„Optimierung verschiedener Hyperparameter einer künstlichen Intelligenz für Objekterkennung, genutzt im Rahmen eines Museums Showcase.“

1. Projektarbeit

vorgelegt am 31. August 2020

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Kurs WWI2019E

von

Luca Alexander Weissbeck

Betreuer in der Ausbildungsstätte:

IBM Deutschland GmbH

Anton Griessbach

Associate Consultant

DHBW Stuttgart:

Prof. Dr. Marcus Vogt

Inhaltsverzeichnis

Abkürzungsverzeichnis

KNN	=	Künstliches neuronales Netz
GPU	=	Graphics Processing Unit
KI	=	Künstliche Intelligenz
AI	=	Artificial Intelligence
ML	=	Machine Learning
DL	=	Deep Learning
SGD	=	Stochastic Gradient Descent
SSH	=	Secure Shell

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Das Thema der künstlichen Intelligenz (KI) ist heutzutage unumgänglich und hat längst das Nischendasein verlassen. Ob beim Einkauf im Internet oder beim Nutzen von sozialen Medien, KI ist überall in unserem Alltag vertreten. In vielen Fälle bekommt der Nutzer von der Verwendung dieser nichts mit. Der Musik-Streaming-Dienst „Spotify“ nutzt beispielsweise eine KI im Hintergrund, die dem Nutzer der Applikation neue Musik vorschlägt, basierend auf dessen individuellen Musikgeschmack[1]. Große Fortschritte der KI werden heutzutage unter Anderem in den Bereichen der Medizin und des autonomen Fahrens gemacht. Besonders europäische Start-ups im Bereich Robotik und KI sind erfolgreich im Anwendungsbereich der KI für autonomes Fahren[2]. Die Relevanz des Themas wird durch eine Analyse der Suchstatistiken des Wortes *Artificial Intelligence* in der Suchmaschine Google besonders deutlich. Im Februar 2020 erreichte das Stichwort die höchste Anzahl an Suchanfragen seit 2004 (siehe Abbildung 1).

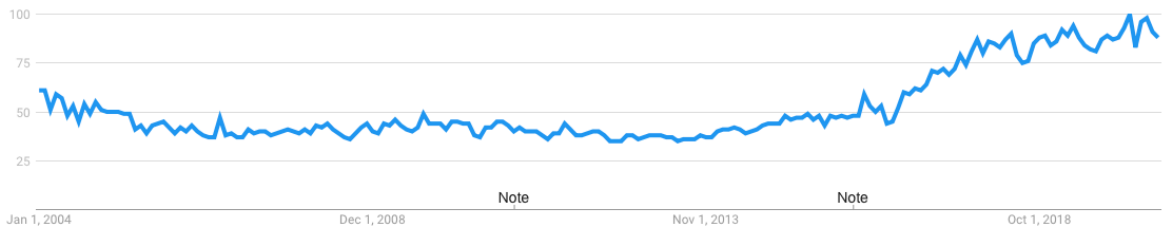

Abbildung 1: Google Trends Artificial Intelligence[3]

Ein wichtiger Anwendungsfall der KI ist die Objekterkennung, welche die technische Grundlage für das autonome Fahren darstellt. Konkrete Beispiele der Objekterkennung sind jedoch nur selten leicht verständlich und daher für Erklärungszwecke ungeeignet. In der folgenden Arbeit wird sich mit KI im Zusammenhang mit der Objekterkennung beschäftigt.

2020 eröffnet in Nürnberg eine Zweigstelle des Deutschen Museums mit Exponaten, die sich überwiegend auf zukunftsorientieren Themen fokussieren. Die IBM hat sich angeboten ein Exponat bereitzustellen, das ein neuronales Netz spielerisch und anschaulich erklären soll. Bei der Zielgruppe handelt es sich um Kinder zwischen dem 7. und 14. Lebensjahr. Es wurde

[1] Vgl. Elverson 2018, S. 1
[2] Wischmann/Rohde 2019, S. 8
[3] Google Trends 2020

entschieden, das biologische, neuronale Netz des Gehirns mit Hilfe eines künstlichen, neuronalen Netzwerks (KNN), welches nach einem ähnlichen Prinzip funktioniert, zu veranschaulichen. In dieser Arbeit wird ein solches KNN nach spezifischen Anforderungen optimiert.

1.2 Problemdefinition und Problemabgrenzung

Vereinfacht sieht das Exponat im Deutschen Museum wie folgt aus: Eine Kamera ist über einem Touchscreen Display montiert. Der Museumsbesucher hat die Möglichkeit einen bereitgestellten Gegenstand auf das Display zu stellen. Das KNN ist auf diesen Gegenstand bereits trainiert. Wird ein solcher Gegenstand auf das Display platziert, so wird dieser durch die Kamera erfasst. Die Aufgabe des KNN ist es nun diesen Gegenstand einem zuvor definierten Objekt zuzuordnen. Eine Anforderung an das KNN ist die Genauigkeit. So ist es relevant, dass die Objekte zuverlässig erkannt und klassifiziert werden. Neben der Genauigkeit liegt eine weitere Herausforderung darin, die Objektzuordnung in Echtzeit vorzunehmen, sodass keine Latenz für den Museumsbesucher auf dem Touchscreen erkennbar ist. Die Schnelligkeit und Genauigkeit dieser Objekterkennung ist abhängig von verschiedenen Hyperparametern im Quelltext, die beim Prozess des Trainings definiert werden, sowie der Leistungsfähigkeit des Edge Computers. Die Hyperparameter werden im weiteren Verlauf der Arbeit untersucht und optimiert.

1.3 Zielsetzung

Ziel dieser Arbeit ist es die performanteste Kombination verschiedener Hyperparameter mittels unterschiedlicher Simulationen zu ermitteln. Damit ein KNN als performant bezeichnet werden darf, muss dieses möglichst schnell und genau Objekte erkennen können. Die Optimierung von Hyperparametern ist ein komplexes Thema und bedarf einiger Grundlagen, die in Kapitel 2 erklärt werden. Zunächst wird allgemein auf das Themengebiet der KI eingegangen. Daraufhin wird die Funktionsweise der Objekterkennung erklärt. Zuletzt wird in Kapitel 2.4 Wissen über den Trainings- und Inferenzprozess erarbeitet, das eine wichtige Voraussetzung für das spätere Tuning der Parameter in Kapitel 4 darstellt. Bevor die Hyperparameter in Kapitel 4 optimiert werden, wird auf eine beispielhafte Implementierung eines KNN für Objekterkennung im Deutschen Museum Nürnberg in Kapitel 3 eingegangen. Anhand der gewonnenen Erkenntnisse kann eine Abwägung der performantesten Parametereinstellungen in Kapitel 4 vorgenommen werden. Dies geschieht in Anbetracht der Ergebnisse geeigneter Simulationen. Zuletzt wird in Kapitel 5 eine kurze Zusammenfassung der Ergebnisse erstellt. Diese Arbeit wird mit der Darstellung verschiedener Verbesserungsmöglichkeiten und einem Ausblick für die Zukunft abgeschlossen.

2 Theoretische Grundlagen

2.1 Künstliche Intelligenz

Das Thema der KI ist in den letzten Jahren aktueller denn je, dennoch liegen die Wurzeln dieser Technik bereits im Jahr 1936[4]. Alan Turing, häufig auch als „father of AI“[5] bezeichnet, entwarf die erste Literatur zu diesem Thema, die im späteren Verlauf historischen Wert erlangte. Zudem entwickelte er den bekannten „Turing Test“. Dieser galt als Lösung für das Problem der Messbarkeit der Intelligenz eines Computers. Der Test besteht aus zwei Personen und dem zu messenden Computer. Ein Prüfer fragt den Computer und die Personen offene Fragen und muss dann entscheiden, welche der beiden Antworten menschlich ist, ohne vorher zu wissen, ob die gegebene Antwort von dem Computer oder von der Person stammt. Ist der Prüfer nicht in der Lage die Antworten korrekt zuzuordnen, so gilt der Computer als intelligent.[6]

Aufgrund rapider Veränderungen im Bereich der Computertechnik zwischen 1956 und 1973 gewann die KI immer mehr an Bedeutung[7]. Das Interesse ging allerdings größtenteils von staatlichen Organisationen wie der „Advanced Research Projects Agency“ aus, die als Reaktion auf den russischen Sputnik Satelliten von der USA gegründet wurde. IBM war die einzige Firma zu der Zeit, die im privaten Sektor Interesse für KI zeigte. Mitte der 1950er Jahre zog sich jedoch auch die IBM aus dem Sektor zurück, da erste Probleme und Kritik an KI geäußert wurden. Die Bevölkerung äußerte sich gegenüber KI und damit auch der IBM kritisch, aufgrund der Gefahr, dass diese Technologie Arbeitsplätze ersetzen könnte. Eine Gefahr die noch heute aktuell ist.[8]

Grundlegend für das Wiederbeleben und Entwicklung von KI war die Verbesserung der Hardware. Frühe Forschungen wurden durch die mangelnde Leistung der Computer gebremst. Speziell die Weiterentwicklung der *Graphics Processing Unit* (GPU) ist ausschlaggebend im Bereich der KI. GPU ermöglichen eine verbesserte Parallelisierbarkeit von wenig rechenintensiven Aufgaben. Dies stellt einen bedeutsamen Vorteil gegenüber *Central Processing Units* (CPUs) dar. Ein Modell, das auf einer GPU in ein bis zwei Tage verarbeitet wird, kann auf einer CPU Wochen oder Monate dauern.[9]

---

[4] Vgl. Taulli 2019, S. 13
[5] Vgl. Taulli 2019, S. 13
[6] Vgl. Taulli 2019, S. 13
[7] Vgl. Taulli 2019, S. 19
[8] Vgl. Taulli 2019, S. 19
[9] Vgl. Taulli 2019, S. 25-27

Ein weiterer Anlass für den Fortschritt der KI ist die immense Verfügbarkeit von Daten auf öffentlich zugänglichen Datenbanken wie ImageNet oder Sports1M[10]. Zudem wurde der Zugang zu KI durch benutzerfreundliche Frameworks wie Caffe, TensorFlow oder MxNet erleichtert[11].

Eine eindeutige, allumfassende Definition von KI aufzustellen ist nahezu unmöglich. Verwandte Wissenschaften wie die Biologie, Psychologie, etc. scheitern ebenfalls an dieser Herausforderung[12]. Eine mögliche Definition der KI ist „der Versuch ein System zu entwickeln, das eigenständig komplexe Probleme bearbeiten kann“[13]. Die meisten Definitionen können jedoch unter den folgenden vier Definitionen zusammengefasst werden: „systems that think like humans“, „systems that act like humans”, “systems that think rationally”, “systems that act rationally”[14]. Folglich kann der Begriff KI einem Computer immer dann zugeordnet werden, wenn dieser kognitive Funktionen nachahmen kann, die sonst nur Menschen zugeordnet werden. Darunter fallen Fähigkeiten wie das Lernen und das Lösen von Problemen[15]. Beispiele für KI in der heutigen Zeit sind das autonome Fahren, das Verständnis eines Computers von menschlicher Sprache oder ein computergesteuerter Gegner in Strategiespielen[16].

Bevor genauer auf die einzelnen Teilbereiche von KI eingegangen wird, lässt sich diese zunächst in die zwei Bereiche *Weak/Narrow AI* und *Strong/General AI* klassifizieren. *Weak AI* kann spezielle Probleme lösen, solange diese auf einen bestimmten Bereich begrenzt sind. Hierunter fallen beispielsweise der Sprachassistent Siri oder Alexa. Beide KI sind nicht in der Lage ihre Funktionalität in unvorhergesehenen Bereichen auszuüben[17]. Im Klartext bedeutet das, dass die KI auf jeden Befehl vom Nutzer vorher trainiert sein muss. Die KI versteht die Schlüsselwörter in der Frage und kann dementsprechend eine Antwort geben. Der Computer erscheint somit als intelligent, obwohl die Applikation nicht aktiv denkt, sondern lediglich vorherig trainierten Aktionen folgt[18].

