hochschule hof
University of Applied Sciences

# Angewandtes maschinelles Lernen – SS2019

Künstliche Intelligenz (KI) und maschinelles Lernen sind Themen, welche aktuell sehr stark im öffentlichen Fokus liegen. Moderne Rechentechnik macht es möglich, aufwendige Optimierungsrechnungen zeitnaher durchführen zu können als noch vor 10 oder 20 Jahren. Dies hat zur Folge, dass die technischen Anforderungen so stark gesunken sind, dass man fast auf jedem Rechner heute ein KI-Modell berechnen kann. Ebenso ermöglicht die gestiegene Performance, dass die Größe und die Komplexität von KI-Modellen deutlich steigen darf während die Zeit zum berechnen des Modells weiterhin akzeptabel bleibt.

Aufbauend auf diesen Entwicklungen entstanden in den vergangenen Jahren sehr viele Werkzeuge. Diese erlauben selbst unerfahrenen Nutzern einen schnellen Einstieg in die Welt des maschinellen Lernens. Hierbei kommt meist das Prinzip eines Baukastens zum Einsatz. Der Nutzer bedient sich fertiger Bausteine, um ein Modell zu erstellen. Ist es erstellt kann er es testen und gegebenenfalls umbauen. Aus wissenschaftlicher Sicht wäre es erstrebenswert, wenn der Nutzer in der Lage wäre zu verstehen, was den Baustein ausmacht und warum er sich so verhält oder wie er weitere Bausteine eigenständig erstellen kann. Das würde die Einstiegshürde in den Bereich der KI aber deutlich erhöhen. Im Bereich des Wohnungsbaus wird von einem Bauunternehmer ja auch nicht erwartet, dass er die Zusammensetzung von Ziegelsteinen versteht. Wichtiger ist, dass er in der Lage ist einem vorgegebenen Bauplan zu folgen und die vorhandenen Elemente korrekt zu verbauen.

Genau dieser Einstieg in die Materie des maschinellen Lernens wurde von den Studenten evaluiert. Jeder Student bzw. jede Gruppe von Studenten hat sich eine Aufgabenstellung ausgewählt und diese versucht mit bestehenden Modellen zu lösen beziehungsweise vorhandene Modelle weiter zu adaptieren. Die Themen erstrecken sich auf sehr unterschiedliche Bereiche: beginnend bei der Erkennung von Personen oder deren Emotionen, über die Bewertung von Geräuschen, bis hin zur Entwicklung von eigenständigen Spielern, welche selbstständig lernen ein Spiel zu meistern, aber auch die Identifikation von Meinungen aus Texten oder die Generierung von Texten oder Beschreibung von Bildern wurde umgesetzt.

Ziel dieser Veröffentlichung ist es den Leser zu motivieren die Beispiele nachzuvollziehen und selbst Erfahrungen mit KI zu sammeln. Hierzu sind auch die Softwarequellen frei zugänglich.

**Sebastian Leuoth (Hrsg.)**
**Hochschule für Angewandte Wissenschaften Hof, 2019**
**ISBN: 978-3-935565-17-2**

# Inhaltsverzeichnis

# Leitfaden zum Entwurf eines Convolutional Neural Networks

Eine kurze Einführung in den Bereich der maschinellen Bildverarbeitung

Robin Reiß
Fakultät Informatik
Hochschule Hof
Hof, Bayern, Deutschland
robin.reiss@hof-university.de

## ZUSAMMENFASSUNG

Die vorliegende Studienarbeit dient als Leitfaden zur Entwicklung von *Convolutional Neural Networks* (*CNNs*). Nach einer kurzen Einführung in den Bereich der „Künstlichen Intelligenz“, werden zunächst die Fachbegriffe „Maschinelles Lernen“, „Muster- und Bilderkennung“ und „Künstliche Neuronale Netzwerke“ genauer erklärt. Anschließend werden die Konzepte und Einsatzgebiete von CNNs vorgestellt, sodass auch nachfolgende Erläuterungen in Bezug auf den schichtenbasierten Entwurf von CNNs verstanden werden können. Aufbauend auf diesen Erkenntnissen, werden mit Hilfe der Skriptsprache *Python* [1] und der Open Source Deep-Learning-Bibliothek *Keras* [2] mehrere verschiedene CNNs zur Klassifikation von Hunderassen implementiert und auf ihre Vor- und Nachteile überprüft. Hierbei wird Bildmaterial von zehn verschiedenen Hunderassen aus dem *Stanford Dogs Dataset* [3] verwendet. Durch die detaillierten Erklärungen zu jedem Bestandteil der verschiedenen CNNs und der Visualisierung von internen Vorgängen ist es dem Leser jetzt nicht nur möglich die Konzepte der maschinellen Bilderkennung auf Basis von künstlichen neuronalen Netzwerken zu verstehen, sondern auch auf eigene Probleme anzuwenden und dabei größtenteils gute Ergebnisse zu erzielen.

## 1 Einleitung

### 1.1 Motivation

Bereits im 18. Jahrhundert beschäftigte sich der französische Arzt und Schriftsteller *Julien Offray de La Mettrie* in seinem Werk *L'Homme Machine* mit dem Gedanken, ob sich die menschliche Intelligenz oder auch Vorgänge des menschlichen Denkens automatisieren oder mechanisieren lassen können. (vgl. [1])

Seitdem er sein Werk im Jahre 1748 veröffentlichte, sorgten seine Ideen für heftige Kontroversen und bescherten ihm dadurch anhaltende Popularität. Heute gilt sein berühmtestes Werk als einer der theoretischen Vorläufer der „Künstlichen Intelligenz“. (vgl. [1, 2])

Seinen Anfang hatte das akademische Fachgebiet „Künstliche Intelligenz“ jedoch erst über 200 Jahre später in den Vereinigten Staaten von Amerika. Dort wurde es im Rahmen der Dartmouth Conference, die im Jahr 1956 am Dartmouth College in Hanover (New Hampshire) abgehalten wurde, gegründet. (vgl. [1, 3])

Der Themenkomplex ist von diesem Zeitpunkt an Gegenstand aktueller Forschung. Als Teilgebiet der Informatik befasst sich der Bereich „Künstliche Intelligenz“ mit der Automatisierung von intelligenten Verhalten und dem „Maschinellen Lernen“. (vgl. [4])

Von *Smart Predictive Maintenance* in der Industrie 4.0, über Sprachassistenten in Smartphones bis hin zu autonomen Fahrzeugen ist künstliche Intelligenz, genauer gesagt maschinelles Lernen, nicht mehr aus unserem Alltag wegzudenken. Durch die steigende Relevanz der Bilderkennung insbesondere in lebenswichtigen Bereichen, wie z.B. der Medizin, nimmt sich der Autor der Studienarbeit diese Entwicklung zum Anlass, um eines der gängigsten und gleichzeitig vielversprechendsten Konzepte der Bildverarbeitung im Bereich des maschinellen Lernens, nämlich *Convolutional Neural Networks* (CNNs), genauer zu betrachten.

### 1.2 Projektumfeld

Dieses Projekt wurde in Form einer Studienarbeit im Rahmen des Moduls „Angewandtes maschinelles Lernen“ im Sommersemester 2019 an der Fakultät Informatik der Hochschule Hof erstellt. Die Veranstaltung wird als seminaristischer Unterricht durchgeführt. Dozent der im Verlauf des Moduls stattfindenden Vorlesungen ist Herr Prof. Dr. Sebastian Leuoth. Neben seiner Tätigkeit als Leiter des Seminars fungiert er ebenfalls als Ansprechpartner für alle technischen Fragen rund um das Thema „Maschinelles Lernen“ und als Betreuer der verschiedenen Projektgruppen.

In den Vorlesungen der Veranstaltung lernen die Studierenden neben den grundlegenden Begriffen des maschinellen Lernens und der korrekten Vorverarbeitung von Daten auch die drei Teilgebiete „Überwachtes Lernen“, „Unüberwachtes Lernen“ und „Bestärkendes Lernen“ des maschinellen Lernens kennen. Ziel der Veranstaltung ist es, den Studierenden eine Einführung in das Gebiet des maschinellen Lernens zu geben und die vermittelten Inhalte in der Praxis testen zu lassen. (vgl. [5])

[1] Weiterführende Informationen: https://www.python.org/
[2] Weiterführende Informationen: https://keras.io/
[3] Weiterführende Informationen: http://vision.stanford.edu/aditya86/ImageNetDogs/

Um die vom Autor der vorliegenden Arbeit erlangten Kenntnisse im Bereich des angewandten maschinellen Lernens aufzuzeigen, wurde im Rahmen dieses Projekts ein Leitfaden zum Entwurf von *Convolutional Neural Networks* (CNNs) erstellt. Dieser soll dem Leser zunächst den Einstieg in den Bereich der Bildklassifizierung erleichtern und darüber hinaus Empfehlungen zum Entwurf und zur Verwendung von *Convolutional Neural Networks* bieten.

Hierfür werden die verschiedenen Bestandteile von CNNs und deren Eigenschaften besprochen und im Anschluss anhand eines frei verfügbaren Datensatzes mit Hilfe von mehreren konkreten Implementierungen evaluiert.

## 2 Maschinelles Lernen

Maschinelles Lernen (engl.: Machine Learning) ist ein Teilgebiet der „Künstlichen Intelligenz“ und damit auch Teil der Informatik. Der Themenbereich befasst sich mit dem Erwerb neuen Wissens durch IT-Systeme. Diese erlangen ihr „künstliches“ Wissen durch Algorithmen, welche in vorhandenen Datenbeständen Muster und Gesetzmäßigkeiten erkennen können. Die gewonnen Erkenntnisse lassen sich durch das System verallgemeinern, sodass es auch bei neuen sowie unbekannten Problemstellungen etwaige Zusammenhänge erkennen und Rückschlüsse ziehen kann. Das menschliche Lernen funktioniert nach ähnlichen Prinzipien. (vgl. [6, 7])

Bevor nun Vorhersagen getroffen oder Lösungen zu Problemstellungen gefunden werden können, müssen der Software des IT-Systems zunächst alle für das Lernen benötigte Daten und auf den Anwendungsfall abgestimmte Algorithmen zur Verfügung gestellt werden. Mit diesen wird die Software dann anschließend trainiert. Die verwendeten Algorithmen lassen sich hierbei grob in die drei Kategorien „Überwachtes Lernen“ (engl.: Supervised Learning), „Unüberwachtes Lernen“ (engl.: Unsupervised Learning), sowie „Bestärkendes Lernen“ (engl.: Reinforcement Learning) einteilen. (vgl. [6, 7, 8])

Ziel des überwachten Lernens ist das Finden einer Funktion, mit deren Hilfe man unbekannte Beobachtungen einer bestimmten Klasse (Klassifikation) oder einem bestimmten Wert (Regression) zuweisen kann. Hierfür werden zunächst alle vorhandenen Daten mit einer zugehörigen Beschriftung (engl.: label) versehen und der zu trainierenden Software übergeben. Anhand der Beschriftungen kann das System seine Ergebnisse (z.B. Objekt in einem Bild ist ein Apfel) auf Korrektheit überprüfen (z.B. Objekt in einem Bild war mit „Tomate“ beschriftet) und Anpassungen an den zugrundeliegenden Parametern seines Lernprozesses vornehmen. Mit einer ausreichend großen Menge von Eingabedaten kann das System nun Muster erkennen, die z.B. zum anschließenden Klassifizieren von Objekten in Bildern (z.B. der Hund auf einem Bild gehört zu einer bestimmten Hunderasse) dienen können. (vgl. [6, 9])

Beim unüberwachten Lernen hingegen werden keine gelernten Zusammenhänge in unbekannten Beobachtungen gesucht, sondern eigenständig neue Muster erkannt und in Beziehungen zueinander gesetzt. Dieses Verfahren wird auch „Clustering“ genannt und ist im weitesten Sinne eine Klassifizierung. (vgl. [6, 9])

Als eine der drei wichtigsten Methoden des maschinellen Lernens gliedert sich nun ebenfalls noch das „Bestärkende Lernen“ (engl.: Reinforcement Learning) zwischen den beiden bereits genannten Lernverfahren ein. Anders als bei den Methoden des überwachten und unüberwachten Lernen setzt bestärkendes Lernen keine Daten voraus. Diese werden nämlich eigenständig von einem Software-Agenten[4] während dessen Training in einer Simulationsumgebung erzeugt und mit entsprechenden Beschriftungen versehen. Hierbei kommt die sog. „Trial-and-Error“-Methode zum Einsatz. Agenten tätigen im Rahmen dieser Methode Aktionen und erhalten von der Simulationsumgebung im Anschluss eine Rückmeldung. Ist eine Aktion zielführend, so erhält der Agent für diese eine Belohnung, andernfalls ändert er anhand von bereits gewonnenen Erfahrungen seine anfängliche Strategie. Ziel des Lernverfahrens ist somit die Maximierung der Anzahl von Belohnungen in der Simulationsumgebung. Mit Hilfe von passenden Algorithmen wird der Agent nach und nach bessere Strategien entwickeln, um dieses Ziel auch zu erreichen. Da dieses Konzept auf eine Vielzahl von Problemen angewendet werden kann, ergeben sich hieraus enorme Potenziale für das bestärkende Lernen. (vgl. [8])

## 3 Muster- und Bilderkennung

Mustererkennung (engl.: Pattern Recognition) ist im Allgemeinen die Fähigkeit Muster, Gesetzmäßigkeiten oder Ähnlichkeiten in einer Menge von Daten zu erkennen. In der Informatik bezeichnet man damit hingegen ein Verfahren, welches gemessene Signale automatisch kategorisieren bzw. klassifizieren kann. (vgl. [10])

Mit Hilfe dieser Mustererkennungsverfahren ist es Computern und anderen Maschinen möglich auch weniger präzise Signale aus der natürlichen Umgebung zu verarbeiten. Dies ist zum Beispiel im Bereich der Sprach- und Gesichtserkennung der Fall. In beiden Anwendungsgebieten lassen sich nämlich keine präzisen Daten erheben. (vgl. [10])

Die bei der Musterkennung verwendeten Verfahren lassen sich dabei in drei verschiedene Ansätze einteilen. Der älteste ist hierbei die „syntaktische Mustererkennung“. Bei diesem Ansatz versucht man Objekte durch Folgen von Symbolen so zu beschreiben, dass Objekte mit identischen Beschreibungen in die gleiche Kategorie einsortiert werden können. Da die syntaktische Mustererkennung vor allem bei komplexen Sachverhalten keine zufriedenstellenden Lösungen bietet, sondern das grundlegende Probleme nur hinauszögert, findet dieser Ansatz nur noch wenig Beachtung. (vgl. [10])

Anders verhält es sich mit dem statistischen Ansatz. Dieser ist der meistgenutzte Ansatz der Mustererkennung und beinhaltet den größten Teil der heutigen Standardmethoden. Zu diesen Methoden gehören neben sog. „Support Vector Machines“[5] auch künstliche neuronale Netzwerke, die im nachfolgenden Kapitel noch genauer betrachtet werden. (vgl. [10])

[4] “ […] Computerprogramm, das zu gewissem (wohl spezifiziertem) eigenständigem und eigendynamischem (autonomem) Verhalten fähig ist.” [33]

[5] Weiterführende Informationen: https://de.wikipedia.org/wiki/Support_Vector_Machine

Bei der statistischen Mustererkennung versucht man prozentuale Zugehörigkeiten eines Objekts zu einer oder mehrerer Kategorien zu bestimmen. Nach der Berechnung aller Wahrscheinlichkeiten wird das Objekt in die Kategorie mit der höchsten Zugehörigkeit einsortiert. Die Merkmale werden nun nicht mehr nach definierten Regeln ausgewertet, sondern als numerische Werte gemessen und in einen sog. „Merkmalsvektor“ gebündelt. Dieser wird daraufhin als Parameter an eine mathematische Funktion übergeben, welche den Merkmalsvektor einer der vordefinierten Kategorien eindeutig zuordnet. (vgl. [10])

Als Letzter der drei zu benennenden Ansätze kombinieret die strukturelle Mustererkennung verschiedene syntaktische und/oder statistische Verfahren zu einem einzigen neuen Verfahren. Für die Merkmalsklassifikation werden hierbei in erster Linie statistische Verfahren verwendet. Durch die entstandenen Einzelergebnisse in Form von prozentualen Zugehörigkeiten erhält man mit Hilfe von übergeordneten strukturellen Verfahren ein Gesamtergebnis, das wiederum der Zugehörigkeit eines bestimmten Objektes zu einer Kategorie entspricht. Strukturelle Verfahren werden vor allem bei sehr komplexen Sachverhalten, wie bei bildgebenden Verfahren in der Medizin, eingesetzt. (vgl. [10, 11])

Da die Definitionen der drei Verfahren noch keine Einblicke in die zugrundeliegenden Mustererkennungsprozesse bieten, werden diese nun etwas genauer betrachtet. Grundsätzlich lassen sich die Prozesse in mehrere verschiedene Teilschritte zerlegen, bei denen am Anfang die Erhebung und Digitalisierung von analogen Daten steht. Sollten die Daten bereits digitalisiert vorliegen, so entfallen die Schritte der Digitalisierung. Damit die gewonnenen Daten frei von unerwünschten oder irrelevanten Bestandteilen sind, findet im Anschluss an die Datenerhebung eine Vorverarbeitung statt. Diese kann beispielsweise dazu dienen, Störsignale in Audiomaterial zu verringern und dadurch das Erkennen von Mustern zu verbessern. Auf die Vorverarbeitung folgt nun die Merkmalsextraktion. Dabei werden Merkmale aus den aufbereiteten Daten gewonnen und zur Merkmalsreduktion weitergeleitet. In diesem Prozess werden alle Merkmale auf ihre Relevanz bei der Klassifizierung von Objekten untersucht. Ist ein bestimmtes Merkmal für die Klassifikation von Objekten unrelevant, so wird es zukünftig nicht mehr verwendet. Im letzten Teilschritt findet nun die tatsächliche Klassifikation der Objekte anhand verschiedener Klassifikationsverfahren, wie dem überwachten Lernen aus dem Bereich des maschinellen Lernens, statt. Im Bereich der Bilderkennung, die ebenfalls ein Teilgebiet der Mustererkennung ist, folgt auf die Klassifizierung von Bildern nur noch die Segmentierung von Objekten in diesen. Damit lassen sich Bilder nicht nur einer Kategorie zuordnen, sondern auf Grund der enthaltenen Objekte, gleich mehrerer. Die gefundenen Objekte werden genauso wie bei der „normalen“ Mustererkennung weder interpretiert noch auf etwaige Zusammenhänge untersucht. Solche Aufgaben sind Gegenstand der Musteranalyse[6]. (vgl. [10, 12])

Die gerade beschriebenen Einzelschritte stellen zusammen mit den drei genannten Ansätzen der Mustererkennung den Ausgangspunkt für alle weiteren Vorgehen in dieser Studienarbeit dar.

[6] Weiterführende Informationen: https://de.wikipedia.org/wiki/Musteranalyse

## 4 Künstliche Neuronale Netzwerke

Künstliche neuronale Netzwerke (engl.: artificial neural network), auch künstliche neuronale Netze (KNN) genannt, sind Netzwerke aus künstlichen Neuronen. Sie sind dem biologischen Vorbild der natürlichen Neuronen bzw. Nervenzellen im Nervensystem eines Lebewesens nachempfunden und dienen, anders als dieser Begriff suggerieren möchte, nicht etwa zur Nachbildung von biologischen neuronalen Netzen, sondern zum Lösen von konkreten Problemen aus Bereichen wie z.B. der Statistik, den Ingenieurwissenschaften oder den Wirtschaftswissenschaften. (vgl. [13, 14])

KNNs werden im Wesentlichen bei Anwendungen eingesetzt, bei denen kein oder nur geringes explizites Wissen[7] über das zu lösende Problem vorhanden ist. Dazu zählt z.B. die Gesichts- oder Bilderkennung. In beiden Bereichen gibt es bis zum heutigen Tag keinen Algorithmus, der Bilder oder Teile davon zu 100% korrekt klassifizieren kann. Mit CNNs, einer Sonderform der künstlichen neuralen Netze, lassen sich allerdings bereits Ergebnisse erzielen, die selbst Menschen übertreffen[8]. (vgl. [13, 15])

Damit KNNs Problemstellungen eigenständig lösen können, müssen diese zunächst mit Hilfe von Software entworfen werden. Dabei sollte die Topologie, also der Aufbau bzw. die Struktur, des Netzes so gewählt werden, dass es seine vorgegebenen Aufgaben möglichst effizient und korrekt bearbeitet. Grundsätzlich bestehen die abstrahierten Modelle der künstlichen neuronalen Netzwerke dabei aus künstlichen Neuronen, die sich aus den in Abbildung 1 dargestellten Bestandteilen zusammensetzen. (vgl. [13])

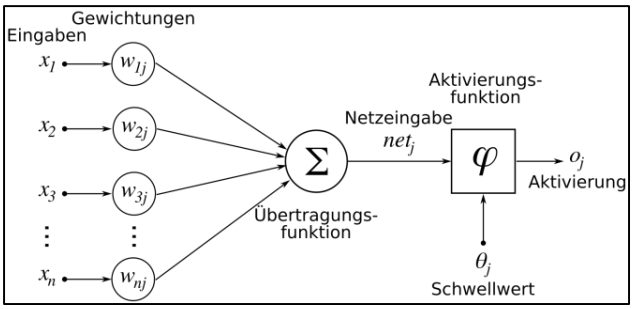



**Abbildung 1: Bestandteile eines künstlichen Neurons [16]**

Ein künstliches Neuron $j$ ist wie sein biologisches Vorbild in der Lage mehrere verschiedene Eingaben $(x_1, x_2, \dots x_n)$ zu verarbeiten und über seine sog. Aktivierung $(o_j)$ auf diese zu reagieren. Dabei bestimmen Gewichtungen $(w_{ij})$ wie groß der Einfluss einer jeden Eingabe $(x_i)$ in der Berechnung der späteren Aktivierung ist. Mit Hilfe der Übertragungsfunktion Σ werden dabei alle Eingaben mit ihrer entsprechenden Gewichtung $(w_{1j}, w_{2j}, \dots w_{nj})$ multipliziert und anschließend zu einem Wert $(net_j)$ aufsummiert. (vgl. [16])

[7] Weiterführende Informationen: https://de.wikipedia.org/wiki/Explizites_Wissen
[8] Weiterführende Informationen: https://www.sueddeutsche.de/wissen/bilderkennung-tiefer-blick-1.4322209

Dieser Wert wird als die Netzeingabe des Neurons bezeichnet und mit einem Schwellenwert $\theta_j$ an eine Aktivierungsfunktion $\varphi$ übergeben. Bevor dies geschieht, wird der Schwellenwert aber noch zur Netzeingabe addiert, wodurch sich die gewichteten Eingaben um einen gewissen Betrag verschieben. Anschließend wird durch die Aktivierungsfunktion die Ausgabe eines Neurons, auch dessen Aktivierung genannt, berechnet und z.B. an andere Neuronen oder die Umwelt bzw. die zugrundeliegende Software übermittelt. Als Aktivierungsfunktion können dabei verschiedene Funktionstypen verwendet werden. (vgl. [16])

In dieser Arbeit werden jedoch nur die Funktionen „Rectifier“ und „Softmax“ verwendet. Die Aufgabe der Aktivierungsfunktion Rectifier, (dt.: Gleichrichter) ist dabei recht trivial. Alle Eingaben, die kleiner als der Wert 0 sind erhalten den Wert 0; Eingaben, die größer als 0 sind behalten ihren ursprünglichen Wert. Abbildung 2 soll dies noch einmal verdeutlichen. (vgl. [17])

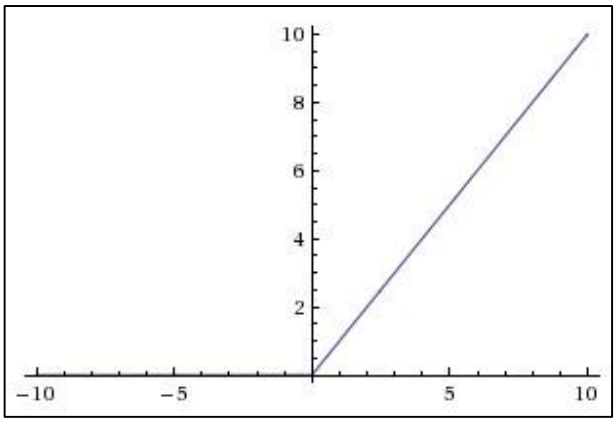



**Abbildung 2: Rectifier-Funktion [18]**

Während die Rectifier-Funktion vor allem die Trainingsdauer von größeren KNNs verringert (vgl. [19]), wird die Softmax-Funktion hauptsächlich zur Berechnung von kategorischen Wahrscheinlichkeitsverteilungen benutzt. Hierbei werden alle Eingaben zwischen 0 und 1 normiert, sodass durch eine anschließende Division Werte entstehen, die in Summe 1 ergeben. Die Einzelwerte entsprechen der prozentualen Zugehörigkeit eines Objektes zu einer Kategorie. (vgl. [17])

Um aus den genannten Neuronen nun ein KNN zu entwickeln, werden diese über Kanten verbunden. Eine Kante führt dabei vom Ausgang bzw. der Aktivierung eines Neurons zum Eingang eines anderen Neurons. Erst durch solche Verknüpfungen gelingt es den KNN intelligentes Verhalten aufzuweisen. Etwaige Beweise oder Erklärungen zur Plausibilität dieses Verhaltens sind indessen nicht mehr Teil dieser Arbeit und können ggf. in Fachliteratur gefunden werden. (vgl. [13])

Weiterhin gilt anzumerken, dass KNNs in der Praxis typischerweise in Schichten (engl.: layer) aus Neuronen organisiert werden und hintereinander angeordnet sind. Der erste Layer eines Netzes wird dabei „input layer“ genannt. Er nimmt Informationen aus der Umwelt auf und gibt diese an seine Neuronen weiter. (vgl. [18])

Die Neuronen berechnen daraufhin ihre jeweiligen Aktivierungen und leiten diese über Verbindungen bzw. Kanten an den nächsten Layer weiter. Bei diesem handelt es sich in den meisten Fällen um einen sog. „hidden layer“, dessen Ausgaben bzw. Aktivierungen wieder mit den Eingängen eines weiteren Layers verbunden sind. Am Ende dieser Kette folgt immer noch ein „output layer“, dessen Aktivierungen wieder an die anfängliche Umwelt zurückgegeben werden. (vgl. [18])

Abbildung 3 soll diesen schichtenbasierten Aufbau anhand von voll verbundenen Schichten (engl.: „fully-connected layers“) noch einmal veranschaulichen.

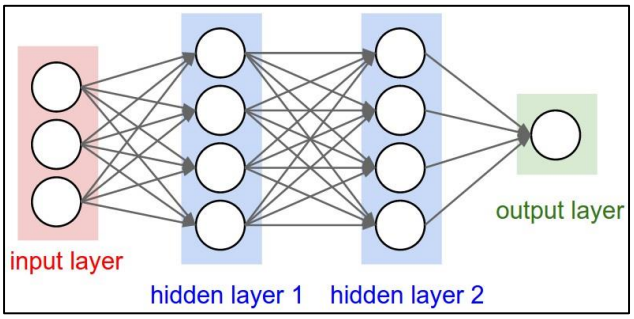



**Abbildung 3: KNN mit voll verbundenen Schichten [18]**

Neben den in Abbildung 3 dargestellten Aufbau existieren jedoch noch viele weitere Strukturen, die teilweise weder voll verbunden noch aus Input- oder Hidden-Layers bestehen. Auch diese werden nicht mehr in der vorliegenden Arbeit behandelt.

Da neuronale Netze in den Bereich des maschinellen Lernens einzuordnen sind, müssen diese nach ihrem Entwurf, wie schon in Kapitel 2 besprochen, trainiert werden. Ohne ein entsprechendes Training sind sie nicht zur Lösung von Problemstellungen fähig. Die Lernerfolg von KNNs resultiert dabei hauptsächlich aus der Modifikation der Gewichtungen ihrer Neuronen. (vgl. [13, 18])

Je nachdem welche Aufgaben ein KNN lösen muss bieten sich die drei in Kapitel 2 besprochenen Lernverfahren „Unüberwachtes Lernen“, „überwachtes Lernen“ und „bestärkendes Lernen“ an.

## 5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), auch faltende neuronale Netzwerke genannt, sind künstliche neuronale Netzwerke, welche von biologischen Prozessen, wie z.B. denen im visuellen Kortex, inspiriert sind. Als ihr Begründer gilt Yann LeCun. (vgl. [15, 20])

CNNs sind, genauso wie künstlich neuronale Netze, Konzepte im Bereich des maschinellen Lernens. Sie finden hauptsächlich im Bereich der maschinellen Bild- und Tonverarbeitung Anwendung und konnten dort bereits hervorragende Resultate erzielen. Grundsätzlich werden CNNs zur Klassifizierung von Daten, wie Bildern oder Audiomaterial, verwendet. Die Topologie eines CNN besteht dabei in der Regel aus Wiederholungen von sog. „Convolutional“ und „Pooling Layern“. (vgl. [15, 20])

Diese und noch weitere in der Studienarbeit verwendete Typen von Schichten werden in den folgenden Kapiteln näher erläutert.

Da CNNs, wie bereits erwähnt, bei der Klassifizierung von Daten verwendet werden, müssen diese mit einem Algorithmus aus dem Bereich des überwachten Lernens trainiert werden. Dabei benötigt man Daten, welche zuvor mit einem Label versehen worden sind. Sobald einem Lernverfahren diese Daten vorliegen, kann mit dem Training begonnen werden. Beim eigentlichen Training wird nun der statistische Ansatz der Muster- und Bilderkennung verfolgt. Zur Erzeugung von numerischen Werten und der anschließenden Bündelung dieser Werte in einem Merkmalsvektor kommen dabei wieder Aktivierungsfunktionen, wie die beiden bereits genannten Funktionen „Rectifier“ und „Softmax“, zum Einsatz.

Weiterhin werden im Verlauf der Studienarbeit keine analogen Daten erhoben und digitalisiert, da auf einen bereits vorhandenen Datensatz mit Bildern zurückgegriffen wird. (siehe Kapitel 6)

Ebenfalls finden keine Vorverarbeitungen zur Entfernung oder Reduzierung von unerwünschten Bestandteilen statt. Lediglich die Darstellung der verwendeten Bilder (z.B. Größe) wird angepasst.

Alle nicht besprochenen Teile der Muster- und Bilderkennung (Merkmalsextraktion, -reduktion und Klassifikation) finden in den nachfolgenden Typen von Schichten statt und werden zusammen mit diesen noch genauer erklärt.

## 5.1 Convolutional Layer (2D)

Ein Convolutional Layer stellt eine Schicht aus Neuronen dar, die sich an die Dimensionalität ihrer Eingabedaten anpasst. Das heißt, dass Daten, die in dreidimensionaler Form (z.B. Farbbild) an eine solche übermittelt werden, auch durch Neuronen dreidimensional repräsentiert werden. Anschließend werden die Netzeingaben der einzelnen Neuronen über eine sog. diskrete Faltung (convolution) berechnet und einer Aktivierungsfunktion übergeben. (vgl. [15])

Zur Berechnung der Netzeingabe eines Neurons kommen bei der diskreten Faltung sog. „Filterkernel“, auch „Faltungsmatrizen“ genannt, zum Einsatz. Die Größe einer solchen Matrix wird dabei softwareseitig festgelegt und steht danach fest. Die Gewichtungen einer jeden Eingabe sind durch die Faltungsmatrizen vorgegeben und werden im Laufe des Trainings unter Zuhilfenahme von sog. „Backpropagation“ angepasst. Bei der eigentlichen Faltung wird nun schrittweise eine Faltungsmatrix mit einer festgelegten Größe über die Eingabe gelegt (siehe Abbildung 4). Schrittweiten und Größen der Matrizen können dabei beliebig gewählt werden. Anschließend wird die Faltungsmatrix mit den darunterliegenden Eingaben multipliziert und die resultierenden Werte aufsummiert. Die dabei entstandene Netzeingabe wird dem Neuron zugeordnet, das auf der gleichen räumlichen Position wie der durch die Mitte des Filterkernels bestimmte Eingabewert (hier: 7) der Eingabe ist. Die aus einer anschließenden Aktivierungsfunktion resultierenden Aktivierungen lassen sich durch dieses Verfahren, welches dem biologischen Vorbild des rezeptiven Feldes folgt, als Reaktion auf Reize in einem lokalen Bereich der vorherigen Schicht auffassen. Somit werden einzelne Eingaben von den umliegenden Eingaben abhängig gemacht. Je nach Anzahl der verwendeten Filter besteht auch die Ausgabe des Convolutional Layers aus derselben Anzahl von Kanälen. Diese werden auch als „Feature Maps“ bezeichnet und stellen grundsätzlich die Muster und Merkmale dar, die durch den bereits genannten Teilschritt „Merkmalsextraktion“ der Bild- und Mustererkennung gewonnen werden sollen. (vgl. [13, 15])

Abbildung 4 und Abbildung 5 sollen den besprochenen Ablauf noch einmal verdeutlichen. Wie aus Abbildung 5 ersichtlich wird, werden in der Praxis sog. „Paddings“ verwendet, die dafür sorgen, dass alle Feature Maps die gleiche räumliche Ausdehnung wie die ursprüngliche Eingabe haben. Auch in dieser Arbeit wird Padding, bei dem alle Werte aus Nullen bestehen, verwendet. (vgl. [15, 21])

Weiterhin finden in dieser Studienarbeit nur zweidimensionale Faltungsmatrizen Anwendung, da sie gleichzeitig auf alle Kanäle einer dreidimensionalen Eingabe gelegt werden können und im Anschluss zweidimensionale Kanäle für die Weiterverarbeitung erzeugen. Dies ist durch das Zusammenfassen aller Kanäle der Eingabe zu einem einzigen zweidimensionalen Kanal möglich.

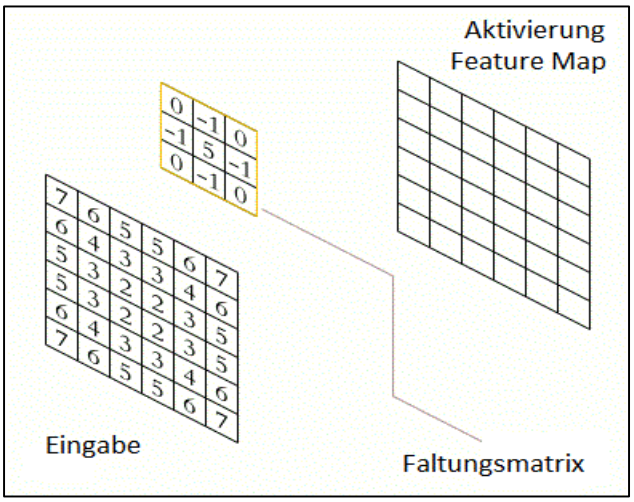



**Abbildung 4: Ausgangssituation bei einer diskreten Faltung mit einem 3x3 Filterkernel, (abgeändert) [15]**

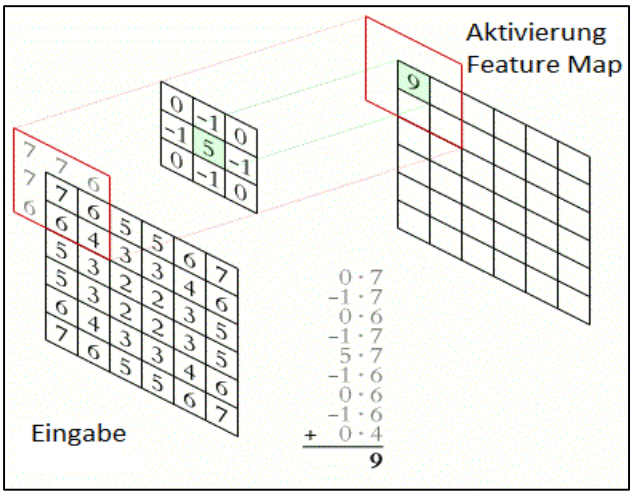



**Abbildung 5: Erzeugung von Feature Maps, (abgeändert) [15]**

## 5.2 Pooling Layer

Pooling Layer sind Schichten aus künstlichen Neuronen, in denen überflüssige Informationen verworfen werden. Sie können somit als Werkzeug für den Teilschritt *Merkmalsreduktion* der Bild- und Mustererkennung verwendet werden. Da einzelne Pixel in einem Bild von vernachlässigbarem Interesse sind stellt dieses Vorgehen auch kein Problem dar. Ausschlaggebend sind lediglich Merkmale als Ganzes. Der große Vorteil dieser Technik ist die Vergrößerung der rezeptiven Felder in den nachfolgenden Convolutional Layers. Diese sind dadurch in der Lage komplexere Features zu erkennen und arbeiten demnach ähnlich wie der visuelle Kortex im Gehirn des Menschen. (vgl. [15])

In dieser Arbeit werden zwei verschiedene Arten von Pooling Schichten verwendet. Zum einen sog. „Max Pooling“ Layer und zum anderen sog. „Global Average Pooling“ Layer. Beide werden nachfolgend genauer erklärt.

**Max Pooling Layer (2D)**

Dieser Pooling Layer behält nur die Aktivierung des aktivsten und damit wichtigsten Neurons aus einem von mehreren festgelegten quadratischen Bereichen eines jeden zweidimensionalen Kanals einer Eingabe und gibt diese Werte anschließend an die nächste Schicht weiter. Wie auch schon bei Convolutional Layers kann die Größe und die Schrittweite der Bereiche bei der Entwicklung des CNNs festgelegt werden. Anders als bei 2D-Convolutional Layers, wird dabei jeder einzelne Kanal der Eingabe bearbeitet. Die Dimensionen der Eingabe (z.B. 3 Dimensionen) bleiben somit erhalten. Die genannten Bereiche haben bei 2D-Kanälen meistens eine räumliche Ausbreitung von 2x2 und eine Schrittweite von 2, was in diesem Beispiel dazu führt, dass jeder Kanal der Eingabe in beiden Dimensionen halbiert wird. In Abbildung 6 wird dieser Vorgang noch einmal visualisiert. (vgl. [15])

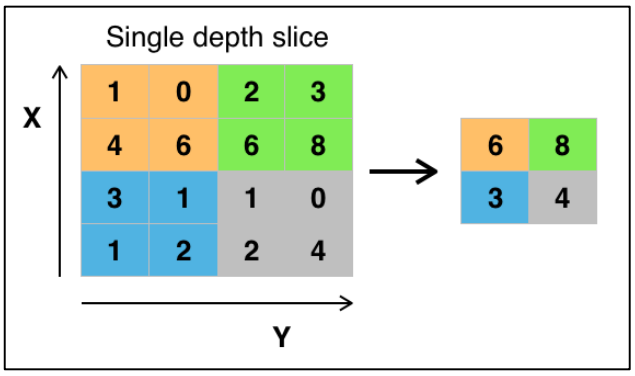



**Abbildung 6: Max Pooling mit einer 2x2 Matrix und einer Schrittweite von zwei [15]**

**Global Average Pooling Layer (2D)**

Dieser Pooling Layer berechnet den Durchschnitt (engl.: average) aus allen Eingaben eines jeden zweidimensionalen Kanals aus der Eingabe (z.B. RGB-Bild mit 3 Farbkanälen) und gibt diese Werte als eindimensionalen Vektor, mit einer Länge, die der Anzahl der Kanäle der Eingabe entspricht (z.B. 3 Kanäle ≙ eindimensionaler Vektor der Länge 3), an die nachfolgende Schicht weiter. Dieses Pooling-Verfahren soll durch Reduzierung trainierbarer Parameter der Überanpassung[9] (engl.: overfitting) von Netzen, insbesondere CNNs, an Trainingsdaten entgegenwirken und Fully-Connected Layer in zukünftigen Netzarchitekturen von Softwareentwicklern und Data Scientists ablösen. (vgl. [22])

## 5.3 Flatten Layer

Dieser Layer überführt zwei- oder mehrdimensionale Eingaben in eindimensionale Vektoren. Die Länge dieser Vektoren ergibt sich aus der Multiplikation der räumlichen Ausbreitungen der Eingabe. Eine Eingabe der Form 32x32x32 würde somit einen Vektor der Länge 32.768 ergeben, der im Anschluss zum Beispiel mit einem Fully Connected Layer verknüpft werden kann. (vgl. [23])

## 5.4 Fully Connected Layer

Fully Connected Layers stellen den Abschluss eine CNNs dar und werden hauptsächlich zur Klassifizierung von Objekten, wie etwa Bildern, angewendet. Die letzte Schicht in einem CNN besteht für gewöhnlich aus künstlichen Neuronen, deren Anzahl sich aus den vom CNN zu unterscheidenden Klassen ergibt. Anders als bei den bereits vorgestellten Schichten, werden alle künstlichen Neuronen des Fully Connected Layers mit allen seinen Eingaben verbunden, die dafür in eindimensionaler Form vorliegen müssen. Dies lässt sich zum Beispiel durch den bereits beschriebenen Flatten Layer erreichen. Die Aktivierungen der letzten Schicht werden meistens über die Softmax-Funktion berechnet und stellen die prozentuale Zugehörigkeit eines Objekts zu einer Klasse dar. (vgl. [15, 24])

## 5.5 (Spatial) Dropout Layer (2D)

Dieser Layer wird ausschließlich während des Trainings aktiviert und soll einen regularisierenden Effekt auf neuronale Netzwerke haben, indem pro Trainingsschritt zufällig ausgewählte Neuronen in diesem ausgeschalten werden. Durch Dropout Layer lässt sich nachweislich die Fähigkeit von künstlichen neuronalen Netzen zur Generalisierung verbessern. Auch Überanpassung lässt sich durch den Einsatz dieser Layer verhindern. (vgl. [25])

Da diese Effekte bei CNNs nicht im gleichen Ausmaß wie bei KNNs beobachtet werden konnten, wird die Verwendung von sog. „Spatial Dropout Layers“ empfohlen. Diese Layers schalten nicht mehr einzelne Neuronen aus, sondern ganze Feature Maps. Durch dieses Vorgehen lassen sich zum Teil deutlich bessere Ergebnisse während des Trainings von CNNs erzielen. (vgl. [26])

[9] Weiterführende Informationen: http://de.wikipedia.org/wiki/%C3%9Cberanpassung

### 5.6 Batch Normalization Layer (2D)

Ähnlich wie einige der bereits besprochenen Schichten, dient auch der „Batch Normalization Layer“ zur Regularisierung von KNNs. Hierbei werden alle Kanäle der Eingabe des Netzwerks in einem Verfahren, das aufgrund seiner Komplexität nicht näher erläutert wird, normalisiert und an die nachfolgende Schicht weitergeleitet. Durch diese Normalisierung konnten anders als bei den genannten Verfahren nicht nur regularisierende Effekte, sondern auch höhere Lernraten und bessere Ergebnisse erzielt werden. (vgl. [27])

## 6 Stanford Dogs Dataset

Der Stanford Dogs Dataset (dt.: Stanford Hunde Datensatz), der in dieser Arbeit zum Trainieren von mehreren verschiedenen CNNs verwendet wird, ist ein frei zugänglicher Datensatz der Universität Stanford (Stanford, Kalifornien) und enthält aktuell 20.580 Bilder von 120 verschiedenen Hunderassen.(vgl. [28])

Im Zuge dieser Studienarbeit werden allerdings nur Bilder von zehn verschiedenen Hunderassen verwendet, sodass das Training der entworfenen CNNs weniger Zeit in Anspruch nimmt und eine schnellere Auswertung der Ergebnisse von etwaigen Änderungen an der Netztopologie stattfinden kann.

Weiterhin muss in Hinblick auf die später stattfindende Datenvorverarbeitung darauf hingewiesen werden, dass sich sowohl die Auflösungen als auch die Seitenverhältnisse aller Bilder z.T. sehr stark voneinander unterscheiden.

Die Bilder der ausgewählten Hunderassen liegen der Arbeit bei und können beliebig erweitert werden. Alle Bilder von neuen und unbekannten Hunderassen sollten, wie auch alle anderen Bilder, in einem Ordner mit dem Namen der jeweilig repräsentierten Hunderasse zusammengefasst und unter „images“ abgespeichert werden.

## 7 Entwurf

Um auf Basis der in dieser Arbeit vermittelten Grundlagen eigene, leistungsfähige Convolutional Neuronal Networks zur Lösung von Problemstellungen entwickeln zu können, bedarf es zunächst einer auf die Art der Problemstellung angepassten Netztopologie.

Da CNNs, wie bereits in Kapitel 5 erwähnt, hauptsächlich zur Klassifizierung von Mustern in Bild- und Tonmaterial verwendet werden, richten sich auch die folgenden Entwürfe von mehreren verschiedenen CNNs an diese Art von Daten. Damit auch etwaige Vor- und Nachteile von verschiedenen Schichttypen herausgestellt oder verifiziert werden können, werden in diesem Kapitel 5 CNNs zur Klassifizierung von Bildern entworfen. Das zum Training und zur Validierung der CNNs verwendete Bildmaterial stammt dabei, wie bereits im Kapitel 6 erwähnt, aus dem Stanford Dogs Dataset, der für jede einzelne Hunderasse durchschnittlich über 150 Bilder bereitstellt. Die Benennungen der entworfenen CNNs sind hierbei analog zu den Listeneinträgen bzw. Auswahlmöglichkeiten des in Abbildung 7 dargestellten Benutzersteuerelement des beigelegten *Jupyter Notebooks* gewählt. Das Jupyter Notebook hat den Namen „Klassifikation_von_Hunderassen.ipynb“ und lässt sich einfach in Jupyter Notebook-Umgebungen, wie *Google Colab*, einbinden.

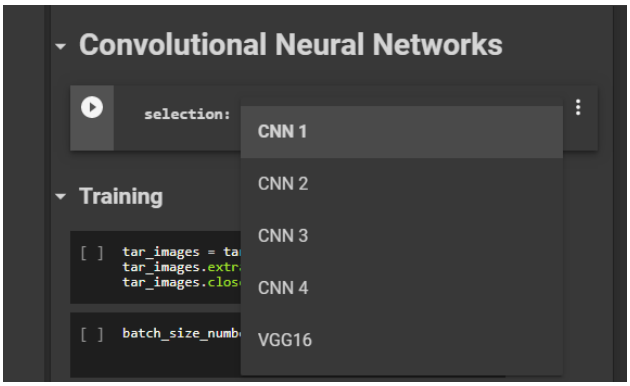



**Abbildung 7: Steuerelement mit Auswahlmöglichkeiten**

Nachdem das Jupyter Notebook in eine entsprechende Umgebung importiert wurde, kann der Benutzer den im Notebook enthaltenen Quellcode direkt ausführen und dessen Ergebnisse betrachten. Das in dieser Arbeit verwendete Notebook wurde unter Zuhilfenahme der Programmiersprache Python entwickelt und mit Textpassagen zu den einzelnen Bestandteilen versehen.

### 7.1 CNN 1

Dieser erste Entwurf stellt die Ausgangsbasis für alle weiteren im Verlauf des Projekts erstellten Netztopologien dar. Das CNN 1 ist nach dem Vorbild des sog. „VGG16“ entworfen worden, welches ebenfalls ein CNN ist und bereits einer der wenigen Gewinner des namhaften Software-Wettbewerbs ILSVRC im Jahre 2014 wurde. Dieser Wettbewerb wird seit 2010 jährlich vom ImageNet-Projekt veranstaltet und soll Aufschluss über die erzielbaren Resultate von Algorithmen zur Objekterkennung und Bildklassifizierung geben. (vgl. [29, 30])

Weiterhin lässt sich die entworfene Netztopologie im Anhang in Abbildung 19 finden. Da die eingesetzte Software-Bibliothek allerdings andere Namen für die Schichten von neuronalen Netzen verwendet, werden die bereits beschriebenen Schichten mit denen der Software-Bibliothek in Tabelle 1 in Bezug zueinander gesetzt.

| Schicht | Bezeichnung Software-Bibliothek |
|---|---|
| Convolutional (2D) | Conv2D |
| Max Pooling (2D) | MaxPooling2D |
| Global Average Pooling (2D) | GlobalAveragePooling2D |
| Flatten | Flatten |
| Fully Connected | Dense |
| Spatial Dropout (2D) | SpatialDropout2D |
| Batch Normalization (2D) | BatchNormalization2D |

Tabelle 1: Bezeichnung von Schichten in der Deep-Learning-Bibliothek Keras

Wie aus Tabelle 1 außerdem hervorgeht, werden in dieser Arbeit ausschließlich Layer verwendet, die 2D-Kanäle erwarten oder alle Eingaben wie eben solche behandeln.

### 7.2 CNN 2

Diese Netztopologie dient zur Überprüfung der unter Kapitel 5.2 genannten Vorteile von Global Average Pooling Layern. Während im ersten Entwurf noch Fully Connected Layer bzw. Dense Layer zum Einsatz kamen, werden diese nun durch eine Global Average Pooling Schicht ersetzt. Ausschließlich der letzte Fully Connected Layer bleibt bestehen, sodass sich über dessen Softmax-Funktion eine kategorische Wahrscheinlichkeitsverteilung berechnen lässt. Der konkrete Entwurf lässt sich in Abbildung 20 im Anhang der Arbeit finden.

### 7.3 CNN 3

Dieser Entwurf zielt, ähnlich wie der Entwurf von CNN 2, auf die Überprüfung der Vorteile von den in Kapitel 5.6 erläuterten Batch Normalization Layern ab. Diese Layers werden, anders als in der Publikation von Sergey Ioffe und Christian Szegedy beschrieben, nach und nicht vor der Rectifier-Funktion eines Layers verwendet. Der konkrete Entwurf lässt sich in Abbildung 21 im Anhang der Arbeit finden.

### 7.4 CNN 4

Das letzte vom Autor dieser Studienarbeit entworfene CNN dient ebenfalls zur Verifizierung von beobachteten, positiven Effekten, welche durch die Verwendung bestimmter Layer, im vorliegenden Fall Spatial Dropout Layer, eintreten sollen. Der konkrete Entwurf lässt sich in Abbildung 22 im Anhang der Arbeit finden.

### 7.5 VGG16

Dieses vortrainierte CNN wird an die Problemstellung der Arbeit, also das Klassifizieren von Hunderassen in Bildern, angepasst und mit den anderen, durch den Autor selbst entworfenen, Netzwerken verglichen. Der konkrete Entwurf lässt sich in Abbildung 23 im Anhang der Arbeit finden.

## 8 Implementierung

Die softwareseitige Implementierung der im vorherigen Kapitel 7 entworfenen Convolutional Neural Networks findet mit Hilfe der Programmiersprache Python, der Deep-Learning-Bibliothek Keras und der Jupyter Notebook-Umgebung Google Colab statt.

Der in diesem Kapitel besprochene Quellcode liegt, wie bereits erwähnt, der Studienarbeit in Form eines Jupyter Notebooks bei. Aufgrund unvorhersehbaren Fehlverhaltens des Quellcodes sowie der Jupyter Notebook-Umgebung, wird dem Leser empfohlen, das Jupyter Notebook „Klassifikation_von_Hunderassen.ipynb“ in die ursprünglich vom Autor verwendete Entwicklungsumgebung, hier Google Colab, zu importieren.

Nach dem erfolgreichen Import ist es dem Leser möglich, alle nachfolgend besprochenen Teile des Quellcodes eigenständig und ohne Konfiguration einer Systemumgebung nachzuvollziehen und ausgiebig zu testen. Da der Code nur im Zusammenspiel mit dem Cloud-Speicher „Google Drive“ korrekt funktioniert, sollte außerdem der beigelegte Ordner „source“ in diesen importiert werden.

Es wird an dieser Stelle empfohlen, den Ordner „source“ in einem neuen Verzeichnis abzulegen und den Pfad zu diesem im Textfeld des dritten Codeblocks des Jupyter Notebooks anzugeben. Ferner sollte beachtet werden, dass die Pfadangabe mit einem Backslash enden muss. Sollte dies nicht der Fall sein, so wird der Pfad nicht als Verzeichnis- sondern als Dateipfad aufgefasst, was zu Fehlern bei der Ausführung des Codes führen kann. In Abbildung 8 wird beispielhaft ein korrekter Verzeichnispfad abgebildet.

```
working_directory: " Angewandtes Maschinelles Lernen/source/ "
```

**Abbildung 8: Korrekter Verzeichnispfad**

Wenn das Verzeichnis mit dem Ordner „source“ erfolgreich in das genannte Textfeld eingegeben wurde, so kann mit der Ausführung des Quellcodes begonnen werden. Hierzu genügt das Betätigen der Schaltfläche am oberen, linken Rand eines jeden Codeblocks. Weiterhin sollten alle Codeblöcke in chronologischer Reihenfolge also nacheinander ausgeführt werden. Andernfalls kann es wieder zu Fehlern bei der Ausführung des Codes kommen. Die einzigen Ausnahmen sind hierbei die beiden Sektionen „Auswertung“ und „Training“ im Abschnitt „Convolutional Neural Networks“. Jeder Bereich kann hierbei unabhängig vom jeweils anderen ausgeführt werden. Jedoch sollten auch hier in der ausgewählten Sektion alle Codeblöcke hintereinander ausgeführt werden.

**Import**

In diesem Codeblock (siehe Abbildung 9) werden alle benötigten Bibliotheken importiert. Die meisten Importe sind dabei auf Keras zurückzuführen. Außerdem werden Bibliotheken oder deren Teile zum Umgang mit Dateien oder Verzeichnissen (google.colab), zur Darstellung von Graphen (matplotlib), zum Entpacken von TAR-Archiven (tarfile), zum Arbeiten mit JSON-Dateien (json) und für mathematische Berechnungen (numpy) in die Umgebung geladen.

```
[ ]  from keras.preprocessing.image import ImageDataGenerator
     from keras.models import Sequential
     from keras.models import load_model
     from keras.layers import Conv2D
     from keras.layers import MaxPooling2D
     from keras.layers import Flatten
     from keras.layers import Dense
     from keras.layers import Dropout
     from keras.layers import GlobalAveragePooling2D
     from keras.layers import BatchNormalization
     from keras.layers import SpatialDropout2D
     from keras.activations import relu
     from keras.activations import softmax
     from keras.optimizers import Adam
     from keras.applications.vgg16 import VGG16
     from keras.models import Model
     from keras.callbacks import ModelCheckpoint
     from keras.preprocessing import image
     from google.colab import drive
     from google.colab import files
     from matplotlib import pyplot
     import tarfile
     import json
     import numpy
```

**Abbildung 9: Importierte Software-Bibliotheken**

**Arbeitsverzeichnis setzen**

Nachdem alle Bibliotheken bzw. deren Teile geladen worden sind, kann das Arbeitsverzeichnis gesetzt werden. Dafür genügt es, den zweiten, dritten und vierten Codeblock im Notebook auszuführen. Da im dritten Codeblock bereits ein Verzeichnispfad stehen sollte, müssen nur noch die Anweisungen des Ausgabefeldes des zweiten Codeblocks befolgt werden. Dieser Block stellt eine Verbindung zum Cloud-Speicher „Google Drive“ her.

**CNN selektieren**

Um eines der entworfenen CNNs zu trainieren oder auszuwerten muss im ersten Codeblock des Abschnitts „Convolutional Neural Networks“ zunächst eine Auswahl durch das Ausführen des Codeblocks getroffen werden. (siehe Abbildung 7)

**Bildmaterial entpacken**

Damit die beigelegten Bilder von verschiedenen Hunderassen aus dem TAR-Archiv „images.tar“ verwendet werden können, müssen diese zunächst extrahiert und in die Anwendung geladen werden. Dies geschieht mit dem in Abbildung 10 dargestellten Codeblock. Hierbei wird zunächst das TAR-Archiv geöffnet und anschließend entpackt. Abschließend wird das TAR-Archiv wieder geschlossen. Der abgebildete Codeblock (siehe Abbildung 10) befindet sich an erster Stelle der Sektion „Training“ im Abschnitt „Convolutional Neural Networks“.

```
[ ]  # Öffnet das TAR-Archiv "images.tar", welches im
     # festgelegten Arbeitsverzeichnis gesucht wird.
     tar_images = tarfile.open(name=working_directory + "images.tar")

     # Extrahiert alle Daten aus dem TAR-Archiv und speichert
     # diese in der Jupyter Notebook-Umgebung.
     tar_images.extractall()

     # Schließt das TAR-Archiv
     tar_images.close()
```

**Abbildung 10: Entpacken des TAR-Archivs „images.tar“**

**Batch Size setzen**

Die Batch Size, auch Stapelgröße genannt, gibt an wie viele Daten bzw. Bilder beim Training des Netzes gleichzeitig in den Arbeitsspeicher geladen werden können. Da eine zu große Stapelgröße zu einem Speicherüberlauf führen kann, sollte man sich zunächst für einen kleineren Wert entscheiden und diesen je nach verfügbaren Ressourcen langsam vergrößern. Einen Anhaltspunkt stellt hierbei der Wert „64“ dar (siehe Abbildung 11). Dieser hat während des Trainings in „Google Colab“ keine Speicherüberläufe oder andere Probleme nach sich gezogen und kann deshalb zum jetzigen Zeitpunkt (Stand: 05 Juli 2019) uneingeschränkt empfohlen werden. Der entsprechende Codeblock befindet sich an zweiter Stelle des Abschnitts „Training“.

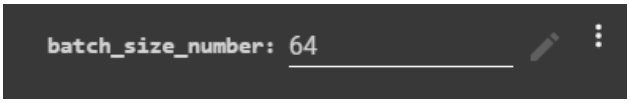



**Abbildung 11: Setzen der Batch Size**

**Laden und Vorverarbeiten der Bilder**

Der dritte Codeblock im Abschnitt „Training“ befasst sich mit der Vorverarbeitung und dem Laden der extrahierten Bilder. Zunächst wird hierbei die Form der Eingabedaten festgelegt. Da es sich bei den verwendeten Bildern um RGB-Farbbilder handelt wird deren Form, neben der Auflösung, um eine weitere, dritte Dimension, nämlich die Anzahl der Farbkanäle, bei RGB-Bildern 3, erweitert. Die Auflösung der Bilder wurde in diesem Projekt auf 256*256 Pixel festgelegt und kann prinzipiell beliebig verändert werden.

Im Anschluss an die Festlegung der Form wird ein Objekt der Klasse „ImageDataGenerator“ erzeugt, welches dazu genutzt wird die Bilder während des Ladevorgangs zu scheren, zu vergrößern und horizontal zu spiegeln. Diese Vorgehensweise wird auch Data oder Image Augmentation genannt. Sie soll zur Vergrößerung des vorliegenden Datensatzes dienen und damit dem Netz zu besseren Ergebnissen bei der Klassifikation verhelfen. Weiterhin wurde ein sog. „validation split“ von 0,2 eingestellt. Dieser Parameter sorgt dafür, dass 20% aller geladenen Daten zur Validierung des Netzes verwendet werden. Bei diesem Verfahren wird überprüft, wie ein KNN auf neue, noch unbekannte Daten reagiert und inwiefern die Ergebnisse von denen des Trainingsprozesses abweichen.

Zusammen mit der gewünschten Form der Eingabedaten, hier: Bilder, lassen sich nun durch die Methode „flow_from_directory“ alle Bilder im angegebenen Verzeichnis, hier „images“, laden und vorverarbeiten. Die Auflösungen werden dabei durch die bereits angegebene Form, hier: 256*256, festgelegt. Überdies werden die Bilder beim Ladevorgang auch noch automatisch mit einem Label versehen. Diese Label tragen dann die Beschriftung des Ordners, in welchem das jeweilige Bild gefunden wurde. Jedes Verzeichnis stellt somit eine Kategorie bzw. Klasse für die Klassifikation dar. Als Rückgabeparameter der Methode erhält man einen Generator, der zur Ausgabe von Schlüsselwertepaaren (Bilder mit zugeörigen Label) dient. Je nachdem welches „subset“ (dt.: Teilmenge) man im Code angegeben hat, erhält man entweder einen Generator mit Trainingsdaten, hier: 80% aller Bilder, oder einen Generator mit Validierungsdaten, hier: 20% aller Bilder. Da beide Generatoren für das Training samt Überprüfung benötigt werden, muss auch die Methode zwei Mal aufgerufen, sowie die Rückgabe zwei Mal gespeichert werden. Außerdem wird die Anzahl der gefundenen Labels für den Entwurf der neuronalen Netze gespeichert. Weitere Details können dem Quellcode entnommen werden.

**Enwurf des selektierten CNNs**

Im vierten Codeblock des Abschnitts „Training“ werden alle unter Kapitel 7 besprochenen Entwürfe implementiert. Je nach Auswahl unterscheiden sich die Implementierungen, wie bereits angemerkt, voneinander. Da detaillierte Erklärungen zu den Parametern den Umfang dieser Arbeit sprengen würden, wird an dieser Stelle auf die Dokumentation von Keras[10] verwiesen. Mit den bereits in der Arbeit getätigten Aussagen zu Faltungsmatrizen (Filtern), Padding und Aktivierungsfunktionen lässt sich die Implementierung auch anhand der Ausgaben im Jupyter Notebook leicht nachvollziehen.

[10] Weiterführende Informationen: https://keras.io/

**Kompilieren des selektieren CNNs**

Nachdem die selektierten CNNs softwareseitig entworfen wurden, können sie im fünften Codeblock kompiliert werden. Hierbei gibt man eine Verlustfunktion (hier: „categorical_crossentropy“), eine Optimierung (hier: Adam) und eine oder mehrere Messgrößen, die vom Netz während des Trainings evaluiert werden sollen, an. Wie auch schon im vorherigen Absatz werden die gesetzten Parameter, sowie deren theoretische Grundlagen in dieser Studienarbeit nicht weiter behandelt. Sie können in Fachliteratur gefunden werden.

Der einzige Parameter, der in diesem Quellcodeblock trotzdem nicht unerwähnt bleiben sollte, ist der Parameter „lr“, welcher die Lernrate des Optimierungsalgorithmus Adam vorgibt. Dieser wird nämlich nur bei der Auswahl „VGG16“ vom Standardwert „1e-3“ auf den Wert „2e-5“ gesetzt. Grund dafür sind die Gewichtungen des CNNs, welche bereits trainiert wurden. Wenn die Lernrate zu hoch eingestellt wird, so werden diese Gewichtungen zu schnell angepasst und die Ergebnisse des CNNs schlechter. Abbildung 12 stellt den besprochenen Codeblock noch einmal dar.

```
[ ]  if selection == "VGG16":
       # Verringerung der Lernrate, da sonst bestehende
       # Gewichtungen zu schnell geändert werden.
       model.compile(
           loss='categorical_crossentropy',
           optimizer=Adam(lr=2e-5),
           metrics=['accuracy']
       )
     else:
       model.compile(
           loss='categorical_crossentropy',
           optimizer=Adam(),
           metrics=['accuracy']
       )

     # Ausgabe der jeweiligen Netztopologie
     print(model.summary())
```

**Abbildung 12: Kompilierung des selektierten CNNs**

**Trainieren des selektierten CNNs**

Nachdem das gewählte CNN sowohl softwareseitig entworfen als auch kompiliert wurde, kann das Training gestartet werden. Dabei wurden zunächst die beiden, bereits erwähnten, Generatoren, ein Callback der nur die beste Trainingsepoche speichert und die Zahl „250“ als Anzahl der zu trainierenden Epochen an die Methode „fit_generator“ des kompilierten CNNs, hier: „model“ übergeben. Nach Beendigung des Trainings lässt sich das CNN als .h5-Datei im Arbeitsverzeichnis finden. Weiterhin werden in diesem letzten Codeblock des Abschnitts „Training“ auch die Trainingshistorien, sowie die verwendeten Labels als JSON-Dateien abgespeichert.

**Visualisieren der Trainingshistorie des selektierten CNNs**

Der erste Codeblock im Abschnitt „Auswertung“ befasst sich mit der Visualisierung der Trainingshistorie des ausgewählten CNNs. Da diese bereits nach dem erfolgreich abgeschlossenen Training im Arbeitsverzeichnis gespeichert wurde, genügt es die Auswahl im ersten Block des Abschnitts „Convolutional Neural Networks“ an das zu untersuchende CNN anzupassen und den ersten Block der Sektion „Auswertung“ auszuführen.

**Klassifizierung mit Hilfe des selektierten CNNs**

Damit man eines der trainierten CNNs zur Lösung der Problemstellung (Klassifikation von Hunderassen) verwenden kann, muss zunächst ein CNN ausgewählt (siehe Abbildung 7) und aus dem Arbeitsverzeichnis geladen werden. Dies geschieht im zweiten Codeblock des Abschnitts „Auswertung“. Neben dem neuronalen Netz werden dabei auch die entsprechenden Labels, die während des Trainings verwendet wurden, geladen. (siehe Abbildung 13)

```
[ ]  # Lädt das selektierte CNN aus dem festgelegten Arbeitsverzeichnis
     model = load_model(working_directory + "{0}.h5".format(selection))

     # Gibt die Struktur des CNNs aus
     print(model.summary())

     # Lädt die Labels, die beim Training des selektierten CNNs verwendet wurden.
     with open(working_directory + "{0}_labels.json".format(selection)) as labels_file:
       labels = json.load(labels_file)
```

**Abbildung 13: Laden des selektierten CNNs**

Anschließend wird bei der Ausführung des dritten Codeblocks ein Upload-Dialog dargestellt (siehe Abbildung 14), mit dessen Hilfe ein Anwender verschiedene Daten in die Laufzeitumgebung laden kann. Die hochgeladenen Daten sollten hierbei Bilder im RGB-Format sein. Andere Eingaben führen, wie auch bei der Angabe des Arbeitsverzeichnisses, zu unerwünschten Fehlverhalten in der Software und sollten daher möglichst vermieden werden.


**Abbildung 14: Upload-Dialog**

Nachdem man die Schaltfläche „Dateien auswählen“ betätigt hat, öffnet sich ein Systemdialog (siehe Abbildung 15), welcher den Anwender auffordert eine Datei zu öffnen bzw. auszuwählen. Hat sich dieser für ein Bild entschieden, so muss der Dialog nur noch durch die Betätigung des Buttons „Öffnen“ (unter Windows 10) quittiert werden. Danach wird das Bild eigenständig in die Jupyter Notebook-Umgebung geladen, auf eine Auflösung von 256*256 Pixel skaliert und klassifiziert. Das Ergebnis dieser Klassifikation lässt sich in Abbildung 16 betrachten.

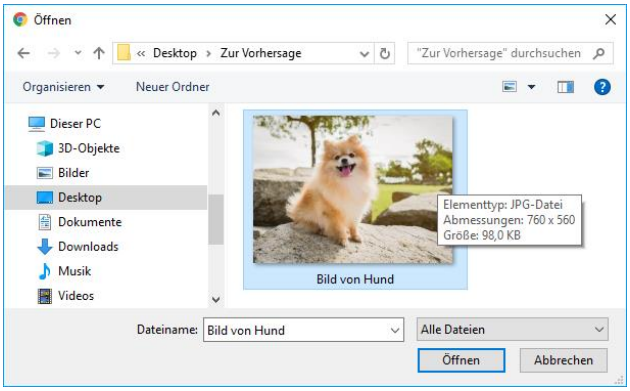



**Abbildung 15: Systemdialog [31]**

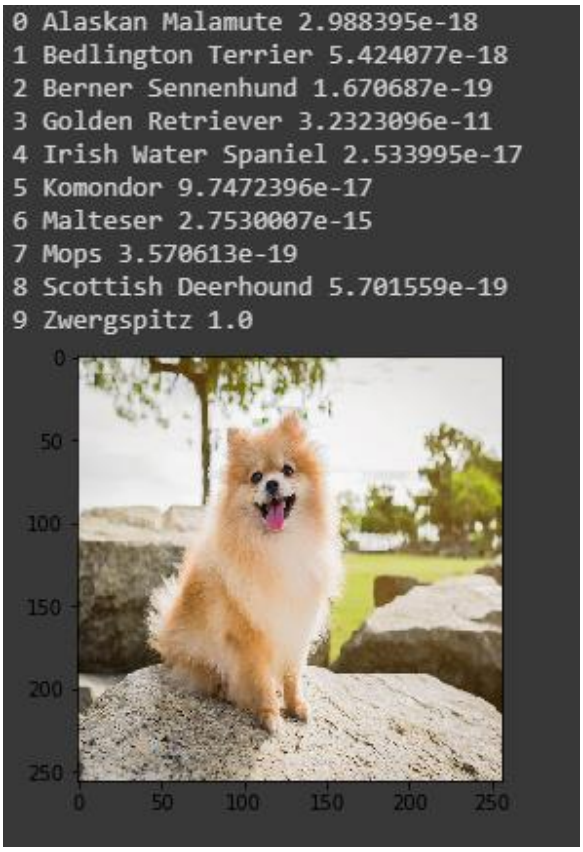



**Abbildung 16: Klassifikation eines Beispielbildes [31]**

**Visualisierung der Aktivierungen des selektierten CNNs**

Um die Funktionsweise im Inneren eines CNNs noch etwas besser verstehen zu können, ist es unter Zuhilfenahme der letzten beiden Codeblöcke im Abschnitt „Auswertung“ möglich einzelne Feature Maps bzw. Aktivierungen eines gewählten CNNs zu visualisieren. Damit dies funktioniert muss zunächst wieder ein CNN selektiert und aus „Google Drive“ geladen werden. Danach genügt es, einen Layer bzw. dessen Position im CNN und eine Feature Map bzw. einen Kanal anzugeben. (siehe Abbildung 17)


**Abbildung 17: Angabe der zu visualisierenden Aktivierung**

Weiterhin muss angemerkt werden, dass Layer und Feature Maps ab der Zahl „0“ gezählt werden. Dies hat zur Folge, dass der erste Layer im Entwurf eines CNNs immer die Position „0“ zugeordnet bekommt, während der zweite Layer immer die Position „1“ hat.

Diese Reihe lässt sich somit bis zu $Anzahl\ Layer - 1$ oder bis zu $Anzahl\ Feature\ Map - 1$ fortführen. Alle Indizes, die außerhalb dieses Bereiches sind, führen zu Fehlverhalten in der Anwendung. Das Ergebnis einer beispielhaften Visualisierung lässt sich in der nachfolgenden Abbildung 18 betrachten.

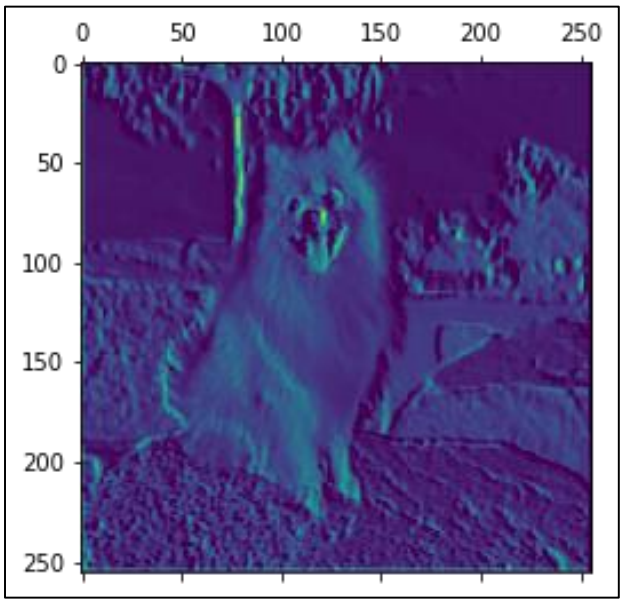



**Abbildung 18: Visualisierung einer Feature Map [31]**

## 9 Ergebnisse

### 9.1 CNN 1

Die Ergebnisse dieses CNNs sind im Vergleich zu allen anderen CNNs die schlechtesten. Wie man in Abbildung 24 und 25 bereits sehen kann, nimmt ab Epoche 25 die „Loss“ (dt.: Verlust) zu, als auch die „Accuracy“ (dt.: Genauigkeit) des Validierungsprozesses ab. Dies bedeutet, dass das neuronale Netz ab Trainingsepoche 25 kontinuierlich schlechter wird, während sich gleichzeitig ebenfalls noch Überanpassung einstellt. Dieses sog. „overfitting“ lässt sich vor allem an den zum Teil gravierenden Unterschieden zwischen den beiden Graphen „Training“ und „Test“ festmachen und sich wie folgt erklären: Lernt ein neuronales Netz die ihm zugeführten Trainingsdaten zu genau, während es bei unbekannten Testdaten deutlich schlechtere Ergebnisse liefert, so spricht man allgemein von „overfitting“. Das Netzwerk hat im Lernprozess die Fähigkeit zur Generalisierung verloren und sich nur die zugeführten Daten eingeprägt.

Da diese Erklärung ausschließlich zum besseren Verständnis des genannten Sachverhalts beitragen soll und keineswegs vollständig ist, sollte abermals auf Fachliteratur zurückgegriffen werden. Eine kleine Einführung zum Thema „overfitting“ lässt sich hier finden: https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765

### 9.2 CNN 2

Die Ergebnisse, die durch das Training des CNN 2 erzielt wurden, dienen zur Überprüfung der unter Kapitel 5.2 genannten Vorteile von Global Average Pooling Layern. Der Fokus dieser Schichten liegt dabei auf der Reduzierung trainierbarer Parameter und damit auch auf der Reduzierung von Überanpassung. Durch Abbildung 26 und 27 lässt sich diese Aussage bestätigen. Die Überanpassung wurde deutlich reduziert (kleinerer Abstand zwischen den beiden Graphen „Test“ und „Train) und auch die Fähigkeit des Netzes zu generalisieren hat sich verbessert. Sichtbar wird dies vor allem an der geringeren „Loss“ als auch an der höheren „Accuracy“.

### 9.3 CNN 3

Ähnlich wie bei den Global Average Pooling Layern, sollen auch Batch Normalization Layer einen regularisierenden Effekt auf alle KNNs haben. Weiterhin sollen mit diesen Schichten auch höhere Lernraten erzielt werden. Während letztere Behauptung durch die Abbildungen 28 und 29 bestätigt wurde, scheinen regularisierende Effekte, wie die Reduzierung von Überanpassung nur im geringen Ausmaß aufzutreten. Die „Loss“ wurde durch die Schichten etwas verringert während die „Accuracy“ etwas erhöht werden konnte.

### 9.4 CNN 4

Wie auch schon bei den beiden vorherigen Schichten soll auch der Spatial Dropout Layer für bessere Generalisierung durch das CNN und reduzierte Überanpassung sorgen. Beide Behauptungen lassen sich, obwohl nur geringe Effekte in den Abbildungen 30 und 31 beobachtet wurden, bestätigen. Neben der „Loss“, konnte auch die „Accuracy“ dahingehend verbessern werden, dass das CNN 4 das beste der selbst entworfenen, nicht vortrainierten Netzwerke ist.

### 9.5 VGG16

Das durch die Abbildungen 32 und 33 dargestellte Ergebnis dieses CNNs ist eindeutig. Das vortrainierte VGG16 schafft es innerhalb von nur wenigen Epochen überdurchschnittlich gute Resultate zu erzielen, welche mit keinem anderen der vier entworfenen CNNs in 250 Epochen erreicht werden konnten. Auch Überanpassung ist bei diesem neuronalen Netz nur im geringen Ausmaß anzutreffen. Der Grund dafür ist die überdurchschnittlich lange Trainingsdauer des Netztes, sowie der Einsatz eines umfangreichen Datensatzes. (vgl. [32, 29])

## 10 Fazit

Im Rahmen dieses Projekts konnten viele Erfahrungen im Bereich des angewandten maschinellen Lernens gesammelt werden. Diese wurden dabei vor allem auf dem Gebiet der Convolutional Neural Networks erworben. Neben der kurzen Einführung in den Bereich der Künstlichen Intelligenz und den anschließenden Erläuterungen zu den Fachbegriffen „Maschinelles Lernen“, „Muster- und Bilderkennung“ und „Künstliche Neuronale Netzwerke“, wurden auch die Konzepte und Einsatzgebiete von CNNs vorgestellt. Auf Basis dieser Grundlagen konnten somit auch Interna, wie der schichtenbasierte Aufbau und die Funktionsweise der einzelnen Schichten, dieser Sonderform eines KNNs nachvollzogen werden. Nachdem die Grundlagen für CNNs im theoretischen Teil dieser Arbeit behandelt wurden, fand der praktische Teil, also der Entwurf, die Implementierung und die Auswertung verschiedener CNNs statt. Neben Erklärungen zum Entwurf, wurden auch der Quellcode und die Ergebnisse des Trainings- und des Validierungsprozesses der CNNs besprochen. Zusammen mit dem theoretischen Hintergrund ergibt sich daraus ein solider Einstieg in die Thematik, der anhand des beiliegenden Quellcodes noch gefestigt werden kann. Alles in allem konnte festgestellt werden, dass das Modul „Angewandtes maschinelles Lernen“ fundamentale Kenntnisse in diesem Bereich vermittelt und den Leser aufgrund dieser Arbeit zur selbständigen Entwicklung von CNNs befähigt. Weiterhin sollten die Layer, die im Rahmen dieses Projekts verwendet wurden, bei der Umsetzung eines CNNs berücksichtigt werden. Die Ergebnisse aus Kapitel 9 unterstützen diese These und sollen als Ansporn zur Entwicklung eigener, leistungsfähiger CNNs dienen.

## Anhang

| Layer (type) | Output Shape |
|---|---|
| conv2d_1 (Conv2D) | (None, 256, 256, 64) |
| conv2d_2 (Conv2D) | (None, 256, 256, 64) |
| max_pooling2d_1 (MaxPooling2 | (None, 128, 128, 64) |
| conv2d_3 (Conv2D) | (None, 128, 128, 128) |
| conv2d_4 (Conv2D) | (None, 128, 128, 128) |
| max_pooling2d_2 (MaxPooling2 | (None, 64, 64, 128) |
| conv2d_5 (Conv2D) | (None, 64, 64, 256) |
| conv2d_6 (Conv2D) | (None, 64, 64, 256) |
| max_pooling2d_3 (MaxPooling2 | (None, 32, 32, 256) |
| conv2d_7 (Conv2D) | (None, 32, 32, 512) |
| conv2d_8 (Conv2D) | (None, 32, 32, 512) |
| max_pooling2d_4 (MaxPooling2 | (None, 16, 16, 512) |
| flatten_1 (Flatten) | (None, 131072) |
| dense_1 (Dense) | (None, 128) |
| dense_2 (Dense) | (None, 128) |
| dense_3 (Dense) | (None, 10) |

**Abbildung 19: Netztopologie von CNN 1**

| Layer (type) | Output Shape |
|---|---|
| conv2d_1 (Conv2D) | (None, 256, 256, 64) |
| conv2d_2 (Conv2D) | (None, 256, 256, 64) |
| max_pooling2d_1 (MaxPooling2 | (None, 128, 128, 64) |
| conv2d_3 (Conv2D) | (None, 128, 128, 128) |
| conv2d_4 (Conv2D) | (None, 128, 128, 128) |
| max_pooling2d_2 (MaxPooling2 | (None, 64, 64, 128) |
| conv2d_5 (Conv2D) | (None, 64, 64, 256) |
| conv2d_6 (Conv2D) | (None, 64, 64, 256) |
| max_pooling2d_3 (MaxPooling2 | (None, 32, 32, 256) |
| conv2d_7 (Conv2D) | (None, 32, 32, 512) |
| conv2d_8 (Conv2D) | (None, 32, 32, 512) |
| global_average_pooling2d_1 ( | (None, 512) |
| dense_1 (Dense) | (None, 10) |

**Abbildung 20: Netztopologie von CNN 2**

| Layer (type) | Output Shape |
|---|---|
| conv2d_1 (Conv2D) | (None, 256, 256, 64) |
| conv2d_2 (Conv2D) | (None, 256, 256, 64) |
| batch_normalization_1 (Batch | (None, 256, 256, 64) |
| max_pooling2d_1 (MaxPooling2 | (None, 128, 128, 64) |
| conv2d_3 (Conv2D) | (None, 128, 128, 128) |
| conv2d_4 (Conv2D) | (None, 128, 128, 128) |
| batch_normalization_2 (Batch | (None, 128, 128, 128) |
| max_pooling2d_2 (MaxPooling2 | (None, 64, 64, 128) |
| conv2d_5 (Conv2D) | (None, 64, 64, 256) |
| conv2d_6 (Conv2D) | (None, 64, 64, 256) |
| batch_normalization_3 (Batch | (None, 64, 64, 256) |
| max_pooling2d_3 (MaxPooling2 | (None, 32, 32, 256) |
| conv2d_7 (Conv2D) | (None, 32, 32, 512) |
| conv2d_8 (Conv2D) | (None, 32, 32, 512) |
| batch_normalization_4 (Batch | (None, 32, 32, 512) |
| global_average_pooling2d_1 ( | (None, 512) |
| dense_1 (Dense) | (None, 10) |

**Abbildung 21: Netztopologie von CNN 3**

| Layer (type) | Output Shape |
|---|---|
| conv2d_1 (Conv2D) | (None, 256, 256, 64) |
| conv2d_2 (Conv2D) | (None, 256, 256, 64) |
| batch_normalization_1 (Batch | (None, 256, 256, 64) |
| spatial_dropout2d_1 (Spatial | (None, 256, 256, 64) |
| max_pooling2d_1 (MaxPooling2 | (None, 128, 128, 64) |
| conv2d_3 (Conv2D) | (None, 128, 128, 128) |
| conv2d_4 (Conv2D) | (None, 128, 128, 128) |
| batch_normalization_2 (Batch | (None, 128, 128, 128) |
| spatial_dropout2d_2 (Spatial | (None, 128, 128, 128) |
| max_pooling2d_2 (MaxPooling2 | (None, 64, 64, 128) |
| conv2d_5 (Conv2D) | (None, 64, 64, 256) |
| conv2d_6 (Conv2D) | (None, 64, 64, 256) |
| batch_normalization_3 (Batch | (None, 64, 64, 256) |
| max_pooling2d_3 (MaxPooling2 | (None, 32, 32, 256) |
| conv2d_7 (Conv2D) | (None, 32, 32, 512) |
| conv2d_8 (Conv2D) | (None, 32, 32, 512) |
| batch_normalization_4 (Batch | (None, 32, 32, 512) |
| global_average_pooling2d_1 ( | (None, 512) |
| dense_1 (Dense) | (None, 10) |

**Abbildung 22: Netztopologie von CNN 4**

| Layer (type) | Output Shape |
|---|---|
| input_1 (InputLayer) | (None, 224, 224, 3) |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) |
| block3_conv3 (Conv2D) | (None, 56, 56, 256) |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) |
| block4_conv3 (Conv2D) | (None, 28, 28, 512) |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) |
| block5_conv3 (Conv2D) | (None, 14, 14, 512) |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) |
| flatten (Flatten) | (None, 25088) |
| fc1 (Dense) | (None, 4096) |
| fc2 (Dense) | (None, 4096) |
| predictions (Dense) | (None, 1000) |

**Abbildung 23: Netztopologie von VGG16**

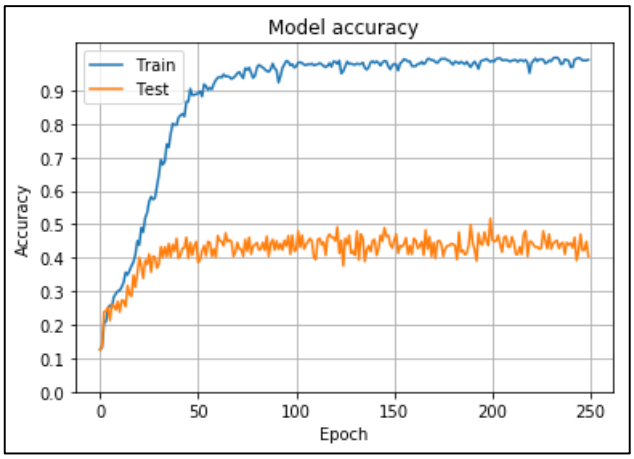



**Abbildung 24: Trainingshistorie von CNN 1 (Accuracy)**

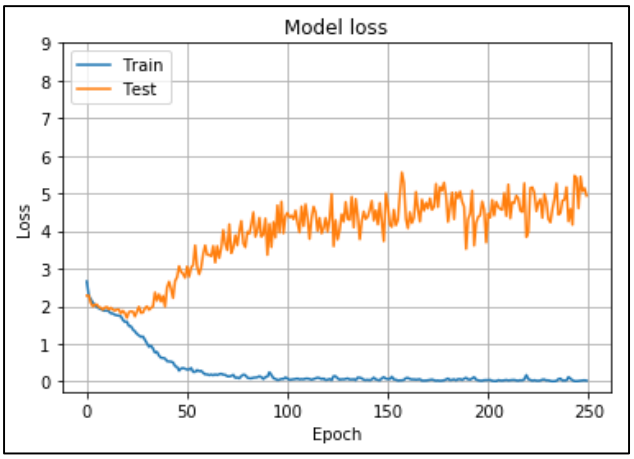



**Abbildung 25: Trainingshistorie von CNN 1 (Loss)**

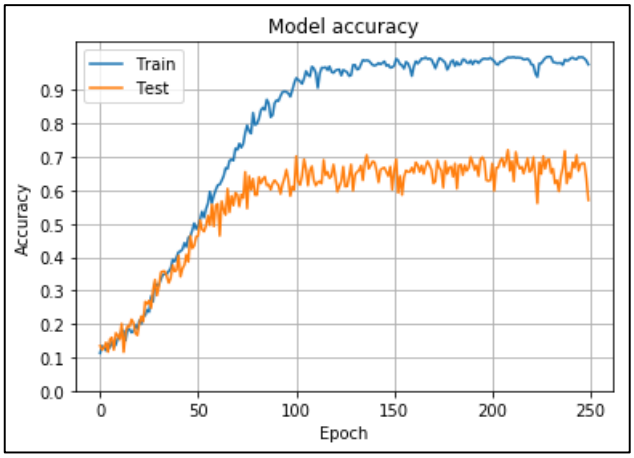



**Abbildung 26: Trainingshistorie von CNN 2 (Accuracy)**

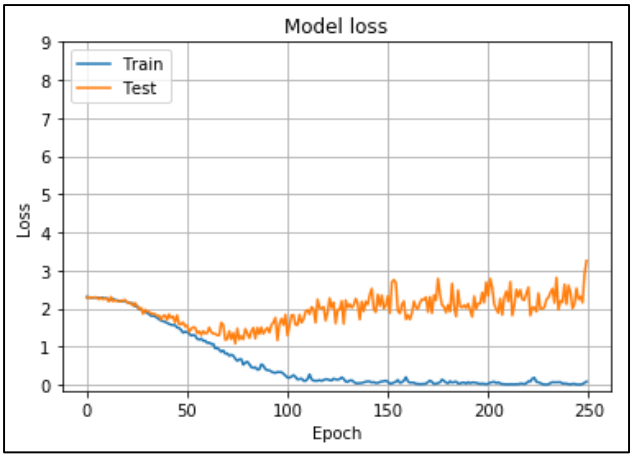



**Abbildung 27: Trainingshistorie von CNN 2 (Loss)**

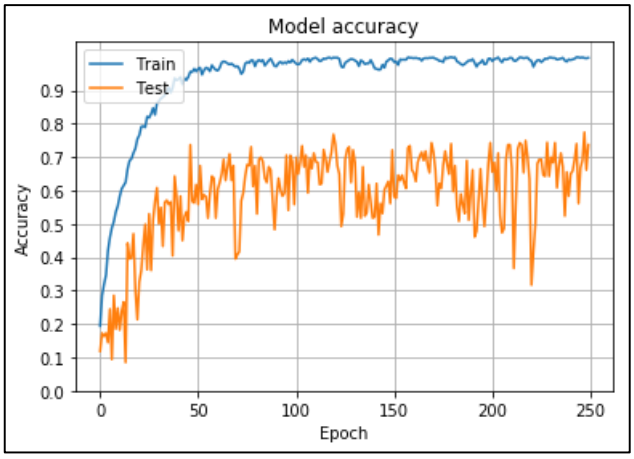



**Abbildung 28: Trainingshistorie von CNN 3 (Accuracy)**

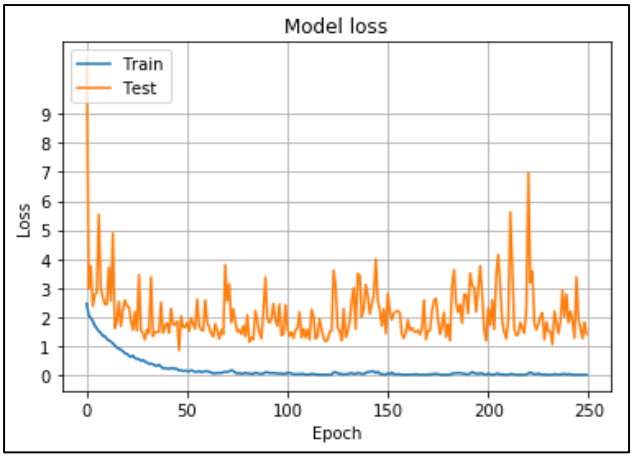



**Abbildung 29: Trainingshistorie von CNN 3 (Loss)**

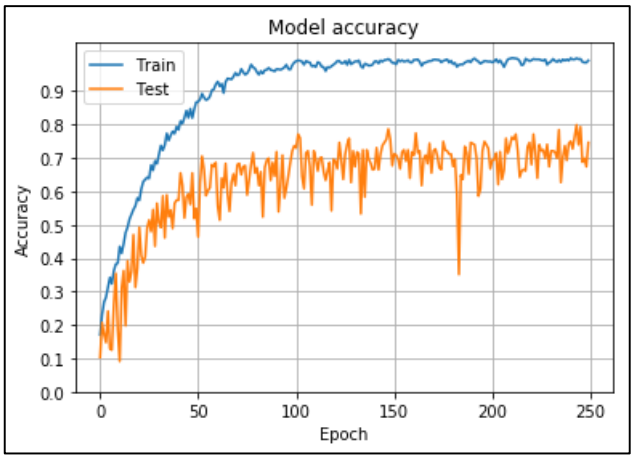



**Abbildung 30: Trainingshistorie von CNN 4 (Accuracy)**

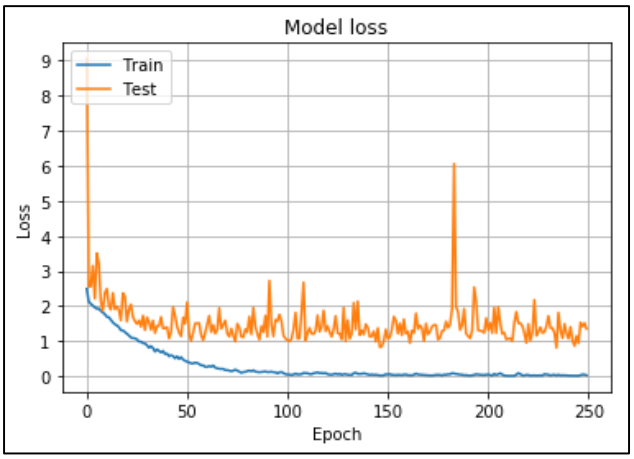



**Abbildung 31: Trainingshistorie von CNN 4 (Loss)**

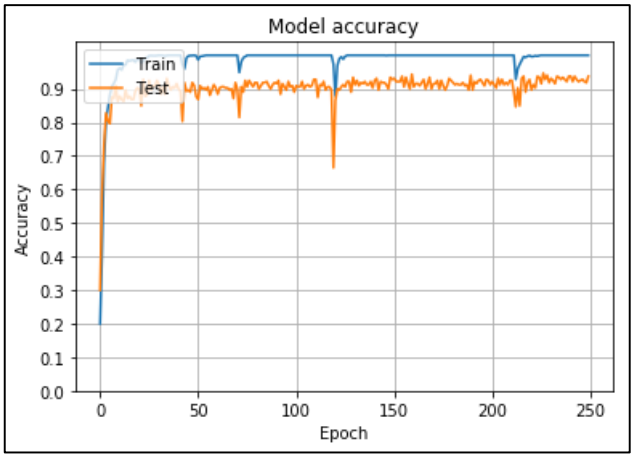



**Abbildung 32: Trainingshistorie von VGG16 (Accuracy)**

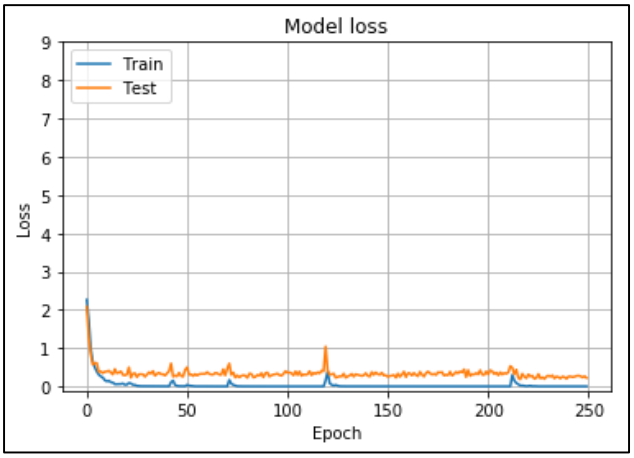



**Abbildung 33: Trainingshistorie von VGG16 (Loss)**

## LITERATURVERZEICHNIS


[1] "Geschichte der künstlichen Intelligenz – Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/Geschichte_der_k%C3%BCnstlichen_Intelligenz. [Accessed 13 06 2019].

