer gehendes Wissen nicht mehr zwingend vorausgesetzt, denn viele komplexen Abläufe werden automatisiert. Das führt zu dem Problem, das Aufbau und zusammenhänge der Bilder häufig nicht verstanden und falsch genutzt werden. Auch im professionellen Bereich wie, Werbung, Medizin, Deep Learning, werden digitale Bilder fast ausschließlich verwendet. Auch hier entstehen oft, durch das fehlende Verständnis der Zusammenhänge, häufig zur Unterschätzung der Probleme und nicht selten zu ineffizienten Lösungen, teuren Fehlern und persönlicher Frustration.
Bildverarbeitung, wird in der heutigen Zeit oft mit Bildbearbeitung assoziiert, also mit der Bearbeitung von Bildern mittels Bearbeitungssoftware. In der Bildverarbeitung geht es anders als in der Bearbeitung, um die Software selber, welche für die Konzeption und Erstellung von digitalen Bildern genutzt wird. In der Programmierung sind Bilder nichts weiter als Zahlen-Matrizen und Arrays, welche man nach Belieben lesen und verändern kann, um seine geplanten Ziele zu erreichen. Grundsätzlich bestehen digitale Bilder aus rastern. Diese besitzen mehrere werte, welche auch Pixel genannt werden. In diesen Pixeln werden Bildinformationen wie Farbe und Position festgehalten. Jeder Pixel besteht dabei oft aus drei Kanälen, den R Kanal für Rot, den G Kanal für Grün und den B Kanal für Blau. Diese bilden in bestimmten Kombinationen und Intensitäten ungefähr 16 Millionen verschiedene Farben. Der beschriebene Aufbau wird bei RGB-Bildern verwendet und auch bei Monitoren, Smartphones, etc. Neben dem RGB gibt es noch einige verschiedene Farbräume (CMY, CMYK, HSV, Lab, etc).

RGB

Der RGB-Farbraum ist der geläufigste und bekannteste Farbraum dieser besteht wie in () kurz beschrieben aus drei Farbräumen (Rot-Grün-Blau). Dieses beschreibt, welche Art von Licht ausgestrahlt werden muss, um eine gewünschte Farbe zu erreichen. Die Kombination der einzelnen Farbräume ist eine additive Farbmischung. RGB ist eher ein Farbmodell, da es viele Farbräume gibt, welche sich davon Ableiten (sRGB, AdobeRGB, etc.).

HSV

Anders als bei RGB ist HSV anders aufgebaut, dennoch gibt es auch hier drei Werte. Das H (Hue) seht für den Farbton, welcher angenommen werden soll, S (Saturation) für die Sättigung der ausgewählten Farbe und V (Value) für den Hell-wert oder auch die Helligkeit der Farbe. Der HSV Farbraum besteht aus diesen Koordinaten. Dieser Farbraum ist gerade in der Kunst beliebt, da die Farbvorstellung besser funktioniert als bei additiven und subtraktiven Farbmischungen.

CMY/CMYK

Anders als der RGB-Farbraum benutzt der CMY-Farbraum den subtraktiven anstelle der

additiven Farbmischung. Hierbei sind die drei Grundfarben C für Cyan, M für Magenta und Y für Yellow. Bekannt ist diese Farbmischung gerade durch das Malen von Wasserfarben, oder für die Farbpigmente, welche in Druckern genutzt werden. Für den durch die Farbmischung kann der CMY kein richtiges Schwarz erreichen, deswegen wurde der Farbraum CMYK mit einem Parameter erweitert. Das K steht für Schwarz, damit auch schwarz richtig dargestellt werden kann und kein dunkles Grau entsteht.

Lab

Der Lab-Farbraum ist ein Messraum, in welchem jede wahrnehmbare Farbe enthalten ist. Der Farbraum wurde auf Grundlage der Gegenfarbentheorie konstruiert. Eine wichtige Eigenschaft, ist die Geräteunabhängigkeit. Das bedeutet, die Farben werden unabhängig der Erzeugung und der Wiedergabe definiert. Die umfassende Definition kann in der DIN 6174 nachgelesen werden. Auf Grundlage der Gegenfarbentheorie liegen sich auf der a Achse die Farben Grün und Rot gegenüber. Die b Achse entspricht den Gegenfarben Blau und Gelb. Senkrecht durch die Ebenen befindet sich die L Achse welche von 0 (weiß) bis 100 (Schwarz) geht. Diese gibt die Helligkeit wieder und hat Zwischenschritte für die Graustufen.

2.4.1 Aufbau von Digitalbildern

Üblicherweise wird bei Digitalbildern mit dem RGB-Farbraum gearbeitet. Aus diesem Grund, wird in dieser Arbeit hauptsächlich auf RGB-Bilder eingegangen. Ein Bild ist wie eine Matrix aufgebaut, dabei kann es sowohl quadratisch wie auch rechteckig Abmessungen haben. Jeder Punkt in der Matrix stellt einen Pixel dar und enthält jeweils einen Rot-, grün-, Blauwert welche zusammen eine beliebige Farbe darstellt. Um die Farbverteilung in einem Bild besser erkennen zu können, werden die Farbwerte in sogenannten Farb-Histogrammen zusammengefasst. Histogramme spielen für die Normalisierungsfunktionen eine Rolle, aus diesem Grund wird das Konzept hinter den Histogrammen und die spätere Manipulation, beschrieben. Für die einfachheitshalber werden Graustufenbildern für die Erklärung verwendet. Anwendbar sind diese auch auf die Histogramme der einzelnen RGB-Kanäle.