Im Gegensatz zu *Weak AI,* sind *Strong AI* nicht auf einen bestimmten Aufgabenbereich limitiert, sondern können Aufgaben in verschiedenen Themenbereichen lösen. Je mehr eine KI sich Fähigkeiten wie Intelligenz, Emotionen oder die Anwendung von themenübergreifendem Wissen aneignet, desto stärker ist diese KI[19]. Der Hauptgedanke hinter *Strong AI* ist die vollständige, kognitive Repräsentation des menschlichen Gehirns. Computer mit einer *Strong AI*

[10] Vgl. u.a. Samek/Wiegand 2017, S. 1
[11] Vgl. u.a. Samek/Wiegand 2017, S. 1
[12] Vgl. Wittpahl 2019, S. 21
[13] Wittpahl 2019, S. 21
[14] u.a Kok/Boers 2010, S. 2
[15] Vgl. Ongsulee 2017, S. 1
[16] Vgl. Ongsulee 2017, S. 1
[17] Vgl. Wischmann/Rohde 2019, S. 100
[18] Vgl. Borana 2016, S. 1
[19] Vgl. Walch 2019, S. 1

als Basis würden somit denselben Intelligenzgrad wie Menschen besitzen und folglich Arbeit auf menschliche Weise verrichten können[20]. Daraus lässt sich die bereits angesprochene Gefahr ableiten, dass Computer den Menschen in ihrer Arbeit ersetzen und somit für Arbeitslosigkeit in vielen Brachen sorgen könnten. Zu dem jetzigen Zeitpunkt ist es der Technik jedoch nicht gelungen eine *Strong AI* zu entwickeln. Derzeitige KI basieren auf *Weak AI*. Ob und wann die Forschung in der Lage ist eine *Strong AI* zu programmieren, ist ungewiss[21].

KI umfasst ein weites Gebiet und besteht aus mehreren Teilgebieten, eines davon ist *Machine Learning* (ML). Nach Munoz ermöglicht ML Computern „the ability to learn without being explicitly programmed"[22]. Dies geschieht durch die Erstellung von Algorithmen, die aus Daten lernen und Vorhersagen über alternative Daten treffen können. Ein einfaches Beispiel für den Einsatz von ML ist das Filtern von Emails nach Spam-Emails[23]. Anhand der vorherigen Emails lernt der Algorithmus, welche Emails in den Spam Ordner gehören. ML wird aufgeteilt in die Bereiche *supervised learning* und *unsupervised learning*[24]. Der Hauptunterschied besteht darin, dass im *supervised learning* Algorithmen mit bereits gelabelten, also markierten Daten trainiert werden. Das bedeutet, dass der korrekte Output dem Algorithmus bereits vorgegeben ist. Der Algorithmus lernt, indem er seinen prognostizierten Output mit dem korrekten Output vergleicht und sich dementsprechend anpasst. Im Ansatz des *unsupervised learning* erhält der Algorithmus einen Datensatz, bei welchem der korrekte Output nicht vorgegeben ist. Der Algorithmus muss daher selbst Strukturen innerhalb der Daten finden, um Vorhersagen treffen zu können. Im Bereich des Marketings kann dies angewandt werden, um gleichartige Segmente von Kunden ausfindig zu machen, damit diese in Vermarktungskampagnen gezielter angesprochen werden können[25].

Der letzte Bereich, *Deep Learning* (DL), ist ein Teilgebiet des ML und somit auch von KI. Der Unterschied zu ML besteht darin, dass ML nahezu ausschließlich mit strukturierten Datensätzen arbeiten kann, während DL Netzwerke auch mit verschiedenen Darstellungen von Daten arbeiten können[26]. Mustererkennungen sind somit einfacher zu entwickeln. Während im konventionellen ML viel Expertise für eine Mustererkennung gefragt ist, ist ein DL Netzwerk in der Lage, aus Rohdaten automatisch Merkmale zu extrahieren, die für die Klassifikation und Erkennung notwendig sind[27].

[20] Vgl. Borana 2016, S. 1
[21] Vgl. Borana 2016, S. 1
[22] Munoz 2015, S. 1
[23] Vgl. Mallampati 2018, S. 1
[24] Vgl. Wittpahl 2018, S. 24
[25] Vgl. Ongsulee 2017, S. 1f.
[26] Vgl. u.a Lecun/Bengio 2015, S. 1
[27] Vgl. u.a Lecun/Bengio 2015, S. 1

2.2 Künstliches Neuronales Netz

KNN bilden den Versuch ab, die Arbeitsweise des menschlichen Gehirns nachzuahmen. Um KNN zu verstehen, muss vorerst die Funktionsweise des menschlichen Gehirns betrachtet werden. Dort werden Reize, wie z.B. das Sehen, Hören oder Fühlen an Rezeptoren aufgenommen. Die Rezeptoren leiten diesen Reiz zunächst an das Gehirn weiter. Im Gehirn wird dieser Reiz von den Neuronen verarbeitet.[28] Kinnebrock definiert Neuronen als „Zellen, die untereinander auf elektrochemischen Wege Signale austauschen und sich gegenseitig erregen können“[29]. Insgesamt besitzt der Mensch 10^{10} Neuronen[30]. Ähnlich wie bei einem Prozessor im Computer, besitzt jedes einzelne Neuron eine Ein- und Ausgabe. Der Sehprozess im vorherigen Beispiel wird als elektrochemischer Reiz an die Eingabe des Neurons geleitet. Gelangen genügend Reize an dasselbe Neuron, so wird ein neuronspezifischer Schwellwert überschritten und das Neuron sendet über die Ausgabe ein Signal an nachgeschaltete Neuronen[31]. Da jedes Neuron in etwa 10.000 Verbindungen mit anderen Neuronen eingeht, liegt die Gesamtzahl der Verbindungen bei 10^{14}. Die Neuronen, gemeinsam mit ihren Verbindungen untereinander, bilden ein sogenanntes neuronales Netz. Verbindungen zwischen Neuronen entstehen erst durch einen Lernprozess. Folglich sind die Neuronen im Gehirn eines Menschen nach seiner Geburt nicht vernetzt[32]. Gleichermaßen werden Verbindungen getrennt, sollten diese lange Zeit nicht verwendet werden. Dieses Phänomen beschreibt das menschliche Vergessen[33].

Darüber hinaus ist zwischen der Eingabe und dem Neuron im Gehirn eine Zelle (Synapse) geschaltet, die den Potentialwert der Eingabe verstärken oder abschwächen kann. Somit kann kontrolliert werden, ob der Schwellwert des Neurons leichter oder schwieriger überschritten werden soll[34]. Synapsen haben eine hohe Relevanz im Lernprozess, da der Potentialwert dieser anpassungsfähig ist. Das bedeutet, dass sich der Potentialwert der Synapsen durch das Erlernen neuer Fähigkeiten verändern kann. Aufgrund der Lernfähigkeit eines neuronalen Netzes ist dieses imstande nach der Trainingsphase zu generalisieren. Generalisieren bedeutet, dass das neuronale Netz „Antworten auf Probleme […] finden [kann], mit denen es [vorher] noch nicht konfrontiert worden ist“.[35]

---

[28] Vgl. Kinnebrock 1994, S. 11
[29] Kinnebrock 1994, S. 13
[30] Vgl. Kinnebrock 1994, S. 13
[31] Vgl. Kinnebrock 1994, S. 13
[32] Vgl. Kinnebrock 1994, S. 13
[33] Vgl. Kinnebrock 1994, S. 13
[34] Vgl. Kinnebrock 1994, S. 15
[35] u.a. Traeger/Eberhart 2003a, S. 1132

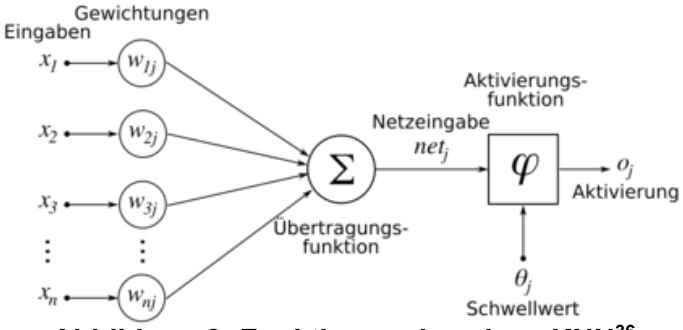

Abbildung 2: Funktionsweise eines KNN[36]

In der Informatik kann ein neuronales Netz als KNN modelliert werden. Ein KNN ähnelt sowohl im Aufbau als auch in seiner Funktionsweise dem biologischen neuronalen Netz. KNN sind gleichermaßen aus einer Vielzahl von Neuronen aufgebaut. Jedes Neuron besitzt eine Aktivierungsfunktion, die festlegt ab welchem Schwellwert das Neuron aktiviert wird, sowie verschiedene gewichtete Eingänge, die das Neuron mit den vorherigen Neuronen im KNN verbindet. Die ankommenden Impulse der vorgeschalteten Neuronen werden im Neuron aufsummiert. In Abbildung 2 ist zu erkennen, dass verschiedene Eingänge unterschiedliche Gewichte tragen. Folglich tragen unterschiedliche vorgeschaltete Neuronen verschieden stark zu der Erfüllung oder Nichterfüllung des Schwellwerts des Neurons bei[37]. In den Gewichtungen an den Eingängen des Neurons ist somit das Wissen des KNN gespeichert[38]. Wird dieses ordnungsgemäß trainiert, so ändern sich lediglich die Gewichte an den Eingängen der Neuronen, sodass das KNN nach dem Training die gewünschte Musterklasse zuordnen kann[39]. Die Aktivierungsfunktion des Neurons wird meist durch eine Sigmoid- oder Tanges-hyperbolicus-Funktion beschrieben[40]. Diese Funktionen verhindern eine Aktivierung des Neurons durch einen zu geringen Impuls. Die Aktivierungsfunktion legt also fest, „wie sich aus einem Aktivierungszustand zum Zeitpunkt t ein Aktivierungszustand t+1 berechnen läßt“[41].