[2] "Julien Offray de La Mettrie – Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/Julien_Offray_de_La_Mettrie. [Accessed 13 06 2019].

[3] J. M. e. al., "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence," 31 08 1955. [Online]. Available: http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html. [Accessed 19 06 2019].

[4] "Künstliche Intelligenz – Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/K%C3%BCnstliche_Intelligenz. [Accessed 13 06 2019].

[5] Hochschule Hof, "Angewandtes maschinelles Lernen," in *Modulhandbücher*, 2019.

[6] K. Manhart, "FAQ Machine Learning: Was Sie über Maschinelles Lernen wissen müssen - computerwoche.de," 19 06 2018. [Online]. Available: https://www.computerwoche.de/a/was-sie-ueber-maschinelles-lernen-wissen-muessen,3329560. [Accessed 13 06 2019].

[7] S. Luber, "Was ist Machine Learning?," 01 09 2016. [Online]. Available: https://www.bigdata-insider.de/was-ist-machine-learning-a-592092/. [Accessed 13 06 2019].

[8] C. Lemke, "Reinforcement Learning – Die Grundlagen, einfach erklärt," 11 04 2019. [Online]. Available: https://www.alexanderthamm.com/de/artikel/einfach-erklaert-so-funktioniert-reinforcement-learning/. [Accessed 15 06 2019].

[9] M. Goram, "Supervised und unsupervised Learning: Ansätze und Vorgehensweisen beim maschinellen Lernen - computerwoche.de," 28 03 2019. [Online]. Available: https://www.computerwoche.de/a/ansaetze-und-vorgehensweisen-beim-maschinellen-lernen,3331036. [Accessed 15 06 2019].

[10] "Mustererkennung – Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/Mustererkennung. [Accessed 19 06 2019].

[11] N. Dannenberger and L. Herzog, "KI-Medizin: Die zwei Gesichter der intelligenten Assistenten," 18 03 2019. [Online]. Available: https://www.faz.net/aktuell/wissen/medizin-ernaehrung/ki-medizin-die-zwei-gesichter-der-intelligenten-assistenten-16095381.html. [Accessed 19 06 2019].

[12] "Bilderkennung – Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/Bilderkennung. [Accessed 20 06 2019].

[13] "Künstliches neuronales Netz – Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/K%C3%BCnstliches_neuronales_Netz. [Accessed 20 06 2019].

[14] "Neuronale Netze - Eine Einführung - Einleitung," [Online]. Available: http://www.neuronalesnetz.de/einleitung.html. [Accessed 20 06 2019].

[15] "Convolutional Neural Network – Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/Convolutional_Neural_Network. [Accessed 20 06 2019].

[16] "Künstliches Neuron – Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/K%C3%BCnstliches_Neuron. [Accessed 21 06 2019].

[17] J. Yang, "ReLU and Softmax Activation Functions · Kulbear/deep-learning-nano-foundation Wiki · GitHub," 11 02 2017. [Online]. Available: https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions. [Accessed 21 06 2019].

[18] "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: http://cs231n.github.io/neural-networks-1/. [Accessed 21 06 2019].

[19] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," [Online]. Available: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf. [Accessed 21 06 2019].

[20] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," 11 1998. [Online]. Available: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf. [Accessed 23 06 2019].

[21] "Convolutional Layers - Keras Documentation," [Online]. Available: https://keras.io/layers/convolutional/. [Accessed 26 06 2019].

[22] M. Lin, Q. Chen and S. Yan, "Network In Network," 04 03 2014. [Online]. Available: https://arxiv.org/pdf/1312.4400.pdf. [Accessed 26 06 2019].

[23] "Core Layers - Keras Documentation," [Online]. Available: https://keras.io/layers/core/. [Accessed 26 06 2019].

[24] R. B. Zadeh and B. Ramsundar, "TensorFlow for Deep Learning," 03 2018. [Online]. Available: https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html. [Accessed 26 06 2019].

[25] "Improving neural networks by preventing co-adaptation of feature detectors," 03 07 2012. [Online]. Available:

https://arxiv.org/pdf/1207.0580.pdf. [Accessed 26 06 2019].

[26] J. Tompson, R. Goroshin, A. Jain, Y. LeCun and C. Bregler, "Efficient Object Localization Using Convolutional Networks," 09 06 2015. [Online]. Available: https://arxiv.org/pdf/1411.4280.pdf. [Accessed 26 06 2019].

[27] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 02 03 2015. [Online]. Available: https://arxiv.org/pdf/1502.03167.pdf. [Accessed 26 06 2019].

[28] A. Khosla, N. Jayadevaprakash, B. Yao and L. Fei-Fei, "Stanford Dogs dataset for Fine-Grained Visual Categorization," [Online]. Available: http://vision.stanford.edu/aditya86/ImageNetDogs/. [Accessed 27 06 2019].

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," 30 01 2015. [Online]. Available: https://arxiv.org/pdf/1409.0575.pdf. [Accessed 27 06 2019].

[30] "ImageNet Large Scale Visual Recognition Competition 2014 (ILSVRC2014)," [Online]. Available: http://www.image-net.org/challenges/LSVRC/2014/results. [Accessed 27 06 2019].

[31] "kleinspitz-hunderassen-760x560.jpg (760×560)," [Online]. Available: http://www.zooroyal.de/magazin/wp-content/uploads/2017/07/kleinspitz-hunderassen-760x560.jpg. [Accessed 10 07 2019].

[32] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 10 04 2015. [Online]. Available: https://arxiv.org/pdf/1409.1556.pdf. [Accessed 07 05 2019].

[33] "Software-Agent – Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/Software-Agent. [Accessed 19 06 2019].

# Maschinelles Lernen mit DeepLab

Semantische Segmentation

Huget Patrick David
Fakultät Informatik
Hochschule Hof
Hof Bayern Deutschland
patrick.huget@hof-university.de

## Motivation

Im Rahmen dieses Projektes im Fach Angewandtes maschinelles Lernen sollen Studenten sich mit fachspezifischen Themen beschäftigen und eigene Erfahrungen mit Anwendungen, die maschinell lernen, machen. Das Projekt soll dazu dienen, Schülern/innen bei Veranstaltungen durch die Hochschule einen ersten Einblick in diese komplexe Thematik bieten zu können.

In dieser Arbeit wurde eine Anwendung betrachtet, die sich mit der Erkennung und Klassifikation von Objekten auf Bildern beschäftigt. Im übertragenen Sinn bedeutet das, Gegenstände bzw. Personen und Tiere auf Bildern zu lokalisieren und im Anschluss auch noch in einer für die Art des Objektes vordefinierten Farbe einzufärben. In Abbildung 1 können sie das Ausgangsbild und das Endresultat sehen. Dazu wird mit der Software DeepLab ein neuronales Netz trainiert. DeepLab wurde von Mitarbeitern/innen des Google Research Teams entwickelt und weiterentwickelt. Es befindet sich momentan in der Version 3+ [1]. DeepLab setzt einiges anders um, als es gewöhnliche „deep convolutional neural network“ tun, was jedoch in späteren Abschnitten ausführlicher beschrieben wird.

**Abbildung 1: Beispiel für die Erkennung und Klassifikation von Objekten auf Bildern (semantische Segmentation) [2]**

Innerhalb des Projektes wurde ein derartiges Netz mit dem umfangreichen PASCAL VOC 2012 Datensatz trainiert. Außerdem erfolgte mit Hilfe von Quelle [3] die Entwicklung eines Shellskripts, welches einige Fehler behebt, die aufgetreten sind, und allgemein den Selbstversuch am PASCAL VOC 2012 Datensatz durch DeepLab vereinfacht. Weiterhin wurde eine bereits vorhandene Demonstrationssoftware so modifiziert, dass trainierte Netze aus dem Google Drive Ordner des Nutzers verwendet werden können.

Mit dem Nutzen verschiedener Datensätze beim Trainieren der Netze können unterschiedliche Objekte erkannt werden. Im Zusammenhang mit DeepLab werden insbesondere die folgenden beschriebenen Datensätze genutzt. Zum Einen der Datensatz „PASCAL VOC 2012“, mit welchem beispielsweise Fahrzeuge, verschiedene Tiere, Personen und unterschiedliche Einrichtungsgegenstände erkannt und segmentiert werden können, und zum Anderen der Datensatz „Cityscapes“, mit welchem Elemente in Straßennähe, wie unter anderem Straßenschilder, Ampeln, verschiedene Fahrzeuge, Straßen, Gehwege, Passanten sowie auch der Himmel, Gebäude und Vegetation auseinander gehalten werden können. Dieser Datensatz kann aber von der Zielgruppe dieses Projektes leider nicht genutzt werden, da er von Seiten des Lizenzgebers nur zur Verfügung gestellt wird, wenn eine Hochschul- oder Arbeitsemailadresse vorliegt. Der letzte Datensatz, der angesprochen werden soll, trägt die Bezeichnung „ADE20K“ und ermöglicht u.a. die Segmentation von Personen, Gebäuden, Treppen und Möbelstücken. In Abbildung 1 wurde der Datensatz „PASCAL VOC 2012“ verwendet.

## 1 Faltung

Anwendungen wie Deeplab basieren in der Regel auf einem neuronalen Netz, das sich den Faltungsprozess zu Nutze macht. Hierbei wandert ein sogenannter Kernel über das Bild und berechnet für dieses eine Repräsentation, die kleiner ist und weniger Informationen trägt. Dieser Kernel besteht aus einer Matrix von Gewichten. Diese werden für die Verkleinerung genutzt, indem sie abschnittsweise mit den im Bild vorhandenen Pixeln multipliziert werden, wie in Abbildung 2 zu sehen ist. Hier wird beispielsweise der untere rechte Pixel des Ergebnisses berechnet. Der Kernel überlappt zu diesem Zeitpunkt die neun Ausgangspixel in der rechten Ecke (gelb dargestellt). Die Berechnung für diesen Ergebnispixel lautet demnach: 1x1+1x0+1x1+1x0+1*1+0x0+ 1*1+0x0+0x1=4. (in Anlehnung an [4])

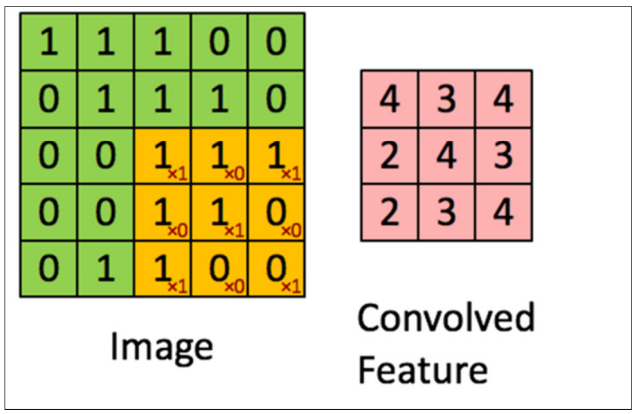



**Abbildung 2: Darstellung des Faltungsvorganges [4]**

## 2 Altrous-Faltung

In DeepLab wird zusätzlich zur normalen Faltung auch die sogenannte Atrous-Faltung genutzt. Mithilfe der Atrous-Faltung ist es möglich, Pixel mit unterschiedlichen Abständen in Zusammenhang zueinander zu bringen. Bei einer normalen Faltung werden nebeneinander liegende Pixel betrachtet. Dies geschieht bei der Atrous-Faltung bei einem Abstandswert von eins. Sobald jedoch der Wert höher wird, werden Lücken zwischen den betrachteten Pixeln gelassen. Der Unterschied ist in Abbildung 3 ersichtlich. (in Anlehnung an[5])

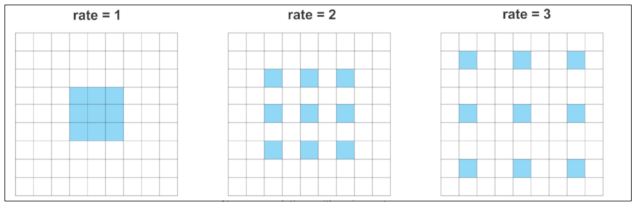



**Abbildung 3: Beispiele für Atrous Rasterungen [5]**

Nur der Abstand zwischen den Pixeln ändert sich mit zunehmenden Werten, die Anzahl der betrachteten Pixel bleibt hingegen gleich. Was bedeutet, dass mehr Kontext gewonnen werden kann, ohne die Anzahl der betrachteten Parameter zu erhöhen. (in Anlehnung an [6] )

Mittels Kombination von mehreren Atrous-Faltungen mit unterschiedlichen Abstandswerten für dasselbe Pixel wird eine deutlich bessere Klassifikation des Pixels ermöglicht. Das Zusammenfügen dieser Ergebnisse erfolgt in der sogenannten Atrous-Pyramide, die exemplarisch in Abbildung 4 zu sehen ist. (in Anlehnung an [6])

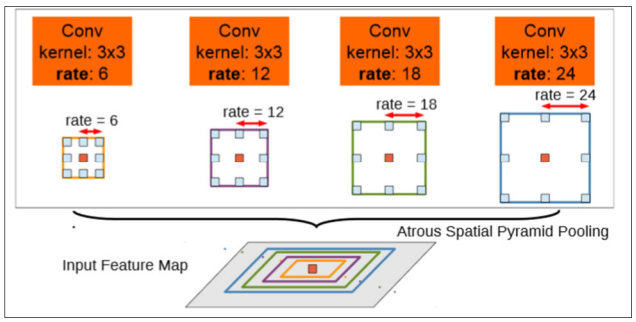



**Abbildung 4: Darstellung der Atrous-Pyramide [6]**

## 3 Skalierung

Ohne Skalierungen wäre die Qualität der Ergebnisse deutlich schlechter, da hierdurch die Größe und die Position der zu erkennenden Objekte an Relevanz verliert, d.h. wenn nur mit Objekten einer Objektgruppe in einem gewissen Größenbereich in den Trainingsdaten gearbeitet wird, ist es dem Netz dennoch möglich, Objekte dieser Gruppe, die sich weiter im Vorder- oder Hintergrund befinden, zu erkennen. Das Nutzen der Eigenschaften unterschiedlicher Skalierungen ist die Voraussetzung, um eine zeitgemäße semantische Bildsegmentation zu ermöglichen. Zur Umsetzung der Skalierungen gibt es zwei Möglichkeiten. Eine davon ist das Nutzen sogenannter skip-nets, bei denen zunächst der Backbone (die Grundlage) trainiert wird und anschließend durch das Extrahieren der Eigenschaften (feature extraction) der multiplen Skalierungen das tiefe Netzwerk repariert bzw. leicht anpasst wird. Da es sich um zwei sequentielle Schritte handelt, führt die Verwendung eines skip-nets zu einer längeren Trainingsdauer. Aus diesem Grund wird in DeepLab durch Verwendung eines share-nets die andere Möglichkeit zur Umsetzung der Skalierungen genutzt. Hierbei werden Eingabebilder in der Größe an unterschiedliche Skalierungen angepasst, welche für das Training eines gemeinsamen tiefen Netzwerkes verwendet werden. Das Endergebnis wird dann aus der Verschmelzung der Eigenschaften der erzeugten Skalierungen berechnet, um eine pixelweise Klassifizierung zu ermöglichen. Die beiden genannten Möglichkeiten der Umsetzung der Skalierungen sind in Abbildung 5 zu sehen. (in Anlehnung an [7])

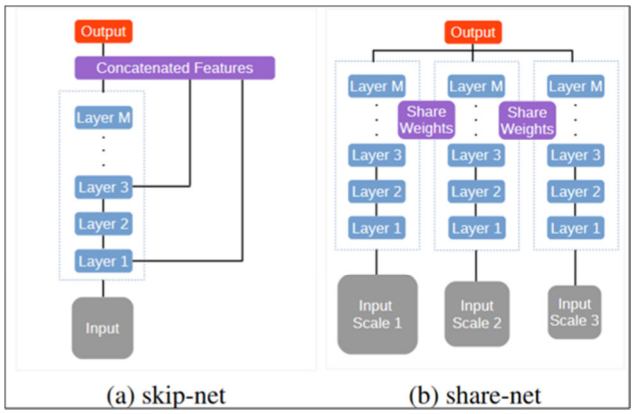



**Abbildung 5: Vergleich von skip-net und share-net [7]**

In DeepLab wird ein share-net benutzt, welches um das Segmentationsmodel adaptiert wird. Das Resultat ist ein sogenanntes Attention Model. Mit dessen Hilfe wird in DeepLab pixelweise klassifiziert. Wie in Abbildung 6 zu sehen ist, werden für jede Skalierung individuelle Ergebnisse in einer Karte, in der pixelweise klassifiziert wird, erzeugt. Bei einem höheren Skalierungswert haben hier kleinere Objekte ein höheres Gewicht (Relevanz) und bei einem kleinen Skalierungswert größere Objekte. Mithilfe dieser Ergebnisse werden Skalierungen in DeepLab ermöglicht. (in Anlehnung an [7])

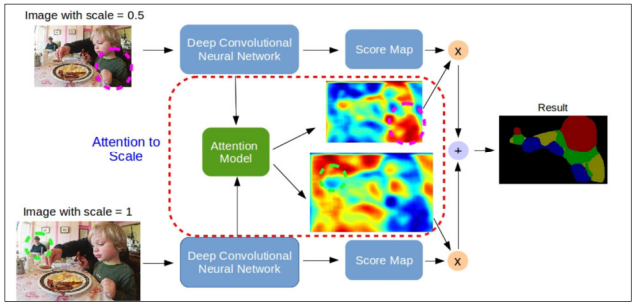



**Abbildung 6: Funktionsweise des Attention Models [7]**

## 4 Training

Es ist möglich, den Trainingsprozess bei DeepLab mittels Trainingsdaten durchzuführen, bei denen die Kennzeichnung der Objekte nur sehr schwach vorhanden ist. Ein Beispiel hierfür ist das Lernen mit Bildern, von denen nur bekannt ist, was auf ihnen zu sehen ist aber nicht exakt wo, wie in Abbildung 7 zu sehen ist. Die für den Trainingsprozess notwendigen Informationen über die Objekte werden in einer entsprechenden Textdatei gespeichert. (in Anlehnung an [8])

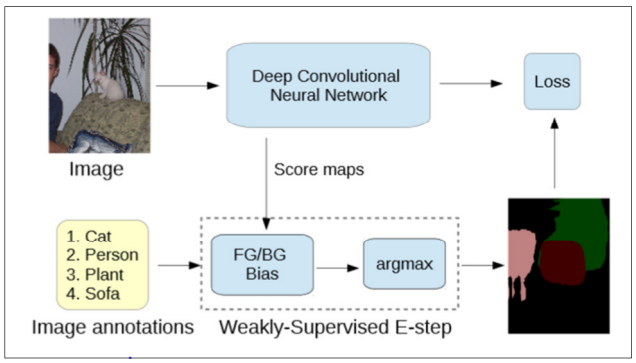



**Abbildung 7: Darstellung des Trainingsvorgangs mit image-level labels [8]**

Es besteht auch die Möglichkeit, mit dem exakten Gegenteil, nämlich mit besonders stark gekennzeichneten Objekten in den Trainingsbildern, welche durch eine vorgegebene pixelweise Klassifikation der Objekte auf den Trainingsbildern ermöglicht wird, zu arbeiten. Hier ist genau bekannt, wo welches Objekt sich auf dem Bild befindet. Weiterhin besteht die Möglichkeit einer Kombination beider genannter Vorgehensweisen. In Abbildung 8 ist eine solche Hybridform zu sehen. Während im oberen Bereich das Lernen mit pixelweiser Klassifikation zu sehen ist, zeigt der untere Bereich das Trainieren mit einem schlechter klassifizierten Datensatz. Die Ergebnisse bei ansonsten gleichen Bedingungen waren bei der pixelweisen Klassifikation früher deutlich besser. Jedoch ist es dem Google Research Team mit DeepLab gelungen, auch mit schlecht identifizierten Bildern und den bereits angesprochenen Hybridformen vergleichbare Ergebnisse zu erzielen, wobei deutlich weniger Aufwand beim Erstellen der entsprechenden Trainingsdatensätze anfällt. (in Anlehnung an [8])

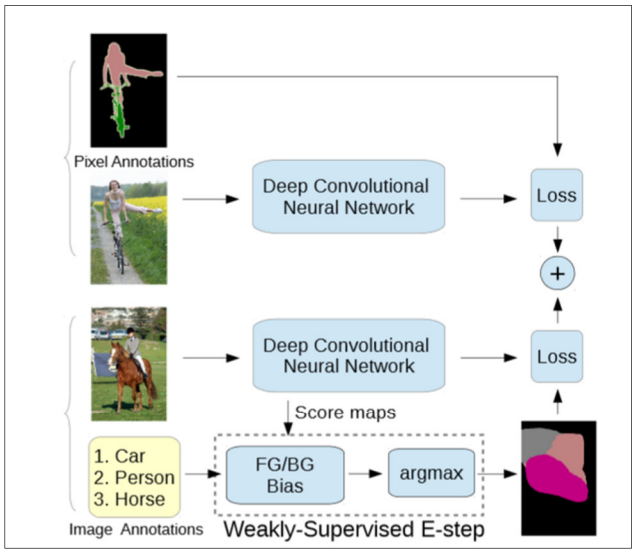



**Abbildung 8: Hybridform aus schwach und stark gekennzeichneten Objekten in den Trainingsdaten [8]**

## 5 Anleitung zum Erstellen eines eigenen Netzes

In der folgenden Anleitung wird Schritt für Schritt in Anlehnung an [3] ein Netz mithilfe des VOC2012 Datensatzes erstellt. Hierfür sind, da dieses Projekt für interessierte Schüler gedacht ist, keine Vorkenntnisse im Bereich des maschinellen Lernens notwendig. Jedoch sollte für das Trainieren des Netzes aufgrund der hohen Anzahl an Trainingsbildern mehrere Stunden zur Erstellung eingeplant werden. Auch wenn hier mit der Programmiersprache Python gearbeitet wird, welches in der Regel plattformunabhängig funktioniert, empfiehlt sich für die Durchführung ein auf Linux- basierendes System zu verwenden, da es bei Versuchen auf Windows innerhalb dieses Projektes zu einigen Schwierigkeiten gekommen ist. Dabei ist aber aufgrund des großen Rechenaufwandes von einer virtuellen Maschine abzuraten. Zunächst muss auf dem Rechner Anaconda eingerichtet werden, welches unter dem folgenden Link heruntergeladen werden kann: https://www.anaconda.com/distribution/. In einem Linux-basierten System wird Anaconda folgendermaßen installiert:

Zunächst ist der Downloadordner zu öffnen, in welchem ein Rechtsklick ausgeführt wird, um anschließend „Open in Terminal“ auszuwählen. Nun sollte sich ein schwarzes Fenster öffnen, in dem Befehle eingegeben werden können. Hier ist nun bash und danach der Name der heruntergeladenen Datei einzugeben, wie es in Abbildung 9 zu sehen ist. Im Anschluss daran beginnt der Installationsprozess. Anschließend muss wiederholt die Entertaste betätigt bzw. auf Anfrage mit „yes“ geantwortet werden. Dadurch wird Sicherheitsbedingungen zugestimmt und es wird die Funktion von conda-Befehlen in der Konsole auf globaler Ebene ermöglicht. Damit ist der Installationsprozess von Anaconda abgeschlossen.

```
(base) patrick@patrick-Switch-SA5-271:~/Downloads$ bash Anaconda3-2019.03-Linux-x86_64.sh

Welcome to Anaconda3 2019.03

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
```

**Abbildung 9: Installation von Anaconda (Screenshot vom Terminal)**

Im nächsten Schritt wird die Zip von der in den Quellen erwähnten GitHub-Seite unter dem Pfad „models“ heruntergeladen, in dem auf die Schaltfläche „Clone or download“ geklickt und anschließend “download ZIP“ auswählt wird, wie in Abbildung 10 zu sehen ist. Die heruntergeladene ZIP Datei ist nun zu extrahieren. Danach wird der durch diesen Prozess gewonnene Ordner geöffnet. Anschließend ist der Unterordner „research“ und danach wiederum dessen Unterordner „deeplab“ zu öffnen. Nun ist aus dem Verzeichnis, in welchem sich auch diese Textdatei befindet, das Shellskript train-voc.sh herunterzuladen und in den Ordner zu kopieren. Jetzt ist in den Ordner „datasets“ zu navigieren und wie bereits beschrieben das Terminal zu öffnen.

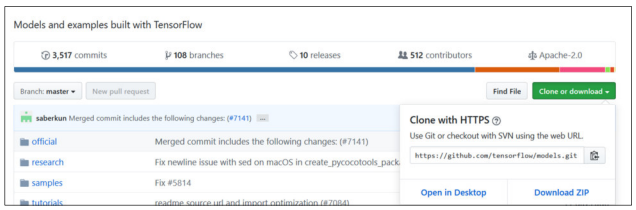



**Abbildung 10: Darstellung zur Vorgehensweise des Herunterladens von GitHub [9]**

Geben sie im Terminal den Befehl: „conda create --name test python=3.6 --channel conda-forge“ ein, um eine neue Anaconda-Arbeitsumgebung zu erststellen. Statt „test“ können sie auch einen beliebigen anderen Namen wählen. Sie müssen nur in den folgenden Befehlen diesen entsprechend wieder verwenden. In Abbildung 11 ist die Erstellung einer Anaconda-Umgebung im Terminal verdeutlicht.

```
(base) patrick@patrick-Switch-SA5-271:~/Desktop/models-master/research/deeplab/datasets$ conda create --name te
st  python=3.6 --channel conda-forge
WARNING: The conda.compat module is deprecated and will be removed in a future release.
Collecting package metadata: done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.6.11
  latest version: 4.7.5
```

**Abbildung 11: Erstellen einer Anaconda-Arbeitsumgebung (Screenshot vom Terminal)**

Sie werden nun gefragt, ob sie einige Pakete installieren wollen. Bitte bestätigen Sie dies mit y. Alle weiteren Befehle in dieser Anleitung müssen in der von Ihnen angelegten Arbeitsumgebung ausgeführt werden. Um diese zu betreten, geben Sie den Befehl „conda activate test“ ein. Diesen Befehl können sie auch jederzeit wieder verwenden, sollten Sie den PC herunterfahren oder versehentlich das Fenster schließen. Vergewissern sie sich jedoch, dass Sie im richtigen Ordner agieren. Anschließend müssen die Befehle „conda install tensorflow“ und „conda install pillow“ eingegeben werden, um weitere notwendige Bibliotheken zu installieren. Bestätigen sie hierbei auf Nachfrage wieder mit y.

Geben Sie den Befehl „bash download_and_convert_voc2012.sh“ in dem im Ordner datasets geöffneten Terminal ein, um den benötigten Datensatz herunterzuladen. Mit dem Kommando c .. im Terminal gelangen Sie anschließend in den Ordner DeepLab. Öffnen Sie nun die Datei „train.py“ im Ordner DeepLab und suchen Sie nach „fine_tune_batch_norm“ und ändern Sie den Wert auf „False“ ab. Geben Sie nun im Terminal „bash train-voc.sh“ ein, um den Trainingsprozess zu beginnen. Dieser kann aufgrund der hohen Anzahl an Bildern durchaus einige Stunden dauern. Die Resultat-Zip finden Sie, indem Sie ausgehend vom Ordner Dataset „pascal_voc_seg“ öffnen und dessen Unterordner „init_models“. Extrahieren Sie diese und Sie erhalten den Ergebnisordner. Mit der durch das Training gewonnen Datei ist es nun möglich, Ihr Resultat durch die in dieser Arbeit beschriebenen Simulation zu testen.

## 6 Simulation eines eigenen bzw. eines heruntergeladenen Netzes

Zunächst müssen sie den Ordner „Models“ in ihre persönliche Google Drive Cloud speichern. In diesen Ordner fügen sie dann die von Ihnen durch das Trainieren gewonnene Datei ein.

Nun öffnen sie die innerhalb dieses Projektes modifizierte Demonstrationsdatei „PASCAL VOC 2012.ipynb“, die sie ebenfalls im Projektordner finden. Laden sie die Seite colab.research.google.com/, wo sie sich mit ihrem Google Konto anmelden müssen. Klicken sie danach auf den Reiter „File“ und anschließend auf „Upload notebook“ (siehe Abbildung 12). Wählen Sie nun in dem sich öffnenden Fenster „Durchsuchen“ und laden Sie die in diesem Projektes modifizierte Demonstrationsdatei „PASCAL VOC 2012.ipynb“ hoch, welche sich ebenfalls im Projektordner befindet.

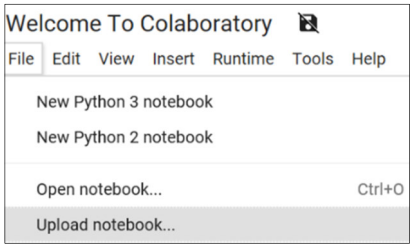
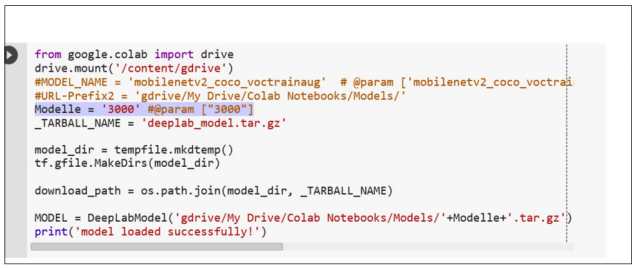



**Abbildung 12: Teil des Upload-Prozesses (Screenshot aus Google Colab)**

Ändern sie zuletzt die in Abbildung 13 zu sehende Zelle, welche die vorletzte auf der Seite ist. Dazu sind die beiden in der Abbildung markierten „3000“ in den Namen ihrer Trainingsdatei umzubenennen, da „3000“ dem Netz, welches innerhalb dieser Arbeit erstellt wurde, entspricht. Falls Sie ihr Netz mit dem innerhalb dieser Arbeit entstandenem Netz vergleichen möchten, können sie die Zeile auch in Modelle = 'NameIhresNetzes' #@param ["3000", "NameIhresNetzes"] abändern.



```
from google.colab import drive
drive.mount('/content/gdrive')
#MODEL_NAME = 'mobilenetv2_coco_voctrainaug'  # @param ['mobilenetv2_coco_voctrai
#URL-Prefix2 = 'gdrive/My Drive/Colab Notebooks/Models/'
Modelle = '3000' #@param ["3000"]
_TARBALL_NAME = 'deeplab_model.tar.gz'

model_dir = tempfile.mkdtemp()
tf.gfile.MakeDirs(model_dir)

download_path = os.path.join(model_dir, _TARBALL_NAME)

MODEL = DeepLabModel('gdrive/My Drive/Colab Notebooks/Models/'+Modelle+'.tar.gz')
print('model loaded successfully!')
```

**Abbildung 13: Codeabschnitt, in dem ein neues Netz eingebunden werden kann [10]**

Um das gesamte Skript auszuführen, klicken sie nun auf den Reiter „Runtime“ und wählen sie „Run All“ aus, wie es in Abbildung 14 dargestellt ist. Beim Ausführen des Codes müssen sie dem Skript das Recht geben, auf Ihre Google Drive Cloud zuzugreifen. Klicken Sie dazu auf den Link, der in der vorletzten Zelle des Programms erscheint, woraufhin das zu verwendende Google Konto ausgewählt werden kann. Nun muss zugestimmt werden, dass auf den drive Ordner zugegriffen werden darf. Kopieren Sie anschließend den daraufhin erscheinenden Authentifikationscode, fügen Sie diesen in das entsprechende Textfeld ein und bestätigen sie mit Enter. Nun können Sie ihr Netz in der letzten Zelle anhand von Bild URLs testen.


**Abbildung 14: Ausführen eines gesamten colab-Skriptes (Screenshot aus Google Colab)**

Sollten sie diese Demo mit einem Netz nutzen wollen, das nicht auf dem Datensatz PASCAL VOC 2012 basiert, müssen sie in der zweiten Zelle von „PASCAL VOC 2012.ipynb“ die Objektnamen abändern, um korrekte Ergebnisse zu erhalten. Die zu ändernden Namen sind in Abbildung 15 zu sehen.

Im GitHub-Projekt sind einige derartige Netze unter dem Link https://github.com/tensorflow/models/blob/master/research/deeplab/g3doc/model_zoo.mdsind zu finden.

```
LABEL_NAMES = np.asarray([
    'background', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus',
    'car', 'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'motorbike',
    'person', 'pottedplant', 'sheep', 'sofa', 'train', 'tv'
])
```

**Abbildung 15: Zuordnung von Objektnamen zur Farbtabelle [10]**

## 7 Beispiele

Es folgen einige interessante Beispielanwendungen des innerhalb dieses Projektes entstandenen Netzes mit Hilfe von Bildern aus dem Internet.

In Abbildung 16 ist zu sehen, dass der Fernseher, die Topfpflanze und die beiden Sofas richtig erkannt werden können.

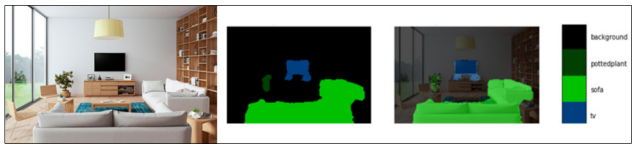



**Abbildung 16: Beispielbild mit richtiger Klassifizierung [10] [11]**

Anders stellt dich das auf Beispielbild 17 dar. Hier ist der Nachteil der Atrous-Faltung erkennbar. Da sich viele Schafe auf dem Bild befinden und Pixel mit größerem Abstand in den Klassifizierungsprozess mit aufgenommen werden, kommt es zu einer Fehlinterpretation durch DeepLab, welche zur Folge hat, dass der Hund fälschlicherweise als Schaf klassifiziert wird.


**Abbildung 17:Fehlergebnis das durch Atrous- Faltung ausgelöst wurde [10] [12]**

Abbildung 18 verdeutlicht, dass Objekte, die zwar zu einer bestimmten im Trainingssatz behandelten Objektgruppe gehören, aber im Aussehen sehr unkonventionell sind, oft vom Programm nicht bzw. falsch erkannt werden.


**Abbildung 18:Unidentifizierbares Objekt da eigentlich zu einer im Training behandelten Objektgruppen gehört [10] [13]**

Sehr interessant ist auch das Phänomen, welches Abbildung 19 aufzeigt. Hier werden nachgebildete Schafe aus Knetmasse ebenfalls als Schafe klassifiziert. Ein ähnliches Verhalten ist auch bei humanoiden Anime- bzw. Manga-Figuren zu beobachten. Diese werden als Personen erkannt.


**Abbildung 19: Reaktion auf künstlich Nachgebildete Objekte [10] [14]**

**QUELLEN**

[1] L. C. Chen et al., https://github.com/tensorflow/models/tree/master/research/deeplab.
[2] https://raw.githubusercontent.com/tensorflow/models/master/research/deeplab/ g3doc/img/vis3.png.
[3] L. C. Chen , Y. Zhu https://github.com/tensorflow/models/blob/master/research/deeplab/g3doc/pascal.md.
[4] S. Saha, A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.
[5] S. Thalles, Deeplab Image Semantic Segmentation Network, https://sthalles.github.io/deep_segmentation_network/.
[6] L. C. Chen et al., DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, https://arxiv.org/pdf/1606.00915.pdf.
[7] L. C. Chen et al., Attention to Scale: Scale-aware Semantic Image Segmentation, https://arxiv.org/pdf/1511.03339.pdf.
[8] Papandreou et al., Weakly- and Semi-Supervised Learning of a Deep Convolutional Network for Semantic Image Segmentation, https://arxiv.org/pdf/1502.02734.pdf.
[9] https://github.com/tensorflow/models.
[10] https:colab.research.google.com/github/tensorflow/models/blob/master/research/deeplab/deeplab_demo.ipynb#scrollTo=7cRiapZ1P3wy.
[11] https://www.holzconnection.de/media/wysiwyg/landingpages/wohnzimmer/holzconnection-wohnzimmer-top-1.jpg.
[12] https://www.hna.de/bilder/2011/04/21/1215967/1027133309-376851498_344-17a7.jpg.
[13] https://www.omlet.de/images/originals/Dog-Dog_Guide-A_Hungarian_Puli_with_a_long_black_corded_coat.jpg.
[14] https://vignette.wikia.nocookie.net/shaunthesheep/images/a/af/Flock2.jpg/revision/latest?cb=20111024105935.

# Entwicklung eines Convolutional Neural Network zur Handschrifterkennung

Tobias Kolb
Hochschule Hof
tobias.kolb@hof-university.de

## ZUSAMMENFASSUNG

In dieser Studienarbeit werden die Ergebnisse festgehalten, die aus der Einarbeitung in das Thema Maschinelles Lernen, der Recherche und anschließenden technischen Umsetzung eines Neuronalen Netzes zur Erkennung von unterschiedlichen handschriftlich geschriebenen Buchstaben von A-J resultieren. Um auf den aktuellen Stand in der Bilderkennung anzuknüpfen, wird ein Convolutional Neural Network (CNN) als Neuronales Netz verwendet. Der Python Source Code des Convolutional Neural Network wurde bereits auf der Plattform Kaggle.com veröffentlich (vgl. Kaggle.com, 2018). Grundlage für die Weboberfläche bietet das GitHub Repository von Siraj Raval (vgl. Raval, 2017). An diesem Source Code wurden Änderungen vorgenommen, um die Oberfläche für dieses Projekt zu verwenden. Bei der Realisierung kommen die Machine Learning Frameworks Tensorflow und Keras sowie die Programmiersprache Python zum Einsatz.

• **Computing methodologies** → *Machine learning approaches*; Artificial intelligence; Neural networks.

## 1 EINLEITUNG

### 1.1 Aufbau der Studienarbeit

Kapitel eins dieser Studienarbeit widmet sich den Anfängen der Bilderkennung. Im zweiten Kapitel „Related Work“ wird ein Überblick über den aktuellen wissenschaftlichen Stand von CNN's, insbesondere in der Bilderkennung gegeben. Im anschließenden Kapitel „Theorie“ werden die in dieser Studienarbeit benötigten Tools, Frameworks und Techniken genauer erläutert. Das Kapitel vier handelt von der Implementierung des Convolutional Neural Network und der Weboberfläche, entwickelt in der Programmiersprache Python. Kapitel fünf beurteilt die gewonnenen Ergebnisse aus dem vorherigem Kapitel. Mit einem Ausblick beendet das sechste Kapitel diese Studienarbeit.

*FWPM: Angewandtes maschinelles Lernen, Sommersemester, 2019, Hof*
.

### 1.2 Maschinelles Lernen

Maschinelles Lernen (ML) ist ein Begriff der sich neben Künstlicher Intelligenz (KI), Deep Learning (DL) und Neuronale Netze wachsender Beliebtheit in der Öffentlichkeit erfreut. Durch die zunehmende Datenmenge die täglich gesammelt wird, braucht es neue Technologien die diese Daten kontrollieren können. Vor ein paar Jahren waren diese Themen für die Öffentlichkeit noch uninteressant und nur für Forschungseinrichtungen relevant. Bereits in den 80er Jahren wurden Forschungen im Bereich der Mustererkennung durchgeführt. Daher ist ML keineswegs eine brandneue Erfindung, sondern kann durch den immer schneller weiterentwickelten technologischen Fortschritt nun sinnvoll eingesetzt werden. Mit den heutigen Grafikprozessoren (GPU) sind Rechengeschwindigkeiten möglich, welche in den 80er Jahren nur theoretisch vorhanden waren. (vgl. Manhart, 2018)

### 1.3 Muster- und Bilderkennung

Für den Fachbegriff Mustererkennung oder auch Pattern Recognition gibt es mehrere Definitionen. Im Grunde beschreiben alle die Mustererkennung als ein Verfahren, in dem gemessene Signale automatisch in Kategorien eingeordnet, um anschließend klassifiziert zu werden. Erste systematische Untersuchungsansätze der Mustererkennung kamen Mitte der 1950er Jahre auf. (vgl. Wikipedia, 2019c)

Zu den richtigen Durchbrüchen der Mustererkennung wird aber erst die Nutzbarmachung von Support Vector Machines und künstlichen Neuronalen Netzen in den späten 1980er Jahren gezählt. Die zunehmende technisch verfügbare Rechenleistung, ist einer der Hauptgründe, warum das Thema ML bzw. Bilderkennung in den letzten Jahren an Beliebtheit gewonnen hat, obwohl sie schon vor Jahrzehnten zu erforschen begonnen wurde. Um das Verfahren der Mustererkennung anzuwenden sind zu aller erst mehrere Schritte vorzunehmen. Abbildung 1 stellt grafisch diese notwendigen Schritte dar.

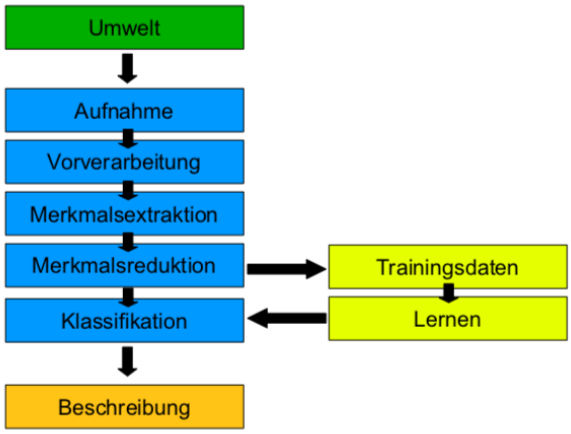



**Abbildung 1: Schritte der Mustererkennung (vgl. Jung, 2016)**

Unsere „Umwelt“ stellt uns die Daten zur Verfügung. Mit Smartphones, Tablet, IoT wachsen diese Daten täglich exponential (vgl. Manhart, 2018). Diese Daten werden anschließend vom Menschen über Sensoren im Schritt der „Aufnahme“ erfasst. Anschließend müssen diese im Schritt, „Vorverarbeitung„ aufbereitet werden. Hierfür wird die Datenmenge reduziert und in ihrer Qualität verbessert. Nachdem die Transformation der Daten in eine problemspezifische Repräsentationsform in der „Merkmalsextraktion„ durch beispielsweise eine Heuristische Extraktion erfolgt, werden diese anschließend in der „Merkmalsreduktion“ durch Graphen repräsentiert und durch Filter reduziert. Waren diese Schritte erfolgreich, werden die Parameter der „Klassifikation„ festgelegt und anschließend die Modelle evaluiert. Danach werden die übrig gebliebenen Daten zufällig in Training Set, Validation Set und Test Set unterteilt. Der letzte Schritt der Mustererkennung ist die „Klassifikation“ der Merkmale in Klassen, meistens unter der Verwendung von Neuronalen Netzen. Am Schluss wird versucht das Muster im Schritt der „Beschreibung“ zu interpretieren. (vgl. Jung, 2016)

In der Bilderkennung schränkt man die gemessenen Eingangssignale auf Bilder ein, deswegen ist sie auch ein Teilgebiet der Mustererkennung. Bei der Entwicklung von Algorithmen zur Bilderkennung spielt Machine Learning eine zentrale Rolle. Denn zum Gewinnen unerschlossenen Wissens ist eine große Datenmenge erforderlich, mit dessen Hilfe die Algorithmen lernen. Der Lernvorgang ist ein ressourcen-intensiver und aufwändiger Prozess. (vgl. Tiedemann, 2018)

## 2 RELATED WORK

Diese Studienarbeit stellt keinesfalls eine Erweiterung der aktuellen Forschung dar, sondern ist vielmehr eine praktische Ausführung des aktuellen Wissensstandes der Forschung zu dem Thema Buchstabenerkennung mithilfe eines CNN.

### 2.1 Convolutional Neuronal Network

CNN ist eine Deep Learning Architektur, welche zur Erkennung von visuellen Mustern mit extremer Variabilität und Robustheit gegenüber Verzerrung und einfachen geometrischen Transformationen entwickelt wurde. Eine extreme Variabilität ist zum Beispiel bei handgeschriebenen Zeichen vorzufinden. Grundlage von CNN für die handschriftlich und maschinell gedruckte Zeichenerkennung bildet das LeNet-5. Dieses Neuronale Netzwerk wurde bereits in den 90er Jahren von Yann LeCun entwickelt.(vgl. LeCun, [n. d.]) (vgl. Y. LeCun, 1989)

Neben der Bilderkennung findet es in vielen Bereichen, wie der Textverarbeitung, Verwendung. CNN ist in der Lage, Input in Form einer Matrix zu verarbeiten. Der Input kann ein Bild sein, welches in einer Matrix (Breite x Höhe x Farbkanäle) dargestellt wird. Ein CNN ist in der Lage Objekte in einem Bild unabhängig von der Position des Objektes in diesem Bild zu erkennen. Der Aufbau besteht aus Filtern (Convolutional Layer) und Aggregations-Schichten (Pooling-Layer), die sich abwechselnd wiederholen und am Ende aus einer oder mehreren Schichten von normalen vollständig verbundenen Neuronen (Dense/ Fully Conntected Layer).(vgl. Becker, 2018) (vgl. Culurciello, 2017)

Das Besondere eines CNN liegt in den Convolutional Layern. Diese Schicht ist die eigentliche Faltungsebene. Hier werden einzelne Merkmale der Eingabedaten erkannt und extrahiert. Bei der Bildverarbeitung können dies Linien, Kanten oder bestimmte Formen sein.

Die nächste Ebene verdichtet und reduziert die Auflösung der erkannten Merkmale. Diese Schicht wird oft auch als Pooling-Schicht oder Subsampling-Schicht bezeichnet. Die Verdichtung und Reduktion verwendet das Maximal-Pooling oder Mittelwert-Pooling. Hierbei werden überflüssige Informationen verworfen und so die Datenmenge reduziert. Dadurch kann sich die Berechnungsgeschwindigkeit erhöhen.

Nach einer vorgegebenen Anzahl an Wiederholungen von Convolutional Layern und Pooling-Layern bildet die vollständig verbundene Schicht den Schluss. Diese letzte Schicht besteht aus Neuronen, welche in mehreren Ebenen angeordnet sein können und deren Anzahl von den Klassen oder Objekten, die das CNN unterscheiden soll, abhängig ist. (vgl. Stefan Luber, 2019)

## 3 THEORIE

Dieses Kapitel dient primär der Erläuterung und Abgrenzung der in der Studienarbeit eingesetzten Technologien.

### 3.1 Maschinelles Lernen

Machine Learning oder auch maschinelles Lernen ist das Erlangen von neuem Wissen ohne dem Computer diese schrittweise einprogrammiert zu haben. Hierbei wird versucht dem Computer das menschliche Lernen anzueignen. Dadurch ist der Computer in der Lage neue Erkenntnisse aufgrund von vorherigem Wissen zu generieren. Neben aufwendigen mathematischen Formeln sind auch große Ansammlungen von Daten zwingend notwendig. Durch Maschinelles Lernen wird

der Mensch dabei unterstützt effektiver und kreativer zu arbeiten. Artificial Intelligenz (AI) oder auch Künstliche Intelligenz (KI) bilden den wissenschaftlichen Bereich, der Maschinelles Lernen enthält aber sich auch mit dem Aufbau einer KI beschäftigt. Denn neben dem eigentlichen Lernen muss Wissen auch effizient abgespeichert, sortiert und abgerufen werden. Machine Learning ist eine Sammlung von mathematischen Methoden der Mustererkennung und bildet somit einen Teilbereich der künstlichen Intelligenz. Mithilfe dieser Algorithmen ist es möglich alltägliche, aber auch sehr spezifische Probleme zu lösen. (vgl. Aunkofer, 2018)

### 3.2 Tensorflow und Keras

Tensorflow (TF) ist eine durchgängige Open-Source Plattform für maschinelles Lernen, welches über ein umfassendes, flexibles Ökosystem von Tools, Bibliotheken und einer starken Community verfügt. ML gestützte Anwendungen können damit problemlos erstellt und bereitgestellt werden. Durch die starke Flexibilität von TF werden Netzwerkberechnungen automatisch auf mehreren CPUs, GPUs, Servern usw. bereitgestellt, was die zunehmende Beliebtheit von TF als Backend-Engine begründet. Tensorflow verwendet das datenstrom-orientierte Paradigma. In diesem wird ein Datenfluss Berechnungsgraph erstellt, welcher aus Knoten und Kanten besteht. Ein Datenfluss Berechnungsgraph kann mehrere Knoten haben, die wiederum mit Kanten verbunden sind. In TF steht jeder Knoten für eine Operation, die Auswirkungen auf eingehende Daten haben. Das generelle Arbeiten mit Tensorflow unterteilt sich in zwei Schritte (vgl. Rosebrock, 2016):

**Erstellen eines Berechnungsgraphen:**
Modellierung des für die spezifische Anwendung benötigte Modell

**Ausführung des Berechnungsgraphen:**
Nachdem das Modell fertig modelliert ist, kann es ausgeführt werden

Keras bildet ein System um komplexere numerische Berechnungsmodule wie Tensorflow. Keras abstrahiert die benötigte Komplexität, die für den Aufbau eines komplexen Netzwerks notwendig ist und bietet dem Benutzer eine einfache und benutzerfreundliche Oberfläche zum schnellen Erstellen, Testen und Bereitstellen von DL-Architekturen. Als Backend-Engine kann auf Tensorflow zurückgegriffen werden. (vgl. Rosebrock, 2016)

### 3.3 Python und Flask

Die Programmiersprache wird nicht nur wegen ihrer Schlichtheit immer beliebter sondern auch wegen ihrer stark zunehmenden Bibliotheken im Bereich ML. Neben Tensorflow gibt es viele weitere Bibliotheken und Frameworks wie scikit-learn, pylearn2 usw. Durch die gute Lesbarkeit von Python Programmen ist die Programmiersprache, eine gute Wahl für das Entwickeln von komplexen Big-Data Applikationen.(vgl. Sadowska, 2018)

Flask ist ein für Python entwickeltes Webframework, bei dem der Fokus auf Erweiterbarkeit und guter Dokumentation liegt. Flask kommuniziert über die WSGI-Schnittstelle, welche zum Zeitpunkt der Erstellung dieser Studienarbeit der aktuelle Stand der Entwicklung für die Kommunikation zwischen Webserver und Webanwendung im Pyhton Umfeld ist. (vgl. Wikipedia, 2019a)

### 3.4 notMNIST Datensatz

Die MNIST (Modified National Institute of Standards and Technology) Datenbank enthält eine große und kostenlose Ansammlung von handschriftlichen Zahlen. Die Datenbank ist im Bereich ML stark verbreitet und wird oft zum Trainieren und Testen verwendet. Die notMNIST Datenbank wurde von Yarosla Bulatov erstellt, indem er öffentlich verfügbare Schriftzeichen entnommen und daraus deren konkrete grafische Darstellung (Glyphe) erzeugt hat. Insgesamt umfasst die Datenbank 18726 Bildern welche in 10 Klassen mit Buchstaben von A bis J gegliedert sind. Die einzelnen Bilder haben eine Auflösung von 28x28 Pixeln und wurden von Hand bereinigt. Der Datensatz ähnelt dem MNIST und dessen Struktur. (vgl. Wikipedia, 2019b)

### 3.5 Convolutional Neural Network

Das in dieser Studienarbeit verwendete Neuronale Netz ist ein Convolutional Neural Network. Der grundlegende Aufbau und die Funktionsweise wurde bereits im vorherigem Kapitel behandelt. Im nächsten Abschnitt wird auf den Aufbau dieses Netzes im Rahmen der Studienarbeit anhand des Python Programmcodes eingegangen.

## 4 IMPLEMENTIERUNG

In diesem Kapitel wird der Python Programmcode erklärt, mit dem das Projekt umgesetzt wurde.

### 4.1 Convolutional Neural Network in Python

Zuerst werden alle notwendigen Bibliotheken importiert (siehe Anhang A). Für diese Studienarbeit wird das Python Framework Keras verwendet, welches bereits im vorherigen Kapitel detailliert erläutert wurde. Die Bibliothek PIL (Pillow) ermöglicht das Öffnen, Bearbeiten und Speichern von verschiedenen Bildformaten. Matplotlib wird zur Darstellung der Ergebnisse verwendet und OS bzw. Zipfile zum Interagieren mit dem Betriebssystem bzw. der Verarbeitung von komprimierten Dateien. Der Datensatz unterteilt die einzelnen Bilder in insgesamt 10 Unterordner, welche mit den Großbuchstaben von A - J gekennzeichnet sind. Darin befinden sich die jeweils dazugehörigen Bilder. Der im Anhang B befindliche Programmcode extrahiert die Bilder aus dem gezippten Ordner. Der Source Code im Anhang C zeigt das Einlesen der einzelnen Bilder. Die einzelnen Bilder werden anschließend in einem Array X gespeichert und am Ende auf die Größe 28x28 Pixel reduziert. Nachdem für die Aufteilung in Training und Test Set, die Daten zufällig durchgemischt

werden, werden 70% in X_train gespeichert und die restlichen 30% in X_test und das Gleiche für labels_train und labels_test.

Im Anhang E wird mithilfe einer for-Schleife eine Tabelle mit 5 Spalten und 3 Zellen erzeugt. Diese dient dazu 15 zufällige Bilder als Beispiel auszugeben.

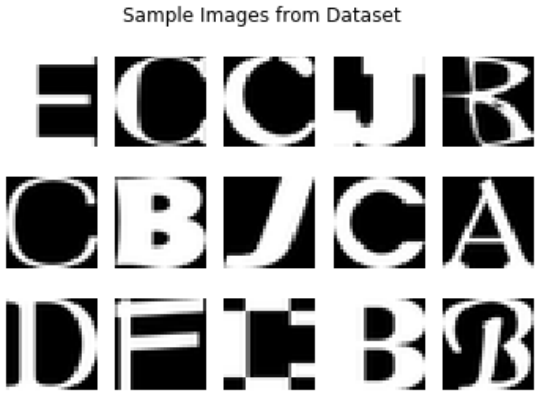



**Abbildung 2: Ausgabe von Beispielbildern**

Abbildung 2 kann man ein Beispiel für diese zufällige Ausgabe von Bildern entnehmen.

Das in dieser Studienarbeit verwendete Convolutional Neural Network besteht aus 2 Convolutional Blöcken, einem Dense Block und einem Output Layer. Der Aufbau des CNN ist dem Anhang F zu entnehmen. Zu erst wird ein Conv2D Layer hinzugefügt. Der erste Parameter „16„ erzeugt 16 Feature Maps. Diese sollen durch einen 3 x 3 Kernel entstehen. Als Aktivierungsfunktion wurde SeLU (Scaled Exponential Linear Units) gewählt. Mit x wird die Dimension des Inputs definert. Als Aktivierungsfunktion hatte sich ReLU (Rectified Linear Units) etabliert. Dennoch wird hier die Funktion SeLU verwendet. SeLUs lernen schneller und besser gegenüber anderen Aktivierungsfunktionen. Zudem ermöglicht SeLU internal normalization.(vgl. Böhm, 2018)

Als nächstes wird ein Pooling-Layer hinzugefügt. Hierbei wird das typische Format gewählt, indem man aus 2 x 2 - Feld das Maximum berechnet. Im Anschluss kommt das Flattening, um die erzeugten Merkmale dem Fully-conntected Layer übergeben zu können. Nach dem Pooling-Layer folgt ein Dropout-Layer. Dieser Layer befindet sich zwischen dem Ersten und dem Zweiten Convolutional Block. Hierbei werden die Dropout-Effekte auf den ersten Block angewandet, welche als Eingabe für den zweiten Convolutional Block dienen. Bei diesem Dropout-Layer besteht eine 50% Chance die Eingabe auf Null zu setzen. (vgl. Brownlee, 2018)

Der zweite Convolutional Block ist vom Aufbau mit dem ersten Block identisch, unterscheidet sich aber in der Wahl der Parameter. So werden im zweiten Block in dem ersten Conv2D Layer 32 Feature Maps erzeugt, welche durch einen 3x3 Kernel entstehen sollen. Der zweite Conv2D Layer dieses Blocks umfasst 64 Feature Maps mit ebenfalls einem 3x3 Kernel. Das Netz als Ganzes hat insgesamt 181 450 Gewichte (siehe Abbildung 7). Um nach dem Trainieren des Netzes noch Auswertungen vornehmen zu können, wird dieses in der training.csv Datei gespeichert. Anschließend wird das Netzwerk trainiert. Insgesamt durchläuft es 40 Epochen mit einer batch size von 64 (siehe Anhang G). Im Anhang H wird das resultierende CNN in der model.h5-Datei gespeichert und die Struktur in der model.json-Datei. Somit kann es anschließend in eine Anwendung eingebunden werden und muss nicht bei jedem Neustart der Anwendung erneut trainiert werden. Anschließend wird im Anhang I eine Bewertung des trainierten Netzes ausgegeben. Hierfür wird die training.csv-Datei ausgelesen, in welche zuvor das Trainingsergebnis gespeichert wurde.

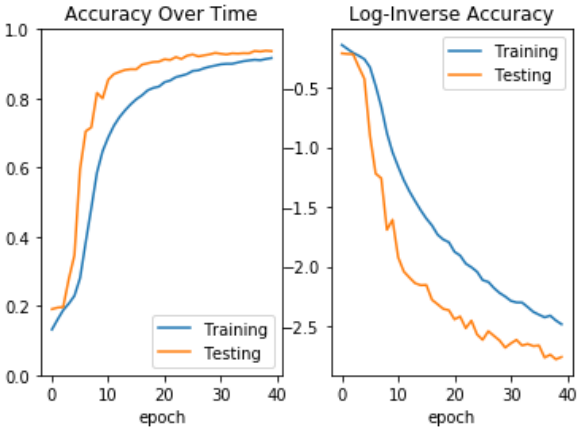



**Abbildung 3: Ausgabe des Plotting Prozesses**

Abbildung 3 liefert das Ergebnis, welches im Anhang I programmiert wurde. Zu sehen sind zwei Graphen. Die Linke Graphik zeigt die Accuracy Over Time. Diese liegt beim Training in den ersten 5 Epochen noch bei unter 20%, steigt danach aber stark an und nähert sich nach 25 Epochen langsam dem Endergebnis von 93,66%. Eine Erhöhung der Trainingsepochen würde das Endergebnis nicht signifikant verbessern. Der Graph, welcher das Testen darstellt, ähnelt sehr stark dem Trainings-Graphen. Einzige Ausnahme ist die vorzeitige und stärkere Steigung. So steigt dieser nach 5 Epochen sehr stark an und erreicht bereits nach 10 Epochen über 80%. Nach 15 Epochen ist kaum noch eine Veränderung wahrzunehmen.

Mithilfe des Codeausschnittes aus Anhang J kann das CNN evaluiert werden. Das Ergebnis (Abbildung 4) ist mit einem insgesamten Verlust (Loss) von 22% und einer Genauigkeit (Accuracy) von 93,66% zu beziffern.

```
Loss: 0.22049779662959895
Accuracy: 93.66%
```

**Abbildung 4: Ausgabe Evaluation Convolutional Neural Network**

## 4.2 Oberfläche

Die Oberfläche in diesem Projekt wurde mit dem Python Framework Flask entwickelt. Um eine vernünftige Web-Oberfläche bereitzustellen, wurde ebenfalls HTML und JavaScript als Web-Programmiersprache eingesetzt. In der Funktion init() (siehe Anhang K) wird zu erst die Struktur des trainierten CNN - welche in JSON Format vorliegt - geladen. Anschließend wird das eigentliche Netz als model.h5-Datei geladen.

Der Anhang L zeigt die Funktion convertImage(). In dieser Funktion wird der über die Oberfläche durch den Benutzer eingegebene Buchstabe in die Bilddatei output.png (siehe Abbildung 5) abgespeichert. Die Datei output.png dient zum Speichern und erneuten Laden zur Auswertung und zum Testen für den Entwickler, um bei möglichen Problemen die Ursache zu ermitteln.

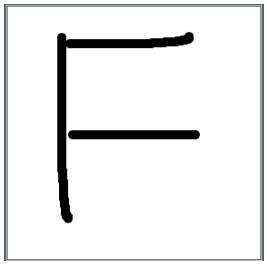


**Abbildung 5: Zwischengespeicherte Eingabe des Benutzers (Originalgröße)**

Die Funktion predict() (siehe Anhang M) bildet die eigentliche REST Schnittstelle. Diese Funktion wird aufgerufen sobald ein Benutzer über die Oberfläche eine Eingabe mit dem Button „predict„ absendet. Zuerst wird der vom Benutzer eingegebene Buchstabe gespeichert. Anschließend wird dieser über die Funktion convertImage() (siehe Anhang L) gespeichert. Danach wird die zuvor abgespeicherte Datei wieder eingelesen, um eine schwarz-weiß Umwandlung des Buchstabens vorzunehmen. Hierbei wird das Bild auf die richtige Größe von 28x28 Pixeln reduziert und in output2.png (siehe Abbildung 6) zwischengespeichert. Abschließend wird das trainierte CNN zur Prognose herangezogen. Das Ergebnis wird an die Oberfläche zurückgesendet und dort für den Benutzer ersichtlich angezeigt.

# 5 ERGEBNISSE

Die Ergebnisse dieses CNN weichen je nachdem wie ordentlich die handschriftliche Eingabe ist stark ab. Bei einer etwas

**Abbildung 6: Zwischengespeicherte Eingabe des Benutzer (reduziert)**

verwackelten Schreibschrift ist das Ergebnis eher ernüchternd. Ist die Eingabe jedoch in deutlich erkennbaren Druckbuchstaben, so liefert das CNN sehr gute Ergebnisse, welche an die 90% grenzen. Da der Datensatz fast nur aus Großbuchstaben besteht, ist es nicht verwunderlich, dass das Ergebnis bei der Eingabe von Großbuchstaben deutlich höher liegt als bei Kleinbuchstaben.

Der Aufbau des in dieser Studienarbeit verwendeten CNN, hat sich daher bewährt. Möchte man das Resultat des Netzes verbessern, wird ein entsprechend größerer Datensatz benötigt, welcher im Internet zwar bereits vorhanden aber kostenpflichtig ist. Der benötigte Datensatz müsste neben allen Buchstaben des Alphabets, sowohl Groß- und Kleinbuchstaben, auch ein viel größeres Volumen aufweisen.

# 6 AUSBLICK

Das im Rahmen dieser Studienarbeit ausarbeitete Projekt zur Handschrifterkennung stellt nur kleines Beispiel des Möglichen dar. Große Tech-Unternehmen wie Microsoft setzen in ihren Office Produkten seit längerem auf eine ausgereifte Handschrifterkennung. In der Software OneNote ist es beispielsweise sehr leicht möglich über ein Tablet-PC Eingaben per Stift in eine Computerschriftart umzuwandeln. (vgl. Malter, 2017)

Mit Zunahme des Tablet-PC- und Smartphone-Marktes wird das Thema Handschrifterkennung von den Herstellern vorangetrieben. Durch die Weiterentwicklung dieser Methodiken wird die digitale Welt immer mehr Einfluss in das normale Leben nehmen.

# LITERATUR

Benjamin Aunkofer. 2018. Machine Learning vs Deep-Learning: Wo liegt der Unterschied? https://data-science-blog.com/blog/2018/05/14/machine-learning-vs-deep-learning-wo-liegt-der-unterschied/. (2018). [Online; Stand 02. Mai 2019].

Roland Becker. 2018. Convolutional Neural Networks - Aufbau, Funktion und Anwendungsgebiete. https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/. (2018). [Online; Stand 16. Mai 2019].

Jason Brownlee. 2018. How to Reduce Overfitting With Dropout Regularization in Keras.

https://machinelearningmastery.com/how-to-reduce-overfitting-with-dropout-regularization-in-keras/. (2018). [Online; Stand 17. Mai 2019].

Timo Böhm. 2018. A first Introduction to SELUs and why you should start using them as your Activation Functions. https://towardsdatascience.com/gentle-introduction-to-selus-b19943068cd9. (2018). [Online; Stand 17. Mai 2019].

Eugenio Culurciello. 2017. The History of Neural Networks. https://dataconomy.com/2017/04/history-neural-networks/. (2017). [Online; Stand 16. Mai 2019].

Bernhard Jung. 2016. Syntaktische und Statistische Mustererkennung. http://bernhard.jung.name/VUSSME/slides/ssme201617-1.pdf. (2016). Folie 31ff. [Online; Stand 16. Mai 2019].

Kaggle.com. 2018. Simple CNN Classifier on notMNIST. https://www.kaggle.com/volperosso/simple-cnn-classifier-on-notmnist. (2018). [Online; Stand 02. Mai 2019].

Yann LeCun. [n. d.]. LeNet-5, convolutional neural networks. http://yann.lecun.com/exdb/lenet/. ([n. d.]). [Online; Stand 16. Mai 2019].

Stefan Malter. 2017. OneNote für Einsteiger. https://onenote-fuer-einsteiger.de/handschrift-erkennung. (2017). [Online; Stand 22. Mai 2019].

Dr Klaus Manhart. 2018. Machine Learning: Was Sie über Maschinelles Lernen wissen müssen. https://www.computerwoche.de/a/was-sie-ueber-maschinelles-lernen-wissen-muessen,3329560. (2018). [Online; Stand 02. Mai 2019].

Siraj Raval. 2017. How to deploy a keras model to production. https://github.com/llSourcell/how$_{t}$o$_{d}$eploy$_{a}$k$_{e}$ras$_{m}$odel$_{t}$o$_{p}$roduction. (2017). [Online; Stand 02. Mai 2019].

Adrian Rosebrock. 2016. Installing Keras with TensorFlow backend. https://www.pyimagesearch.com/2016/11/14/installing-keras-with-tensorflow-backend/. (2016). [Online; Stand 02. Mai 2019].

Luiza Sadowska. 2018. Why is Python so popular in Machine Learning. https://www.merixstudio.com/blog/why-python-so-popular-machine-learning/. (2018). [Online; Stand 02. Mai 2019].

Nico Litzel Stefan Luber. 2019. Was ist ein Convolutional Neural Network. https://www.bigdata-insider.de/was-ist-ein-convolutional-neural-network-a-801246/. (2019). [Online; Stand 02. Mai 2019].

Michaela Tiedemann. 2018. Wie Maschinen sehen lernen - Verfahren zur Bilderkennung. https://www.alexanderthamm.com/de/artikel/wie-maschinen-sehen-lernen-verfahren-zur-bilderkennung/. (2018). [Online; Stand 16. Mai 2019].

Wikipedia. 2019a. Flask. https://de.wikipedia.org/wiki/Flask. (2019). [Online; Stand 02. Mai 2019].

Wikipedia. 2019b. MNIST database. https://en.wikipedia.org/wiki/MNIST$_{d}atabasecite_{n}ote$ − $Multideep$ − 8.2019. $Online$; $Stand$02.$Mai$2019.

Wikipedia. 2019c. Mustererkennung. https://de.wikipedia.org/wiki/Mustererkennung. (2019). [Online; Stand 02. Mai 2019].

J.S. Denker D.Henderson R.E. Howard W. Hubbard L.D. Jackel Y. LeCun, B. Boser. 1989. Backpropagation Applied to Handwritten Zip code Recognition. http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf. (1989). [Online; Stand 16. Mai 2019].

## A BENÖTIGTE PYTHON IMPORTS

```
import keras
from keras.models import Model
from keras.layers import Input, Dense, Dropout
from keras.layers import Reshape, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import to_categorical
import numpy as np
from PIL import Image
import os
from matplotlib import pyplot as plt
import zipfile
from keras.preprocessing.image import
    ↪ ImageDataGenerator

%matplotlib inline
```

## B ZIP HANDLER - AUFRUF

```
zip_ref = zipfile.ZipFile('../notMNIST_small.zip',
    ↪ 'r')
zip_ref.extractall('../notMNIST_small')
zip_ref.close()
```

## C BILDER LADEN

```
X = []
labels = []
DATA_PATH = '../notMNIST_small/notMNIST_small'
# for each folder (holding a different set of
    ↪ letters)
for directory in os.listdir(DATA_PATH):
    # for each image
    for image in os.listdir(DATA_PATH + '/' +
        ↪ directory):
        # open image and load array data
        try:
            file_path = DATA_PATH + '/' + directory
                ↪ + '/' + image
            img = Image.open(file_path)
            img.load()
            img_data = np.asarray(img, dtype=np.
                ↪ int16)
            # add image to dataset
            X.append(img_data)
```

```
            # add label to labels
            labels.append(directory)
        except:
            None # do nothing if couldn't load file

N = len(X) # number of images
img_size = len(X[0]) # width of image
#reshape convert 28x28 picture into 784 vector
X = np.asarray(X).reshape(N, img_size, img_size,1)
labels = to_categorical(list(map(lambda x: ord(x)-
    ↪ ord('A'), labels)), 10)
```

## D AUFTEILUNG DER EINZELNEN BILDER

```
temp = list(zip(X, labels))
np.random.shuffle(temp)
X, labels = zip(*temp)
X, labels = np.asarray(X), np.asarray(labels)
PROP_TRAIN = 0.7 # proportion to use for training
NUM_TRAIN = int(N * PROP_TRAIN) # number to use for
    ↪  training
X_train, X_test = X[:NUM_TRAIN], X[NUM_TRAIN:]
labels_train, labels_test = labels[:NUM_TRAIN],
    ↪ labels[NUM_TRAIN:]
```

## E BEISPIELE

```
num_rows, num_cols = 3, 5
fig, axes = plt.subplots(num_rows, num_cols)
for i in range(num_rows):
    for j in range(num_cols):
        axes[i, j].imshow(X[num_cols*i + j, :, :,
            ↪ 0], cmap='gray')
        axes[i, j].axis('off')
fig.suptitle('Sample Images from Dataset')
plt.show()
```

## F CONVOULTIONAL NEURAL NETWORK

```
shape = X[0].shape
img_in = Input(shape=shape, name='input')
x = Dropout(0.2, name='input_dropout')(img_in)

# conv block 1
x = Conv2D(16, (3,3), activation='selu', name='
    ↪ block1_conv1')(x)
x = Conv2D(32, (3,3), activation='selu', name='
    ↪ block1_conv2')(x)
x = MaxPooling2D((2,2), name='block1_pool')(x)
block_1 = Dropout(0.5, name='block1_dropout1')(x)

# conv block 2
x = Conv2D(32, (3,3), use_bias=False, activation='
    ↪ selu', name='block2_conv1')(block_1)
x = Conv2D(64, (3,3), use_bias=False, activation='
    ↪ selu', name='block2_conv2')(x)
x = MaxPooling2D((2,2), name='block2_pool')(x)
block_2 = Dropout(0.5, name='block2_dropout1')(x)

# dense block 3
x = Flatten(name='block3_flatten')(block_2)
x = Dense(128, activation='selu', name='
    ↪ block3_dense1')(x)
x = Dropout(0.5, name='block3_dropout1')(x)
x = Dense(128, activation='selu', name='
    ↪ block3_dense2')(x)
x = Dropout(0.5, name='block3_dropout2')(x)

# output layer
output = Dense(10, activation='softmax', name='
    ↪ output')(x)

# compile model
model = Model(img_in, output)
model.compile(optimizer='adamax',
            loss='categorical_crossentropy',
            metrics=['accuracy'])
model.summary()
```

## G MODEL TRAINIEREN

```
csv_logger = keras.callbacks.CSVLogger('training.
    ↪ csv', append=True)

# train model
history = model.fit(X_train, labels_train,
          epochs=40, batch_size=64,
          validation_data=[X_test, labels_test],
          callbacks=[csv_logger])
```

## H MODEL ABSPEICHERN

```
# Save the model weights for future reference
model.save('model.h5')

# serialize model to JSON
model_json = model.to_json()
with open( DATA_PATH + "/model.json", "w") as
    ↪ json_file:
    json_file.write(model_json)
```

## I PLOTTING AUSGABE

```
data = np.genfromtxt('training.csv', delimiter=',')
data = data[1:][:,1:]

fig, axes = plt.subplots(1, 2)

# plot train and test accuracies
axes[0].plot(data[:,0]) # training accuracy
axes[0].plot(data[:,2]) # testing accuracy
axes[0].legend(['Training', 'Testing'])
```

```
axes[0].set_title('Accuracy␣Over␣Time')
axes[0].set_xlabel('epoch')
axes[0].set_ybound(0.0, 1.0)

# same plot zoomed into [0.85, 1.00]
axes[1].plot(np.log(1-data[:,0])) # training
    ↪ accuracy
axes[1].plot(np.log(1-data[:,2])) # testing
    ↪ accuracy
axes[1].legend(['Training', 'Testing'])
axes[1].set_title('Log-Inverse␣Accuracy')
axes[1].set_xlabel('epoch')
plt.show()
```

## J EVALUATION

```
# Test
score = model.evaluate(X_test, labels_test, verbose
    ↪ =False)
print('Loss: {}'.format(score[0]))
print('Accuracy: {}%'.format(np.round(10000*score
    ↪ [1])/100))
```

## K INIT() - FUNKTION

```
# initialize these variables
def init():
    json_file = open('model.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(
        ↪ loaded_model_json)
    # load woeights into new model
    loaded_model.load_weights("model.h5")
    print("Loaded Model from disk")

    # compile and evaluate loaded model
    loaded_model.compile(loss='
        ↪ categorical_crossentropy', optimizer='
        ↪ adam', metrics=['accuracy'])
    graph = tf.get_default_graph()

    return loaded_model, graph
```

## L CONVERTIMAGE() - FUNKTION

```
# decoding an image from base64 into raw
    ↪ representation
def convertImage(imgData1):
    imgstr = re.search(r'base64,(.*)', imgData1).
        ↪ group(1)

    with open('output.png', 'wb') as output:
        output.write(imgstr.decode('base64'))
```