Histogramme

Histogramme beschreiben eine Häufigkeitsverteilung [BB09, 42ff.]. Speziell bei Bildern zeigen Histogramme die Häufigkeit der auftretenden Intensität-werte. Am einfachsten lässt sich das mit Graustufenbildern nachvollziehen. Ein Beispiel dafür zeigt die (Abbildung 8) für ein Graustufenbild I mit möglichen Intensität-werten im Bereich $I(n, V)E[0, K-1]$ enthält das Histogramm H genau K Einträge. Die Menge der Einträge bei einem 8 Bit Graustufenbild sind 2^8 Farbabstufungen ($K = 2^8 = 256$). Der Wert des Histogramms an der Stelle i ist $h(i)$ die Anzahl der Pixel von I mit einem Intensitätswert i, für alle

$(0 < i < K)$. Mathematisch ausgedrückt ergibt sich $h(i) = card(u, v)|I(u, v) = i$
daraus lässt sich entnehmen, dass $h(0)$ die Anzahl der Pixel mit dem Wert 0, $h(1)$ die Anzahl des Wertes 1 usw., $h(255)$ ist schließlich der letzte Wert und gibt die Menge aller weißen Pixel in dem Bild an $(K - 1)$. Eine Histogramm Berechnung stellt somit einen eindimensionalen Vektor h der Länge K da.
card beinhaltet die Anzahl der Elemente einer Menge (*Kardinalität*)

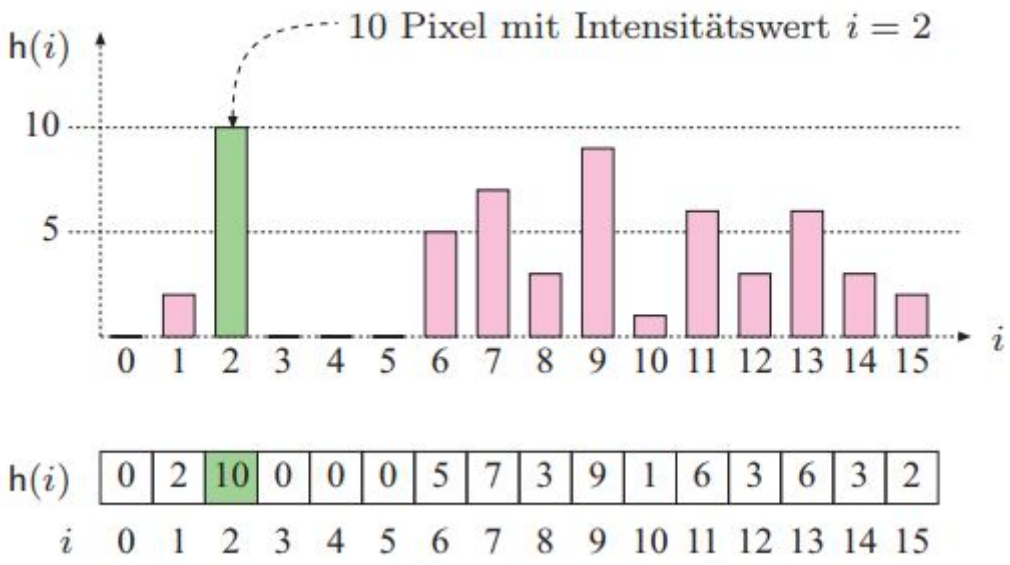

Abbildung 8: *Histogramm eines Graustufen Bildes mit 16 Helligkeitsabstufungen [BB09, S. 42]*

Die Abb.8 zeigt ein mögliches Histogramm mit einer Farbabstufung von 16 werten. Ein Histogramm zeigt die wichtigsten Eigenschaften eines Bildes, wie beispielsweise die Dynamik oder den Kontrast eines Bildes. Außerdem lässt sich erkennen, ob bei der Bildaufnahme Probleme aufgetreten sind. Diese lassen sich erkennen indem Ausschläge ganz links oder ganz rechts, des Graphen zu erkennen sind. Ein Ausschlag bei 0 oder 255 bedeutet das dort keine Farbinformationen mehr vorhanden sind, und das teile des Bildes, Über- beziehungsweise unterbelichtet sind.

Kontrast
Der Kontrast bezeichnet den Bereich von Intensitätstufen, welche in einem Bild effektiv genutzt werden, also die Differenz der maximalen und minimalen Pixelwerten [BB09, S. 44]. Ein Bild welches einen hohen Kontrast hat, nutzt das gesamte Spektrums des Histogramms (Schwarz bis weiß). Der Kontrast, lässt sich also leicht aus einem Histogramm entnehmen, wie in Abb. (NOCH HINZUFÜGEN) die verschiedenen Histogramme zeigen.

Dynamik

Dynamik bedeutet, wie viel verschiedene Pixelwerten in einem Bild genutzt werden, und die jeweilige Anzahl[BB09, S. 44]. Im besten Fall entspricht die Dynamik der insgesamt genutzten Anzahl an Pixelwerten. Also $K-1$ genutzter Farbabstufungen, damit der Wertebereich voll ausgeschöpft wird.

Anders als der Kontrast, welcher immer erhöht werden kann, solange der maximale Wertebereich nicht erreicht worden ist, kann die Dynamik eines Bildes nicht so leicht verbessert werden. Aus diesem Grund ist eine hohe Dynamik von Vorteil, denn dadurch wird die Gefahr von Qualitätsverlust bei Verarbeitungsschritten verringert. Trotz der Tatsache, das viele Ausgabegeräte mit Farbtiefen von 8 Bit arbeiten, nehmen heutige Kameras bis zu 12-14 Bit pro Kanal auf, um einen möglichen Qualitätsverlust vorzubeugen.