[36] Lüdi 2007
[37] Vgl. u.a. Traeger/Eberhart 2003b, S. 1056
[38] Vgl. Universtität Ulm o.J., S. 2
[39] Vgl. Strecker 1997, S. 9
[40] Vgl. Tawil 1999, S. 2
[41] Tawil 1999, S. 2

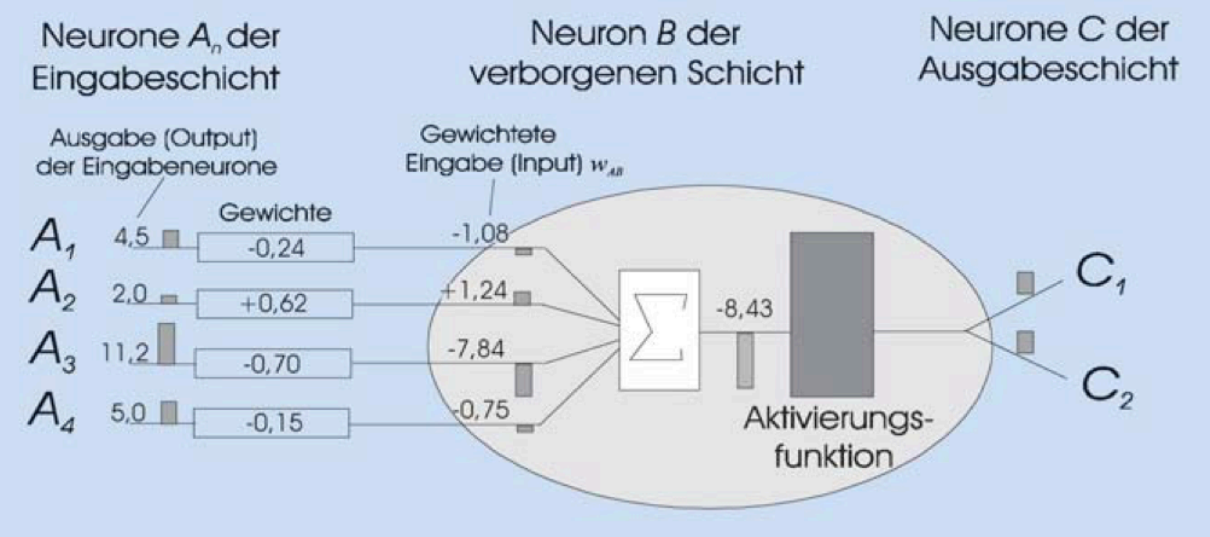

Abbildung 3: Aufbau eines KNN[42]

Der Abbildung 3 sind die bereits angesprochenen Eingabegewichte, sowie die Ausgabe vorheriger Neuronen und die Aktivierungsfunktion zu entnehmen. Der Einfluss der Eingabegewichte auf die konkrete Eingabe wird hier anhand beispielhafter Zahlen verdeutlicht. Zudem ist bereits eine Schichtenstruktur zu erkennen, auf die im Folgenden genauer eingegangen wird.

Ein KNN besteht in der Regel aus drei Schichten; der Eingabeschicht (*Input Layer*), den verborgenen Schichten (*Hidden layers*) und der Ausgabeschicht (*Output layer*)[43]. Während es von der Ein- und Ausgabeschicht jeweils nur eine Schicht gibt, kann es beliebig viele verborgene Schichten geben. Ein KNN mit mehreren Schichten wird als Deep Neural Network bezeichnet[44]. Die Einteilung in verschiedene Schichten grenzt ein KNN in der Informatik von denen in der Biologie ab. Die Eingabeschicht nimmt die externen, zu verarbeitenden Reize bzw. Variablen auf. Dabei gilt, dass die Anzahl der Eingabeneuronen meist der Anzahl der Eingabevariablen gleicht[45]. Bei einem 16x16 binärem schwarz-weiß Bild ergeben sich folglich 256 Eingabeneuronen. In der Regel sind die Werte der Eingabe- und Ausgabesignale binär (0, 1), reel (+, -) oder bipolar (-1, +1)[46]. Im genannten Beispiel also schwarz (0) oder weiss (1). Strecker führt als weiteres Beispiel den Test der Kreditwürdigkeit an, bei dem die Bonitätsklasse als Eingabevariable gilt. Entweder man besitzt eine gute Bonität (1) oder eine schlechte Bonität (0)[47]. Die in der Eingabeschicht aufgenommenen Informationen werden an mindestens eine

[42] u.a. Traeger/Eberhart 2003
[43] Vgl. u.a. Traeger/Eberhart 2003b, S. 1056
[44] Vgl. Li/Zhang 2017, S. 1
[45] Vgl. Strecker 1997, S. 11
[46] Vgl. Strecker 1997, S. 11
[47] Vgl. Strecker 1997, S. 11

der verborgenen Schichten weitergeleitet und netzintern verarbeitet. Mithilfe der Ausgabeschicht kann das Ergebnis der Informationsverarbeitung des Netzes ausgelesen werden[48]. Fehler in der Ausgabeschicht werden mittels des *Backpropagation* Algorithmus an die vorherigen Schichten weitergegeben. Dies funktioniert indem die Ausgabewerte des KNN mit den objektiv richtigen Ausgabewerten verglichen werden und die Abweichung berechnet wird. Um die Abweichung so gering wie möglich zu halten, werden die Gewichte innerhalb des KNN entsprechend angepasst.[49]

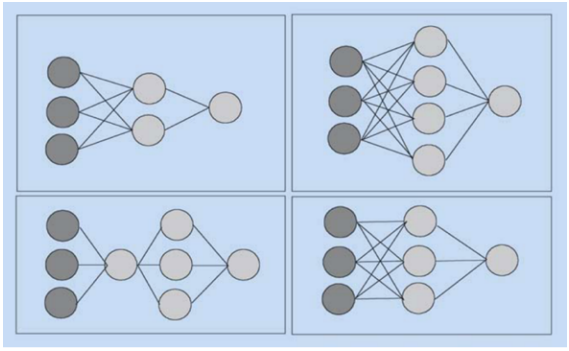

Abbildung 4: Beispielhafte Strukturen eines KNN[50]

Jedes Neuron einer Schicht ist mit jedem Neuron der darauffolgenden Schicht verbunden, wie es auch in Abbildung 4 zu sehen ist. Ebenfalls können der Abbildung vereinfachte, exemplarische Aufbauten von KNN entnommen werden. Diese unterscheiden sich häufig stark in der Anzahl von Neuronen pro Schicht oder in der Anzahl an Schichten.

2.3 Objekterkennung

Die Objekterkennung ist ein Anwendungsbereich von KI, welcher erst in den letzten Jahren an Relevanz erlangt hat. Sie ermöglicht z.B. das Erkennen von Autos, Fußgängern oder sonstigen Hindernissen im autonomen Fahren[51]. Andere Anwendungszwecke beinhalten unter anderem die automatische Identifizierung von Tumoren oder Anomalien in der Medizin, sowie die Gesichtserkennung in der Biometrik, welche heutzutage bereits Standard in vielen Smartphones ist[52].

Die Objekterkennung besteht im Wesentlichen aus drei Schritten: Der Bildklassifikation, der Objektlokalisation und der semantischen Segmentierung[53]. Bevor auf die einzelnen Bereiche

[48] Vgl. Tawil 1999, S. 2
[49] Vgl. Kirste/Schürholz 2018, S. 32
[50] u.a. Traeger/Eberhart 2003
[51] Vgl. Süße/Rodner 2014, S. 5
[52] Vgl. Lammers/Wachenfeld 2003, S. 4
[53] Vgl. Süße/Rodner 2014, S. 589

eingegangen wird, gilt es grundsätzlich vier Hauptprobleme zu beachten, die in den verschiedenen Schritten der Objekterkennung auftreten können. Zunächst können sich Objekte innerhalb des Bildes gegenseitig verdecken[54]. Zu sehen ist dies in Abbildung 5. Dort wird der Elefant von einer Hand verdeckt. Während der Elefant für Menschen unschwer zu erkennen ist, sind spezifische Merkmale des Elenfanten verdeckt, welche von hoher Bedeutung für die Erkennung in einem KNN sein können.

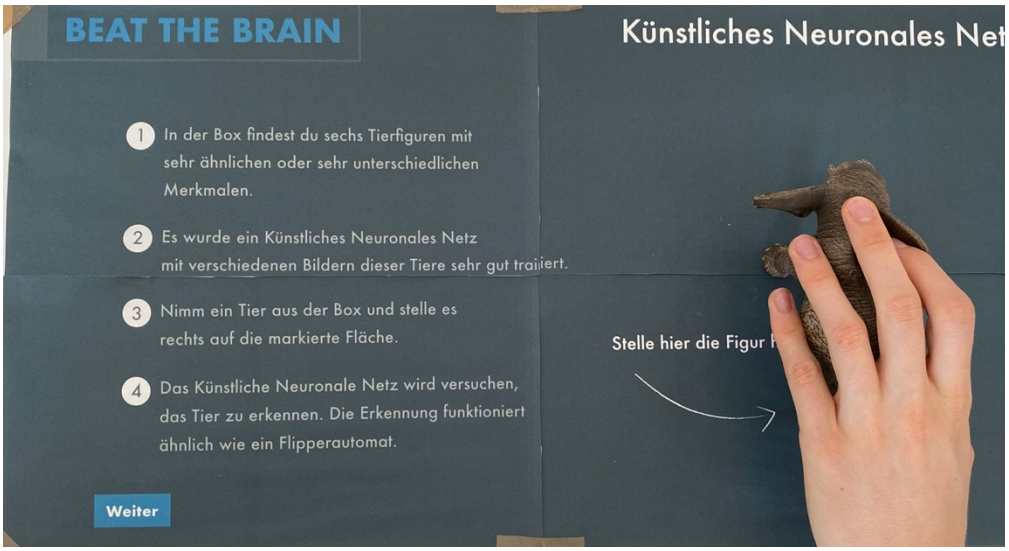

Abbildung 5: Verdeckungsproblem in der Objekterkennung[55]

Ein weiteres Problem ist der Hintergrund[56]. Die KI kann unter Umständen Objekte im Hintergrund, die nicht erkannt werden sollten, fälschlicherweise erkennen. Das dritte Problem resultiert aus dem Intraklassenabstand[57]. Spezifische Bildermerkmale in Bildern der gleichen Objekte, wie eine unterschiedliche Rotation, Skalierung, farbliche Gestaltung oder andere Abhängigkeiten, lassen die Objekte häufig sehr anders für die KI erscheinen. Ein Elefant bei starker Beleuchtung kann unter Umständen für eine KI anders aussehen, als ein Elefant bei schwacher Beleuchtung. Zuletzt gilt es die Interklassendistanz zu beachten. Bestimmte Objekte unterschiedlicher Klassen können sich optisch ähneln, sodass es schwierig ist für die KI diese zu unterscheiden.[58]

Eine Vorbereitung, die getroffen werden muss, bevor Objekte im Bild klassifiziert werden können, ist die Segmentierung des Bildes. Diese beiden Bereiche lassen sich allerdings nicht eindeutig voneinander abgrenzen, da eine Klassifizierung von Objekten unbewusst bereits in der

[54] Vgl. Süße/Rodner 2015, S. 590
[55] Eigene Darstellung
[56] Vgl. Süße/Rodner 2015, S. 591
[57] Vgl. Süße/Rodner 2015, S. 591
[58] Vgl. Süße/Rodner 2015, S. 591

Segmentierung stattfindet[59]. Aufgrund der Vielfalt von Segmentierungsmethoden wird im Folgenden ausschließlich auf die am meisten verbreitete Methode, dem Schwellenwertverfahren (*Thresholding*), eingegangen. Diese Methode ermöglicht die Abgrenzung des Objektes von dessen Hintergrund mit Hilfe eines Schwellenwertes (*Threshold*)[60]. Pixel, die einen geringeren Wert als den Schwellwert besitzen, werden somit dem Hintergrund zugeordnet. Dabei wird erneut zwischen dem globalen und lokalen Schwellenwertverfahren unterschieden[61]. Während im globalen Schwellenwertverfahren alle Pixel des Bildes mit dem Schwellenwert verglichen werden, wird im lokalen Schwellenwertverfahren das originale Bild zunächst in kleinere Unterbilder zerlegt. Für jedes dieser Unterbilder wird ein spezifischer Schwellwert gebildet[62]. Das lokale Schwellenwertverfahren wird besonders dann verwendet, wenn starke Helligkeitsunterschiede innerhalb des Bildes erkennbar sind, da Ergebnisse des globalen Schwellenwertverfahrens mit einer solchen Herangehensweise häufig unbrauchbar sind. Allerdings muss berücksichtigt werden, dass das lokale Schwellenwertverfahren durch die mehrmalige Ermittlung der Schwellwerte deutlich mehr Zeit in Anspruch nimmt.[63] Darüber hinaus wird erneut zwischen verschiedenen Methoden zur Bestimmung des Schwellenwertes unterschieden. Dieser kann entweder manuell durch die Eingabe des Nutzers bestimmt werden, oder automatisch mit Hilfe eines Algorithmus.[64]Das Resultat beider Schwellenwertverfahren ist ein Binärbild. Ein beispielhaftes Ergebnis des Schwellenwertverfahrens ist in der Abbildung 6 zu sehen.