## M PREDICT() - FUNKTION

```
@app.route('/predict/', methods=['GET', 'POST'])
def predict():
    imgData = request.get_data()
    # encode it into a suitable format
    convertImage(imgData)
    # read the image into memory
    x = imread('output.png', mode='L')
    # compute a bit-wise inversion so black becomes
        ↪  white and vice versa
    x = np.invert(x)
    # make it the right size
    x = imresize(x, (28, 28))
    imsave('output2.png',x)

    # convert to a 4D tensor to feed into our model
    x = x.reshape(1, 28, 28, 1)
    # in our computation graph
    with graph.as_default():
        # perform the prediction
        out = model.predict(x)
        print(out)
        print(np.argmax(out, axis=1))
        # convert the response to a string
        response = np.array_str(np.argmax(out, axis
            ↪ =1))

        counter = 0
        for x in response:
            if counter == 1:
                print x
                a = ''.join(str(x))
                b = int(a)
                counter += 1
            counter += 1

        return letters[b+1]
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input (InputLayer) | (None, 28, 28, 1) | 0 |
| input_dropout (Dropout) | (None, 28, 28, 1) | 0 |
| block1_conv1 (Conv2D) | (None, 26, 26, 16) | 160 |
| block1_conv2 (Conv2D) | (None, 24, 24, 32) | 4640 |
| block1_pool (MaxPooling2D) | (None, 12, 12, 32) | 0 |
| block1_dropout1 (Dropout) | (None, 12, 12, 32) | 0 |
| block2_conv1 (Conv2D) | (None, 10, 10, 32) | 9216 |
| block2_conv2 (Conv2D) | (None, 8, 8, 64) | 18432 |
| block2_pool (MaxPooling2D) | (None, 4, 4, 64) | 0 |
| block2_dropout1 (Dropout) | (None, 4, 4, 64) | 0 |
| block3_flatten (Flatten) | (None, 1024) | 0 |
| block3_dense1 (Dense) | (None, 128) | 131200 |
| block3_dropout1 (Dropout) | (None, 128) | 0 |
| block3_dense2 (Dense) | (None, 128) | 16512 |
| block3_dropout2 (Dropout) | (None, 128) | 0 |
| output (Dense) | (None, 10) | 1290 |

Total params: 181,450
Trainable params: 181,450
Non-trainable params: 0

**Abbildung 7: Zusammenfassung des Convolutional Neural Network**

# Object Detection mit Hilfe eines vortrainierten Faster R-CNN

David Kohl
Medieninformatik
Hochschule Hof
Deutschland
david.kohl@hof-university.de

## ABSTRACT

Im Rahmen des Modules "Angewandtes Maschinelles Lernen" an der Hochschule Hof befasst sich diese Arbeit mit dem Training eines Neuronalen Netzes zur Erkennung von Spielkarten. Zur Implementierung wurde sich für ein Faster R CNN (Faster Region Based Convolutional Neural Networks) mit einem selbstständig erstellten Datensatz gewählt.

Ziel ist es durch die Aufnahme eines Bildes durch eine Kamera, dessen Inhalt auf eine Spielkarte zu prüfen und bei positivem Befund diese korrekt zu klassifizieren. Als Klasse wurde dabei der Rang der Karten gewählt. (Ass, König, Dame, Bube, Zehn).

## KEYWORDS

Maschinelles Lernen, Objekterkennung, GoogleColab, Tensorflow, Spielkarten, Faster R-CNN

## 1 Einleitung

Durch die voranschreitende Digitalisierung und der enorme Anstieg der Leistungsfähigkeit von modernen Computern verzeichnet die Anwendung von Maschinellen lernen einen gewaltigen Anstieg in Popularität. Ein großer Bereich im Maschinellen lernen ist die Erkennung und Klassifizierung von Objekten und Personen. Diese Technologie wird beispielsweise in selbstständig fahrenden Kraftfahrtzeugen oder in der automatisieren Überwachung eingesetzt.[1]
Da das trainieren solcher Algorithmen eine große Menge von Rechenleistung bedarf, ist es für einfache Haushaltshadware oftmals zeitaufwändig. Um maschinelles lernen zu fördern und für die Öffentlichkeit zugänglich zu machen, bietet Google mit Google Colab eine virtuelle Entwicklungsumgebung, in welcher Nutzer auf spezialisierter Hardware entwickeln können. Mit Hilfe dieses Services wurde in dieser Studienarbeit die Implementierung eines Neuralen Netzes zu Erkennung und Klassifizierung von Spielkarten in Bildern umgesetzt. Es soll dem Anwender möglich sein, beliebige Spielkarten durch eine Kamera zu fotografieren und durch das trainierte Modell auf das Vorhandensein von Spielkarten zu überprüfen. Dabei unterteilt sich diese Arbeit in vier Kernbereiche. In Kapitel 2 werden die Grundlagen von Neuronalen Netzen erläutert sowie das genauere Vorgehen beim Training und Anwendung von neuronalen Netzen beschrieben.
Danach wird in Abschnitt „Vorbereitung der Daten" das Vorgehen zur Generierung und Aufbereitung der Bilder, welches zum Training des Modells verwendet wurde, erläutert. Dazu gehören die Sammlung von Bildern, Erstellung der Label sowie Bounding Boxen und das Erstellen eines Train/Test Splits.
Im nächsten Teil dieser Arbeit wird das eigentliche Training des Netzes genauer erläutert. Hier wird die Konfiguration der Trainingsparameter, der Aufbau des Neuralen Netzes, die Beurteilung sowie die endgültig Trainiere Modell, welches zur späteren Erkennung genutzt wird, beschrieben.
Abschließend gibt Kapitel 5 eine Zusammenfassung der gesamten Studienarbeit und nennt die wesentlichen Erkenntnisse. Letztendlich findet unter Betrachtung sämtlicher Erkenntnisse eine kritische Würdigung dieser Studienarbeit statt. Es werden konkrete Einschränkungen und mögliche Punkte zur Verbesserung zukünftiger Arbeiten genannt.

## 2. Grundlagen

### 2.1 Grundlagen von Convolutional Neural Networks (CNN)

Bei dieser Art von Netzen handelt es sich um eine Art Netz, welche sich besonders zur Verarbeitung von Bild und Videodaten eignet. Ein CNN besteht meist aus folgenden Bauteilen:

#### Convolutional Schicht (Faltungsschicht)

In dieser Ebene findet die eigentliche Faltung statt. Daher stammt auch der Name aus dem englischen Convolution, was Faltung bedeutet. Sie ist in der Lage, in den Eingabedaten einzelne Merkmale zu erkennen und zu extrahieren. Bei der Bildverarbeitung können dies Merkmale wie Linien, Kanten oder bestimmte Formen sein. Die Verarbeitung der Eingabedaten erfolgt in Form einer Matrix. Diese Matrix wird definiert durch Höhe multipliziert mit der Breite sowie der anzahl von Kanälen.

[1] https://www.analyticsindiamag.com/8-uses-cases-of-image-recognition-that-we-see-in-our-daily-lives/

**Pooling Schicht**

Die Pooling-Schicht verdichtet und reduziert die Auflösung der erkannten Merkmale. Hierfür verwendet werden die Schicht Methoden wie das Maximal-Pooling oder Mittelwert-Pooling. Das Pooling verwirft überflüssige Informationen und reduziert die Datenmenge. Die Leistungsfähigkeit beim maschinellen Lernen wird dadurch nicht verringert. Durch das reduzierte Datenaufkommen erhöht sich die Berechnungsgeschwindigkeit.

**Vollständig verknüpfte Schicht**

Den Abschluss des Convolutional Neural Networks bildet die vollständig verknüpfte Schicht. Sie schließt sich den sich wiederholenden Abfolgen der Convolutional- und Pooling-Schichten an. Alle Merkmale und Elemente der vorangehenden Schichten sind mit jedem Outputmerkmal verknüpft. Die vollständig verbundenen Neuronen können in mehreren Ebenen angeordnet sein. Die Anzahl der Neuronen ist abhängig von den Klassen oder Objekten, die durch ein neuronale Netz unterschieden werden sollen.

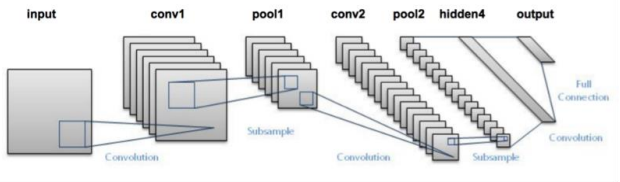



Abbildung 1: Schematische Darstellung eines CNN[2]

Wie in Abbildung 1 zu sehen ist, können die verschiedenen Layer beliebig oft hintereinandergeschaltet werden. Dadurch entstand auch der Begriff „Deep Learning“ (Tiefes Lernen)

## 2.2 Computer Vision

Computer Vision ist ein Teilgebiet der Artificial Intelligence und befasst sich damit, wie Maschinen digitale Bilder und Videos verstehen können. Computer Vision umfasst Filtering, 3D Reconstruction, Segmentation, Depth Estimation Video Compression, Objekterkennung und vielem mehr.
Erstmalig wurden in den frühen 60ern in diesem Gebiet geforscht und erfuhr bis heute einen enormen Anstieg and Bedeutung.
Diese Arbeit konzentriert sich hauptsächlich auf den Bereich der Objekterkennung.

## 2.3 Grundlagen der Objekterkennung mit R-CNN

In der Disziplin der Objekterkennung hat sich im laufe der Zeit eine stätige Weiterentwicklung immer effizienterer Methoden beobachten lassen. Einer dieser Ansätze ist das Faster Region Based Netzwerk (F R-CNN), welchen aus seinen Vorgängern Region Based sowie Fast Region Based Netzwerk entstanden ist. Das F R-CNN wurde erstmal von Shaoqing Ren, Kaiming He, Ross Girshick und Jian Sun in ihrer Veröffentlichung „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks” vorgestellt. Besonderheit dieses Netzes ist, dass es sozusagen aus zwei Netzen besteht, welche zusammen als ein vollständiges Netz arbeiten. Das erste Modul ist ein vollständig verknüpftes tiefes Netz welches, welches Regionen vorschlägt. Dieses Modul nennt man auch ein Region Proposal Network. Diese Vorschläge werden dann durch ein zweites Modul verwendet, um Objekte in dieser Region zu erkennen. Zusammen bilden sie ein vollständiges Netz zur Objekterkennung. Da beide Netze Gegenüber seiner Vorgänger wie R-CNN und Fast R-CNN verzichtet das Faster R-CNN auf die resourcen- und zeitaufwändige Selective Search.
Im ersten Schritt wird durch ein Convolution Layer eine Feature Map erstellt. Außerdem reduziert es die Dimensionen. Diese wird im nächsten Schritt in das Region Proposal Network geleitet. Dort werden sogenannte Anker (engl. Anchor) generiert. Dies geschieht für jedes Layer einer Featuremap durch ein verschiebendes Fenster, welche durch sämtliche Schichten der Ausgabe des ersten Convolution Layers fährt und diese Bereiche mit einer Wahrscheinlichkeit betitelt, dass sich das gewünschte Objekt in einem bestimmten Fenster befindet. Dies geschieht für eine festgelegte Anzahl an Anchor. Diese unterscheiden sich häfig in Größe und Seitenverhältnis. Typische sind Anchor im Format 1:1, 2:1 und 1:2. [3, S.5].

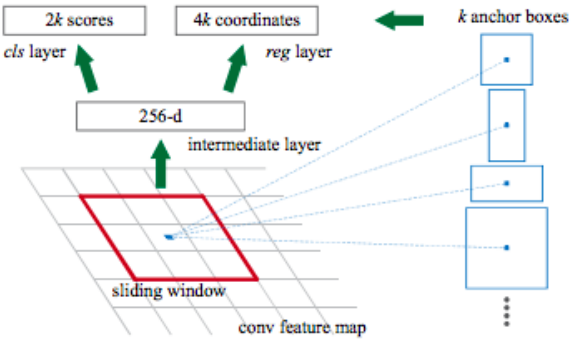



Abbildung 2 : Beispiel Generierung von Ankerboxen ([3] S.3 )

Vorteile dieser Anchor ist es, dass sie gegenüber Transformation des Bildes Resistent sind. Das bedeutet, sollte das Bild aus seiner ursprünglichen Dimensionen verändert werden, so werden seine Anchor automatisch mit verändert und behalten ihre Gültigkeit in Bezug auf die Position des Objektes, da sie relative zum jeweiligen Sliding Window platziert sind.[3, S.3]

Diese sogenannten Region of Interest (RoI), also jene Bereiche, welche mit hoher Wahrscheinlichkeit ein Objekt enthalten werden an das zweite Netz weitergeben, welches dann die eigentliche Objekterkennung und Klassifizierung durchführt.

[2] https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

## 3 Vorbereitung der Daten

Als Grundlage jeder guten Objekterkennung dient ein solides Datenset an Bildern die zum Training des Modelles genutzt werden sollen. Je umfangreicher und vielfältiger das sich Set dabei präsentiert, desto besser ist die zu erwartende Performance. Da es sich bei dieser Arbeit um eine Studentische Arbeit im Rahmen einer Lehrveranstaltung handelt, wurde sich entschlossen, einen eigenen Datensatz zu generieren, um Lerneffekt zu erzielen. Auf Grund von Limitationen durch Zeit und Kapazitäten entschied man sich, lediglich eine Auswahl von Rängen für das Training zu berücksichtigen. Spielkarten werden nicht auf ihre Farbe unterschieden, da dies den Bedarf an generieren Bildern um ein Vielfaches erhöhen würde. Während es bei der Unterscheidung nach Rang die Zugehörigkeit des Blattes keine Rolle spielt, so bedarf es für das Training von Rang und Blatt mehr als das Vierfache an Daten. Die Wahl fiel endgültig auf die Ränge Ass, König, Dame, Bube und Zehn, um die Erkennung von Ass sowie Zahlen mit der Zehn als Repräsentant zu demonstrieren. Bei den Rängen König, Dame und Bube sollte beobachtet werden, wie das Modell mit den doch sehr ähnlichen Abbildungen zurechtkommt. Als Basis wurden Karten aus einem klassischen Bicycle Standard Kartendeck entnommen und vor einer Auswahl von Hintergründen fotografiert. Um eine möglichst hohe Anzahl von möglichst unterschiedlichen Bildern zu erhalten wurde besonderes Augenmerk auf die Auswahl von Hintergrund, Komposition und Entfernung gelegt. Auch Bilder, auf denen mehrere Karten zu sehen sind, wurden in den Datensatz mit aufgenommen. Dies soll dabei helfen, dem späteren Modell die Erkennung von mehreren Objekten, welchen sich gegenseitig verdecken ermöglichen. Dies ist in den folgenden Abbildungen zu sehen.

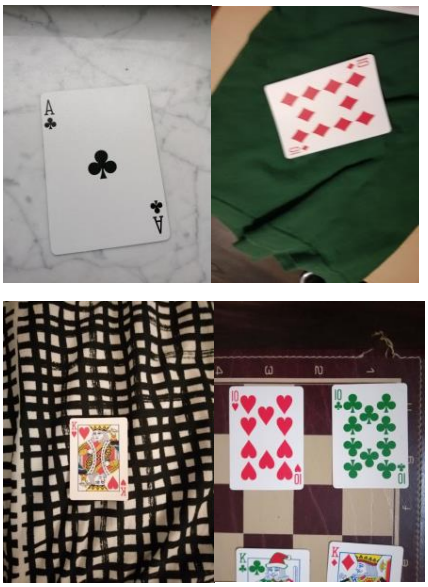


Abbildung 3: Generierung des Datensatzes

Dabei sind Karten teilweise verdeckt, verschwommen oder nicht vollkommen sichtbar. Damit ist der erste Schritt der Datenaufbereitung vollständig. Zum generieren der Bilder wurde eine Moderne Handykamera verwendet, welche Bilder in einer Auflösung von 4160x3120 Pixel erzeugt. Um Ladezeiten des Git Repository zu verkürzen und da die zu erbringende Rechenarbeit mit Größe der Bilder stark anwächst, wurde sämtliche Bilder auf 20% ihrer Originalgröße konvertiert.
Um dem zukünftigen Modell Informationen bezüglich der Objekte zu liefern, auf welches es trainiert werden soll, bedarf es der Anreicherung der Bilder mit einer Kontextdatei. Dazu wurden sämtliche Bilder mit dem kostenlosen Programm lablelimage verwendet.[3] Es können im Voraus eine Liste von Label erstellt werden, welche später durch das Modell erkannt werden sollen. Anschließend durchlaufen sowohl Trainings- als auch Testset dieselbe Prozedur. Im ersten Schritt wird das Bild geladen.

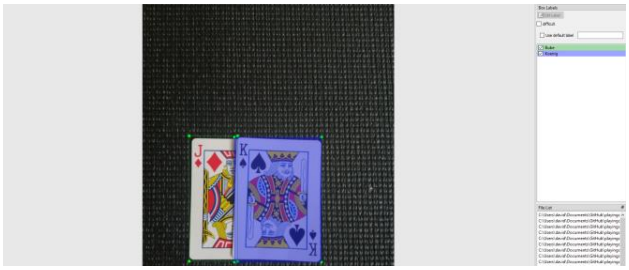


Abbildung 4: Bilder mit Boundingboxes und Labeling

Im zweiten Schritt werden rechteckige Boxen um jene Bereiche gezogen, welche ein später zu erkennendes Objekt enthält. Die Inforationen bezüglich Art des Objekts werden durch ein Label markiert. werden in einem vordefinierten XML Format gespeichert, welches mit dem PASCAL VOC Format entspricht. Dieser Schritt ist sehr zeitaufwändig. Jedoch ist ein umfangreicher und vielfältiger Bilderdatensatz unerlässlich, um ein akzeptables Ergebnis beim späteren Training zu erreichen. Am ende des Daten Preprocessing werden am Schluss die Gesamtheit der Bilder in einem Verhältnis von 80/20 in Trainingset und Testset aufgeteilt.
Zusammen mit den Helferklassen zu Generierung der csv-Dateinen aus den XML - Dateien und dem Skript zu Generierung der Tensorflow Records wurden diese in einem Github Repository zusammengeführt, um einfachen Zugang während der Arbeit in Colab zu ermöglichen.

[3] labelimage

## 4 Training und Anwendung

Als Grundlage dieser Arbeit dient das Framework, welches durch Chengwei Zhang in einem Online Artikel beschrieben wurde. [4] Es wurde dahingehen modifiziert, um das in Kapitel 3 beschriebene Datenset zu verwenden. In seinem Blogeintrag erläutert er, wie mit Hilfe eines vortrainierten Netzes das Erlernen von neuen Objekten beschleunigt werden kann. Grund dafür ist, das zu Beginn des Trainings die Layer des Netzes bereits für die Erkennung von Objekten optimierte Konfiguration besitzen. In dieser Arbeit wurde ein vortrainiertes Modell, welches durch Tensorflow bereitgestellt wurde, dazu verwendet, neue Objekte in Form von Spielkarten zu erkennen. Bei dem ausgewählten Modell handelt es sich um das Faster R-CNN with Inception v2, welches an dem Oxford-IIIT Pets Dataset trainiert wurde. Da es sich bei Spielkarten um eine neue Form von Objekten handelt, die mit den im bisherigen Trainingsdatenset enthaltenen Objekten keine große Ähnlichkeit haben, bedarf es einer größeren Menge an Daten um das Modell wirksam auf Spielkarten zu trainieren. Erste Versuche zeigten ein nicht zufriedenstellendes Ergebnis. Erste Erkenntnisse ließen darauf schließen, dass die Grundgesamtheit der zum Training verwendeten Daten nicht ausreichen war. Besonders die Unterscheidung der Ränge König, Dame und Bube zeigte sich als problematisch, da diese weniger prägnante Unterscheidungsmerkmale aufweisen wie zum Beispiel Zahlenränge oder das Ass. Auch das Erkennen von mehreren Karten sowie teilweise verdecke Katen erwies sich als problematisch. Das Hinzufügen einer neuen Reihe von Trainingsbildern zeigte verfehlte die erhoffte Wirkung und zeigte lediglich einen geringfügigen Anstieg in Präzision und Recall. Erste Modelle wurden mit einer Anzahl von 1000 Trainingsschritten durchgeführt, um eine schnelle Evaluation zu ermöglichen. Dies erwies sich jedoch als nicht weiter haltbar, da mit steigender Datenmenge eine größere Anzahl von trainingsschritten erforderlich war, um deren volles Potential auszuschöpfen. Aus diesem Grund wurde die Anzahl der Schritte um den Faktor zehn erhöht. Nach ersten Testläufen zeigte sich jedoch, dass bereits nach etwa 3000 Trainingsschritten eine Sichtbare Stagnation der Präzision um etwa 0.9 eintrat. Anders verhielt es sich jedoch in der Verbesserung des Recalls, welcher sich deutlich langsamer verbesserte. Aus diesem Grund wurde sich dafür entschieden mit 5000 Trainingsschritten einen Mittelwert zu finden welcher sowohl hinsichtlich auf Performance als auch unter Betrachtung der Zeit einen vertretbar war. Als eine Präzision von 0.9 und ein Recall von ungefähr 0.75 erreicht wurde, entschließ man sich dazu, das Training als abgeschlossen zu betrachten. Dadurch wurde ein akzeptables Ergebnis erzielt. In der folgenden Abbildung ist zu sehen, wie das trainierte Netzt mehrere Spielkarten erfolgreich erkennt. Jedoch lässt sich auch erkennen, dass die Vorhersagewahrscheinlichkeit zwischen den drei Karten sich signifikant unterscheidet.


Abbildung 5: Ergebnis des finalen Netzes - Groundtruth rechts; Vorhersage links

Daraufhin wurde der Modellgraph exportiert, welche sämtliche Parameter des endgültigen Netzes enthält. Um bei späteren Anwendungen das zweistündige Training zu vermeiden wurde der Graph ebenfalls in das Github Repository aufgenommen, aus welchem er bei Bedarf schnell geladen werden kann.
Als nächster Punkt stand die Implementierung der Aufnahme von Bildern durch eine an den Computer angeschlossene Kamera. Glücklicherweise wurde in der Codebeispielsammlung in Colab die Einbettung einer Kamera vorimplementiert. Mit Hilfe einiger Anpassungen wurde das Speicherverzeichnis auf das Verzeichnis geändert, aus welchem später die Bilder zur Objekterkennung geladen werden sollen.

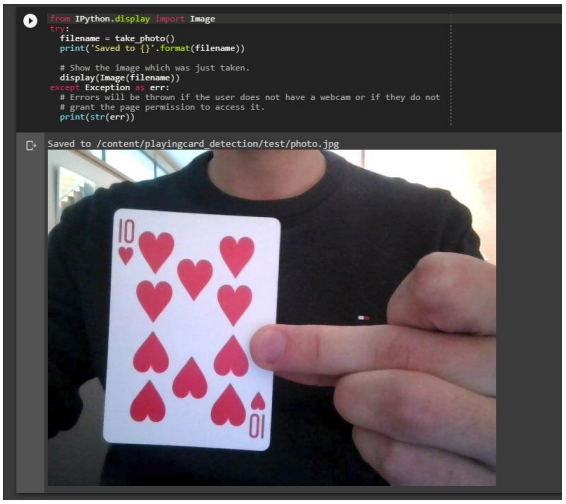



Abbildung 6: Aufnahme des Testbildes

[4] https://www.dlology.com/blog/how-to-train-an-object-detection-model-easy-for-free/

Alternativ ist so auch möglich, Bilder manuell in dieses Verzeichnis hochzuladen, sollte der Computer über keine passende Kamera verfügen. Anschließend wird das zuvor auf die Spielkartenobjekte trainierte Netz geladen und die eigentliche Objekterkennung gestartet. Nach kurzer Zeit wird dann das analysierte Bild mitsamt Boundingbox und vorhergesagtem Label angezeigt. Ebenfalls wird die Wahrscheinlichkeit angezeigt, mit der es sich um das vorhergesagte Objekt handelt.

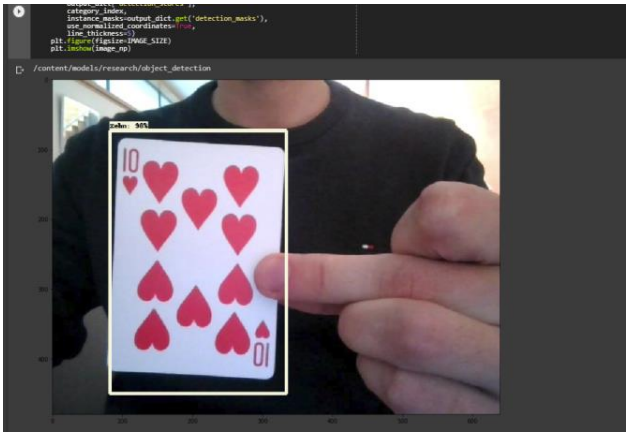



Abbildung 7: Verarbeitetes Bild mit relevaten Informationen

## 5 Fazit

### 5.1 Zusammenfassung

Zusammenfassend hat diese Studienarbeit gezeigt, wie auf Basis einen vortrainieren Modells und mit Hilfe eines eigenen Datensatzes die Erkennung von beliebigen Objekten durch ein Neuronales Netz implementiert werden kann. Auch hat sich gezeigt, dass durch das kontinuierlich hinzufügen von zusätzlichen Bildern die Vorhersagequalität des Modells verbessern lässt. Während erste Versuche noch deutliche Defizite bei der Vorhersage aufwiesen, wurde letztendlich ein akzeptables Ergebnis erzielt. Das finale Modell wurde mit 221 Bildern trainiert, welche durch einen Train/Testsplit im Verhältnis 80/20 aufgeteilt wurden. Das Modell bedarf 5000 Trainingsschritten und 50 Evaluierungsschritten mit einer Batchsize von je 12 Bildern.
Die durchschnittliche Präzision liegt bei 0.9. Wie aus den Tensorboard Graphen ersichtlich, werden sämtliche Bilder in den Kategorien der großen Bounding Boxes eingeordnet. Der durschnittliche Recall liegt bei 0.6. Die Kombination aus hoher Präzision und geringem Recall lässt darauf schließen, dass das Modell strenge Kriterien anwendet um Spielkarten zu erkennen. Dies kann dazu führen, dass Spielkarten teilweise nicht erkannt werden. Jedoch ist die Chance, Objekte welche keine Spielkarten sind als solche zu identifizieren, gering.

### 5.2 Kritische Würdigung

Trotz Erreichens des gesetzten Zieles, Spielkarten durch eine Kamera einzulesen und anschließend deren Inhalt auf Spielkarten zu prüfen, gibt es weiterhin Punkte zur Verbesserung und Limitationen. So bestehen weiterhin Mängel im Erkennen von mehreren verdecken oder nicht vollständig sicherbarer Karten. Eine große Beeinträchtigung stellt die Größe des verwendeten Datensatzes dar. Diese wird als maßgebliche Beeinträchtigung der Leistung vermutet, da durch das kontinuierliche Hinzufügen von neuen Datensätze zwar eine Steigerung der Vorhersagequalität beobachtet werden konnte, jedoch primär Bilder von einzelnen oder vollständig sichtbaren Karten verwendet wurde. Für zukünftige Entwicklung sollte der Datensatz um weitere Einträge ergänzt werden. Auch die Vervollständigung der verbleibenden Ränge kann in Betracht gezogen werden, um ein vollständig funktionsfähiges Modell zu repräsentieren. Auch wurde die Performance unter schwierigen Bedingungen wie geringem oder zu hoher Lichtintensität als Schwachpunkt erkannt. Teilweise schafft die Verwendung einer höherwertigen USB-Kamera Abhilfe, da diese besser auf unterschiedliche Lichtintensitäten reagiert. Trotz all der vorherig genannten Einschränkungen wurde das in der Einleitung definierte Ziel, ein Modell so zu trainieren, dass es Spielkaten erkennen und korrekt zuordnen kann, erfüllt.

## Quellenabgaben:

[1] Tzuta Lin,labelimg, https://github.com/tzutalin/labelImg

[2] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

[3] Chengwei Zhang (2019).. https://www.dlology.com/blog/how-to-train-an-object-detection-model-easy-for-free/

[4] Hao Gao (2017), Faster R-CNN Explained, https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8

# Trainieren eines Neuronalen Netzes für Mobile Anwendungen

Am Beispiel einer Pilzerkennungsapp für Android

Andreas Glaser
Fakultät Informatik
Hochschule Hof
Hof Bayern Deutschland
Andreas.Glaser@hof-university.de

## Motivation

Immer weniger Menschen können die bei uns heimischen Pilze noch eindeutig erkennen, hinzu kommen neue giftige Pilzsorten, die durch den Klimawandel bei uns heimisch werden könnten.[1] Ziel ist es also den angehenden Pilzsammlern eine Hilfe für die Identifizierung von Pilzen zu bieten. Es sollen nur Vermutungen zur Pilzart bereit gestellt werden, mehr ist Moralisch und Rechtlich nicht vertretbar. Im Idealfall muss der Anwender nur noch mithilfe von Fachliteratur, die eindeutigen Merkmale der vermuteten Art und deren giftigen Doppelgänger überprüfen, so liegt die finale Entscheidung beim Anwender, die Anwendung hilft so die Entscheidung zu beschleunigen.

In diesem Artikel wird die Erstellung einer Pilzerkennungsapp und die dafür notwendige Theorie behandelt. Kapitel 1 bis 3 behandeln die dafür notwendigen Programme, Kapitel 4 die Theorie, 5 und 6 die verwendete Technik und in Kapitel 7 wird der Trainingsprozess erläutert. In Kapitel 8 wird erklärt wie das trainierte Netz in eine App eingebunden wird. Abschließend werden in Kapitel 9 und 10 mögliche Verbesserungen vorgestellt und analysiert.

## 1 Tensorflow

Tensorflow ist ein von Google entwickeltes Framework, mit dem leicht, auch ohne tiefere Programmierkenntnisse, Neuronale Netze trainiert werden können.

## 2 Anaconda

Anaconda ist eine Software um Python Umgebungen zu erstellen und zu verwalten, so kann z.B. die selbe Software bequem auf einem Rechner mit verschiedenenen Versionen von Python oder anderer Bibliotheken ausgeführt werden.

Anaconda kann hier heruntergeladen werden: https://www.anaconda.com/distribution/

Installiert wird Anaconda über die Konsole mit folgendem Befehl:

```
$ bash Anaconda3-2019.03-Linux-x86_64.sh
```

Hierbei müssen die Fragen entweder mit „yes“ oder mit betätigen von „Enter“ bestätigt werden. Insbesondere die letzte Frage, hierbei handelt es sich um die Frage ob der „conda“-Befehl im Betriebssystem registriert werden soll, dies ist notwendig um „conda“ überall benutzen zu können.

## 3 Git

Git ist ein Versionierungstool um effizient Code zu verwalten und muss installiert werden um den benötigten Code leicht herunterladen zu können, dies macht man mit folgendem Befehl:

```
$ sudo apt install git
```

[1] https://www.br.de/br-fernsehen/sendungen/quer/181025-quer-themen-100.html

# 4 Theorie

## 4.1 Gewichte

Gewichte sind Fließkommazahlen die vom Netz eingestellt werden um das Eingangsbild richtig zu erkennen. Sie stellen die Regeln dar nach denen das Netz ein Bild einordnet.

## 4.2 Convolution

Diese Ausführungen stützen sich auf den Artikel von Chi-Feng Wang [2].

Bei Convolutions werden Kernel, Matrizen aus Gewichten, über das Bild „geschoben“, sie berechnen daraus ein neues Bild mit weniger Informationen das das Netz besser verarbeiten kann.

Wenn man ein Ausgangsbild mit 12x12 Pixeln und drei Dimensionen die drei Farbkanäle RGB, also ein Bild mit 12x12x3 Werten in eine Convolution mit einem Kernel mit 5x5x3 Gewichten gibt erhält man ein Ausgangsbild mit 8x8x1 Werten, das eine vereinfachte Version des Eingangsbildes darstellt.

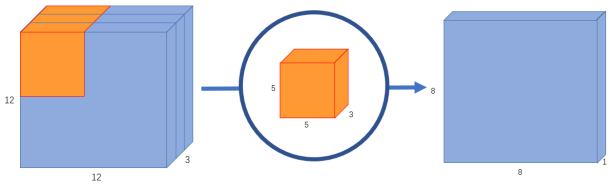



*Abbildung 1: Convolution mit einem Kernel*

Verwendet man mehr als einen Kernel erhält man ebenso viele Ausgangsbilder und diese kann man übereinander legen. Nimmt man z.B. 256 Kernel erhält man ein Ausgangsbild mit 8x8x256 Werten.

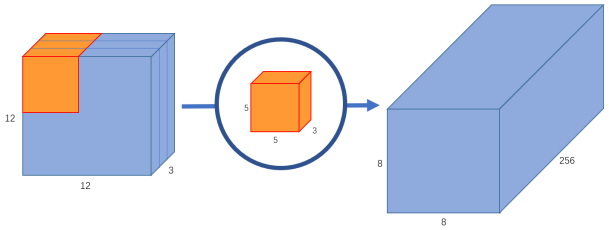



*Abbildung 2: Convolution mit 256 Kernel*

Der Vorteil an dieser hohen Anzahl an Kernel ist nun dass die Kernel sich auf bestimmte Eigenschaften spezialisieren können und diese besonders prägnant in die jeweilige Ebene übertragen können.

## 4.3 Pointwise Convolution

Pointwise Convolutions sind Convolutions mit 1x1 Kernel, die die Dimensionen des Eingangsbildes haben in diesen Fall also 3, so ergeben sich 1x1x3 Gewichte. Es werden die Kernel mit ihren Gewichten auf einzelne Pixel angewendet. Besonderheit hierbei ist dass das Bild hierbei nicht kleiner wird, ein Eingangsbild mit 8x8x3 Werten wird ein Ausgangsbild mit 8x8x1 Werten produzieren.

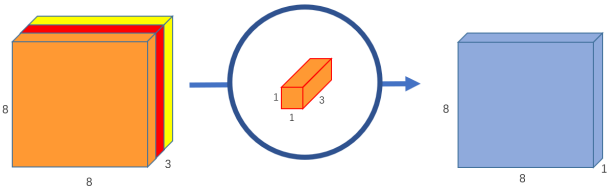



*Abbildung 3: Pointwise Convolution mit einem Kernel*

Auch hier können wieder die Ergebnisse mehrerer Convolution mit verschiedenen Kernel kombiniert werden. So wird mit 256 Kernel ein Ausgangsbild mit 8x8x256 Werten.

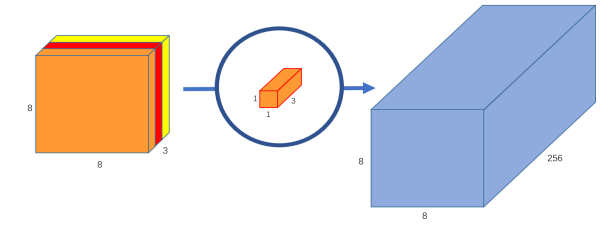
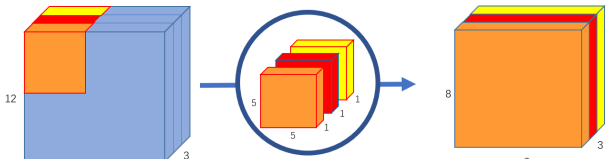



*Abbildung 4: Pointwise Convolution mit 256 Kernel*

## 4.4 Depthwise Convolution

Bei Depthwise Convolutions existiert für jede Dimension des Eingangsbildes ein eigener ein-Dimensionaler Kernel, der nur auf die Werte dieser Dimension angewendet wird. Am Ende werden die Ausgangsbilder jedes Kernels wieder zusammen gefügt.



## 5 Vorstellung tfmobile[2]

Tfmobile ist eine von Google entwickelte Android App die den Videostream der Handykamera nimmt, einzelne Bilder herausnimmt und in ein neuronales Netz füttert. Das Ergebnis wird dann im oberen Bereich des Bildschirms angezeigt.


*Abbildung 6: Die fertige Pilzerkennungsapp im Emulator*

## 6 MobileNet

MobileNet ist eine Architektur die für den Einsatz auf leichtgewichtigen mobilen Geräten optimiert ist. Sie basiert auf den Einsatz von depthwise und pointwise Convolutions. Im Gegensatz zu vergleichbaren Architekturen ist sie deutlich kleiner und erzielt teilweise sogar bessere Ergebnisse, als z.B. GoogleNet, VGG 16 und Squezzenet. [3]

## 7 Anleitung wie man das Netz trainiert

Diese Anleitung ist in Anlehnung an das Tutorial von Johnny Chan [1] entstanden.

[2]https://codelabs.developers.google.com/codelabs/tensorflow-for-poets-2/#0

### 7.1 Skripte herunterladen

Zu nächst wird ein Ordner für das Projekt angelegt:

```
$ mkdir fungi
```

Und in diesen gewechselt:

```
$ cd ./fungi
```

Dann werden Skripte von Git heruntergeladen:

```
$ git clone
https://github.com/tzutalin/ImageNet_Utils my-
ImageNet_Utils
$ git clone
https://github.com/googlecodelabs/tensorflow-
for-poets-2 my-tensorflow-for-poets
```

### 7.2 Anaconda Workspace einrichten

Nach der Installation von Anaconda, kann ein Workspace eingerichtet werden, hierfür wurde die Python Version 3.6 gewählt. Der Befehl hierfür lautet:

```
$ conda create --name fungi python=3.6
```

Nach dem Erstellen kann der Workspace mit folgendem Befehl geöffnet werden:

```
$ conda activate fungi
```

Wenn der Workspace erfolgreich geöffnet wurde erkennt man das daran dass der Name in der Konsole angezeigt wird:

```
(fungi) $
```

Um Tensorflow im Anaconda Workspace zu installieren muss folgender Befehl eingegeben werden:

```
(fungi) $ pip install "tensorflow==1.4.1"
```

Ob Tensorflow erfolgreich installiert wurde, kann mit dem „conda list“ Befehl überprüft werden. In der Ausgabe muss folgendes vorkommen:

```
tensorflow                1.4.1
```

## 7.3 Bilder für das Training herunterladen

Als nächstes werden die Bilder heruntergeladen, alternativ können die Bilder auch direkt von der Seite: http://image-net.org/explore?wnid=n12992868 heruntergeladen werden, hierfür ist aber eine Registrierung notwendig, damit sichergestellt werden kann dass die Bilder nur für Forschungszwecke verwendet werden. Dazu wird der folgende Befehl verwendet:

```
(fungi) $ cd ./fungi/my-ImageNet_Utils
(fungi) $ ./downloadutils.py --downloadImages --wnid n13003061
```

Mit der wnid= n13003061 werden Bilder von Fliegenpilzen heruntergeladen. Hierfür können beliebige Kategorien gewählt werden, sie sind auf der Image.Net Seite zu finden, für den Anfang sollten noch zusätzlich folgende Kategorien heruntergeladen werden:

```
n13030337 → Zinnoberroter Kelchbecherling
n13040629 → Gemeine Stinkmorchel
n13044375 → Riesenbovist
n13003061 → Erdstern
```

Beim Herunterladen werden mehrere Fehler geworfen wie „Not found“ oder „Access Denied“, diese sind aber nicht weiter problematisch solange nach dem nächsten Schritt, dem Aussortieren, wenigstens 250 Bilder pro Kategorie verbleiben.

## 7.4 Bilder aussortieren

Sehr wichtig ist es die heruntergeladenen Bilder, von Hand auszusortieren. Grundsätzlich müssen alle Bilder entfernt werden die keinen Pilz der Kategorie zeigen auch müssen Bilder entfernt werden die den Pilz nicht vollständig darstellen, bei Pilzen sind das oft Bilder von Pilzgerichten. Wichtig ist dass nach dem Aussortieren mindestens 250 Bilder pro Kategorie verbleiben, es ist kein Problem wenn die Menge der Bilder pro Kategorie stark variiert.

Danach müssen im Ordner „my-tensorflow-for-poets/tf_files“ jeweils ein Ordner für Trainingsbilder und einen für Validierungsbilder, mit den Namen „trainbilder“ und „valbilder“, angelegt werden. In jeden dieser Ordner wird noch ein Ordner pro Kategorie, mit der Pilzart als Namen, angelegt. Pro Kategorie werden in dem Ordner „trainbilder“ nun mindestens 200 Bilder, in den entsprechenden Ordner, eingefügt und in dem Ordner „valbilder“ exakt 50 Bilder pro Kategorie, die nicht im Ordner „trainbilder“ vorkommen dürfen.

## 7.5 Tensorboard starten

Tensorboard zu starten ist optional und dient nur der Überwachung des Lernprozesses, wenn dies nicht gewünscht ist kann dirket mit 7.5 Training weitergemacht werden.

Für Tensorboard ist es sinnvoll eine zweite Konsole zu öffnen da Tensorboard gelegentlich Statusmeldungen zurückgibt. Zuerst aktiviert man wieder den Anaconda Workspace und wechselt dann in den „my-tensorflow-for-poets“ Ordner:

```
$ conda activate fungi
(fungi) $ cd ./fungi/my-tensorflow-for-poets
```

Tensorboard wird mit folgendem Befehl gestartet:

```
(fungi) $ tensorboard --logdir tf_files/training_summaries --host=localhost &
```

Tensorboard kann nun im Browser geöffnet werden indem man auf die Seite: http://localhost:6006 navigiert. Die Seite sollte dann wie folgt aussehen:

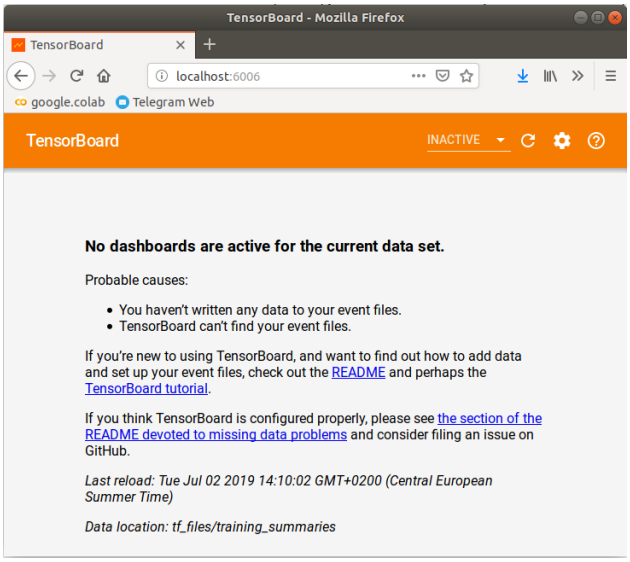



*Abbildung 7: Tensorboard nach dem ersten Start*

Beenden kann man Tensorboard mit folgenden Befehl:

```
(fungi) $ pkill -f "tensorboard"
```

Sobald das Training begonnen hat und die ersten Ergebnisse vorhanden sind, können sie auf Tensorboard analysiert werden.

## 7.6 Training

Für das Training muss zum „my-tensorflow-for-poets“ Ordner navigiert werden falls das in Schritt 7.4 nicht bereits geschehen ist:

```
(fungi) $ cd ./fungi/my-tensorflow-for-poets
```

Zuerst werden einige Umgebungsvariablen definiert:

```
(fungi) $ export TFP_IMAGE_SIZE="224"
(fungi) $ export TFP_RELATIVE_SIZE="0.50"
(fungi) $ export
TFP_ARCHITECTURE="mobilenet_$
{TFP_RELATIVE_SIZE}_${TFP_IMAGE_SIZE}"
(fungi) $ export TFP_IMAGES_DIR="trainbilder"
```

Für die Variable „TFP_IMAGE_SIZE“ wird beispielhaft 224 gewählt. Dies bedeutet das alle Bilder die in das Netz gehen auf 224x224 Pixel skaliert werden. Alternativ kann auch 128, 160 und 192 gewählt werden, mit geringeren Werten berechnet das Netz das Ergebnis schneller, das Ergebnis wird aber schlechter.

„TFP_RELATIVE_SIZE“ steht für die Komplexität des Netzes im Verhältnis zur Komplexesten Variante. So ist das Netz mit einem Wert von 0,5 nur halb so groß wie mit 1,0. Hier gilt auch wieder kleiner bedeutet deutlich schneller und größer dafür Zuverlässiger. Gültige Werte hierfür sind 0,25, 0,50, 0,75 und 1,0.

„TFP_ARCHITECTURE“ fass die vorherigen zwei Variablen einfach nur zussammen.

In „IMAGES_DIR“ wird der Name des Ordners, in dem die Bilder für das Training sind, angegeben. In diesem Fall „trainbilder“.

Um das Training nun zu starten muss lediglich der folgende Befehl ausgeführt werden:

```
(fungi) $ python -m scripts.retrain \
  --image_dir=tf_files/${TFP_IMAGES_DIR} \
  --bottleneck_dir=tf_files/bottlenecks \
  --model_dir=tf_files/models/ \

--summaries_dir=tf_files/training_summaries/ba
sic/"${TFP_ARCHITECTURE}" \
  --output_graph=tf_files/retrained_graph.pb \
  --output_labels=tf_files/retrained_labels.txt \
  --how_many_training_steps=500 \
  --architecture="${TFP_ARCHITECTURE}"
```

Der Parameter „how_many_training_steps“ ist auf 500 gesetzt, bei nur 5 Kategorien ist das vollkommen ausreichend, erst wenn die Anzahl der Kategorien erheblich gesteigert wird muss dieser Wert erhöht werden. Er bedeutet dass jedes Bild 500 Mal in das Netz eingespeist wird.

## 7.7 Mit Tensorboard das Training Evaluieren

Hierfür ist Schritt 7.4 Tensorboard starten notwendig. Man muss lediglich die Tensorboard Seite aktualisieren um die Zwischenstände des Trainings zu sehen. Besonderst interessant hiervon sind die „accuracy“ die Genauigkeit und die „cross_entropy“. Die Genauigkeit sagt einfach aus mit welcher wahrscheinlichkeit das Netz ein Bild richtig einordnet, bei einer Genauigkeit von 0,98 also mit einer 98% Wahrscheinlichkeit. „cross_entropy“ hingegen ist ein Maß wie viel das Netz sich anpassen musste um das aktuelle Bild richtig einzuordnen. Genauigkeit sollte möglichst gegen 1 gehen und „cross_entropy“ gegen 0, dies könnte wie folgt aussehen:

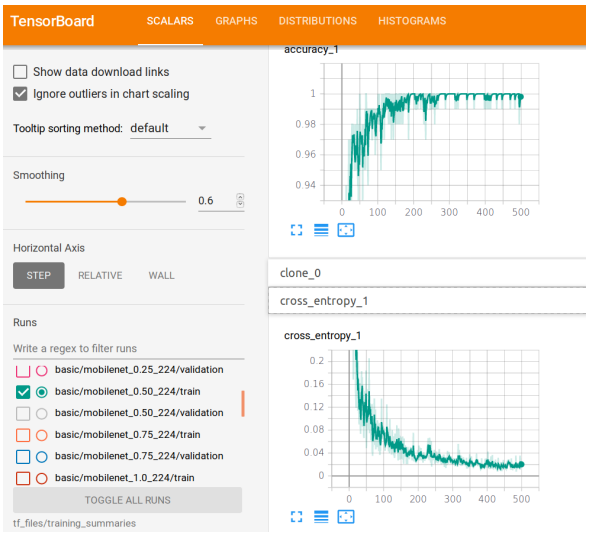



*Abbildung 8: Das Traininsergebnis in tensorboard*

## 7.8 Einordnen eines Testbildes

Um ein Bild vom Netz einordnen zu lassen wird folgender Befehl verwendet:

```
(fungi) $ python -m scripts.label_image \
    --graph=tf_files/retrained_graph.pb  \
    --image=tf_files/valbilder/fliegenpilz/fly-
agaric.jpg
```

Der Wert von „image“ kann mit dem Pfad eines beliebigen Bildes belegt werden. Das Ergebnis sieht dann wie folgt aus:

```
Evaluation time (1-image): 0.228s

Fliegenpilz 0.979888
Gemeine Stinkmorchel 5.1072e-03
Erdstern 5.5167e-08
Riesenbovist 3.83321e-08
Zinnoberroter Kelchbecherling 2.41322e-08
```

Hierbei handelt es sich laut dem Netz mit 98% Wahrscheinlich um einen Fliegenpilz.

# 8 Einbinden des Netzes in tfmobile

In Anlehnung an das Tutorial von Google [4].

Um das trainierte Netz mit tfmoblie nutzen zu können müssen lediglich die Dateien „retrained_graph.pb“ und „retrained_labels.txt“ im Ordner tf_files umbenannt und in den Ordner android/tfmobile/assets verschoben werden. Hierfür können folgende Befehle genutzt werden:

```
(fungi) $ cp retrained_graph.pb
../android/tfmobile/assets/graph.pb
(fungi) $ cp retrained_labels.txt
../android/tfmobile/assets/labels.txt
```

Dann muss Android Studio installiert werden, nach dem Starten von Android Studio muss das Projekt importiert werden:

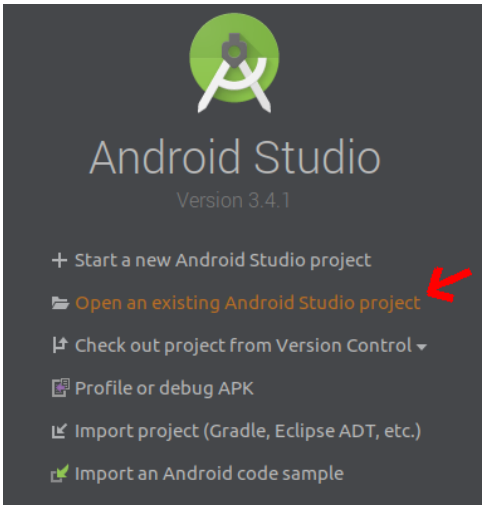



*Abbildung 9: Öffnen des Projektes in Android Studio*

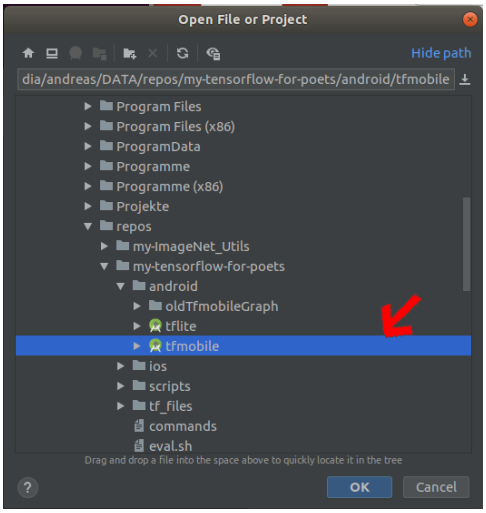



*Abbildung 10: Auswahl des Projektes "tfmobile"*

Unter „Tools“ findet man den AVD Manager mit dem virtuelle Android Geräte erstellt werden können. Alternativ kann auch ein echtes Gerät angeschlossen werden dafür muss aber USB-Debugging auf dem Gerät aktiviert werden weshalb ein virtuelles Gerät empfohlen wird.

Im AVD-Manager muss die Schaltfläche unten links mit „+ Create Virtual Device“ betätigt werden.

Erst muss die Auswahl welches Android Gerät emuliert werden soll getroffen werden, in diesem Fall das Pixel 2:

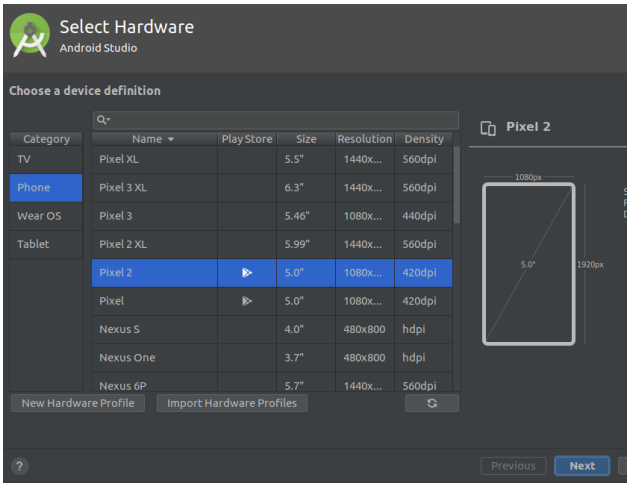



*Abbildung 11: Auswahl des Gerätes*

Im nächsten Fenster muss die Android Version festgelegt werden, in diesem Fall Android Q, zuvor muss die Version mit einem Klick auf Download noch heruntergeladen werden.

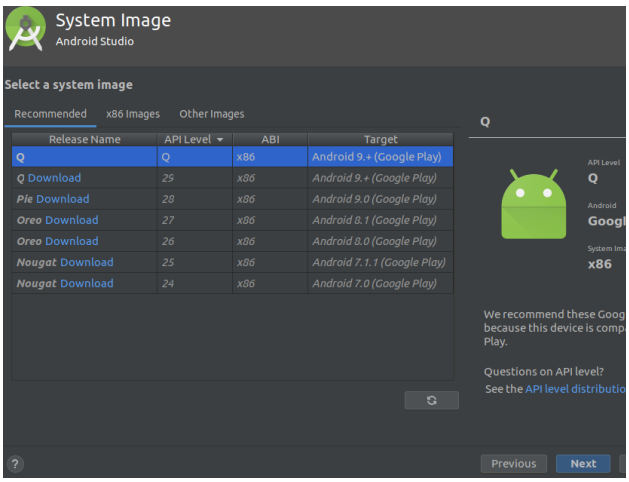



*Abbildung 12: Auswahl der Android Version*

Im letzten Fenster muss noch in den Advanced Settings noch die Rückkamera auf VirtualScene gestellt werden. Dadurch wird ein Raum für die Kamera emuliert in dem eigene Bilder angezeigt werden können. Alternativ kann auch auf Emulated gestellt werden dadurch verwendet die Emulation die Geräte Kamera dies funktioniert allerdings nur wenn es auch eine Kamera hat.

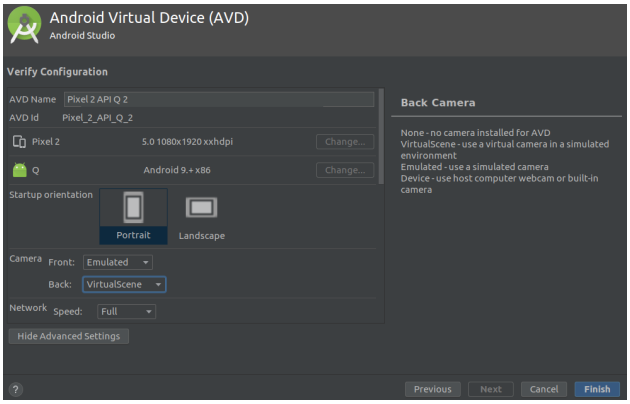



*Abbildung 13: Einstellen der Rückkamera*

Um die Emulation zu starten muss der Task „tfmobile“ gestartet werden:

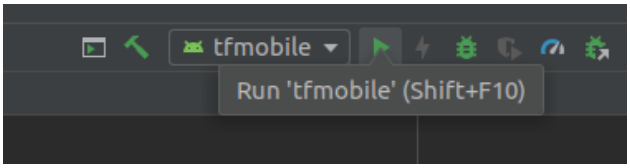



*Abbildung 14: Installieren und Starten der App*

Danach muss das gerade erstellte Pixel 2 nur noch ausgewählt werden:

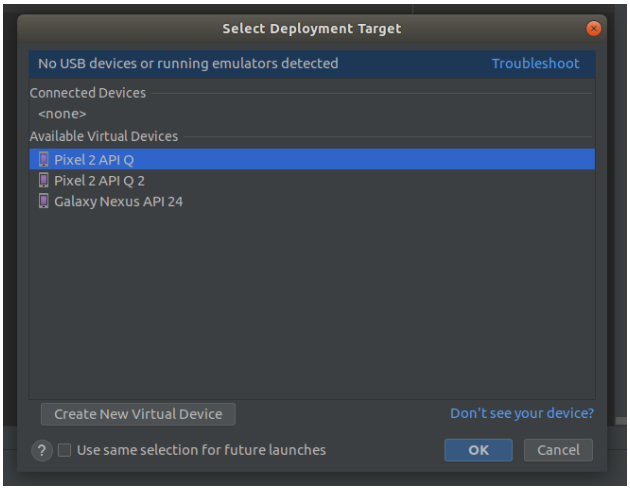



*Abbildung 15: Auswahl des emulierten Gerätes*

In der Emulation kann ein Bild eingestellt werden welches an die Wand des virtuellen Raumes geworfen wird, hierzu muss die Schaltfläche mit den drei Punkten betätigt werden und dann zum Punkt Camera gewechselt werden:

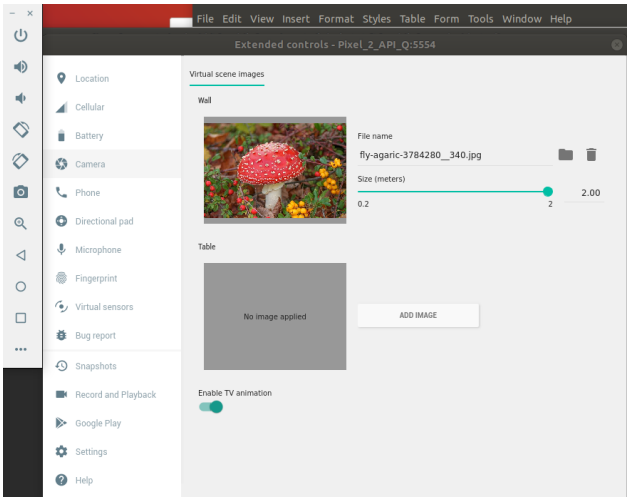



*Abbildung 16: Einstellen eines Testbildes*

# 9 Mögliche Verbesserungen

Wenn man die App auf ein echtes Gerät spielt und die Bilder über die Handykamera aufgenommen werden, werden diese leider so stark abgeändert das das Netz sie nicht mehr richtig erkennt. Dies hängt von der Handykamera ab und folgende Skripte können das Ergebnis trotz Handykamera leicht verbessern. Sie erzeugen dazu Abwandlungen der Trainingsbilder.

## 9.1 Bildform verändern

In Anlehnung an das Tutorial von Francois Chollet[5].

Dieses Skript dreht, verschiebt, spiegelt und schert die Trainingsbilder zufällig und erstellt dadurch 20 neue Varianten die auch zum Trainieren genutzt werden können.

```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
import os
datagen = ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
for image in os.listdir(os.getcwd()):
        if(os.path.isfile(image)):
                img = load_img(image)
                x = img_to_array(img)  # ein Numpy Array mit der Form (3, 150, 150)
                x = x.reshape((1,) + x.shape)  # ein Numpy Array mit der Form (1, 3, 150, 150)
                # die Funktion datagen.flow() generiert 20 augmentierte Bilder pro urspruenglichen Bild
                # und speichert diese in den Ordner 'preview'
                i = 0
                for batch in datagen.flow(x, batch_size=1,
                                          save_to_dir='preview', save_prefix='fly-agaric',
                                                save_format='jpeg'):
                        i += 1
                        if i > 20:
                                break  # ansonsten würden unendlich viele bilder generiert
```

*Abbildung 17: Skript um die Bildform zu verändern*

## 9.2 Bildfarben verändern

Das zweite Skript verändert den Kontrast, die Helligkeit und die Farbsättigung der Bilder und erzeugt so nochmal eine Kopie jedes Bildes die dem Bild einer Handykamera näher kommt.

```
import random, os
from PIL import Image,ImageFilter,ImageEnhance
for image in os.listdir(os.getcwd()):
        if(os.path.isfile(image)):
                #Bild einlesen
                im = Image.open(image)

                #Zufallszahlen generieren
                modContr = round(random.uniform(0.5,0.9),2)
                modBright = round(random.uniform(1.3,2.0), 2)
                modColor = round(random.uniform(0.7,1.0), 2)

                #Modifikatoren anwenden
                contr = ImageEnhance.Contrast(im)
                im = contr.enhance(modContr)
                brigth = ImageEnhance.Brightness(im)
                im = brigth.enhance(modBright)
                color = ImageEnhance.Color(im)
                im = color.enhance(modColor)

                #Speichern des modifizierten Bildes
                im.save(""+image+"_"+str(modContr)+"_"+str(modBright)+"_"+ str(modColor)+".jpg")
```

*Abbildung 18: Skript um die Farbwerte zu verändern*

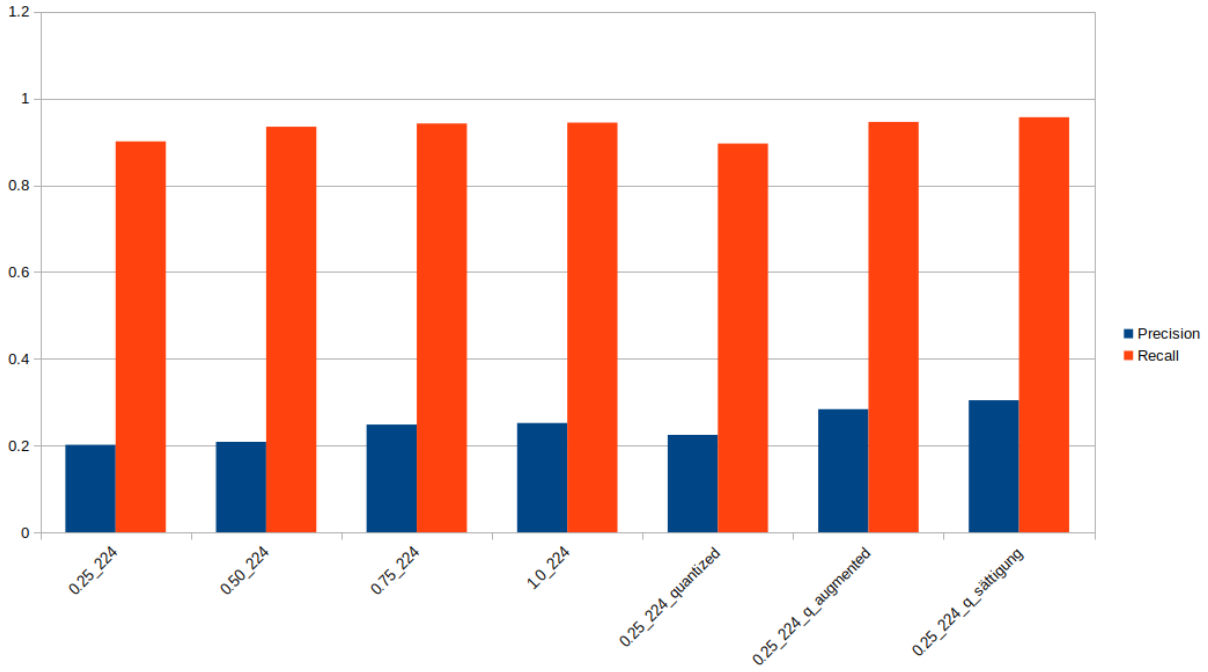



*Abbildung 19: Testergebnis verschiedener Konfigurationen*

## 10 Test verschiedener Konfigurationen

Abbildung 19 zeigt die Ergebnisse der Architektur mit verschiedenen Komplexitäten oder verschiedenen Trainingsdaten. „Recall“ bedeutet ob das Netz einen Pilz richtig erkannt hat und „Precision“ bedeutet mit welcher Wahrscheinlichkeit das Netz ein Bild ohne Pilz als einen Pilz erkannt hat. „Recall“ sollte so hoch wie möglich und „Precision“ so niedrig wie möglich sein. Eine Vervierfachung der Komplexität führt nur zu einer geringen Verbesserung des „Recalls“, gleichzeitig wird auch die „Precision“ schlechter. Wird das Netz „Quantisiert“ das bedeutet dass die Gewichte gerundet werden, nimmt der Speicherbedarf des Netzes enorm ab und der „Recall“ wird dadurch nur geringfügig verschlechtert. Werden die Trainingsdaten nun noch augmentiert und die Farbwerte verändert, kann wieder ein „Recall“ wie bei der Komplexesten Variante hergestellt werden. Die „Preciscion“ leidet hierbei leider merklich, dies ist in diesem Anwendungsfall allerdings kein Problem, da das Netz nicht Pilze von nicht-Pilzen unterscheiden soll sondern die Pilze untereinander.

## Referenzen

[1] Johnny Chan, *Train a basic wild mushroom classifier*, http://fungai.org/2017/12/13/train-a-basic-wild-mushroom-classifier/

[2] Chi-Feng Wang, *A Basic Introduction to Separable Convolutions*, https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

[3] A.G. Howard, M. Zhu, Bo Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, *MobileNets: Efficient Convolutional Neural Networks for Mobile VisionApplications,* https://arxiv.org/pdf/1704.04861.pdf

[4] TensorFlow for Poets 2: TFMobile, https://codelabs.developers.google.com/codelabs/tensorflow-for-poets-2/#0

[5] Francois Chollet, Building powerful image classification models using very little data, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

## Bildnachweis

[2] Chi-Feng Wang, *A Basic Introduction to Separable Convolutions:* 1-5

Autor: 6-19

# Planespotting – Flugzeugerkennung mit TensorFlow und Google Cloud Platform

Daniel Urban
Fakultät Informatik
Hochschule Hof
Hof Bayern Deutschland
Daniel.Urban@hof-university.de

**Abstrakt :**

Diese Studienarbeit beschäftigt sich mit dem Projekt "Planespotting - Flugzeugerkennung mit Tensorflow und Google Cloud Platform", welches die Studierenden des Studiengangs Informatik an der Hochschule Hof im Rahmen der Veranstaltung „Angewandtes maschinelles Lernen" bei Herr Prof. Dr. Sebastian Leuoth in Form einer Studienarbeit durchführen müssen. Es geht dabei speziell um die praktische Anwendung von lernender Software, die sich selbst bei der Durchführung einer bestimmten Aufgabe durch wiederholtes Trainieren immer weiter verbessern soll. Speziell befasst sich dieses Dokument mit der Erkennung und Markierung von Flugzeugen auf großen Luftaufnahmen.

Zum Training wurde das im Rahmen der Vortragsreihe „Tensorflow and deep learning without a PhD" von Martin Görner entwickelte Modell zur Flugzeugerkennung verwendet [1]. Das Training selbst wurde über der von Google zur Verfügung gestellten Cloud Plattform „Google Cloud Platform" auf einem Rechen-Server durchgeführt. Das fertig trainierte Modell konnte anschließend über eine von Martin Görner entwickelte Web-Anwendung getestet und validiert werden. Die Ergebnisse des Projektes werden im Zuge dieser Arbeit erläutert.

## 1 Einführung

Der Computer wird in unserer modernen Welt immer wichtiger. Die zunehmende Digitalisierung erreicht heutzutage immer mehr Bereiche, wodurch sich auch die Anforderungen an Computer-Programme stetig ändern. Ein Bereich, der in den letzten Jahren immer mehr in den Fokus rückt, ist die so genannte Künstliche Intelligenz. Doch was bedeutet das eigentlich? Wie funktionieren selbständig denkende Maschinen? Diese Fragen beschäftigen viele Menschen. Die Antwort darauf ist ganz einfach. Hinter jeder künstlichen Intelligenz steckt im Grunde nur ein Neuronales Netz, welches auf Basis von so genannten künstlichen Neuronen das menschliche Gehirn simulieren soll. Ähnlich wie bei Menschen, sind auch diese Neuronalen Netze dazu in der Lage, durch Übung und Training dazuzulernen. Die Neuronen können mehrere Eingaben verarbeiten, diese gewichten und an eine Aktivierungsfunktion übergeben, welche die Werte für nachfolgende Neuronenschichten berechnet [2].

Neuronale Netze werden vor allem da eingesetzt, wo konventionelle Programme mit einer expliziten Modellierung an ihre Grenzen stoßen. Beispiele hierfür sind die Audio- und Bilderkennung, zu der auch das in dieser Arbeit behandelte Projekt zählt. Je nach Aufgabe muss eine individuelle Architektur des neuronalen Netzes gefunden werden, da schon kleine Änderungen dazu führen können, dass ein Netz mal mehr oder weniger gut konvergiert. Die Architektur, die für dieses Projekt verwendet wurde nennt man Squeezenet. Sie wird in einem späteren Kapitel behandelt.

Eine Form von Neuronalen Netzen, die sich im Bereich der Bildverarbeitung als bewährt erwiesen hat, sind so genannte Convolutional Neural Networks (dt.: faltendes neuronales Netzwerk). Diese Netzwerke bestehen in der Regel aus einem oder mehreren Convolutional Layers gefolgt von einer Pooling Layer. Genaueres wird in einem separaten Kapitel behandelt. [3] Ein Convolutional Network liegt auch dem Planespotting-Programm dieses Projektes zugrunde.

Ein weiterer Ansatz der bei der Entwicklung des Planespotting-Modells verwendet wurde ist YOLO (You Only Look Once, dt.: du schaust nur einmal). Bei diesem Ansatz wird das eingehende Bild, wie der Name schon vermuten lässt, nur einmal betrachtet. Dieser Ansatz ist wichtig, da die Flugzeuge auf den Luftaufnahmen mit grünen Kästchen (Bounding Boxes) markiert werden sollen und YOLO für das passende Erzeugen solcher Kästchen prädestiniert ist. Auch hier werden Einzelheiten in einem späteren Kapitel erläutert.

Die für das Programm verwendete Programmiersprache ist Python. Hier wird hauptsächlich das von Google entwickelte Framework "TensorFlow" verwendet, welches die einfache Entwicklung von Neuronalen Netzen ermöglicht.

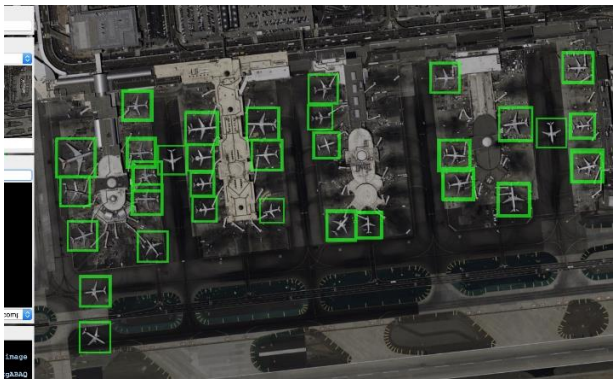


*Abbildung 1: Ausschnitt aus der Web-Anwendung zur Flugzeugerkennung*

# 2 Tensorflow und Google Cloud Platform

## 2.1 TensorFlow

Wie bereits anfänglich erwähnt, wurde für die Entwicklung des neuronalen Netzes die Plattform „TensorFlow“ verwendet. TensorFlow ist eine Ende-zu-Ende open Source Plattform für maschinelles Lernen, die von der Firma Google entwickelt wurde. Es hilft dabei, relativ schnell und einfach Modelle zum maschinellen Lernen zu entwickeln und zu trainieren. Die Plattform wird von Firmen wie Coca Cola, airbnb, intel und Google selbst zum Entwickeln und Trainieren von Neuronalen Netzen genutzt. Die Bibliotheken und Module sind in Python geschrieben, da sich diese Programmiersprache aufgrund ihrer Schnelligkeit und Einfachheit am besten für solche Anwendungen eignet. [4]

## 2.2 Google Cloud Platform

Google Cloud Platform ist ein Cloud Computing Service der Firma Google über den die Benutzer verschiedenste, rechenintensive Programme auf Rechen-Servern laufen lassen können, die von Google zur Verfügung gestellt werden. Ein Vorteil daran ist, dass man solche Prozesse wie z. B. das Training eines neuronalen Netzes, was unter Umständen mehrere Stunden bis Tage dauern kann, nicht auf seinem eigenen PC durchführen muss und somit keine eigene Rechenkapazität über einen längeren Zeitraum in Anspruch genommen wird. Zusätzlich bietet die Plattform auch alle weiteren Dienste, die eine Cloud mit sich bringt. Eine eingebaute Projektverwaltung hilft dabei, den Überblick über seine Projekte zu behalten. So genannte Buckets können dazu genutzt werden, die errechneten Ergebnisse abzuspeichern oder zu exportieren. Grundvoraussetzung für die Verwendung der Plattform ist ein gültiges Google-Konto sowie die Angabe einer Kreditkarte, um zu verifizieren, dass man auch wirklich eine natürliche Person und kein Computer ist. Das Planespotting Projekt wurde von Martin Görner für die Verwendung von Google Cloud Platfrom optimiert, weshalb der Dienst auch im Rahmen dieses Projektes genutzt wurde.

# 3 Convolutional Neural Networks

## 3.1 Convolutional Layer

Ein Convolutional Neural Network besteht in der Regel aus einer bis mehreren Convolutional Layers gefolgt von einer Pooling Layer. Ein solches Netzwerk aus künstlichen Neuronen wird vor allem in der Bild- und Audioerkennung eingesetzt. Bei dem Input handelt es sich deshalb meistens um 2 bzw. 3-dimensionale Mitritzen, die z. B. die Pixel und deren RGB-Farbwerte eines Bildes darstellen sollen. Convolutional Neural Networks arbeiten hierbei auf Basis einer diskreten Faltung. Bei diesem Verfahren wird über die Matrizen eine Faltungsmatrix oder auch Filterkernel bewegt, welcher dazu genutzt wird, um den Input für die nächste Schicht zu berechnen. Das Ergebnis dieser Faltung ist das innere Produkt des Filterkernels mit dem darunter liegenden Bildausschnitt. Ziel ist es, so genannte Feature-Maps zu erzeugen, die bestimmte Merkmale eines Bildausschnitts hervorheben sollen. Nach der Faltung wird der Input eines jeden Neurons durch eine Aktivierungsfunktion zu dem Output, der die relative Feuerfrequenz eines echten Neurons simulieren soll. Je tiefer ein Convolutional Network aufgebaut wird, also je mehr Schichten es besitzt, umso mehr steigen die Größe der rezeptiven Felder sowie die Komplexität der erkannten Features. [3]

## 3.2 Pooling Layer

Die zuvor durch eine oder mehrere Convolutional Layer berechneten Werte werden nun in eine Pooling Layer gegeben. Ziel einer Pooling Layer ist es, überflüssige Informationen aus kleineren Bildausschnitten herauszufiltern. Dabei gibt es verschiedene Verfahren. Wohl am meisten verbreitet ist das so genannte Max-Pooling. Hier wird aus einem 2 x 2 Quadrat nur das aktivste Neuron, also das mit dem höchsten Wert, herausgefiltert und an die nächste Schicht übergeben. Im Falle des Planespotting Projektes wurde ein anderer Ansatz verfolgt, nämlich das so genannte Average Pooling. Wie der Name bereits vermuten lässt, wird hier der Durchschnittswert aus einem quadratischen Feld berechnet und an die nächste Schicht übergeben. Dieser Ansatz einer Pooling Layer bietet mehrere Vorteile. Zum einen wird durch die Reduktion von irrelevanten Informationen die Durchführungsgeschwindigkeit erhöht. Außerdem wird dem so genannten „Overfitting“ (dt.: Überanpassung) entgegengewirkt, welches auftritt, wenn ein Neuronales Netz zu stark mit eintönigen Daten trainiert wird und sich somit auf die Erkennung eines ganz bestimmten Merkmals spezialisiert hat. Dadurch kann es z.

B. im Bereich der Handschrifterkennung nur noch die Handschrift einer bestimmten Person erkennen, aber nicht mehr die von anderen. [3]

## 3.3 Fully-Connected Layer

Abschließend wird an ein Convolutional Network häufig eine oder mehrere Fully-Connected Layer (dt. etwa: Voll vernetzte Schicht) angehängt. Eine Fully-Connected Layer dient in erster Linie der Klassifikation, wobei die Anzahl an Neuronen der letzten Schicht der Anzahl an unterschiedlichen Klassen entspricht. Wenn es nun also die beiden Klassen „Flugzeug“ und „kein Flugzeug“ gibt, besteht die letzte Schicht aus zwei Neuronen, eines für jede Klasse. Die Ausgabe der letzten Schicht wird nun noch häufig durch eine Softmax-Funktion in eine Wahrscheinlichkeitsverteilung überführt, die die Wahrscheinlichkeit zur Zugehörigkeit einer bestimmten Klasse festlegen soll. [3]

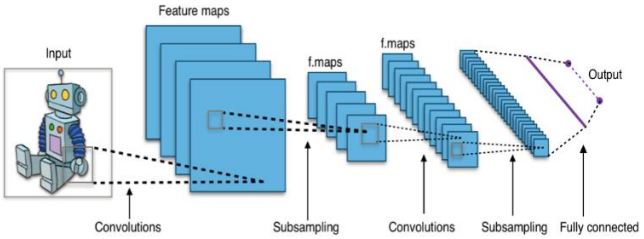



*Abbildung 2: Beispielhafter Aufbau eines Convolutional Neural Networks*

# 4 YOLO

YOLO (You Only Look Once) ist eine Technik, die in der Bilderkennung zur Echtzeit-Objekt-Erkennung verwendet wird. Dabei wird über das eingehende Bild ein N x N Netz gelegt, welches das Bild in N x N Zellen aufteilt. Jede einzelne dieser Zellen erzeugt für sich selbst eine bestimmte Anzahl von so genannten Bounding Boxen, deren Zweck es ist, ein beinhaltetes Objekt optimal einzuschließen. Zusätzlich wird für jede Zelle eine Klassenwahrscheinlichkeit ermittelt, welche angibt, mit welcher Wahrscheinlichkeit ein Objekt auf dem Bild zu einer bestimmten Klasse gehört. Das Konzept der Klassenbildung wurde bereits bei den Convolutional Networks eingeführt. Im Falle des Projekt Planespotting gibt es im Grunde nur zwei Klassen: „Flugzeug“ und „kein Flugzeug“. Da aber „kein Flugzeug“ gleichzusetzen ist mit „kein Objekt“ wird hier nur eine einzige Klasse verwendet.

Die Bounding Boxes besitzen zusätzlich noch 5 Werte, die für die weitere Verarbeitung essentiell sind. Diese Werte sind x, y, w, h und der Confidence Score. x und y beschreiben die relative Position der Box innerhalb einer Zelle. Sie können einen Wert zwischen -1 und +1 annehmen. w und h sind die relative Größe der Box im Bezug zum gesamten Bild. Sie können Werte im Intervall von 0 bis 1 annehmen, wobei 0 extrem klein und 1 so groß wie das gesamte Bild bedeutet. Der Confidence Score gibt an, wie sicher sich das Modell ist, dass eine Bounding Box ein Objekt enthält und wie es klassifiziert wurde. Die erzeugten Bounding Boxen werden mit den ermittelten Klassenwahrscheinlichkeiten verrechnet und ergeben somit die finalen Bounding Boxen, die im Idealfall ein Objekt wie z. B. ein Flugzeug mit passender Größe einschließen und dieses passend klassifizieren. [1] [5]

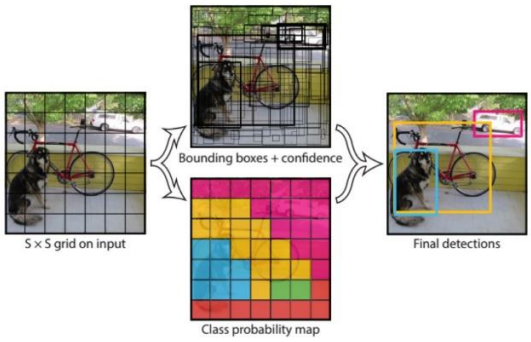



*Abbildung 3: Beispielanwendung für YOLO*

# 5 Squeezenet

Squeezenet ist eine Architektur, die zum Aufbau von Convolutional Networks benutz wird. Hierbei wird ein bestimmter Ansatz verfolgt, bei dem die eingehenden Daten zunächst zu einer 1x1 Convolutional Layer verdichtet (squeeze) werden. Sinn dieser Vorgehensweise ist es, die Anzahl an Gewichtungen im Netzwerk zu reduzieren. Anschließend werden die Ergebnisse der 1x1 Convolutional Layer parallel in einer weiteren 1x1 Schicht und einer 3x3 Schicht weiterverarbeitet (expand). Durch Konkadenation der daraus resultierenden Ergebnisse wird ein neuer „Daten-Würfel“ erzeugt, welcher nun in weiteren Schichten weiter verarbeitet werden kann. Diesen Vorgang kann man beliebig oft wiederholen. Planespotting verwendet ebenfalls diesen Ansatz. [1]

# 6 Aufbau und Funktion des Planespotting Modells

Nachdem in den vorhergehenden Kapiteln die grundlegende Theorie behandelt wurde, wird nun das eigentliche Projekt Planespotting genauer durchleuchtet. Das finale Modell besteht aus einem 17 Schichten Convolutional Neural Network, welches der Squeezenet Architektur nachempfunden wurde. Das bedeutet, dass die eingehenden Bilder zunächst in ein 1x1 Convolutional Layer überführt werden, gefolgt von der parallelen Verarbeitung in einer weiteren 1x1 und 3x3 Schicht. Am Ende des Colnvolutional Networks wird der YOLO Ansatz auf die ausgegebenen Daten angewandt. [1]

Wie bereits erwähnt, ist das Modell darauf ausgelegt, Flugzeuge auf großen Luftaufnahmen von Flughäfen zu erkennen und zu markieren. Diese Luftaufnahmen haben eine anfängliche Größe von 6000 x 8000 Pixeln. Weil das Modell selber nur Bilder mit einer Größe von 256 x 256 Pixeln direkt verarbeiten kann, werden die großen Luftaufnahmen zufällig in mehrere kleinere Bilder mit oben genannter Größe aufgeteilt. Dabei wird darauf geachtet, dass es in etwa genauso viele Bilder mit Flugzeugen gibt wie ohne. Zum Training des Modells kann man sowohl die großen Luftaufnahmen als auch die kleineren Bilder verwenden, wobei die großen Luftaufnahmen durch ein Skript vor dem eigentlichen Training in die kleineren Bilder aufgeteilt werden. Beide Trainingsdatensätze sind über die Google Cloud Plattform verfügbar. Die kleineren Bilder werden dann an das Squeezenet Netzwerk übergeben. Der daraus resultierende Output, welcher aus mehreren Datenschichten besteht, wird anschließend nach dem YOLO Ansatz durch ein 16 x 16 Netz in mehrere „Säulen“ unterteilt wie in Abbildung 4 dargestellt. [1]

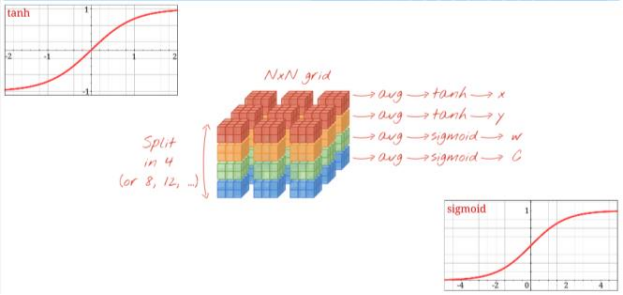



*Abbildung 4: Praktische Anwendung von YOLO auf die durch Squeezenet erzeugten Daten*

Jede dieser Säulen stellt dabei eine Zelle des Netzes dar. Was nun noch fehlt sind die Bounding Boxen. Planespotting erzeugt 2 Bounding Boxen pro Zelle, wodurch die einzelnen Datensäulen nochmal in 4 weitere Teile unterteilt werden (zwei Schichten pro Block bedeutet 2 Bounding Boxen). Zur Ermittlung der in Abschnitt 4 beschriebenen Werte der Bounding Boxen werden die einzelnen Datenblöcke, je nach gewünschtem Ergebnis, an eine tanh- oder sigmoid-Funktion übergeben. Ergebnis einer tanh-Funktion ist ein Wert zwischen -1 und +1, was für die Werte x und y benötigt wird. Die sigmoid-Funktion hingegen erzeugt Werte im Intervall von 0 bis 1, was den gewünschten Werten von w und h entspricht. [1]

Jede Zelle bekommt nun noch eine Klassenwahrscheinlichkeit zugeordnet. Wie bereits erwähnt, kennt das Modell von Planespotting im Grunde nur zwei Klassen: „Flugzeug“ und „kein Flugzeug“. Weil alle Objekte, die kein Flugzeug darstellen, ohnehin vom Modell ignoriert werden, bedeutet die Klasse „kein Flugzeug“ im Grunde dasselbe wie „kein Objekt“ und kann daher wegfallen. Somit erhält jede Zelle nur noch einen Wahrscheinlichkeitswert, der aussagt, mit welcher Wahrscheinlichkeit sich ein Flugzeug in dieser Zelle befindet.

Die durch das Modell ermittelten Werte für die Bounding Boxen werden nun mit den Klassenwahrscheinlichkeiten verrechnet. Ergebnis sind die finalen Boxen, die im Idealfall alle Flugzeuge auf dem eingegebenen Bild markieren, wie bereits in Abbildung 1 dargestellt. Zum Validieren der Ergebnisse kann die von Martin Görner entwickelte Web-Anwendung verwendet werden. Diese ist über eine REST-Schnittstelle mit der Google Cloud Plattform verbunden und ermöglicht es dadurch, schnell und einfach das Trainierte Modell über die Schnittstelle aufzurufen und zu verwenden. Zusätzlich kann über eine Google-Maps-Schnittstelle aus einer Liste einer von mehreren Flughäfen ausgewählt werden. Wichtig ist nur, dass man sich innerhalb der Anwendung vorher mit dem Google Konto registriert. Anschließend muss nur noch der Pfad zum fertig trainierten Modell angegeben werden und schon kann das Modell genutzt werden. [1]

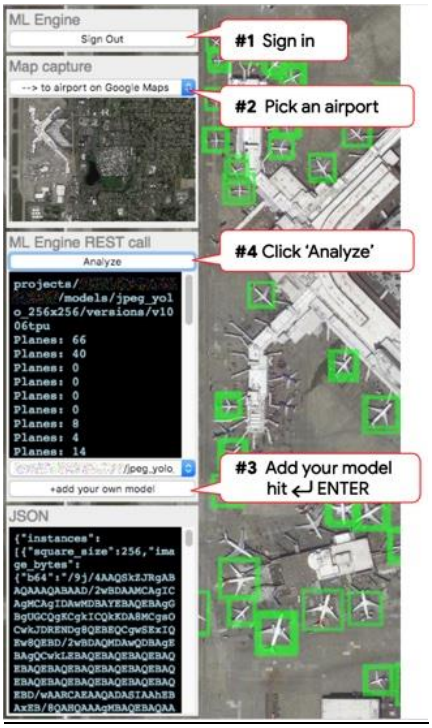



*Abbildung 5: Interface der Web-Anwendung*

# 7 Anleitung zum Trainieren mit Google Cloud Plattform

In diesem Kapitel folgt eine kurze Anleitung, mit der jeder Leser dieser Arbeit das Modell selbst trainieren und testen kann. Das Projekt selbst ist als git-Projekt unter der URL https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/tree/master/tensorflow-planespotting zum Download verfügbar. Hier befindet sich auch eine von Martin Görner selbst auf Englisch verfasste Anleitung. Dem oben genannten Projekt liegen zusätzliche Shell-Skripte bei. Diese und alle aufgeführten Kommandos benötigen zur Ausführung eine Linux-Shell. Das Skript zum automatischen Trainieren auf der Google Cloud Plattform heißt cloudrun_yolo.bash und befindet sich im Ordner „tensorflow-planespotting“. Dieses Skript trainiert automatisch

ein 17-Schichten Squeezenet/YOLO-Modell auf Google Cloud Plattform.

Vor dem eigentlichen Training müssen noch ein paar Vorbedingungen erfüllt werden:

- Man benötigt ein aktives Google-Konto, mit dem man sich unter https://console.cloud.google.com/ anmeldet
- In der Cloud Console muss man ein neues Projekt erstellen und eine Kreditkarte als Zahlungsmethode hinterlegen (das erste Jahr ist kostenlos, die Karte muss aber trotzdem angegeben werden).
- In der Konsole muss ein neuer "Bucket" angelegt werden. Als Speicher-Klasse wird "regional" angegeben und die gewünschte Region ausgewählt. Die Region (also der Server) muss aber über ML-Engine Ressourcen verfügen.
- Zusätzlich wird das Google Cloud SDK benötigt, dieses kann im Internet kostenlos heruntergeladen werden. Mit dem SDK kann die "gcloud" Kommandozeile verwendet werden.
- Nach Installation des SDK muss in diesem "*gcloud init*" ausgeführt und sich mit seinem Google Account angemeldet werden.

Nun muss man im cloudrun_yolo.bash Skript an der erforderlichen Stelle den Bucket-Namen, Projekt-Namen und die ausgewählte Region einfügen. Wurden die bisherigen Schritte ausgeführt, kann das Modell mit dem Befehl „*./cloudrun_yolo.bash*" im Verzeichnis „tensorflow-planespotting" ausgeführt werden. Das Training kann nun einige Stunden bis hin zu einem Tag dauern. Wurde das Modell fertig trainiert, kann auf der Google Cloud Plattform im Bereich „KI-Plattform", hier im Unterbereich „Modelle" ein neues Modell erstellt werden. Dazu auf „neues Modell" und anschließend auf „neue Version". Hier muss nur das trainierte Modell aus dem Bucket ausgewählt werden. Der Pfad dazu sieht in etwa so aus: *gs://<Dein-Bucket>/jobs/airplaneXXX/export/planespotting/1529794205/*. Wurde das Modell erstellt, kann es getestet werden.

Die Test-Anwendung wurde in HTML und JavaScript geschrieben und besitzt eine REST-Schnittstelle zur Google Coud Plattform. Sie ist im Verzeichnis „tensorflow-planespotting/webui" verfügbar. Um das Modell zu testen, muss zunächst (unter Linux) ein lokaler Web-Server gestartet werden. Dazu im Verzeichnis „webui" den Befehl „*python3 -m http.server --bind localhost 8000*" eingeben und ausführen. Die Web-Anwendung ist anschließend über einen Browser unter „*htttp://localhost:8000*" verfügbar. In der Anwendung registriert man sich nun mit seinem Google-Account, sucht sich einen Flughafen raus, fügt über den Button „add your own model" ein trainiertes Modell hinzu und klickt anschließend auf „Analyze", wie in Abbildung 5 dargestellt. Die Anwendung unterteilt nun das Bild in sich überlappende 256 x 256 Pixel große Bilder und übergibt diese dem Modell. Nach einiger Zeit werden nun die erkannten Flugzeuge mit grünen Boxen markiert.

# 8 Fazit

In dieser Studienarbeit wurde darauf eingegangen, wie genau Künstliche Intelligenz und Neuronale Netze funktionieren, wie sie aufgebaut sind und wie man sie, mit ein wenig Programmiererfahrung, ganz einfach selber trainieren kann. Es wurde zunächst die grundlegende Theorie erklärt und anschließend auf das eigentliche Projekt Planespotting Stellung bezogen.

Schlussendlich lässt sich über das Projekt sagen, dass es trotz anfänglicher Schwierigkeiten, das Modell zu laufen zu bringen, über Google Cloud Plattform recht einfach und intuitiv anzuwenden war. Die Tatsache, dass sowohl die Datensätze als auch die Rechen-Kapazität von Google zu Verfügung gestellt wurden, erleichterte die Entwicklung zusätzlich. Die angebotene Web-GUI ermöglichte es außerdem, spielerisch das fertig trainierte Modell zu testen und zu verwenden.

# Literaturverzeichnis

[1] Martin Görner, „github.com," [Online]. Available: https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd. [Zugriff am 06. Juli 2019].

[2] chrislb, „wikipedia.org," 28. April 2019. [Online]. Available: https://de.wikipedia.org/wiki/Künstliches_Neuron. [Zugriff am 06. Juli 2019].

[3] Dopexxx, „wikipedia.org," 20. Mai 2019. [Online]. Available: https://de.wikipedia.org/wiki/Convolutional_Neural_Network. [Zugriff am 06. Juli 2019].

[4] Google, „tensorflow.org," Google, [Online]. Available: https://www.tensorflow.org/. [Zugriff am 06 Juli 2019].

[5] M. Chablani, „towardsdatascience.com," 21. August 2017. [Online]. Available: https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006. [Zugriff am 07. Juli 2019].

# Abbildungsverzeichnis

# Erkennen von Emotionen mittels Supervised Learnings

Angewandtes maschinelles Lernen

Lucas Drechsel
Informatik
Hochschule für angewandte Wissenschaften
Hof, Germany
lucas.drechsel@hof-university.de

## Abstract

Im Rahmen der Studienarbeit „Erkennen von Emotionen mittels Supervised Learnings“ wurde sich mit der Thematik der Erkennung von Emotionen mittels Convolutional Neural Network und der praktischen Anwendung genauer auseinandergesetzt. Dabei wird ein neuronales Netz auf sieben Grundemotionen trainiert und anschließend Bilder von Gesichtern des Nutzers hinsichtlich der gezeigten Emotionen analysiert.

## 1 Hinleitung

Das Erkennen von Emotionen spielt nicht nur bei Unterhaltungen, sondern auch innerhalb zwischenmenschlicher Beziehungen eine enorm wichtige Rolle. Menschen haben im Allgemeinen die Fähigkeit, die Grundemotionen der Person gegenüber ohne Probleme zu erkennen und diese zu deuten. Auch im Tierreich erkennen beispielsweise Hunde die Emotionen ihres Herrchens. Wenn dieses ihm zum Beispiel einen harschen Blick zuwirft, reagiert der Vierbeiner meist mit dem sogenannten Hundeblick.

Nach einer langen Zeit der Recherche und Einarbeitung in die Thematik, bin ich auf die Arbeit von Sefik Ilkin Serengil aufmerksam geworden, welcher sich beruflich mit den Fragen der Kryptografie und maschinellem Lernen beschäftigt. In seiner Freizeit schreibt der Softwareentwickler einen Technik-Blog oder erstellt Kurse für Udemy.

In zwei Blogbeiträgen[1] beschreibt er das Vorgehen, mit Hilfe des Datensatzes der Kaggle-Challenge zur ICML 2013 „Representation Learning: Facial Expression Recognition Challenge“ ein Netz zu trainieren, um anschließend Emotionen von Bildern zu analysieren.

Darüber hinaus stütze ich mich auf den Artikel „Training a TensorFlow model to recognize emotions“[2] des Entwicklers Jose Flores, welcher den gleichen Kaggle-Datensatz als Grundlage seiner Arbeit verwendet.

Im Rahmen dieser Studienarbeit wird ein neuronales Netz mit Bildern von Gesichtern anhand der gezeigten Emotionen trainiert. Auf Basis des Rades der Emotionen nach Robert Plutchiks werden hierfür die Gefühle Angst, Wut, Ekel, Freue, Trauer, Überraschung und der neutrale Gesichtsausdruck analysiert.

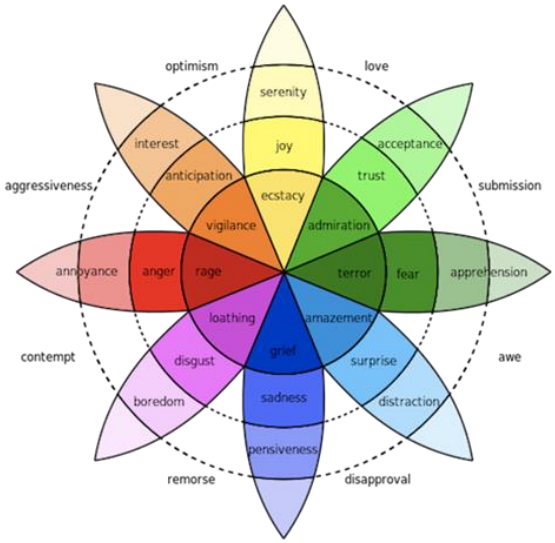



**Abbildung 1: Das Rad der Emotionen nach Robert Plutchiks[3]**

## 2 Vorbereitungen

Im ersten Schritt werden hunderte Bilder von Gesichtern mit den dargestellten Emotionen in Form von *labled Data* benötigt. Alternativ kann dabei auch eine Liste an prozentual gezeigten Gesichtsausdrücken angegeben werden (zum Beispiel: traurig: 25%, wütend: 57%, ängstlich: 18%).

Diese Daten können selbst erhoben werden, indem man zum Beispiel einfach mittels schmaler, selbstgeschriebener *Progressive Web Applikation* seinen Freunden und Bekannten die Möglichkeit bereitstellt, Bilder mit entsprechenden Emotionen hochzuladen und das Gesicht passend textuell zu beschreiben. Dabei kann gleich der Bildausschnitt mittels einer Schablone festgelegt werden.

Da in dieser Vorlesung der Fokus nicht auf App-Programmierung, sondern auf angewandtem maschinellem Lernen liegt, wird im Rahmen der Studienarbeit auf online Repositorys zurückgegriffen. In denen lassen sich verschiedenste Datensätze, angefangen bei Sammlungen von Katzenbildern bis

[1] http://sefiks.com/2018/01/01/facial-expression-recognition-with-keras/ und https://sefiks.com/2018/01/10/real-time-facial-expression-recognition-on-streaming-data/

[2] https://medium.com/@jsflo.dev/training-a-tensorflow-model-to-recognize-emotions-a20c3bcd6468

[3] https://de.wikipedia.org/wiki/Robert_Plutchik

hin zu weltweiten Flugdaten, abfragen und nach kurzer Registrierung herunterladen.

## 2.1 Facial Expression Recognition Challenge

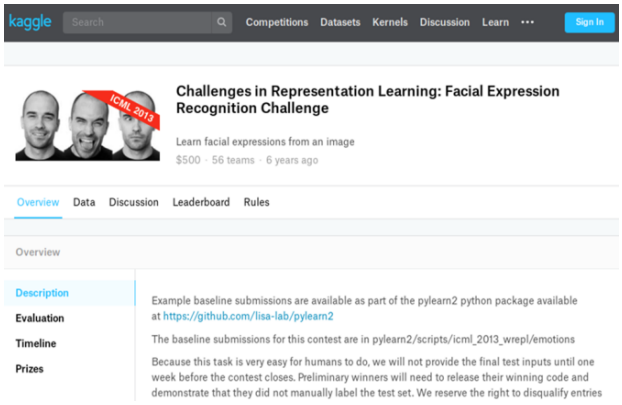



**Abbildung 2: Startseite zur Kaggle Facial Expression Challenge**

Dabei wird auf einen Datensatz eines Wettbewerbes zur *International Conference on Machine Learning* (ICML 2013) der Plattform Kaggle.com zurückgegriffen[4]. Die Daten bestehen aus 48x48 Pixel Graustufenbildern von Gesichtern, welche automatisch zentriert wurden, sodass in etwa jedes Gesicht die gleiche Fläche in jedem Bild einnimmt. Das Trainingsset besteht aus 28709 Beispielbildern - das public und private Testset aus jeweils 3589 Beispielbildern.

## 2.2 Erster Kontakt mit Rohdaten

Wie in der beigefügten Readme-Datei beschrieben, werden zunächst die Rohdaten analysiert. Dabei wird die CSV-Datei mit dem Datensatz auf der Kaggle-Website des Wettbewerbes nach kurzer Anmeldung und Validierung des Accounts via SMS heruntergeladen und als DataFrame eingelesen. Der Befehl info() zeigt alle relevanten Informationen der Datei:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35887 entries, 0 to 35886
Data columns (total 3 columns):
emotion    35887 non-null int64
pixels     35887 non-null object
Usage      35887 non-null object
dtypes: int64(1), object(2)
memory usage: 841.2+ KB
```

**Abbildung 3: Informationen zum heruntergeladenen Datensatz**

Dabei kann man erkennen, dass der Datensatz eine Spalte mit der entsprechenden Emotion (einer Zahl zwischen 0 und 6), eine Spalte mit den Pixeldaten des Bildes als Ganzzahlarray und eine Spalte mit dem Usage, also ob es sich hierbei um Trainingsdaten, Public- oder Private-Test handelt, besitzt.