2.4.2 Einfluss von Belichtungen

In dem Kapitel2.4.1 wurde Beschrieben wie die Farben in Bildern gespeichert werden und welche unterschiedlichen Arten von Farbräumen es gibt [BB09, 41ff.]. Bei der Aufnahme von Bildern ist gerade die Belichtung entscheiden. Eine hohe Belichtung sorgt dafür, dass Farben heller wirken, wobei wenig licht dafür sorgt das Farben dunkler erscheinen. Diese Tatsache kann beim Deep Learning zu Problemen führen, da ein und dasselbe Objekt unterschiedliche Farben annehmen kann und je nach Belichtungsintensität variiert. Ein künstliches neuronales Netz kann Objekte welche es normalerweise trainiert hat nicht immer erkennen, wenn die Farbe zu stark von der eigentlichen Farbe abweicht. Um Belichtungen auf Bildern auszugleichen und zu normalisieren, gibt es einige verfahren, welche diesem Problem entgegenwirken. Die Ausgewählten verfahren werden aufgeführt und beschrieben. Diese stellen nicht die Originalfarbe des Objektes wieder her, sondern sorgen für eine konstante Farbvariation, welche sich nicht stark verändert.

2.4.3 Arten der Farbnormalisierung

Die Farbnormalisierung ist ein Thema der Bildverarbeitung und wird hauptsächlich mit künstlicher Farbsicht befasst. Die Verteilung und Darstellung von Farben auf Bildern hängt hauptsächlich von Beleuchtungsbedingungen und der Kamera ab. Das bedeutet das sich die Farben bei der Aufnahme, je nach Beleuchtung verändern. Das ist gerade im Bereich von *Maschine Learning* problematisch, da diese Farbveränderungen zu Fehlern im Lernprozess und der späteren Genauigkeit führen. Farbnormalisierung-Algorithmen, sollen dafür sorgen, dass die Farbabweichungen durch Lichteinfluss geringere Auswirkungen habe und zu besseren Ergebnissen führen. Im folgenden werden die ausgewählten Algorithmen erläutert, welche im Rahmen dieser Arbeit getestet werden.

Gray-World-Algorithmus

Bei der Gray-World Normalisierung geht es nicht, wie der Name vermuten lässt um die Umwandlung in ein Graustufenbild, eher werden die verschieden Farbräume abgedunkelt und eingegrenzt. Es wird davon ausgegangen, dass das Farbspektrum durch drei konstante Faktoren modelliert werden kann. Dafür skalieren die Konstanten α, β und γ an die Kanäle R, G und B. Dadurch kann eine Konstanzlösung erzielt werden, welcher bei unterschiedlichen Belichtungen unveränderlich ist, indem jeder Farbkanal durch seinen Durchschnittswert geteilt wird, wie in folgender Formel beschrieben:

$$(\alpha R, \beta G, \gamma B) \rightarrow \left(\frac{\alpha R}{\frac{\alpha}{n} \sum_i R}, \frac{\beta G}{\frac{\beta}{n} \sum_i G}, \frac{\gamma B}{\frac{\gamma}{n} \sum_i B} \right) \tag{11}$$

Wie schon erwähnt, ist diese Art der Farbnormalisierung für verschiedene Farbvariationen unveränderlich. Ein größeres Problem dieses Normalisierung-verfahren besteht darin, das nicht alle Variationen der Beleuchtungsintensität berücksichtigt und auch nicht einfach für dynamische Szenen verwendet werden kann. Um solche Probleme zu lösen, gibt es mehrere Variationen dieses Algorithmus.

Histogramm Ausgleichung

Die Histogramm Ausgleichung ist eine Nichtlineare Transformation, welche auf Grundlage der Histogramme wie in Kapitel 2.4.1 beschrieben arbeitet. Der Pixelrang wird dabei nicht verändert und kann bei jeder monoton steigenden Farbformation-Funktion verwendet werden. Diese Normalisierung-Funktion gilt stärker als der Gray-World-Algorithmus. Durch die Histogramm Ausgleichung entsteht ein dominanter blauer Kanal, wodurch das Bild oft unnatürlich erscheint.

Tabelle 1: *Tabellarisches Histogramm des Zielbildes*

Graustufen	0	1	2	3	4	5	6	7
Anzahl der Pixel	8	10	10	2	12	16	4	2

0	1	5	1	7	2	0	3
0	0	5	5	5	2	4	5
4	5	1	4	1	5	1	4
5	1	2	4	5	2	6	3
5	2	6	4	0	4	0	5
4	0	2	4	7	4	6	2
5	1	6	1	0	1	1	5
4	3	2	4	2	5	2	5

Aus der Pixelverteilung wird nun ein Ausgleich berechnet welche den kumulativen Pixelwert durch die Anzahl aller Pixel teilt und durch die höchste Graustufe teilt. Das Ergebnis wird gerundet und der passenden Graustufe zugeordnet. Die Vorgehensweise mit Farbbildern funktioniert gleich, nur das die R, G und B Kanäle verwendet werden.