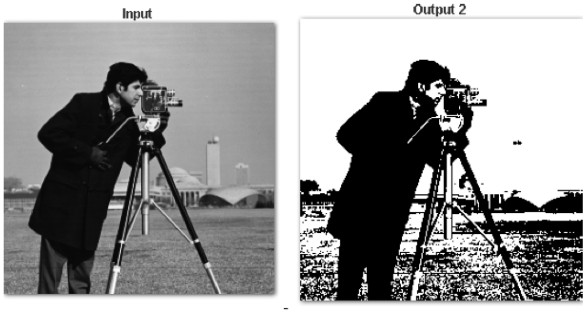

Abbildung 6: Ergebnis Schwellenwertverfahren[65]

In viele Fällen ist eine Segmentierung jedoch nicht ausreichend, um bestimmte Bildinhalte zu extrahieren. Daher ist es sinnvoll bestimmte Merkmale gezielt im Bild zu erkennen. Dieser Teilschritt nennt sich Merkmalsextraktion und ist Teil der Objektlokalisierung[66]. Die Kantendetektion ist beispielsweise ein Verfahren der Merkmalsextraktion. Eine Kante dient häufig als

[59] Vgl. Süße/Rodner 2015, S. 211
[60] Vgl. Erhardt 2017, S. 7
[61] Vgl. Erhardt 2017, S. 7
[62] Vgl. Erhardt 2017, S. 7
[63] Vgl. Erhardt 2017, S. 8
[64] Vgl. Erhardt 2017, S. 7
[65] Peterwitz 2006
[66] Vgl. Peterwitz 2006, S. 7

Signal für den Abschluss eines Objektes[67], sodass es sinnvoll erscheint, diese im Bild zu suchen. In der Praxis geschieht dies durch einen Algorithmus, der die Kanten im Anschluss als Schwarzweiß Bild wiedergibt. Andere mögliche Merkmalsextraktionsverfahren sind Eckendetektoren, wie der Moravec's Corner Detector.[68]

Ist nun mittels Segmentierung und Merkmalsextraktion ein Objektbereich identifiziert, gilt es diesen einer Klasse zuzuordnen. Dies geschieht im Schritt der Bildklassifikation. Die Bildklassifikation hat die Zuweisung von Bildern zu Objektkategorien als Aufgabe[69]. Peterwitz beschreibt die Klassifizierung als „einen Vorgang oder eine Methode zur Einteilung von Objekten in Klassen oder Kategorien"[70]. Es gibt verschiedene Herangehensweisen für die Bildklassifikation, jedoch wird in den meisten Fällen vorausgesetzt, dass bereits beschriftete Bilder für jede Objektkategorie vorliegen. In diesen Bildern ist, mittels Begrenzungsrahmen (*Bounding boxes*), vorgemerkt welche Objektkategorie sich wo in dem Bild befindet. Für die Klassifizierung werden die gesammelten Informationen über den zu klassifizierenden Objektbereich mit den Eigenschaften der bereits beschrifteten Bilder verglichen und anhand dessen die entsprechende Klasse bestimmt[71]. Peterwitz führt hierfür das Beispiel eines Hundes an[72]. In den beschrifteten Bildern tragen Hunde häufig die Eigenschaft klein und vierbeinig, während dagegen Menschen die Eigenschaft zweibeinig und groß besitzen. Sollte der zu identifizierende Bildbereich die Eigenschaft zweibeinig tragen, so ist es unwahrscheinlich, dass sich im Objektbereich ein Hund befindet und das KNN würde diesen Bereich als Mensch klassifizieren.

2.4 Trainings- und Inferenzprozess

Bei der Verwendung eines KNN wird unterschieden zwischen dem Trainings- und Inferenzprozess[73]. Im Trainingsprozess findet das eigentliche Trainieren, oder auch Anlernen, des vorerst untrainierten KNN mit einem vorgefertigtem Datensatz statt.[74] Im Inferenzprozess wird keine Anpassung innerhalb des KNN vorgenommen, sondern eine Mustererkennung durchgeführt[75]. Dabei wird das antrainierte Wissen des KNN verwendet um ein Ergebnis zu inferieren[76]. Für den Inferenzprozess werden immer Daten verwendet, mit denen das KNN nicht trainiert wurde. Der Inferenzprozess findet zeitlich immer nach dem Trainingsprozess statt, da ein trainiertes KNN benötigt wird. Die Genauigkeit und die Schnelligkeit der Prognosen im

[67] Vgl. Peterwitz 2006, S. 7
[68] Vgl. Peterwitz 2006, S. 10
[69] Vgl. Süße/Rodner 2015, S. 591
[70] Vgl. Peterwitz 2006, S. 18
[71] Vgl. Peterwitz 2006, S. 18
[72] Vgl. Peterwitz 2006, S. 18
[73] Vgl. Kirste/Schürholz 2018, S.32
[74] Vgl. Kirste/Schürholz 2018, S.32
[75] Vgl. Kirste/Schürholz 2018, S. 32
[76] Vgl. MITX PC o.J., S.1

Inferenzprozess sind unter anderem abhängig von sogenannten Hyperparametern, die sich im Quelltext des Trainings ausfindig machen lassen. Diese Hyperparameter sollten optimiert werden, sodass im Inferenzprozess eine möglichst hohe Genauigkeit der Ergebnisse und eine schnelle Inferenzgeschwindigkeit erzielt werden kann. Aufgrund der großen Anzahl an existierenden Hyperparametern werden im nachfolgenden Verlauf ausschließlich die relevantesten betrachtet. Der Vorgang der Hyperparameteroptimisierung ist allerdings noch nicht vollständig erforscht. Zu dem derzeitigen Zeitpunkt gibt es keine eindeutige, zielführende Methode die Parameter zu optimieren[77].

2.4.1 Optimizer & Learning rate

Die Hauptaufgabe der Optimierung ist die Minimierung der loss Funktion durch die optimale Einstellung der entsprechenden Parameter[78]. Die loss Funktion stellt in diesem Kontext die Abweichung der prognostizierten Ergebnisse des KNN von den objektiv richtigen Ergebnissen dar[79]. Sie beinhaltet unter anderem die verschiedenen Gewichtungen an den Neuroneneingängen. Die loss Funktion beschreibt also, wie gut eine bestimmte Kombination von Hyperparametern funktioniert[80]. Im Idealfall ist die loss Funktion = 0. Dies ist allerdings unter echten Bedingungen unrealistisch, sodass nur eine asymptotische Annäherung an das Minimum angestrebt werden kann.

Die loss Funktion wird durch die Verwendung von sogenannten *gradient descent algorithms* minimiert[81]. Ein Großteil der aktuellen Frameworks für DL, wie z.B. MxNet, besitzen bereits eine Implementation von verschiedenen *gradient descent algorithms,* um die loss Funktion zu optimieren[82]. Der *gradient descent* ist eine der gebräuchlichsten Algorithmen für die Optimierung von KNN[83]. Die Funktionsweise eines *gradient descent algorithm* besteht darin, die Gewichte des KNN ständig zu aktualisieren, um die loss Funktion zu minimieren[84]. Jedes Gewicht im KNN wird in entgegengesetzter Richtung zur Steigung verändert. Wie stark die Gewichte geändert werden, hängt von einem weiteren Hyperparameter ab, nämlich der *learning rate*, auf die im späteren Verlauf genauer eingegangen wird[85]. Im Allgemeinen beschreibt das

[77] Vgl. Smith 2018, S. 1
[78] Vgl. GitHubIO o.J., S.1
[79] Vgl. GitHubIO o.J., S.1
[80] Vgl. GitHubIO o.J., S.1
[81] Vgl. MxNet o.J., S.1
[82] Vgl. Ruder 2016, S. 1
[83] Vgl. Ruder 2016, S. 1
[84] Vgl. u.a. Yamashita/Nishio 2018, S. 617
[85] Vgl. u.a. Yamashita/Nishio 2018, S. 617

Framework MxNet die Funktion des *gradient descent algorithms* als ein „general purpose algorithm for minimzing a function using information from the gradient of the function with respect to its parameters"[86].

In Frameworks wie MxNet gibt es verschiedene Implementationen von *gradient descent algorithms* mit jeweils spezifischen Stärken und Schwächen. Die gängigste Implementierung wird im Folgenden beschrieben. *Gradient descent algorithms* benötigen weitere Hyperparameter als Parameterangabe im Code, die anhand des ausgewählten Algorithmus erklärt werden. Der im Folgenden behandelte *gradient descent algorithm* lautet *stochastic gradient descent* (SGD). Da die loss Funktion eine Funktion ist, die alle Gewichtungen und Trainingsdaten des KNN miteinbezieht, muss deren Gradient unter der Verwendung aller Trainingsdaten berechnet werden[87]. Während im sogenannten *total gradient descent algorithm* die Gewichtungen im KNN einmalig nach jedem Durchlauf des Trainingsdatensatzes optimiert werden, geschieht die Anpassung der Gewichtungen im SGD Algorithmus sobald der jeweilige einzelne Bestandteil des Trainingsdatensatzes vom KNN prozessiert wurde. Wie stark die Gewichtung verändert wird hängt im SGD Algorithmus vom Gradienten der loss Funktion ab[88]. Hierbei wird allerdings nicht der Gradient der loss Funktion des gesamten Datensatzes genommen, sondern der Gradient der loss Funktion spezifisch für den einzelnen Bestandteil des Datensatzes[89]. Laut Bottou hat dies zur Folge, dass der SGD Algorithmus somit deutlich schneller, verlässlicher und weniger anfällig für lokale Minima ist.[90]

Bei der Implementierung eines SGD Algorithmus im Code müssen weitere Parameter angegeben werden. Welche Parameter angegeben werden müssen, hängt von der Auswahl des *gradient descent algorithm* ab.

```
trainer = gluon.Trainer(
    net.collect_params(), 'sgd',
    {'learning_rate': 0.001, 'wd': 0.0005, 'momentum': 0.9})
```

Abbildung 7: Implementierung der Hyperparameter im MxNet Code[91]

In Abbildung 7 ist neben *weight decay* (in Abbildung 7 als „wd") und *momentum*, welche im späteren Verlauf erläutert werden, ebenfalls der Hyperparameter *learning rate* angegeben. Laut Yamashita ist die *learning rate* eines der wichtigsten Parameter, der vor dem Training angegeben werden muss[92]. Wie bereits erwähnt werden die Gewichtungen in einem KNN in

[86] Vgl. MxNet o.J., S.1
[87] Vgl. Bottou 1991, S. 1
[88] Vgl. Bottou 1991, S. 1
[89] Vgl. Bottou 1991, S. 1
[90] Vgl. Bottou 1991, S. 1
[91] Eigene Darstellung
[92] Vgl. u.a. Yamashita/Nishio 2018, S. 617

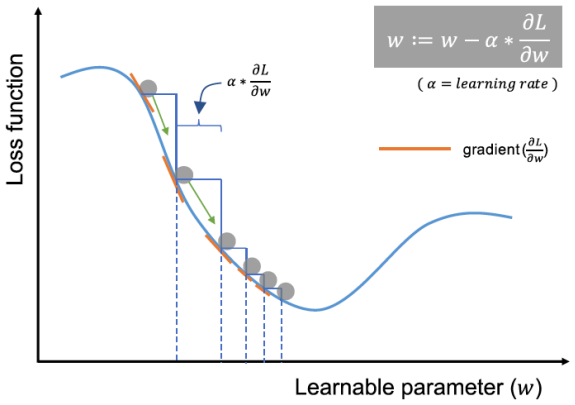

Abbildung 8: Einfluss der learning rate auf die loss Funktion[93]

die entgegengesetzte Richtung des Gradienten der loss Funktion optimiert. Der Hyperparameter *learning rate* gibt die Schrittgröße vor, also wie groß die Zunahme der Änderung in jedem Schritt ist[94].