Um ein Bild anzuzeigen, wird das jeweilige Pixelarray geladen, jeder Eintrag durch 255 geteilt und über die Funktion reshape() als Matplot ausgegeben.

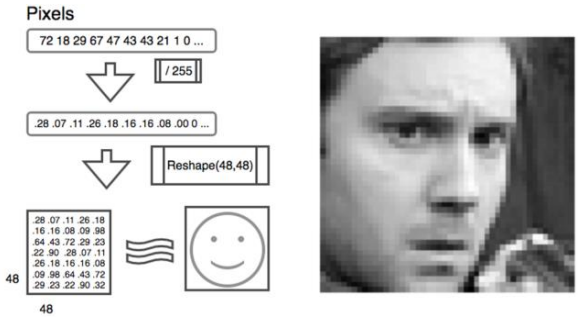



**Abbildung 4: Ablauf zur Darstellung eines Bildes aus Datensatz[5]**

Im beigefügten *Colab Notebook* kann mittels eines Sliders der Datensatz Bild für Bild ausgegeben und betrachtet werden.

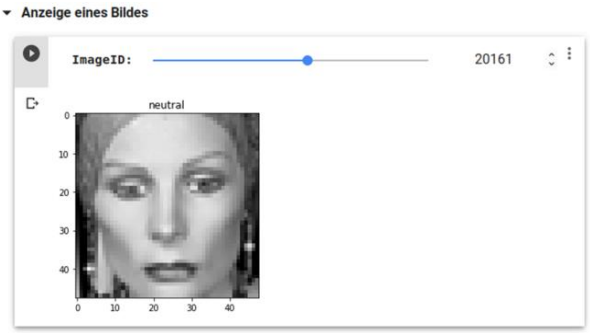



**Abbildung 5: Anzeige eines Bildes durch den User mittels Slider-Widgets**

## 2.3 Eigenschaften des Bildmaterials

Das Bildmaterial des Datensatzes ist von sehr unterschiedlicher Qualität. Neben unzähligen Bildern mit Texten und Mustern im Gesicht (z.B. Bild IDs 15361, 5117), haben einige Personen teilweise ihre Hände vor dem Gesicht (z.B. ID 35879) oder sind einfach komplett unscharf (z.B. ID 19286). Des Weiteren sind die Gesichter auf den Bildern teilweise zentriert oder komplett abgeschnitten, sodass die Mund oder Augenpartien fehlen (z.B. ID 18012).
Bei einigen wird das Gesicht schräg oder nur seitlich gezeigt (z.B. IDs 35824, 21875).
Zudem ist die Altersgruppe sehr unterschiedlich. Neben älteren Personen wurden im CSV-Datensatz auch Bilder von Babys uns Kindern gesammelt. Ebenso befinden sich Bilder von Personen mit hellen und dunklen Hauttypen in der Sammlung, sowie von Personen mit und ohne Brille.

[4] https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

[5] Jose Flores. 2018. https://medium.com/@jsflo.dev/training-a-tensorflow-model-to-recognize-emotions-a20c3bcd6468

## 2.4 Einschränkungen

Bei der Durchsicht des Materials und Lesen der Beschreibung, stößt man auch auf die ein oder anderen Probleme oder Einschränkungen:

2.4.1 *Anzahl der Emotionen.* In der Beschreibung des Datensatzes ist zu lesen, dass nur 7 Gesichtsausdrücke erfasst sind (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). Darüber hinaus gibt es allerdings weitere Emotionen, welche im verwendeten Datensatz nicht erfasst wurden. Sollen weitere Emotionen gelernt werden, muss ein anderer Datensatz gefunden oder die Daten selbst gesammelt werden

*2.4.2 Zentrierung der Gesichter.* Der Beschreibung des Datensatzes ist zu entnehmen, dass die Gesichter automatisch zentriert wurden. Um später diese Zentrierung zu replizieren, muss analysiert werden, wie das Bild zentriert wurde. Eine nachträgliche Beschneidung im Preprocessing beispielsweise mittels OpenCV ist aufgrund der Knappen Ränder der Gesichter kaum mehr möglich.

*2.4.3 Auflösung der Bilder.* Die Bilddateien sind mit 48 mal 48 Pixeln relativ klein. Bei der Skalierung hochauflösender Bilder gehen leicht feine Details verloren. Gerade diese Details entscheiden bei Emotionen häufig, ob zum Beispiel eine Person angewidert oder wütend schaut. Das ist vermutlich das größte Problem. Hochauflösende Bilder würden zwar diese feinen Details liefern, jedoch würde sich der Lernprozess als deutlich zeit- und rechenintensiver gestalten.

# 3 Netz

## 3.1 Architektur

In dieser Studienarbeit wird sich mit dem überwachten Lernen auseinandergesetzt, was einem Teilgebiet des maschinellen Lernens entspricht. Dabei wird versucht, durch *labled Data* das System anzulernen und Gesetzmäßigkeiten nachzubilden.
Bei der Architektur handelt es sich um ein *Convolutional Neural Network* (CNN), welches normalerweise bei der maschinellen Verarbeitung von Bild- oder Audiodaten Verwendung findet, und wird in unserem Fall mit *Keras* unter Verwendung von *TensorFlow* im Backend gebaut.
Keras ist eine High-Level-API für neuronale Netzwerke und ermöglicht eine einfache Modellierung unseres Models.
Nach einigen Anpassungen an der Architektur, die im folgenden Kapitel genauer erörtert werden, ergibt die Abfrage des Befehls model.summary() folgende Ausgabe:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_88 (Conv2D) | (None, 46, 46, 32) | 320 |
| conv2d_89 (Conv2D) | (None, 46, 46, 32) | 9248 |
| max_pooling2d_17 (MaxPooling | (None, 23, 23, 32) | 0 |
| conv2d_90 (Conv2D) | (None, 21, 21, 64) | 18496 |
| conv2d_91 (Conv2D) | (None, 19, 19, 64) | 36928 |
| average_pooling2d_33 (Averag | (None, 9, 9, 64) | 0 |
| conv2d_92 (Conv2D) | (None, 7, 7, 128) | 73856 |
| conv2d_93 (Conv2D) | (None, 5, 5, 128) | 147584 |
| average_pooling2d_34 (Averag | (None, 2, 2, 128) | 0 |
| global_average_pooling2d_16 | (None, 128) | 0 |
| dense_49 (Dense) | (None, 64) | 8256 |
| dropout_43 (Dropout) | (None, 64) | 0 |
| dense_50 (Dense) | (None, 64) | 4160 |
| dropout_44 (Dropout) | (None, 64) | 0 |
| dense_51 (Dense) | (None, 7) | 455 |

Total params: 299,303
Trainable params: 299,303
Non-trainable params: 0

**Abbildung 6: Der Befehl model.summary() liefert iterativ eine Übersicht der Architektur**

Man erkennt hierbei den grundlegenden Aufbau eines CCNs:
Dabei wechseln sich Faltungsebenen, die sogenannten *Convolutional Layer*, mit *Pooling Layern* mehrfach ab. Anschließend wird durch *Fully Conected Layers* die Architektur abgeschlossen.
Zusätzlich soll beispielsweise mit Dropout Layern Overfitting verhindert werden. Hierbei werden in jedem Schritt des Trainings zufällig ausgewählte Neutronen aus dem Netz gelöscht.

## 3.2 Anpassungen

Die Basisarchitektur ist an dem Entwurf von Serengil angelehnt. Dabei wurden zusätzlich *Convolution Layer* ergänzt, die Dropout-Rate erhöht und *Flatten* mit *GlobalAveragePooling2D* getauscht.
Zusätzlich wurde in Hinblick auf Data Augmentation der *ImageDataGenerator* um eine zufällige Rotation um maximal jeweils 27° und ein zufällig horizontales Drehen des Bildes erweitert. Daneben wurde die Variable *steps_per_epoch*, welche im ursprünglichen Quellcode gleich der *batch_size* gesetzt wurde, separiert und einer eigenen Variablen zugewiesen.
Auch wurden die Epochen von fünf auf aktuell 200 gesetzt, welche aber durchaus erweiterbar sind. Das Training des Models wird kurz vorher durch den hinzugefügten Callback *EarlyStopping* beendet. Daneben wurden die Callbackfunktionen *ModelCheckpoint*, also ein automatisches Sichern der optimalen Gewichte, *TensorBoard*, für alle relevanten Informationen bezüglich Loss und Accurancy von Training und Validation und *ReduceLROnPlateau*, was die Learningrate bei Erreichen von Plateaus anpasst.

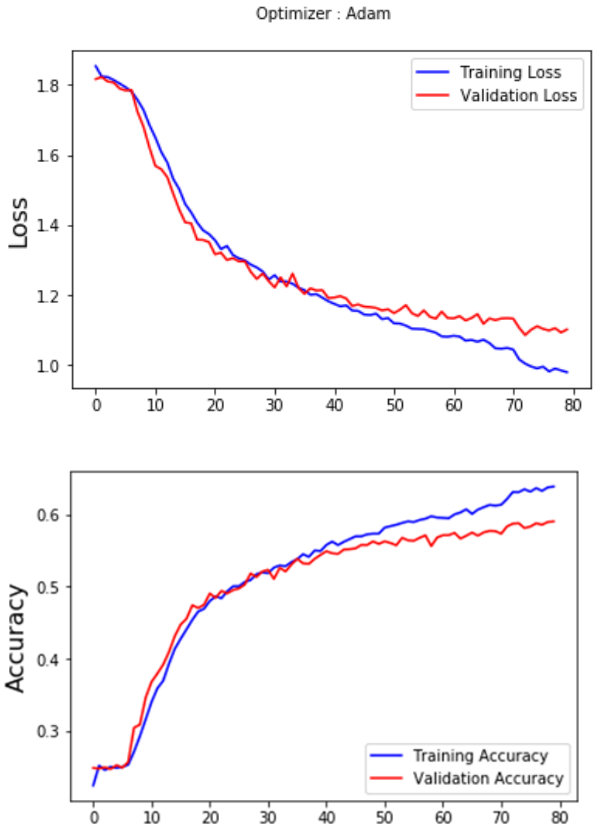



**Abbildung 7: Plot zur Accuracy und Loss**

Zusätzlich wird das abgeschlossene Training mit einer eigenen separaten Testmenge zum Schluss evaluiert und Loss und Accurancy dieses Testes ausgegeben.

```
9s 93ms/step - loss: 0.9847 - acc: 0.6372 - val_loss: 1.0928 - val_acc: 0.5894
9s 93ms/step - loss: 0.9790 - acc: 0.6390 - val_loss: 1.1014 - val_acc: 0.5901
Test 2 loss: 1.0541986631928595
Test 2 accuracy: 60.15603232181116
```

**Abbildung 8: 60% Accuracy mit der unabhängigen Testmenge**

Nach erfolgreichem Training kann zu Testzwecken direkt ein Bild analysiert werden:

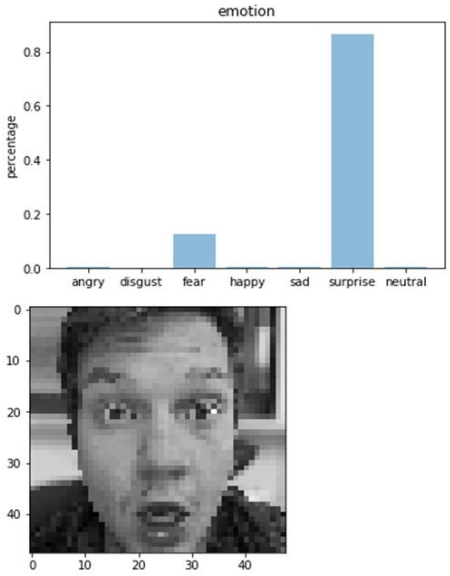



**Abbildung 9: Nach erfolgreichem Training: Testanalyse eines Bildes**

## 4 Anwendung

Nach dem das Netz nun trainiert wurde, können eigene Bilder im Rahmen der Anwendung analysiert werden. Dabei muss bei der Anwendung kein Fit-Prozess des Models vorangegangen sein, denn Ziel der Studienarbeit war eine unabhängige und modulare Ausführung einzelner Elemente.
Zu Beginn der Anwendung kann das Model und die Gewichte einfach geladen werden, die durch die Verbindung zu Google Drive jederzeit aufrufbar sind. Nach kurzer Authentifizierung und aller Vorbereitungen der Readme-Datei können die Rückgabewerte der Vorhersage des Models verarbeitet werden.

### 4.1 Grundfunktion

Bei der Ausführung ist der Kerngedanke eines jeden Anwendungsfalles identisch: das Bild wird mittels OpenCV hinsichtlich eines Gesichtes untersucht, der gefundene Ausschnitt zu Graustufen konvertiert, zu 48 mal 48 Pixeln verkleinert, als Array gespeichert, zu Werten zwischen 0 und 1 normalisiert und anschließend das Ergebnis der Prediction zur weiteren Verarbeitung gespeichert.
Danach wird das maximal indexierte Array bestimmt, also der prozentual höchste Wert der bestimmten Emotion, welches neben dem durch OpenCV erkannte Gesicht in Form des Emotionsnamens angezeigt wird.
Jeweils rechts oder links, je nach Position des Gesichtes, werden alle zurückgelieferten Prozentwerte und Namen der Gefühle aufgeschlüsselt. Das stärkste wird zusätzlich hell hervorgehoben.
Zwar ist die Grundfunktion identisch, doch wird durch den gezielten modularen Aufbau und den einzelnen Anpassungen bei Videoverarbeitung, Aufnahme von Einzelbildern oder Bilderupload der Code an die jeweiligen Anforderungen angepasst.

## 4.1 Bilderupload

Der Bilderupload ermöglicht das Hochladen und die anschließende Stapelverarbeitung von Bildern des lokalen Computers. Nach Auswahl der Bilder und dem Upload, dessen Status im Output der Konsole erscheint, wird Bild für Bild das Gesicht ermittelt und die weiteren Schritte zur Bestimmung des Gesichtsausdruckes, wie oben erklärt, ausgeführt.


**Abbildung 10: Nach erfolgreichem Upload wird der Name der hochgeladenen Datei ausgegeben**


**Abbildung 11: Nach erfolgreicher Ausführung erscheinen alle analysierten Bilder im Output der Konsole**

## 4.2 Bilder über Webcam

OpenCV erlaubt ohne weitere Anpassungen die Verarbeitung von Videostreams verschiedener Videoinputs. Da als Grundlage *Google Colaboratory* gewählt wurde, wird diese Funktion dort leider nicht unterstützt.
Aus diesem Grund wurde der Basis-Code der Bildaufnahme so angepasst, dass automatisch alle Sekunde ein Bild von der durch den Nutzer im Browser gewählten Videoquelle aufgenommen, gespeichert und verarbeitet wird, um den Effekt eines Streams nachzuempfinden. In den Browsern sollte der Haken bei „Immer Zugriff auf Videogerät erlauben“ gesetzt werden, da sonst ohne Bestätigung des Benutzers kein Bild aufgenommen werden kann.

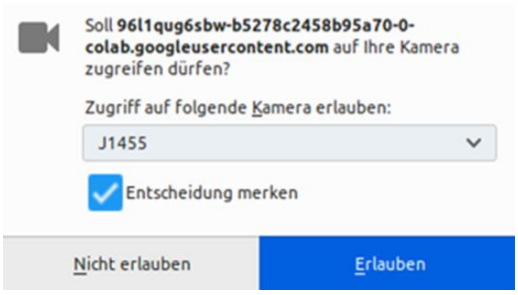



**Abbildung 12: Bestätigt der Nutzer den dauerhaften Zugriff, wird jede Sekunde ein Bild aufgenommen und hochgeladen**

Da jedoch die interne Webcam meines Notebooks kontrastarme und meist über- oder unterbelichtete Fotos liefert, welche von OpenCV nur mit Mühe verarbeitet werden konnten, musste eine alternative Bild-Quelle gesucht werden.
Zunächst wurde mit einer Android-App namens „IP Webcam“ das Bild der Handykamera auf einen über die App bereitgestellten lokalen Webserver geladen, was danach durch einen Rechner im Netzwerk über FTP auf einen passwortgeschützten Webhost hochgeladen wurde und in der *Colab* Konsole durch einen wget-Aufruf für die Verarbeitung gespeichert wurde. Das Problem hierbei war jedoch die große Latenz, welche zwischen Aufnahme des Bildes und Ausgabe der Konsole durch den mehrfachen Transport entstand.
Als Favorit stellte sich eine simple Lösung heraus. Dabei wurde eine kleine Action-Kamera mit Weitwinkelobjektiv via USB an den Laptop angeschlossen und die Kamera auf den Webcam-Modus gestellt. Dadurch erkennt das Betriebssystem die Kamera als zusätzliches Bildgerät, welches in den Einstellungen des Browsers ohne Probleme selektiert werden kann. Das Bildmaterial der nun angeschlossen Kamera liefert ein kontrastreicheres und nun besser belichtetes Bild für die Weiterverarbeitung mit OpenCV.

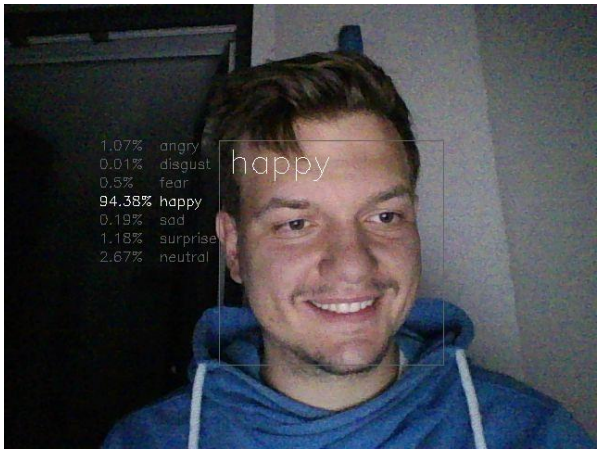



**Abbildung 13: Ausgabe eines Bildes: der Hauptgesichtsausdruck wird dabei groß vermerkt.**

## 4.3 Videoverarbeitung

Um noch einen Schritt weiter zu gehen, wird im Folgenden bewegtes Bildmaterial in Form von Videodateien analysiert.
Dabei wird das Video über die Funktion cv2.VideoCapture(inputvideo) geladen und anschließend Frame für Frame über cap.read() für die Bildanalyse zwischengespeichert. Anschließend wird der aktuelle Frame, wie oben zuvor beschrieben, hinsichtlich des jeweiligen Gesichtsausdruckes analysiert und für die Ausgabe gespeichert.
Bei den Parametern kann der Benutzer die Input-File und Input-FPS wählen. Zusätzlich wird die Möglichkeit bereitgestellt, den Motion-Flag zu setzten. Hierbei bleibt die Beschriftung des jeweiligen Frames entweder statisch in der oberen linken Ecke, oder bewegt sich dynamisch mit dem Gesicht im aktuellen Einzelbild.

Nachdem alle Frames verarbeitet wurden, wird das Videofile gespeichert und kann im Arbeitsverzeichnis der Runtime durch den Benutzer heruntergeladen und auf dem lokalen Rechner betrachtet werden.

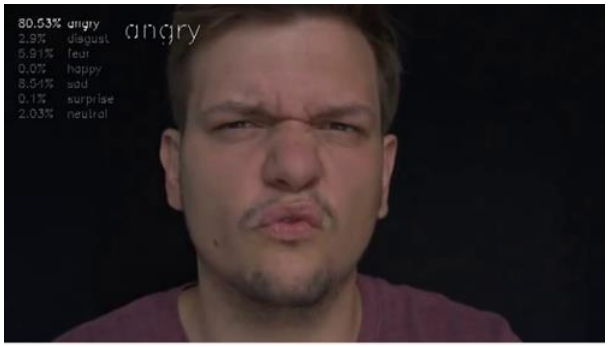



**Abbildung 14: Im Frame des Videos: wütender Gesichtsausdruck**


**Abbildung 15: Im Frame des Videos: überraschter Gesichtsausdruck**

## 5 Abschließende Gedanken

Die Analyse von Emotionen ist ein sehr spannendes und umfangreiches Thema. Schon bei der Sichtung der Literatur war ich von den technischen Möglichkeiten beeindruckt und konnte mir nicht wirklich vorstellen, selbst eigene Bilder durch den angepassten Quellcode zu analysieren.

Mein persönliches Highlight war der Moment, an dem ich genau dies machen konnte: eigene Bilder hochladen, analysieren lassen und das Ergebnis anzeigen lassen. Das spornte mich erst recht an, weitere Anwendungsfälle, wie zum Beispiel die Verarbeitung von Videomaterial, abzudecken.

Mich faszinieren die technischen Möglichkeiten, welche sich durch die Anwendung von neuronalen Netzen ergeben. Als praktischen Anwendungsfall könnte ich mir eine leicht abgewandelte Art dieser Arbeit bei der tiefgreifenden Entwicklungsstörung Autismus vorstellen.

Autisten fällt es schwer, die eigenen und die Gefühle anderer Menschen richtig einzuordnen.[6] Eine kleine versteckte Kamera an der Brille könnte dabei unauffällig die Emotionen der Person gegenüber im Hintergrund festhalten. Die Aufzeichnung kann dann durch ein trainiertes Netz analysiert und die Ergebnisse durch Vibrationen oder durch LEDs an den Träger weitergegeben werden. Wir können gespannt sein, was die Zukunft mit sich bringt.

[6] Autismus Westerwald-Mittelrhein e.V. Sozialverhalten - Leben in der eigenen Welt. https://www.autismus-wemi.de/erkennungszeichen/articles/sozialverhalten.html

# Entwicklung eines Personenidentifikationssystems

Prototypische Realisierung unter Zuhilfenahme von Gesichtsmerkmalen

Marco Schiener
Fakultät Informatik
Hochschule Hof
Hof, Bayern, Deutschland
marco.schiener@hof-university.de

## ZUSAMMENFASSUNG

Die folgende Arbeit soll den grundlegenden Aufbau und Ideen zur Realisierung eines Personenidentifikationssystems, welches Gesichtsmerkmale verarbeiten und verschiedenen Identitäten zuordnen kann, näher erläutern. Technisch wird sich hierzu den Mitteln aus dem maschinellen Sehen, einem Teilgebiet des maschinellen Lernens, bedient. In der vorliegenden Arbeit werden die Datenvorverarbeitung von Bildmaterial zur anschließenden Personenidentifikation sowie zwei Ansätze zur Realisierung eines neuronalen Netztes zur Umsetzung dergleichen diskutiert.

Es konnte gezeigt werden, dass sich die praktische Realisierung eines Systems zur Personenidentifikation auf Basis konventioneller Methoden deutliche Nachteile mit sich bringt. Als konventionell werden Ansätze bezeichnet, welche ein Netz mit Beispieldaten trainieren und auf die Merkmalsextraktion eines allgemein klassifizierenden neuronalen Netzes zurückgreifen, um anschließend eine Feinabstimmung des neuronalen Netzes durchführen zu können. Der hierbei verwendete Binärklassifizierer konnte trotz akribischer Abstimmung von Metaparametern des neuronalen Netzes keine zufriedenstellenden Ergebnisse in der Klassifizierungsgenauigkeit liefern. Da sich die Anzahl der Trainingsbeispiele in praktischen Anwendungen in Grenzen halten sollte oder dies auch die Gegebenheiten so verlangen, wurde aus diesem Grunde mit nur 180 Beispielaufnahmen eines Gesichts in verschiedenen Belichtungsverhältnissen und Räumlichkeiten sowie 352 Gegenbeispielen trainiert, bei einem Split von Trainings- zu Validierungsdaten von 80 zu 20. Bereits nach der ersten Trainingsepoche wurde eine Validierungsgenauigkeit von 100 Prozent erreicht. Gründe hierfür können sein, dass sich die Gesichter aufgrund der optischen Normierung nur bedingt voneinander unterscheiden und sich somit in weiterführenden Trainingsepochen kaum neue Merkmale erlernen lassen oder auch nur sehr wenig (Gegen-) Beispiele zur erlernenden Person vorliegen. Wurden die Gegenbeispiele erhöht, so sank die Erkennungsrate der zu erlernenden Person rapide ab. Die Fehler bei der Nichterkennung der erlernten Person hielten sich bei neuem Datenmaterial in Grenzen, während die fehlerhafte Klassifikation von einer anderen Identitäten trotz der hohen Korrektklassifizierungsrate der Validierungsdaten sehr hoch lag. Auch der Einsatz von Data Augmentation konnte an dieser Stelle nur bedingt helfen und verschlechterte bei starker Augmentierung das Ergebnis, was an den sehr ähnlichen, zugeschnittenen und vorverarbeiteten Bildausschnitten liegt, welche die zu erlernenden Merkmale in komprimierter Form enthalten. Aggressives Data Augmenting führte zu zu starker Generalisierung der Trainingsbeispiele.

Aus den besagten Problemen heraus wurde die auf Gesichter trainierte Faltungsbasis *VGG-Face* verwendet, welche als Convolutional Autoencoder genutzt wurde. Hierbei wurde der Klassifizierer des neuronalen Netzes verworfen und stattdessen mit der Vektorrepräsentation des vorhergehenden Layern gearbeitet. Die euklidschen bzw. Cosinus-Abstände zwischen verschiedenen Vektorrepräsentationen eines Eingabebildes waren viel zuverlässiger in der Bestimmung von Identitäten als der konventionelle Ansatz. Es wurden durch den Ansatz des Convolutional-Autoencoders auch weitere Probleme, wie z.B. die Anzahl der Trainingsdaten sowie Schwankungen in der Belichtung gelöst bzw. verbessert. Zudem ergaben sich aus diesem Ansatz heraus auch komplett neue Anwendungsfälle, wie beispielsweise das Clustering von Identitäten.

## SCHLAGWORTE

Angewandtes maschinelles Lernen, Künstliche Intelligenz, Merkmalsextraktion, Personenidentifikation

## 1 Einleitung

Im Folgenden wird geklärt, welche Beweggründe der vorliegenden Arbeit zugrunde liegen. Weiterhin wird das Umfeld, im welchem diese Arbeit angefertigt wurde, näher erläutert.

### 1.1 Motivation

Wie aus der Publikation "Gesichter als Stimuli im Säuglingsalter" hervorgeht, ist das Erkennen von Gesichtern eine bereits bei Säuglingen ausgeprägte Fähigkeit: *"Als einen der wichtigsten visuellen Reize konnte die bisherige Forschung [..] das menschliche Gesicht ausmachen. Nur wenige Stunden alte Säuglinge können beispielsweise über die Stimme ihrer Mutter, die sie bereits aus dem Mutterleib kennen, ihr Gesicht enkodieren und vom Gesicht einer anderen Frau unterscheiden."* ( [1] )

Wenn eine solche menschliche Fähigkeit über ein neuronales Netz mit Spezialisierung auf maschinelles Sehen abgebildet wird, entsteht ein sehr mächtiges Werkzeug, da typisch menschliche Fähigkeiten mit automatisierter Verarbeitung kombiniert werden können. Bei der richtigen Umsetzung können die Vorteile beider Seiten zum Tragen kommen: Es können typische Aufgabenstellung in der Personenidentifikation, für welche bisher der Mensch, aufgrund seiner intrinsisch im Menschsein vorhandenen Fähigkeiten, eingesetzt wurde, durch Maschinen und somit mit Zeit- und auch Kostenersparnis nach der Ammortisation gelöst werden.

Der technische Einsatz von neuronalen Netzen, welche Personen anhand ihres Gesichts erkennen und zuordnen können, ist sehr vielseitig. Anwendungsfälle sind beispielsweise die Autorisierung zur Nutzung von Geräten und Zugängen, das Clustering von ähnlichen bzw. gleichen Personen oder auch das automatisierte Filtern von Bild- und Videomaterial nach Identitäten, z.B. im Rahmen der Strafverfolgung oder auch zur besseren Strukturierung von privaten Bildgalerien nach Personen. Jüngste Gesetzgebungen, wie die Datenschutzgrundverordnung der Europäischen Union, schreiben vor, dass personenbezogene Daten nur durch Zustimmung der betroffenen Person veröffentlicht werden dürfen (vgl. [2]). Denkbar sind deshalb auch Systeme, welche die Rechte von Einzelpersonen schützen, indem nach unrechtmäßig veröffentlichten Bildern zu einer Person im öffentlichen Internet gesucht wird. Die Bilder können anschließend über ein solches System evaluiert werden. Es gilt jedoch anzumerken, dass ein derartiges System auch selbst im weiterführenden Sinne personenbezogene Daten verarbeitet, weshalb auch hier die DSGVO zur Anwendung kommt und gesetzliche Regelungen beachtet werden müssen.

Es sollte auch nicht vernachlässigt werden, dass solche technologischen Entwicklungen auch Möglichkeiten bieten, diese entgegen der Persönlichkeitsrechte einzusetzen. Die autoritäre Regierung der Volksrepublik China beispielsweise hat das weltweit umfassendste System von Massenüberwachung seiner Bürger etabliert (vgl. [3]). Ein Personenidentifikationssystem könnte somit als technisches Werkzeug zur Überwachung von Personen im öffentlichen Raum missbraucht werden. Ein derartiges System kommt auch schon in Teilen Chinas zur Anwendung. Die FAZ schreibt hierzu in einem Onlinebeitrag, dass in China immer mehr Städte mit Systemen ausgestattet werden, welche Bilder per Gesichtserkennung auswerten (vgl. [4]). Einer Schätzung gemäß des Analyse-Unternehmens IHS Markit heiße es, dass die Zahl der Überwachungskameras von derzeit 170 Millionen bis 2020 auf über 400 Millionen steigen wird (vgl. [4]).

Aufgrund der besagten, vielseitigen Anwendungsfälle und aus den aktuellen Anlässen heraus stellt sich die Frage, welche technischen Konzepte Personenidentifikationssystemen zugrunde liegen.

## 1.2 Projektumfeld

Im Rahmen der Vorlesung "Angewandtes maschinelles Lernen" lernen die Studierenden im 6. Semester der Fakultät Informatik an der Hochschule Hof die theoretischen Hintergründe und grundlegenden Konzepte von maschinellem Lernen näher kennen. Dozent der Vorlesung ist Herr Prof. Dr. Sebastian Leuoth, welcher zugleich auch als Betreuer der Studienarbeiten fungiert.

Im Seminar werden die verschiedenen Typen des maschinellen Lernens näher untersucht, wie zum Beispiel überwachtes Lernen, unüberwachtes Lernen oder auch bestärkendes Lernen. Auch vorgelagerte Prozesse, wie beispielsweise die Datenaufbereitung zur anschließenden maschinellen Verarbeitung werden von den Studierenden näher beleuchtet. Ziel der Vorlesung ist die Einführung in das Gebiet des maschinellen Lernens. Nach der theoretischen Erläuterung ausgewählter Techniken werden diese praktisch evaluiert. (vgl. [5])

Begleitend zum Seminar wurde diese Studienarbeit erstellt, um die erlernten Prinzipien im praktischen Rahmen anwenden und weiter verinnerlichen zu können. Sie knüpft an den Erkenntnissen der Vorlesung an und dient als wichtiger Erfahrungswert in der Konzeption und dem Verständnis von neuronalen Netzen. Aufgrund dessen wurde zu dieser Arbeit ein Jupyter-Notebook auf Basis von Google Colaboratory [1] erstellt, welches die unter *2 Aufgabenstellung* genannten Ziele verfolgt und dazu dient, diese praktisch zu evaluieren.

# 2 Aufgabenstellung

Die konkrete Aufgabenstellung konnte von den Studierenden selbst frei gewählt werden. Im weiteren Sinne muss die zugrundeliegende Thematik am maschinellen Lernen anknüpfen, um praktischen Erfahrungswert auf dem Fachbereich bieten zu können. Nachfolgend werden die konkret gewählte Aufgabenstellung und die zugrundeliegenden Ziele, welche mit dem Projekt verfolgt wurden, näher erläutert.

Im Allgemeinen soll im Rahmen der Studienarbeit mithilfe von maschinellem Sehen unter Anwendung eines neuronalen Netzes prototypisch evaluiert werden, wie Identitäten von Personen klassifiziert werden können. Dem Erlernen und der anschließenden Bestimmung von Identitäten liegt entsprechendes Bildmaterial zugrunde, welches vom neuronalen Netz verarbeitet werden kann. Dieser Prozess geschieht unter Zuhilfenahme von Gesichtsmerkmalen. Der Evaluationsprozess soll in zwei Teilbereiche gegliedert werden. Zum soll entschieden werden können, ob und wo sich Gesichter im zugrundeliegenden Bildmaterial befinden. In einem zweiten Schritt soll die konkrete Identität, welche sich von anderen abgrenzt, erlernt werden, um das vom neuronalen Netz gespeicherte Wissen reproduzierend auf neues Bildmaterial anwenden zu können.

[1] Weiterführende Informationen unter:
https://colab.research.google.com/notebooks/welcome.ipynb

Des Weiteren sollen die in der Praxis auftretenden Probleme bei der maschinellen Bestimmung von Identitäten beachtet werden, um einen hohen Praxisbezug gewährleisten zu können.

Zu den in der Praxis auftretenden Problemen gehört unter anderem der Mangel an vorhandenen Trainingsbeispielen, welche für ein neuronales Netz von fundamentaler Bedeutung sind. Sollte ein Personenidentifikationssystem praktisch umgesetzt werden, so ist es zielführend, wenn das zur Bestimmung der Identität nötige Wissen aus nur wenigen Trainingsbeispielen erlernt werden kann bzw. die zugrundeliegende Technik es erlaubt, eine Abbildung einer Identität aus möglichst wenig Bildmaterial zu extrahieren. Aus diesem Grunde wird auch evaluiert, inwiefern die für das Training von neuronalen Netzen einsetzbare Data Augmentation positiven Einfluss auf die Genauigkeit des neuronalen Netzes haben kann.

Weiterhin soll evaluiert werden, inwiefern eine Resistenz gegenüber Belichtung und Pose von Bildern geschaffen werden kann, um möglichst gute Ergebnisse in der Bestimmung von Identitäten erreichen zu können. Aus einer Veröffentlichung von Google Inc. Mitarbeitern, welche sich mit derselben Themenstellung befassen, geht hervor:*„Pose and illumination have been a long standing problem in face recognition“* ( [6] ). Auch die Masterarbeit von Herrn Constantin Kirsch befasst sich in Teilen mit möglichen Schritten, um den Einfluss von Belichtung und Pose auf den Lernerfolg von neuronalen Netzen zu minimieren (vgl. [7]). Beide Arbeiten dienen als Grundlage zur praktischen Evaluierung möglicher Lösungsansätze.

Ein weiteres Ziel ist die Erweiterbarkeit der Anwendung, ohne sämtliche Trainingsdaten von Grund auf neu erlernen zu müssen. Sollte sich bei konventionellen Klassifizierungsproblemen und entsprechenden Lösungsansätzen die Klassenanzahl verändern, so würde dies ein erneutes Training mit allen Beispieldaten nach sich ziehen. Um dies zu vermeiden, wird ein erweiterbares System angestrebt, welches die Löschung oder auch die Hinzunahme von Identitäten erlaubt, ohne einen größeren Mehraufwand im Training des neuronalen Netzes zu verursachen.

## 3 Programmierwerkzeuge

Zur praktischen Durchführung des Projekts wurde ein Jupyter-Notebook in der Programmiersprache Python erstellt. Das Notebook wurde für die Ausführung auf der Plattform „Google Colaboratory“ ausgelegt. Für maschinelles Lernen kommt in erster Linie die Programmbibliothek *keras* zum Einsatz, der High-Level API von TensorFlow. Zur Datenvorverarbeitung wird OpenCV verwendet. Auch weitere Bibliotheken, welche die Umsetzung Unterstützen oder welche durch die genannten Programmbibliotheken genutzt werden, wie beispielsweise Numpy, werden verwendet.

## 4 Datenquellen

Um ein Personenidentifikationssystem, welches auf Gesichtsmerkmalen basiert, umsetzen zu können, sind entsprechende Beispieldaten notwendig. Die Beispieldaten fungieren in erster Linie als Gegenbeispiele zur eigentlich zur erlernenden Person. Die Daten entstammen der *Database of Faces*[2] der *AT&T Laboratories Cambridge*. Diese Datenbasis umfasst jeweils 10 Bildaufnahmen von Gesichtern von insgesamt 40 Personen. Die Bildaufnahmen liegen im Graustufenformat vor. Von den daraus resultierenden 400 Bildaufnahmen kommen aber lediglich nur 352 zur Anwendung, da einige Bilder für die Datenaufbereitung unbrauchbar waren. Nähere Informationen hierzu werden im folgenden Kapitel genannt. Weitere Daten entstammen aus eigens angefertigten Aufnahmen, um beispielsweise die Identifikation der eigenen Person vornehmen zu können.

## 5 Datenvorverarbeitung

Ziel der Datenvorverarbeitung ist es, das Bildmaterial für das anschließende Erlernen bzw. Extrahieren von Identitäten vorzubereiten. Im Folgenden werden die in der praktischen Arbeit durchgeführten Schritte näher erläutert.

*Abbildung 1* visualisiert das allgemeine Vorgehen, welches im weiteren Verlauf näher erläutert wird:

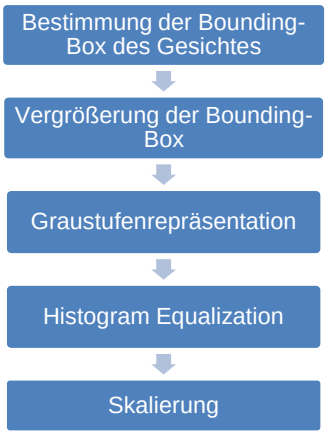



**Abbildung 1: Schematische Darstellung der Datenvorverarbeitung**

[2]Weiterführende Informationen unter:
https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

## 5.1 Gesichtserkennung

Als erster Schritt zur Datenvorverarbeitung ist die Lokalisation von Gesichtern im Bild notwendig. Hierzu wird sich der freien Programmbibliothek OpenCV (cv2) bedient.

OpenCV (Open Source Computer Vision Library) ist eine frei verfügbare Programmbibliothek, welche unter der BSD-Lizenz für maschinelles Sehen eingesetzt wird. (vgl. [8])
OpenCV unterstützt diverse *Haar-Cascades*, mithilfe welcher über den Viola-Jones Algorithmus Rapid Object Detection durchgeführt werden kann. Der Viola-Jones Algorithmus basiert auf dem Sliding-Window Verfahren. Die ausführlich genannten Haar feature-based cascade classifiers (vgl. [7]) enthalten die über überwachtes maschinelles Lernen extrahierten Merkmale. In der prototypischen Anwendung kommt die Haar-Cascade *haarcascade_frontalface_alt2.xml* zum Einsatz, welche auf die Erkennung von Frontalaufnahmen von Gesichtern trainiert wurde. Nach Ausführung der Gesichtserkennung wird eine Python-Liste von Tupeln der Form

$$[(x_1, y_1, w_1, h_1), \quad (x_2, y_2, w_2, h_2), \quad \dots, (x_n, y_n, w_n, h_n)]$$

zurückliefert. *x* sowie *y* stehen in diesem Kontext für die Bounding-Box bezeichnenden Koordinaten auf der Abszisse x und der Ordinate y. *w* und *h* geben entsprechenden die Größe der Bounding-Box, genauer gesagt die Breite (engl. width) und Höhe (engl. height) dieser an. Um mehr Merkmale, auch um das Gesicht, einfangen zu können, wird die Bounding-Box bezüglich ihrer Fläche im Nachgang um 10 % vergrößert, da ein größerer Ausschnitt zur besseren Klassifizierung beitragen kann (vgl. [7]). *Abbildung 2* zeigt das aus den beschriebenen Schritten resultierende Ergebnis, indem die Bounding-Box eines erkannten Gesichtes eingezeichnet wurde.

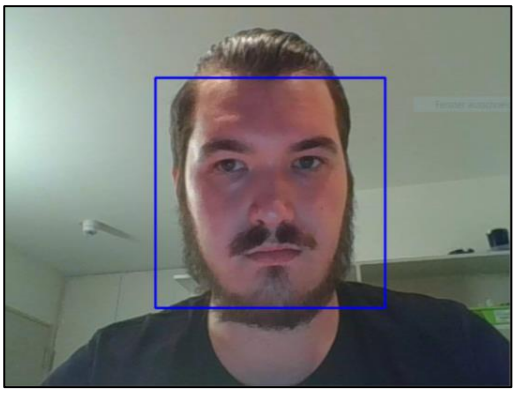


**Abbildung 2: Gesichtserkennung mit Haar-Cascades**

Ein Vorteil bei der Verwendung von Haar-Cascades ist es, dass Gesichter auch Live in Videostreams erkannt und verarbeitet werden können. Da die Haar-Cascade „Frontalface“ nur Frontalaufnahmen von Gesichtern erkennt, werden andere Posen im Voraus ausgeschlossen. Durch die Vergrößerung des Bildausschnittes können auch mehr Merkmale einer Identität in die Entscheidungsfindung des Klassifizierers eingehen.

## 5.2 Optische Normierung

Mit der optischen Normierung soll in erster Linie eine gewisse Resistenz gegenüber diverser Belichtungssituationen erwirkt werden. Weiterhin erfolgt die Skalierung auf eine einheitliche Größe, welche zur anschließenden Verarbeitung durch ein neuronales Netz benötigt wird.

Im ersten Schritt wird das Bild in Graustufen konvertiert. Dies hat den Vorteil, dass über OpenCV eine sogenannte Histogram-Equalization durchgeführt werden kann. Somit wird der Kontrast und das Verhältnis zwischen schwarzen und weißen Bildanteilen insofern normiert, als dass sie einer Gleichverteilung genügen.

*Abbildung 3* und *Abbildung 4* zeigen die Ergebnisse der Gesichtserkennung mit anschließender Histogram-Equalization bei besserem bzw. schlechteren Lichtverhältnissen. Auf der rechten Seite des Bildes wird die Gesichtserkennung visualisiert, während auf der linken Seite die normierte Darstellung des Bildausschnittes erfolgt.

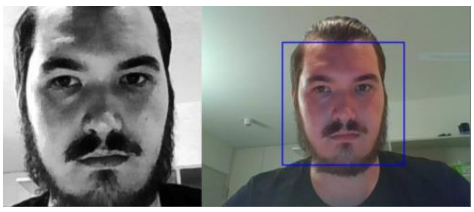


**Abbildung 3: Optische Normierung bei stärkerer Belichtung**

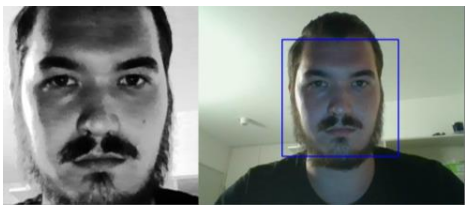


**Abbildung 4: Optische Normierung bei schwächerer Belichtung**

In den Abbildungen ist zu erkennen, dass die Belichtungsverhältnisse durch die Normierung ein Stück weit ausgeglichen werden können.

Weiterhin gibt es noch weitere Formen der optischen Normierung, auf welche jedoch an dieser Stelle aus zeitlichen Gründen verzichtet wurde. Hierbei würde das sogenannte Alignment (engl. für Ausrichtung) des Gesichtes zu einer annähernd einheitlichen Pose führen. Hierbei kann die Programmbibliothek Dlib zur Verwendung kommen (vgl. [7]). Durch die Bestimmung der Position der Augen und der Bildung einer Geraden durch den Schwerpunkt der Augen kann das Bild zur horizontalen Ausrichtung der Geraden gedreht werden. Um eine einheitliche Pose zu gewährleisten, können auch weitere Schritte, wie beispielsweise Scherungen oder weitere grafische Operationen durchgeführt werden, um jedes Gesicht gleich auszurichten.

## 6 Durchführung

Nachdem die Datenvorverarbeitung abgeschlossen wurde, konnte ein Prototyp zur Gesichtserkennung innerhalb eines Jupyter-Notebooks aufgebaut werden. Hierbei wurden zwei verschiedene Ansätze verfolgt, welche in den nächsten Kapiteln näher erläutert werden.

### 6.1 Konventioneller Binärklassifizierer

Als konventionell wird dieser Ansatz bezeichnet, da auf klassische Merkmalsextraktion eines vortrainierten Netztes gesetzt wird und lediglich der Klassifizierer getauscht wird. Die Aktivierungsfunktion des letzten Layers des Klassifizierers ist *sigmoid*, weshalb die Ausgabeschicht Ergebnisse im Intervall von $[0, 1] \subseteq \mathbb{R}$ liefert. Als Faltungsbasis wurde die VGG-16 Architektur (siehe *Abbildung 5*) mit imagenet-Kernel verwendet.

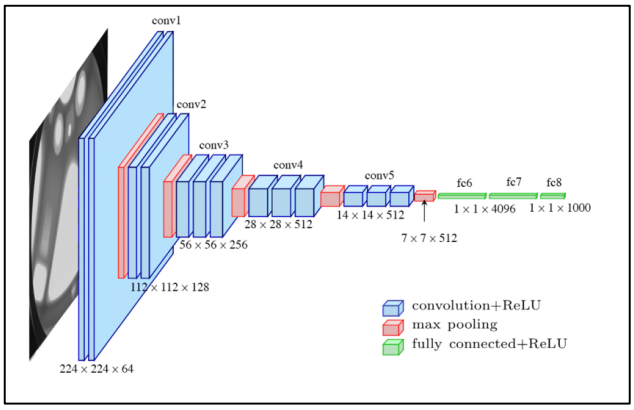



**Abbildung 5: Schematische Darstellung der VGG-16 Architektur (vgl.** [9]**)**

Zunächst wurde eine allgemeine Struktur bereitgestellt, um diverse Personen anlegen, löschen, trainieren und validieren zu können. Hierbei wurden auch Beispieldaten bereitgestellt sowie die Unterstützung der Webcam eingebunden, sofern diese vorhanden ist. Weitere Details befinden sich in der praktischen Arbeit, welche durchgängig kommentiert wurde.

Der Klassifizierer wurde gemäß folgendem Schema aufgebaut (summary-Ausgabe der Programmbibliothek Keras):

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               3211392
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 17,926,209
Trainable params: 17,926,209
Non-trainable params: 0
```

Um dem geforderten Ziel der Erweiterbarkeit gerecht zu werden, wird nun pro zu klassifizierender Person ein neuronales Netz trainiert. Das Verhältnis der Trennung von Test- und Validierungsdaten betrug hierbei 80 zu 20. Wird nun eine Person auf einem Bild bestimmt, so muss ein Schwellwert definiert werden, ab welchem potenziell erkannte Personen als möglich erkannte Identitäten in die nähere Auswahl gelangen. Im Idealfall stellt die nach dem Filterkritierum des Schwellwerts aussortierten Identitäten höchste Ausgabe des letzten Layers die korrekte Identität dar.

Zum prototypischen Training eines Klassifizierers kommen 180 Bilder der eigenen Person sowie 352 Bilder von Gegenbeispielen zum Einsatz (siehe *4 Datenquellen*). Diese wurden auch der bereits erläuterten Datenvorverarbeitung unterzogen.

Nachdem das Training des Klassifizierers mit automatischer Reduktion der Lernrate sowie dem automatischen Beenden des Trainings, sofern keine Verbesserungen mehr erzielt werden, abgeschlossen wurde, konnten die Ergebnisse der Trainings- und Testkorrektklassifizierungsrate sowie die Ergebnisse der Trainings- und Testverlustfunktion näher analysiert werden. *Abbildung 6* zeigt hierbei die Ergebnisse der Korrektklassifizierungsrate, während in *Abbildung 7* die Ergebnisse der Verlustfunktion aufgezeigt werden.

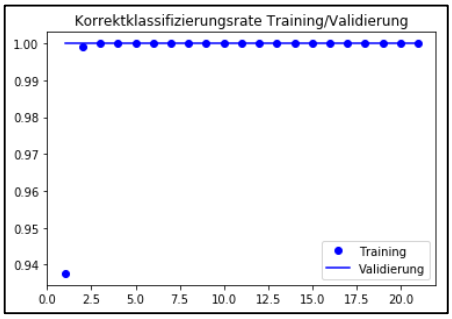



**Abbildung 6: Korrektklassifizierungsrate während des Trainings und der Validierung**

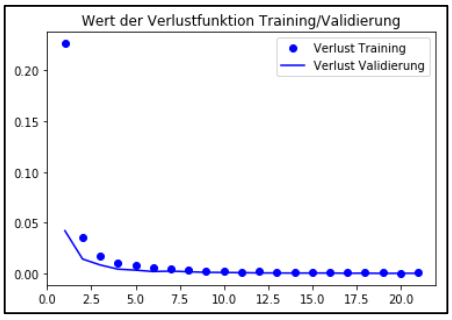



**Abbildung 7: Verlustfunktion während des Trainings und der Validierung**

Wie in den Abbildungen zu erkennen ist, steigt die Korrektklassifizierungsrate bei der Klassifizierung bereits nach der ersten Epoche rapide an.

Nach näherer Untersuchung konnte festgestellt werden, dass es mehrere Gründe für dieses Verhalten geben kann. Zum einen ist es möglich, dass aufgrund der sehr ähnlich beschaffenen Bildausschnitte kaum neue Merkmale in zusätzlichen Epochen erlernt werden konnten oder auch zu wenig Bilddaten vorhanden sein könnten. Der genaue Auslöser für dieses Verhalten konnte jedoch nicht empirisch verifiziert werden.

Bei der praktischen Evaluation des Binärklassifizierers konnte festgestellt werden, dass sich der Alpha-Fehler (Ausschluss einer Identität, obwohl diese vorliegt) in Grenzen hielt, während der Beta-Fehler (Erkennung einer Identität, obwohl diese nicht vorliegt) ziemlich hoch lag. *Abbildung 8* zeigt die korrekte Interpretation des Klassifizierers, während *Abbildung 9* ein Beispiel eines Beta-Klassifizierungsfehlers aufzeigt.

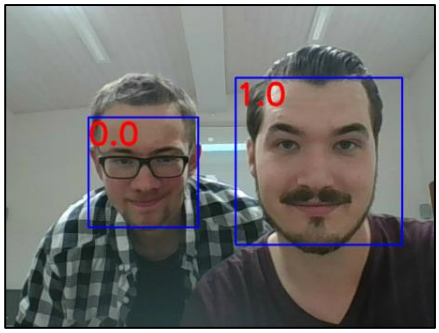



**Abbildung 8: Korrektes Klassifizierungsergebnis des Binärklassifizierers**

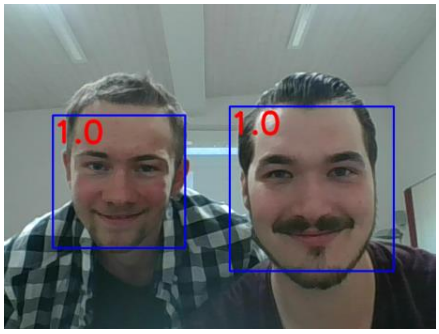



**Abbildung 9: Beta-Fehler bei der Evaluierung des Binärklassifizierers**

Wurden die Anzahl der Gegenbeispiele erhöht, so ergab sich die Problematik, dass im praktischen Test fast immer alle Personen als die zu testende Person erkannt worden sind, auch wenn dies nicht der Fall war (Maximierung des Beta-Fehlers). Auch Data Augmentation konnte nur bedingte Verbesserung hervorrufen. Wenn Data Augmentation zu stark eingesetzt wurde, so sank die Korrektklassifizierungsrate in unabhängigen Tests rapide ab. Dies liegt an der Beschaffenheit der Daten, da durch die Normierung dem neuronalen Netz nur Ausschnitte des Gesichtes, welche frontal aufgenommen sein müssen (dies liegt intrinsisch in der Funktionsweise der Frontalface-Haar-Cascades, da diese nur Frontalaufnahmen erkennen) zugeführt werden. Aggressives Data Augmenting verfremdet die Trainingsdaten entsprechend zu stark, sodass die erlernten Repräsentation im Klassifizierer des neuronalen Netzes dazu neigten, zu stark zu generalisieren.

Weitere Anpassung, wie beispielsweise

- die Anpassung der Learning-Rate
- die Verwendung von GlobalAveragePooling2D statt Flatten
- Veränderung der Klassifiziererarchitektur (Dense-Layer) sowie der Unit-Größe
- Veränderung der Batchsize
- das Durchmischen der Daten, um einen gleichmäßigen Trainings- und Validierungssplit zu gewährleisten, wenn eine größere Menge an Bilddaten in ähnlichen Verhältnissen aufgenommen worden sind, welche jedoch nur in die Validierungsmenge und nicht in die Trainingsdatenmenge fallen
- die Veränderung des Trainings-/Validierungssplits
- Weiterführende Feinabstimmung der letzten 3 Convolutional Layer der Faltungsbasis mit niedriger Learning-Rate
- dem Bereitstellen von neuen Trainingsdaten mit diversen Lichtverhältnissen

sowie auch weiterer Metaparameter gab keine Verbesserungen. Aus diesem Grunde wurde der Einsatz eines konventionellen Binärklassifizierers an dieser Stelle nicht weiter verfolgt und nach einem weiteren Lösungsansatz gesucht.

## 6.2 Convolutional-Autoencoder

Der weiterhin verfolgte Ansatz basiert auf einem Blogbeitrag von Sefik Ilkin Serengil, von welchem auch die Codebeispiele als Grundlage für die Evaluierung des zweiten Ansatzes verwendet wurden. (vgl. [10])

Dieser Ansatz basiert auf der Merkmalsextraktion des VGG-Face netztes, welches ähnlich dem VGG-16 Netz aufgebaut ist, jedoch auf Gesichter trainiert wurde und sich für die Merkmalsextraktion bei der Identitätsbestimmung demnach bestens eignet.

Die Netzarchitektur des VGG-Face Netztes ist in *Abbildung 10* dargestellt.

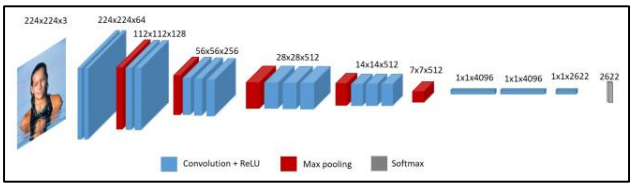



**Abbildung 10: Netzarchitektur des VGG-Face-Netzes (vgl.** [10]**)**

Die verwendete Technik bei der Identitätsbestimmung mit dem Convolutional-Autoencoder Ansatz unterscheidet sich grundlegend von der des konventionellen Binärklassifizierers.

Anstatt Beispiele für Personen und Gegenbeispiele zu trainieren, wird zunächst auf das VGG-Face-Netz als Faltungsbasis gesetzt, um eine optimale Merkmalsextraktion für Gesichter bereitzustellen. Als Convolutional-Autoencoder wird dieser Ansatz bezeichnet, da der Klassifizierer der Faltungsbasis verworfen wird und auf der Vektorrepräsentation des letzten Convolutional-Layers weitergearbeitet wird. Demnach wird ein potenziell hochdimensionales Eingabebild auf eine viel kleinere Vektorrepräsentation des Gesichtes eingedampft (2622 Parameter). (vgl. [10])

Diese Vorgehensweise garantiert auch eine gewisse Resistenz gegenüber Belichtung und Pose, da dieses Wissen schon in gewissem Maße in den Gewichten des neuronalen Netzes vorhanden ist, welches schon im Vornherein mit vielen Beispieldaten trainiert wurde. Die unter *5.2 Optische Normierung* erwähnte Vorgehensweise des Alignments könnte jedoch nochmal eine deutliche Leistungssteigerung hervorrufen, welche jedoch praktisch evaluiert werden müssten und an dieser Stelle aus zeitlichen Gründen auch nicht weiter verfolgt wurde.

Die aus der Faltungsbasis resultierende Vektorrepräsentation dient nun als Grundlage einer One-Shot Klassifikation. Die Vektorrepräsentation des Eingabebildes erzeugt bei ähnlichen Eingabebildern auch ähnliche Ausgaben. Wird nun der euklid'sche Abstand bzw. der Cosinusabstand der Vektorrepräsentationen berechnet, dient das resultierende Ergebnis als Metrik der Ähnlichkeit von Gesichtern. Im Blogbeitrag von Sefik Ilkin Serengil werden Schwellwerte für die Gleichheit von Gesichtern bei der euklid'schen Abstandsfunktion von 120, bei dem Ansatz mit der Berechnung des Cosinusabstands von 0,40 angegeben. Unterschreitet der Abstand der Vektorrepräsentation von Bildern von Gesichtern diese Schwellwerte, so kann von einer Gleichheit einer Identität ausgegangen werden. (vgl. [10])

Dieser Ansatz bietet enorme Vorteile. So ist dieser Ansatz nicht nur dynamisch und einfach erweiterbar, vielmehr ist das Training eines spieziellen Netzes nur ein einziges Mal nötig, nämlich für die Faltungsbasis. Alle weiteren Schritte können als One-Shot-Klassifikation verstanden werden, da lediglich potenziell nur ein Referenzbild pro Person benötigt wird, um die zugrundeliegende Identität durch einmalige Berechnung der Vektorrepräsentation auch bei weiteren Bildern feststellen zu können. Der Ansatz könnte verfeinert werden, indem das arithmetische Mittel von mehreren Vektorrepräsentationen einer Person gebildet wird.

Nicht nur die Bestimmung von Personen unter Zuhilfenahme von nur einem Referenzbild kann mit diesem Ansatz gelöst werden, es ergeben sich sogar komplett neue Einsatzmöglichkeiten. Somit können über diesen Ansatz sogar Identitäten geclustert werden, indem die Ähnlichkeit von ungelabelten Eingabedaten berechnet wird und ähnliche Identitäten zusammengefasst werden.

Bei der praktischen Evalution dieses Ansatzes im Rahmen der Studienarbeit konnten deutlich zuverlässigere und bessere Ergebnisse erzielt werden, als mit dem konventionellen Binärklassifizierer.

*Abbildung 11* zeigt beispielhaft ein Referenzbild, zu welchem mithilfe des Convolutional-Autoencoder Ansatzes die euklid'schen Abstände zu den in *Abbildung 12* dargestellten Gesichter berechnet wurden. Diese euklid'schen Abstände wurden in *Abbildung 12* auch mit eingezeichnet.

**Abbildung 11: Referenzbild zur Evaluierung des Convolutional-Autoencoder Ansatzes**

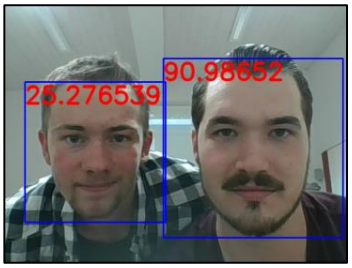



**Abbildung 12: Testbild zur Evaluierung des Convolutional-Autoencoder Ansatzes**

Abschließend kann festgestellt werden, dass der Einsatz eines Convolutional-Autoencoders bei der Personenidentifikation deutliche Vorteile im Gegensatz zum konventionellen Binärklassifizierer aufzeigt. Aus diesem Grunde lieferte dieser Ansatz in der praktischen Evaluation auch bessere Ergebnisse.

## 7 Fazit

Die Arbeit konnte sehr stark zum grundlegenden Verständnis von neuronalen Netzen und künstlicher Intelligenz beitragen. Das selbst beschaffte Begleitwerk „Deep Learning mit Python und Keras" von Francois Chollet, dem Entwickler der Keras-Bibliothek, konnte zu den in der Vorlesung behandelten Grundlagen optimale Ergänzungen zum Verständnis der Thematik liefern. Im Allgemeinen konnte festgestellt werden, dass der technische Einsatz von neuronalen Netzen diverse Probleme mit sich bringt, angefangen bei der Datenvorverarbeitung, den Metaparametern des

neuronalen Netzes bishin zur eigentlichen Beschaffung der Trainingsdaten. Die Arbeit ermöglichte es, den sinnvollen Einsatz von diversen Techniken (Datenvorverarbeitung, Data Augmenting, Merkmalsextraktion und Feinabstimmung, Convolutional-Autoencoder Ansatz usw.) am praktischen Beispiel gewinnbringend zu evaluieren, um auch abseits der Studienarbeit neuronale Netze sinnvoll zur Lösung von praktischen Problemstellungen einsetzen zu können.

## LITERATURVERZEICHNIS


[1] M. A. I. Faßbender, „PUB - Publikationen an der Universität Bielefeld," 2015. [Online]. Available: https://pub.uni-bielefeld.de/download/2900351/2900352. [Zugriff am 05 07 2019].

[2] „Inhalte und Hinweise zur DSGVO / EU-Datenschutz-Grundverordnung," 27 04 2016. [Online]. Available: https://www.datenschutz-grundverordnung.eu/. [Zugriff am 05 07 2019].

[3] A. Dorloff und D. Satra, „China: Auf dem Weg zur totalen Überwachung | tagesschau.de," ARD-Studio Peking, 24 03 2019. [Online]. Available: https://www.tagesschau.de/ausland/ueberwachung-china-101.html. [Zugriff am 05 07 2019].

[4] „Gesichtserkennung in China verwechselt Bus mit Fußgänger," FAZ, 23 11 2018. [Online]. Available: https://www.faz.net/aktuell/gesellschaft/menschen/gesichtserkennung-in-china-verwechselt-bus-mit-fussgaenger-15905254.html. [Zugriff am 05 07 2019].

[5] „Modulhandbücher," [Online]. Available: https://www.hof-university.de/studierende/info-service/modulhandbuecher.html?tx_modulhandbuch_modulhandbuch%5Bid2%5D=6636&tx_modulhandbuch_modulhandbuch%5Bid%5D=22654&tx_modulhandbuch_modulhandbuch%5Byear%5D=SS%202019&tx_modulhandbuch_modulhandbuch%5Bclas. [Zugriff am 05 07 2019].

[6] F. Schroff, D. Kalenichenko und J. Philbin, „arxiv.org," 17 06 2015. [Online]. Available: https://arxiv.org/pdf/1503.03832.pdf. [Zugriff am 06 07 2019].

[7] C. Kirsch, „www.h-brs.de," 29 08 2017. [Online]. Available: https://www.h-brs.de/files/20170829_fbinf_mclab_ss17_ma_constantin kirsch_neuronalenetze_kj_0.pdf. [Zugriff am 06 07 2019].

[8] „About," [Online]. Available: https://opencv.org/about/. [Zugriff am 06 07 2019].

[9] M. Ferguson, „Fig. A1. The standard VGG-16 network architecture as proposed in [32].... | Download Scientific Diagram," 12 2017. [Online]. Available: https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture-as-proposed-in-32-Note-that-only_fig3_322512435. [Zugriff am 06 07 2019].

[10] S. I. Serengil, „Deep Face Recognition with VGG-Face in Keras | sefiks.com," 06 08 2018. [Online]. Available: https://sefiks.com/2018/08/06/deep-face-recognition-with-keras/. [Zugriff am 06 07 2019].

# Bestärkendes Lernen

Martin Heckel
martin.heckel@hof-university.de
Hochschule für Angewandte Wissenschaften Hof
Hof

## ZUSAMMENFASSUNG

In dieser Studienarbeit werden einige grundlegende Ansätze und Hintergründe des bestärkenden Lernens vorgestellt. Im Anschluss folgt eine Vorstellung des im Modul "Angewandtes maschinelles Lernen" umgesetzten Programms zur anschaulichen Demonstration der verwendeten Techniken. Es folgen Erklärungen und Statistiken der im Rahmen dieser Studienarbeit gesammelten Erfahrungen zum Layout und Verhalten des Netzes.

## 1 EINLEITUNG

Häufig werden neuronale Netze trainiert, indem das Netz ein Problem löst, für das die Lösung bereits bekannt ist. Dieser Ansatz wird als **überwachtes Lernen** bezeichnet. Eine typische Problemklasse, bei der dieser Ansatz zum Einsatz kommt, sind Klassifizierungsprobleme. Dabei soll ein Netz lernen, Objekte (z.B. Bilder) einer von mehreren gegebenen Kategorien zuzuordnen. Das Netz wird mit beschrifteten Datensätzen trainiert, d.h. jedes Objekt des Datensatzes wurde bereits klassifiziert. Das Netz lernt auf Basis dieser Daten, entsprechende Objekte zu klassifizieren.

Solche Netze lassen sich meist gut trainieren und erzielen gute Ergebnisse. Der Nachteil besteht allerdings darin, dass zum Training entsprechende beschriftete Datensätze notwendig sind. Bei manchen Problemstellungen ist dies nicht möglich, weshalb dieser Ansatz nicht zur Lösung aller Problemtypen verwendet werden kann.

Der Ansatz des **bekräftigenden Lernen** löst dieses Problem, indem die Datensätze zum Training dynamisch während der Interaktion des Netzes mit der Umgebung gesammelt werden. Ein typischer Anwendungsbereich des bekräftigenden Lernen sind Spiele: Ein Spieler interagiert mit der Umgebung, indem er Aktionen ausführt. Diese Aktionen können den Zustand der Umgebung verändern. Das Ziel besteht darin, mit Hilfe des Netzes eine gute Aktion in einem gegebenen Zustand zu wählen.

Dabei besteht allerdings das Problem, dass es häufig nicht möglich ist, eine Aktion während eines Spiels zu bewerten, da die Bewertung dieser Aktion von den folgenden Aktionen abhängt. Im Rahmen dieser Arbeit werden einige Grundlagen und Ansätze vorgestellt, dieses Problem zu lösen.

Im Rahmen der Studienarbeit sollte eine Anwendung entwickelt werden, die das Verhalten des entwickelten Netzes möglichst anschaulich demonstriert. Im Abschnitt 2 "Theorie" wurden einige der möglichen Ansätze vorgestellt. In den folgenden Abschnitten wurde das Netz auf Basis von Deep-Q-Learning implementiert. Darauf hin wurde eine grafische Oberfläche entwickelt, um die Fähigkeiten und Grenzen des Netzes anschaulicher nachvollziehen zu können. Außerdem wurden verschiedene Trainingsmodi implementiert, um diese miteinander vergleichen zu können.

*FWPM: Angewandtes maschinelles Lernen, Sommersemester, 2019, Hof*

## 2 THEORIE

### 2.1 Allgemeine Ansätze

Die Herausforderung beim bestätigenden Lernen besteht darin, ein neuronales Netz so zu trainieren, dass es mit Hilfe dieses Netzes möglich ist, in einem gegebenen Zustand der Umgebung eine möglichst gute Aktion zu wählen. Um dies zu erreichen, gibt es zwei grundlegende Ansätze: das Lernen der **Aktion** und das Abschätzen des **Wertes** eines gegebenen Zustands.

Wenn das Netz lernen soll, welche Aktion $a$ es in einem Zustand $s$ wählen soll, ist es notwendig, über entsprechende Tainingsdaten zu verfügen. Allerdings ist es häufig nicht möglich, direkt abzuschätzen, wie gut eine Aktion in einem Zustand ist, da der Wert dieser Aktion von den folgenden Aktionen abhängt.

Ein gutes Beispiel dafür ist das Bauernopfer beim Schach: Ein Spieler opfert einen Bauern, um dadurch einen anderweitigen Vorteil im Spiel zu erlangen. Wenn man diese Aktion direkt abschätzen würde, wäre die Aktion ziemlich schlecht, da der Spieler ja eine Figur verloren hat, ohne dadurch (direkt) einen Vorteil zu haben. Der Vorteil wird erst durch Distanz deutlich (z.B. da der Spieler im nächsten Zug eine Figur des Gegners schlägt). Bei komplexeren Strategien kann es mehrere Züge dauern, bis der durch einen früheren Zug erzielte Vorteil zur Geltung kommt. Der einzige Zeitpunkt, an dem der Wert einer Aktion direkt angegeben werden kann, ist das Spielende, da es ab diesem Zeitpunkt nicht mehr möglich ist, durch vorherige Züge erhaltene Vorteile zu nutzen.

Ein analoges Problem besteht beim Lernen des Wertes $v$ eines gegebenen Zustands $s$: Es ist möglich, dass ein durch eine Aktion erreichter Zustand einen schlechten Werte hätte (z.B. da der Gegner mehr Figuren im Spiel hat), dadurch allerdings ein Vorteil erzielt wird, der einige Züge später genutzt werden kann.

Im Abschnitt "Q-Learning" wird beschrieben, wie es möglich ist, einen Zustand, der kein Endzustand ist, dennoch zu bewerten.

Bei neuronalen Netzen handelt es sich um hochdimensionale Optimierungsprobleme, da der gesamte Abstand der Vorhersagen des Netzes zu den gewünschten Ergebnissen minimiert wird. Die Parameter ("Gewichten") des Netzes sollen also so angepasst werden, dass dieser Abstand minimal wird. Das Ziel des Trainings ist es, ein gutes Minimum zu finden. Dabei kann das Problem auftreten, dass die Parameter so optimiert wurden, dass ein Minimum gefunden wurde, das gefundene Minimum allerdings nicht besonders gut ist. Um die Darstellung etwas zu vereinfachen, wurde eine Funktion mit einem Parameter verwendet ($f(x) = 3sin(x + 25) + 3cos(x + 25) - sin(0.5(x + 25)) - 20cos(0.1(x + 25)) + 30$):

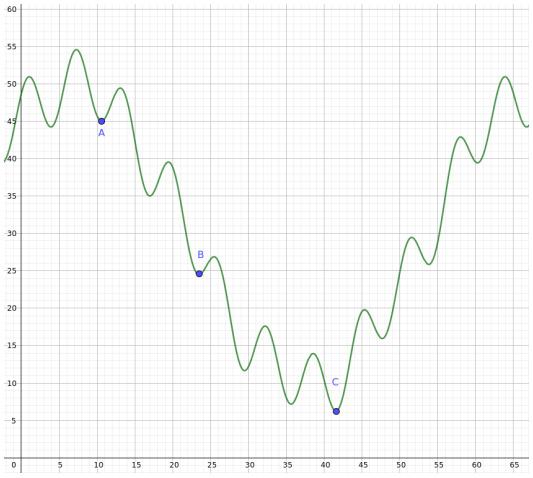



**Abbildung 1: Minima einer Funktion mit einem Parameter**

Wie man an Abbildung 1 gut sehen kann, handelt es sich bei $A$ um ein Minimum, allerdings ist dieses Minimum global betrachtet nicht besonders gut, da $B$ und $C$ deutlich niedriger liegen. Wenn man allerdings von $A$ nach einen Minimum sucht, sind die Funktionswerte der Umgebung höher, weshalb wieder $A$ als Minimum gefunden werden würde. Die Konsequenz davon ist, dass das Netz die Minima bei $B$ und $C$ nicht finden würde.

Um dieses Problem zu beheben, lässt man das Netzwerk "erkunden". Dabei wird mit einer Wahrscheinlichkeit $\epsilon$ eine zufällige Aktion gewählt. Durch diesen Ansatz kann das Netz auch bessere als bereits gefundene Minima finden. Natürlich besteht auch die Möglichkeit, dass die Zufällige Aktion schlechter ist. In diesem Fall würde das Netz lernen, dass diese Aktion nicht gut war und bei der bekannten Aktion bleiben.

Das Ziel besteht an dieser Stelle darin, einen guten Wert für $\epsilon$ zu finden. Ist $\epsilon$ zu groß, werden vom Netz gelernte Eigenschaften zu wenig berücksichtigt, ist $\epsilon$ zu klein, findet zu wenig Erkundung statt und das bereits gelernte nicht optimale Verhalten wird nicht verbessert.

## 2.2 Markov-Entscheidungsprozess

Der Markov-Entscheidungsprozess [2] ist eine Möglichkeit, Entscheidungsprobleme mathematisch zu formulieren: $MEP = (S, A, t, r)$ mit

- $S$: Menge aller Zustände, die die Umgebung annehmen kann
- $A$: Menge aller Aktionen, die der Agent durchführen kann
- $t$: Übergangsfunktion, die abhängig von einem Zustand und der in diesem Zustand gewählten Aktion den Folgezustand ausgibt ($t(s, a) = s'$)
- $r$: Wertfunktion, die den Wert einer Aktion in einem gegebenen Zustand ausgibt ($r(s, a) = v$)

Die Lösung des MEP ist eine Funktion $\pi(s) = a$, die zu einem Zustand die optimale in diesem Zustand wählbare Aktion ausgibt. Diese Funktion wird als **Policy** bezeichnet.

Im praktischen Teil dieser Studienarbeit wurde das Spiel "TicTacToe" betrachtet. Dieses Spiel kann als $MEP = (S, A, t, r)$ formuliert werden:

- Der Zustand des Spielfelds kann codiert werden, indem die Felder von links oben nach rechts unten zu einer Reihe zusammengefasst werden. Jedes leere Feld wird durch eine 0, jedes eigene Feld durch eine 1 und jedes gegnerische Feld durch eine 2 dargestellt. Zusätzlich gibt es einen ungültigen Zustand (z.B. 333333333). Entsprechend gilt also: $S = \{000000000, 000000001, \ldots, 012021000, \ldots, 333333333\}$
- Der Spieler kann jedes Feld als Aktion auswählen: $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$
- Die Übergangsfunktion der Zustände wird entsprechend der Spielregeln erstellt, z.B.: $t(000000000, 0) = 100000000$
- Die Wertfunktion kann für Zustände und Aktionen, die zu Endzuständen führen (Ein Spieler gewinnt, alle Felder sind belegt, ungültiger Zustand) angegeben werden, z.B.: $v(120102000, 6) = 42$, wenn Werte für diese Zustände definiert werden.

$S$, $A$ und $t$ sind entsprechend vollständig von der Umgebung vorgegeben Das Problem besteht darin, dass $r$ nur teilweise durch die Umgebung vorgegeben ist. Wenn $r$ vollständig bekannt wäre, könnte man das MEP lösen und die Funktion $\pi(s)$ aufstellen.

Im Fall von TicTacToe ist es möglich, die Funktion $r$ vollständig aufzustellen und das MEP zu lösen. Bei komplexeren Umgebungen (z.B. Schach) wäre dieses Vorgehen praktisch nicht anwendbar. Aus diesem Grund kann man ein neuronales Netz verwenden, welches die Funktion $\pi(s)$ lernen soll.

## 2.3 Q-Learning

Um die Policy-Funktion $\pi(s)$ zu lernen, muss ein neuronales Netz mit Datensätzen trainiert werden, die folgende Informationen enthalten:

- $s$: Zustand
- $a$: gewählte Aktion
- $v$: Wert der Aktion in diesem Zustand

Ein Trainingsdatensatz kann also als Tupel $(s, a, v)$ dargestellt werden. Dadurch tritt allerdings das im vorherigen Abschnitt beschriebene Problem auf, dass $v$ nur für Endzustände bekannt ist. Dieses Problem lässt sich dadurch lösen, dass jedes Tupel aus Zustand und Aktion einen **Q-Wert** erhält, der mit Hilfe der Bellman-Gleichung berechnet wird: $Q(s, a) = r + \gamma(max(Q(s', a'))$ Anschaulich gesprochen wird der Q-Wert also berechnet, indem der direkte Wert (0, wenn es sich nicht um einen Endzustand handelt) mit dem Q-Wert der besten Aktion im Folgezustand addiert wird.

Da der Zug, der zum Sieg geführt hat, einen besseren Wert haben sollte als vorhergehende Züge, wird der Q-Wert der besten Aktion im Folgezustand mit einem Reduktionsfaktor $\gamma$ multipliziert. Dadurch nähert sich der Q-Wert umso stärker 0 je größer die Distanz zum Endzustand ist.

Bei einfachen Umgebungen können die auf diese Weise berechneten Werte als Funktion $r$ in den MEP eingesetzt werden. Darauf hin kann der MEP gelöst und die Funktion $\pi(s)$ berechnet werden. Im Folgenden wird dieser Ansatz anhand einer einfachen Umgebung erläutert:

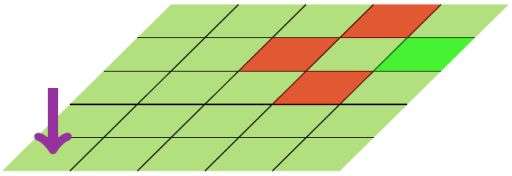


**Abbildung 2: Einfache Umgebung**

Der Spieler startet auf dem unteren linken Feld und kann sich in 4 Richtungen bewegen (rechts, links, nach vorn, nach hinten). Wenn der Spieler in das hervorgehobene grüne Feld läuft, hat er gewonnen. Läuft er in ein rotes Feld, hat er verloren. Im nächsten Schritt werden die Werte dieser Endfelder definiert. Gewinnen ist als +10, Verlieren als −10 definiert:

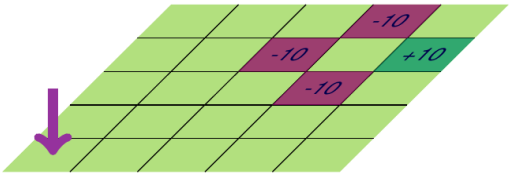



**Abbildung 3: Einfache Umgebung mit Werten auf Endfeldern**

Darauf hin wird die Bellman-Gleichung angewendet. Um zu ungerade Zahlen zu verhindern und dadurch die Darstellung zu verbessern wurde in diesem Beispiel nicht mit einem Faktor multipliziert, sondern der Wert des besten Folgefeldes verwendet und davon 1 subtrahiert:

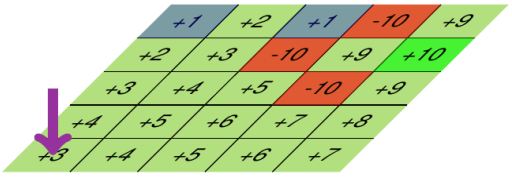



**Abbildung 4: Einfache Umgebung mit Werten**

Nun ist einfach zu erkennen, dass $\{r, r, r, r, v, v, v\}$ eine optimale Folge von Aktionen ist, um in der gegebenen Umgebung mit einer möglichst geringen Anzahl von Aktion zu gewinnen.

Dieses Vorgehen ist bei komplexen Umgebungen allerdings nicht geeignet, da der Aufwand für die Berechnung aller Q-Werte bei komplexen Umgebungen sehr groß ist. Um Q-Learning auch für komplexe Umgebungen verwenden zu können, kann ein neuronales Netz verwendet werden, das die Policy-Funktion $\pi(s)$ lernt. Mit der oben beschriebenen Herangehensweise kann das Netz also ein Spiel spielen. Die dabei gesammelten Tupel werden zu einem Array hinzugefügt. Der Wert für $v$ wird dabei jeweils auf 0 gesetzt, wenn die Aktion keine direkte Belohnung erzielt hat. Am Ende eines Spiels ist der Wert der letzten Aktion bekannt. Nun können die Werte für $r$ berechnet werden. Darauf hin wird das Netz mit diesem Datensatz trainiert.

Dieser Prozess wird so lange wiederholt bis das Netz entweder die gewünschte Qualität hat (z.B. 90% der Spiele gewinnen) oder das Netz trotz vieler Wiederholungen keine weitere Verbesserung mehr erreicht.

## 2.4 Deep-Q-Learning

Deep-Q-Learning ist ein auf Q-Learning aufbauender Algorithmus, der im Gegensatz zum klassischen Q-Learning ein komplexes neuronales Netz mit mehreren Schichten verwendet.

Da die im Verlauf des Spiels gespeicherten Erfahrungen stark korreliert sind (Der aktuelle Zustand hängt sehr stark von den vorher gewählten Aktionen ab), kann es sinnvoll sein, diese Korrelation der Daten aufzulösen. Dafür werden mehrere Spiele gespielt und die Erfahrungen in einen Puffer geschrieben. Danach werden die einzelnen Q-Werte berechnet und das Netz trainiert. Beim Training werden die Datensätze allerdings in zufälliger Reihenfolge verwendet, wodurch die Datensätze nicht mehr korreliert sind.

Eine optionale Erweiterung für DQN besteht darin, ein zweites Netz zu trainieren, das den Wert eines Zustands abschätzt. Dieser Ansatz verbindet also die Vorteile des Lernens einer Aktion und des Lernens der Werte von Zuständen.

Wenn das DQN lernen soll, Videospiele zu spielen, schaltet man häufig einige convolutional-Layers vor die eigentlichen Schichten des DQN, um eine möglichst gute Vorverarbeitung der Bilddaten des Videospiels zu erreichen. Außerdem werden häufig mehrere Bilder gleichzeitig zum Trainieren verwendet, um die Spieldynamik darzustellen: Bei dem Spiel Pong ist es z.B. wichtig zu wissen, in welchem Winkel sich der Ball zwischen den Schlägern bewegt. Diese Information lässt sich allerdings nicht aus einem einzelnen Bild gewinnen. Sind hingegen zwei Bilder vorhanden, kann die Information des Winkels verwendet werden.

## 2.5 A3C

Der Algorithmus A3C [1] “Asynchronous Advantage Actor-Critic” verbindet drei grundlegende Ansätze:

- **Asynchronous**:
  Es gibt mehrere Agenten, von denen jeder über eine eigene von den anderen Agenten unabhängige Kopie der Umgebung verfügt. Zusätzlich verfügt jeder Agent über eine eigene Version des neuronalen Netzes. Mehrere Agenten werden parallel mit ihrer jeweiligen Umgebung trainiert. Nach einer bestimmten Anzahl von Trainingsschritten werden die gelernten Gewichte mit einem globalen Netz synchronisiert. Dazu gibt es generell mehrere Ansätze. Der einfachste Ansatz ist das Synchronisieren durch eine “laufenden Durchschnitt”. Dabei werden die jeweiligen Gewichte mit einem Faktor multipliziert und addiert. Die Summe der Faktoren ergibt 1. Beim ersten Synchronisationsschritt wird das Gewicht des globalen Netzes mit 0 und das Gewicht des Subnetzes mit 1 multipliziert, das globale Netz wird also praktisch überschrieben. Beim zweiten Synchronisationsschritt wird das Gewicht des globalen Netzes mit $\frac{1}{2}$ und das Gewicht des Subnetzes

mit $\frac{1}{2}$ multipliziert. Beim n-ten Synchronisationsschritt wird das Gewicht des globalen Netzes mit $\frac{n-1}{n}$ und das Gewicht des Subnetzes mit $\frac{1}{n}$ multipliziert.

- **Actor-Critic**:
  Das Netz besteht aus einem Actor- und einem Critic-Teil. Der Actor-Teil ist für die zu wählende Aktion zuständig, der Critic-Teil für die Abschätzung des Werts eines Zustandes.
- **Advantage**:
  Die Critic gibt an, wie gut ein Zustand ist ($c(s)$), der Actor gibt an, wie gut eine Aktion in einem Zustand ($a(s, a)$). Der Vorteil durch das Wählen einer Aktion kann also wie Folgt berechnet werden: $A(s, a) = a(s, a) - v(s)$. Wenn die Umgebung erlaubt, keine Aktion durchzuführen, kann der Advantage dazu genutzt werden, um zu entscheiden, ob überhaupt eine Aktion durchgeführt wird: Ist $A(s, a) \leq 0$, wird keine Aktion durchgeführt.

A3C bietet einige Vorteile gegenüber den vorher beschriebenen Ansätzen:

- Geschwindigkeit:
  Durch das parallele Trainieren mit mehreren Umgebungen kann zur Verfügung stehende Rechenleistung besser genutzt werden. A3C könnte z.B. auf einem Cluster laufen.
- Keine Korrelation:
  Da die verschiedenen Agents parallel trainieren, besteht keine Korrelation zwischen den Trainingsdaten, da diese zwar innerhalb eines Agents korreliert sind, diese Korrelation allerdings durch das Mergen verloren geht.
- Keine Aktion wählen:
  Es ist möglich, zu erkennen, ob das Wählen einer Aktion die Situation verbessert oder nicht. Wenn die Umgebung das zulässt, kann also auch keine Aktion gewählt werden

# 3 UMSETZUNG

## 3.1 Allgemeines

Um zu veranschaulichen, wie künstliche neuronale Netze trainiert werden, wurde im Rahmen dieser Studienarbeit ein Programm erstellt, mit dem das Spiel "TicTacToe" gegen ein künstliches neuronales Netz gespielt werden kann. Außerdem kann das Netz Trainiert werden, wobei einige Parameter des Trainings angepasst werden. Es werden Statistiken zur Qualität des Netzes ausgegeben.

Da sich der Fokus im Verlauf der Studienarbeit von der Implementierung eines hochwertigen Algorithmus und dem Erreichen einer möglichst hohen Gewinnquote zur anschaulichen Darstellung des Trainings verschoben hat, wurde ein DQN-ähnlicher Algorithmus implementiert.

Beim Training spielt ein Agent, der mit dem Netz verbunden ist, gegen einen Computergegner wählbarer Schwierigkeit. Dabei wird aufgezeichnet, wie viele Spiele das Netz gewonnen oder verloren hat, wie oft es unentschieden gespielt hat und wie viele Spiele es durch einen ungültigen Zug verloren hat. Diese Statistikdaten werden auch global erhoben. Zusätzlich wird die Anzahl insgesamt gespielter Spiele abgespeichert. Das Netz und die Statistikdaten können gespeichert und wieder geladen werden.

Während eines Trainingsspiels wird gespeichert, welche Aktion in welchem Zustand welchen Wert erreicht hat. Am Ende des Spiels werden die Werte wie in den vorherigen Abschnitte beschrieben berechnet. Anschließend wird das Netz mit diesen gespeicherten Datensätzen trainiert.

Zusätzlich hat der Benutzer auch die Möglichkeit, selbst gegen das Netz zu spielen. Dabei kann er wählen, ob das Netz aus diesem Spiel lernen soll oder nicht.

## 3.2 Programm

Das Programm wurde in der Programmiersprache Python [3] mit der Bibliothek PySide2 [4], einem Wrapper für QT [5] unter Python, erstellt. Das künstliche neuronale Netz wurde mit Keras [6] und dem TensorFlow [7] Backend erstellt.

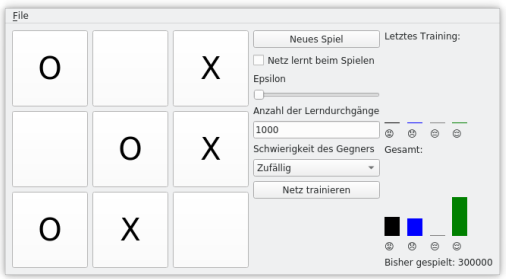



**Abbildung 5: Oberfläche des Spiels**

Im linken Bereich befindet sich das Spielfeld. Dort kann der Benutzer gegen das Netz spielen, wenn es nicht gerade trainiert wird. In der Menübar befindet sich das Menü "Datei". Über dieses Menü kann ein neues Netz erstellt, das aktuelle Netz gespeichert oder ein gespeichertes Netz geladen werden. Wird ein neues Netz erstellt, wenn das aktuelle Netz nicht gespeichert ist, erscheint direkt ein Speichern-Dialog, mit dem das aktuelle Netz gespeichert werden kann.

Im mittleren Bereich befindet sich die Steuerung. Dort kann ein neues Spiel gestartet werden. Es gib eine Checkbox zur Auswahl, ob das Netz beim Spielen lernen soll oder nicht. Zusätzlich kann mit dem Slider "Epsilon" angepasst werden, mit welcher Wahrscheinlichkeit der Agent die Ausgabe des Netzes ignoriert und einen zufälligen Zug wählt. Die Textbox "Anzahl der Lerndurchgänge" enthält die Zahl der Spiele, die das Netzt bei einem Klick auf den Button "Netz trainieren" spielen soll. Mit dem DropDown-Menü "Schwierigkeit des Gegners" kann eine von vier Schwierigkeitsstufen des Gegners, gegen den das Netz trainiert wird, gewählt werden:

- **Zufällig**:
  Es wird ein zufälliger, möglicherweise auch ungültiger Zug gewählt
- **Einfach**:
  Es wird ein zufälliger, garantiert gültiger Zug gewählt
- **Mittel**:
  Wenn der Gegner im nächsten Zug gewinnen kann, wird dies verhindert. Sonst wird ein zufälliger gültiger Zug gewählt
- **Schwer**:
  Wenn es möglich ist, in diesem Zug zu gewinnen, wird dieser

Zug genommen. Kann der Gegner im nächsten Zug gewinnen, wird dies verhindert. Sonst wird ein zufälliger gültiger Zug gewählt

Im rechten Bereich des Fensters werden Statistiken angezeigt. Dabei zeigt der obere Teil die Statistiken des letzten Trainings an (ein Spiel, bei dem das Netzwerk lernt, zählt auch als Training). Der untere Teil zeigt die globale Statistik des Netzes seit dessen Erstellung an.

Der linke Balken gibt den Anteil der Spiele an, die das Netz durch einen ungültigen Zug verloren hat. Der Balken daneben zeigt den Anteil der Spiele, die das Netz verloren hat. Der Balken daneben zeigt, wie viele Spiele unentschieden ausgegangen sind. Der rechte Balken gibt den Anteil der Spiele, die das Netz gewonnen hat, an.

Darunter steht die Anzahl der von dem Netz seit der Erstellung gespielten Spiele.

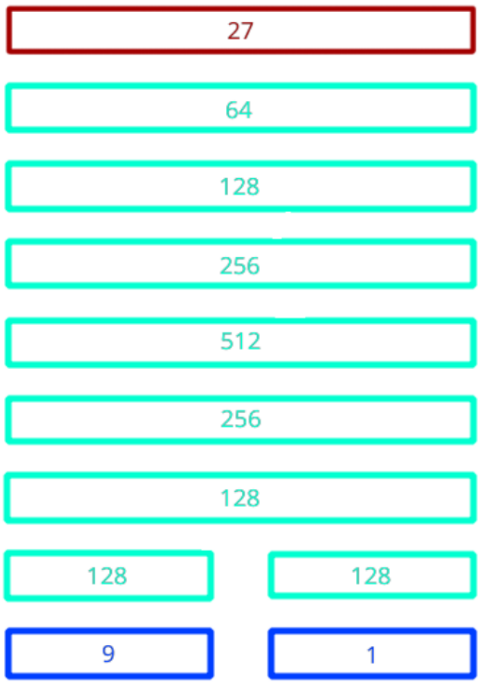



**Abbildung 6: Komplexes Netz**

## 3.3 Erfahrungen

Im Verlauf des Projekts gab es einige Dinge, durch deren Anpassung deutliche Verbesserungen erreicht wurden. Bei den in diesem Abschnitt verwendeten Abbildungen sind rote Schichten Eingabeschichten, cyanfarbene Schichten innere Schichten und blaue Schichten Ausgabeschichten.

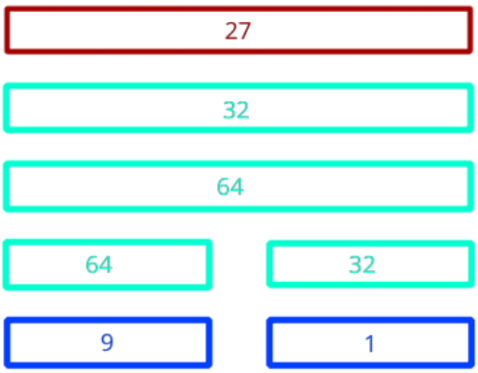



**Abbildung 7: Einfaches Netz**

*3.3.1 Gesamtkomplexität des Netzes.* Ein relativ komplexes Netz (8 Schichten, jeweils ca. 300 Knoten) lernte schnell, ungültige Züge zu vermeiden, wenn als Eingabevektor die Information übergeben wurde, ob ein Feld leer (0) oder belegt (1) ist. Wurde zum Eingabevektor zusätzlich die Information hinzugefügt, welcher Spieler auf ein Feld gesetzt hatte, war das Netz nicht mehr in der Lage, ungültige Züge zu vermeiden. Durch das Verringern der Komplexität (Reduktion auf 3 Schichten, jeweils ca. 50 Knoten) konnten deutlich bessere Ergebnisse erzielt werden.

*3.3.2 Eingaben aufteilen und in verschiedenen Schichten verwenden.* Durch das Aufteilen des Eingabevektors in zwei Teilvektoren, die an verschiedenen Stellen in das Netz gegeben werden, wurden die Ergebnisse deutlich verbessert. Dabei wurde der aus 27 Elementen bestehende Eingabevektor (9 für leere Felder, 9 für Spieler 1, 9 für Spieler 2) in einen Eingabevektor für die leeren Felder und einen Eingabevektor für die Spieler aufgeteilt:

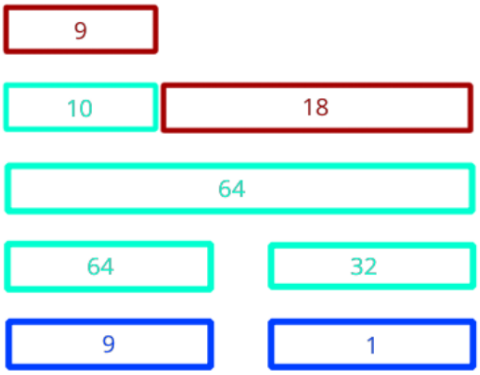



**Abbildung 8: Netz mit aufteteilten Eingabevektor**

*3.3.3 A3C-Ansatz verwerfen.* Aufgrund des Fokus auf die Implementierung des A3C-Algorithmus am Anfang der Studienarbeit, basierte das Layout des Netzes auch auf dem in A3C vorgeschlagenen Layout. Es gab also eine Ausgabe für den Wert und eine Ausgabe für die zu wählende Aktion. Da der Wert allerdings nicht verwendet wurde, konnte er entfernt werden, was zu einer Verbesserung des Netzes führte.

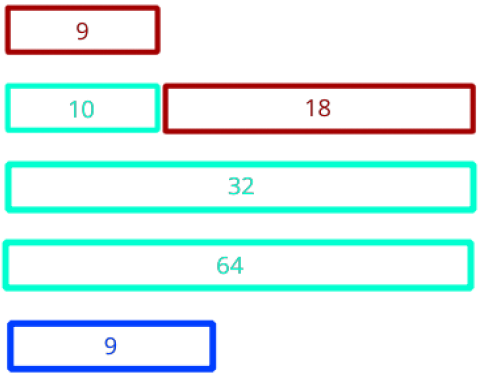



**Abbildung 9: Netz nach Entfernen des Value-Teil**

*3.3.4 Verhalten des Netzes bei verschiedenen Gegnern.* Es folgen einige Statistiken zu den Ausgängen von jeweils 1000 Spielen. Jede Statistik enthält 10 solcher Datenpunkte.

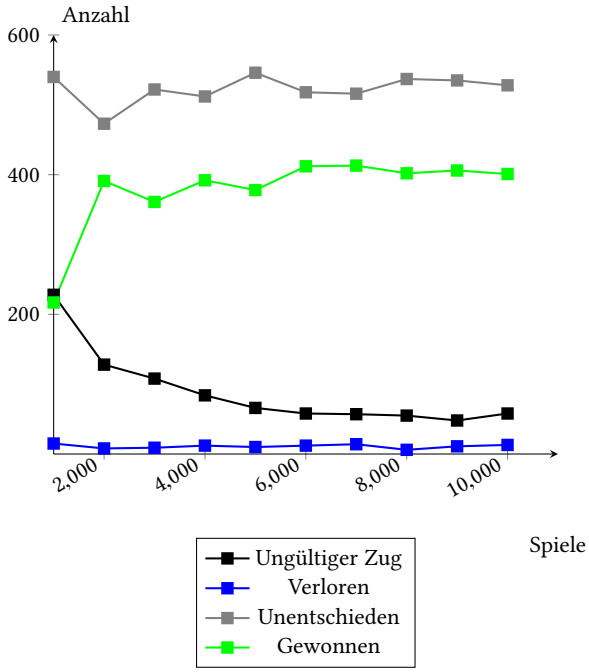



**Abbildung 10: Zufälliger Gegner**

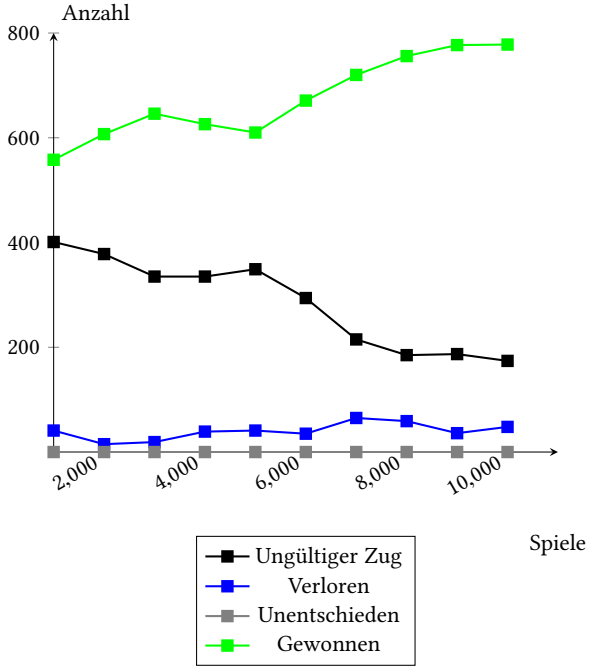



**Abbildung 11: Einfacher Gegner**

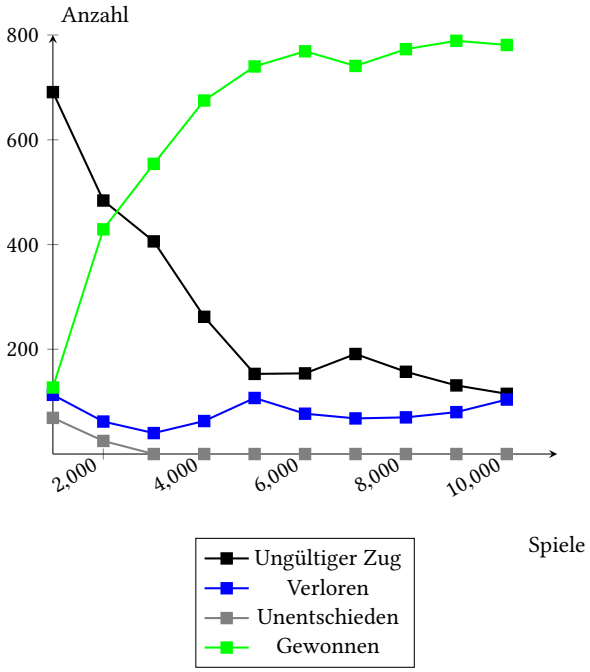



**Abbildung 12: Mittlerer Gegner**

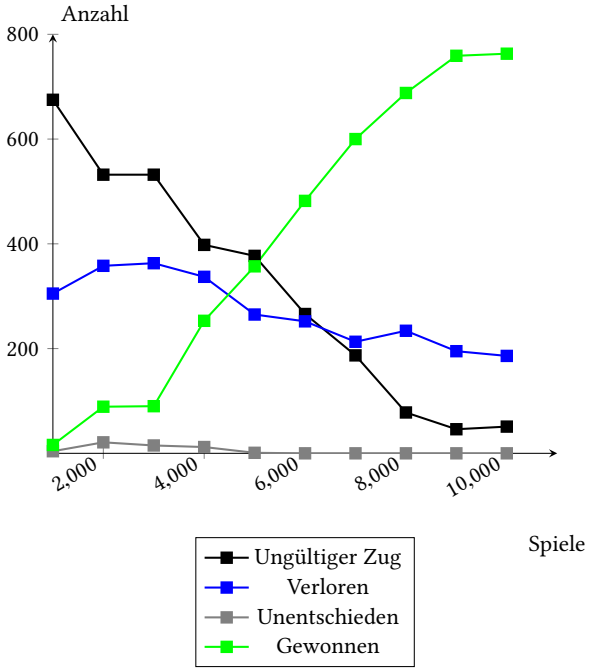



**Abbildung 13: Schwerer Gegner**

Dabei fällt auf, dass das Netz gegen bessere Gegner deutlich bessere Ergebnisse erreicht. Außerdem ist auffällig, dass die Anzahl unentschiedener Spiele gegen einen zufälligen Gegner im Gegensatz zu allen anderen Gegnern sehr hoch ist. Das hängt damit zusammen, dass ein Spiel, bei dem der Gegner durch einen ungültigen Zug verloren hat, als "unentschieden" gewertet wird.

## 4 AUSBLICK

*4.0.1 Bessere Gegenspieler.* In der aktuellen Version kann zwischen vier verschiedenen Gegnern gewählt werden. Die detailierten Charakteristiken der Gegner sind im Abschnitt 3.2 beschrieben. Es wäre möglich, einen weiteren Gegner zu entwickeln, der (z.B. mit Hilfe des MiniMax-Algorithmus [8]) in jeder Situation den bestmöglichen Zug wählt. Da es bei TicTacToe nicht möglich ist, gegen einen optimal spielenden Gegner zu gewinnen, kann das Netz im besten Fall unentschieden spielen. Anderenfalls verliert das Netz, ggf. durch einen ungültigen Zug.

Man könnte untersuchen, ob sich das Netz gegen diesen Gegner verbessert oder ob es nicht dazu in der Lage ist. In diesem Fall könnte man das Layout des Netzes überarbeiten, um möglichst gute Ergebnisse gegen diesen idealen Gegner zu erzielen. Im Anschluss könnte man untersuchen, wie sich das Netz gegen einfachere Gegner verhält und ob aus guten Ergebnissen gegen einen optimalen Gegner auch gute Ergebnisse gegen schlechtere Gegner folgen.

*4.0.2 Netz gegen sich selbst trainieren.* Anstelle der bisher verwendeten Computergegner könnte man das Netz gegen sich selbst spielen lassen und mit den dadurch gewonnenen Erfahrungen trainieren. Analog zum vorherigen Abschnitt 4.0.1 "Bessere Gegenspieler" wäre darauf hin untersuchbar, wie sich das Netz gegen andere Gegner verhält.

*4.0.3 Editor für Layout des Netzes.* Zur verbesserten Demonstration des Einflusses des Layouts eines Netzes auf die erzielten Ergebnisse könnte man das Programm um einen Editor ergänzen, mit dem das Layout neu erstellter Netze grafisch festgelegt werden kann. Außerdem könnte man damit das Layout geladener Netze anzeigen. Durch diese Erweiterung könnte der Einfluss verschiedener Layouts (vgl. Abschnitt 3.3.1 "Gesamtkomplexität des Netzes" besser dargestellt und untersucht werden.

## 5 LITERATUR

[1] Asynchronous Methods for Deep Reinforcement Learning (https://arxiv.org/pdf/1602.01783.pdf)
[2] An Introduction toMarkov Decision Processes (https://www.cs.rice.edu/~vardi/dag01/givan1.pdf)
[3] Python
[4] PySide2
[5] QT
[6] Keras
[7] TensorFlow
[8] Adversariale Suche für optimales Spiel: DerMinimax-Algorithmus und die Alpha-Beta-Suche (http://www.juliusadorf.com/pub/alphabeta-seminar-paper.pdf)

# Einer KI mittels Reinforcement-Learning beibringen Pong zu spielen

Jennifer Schmidbauer

Hochschule für Angewandte Wissenschaften Hof
95028 Hof
jennifer.schmidbauer@hof-university.de

Haiko Tammler

Hochschule für Angewandte Wissenschaften Hof
95028 Hof
haiko.tammler@hof-university.de

**Abstrakt:** In diesem Beitrag wird die Umsetzung, wie man einer selbst gebauten KI mittels Reinforcement-Learning beibringt, das Retrospiel Pong zu spielen.

Das Ziel ist es, mittels Keras, dem OpenAI Gym und Python Code ein neuronales Netz so zu trainieren, dass es sehr gut Pong spielen kann.

**Schlüsselwörter:** Reinforcement-Learning, Neuronales Netz, Künstliche Intelligenz

## 1 Einleitung

Bei dem Atari Spiel Pong bewegt sich ein Ball auf dem Bildschirm hin und her. Beide Spieler bewegen dabei einen Schläger, welcher den Ball in Richtung des Gegners ablenkt. Ziel ist es, dabei den Ball am gegnerischen Schläger vorbeizutreffen und somit einen Punkt zu erzielen. Ein Spiel besteht aus mehreren Runden. Sieger ist, wer zuerst 21 Runden gewonnen hat. In diesem Fall wird der linke Schläger von einem Computer gesteuert und der Rechte von der programmierten KI.

In diesem Beitrag beschreiben wir die theoretische sowie die technische Umsetzung.

## 2 Theoretische Erklärungen

Im den folgenden Unterkapiteln wird der Aufbau der Pong-KI und die notwendigen Zwischenschritte genauer erläutert.

### 2.1 Reinforcement-Learning

Das Reinforcement-Learning setzt auf das trail-and-error Prinzip. Das heißt, es versucht Dinge mit positiven Rückkopplungen zu wiederholen sowie welche mit negativen zu unterlassen. [1]

#### 2.1.1 Reinforcement-Learning bei der Pong-KI

Auf die Pong-KI bezogen kann man das bestärkende Lernen wie folgt genauer erläutern. Die KI (Agent) befindet sich in einer Umgebung, dem sogenannten Spielfeld, und besitzt zudem einen internen Zustand. Dieser besteht lediglich aus dem Inhalt des Bildschirms. Aus diesem Zustand berechnet der Agent Wahrscheinlichkeiten, welche mithilfe der zur Verfügung stehenden Aktionen zu einer Belohnung führen werden. Anhand der berechneten Wahrscheinlichkeit wählt er im folgenden Schritt eine Aktion aus. Damit nimmt er Einfluss auf den nächsten Zustand, welchen er wieder beobachten kann. [2]

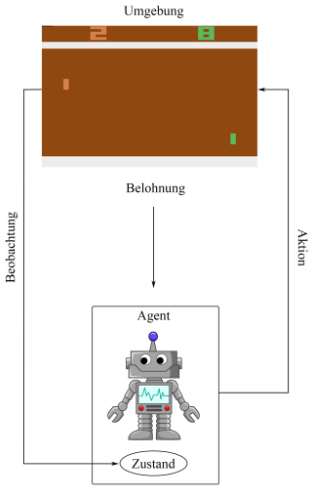



**Abb. 1:** Reinforcement-Learning

### 2.2 Erlernen der Strategie

Damit die Pong-KI lernen kann, benötigt sie eine Strategie. Diese wird als Funktion implementiert, welche jedem Zustand eine bestimmte Wahrscheinlichkeitsverteilung zuweist. Dies wird zumeist mithilfe einer Look-up-Table realisiert.

Das Ziel hierbei soll sein, dass die Summe der Belohnungen, welche der Agent nach jeder ausgeführten Aktion bekommt, möglichst hoch ist. [2]

Im Folgenden werden die einzelnen Schritte genauer erläutert. Zudem wird beschrieben, wie diese zu einem passenden Trainingsalgorithmus zusammengefügt werden.

### 2.2.1 Differenz des Bildschirminhaltes

Die Veränderung des Zustandes wird durch die Beobachtung des gesamten Bildschirms wahrgenommen. Damit der Agent herausfindet wie sich der aktuelle Bildschirminhalt verändert hat, bildet er die Differenz zwischen dem vorangegangenen sowie dem aktuellen Bildschirminhalt. Somit können die Unterschiede, wie sich beispielsweise der Ball oder der gegnerische Schläger verändert hat, erkannt werden. Um dies für den Agenten einfacher zu gestalten wird der eigentliche Spielfeldinhalt ausgeschnitten und die Auflösung verringert. Zudem wird der Hintergrund auf schwarz und die Farbe aller Objekte auf weiß gesetzt.

Anschließend wird der aktuelle Spielfeldinhalt von dem vorherigen Spielfeldinhalt abgezogen. Dieses Bild dient dem neuronalen Netz anschießend als aktueller Zustand. [2]

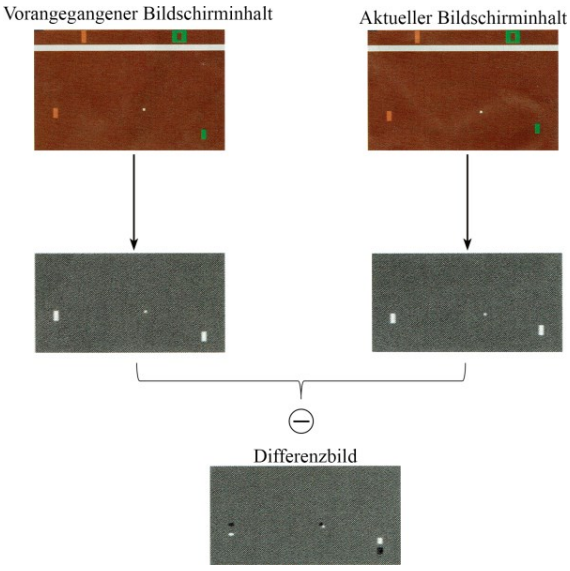



**Abb. 2:** Veranschaulichung der Differenzbilderstellung

### 2.2.2 Aktionismus

Der Agent hat in allen Zuständen die Möglichkeit aus drei Aktionen auszuwählen. Diese wären: „Schläger nach oben“, „Schläger nach unten“ und „Schläger nicht bewegen“.

Wird eine Aktion ausgeführt, welche zu einem Punkt geführt hat, erhält der Agent als Belohnung 1,0. Hat der letzte Schritt jedoch zu einem gegnerischeren Punkt geführt wird eine negative Belohnung von -1,0 vergeben. Diese wird als Bestrafung eingeordnet. Ist in der letzten Aktion nichts passiert erhält er 0,0 - welche als neutral betrachtet werden kann. [2]

### 2.2.3 Lookup-Table (ohne neuronales Netz)

In einer sogenannten Lookup-Table wird für jeden Zustand, jede mögliche Aktion mit einer dazugehörigen Wahrscheinlichkeit abgespeichert. Überträgt man dies auf das Pong Spiel, befinden sich in der Tabelle nun zu jedem möglichen Differenzbild drei Wahrscheinlichkeitswerte. Während eines Spielzuges berechnet die KI das Differenzbild und schaut die Wahrscheinlichkeiten in der Tabelle nach.

Auf Grundlage dieser Werte wird im nächsten Schritt nun eine zufällige Aktion, auch Sampling genannt, ausgeführt. Beträgt die Wahrscheinlichkeit, dass er den Schläger nach oben bewegen soll beispielsweise lediglich 5%, würde er nur in 5 von 100 Fällen die Aktion „Schläger nach oben“ ausführen. Nachdem eine Aktion festgelegt wurde geht das Spiel anschließend wieder in den nächsten Zustand über. [2]

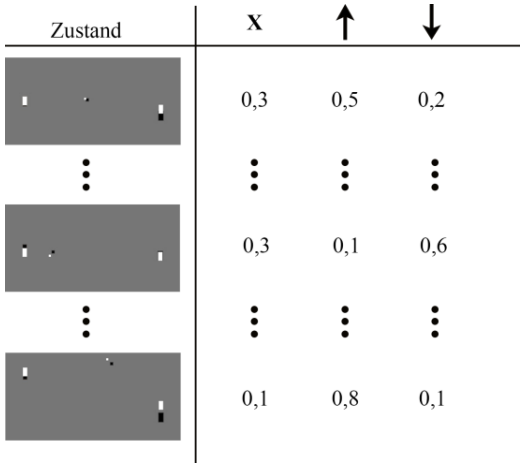



**Abb. 3:** Veranschaulichung einer Lookup-Table

### 2.2.4 Trainingsalgorithmus und Lernprozess

*Anm.: Als Referenz für den folgenden Abschnitt diente, wenn nicht anders angegeben, hauptsächlich [2].*

Um eine effektive Strategie für unsere KI zu entwickeln ist es sinnvoll den Algorithmus so anzupassen, dass am Ende keine Belohnung im Wert von 0,0 erreicht wird. Dies schafft man, indem alle Aktionen, welche in einer Runde ausgeführt wurden, berücksichtigt werden.

Dafür wird von der KI eine Aktion, welche sich an den Wahrscheinlichkeiten aus der Lookup-Table orientiert, ausgeführt. Anschließend merkt sie sich, welche Aktion ausgeführt wurde und in welchem Zustand sie sich befand. Dieser Vorgang wird so oft wiederholt, bis eine Runde vorbei ist. Folgend bekommt die KI eine Belohnung, welche wiederspiegelt ob die Runde gewonnen oder verloren wurde.

Bei einem Sieg, wird aufgrund des Erfolgs, die Wahrscheinlichkeit für alle ausgeführten Aktionen und Zustände erhöht. Wurde die Runde jedoch verloren, wird die Wahrscheinlichkeit verringert. Bei einem Punktgewinn /-verlust wird zusätzlich überprüft ob eine Aktion viele Ballwechsel hatte. Aufgrund der Tatsache, dass Aktionen zu Beginn des Ballwechsels weiniger mit dem Sieg zu tun haben als die zuletzt durchgeführten Aktionen, werden die Wahrscheinlichkeiten von Aktionen (welche weiter in der Vergangenheit liegen) vom Trainingsalgorithmus weniger stark angepasst (exponentiell mit der Basis 0,99).

Ein Nachteil der Lookup-Table ist jedoch, dass es sehr viele Zustände gibt, wovon bei allen die Wahrscheinlichkeiten explizit abgespeichert werden müssen. Insbesondere der Arbeitsspeicher unterliegt hierbei einem sehr hohen Ressourcenverbrauch. Aufgrund dessen ist es sinnvoll eine Funktion zu implementieren, welche einen Zustand als Argument entgegennimmt und die Wahrscheinlichkeit der Aktionen als Ausgabe zurückgibt.

Hierfür eignet sich als Funktion ein neuronales Netz. Dort kann der aktuelle Zustand in das Netz einfließen und die Ausgabeneuronen repräsentieren - die direkten Wahrscheinlichkeiten. Damit die Wahrscheinlichkeiten immer abgeändert werden, passt der Algorithmus die Gewichte der Neuronen im Netzwerk an.

Die KI wählt nun während eines Spiels Aktionen aus, welche abhängig vom aktuellen Zustand sind. Sobald ein Spiel vorbei ist, trainiert der Algorithmus das Netzwerk mit den gewonnenen Informationen. Dafür verwendet er die, nach jeder Aktion zwischengespeicherten, Informationen wie „aktueller Zustand“, „Wahrscheinlichkeitsverteilung der Aktion“ sowie die dazugehörige Belohnung. Dieser Vorgang wird so lange wiederholt bis man das Programm beendet.

Damit der Optimierungsalgorithmus die aktuelle Wahrscheinlichkeitsverteilung entsprechend anpassen kann, um in der nächsten Runde eine Aktion mehr oder weniger wahrscheinlich zu machen, zieht der Code die vom neuronalen Netzwerk berechnete Wahrscheinlichkeitsverteilung von einem One-Hot-Vektor ab. Dadurch entsteht der sogenannte Aktionsgarident. Dieser zeigt nun dem Optimierungsalgorithmus, wie die Netzwerkausgabe entsprechend angepasst werden muss.

Da eine KI auch nach einem langen Training nicht deutlich besser werden würde, sollten auch länger zurückliegende Aktionen häufiger gewählt werden. Das heißt, wenn eine Aktion zu einem Punktgewinn führte, sorgt eine Funktion für die richtige Zuweisung der Belohnungen. Diese addiert am Ende jeder Runde die entsprechenden Belohnungen in einer leicht abgeschwächten Form auf frühere Aktionen. Ebenso werden die statischen Eigenschaften der Belohnungen angepasst, indem der Durchschnitt abgezogen wird und alle Werte durch die Standardabweichung dividiert werden.

## 3 Technische Umsetzung

Im Folgenden wird die Technische Umsetzung des neuronalen Netzes genauer erläutert.

### 3.1 Installation

Für die Implementierung der Keras Bibliothek (siehe 3.2.2) wurde der Schnittstellengenerator Swig implementiert.

Zudem wurde OpenAI Gym, Keras, numpy und Wheel benötigt. Diese wurde mit dem Python „package management system“ PIP installiert.

### 3.2 Verwendete Bibliotheken

Für die Umsetzung wurden mehrere Bibliotheken verwendet. Die zwei Hauptbibliotheken und deren Verwendung werden im Folgenden etwas genauer beschrieben.

#### 3.2.1 OpenAI Gym

OpenAI Gym ist speziell zum Entwickeln und Vergleichen von Algorithmen für verstärkendes Lernen entwickelt und bringt zudem auch Standardprobleme des verstärkenden Lernens mit. [3]

Damit das Atari-Spiel Pong auf einem PC ausgeführt werden kann benötigt man Emulatoren. Die Python-Bibliothek „OpenAI Gym“ interagiert problemlos mit einem Atari-Emulator und ermöglicht es, ohne großen Aufwand, Pong auf dem PC auszuführen.

### 3.2.2 Keras

Die Bibliothek Keras ermöglicht eine schnelle Implementierung neuronaler Netzwerke für Anwendungen des Deep Learnings. Diese bietet uns, mit wenigen Zeilen Code, ein neuronales Netz für die Strategie und die Möglichkeit, unsere KI ohne Umwege zu trainieren. [2]

### 3.2.3 numpy

Numpy bietet mathematische Funktionen an, um mit Matrizen und Arrays zu arbeiten und diese auch berechnen zu können.

## 3.3 Programmcode

In den folgenden Unterkapiteln wird der jeweilige Programmcode, welcher zur Umsetzung des Projektes dient, genauer erläutert.
Für die Umsetzung wurde die Programmiersprache Python sowie die Entwicklungsumgebung Jupyter verwendet.
Im vorletzten Unterkapitel (siehe 3.3.2) befindet sich eine kurze Erläuterung, wie der Phyten Code in der Konsole ausgeführt werden kann.

### 3.3.1 Programmablauf in Jupyter

Bevor wir den Code in Jupyter ausführen können, müssen zuvor alle notwendigen Installationen getätigt worden sein (siehe 3.1).

Wenn unsere KI trainieren soll, muss die Datei „train_pong“ geöffnet werden. Anschließend besteht die Möglichkeit ein gewünschtes Gewicht auszuwählen. Das Training baut auf den bisher erreichten Trainingsfortschritt der entsprechenden Datei auf.
Zur Auswahl stehen folgende, bereits vortrainierte, Gewichte:
- trained_weights_new.h5 (im Trainingsmodus)
- trained_weights_start.h5
- trained_weights_middle.h5
- trained_weights_advanced.h5
- trained_weights_complete.h5

Zudem besteht die Möglichkeit den Spielbildschirm nicht anzuzeigen. Dazu kann die Variable „render“ auf „False“ gesetzt werden. Dies hat den Vorteil, dass die KI schneller trainiert wird.

Um mit der trainierten KI spielen zu können, muss die Datei „play_pong“ ausgeführt werden. Hier kann ein Spiel mit einer Gewichtsdatei in Echtzeit mit „Python“ ausgeführt werden. Dabei findet kein Lernen des Agenten mehr statt.
Zudem enthält „play_pong“ ein zweites Argument, mit dem jeder Frame als eigenständiges Bild (.png) in den Ordner „media“ abgelegt werden kann. Hierbei ist jedoch auf die Anzahl der Bilder zu achten, da recht schnell ein hoher Speicherbedarf entstehen kann. Ist dies gewünscht, muss der Parameter „imageExport“ auf „True“ gesetzt werden.

#### 3.3.1.1 train_pong