Tabelle 2: *Mathematische Umsetzung der Histogrammausgleichung*

r_k	P_k	Kumulative Werte	$Kumulative/Gesamt * (L-1)$	Gerundete nahe Grauwerte
0	8	8	$8/64 * 7 = 0.875$	1
1	10	18	$18/64 * 7 = 1.968$	2
2	10	28	$28/64 * 7 = 3.0625$	3
3	2	30	$30/64 * 7 = 3.2812$	3
4	12	42	$42/64 * 7 = 4.5937$	5
5	16	58	$58/64 * 7 = 6.3437$	6
6	4	62	$62/64 * 7 = 6.78125$	7
7	2	64	$64/64 * 7 = 7$	7

Histogramm Spezifikation

Ähnlich wie die Histogramm Ausgleichung arbeitet die Histogramm Spezifikation. Die Histogramme der roten, grünen und blauen Kanäle werden so umgewandelt, dass sie den Formen von drei spezifischen Histogrammen entsprechen, anstatt sie einfach auszugleichen. Mithilfe dieses Vorgehens wird darauf abgezielt, Bilder zu erhalten welche ein Histogramm einer bestimmten Form haben. Zunächst muss das Bild so konvertiert werden, dass das Histogramm eine bestimmte Form hat. Für diese Methode werden üblicher weise zwei ähnliche Bilder verwendet, einmal das Hauptbild und das Referenzbild. Das Referenzbild hat die gewünschte Form des Histogramms, hingegen das Hauptbild nicht optimiert ist. Zunächst wird mit beiden Bildern eine Histogramm Ausgleichung durchgeführt. Im Weiteren wird das Referenz-Histogramm mit dem des Hauptbildes kombiniert und angepasst.

Tabelle 3: *Tabellarisches Histogramm des Referenz-Bildes (B1)*

Graustufen	0	1	2	3	4	5	6	7
Anzahl der Pixel	8	10	10	2	12	16	4	2

Um die Funktionsweise verständlicher darzustellen, wird kurz auf das Vorgehen und die mathematische Herleitung eingegangen. Dafür gibt es in den Tabellen?und? zwei Histogramme mit 8 Grauabstufungen. In diesen Tabellen wird aufgeführt, wie oft der jeweilige

Tabelle 4: *Tabellarisches Histogramm des Zielbildes (B2)*

Graustufen	0	1	2	3	4	5	6	7
Anzahl der Pixel	0	0	0	0	20	20	16	8

Grauwert in dem Bild vorkommt. Nun werden die Histogramme wie in Absch2.4.1 beschrieben, ausgeglichen. Von 0 bis $L-1$ werden die Summen der Pixel berechnet, durch die Anzahl der im Bild enthaltenden Pixel geteilt und mit der höchsten Graustufe $(L-1)$ multipliziert. Das Ergebnis des Ausgleiches wird in Tabelle? Aufgeführt. Nun werden die Grauwertverteilungen von B1 und die Verteilung von B2 angepasst dafür wird der nächste Wert aufgerundet von B1 übernommen.

In einem Beispiel:

1 (B1) am nächsten an 2 (B2) liegt bei Graustufe 4
2 (B1) am nächsten an 2 (B2) liegt bei Graustufe 4
3 (B1) am nächsten an 4 (B2) liegt bei Graustufe 5
3 (B1) am nächsten an 4 (B2) liegt bei Graustufe 5
.....

Tabelle 5: *Von zwei Ausgeglichenen Histogrammen zu einem Spezifizierten Histogramm*

Graustufe	Ausgleichung B1	Ausgleichung B2	Spezifiziertes Histogramm
0	1	0	4
1	2	0	4
2	3	0	5
3	3	0	5
4	5	2	6
5	6	4	6
6	7	6	7
7	7	7	7

Aus der neuen Stufen Verteilung entsteht ein neues Histogramm welches weniger Farbstufen enthält. Dadurch verschiebt sich das Histogramm Richtung weiß und das Bild wird erhellt.

Tabelle 6: *Tabellarisches Histogramm des Refernzbildes (B1) auf Basis des Zielbildes (B2)*

Graustufen	0	1	2	3	4	5	6	7
Anzahl der Pixel	0	0	0	0	18	12	28	6

3 Methodik und Durchführung

Nach der theoretischen Grundlage soll sich der zweite Teil der Arbeit um die Versuchsbeschreibung, Durchführung und Auswertung der Ergebnisse handeln. Dabei wird zunächst die Versuchsumgebung und die verwendeten Frameworks eingegangen, und wie diese für den Versuch genutzt werden.

3.1 Frameworks und Entwicklungsumgebung

Wie in vorherigen Kapiteln schon beschrieben ist das Trainieren eines künstlichen neuronalen Netzes oder eines faltenden neuronalen Netzes sehr rechenintensiv. Die enthaltenen Daten werden parallel mittels Matrizenmultiplikation verarbeitet. Aus diesem Grund werden für die Verarbeitung Grafikprozessoren hauptsächlich verwendet da diese im Gegensatz zu normalen Prozessoren für die Berechnung von Matrizen konzipiert wurden.
Auf Softwareseite gibt es eine Menge an Frameworks, mit welcher sich CNNs realisieren lassen. Einige von denen sind beispielsweise Tensorflow, Keras und Theano. Wegen der vorliegenden Erfahrung und der Möglichkeit Trainingsfortschritte grafisch anzeigen zu können, wird der praktische Teil der Arbeit mittels Tensorflow durchgeführt. Tensorflow ist ein Open Source Framework von Google, welches zum entwickeln künstlicher neuronaler Netze genutzt werden kann. Programmiert wurde das Framework in den Programmiersprachen C und Python, welche auch für die Entwicklung genutzt werden. Mit dem Framework Tensorflow kommen einige Möglichkeiten. Zum einen gibt es eine übersichtliche Dokumentation, zum anderen werden vortrainierte Netze, Open Source und für die Entwicklung die Tensorflow Object Detection API angeboten. Zur Verfügung stehen verschiedenste Netze auf Basis von verschiedenen Datensätzen.