Abbildung 8 zeigt den Verlauf einer exemplarischen loss Funktion und die Funktionsweise des *gradient descent algorithms*, wobei (w) für den *learnable parameter* steht, (a) für die *learning rate* und (L) für die loss Funktion[95]. Ziel des *gradient descent algorithms* in der Abbildung ist es, das lokale Minimum zu ermitteln, das in der Funktion deutlich erkennbar ist. Aus der Gleichung ist ebenfalls zu entnehmen, dass eine größere *learning rate* eine größere Schrittgröße zum nächsten *learnable paramter* ermöglicht ($a * \frac{L}{w}$). Die *learning rate* wird in einem Bereich zwischen 0 und 1 angesetzt.

2.4.2 Momentum

Der SGD Algorithmus wird zudem durch den Hyperparamter *momentum* erweitert.[96] Dieser wird im Code des Algorithmus als Parameter definiert. *Momentum* hilft dem Algorithmus ein genaueres Ergebnis zu ermitteln, indem der Gradient der loss Funktion durch die Miteinberechnung der vorherigen Gradientenberechnung ermittelt wird. Es kann also behauptet werden, dass SGD mit *momentum* sich die vorherigen Schritte (Berechnungen) „merkt“, sodass diese in der nächsten Iteration miteinbezogen werden können[97].

[93] u.a. Yamashita/Nishio 2018
[94] Vgl. u.a. Yamashita/Nishio 2018, S. 619
[95] Vgl. u.a. Yamashita/Nishio 2018, S. 619
[96] Vgl. Ruder 2016, S. 4
[97] Vgl. MxNet o.J., S. 1

2.4.3 Weight decay & Epochs

Weight decay verbessert die Fähigkeit der Generalisation eines KNN nach dem Training[98]. Generalisation, wie bereits oben beschrieben, ist die Fähigkeit eines KNN „Antworten auf Probleme zu finden, mit denen es [vorher] noch nicht konfrontiert worden ist“.[99] Des Weiteren wird mit der Verwendung des Parameters *weight decay* die Gefahr des *overfitting* verringert. *Overfitting* beschreibt ein Szenario, bei dem das KNN die Fähigkeit der Generalisierung aufgrund von zu intensivem Training mit den Trainingsdaten verliert[100]. Wichtig ist die Unterscheidung zwischen einem größeren Trainingsset mit unterschiedlichen Bildern und der Anzahl an Durchläufen desselben Bildes durch das KNN. Diese Anzahl der Durchläufe eines einzelnen Bildes wird bestimmt durch den Hyperparameter *epoch*. Mithilfe von Trainings- und Validationsdaten kann festgestellt werden, ob ein KNN *overfitted* ist. So wird ein KNN meist nur mit einem Großteil der Gesamtdaten trainiert, sodass die restlichen Daten (Validationsdatensatz) als Überprüfung der Genauigkeit des KNN dienen können[101]. Da das KNN nie auf die Trainingsdaten im Validationsdatensatz trainiert wurde, dienen diese ebenfalls als Test für die Generalisierungfähigkeit. Sollte das KNN gute Ergebnisse beim Testen auf den Trainingsdatensatz zeigen aber schlechte Ergebnisse im Test auf den Validationsdatensatzes, so kann man davon ausgehen, dass das KNN *overfitted* ist[102]. Werden schlechte Ergebnisse im Test beider Datensätze erzielt, so ist das KNN *underfitted* und muss intensiver mit dem Trainingsdatensatz trainiert werden. Es gilt also die richtige Balance zwischen *underfitting* und *overfitting* eines KNN zu finden. Dies ist anschaulich in der Abbildung 9 visualisiert. Die Anzahl der Iterationen (*epochs*) ist in Abhängigkeit der loss Funktion dargestellt. Im Graphen der Abbildung sind zwei Kurven zu erkennen, wobei die obere den Validationsdatensatz repräsentiert und die untere den Trainingsdatensatz. Beide Kurven haben einen ähnlichen Verlauf und verdeutlichen, dass der anzustrebende Bereich zwischen *underfitting* und *overfitting* relativ gering ist. Neben der Möglichkeit den Trainingsdatensatz mit weiteren Bildern zu erweitern, kann gegen *overfitting* mittels des *weight decay* Hyperparameters vorgegangen werden. Dieser sorgt dafür, dass die

[98] Vgl. MxNet o.J., S. 1
[99] Vgl. u.a. Traeger/Eberhart 2003a, S. 1132
[100] Vgl. u.a. Yamashita/Nishio 2018, S. 618
[101] Vgl. u.a. Yamashita/Nishio 2018, S. 618
[102] Vgl. u.a. Yamashita/Nishio 2018, S. 620

Gewichtungen im KNN klein gehalten werden, indem größere Gewichtungen mathematisch gemaßregelt werden[103].

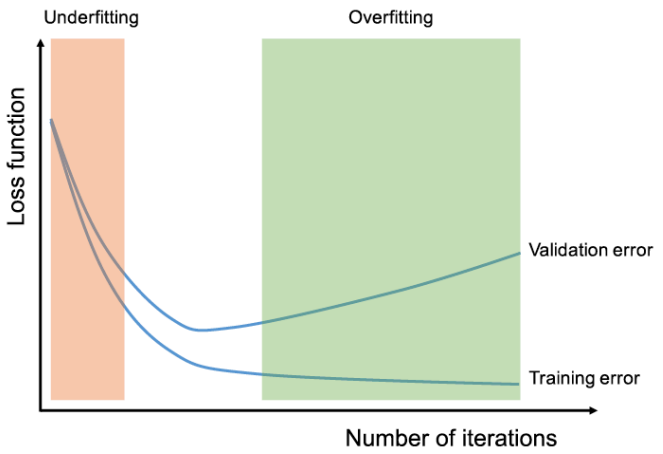

Abbildung 9: Overfitting/Underfitting[104]

[103] Vgl. MxNet o.J, S.1
[104] u.a. Yamashita/Nishio 2018

3 Implementierung eines KNN am Beispiel des Deutschen Museum Showcase

3.1 Aufbau des Exponats

Die IBM wurde Anfang 2019 vom Deutschen Museum Nürnberg mit dem Projekt beauftragt ein Exponat zu erstellen, welches das menschliche Gehirn anhand eines KNN erklärt. Dieses Exponat soll im Deutschen Museum in Nürnberg ausgestellt werden, welches Anfang 2021 eröffnet.

Der Abbildung 10 ist der Aufbau des Exponats zu entnehmen. Dargestellt ist ein 55 Zoll 4k Touchscreen Monitor, welcher senkrecht auf einem Tisch in 90cm Höhe platziert ist. Unter dem Tisch ist ein leistungsfähiger Edge Computer positioniert, an welchem eine Full-HD Kamera angeschlossen ist. Diese ist 140 cm über dem Touchscreen mittig platziert, sodass der gesamte Touchscreen über den Kamerafeed erkennbar ist.

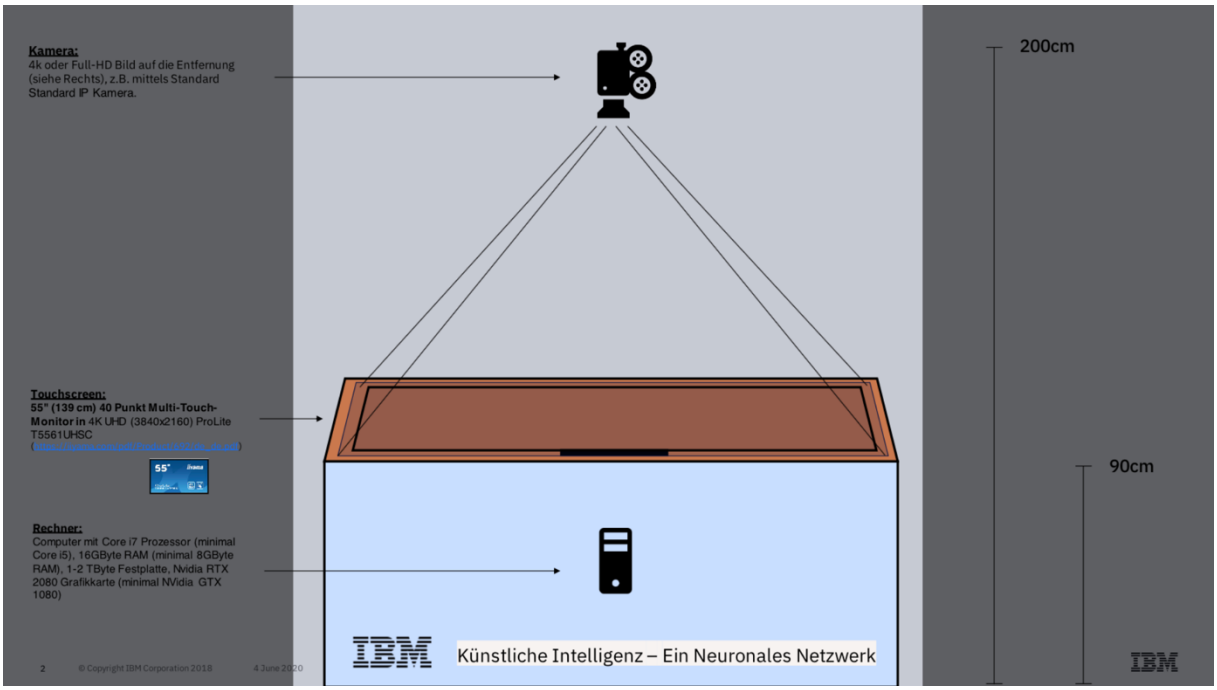

Abbildung 10: Aufbau des Exponats im Deutschen Museum Nürnberg[105]

Auf dem Computer wird ein Programm laufen, das dem Nutzer die Funktionsweise eines künstlichen neuronalen Netzes erklärt. Unter anderem wird der Nutzer gebeten eine Hartgummi Figur auf dem Display zu positionieren. Vom KNN soll erkannt werden, um welches Tier es sich handelt, sowie wo sich dieses Tier auf dem Display befindet. Auf dem Display soll ein

[105] IBM Projektdaten

Rechteck um das Tier erscheinen, sowie der Namen des Tieres und dessen Erkennungsgenauigkeit in Prozent. Neben den bereitgestellten Tieren sollen ebenfalls Alltagsgegenstände wie Uhren, Smartphones, Gelscheine, Gelbeutel und Schlüssel erkannt werden.

3.2 Technische Funktionsweise

Es wird sich im Weiteren auf den technischen Kontext der Objekterkennung der Hartgummifiguren und der Alltagsgegenstände beschränkt. Damit verschiedene KNN trainiert werden können, wurde für diese Arbeit ein Trainingsskript in Python mittels des Frameworks MxNet geschrieben. Für das Trainingsskript wird ein vortrainiertes Modell aus einer Sammlung verschiedener Netze gewählt, welches bereits auf den „Coco“ Datensatz trainiert ist. Dieser Datensatz enthält unter anderem verschiedene Fotos von Tieren. Aufgrund der Tatsache, dass das Netzwerk zusammen mit seiner *Input- und Hiddenlayers* bereits auf die Aufgabe der Objekterkennung trainiert wurde, ist eine Anpassung des Netzwerks zur Erkennung eigens definierter Klassen mit einer relativ geringen Menge an Daten möglich. Um dies zu realisieren, wird die letzte Schicht des vortrainierten Netzwerks, der *Output-Layer*, gelöscht und mit neuen Klassen befüllt.