```
import numpy as np
import sys
from os.path import isfile
import gym
import keras
from keras.layers import Dense
from keras.layers.core import Flatten

# Trainingsgewichte laden bzw. anlegen
path = '__path__/pong/trained_weights/'

#Es können verschiedene Trainingsgewichte ausgewählt werden

#new: Ist mit zufälligen Inhalten und initialisierten Gewichten
erstellt, welche zum Trainieren der anderen, trainierten Gewichte
verwendet wurden
filename = path + 'trained_weights_new.h5'
#start: Wenig gelernte Spiele vorhanden
#filename = path + 'trained_weights_start.h5'
#middle: ca. 1.200 Spiele angelernt
#filename = path + 'trained_weights_middle.h5'
#advanced: ca. 20.000 Spiele angelernt
#filename = path + 'trained_weights_advanced.h5'
#complete: ca. 32.000 Spiele angelernt
#filename = path + 'trained_weights_complete.h5'
#logfile laden bzw. anlegen
filehandler = open(filename + '.log', 'a')


# Legt den Wert der Datenformatkonvention fest
# Bilddaten werden in einem dreidimensionalen Array dargestellt,
wobei der letzte Kanal die Farbkanäle darstellt
keras.backend.set_image_data_format('channels_last')

# Spielanimation anzeigen bzw. ausblenden
# env.render()
render = True


def preprocess(image):

# Bild in das richtige Format bringen
# Einige Informationen wie Punktestand, hohe Auflösung und
Farben haben für den KI-Agenten keine Bedeutung und werden
daher ausgeblendet
# Die aus „Gym reset()“ gelieferten Bildschirminhalte bieten
verschiedene Bearbeitungsmethoden

    # Schneidet das Spielfeld aus und nimmt eine Unterabtastung
    von 2 vor
    image = image[35:195:2, ::2, 0]

    # Hintergrund auf "schwarz" setzen
     image[np.logical_or(image == 144, image == 109)] = 0
```

```
    # Paddel und Ball auf "weiß" stellen
    image[image != 0] = 1

    # Das Differenzbild dient dem neuronalen Netz als aktuellen
    Zustand und ist auch die Information, die während des
    Spielens und Trainierens in das Netz gefüttert wird
    return image.astype(np.float)


def propagate_rewards(r, gamma=0.99):

    # Belohnungen zuteilen – ebenso auch rückwirkend auf
    Aktionen, bei denen keine Belohnung vergeben wurde
    # Belohnungen für frühere Aktionen werden dabei
    abgeschwächt

    discounted_r = np.zeros_like(r)
    running_add = 0

    for t in range(r.size - 1, 0, -1):
        if r[t] != 0: running_add = 0
        running_add = running_add * gamma + r[t]
        discounted_r[t] = running_add
    discounted_r -= np.mean(discounted_r)
    discounted_r /= np.std(discounted_r)

    return discounted_r

# Arrays anlegen - diese werden als Trainingsdaten benötigt
states, action_prob_grads, rewards, action_probs = [], [], [], []
reward_sum = 0
episode_number = 0

# OpenAI Gym Umgebung setzen
# reset(): Setzt die mit make() geladene Umgebung zurück.
Zudem liefert reset() eine aktuelle Beobachtung der Umgebung
sowie den Bildschirminhalt bevor die Partie losgeht
env = gym.make("Pong-v0")
obs = env.reset()

# preprocess(): Bild in das richtige Format bringen
previous_obs = preprocess(obs)


# Keras neuronales Netzwerk anlegen
# Eingabeschicht, mittlere Schicht und Ausgabeschicht:
Alle Neuronen einer Schicht sind mit allen Neuronen der nächsten
Schicht verbunden

# Sequential(): erzeugt ein neues Modell und teilt Keras mit, dass
diese einen einfachen Aufbau benutzt
model = keras.models.Sequential()

# Add() fügt eine Schicht zum neuronalen Netzwerk hinzu
# Die drei folgenden Add()-Befehle definieren das gesamte
neuronale Netz

# Eingabeschicht:
# Flatten() nimmt das vorbereitete Differenzbild und wandelt es in
einen eindimensionalen Vektor um, da für die folgenden
Schichten ein zweidimensionales Differenzbild nicht nötig ist
model.add(Flatten(input_shape=((80, 80, 1))))

# mittlere Schicht:
# Dense() erzeugt eine Schicht, in der alle Neuronen über
trainierbare Gewichte mit allen Neuronen der Eingabeschicht
verbunden sind. 512 bestimmt hierbei die Anzahl der Neuronen in
dieser Schicht
# „activation“ gibt an, welche Funktion (Aktivierungsfunktion)
auf die Ausgaben der Neuronen angewendet wird, bevor diese
Aktivierung an die nächste Schicht weitergeleitet wird
# relu (Rectified Liner Unit) Standardfunktion für schnelle
Berechnungen
# kernel_initializer bestimmt wie Keras die Anfangswerte der
trainierbaren Gewichte wählt
# „glorot_normal“: Alle Initialisierungsfunktionen belegen die
Parameter mit Zufallszahlen, streuen diese aber unterschiedlich.
"glorot_normal" streut entsprechend einer Normalverteilung und
passt die Breite der Ein- und Ausgabeschicht an
model.add(Dense(512, activation='relu',
kernel_initializer='glorot_normal'))

# Ausgabeschicht:
# Als Ausgabeschicht folgt eine Schicht mit sechs Neuronen,
welche denen von OpenAi Gym vorgegeben sechs Aktions-
möglichkeiten für Pong entsprechen
# activation='softmax': Führt dazu, dass sich die Ausgaben aller
sechs Neuronen zu „1“ summieren. Dadurch kann die Ausgabe
eines Neurons direkt als Wahrscheinlichkeit für eine bestimmte
Aktion interpretiert werden
model.add(Dense(6, activation='softmax'))

# Damit Keras es trainieren kann, muss es kompiliert werden
model.compile(loss='categorical_crossentropy', optimizer='adam')


# Gibt den Aufbau des neuronalen Netzes aus
# Zeigt an, wie viele Parameter für jede Schicht und das gesamte
Netzwerk trainiert werden müssen
# Für dieses Netz: ca. 3.000.000 Parameter
model.summary()

# Laden der trainierten Gewichten des letzten Laufs (falls
vorhanden)
if isfile(filename):
    model.load_weights(filename)


reward_sums = []

while True:
    if render:
        env.render()

    # preprocess(): Bild in das richtige Format bringen
    current_obs = preprocess(obs)
```

```
# Differenz zwischen dem letzten und dem aktuellen, vorverarbeiteten Frame – wird zum aktuellem Status des Agenten
state = current_obs - previous_obs
previous_obs = current_obs

# Prognostiziert die Wahrscheinlichkeiten anhand des Keras-Modells und der Beispielaktion
action_prob = model.predict_on_batch(state.reshape(1, 80, 80, 1))[0, :]
action = np.random.choice(6, p=action_prob)

# Ausführen einer Aktion in der Umgebung (Spiel)
# step(): Führt eine mögliche Aktion aus (Schläger rauf bzw. runter)
# reward: Gibt für diese Aktion die erhaltene Belohnung zurück
# done: Gibt an ob die Episode (Spiel) fertig ist (Spieler hat 21 Punkte)
# obs: die neue Beobachtung (Framebild)
obs, reward, done, info = env.step(action)
reward_sum += reward


# Trainieren des Modells

# Ermittelte Daten in das zuvor erstelle Arrays speichern
# Größere Datenmengen sind zum Trainieren effizienter, deshalb wird eine ganze Episode gespielt (d.h. einer der Spieler hat 21 Punkte) sowie nach jeder Runde die Werte in ein Array gespeichert.
states.append(state)
action_probs.append(action_prob)
rewards.append(reward)

# Gradienten der Aktionswahrscheinlichkeiten
# Diese Gradienten zeigen dem Optimierungsalgorithmus wie die Netzwerkausgabe angepasst werden muss, um höhere Wahrscheinlichkeiten mit Hilfe der Aktionen zu erreichen.
# „6“ => sechs Neuronen in der Ausgabeschicht
y = np.zeros(6)
y[action] = 1
action_prob_grads.append(y - action_prob)

if done:

    # Sobald ein Spiel beendet ist (d.h. einer der Spieler hat 21 Punkte) wird das Netz trainiert
    episode_number += 1

    # Aus den letzten 40 Belohnungssummen wird die mittlere Belohnungssumme berechnet
        reward_sums.append(reward_sum)
        if len(reward_sums) > 40:
            reward_sums.pop(0)

    # Aktuelle Leistung des Agenten in der Konsole ausgeben…
    s = 'Anzahl der Spiel-Episoden: %d Belohnung: %f , Durchschnitt: %f' % (
        episode_number, reward_sum, np.mean(reward_sums))
    print(s)

        # ... und in das Logfile schreiben
    filehandler.write(s + '\n')
    filehandler.flush()

    # Belohnungen zuteilen – ebenso auch rückwirkend auf Aktionen, bei denen keine Belohnung vergeben wurde
    # Belohnungen für frühere Aktionen werden dabei abgeschwächt
    rewards = np.vstack(rewards)
    action_prob_grads = np.vstack(action_prob_grads)

    # Belohnungen zuteilen
    rewards = propagate_rewards(rewards)


    # Keras trainiert das Netzwerk

    # Die Ausgaben sind die, anhand der erhaltenen Belohnungen angepassten, Aktionswahrscheinlichkeiten
    # Lernrate: eine zu  aggressive Änderung der Wahrscheinlichkeit kann zu Instabilitäten während des Trainings führen. Daher ist eine geringe Lernrate, von beispielsweise „0.1“, empfehlenswert
    learning_rate = 0.1
    X = np.vstack(states).reshape(-1, 80, 80, 1)
    Y = action_probs + learning_rate * rewards *
    action_prob_grads

    # Keras trainiert das Netzwerk sobald „train_on_batch()“ aufgerufen wird
    model.train_on_batch(X, Y)

    # Aktuelle Gewichte des Modells speichern
    model.save_weights(filename)

    # Alles für das nächste Spiel zurücksetzen
    states, action_prob_grads, rewards, action_probs = [], [], [],[]
    reward_sum = 0
    obs = env.reset()
```

### 3.3.1.2 play_pong

```
import sys
import os
import gym
import numpy as np
import keras
from keras.layers import Dense
from keras.layers.core import Flatten
from time import sleep

# Trainingsgewichte laden bzw. anlegen
path = '__path__/pong/trained_weights/'


#Es können verschiedene Trainingsgewichte ausgewählt werden

#start: Wenig gelernte Spiele vorhanden
filename = path + 'trained_weights_start.h5'
#middle: ca. 1.200 Spiele angelernt
#filename = path + 'trained_weights_middle.h5'
#advanced: ca. 20.000 Spiele angelernt
#filename = path + 'trained_weights_advanced.h5'
#complete: ca. 32.000 Spiele angelernt
#filename = path + 'trained_weights_complete.h5'
# „True“: Einzelbilder werden abgespeichert.
imageExport = False

# Legt den Wert der Datenformatkonvention fest
# Bilddaten werden in einem dreidimensionalen Array dargestellt,
wobei der letzte Kanal die Farbkanäle darstellt
keras.backend.set_image_data_format('channels_last')


def preprocess(image):

# Bild in das richtige Format bringen
# Einige Informationen wie Punktestand, hohe Auflösung und
Farben haben für den KI-Agenten keine Bedeutung und werden
daher ausgeblendet
# Die aus „Gym reset()“ gelieferten Bildschirminhalte bieten
verschiedene Bearbeitungsmethoden

    # Schneidet das Spielfeld aus und nimmt eine Unterabtastung
    von 2 vor
    image = image[35:195:2, ::2, 0]

    # Hintergrund auf "schwarz" setzen
    image[np.logical_or(image == 144, image == 109)] = 0
    # Paddel und Ball auf "weiß" stellen
    image[image != 0] = 1

    # Das Differenzbild dient dem neuronalen Netz als aktuellen
    Zustand und ist auch die Information, die während des
    Spielens und Trainierens in das Netz gefüttert wird
    return image.astype(np.float)

    # OpenAI Gym Umgebung setzen
    # reset(): Setzt die mit make() geladene Umgebung zurück.
    Zudem liefert reset() eine aktuelle Beobachtung der Umgebung
    sowie den Bildschirminhalt bevor die Partie losgeht
    env = gym.make("Pong-v0")
    obs = env.reset()


    # preprocess(): Bild in das richtige Format bringen
    previous_obs = preprocess(obs)


# Keras neuronales Netzwerk anlegen
# Eingabeschicht, mittlere Schicht und Ausgabeschicht:
Alle Neuronen einer Schicht sind mit allen Neuronen der nächsten
Schicht verbunden

# Sequential(): erzeugt ein neues Modell und teilt Keras mit, dass
dieses einen einfachen Aufbau benutzt
model = keras.models.Sequential()

# Add() fügt eine Schicht zum neuronalen Netzwerk hinzu
# Die drei folgenden Add()-Befehle definieren das gesamte
neuronale Netz

# Eingabeschicht:
# Flatten() nimmt das vorbereitete Differenzbild und wandelt es in
einen eindimensionalen Vektor um, da für die folgenden
Schichten ein zweidimensionales Differenzbild nicht nötig ist
model.add(Flatten(input_shape=((80, 80, 1))))

# mittlere Schicht:
# Dense() erzeugt eine Schicht, in der alle Neuronen über
trainierbare Gewichte mit allen Neuronen der Eingabeschicht
verbunden sind. 512 bestimmt hierbei die Anzahl der Neuronen in
dieser Schicht
# „activation“ gibt an, welche Funktion (Aktivierungsfunktion)
auf die Ausgaben der Neuronen angewendet wird, bevor diese
Aktivierung an die nächste Schicht weitergeleitet wird
# relu (Rectified Liner Unit) Standardfunktion für schnelle
Berechnungen
# kernel_initializer bestimmt wie Keras die Anfangswerte der
trainierbaren Gewichte wählt
# „glorot_normal“: Alle Initialisierungsfunktionen belegen die
Parameter mit Zufallszahlen, streuen diese aber unterschiedlich.
"glorot_normal" streut entsprechend einer Normalverteilung und
passt die Breite der Ein- und Ausgabeschicht an
model.add(Dense(512, activation='relu',
kernel_initializer='glorot_normal'))

# Ausgabeschicht:
# Als Ausgabeschicht folgt eine Schicht mit sechs Neuronen,
welche denen von OpenAi Gym vorgegeben sechs Aktions-
möglichkeiten für Pong entsprechen
# activation='softmax': Führt dazu, dass sich die Ausgaben aller
sechs Neuronen zu „1“ summieren. Dadurch kann die Ausgabe
```

```
eines Neurons direkt als Wahrscheinlichkeit für eine bestimmte
Aktion interpretiert werden
model.add(Dense(6, activation='softmax'))

# Damit Keras es trainieren kann, muss es kompiliert werden
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.summary()

# Laden von trainierten Gewichten
model.load_weights(filename)


i = 0
while True:
    env.render()

    # Speichert den Spielverlauf ggf. in Einzelbildern ab
    if imageExport:
        env.env.ale.saveScreenPNG('media/bild_%06d.png' % (i))
    i += 1
    # preprocess(): Bild in das richtige Format bringen
    current_obs = preprocess(obs)
    # Differenz zwischen dem letzten und dem aktuellen,
     vorverarbeiteten Frame – wird zum aktuellem Status des
    Agenten
    state = current_obs - previous_obs
    previous_obs = current_obs


    # Prognostiziert die Wahrscheinlichkeiten anhand des Keras-
    Modells und der Beispielaktion
    action_prob = model.predict_on_batch(state.reshape(1, 80, 80,
    1))[0, :]
    action = np.random.choice(6, p=action_prob)


    # Ausführen einer Aktion in der Umgebung (Spiel)
    # step(): Führt ein mögliche Aktion aus (Schläger rauf bzw.
    runter)
    # reward: Gibt für diese Aktion die erhaltene Belohnung zurück
    # done: Gibt an ob die Episode (Spiel) fertig ist (Spieler hat 21
    Punkte)
    # obs: die neue Beobachtung (Framebild)
    obs, reward, done, info = env.step(action)


    # Spiel in Echtzeit ablaufen lassen
    sleep(1/60)

    # Ein Spiel ist beendet, Umgebung zurücksetzen
    if done:
        obs = env.reset()
```

### 3.3.2 Ausführung in der Konsole

Damit der Python Code auf der Konsole ausgeführt werden kann, sind die nachfolgend beschriebenen Schritte notwendig. Des Weiteren müssen auch bereits alle notwendigen Installationen (siehe 3.1) getätigt worden sein.

Ist dies erledigt, kann der Agent trainiert werden. Dazu muss die Datei „train_pong ../trained_weights/trained_weights_new.h5“ ausgeführt werden. Wenn die Datei "trained_weights_new.h5" bereits vorhanden ist, wird das Training fortgesetzt. Ansonsten wird ein neuer Trainingslauf gestartet und der Fortschritt wird in einer neuen Gewichtsdatei gespeichert. Die Protokolldatei mit dem Trainingsfortschritt wird in „trained_weights_start.h5.log“ geschrieben.

Soll ein anderes Gewicht weiter trainiert werden, befinden sich im Repository „trained_weights“ hierfür die folgenden Gewichte:
- trained_weights_new.h5 (im Trainingsmodus)
- trained_weights_start.h5
- trained_weights_middle.h5
- trained_weights_advanced.h5
- trained_weights_complete.h5

Wenn der Spielbildschirm nicht anzeigt werden soll, um das Training zu beschleunigen, sollte die Variable “render“ im Skript auf „False“ gesetzt werden.

Um den Agenten verwenden zu können muss die entsprechende Datei mit dem gewünschten Gewicht „python play_pong trained_weights_XYZ.h5“ ausgeführt werden. Hierbei findet kein Lernen des Agenten mehr statt.

Zudem enthält „play_pong“ ein zweites Argument, mit dem jeder Frame als eigenständiges Bild (.png) in den Ordner „media“ abgelegt werden kann. Hierbei ist jedoch auf die Anzahl der Bilder zu achten, da recht schnell ein hoher Speicherbedarf entstehen kann.

## 3.4 Best Practice für den erfolgreichen KI-Einsatz

Im den folgenden Unterkapiteln wird genauer erläutert welche Komponenten verändert werden können und welche Erkenntnisse wir durch die jeweilige Veränderung gewonnen haben.

### 3.4.1 Veränderbare Komponenten

In dem Programmcode gibt es hauptsächlich vier Werte bzw. Eigenschaften welche verändert oder angepasst werden können.

Für die gesamte mittlere Schicht (hidden layer) werden 512 Neuronen verwendet. Die Ausgabeschicht (output layer) wird mit 6 Neuronen (Aktionen Pong) belegt. Hieraus entstehen 3.280.390 programmierbare Trainingsparameter.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| flatten_1 (Flatten) | (None, 6400) | 0 |
| dense_1 (Dense) | (None, 512) | 3277312 |
| dense_2 (Dense) | (None, 6) | 3078 |

Total params: 3,280,390

**Abb. 4:** Übersicht der Trainingsparameter zur jeweiligen Funktion

Eine Erhöhung der Neuronen im „hidden layer“ die dadurch einhergehende Vergrößerung der programmierbaren Trainings-parameter, ergaben nach 25.000 gespielten Runden keine merkliche Verbesserung der Lernkurve.

Des Weiteren könnte man die Anzahl der Aktionen bei der Funktion „model.add(Dense(6, activation='softmax'))“ von „6“ auf „3“ reduzieren, da Pong eigentlich nur drei Aktionen besitzt (siehe 2.2.2). Jedoch gibt es durch die Abbildung des Atari Controllers einige redundante Aktionen, welche auch manuell entfernt werden könnten. Somit würde man das Netz anschließend mit den drei möglichen Aktionen trainieren. Jedoch kann das neuronale Netz die Redundanz selbst herausfinden – was das händische Entfernen nicht nur obsolet macht, sondern auch negative Auswirkungen hat, wie sich bei längeren Tests zeigte.
So ergab eine Verkleinerung der Ausgabeschicht auf drei Neuronen (Aktionen „Schläger nach oben“, „Schläger nach unten“ und „Schläger nicht bewegen“) ohne die redundante Atari Controller Aktionen, eine Verschlechterung der Steuerung der Schläger durch die KI.

Bei Verwendung der oben genannten Parameter (siehe Abb. 4) konnte nach ca. 27.500 Spielrunden keine erhebliche Verbesserung der Lernkurve mehr festgestellt werden. Das trainierte Netz konnte zwar alle folgenden Spiele für sich entscheiden, jedoch mit schwankender Anzahl an Gegentreffern.

### 3.4.2 Alternativen ohne Deep Learning mit Keras

Eine praktikable Alternative um Pong ohne den Einsatz von künstlichen neuronalen Netzwerken anzulernen, war nicht möglich. Die weiter oben beschriebene Lookup-Table (siehe 2.2.3), wurde durch die Anzahl der möglichen Zustände zu groß und konnte schließlich nicht mehr vom Arbeitsspeicher adressiert werden.

### 3.5 Verwendete Keras Parameter

Verwendete Keras Parameter, mit denen das Netzwerk im Test die besten Lernerfolge erzielt hatte.

- relu (Rectified Liner Unit): Standardfunktion für schnelle Berechnungen

- kernel_initializer: Bestimmt wie Keras die Anfangswerte der trainierbaren Gewichte wählt

- glorot_normal: Alle Initialisierungsfunktionen belegen die Parameter mit Zufallszahlen, streuen diese aber unterschiedlich

- "glorot_normal": Streut entsprechend einer Normalverteilung und passt die Breite der Ein- und Ausgabeschicht an

- activation='softmax': Führt dazu, dass die Ausgaben aller sechs Neuronen zu „1“ summiert werden

## 4 Zusammenfassung

Die Pong-KI trainierte auf einem MacBook Pro auf einer Intel Core i5 CPU und einer Intel Iris Plus Graphics 650 Grafikkarte. Hierbei benötigte sie über 120 Stunden um auf die gewünschte Spielanzahl von 32.000 Spielen zu kommen. Bereits bei dieser Spielanzahl gewinnt unsere KI jedes Spiel gegen den Computergegner.
Bei unseren eingebauten Zwischenstufen konnte man deutlich erkennen, wie die KI sich stetig verbesserte. So erreichte sie nach 100 angelernten Spielen noch keinen Rundengewinn gegen den Computergegner. Ab 1.200 Spielen jedoch, war es ihr schon vereinzelt möglich Runden für sich zu entscheiden. Nach etwa 21.000 Spielen konnte man eine relativ ausgewogenen Gewinn- und Verlustrate erkennen, während die KI schlussendlich nach Spiel 27.487 kein weiteres Mal mehr verlor (siehe Abb. 5).

Somit kann man abschließend festhalten, dass die KI ab diesem Moment für den Atari-Computergegner unbezwingbar geworden war. Sie müsste jedoch eine noch deutlich größere Anzahl an Spielen bewältigen, um sogar ungeschlagen (21:0) gegen ihren Computergegner gewinnen zu können.

## Verweise

[1] J. Forchte. Maschinelles Lernen: Grundlagen und Algorithmen in Python, 2. Auflage, S. 331, 2019.
[2] S. Stabinger. Zuckerbrot und Peitsche. c´t, Heft 21, S 166-173, 2018.
[3] „OpenAI Gym – Documentation“ unter: https://gym.openai.com/docs/, Stand: 02.07.2019

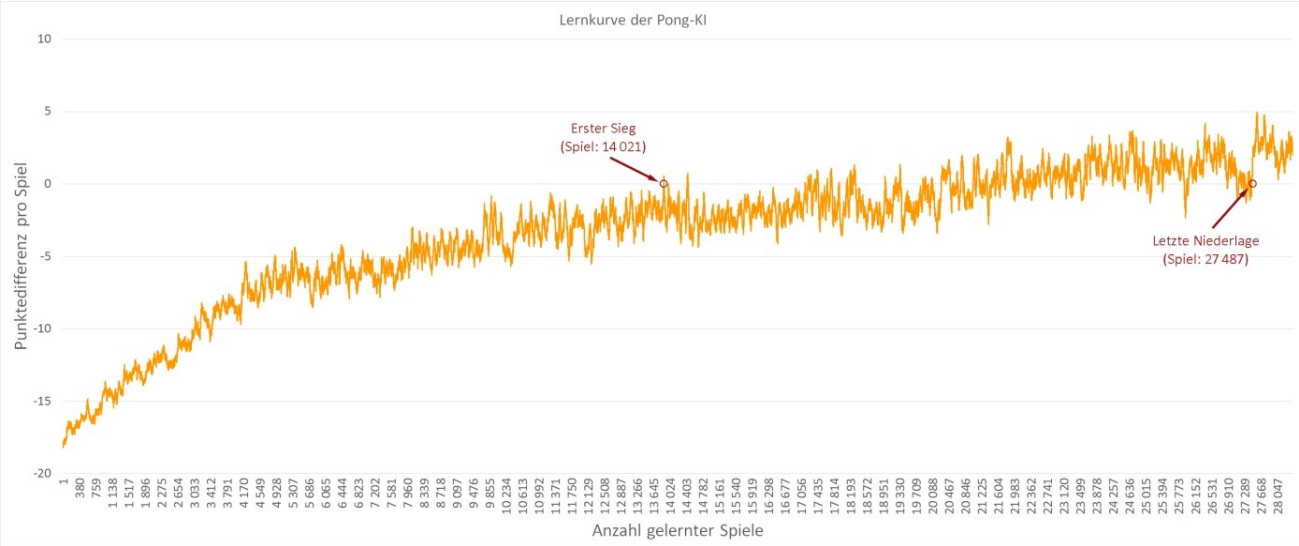



**Abb. 5:** Die Abbildung zeigt jeweils den Durchschnitt der Punktedifferenz pro 40 angelernten Spielen. Positive Werte bedeuten, die angelernte KI hat gewonnen – bei negativen der Computergegner

# Reinforcement Learning mit „Sonic The Hedgehog“

Jonas Schmitt
Hochschule Hof
Hof, Deutschland
jonas.schmitt@hof-university.de

## ZUSAMMENFASSUNG

In der Arbeit wird das Training eines Neuronalen Netzes beschrieben. Dem Netzwerk wird mittels „Reinforcment Learning“ das Spielen von „Sonic The Hedgehog“ beigebracht.

• **Machine Learning approches**;

## KEYWORDS

Neurale Netzwerke, Videospiele, Maschinelles Lernen, Reinforcement Learning

*FWPM: Angewandtes maschinelles Lernen, Sommersemester, 2019, Hof*
Jonas Schmitt.

## 1 EINLEITUNG

Im Jahr 2018 wurde ein Wettbewerb von „OpenAI“ ausgetragen, bei dem es darum ging, das beste Neuronale Netzwerk für das Spiel „Sonic The Hedgehog“ zu trainieren. Dieser Wettbewerb war die Inspiration für diese Arbeit. Bei dem genannten Wettbewerb ging es darum, dass trainierte Agent einige unbekannte Level absolvieren musste. Diese Arbeit wird sich darauf fokosieren, ein bereits im Spiel vorhandenes Level zu absolvieren.

## 2 SONIC THE HEDGEHOG

Wie bereits in der Einleitung erwähnt, geht es darum, ein Level im Spiel „Sonic The Hedgehog“ zu absolvieren. „Sonic“ ist eine Spielereihe von „Sega“, die im Jahr 1991 ihr Debüt mit „Sonic The Hedgehog“ auf dem „Sega Genesis“ feiert (vgl. [9]). In dem Spiel geht das darum, so schnell wie möglich das Ende des Levels zu erreichen. Der namensgebende Charakter „Sonic“ erreicht dabei hohe Geschwindigkeiten und muss Hindernisse, wie Loopings und Abgründe, überwinden.


**Abbildung 1: Sonic The Hedgehog - Genesis - Level 1 (https://www.letsplaysega.com/wp-content/uploads/images/gen/Sonic_the_Hedgehog.png).**

## 3 ZIELE

Der Agent soll komplett selbständig lernen, wie man ein Level in „Sonic The Hedgehog“ absolviert. Dabei gibt keine vortrainierte Daten, das Training wird also ohne Vorwissen gestartet. Wichtig ist außerdem, dass das Netz am Ende des Trainings das Level konstant schafft. Das heißt bei wiederholten Versuchen sollen so viele, wie möglich, erfolgreich abgeschlossen werden.

# 4 REINFORCEMENT LEARNING

## 4.1 Allgemein

(vgl. [1]) Reinforcement Learning ist eine besondere Art des Machinellen Lernens und basiert darauf, dass ein Agent via „trial and error“ aus den eigenen Aktionen lernt. Reinforcement Learning benutzt hierfür Belohnungen, die den Agenten, abhängig von der getätigten Aktion, belohnen. Je besser die Aktion, desto höher die Belohnung. Ziel ist es, die Belohnung zu maximieren. Beim Reinforcement Learning gibt es folgende Elemente:

- Umgebung: Die Welt, in der der Agent operiert
- Status: Der aktuelle Status des Agenten
- Belohnung: Die Belohnung für die aktuelle Aktion
- Strategie: Wie wird der aktuelle Status auf die Aktionen abgebildet

Der allgemeine Ablauf beim Reinforcement Learning wird in Abbildung 2 dargestellt.

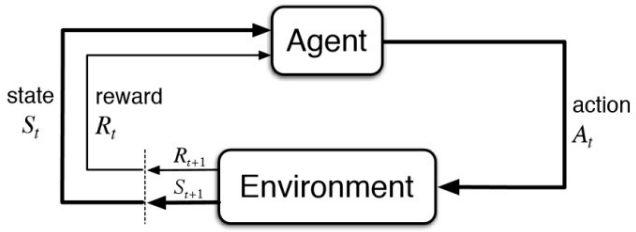



**Abbildung 2: Reinforcement Learning Allgemein (https://www.kdnuggets.com/images/reinforcement-learning-fig1-700.jpg).**

## 4.2 Sonic The Hedgehog

Für „Sonic The Hedgehog“ lassen sich die Elemente folgendermaßen übertragen:

| Element | Bedeutung in „Sonic The Hedgehog“ |
|---|---|
| Umgebung | Gym-Retro mit geladenem Spiel |
| Status | Die aktuellen Spieldaten(Pixeldaten) |
| Belohnung | x-Position des Spielers |
| Stategie | CNNPolicy (gegeben durch stable-baselines) |

# 5 VORGEHENSWEISE

## 5.1 Gym-Retro

(vlg. [7] und [2]) Das Spiel wird über die Schnittstelle „Gym-Retro“ angesteuert. „Gym-Retro“ ist eine Emulatorsoftware, die es ermöglicht eine große Auswahl von alten Spielen anzusteuern. Es emuliert unterschiedliche Konsolen und bietet verschiedene Möglichkeiten mit diesen Spielen, vor allem mittels Maschnienellem Lernen, zu interagieren. Das Spiel wurde im Rahmen dieser Arbeit auf der Plattform „Steam“erworben, da die Spiele nicht direkt in „Gym-Retro“enthalten sind. Einen guter Einstieg zum Verständnis der Umgebung bietet eine Video-Reihe von „Lucas Thompson“ (vgl. [4]), die mittels Beispielen das Arbeiten mit „Gym-Retro“ erklärt. Zu den wichtigsten Funktionen von „Gym-Retro“ gehören:

*Speicheradressen.* Für viele Spiele gibt es eine Datei, in der Speicheradressen für nützliche Werte des Spiels aufgelistet sind. Für „Sonic“ ist zum Beispiel die Speicheradresse „16764936“ als „x“ gelistet. Dieses „x“ gibt die aktuelle Position des Spielers an und kann überall verwendet werden. Es können bei Bedarf auch zusätzliche Speicheradressen hinzugefügt werden.

*Belohungsfunktionen.* Beim „Reinforcement Learning“ wird das Model, abhängig von seiner letzten Aktion, bewertet. Je besser die Aktion, desto mehr wird das Model für seine Aktion belohnt. „Gym-Retro“ bietet verschiedene Möglichkeiten Belohnungen zu definieren. Im Allgemeinen wird die Belohnung in der Datei „Scenario.json“, die zu dem entsprechenden Spiel gehört, festgelegt. In dieser Datei kann auf die oben beschriebenen Speicheradressen zugegriffen werden. Es können also Belohnungsfunktionen abhängig von aktuellen Werten im Spiel abhängig gemacht werden.

*Savestates.* Ein weiteres nützliches Feature sind „Savestates“. Mit „Savestates“ kann das Spiel an unterschiedlichen Stellen gespeichert und wieder geladen werden. So kann zum Beispiel für jedes Level ein „Savestate“ erstellt werden, welcher dann direkt geladen wird. Das Erstellen von „Savestates“ wird mit einem Tool, das direkt mit „Gym-Retro“ mitgeliefert wird, ermöglicht.

*Replays.* Während des Trainings oder Spielens kann zusätzlich eine Funktion zum Mitschnitt des aktuellen Spielgeschehens aktiviert werden. Damit wird eine bessere Auswertung des Trainings ermöglicht, da prinzipiell zu jedem Zeitpunkt des Trainings ein Einblick in das aktuelle Geschehen möglich ist.

## 5.2 Stable Baselines

(vgl. [5]) „Stable Baselines“ basiert auf „Tensorflow“ und bietet eine Sammlung von nützlichen „Reinforcement Learning“ Algorithmen. Ein großer Vorteil ist die Kompatibilität mit „Gym-Retro“. Das heißt, dass die Algorithmen direkt mit der „Gym-Retro“ Umgebung kommunizieren können. Ein weiterer Vorteil ist, dass eine „Tensorboard“-Integration gegeben ist. Damit können die Ergebnisse auf „Tensorboard“ dargestellt und analysiert werden.

## 5.3 PPO2

„PPO2“, kurz für „Proximal Policy Optimization“ (Version 2), ist ein „Reinforcement Learning“ Algorithmus, der sich hervorragend für Videospiele eignet. Der Vorteil von PPO gegenüber anderen „policy gradient methods“ ist die einfache Implementierung des Algorithmus, die Effizienz und die Möglichkeit der Parallelisierung (vgl. [6]). „PPO2“ ist in „stable-baselines“integriert. Die Schöpfer von PPO beschreiben ihren Algorithmus folgendermaßen:

„We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a „surrogate“ objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much

simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.“[6]

# 6 TRAINING

## 6.1 Erstellen der Umgebung

Bevor das Netzwerk trainiert werden kann, muss die Umgebung erstellt werden. Für die Umgebung in „Gym-Retro“ werden folgende Parameter übergeben:

- Der „Savestate“, in dem das Spiel/Training starten soll
- Die Belohnungsfunktion
- Pfad für „Replays“

Außerdem wird die Umgebung um einige Funktionen erweitert, die es standardmäßig nicht gibt:

*Limiter.* Im Fall von Sonic bietet sich an, die Aktion, die das Netzwerk zur Auswahl hat einzuschränken. Normalerweise hat das Netzwerk alle Buttons des „Sega Genesis“-Controllers (vgl. Fig. 2) zur Auswahl. Da für „Sonic“ aber eigentlich nur die Aktionen „Links“, „Rechts“ und „B“ (Springen) wichtig sind, werden alle anderen Aktionen aus dem Pool genommen. Der Code hierfür basiert auf dem Code des Github-Nutzers „unixpickle“ (vgl. [8]).


**Abbildung 3: Sega Genesis Controller (https://retropie.org.uk/forum/topic/11647/configure-hotkey-sega-genesis-gamepad-6-buttons).**

*Belohnungen skalieren.* Der Gleiche Benutzer hat herausgefunden, dass sich beim Training mit „PPO2“ kleinere Belohnungen (relativ gesehen) positiv auf die Geschwindigkeit des Trainings auswirken. Daher wird die „RewardScaler“-Funktion eingebaut, die die Belohnungen, die der Agent erhält, nach unten skaliert. Der Code hierfür basiert auf dem gleichen Github-Repository (vlg. [8]).

*Logger.* Die Umgebung wird um eine weitere Funktion erweitert. Ein Logger für die Belohnungen ist sehr nützlich, um die Ergebnisse auszuwerten. Hierfür wird einfach bei jedem Schritt des Netzwerkes die erhaltene Belohnung in eine Datei geschrieben. Außerdem wird der Agent zurückgesetzt, falls er zu lange an der gleichen Stelle hängen bleibt.

## 6.2 Trainings-Loop

Vor Beginn des Trainings muss die Umgebung erstellt werden (vgl. Kapitel 6). Dann muss das Model erstellt und mit den entsprechenden Parametern vorbereitet werden. Diesem Model wird dann die Umgebung zugewiesen. Außerdem wird eine „Callback“-Funktion definiert, die nach jedem Schritt aufgerufen wird und den Fortschritt überwacht. Sobald in den letzten 10 Versuchen bei mindestens 8 Versuchen das Ende des Levels erreicht wird, wird das Training abgeschlossen und gilt als Erfolg. Außerdem wird immer das beste Models gespeichert. Während des Trainings wird das Spiel nicht auf dem Bildschirm dargestellt, um Rechenleistung zu sparen (die Möglichkeit gibt es aber prinzipiell).

## 6.3 Verwendete Hardware

Für das Training des Netzwerks wird folgende Hardware verwendet:

- Prozessor: AMD Ryzen 2600X mit 6 Kernen/12 Threads
- Grafikkarte: NVIDIA Geforce GTX 1070
- Arbeitsspeicher: 16GB DDR4

# 7 ERGEBNISSE

Nach einer ganzen Nacht Training und über 250 gespielten Runden wurde das Training des Models erfolgreich beendet, da das Abschlusskriterium erreicht wurde. Im Laufe des Trainings ist aufgefallen, dass der Erfolg nicht konstant ist, sonder eher stufenweise. Das heißt, „Sonic“ bleibt immer wieder an schwierigen Stellen hängen und überwindet diese teilweise erst deutlich später. Vor Allem der Looping des ersten Levels hat hier große Probleme bereitet (vgl. Abbilung 4).


**Abbildung 4: Looping im ersten Level von Sonic**

## 7.1 Die ersten Runden

In den ersten Runden fällt auf, dass viele Aktionen noch sehr zufällig stattfinden und „Sonic“ auch nicht wirklich von der Stelle kommt. Sobald eine kleine Erhöhung, über die Sonic springen muss, kommt, ist der Agent zunächst nicht in der Lage, dieses Problem zu lösen. Ein Video mit den ersten Versuchen des Agenten ist hier zu finden:

https://youtu.be/0QRBSy52ywQ

### 7.2 Endergebnis

Wenn man das Model nach dem Training ausprobiert, spielt das Netzwerk ohne Probleme das erste Level komplett durch. Hierbei fällt auf, dass auch die Geschwindigkeit, mit der Sonic durch das Level kommt sehr hoch ist und locker mit einem Menschen mithalten kann. Ein Test des fertig trainierten Netzwerks ist unter folgendem Link zu finden:

https://youtu.be/ewEusin7otU

## 8 AUSWERTUNG

### 8.1 Heatmap

Da jedes Spiel während des Trainings aufgezeichnet wird, kann im Zuge der Auswertung eine Heatmap erstellt werden, die anzeigt, an welchen Stellen Sonic am meisten gewesen ist. Das Tool zur Auswertung der „Replays“ wurde von „Tristan Sokol“ erstellt (vgl. [3]). Die generierte Heatmap ist auf Abbildung 4 zu sehen. Es ist deutlich zu erkennen, dass vor allem Anhöhen, Abgründe oder Loopings die größten Hindernisse darstellen, an denen am meisten Zeit während des Trainings verbracht wird. Interessant ist außerdem, dass nachdem der Looping geschafft ist, kaum mehr Stellen vorhanden sind, an denen der Agent hängen bleibt. Das ist damit zu erklären, dass im hinteren Teil des Levels keine neuen Hindernisse mehr kommen, mit denen der Agent noch nicht gelernt hat um zu gehen.

### 8.2 Tensorboard

Während des Trainings werden außerdem Daten für „Tensorboard“ mitgeschrieben. Diese Daten können dann in Form eines Graphen angezeigt werden. So zeigt der Graph (vgl. Abbildung 5), der die Belohnung im Laufe des Trainings (Anzahl an Schritten) darstellt, dass es teilweise starke Schwankungen gibt. Dies ist vermutlich damit zu erklären, dass der Agent immer wieder an Hindernissen hängen geblieben ist, diese dann teilweise schafft und später aber wieder hängen bleibt. Gegen Ende des Trainings ist ersichtlich, dass ein großes Hindernis überwunden wurde. In diesem Fall handelt es sich vermutlich um den Looping, da dort, wie auch schon aus der Heatmap ersichtlich, am meisten Zeit verbraucht wurde.

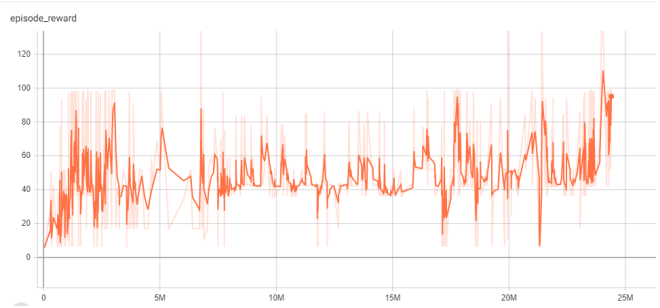



**Abbildung 5: Belohnung über die Dauer des Trainings**

## 9 FAZIT

Dank sehr guter Bibliotheken und deren Dokumentation, sowie dem ausgetragenem Wettbewerb von „Open AI“, ist es gelungen, selbst ein Neuronales Netz zu trainieren, das „Sonic The Hedgehog“ spielt. Das aktuell trainierte Netz absolviert das erste Level. Das Prinzip lässt sich aber auch auf weitere Level anwenden.

## LITERATUR


[1] Shweta Bhatt. 2018. *5 Things you need to know about Reinforcement Learning.* Retrieved July 7, 2019 from https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

[2] christopherhesse. 2019. *Gym Retro Repository.* Retrieved July 7, 2019 from https://github.com/openai/retro

[3] Tristan Sokol. 2018. *Making fun visuals, history maps and other tool improvements.* Retrieved July 7, 2019 from https://medium.com/@tristansokol/making-fun-visuals-history-maps-and-other-tool-improvements-eb5ffe187fd3

[4] Lucas Thompson. 2018. *Youtube Tutorial - Sonic AI Bot with NEAT.* Retrieved July 7, 2019 from https://www.youtube.com/playlist?list=PLTWFMbPFsvz3CeozHfeuJIXWAJMkPtAdS

[5] Ashley Hill und Antonin Raffin und Maximilian Ernestus und Adam Gleave. 2019. *Stable Baselines Repository.* Retrieved July 7, 2019 from https://github.com/hill-a/stable-baselines

[6] John Schulman und Filip Wolski und Prafulla Dhariwal und Alec Radford and Oleg Klimov. 2017. *Proximal Policy Optimization Algorithms.* Retrieved July 7, 2019 from https://arxiv.org/abs/1707.06347

[7] Vicki PfauAlex und NicholChristopher und HesseLarissa und SchiavoJohn und SchulmanOle and Klimov. 2018. *Gym Retro.* Retrieved July 7, 2019 from https://openai.com/blog/gym-retro/

[8] unixpickle. 2018. *Sonic Util Repo.* Retrieved July 7, 2019 from https://github.com/openai/retro-baselines/blob/master/agents/sonic_util.py

[9] Sonic Fandom Wiki. 2018. *Sonic the Hedgehog Genesis.* Retrieved July 7, 2019 from https://sonic.fandom.com/de/wiki/Sonic_the_Hedgehog_Genesis


**Abbildung 6: Sonic Heatmap - Generiert aus den Trainingsdaten (Orginal Levelbild: http://www.sonicgalaxy.net/wp-content/img/maps/gen/sonic/ghz-1.png).**

# Flappy Bird anhand von Bekräftigungslernen lösen

Walter Ehrenberger
Allgemeine Informatik
Hof University of Applied Sciences
wehrenberger@hof-university.de

## ABSTRACT

In der vorliegenden Arbeit werden insgesamt drei Algorithmen im Bereich des Bekräftigungslernen behandelt. Die aufeinander aufbauenden Techniken werden auf das Spiel Flappy Bird mit unterschiedlichem Erfolg angewandt. Das letzte Verfahren schafft es menschliche Gegner in dem Spiel zu überbieten. Damit wird der Erfolg in der Forschung dieses Gebiets in den letzten Jahren verdeutlicht.

## 1 Einführung

Maschinelles Lernen ist die Kunst einen Computer dazu zu bringen ein Problem zu lösen, ohne ihn speziell darauf zu programmieren. Dies führte in den letzten Jahren zu enormen technologischen Fortschritten, namentlich selbstfahrenden Autos, Spracherkennung sowie einem fundierteren Wissen über das menschliche Genom.[1]

Bekräftigungslernen, zu Englisch *Reinforcement Learning*, ist ein Teilbereich des maschinellen Lernens und bildet damit die Grundlage dieser Arbeit. Die Idee ist es, einen Agenten eine Umgebung eigenständig erkunden zu lassen bis er diese verstanden hat. Das Verständnis wird anhand eines Belohnungssystems aufgebaut. Alles ohne menschliche Führung. Kinder verhalten sich in den jungen Jahren ihrer Entwicklung ähnlich.[2] Sie erkunden ihre Umgebung und bekommen direktes Feedback. Ein Beispiel dafür ist ein Kind, das sich im Sommer weigert die Schuhe anzuziehen. Bestraft wird es durch aufgeheizten Teer und schmerzenden Fußsohlen, wodurch ein Lernerfolg erzielt wird. Die ersten Kapitel befassen sich damit wie das Lernen bei Computern ablaufen kann.

Ein weiteres beeindruckendes Beispiel ist die Arbeit des Teams Open AI[3]. Mit ihrer gleichnamigen künstlichen Intelligenz, *OpenAi Five,* haben sie es in dem Spiel Dota 2 geschafft professionelle Spieler zu besiegen. Dies ist insofern beeindruckend, da mittels dieses Spiels Turniere um bis zu 25 Millionen Euro Preisgeld ausgetragen werden. Der Sieg wurde am 13.April.2019 errungen, und ist damit ein recht junger aber enormer Meilenstein. Der verwendete Algorithmus, *Proximal Policy Optimization* (PPO), ist zwar nicht Gegenstand dieser Arbeit, baut aber auf den hier vorgestellten Techniken und Algorithmen auf.
Videospiele tragen eine besondere Rolle in diesem Gebiet. Anhand von simulierten Umgebungen lassen sich endlose Datenmengen generieren, die für den Trainigsprozess notwendig sind.

## 2 Hintergrund

Hierbei behandelte Techniken und Algorithmen sind das Produkt zweier Werke. Ersteres ist das Buch *Deep Reinforcement Learning Hands-On*, von Maxim Lapan, welches die Algorithmen aber auch die Theorie für dieses Thema behandelt. Des Weiteren ist die Arbeit *Playing Atari with Deep Reinforcement Learning* ausschlaggebend. Sie befasste sich 2013 damit, anhand von Bekräftigungslernen eine Vielzahl der Atari Spiele besser zu lösen als es Menschen möglich ist. Bei drei von sieben gewählten Spielen ist ihnen das gelungen.

### 2.1 Bekräftigungslernen

Neben dem überwachten Lernen, sowie dem unüberwachten Lernen bildet Bekräftigungslernen eines der großen Teilgebiete des maschinellen Lernens.

Im Gegensatz zum überwachten Lernen werden Informationen nicht direkt an den Agenten übermittelt, indem sie beschriftet sind. Allerdings wird darauf wie beim unüberwachten Lernen nicht komplett verzichtet. Anstatt dessen wird dem Agenten gesagt in welche Richtung er sich bewegt, nämlich ob die getroffenen Entscheidungen zu einem positiven oder negativen Ergebnis geführt haben.

Prinzipiell lässt sich die Technik innerhalb eines Satzes zusammenfassen. Ein Agent soll in einer Umgebung die bestmöglichen Entscheidungen treffen um die größtmögliche Belohnung zu erhalten.

[1] Andrew Ng, *What is Machine Learning*, https://www.coursera.org/lecture/machine-learning/what-is-machine-learning-Ujm7v, zuletzt besucht im Juli.2019
[2] Andrew Ng, Reinforcement Learning: *An Introduction*, Kapitel 1, Seite 1
[3] Greg Brockman David Fahri, Szymon Sidor und weitere, *OpenAI's mission is to ensure that artificial general intelligence benefits all of humanity*, https://openai.com/five/, veröffentlicht am 13.04.2019

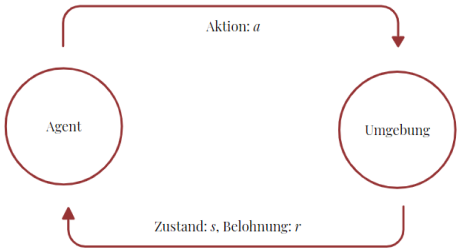



**Abbildung 1:** Folgende Abbildung erklärt wichtige Begriffe sowie den generellen Ablauf eines Lernprozesses. Ein Agent hat die Möglichkeiten, Aktionen auf eine Umgebung auszuführen. Die Umgebung liefert dem Agenten Beobachtungen über ihren eigenen Zustand. Anhand der Zustände, sowie der Belohnungen, die zusammen mit einem Zustand auftreten, soll der Agent lernen ausgehend von jedem Zustand die beste Entscheidung zu treffen.

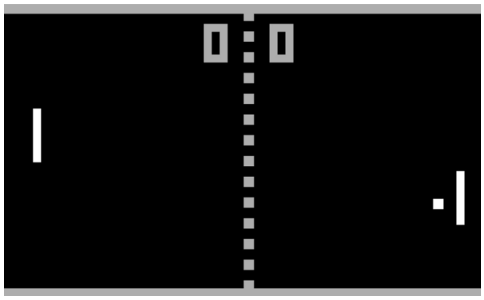


**Abbildung 2[4]:** Exemplarisch lässt sich das Prinzip am Spiel Pong veranschaulichen. Der Agent ist ein beliebiger Algorithmus, welcher beispielweise über ein neuronales Netzwerk lernt. Anhand der Aktionen steuert der Agent seine Plattform auf den Ball hinzu. Die Umgebung, das Spiel, liefert Zustände in Form der Pixeldaten, die auch ein Mensch wahrnimmt. Andere Formen der Zustände, wie die Positionen der Figuren auf einer Koordinatenachse sind auch denkbar. Schließlich erhält der Agent Resonanz darüber, ob seine Entscheidung dazu geführt hat, ob der Gegner oder er einen Punkt erzielt hat.

Anhand dieses Schemas lassen sich folgende Probleme ableiten:

**1.** Was geschieht, wenn die Belohnung verspätet eintritt?
**2.** Wie soll auf eine wahrscheinlichkeitsbedingte Umgebung reagiert werden?
**3.** Welche Belohnungen sind für den Agenten relevant?
**4.** Wie sollen die Zustände dem Agenten vermittelt werden?

## 2.2 Markov Decision Process

Gelöst werden eben behandelte Probleme zum großen Teil über den *Markov decision process*. Dies ist ein mathematisches Modell von Andrey Markov, um Umgebungen zu beschreiben[5]. Es bildet eine der wichtigsten Grundlagen des Bekräftigungslernens.

### *2.2.1 Markov Chain*

Elementar baut der *Markov decision process* auf der *Markov chain* auf. Zum einfacheren Verständnis wird sich wieder am im letzten Kapitel besprochenen Spiel Pong bedient.

Über den kompletten Zeitraum eines Spiels, genannt **Episode**, entstehen eine Vielzahl von Beobachtungen. Ein Zustand wird dabei über $\boldsymbol{s}$ definiert. Diese können in einer Menge zusammengefasst werden, die mit $\boldsymbol{S}$ beschrieben wird. Mit ablaufender Zeit ändern sich die Zustände. Wird der Ball von der gegnerischen Plattform getroffen und nach oben geschossen, bewegt sich der Ball fortlaufend in eine gewisse Richtung, bis ein Rand getroffen wird. Die Menge $\boldsymbol{S}$ der Zustände darf laut dem Model nicht unendlich groß sein, aber nahezu, sodass die Anzahl für gewöhnlich kein Hindernis darstellt.

Ein Schlüsselelement ist dabei die **Markov-Eigenschaft**. In der Zukunft liegende Zustände dürfen nur von dem jetzigen Zustand abhängen. Daraus lässt sich über den jetzigen Zustand $\boldsymbol{s_t}$ eine Funktion bilden aus der sich alle nachfolgenden Zustände $\boldsymbol{s_{t+k}}$ ableiten lassen. Dies bedeutet, dass mittels dieser Funktion der weitere Verlauf des Systems beschrieben werden kann. Es lässt sich also bestimmen, welcher Zustand $\boldsymbol{s'}$ mit einer gewissen Wahrscheinlichkeit als Nächstes eintritt.

Auf Pong bezogen bedeutet das, dass wenn der Ball nach oben geschossen wird er auch nach oben fliegt. Die gegnerische Plattform bewegt sich abhängig von der Implementation möglicherweiße zufällig oder zeitversetzt zum Ball.

Umgebung können nur über den *Markov decision process* beschrieben werden, wenn sie die **Markov-Eigenschaft** erfüllen. Folgerichtig darf sich die Wahrscheinlickeit für das Eintreten eines bestimmen nächsten Zustandes $\boldsymbol{s'}$ abhängig von $\boldsymbol{s}$ nicht ändern.

### *2.2.2 Markov Reward Process*

Um die *Markov chain* weiterzuführen werden die zuvor behandelten Belohnungen über $\boldsymbol{R_t}$ eingeführt. Dabei entsteht ein *Markov reward process*. Jeder Zustand liefert nun eine Belohnung in der Form eines Zahlenwertes mit sich.

Vereinfacht bedeutet das im Spiel Pong, dass der Agent im Falle eines Tors eine Belohnung von **+1** erhält. Im Falle eines gegnerischen Tors erhält er eine Belohnung von **-1**. Für den Fall, dass nichts passiert beträgt die Belohnung **0,1**.

[4] Bild des Spiels Pong: https://www.funstockretro.co.uk/news/atari-pong-is-45-years-old-today/, zuletzt besucht am 17.06.2019

[5] Maxim Lapan: "Deep Reinforcement Learning Hands-On", Kapitel 1, Markov decision process

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots$$

$$= \Sigma_{k=0} \gamma^k R_{t+k+1}$$

Beschreiben lässt sich der Prozess über die gegebene Formel. Aufgelöst wird nach $G_t$. $G_t$ steht für die maximale Belohnung ausgehend vom Zeitpunkt $t$. Für jeden nachfolgenden Zeitpunkt $t$ werden die entsprechenden Belohnungen $R$ der Zustände $S$ aufsummiert. Von besonders großer Bedeutung ist $\gamma$. Gamma löst das Problem der zeitversetzten Belohnung wie in Kapitel 2.1 besprochen. In der Zukunft liegende Belohnungen dürfen zwar nicht ignoriert werden, haben aber eine andere Bedeutung als Belohnungen, die mit dem jetzigen Zustand einhergehen. Mittels der Konstante Gamma hoch der zeitlichen Entfernung $k$ werden diese minimiert.

Wenn für Gamma eins festgelegt wird, hat jede Belohnung über eine Episode den gleichen Wert. Für den entgegengesetzten Fall, Gamma ist gleich null, hat nur die Belohnung für den jetzigen Zeitpunkt eine Bedeutung. Ersteres wäre bei Spielen wie *Tic Tac Toe* realisierbar, da die Episode maximal über neun Zeitschritte verfügt. In der zu Beginn dieses Kapitels besprochenen Literatur wird für Gamma der Wert **0.9** empfohlen[6]. Dies hat sich, wie in den nachfolgenden Kapiteln erkennbar wird, auch in dieser Arbeit bewährt.

Mittels der Einführung von Belohnungen in die Gleichung, ist es somit möglich die Abfolge von Zuständen zu finden, welche die größtmögliche Belohnung mit sich bringen.

*2.2.3 Markov Decision Process*

Um das dargestellte Schema in der ersten Abbildung zu vervollständigen fehlt lediglich die Möglichkeit Aktionen auszuüben. Dementsprechend wird der Gleichung die Menge $A$ mit der auf die Umgebung möglichen auszuführenden Aktionen hinzugefügt. Anhand einer Aktion $a$ kann die Wahrscheinlichkeit des nächsten Zustandes $s'$ somit beeinflusst werden.

Bezogen auf Pong ist das die naheliegende Möglichkeit die Plattform nach oben oder unten zu lenken. Dabei entsteht das Herzstück eines Agenten, die **Policy** $\pi$. Sofern eine Umgebung über den Markov decision process beschrieben werden kann, ist es möglich über einen Zustand anhand der Policy $\pi(s)$ die bestmögliche Entscheidung zu treffen.

Was jedoch zuerst erlernt werden muss, ist die Policy, welche die Strategie eines Agenten wählt. Ein zufälliges Bewegen der Plattform, auch wenn ohne Erfolg, kann genauso über die Policy beschrieben werden wie die erfolgssichersten Bewegungen.

### 2.3 Die Umgebung Flappy Bird

Um die nachfolgenden Algorithmen zu testen wurde als Umgebung Flappy Bird verwendet, siehe Abbildung 3. In diesem Spiel ist es das Ziel solange zu überleben wie möglich. Röhren bewegen sich von rechts in konstanter Geschwindigkeit auf den Spieler hinzu. Dieser muss durch diese hindurch navigieren ohne die Röhren oder den Boden zu berühren. Für durch jede erfolgreich navigierte Röhre erhält der menschliche Spieler einen Punkt.

Implementiert wurde das Spiel eigens in *python3* mit dem Framework *pygame*. Die Schnittstelle für den Agenten ist inspiriert an *gym*. Das open-source Framework enthält eine Vielzahl von Umgebungen die für etwaige Algorithmen als Benchmark dienen.

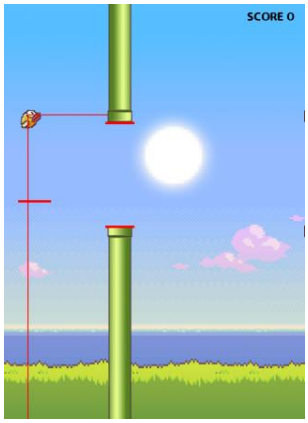



**Abbildung 3:** In rot markiert sind die Zustände, welche der Agent von der Umgebung für jeden Zeitschritt erhält. Dies sind die Bewegungsgeschwindigkeit des Spielers auf der y-Achse, die Entfernung zum Boden sowie die Entfernung zum nächsten Röhrenpaar und die Eingangskoordinaten der Röhren.

Es wurde bewusst darauf verzichtet die Pixelinformationen des Spiels zu verwenden, da hier, nicht wie in der Arbeit *Playing Atari with Deep Reinforcement Learning*[7] mehrere Umgebungen mit einem einzigen Algorithmus in der gleichen Konfiguration gelöst werden sollen. Dies ermöglicht die Implementation eines einfachereren Algorithmus, sowie eine schnellere Konvergenzzeit. Nachteil ist, dass die Umgebung sowie die zu erhaltenden Zustände iterativ optimiert werden mussten.
Eine der entscheidendsten Änderungen ist das Erzeugen der Röhren. Zu Beginn wurden diese zufällig auf der kompletten y-Achse verteilt erzeugt. Wie später behandelt wird, führte das zu Konvergenzproblemen bei den in Verwendung getretenen Algorithmen. In der finalen Version werden diese auf fünf unterschiedlichen, gleichbleibenden Positionen erzeugt.

Als Belohnung erhält der Agent für den Fall, dass nichts passiert eine Belohnung mit dem Wert **+0,1**, für den Fall, dass der Boden

[6] Maxim Lapan: *Deep Reinforcement Learning Hands-On*, Kapitel 1, *Markov decision process*

[7] Volodymyr Mnih Koray Kavukcuoglu David Silver etc.: Playing Atari with Deep Reinforcement Learning, veröffentlicht im Januar.2013

oder eine Röhre getroffen wird eine Belohnung von **-1** und für den Fall, dass eine Röhre erfolgreich überwunden wird eine Belohnung von **+1**. Das Skalieren der Werte auf einen Bereich von **-1** bis **+1** hat insbesondere den letzten Algorithmus bei der Konvergenz enorm unterstützt.

Um mittels Bekräftigungslernen gelöst werden zu können, muss schließlich definiert werden, dass die Umgebung die Voraussetzungen eines *Markov decision process* erfüllt. Der Spieler selbst wird zu Beginn des Spiels auf der immer gleichen Höhe erzeugt. Alle Röhrenanordnungen werden mit der gleichen Wahrscheinlichkeit erzeugt, während an der Umgebung mit fortlaufender Zeit nichts an diesen Wahrscheinlichkeiten geändert wird. Demnach erfüllt diese Implementation die Voraussetzungen.

## 3 Implementierung der Algorithmen

Die ersten zwei Implementationen der folgenden Algorithmen entstammen aus dem in Kapitel 2 behandelten Buch von Maxim Lapan. Der dritte Algorithmus, das Deep Q-Network wurde von Greg Surma implementiert[8].

### 3.1 Unterscheidung der Algorithmen

Prinzipiell lassen sich Algorithmen, die auf Bekräftigungslernen basieren in drei Kategorien einteilen[9]. Diese Kategorien sind abhängig von der Umgebung bei der Auswahl eines Algorithmus zu bedenken.

- Modellfrei oder modellbasiert
- Wertebasiert oder Policy-basiert
- On-Policy oder Off-Policy

**Modellfrei** bedeutet, dass kein Modell der Umgebung aufgebaut wird. Zustände werden direkt berechnet und nur abhängig von diesen wird eine Entscheidung gefällt. Im Gegensatz dazu versuchen **modellbasierte** Algorithmen den nächsten Zustand oder die nächste Belohnung vorherzusagen. Die beste Entscheidung wird auf Grundlage der Vorhersage gewählt. Ersteres Verfahren wurde in den letzten Jahren der Forschung die größere Aufmerksamkeit geschenkt. In den meisten Fällen ist es schwer ein reiches Modell einer komplexen Umgebung aufzubauen.

**Policy-basierte** Verfahren versuchen in den meisten Fällen anhand einer Wahrscheinlichkeitsverteilung über die verfügbaren Aktionen die Beste zu wählen. Die Policy wird approximiert. **Wertebasierte** Verfahren berechnen für jeden Zeitschritt einen Wert für alle gegebenen Aktionen. Dabei wird die Aktion mit dem besten Wert gewählt.

Jeder bekräftigungslernender Algorithmus verwendet die Policy um zu entscheiden welche Aktion abhängig von einem gegebenen Zustand gewählt wird. **Off-Policy** bedeutet in dem Zusammenhang, dass die erlernte Policy während des Lernvorgangs ignoriert wird. Entgegengesetzt dazu beziehen andere Algorithmen diese bei **On-Policy** Verfahren mit ein. Einer der bekanntesten Off-Policy Algorithmen ist das *Deep Q-Network,* welches mittels eines Erkundungswertes die Umgebung zunächst unabhängig vom Gelernten erkundet. Dies ist auch der letzte Algorithmus, der in dieser Arbeit behandelt wird.

### 3.2 Cross Entropy

*Cross entropy* fällt in die modellfreie, Policy-basierte und On-Policy Kategorie[9].

Gelernt wird in dieser Implementierung über ein neuronales Netzwerk. Da die *Policy* über das Netzwerk approximiert wird (Policy-basiert) und die Gewichte zu Beginn zufällig verteilt sind, fällt der Agent solange zufällige Aktionen bis der Lernprozess Früchte trägt.

Jede Episode, die der Agent durchläuft ist eine Menge von Zuständen, Aktionen die der Agent ausgeführt hat und Belohnungen für die Aktionen. Für jede Episode kann somit die maximale Belohnung berechnet werden, Gamma mit einbezogen. Da der Agent sich zu Beginn dem Zufall entsprechend verhält, sind einige Episoden erfolgreicher als andere. Ziel von *cross entropy* ist es nur auf erfolgreichen Episoden zu lernen. Der Algorithmus ist dabei wie folgt implementiert.

**Algorithmus:**

**1.** Spiele ***N*** Episoden mit dem momentanen Modell.

**2.** Berechne die erreichte Belohnung für jede Episode und bestimme einen Grenzwert **B**.

**3.** Werfe alle Episoden weg, deren erzielte Belohnung geringer als ***B*** ist.

**4.** Trainiere auf den verbleibenden Elite Episoden.

**5.** Wiederhole Schritt 1 bis das Ergebnis zufriedenstellt.

Kategorisiert werden alle Episoden in Behalten oder in Wegwerfen. Dieses Vorgehen wird durch den Grenzwert in einer konstanten Prozentzahl definiert. In diesem Fall werden nur die 30% der ***N*** besten Episoden für die nächste Iteration behalten.

**Ergebnis:**


[8] Greg Surma, *Cartpole - Introduction to Reinforcement Learning,* https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288, zuletzt besucht im Juli.2019

[9] Maxim Lapan: *Deep Reinforcement Learning Hands-On*, Kapitel 3, *The Cross-Entropy Method*

**Abbildung 4**[10]**:** Auf dem Graphen sind zwei Agenten des Trainingsvorgangs zu sehen. Der schwarze Strich im oberen Bereich signalisiert den Belohnungswert, welcher beim Erreichen der ersten Röhre erhalten wird. Beide Durchläufe wurden für 14 Stunden trainiert. Für Cross_Entropy_1 wurde für ***Gamma*** ein Wert von **0,9** gewählt. Für Cross_Entropy_2 wurde ein Wert von **0,99** gewählt. Dies machte in diesem Szenario keinen Unterschied. Beide basieren auf einem neuronalen Netzwerk mit einer versteckten Schicht welche 512 Knoten trägt. Ebenfalls wurde in beiden Fällen die *learning rate* **0,01** für das Netzwerk verwendet. Für ***N*** wurde in beiden Fällen ein Wert von 100 gewählt.

Ein Hinweis darauf, dass die Umgebung von Flappy Bird zu komplex für diesen Algotrithmus ist, sind die Ergebnisse. Der trainierte Algorithmus konnte in seiner besten Variante im Durschnitt zwei Röhren durchqueren. Mit Glück waren bis zu zehn Röhren möglich.

### 3.3 Value Iteration

Bevor *value iteration* behandelt werden kann, muss noch ein weiteres, wichtiges Konzept erörtert werden. Wie anhand von Markov beschrieben, kann jeder Zustand einer Umgebung einen Wert haben.

$$V(S) = \Sigma_{t=0} R_t \gamma^t$$

***V(S)*** definiert diesen Wert eines Zustandes. Anhand eines Zustandes kann man die maximal erreichbare Belohnung für den Weiterverlauf des Systems beschreiben. Da der nächste Zustand die Folge einer gewählten Aktion ist, kann man auch über den Wert einer Aktion sprechen. Dies vereinfacht die Anwendung in der Praxis enorm.

Richard Bellmann, ein bedeutender Mathematiker, hat mit seiner Belmanngleichung auf dieser Idee aufgebaut[11]. Um die beste Aktion zu wählen, muss der Wert einer Aktion über alle gegebenen Aktionen bestimmt werden. Der optimale Wert ist somit die Aktion, welche die maximal mögliche unmittelbare Belohnung ist, plus die Langzeitbelohnung für den nächsten Zustand. Diese Definition ist rekursiv und damit auf den Ablauf einer kompletten Episode anwendbar. Der Wert der Aktion wird über ***Q(s, a)*** definiert. Daher stammt auch der Name für den dritten Algorithmus, *Deep Q-Network* (*Quality of an action*).

*Value iteration* fällt in die modellfreie, wertebasierte und On-Policy Kategorie. Ziel dieser Methode ist es, die Funktion ***V(s)*** iterativ zu optimieren[12]. Der Algorithmus läuft nach folgendem Schema ab.

**Algorithmus:**

**1.** Initialisiere den Wert aller Zustände auf 0.

**2.** Berechne für jeden Zustand von der Umgebung die Bellmangleichung für ***V(s)*** und update die Werte der Zustände.

**3.** Wiederhole Schritt 1 bis das Ergebnis zufriedenstellt.

In der Praxis werden die Zustände inklusive der berechneten Werte in einer Tabelle eingetragen. Anhand der Tabelle entscheidet der Agent je nach Zustand welche Aktion gewählt werden soll.

**Ergebnis:**

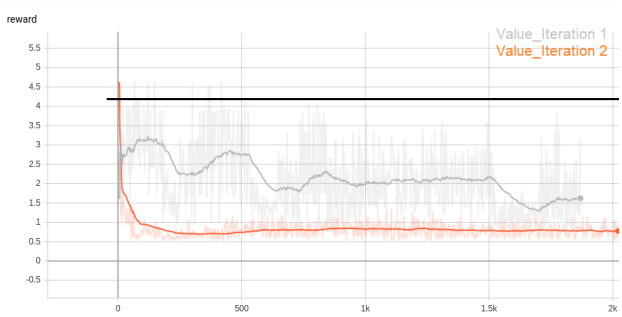



**Abbildung 5:** Es werden zwei Agenten auf dem Graphen dargestellt. Die Trainingsdauer liegt jeweils bei 19 Stunden und für beide Algorithmen wurde für Gamma der Wert **0,9** verwendet. Value_Iteration_2 unterscheidet sich dahingehend, dass die Zustände der Umgebungen auf 2 Nachkommastellen gerundet wurden. Die Idee war es die Anzahl der möglichen Zustände zu verringern. Dies hat das Ergebnis allerdings verschlechtert, da die Zustände wahrscheinlich zu wenig Informationen enthielten.

Auch hier lässt sich erkennen, dass das Verfahren die Umgebung nicht meistern konnte. Dies liegt vermutlich daran, dass die Anzahl der Zustände zu mächtig ist, um sie in einer Tabelle einzufangen. Da nicht einmal ein suboptimaler Wert für ***V(s)*** berechnet werden konnte, hat das Verfahren auch im trainierten Zustand keine Röhre durchqueren können. Anzumerken ist, dass die Werte nicht wie etwa bei *cross entropy* zufällig initialisiert werden. Deshalb wird immer die gleiche Aktion gewählt, solange keine bessere gefunden wurde.

### 3.3 Deep Q-Network

Unter die modellfreie, Policy-basierte und Off-Policy Kategorie fällt der letzte Algorithmus dieser Arbeit. Ein Problem der *value iteration* Methode ist die enorme Anzahl der Zustände. Und das, obwohl viele Zustände für den Agenten irrelevant sind. Um auf das ursprüngliche Beispiel Pong zu verweisen, alle Zustände, in denen der Ball in die Richtung der gegnerischen Plattform fliegen

[10] Die Graphen wurden mit Hilfe der Bibliothek *tensorboard* erstellt. Basierend auf tensorflow, einem *Machine Learning* Framework von Google, https://www.tensorflow.org/guide/summaries_and_tensorboard, zuletzt besucht im Juli.2019

[11] Maxim Lapan: *Deep Reinforcement Learning Hands-On*, Kapitel 5, *Tabular Learning and the Bellman Equation*

haben keine Relevanz für den Lernfortschritt. Mitunter deshalb wird die Bellmanngleichung in dieser Implementierung mittels eines neuronalen Netzwerks approximiert. So verliert die Mächtigkeit der maximalen Zustandsmenge an Bedeutung.

Weiterhin wird in diesem Off-Policy Algorithmus eine Strategie namens **erkunden** und **ausnutzen** genutzt. Über den Wert $\varepsilon$ wird gesteuert, zu wie viel Prozent ein Agent zufällige Aktionen wählt. In der Norm wird dieser Wert über den Trainingsverlauf auf einen Mindestwert gesenkt. Problematisch ist es, wenn der Mindestwert zu groß gewählt wird. In diesem Fall kann für den Agenten der Verdacht aufkommen, dass die Umgebung keine besseren Belohnungen bereithält. Eigene Erfahrungen haben gezeigt, dass der Agent erst beginnt zu lernen, sobald der Mindestwert erreicht ist.

Ein weiteres Problem ist die statistische Verteilung der Zustände. Zustände die nahe bei einander liegen sind sich sehr ähnlich. Das führt zu einer Gleichverteilung die dem neuronalen Netzwerk Probleme bereitet. Anhand einer Technik namens ***experience replay*** werden Paare aus Zustand, Aktion, Belohnung und neuer Zustand **(s, a, r, s')** ein einem Puffer gespeichert. Der Agent lernt dann nicht mehr aus der Umgebung, sondern aus zufälligen Paaren aus dem Puffer. Dieser hat eine maximale Kapazität, ist diese erreicht wird das älteste Paar gelöscht.

Schließlich liefert die Umgebung dem Agenten in dieser Iteration vier Paare von Zuständen. Dies spiegelt die Dynamik der Umgebung wider und hilft dem Netzwerk bei der Konvergenz enorm.

**Algorithmus:**

**1.** Initialisiere alle Parameter der Q-Funktion mit zufälligen Werten und leere den *replay buffer*.

**2.** Wähle anhand von $\varepsilon$ eine Aktion aus und führe diese auf eine Testumgebung aus.

**3.** Speicher Zustand, Aktion, Belohnung und neuen Zustand im *replay buffer*.

**4.** Lerne einen zufälligen Eintrag aus dem *replay buffer*.

**5.** Berechne die *Loss-Funktion* des Netzwerks.

**6.** Berechne den Q-Wert für jeden Eintrag aus dem Buffer.

**7.** Wiederhole Schritt 2 bis zur Konvergenz.

**Ergebnis:**

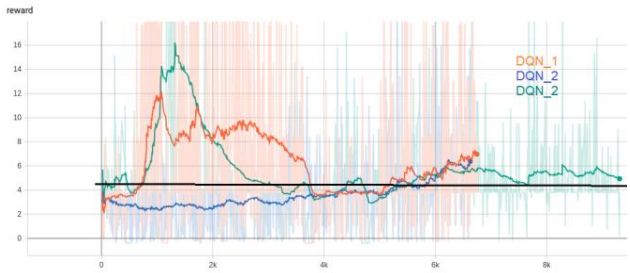



**Abbildung 6:** Wie anhand des Graphen zu erkennen ist hat dieses Verfahren die Umgebung erfolgreich gemeistert. Der Trainingsprozess dauerte jeweils bis zu zwölf Stunden. Es gibt zwei wichtige Punkte, die anzumerken sind. Erstens fällt der Erkundungswert während eines Trainings nie unter einem Prozent. Demnach ist es unwahrscheinlich, dass die Episode während des Trainings nicht terminiert. Zweitens erkennt man, dass die Graphen mit längerem Verlauf wieder absinken und versuchen von neuem zu konvergieren. Eine mögliche Theorie ist, dass das Netzwerk *overfitted*, also versucht die Umgebung auswendig zu lernen. Dies wird für gewöhnlich mit Verfahren wie *dropout* beseitigt, dass hier keine Anwendung gefunden hat.

Alle Agenten verwenden ein neuronales Netzwerk mit einer versteckten Schicht welche 512 Knoten trägt. Ebenfalls wird jeweils für Gamma der Wert **0,9** gewählt. Die Agenten unterscheiden sich jeweils über die Schwierigkeit der Umgebung. Für DQN 1 wurde ein weiter Abstand zwischen zwei Röhren verwendet, für DQN 2 ein mittlerer und für DQN 3 ein sehr geringer. Die ersten zwei Durchläufe konnten die Umgebung im trainierten Zustand meistern. Das heißt, dass der Agent unabhängig von der Episodendauer niemals eine Röhre oder den Boden berührt. Bei DQN 3 ist dies nicht der Fall. Wenn zwei aufeinanderfolgende Röhrenpaare den maximal möglichen Abstand ausweisen, ist es dem Agenten nicht möglich dieses Hindernis zu überwinden. Dies liegt wahrscheinlich an den physikalischen Gegebenheiten der Umgebung. Anzumerken ist, dass DQN 3 im Gegensatz zu den anderen Agenten die nächste Röhre unmittelbar beim Betreten der ersten Röhre sieht. Die anderen Agenten sehen die nachfolgende Röhre erst, wenn die erste komplett überwunden ist. Auch anzumerken ist, dass DQN 3 auf dem bereits trainierten Modell von DQN 2 weiter trainiert hat.

[12] Maxim Lapan: *Deep Reinforcement Learning Hands-On*, Kapitel 5, *Tabular Learning and the Bellman Equation, The Value Iteration Method*

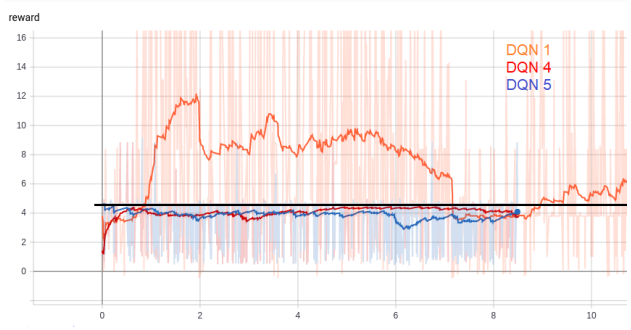



**Abbildung 7:** Folgende Abbildung zeigt den Lernprozess zweier weiterer Agenten. DQN 1 ist der in Abbildung 6 verwendete gleichnamige Agent. Das neuronale Netzwerk von DQN 4 hat 128 Knoten in der versteckten Schicht, das von DQN 5 hat 512. Keine der beiden Agenten verwendet den *replay buffer*. Offensichtlich können die Agenten ohne diesen nicht konvergieren.

## 5 Diskussion

Hinsichtlich des Lernprozesses in Abbildung 6 und 7 erkennt man, dass besonders das *Deep Q-Network* nicht optimal angewandt wurde. Speicherstände eines Netzwerks, das konvergiert hat konnten die Umgebung nach einer weiteren Iteration nicht mehr erfolgreich lösen. Möglichkeiten um dagegen vorzugehen wurden bereits besprochen. Auch die Umgebung musste in vielen Iterationen überarbeitet werden. Mit einer längeren Trainingsdauer wäre eine Konvergenz vielleicht möglich gewesen. Nichtsdestotrotz belegen die Ergebnisse die Mächtigkeit von *Deep Q-Networks*. Nicht umsonst ist dieses Verfahren ein zentraler Punkt der Forschung im Feld des maschinellen Lernens der letzten Jahre inklusive darauf aufbauender Verfahren wie die *PPO*.

## ANHANG

Die trainierten Algorithmen sowie die Umgebung können unter folgendem Link auf Github gefunden werden, https://github.com/Walterooo/ReinforcementLearning.

Die notwendigen Abhängigkeiten sind in einer Textdatei vermerkt.

## LITERATURVERZEICHNIS

[1] Andrew Ng, *What is Machine Learning*, https://www.coursera.org/lecture/machine-learning/what-is-machine-learning-Ujm7v, zuletzt besucht im Juli.2019

[2] Maxim Lapan: “Deep Reinforcement Learning Hands-On”, veröffentlicht im Juni.2018

[3] Ryan Wong: https://towardsdatascience.com/getting-started-with-markov-decision-processes-reinforcement-learning-ada7b4572ffb, zuletzt besucht am 17.06.2019

[4] Volodymyr Mnih Koray Kavukcuoglu David Silver etc.: Playing Atari with Deep Reinforcement Learning, veröffentlicht im Januar.2013

[5] James MacGlashan, https://www.quora.com/What-is-the-difference-between-model-based-and-model-free-reinforcement-learning, zuletzt besucht am 17.06.2019

[6] Greg Surma, *Cartpole - Introduction to Reinforcement Learning*, https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288, zuletzt besucht im Juli.2019

# Textgeneration mit Hilfe von TensorFlow und Keras

Jonas Beierlein
jonas.beierlein@hof-university.de
Hochschule Hof

## 1 EINFÜHRUNG

Mit Hilfe von maschinellem Lernen kann einem nicht intelligenten Gerät etwas ”beigebracht” werden. Diese Arbeit befasst sich damit, durch das Anwenden von maschinellem Lernen, einem Computer beizubringen Texte zu generieren. Dafür wird die Programmiersprache Python verwendet, sowie Frameworks, welche für Python geschrieben wurden. Neben dem eigentlichen maschinellem Lernen wird ebenfalls aufgezeigt, wie man die Daten für das Training bekommt und wie die Daten vorbereitet werden müssen.

## 2 VERWANDTE ARBEITEN

Diese Arbeit bezieht sich hauptsächlich auf den Kernel von Shivam Bansal, welcher auf kaggle.com veröffentlicht wurde.[1]

## 3 VERWENDETE PROGRAMMIERSPRACHE UND FRAMEWORKS

### 3.1 Python

Python ist eine einfach zu benutzende Programmiersprache, welche es dem Entwickler ermöglicht schnell Ergebnisse zu produzieren. Eine Vielzahl an Frameworks, die in eigene Projekte eingebunden werden können, vereinfacht die Benutzung weiter. Unter diesen Frameworks befinden sich auch solche, die speziell für das Arbeiten mit maschinellem Lernen konzipiert worden sind und dieses Thema für den Benutzer zugänglich machen, ohne ein höheres Verständnis der Mathematik dahinter abzuverlangen. Solche Frameworks wurden auch für diese Arbeit benutzt und werden nun näher beleuchtet.

### 3.2 TensorFlow

TensorFlow ist eine Open Source Platform für maschinelles Lernen. TensorFlow bietet unter anderem ein Framework und APIs für Python an. Mit Hilfe von TensorFlow kann man Aufgabenstellungen des maschinellen Lernens angehen, ohne die genaue Mathematik dahinter vollends zu verstehen. Da TensorFlow verschiedene Abstraktionsebenen unterstützt, können aber auch fortgeschrittene Benutzer komplexe Probleme angehen.[4]

### 3.3 Keras

Keras ist ein Python Framework, welches auf TensorFlow aufsetzt. Keras wurde speziell dazu entwickelt, um Entwicklern eine Möglichkeit zu bieten ihre Ideen schnell in einen lauffähigen Prototypen zu verwandeln. Weiterhin bietet Keras verschiedene Arten von neuronalen Netzen an, also sowohl ”Convolutional Neural Networks”,als auch ”Recurrent Neural Networks”. Letzteres wird in dieser Arbeit verwendet.[2]

*FWPM: Angewandtes maschinelles Lernen, Sommersemester, 2019, Hof*

### 3.4 Scrapy

Scrapy ist ebenfalls ein Python Framework, mit dessen Hilfe man Webseiten durchforsten (crawlen) und Informationen daraus extrahieren (scrapen) kann. Dies geschieht, in dem man sogenannte ”Spiders” programmiert, welche mit Scrapy gestartet werden. Näheres dazu weiter unten.

## 4 THEORIE

### 4.1 Maschinelles Lernen

Maschinelles Lernen ist ein Teilgebiet der Informatik und soll dem Computer beibringen Aufgaben ”intelligent” auszuführen. Dies versucht man zu erreichen, in dem man Modelle bestehend aus künstlichen neuronalen Netzen mit Daten trainiert. Diese trainierten Modelle können dann zum Beispiel Vorhersagen treffen.

In Bezug auf diese Arbeit bedeutet, dass das Modell an Hand von einem Wort herausfinden soll, welches Wort als nächstes im Satz am wahrscheinlichsten stehen sollte.

### 4.2 LSTM

Wie schon erwähnt wird das Modell, welches in dieser Arbeit behandelt wird ein sequentielles Modell. LSTM steht für Long-Short-Term-Memory, was übersetzt ”langes Kurzzeitgedächtnis” bedeutet. LSTMs werden verwendet, um die Leistungsfähigkeit von rekurrenten neuronalen Netzen (RNN) zu verbessern. Dies wird erreicht in dem die RNNs die Fähigkeit erhalten, sich an frühere Trainingsdurchläufe zu erinnern. Eine LSTM-Zelle besteht dabei aus der ”Erinnerung”, sowie verschiedenen Toren, welche bestimmen in welchem Umfang neue Informationen aufgenommen werden, ob Informationen aufgenommen oder vergessen werden sollen und in welchem Umfang die Information wieder ausgegeben werden sollen.[3]

## 5 DATENBESCHAFFUNG

Um ein Modell zu trainieren benötigt man Daten. In dieser Arbeit soll ein Modell geschaffen werden, welches in der Lage ist Text zu generieren. Als Beispiel soll es Überschriften bzw. Titel erzeugen können. Dazu werden zwei verschiedene Datensätze benutzt. Wie diese Datensätze beschafft werden können und für die Weiterverarbeitung vorbereitet werden müssen, soll in diesem Kapitel beschreiben werden.

Wie oben bereits erwähnt wurde für diese Aufgabenstellung das Python Framework Scrapy verwendet. Im Nachfolgenden werden nun die ”Spiders” für die beiden Datensätze aufgezeigt und erläutert.

## 5.1 SpiegelSpider