3.2 Verwendetes Modell

Das Unternehmen COCO [Con18] (Common Objects in Contexts) besitzen eine vielzahl an Datensätzen mit welchen, neuronale Netze trainiert werden können. Einige dieser Netze wurde auf der Object detection API von Tensorflow trainiert und stehen als Open Source zur Verfügung. Die folgende Tabelle führt ausgewählte Modelle auf, um die Genauigkeit und Geschwindigkeit vergleichen zu können.

Das faster_rcnn_inception_v2_coco soll für das Projekt verwendet werden. Dieses hat eine mAP von 28 und eine Geschwindigkeit von 32 ms. Dadurch das es die niedrigste mAP hat, sollten die Unterschiede in der Genauigkeit stärker ausfallen. Das ausgewählte Modell wird im folgenden auf die Datensätze, welche im anschließenden Unterkapitel aufgeführt sind.

Tabelle 7: *Modelle [LLC18] welche mit dem COCO Dataset Trainiert wurden [Con18]*

Modell	mAP	Geschwindigkeit
faster_rcnn_inception_v2_coco	28	58
faster_rcnn_resnet50_coco	30	89
rfcn_resnet101_coco	30	92
ssd_mobilenet_v1_coco	30	21
ssd_resnet_50_fpn_coco	35	76

3.3 Trainingsdaten

Damit am Ende des Projektes möglichst gute Vergleichswerte entstehen, werden mehrere Datensätze vor den Versuch verwendet. Insgesamt werden die neuronalen Netze mit 3 unterschiedlichen Datensätzen trainiert. Dafür wurden 3 Datensätze heraus gesucht. Der erste Datensatz welcher getestet wird, ist der Nahrungsmittel Datensatz eines früheren Projektes, zum anderen der Datensatz von *Pascal Visual Object Classes* (PascalVOC) und der Hunderassen Datensatz, Stanford Dog Dataset.

In den folgenden Tabellen werden alle Datensätze mit den zugehörigen Klassen und Anzahl an Trainingsbildern aufgeführt:

Tabelle 8: *Datensatz mit Nahrungsmitteln*

Objektklasse	Objektklasse
Milch - Packung	Orangensaft - Packung
Wasser - Flasche	Bier - Flasche
Brunch - Aufstrich	Margarine - Aufstrich

Der PascalVOC Datensatz umfasst 20 Klassen und besteht aus 5.000 Bildern. Von 2005 bis 2012 wurde jährlich die PascalVOC Challenges durchgeführt. Dabei sollte das beste verfahren für die Segmentierung, Klassifikation und Objekterkennung ermittelt werden. Inhaltlich befasst sich der Datensatz mit einer reihe unterschiedlicher Klassen, welche in Tabelle 9 zu erkennen sind. Es gibt einige Unterschiede in der Häufigkeit der auftretenden Klassen. Beispielsweise sind in rund 2000 Bildern insgesamt 4.690 Personen enthalten, weswegen es möglicherweise Unterschiede in der Genauigkeit untereinander, der Klassen geben könnte. Die Aufnahmen der Bilder sind thematisch und und von der Art der Aufnahme unstrukturiert, was bedeutet, dass teilweise Bilder von einzelnen Objekten, Gruppen von Objekten oder auch ganze Szenen enthalten sind.

Der dritte Datensatz, welcher verwendet wird, ist der Stanford Dogs Datensatz. Dieser beinhaltet mehr als 24.000 Bilder und 120 Klassen, wobei es inhaltlich um die Erkennung der Hunderasse geht. Dabei ist der Datensatz so aufgebaut, dass die enthaltenen Rassen

Tabelle 9: *Pascal Visual Object Classes [Eve+]*

Objektname	Objektname	Objektname	Objektname
Person	Vogel	Katze	Kuh
Hund	Pferd	Schaf	Zug
Flugzeug	Fahrrad	Boot	Bus
Auto	Motorrad	Flasche	Stuhl
Tisch	Blumentopf	Sofa	Bildschirm

mit den zugehörigen Annotationen in einzelnen Ordnern geordnet sind. Für die Übersicht wurden 20 verschiedene Rassen ausgewählt und für das Training verwendet.

Tabelle 10: *Stanford Dog Dataset [Kho+11]*

Objektname	Objektname	Objektname	Objektname
Chihuahua	Japanese spaniel	Maltese dog	Pekinese
Shih-Tzu	Blenheim Spaniel	Papillon	Toy Terrier
Rhodesian Ridgeback	Afghan Hound	Basset	Beagle
Bloodhound	Bluetick	coonhound	Walker Hound
Redbone	Borzoi	Irish Wolfhound	Italian Greyhound

3.4 Normalisierung-Algorithmen

Für die Normalisierung der Datensätze ist eine Methode nötig, um mehrere Bilder möglichst schnell hintereinander zu bearbeiten. Für die Normalisierung wurden Python Programme geschrieben, welche nacheinander die Bilder, anhand eines Algorithmus verarbeitet. Dafür wurde mit unter die *Python Image Libary* und OpenCV genutzt. Beide Bibliotheken werden für das verarbeiten von Bildern benötigt.