Es wurden 700 individuelle Bilder mit einer Kamera aufgenommen. Dieser Datensatz wurde daraufhin auf 2400 Bilder erweitert, indem eine Augmentation vorgenommen wurde. Es wurden anhand der 700 Bilder 1700 weitere Bilder erstellt, die sich lediglich in der Helligkeit, dem Farbenspektrum und der Rotation unterscheiden. Während die augmentierten Bilder für das menschliche Auge nahezu wie die Originalbilder aussehen, erscheinen sie für ein KNN wie ein neues Bild. Der Datensatz von 2400 Bildern wurde in einen Trainings- und Validationsdatensatz eingeteilt. Die folgenden Simulationen nutzen 2000 Trainingsbilder und 400 Validationsbilder.

Neben dem Trainingsskript wurde zudem ein Programm geschrieben, welches einen live Webcam feed als Input entgegennimmt. In den Bildern des Webcam feeds soll das KNN die verschiedenen Hartgummifiguren und Alltagsgegenstände in Echtzeit erkennen können. Dies ist allerdings nur möglich, wenn das KNN auf performant optimiert ist, sodass die Objekte zuverlässig und schnell erkennt werden können. Die Optimierung des KNN wird im weiteren Verlauf dieser Arbeit vorgenommen.

4 Parametertuning

4.1 Hyperparametertuning im Trainingsprozess

Im Folgenden werden unterschiedliche KNN auf 2000 Bilder von Hartgummifiguren sowie Alltagsgegenständen trainiert. Die verschiedenen Klassen, auf die die KNN trainiert werden, lauten: Leopard, Giraffe, Tiger, Nashorn, Elefant, Papagei, Schlüssel, Uhr, Smartphone, Geldschein, Hand und Geldbeutel. 400 weitere Bilder stehen als Validationsdatensatz für die Überprüfung der Genauigkeit des Netzes zur Verfügung. Das Training erfordert allerdings hohe Anforderung an die GPU. Aufgrund dessen ist es sinnvoll das Training auf dem Exponatrechner zu tätigen, der zu dem derzeitigen Zeitpunkt im IBM HQ in Ehningen steht. Aufgrund der aktuellen Corona Pandemie ist es nicht möglich das Training vor Ort laufen zu lassen, sodass dieses via *Secure Shell* (SSH) Zugriff auf dem Exponat Rechner im Docker Container läuft. In der Abbildung 11 ist der Versuchsaufbau visualisiert. Jedes Training erstellt eine spezifische „.params“ Datei, die das Wissen des jeweiligen trainierten KNN enthält. Das trainierte KNN wird daraufhin auf Schnelligkeit und Genauigkeit in weiteren Skripten überprüft. Ist die Schnelligkeit des KNN ermittelt, wird im nächsten Schritt die Genauigkeit des KNN mittels eines weiteren Skripts anhand des Validationsdatensatzes berechnet. Zum Schluss soll das KNN ermittelt werden, das die beste Kombination aus Schnelligkeit und Genauigkeit bietet.

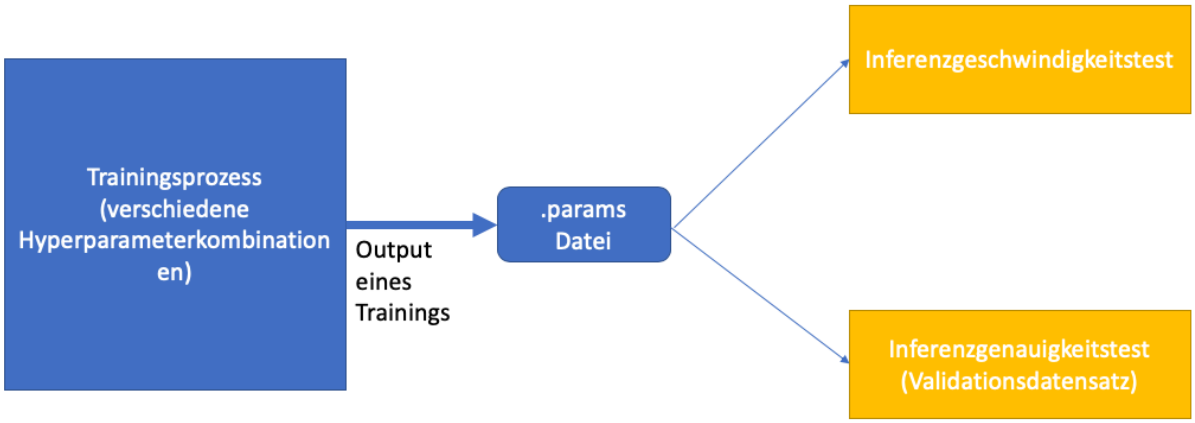

Abbildung 11: Versuchsaufbau Hyperparameteroptimierung[106]

Im Folgenden wird versucht die optimale Kombination der bereits oben angesprochenen Hyperparameter durch das Training verschiedener KNN zu bestimmen. In Abbildung 11 befindet sich dieser Schritt also im blauen Kasten „Trainingsprozess“. Für eine solche Hyperparameteroptimisierung gibt es verschiedene Verfahren, die verwendet werden können. Es wurde sich in dieser Arbeit für die bekannteste Methode, *grid search*, entschieden. In diesem Verfahren treten unterschiedliche Kombinationen von Hyperparametern gegeneinander an, sodass das

[106] Eigene Darstellung

beste Modell auf Basis von Schnelligkeit und Genauigkeit ermittelt werden kann[107]. Um die Kombination zu ermitteln, werden vom Nutzer im Vorhinein die Wertebereiche der Parameter in einen sinnvollen Bereich eingegrenzt, sodass mithilfe des kartesischen Produktes unterschiedliche Kombinationen errechnet werden können[108]. Das Training einer Kombination dauert in etwa 6 Stunden, daher ist es sinnvoll für jeden Hyperparameter eine Einschränkung des Wertebereichs vorzunehmen, sodass Zeit und Computerressourcen gespart werden.

4.2 Eingrenzung des Wertebereichs der Hyperparameter

Da ein KNN optimiert werden soll, dass das komplexe Problem der Objektlokalisierung und Objektklassifizierung in fremden Bildern lösen soll, empfehlen Wilson/Martinez die *learning rate* in geringen Spektren anzusetzen[109]. Eine zu hohe *learning rate* könnte zudem die Fähigkeit der Generalisierung der KNN negativ beeinflussen und das Training stark verlangsamen. Die Gefahr einer zu geringen *learning rat*e besteht darin, dass das Training erheblich länger dauert und somit Computerressourcen verschwendet werden[110,111]. Des Weiteren kann eine zu große *learning rate* die Gefahr des *overfitting* erhöhen[112]. Dieses Risiko wird für die Simulationen minimiert, indem der Wert der *learning rate* auf lediglich zwei Stellen hinter dem Komma gesetzt wird. Die Ergebnisse verschiedener Simulationen von Wilson/Martinez zeigen deutlich, dass die performantesten Werte für die *learning rate* im Bereich von 0.01 bis 0.05 liegen[113]. Dieser Werte befinden sich im geringen Spektrum der learning rates und sorgen daher für eine starke Beanspruchung von Computerresourcen. Da der Optimierungsvorgang jedoch ausschließlich einmal stattfindet, wird dies in Kauf genommen. Die optimale Einstellung dieses Hyperparameter ist stark abhängig von dem zu lösenden Problem, weshalb Kombinationen mit 0.01, 0.02, 0.035 und 0.05 als *learning rate* simuliert werden.

Die Hyperparameter sind abhängig voneinander und müssen aufeinander abgestimmt sein. Wählt man beispielsweise eine große *learning rate* sollte der Wert des *weight decays* geringgehalten werden, um der Gefahr des *overfittings* entgegenzuwirken. Dasselbe gilt im umgekehrten Fall[114]. Für große *learning rates* Werte empfiehlt Smith daher einen *weight decay* Wert von 10^{-4}. Für geringe *learning rate* Werte sollte das *weight decay* im 10^{-3} Bereich liegen[115]. Der

[107] Vgl. Ebert 2019, S. 36
[108] Vgl. Hutter 2014, S. 7
[109] Vgl. Wilson/Martinez 2001, S. 115
[110] Vgl. Smith 2017, S. 464
[111] Vgl. Wilson/Martinez 2001, S. 115
[112] Vgl. Smith 2018, S. 6
[113] Vgl. Wilson/Martinez 2001, S. 115
[114] Vgl. Smith 2018, S. 7
[115] Vgl. Smith 2018, S. 7

Wertebereich der obigen gewählten *learning rates* ist zwar gering, dennoch sollten *weight decay* Werte im Bereich 10^{-4} in der Simulation enthalten sein. Daher werden die folgenden Werte für den Hyperparameter *weight decay* vorgeschlagen: 0.0005, 0.0008 und 0.0011.

Ein weiterer zu bestimmender Hyperparameter ist *momentum*. Dieser Parameter hängt ebenfalls stark mit der *learning rate* zusammen. Bei ähnlichen *learning rates*, wie sie bereits für diese Arbeit festgelegt wurden, erzielten *momentum* Werte im Bereich zwischen 0.7 bis 0.95 die besten Ergebnisse.[116] Erneut ist dies stark abhängig von dem jeweiligen Problem, es macht jedoch Sinn nach dem optimalen *momentum* Wert in diesem Bereich zu suchen. Die zu testenden Werte sind demnach: 0.75, 0.85 und 0.95.

Der letzte, zu bestimmende Parameter ist *epoch*. *Epoch* ist ein komplexer Hyperparameter, da dieser stark vom Datensatz abhängig ist, auf den das KNN trainiert werden soll. Aus früheren Experimenten mit dem Datensatz lassen sich brauchbare Ergebnisse erst ab circa 80 *epochs* auslesen. Dieser Hyperparameter wird für die folgenden Simulationen als Konstante bei 90 belassen. Im Rahmen dieser Praxisarbeit ist es zeitlich nicht möglich *epochs* als einen weiteren, unbestimmten Parameter anzuführen, da dies zu einem erheblichen Anstieg aller möglichen und somit zu testenden Kombinationen führen würde. Die Mengen LR:{0.01, 0.02, 0.035, 0.05}, WD:{0.0005, 0.0008, 0.0011} und M:{0.75, 0.85, 0.95} repräsentieren in der Mengenlehre die zu bestimmenden Parameter. Dabei steht LR für die *learning rate*, WD für das *weight decay* und M für das *momentum*. Aus diesen Mengen werden nun alle denkbaren Kombinationen ermittelt. Es wird für jede Kombination ein KNN erstellt. Die möglichen Kombinationen sind der Tabelle 1 zu entnehmen und werden im Folgenden über ihren Index referenziert.