```
import scrapy

class SpiegelSpider(scrapy.Spider):
    name = 'SpiegelSpider'
    allowed_domains = ['spiegel.de']

    def start_requests(self):
        urls = []
        for i in range(1947, 2020):
            urls.append("https://www.spiegel.de/spiegel/
                print/index-"+str(i)+".html")

        for url in urls:
            yield scrapy.Request(url=url, callback=self.
                parse)

    def parse(self, response):
        for title in response.xpath('//div[@class="
            spiegel-gallery-item-title"]/a[1]/@title'):
            yield{
                "title": title.get().upper().split("(
                    SIEHE")[0].split("- DER SPIEGEL")
                    [0].split("(S.")[0].replace("\"", ""
                    ),
                "issue": title.get().upper().split("- DER
                     SPIEGEL ")[1]
            }
```

**Listing 1: SpiegelSpider.py**

Mit Hilfe des oben stehenden Spider lassen sich die Titel aller bisher erschienen Spiegel-Zeitschriften herunterladen. Dazu werden als erstes die URLs generiert, welche durchsucht werden sollen. In diesem Fall sind die einzelnen URLs wie folgt aufgebaut:

https://www.spiegel.de/spiegel/print/index-”jahr”.html

Wobei ”jahr” für das Erscheinungsjahr der einzelnen Ausgaben steht. Auf jeder der Webseiten die durch diese URLs erreicht werden, sind die einzelnen Ausgaben dieses Jahres gelistet. Weiterhin werden nun die tatsächlichen Titel auf den einzelnen Seiten gesucht und in einem JSON gespeichert. Davor werden noch ein paar Anpassungen an den Titeln vorgenommen, z.B. werden nicht gewollte Inhalte aus den Titeln entfernt und alle Zeichen werden zu Großbuchstaben gemacht. Das erzeugte JSON kann dann weiter verarbeitet werden.

## 5.2 TagesschauNewsSpider

```
import scrapy
from datetime import *
from dateutil.relativedelta import *
from dateutil.parser import *

class TageschaunewsspiderSpider(scrapy.Spider):
    name = 'tageschauNewsSpider'
    allowed_domains = ['tagesschau.de']
    start_urls = ['http://tagesschau.de/']

    def start_requests(self):
        urls = []
        i = 0
        NOW = datetime.now()
        for i in range(0, 365):
            tempDate = NOW+relativedelta(days=-i)
            datestring = tempDate.strftime("%Y%m%d")
            urls.append("http://www.tagesschau.de/archiv/
                meldungsarchiv100~_date-"+datestring+".
                html") #20190605

        for url in urls:
            yield scrapy.Request(url=url, callback=self.
                parse)

    def parse(self, response):
        for title in response.xpath('//div[@class="
            linklist withnolinkcontentbutnoproblem"]/ul/
            li/a/text()'):
            yield{
                "title": title.get().upper().split(" |")
                    [0].replace("\"","")
            }
```

**Listing 2: tageschauNewsSpider.py**

Funktioniert analog zu dem SpiegelSpider. Allerdings wurden hier zusätzliche Python Frameworks benutzt, um die URLs zu erstellen.

Dieser Spider liest sämtliche Titel von den Meldungen vom tageschau.de Archiv der letzten 365 Tage aus (Es sind immer nur die Meldungen der letzten 365 Tage online verfügbar). Die Titel werden ebenfalls in ein JSON gespeichert.

# 6 IMPLEMENTIERUNG

Im Nachfolgenden soll die Implementierung des Modells dargelegt werden. Der gezeigte Code bezieht sich größtenteils auf einen Kernel von Shivam Bansal, der auf kaggle.com veröffentlicht wurde.[1]

Es wurden ein paar Änderungen zugunsten der Übersichtlichkeit getätigt. Außerdem wurde der Code an die hier verwendeten Datensätze angepasst.

## 6.1 Imports

```
import tensorflow as tf
import json
from google.colab import files
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Embedding, LSTM, Dense, Dropout
from keras.preprocessing.text import Tokenizer
from keras.callbacks import EarlyStopping
from keras.models import Sequential
import keras.utils as ku
from keras.models import load_model

from tensorflow import set_random_seed
from numpy.random import seed
set_random_seed(2)
seed(1)

import pandas as pd
import numpy as np
import string, os
```

**Listing 3: Imports**

Für das Modell und die Datenverarbeitung werden Frameworks wie TensorFlow und Keras benötigt. Außerdem noch das Framework json, um die in Kapitel 5 erzeugten JSONs verarbeiten zu können.

Ebenfalls werden sog. ”Seeds” gesetzt, um nachher die Ergebnisse vergleichbar zu machen.[1]

## 6.2 Datenverarbeitung

Bevor die Texte dem Modell zum Trainieren übergeben werden können, müssen sie vorbereitet und verarbeitet werden.

```
newsTitel = []

with open('news.json') as json_file:
  data = json.load(json_file)
  for entry in data:
    newsTitel.append(entry["title"])
```

**Listing 4: JSON mit Titeln einlesen**

Die einzelnen Titel, welche in Kapitel 5 in ein JSON gespeichert wurden, werden nun aus diesem wieder ausgelesen und für die Weiterverarbeitung in ein Array gespeichert. Somit ist jeder Titel einzeln aus diesem Array abrufbar.

```
tokenizer = Tokenizer()

def generate_tokens(newsTitel):
  tokenizer.fit_on_texts(newsTitel)
  total_words = len(tokenizer.word_index) + 1
  return total_words

def generate_input_sequences():
  input_sequences = []
  for title in newsTitel:
    token_list = tokenizer.texts_to_sequences([title])[0]
    for i in range(1, len(token_list)):
      n_gram_sequence = token_list[:i+1]
      input_sequences.append(n_gram_sequence)
  return input_sequences

total_words = generate_tokens(newsTitel)
inp_sequences = generate_input_sequences()
```

**Listing 5: Wörter in Tokens umwandeln und Sequenzen erstellen**

Da das Modell nur mit Zahlen umgehen kann, müssen sämtliche Wörter als erstes in Zahlen, sogenannte Tokens, umgewandelt werden. Anschließend werden die Titel mit Hilfe der Tokens in Input-Sequenzen umgewandelt.

```
def generate_padded_sequences(input_sequences):
    max_sequence_len = max([len(x) for x in
         input_sequences])
    input_sequences = np.array(pad_sequences(
         input_sequences, maxlen=max_sequence_len,
         padding='pre'))

    predictors, label = input_sequences[:,:-1],
         input_sequences[:,-1]
    label = ku.to_categorical(label, num_classes=
         total_words)
    return predictors, label, max_sequence_len

predictors, label, max_sequence_len =
     generate_padded_sequences(inp_sequences)
```

**Listing 6: Auffüllen der Sequenzen**

Sämtliche Input-Sequenzen mit denen das Modell trainiert werden soll, müssen die gleiche Länge haben. Um das zu erreichen werden alle Sequenzen, die kleiner sind als die längste Sequenz, mit Nullen aufgefüllt. Die Null hat als Token einen besonderen Stellenwert, da sie speziell zum Auffüllen genutzt wird und somit vom Modell auch gesondert zu den eigentlich relevanten Daten betrachtet wird.

## 6.3 Modell

```
lstm_size = 200
dropout = 0.1

def create_model(max_sequence_len, total_words):
    input_len = max_sequence_len - 1
    model = Sequential()

    model.add(Embedding(total_words, 10, input_length=
         input_len))

    model.add(LSTM(lstm_size))
    model.add(Dropout(dropout))

    model.add(Dense(total_words, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
         optimizer='adam', metrics=['accuracy'])

    return model

model = create_model(max_sequence_len, total_words)
```

**Listing 7: Modell captionpos**

Wie bereits erwähnt wird eine LSTM-Schicht benutzt. Außerdem eine Embedding-Schicht, Dropout und eine Dense-Schicht. Die gewählte Größe der LSTM-Schicht hat sich als guter Mittelweg zwischen der Trainingsdauer und der Qualität der Ergebnisse herausgestellt.

Nachfolgend die Zusammenfassung des Modells:

```
_________________________________________________
Layer (type)             Output Shape     Param #
=================================================
embedding_4 (Embedding) (None, 12, 10)  158990
_________________________________________________
lstm_4 (LSTM)            (None, 200)     168800
_________________________________________________
dropout_4 (Dropout)      (None, 200)     0
_________________________________________________
dense_4 (Dense)          (None, 15899)   3195699
=================================================
Total params: 3,523,489
Trainable params: 3,523,489
Non-trainable params: 0
_________________________________________________
```

**Listing 8: Modell Zusammenfassung captionpos**

## 6.4 Testen des Modells

```
def generate_text(seed_text, next_words, model,
     max_sequence_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([
             seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=
             max_sequence_len-1, padding='pre')
        predicted = model.predict_classes(token_list,
             verbose=0)

        output_word = ""
        for word,index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " "+output_word
    return seed_text.upper()
```

**Listing 9: Methode zum Testen des Modells captionpos**

Um zu überprüfen, ob das Modell richtig trainiert wurde bedarf es einer Methode, die mit Hilfe des trainierten Modells arbeitet. Da dieses Modell mit verschiedenen Titeln trainiert worden ist, wird versucht Titel anhand von einem ersten Wort zu reproduzieren. Der Methode werden also folgende Parameter mitgegeben: das erste Wort, die Anzahl der Wörter die darauf folgen sollen, das trainierte Modell, sowie die maximale Sequenzlänge.

## 7 ERGEBNISSE

Nach einem Training von 100 Epochen sind die Ergebnisse schon sehr gut. Bei dem tageschau.de-Datensatz wurde eine Genauigkeit von ca. 0,75 erreicht. Bei dem Spiegel-Titel-Datensatz wurde eine Genauigkeit von ca. 0,86 erreicht, was vermutlich an der geringeren Anzahl an Titeln liegt.

Teilweise wurden Titel exakt wie in den Trainingsdaten vorhanden reproduziert. Da man in der Methode zur Textgenerierung (siehe Listing 9) eine feste Anzahl an nachfolgenden Wörtern vorgibt, kommen nicht immer ”sinnvolle” Texte zustande. Das liegt daran, da die Titel auch nicht immer eine feste Anzahl an Wörtern haben. Mache sind etwas länger und manche sind etwas kürzer. Lässt man diesen Aspekt allerdings außen vor und betrachtet die zusammenhängenden Titel die generiert worden sind, ist das Ergebnis als brauchbar zu betrachten.

## 8 AUSBLICK

Um das Modell und die daraus folgenden Ergebnisse zu verbessern, kann man an mehreren Parametern ansetzen. Zum einen könnte man das Modell weiter verfeinern.[1]

Zum Beispiel durch zusätzliche LSTM-Schichten die hinzugefügt werden oder in dem man die vorhandene Schicht vergrößert. Außerdem könnte man die Anzahl der Trainingsepochen noch erhöhen, um die Genauigkeit der Vorhersagen zu verbessern.

Um eine optimale Kombination aus Modell und Trainingsepochen zu erreichen müsste man weiterhin mit den Parametern experimentieren.

## LITERATURVERZEICHNIS

[1] S. Bansal. Beginners guide to text generation using lstms. https://www.kaggle.com/shivamb/beginners-guide-to-text-generation-using-lstms/notebook, Jan. 2019. [Online; accessed 28-Jun-2019].

[2] keras-team. Keras github. https://github.com/keras-team/keras. [Online; accessed 28-Jun-2019].

[3] S. Luber. Was ist ein long short-term memory? https://www.bigdata-insider.de/was-ist-ein-long-short-term-memory-a-774848/, Nov. 2018. [Online; accessed 28-Jun-2019].

[4] Tensorflow. About tensorflow. https://www.tensorflow.org/about/. [Online; accessed 28-Jun-2019].

# Angewandtes maschinelles Lernen - Opinion Mining

Kee Man Pun
Hochschule Hof
kee-man.pun@hof-university.de

Jonas Kemnitzer
Hochschule Hof
jonas.kemnitzer@hof-university.de

Dominik Thalhammer
Hochschule Hof
dominik.thalhammer@hof-university.de

Angelina Scheler
Hochschule Hof
angelina.scheler@hof-university.de

## ZUSAMMENFASSUNG

Zentrales Thema dieser Studienarbeit ist die Verwendung eines Recurrent Neural Networks (RNN). Anhand von Bewertungen eines E-Commerce Shops, welcher sich auf Damenkleidung spezialisiert hat, soll ein maschinell lernendes Netzwerk darauf trainiert werden, die Wahrnehmung (Polarität) eines Textes oder Tokens zu quantifizieren. Quantifizieren bedeutet in unserem Fall, dass das Netzwerk analysiert, ob der Text im Allgemeinen als positiv (Wörter wie z. B. “wunderschön”) oder negativ (z. B. “hässlich”) wahrgenommen wird oder eher neutral ist. Hier kommt zudem das Teilgebiet des Natural Language Processing zum Einsatz. Als Programmiersprache für das Projekt verwenden wir Python. Tensorflow und Keras liefern nötige Funktionen, um effizient ein neuronales Netzwerk aufzubauen. Das Colab Notebook von Google nutzen wir als Entwicklungsumgebung. Im Allgemeinen soll ein Grundverständis für RNN’s als auch notwendige Teilbereiche sowie die Herangehensweise solcher Projekte erläutert werden.

**ACM Reference Format:**
Kee Man Pun, Jonas Kemnitzer, Dominik Thalhammer, and Angelina Scheler. . Angewandtes maschinelles Lernen - Opinion Mining. In *FWPM: Angewandtes maschinelles Lernen, Sommersemenster, 2019, Hof* . ACM, New York, NY, USA, 9 pages.

## 1 EINLEITUNG

Heutzutage gewinnt maschinelles Lernen, als Teilbereich der künstlichen Intelligenz immer mehr an Bedeutung. Künstliches Wissen wird aus Erfahrungen generiert. Damit eine Software eigenständig lernen und Lösungen finden kann, muss diese mit entsprechenden Daten und Algorithmen versorgt werden. Ein Gebiet, das maschinelles Lernen verwendet, ist das Opinion Mining. Durch Analyse der natürlichen Sprache, Natural Language Processing, wird die Stimmung oder Meinung der Öffentlichkeit zu einem bestimmten Produkt erfasst. Ob dieses eher negativ oder positiv bewertet wurde. Ziel unserer Studienarbeit ist es ein Programm zu schreiben, welches Nutzerbewertungen von Frauenkleidung analysiert.

## 2 RELATED WORK

Im Gebiet der Sentimental Analysis gibt es bereits eine Reihe weiterer Projekte, die sich mit dem Thema beschäftigen. Ein beliebter Einsteigerkurs ist das IMDB-Movie Review Set. Dieser Datensatz dient in einer Reihe von Tutorials als Grundlage. Für Einsteiger sind vor allem die Bücher von Giancarlo Zaccone (2018) sowie von Thushan Ganegedara (2018) zu empfehlen.

*FWPM: Angewandtes maschinelles Lernen, Sommersemenster, 2019, Hof*
.

## 3 MOTIVATION

Sei es die fiktive Darstellung eines selbst denkenden Androiden, das eines hoch entwickelten Computers, der aus künstlichem neuralen Netzwerk besteht (Positronisches Gehirn vom Schriftsteller Isaac Asimov) oder eine menschenähnliche Maschine, die aus der

**Abbildung 1: Abbildung fiktiver Android**
**Wikipedia: URL: https://en.wikipedia.org/wiki/Data_(Star_Trek) [Stand 16.10.2019]**

Zukunft kommt und mit einer neuralen Netzwerkprozessor ausgestattet ist, das selbst lernen kann (Terminator 2 Judgment Day von James Cameron). Beide genannten fiktiven Beispiele haben einen

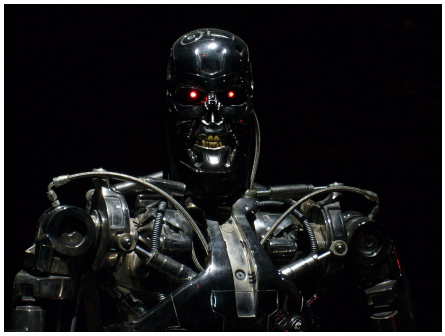


**Abbildung 2: Abbildung einer Maschine aus dem Film Terminator**
**Wikipedia: URL: https://de.wikipedia.org/wiki/Terminator_(Film) [Stand 16.10.2019]**

gemeinsamen Nenner, die zum maschinellen Lernen hinführen. Aus damaligen Fiktionen ist heute vieles zur Realität geworden. Man nehme hier, als Beispiel den tragbaren Tricorder für die Kommunikation aus Star Trek welches dem heutigen Smartphone ähnelt. Die Wissenschaft im Entwicklungsbereich künstliche Intelligenz wird hierbei sogar mit einem Preisgeld von fünf Millionen Dollar unterstützt. Dieses Preisgeld wird von der Organisation XPRIZE vergeben. Mit der Zielsetzung und Motivation, eine künstliche In-

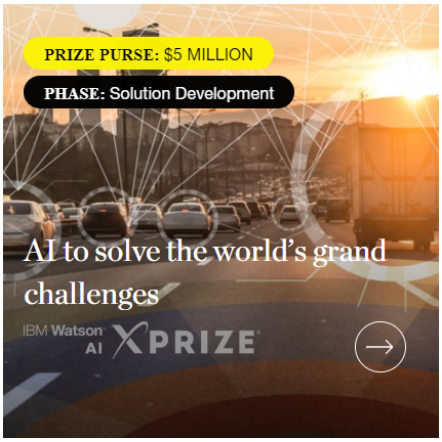



**Abbildung 3: Ausschreibung des xprize AI Wettbewerb XPRIZE Foundation: URL: https://xprize.org/prizes [Stand 16.10.2019]**

telligenz selbst zu erschaffen, die nicht nur Texte einfach lesen kann, sondern auch das Gelesene wirklich versteht, fallen diese Anforderungen eindeutig im Aufgabenbereich von maschinellem Lernen hinein. Aufgrund der zeitlichen Rahmenbedingung einer Studienarbeit befassen wir uns mit einem realisierbaren Ziel in der Erkennung von positiven oder negativen Textbewertungen. Die Skala, die wir hernehmen ist, dass null als das Negative gesehen wird und eins als das Positive. Zwischen null und eins sind Werte erlaubt und soll die Tendenz zur Negativen oder zur Positiven verdeutlichen. Diese Erkennung bildet die Grundlage für das weitere Verständnis in der Stimmungsanalyse. Die daraus resultierenden Ergebnisse dieser Studienarbeit sollen als Beispiel für weitere Stimmungsanalysen frei verfügbar sein und im Idealfall zukünftige oder auch nachfolgende Studierende für diesen Teilbereich vom maschinellen Lernen das nötige Interesse wecken.

### 3.1 Datensatz

Opinion Mining und Stimmungsanalyse sind zwei Begriffe, die in vielen Fällen synonym behandelt werden. Für die Analyse bedarf es eine große Menge an Daten, welches Kaggle der Online-Community von Datenwissenschaftlern und Maschinenlernern, die sich im Besitz von Google LLC befindet (Auszug Wikipedia), bereitstellt. Die Suche nach dem passenden Datensatz war nicht einfach, da wir mit einem Datensatz arbeiten wollen, der emotionale Schlagwörter beinhalten soll. Schlussendlich haben wir uns für den Datensatz Womens Clothing E-Commerce Reviews entschieden. Dieser Datensatz beinhaltet unter den vielen Daten, zwei wichtige Spaltenbereiche, die für unsere Studienarbeit von Wichtigkeit sind. Die Spalte „Review Text“ beinhaltet den Bewertungstext mit emotionalen Schlagwörtern und die Spalte „Rating“ können zur Evaluierung des Bewertungstextes herausgenommen werden. Das Rating hat eine Skala von eins bis fünf, worauf eins als sehr negativ empfunden wird und fünf als sehr positiv. Eine weitere Spalte, die uns im Sinn kam mit in die Bewertung, als Bewertungstext zu nehmen, war die Spalte „Title“. In der Spalte „Title“ sind ebenfalls emotionale Schlagwörter enthalten, die im Vergleich zu dem eigentlichen Bewertungstext kürzer und prägnanter sind. Der Gedankengang, die Spalte „Title“ als weiterer Bewertungstext heranzunehmen, würde den Datensatz von der Menge her verdoppeln, jedoch aber auch den originalen Datensatz möglicherweise verfälschen. Daher wurden schlussendlich nur die Spalten „Review Text“ und „Rating“ für dieses Projekt verwendet. Der originale Datensatz bietet 23486 Zeilen an verwertbaren Roh-Daten an.
Folgendes Beispiel zeigt ein Review Text mit Rating aus dem Datensatz: *Review: This shirt is very flattering to all due to the adjustable front tie. it is the perfect length to wear with leggings and it is sleeveless so it pairs well with any cardigan. love this shirt!!! Rating: 5*
Man erkennt bzw. liest heraus, dass der Bewertungstext positiv gestimmt ist. Die zugehörige Bewertung zeigt eine fünf an, was als höchste positive Wertung in der Skala gilt. Anzumerken ist auch, dass die Bewertungstexte und Bewertungen im Datensatz größtenteils unterschiedlich sind. Es kann jedoch nicht ausgeschlossen werden, dass sich dieser öffentliche Datensatz in absehbarer Zeit verändern wird und die darin enthaltenen Bewertungstexte und Bewertungen sich nicht mehr mit unseren Bewertungstexten und Bewertungen übereinstimmen.

## 4 THEORIE

Im Folgenden wird der allgemeine Hintergrund unserer Vorgehensweise beschrieben. Welche Prozesse der Text durchlaufen muss, um am Ende eine Aussage über den Inhalt treffen zu können, ob dieser positiv oder eher negativ ist. Allgemein betrachtet, sind folgende Schritte nötig:

- Natural Language Processing
- Text-Optimierung
- Tokenizer
- Embedding
- Recurrent Neural Network
- Sigmoid function

### 4.1 Natural Language Processing

Die Unterhaltung zwischen zwei Menschen wird als natürliche Sprache bezeichnet. Sie ist nicht immer eindeutig wie die formale Sprache. Dies führt dazu das ein Computer die natürliche Sprache nicht eindeutig interpretieren kann. In der Informatik versucht man deshalb die natürliche Sprache soweit wie möglich zu analysieren, um sie für die Verarbeitung bereitzustellen. Von dieser Verarbeitung profitiert die künstliche Intelligenz, die man in vielen Fällen auch als maschinelles Lernen bezeichnet. Die Lernergebnisse aus dem maschinellen Lernen finden sich dann in der Anwendung auf die

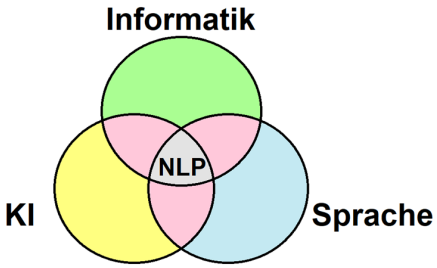



**Abbildung 4: Venn-Diagramm NLP**

natürliche Sprache wieder. Zwischen diesen drei Bereichen befindet sich Natural Language Processing (NLP), die als Schnittstelle fungiert.

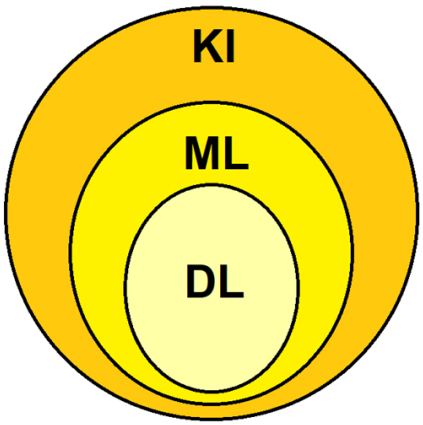



**Abbildung 5: Venn-Diagramm KI-ML-DL**

Um die Einordnung zwischen künstliche Intelligenz, maschinelles Lernen und Deep Learning grafisch zu verdeutlichen, sieht man in der Abbildung den jeweiligen Bereich als eine Menge dargestellt. DL steht für Deep Learning und ist eine Teilmenge von ML also Machine Learning. Machine Learning ist wiederum eine Teilmenge von der künstlichen Intelligenz kurz KI genannt. Wenn man von künstlicher Intelligenz spricht, so kann man anhand dieser Abbildung erkennen, das maschinelles Lernen als auch Deep Learning darin mit inbegriffen sind.

Mithilfe von NLP können natürliche Sprachen auf verschiedene Weisen verarbeitet werden und unterstützen hierdurch anhand der Resultate die zu lernende Maschine. Die Verwendungsbereiche von NLP sind demnach:

- Stimmungsanalyse
- Chatbot
- Speech Recognition
- Maschinelle Übersetzung
- Rechtschreibprüfung
- Keywordsuche
- Extrahierung von Informationen aus Dokumenten oder Web

In der Stimmungsanalyse werden zum Beispiel die Emotionen aus Bewertungen analysiert ob sie positiv oder negativ sind. Weiterführende Feinheiten liegen in der Erkennung von Ironie oder Sarkasmus. Ein Chatbot liegt im Bereich von Conversational AI und hat zum Beispiel den Einsatzbereich im Kundenservice. Sie hat eine NLP gestützte Spracheingabe und Sprachausgabe und betreut somit dem direkten Kunden. Bekannte Vertreter von Speech Recognition sind zum Beispiel Siri von Apple, Alexa von Amazon, Google Assistent oder Cortana von Microsoft. Der Google Translate gilt als Beispiel für maschinelle Übersetzung. Namenhafte Textverarbeitungssoftware beinhalten Rechtschreibprüfung als Basisfeature. Die Keywordsuche kommt in viele Suchmaschinen zum Einsatz. Bezüglich Extrahierung von Informationen aus Dokumenten oder Webseiten kann man hier ebenfalls Suchmaschinen als Beispiel nehmen, da die Webseiten nach Keywords durchsucht werden und anschließend als Suchergebnisse zurückgeliefert werden.

Die Vorgehensweise mit NLP in Bezug auf Texte ist:

- Tokenization
- Stemming
- Lemmatisierung
- Part-Of-Speech Tagging
- Named Entity Recognitio
- Chunking

In Tokenization geht es darum, ein Text in kleine Stücke zu zerteilen. Zum Beispiel in einzelne Wörter. Dieser ersten Vorgehensweise auf Roh-Texte ist in vielen Fällen üblich und kommt auch in unserem Projekt zum Einsatz. In Stemming werden die Wörter im Text, zu ihrem Wortstamm reduziert. Als Beispiel werden hier die Wörter einfahren, ausfahren und hochfahren dem Wortstamm fahr zugeordnet. In Lemmatisierung werden Wortähnlichkeiten festgestellt. Nimmt man die Wörter laufen, rennen und sprinten als Beispiel, so gehörten alle drei Wörter einen anderen Wortstamm an. Durch die Lemmatisierung werden diese drei Wörter die Gruppierung bewegen eingeordnet. In Part-Of-Speech Tagging auch kurz POS genannt, wird der Text grammatikalisch geprüft und in ihren Wortarten zugeordnet. Die Wortarten mit entsprechendem englischem Kürzel die POS unterscheidet und einordnet sind:

- Nomen = NN
- Nomen (Plural) = NNS
- Pronomen = PRO
- Verb = VRB
- Adverb = ADV
- Adjektiv = ADJ
- Konjunktion = CON
- Präposition = PRE
- Interjektion = INT

Es existieren noch weitere Wortarten für die POS Unterscheidung, die aber nicht alle hier aufgelistet sind. Die Einteilung der Wortarten in POS-Verfahren ermöglicht jedoch nicht die Struktur des Satzes zu erkennen. Je nach Anwendungsbereich käme das Verfahren der Eigennamenerkennung und anschließend Chunking. In Named Entity Recognition also die Eigennamenerkennung werden die Wörter im Text nach Entitäten eingeordnet. Als Beispiel kann die Erkennung von Person, Organisation, Standort und noch weitere Entitäten im Text definiert werden. In Chunking werden kleine

Informationsteile vom großen Ganzen entnommen und anschließend omnidirektional gechunkt. Die Richtung, in der sie gechunkt wird, ermöglicht der Informationsteil genauer zu spezifizieren, Ähnlichkeiten zu finden oder die Abstraktion zu erhöhen, um daraus einen besseren Überblick zu erhalten. Für das Chunking wird in vielen Fällen das POS-Verfahren zuvor verwendet, da hier der Text in seine grammatikalischen Wortarten zerlegt und zugeordnet wird.

Im Folgenden werden einige NLP Bibliotheken aufgelistet, die für maschinelles Lernen verwendet werden können:

- Natural Language Toolkit
- Scikit-learn
- TextBlob
- spaCy

Für unser Projekt wird die Bibliothek Natural Language Toolkit (NLTK) verwendet. NLTK stellt neben den genannten Vorgehensweisen auch eine Ansammlung von Stopwörter bereit, um irrelevante Wörter, die den Inhalt oder Kontext nicht verändern, vorab zu entfernen. Die Stopp-Wörter kann anwendungsspezifisch erweitert werden. Da unser Datensatz aus dem englischen Sprachraum ist, greifen wir auf die englischen Stopp-Wörter von NLTK für die Filterung zu.

## 4.2 Text-Optimierung

Damit im nächsten Schritt nicht zu viele unnötige Tokens entstehen, werden zunächst alle sogenannten Stopp Wörter eliminiert. Stopp Wörter, sind Füllwörter oder Wörter, die für die Aussage des Inhalts nicht relevant sind, z. B. „was, hat, welche“. Dafür gibt es vorgefertigte Listen oder können auch selbst implementiert werden.

## 4.3 Tokenizer

Der Hauptteil geschieht im RNN Recurrent Neural Network, dieses kann allerdings nicht direkt mit dem Text arbeiten. Dementsprechend, muss dieser vorbereitet werden. Der Tokenizer spaltet den Text in die einzelnen Wörter auf und konvertiert diese zu Integer Werten.

## 4.4 Embedding

Um mit den Tokens weiter arbeiten zu können, müssen alle Daten im Datensatz, dieselbe Länge haben, entweder stellen wir sicher, das alle Sequenzen dieselbe Länge haben, oder wir schreiben einen Datengenerator, der kürze Datensätze mit Nullen auffüllt, oder längere kürzt. Die maximale Anzahl der Tokens wird über den Durchschnitt und einer Standardabweichung bestimmt. Die Datensätze können entweder am Anfang oder Ende gefüllt bzw. abgeschnitten werden. Je nachdem für welchen Ansatz man sich entscheidet, muss dieser konsequent durchgeführt werden, um eine konsistente Auswertung zu gewährleisten. Anschließend werden die einzelnen Tokens in Vektoren umgewandelt mit Werten zwischen -1,0 und 1,0, da das RNN mit Matrixoperationen arbeitet.

## 4.5 Recurrent Neural Network

Als rekurrente bzw. rückgekoppelte neuronale Netze bezeichnet man neuronale Netze, die sich im Gegensatz zu den Feedforward-Netzen durch Verbindungen von Neuronen einer Schicht zu Neuronen derselben oder einer vorangegangenen Schicht auszeichnen. Rückgekoppelte neuronale Netze bzw. Backpropagation oder auch Backpropagation of Error bzw. auch Fehlerrückführung genannt, ist ein verbreitetes Verfahren für das trainieren von künstlichen neuronalen Netzen. Es gehört zur Gruppe der überwachten Lernverfahren und wird als Verallgemeinerung der Delta-Regel auf mehrschichtige Netze angewandt. Dazu muss ein externer Lehrer existieren, der zu jedem Zeitpunkt der Eingabe die gewünschte Ausgabe, den Zielwert, kennt. Die Rückwärtspropagierung ist ein Spezialfall eines allgemeinen Gradientenverfahrens in der Optimierung, basierend auf dem mittleren quadratischen Fehler. Der Backpropagation-Algorithmus läuft in folgenden Phasen: Ein Eingabemuster wird angelegt und vorwärts durch das Netz propagiert. Die Ausgabe des Netzes wird mit der gewünschten Ausgabe verglichen. Die Differenz der beiden Werte wird als Fehler des Netzes erachtet. Der Fehler wird nun wieder über die Ausgabe- zur Eingabeschicht zurück propagiert. Dabei werden die Gewichtungen der Neuronenverbindungen abhängig von ihrem Einfluss auf den Fehler geändert. Dies garantiert bei einem erneuten Anlegen der Eingabe eine Annäherung an die gewünschte Ausgabe. In künstlichen neuronalen Netzen werden rekurrente Verschaltung von Modellneuronen benutzt, um zeitlich codierte Informationen in den Daten zu entdecken. Rekurrente Netze lassen sich folgendermaßen unterteilen:

- direkt: Bei einer direkten Rückkopplung (englisch direct feedback) wird der eigene Ausgang eines Neurons als weiterer Eingang genutzt.
- indirekt: Die indirekte Rückkopplung (englisch indirect feedback) verbindet den Ausgang eines Neurons mit einem Neuron der vorhergehenden Schichten.
- seitlich: Die seitliche Rückkopplung (englisch lateral feedback) verbindet den Ausgang eines Neurons mit einem anderen Neuron derselben Schicht.
- vollständig: Bei einer vollständigen Verbindung hat jeder Neuronenausgang eine Verbindung zu jedem anderen Neuron.

Das RNN besteht aus Grundbausteinen den Recurrent Unit’s, kurz RU. Von den wiederkehrenden Einheiten gibt es verschiedene Variationen unter anderem die GRU(Gated Recurrent Unit)und die LSTM(Long short-therm memory), welche vollständige RNN sind. Diese unterscheiden sich in der Komplexität und Anzahl der Gates. GRU ist weniger komplex und hat zwei Gates. Wir betrachten das GRU. Jede RU hat einen internen Status, der bei jeder Eingabe aktualisiert wird. Dieser interne Status dient als eine Art Gedächtnis für Gleitkommawerte, welche durch Matrixoperationen gelesen und geschrieben werden. Der Statuswert, hängt von der vorherigen und aktuellen Eingabe ab. Wenn zum Beispiel zuerst der Wert für "nicht"gespeichert wurde und die aktuelle Eingabe "gut" ist, müssen wir einen neuen Wert "nicht gut" speichern, der speichert, was auf eine negative Stimmung hinweist. Der Teil der RU der für das Abbilden der alten Werte und Eingabe der neuen Werte verantwortlich ist, nennt man Gate. Ein Gate ist im eigentlichen Sinne eine Matrixoperation. Für die Berechnung der Ausgangswerte gibt es ein weiteres Gate. Um die RU zu trainieren, müssen die Gewichtsmatrizen der Gates schrittweise geändert werden, damit die Recurrent Units für die Eingabe Sequenz den gewünschten Output liefert. In der Praxis durchläuft die Sequenz mehrere RU Schichten, nach jeder

Aktualisierung, wird das Ergebnis in die nächste Schicht weiter gereicht und nachdem die Sequenz alle Schichten, auch Layer genannt durchlaufen hat, ergibt das Gesamtergebnis den Output des RNN.

## 4.6 Sigmoid function

Sigmoidfunktionen werden oft in künstlichen neuronalen Netzwerken als Aktivierungsfunktion verwendet. Als Aktivierungsfunktion eines künstlichen Neurons wird die Sigmoidfunktion auf die Summe der gewichteten Eingabewerte angewendet, um die Ausgabe des Neurons zu erhalten. Die Sigmoid function, nimmt den Output des RNN und errechnet daraus einen Wert zwischen 0 und 1. Liegt der Wert in der Nähe von null, schließen wir daraus, dass der Text eher eine negative Aussage hat. Umgekehrt siedelt sich der Wert bei 1 an, ist die Aussage des Textes eher positiv. Sie wird folgendermaßen definiert.

$$\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

**Abbildung 6: Sigmoid function**

# 5 IMPLEMENTIERUNG

Diese Kapitel beschäftigt sich mit den konkreten Funktionen unseres Code. Es dient außerdem als Dokumentation um die Herangehensweise anhand unseres Beispiel genauer zu erläutern. Im wesentlicher lässt sich unser Code in 3 Abschnitte einteilen. Diese 3 Teile sind:

- Vorarbeit und Textaufbereitung
- Erstellen und Trainieren des RNN
- Auswertung und des Netzwerk

Letztes wird in einem späteren Kapitel Auswertung sowie Ausblick genauer behandelt. Dieser Teil fokussiert sich auf die ersten beiden Punkte.

## 5.1 Vorbereitung und Textverarbeitung

Bevor wir mit der Textverarbeitung anfangen können, müssen wir zunächst alle notwendigen Daten in die Entwicklungsumgebung laden. Die verwendeten Daten sind frei zugänglich und wurden von der Website Kaggle beschafft. Im ersten Schritt werden zunächst alle notwendigen import-statements durchgeführt um die notwendigen Funktionen und die Entwicklungsumgebung zu laden.

**Listing 1: Import Statements**

```
import numpy as np
import tensorflow as tf
import pandas as panda
import nltk
import pickle

from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, GRU, Embedding
from tensorflow.python.keras.optimizers import Adam
from tensorflow.python.keras.preprocessing.text import Tokenizer
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
```

Wurden alle imports vollständig ausgeführt, so können wir nun mithilfe der bereitgestellten Funktionen, unsere CSV-Datei einlesen und uns gleichzeitig einen kleinen Teil anzeigen lassen. Dieser Schritt dient dazu, um die Struktur der Datei zu verifizieren.

**Listing 2: Einlesen der CSV-Datei**

```
csv = panda.read_csv("/content/drive/My Drive/Colab Notebooks/WomensClothing.csv",sep=";")
print(csv.shape)
print(csv.head())

csv2 = csv.iloc[:,4]

total_rows = csv['Review Text'].count()
print (total_rows)
print(csv2)
```

## 5.2 Textaufbereitung

Unser Netzwerk weiß zunächst nichts mit menschlichen Wörtern anzufangen. Deshalb ist der nun folgende Schritt für unser RNN extrem wichtig. Wir müssen den Text in eine für das Netzwerk verständliche Form umwandeln. Kernbestandteil ist hier das Natural Language Processing. Der Text enthält Wörter die für unsere Analyse unwichtig sind. Man spricht hier von sogenannten Stopp-Wörtern. Also Wörter die keinen Einfluss auf die Wertigkeit des Textes haben. All dieser Wörter speichern wir in ein Wörterbuch. Wir können theoretisch unser Wörterbuch manuell mithilfe der CSV-Datei erstellen, in dem wir durch die Sätze einzeln durch iterieren. Jedoch decken wir damit eventuell nicht alle Wörter einer Sprache ab. Die Reviewtexte sind auf Englisch somit laden wir uns mithilfe des Toolkit ein bereits vorgefertigtes Wörterbuch. Der Vorteil liegt darin, dass wir dadurch wesentlich mehr Wörter der englischen Sprache abdecken.

**Listing 3: Einlesen der Stopp-Wörter**

```
nltk.download('punkt')
nltk.download('stopwords')
stopWords = set(stopwords.words('english'))
```

Weiterhin erstellen wir ein Array, das alle Sonderzeichen sowie Zahlen enthält. Sonderzeichen und Zahlen sind wie die Stopp-Wörter für unser Netzwerk irrelevant und werden somit aus dem Text später entfernt.

**Listing 4: Filterung von Zahlen und Sonderzeichen**

```
numbersArray = ["1","2","3","4","5","6","7","8","9","0"]
sonstigesArray = ["—","#", "'s", "(", ")", ",","!", "&", "'m", "'", "i",","]
```

Mithilfe des zuvor erstellen Wörterbuchs sowie den beiden Arrays, können wir nun mithilfe einiger for-Schleifen den Text bereinigen. Wir erhalten nach der Bereinigung eine neue Liste, in der die Wörter beinhaltet sind, die das neuronale Netzwerk später analysiert.

**Listing 5: Erstellen der Listen mit relevanten Wörtern**

```
listForKM = []
listForDomi = set([])
for testdata in csv['Review Text']:
    wordsFiltered = []
    wordsFiltered2 = []
    words = word_tokenize(str(testdata))
    for w in words:
        lowerW = w.lower()
        youshallnotpass = False
        if lowerW in stopWords:
            pass
        else:
            for eachLetter in lowerW:
                if eachLetter in numbersArray:
                    youshallnotpass = True
            if youshallnotpass == False:
                wordsFiltered.append(lowerW)

    for w in wordsFiltered:
        if w in sonstigesArray:
```

```
            pass
        else:
            wordsFiltered2.append(w)
            listForDomi.add(w)
    listForKM.append(wordsFiltered2)
```

Unser Text ist nun bereinigt. Das RNN kann den Text jedoch in seiner jetzigen Form nicht analysieren. Hierfür müssen wir unsere zuvor erstelle Liste in Tokens umwandeln. Es wird also jedem Wort ein Integer als eine Art Index zugewiesen. Dies geschieht mithilfe von folgendem Code.

**Listing 6: Verwendung des tokenizer**

```
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=20000)
tokenizer.fit_on_texts(listForDomi)
print(tokenizer.word_index)
with open('tokenizer_3.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In **listForDomi** stehen nun keine Wörter mehr, sondern der jeweilige Integer Wert, der das jeweilige Wort repräsentiert. Wir haben nun unsere Indizes, jedoch betrachten wir hier alle Wörter des Reviewtext. Um die Wörter in dem Kontext des zugehörigen Satzes zu betrachten, erstellen wir eine Sequenz von Integer Werten, die den einzelnen Satz darstellt.

**Listing 7: Sequenzierung**

```
kmtokens = tokenizer.texts_to_sequences(listForKM)

print(listForKM[0])
print(kmtokens[0])
```

Betrachten wir nun die Struktur des Satzes so erhalten wir.

**Listing 8: Darstellung Satz als Integersequenz**

```
['absolutely', 'wonderful', 'silky', 'sexy', 'comfortable']
[4531, 510, 932, 274, 136]
```

Im Moment haben unsere Reviewsätze noch unterschiedliche Längen. Bevor die Daten dem RNN übergeben werden können, müssen wir die Längen der Sequenzen jedoch angleichen. Wir bestimmen als Nächstes also die maximale Länge, die eine Sequenz haben kann.

**Listing 9: Bestimmen der maximalen Sequenzlängen**

```
num_tokens = [len(kmtokensElement) for kmtokensElement in kmtokens]
num_tokens = np.array(num_tokens)


max_tokens = np.mean(num_tokens) + 2 * np.std(num_tokens)
max_tokens = int(max_tokens)
print(max_tokens)



p = np.sum(num_tokens < max_tokens) / len(num_tokens)
print(p)
# Prozentsatz der damit erreichten Abdeckung
```

Sätze welche die vorgegebene Länge nicht einhalten werden mithilfe des **Padding** mit 0 aufgefüllt. Dafür gibt es 2 unterschiedliche Arten. Das **pre-padding** bei dem die 0er zu Beginn der Sequenz aufgefüllt werden oder das **post-padding**. Hier werden die 0er am Ende platziert. Wir verwenden das **pre-padding**.

**Listing 10: Angleichen der Sequenzlängen**

```
pad = 'pre'
domi_token_pad = pad_sequences(kmtokens, maxlen=max_tokens, padding = pad, truncating=pad)
```

## 5.3 Erstellen des eigentlichen RNN

Nachdem wir unseren Input nun vollständig bereinigt sowie vorbereitet haben, können wir nun das eigentliche RNN erstellen. Das Netz verfügt über 3 GRU's sowie ein dense-layer, welches 3 Outputs liefert. Es werden deshalb 3 Outputs geliefert, da wir eine Kategorisierung zwischen Schlecht, Mittel sowie Gut vornehmen.

**Listing 11: Erstellen des eigentlichen RNN**

```
model = Sequential()
embedding_size = 8
model.add(Embedding(input_dim=200000, output_dim=embedding_size, input_length=max_tokens, name='
    ↪ layer_embedding'))

model.add(GRU(units=16, return_sequences=True))
model.add(GRU(units=8, return_sequences=True))
model.add(GRU(units=4))

#model.add(Dense(1, activation='sigmoid'))
model.add(Dense(3, activation='sigmoid'))

optimizer = Adam(lr=1e-3)

model.compile(loss='categorical_crossentropy', optimizer = optimizer, metrics = ['accuracy'])
model.summary()
history = None
```

Als Bezugspunkt für unsere Auswertung benötigen wir weiterhin das in der CSV vorhandene Rating.

**Listing 12: Einholen des Rating**

```
for testdata in csv['Rating']:
    if testdata < 3:
        ratingList.append([1.0,0.0,0.0]);
    if testdata == 3:
        ratingList.append([0.0,1.0,0.0]);
    if testdata > 3:
        ratingList.append([0.0,0.0,1.0]);

ratingList = np.array(ratingList)
```

Im letzten Schritt trainieren wir das RNN. Unserer Erkenntnis nach erhalten wir mit 100 Epochen ein akzeptables Ergebnis ohne das wir das neuronale Netz stundenlang trainieren müssen. Wir erhalten somit eine Genauigkeit von **0.9866**.

**Listing 13: Trainieren des RNN**

```
mhistory = model.fit(trainData, trainLabels, epochs=100, batch_size=128, verbose=1)
```

# 6 AUSWERTUNG

## 6.1 Auswertung des trainierten Netzwerk

In den Auswertungen zeigt sich, das minimale Änderungen der Sätze, große Veränderungen bewirken können. Man nehme hier als Beispiel, das Rating von eins bis fünf. Im ersten Versuch die Ratingstufen prozentual einzuordnen wurde ein unbefriedigendes Resultat erreicht.

Die Ratings waren:

- 1 Stern = 0.00
- 2 Sterne = 0.25
- 3 Sterne = 0.50
- 4 Sterne = 0.75
- 5 Sterne = 1.00

Obwohl diese prozentuale Einordnung logisch nachvollziehbar erscheint, konnte das Lernmodell nur eine Treffergenauigkeit von circa 59 Prozent erreichen. Ebenfalls wurde dieses Ergebnis schon nach 15 Epochen erreicht und hat sich bis zu 100 Epochen hin nur noch weiter gefestigt. Die Genauigkeit von circa 59 Prozent ist hierbei fast gleichzusetzen wie ein einfaches Erraten. Dies ist natürlich nicht zielführend. Jedoch leitet man aus der Erkenntnis zwei

Vermutungen her. Die erste Vermutung liegt darin, dass die vorhandenen Daten in der Menge nicht ausreichen, um die gewünschten Ergebnisse zu erzielen. Die zweite Vermutung war, dass die Gewichtungen für die Ratingstufen nicht optimiert sind, weshalb wir uns mit dieser Vermutung intensiver befasst haben. Der Gedankengang das eine drei Sterne Bewertung den Mittelwert 0.5 darstellt, bleibt weiterhin bestehen. Ebenfalls ist die untere und obere Grenze des Rating eindeutig mit 0.0 und 1.0 festgelegt. Für die Anpassungen kommen somit nur noch zwei Sterne und vier Sterne infrage. Die Psychologie von Bewertungen greift hier mit einer gewissen Komplexität ein und beeinflusst somit die Tendenz der zwei Sterne und vier Sterne. Ein zwei Sterne Rating wurde deshalb dem negativen 0.0 Wertung eingeordnet und eine vier Sterne Rating entsprechend der positiven 1.0 Wertung. Die neuen Ratings sind:

- 1 Stern = 0.00
- 2 Sterne = 0.00
- 3 Sterne = 0.50
- 4 Sterne = 1.00
- 5 Sterne = 1.00

Daraus konnte im zweiten Versuch eine Genauigkeit von circa 87 Prozent erzielt werden, was das Lernmodell akzeptabel macht aber immer noch eine hohe Fehlerquote aufzeigt. Für den dritten Versuch wurde die Erkenntnisse aus dem zweiten Versuch weiter optimiert. Jede einzelne Bewertung liefert nun kein ganzes Ergebnis mehr, sondern sie wird in den drei Spalten negativ, neutral und positiv zugeordnet. Hierdurch soll ersichtlich werden wie stark die drei Spalten innerhalb einer Bewertung vertreten sind. Nach dieser Optimierung lieferte der dritte Versuch eine Genauigkeit von circa 98 Prozent. Mit dem Modell aus dem dritten Versuch wurden die Testdaten evaluiert und eine Genauigkeit von circa 78 Prozent erreicht. Nach der erfolgreichen Evaluierung der Testdaten, soll die Evaluierung durch selbst geschriebene Bewertungen stattfinden. Die acht selbst geschriebenen Bewertungen beinhalten ein Muster, um das Modell zu prüfen.

**Listing 14: EIGENER INPUT**

```
#Verwendung von Selbstgeschriebenen Test
testList = []
testList.append("the_clothes_is_very_nice_wonderful_pretty_perfect_super_")
testList.append("the_clothes_is_very_nice_wonderful_color_pretty_smooth_perfect_fit_super_elastic_low_price")
testList.append("the_clothes_is_very_nice")
testList.append("the_clothes_is_very_nice_to_wear_wonderful_color")

testList.append("the_material_of_the_pants_could_be_better")

testList.append("the_quality_of_the_pants_is_bad")
testList.append("the_quality_of_the_pants_is_absolutly_bad_and_uncomfortable")
testList.append("i'm_very_unhappy_with_the_design")

ownCodedOpinionIntokens = tokenizer.texts_to_sequences(testList)
```

## 6.2 Auswertung mithilfe der Diagramme

In der Bewertung von Abbildung 7 wurden mehrere emotionale Schlagwörter nacheinander geschrieben. Ein positives Ergebnis war zu erwarten und dieses wurde anschließend auch bestätigt.

In der Bewertung von Abbildung 8 wurden mehrere emotionale Schlagwörter betont. Ein positives Ergebnis war zu erwarten und dieses wurde anschließen auch bestätigt.

In der Bewertung von Abbildung 9 wurde der positive Text kurz

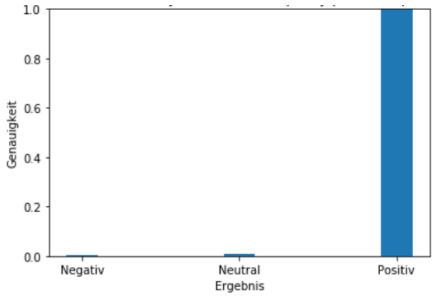



**Abbildung 7: Bewertung des 1. Satzes**

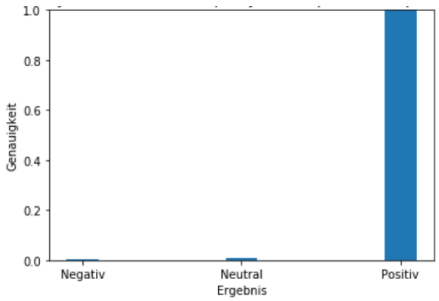



**Abbildung 8: Bewertung des 2. Satzes**

gehalten. Ein positives Ergebnis war zu erwarten und dieses wurde anschließend auch bestätigt. Am Diagramm sieht man hier, dass das Modell die Bewertung als positiv erkannt hat, jedoch nur knapp über 20 Prozent. Die Erklärung hierfür liegt am verwendeten Datensatz, mit dem das Modell angelernt wurde. Ein Großteil der Bewertungstexte ist ausführlich geschrieben und haben somit eine gewisse Textlänge. Es müssen mehrfach kurze Bewertungen im Datensatz gegeben sein, damit das Modell hier eine höhere Genauigkeit erzielt. Rückblickend auf dem Datensatz könnten alternativ die Inhalte aus der Spalte „Title“ in das Modell miteinbezogen werden.

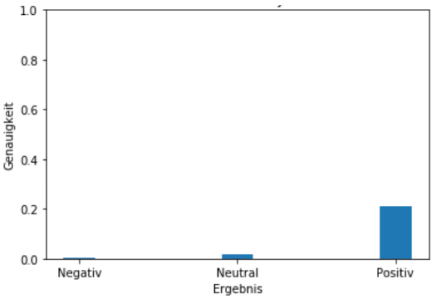



**Abbildung 9: Bewertung des 3. Satzes**

In der Bewertung von Abbildung 10 wurde der positive Text aus Bewertung Nr. 3 etwas erweitert, um zu prüfen ob das Modell den Text nun genauer einordnen kann. Ein positives Ergebnis war zu

erwarten und dieses wurde anschließend auch bestätigt.

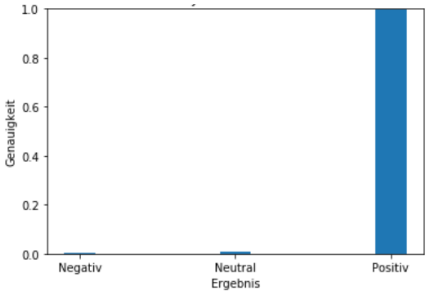



**Abbildung 10: Bewertung des 4. Satzes**

In der Bewertung von Abbildung 11 wurde eine Bewertung geschrieben, welche nur eine Empfehlung darstellt und kaum positive oder negative Schlagwörter beinhaltet. Ein neutrales Ergebnis war zu erwarten und dieses wurde anschließend auch bestätigt. Das Diagramm zeigt, dass das Modell den Textinhalt als neutral erkannt hat, jedoch mit geringer Genauigkeit. Um die neutrale Bewertung mit höherer Genauigkeit zu trainieren, bedarf es wiederum eine große Menge an Daten im Bewertungsbereich von drei Sternen. Ferner müsste das Modell auch mit Widersprüchen angelernt werden, um akzeptable Ergebnisse zu liefern.

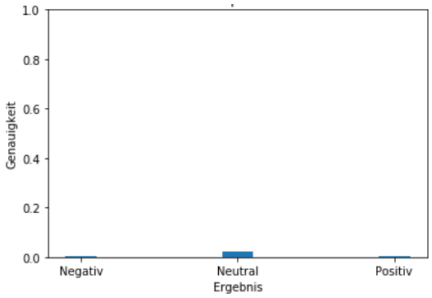



**Abbildung 11: Bewertung des 5. Satzes**

In der Bewertung von Abbildung 12 wurde eine Bewertung mit negativen Inhalt geschrieben und hat eine kurze Textlänge. Ein negatives Ergebnis war zu erwarten, dieses wurde entgegen unserer Erwartung nicht bestätigt. Das Diagramm zeigt ein positives und neutrales Wachstum mit geringer Genauigkeit. Trotz hoher Lerngenauigkeit von circa 98 Prozent liegt hier der Fehler wieder darin, dass das Modell mehr Daten mit kurzer Textlänge zum Lernen braucht.

In der Bewertung von Abbildung 13 wurde die gleiche negative Bewertung aus der Bewertung von Abbildung 12 geschrieben, jedoch mit mehr negativen Schlagwörter. Ein negatives Ergebnis war zu erwarten, dieses wurde anschließend auch bestätigt. Das Diagramm zeigt im Vergleich zu der vorherigen Bewertung eine eindeutige Tendenz, dass der Text negativ vom Modell erkannt wurde.

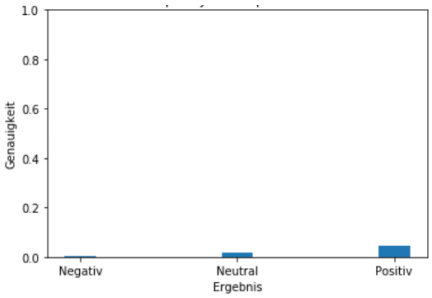



**Abbildung 12: Bewertung des 6. Satzes**

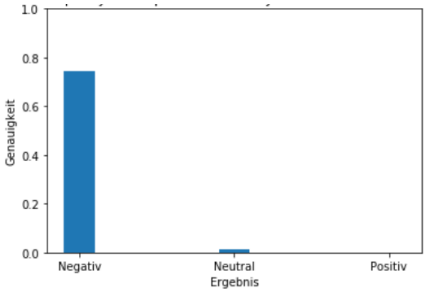



**Abbildung 13: Bewertung des 7. Satzes**

In der Bewertung von Abbildung 14 wurde wieder eine negative Bewertung geschrieben. Ein negatives Ergebnis war zu erwarten, dieses wurde anschließend auch bestätigt. Das Diagramm zeigt trotz der negativen Erkennung ein kleines positives Wachstum in der Genauigkeit. Die neutrale Wertung ist, dem Diagramm nach, nicht vorhanden. Die Vermutung läge darin, dass manche Wörter aus der Bewertung im positiven Zusammenhang stehen.

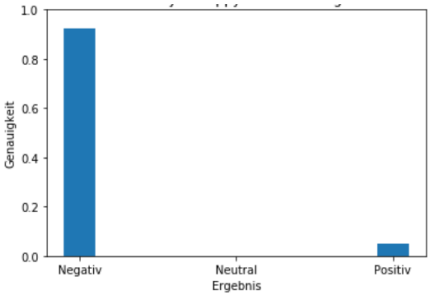



**Abbildung 14: Bewertung des 8. Satzes**

Alle Auswertungen zeigen auf, dass das Modell zum anlernen eine noch größere Menge an Daten benötigt. Die Ergebnisse werden mit zunehmender Datenmenge genauer und reduzieren gleichzeitig die Fehlerquote.

**Listing 15: Auswertung**

```
result = model.predict(owncoded_token_pad)
print(result)

[[1.7368197e−03 9.5579028e−03 9.9944061e−01]
 [1.7224550e−03 9.6459389e−03 9.9950230e−01]
 [2.5086403e−03 1.6379207e−02 2.1177730e−01]
 [1.8664896e−03 9.2240870e−03 9.9823225e−01]
 [5.8086514e−03 2.0626158e−02 3.9716959e−03]
 [3.8374662e−03 1.6482115e−02 4.6136826e−02]
 [7.4589670e−01 1.0926694e−02 6.5103173e−04]
 [9.2276520e−01 9.9021196e−04 4.8039079e−02]]
```

## 7 GRAPHICAL USER INTERFACE

Auf Basis des trainierten Models wurde eine Webanwendung als GUI für eine einfachere Verwendung entwickelt. Die GUI bietet die Möglichkeit einen Satz einzugeben und zeigt anschließend die Auswertung des Netzwerks als Balkendiagramm an. Die GUI teilt sich in zwei Bereiche, einen Client teil welcher mithilfe von HTML5 und Javascript realisiert wurde, sowie einem Serverteil welcher als Python3 Skript realisiert wurde. Die Abbildung 15 zeigt das GUI in der finalen Version.

### 7.1 Serverteil

Der Serverteil wurde als Python3 Skript realisiert und verwendet die Python Bibliothek Flask welche eine Webserver bereitstellt und es so ermöglicht eine JSON API für das Netzwerk zu erstellen. Bei der Implementierung wurden nach kurzer Zeit Probleme ersichtlich, da Tensorflow offenbar mit Multithreading besitzt, sodass zurzeit das Model bei jeder Anfrage neu geladen wird, um diese Probleme zu vermeiden. Leider hat dies negative Auswirkungen auf die Zeit, die für eine Auswertung benötigt wird, welche mit diesem Ansatz bei rund 800 Millisekunden liegt. Dies ist für eine Produktionsumgebung zu viel, für einen Prototyp jedoch in Ordnung. Wird das Skript gestartet, so werden zuerst die vorgefertigten Stopp-Wörter Bibliotheken heruntergeladen und anschließend der trainierte Tokenizer wieder hergestellt. Anschließend werden 3 Routen für den mit Flask realisierten Webserver definiert. Die erste Route stellt eine JSON API bereit, welche es ermöglicht eine Prediction durchzuführen. Sie erwartet einen Text in einem JSON Objekt, lädt das gespeicherte Model, konvertiert den übergebenen Text mithilfe des Tokenizers in ein Zahlenarray und wendet alle Preprocessing Schritte an. Anschließend übergibt es an das Neuronale Netzwerk und gibt das Ergebnis als JSON verpackt zurück an den anfragenden Client. Die zweite Route dient dazu die statischen Dateien, welche die GUI darstellen auszuliefern und gibt lediglich den Inhalt der Dateien im Ordner „static“ zurück. Die dritte Route dient dazu den Standardpfad „/“ auf die index.html zu mappen damit nicht der exakte Dateiname im Browser eingegeben werden muss.

### 7.2 Clientteil

Als Client wurde eine einfache HTML5 Webseite entwickelt, welche die „Milligramm“ CSS Bibliothek verwendet und mithilfe eines kurzen Javascripts ein Balkendiagramm der Auswertung anzeigt. Zur Erstellung der Diagramme wurde die Javascript Bibliothek Charts.js verwendet. Alle benötigten Dateien werden durch den Flask Webserver bereitgestellt, sodass nach dem Starten des Serverskripts lediglich der Browser mit „http://localhost:5000/“ geöffnet werden muss, um die Benutzeroberfläche zu betrachten.

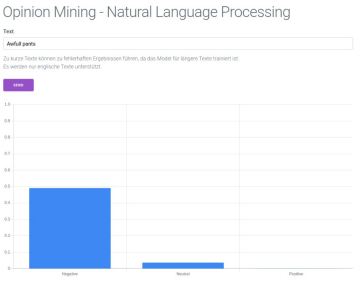



**Abbildung 15: Abbildung des GUI**

## 8 AUSBLICK

Um bessere Ergebnisse zu erzielen, wäre eine Möglichkeit Sarkasmus sowie Ironie erkennen zu können. Des Weiteren könnte durch Trainieren größerer Datenmengen, welche lange und kurze Texte enthalten, Fehlinterpretationen vermieden werden. Irgendwann erreicht dies allerdings eine Grenze. Die Zukunft des Opinion Mining, liegt in Predictive Analytics, was bedeuten würde, man könnte vorhersagen, wie ein Produkt bei den Kunden ankommt.

## LITERATUR

Thushan Ganegedara. 2018. *Natural Language Processing with TensorFlow*. packt. https://www.buecher.de/shop/fachbuecher/natural-language-processing-with-tensorflow-ebook-epub/ganegedara-thushan/products_products/detail/prod_id/54536439/

Giancarlo Zaccone. 2018. *Deep Learning with TensorFlow*. packt. https://www.bookdepository.com/Deep-Learning-with-TensorFlow-Giancarlo-Zaccone/9781788831109

# Can Computers Create Art?

Exploring the basics of neural style transfer

Kayla Kinadeter
Computer Science
Hof University of Applied Sciences
Hof, Bavaria, Germany
kayla.kinadeter@hof-university.de

## ABSTRACT

In the Applied Machine Learning seminar, students learned the basics of convolutional neural networks and how they are used to process and categorize images. Building on that knowledge, I looked into techniques using neural networks to generate images and decided to explore neural style transfer. This technique uses feature extraction layers of a pretrained network to combine two images, and "paint" the content of one image in the style of the other. I adapted an existing Colab demo to make it more user-friendly and try it out for myself.