3.4.1 Gray-World-Algorithmus

```
def grey_world(nimg):
    nimg = nimg.transpose(2, 0, 1).astype(np.uint32)
    mu_g = np.average(nimg[1])
    nimg[0] = np.minimum(nimg[0]*(mu_g/np.average(nimg[0])),255)
    nimg[2] = np.minimum(nimg[2]*(mu_g/np.average(nimg[2])),255)
    return  nimg.transpose(1, 2, 0).astype(np.uint8)
```

In dem verwendeten Gray World Algorithmus [Aih12] Zunächst wird das Bild

Abbildung 9: *Auf der linken Seite ist das Originalbild. Auf der rechten Seite ist das Bild welches mit mit dem Gray World Algorithmus normalisiert wurde*

3.4.2 Histogramm Ausgleich

```
def histogram_equalisation(img):
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    R, G, B = cv.split(img)
    output1_R = cv.equalizeHist(R)
    output1_G = cv.equalizeHist(G)
    output1_B = cv.equalizeHist(B)
    output1 = cv.merge((output1_R, output1_G, output1_B))
    im = Image.fromarray(output1)
    return im
```

Für die Histogramm Ausgleichung [Jaz16] wird das Bild zunächst vom BGR Farbraum in den RGB Farbraum umgewandelt. Der Farbraum ist abweichend von der Reihenfolge der Kanäle identisch mit dem RGB Farbraum. Im zweiten Schritt werden die Farbkanäle des Bildes aufgesplittet. Der Rote, Grüne und Blaue Farbkanal bekommen jeweils einen Histogramm Ausgleich, wie in Kapitel2.4.1 beschrieben. Die normalisierten Farbkanäle müssen wieder miteinander verschmolzen und werden und wird schlussendlich in einem Bildformat zurückgegeben. Das Ergebnis wird als JPG Bild abgespeichert.

Abbildung 10: *Auf der linken Seite ist das Originalbild. Auf der rechten seite ist das Bild welches mit einer Histogramm Ausgleichung bearbeitet wurde*

3.4.3 Histogramm Spezifikation

```
def histeq(im,nbr_bins=256):

    #get image histogram
    imhist,bins = histogram(im.flatten(),nbr_bins,normed=True)
    cdf = imhist.cumsum() #cumulative distribution function
    cdf = 255 * cdf / cdf[-1] #normalize

    #use linear interpolation of cdf to find new pixel values
    im2 = interp(im.flatten(),bins[:-1],cdf)

    return im2.reshape(im.shape), cdf

imsrc = imread("test1.jpg")
imtint = imread("image1.jpg")

nbr_bins=255
if len(imsrc.shape) < 3:
     imsrc = imsrc[:,:,np.newaxis]
```

```
    imtint = imtint[:,:,np.newaxis]

imres = imsrc.copy()
for d in range(imsrc.shape[2]):
    imhist,bins = np.histogram(imsrc[:,:,d].flatten(),nbr_bins,density=Tr
    tinthist,bins = np.histogram(imtint[:,:,d].flatten(),nbr_bins,density

    cdfsrc = imhist.cumsum() #cumulative distribution function
    cdfsrc = (255 * cdfsrc / cdfsrc[-1]).astype(np.uint8) #normalize

    cdftint = tinthist.cumsum() #cumulative distribution function
    cdftint = (255 * cdftint / cdftint[-1]).astype(np.uint8) #normalize

    im2 = np.interp(imsrc[:,:,d].flatten(),bins[:-1],cdfsrc)
    im3 = np.interp(im2,cdftint, bins[:-1])
    imres[:,:,d] = im3.reshape((imsrc.shape[0],imsrc.shape[1] ))

try:
    imsave("histnormresult.jpg", imres)
except:
    imsave("histnormresult.jpg", imres.reshape((imsrc.shape[0],imsrc.shap
```

4 Ergebnisse

Nachdem alle Trainingseinheiten mit den verschiedenen Normalisierungsverfahren durchlaufen wurden, sollen die Ergebnisse aufgeführt und verglichen werden. Als Vergleichswerte konnte mithilfe von Tensorboard die Genauigkeiten des Netzes erhoben werden. Jede Klasse hat eine eigene durchschnittliche Genauigkeit (AP) und eine Genauigkeit für das gesamte Netz in Form einer mAP. In den folgenden Tabellen werden bestimmte Abkürzungen verwendet: N = Normaler Datensatz, GW = Gray-World-Algorithmus, HA = Histogramm Ausgleich, HS = Histogramm Spezifikation

4.1 Nahrungsmittel Datensatz

Der erste Datensatz welcher untersucht wurde ist der Datensatz, welcher in der Tabelle 11 aufgeführt ist. Dieser enthält im Vergleich zu den anderen Datensätzen weniger Objektklassen und weniger Bilder. Außerdem ist, sind die Objekte unter ähnlichen Bedingungen entstanden. Das bedeutet, dass die Lichteinstrahlung in den meisten Fällen gleich ist und nur auf wenigen unterschiedlichen Hintergründen aufgenommen wurden. In der Tabelle

Tabelle 11: *Genauigkeits-Berechnungen des Trainierten Modells ohne Normalisierungsverfahren*

Klassenname	AP(N)	AP(GW)	AP(HA)	AP(HS)
Wasser - Flasche	0.948	**0.955**	0.934	0.000
Orangensaft - Packung	**0.999**	0.998	**1.000**	0.000
Milch - Packung	**1.000**	**1.000**	**1.000**	0.000
Margariene	**1.000**	**1.000**	**1.000**	0.000
Brunch - Aufstrich	**1.000**	0.995	0.959	0.000
Flasche - Bier	**0.997**	0.970	0.986	0.000
mAP	**0.991**	0.987	**0.980**	0.000

wurde der erste Datensatz verwendet und enthält die Informationen des originalen Datensatzes und des Gray-World Algorithmus. Im genauen Betrachten fällt auf, dass kaum nennenswerte Unterschiede entstanden sind. In manchen Punkten können Verbesserungen in der Genauigkeit ausgewertet werden(Wasser, Orangensaft, Bier), in anderen wiederum ist die Genauigkeit gesunken(Milch, Brunch, mAP).
In der Tabelle sind die beiden Histogramm Normalisierungen (Ausgleich und Spezifizierung) aufgeführt. Auch hier sind die Unterschiede nur in einzelnen Objektklassen besser geworden und in anderen zurückgegangen. Die Varianz der verschiedenen Netze liegt bei gerade mal 0.05, was in der Praxis keinen bedeutenden Unterschied macht.