Index	LR	WD	M	Index	LR	WD	M
1	0.01	0.0005	0.75	19	0.035	0.0005	0.75
2	0.01	0.0005	0.85	20	0.035	0.0005	0.85
3	0.01	0.0005	0.95	21	0.035	0.0005	0.95
4	0.01	0.0008	0.75	22	0.035	0.0008	0.75
5	0.01	0.0008	0.85	23	0.035	0.0008	0.85
6	0.01	0.0008	0.95	24	0.035	0.0008	0.95
7	0.01	0.0011	0.75	25	0.035	0.0011	0.75
8	0.01	0.0011	0.85	26	0.035	0.0011	0.85
9	0.01	0.0011	0.95	27	0.035	0.0011	0.95
10	0.02	0.0005	0.75	28	0.05	0.0005	0.75
11	0.02	0.0005	0.85	29	0.05	0.0005	0.85
12	0.02	0.0005	0.95	30	0.05	0.0005	0.95
13	0.02	0.0008	0.75	31	0.05	0.0008	0.75
14	0.02	0.0008	0.85	32	0.05	0.0008	0.85
15	0.02	0.0008	0.95	33	0.05	0.0008	0.95
16	0.02	0.0011	0.75	34	0.05	0.0011	0.75
17	0.02	0.0011	0.85	35	0.05	0.0011	0.85
18	0.02	0.0011	0.95	36	0.05	0.0011	0.95

Tabelle 1: Kombination der Hyperparameter[117]

[116] Vgl. Smith 2018, S. 9
[117] Eigene Darstellung

4.3 Inferenzgeschwindigkeit Simulation

Die Inferenzgeschwindigkeit kann für die Verwendung eines KNN eine ausschlaggebende Metrik sein. Soll ein KNN für Objekterkennung in Echtzeit Objekte in einem Videofeed erkennen können, so ist es von Bedeutung, dass die einzelnen Bilder des Videofeeds möglichst schnell von dem KNN verarbeitet werden können, sodass eine Echtzeiterkennung realisiert wird. Die Verarbeitungsgeschwindigkeit der einzelnen Bilder ist abhängig von der Konfiguration des KNN. Die verschiedenen Kombinationen der Hyperparameter haben also einen Einfluss auf diese. Darüber hinaus ist die Verarbeitungsgeschwindigkeit abhängig von der Computerhardware, auf der die Bilder vom KNN inferiert werden. Die nachfolgende Inferenzgeschwindigkeitssimulation wurde auf einem MacBook Pro 13‘‘ mit einer 2,3 GHz Dual-Core Intel i5 durchgeführt. Die Hardware gilt allerdings als Konstante in der Simulation und die Messwerte sind relativ zueinander zu betrachten. Für eine Echtzeiterkennung ist es sinnvoll das KNN auf einem Computer mit einer leistungsfähigen Grafikkarte laufen zu lassen. Damit jedes KNN unter denselben Voraussetzungen auf dessen Geschwindigkeit getestet wird, wurden 100 zufällige Bilder aus dem Validationsdatensatz ausgewählt. Die ausgewählten Bilder werden von dem KNN inferiert. Während des Inferenzprozesses wird die Zeit gemessen, die das KNN in Anspruch nimmt. Mit Hilfe der Zeit lassen sich die verarbeiteten Bilder pro Sekunde

Index	Inferenzzeit (in s)	FPS	Index	Inferenzzeit (in s)	FPS
1	41,27	2,42	19	46,13	2,17
2	41,45	2,41	20	41,00	2,44
3	42,83	2,33	21	41,41	2,41
4	43,12	2,32	22	44,42	2,25
5	40,49	2,47	23	42,12	2,37
6	41,09	2,43	24	44,43	2,25
7	42,29	2,36	25	42,17	2,37
8	41,01	2,43	26	42,51	2,35
9	42,62	2,35	27	45,27	2,21
10	42,32	2,36	28	40,97	2,44
11	44,51	2,25	29	40,77	2,45
12	44,34	2,26	30	46,75	2,13
13	41,65	2,40	31	42,42	2,36
14	44,94	2,23	32	44,32	2,26
15	43,00	2,33	33	41,65	2,40
16	43,18	2,32	34	42,50	2,35
17	41,26	2,42	35	44,12	2,27
18	46,28	2,16	36	45,90	2,18
19	46,13	2,17			

Tabelle 2: Inferenzgeschwindigkeitsmessungen[118]

[118] Eigene Darstellung

errechnen, indem die 100 Bilder durch die in Anspruch genommenen Gesamtzeit geteilt werden. Diese Metrik wird im Folgenden als FPS bezeichnet.

Die unterschiedlichen Inferenzgeschwindigkeiten der in 4.1.1 beschriebenen, verschiedenen KNN lassen sich aus der Tabelle 2 ablesen. Auf den ersten Blick erscheinen die Werte der Ergebnisse dicht aneinander. Besonders in der FPS Metrik sind die Ergebnisse aller Messreihen im Zweierbereich. Es ist jedoch zu beachten, dass die Messergebnisse auf einem leistungsschwachen Prozessor erzielt worden sind. Sollten dieselben Simulationen auf einem performanteren Prozessor oder einer performanteren Grafikkarte durchgeführt werden, so wäre die Rangordnung der Ergebnisse zwar stets gleich, die Differenz zwischen den Messergebniswerten jedoch deutlich stärker erkennbar. Darüber hinaus sind selbst Verbesserungen im Nachkommabereich der FPS Metrik nicht von der Hand zu weisen. Ein 2-minütiges Video mit 30 FPS würde von einer KNN mit einer 2,4 FPS Inferenzgeschwindigkeit in 25 Minuten verarbeitet werden. Dahingegen würde dasselbe Video bei einer KNN mit einer Inferenzgeschwindigkeit von 2,2 FPS mehr als zwei Minuten länger benötigen. Stellt man denselben Vergleich für Videos mit mehreren Stunden Länge auf, so wird die Relevanz der Nachkommastelle umso deutlicher.

Anhand der Tabelle 2 ist deutlich erkennbar, dass die Kombination fünf am performantesten inferiert (grün gekennzeichnet). Mit einer Inferenzzeit von 40,49 Sekunden erzielt das KNN eine FPS Anzahl von 2,47 Sekunden und liegt somit 0,03 FPS vor den zweitbesten Messungen, der Kombination 20 und 28, welche beide eine Inferenzgeschwindigkeit von 2,44 FPS erzielen. Eine eindeutige Abhängigkeit der Inferenzgeschwindigkeit von der konkreten Einstellung der Hyperparameter lässt sich nicht ermitteln. Dennoch ist deutlich zu erkennen, dass die Kombination 30 mit einer gemessenen Inferenzgeschwindigkeit von 2,13 FPS am langsamsten inferiert. Es ist zu vermuten, dass die Kombination einer recht großen *learning rate* sowie eines großen *momentums* mit einem geringen *weight decay* schlichtweg nicht performant ist. Doch auch eine solche Vermutung lässt sich nicht unbedingt mittels anderer Messreihen bestätigen.

4.4 Inferenzgenauigkeit Simulation

Nachdem die Inferenzgeschwindigkeit der Messreihen ermittelt wurde, gilt es im Folgenden die Genauigkeit der verschiedenen Messreihen zu untersuchen. Die Inferenzgenauigkeit stellt die nahezu wichtigste Metrik in einem KNN dar. Ein schnelles KNN, das Objekte konstant falsch erkennt, ist in den wenigstens Fällen hilfreich. Daher gilt es zu überprüfen mit welcher Konfidenzrate ein KNN bestimmte Klassen ermittelt. Die folgenden Simulationen wurden daher anhand von zehn zufällig gewählten Bildern aus dem Validationsdatensatz getätigt. In diesen zehn Bildern wurden die Klassengenauigkeiten für die folgenden sechs Klassen ermittelt:

Elefant, Papagei, Tiger, Leopard, Giraffe und Nashorn. Ist ein Objekt in dem Bild korrekt erkannt worden, so wird die Konfidenzrate der KNN für diese Erkennung notiert. Die Konfidenzrate ist grundsätzlich in Prozent angegeben. Anhand der obigen ausgewählten Klassen lässt sich ein Durchschnitt für das gesamte KNN ermitteln, der als Vergleichsmetrik zu anderen KNN dient. Dieser Durchschnitt wird berechnet, indem die verschiedenen Klassendurchschnittskonfidenzraten zusammenaddiert und durch die Anzahl der Klassen geteilt werden. Dieser Durchschnitt wird im Folgenden als Gesamtklassendurchschnittskonfidenzrate bezeichnet.

	Elefant	Papagei	Tiger	Leopard	Giraffe	Nashorn	Ø
1	97,3	96,8	87,6	95,6	96,6	99,0	95,6
2	98,4	95,8	91,0	96,8	99,0	98,6	96,6
3	0,0	0,0	0,0	0,0	0,0	0,0	0,0
4	90,4	93,0	98,4	93,0	97,1	92	94,0
5	88,5	93,0	98,4	92,9	97,1	91,8	93,9
6	85,0	67,5	74,4	83,3	54,8	54,7	69,4
7	97,1	98,4	89,1	97,8	97,3	94,8	95,8
8	92,9	76,6	87,2	0,0	93,6	51,6	67,0
9	81,1	94,9	85,8	67,1	91,9	91,6	85,4
10	93,2	83,4	94,8	70,2	97,1	92,5	88,5
11	92,7	85,4	83,7	87,4	74,3	63,1	81,1
12	76,7	74,3	73,7	75,0	89,5	0,0	64,0
13	88,5	94,2	77,4	87,3	86,8	73,5	84,6
14	86,9	87,7	87,6	93,5	92,2	74,7	87,1
15	77,7	0,0	83,3	0,0	79,1	76,8	52,0
16	94,5	92,9	86,4	95,0	86,2	53,0	84,7
17	78,3	94,5	64,7	0,0	81,6	78,9	66,3
18	0,0	0,0	0,0	0,0	0,0	0,0	0,0
19	0,0	59,9	69,1	71,8	60,4	51,2	52,1
20	60,0	78,6	59,0	0,0	62,5	58,3	53,1
21	73,7	89,3	96,2	80,1	86,2	84,5	85,0
22	68,9	0,0	0,0	0,0	0,0	0,0	11,5
23	65,8	80,7	72,1	79,5	94,5	63,0	75,9
24	0,0	0,0	0,0	0,0	0,0	0,0	0,0
25	74,4	74,9	70,2	75,8	67,8	0,0	60,5
26	56,7	63,7	0,0	0,0	0,0	0,0	20,0
27	0,0	0,0	0,0	0,0	0,0	0,0	0,0
28	77,7	58,8	68,1	88,7	79,3	0,0	62,1
29	0,0	0,0	0,0	0,0	0,0	0,0	0,0
30	0,0	0,0	0,0	0,0	0,0	0,0	0,0
31	90,4	74,1	78,9	0,0	0,0	0,0	40,6
32	0,0	0,0	0,0	0,0	0,0	0,0	0,0
33	90,9	0,0	0,0	0,0	85,6	0,0	29,4
34	89,0	80,3	66,5	0,0	82,0	0,0	53,0
35	0,0	0,0	0,0	0,0	0,0	0,0	0,0
36	0,0	0,0	0,0	0,0	0,0	0,0	0,0

Tabelle 3: Inferenzgenauigkeitsmessungen[119]

[119] Eigene Darstellung

In Tabelle 3 sind sowohl die Gesamtklassendurchschnittskonfidenzraten als auch die spezifischen Klassenkonfidenzraten pro KNN dargestellt. Ebenfalls sind einige Teile verschiedener Messreihen grau markiert und durchgestrichen. In diesen Messreihen hat das KNN kein Objekt der Klasse in den Bildern erkennen können, sodass sich eine Konfidenzrate von Null ergibt. Für den Gesamtdurchschnitt wurden diese als null in die Kalkulation aufgenommen. Die Ergebnisse dieser Messreihen differenzieren sich deutlicher voneinander als die Ergebnisse der Inferenzgeschwindigkeit. Aus der Tabelle lässt sich entnehmen, dass die Kombination zwei mit einer Klassendurchschnittskonfidenzrate von 96,6% die höchste Genauigkeit aller Messreihen besitzt (grün markiert in Tabelle 3). Dicht gefolgt von Kombination sieben, eins und vier mit jeweils 95,8%, 95,6% und 94%. Einige Messreihen konnten keine der ausgewählten Klassen in den zehn Bildern ausfindig machen, sodass sich eine Gesamtklassendurchschnittskonfidenzrate von 0% ergibt (rot markiert in Tabelle 3). Im Gegensatz zu den Messreihen für die Inferenzgeschwindigkeit lässt sich in diesen Messreihen ein Muster erkennen. Grundsätzlich kann behauptet werden, dass die Inferenzgenauigkeit ab einer *learning rate* größer gleich 0.035 stark leidet. Genauigkeitsergebnisse von Kombinationen mit einer *learning rate* gleich 0.05 sind nahezu ausnahmslos null Prozent. Es können also keine Objekte im Bild festgestellt werden. Zudem ist stark auffällig, dass besonders das Nashorn schwer erkannt wird. Diese Klasse trägt die höchste Anzahl an 0% Konfidenzraten. Dies liegt an einem Problem, das bereits in Kapitel 2.3 beschrieben wurde. Die Interklassendistanz zwischen dem Elefanten und dem Nashorn ist zu gering, sodass das Nashorn häufig fälschlicherweise als Elefant klassifiziert wird. Dies ist ersichtlich in Abbildung 12.