## 1 Introduction

Tensorflow, a machine learning platform, provides a tutorial of style transfer as a demonstration of the Keras framework. They describe the process as follows:

"The principle of neural style transfer is to define two distance functions, one that describes how different the content of two images are, $L_{content}$ and one that describes the difference between two images in terms of their style, $L_{style}$. Then, given three images, a desired style image, a desired content image, and the input image (initialized with the content image), we try to transform the input image to minimize the content distance with the content image and its style distance with the style image. In summary, we'll take the base input image, a content image that we want to match, and the style image that we want to match. We'll transform the base input image by minimizing the content and style distances (losses) with backpropagation, creating an image that matches the content of the content image and the style of the style image." (Tensorflow Team, 2018)

As an example, they combine a photo of a sea turtle with Katsushika Hokusai's famous painting *The Great Wave off Kanagawa*.

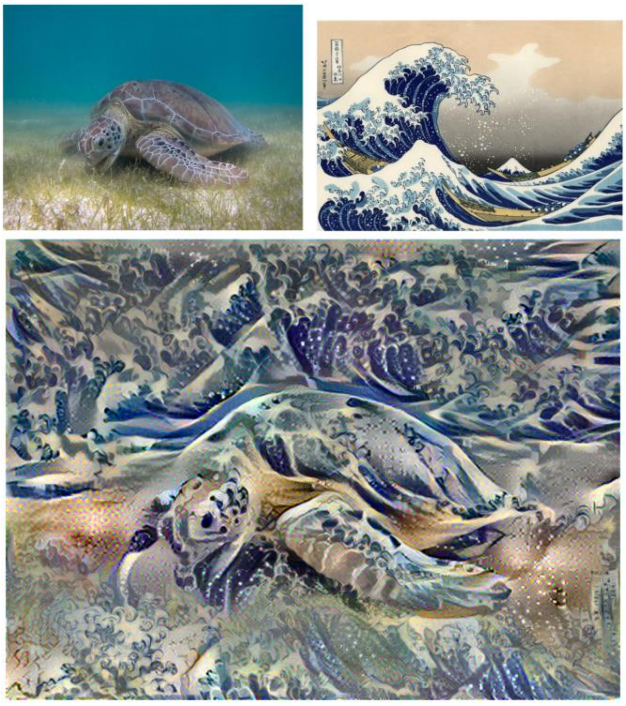


**Figure 1: Neural style transfer "paints" the photograph (content image) in the style of the artwork (style image). (Tensorflow Team, 2018)**

The tutorial is provided in the form of a Python notebook that the user can easily execute themselves to try it out. I used this code as a base for my own interactive demo, which can be found here:

https://colab.research.google.com/drive/1SoMa2oYcfQCBvLVXMOUuquBfDhBGXcfV

## 2 Theory

### 2.1 Pretrained Network- VGG19

Rather than training a network from scratch, style transfer uses a pretrained network as a base and repurposes it to create new output. To truly grasp how style transfer works, a solid understanding of the base network and convolutional neural networks in general was necessary. The demo briefly mentions which network was used and why it was chosen:

"We use VGG19, as suggested in the paper. In addition, since VGG19 is a relatively simple model (compared with ResNet, Inception, etc) the feature maps actually work better for style transfer." (Tensorflow Team, 2018)

VGG19 is a convolutional neural network (CNN) trained on the ImageNet dataset and was the winner of ImageNet's 2014 challenge in the category of object localization and classification. It distinguishes itself from other CNNs by being very deep- this version uses a stack of 19 convolutional layers, hence the name. In exchange the layers are small, with a receptive field of only 3 x 3, which keeps the total number of trainable weights (in VGG19, 144 million) in line with shallower nets. Pooling layers are interspersed between the convolutional layers, followed by three fully-connected layers and finally a soft max to classify images into one of 1000 categories. ReLU is used as the activation function. (Simonyan & Zisserman, 2015)

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64 **LRN** | conv3-64 **conv3-64** | conv3-64 conv3-64 | conv3-64 conv3-64 | conv3-64 conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128 **conv3-128** | conv3-128 conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 |
| maxpool | | | | | |
| conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 **conv1-256** | conv3-256 conv3-256 **conv3-256** | conv3-256 conv3-256 conv3-256 **conv3-256** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

**Figure 2: The creators of VGG experimented with different laver configurations (A through E), but the general structure of groups of convolutional layers separated by pooling layers remains the same.**

The original paper compared the effectiveness of different depths. 19 depth layers (here column E) was the most successful and was released to the public. It is available by default in Keras.

Parts of this network are used as the basis for the style transfer network. In the process of learning to classify images, a CNN breaks the image down into so-called "feature maps." Each convolutional layer of the network outputs a differently-filtered version of the input image depending on the particular feature that layer recognizes. (Gatys, Ecker, & Bethge, 2015) These features range from low-level information like edges to higher-level patterns or even entire objects.

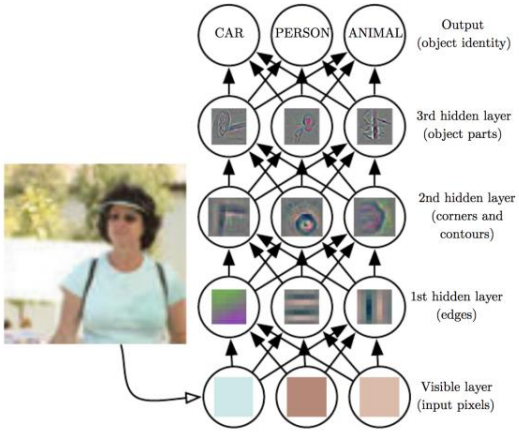



**Figure 3: Early convolutional layers react to simple forms like edges, while deeper layers can recognize complex forms. (Narayanan, 2017)**

The style transfer technique utilizes this knowledge of images to transform an input image rather than classify it. Therefore, it does not need the fully connected layers at the end to perform classification but only utilizes the output of selected convolutional layers. Additionally, it was found that exchanging the max pooling layers for average pooling yields slightly more appealing results. (Gatys, Ecker, & Bethge, 2015)

### 2.2 Defining Style and Content

In order to understand how these images are produced, it is necessary to define what exactly "style" and "content" are to a neural network. The original paper describing the style transfer technique explains it as follows:

As a CNN is trained on object recognition, its representation of an image becomes increasingly focused on the actual content of an image and not its exact pixel values. (Gatys, Ecker, & Bethge, 2015) The lower layers simply reconstruct the original pixel values, but the higher layers have a more "abstract" understanding of what the image contains and can be directly used as the *content representation.*


**Figure 4: Early layers of the CNN output nearly the exact pixel values of the original image, while deeper layers output the network's own concept of what defines the image. (Narayanan, 2017)**

Meanwhile, the "style" of an image is mainly the texture information without the global arrangement of the scene. It is built on top of the original CNN layers and computes correlations between features of different layers to create a general representation of the image in terms of color and localized structure. (Gatys, Ecker, & Bethge, 2015) This newly computed layer is used as the *style representation*.

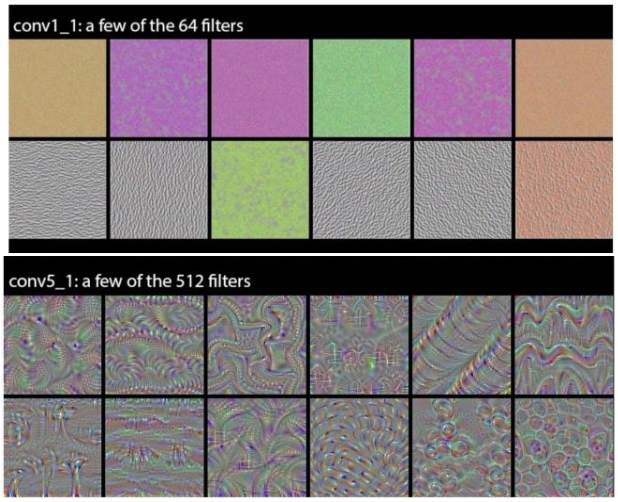



**Figure 5: Examples of textures recognized by early network layers (conv1_1) and deeper layers (conv5_1). (Singh, 2017)**

The demo code uses chosen layers for style and content that the authors found most effective, but it would be possible to select other layers and experiment with its effect on the results.

```
# Content layer where will pull our feature maps
content_layers = ['block5_conv2']

# Style layer we are interested in
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1'
               ]
```

**Figure 6: Code excerpt where the content and style layers for the style transfer network are selected from the original layers of the VGG19 network. (Tensorflow Team, 2018)**

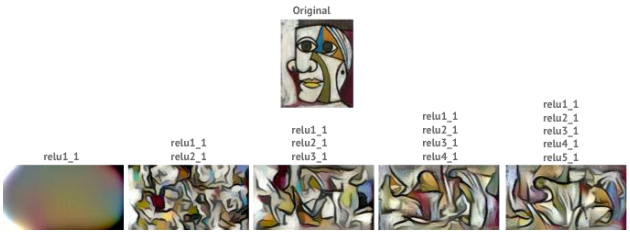



**Figure 7: Using more base layers (one from each group) when computing the style representation yields better results. (Singh, 2017)**

## 2.3 Loss Function

Importantly, these representations of style and content are separable, which allows us to combine the style of one image with the content of another. The goal of our neural network is to find an image that simultaneously matches the content of one input and the style of the other input. This is a question of optimization and can be achieved with the proper loss function.

The optimization loop begins with a white noise image and performs gradient descent to find another image that produces similar feature responses as the original image. This is performed separately for style and content using two different loss functions, which are described in detail in (Gatys, Ecker, & Bethge, 2015). For our purposes, we will look at the general loss function without going too deep into the details.

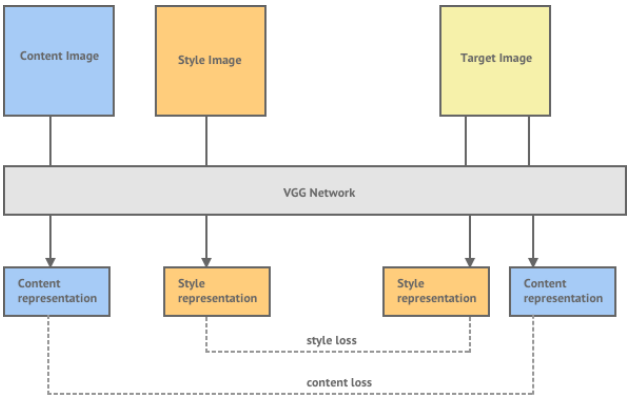



**Figure 8: The content image, style image and target image (originally noise) are repeatedly fed through the network and the style loss and content loss are computed. The target image is then changed using backpropagation to minimize loss. (Singh, 2017)**

The total loss function is a combination of the style and content loss functions, defined as follows:

"So let $\vec{p}$ be the photograph and $\vec{a}$ be the artwork. The loss function we minimise is

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

where $\alpha$ and $\beta$ are the weighting factors for content and style reconstruction respectively." (Gatys, Ecker, & Bethge, 2015)

The style transfer network requires no training; the weights alpha and beta are set by the user. Most of the example images shown used an alpha/beta ratio of 1 x $10^{-3}$ or 1 x $10^{-4}$, but it is possible to experiment with different ratios and produce images that are more or less stylized.

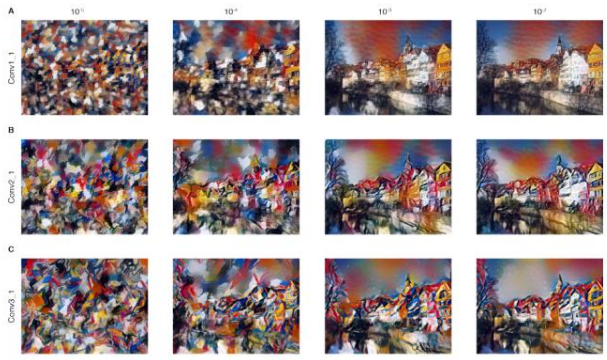


**Figure 9: The columns use alpha/beta ratios ranging from $10^{-5}$ to $10^{-2}$ and the rows are reconstructions of different layers of the network. A low ratio produces images with nearly no recognizable content, while a high ratio is only slightly stylized. Lower layers (the upper row) match the original pixels closely; higher layers become increasingly abstract. (Gatys, Ecker, & Bethge, 2015)**

## 3 Implementation

Since the style transfer network is built from a pretrained network and the weights for style and content are also explicitly specified rather than trained, there was no need for me to build and train a network of my own. I instead opted to tweak the demo to make it more user-friendly and accessible for someone with no knowledge of machine learning or programming.

The most important feature is, in my opinion, the ability to easily experiment with a variety of images. To that end I preloaded a number of royalty-free photographs and artworks from Wikimedia Commons that the user can choose from. Additionally, the user can upload images from their own computer or provide a link to an external image. The style and content images to use can be selected with Colab's user input fields.


**Figure 10: Fields where the user can enter image URLs or choose from preloaded images.**

In order to work with images from varying sources, tweaks had to be made to the image loading functions and others that worked with them. Mainly, an actual image object must be passed as a parameter rather than a file path, since user-chosen images are not saved to the virtual machine where the program is running.

Secondly, variables were introduced to give the user some control over the optimization loop. The number of iterations can be selected, along with the strength of stylization (the alpha/beta ratio described in 2.3).


**Figure 11: Input fields to select the number of iterations and different style and content weight ratios.**

It was also necessary to change the runtime type to GPU. This reduced the time to process an image in 1000 iterations from over an hour to under 5 minutes. Due to the high number of images being created and processed, a GPU is more suited to the task than a standard processor.

Other than the aforementioned changes, the code (including comments) is taken directly from the demo. The text between the code blocks was written by me, and should lead the user through the steps in the simplest way possible.

## 4 Results

The resulting program is a simple, accessible application that anyone can use to experiment with neural style transfer. Neural networks can seem like a daunting concept, especially to those outside of the IT field, so a hands-on application that quickly produces interesting results can be a valuable tool for sparking interest. My program aims to combine ease-of-use for the general public with the freedom of choice a curious developer wants.

Some examples of images I created:

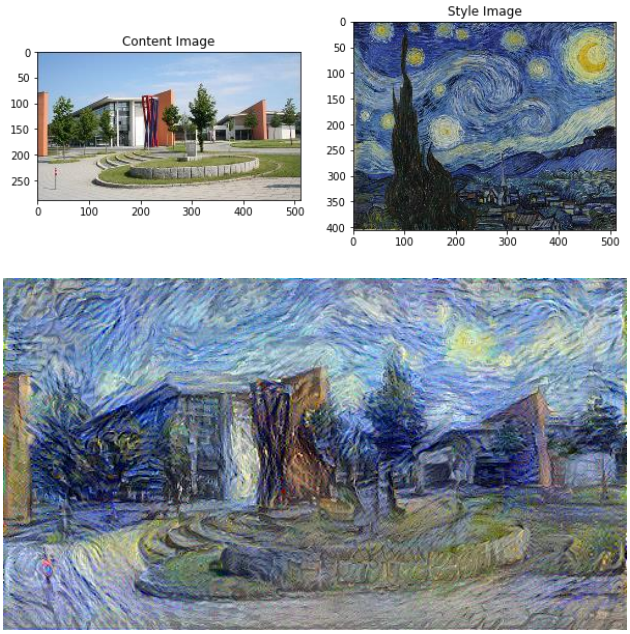



**Figure 12: Hof University combined with van Gogh's *Starry Night*; 1000 iterations with default weights.**


**Figure 13: The Eiffel Tower combined with Monet's *Femme à l'ombrelle*; 1000 iterations with default weights.**

**Figure 14: A cat combined with Picasso's *Three Musicians*. The first image is with style strength set to 1; the second with style strength set to 5.**

## 5 Looking Forward

There are a few ways my program could be extended to be more customizable. You could give the user the choice to select different layers as the base for content and style representations, or to use a different base network altogether. Improved CNNs are released yearly that could yield even better results than VGG19.

I find style transfer to be an interesting prospect for artists and non-artists alike. Typically, "creativity" is thought to be one of the main traits that separates humans from machines, but machines are now able to generate images that spark the same intrigue as human-created artwork. While I doubt that machines will ever be able to replace artists completely, I have hope that these new tools can be used to create entirely new and exciting genres of collaborative art.

## REFERENCES

Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). *A Neural Algorithm of Artistic Style.* Tübingen. Retrieved from https://arxiv.org/abs/1508.06576

Narayanan, H. (2017, March 31). *Convolutional neural networks for artistic style transfer*. Retrieved from harishnarayanan.org: https://harishnarayanan.org/writing/artistic-style-transfer/

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR.* Retrieved from https://arxiv.org/abs/1409.1556

Singh, M. (2017, September 4). *Artistic Style Transfer with Convolutional Neural Network.* Retrieved from Medium: https://medium.com/data-science-group-iitr/artistic-style-transfer-with-convolutional-neural-network-7ce2476039fd

Tensorflow Team. (2018, November 5). *Neural Style Transfer with tf.keras*. Retrieved from Github: https://github.com/tensorflow/models/blob/master/research/nst_blogpost/4_Neural_Style_Transfer_with_Eager_Execution.ipynb

# Audio-Klassifizierung mithilfe eines Convolutional Neural Networks anhand des Fallbeispiels von Tiergeräuschen

Kubilay Cinar
Hochschule Hof
kubilay.cinar@hof-university.de

## ZUSAMMENFASSUNG

Für die Thematik der maschinellen Audio-Klassifizierung haben sich bis heute zwei Ansätze bewährt. Zum Einen die Erkennung von Mustern anhand eines Spectograms und zum Anderen die Nutzung der herkömmlichen Wavelet-Form der jeweiligen Audiodaten. In dieser Arbeit wird der Ansatz der Erkennung über Wavelet erläutert und am Fallbeispiel des Datensatzes „Audio Cats and Dogs“ und dem Framework TensorFlow wird ein „Convolutional Neural Network“ (CNN) implementiert, das die Audiodateien des Datensatzes nach dem jeweiligen Tier klassifizieren kann. Im Zuge dessen wird auf die nötigen theoretischen Grundlagen eingegangen sowie Optimierungsstrategien bei Neuronalen Netzen mit solch kleinen Datensätzen vorgestellt.
Das Ergebnis der Arbeit ist ein zu 95,92 % genaues Neuronales Netz, dass durch die von dem Datensatz zur Verfügung stehenden Hunde- und Katzengeräusche lernt und diese erkennen kann.

• **Computing methodologies → Neural networks**; *Supervised learning by classification*; Modeling methodologies.

## KEYWORDS

datasets, neural networks, audio, optimization, CNN, tensorflow, keras, animal sounds, fine-tuning, hyperparameter

## 1 EINFÜHRUNG

„Künstliche Intelligenz“ (KI) scheint für das zweite Jahrzehnt des 21. Jahrhunderts das Schlagwort schlechthin zu sein. In der letzten Dekade hat sich das Thema KI, nicht zuletzt wegen der Entwicklung des Smartphones, immer mehr in den Mittelpunkt der Wissenschaft, unserer Gesellschaft und besonders der Medien gerückt. Dabei wird KI bereits seit 1956 thematisiert. Die damalige Dartmouth-Konferenz wird als offizielle Geburt des akademischen Feldes „künstliche Intelligenz“ bezeichnet. [20] Seitdem wird an zahlreichen Fachrichtungen der KI geforscht, die besonders an der kognitiven Fähigkeit und dem Aufbau des menschlichen Gehirns angelehnt ist. Zwar ist die Forschung von einer echten KI noch einige Jahre entfernt, jedoch ist die Subdisziplin „Machine Learning“ oder „Deep Learning“ bereits sehr tief im Alltag eingegliedert.

*FWPM: Angewandtes maschinelles Lernen, Sommersemenster, 2019, Hof*

Sowohl in betrieblicher Umgebung, bei der Neuronale Netze genutzt werden, um beispielsweise Qualitätswerte, Maschinenausfälle oder Engpässe vorherzusagen, als auch im alltäglichen Leben befinden sich zahlreiche Anwendungsfelder für maschinelles Lernen. Die meisten Anwendungen sind unscheinbar und im Hintergrund agierend. Apps, die der Großteil der Menschen tagtäglich nutzen, beinhalten bereits seit Jahren maschinelles Lernen, um beispielsweise unsere E-Mails zu filtern oder dafür zu sorgen, dass wir mit unserer Google-Suche tatsächlich das finden, wonach wir suchen. Neben den Anwendungsbeispielen, die sich eher im Hintergrund abspielen, gibt es aber auch Tools, die die Nutzer für die unmittelbare Mensch-Maschinen-Interaktion nutzen. Das beste Beispiel hierfür sind Sprachassistenten in Smartphones oder Smart-Home-Geräten. Diese nutzen die Kombination aus Audio-Erkennung und Neurolinguistik, um die Spracheingaben der Nutzer zu erfassen, zu verstehen und die entsprechenden Befehle auszuführen. Besonders die großen Tech-Konzerne Google und Co. forschen und perfektionieren seit Jahren diese Technologie, um sie nicht nur in ihren Endgeräten, sondern auch in den verschiedenen Betriebssystemen und Softwareprodukten zu implementieren und die Nutzung dieser Anwendungen zu erleichtern und natürlicher zu gestalten.
Ein weiteres Beispiel für die ausgereifte Nutzung von Audio-Erkennung ist der bekannte Musikerkennungs-Dienst „Shazam“, der seit 2002 seinen Nutzern die Möglichkeit anbietet, über die Smartphone-App und das Halten des Endgerätes in Richtung der Audioquelle, den Titel zu erkennen und über die zugehörige Datenbank weitere Infos wie Interpret, Album oder Erscheinungsjahr herauszufinden. [9] Dieser Prozess kann in drei große Schritte gegliedert werden:

(1) Konvertierung der physischen Schallwellen in ein digitales Signal mithilfe eines Analog-Digital-Koverters (A/D-Wandlung)
(2) Datenvorbereitung und Erkennung von Mustern, Phrasen oder Soundclustern durch ein Neuronales Netz
(3) Weiterverarbeitung der Klassifizierung beispielsweise durch Abgleich einer Datenbank und darauffolgender Ausführung von Befehlen oder Algorithmen

Diese Arbeit handelt von genanntem zweiten Schritt des Audio-Erkennungsprozesses. Mithilfe eines verständlichen Fallbeispiels wird ein Neuronales Netz entwickelt, dass eine binäre Klassifizierung von Hunde- und Katzengeräuschen aus Wavelet-Audiodaten durchführt. Dabei wird der Datensatz „Audio Cats and Dogs“ [13] von der öffentlich zugänglichen Datensatz-Plattform Kaggle genutzt. Das Neuronale Netz wird mithilfe von verschiedenen Frameworks wie TensorFlow in der Programmiersprache Python und der „Convolutional Neural Network“-Architektur (CNN) realisiert.
Bedingt durch die Datensatzgröße gestaltet sich ein optimales Erlernen der Klassifizierung durch das Netz recht schwierig. Aus diesem

Grund werden zudem Möglichkeiten für die Optimierung des Neuronalen Netzes präsentiert, bei denen festgelegte Hyper-Parameter des Models entsprechend kalibriert werden können, sodass auch bei kleinen Datensätzen eine akzeptable Genauigkeit und Performanz des Netzes zustandekommen kann.
Im ersten Kapitel nach der Einleitung werden zunächst verwandte Veröffentlichungen zu dem Thema Audio-Klassifizierung und Geräuscherkennung dargestellt. Im darauffolgenden Kapitel wird auf theoretische Grundlagen zu digitalen Audiosignalen, den Besonderheiten des in der Arbeit genutzten CNN und auf den Datensatz „Audio Cats and Dogs“ eingegangen. Im vierten Kapitel wird die Implementierung des Neuronalen Netzes erläutert. Anschließend befassen sich je ein Kapitel mit der möglichen Optimierung, den daraus resultierenden Ergebnissen und einem abschließendem Fazit.

## 2 VERWANDTE ARBEITEN

Im Bereich der Audio-Erkennung von Neuronalen Netzen gibts es zum Zeitpunkt des Schreibens dieser Arbeit vergleichsweise wenige thematisch ähnliche wissenschaftliche Arbeiten, ungleich der Anzahl derer über beispielsweise Bilderkennung. Besonders im Gebiet der Tiergeräuscherkennung ist eine gründliche Recherche nach verwandten Werken nötig. Nichtsdestotrotz werden im Folgenden Arbeiten vorgestellt, die neben Klassifizierung von Tiergeräuschen auch Themen enthalten, die mit dem Zweck dieser Arbeit korrelieren.
Um die Forschungsaktivität im Bereich der Audio-Erkennung von Neuronalen Netzen weiter voranzubringen und zu fördern, haben Gemmeke et al. [10] die Erstellung eines groß angelegten manuell gelabelten Audio-Datensatzes „AudioSet“ behandelt, das von Google als Open-Source-Datensatz zur freien Verfügung steht. Der Datensatz verfügt über 635 Audio-Klassen aus 10 sekündigen Audio-Events aus Youtube Videos mit entsprechender Dokumentation. So konnte de Benito-Gorron et al. [5] die von AudioSet zur Verfügung stehenden Daten nutzen, um verschiedene Arten von Neuronalen Netzen auf Audio-Erkennung zu testen. Dabei wurden sowohl „Deep Feed Forward Network“ (DFF) und CNN, als auch „Long/Short-Term Memory Network“ (LSTM) sowie „Recurrent Neural Network“ (RNN) [14] zum testen herangezogen. Das Ergebnis dieser Arbeit brachte de Benito-Gorron et al. zur Erkenntnise, dass ein CNN in Kombination mit LSTM in diesem Fall eine hohe Performanz von 85 % liefert.
Werden üblicherweise an Netzen zur Erkennung von Musik wie bei Shazam [9] oder für „Nautral Language Processing“ (NLP) geforscht, hat sich Bountourakis et al. [4] der automatischen Erkennung und Klassifizierung von Umweltgeräuschen mit dem Fokus auf reine Umweltgeräusche ohne Musik oder Sprache gewidment. Dabei wurden, ähnlich wie in dieser Arbeit, bereits etablierte Techniken und Architekturen aus den zwei anderen Bereichen der Audio-Erkennung adaptiert. Die Techniken wurden entsprechend verglichen, um eine passende Methode für die Klassifizierung diskreter Umweltgeräusche zu definieren.
Kiskin et al. [11] präsentiert in seiner Arbeit den Vergleich zweier CNNs mit verschiedenen Ansätzen die Daten vorzubereiten. Für die Datenvorbereitung werden hier zum Einen der konventionelle Ansatz der „Short-Time Fourier Tranformation“ [16] verwendet, das für die spektrale Frequenzteilung eines Audiosignals genutzt wird, und zum Anderen ein CNN genutzt, das auf einer Wavelet-Transformation (siehe Kapitel 3.1) eines Audiosignals angewendet wird. Zweiteres wird in derselben Variante auf den Datensatz „Audio Cats and Dogs“ [13] angewendet.
Im Themenbereich der Audio-Klassifizierung gibt es zudem wissenschaftliche Arbeiten, die sich mit medizinischen oder psychologischen Themen wie die „Audiovisuelle Erkennung von Emotionen“ von Avots et al. [1] befassen. Auch Aykanat et al. [2] schreibt über die Klassifizierung von Lungengeräuschen aus Aufzeichnung von Stethoskopen. Dabei greift Aykanat et al. auf 17930 Audioaufnahmen von Lungen von insgesamt 1630 Patienten zurück. Dieser große Datensatz wird für verschiedene Klassifizierungsvarianten genutzt, die nicht nur binärer Natur sind, wie beispielsweise Vergleich gesunder und pathologischer Lungengeräusche, sondern auch für mehrklassige Segmentierungen, die dann auch in einzelne Erkrankungen aufteilt.

## 3 THEORIE

### 3.1 Aufbau digitaler Wavelet-Audiosignale

Bei der Klassifizierung von Audio wird in der Regel eine Form von Fourier Transformation wie zum Beispiel die „Fast Fourier Transformation“ (FFT) [16] verwendet, um das Signal in seine einzelnen Frequenzteile über das ganze Spektrum hinweg aufzuteilen und diese als Spectogramm weiter zu verarbeiten beziehungsweise als tatsächliches Bild in ein Neuronales Netz einzugeben. So wird das Audiosignal im Grunde in ein dreidimensionales Bild verwandelt, in dem die x-Achse nach wie vor Zeit, die y-Achse nun aber die einzelnen Frequenzen anzeigt und zuletzt die Farbe des entsprechenden Pixels des Bildes den Pegel oder Ausschlag der Frequenz zu diesem Zeitpunkt (oder Sample) angibt. In Abbildung 1 kann man diese drei Dimensionen erkennen.

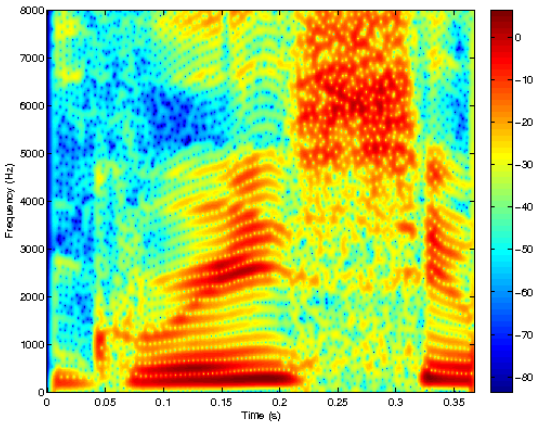



**Abbildung 1: Spectogramm eines Audiosignals [6]**

In dieser Arbeit hingegen werden die Audiosignale in ihrer ursprünglichen Wavelet-Form (siehe Abbildung 2) analysiert und antrainiert.

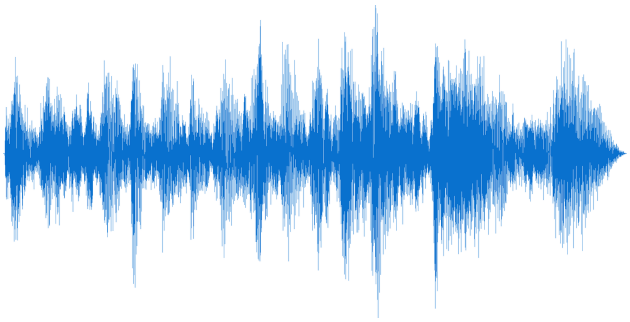


**Abbildung 2: Wavelet-Form eines Audiosignals [7]**

Eine Wavelet-Form ist im Grunde die digitale Übersetzung eines physischen Tons. Bei der A/D-Wandlung eines Tons werden die Pegel der Schallwellen abgetastet. Eine einzelne Abtastung wird Sample genannt und erfolgt bei Audio schrittweise in einem Frequenzbereich zwischen 32 und 96 kHz pro Sekunde Audiomaterial (siehe Abbildung 3). Der Standard für die Sampling-Rate eines digitalen Audiosignals ist 44.1 kHz (CD-Qualität), da bei dieser Rate die maximale Ton-Frequenz, die abgetastet werden kann, etwas über der Hälfte bei circa 22 kHz liegt und somit gerade so das Frequenzspektrum des menschlichen Gehörs wiedergeben kann. [8] Der Grund dafür ist das Nyquist-Shannon-Sampling-Theorem, das besagt, dass für die akurate Rekonstruktion eines Signals in der Bandbreite, welches vom menschlichen Gehör erfasst werden kann (also zwischen 20 Hz und 20 kHz), muss die Sampling-Rate mindestens doppelt so hoch sein wie die höchste zu samplende Frequenz. [15]

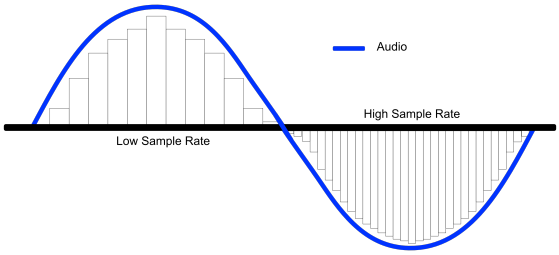



**Abbildung 3: Schaubild für die Sampling-Rate eines Signals [19]**

Weil Computer nicht kontinuierlich sondern in diskreten Werten rechnen, muss die Pegelwerterfassung ebenfalls schrittweise in Bit ablaufen. Pro Sample wird somit ein Pegelwert abgetastet, für das eine Anzahl von Bitwerten zu Verfügung steht. Diese Anzahl an Bitwerten, oder auch „Bit-Tiefe“ genannt, sagt aus wie präzise die Werte im Vergleich zum analogen Ton sind. Je höher die Bit-Tiefe desto akurater also die Wandlung des Tons in ein digitales Audiosignal. [15] Bei einer Bit-Tiefe von 16 Bit können beispielsweise 65.536 mögliche Pegelwerte pro Sample abgetastet werden. Das heißt, dass der Dynamikumfang wesentlich besser wiedergegeben werden kann im Vergleich zu einer 8 Bit-Tiefe, die nur 256 mögliche Pegelwerte pro Sample erfassen könnte. In Abbildung 4 werden die zwei Standardwerte dargestellt, die 16 und 24 Bit sind. [15]

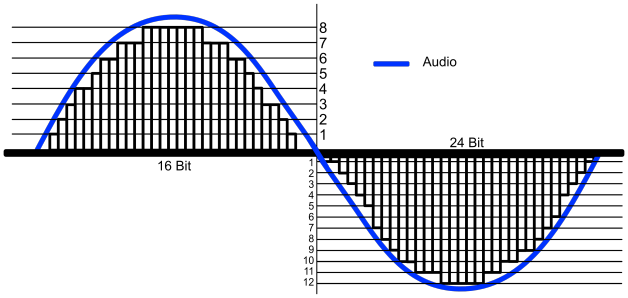



**Abbildung 4: Schaubild für die Bit-Tiefe eines Signals [19]**

Die Audiodateien in dieser Arbeit haben eine Sample-Rate von 16 kHz und eine Bit-Tiefe von 16 Bit. Daraus lässt sich schließen, dass hohe Frequenzen bedingt durch die niedrige Abtastrate nicht wiedergegeben werden können. Praktisch heißt das, dass die Hunde- und Katzengeräusche vermutlich sehr dumpf klingen, aber durch die 16 Bit-Tiefe ausreichend Dynamikumfang haben und sehr viele verschiedene Pegelwerte pro Sample annehmen können, was eine positive Auswirkung für den Trainingsprozess des Neuronalen Netz haben wird.

## 3.2 Eindimensionale CNN

Convolutional Neural Networks haben sich besonders für Anwendungsfälle für Bilderkennung erwiesen und werden mittlerweile sehr oft für die Erkennung von statischen Bildern genutzt. Jedoch haben sich CNN auch für die Erkennung von Audio als eine gute Wahl etabliert, da die Feature-Erkennung adaptiv auch auf Audio-Spectogramme angewendet werden kann. Ein Neuronales Netz würde somit ebenfalls Features im Spectogramm erlernen, dass zur korrekten Klassifizierung des entsprechenden Audiosignals führt. Entsprechende Beispiele sind in Kapitel 2 aufgeführt.
Für den Anwendungsfall dieser Arbeit wird ebenfalls ein CNN in Betracht gezogen, obwohl die zu analysierenden Audiodateien nicht mithilfe der FFT in Spectogrammbilder umgewandelt wird. Tatsächlich kann für die Feature-Erkennung in einem CNN eine eindimensionale Input-Matrix verwendet werden. Diese erfolgt somit mit eindimensionalen Filtern, oder auch Kernels genannt, die je nach Größe über einen Bereich an Samples rechnet. Bei einer Kernel-Größe von beispielweise 3 werden Pegelwerte von jeweils 3 Samples verrechnet und ausgegeben. Zur Veranschaulichung wird in Abbildung 5 eine Beispielrechnung eines eindimensionalen „Convolutional-Layers“ dargestellt.
So können alle Funktionen eines CNN wie zum Beispiel MaxPooling-Schichten ebenfalls für das Training eines eindimensionalen Inputs genutzt werden.

## 3.3 Datensatz „Cats and Dogs“

Der Datensatz „Audio Cats and Dogs“ beinhaltet 277 gekennzeichnete Sounddateien im Waveform Audio File Format (WAVE) und

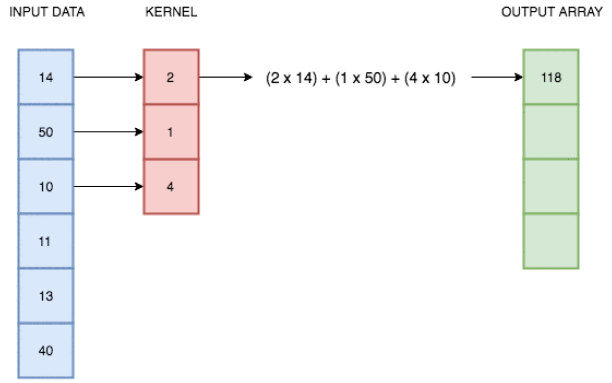



**Abbildung 5: Beispielrechnung eines eindimensionalen Inputs und einer Kernelgröße von 3 [12]**

| | Test | Training | Summe |
|---|---|---|---|
| Dog | 49 | 64 | 113 |
| Cat | 49 | 115 | 164 |
| Summe | 98 | 179 | **277** |

**Abbildung 6: Aufteilung des Datensatz in Trainings- und Testdaten [13]**

wurde von Moreaux [13] auf kaggle.com zur freien Verfügung gestellt. Moreaux hat die Dateien von dem AE-Datensatz von Takahashi et al. [18] extrahiert und entsprechend gelabelt. Der Datensatz beinhaltet zwei Klassen „Dog“ und „Cat“ und wird mithilfe einer CSV-Datei in Test- und Trainings-Daten aufgeteilt, wobei die Aufteilung in Abbildung 6 dargestellt ist.
Bedingt durch die sehr kleine Anzahl an Daten ist in dieser Arbeit besonders die Generalisierung der Trainingsergebnisse eine Herausforderung. In Kapitel 6 werden mögliche Optimierungen in Betracht gezogen, um die Größe des Datensatzes zu kompensieren.

# 4 IMPLEMENTIERUNG

Für die Implementierung wurden die zahlreichen Open-Source Kernels auf der Seite des Datensatzes auf kaggle.com [13] herangezogen, die verschiedene Ansätze für den Bau des Neuronalen Netzes anbieten und von verschiedenen Nutzern der Plattform hochgeladen wurden. Die Implementierung für diese Arbeit orientiert sich weitestgehend am Kernel von Zhou [22]. In diesem Kapitel wird auf die wichtigsten Funktionen des Codes eingegangen. Der gesamte Source-Code befindet sich im Anhang dieser Arbeit. Die Implementierung erfolgt mit dem Online-Tool „Google-Colaboratory“, das für das Training von Neuronalen Netzen besonders praktisch ist, da die Rechenleistung für den Trainingsprozess von externen Ressourcen, also extra Server-Ressourcen von Google, bereitgestellt wird.

## 4.1 Datenvorbereitung

Für den Zugang des Colab-Notebooks zum Datensatz wird zunächst der Zugriff zum Google-Drive-Verzeichnis benötigt, indem der Datensatz vorher gespeichert wurde (siehe Auflistung 1).

**Auflistung 1: Verbindung zum GoogleDrive-Verzeichnis**

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Wie in Kapitel 3.1 erwähnt ist die Sampling-Rate der Audiodateien in dem Datensatz 16 kHz. So wird diese auch dafür genutzt, um die Audiodateien auf den gleichen Nenner zu bringen, indem die Audiodateien, die länger als 20 Sekunden sind, auf 20 runtergeschnitten werden (siehe Auflistung 2). Ebenso wichtig wird im späteren Verlauf eine mögliche Down-Sampling-Rate sein, die, wie der Begriff bereits andeutet, die ursprüngliche Sampling-Rate um einen Faktor herunterbricht und die Anzahl an Samples der Audiodaten massiv verkleinert. Dabei ist natürlich zu bemerken, dass dann wesentlich weniger Pegelwerte für das Lernen zur Verfügung stehen, da beim Down-Sampling Pegelwerte entsprechend einfach wegfallen.

**Auflistung 2: Setzen der globalen Variablen**

```
WAVE_FOLDER = '/content/gdrive/MyDrive/audio-cats-and-
    ↪ dogs/cats_dogs'
#Sampling-Rate
FRAMERATE = 16000
#max 20sec
MAX_WAV_SAMPLES = 20*FRAMERATE
#Down-Sampling variables
DOWNSAMPLING_SCALE = 1
```

Im nächsten Schritt wird die CSV-Datei geladen, die die Dateinamen beinhaltet und diese in die vier Kategorien verteilt wie in der Matrix in Abbildung 6 zu sehen ist. In Auflistung 3 ist dargestellt wie die Spalten der CSV-Datei jeweils in Data-Frames geladen und anschließend jeweils in Trainings- und Test-Set zusammengeführt werden.

**Auflistung 3: Erstellung des Trainings- und Test-Set**

```
1  #load csv from gdrive
2  df = pd.read_csv("/content/gdrive/My_Drive/audio-cats
       ↪ -and-dogs/train_test_split.csv")
3
4  #load into data-frames
5  test_cat = df[['test_cat']].dropna().rename(index=str,
       ↪  columns={"test_cat": "file"}).assign(label=0)
       ↪
6  test_dog = df[['test_dog']].dropna().rename(index=str,
       ↪  columns={"test_dog": "file"}).assign(label=1)
       ↪
7  train_cat = df[['train_cat']].dropna().rename(index=
       ↪ str, columns={"train_cat": "file"}).assign(
       ↪ label=0)
8  train_dog = df[['train_dog']].dropna().rename(index=
       ↪ str, columns={"train_dog": "file"}).assign(
       ↪ label=1)
9
10 # Concat into test- and training-set
```

```
test_df = pd.concat([test_cat, test_dog]).reset_index
    ↪ (drop=True)
train_df = pd.concat([train_cat, train_dog]).
    ↪ reset_index(drop=True)
```

## 4.2 Vorbereitung der Audiofiles

Zur Veranschaulichung der Audiodateien werden in Auflistung 4 zunächst zufällige Audiodateien von beiden Kategorien geladen und visualisiert. Der Output wird in Abbildung 7 dargestellt.

**Auflistung 4: Visualisierung der Wavelets**

```
# Plot of 5 random files from both classes
for i in range(0,10,2):
    plot_wavelet(os.path.join(WAVE_FOLDER, test_cat.
        ↪ iloc[i]['file']))
    plot_wavelet(os.path.join(WAVE_FOLDER, test_dog.
        ↪ iloc[i]['file']))
```

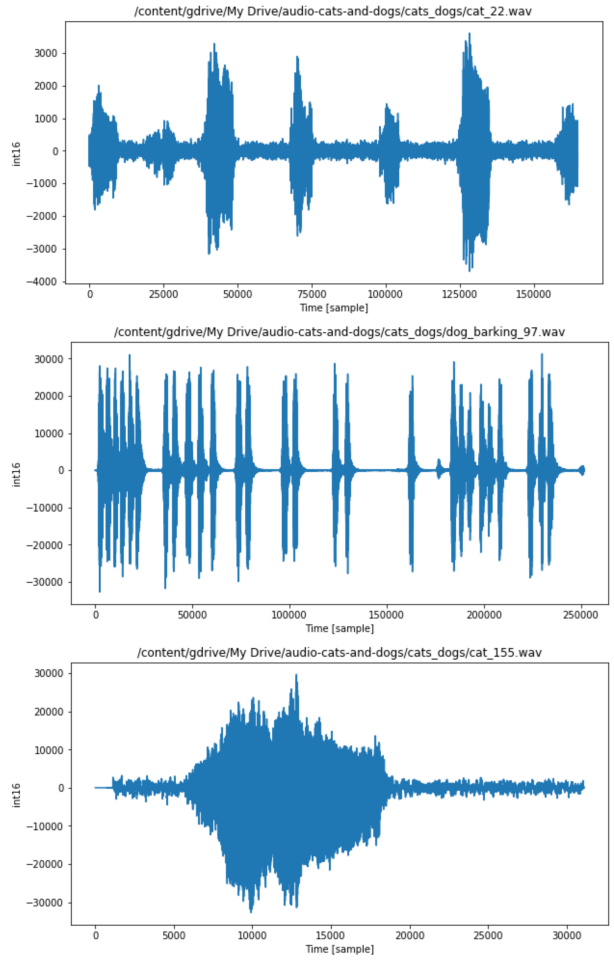



**Abbildung 7: Als Wavelet geplottete zufällige Audiodateien**

Aus Abbildung 6 ist leicht erkennbar, dass die Häufigkeiten beider Klassen ungleichmäßig sind. Um dem entgegen zu wirken wird in Auflistung 5 ein Random-Oversampler genutzt, um diese Häufigkeiten auszugleichen und einige Dateien und zugehörige Labels in der Klasse „Dog“ zu duplizieren. Anschließend wird die Reihenfolge der Dateien und ihren Labels randomisiert.

**Auflistung 5: Random-Oversampler für den Ausgleicher beider Klassen**

```
# RandomOversampler leveling both classes
random_oversampler = RandomOverSampler()
idx = np.arange(0, len(train_df)).reshape(-1, 1)
idx_sampled, _ = random_oversampler.fit_sample(idx,
    ↪ train_df['label'])
train_files, train_labels = train_df.iloc[idx_sampled.
    ↪ flatten()]['file'].values, train_df.iloc[
    ↪ idx_sampled.flatten()]['label'].values
train_files, train_labels = sklearn.utils.shuffle(
    ↪ train_files, train_labels)
test_files, test_labels = test_df['file'].values,
    ↪ test_df['label'].values
```

## 4.3 Aufbau des Models

Für den Aufbau des CNN-Models wird wie in Kapitel 3.2 bereits erwähnt, eindimensionale CNN-Schichten genutzt mit jeweils 32 Knoten genutzt. Die Filtergröße wird für die grundlegende Implementierung auf fünf sowie die Aktivierungsfunktion auf die beliebte „Rectified Linear Units“ (RELU) festgelegt. Die Initialisierung der Kernel-Werte erfolgt mit der von der Keras API vorgefertigten „Orthogonal“-Funktion, die randomisiert nach einem bestimmten Schema für die Filter erstellt. Im Anschluss zum eindimensionalen Convolutional-Layer wird ein Max-Pooling-Schicht sowie eine Batch-Normalisierungs-Schicht hinzugefügt. Dieser Satz an drei Schichten wird 14-fach wiederholt, sodass 14 Convolutional-Layer im Model platziert sind.

Die letzten Schichten schließen wie in üblichen CNN das Model mit einer „Flatten“-Schicht und einer „Fully Connected“-Schicht ab, wobei eine weitere Batch-Normalisierung dazwischengeschalten wird, um die Werte nochmals auf den gleichen Nenner zu bringen. Für die vollverbundene letzte Schicht wird die oft genutzte Softmax-Aktivierungsfunktion genutzt.

Das ganze Model wird sukzessive mit der Optimierungsfunktion „Adam“ um den Wert 0.01 angepasst und nutzt für als Verlustfunktion die „Sparse-Categorical-Crossentropy“. Das ganze Model wird in Auflistung 6 dargestellt.

**Auflistung 6: Aufbau des CNN-Models mit der Keras API**

```
#Sequential-model
model = tf.keras.models.Sequential()
#Reshape input
model.add(tf.keras.layers.Reshape((MAX_WAV_SAMPLES//
    ↪ DOWNSAMPLING_SCALE,1), input_shape=(
    ↪ MAX_WAV_SAMPLES//DOWNSAMPLING_SCALE,)))
for i in range(14):
    #1D-Conv-Layer
    model.add(tf.keras.layers.Conv1D(32,
```

```
8           kernel_size=5,
9           padding='same',
10          activation='relu',
11          kernel_initializer=tf.keras.initializers.
                ↪ Orthogonal(),))
12      #MaxPooling
13      model.add(tf.keras.layers.MaxPooling1D(pool_size
            ↪ =3, strides=2))
14      #BatchNormalization
15      model.add(tf.keras.layers.BatchNormalization())
16  #Flatten-Input
17  model.add(tf.keras.layers.Flatten())
18  #BatchNormalization
19  model.add(tf.keras.layers.BatchNormalization())
20  #FullyConnected
21  model.add(tf.keras.layers.Dense(2, activation='
        ↪ softmax'))
22  #Compile with Adam-Optimizer at 0.01 and
        ↪ sparse_categorical_crossentropy
23  model.compile(optimizer=tf.keras.optimizers.Adam
        ↪ (0.01),
24              loss=tf.keras.losses.
                    ↪ sparse_categorical_crossentropy,
25              metrics=['accuracy'])
26  model.summary()
```

## 5 ERGEBNISSE

Wird das neuronale Netz wie in Kapitel 4 beschrieben trainiert, wird ein Netz mit 53 Schichten und insgesamt 72.418 Parametern generiert, von denen 70.370 trainierbar sind. Die Genauigkeit von diesem Netz beträgt nach 15 Epochen schließlich 82,65 %. Dieser Wert wird in der Optimierungsphase als Vergleichsansatz verwendet und als „Grundimplementierung “ benannt. Die Abbildung 9 stellt den Verlauf der Genauigkeit von Training- und Test-Set dar. Die durch das Test-Set berechnete Validierungs-Genauigkeit und die Konfusionsmatrix in Abbildung 8 zeigen, dass insgesamt 17 Test-Dateien falsch erkannt wurden. Weitere Information zu den wichtigsten Hyper-Parametern ist dem Anhang zu entnehmen.
Eine Genauigkeit von 82,65 % ist für ein neuronales Netz zwar bereits sehr gut, bedeutet bei so einem kleinen Datensatz jedoch, dass 17 der 98 Test-Dateien falsch vorhergesagt werden. Während man bei sehr großen Datensätzen mit mehreren Tausend Daten 83 %-ige Akkuratesse als gut befindet, muss in diesem Fall das Netz in puncto Genauigkeit auf mindestens 90 % optimiert werden. Im folgenden Kapitel werden verschiedene Ansätze zur Optimierung erläutert und einige dieser auf das CNN angewendet, um die Zielgenauigkeit von über 90 % zu erreichen.

## 6 OPTIMIERUNG DES NETZES

Mit dem Thema „Optimierung eines Neuronalen Netzes“ befassen sich immer mehr wissenschaftliche Arbeiten. Beispiele hierfür sind zum Einen Yosinski et al. [21], der in seiner Arbeit über die Optimierungsstrategie über Transferierung von Features aus anderen vortrainierten Neuronalen Netzen berichtet. Statt einer randomisierten Initialisierung der Netz-Parameter, wird hier ein bereits auf einen ähnlichen Datensatz trainiertes Netz geladen und mit den

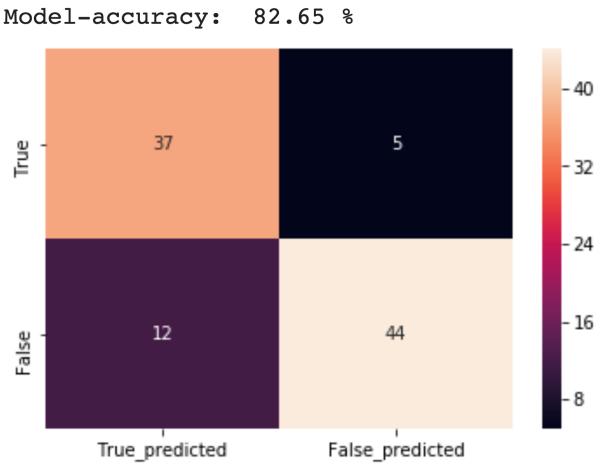



**Abbildung 8: Konfusionsmatrix des Netzes in der Grundimplementierung**

datensatz-spezifischen Features und Schichten verfeinert. [21] Zum Anderen hat sich Becherer et al. [3] mit dem Parameter-Fine-Tuning beschäftigt, dass sowohl Transfer-Learning wie sie Yosinski et al. nutzt, aber auch einzelne Hyper-Parameter gezielt umkalibriert, um die Performanz des Neuronalen Netzes zu erhöhen.
In dieser Arbeit werden bestimmte Hyper-Parameter entsprechend angepasst und systematisch nach einer Optimierung geprüft. Dabei werden insgesamt 14 Tests durchlaufen und sukzessiv Parameter verändert. Entsprechende Protokollierung der Hyper-Parameter der einzelnen Durchläufe sind im Anhang aufzufinden.
Versucht man die Performanz eines Neuronalen Netzes zu optimieren, gibt es verschiedene Methoden, die besonders in diesem Fall zur Wahl stehen. Dabei unterscheidet man in Maßnahmen zur Optimierung der Genauigkeit und der Lernzeit. Für Ersteres können folgende Maßnahmen in Betracht gezogen werden:

- Nutzung von Maxpooling-Schichten
- Hinzufügen einer Dropout-Schicht , um Overfitting zu vermeiden
- Data-Augmentation, welches die zur Verfügung stehenden Daten dupliziert und gewisse Anomalien hinzufügt, um sie dem Netz als zusätzliche Daten anzubieten

Für die Optimierung in der Lernzeit können folgende Schritte hilfreich sein:

- Verkleinerung der Filtergröße in den hinteren Schichten
- Batch-Normalisierung
- Downsampling des Audiomaterials

Für den Anwendungsfall dieser Arbeit haben sich die meisten der oben genannten Punkte bewährt. In der Grundimplementierung ist ohnehin Batch-Normalisierung sowie die zahlreiche Maxpooling-Schichten zwischen den Convolutional-Schichten aufzufinden. Im Verlauf der Durchläufe werden nach und nach weitere der Optimierungspunkte hinzugefügt, sodass am Ende das in Auflistung 7 dargestellte Model **eine Genauigkeit von 95,92 %** erreicht.

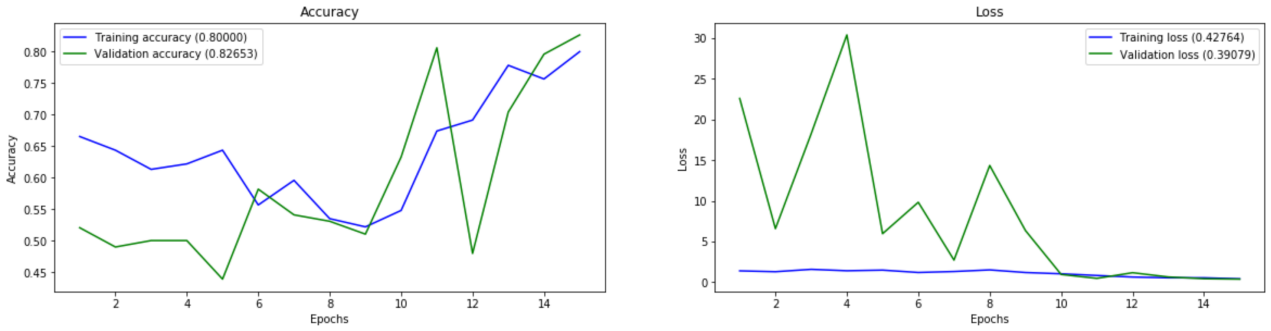



**Abbildung 9: Genauigkeit des Netzes in der Grundimplementierung**

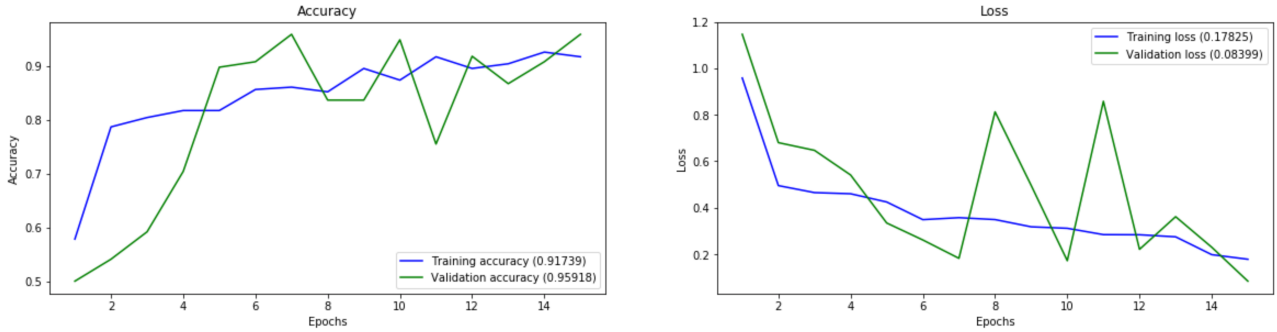



**Abbildung 10: Genauigkeit des optimierten Netzes**

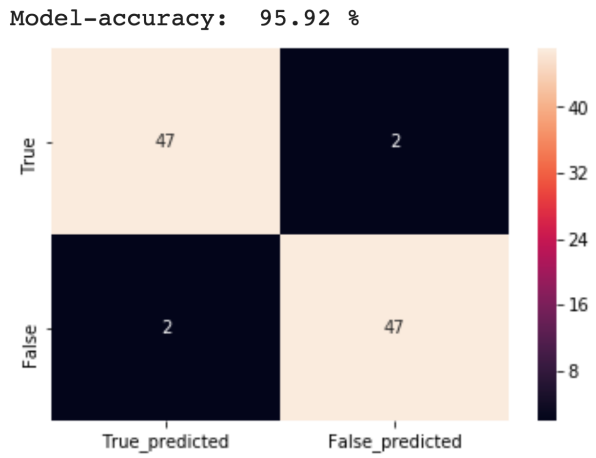



**Abbildung 11: Konfusionsmatrix des optimierten Netzes**

Das Model hat hier eine erste Convolutional-Schicht mit 64 Neuronen und einer Filtergröße von 5, während die restlichen 32 Neuronen und jeweils eine Filtergröße von 3 haben. Vor der Dense-Schicht wird eine Dropout-Schicht mit 25 % platziert, die Overfitting verhindern soll, die bei so kleinen Datensätzen oft auftritt. Darüber hinaus wird der Wert für den Optimizer auf 0.005 gesetzt, dass eine granularere Annäherung zum Minimum der Verlustfunktion mit sich bringt. Für dieses Netz hat sich schließlich eine Batch-Größe von 5 und keine Vermehrung der Epochen bewährt. Auf Downsampling und Data-Augmentation wurde an dieser Stelle verzichtet, da das Netz bereits eine hohe Performanz aufweist und lediglich 9 Sekunden pro Epoche benötigt. In Abbildung 11 ist schließlich die neue Konfusionsmatrix zu sehen, bei der man erkennt, dass bei einer Genauigkeit von 95,92 % insgesamt lediglich 4 falsche Vorhersagen getroffen wurden. Zum Vergleich wird in Abbildung 10 der Kurvenverlauf der Genauigkeit des optimierten Netzes dargestellt.

**Auflistung 7: Aufbau des optimierten CNN-Models**

```
1 model = tf.keras.models.Sequential()
2 model.add(tf.keras.layers.Reshape((MAX_WAV_SAMPLES//
      ↪ DOWNSAMPLING_SCALE,1), input_shape=(
      ↪ MAX_WAV_SAMPLES//DOWNSAMPLING_SCALE,)))
3 model.add(tf.keras.layers.Conv1D(64, kernel_size=5,
4       padding='same',
5       activation='relu',
6       kernel_initializer=tf.keras.initializers.
            ↪ Orthogonal(),))
7 model.add(tf.keras.layers.MaxPooling1D(pool_size=3,
      ↪ strides=2))
8 model.add(tf.keras.layers.BatchNormalization())
9 for i in range(15):
```

```
10      model.add(tf.keras.layers.Conv1D(32, kernel_size
            ↪ =3,
11          padding='same',
12          activation='relu',
13          kernel_initializer=tf.keras.initializers.
                ↪ Orthogonal(),))
14      model.add(tf.keras.layers.MaxPooling1D(pool_size
            ↪ =3, strides=2))
15      model.add(tf.keras.layers.BatchNormalization())
16  model.add(tf.keras.layers.Flatten())
17  model.add(tf.keras.layers.BatchNormalization())
18  model.add(tf.keras.layers.Dropout(0.25))
19  model.add(tf.keras.layers.Dense(2, activation='
        ↪ softmax'))
20  model.compile(optimizer=tf.keras.optimizers.Adam
        ↪ (0.005),
21      loss=tf.keras.losses.
            ↪ sparse_categorical_crossentropy,
22      metrics=['accuracy'])
23  model.summary()
```

## 7 FAZIT

Betrachtet man die Ergebnisse dieser Arbeit, kann gesagt werden, dass die unübliche Variante der Audio-Klassifizierung mit Wavelet-Daten durchaus zu sehr guter Genauigkeit des Neuronalen Netzes führen kann. Um diese noch weiter Richtung 100 % zu optimieren benötigt es noch weitaus feinere Parameterkalibrierung. Zudem sollte nicht außer Acht gelassen werden, dass der Audio Cats and Dogs-Datensatz [13] zu einem der kleinsten Datensätze gehört und rein für Demonstrationszwecke nutzbar ist.

Darüber hinaus besteht noch weiterhin die Möglichkeit den Ansatz von Yosinski et al. [21] zu nutzen, um durch Transfer-Features bereits eine grundlegende Genauigkeit mitzubringen und diese durch das von dieser Arbeit genutzte Neuronale Netz zu erweitern. Ebenfalls möglich ist eine zweistufige Audio-Klassifizierung, die Scholler and Purwins [17] in seiner Arbeit über die Audio-Klassifizierung von Percussionsinstrumenten nutzt. Dieser ist aus der Biologie inspiriert und nutzt im ersten Schritt „Unsupervised Learning“, um zunächst die Schlagzeug-Phrasen zu erkennen. Im zweiten Schritt werden dann einzelne Instrumente des Schlagzeugs extrahiert. Der einfachheithalber konkretisiert sich Scholler and Purwins dabei auf 3 Klassen (Bass Drum, Snare, High-Hat). Durch diese Art von zwei aufeinanderfolgenden Netzen wird in diesem Beispiel ebenfalls eine Genauigkeit von 91,7 % erlangt.

Abschließend kann gesagt werden, dass bei der Klassifizierung von Audiodaten im ersten Schritt betrachtet werden muss, um welche Art von Audio es sich bei dem zu trainierenden Datensatz handelt. Nach den Ergebnissen dieser Arbeit kann durchaus die Variante der Wavelet-Daten genutzt werden. Jedoch weist diese Methode einen erheblichen Nachteil auf, wenn man Audiodaten behandelt, die mehrere Spuren oder Tonquellen aufweisen. Für die von Shazam [9] genutzte Musikanalyse muss nach wie vor die Fast Fourier-Transformation genutzt werden, um dann auch die einzelnen Frequenzbänder betrachten zu können. Möchte man jedoch einzeln dastehenden Ton verarbeiten, erweist sich, wie de Benito-Gorron et al. [5] ebenfalls erläutert, die Wavelet-Form der Audiodaten als die bessere Wahl.

## LITERATUR


[1] Egils Avots, Tomasz Sapiński, Maie Bachmann, and Dorota Kamińska. 2019. Audiovisual emotion recognition in wild. *Machine Vision and Applications* 30, 5 (01 Jul 2019), 975–985. https://doi.org/10.1007/s00138-018-0960-9

[2] Murat Aykanat, Özkan Kılıç, Bahar Kurt, and Sevgi Saryal. 2017. Classification of lung sounds using convolutional neural networks. *EURASIP Journal on Image and Video Processing* 2017, 1 (11 Sep 2017), 65. https://doi.org/10.1186/s13640-017-0213-2

[3] Nicholas Becherer, John Pecarina, Scott Nykl, and Kenneth Hopkinson. 2017. Improving optimization of convolutional neural networks through parameter fine-tuning. *Neural Computing and Applications* (25 Nov 2017). https://doi.org/10.1007/s00521-017-3285-0

[4] Vasileios Bountourakis, Lazaros Vrysis, and George Papanikolaou. 2015. Machine Learning Algorithms for Environmental Sound Recognition: Towards Soundscape Semantics. In *Proceedings of the Audio Mostly 2015 on Interaction With Sound (AM '15)*. ACM, New York, NY, USA, Article 5, 7 pages. https://doi.org/10.1145/2814895.2814905

[5] Diego de Benito-Gorron, Alicia Lozano-Diez, Doroteo T. Toledano, and Joaquin Gonzalez-Rodriguez. 2019. Exploring convolutional, recurrent, and hybrid deep neural networks for speech and music detection in a large audio dataset. *EURASIP Journal on Audio, Speech, and Music Processing* 2019, 1 (17 Jun 2019), 9. https://doi.org/10.1186/s13636-019-0152-1

[6] Peter L. Søndergaard et. al. 2019. The Large Time-Frequency Analysis Toolbox. (2019). Retrieved Juli 07, 2019 from https://ltfat.github.io

[7] Meir Feinberg. 2015. How to Generate Waveform Images From Audio Files. (2015). Retrieved Juli 07, 2019 from https://cloudinary.com/blog/how_to_generate_waveform_images_from_audio_files

[8] Mitch Gallagher. 2015. 7 Questions About Sample Rate. (2015). Retrieved Juli 07, 2019 from https://www.sweetwater.com/insync/7-things-about-sample-rate/

[9] Coding Geek. 2015. How does Shazam work. (2015). Retrieved Juli 06, 2019 from http://coding-geek.com/how-shazam-works/

[10] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. 2017. Audio Set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*. New Orleans, LA.

[11] Ivan Kiskin, Davide Zilli, Yunpeng Li, Marianne Sinka, Kathy Willis, and Stephen Roberts. 2018. Bioacoustic detection with wavelet-conditioned convolutional neural networks. *Neural Computing and Applications* (01 Aug 2018). https://doi.org/10.1007/s00521-018-3626-7

[12] Samuel Lynn-Evans. 2018. Reading Minds with Deep Learning. (2018). Retrieved Juli 07, 2019 from https://blog.floydhub.com/reading-minds-with-deep-learning/

[13] Marc Moreaux. 2017. Audio Cats and Dogs. (2017). Retrieved Juli 06, 2019 from https://www.kaggle.com/mmoreaux/audio-cats-and-dogs

[14] Julian Möser. 2017. Künstliche neuronale Netze - Aufbau und Funktionsweise. (2017). Retrieved Mai 25, 2019 from https://jaai.de/kuenstliche-neuronale-netze-aufbau-funktion-291/

[15] Presonus. 2019. Digital Audio Basics: Sample Rate and Bit Depth. (2019). Retrieved Juli 07, 2019 from https://www.presonus.com/learn/technical-articles/sample-rate-and-bit-depth

[16] V. U. Reddy. 1998. On fast Fourier transform. *Resonance* 3, 10 (01 Oct 1998), 79–88. https://doi.org/10.1007/BF02841425

[17] Simon Scholler and Hendrik Purwins. 2010. Sparse Coding for Drum Sound Classification and Its Use As a Similarity Measure. In *Proceedings of 3rd International Workshop on Machine Learning and Music (MML '10)*. ACM, New York, NY, USA, 9–12. https://doi.org/10.1145/1878003.1878007

[18] Naoya Takahashi, Michael Gygli, Beat Pfister, and Luc Van Gool. 2016. Deep Convolutional Neural Networks and Data Augmentation for Acoustic Event Recognition. (2016). Retrieved Juli 06, 2019 from https://data.vision.ee.ethz.ch/cvl/ae_dataset/

[19] Mastering the Mix. 2016. Sample Rates and Bit Depth... In a nutshell. (2016). Retrieved Juli 06, 2019 from https://www.masteringthemix.com/blogs/learn/113159685-sample-rates-and-bit-depth-in-a-nutshell

[20] welove.ai. 2019. Die Geschichte der künstlichen Intelligenz. (2019). Retrieved Juli 06, 2019 from https://www.welove.ai/de/blog/post/geschichte-kuenstlicher-intelligenz.html

[21] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How Transferable Are Features in Deep Neural Networks?. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 3320–3328. http://dl.acm.org/citation.cfm?id=2969033.2969197

[22] Lingyan Zhou. 2018. CNN - Audio Cats and Dogs. (2018). Retrieved Juli 07, 2019 from https://www.kaggle.com/zhoulingyan0228/moew-woof-cnn/data

# ANHANG

## A GRUNDIMPLEMENTIERUNG

**Auflistung 8: Source-Code der Grundimplementierung**

```
from google.colab import drive
drive.mount('/content/gdrive')

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import scipy
import scipy.io.wavfile as wavfile
import sklearn
import sklearn.metrics
import seaborn as sns
import randomp
import math
import sklearn.utils
import sklearn.metrics
import matplotlib.pyplot as plt
import glob
import os
import scipy
import scipy.signal
import tensorflow as tf
from imblearn.over_sampling import RandomOverSampler
import IPython

# Setzt die global wichtigen Variablen
WAVE_FOLDER = '/content/gdrive/My Drive/audio-cats-and-dogs/cats_dogs'
FRAMERATE = 16000
MAX_WAV_SAMPLES = 20*FRAMERATE
DOWNSAMPLING_SCALE = 1

# Laedt die CSV-Datei aus GDrive
df = pd.read_csv("/content/gdrive/My Drive/audio-cats-and-dogs/train_test_split.csv")

# Teilt die Spalten zu verschiedenen Klassen und Testklassen
test_cat = df[['test_cat']].dropna().rename(index=str, columns={"test_cat": "file"}).assign(label=0)

test_dog = df[['test_dog']].dropna().rename(index=str, columns={"test_dog": "file"}).assign(label=1)

train_cat = df[['train_cat']].dropna().rename(index=str, columns={"train_cat": "file"}).assign(label=0)
train_dog = df[['train_dog']].dropna().rename(index=str, columns={"train_dog": "file"}).assign(label=1)

# Setzt Test und Trainingsdaten zusammen in eine Tabelle
test_df = pd.concat([test_cat, test_dog]).reset_index(drop=True)
train_df = pd.concat([train_cat, train_dog]).reset_index(drop=True)

print('Test-wavs (dog,cat):')
print(len(test_dog))
print(len(test_cat))
print('Train-wavs (dog,cat)')
print(len(train_dog))
print(len(train_cat))

print('Test-wavs')
print(len(test_df))
print('Train-wavs')
print(len(train_df))

def plot_wavelet(file):
    # Samplerate der wavfile
    x = wavfile.read(file)[1]
    plt.figure(figsize=(10,5))
    plt.title(file)
    plt.plot(x)
    # Labelling der Achsen
    plt.ylabel('int16')
    plt.xlabel('Time [sample]')
    plt.show()

# Waehlt aus beiden Kategorien jeweils 5 Beispiele aus und plottet diese zur Veranschaulichung
for i in range(0,10,2):
    plot_wavelet(os.path.join(WAVE_FOLDER, test_cat.iloc[i]['file']))
    plot_wavelet(os.path.join(WAVE_FOLDER, test_dog.iloc[i]['file']))

# Gibt ein Histogramm der zwei Labels aus
train_df['label'].plot.hist(bins=2);

# RandomOversampler fuer ausgeglichene Anzahl beider Klassen
random_oversampler = RandomOverSampler()
idx = np.arange(0, len(train_df)).reshape(-1, 1)
idx_sampled, _ = random_oversampler.fit_sample(idx, train_df['label'])
train_files, train_labels = train_df.iloc[idx_sampled.flatten()]['file'].values, train_df.iloc[idx_sampled.flatten()]['label'].values
train_files, train_labels = sklearn.utils.shuffle(train_files, train_labels)
test_files, test_labels = test_df['file'].values, test_df['label'].values

pd.Series(train_labels).plot.hist(bins=2);

def fit_generator(train_files, train_labels, wavs_per_batch=20, augments=5):
```

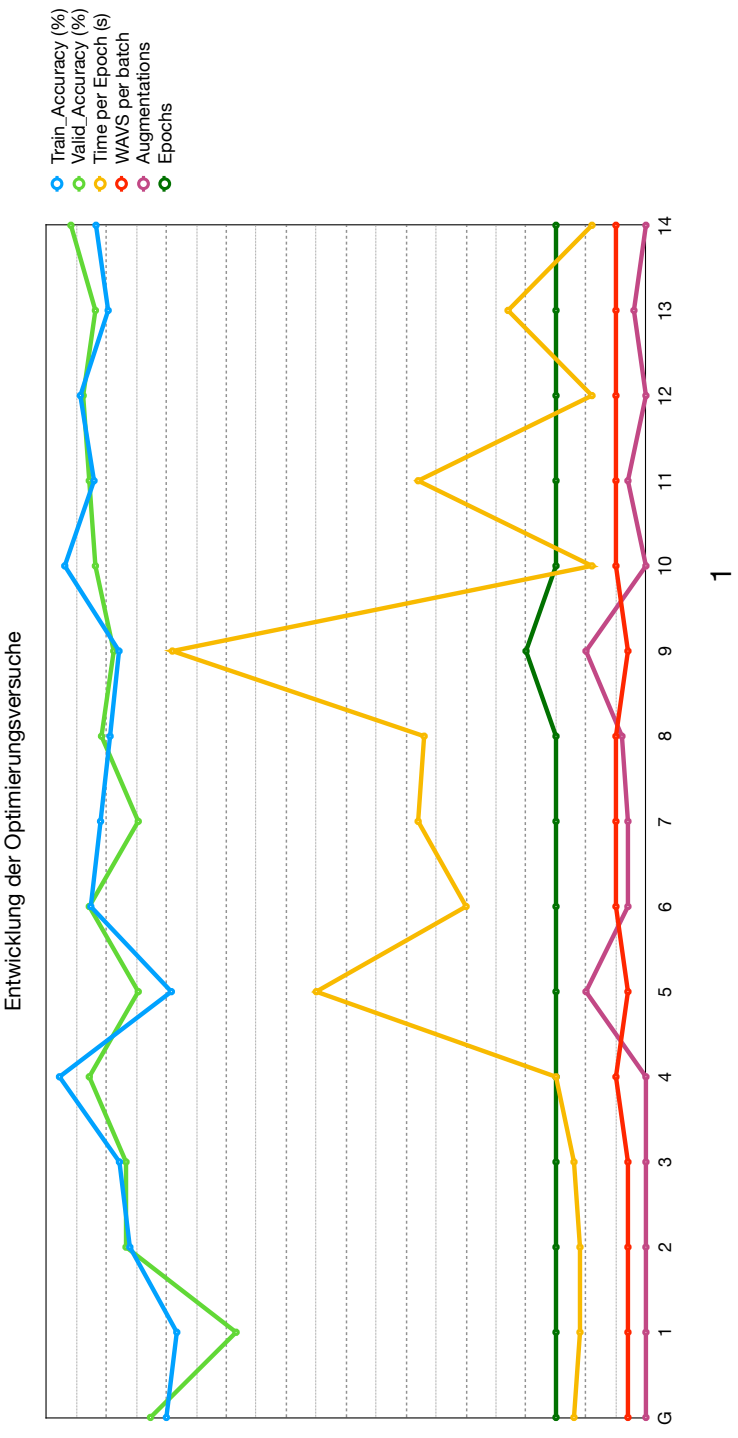
```
    while True:
        maxidx = len(train_files)
        for i in range(0, maxidx, wavs_per_batch):
            waves_batch = []
            labels_batch = []
            for j in range(i, min(maxidx, i+
                ↪ wavs_per_batch)):
                file, label = train_files[j],
                    ↪ train_labels[j]
                wave_raw = wavfile.read(os.path.join(
                    ↪ WAVE_FOLDER, file))[1]
                wave_raw = wave_raw/np.std(wave_raw)
                length = len(wave_raw)
                waves_batch.append(np.pad(wave_raw,
                    ↪ pad_width=((0, MAX_WAV_SAMPLES
                    ↪ - length)), mode='wrap'))
                labels_batch.append(label)
                #Data-Augmentation
                for _ in range(augments):
                    wave_rotated = np.roll(wave_raw,
                        ↪ random.randint(0, length))
                    while random.choice([True, False]):
                        wave_rotated += np.roll(wave_raw
                            ↪ , random.randint(0,
                            ↪ length))
                    wave = np.pad(wave_rotated,
                        ↪ pad_width=((0,
                        ↪ MAX_WAV_SAMPLES - length)),
                        ↪  mode='wrap')
                    #wave = scipy.signal.decimate(wave,
                        ↪  DOWNSAMPLING_SCALE)
                    waves_batch.append(wave)
                    labels_batch.append(label)
            yield np.array(waves_batch), np.array(
                ↪ labels_batch)

def validate_generator(test_files, test_labels,
    ↪ wavs_per_batch=20):
    while True:
        maxidx = len(test_files)
        for i in range(0, maxidx, wavs_per_batch):
            waves_batch = []
            labels_batch = []
            for j in range(i, min(maxidx, i+
                ↪ wavs_per_batch)):
                file, label = test_files[j],
                    ↪ test_labels[j]
                wave_raw = wavfile.read(os.path.join(
                    ↪ WAVE_FOLDER, file))[1]
                wave_raw = wave_raw/np.std(wave_raw)
                length = len(wave_raw)
                left = 0
                right = MAX_WAV_SAMPLES - left - length
                wave = np.pad(wave_raw, pad_width=((
                    ↪ left, right)), mode='wrap')
                #wave = scipy.signal.decimate(wave,
                    ↪ DOWNSAMPLING_SCALE)
                waves_batch.append(wave)
                labels_batch.append(label)
            yield np.array(waves_batch), np.array(
                ↪ labels_batch)

def steps_per_epoch(wavs_per_epoch, wavs_per_batch):
    return int(math.ceil(wavs_per_epoch/
        ↪ wavs_per_batch))

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Reshape((MAX_WAV_SAMPLES//
    ↪ DOWNSAMPLING_SCALE,1), input_shape=(
    ↪ MAX_WAV_SAMPLES//DOWNSAMPLING_SCALE,)))
for i in range(14):
    model.add(tf.keras.layers.Conv1D(32, kernel_size
        ↪ =5,
                                     padding='same',
                                     activation='relu',
                                     kernel_initializer=
                                         ↪ tf.keras.
                                         ↪ initializers.
                                         ↪ Orthogonal(),
                                         ↪
                                    ))
    model.add(tf.keras.layers.MaxPooling1D(pool_size
        ↪ =3, strides=2))
    model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(2, activation='
    ↪ softmax'))

model.compile(optimizer=tf.keras.optimizers.Adam
    ↪ (0.01),
              loss=tf.keras.losses.
                  ↪ sparse_categorical_crossentropy,
              metrics=['accuracy']
             )

model.summary()

WAVS_PER_BATCH = 3
AUGMENTS = 0
EPOCHS=15
history = model.fit_generator(fit_generator(
    ↪ train_files, train_labels, WAVS_PER_BATCH,
    ↪ AUGMENTS),
    steps_per_epoch=steps_per_epoch(len(train_files),
        ↪  WAVS_PER_BATCH),
    epochs = EPOCHS,
    validation_data=validate_generator(test_files,
        ↪ test_labels, WAVS_PER_BATCH),
    validation_steps=steps_per_epoch(len(test_files),
        ↪  WAVS_PER_BATCH),
    verbose=1)
```

```
predicted_probs = model.predict_generator(
    validate_generator(test_files, test_labels,
        ↪ WAVS_PER_BATCH),
    steps=steps_per_epoch(len(test_files),
        ↪ WAVS_PER_BATCH))
predicted_classes = np.argmax(predicted_probs, axis
    ↪ =1)
acc = round(sklearn.metrics.accuracy_score(
    ↪ predicted_classes, test_labels),4) *100
print('Model-accuracy:␣',acc,'%')
xticks = np.array(['True_predicted','False_predicted'
    ↪ ])
yticks = np.array(['True', 'False'])
sns.heatmap(sklearn.metrics.confusion_matrix(
    ↪ predicted_classes, test_labels), annot=True,
    ↪ xticklabels=xticks, yticklabels=yticks);

#Plot loss and accuracy for the training and
    ↪ validation set.
def plot_history(history):
    loss_list = [s for s in history.history.keys() if
        ↪  'loss' in s and 'val' not in s]
    val_loss_list = [s for s in history.history.keys()
        ↪  if 'loss' in s and 'val' in s]
    acc_list = [s for s in history.history.keys() if
        ↪ 'acc' in s and 'val' not in s]
    val_acc_list = [s for s in history.history.keys()
        ↪  if 'acc' in s and 'val' in s]
    if len(loss_list) == 0:
        print('Loss␣is␣missing␣in␣history')
        return
    plt.figure(figsize=(22,10))
    ## As loss always exists
    epochs = range(1,len(history.history[loss_list
        ↪ [0]]) + 1)
    ## Accuracy
    plt.figure(221, figsize=(20,10))
    ## Accuracy
    # plt.figure(2,figsize=(14,5))
    plt.subplot(221, title='Accuracy')
    for l in acc_list:
        plt.plot(epochs, history.history[l], 'b',
            ↪ label='Training␣accuracy␣(' + str(
            ↪ format(history.history[l][-1],'.5f'))+
            ↪ ')')
    for l in val_acc_list:
        plt.plot(epochs, history.history[l], 'g',
            ↪ label='Validation␣accuracy␣(' + str(
            ↪ format(history.history[l][-1],'.5f'))+
            ↪ ')')
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    ## Loss
    plt.subplot(222, title='Loss')
    for l in loss_list:
        plt.plot(epochs, history.history[l], 'b',
            ↪ label='Training␣loss␣(' + str(str(
            ↪ format(history.history[l][-1],'.5f'))+
            ↪ ')'))
    for l in val_loss_list:
        plt.plot(epochs, history.history[l], 'g',
            ↪ label='Validation␣loss␣(' + str(str(
            ↪ format(history.history[l][-1],'.5f'))+
            ↪ ')'))
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

# plot history
plot_history(history)
```

## B PARAMETER-PROTOKOLL

Übersicht der Hyper-Parameter der einzelnen Optimierungsversuche

| Train_Accuracy (%) | Valid_Accuracy (%) | Time per Epoch (s) | Layers | WAVS per batch | Augmentations | Epochs | Dropout | Nodes per Layer | filter size | pool size | strides | optimizer | total params | trainable params | non-trainable params |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 80 | 82,65 | 12 | 14 | 3 | 0 | 15 | 0 | 32 | 5 | 3 | 2 | 0,01 | 72.418 | 70.370 | 2.048 |
| 78,26 | 68,37 | 11 | 14 | 3 | 0 | 15 | 0 | 32 | 3 | 3 | 2 | 0,01 | 45.730 | 43.682 | 2.048 |
| 86,09 | 86,73 | 11 | 14 | 3 | 0 | 15 | 0 | 32 | 3 | 3 | 2 | 0,001 | 45.730 | 43.682 | 2.048 |
| 87,83 | 86,73 | 12 | 16 | 3 | 0 | 15 | 0 | 32 | 3 | 3 | 2 | 0,001 | 45.730 | 43.682 | 2.048 |
| 97,83 | 92,86 | 15 | 16 | 5 | 0 | 15 | 0 | 64/32 | 5/3 | 3 | 2 | 0,001 | 52.770 | 51.490 | 1.280 |
| 79,17 | 84,69 | 55 | 16 | 3 | 10 | 15 | 0 | 32 | 5/3 | 3 | 2 | 0,005 | 49.378 | 48.162 | 1.216 |
| 92,61 | 92,86 | 30 | 16 | 5 | 3 | 15 | 0 | 64/32 | 5/3 | 3 | 2 | 0,005 | 52.770 | 51.490 | 1.280 |
| 90,98 | 84,69 | 38 | 16 | 5 | 3 | 15 | 0 | 64/64/32 | 5/5/3 | 3 | 2 | 0,001 | 70.338 | 68.994 | 1.344 |
| 89,39 | 90,82 | 37 | 16 | 5 | 4 | 15 | 0 | 64/32 | 5/3 | 3 | 2 | 0,001 | 52.770 | 51.490 | 1.280 |
| 87,90 | 88,78 | 79 | 16 | 3 | 10 | 20 | 0 | 64/32 | 5/3 | 3 | 2 | 0,001 | 52.642 | 51.362 | 1.280 |
| 96,95 | 91,84 | 9 | 16 | 5 | 0 | 15 | 0,25 | 64/32 | 5/3 | 3 | 2 | 0,001 | 52.642 | 51.362 | 1.280 |
| 92,07 | 92,86 | 38 | 16 | 5 | 3 | 15 | 0,25 | 64/64/32 | 5/5/3 | 3 | 2 | 0,001 | 70.338 | 68.994 | 1.334 |
| *94,35* | *93,88* | *9* | *16* | *5* | *0* | *15* | *0,25* | *64/32* | *5/3* | *3* | *2* | *0,005* | *52.642* | *51.362* | *1.280* |
| 89,71 | 91,84 | 23 | 16 | 5 | 2 | 15 | 0,25 | 64/32 | 5/3 | 3 | 2 | 0,0005 | 52.642 | 51.362 | 1.280 |
| **91,74** | **95,92** | **9** | **16** | **5** | **0** | **15** | **0,25** | **64/32** | **5/3** | **3** | **2** | **0,005** | **52.770** | **51.490** | **1.280** |



# Convolutional Neural Network für die „Urban Sound Classification“

Moritz Egelkraut
Hof University
moritz.egelkraut@hof-university.de

## ZUSAMMENFASSUNG

Die automatisierte Erkennung von urbanen Umgebungsgeräuschen durch Informationssysteme gewinnt zunehmend an Interesse in Industrie und Forschung. In dieser Arbeit wird der aktuelle wissenschaftliche Stand dargestellt und die grundlegende Theorie über maschinelles Lernen präsentiert. Mithilfe des Datensatzes „Urban Sound“ und dem Framework TensorFlow wird ein „Convolutional Neural Network (CNN)“ vorgestellt, welches typische urbane Umgebungsgeräusche selbstständig erkennen kann.

• **Computing methodologies** → *Machine learning approaches*; Artificial intelligence; Neural networks.

## KEYWORDS

datasets, neural networks, cnn, urban sound, fourier, tensorflow, keras, librosa

## 1 EINLEITUNG

„Hey Siri...“, „Okay Google!“ und „Alexa!“ sind die drei bekanntesten Sätze, welche seit den letzten Jahren immer mehr zu hören sind. Eine Notiz anlegen oder einen Wecker stellen, all das ist problemlos mittels Spracheingabe in ein mobiles Endgerät möglich. Immer mehr Menschen nutzen die Spracherkennung ihres Gerätes, um damit ihren Alltag zu vereinfachen. Die Sprachassistenten werden auch immer mehr als Brücke zu anderen Innovationen verwendet, wie zum Beispiel dem Smart Home. Auch deshalb steigen seit Jahren die Investitionen im Bereich der Audio-Erkennung und Verarbeitung. [1]
Die maschinelle Erkennung und Klassifizierung von Audio findet vor allem im Umfeld der Künstlichen Intelligenz statt. Dabei gibt es verschiedene Vorgehensweisen, von denen sich einige als vorteilhaft erwiesen haben. Dennoch ist festzustellen, dass die Forschung sich hierbei erst in einem vergleichsweise frühen Stadium befindet und vor allem durch die Kooperation mit großen Konzernen wie Google vorangetrieben wird. Dies ist auch durch das breite Einsatzspektrum der Audio-Erkennung bedingt, da die verschiedenen Vorgehensweisen sehr fallspezifisch sind und es keine allgemein gültigen „Faustregeln“ für die automatische Erkennung und Verarbeitung von Audio-Input gibt. [10]
Ein Teilbereich davon ist die Erkennung von Umgebungsgeräuschen. Besonders im urbanen Umfeld, welches geprägt ist von stark unterschiedlichen Geräuscharten, wird der Erkennung von Umgebungsgeräuschen eine immer größere Bedeutung zugemessen. Ein Problem dabei ist allerdings, dass es sehr wenige Datensätze gibt, welche ausreichend viele klassifizierte, typische Geräusche einer Stadt beinhalten. Dieses Problem hat auch die New York University erkannt und deshalb den Datensatz „Urban Sound Classification“ erstellt und der Forschergemeinde zur Verfügung gestellt. [11]
In dieser Arbeit geht es um den eben genannten Datensatz „Urban Sound Classification“ und dessen Verwendung in einem neuronalen Netz. Dieses wird mithilfe von verschiedenen Frameworks in der Programmiersprache Python realisiert. In dieser Arbeit sollen die Vor- und Nachteile der Verwendung eines CNN zur Umgebungsgeräuscherkennung an einem praktischen Beispiel dargestellt werden. Im ersten Kapitel nach der Einleitung wird zunächst der aktuelle wissenschaftliche Stand im Bereich der Geräuscherkennung mithilfe von Künstlicher Intelligenz dargestellt. Im darauf folgenden Kapitel werden die theoretischen Grundlagen des maschinellen Lernens, der digitalen Audioverarbeitung und weiteren in dieser Arbeit relevanten Themen gelegt. Anschließend befassen sich je ein Kapitel mit der Implementierung und den daraus resultierenden Ergebnissen und deren Bewertung. In einem letzten Kapitel findet sich ein Ausblick auf mögliche Verbesserungen und alternative Vorgehensweisen.