4.2 PascalVOC Datensatz

Beim Auswerten der Genauigkeit des normalen Modells fällt auf, das teilweise große Unterschiede der einzelnen Objektklassen auftreten. Das kann daher resultieren, dass unterschiedlich viele Bilder pro klasse enthalten sind. Ähnlich wie beim vorherigen Datensatz konnte zwischen dem originalen Datensatz und der Gray-World-Normalisierung kein deutlicher Unterschied festgestellt werden. Einige Klassen haben zwar eine höhere durchschnittliche Wahrscheinlichkeit, dennoch gibt es auch hier mehrere Bereiche, wo die Genauigkeit abgenommen hat. Auch die mAP hat ca. 0.8% Genauigkeit verloren. Anders als beim Nahrungsmittel Datensatz schneidet der Histogramm Ausgleich schlechter ab. Nur ein paar Objektklassen konnten besser erkannt werden. Insgesamt verliert das neuronale Netz 3.4% der Genauigkeit, was einen wichtigen Unterschied macht. Einige Objektklassen sinken durch die Ausgleichung unter die 50% Genauigkeit, was vergleichbar mit raten wäre.

Tabelle 12: *Genauigkeitsberechnungen des Trainierten Modells mit Histogramm Ausgleich und Spezifizierung*

Objektklasse	AP (N)	AP(GW)	AP(HA)	AP(HS)
sofa	0.611	0.612	**0.637**	0.000
aeroplane	0.845	**0.855**	0.826	0.000
horse	**0.924**	0.910	0.891	0.000
train	0.790	**0.804**	0.797	0.000
bird	**0.750**	0.733	0.691	0.000
tvmonitor	0.729	**0.733**	0.725	0.000
boat	**0.680**	0.644	0.626	0.000
pottedplant	**0.430**	0.415	0.425	0.000
bus	0.797	**0.810**	0.763	0.000
diningtable	0.532	**0.545**	0.507	0.000
car	0.812	**0.815**	0.789	0.000
bottle	**0.553**	0.510	0.498	0.000
cat	**0.903**	0.902	0.872	0.000
person	**0.791**	0.779	0.775	0.000
chair	0.499	**0.515**	0.460	0.000
bicycle	0.744	0.730	**0.752**	0.000
cow	**0.718**	0.698	0.694	0.000
motorbike	**0.739**	0.722	0.725	0.000
dog	**0.867**	0.862	0.835	0.000
sheep	0.645	0.619	**0.664**	0.000
mAP	**0.718**	0.710	0.698	0.000

4.3 Stanford Dogs Dataset

Der dritte Datensatz welcher in dieser Arbeit verwendet wird, ist der Hunde Datensatz aus Stanford und beinhaltet 120 verschiedene Hunderassen. Jede Klasse hat um die 200 Bilddaten. Wegen der Struktur des Datensatzes, in einzelnen Ordnern, kann eine Auswahl der priorisierten Hunderassen zusammengestellt werden. Für den Versuch und damit eine bessere Übersicht erreicht werden kann, wurden 20 Hunderassen ausgewählt. Die Annotationen der Daten sind in Form von XML Dateien beigefügt. In der folgenden Tabelle werden die Klassen und Ergebnisse der Tests aufgeführt.

Tabelle 13: *Genauigkeitsberechnungen des Trainierten Modells mit Histogramm Ausgleich und Spezifizierung*

Objektklasse	AP (N)	AP(GW)	AP(HA)	AP(HS)
Chihuaua	0.729	**0.854**	0.000	0.000
Shih-Tzu	0.864	**0.879**	0.000	0.000
Rhodesian Ridgeback	**0.741**	0.706	0.000	0.000
Bloodhound	0.872	**0.878**	0.000	0.000
Redbone	**0.617**	0.546	0.000	0.000
Japanese Spaniel	0.886	**0.919**	0.000	0.000
Blenheim Spaniel	**0.980**	0.966	0.000	0.000
Afghan Hound	**0.955**	0.946	0.000	0.000
Bluetrick	**0.897**	0.892	0.000	0.000
Borzoi	**0.931**	0.922	0.000	0.000
Maltese dog	**0.905**	0.901	0.000	0.000
Papillon	0.971	**0.984**	0.000	0.000
Basset	0.763	**0.779**	0.000	0.000
Coonhound	0.863	**0.864**	0.000	0.000
Irish Wolfhound	**0.945**	0.930	0.000	0.000
Pekinese	0.723	**0.840**	0.000	0.000
Toy Terrier	0.838	**0.856**	0.000	0.000
Beagle	0.725	**0.743**	0.000	0.000
Walker Hound	**0.767**	0.705	0.000	0.000
Italian Greyhound	0.822	**0.845**	0.000	0.000
mAP	0.840	**0.848**	0.000	0.000

5 Diskussion

Die Ergebnisse der trainierten Netze zeigen, das mit die verwendeten Datensätzen zusammen mit den angewandten Normalisierungsmethoden keine Erhöhung der Genauigkeit erzielen. Ein möglicher Grund könnte zum einen in den vortrainierten Netzen liegen, welche schon zu genau und hochwertig eingestellte Perzeptronen besitzen und durch die vorherigen Datensätze nicht optimal für den Versuch waren. Zum anderen Könnte es an den Datensätzen selber liegen welche Normalisiert wurden. Dafür wurden Objekte mit gleicher Lichteinstrahlung und gleicher Entfernung auf unterschiedlichen Hintergründen Aufgenommen. Das Objekt wurde vom Hintergrund segmentiert, um zu überprüfen wie sich die Histogramme voneinander unterscheiden (Abb.). Dieser Vorgang wurde daraufhin auch mit der Histogramm Ausgleichung überprüft (Abb.).

Abbildung 11: *Links: das aufgenommene Bild im Datensatz ohne Normalisierung, Mitte: Segmentiertes Objekt aus dem Hintergrund entnommen, Rechts: Histogramm inkl. Rot-, Grün-, Blau-Kanal*

Bei den Normalen Datensätzen, ist aufgefallen, das bei gleicher Lichteinstrahlung und unterschiedlichen Hintergründen die Objekte eine unterschiedliche Farbverteilung aufweisen. Das könnte durch Lichtreflexionen anderer Objekte in der Umgebung erklärt werden. Der Normalisierte Datensatz weißt sogar noch größere Unterschiede in den Klassen auf. Das liegt daran, dass Die Normalisierung nicht nur die Objekte auf den Bildern normalisiert, sondern auch den Hintergrund, welcher bei vielen Bildern unterschiedlich ist verrechnet. Das bedeutet, das nicht der Datensatz untereinander Normalisiert wurde, sondern jedes Bild in dem Datensatz für sich.

Die Datensätze sind mit dem verwendeten Aufbau und den Normalisierungs Algorithmen nicht gut geeignet. Die Kriterien für einen Datensatz welcher durch Farbnormalisierung eine höhere Genauigkeit erreichen könnte, liegt in dem Hintergrund welcher verwendet wird. Dabei müssen alle Klassen mit dem selben Hintergrund aufgenommen werden. Dadurch würden alle Klassen auf der selben Basis normalisiert werden. Gerade die Histogramm Spezifikation (Absch. 2.4.3) hätte hier ein großes Potenzial, da die Auswahl eines Refe-

renzbildes einfacher und effektiver wird, durch den einheitlichen Hintergrund. Um diese Theorie zu überprüfen, wird ein weiterer Datensatz generiert und gelabelt.

6 Fazit

Literatur

[Aih12] Shunsuke Aihara. *color correction algorithm in python*. 2012. URL: https://gist.github.com/shunsukeaihara/4603234.

[Ang18] Bistra Angelova. *Optical character recognition für chinesische Schrift*. 2018. URL: http://pille.iwr.uni-heidelberg.de/~ocr01/nnet.html.

[BB09] Wilhelm Burger und Mark James Burge. *Digitale Bildverarbeitung: Eine Algorithmische Einführung Mit Java*. Springer-Verlag, 2009.

[Con18] Common Objects in Context. *COCO Datenset*. 2018. URL: http://cocodataset.org/#home.

[CSD18] University of California San Diego. *Why are neuron axons long and spindly? Study shows they're optimizing signaling efficiency*. 2018. URL: https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html.

[Dee16] DeepMind. *Google DeepMind Challenge Match: Lee Sedol vs AlphaGo*. 2016. URL: https://www.youtube.com/watch?v=l-GsfyVCBu0.

[Ert13] Wolfgang Ertel. *Grundkurs künstliche Intelligenz: eine praxisorientierte Einführung*. Springer-Verlag, 2013.

[Eve+] M. Everingham u. a. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. http://www.pascal-network.org/challenges/VOC/voc2007/worksho

[Goo+16] Ian Goodfellow u. a. *Deep learning*. Bd. 1. MIT press Cambridge, 2016.

[HE16] Joachim Hoffmann und Johannes Engelkamp. *Lern-und Gedächtnispsychologie*. Springer-Verlag, 2016.

[Jaz16] Mohammad Al Jazaery. *OpenCV Python equalizeHist colored image*. 2016. URL: https://stackoverflow.com/questions/31998428/opencv-python-equalizehist-colored-image.

[Kho+11] Aditya Khosla u. a. „Novel Dataset for Fine-Grained Image Categorization". In: *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*. Colorado Springs, CO, 2011.

[LLC18] Google LLC. *Tensorflow detection model zoo*. 2018. URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

[Mis19] Mayank Mishra. *Convolutional Neural Networks, Explained*. 2019. URL: https://www.datascience.com/blog/convolutional-neural-network.

[Pen+16] Min Peng u. a. „NIRFaceNet: A Convolutional Neural Network for Near-Infrared Face Identification“. In: *Information* 7.4 (2016). ISSN: 2078-2489. DOI: 10.3390/info7040061. URL: https://bit.ly/2GGNqUP.

[SCL12] Pierre Sermanet, Soumith Chintala und Yann LeCun. „Convolutional neural networks applied to house numbers digit classification“. In: *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE. 2012, S. 3288–3291.

[ST13] Robert F Schmidt und Gerhard Thews. *Physiologie des Menschen*. Springer-Verlag, 2013.

[tzu19] darrenl tzutalin. *labelimg*. 2019. URL: https://github.com/tzutalin/labelImg.

[Wik18] Computer science Wiki. *Max-pooling / Pooling*. 2018. URL: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling.

[Wik19] Wikipedia. *Künstliches Neuron*. 2019. URL: https://de.wikipedia.org/wiki/K%C3%BCnstliches_Neuron.

Anhang

Eidesstattliche Erklärung

Ich versichere, dass ich die Bachelorarbeit selbständig angefertigt und keine anderen als die von mir angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt und die vorliegende Arbeit an keiner anderen Stelle zur Erlangung eines Abschlusses vorgelegt habe.

Datum, Ort

Unterschrift