## 2 VERWANDTE ARBEITEN

Wissenschaftliche Arbeiten über den Themenbereich „Urban Sound Classification“ sind zum Zeitpunkt des Schreibens dieser Arbeit sehr schwer zu finden. Deshalb ist es nötig, den übergeordneten Bereich der allgemeinen „Sound Classification“ zu betrachten. Da die Problemstellungen in anderen Bereichen der Sound-Klassifizierung denen der urbanen Umgebungsgeräuscherkennung ähneln, können die Ergebnisse teilweise darauf projiziert werden. Die meisten Innovationen haben, neben dem vom Entwickler meist forcierten technischen Fortschritt, auch einen betriebswirtschaftlichen Hintergrund. So sollen Kosten gesenkt oder die Qualität der Produkte verbessert werden. Deshalb ist es auch nicht verwunderlich, dass die Einsatzmöglichkeiten der „Sound Classification“ nicht auf die bisher benannten Einsatzmöglichkeiten beschränkt sind, sondern sich durchaus auch komplett andere Möglichkeiten ergeben.
So stellen Yang and Rai [16] fest, dass besonders im Bereich des Maschinen-Monitorings auch die akustischen Signale ein sehr wichtiger, nicht zu unterschätzender, Faktor sind. In ihrer Arbeit zeigen die Wissenschaftler, wie Unregelmäßigkeiten im Betrieb einer Maschine mithilfe eines Mikrofons erkannt werden können. Durch die Verwendung eines CNN werden Ratter-Geräusche bemerkt und es kann eine Prüfung der Maschine vor dem tatsächlichen Schadenseintritt stattfinden. Die Entwicklung eines solchen Projekts ist allerdings nicht trivial. Eine der größten Schwierigkeiten bei

*FWPM: Angewandtes maschinelles Lernen, Sommersemester, 2019, Hof*

der Entschlüsselung aufschlussreicher Informationen aus Tonbits besteht darin, dass Zeitstruktur-Tonsignale durch zeitliche Ausdehnung und Komprimierung maskiert werden. Das bedeutet, dass die Klassifizierung der Daten durch ein neuronales Netzwerk sehr schwierig ist und deshalb sehr viele Daten benötigt werden. [16]
Souli and Lachiri [12] beschäftigten sich im Jahr 2011 mit der „Urban Sound Classification“. Als Modell wurde eine „support vector machine (SVM)“ verwendet, eine sogenannte „supervised learning method“. In Kapitel 3.3 werden Modelle im Umfeld des maschinellen Lernens näher erklärt. Ebenso wie der Datensatz, welcher in dieser Arbeit verwendet wird, haben Souli and Lachiri [12] einen Datensatz mit zehn zu klassifizierenden Geräuschen verwendet und dabei eine Genauigkeit von bis zu 91,82 % erreicht. Die Audiodateien wurden bei diesem Ansatz in Spektrogramme umgewandelt und so durch das Modell als Bilder verarbeitet [12].

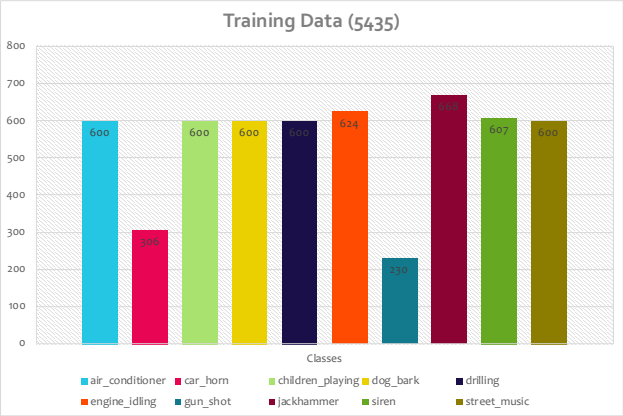



**Abbildung 1: Verteilung der Klassen der Trainingsdateien im Datensatz „Urban Sound“**

# 3 THEORIE

## 3.1 Künstliche Intelligenz & Maschinelles Lernen

Bis zu den Anfängen der Informatik in den 1950er und 1960er reichen die Ursprünge der Künstlichen Intelligenz zurück. Aufgrund der massiven Entwicklung der Rechenleistung hat sich die Künstliche Intelligenz vor allem im letzten Jahrzehnt stark weiterentwickelt. Cowling et al. [6] definiert Künstliche Intelligenz als Unterfangen, Computer Funktionen ausführen zulassen, welche allgemein geglaubt Intelligenz verlangen. Der Begriff ist auch deshalb schwer allgemeingültig zu definieren, da um den Begriff „Intelligenz“ an sich schon eine Debatte um die richtige Definition stattfindet. Dennoch kann festgestellt werden, dass die Künstliche Intelligenz sehr komplexe Probleme lösen kann. Dabei versteht sich der Begriff als allgemeiner Bereich in der Informatik, welcher viele Vorgehensweisen und Disziplinen zur Lösung komplexer Probleme mithilfe eines Computers unter sich vereint. [6]
„Maschinelles Lernen“ ist ein Begriff, welcher im Rahmen der Künstlichen Intelligenz verwendet wird, um eine automatisierte Verbesserung anzuzeigen, die auf Erfahrungswerten oder empirischen Daten zur Erfüllung einer bestimmten Aufgabe wie der Optimierung einer Zielfunktion basiert. [7]

## 3.2 Datensatz „Urban Sound Classification“

„Urban Sound“, oder auch „Urban Sound Classification“ genannt, ist ein Datensatz, welcher 8732 gekennzeichnete Sounddateien im Waveform Audio File Format (WAVE) beinhaltet und von der New York University (NYU) erstellt wurde. Diese sind aufgeteilt in zehn verschiedene Klassen und jeweils kleiner gleich vier Sekunden lang. Die Sounddateien sind nummeriert und in dem Datensatz zugehörigen CSV-Dateien den zehn Klassen zugeordnet. Diese Klassen sind „Air Conditioner“, „Car Horn“, „Children Playing“, „Dog bark“, „Drilling“, „Engine Idling“, „Gun Shot“, „Jackhammer“, „Siren“ und „Street Music“. Aufgeteilt ist der Datensatz in Trainingsdateien (5435 Dateien) und Testdateien (3297), wobei die Verteilung der Trainingsdateien in Abbildung 1 zu sehen ist. [11]

## 3.3 Modelle

Um den verschiedensten Anforderungen der unzähligen Anwendungsmöglichkeiten gerecht zu werden, wurden im Laufe der Jahre viele verschiedene Modelle im Bereich des maschinellen Lernens entwickelt. Diese bedienen sich sehr komplexen mathematischen Funktionen und unterscheiden sich in der Vorgehensweise zur Erfüllung einer bestimmten Aufgabe. Die Wahl des richtigen Modells hängt stark von den zur Verfügung stehenden Daten ab [8]. Die Modelle werden deshalb grundsätzlich in drei Kategorien eingeteilt, welche in Abbildung 2 zu sehen sind. Um zu entscheiden, welche Kategorie verwendet werden muss, erfolgt zunächst ein Blick auf die Datenstruktur. In Kapitel 3.2 wird diese genauer betrachtet. Da es sich bei den Daten um klassifizierte und gelabelte Daten handelt, ist in diesem Fall ein Modell aus dem Bereich der Klassifizierung zu wählen. Unterschieden werden Modelle weiterhin durch die Art des Lernens [8]. Auch hier gibt es drei verschiedene Kategorien. Das Lernen mit Daten und bekannten Ergebnis, dem sogenannten überwachten Lernen (supervised learning). Unüberwachtes Lernen (unsupervised learning) durch das Lernen von Mustern in Daten und das Bekräftigungslernen (reinforcement learning), bei welchem über die Richtigkeit nach der Entscheidung entschieden wird. Da es sich bei den Daten um Beispiele handelt, welche ein klares Ergebnis möglich machen (siehe Kapitel 3.2), muss bei der Art des Lernens die Kategorie überwachtes Lernen gewählt werden.
Wie bereits angemerkt, gibt es oft keine eindeutig richtige Wahl eines bestimmten Modells für viele Probleme, sondern nur Modelle, welche sich gegenüber anderen als besser für bestimmte Anwendungsfälle erwiesen haben. Der am meisten verwendete Indikator, ob ein Modell gut ist, ist die Genauigkeit der gelieferten Ergebnisse (Output). Doch sind andere Faktoren, wie zum Beispiel die benötigte Rechenleistung oder Speicherkapazitäten, bei der Wahl des Modells nicht komplett außer Acht zu lassen. [8]
Zusammenfassend für dieses Kapitel lässt sich feststellen, dass für die in dieser Arbeit verwendeten Daten ein Modell aus dem Bereich der Klassifizierung mit überwachtem Lernen gewählt werden sollte.

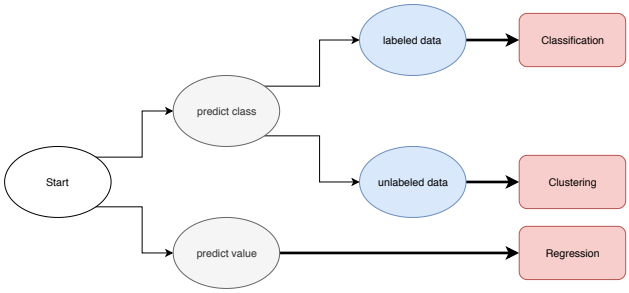



**Abbildung 2: Kategorien der Modelle des maschinellen Lernen**

## 3.4 CNN

Neuronale Netze im Bereich der Informatik sind in ihrem Design vor allem durch den Aufbau des menschlichen Gehirns inspiriert. Durch die Verwendung von sehr vielen Neuronen, welche logische Zustände annehmen können, kann ein Netz bestimmte Muster antrainieren und diese erkennen. Ein Neuron besteht aus vier Komponenten, namentlich den gewichteten Kanten, einem Schwellwert (Bias), einem Addierer der gewichteten Eingangswerte und einer Funktion, welche die gewichtete Summe in einen Ausgangswert transformiert. Durch Training werden die Gewichte angepasst, bis ein akzeptables Ergebnis durch das gesamte Netz erreicht wird. Ein Convolutional Neural Network zeichnet sich durch das Wort „convolutional“ aus, was im Deutschen Faltung bedeutet. [13]
Bei der Erkennung, von zum Beispiel Graphen, und damit Bildern, werden alle Pixel als Zahlenwerte in einer Matrix dargestellt. Wird ein Netz mit Bildern trainiert, hängt die Genauigkeit bei einem klassischen neuronalen Netz stark davon ab, ob die gleichen Klassen immer an der selben Stelle im Bild sind und so ein Muster erkannt werden kann. Ein weit verbreitetes Beispiel hierfür ist die Erkennung von handschriftlichen Zahlen durch ein neuronales Netz. Schreibt man die Zahlen immer an die selbe Stelle in den Bildern kann ein Netz sehr schnell gute Erfolge erzielen. Sobald allerdings eine Zahl nicht an der üblichen Stelle platziert ist, sinken die Erfolgsquoten rapide. Die Lösung für dieses Problem sind Faltungen, also „convolutions“. So wird zwischen zwei Matrizen eine Multiplikation durchgeführt, welche allerdings keine klassische Matrix-Multiplikation darstellt. Wichtig hierbei ist, dass es eine kleinere Matrix geben muss, welche allgemein „filter“ oder „kernel“ genannt wird. Die Werte für die kleinere Matrix stammen aus einem statistischen Gradientenabstieg. Ein weiterer wichtiger Begriff ist das „Pooling“. Neben Pooling-Methoden, welche Summen und Durchschnitte verwenden, ist das „Max-Pooling“ eine der am meisten verwendeten Methoden. Es wird die große Matrix in so große Felder aufgeteilt, wie groß die kleine Matrix nach der Faltung ist. Ist die kleine Matrix nach der Faltung 2x2 groß und die große Matrix 4x4 groß, so wird die große Matrix in insgesamt vier Felder unterteilt. Beim „Max-Pooling“ wird der jeweils größte Wert der Unterteilungen der großen Matrix, also vier, in die kleine Matrix übernommen. Durch dieses Verfahren wird das oben genannte Problem der einfachen neuronalen Netze eliminiert. In Kapitel 6 werden noch andere Alternativen und Lösungsansätze für differenzierte Probleme erläutert. [4]

## 3.5 Fourier-Transformation

Die „Fast Fourier transform (FFT)“ ist ein Algorithmus, welcher die diskrete Fourier Transformation (DFT) effizient berechnen kann. Dadurch ist es möglich, ein digitales Signal in seine Frequenzanteile zu zerlegen und diese dann zu analysieren. Dieser Algorithmus ist deshalb von herausragender Bedeutung, da er in allen wissenschaftlichen Bereichen Anwendung findet und auch ein wichtiger Bestandteil bei der Implementierung von Sounderkennungen ist, also auch bei der „Urban Sound Classification“. [9]

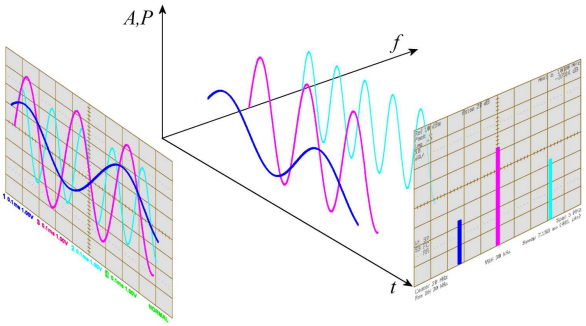



**Abbildung 3: Betrachtungsrichtungen für den Zeitbereich und den Frequenzbereich [14]**

Konkret lassen sich mit der FFT hauptsächlich vorkommende Frequenzen und deren Amplituden in einem abgetasteten Signal bestimmen. Abbildung 3 veranschaulicht den Nutzen einer DFT. Die einzelnen Frequenzen und deren Amplituden des Frequenzspektrum sind gut zu erkennen und lassen sich dadurch auch digital analysieren und gegen andere Spektren abgrenzen. [14]

## 3.6 TensorFlow, Keras und LibROSA

In diesem Kapitel sollen die drei wichtigen Frameworks TensorFlow, Keras und LibROSA kurz vorgestellt werden. Diese spielen nicht nur bei der Implementierung der „Urban Sound Classification“ eine Rolle, sondern sind im allgemeinen sehr hilfreich bei der Implementierung von Lösungen im Bereich des maschinellen Lernens.
TensorFlow ist ein Framework, um Modelle des maschinellen Lernens zu bauen und zu trainieren. Das Framework wurde in Python und C++ implementiert, wobei die Verwendung durch Python erfolgt. TensorFlow ist ein Projekt von Google, welches das Framework selbst in den eigenen Produkten, wie dem Assistant oder Maps, einsetzt. TensorFlow wurde deshalb auch so entwickelt, dass die Modelle einfach in der Produktion eingesetzt werden können beziehungsweise ihrer eigentlichen Verwendung schnell zugeführt werden können. Des weiteren bietet TensorFlow noch weiterführende Tools, um wissenschaftliche Teams bei ihren Forschungen zu unterstützen. Ein Vorteil von TensorFlow sind die verschiedenen Abstraktionslevel. Mithilfe der bereits in TensorFlow implementierten Keras API ist es sehr einfach und ohne großes Hintergrundwissen möglich, eigene Modelle zu bauen. Für Fortgeschrittene und Experten stehen zahlreiche tiefer greifende Funktionen zur

Verfügung. In Kapitel 4 wird genauer auf die Implementierung von Modellen mithilfe der Keras API eingegangen. [3]
LibROSA ist ein Python-Framework, welches zur Musik- und Audioanalyse eingesetzt wird. LibROSA kann sehr viele Eigenschaften aus Audiodateien gewinnen. In dieser Arbeit besonders interessant ist die Darstellung von Spektrogrammen und einige weitere Funktionen, die in Kapitel 4 gezeigt werden. [2]

## 4 IMPLEMENTIERUNG

In weiten Teilen bezieht sich die Implementierung auf das Notebook von Yang [15] auf kaggle.com, einer Plattform für Data Science Projekte. Der Verständlichkeit wegen, wird in diesem Kapitel nur auf die wichtigen Teile der Implementation eingegangen, wobei der komplette Programmiercode im Anhang zu finden ist.

**Auflistung 1: Laden des Datensatzes**

```
SKIP_AUDIO_RELOAD = True

from google.colab import drive
drive.mount('/content/gdrive')

#location of the sound files
INPUT_PATH='/content/gdrive/My Drive/Studium/
    ↪ Angewandtes maschinelles Lernen/urban-sound-
    ↪ classification'

TRAIN_INPUT=INPUT_PATH+'/train'
TRAIN_AUDIO_DIR=TRAIN_INPUT+'/Train'

TEST_INPUT=INPUT_PATH+'/test'
TEST_AUDIO_DIR=TEST_INPUT+'/Test'
```

Im ersten Schritt nach dem Import der benötigten Pakete werden die Daten geladen. Da es sich hierbei um sehr große Datenmengen mit mehreren Gigabyte handelt und die Implementierung auf einem Jupyternotebook online stattfindet (colab.research.google.com), wird eine Verbindung zu einem Cloud-Dienst hergestellt, in diesem Fall Google Drive. Interessant ist auch die erste Zeile Code in der Auflistung 1. Was diese genau zu bedeuten hat, wird später deutlich. Anschließend werden die Pfade zu den bereits aufgeteilten Trainings- und Testdaten gesetzt.

**Auflistung 2: LibROSA-Beispiel**

```
# sample-1: spectrogram
librosa.display.specshow(ps, y_axis='mel', x_axis='
    ↪ time')
```

In Auflistung 2 wird von einer Audiodatei ein Spektrogramm ausgegeben, um einen Eindruck zu schaffen, wie einfach mit libROSA Audiodateien zu analysieren sind. Das Ergebnis ist in Abbildung 4 zu sehen. Die Sounddatei stammt aus der Kategorie „police_siren“. In Abbildung 5 ist ein Spektrogramm aus der Kategorie „street_music“ zu sehen. Die Unterschiede zwischen den beiden Klassen sind in den Spektrogrammen klar erkennbar.
Ein sehr wichtiger Schritt bei der Vorbereitung der Daten ist, neben der Unterteilung in Trainings- und Testdaten, die Zuschneidung

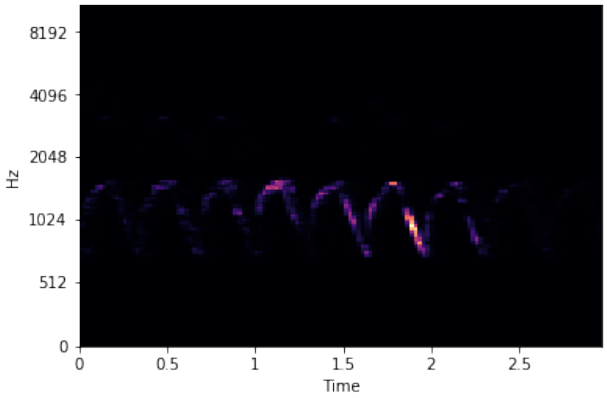



**Abbildung 4: Beispiel Spektrogramm police_siren**

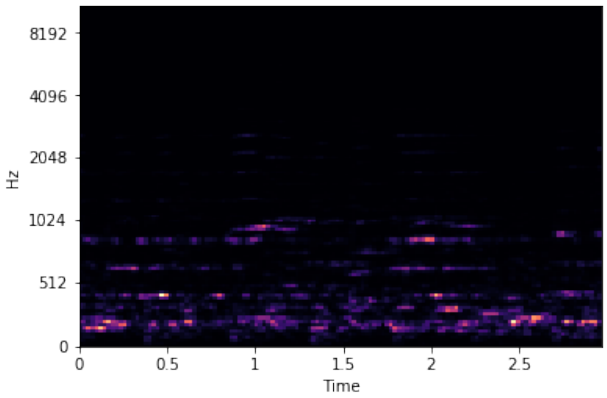



**Abbildung 5: Beispiel Spektrogramm street_music**

der Audiodateien auf die selbe Länge. Im Datensatz sind die Audiodateien nicht genau gleich lang. Dies ist aber nötig, damit alle Spektrogramme und sonstigen Bilder die selbe Größe haben. In diesem Fall werden 3 Sekunden Audiolänge gewählt, was einer Größe von 128x128 entspricht. Da der Algorithmus zur Kürzung keine Erkenntnisse für das maschinelle Lernen liefert, wird dieser hier nicht erläutert.

**Auflistung 3: Vereinheitlichung der Audiolängen der Trainingsdaten**

```
1  %%time
2  %%memit
3  # Dataset
4  train_audio_data = []
5  train_object_file='/content/gdrive/My Drive/Studium/
       ↪ Angewandtes maschinelles Lernen/
       ↪ prepared_data_models/saved_train_audio_data.p
       ↪ '
6
7  #load training data
8  if SKIP_AUDIO_RELOAD is True:
```

```
9       print ("skip re-loading TRAINING data from audio
            ↪ files")
10  else:
11      print ("loading train audio data, may take more
            ↪ than 15 minutes. please wait!")
12      for row in tqdm(valid_train_data.itertuples()):
13          ps = load_audio_file(file_path=row.path,
                ↪ duration=2.97)
14          if ps.shape != (128, 128): continue
15          train_audio_data.append( (ps, row.Class) )
16      print("Number of train samples: ", len(
            ↪ train_audio_data))
17
18  # load saved audio object
19  if SKIP_AUDIO_RELOAD is True:
20      train_audio_data = cPickle.load(open(
            ↪ train_object_file, 'rb'))
21      print ("loaded train data [%s] records from
            ↪ object file" % len(train_audio_data))
22  else:
23      cPickle.dump(train_audio_data, open(
            ↪ train_object_file, 'wb'))
24      print ("saved loaded train data :",len(
            ↪ train_audio_data))
```

In den Zeilen 5 und 7 der Auflistung 3 wird deutlich, dass der Prozess der Verkürzung nicht jedes mal erfolgen muss, sobald das Programm ausgeführt wird, sondern die verkürzten Audiodateien komprimiert abgespeichert werden können und bei Bedarf einfach wieder geladen werden können. Die Verkürzung der Trainingsdateien dauerte im Durchlauf auf colab.research.google.com 1h 43min und die Testdateien 1h 14min. Der Code zu den Testdateien ist analog zu dem Code in Auflistung 3. Ist der Flag nicht auf „true“ gesetzt, beginnt der Prozess erneut. Auch beim Zwischenspeichern muss aufgrund der Dateigröße wieder auf eine Cloud gesetzt werden.

**Auflistung 4: Labels**

```
list_labels = sorted(list(set(valid_train_data.Class.
    ↪ values)))
```

In Abbildung 1 aus Kapitel 3.2 sind die Klassen des Datensatz bereits ersichtlich, allerdings müssen diese auch dem Programm eindeutig bekannt sein. Dies geschieht in TensorFlow mit dem Code aus Auflistung 4.

**Auflistung 5: Validation**

```
X_test, X_val, y_test, y_val = train_test_split(
    ↪ X_test, y_test, test_size=0.30, random_state
    ↪ =42)
```

Aus der Menge der Testdateien werden in Auflistung 5 30 % als validation-Dateien gekennzeichnet und somit vollkommen von dem CNN abgegrenzt. Dies erlaubt später eine genauere Auswertungen über die Genauigkeit des Netzes.

**Auflistung 6: Bau des Modells**

```
1   model = Sequential()
2   input_shape= X_train.shape[1:]
3
4   model.add(Conv2D(24, (5, 5), strides=(1, 1),
        ↪ input_shape=input_shape))
5   model.add(MaxPooling2D((4, 2), strides=(4, 2)))
6   model.add(Activation('relu'))
7
8   model.add(Conv2D(48, (5, 5), padding="valid"))
9   model.add(MaxPooling2D((4, 2), strides=(4, 2)))
10  model.add(Activation('relu'))
11
12  model.add(Conv2D(48, (5, 5), padding="valid"))
13  model.add(Activation('relu'))
14
15  model.add(Flatten())
16  model.add(Dropout(rate=0.5))
17
18  model.add(Dense(64))
19  model.add(Activation('relu'))
20  model.add(Dropout(rate=0.5))
21
22  model.add(Dense(len(list_labels)))
23  model.add(Activation('softmax'))
24  model.summary()
```

In Auflistung 6 wird nun mithilfe der Keras API das Modell des CNNs gebaut. In Zeile 4 wird der erste Layer hinzugefügt. Dabei steht „Conv2D“ für einen Convolutional-Layer. Die 24 steht für die Anzahl der Output-Filter und das Argument „(5, 5)“ für die Kernel-Größe. Das Argument „strides“ legt den Betrag fest, um den sich der Filter verschiebt. In diesem Fall jeweils um eine Einheit in Breite und Höhe. Des Weiteren muss bei dem ersten Layer eines sequentiellen Modells immer die Eingangswerte definiert werden. Diese bestehen aus der Höhe, Breite und der Channel-Dimension. In diesem Fall betragen Höhe und Breite 128 und die Channel-Dimension ist 1. In Zeile 5 wird nun das Pooling durchgeführt und in Zeile 6 wird als Aktivierungsfunktion die lineare ReLU-Funktion festgelegt. Diese legt fest, aufgrund welcher Funktion ein Neuron im Netz aktiviert wird. Ein Beispiel für eine nicht-lineare Aktivierungsfunktion ist die sogenannte sigmoid-Funktion. In Zeile 8 ist besonders das Argument „padding“ interessant, da in diesem Fall „valid“ gewählt wurde. Dies bedeutet, dass überstehende Werte bei der Faltung ersatzlos gestrichen werden und durch Padding kein weiteres Feld mit hinzugefügten Werten erzeugt wird. Mit Zeile 16 wird einer Überanpassung, „overfitting“ genannt, des Netzes entgegengewirkt. In Zeile 22 wird die letzte Schicht hinzugefügt. Diese ist exakt so groß gewählt, dass als Ergebnis nur eine der Klassen des Datensatzes herauskommen kann. Das bedeutet, das Netz entscheidet sich immer für eine Klasse des Datensatzes. Hierbei wird als Aktivierungsfunktion die „softmax“-Funktion eingesetzt. Diese gibt die Wahrscheinlichkeit an, dass eine der Klassen wahr ist.

**Auflistung 7: Training**

```
history = model.fit(x=X_train, y=y_train,
                    epochs=MAX_EPOCHS,
                    batch_size=MAX_BATCH_SIZE,
                    verbose=0,
                    validation_data= (X_val, y_val),
```

```
callbacks=callback)
```

In Auflistung 7 wird das Modell anschließend trainiert. „MAX-_EPOCHS“ legt die Anzahl der Wiederholungen des Lernvorgangs fest und wird in diesem Beispiel auf 20 gesetzt. „MAX_BATCH-_SIZE“ legt die Größe der Anteile des Datensatzes fest, welche durch das Netz propagiert werden. Somit wird das Netz nicht auf einmal mit allen Daten gemeinsam trainiert, sondern aufeinander folgend wird das Netz mit allen Daten trainiert.

## 5 ERGEBNISSE

In Abbildung 6 ist die Genauigkeit des Netzes zu sehen, wobei die Werte aus Kapitel 4 verwendet wurden. Die Genauigkeit des Netzes, getestet mit den Daten aus der Validation, beträgt 78 %. Die aus dieser Grafik nicht ersichtliche Genauigkeit mit den Testdaten beträgt 76 %. Nutzt man die Evaluierungs-Funktion der Keras API, wird bei den Trainingsdaten ein Ergebnis von 86 % erreicht. Die Bewertung der Ergebnisse ist schwierig. Grundsätzlich sind knapp 80 % positiv zu bewerten und für einige Anwendungsfälle zunächst ausreichend. Dennoch kann kritisiert werden, dass die Genauigkeit von CNNs in anderen Bereichen wesentlich höher ist. Dort erreichen CNNs weit über 90 % Genauigkeit, in einigen Beispielen ist die Genauigkeit nicht sehr weit von 100 % entfernt. Insoweit besteht durchaus noch die Möglichkeit, die Genauigkeit des Netzes durch andere Verfahren zu verbessern. Des Weiteren neigt das Netz zur Überanpassung, was durch den Dropout-Layer zwar gemindert wird, allerdings dennoch ein Problem darstellt. Eine weitere Vorgehensweise, das Beenden des Trainingsprozesses, bevor eine starke Überanpassung einsetzt, klappt allerdings verhältnismäßig gut. Weiterhin lässt sich eine Überanpassung theoretisch durch den Einsatz von mehr Daten vermindern, wobei dies in diesem Fall aufgrund zu weniger Daten nicht möglich ist. Ein weiterer Hinweis auf zu wenigen Daten ist die schlechte Loss-Funktion, da diese kontinuierlich sinken sollte, besonders bei zunehmender Epochenanzahl (vgl. Abbildung 7).

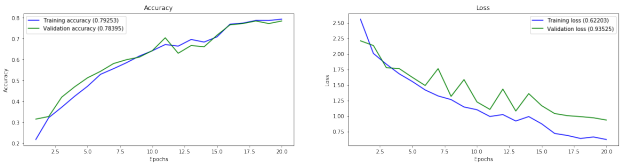



**Abbildung 6: Genauigkeit des Netzes (20 Epochen)**

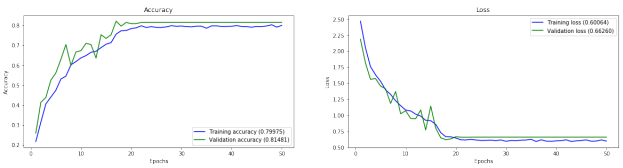



**Abbildung 7: Genauigkeit des Netzes (50 Epochen)**

## 6 AUSBLICK

Bedenkt man die Einsatzmöglichkeiten der Audioerkennung, dann wird schnell erkannt, dass viele Geräusche und gesprochenes Wort erst in einem Zusammenhang mit wiederum anderen Geräuschen und gesprochenem Wort einen großen praktischen Nutzen haben. Deshalb ist es neben der Erkennung von einzelnen Klassen oft nötig, die Kombination aus vielen Klassen zu erkennen. Bei einem CNN erfolgt die Erkennung genau einer Klasse, da das Netz per se keinen eigenen Speicher hat, also ein statisches Muster. Die Erkennung zeitlicher Muster ist somit bei einigen Anwendungen ebenfalls bedeutsam. Diese Fragestellung lässt sich mit einem „Reccurent Neural Network (RNN)“ lösen. Im Bereich der Urban Sound Classification kann also ein RNN ebenfalls gute Ergebnisse liefern, besonders bei der Erkennung der Geräusche während einer Audiosequenz. [5]

Eine weitere Möglichkeit zur Verbesserung ist es, den Datensatz zu vergrößern und noch mehr Daten zu sammeln. Bei klanglich stark unterschiedlichen Klassen beziehungsweise Geräuschen führt dies sicherlich zu einer verbesserten Genauigkeit. Werden allerdings mehr Klassen hinzugefügt, so kann sich das Problem der Genauigkeit verschärfen, sollten die Klassen sich ähnlicher werden. Dies ist besonders im Urbanen der Fall, da viele Geräusche sich sehr ähneln. Allein die Bildung der Klassen ist diskutabel. Ist es wichtig Sirenen zu unterscheiden? Und ist bei Straßenmusik nötig, verschiedene Strömungen zu erkennen?

Die Erkennung von urbanen Umgebungsgeräuschen wird auch in Zukunft noch Grund genug weiterer Forschung sein. Große Internetkonzerne und öffentlich finanzierte Forschungsinstitutionen sind auch weiterhin auf der Suche nach immer besseren Verfahren, um die Genauigkeit und Generalisierung zu verbessern. Die Motivationen dabei sind sicherlich unterschiedlich, auch wenn das Ziel dasselbe ist.

## LITERATUR

[1] 2017. *c't wissen Smart Home (2017/2018): Bequemer leben mit intelligenter Technik*. Heise Media. https://books.google.de/books?id=qzQ7DwAAQBAJ

[2] 2018. LibROSA. (2018). Retrieved June 26, 2019 from https://librosa.github.io/librosa/

[3] 2019. Why Tensorflow. (2019). Retrieved June 26, 2019 from https://www.tensorflow.org/about/

[4] V. Kishore Ayyadevara. 2018. *Convolutional Neural Network*. Apress, Berkeley, CA, 179–215. https://doi.org/10.1007/978-1-4842-3564-5_9

[5] V. Kishore Ayyadevara. 2018. *Recurrent Neural Network*. Apress, Berkeley, CA, 217–257. https://doi.org/10.1007/978-1-4842-3564-5_10

[6] Peter I. Cowling, Harvey J. Greenberg, and Kenneth De Jong. 2013. *Artificial Intelligence*. Springer US, Boston, MA, 78–83. https://doi.org/10.1007/978-1-4419-1153-7_42

[7] Saul I. Gass and Michael C. Fu (Eds.). 2013. *Machine Learning*. Springer US, Boston, MA, 909–909. https://doi.org/10.1007/978-1-4419-1153-7_200425

[8] Michael Paluszek and Stephanie Thomas. 2019. *An Overview of Machine Learning*. Apress, Berkeley, CA, 1–18. https://doi.org/10.1007/978-1-4842-3916-2_1

[9] V. U. Reddy. 1998. On fast Fourier transform. *Resonance* 3, 10 (01 Oct 1998), 79–88. https://doi.org/10.1007/BF02841425

[10] Tara Sainath and Carolina Parada. 2015. Convolutional Neural Networks for Small-Footprint Keyword Spotting. In *Interspeech*.

[11] Pavan Sanagapati. 2018. Urban Sound Classification. (2018). Retrieved June 09, 2019 from https://www.kaggle.com/pavansanagapati/urban-sound-classification

[12] Sameh Souli and Zied Lachiri. 2011. Environmental Sounds Classification Based on Visual Features. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, César San Martin and Sang-Woon Kim (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 459–466.

[13] Pang-Ning Tan. 2016. *Neural Networks*. Springer New York, New York, NY, 1–5. https://doi.org/10.1007/978-1-4899-7993-3_560-2

[14] Christian Wolff. [n. d.]. Time-Domain versus Frequency-Domain. ([n. d.]). Retrieved June 26, 2019 from http://www.radartutorial.eu/10.processing/sp53.de.

html
[15] Min Yang. 2019. Automatic Urban Sound Classification with CNN. (2019). Retrieved June 26, 2019 from https://www.kaggle.com/mychen76/automatic-urban-sound-classification-with-cnn/notebook
[16] Ruo-Yu Yang and Rahul Rai. 2019. Machine auscultation: enabling machine diagnostics using convolutional neural networks and large-scale machine audio data. *Advances in Manufacturing* (21 May 2019). https://doi.org/10.1007/s40436-019-00254-5

## A IMPLEMENTIERUNG CNN

Vgl. Yang [15]

```
import keras
from keras.layers import Activation, Dense, Dropout, Conv2D, \
                         Flatten, MaxPooling2D
from keras.models import Sequential
from keras.callbacks import EarlyStopping, ReduceLROnPlateau,ModelCheckpoint,TensorBoard, ProgbarLogger
from sklearn.model_selection import train_test_split
import librosa
import librosa.display
import numpy as np
import pandas as pd
import random
import warnings
warnings.filterwarnings('ignore')

#object serialization
import _pickle as cPickle #python 3 change
import os

%matplotlib inline
#%%
#enable memory profiler for memory management usage %%memit
from memory_profiler import memory_usage
%load_ext memory_profiler

#enable garbage collection control
import gc
gc.enable()
#%%
#progress tracker
from tqdm import tqdm, tqdm_notebook
#%%
# when set to TRUE, training data get loaded from a saved serialized data object file
# All audio files data get saved to a serialized object file to save reloading time on training runs
#
# Note:
# On first time run, if serialized file doesn't exist, this flag will get overridient
#
SKIP_AUDIO_RELOAD = True
#%%
from google.colab import drive
drive.mount('/content/gdrive')
#%%
#location of the sound files
INPUT_PATH='/content/gdrive/My Drive/Studium/Angewandtes maschinelles Lernen/urban-sound-classification'

TRAIN_INPUT=INPUT_PATH+'/train'
TRAIN_AUDIO_DIR=TRAIN_INPUT+'/Train'

TEST_INPUT=INPUT_PATH+'/test'
TEST_AUDIO_DIR=TEST_INPUT+'/Test'
#%%
def load_input_data(pd, filepath):
    # Read Data
    data = pd.read_csv(filepath)
    return data
#%%
# training file
TRAIN_FILE=TRAIN_INPUT+'/train.csv'

#show info
train_input=load_input_data(pd,TRAIN_FILE)
train_input.head()
#%%
# training file
TEST_FILE=TEST_INPUT+'/test.csv'

#show info
test_input=load_input_data(pd,TEST_FILE)
test_input.head()
#%%
#labels
valid_train_label = train_input[['Class']]
#x=data['label'].unique()
valid_train_label.count()

#unique classes
x = train_input.groupby('Class')['Class'].count()
x
#%%
# train data size
valid_train_data = train_input[['ID', 'Class']]
valid_train_data.count()
#%%
# test data size
valid_test_data = test_input[['ID']]
valid_test_data.count()
#%%
# sample-1 load
sample1=TRAIN_AUDIO_DIR+'/943.wav'
duration=2.97
sr=22050

y, sr = librosa.load(sample1, duration=duration, sr=sr)
```

```
ps = librosa.feature.melspectrogram(y=y, sr=sr)

input_length=sr*duration
offset = len(y) - round(input_length)
print ("input:", round(input_length), "␣load:", len(y ) , "␣offset:", offset)
print ("y␣shape:", y.shape, "␣melspec␣shape:", ps.shape)
#%%
# sample-1 waveplot
librosa.display.waveplot(y,sr)
#%%
# sample-1: audio
import IPython.display as ipd
ipd.Audio(sample1)
#%%
# sample-1: spectrogram
librosa.display.specshow(ps, y_axis='mel', x_axis='time')
#%%
# sample-2 load
sample2=TRAIN_AUDIO_DIR+'/1.wav'
duration=2.97
sr=22050

y2, sr2 = librosa.load(sample2, duration=duration, sr=sr)
ps2 = librosa.feature.melspectrogram(y=y2, sr=sr2)

input_length=sr*duration
offset = len(y) - round(input_length)
print ("input:", round(input_length), "␣load:", len(y ) , "␣offset:", offset)
print ("y␣shape:", y.shape, "␣melspec␣shape:", ps2.shape)
#%%
# sample-2: audio
ipd.Audio(sample2)
#%%
# sample-2: spectrogram
librosa.display.specshow(ps2, y_axis='mel', x_axis='time')
ps.shape
#%%
#training audio files
valid_train_data['path'] = TRAIN_AUDIO_DIR+'/' + train_input['ID'].astype('str')+".wav"
print ("sample",valid_train_data.path[1])
valid_train_data.head(5)
#%%
#test audio files
valid_test_data['path'] = TEST_AUDIO_DIR+'/' + test_input['ID'].astype('str') +".wav"
print ("sample",valid_test_data.path[1])

valid_test_data.head(5)
#%%
#
# set duration on audio loading to make audio content to ensure each training data have same size
#
# for instance, 3 seconds audio will have 128*128 which will be use on this notebook
#
def audio_norm(data):
    max_data = np.max(data)
    min_data = np.min(data)
    data = (data-min_data)/(max_data-min_data+0.0001)
    return data-0.5

#fix the load audio file size
audio_play_duration=2.97

def load_audio_file(file_path, duration=2.97, sr=22050):
    #load 5 seconds audio file, default 22 KHz default sr=22050
    # sr=resample to 16 KHz = 11025
    # sr=resample to 11 KHz = 16000
    # To preserve the native sampling rate of the file, use sr=None
    input_length=sr*duration
    # Load an audio file as a floating point time series.
    # y : np.ndarray [shape=(n,) or (2, n)] - audio time series
    # sr : number > 0 [scalar] - sampling rate of y
    y, sr = librosa.load(file_path,sr=sr, duration=duration)
    dur = librosa.get_duration(y=y)
    #pad output if audio file less than duration
    # Use edge-padding instead of zeros
    #librosa.util.fix_length(y, 10, mode='edge')
    if (round(dur) < duration):
        offset = len(y) - round(input_length)
        print ("fixing␣audio␣length␣:", file_path)
        print ("input:", round(input_length), "␣load:" , len(y) , "␣offset:", offset)
        y = librosa.util.fix_length(y, round(input_length))
    # y = audio_norm(y)
    # using a pre-computed power spectrogram
    # Short-time Fourier transform (STFT)
    #D = np.abs(librosa.stft(y))**2
    #ps = librosa.feature.melspectrogram(S=D)
    ps = librosa.feature.melspectrogram(y=y, sr=sr)
    return ps
#%%
%%time
%%memit
# Dataset
train_audio_data = []
```

```
train_object_file='/content/gdrive/My Drive/Studium/Angewandtes maschinelles Lernen/prepared_data_models/saved_train_audio_data.p'

#override the reload flag if serized file doesn't exist
#if not os.path.isfile(train_object_file):
# SKIP_AUDIO_RELOAD = False

#load training data
if SKIP_AUDIO_RELOAD is True:
    print ("skip re-loading TRAINING data from audio files")
else:
    print ("loading train audio data, may take more than 15 minutes. please wait!")
    for row in tqdm(valid_train_data.itertuples()):
        ps = load_audio_file(file_path=row.path, duration=2.97)
        if ps.shape != (128, 128): continue
        train_audio_data.append( (ps, row.Class) )
    print("Number of train samples: ", len(train_audio_data))
#Number of train samples: 5382
#peak memory: 1120.78 MiB, increment: 598.80 MiB
#CPU times: user 14min 42s, sys: 9min 28s, total: 24 min 10s
#Wall time: 1h 43min 30s
#%%
# load saved audio object
if SKIP_AUDIO_RELOAD is True:
    train_audio_data = cPickle.load(open(train_object_file, 'rb'))
    print ("loaded train data [%s] records from object file" % len(train_audio_data))
else:
    cPickle.dump(train_audio_data, open(train_object_file, 'wb'))
    print ("saved loaded train data :",len(train_audio_data))
#%%
%%time
%%memit
#load test data
test_audio_data = []
test_object_file='/content/gdrive/My Drive/Studium/Angewandtes maschinelles Lernen/prepared_data_models/saved_test_audio_data.p'

#override the reload flag if serized file doesn't exist
#if not os.path.isfile(test_object_file):
# SKIP_AUDIO_RELOAD = False

if SKIP_AUDIO_RELOAD is True:
    print ("skip re-loading TEST data from audio files")
else:
    print ("loading test audio data, may take more than 15 minutes. please wait!")
    for row in tqdm(valid_test_data.itertuples()):
        ps = load_audio_file(file_path=row.path, duration=2.97)
        if ps.shape != (128, 128):
            print ("***data shape is wrong, replace it with zeros", ps.shape, row.path)
            ps = np.zeros([128, 128])
            #continue
        test_audio_data.append( (ps, row.ID) )
    print("Number of train samples: ", len(train_audio_data))

#Number of train samples: 5382
#peak memory: 3334.95 MiB, increment: 0.84 MiB
#CPU times: user 9min 4s, sys: 9min 45s, total: 18min 50s
#Wall time: 1h 14min 31s
#%%
# load saved data
if SKIP_AUDIO_RELOAD is True:
    test_audio_data = cPickle.load(open(test_object_file, 'rb'))
    print ("loaded test data [%s] records from object file" % len(test_audio_data))
else:
    cPickle.dump(test_audio_data, open(test_object_file, 'wb'))
    print ("save loaded test data :", len(test_audio_data))
#%%
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from keras.utils import to_categorical
from numpy import argmax

# get a set of unique text labels
list_labels = sorted(list(set(valid_train_data.Class.values)))
print ("unique text labels count: ",len(list_labels))
print ("labels: ",list_labels)

# integer encode
label_encoder = LabelEncoder()
label_integer_encoded = label_encoder.fit_transform(list_labels)
print("encoded labelint values", label_integer_encoded)

# one hot encode
encoded_test = to_categorical(label_integer_encoded)
inverted_test = argmax(encoded_test[0])
#print(encoded_test, inverted_test)
```

```
#map filename to label
file_to_label = {k:v for k,v in zip(valid_train_data.path.values, valid_train_data.ID.values)}

# Map integer value to text labels
label_to_int = {k:v for v,k in enumerate(list_labels)}
#print ("test label to int ",label_to_int["Applause"])

# map integer to text labels
int_to_label = {v:k for k,v in label_to_int.items()}
#%%
#full dataset
dataset = train_audio_data
random.shuffle(dataset)

RATIO=0.9
train_cutoff= round(len(dataset) * RATIO)
train = dataset[:train_cutoff]
test = dataset[train_cutoff:]

X_train, y_train = zip(*train)
X_test, y_test = zip(*test)

# Reshape for CNN input
X_train = np.array([x.reshape( (128, 128, 1) ) for x in X_train])
X_test = np.array([x.reshape( (128, 128, 1) ) for x in X_test])

print ("train ",X_train.shape, len(y_train))
print ("test ", X_test.shape, len(y_test))
#%%
# Apply sck-learn label text encoding to integer
label_encoder = LabelEncoder()
y_train_integer_encoded = label_encoder.fit_transform(y_train)
y_test_integer_encoded = label_encoder.fit_transform(y_test)
#%%
# Apply Keras One-Hot encoding for classes
y_train = np.array(keras.utils.to_categorical(y_train_integer_encoded, len(list_labels)))
y_test = np.array(keras.utils.to_categorical(y_test_integer_encoded, len(list_labels)))
#%%
#split up test into test and validation
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.30, random_state=42)

print ("test ",X_test.shape, len(y_test))
print ("valid ", X_val.shape, len(y_val))
#%%
# build convolution model
# input shape = (128, 128, 1)
model = Sequential()
input_shape= X_train.shape[1:]

model.add(Conv2D(24, (5, 5), strides=(1, 1), input_shape=input_shape))
model.add(MaxPooling2D((4, 2), strides=(4, 2)))
model.add(Activation('relu'))

model.add(Conv2D(48, (5, 5), padding="valid"))
model.add(MaxPooling2D((4, 2), strides=(4, 2)))
model.add(Activation('relu'))

model.add(Conv2D(48, (5, 5), padding="valid"))
model.add(Activation('relu'))

model.add(Flatten())
model.add(Dropout(rate=0.5))

model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(rate=0.5))

model.add(Dense(len(list_labels)))
model.add(Activation('softmax'))
model.summary()
#%%
%%time
%%memit

# NOTE:
# Increase number if epochs from 1 to 60 or 100 for higher prediction accuracy
# default is set to 1 for faster commit
MAX_EPOCHS=20
MAX_BATCH_SIZE=23
# learning rate reduction rate
MAX_PATIENT=2

# saved model checkpoint file
best_model_file="./best_model_trained.hdf5"

# callbacks
# removed EarlyStopping(patience=MAX_PATIENT)
callback=[ReduceLROnPlateau(patience=MAX_PATIENT, verbose=1),
          ModelCheckpoint(filepath=best_model_file, monitor='loss', verbose=1, save_best_only=True)]

#compile
model.compile(optimizer="Adam",loss="categorical_crossentropy",metrics=['accuracy'])

#train
print('training started.... please wait!')
```

```
history = model.fit(x=X_train, y=y_train,
                    epochs=MAX_EPOCHS,
                    batch_size=MAX_BATCH_SIZE,
                    verbose=0,
                    validation_data= (X_val, y_val),
                    callbacks=callback)
print('training finished')

# quick evaluate model
print('Evaluate model with test data')
score = model.evaluate(x=X_test,y=y_test)

print('test loss:', score[0])
print('test accuracy:', score[1])
#%%
%%time
%%memit

import matplotlib.pyplot as plt
#Plot loss and accuracy for the training and validation set.
def plot_history(history):
    loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' not in s]
    val_loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' in s]
    acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' not in s]
    val_acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' in s]
    if len(loss_list) == 0:
        print('Loss is missing in history')
        return
    plt.figure(figsize=(22,10))
    ## As loss always exists
    epochs = range(1,len(history.history[loss_list[0]]) + 1)
    ## Accuracy
    plt.figure(221, figsize=(20,10))
    ## Accuracy
    # plt.figure(2,figsize=(14,5))
    plt.subplot(221, title='Accuracy')
    for l in acc_list:
        plt.plot(epochs, history.history[l], 'b', label='Training accuracy (' + str(format(history.history[l][-1],'.5f'))+')')
    for l in val_acc_list:
        plt.plot(epochs, history.history[l], 'g', label='Validation accuracy (' + str(format(history.history[l][-1],'.5f'))+')')
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    ## Loss
    plt.subplot(222, title='Loss')
    for l in loss_list:
        plt.plot(epochs, history.history[l], 'b', label='Training loss (' + str(format(history.history[l][-1],'.5f'))+')')
    for l in val_loss_list:
        plt.plot(epochs, history.history[l], 'g', label='Validation loss (' + str(format(history.history[l][-1],'.5f'))+')')
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

# plot history
plot_history(history)
#%%
#Evaluate model use Keras reported accuracy:
score = model.evaluate(X_train, y_train, verbose=0)
print ("model train data score       :",round(score[1]*100) , "%")

score = model.evaluate(X_test, y_test, verbose=0)
print ("model test data score        :",round(score[1]*100) , "%")

score = model.evaluate(X_val, y_val, verbose=0)
print ("model validation data score  :", round(score[1]*100), "%")
#%%
print ("Prediction with [train] data")
y_pred = model.predict_classes(X_train)
missed=[]
matched=[]
for i in range(len(y_pred)):
    y_val_label_int = argmax(y_train[i])
    if (y_pred[i]!=y_val_label_int):
        missed.append( (y_pred[i], "-", int_to_label[y_pred[i]], " - ", int_to_label[y_val_label_int] ))
    else:
        matched.append((y_pred[i], "-", int_to_label[y_pred[i]], " - ", int_to_label[y_val_label_int]))

print ("  |__match     :", len(matched))
print ("  |__miss      :", len(missed))
print ("  |__accuracy  :", round((len(matched)-len(missed))/len(matched)*100,2), "%")
print ("")
#print ("Value missed : \n",missed)

# show sample results
print ("---samples---")
```

```
for i in range(5):
    print (i,"predict = ", int_to_label[y_pred[i]])
    print (i,"original=", int_to_label[argmax(y_train
        ↪ [i])])
    print ("")
#%%
# prediction class
print ("Prediction with [test] data")
y_pred = model.predict_classes(X_test)
missed=[]
matched=[]
for i in range(len(y_pred)):
    y_val_label_int = argmax(y_test[i])
    if (y_pred[i]!=y_val_label_int):
        missed.append( (y_pred[i], "-", int_to_label[
            ↪ y_pred[i]], " - ", int_to_label[
            ↪ y_val_label_int] ))
    else:
        matched.append((y_pred[i], "-", int_to_label[
            ↪ y_pred[i]], " - ", int_to_label[
            ↪ y_val_label_int]))

print ("  |__match    :", len(matched))
print ("  |__miss     :", len(missed))
print ("  |__accuracy :", round((len(matched)-len(
    ↪ missed))/len(matched)*100,2), "%")
print ("")
#print ("Value missed : \n",missed)

# show sample results
print ("---samples---")
for i in range(8):
    print (i,"predict = ", int_to_label[y_pred[i]])
    print (i,"original=", int_to_label[argmax(y_test[
        ↪ i])])
    print ("")
#%%
# prediction class
print ("Prediction with [validation] data")
y_pred = model.predict_classes(X_val)
missed=[]
matched=[]
for i in range(len(y_pred)):
    y_val_label_int = argmax(y_val[i])
    if (y_pred[i]!=y_val_label_int):
        missed.append( (y_pred[i], "-", int_to_label[
            ↪ y_pred[i]], " - ", int_to_label[
            ↪ y_val_label_int] ))
    else:
        matched.append((y_pred[i], "-", int_to_label[
            ↪ y_pred[i]], " - ", int_to_label[
            ↪ y_val_label_int]))

print ("  |__match    :", len(matched))
print ("  |__miss     :", len(missed))
print ("  |__accuracy :", round((len(matched)-len(
    ↪ missed))/len(matched)*100,2), "%")
print ("")
#print ("Value missed : \n",missed)

# show sample results
print ("---samples---")
for i in range(8):
    print (i,"predict = ", int_to_label[y_pred[i]])
    print (i,"original=", int_to_label[argmax(y_val[i
        ↪ ])])
    print ("")
#%%
print ("test data size ",len(test_audio_data))
sub_test = test_audio_data[1:22]
tx_test, ty_test = zip(*test_audio_data)

# make prediction
tx_test2 = np.array([x.reshape((128, 128, 1)) for x
    ↪ in tx_test])
print ("test data shape ", tx_test2.shape)
#%%
# run prediction data
y_pred = model.predict_classes(tx_test2, batch_size
    ↪ =1)
print ( len(y_pred), len(tx_test2))
#%%
# save result for submission
prediction_output_file='prediction_result_1.csv'
with open(prediction_output_file,"w") as file:
    file.write("ID,Prediction\n")
    i=0
    for i in range( (len(valid_test_data)-1)) :
        #print(i, y_pred[i])
        file.write(str(valid_test_data['ID'][i])+','+
            ↪ int_to_label[y_pred[i]])
        file.write('\n')
        i=i+1

print (len(y_pred))
output = pd.read_csv(prediction_output_file)
output.head(20)
```

# pix2code

Analyse und Erweiterung

Florian Kriegel
Hochschule fur angewandte Wissenschaften Hof
florian.kriegel@hof-university.de

Markus Solisch
Hochschule fur angewandte Wissenschaften Hof
markus.solisch@hof-university.de

## ABSTRACT

Die vorliegende Arbeit behandelt neben der Analyse des Neuronalen Netzes pix2code auch durch die Studierenden eingebaute Erweiterungen.

pix2code bietet umfassende Funktionalitäten im Bereich der Bildumsetzung mittels neuronaler Netze an. Jedoch stellte sich nach eingehender Analyse und Verwendung des Netzes heraus, dass es an einigen Stellen an Bedienbarkeit und Nachvollziehbarkeit der Arbeit des Models fehlte. Dafür wurden neben Analysefunktionen wie Tensorbard und Modelcheckpoints auch der Einbau eines Evaluierungsdatensatzes realisiert. Darüber hinaus wurde auch ein Webfrontend zur besseren Verwendbarkeit konzipiert.

In den nachfolgenden Abschnitten werden das Model selbst, sowie auch die implementierten Optimierungen vorgestellt.

## 1. Einleitung

Bei der Implementierung von Software kommt es oft vor, dass Entwickler von Designern entworfene Modelle umsetzen müssen. Dies ist aufwändig und beansprucht Zeit, die dann eventuell bei der Entwicklung der Logik der Anwendung fehlt. Aus diesem Grund wurde das Projekt „pix2code“, vom französischen Informatiker Tony Beltramelli, ins Leben gerufen[1]. pix2code ermöglicht es aus Bild-Dateien, welche den Screenshot einer Webseite, Android- oder iOS Oberfläche zeigen, Programmcode in der gewünschten Zielsprache zu generieren. Beltramelli stellt das Projekt kostenfrei auf Github zu Forschungszwecken zur Verfügung.

In dieser Arbeit soll pix2code nun im Rahmen des Seminars „Angewandtes Maschinelles Lernen“ analysiert und eingebaute Erweiterungen erläutert werden. Besonderer Fokus wurde auf das Model gelegt, welches für die Generierung von HTML Code aus Bildern konzipiert wurde. Im Zuge der Erarbeitung wurde das Projekt aufgrund der hohen Komplexität in mehrere Teile aufgespalten, die von den Teammitgliedern gesondert untersucht und verstanden wurden. Dazu zählen insbesondere auch das Aufarbeiten und Visualisieren der verwendeten Code-Klassen und Algorithmen. Anschließend wurden die erarbeiteten Erkenntnisse im Team diskutiert und in Vorträgen dem Plenum vorgestellt.

Neben der theoretischen Analyse anhand des Quellcodes, sowie veröffentlichten wissenschaftlichen Papers[2] wurde das Model trainiert und in einigen Beispielanwendung getestet. Dabei wurde die Funktionalität von pix2code in Teilen erweitert, indem ein zweites neuronales Netz zur Generierung von Beispieltexten für die entstandenen Webseiten eingebaut wurde. Außerdem sollte mit einer Kombination aus ReactJS und Python Flask eine Oberfläche erzeugt werden, die auch jüngeren Interessenten ohne Vorwissen die Möglichkeit gibt die Funktionalität von pix2code zu testen.

Im folgenden Abschnitt wird deutlich, wie das pix2code Model aufgebaut ist und welche Architektur während der unterschiedlichen Phasen wie Training und Testing verwendet wird.

1 Beltramelli, Tony: pix2code (Stand 13. Dezember 2017) https://github.com/tonybeltramelli/pix2code [08. Juli 2019]

2 Beltramelli, Tony: pix2code: Generating Code from a Graphical User Interface Screenshot (Stand 22. Mai 2017) https://arxiv.org/abs/1705.07962 [08. Juli 2019]

## 2. Verwandte Arbeiten

Bereits vor pix2code gab es einige Ansätze zur automatisierten Generierung lauffähigen Programmcodes. Einige dieser Ansätze hat Beltramelli in seinem Paper erwähnt.

Eines der Projekte, die bereits vor pix2code ins Leben gerufen wurden, ist beispielsweise DeepCoder[3]. Dieses neuronale Netz ermöglicht die Generierung von Code, indem es aus Seiten wie z.B. StackOverflow Programmcode extrahiert und zur Generierung eigener Software verwendet.

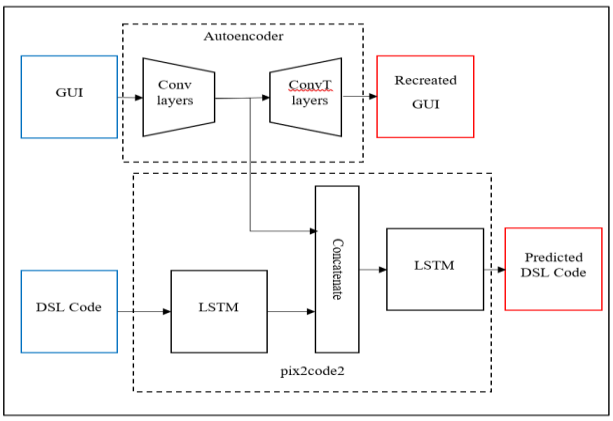
In einer weiteren Arbeit der Autoren Nguyen et al. wird versucht aus gegebenen Screenshots von Android User Interfaces, den zugehörigen Programmcode zu rekonstruieren[4]. Folglich sind hier einige parallelen zu pix2code erkennbar. Beltramelli selbst bezeichnet dies als das am nächsten zu pix2code verwandte Projekt. Eine Implementierung, welche zeitgleich mit pix2code begonnen wurde ist Sketch2Code von airbnb[5]. Im Gegensatz zu pix2code, welches der Entwickler selbst nur als Forschungsprojekt bezeichnet, ist Sketch2Code für den tatsächlichen Einsatz im Softwareproduktionsprozess vorgesehen. Ferner wird, im Unterschied zu pix2code, versucht aus handgemalten Skizzen lauffähigen Code zu generieren. Airbnb nutzt diese Technologie schon heute in Meetings um Ideen zu testen.

Bemerkenswert ist die Tatsache, dass einige Entwickler die Idee von pix2code aufgegriffen und darauf aufbauend eigene Netze entwickelt haben. Auch wenn pix2code in diesem Sinne keine neue Epoche der codegenerierenden neuronalen Netze eingeleitet hat, sollen im Folgenden, neben den bereits vor pix2code existenten Implementierungen auch diejenigen Ansätze vorgestellt werden, die auf pix2code aufbauend oder zeitgleich dazu entwickelt wurden.

Implementierungen, welche auf pix2code basieren sind oftmals ein Versuch die Präzision des Models von pix2code zu optimieren, dessen Genauigkeit mit etwa 77% verbesserungsbedürftig ist. So existiert beispielsweise das Projekt pix2code2[6]. Hierbei wird versucht, durch separat voneinander trainierter und schließlich zu einem Gesamtmodel zusammengeführter Komponenten bessere Ergebnisse in der Genauigkeit zu erzielen. Auch die Architektur wurde, entsprechend der nachfolgenden Abbildung optimiert.

## 3. Aufbau des Models

Im Folgenden wird der Aufbau des Models von pix2code erklärt. Da der ursprüngliche Entwickler sich in seiner veröffentlichten Arbeit auch auf bereits bekannte und bewährte Methoden bezieht werden diese ebenfalls aufgenommen.




Bei der Erstellung des Models wurde besonders auf die Open Source Deep-Learning Bibliothek „Keras“ in Verbindung mit dem von Google entwickelten Framework zur datenstrom-orientierten Programmierung, „TensorFlow“ gesetzt[7][8].

Um eine einfache und einheitliche Möglichkeit des Speicherns und Ladens der von pix2code erzeugten Gewichte zu erhalten, wurde vom Entwickler eine Oberklasse „AModel“ definiert, die zwei Methoden eben dafür bereitstellt. Ferner befindet sich das eigentliche Model mit den einzelnen Schichten in einer eigenen Klasse.

Essentiell für den Aufbau und die gute Erfolgsrate von pix2code ist, dass das Model nicht nur aus einem einzigen neuronalen Netz besteht, sondern stattdessen drei Netze für ihren ganz speziellen Anwendungsbereich konzipiert wurden. In den nachfolgenden Absätzen sollen die einzelnen Netze nun genauer betrachtet werden.

3 M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. arXiv preprint arXiv:1611.01989, 2016.

4 T. A. Nguyen and C. Csallner. Reverse engineering mobile application user interfaces with remaui (t). In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, pages 248–259. IEEE, 2015.

5 Airbnb: Sketch2Code: Sketching Interfaces, Generating code from low fidelity wireframes (Stand unknown) https://airbnb.design/sketching-interfaces/ [08. Juli 2019]

6 Fjbriones: pix2code2: An attempt to improve pix2code through pretrained autoencoders (Stand 30. Dezember 2017) https://github.com/fjbriones/pix2code2 [08. Juli 2019]

7 Keras-Team: keras, deep learning for humans (Stand 07. Juli 2019) https://github.com/keras-team/keras [08. Juli 2019]

8 Tensorflow: tensorflow, an open source machine learning framework for everyone (Stand 08. Juli 2019) https://github.com/tensorflow/tensorflow [08. Juli 2019]

### 3.1. Image Model

Um die Bilder, aus denen anschließend lauffähiger Programmcode generiert werden soll, verarbeiten zu können bedarf es eines Ansatzes, bei welchem bestimmte Charakteristika und Merkmale des Eingabebildes gelernt und gespeichert werden. Für diesen Anwendungsfall eignet sich deshalb das unüberwachte „Feature Learning".

Die Keras-API bietet unter den beiden Oberbegriffen „Sequential"-Learning und „Functional"-API zwei unterschiedliche Strukturierungsmöglichkeit mit ihren jeweiligen Vor- und Nachteilen an. Um zu verstehen, warum pix2code auf ein sequentielles Lernen setzt ist es wichtig die beiden Herangehensweisen zu verstehen und um ihre Eigenschaften zu wissen.

Ein funktionales Model eignet sich in der Praxis vor allem für komplexe Netzarchitekturen. So kann es zum Beispiel beim Einsatz mehrerer Eingabequellen oder multipler Ausgabequellen sinnvoll sein auf die Functional-API zurückzugreifen. Auch eignet sich diese Art der Architektur für die Wiederverwendung von Neuronenschichten zu einem späteren Zeitpunkt. Eine solche Verwendung ist zwar auch mit einem sequentiellen Model möglich, dennoch lässt sich es sich einfacher mit einem funktionalen Model realisieren.

Im konkreten Fall „pix2code" gibt es nur eine Eingabequelle für das ImageModel, nämlich das Abbild der zu generierenden Webseite. Der Einsatz eines funktionalen Models ist auch unter dieser Voraussetzung grundsätzlich möglich; seine längere Laufzeit im Vergleich zu einem sequentiellen Model aber nur schwer rechtfertigbar.

Im Bereich der Bildanalyse sind sogenannte Convolutional Neural Networks seit einigen Jahren das Mittel der Wahl. So bedient sich auch pix2code einem Aufbau ähnlich dem bewährten „VGGNet", welches 2015 von zwei Mitarbeitern der University of Oxford erstmals vorgestellt wurde[9].

Dem sequentiellen Model von pix2code werden insgesamt sechs Faltungsschichten, unterschiedlicher Größe (32, 64, 128 Schichten), sogenannte „Convolutional Layer" hinzugefügt. Nach jeder zweiten Faltungsschicht wird eine Filterschicht eingefügt, welche durch sogenanntes MaxPooling mit einem 2x2 Filter die Dimension der Eingabematrix verkleinert. So gehen zwar Bild-Informationen verloren, wodurch jedoch die Performanz des Netzes insgesamt deutlich erhöht wird. Um ein Auswendiglernen oder „Overfitting" des Models zu vermeiden wird eine Dropoutschicht eingefügt, welche dafür sorgt, dass 25% der verwendeten Neuronen pro Trainingseingabe ausgeschalten werden. Den Abschluss des ImageModels bilden zwei vollverknüpfte Schichten, welche das prozessierte Eingabebild schließlich auf einen eindimensionalen Vektor abbilden.

Wird nun ein Trainingsbild eingegeben ergibt sich folgender Ablauf: Das Bild wird auf eine Größe von 256x256 Pixel skaliert, anschließend mithilfe der Bibliothek OpenCV in numerische Werte umgewandelt und am Ende normalisiert[10]. Bei der Faltung der Matrizen wird ein 3x3 Aufnahmefeld gewählt, welches dann um jeweils eine Stelle verschoben wird. Dieser Schritt wird in jeder der sechs Faltungsschichten wiederholt. Gelangen die Daten zu den beiden vollverknüpften Schichten, so werden diese durch eine Glättungsschicht in einen eindimensionalen Vektor umgewandelt, um sie anschließend durch zwei aufeinanderfolgende Dense Layer zu prozessieren.

### 3.2. Sprach Model

Neben dem Erlernen von Merkmalen der Eingabebilder, hat pix2code auch die Aufgabe den erlernten Elementen die richtigen Tokens der zugehörigen DSL zuzuweisen. So weiß die Software nicht nur welche DOM-Elemente verwendet wurden, sondern kann sie auch wieder in Code umwandeln. Aus diesem Grund ist neben dem Image-Model auch noch ein Sprach Model von Nöten. Da das Erlernen des Tokenformats eine weitaus weniger komplexe Aufgabe darstellt als das im Image Model verwendete Feature Learning, ist auch die Architektur weitaus simpler.

Wie auch im Image Model wird beim Sprach Model eine sequentielle Anordnung der einzelnen Neuronen-Schichten verwendet. Dem Model werden daraufhin noch zwei sogenannte „Long Short Term Memory" kurz LSTM Schichten hinzugefügt. LSTMs haben sich in den letzten Jahren vor allem in Bereichen der Textgenerierung durchgesetzt. Sie lösen das Vanishing-Gradient-Problem, welches im schlimmsten Fall zu einem Lernstop bei dem Model führt, und ermöglichen es, Verknüpfungen zwischen unterschiedlichen Datensegmenten herzustellen, selbst wenn einzelne Neuronenschichten weit auseinander liegen. Darüber hinaus sind LSTMs besonders für die Verarbeitung sequentieller Daten oder Daten, die eine temporäre Verbindung untereinander haben geeignet. Da es sich bei pix2code ebenfalls um eine Art der Textgenerierung handelt, ist der Einsatz eines LSTMs hier nicht nur hilfreich, sondern auch ratsam.

9 K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

10 OpenCV: opencv, open source computer vision library (Stand 07. Juli 2019) https://github.com/opencv/opencv [08. Juli 2019]

### 3.3. Decoder

Um ein gegebenes Eingabebild in ein oder mehrere DSL Tokens zu überführen, bedarf es einer Struktur, die es ermöglicht die Verknüpfung zwischen Eingabebild und DSL Token zu erlernen. Üblich ist hier der Einsatz einer sogenannte Encoder-Decoder Architektur[11]. Dabei wird versucht einen durch den Encoder codierten Datensatz durch den Decoder möglichst genau zu rekonstruieren. Mathematisch lässt sich dies wie folgt ausdrücken:

- $f : X \to F$ (Encoder-Funktion)
- $g : F \to X$ (Decoder-Funktion)

Im konkreten Anwendungsfall von pix2code bedeutet dies, dass im Lernprozess des Models bei gegebenen Eingabevektoren, bestehend aus dem codierten Eingabebild, sowie den codierten DSL Token, die den dazugehörigen Quellcode repräsentieren, die Verknüpfung zwischen den beiden erlernt wird.

Nachdem das Training abgeschlossen ist, wird nur noch das Bild als einzige Eingabequelle eingespeist und das Model bildet dieses auf eine Menge von DSL Tokens ab: Verwendet wird abermals eine Schicht von zwei LSTMs, ähnlich dem Aufbau des Language-Models. Abschließend wird noch eine Dense-Schicht hinzugefügt, welche die gelernten Informationen aus allen Schichten verknüpfen und somit ein Gesamtergebnis für die Vorhersage der Token liefert.

## 4. Training des Models

Zu Beginn wurde versucht das Model lokal auf dem Rechner zu trainieren. Aufgrund der Größe des Datensatzes und der Komplexität des Models hätte dies jedoch trotz leistungsstarkem Rechner mindestens 17 Stunden gedauert.

Nach Rücksprache mit dem Dozenten wurde die von Google speziell für die Arbeit mit neuronalen Netzen optimierte Cloud Plattform „Google Colab“ verwendet, um das Training von pix2code durchzuführen[12]. Die erwähnte Komplexität des Models führt jedoch zu einem weiteren Problem: Google Colab ermöglicht es zwar eigene Daten hochzuladen. Jedoch interpretiert Google Colab zu große Datensätze als „Binary Large Object“ (BLOB), was einen direkten Upload ausschließt. Auch die Tatsache, dass alle Dateien nach der Beendigung einer Sitzung automatisch wieder gelöscht werden ist hinderlich, da trainierte Gewichte oder Speicherpunkte erhalten bleiben sollen.

Um dies zu ermöglichen wurde die Schnittstelle zur Google Drive API für den Up- und Download von eigenen Daten genutzt. Mit dieser ist es möglich das pix2code-Model in Google Drive hochzuladen und von dort aus zu nutzen. Ein weiteres Problem ergab sich dadurch, dass Zugriffe auf Google Drive viel Zeit kosten und beim Training des Models oft auf die einzelnen Datensätze zugegriffen wird. Als Lösung wurde eine Hybridstruktur geschaffen: Die Datensätze werden jedes Mal aus dem Git Repository geklont, entpackt und generiert, sodass sie sich im Datei Explorer von Google Colab befinden. Das Model selbst befindet sich jedoch weiterhin in Google Drive.

## 5. Erweiterungen

Gegenwärtig liegt das Hauptaugenmerk von pix2code auf der korrekten Umwandlung von Bildern in valide HTML Dokumente, nicht aber auf der User Experience und dem äußeren Erscheinungsbild der Webseiten.

Auch die Erstellung von Speicherpunkten während des Trainingsverlaufs oder die Erstellung von Tensorboard-Logs wurden im ursprünglichen Projekt nicht umgesetzt. Um diese Features dennoch nutzen zu können, ist ein Ziel der Studienarbeit die Erweiterung von pix2code hinsichtlich Nutzererfahrung, Stabilität und verbesserter Analysemöglichkeiten. Obgleich es sich wie vom Entwickler selbst beschrieben, nur um ein Forschungsprojekt handelt, dienen die durchgeführten Optimierungen auch anderen Studierenden und Schülern, die sich für den Bereich der Bildumsetzung mittels neuronaler Netze interessieren und in diesem Zusammenhang pix2code ebenfalls ausprobieren oder analysieren möchten.

Als weitere Verbesserung wurde ein Generator zur Erstellung eines Evaluationsdatensatzes implementiert. Dadurch ist es möglich nach jedem Trainingsdurchlauf den Fortschritt des Models anhand von unterschiedlichen Daten zu beobachten und zu dokumentieren. Dadurch entsteht die Möglichkeit den Trainingsvorgang zu optimieren.

Zuletzt wurde mithilfe von ReactJS eine Weboberfläche entwickelt, mit der interessierte und nicht versierte Besucher einen Eindruck von der Arbeits- und Funktionsweise von pix2code erhalten können.

11 AI-United-Redaktion: Autoencoder: Architekturtypen und Anwendungen (Stand: 27. Februar 2019) https://www.ai-united.de/autoencoder-architekturtypen-und-anwendungen/ [08. Juli 2019]

12 Fuat: Goodle Colab Free GPU Tutorial (Stand 26. Januar 2018) https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d [08. Juli 2019]

## 5.1. Verbesserung der User Experience

Wie im vorherigen Absatz ausgeführt, ist die Verbesserung der Nutzererfahrung ein zentraler Punkt in der Erweiterung von pix2code durch die Autoren. Um die durchgeführten Veränderungen leichter nachvollziehbar zu machen, wird im folgenden Absatz zuerst auf die ursprüngliche Implementierung eingegangen, um dann zwei Optimierungsmöglichkeiten vorzustellen und deren Vor- und Nachteile zu evaluieren.

Für die Umwandlung des generierten DSL Codes in valide HTML Dokumente dient vor allem das Skript „web_compiler.py“ in Verbindung mit „compiler.py“, welches die unabhängige Oberklasse für alle Compiler darstellt. Um die Transformation von DSL in HTML zu starten, muss folgender CLI Befehl ausgeführt werden.

```
./web-compiler.py <input file path>.gui
```

Nun wird die Datei vom Compiler ausgewertet und in HTMLNode-Objekte umgewandelt, welche bereits den auszuführenden HTML Code enthalten. Dieser Vorgang lässt sich in folgenden Schritten darstellen.

1. Generierung des initialen Node Objekts unabhängig von eingegebener DSL Datei

2. Setzen des erstellten Node Objekts als globalen Elternknoten

3. Für jeden Token im DSL File:

    3.1. Falls Token = öffnendes Tag

    3.1.1 Erstellen eines Node Objekts mit Verweis auf den zugehörigen Elternknoten

    3.1.2. Setzen des globalen Elternknotens auf das neu erstellte Node Objekt

    3.2. Falls Token = schließendes Tag

    3.2.1. Setzen des globalen Elternknotens-Objekts auf den „Großelternknoten“

    3.3. Falls Token != schließendes Tag und Token != öffnendes Tag:

    3.3.1. Erstellen eines Node Objekts für jeden gefundenen Token in dieser Hierarchie

    3.3.2. Hinzufügen des erstellten Node Objekts in die List der Kinder des globalen Elternknotens

Anschließend wird die erstellte Node-Hierarchie gerendert. Dabei werden Objekten die darzustellenden Texte zugewiesen. Im ursprünglichen Projekt werden diese Texte zufällig erzeugt.

An dieser Stelle soll nun die erste Möglichkeit der Optimierung vorgestellt werden, die Generierung von Texten durch ein weiteres neuronales Netz:

Es wurde versucht, das Netz mittels Sportnachrichten zu trainieren, sodass es nach Eingabe einer Startsequenz von Wörtern selbst Artikel generiert. Ein Vorteil ist, dass man je nach gewählten Trainingsparametern eine große Diversität in den Texten erreichen kann. Ein möglicher Nachteil ist die Suche eines geeigneten Datensatzes. Dieser Nachteil stellte sich als schwerwiegend dar, da die Zeit für das Seminar begrenzt war und der Fokus auf der Optimierung von pix2code liegen sollte. Hinzu kommt die Tatsache, dass die Ergebnisse trotz geeignetem Datensatz nur mäßig zufriedenstellend waren. Aus diesem Grund wurde diese Idee nach einigen Testimplementationen nicht weiterverfolgt.

Der zweite Ansatz ist die Verwendung eines Webcrawlers zur Extrahierung von echten Nachrichten. In diesem Fall konkret: Der Einsatz des webcrawling Frameworks „scrapy“[13]. Dieses ermöglicht es echte Nachrichtenartikel zu verwenden, um zufällig generierte Titel und Texte durch „echte“ Nachrichten zu ersetzen.

Dazu wurden aus dem Archiv von „sport.de“, Schlagzeilen und die zugehörigen Artikel extrahiert. Der große Vorteil hierbei ist, dass Überschrift und dazugehöriger Text gemeinsam extrahiert werden und die resultierende Kombination Sinn ergibt. Im Gegensatz zur dynamischen Generierung von Texten durch ein weiteres neuronales Netz ist es durchaus möglich, dass Texte wiederholt auftreten, was bei einer Sammlung von über 2500 unterschiedlichen Artikeln jedoch eher unwahrscheinlich ist. Nach der Evaluierung und Rücksprache mit dem Dozenten hat sich die Methode der Textextrahierung mittels Webcrawler als die bessere herausgestellt und wurde implementiert.

Nachdem die Texte extrahiert wurden, muss bei der Verwendung dieser darauf geachtet werden, dass zwar zufällige, aber dennoch zueinander gehörende Überschrift-Text-Paare in die HTML Knoten eingefügt werden. Aufgrund der Tatsache, dass die HTML-Struktur rekursiv erzeugt wird, ist es bei der Umsetzung wichtig die zugehörigen Texte den

13 Scrapy: scrapy, a fast high-level web crawling & scraping framework for Python (Stand 07. Juli 2019) https://github.com/scrapy/scrapy [08. Juli 2019]

Elternknoten während des Compile-Prozesses zuzuweisen. Das Ergebnis ist, dass allen Kindknoten eine Kombination aus Schlagzeile/Artikel zugeordnet wird. Nun enthalten alle HTML “title”-Tags den Titel der Schlagzeile und alle HTML “text”-Tags den zur Schlagzeile zugehörigen Artikel.

## 5.2. Erweiterung der Analysemöglichkeiten

Beim Start der Analyse von pix2code fiel es teilweise schwer die genaue Funktionsweise und den Aufbau des Models nachzuvollziehen. Auch aus dem Grund, dass es sich bei pix2code um ein Forschungsprojekt handelt, sollte die Transparenz der Arbeit des Netzes zu jeder Zeit ermöglicht werden. Aus diesen Gründen wurden im Rahmen der Optimierung einige Funktionalitäten eingebaut, welche die Analyse und Dokumentierung vereinfachen sollen:

Zuerst wurde das Tensorflow Plugin, Tenosorboard, mitaufgenommen[ 14 ]. Damit dieses nach jedem Trainings-durchlauf aufgerufen wird, muss die entsprechende Callback-Funktion noch ins Callback-Array der model.fit()-Funktion von pix2code integriert werden. Dadurch werden nach jeder Trainingsepoche detaillierte Daten zum jeweiligen Durchlauf gespeichert. Einsehbar werden diese durch den Start der Tensorboard Applikation. Zwar wurde Google Colab für rechenintensives Training verwendet, dennoch ist pix2code nicht direkt in Colab implementiert und existiert stattdessen als eigenständige Anwendung. Diese Tatsache macht einen kleinen Umweg nötig, um die generierten Log Dateien mittels Tensorboard einsehbar zu machen.

Zunächst wird die Tensorboard Applikation im Hintergrund gestartet:

```
#start tensorboard in the background
LOG_DIR = "/content/gdrive/My\ Drive/logs/scalars"
get_ipython().system_raw(
    'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'
    .format(LOG_DIR)
)
print(LOG_DIR)
get_ipython().system_raw('./ngrok http 6006 &')
```

Anschließend wird mittels ngrok ein Tunnel aufgebaut, welcher es möglich macht, die erzeugten Dateien einzusehen.

```
#access tensorboard using the public url by ngrok
!curl -s http://localhost:4040/api/tunnels | python3 -c \
  "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"
```

Des Weiteren kann durch Tensorboard auch die komplette Modelarchitektur dargestellt und daraus resultierend Verknüpfungen zwischen den einzelnen Schichten leichter nachvollzogen werden. Auch wird der Verlauf der Daten dadurch sehr viel klarer.

Eine weitere Schwäche des Netzes ist der Verlust aller bisherigen Trainingserfolge und Gewichtungen des aktuellen Trainingvorgangs, sollte die Anwendung während des Lernprozesses abstürzen. Besonders beim Training in der Cloud in Kombination mit einer instabilen Internetverbindung wurde dieses Szenario vergleichsweise häufig beobachtet. Um diese Schwachstelle zu eliminieren, wurden sogenannte Modelcheckpoints eingefügt. Modelcheckpoints ermöglichen es nach jeder Trainingsepoche die errechneten Gewichte zwischen zu speichern und somit den Verlust bei einem Absturz zu minimieren. Insbesondere bei langen Trainingsdauern ist dies ein häufig verwendetes Mittel, um Datenverlust vorzubeugen.

Da Tensorflow selbst immer weiterentwickelt wird und frequentiert neue Versionen und Updates veröffentlicht, ist dessen Funktionalität im optimierten Model im Moment nur eingeschränkt verfügbar, da es bei Ende eines Trainingsdurchlaufs während des Speicherns des Models zu einer Fehlermeldung kommt.

Nach einiger Recherche wurde festgestellt, dass der Fehler bereits als offizieller Bug in dem Github Repository aufgenommen wurde[15]. Als Workaround wurde die Funktion lediglich aus dem Callback Array entfernt. Nachdem der Bug behoben ist, muss sie nur erneut hinzugefügt werden und ermöglicht so wieder das Erstellen von Checkpoints.

## 5.3. Bereitstellung von Evaluierungsmöglichkeiten

Beim Training von neuronalen Netzen und der Überwachung des Trainings ist es üblich neben dem Trainingsdatensatz einen weiteren Datensatz zur Überprüfung der Trainingsergebnisse zu haben. Bei dem sogenannten Evaluierungsset handelt es sich um eine vom Trainigssatz divergente Datensammlung. Im Vergleich zu der Menge der zu trainierenden Daten ist dieser auch bedeutend kleiner. Normalerweise entspricht die Größe des Evaluierungssets circa 10% des Trainingssatzes. Verwendet wird dieser, um den Trainingserfolg zu verifizieren.

pix2code bietet zwar beide Formen der Datensätze an, verwendet den Evaluierungssatz jedoch nicht direkt beim Training des Models. Grundsätzlich ist die Verwendung eines Evaluierungssatzes nicht essentiell für das Training wichtig, dennoch lassen sich daraus Schlüsse, in Bezug auf die Arbeit

14 Tensorflow: Tensorboard (Stand 08. Juli 2019) https://www.tensorflow.org/tensorboard/ [08. Juli 2019]

15 Tensorflow Github Repository: Issue #29791 (Stand 08. Juli 2019) https://github.com/tensorflow/tensorflow/issues/29791 [08. Juli 2019]

des Models als auch auf eventuell auftretende Schwächen, ziehen.

Um die an dieser Stelle eingefügte Optimierung besser nachvollziehen zu können, wird nun kurz auf die beiden möglichen Trainingsmethoden und die Datengenerierung zur Trainingszeit eingegangen:

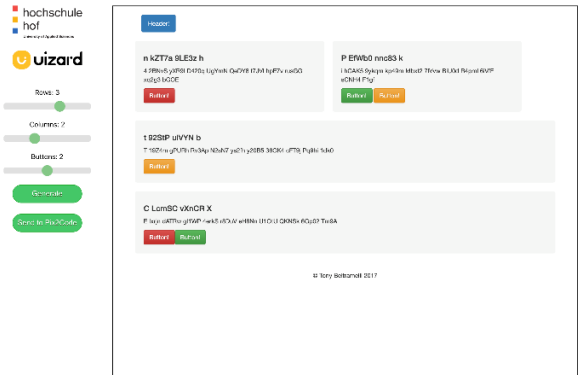
Keras bietet zum Training eines Models zwei Methoden an. Bei der einen, werden alle Daten erst komplett geladen und dann als Ganzes in die model.fit() Methode übergeben. Bei der anderen werden die Daten nach Bedarf geladen. Besonders bei großen oder vielen Daten und begrenzter Rechenkapazität ist „model.fit_generator()“ die Methode der Wahl. Hierzu muss ein Generatorobjekt übergeben werden, welches eine Methode zur konsekutiven Erzeugung von Daten bereitstellt. Neben der Erzeugung von Trainingsdaten können so auch die Evaluierungsdaten erzeugt werden.

An dieser Stelle wurde angesetzt und neben einem bereits bestehenden Trainingsdatengenerator auch ein Evaluierungsdatengenerator instanziiert und an die model.fit_generator() Methode übergeben. Der Nutzen dieser Erweiterung liegt klar auf der Hand: es findet nun eine Evaluierung des Models nach jeder Epoche statt. Darüber hinaus ermöglicht es die graphische Darstellung der Modelperformace, sowohl auf den Trainings- als auf den Evaluierungsdaten.

### 5.4. GUI für weitere Datensätze

pix2code ist vor allem ein wissenschaftlicher Versuch, aus einem Bild Code zu generieren. Beltramelli selbst hatte keine Ambitionen die Ergebnisse in einer Anwendung einzusetzen. Damit richtet sich pix2code vor allem an ein Fachpublikum, das diese Erkenntnisse mit dem Klonen des Repositories selbst nachstellen kann.

Um pix2code einem fachfremden Publikum präsentieren zu können, wurde eine grafische Oberfläche entwickelt, mit der Interessierte eine Beispielwebsite generieren und anschließend an das vortrainierte Model senden können. Der Rahmen ist durch pix2code in der Datei „mapping.json“ selbst definiert. Die verfügbaren Elemente können nur schwer selbst erweitert werden, da in diesem Fall das Model auf ebenjenes Element neu trainiert werden muss.




Auf der linken Seite kann der Nutzer mithilfe von Schiebereglern eine Obergrenze für die Anzahl der Reihen, Spalten pro Reihe und Buttons pro Spalte angeben. Mit einem Klick auf „Generate“ erscheint die selbsterzeugte Website in der rechten Hälfte.

Ursprünglich war geplant, dass mit einem Klick auf „Send to pix2code“ das Bild an einen Webserver (Flask) gesandt wird, welcher das Bild entgegennimmt, es an pix2code übergibt und auf das Ergebnis von „sample.py“ wartet. Die erhaltene Antwort soll dann als Response an die React-Anwendung übergeben werden, wo der Input und der Output dem Nutzer zum Vergleich präsentiert werden.

Bei der Implementation kam es jedoch zu der Schwierigkeit, dass das geplante Vorgehen unter anderem echte Hardware (zB. in Form eines Root-Servers) erfordert, da die von pix2code eingesetzten pip-Module eine echte CPU erforderlich machen. Da es zum Zeitpunkt der Abgabe keinen konkreten Termin für eine solche Präsentation gegeben hat, wurde auf das Mieten eines Root-Servers verzichtet und das Frontend als Konzept für eine Weiterentwicklung verstanden.

Derzeit wird mit einem Klick auf „Send to pix2code“ der Screenshot der Website stattdessen in einem neuen Tab angezeigt. Zu Testzwecken kann dieses Bild lokal abgespeichert und sample.py anschließend manuell aufgerufen werden.

## 6. Konklusion und Diskussion

In dieser Arbeit wurden, neben der Analyse von pix2code selbst, zahlreiche implementierte Erweiterungen und deren konkreter Nutzen vorgestellt. Begründet durch die Tatsache, dass pix2code ein Forschungsprojekt und dadurch für andere Interessierte ein Mehrwert entstehen soll, wurde unter anderem an der Zugänglichkeit des Models gearbeitet. So ist es durch den Einsatz von Analysewerkzeugen wie Tensorboard auch unerfahreneren Nutzern nun möglich einen leichten Einstieg in die Funktionsweise des Models zu erhalten. Auch die Präsentation des Models wurde durch das Ersetzen der zufällig generierten Texte durch echte Artikel verbessert. Ebenso ist es nun möglich durch den Einbau des Evaluierungsdatensatzgenerators den Trainingserfolg besser nachzuvollziehen und Zwischenerfolge durch Checkpoints zu speichern.

Obgleich der Tatsache, dass pix2code annehmbare Ergebnisse liefert, können diese noch verbessert werden. Durch den Einsatz eines größeren Datensatzes oder einer tieferen Netzstruktur ist es möglich, die Genauigkeit des Models zu optimieren. Dies wurde auch deutlich durch den Einbau des Evaluierungssets. Hierbei wurde festgestellt, dass das Model leicht zum Overfitting neigt. Zwar wurde versucht durch verschiedene Dropout-Raten den Fehler zu beheben, jedoch ohne merklichen Erfolg.

Abschließend hat sich herausgestellt, dass die Analyse und Anwendung durch die eingebauten Änderungen deutlich einfacher gestaltet werden können. Sei es durch die bessere Visualisierung der Arbeit des Models oder den optimierten Auftritt der generierten Webseiten.