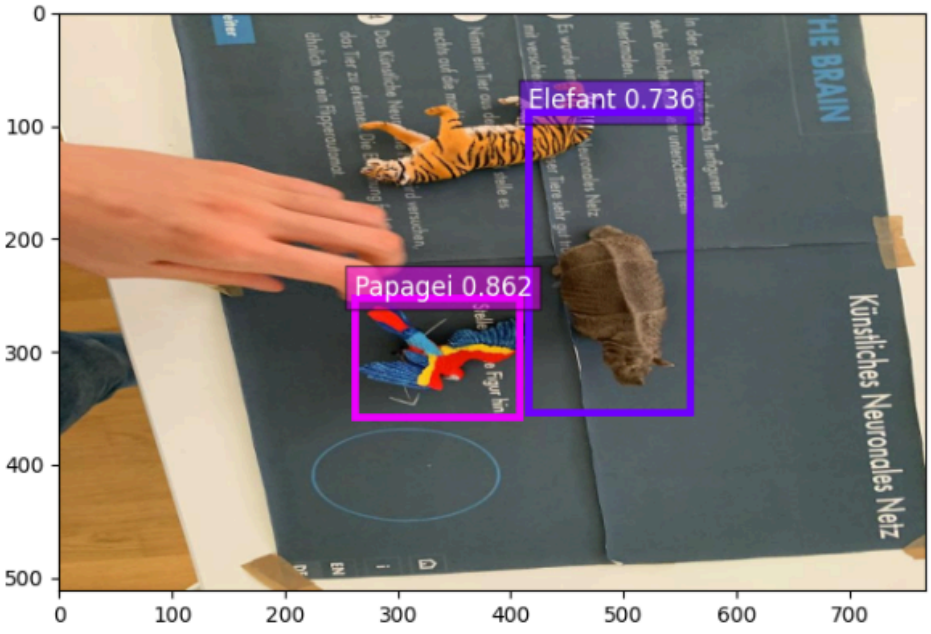

Abbildung 12: Problem der Interklassendistanz[120]

[120] Eigene Darstellung

4.5 Kombinationsauswahl

Nachdem sowohl Messungen für die Inferenzgeschwindigkeit als auch für die Inferenzgenauigkeit vorliegen, gilt es nun die passende Kombination auszuwählen. Anhand der Inferenzgeschwindigkeitsergebnisse wurde bereits ermittelt, dass die Kombination fünf am schnellsten inferiert mit 2,47 FPS. Die Kombination fünf erzielt in den Inferenzgenauigkeitstests eine Klassendurchschnittskonfidenzrate von 93,9%. Die beste Klassendurchschnittskonfidenzrate erzielt die Kombination zwei mit 96,6%. Diese erzielt eine FPS Rate von 2,41. Welche Kombination nun ausgewählt wird ist stark abhängig von dem Einsatz des KNN. Ist die Genauigkeit des Ergebnisses einer KNN von hoher Bedeutung und die Inferenzzeit kein allzu großer Faktor, so sollte sich für die Kombination zwei entschieden werden. Ein Beispiel, in dem Genauigkeit eine höhere Priorität besitzt als die Zeit, ist das Erkennen von Tumoren in Bildern. Eine hohe Genauigkeit ist hier von größerer Bedeutung. Hingegen sollte sich für Kombination zwei entschieden werden, wenn eine Erkennung in Echtzeit gefragt ist und die Hardware des Computers keine Echtzeiterkennung mittels Kombination zwei bewältigen kann.

Im Deutschen Museum wird eine KNN für Objekterkennung eingesetzt, die eine Erkennung in Echtzeit bewältigen muss. Zwar könnte behauptet werden, dass die Anforderung an die Echtzeiterkennung darauf schließen lässt, dass das KNN mit der Kombination fünf gewählt wird. Jedoch ist in dem Showcase ein leistungsstarker Computer mit einer Hochleistungsgrafikkarte im Einsatz. Diese ermöglicht die Verwendung der Kombination zwei trotz Echtzeiterkennung.

Neben Kombination zwei und fünf gibt es eine weitere Kombination, die in Betracht kommen könnte. Diese ist ein Kompromiss zwischen Qualität und Schnelligkeit. Sollte Kombination zwei nicht von der Hardware bewältigt werden können, Kombination fünf jedoch eine zu geringe Genauigkeit liefern, so könnte Kombination eins ausgewählt werden. Diese ist die einzige Kombination, die eine Inferenzgenauigkeit im 90% Bereich liefert, bei einer Inferenzgeschwindigkeit von über 2,4 FPS. Andere Kombination im 90% Bereich sind deutlich langsamer im Inferenzprozess und stellen somit keinen lohnenswerten Kompromiss dar. Die Kombination eins liegt mit einer Inferenzgenauigkeit von 95,6% und einer Inferenzgeschwindigkeit von 2,42 FPS zwischen Kombination zwei und fünf. Die Geschwindigkeitsdifferenz zwischen Kombination zwei und fünf liegt allerdings bei lediglich 0,01 FPS. Ob sich diese minimale Geschwindigkeitsverbesserung lohnt muss für das jeweilige Projekt entschieden werden.

5 Schlusskapitel

5.1 Zusammenfassung

Im Laufe dieser Arbeit wurden 36 verschiedene KNN erstellt. Diese wurden auf 2000 selbstgemachte Bilder von Hartgummitierfiguren trainiert. Die KNN unterscheiden sich in ihrer Angabe der Hyperparameter. Genauer differenziert sich jedes KNN in der Angabe der *learning rate*, des *momentum,* und des *weight decay.* Da diese Hyperparameter sowohl die Genauigkeit als auch die Geschwindigkeit in dem Inferenzprozess beeinflussen, wurden die verschiedenen KNN auf diese beiden Kategorien nach dem Training getestet. Die Genauigkeit und die Geschwindigkeit sind die beiden größten Anforderungen an ein KNN. Dabei gilt zu beachten, dass unterschiedliche Projekte unterschiedlich viel Wert auf die genannten Anforderungen legen. Zunächst wurde die Inferenzgeschwindigkeit der verschiedenen KNN getestet und notiert. Daraufhin ist die Inferenzgenauigkeit ermittelt worden, indem sechs verschiedene Klassen ausgewählt wurden und deren Klassendurschnittskonfidenzraten errechnet wurden. Im Anschluss wurde bewertet welche Kombination sich für welche Nutzzwecke anbietet. Insgesamt konnten drei sinnvolle Kombinationen ermittelt werden, die je nach Bedarf verwendet werden können.

5.2 Verbesserungsmöglichkeiten

Es wurde in dieser Arbeit deutlich, dass das Hyperparametertuning Grenzen aufweist. So konnten im Kontext dieser Arbeit lediglich 36 Kombinationen getestet werden, wobei nur 3 Parameter variabel waren. Es gibt viele weitere Hyperparameter, die in darauffolgenden Arbeiten optimiert werden könnten. Dazu gehört unter anderem der Hyperparameter *epoch*. Besonders aus den Ergebnissen der Inferenzgeschwindigkeitstest wurde ersichtlich, dass teilweise kein eindeutiger Zusammenhang der Ergebnisse ermittelt werden konnte und es somit darauf ankommt, so viele Kombinationen wie möglich zu testen. In fortführenden Arbeiten könnte ebenfalls ein noch kleinerer Bereich der *learning rates* getestet werden, da besonders die *learning rates* mit geringen Werten vielversprechende Ergebnisse aufwiesen.

Zudem wurden einige Einschränkungen für diese Arbeit getroffen, die in fortführenden Arbeiten abgeändert werden könnten. So wurde beispielsweise lediglich der *SGD optimizer* verwendet. In weiteren Arbeiten könnte ein Vergleich der verschiedenen *optimizer* aufgestellt werden.

Die Inferenzgeschwindigkeit und Qualität sind außerdem abhängig von weiteren Bedingungen, wie der Kameraauflösung, der Struktur des KNN und der Hardware des PCs. Auch diese Bedingungen könnten verglichen und optimiert werden.

5.3 Ausblick

In dieser Arbeit wurde eine hoch performante Hyperparameterkombination ermittelt, die eine Echtzeitverarbeitung von einem live Webcam feed ermöglicht. Die in dieser Arbeit gefundenen Kombinationen können somit im KNN für Objekterkennung im Deutschen Museum Showcase der IBM implementiert werden. Die Eröffnung des Museums ist bereits im Frühjahr 2021 geplant.

Literaturverzeichnis

Borana, J. (2016): Industry Applications of Artificial Intelligence. *Proceeding of International Conference on Emerging Technologies in Engineering, Biomedical, Management and Science [ETEBMS-2016], 5-6 March 2016*, 4. https://pdfs.semanticscholar.org/d5b0/61e6565ce421b4b0b7d56296e882085dc308.pdf

Bottou, L. (1991): Stochastic Gradient Learning in Neural Networks. *Proceedings of Neuro-Nımes*, *91*(8), 12. https://leon.bottou.org/publications/pdf/nimes-1991.pdf

Deepika Mallampati. (2018): An Efficient Spam Filtering using Supervised Machine Learning Techniques. *International Journal of Scientific Research in Computer Science and Engineering*, *6*(2), 33–37. https://doi.org/10.26438/ijsrcse/v6i2.3337

Ebert, D. (2019): *Bildklassifizierung mit Neuronalen Netzen*. https://opendata.uni-halle.de/handle/1981185920/14186

Elverson, A. (2018): *Spotify: Can machine learnign drive content generation?* MBA Student Perspectives. https://digital.hbs.edu/platform-rctom/submission/spotify-can-machine-learning-drive-content-generation/

Erhardt, A. (2017): *Segmentierung von Lungen auf Röntgenaufnahmen*. http://www.is.informatik.uni-wuerzburg.de/fileadmin/10030600/Mitarbeiter/Reul_Christian/Studentische_Arbeiten/2017/LungenSegmentierung_Bachelorarbeit_Erhardt.pdf

GitHub IO. (n.d.): *Convolutional Neural Networks for Visual Recognition*. https://cs231n.github.io/convolutional-networks/

***Google Trends*. (2020):** https://trends.google.com/trends/?geo=US

Grundlagen und Aufbau von neuronalen Netzen. (n.d.): *Ulm, Universität*. https://www.informatik.uni-ulm.de/ni/Lehre/SS02/Evosem/OptimierungKNNFolien.pdf

Halgamuge, M. N., Daminda, E., & Nirmalathas, A. (2020): Best optimizer selection for predicting bushfire occurrences using deep learning. *Natural Hazards*, *0123456789*. https://doi.org/10.1007/s11069-020-04015-7

Haralick, R., & Shapiro, L. (1984): Image Segmentation Techniques. *Machine Vision International, Ann Arbor, Michigan*, 112. https://www.sciencedirect.com/science/article/pii/S0734189X85901537

Hutter, F., Kotthoff, L., & Vanschoren, J. (Eds.). (2019): *Automated Machine Learning : Methods, Systems, Challenges*. Springer Nature PP - Cham. https://doi.org/10.1007/978-3-030-05318-5

Kinnebrock, W. (1994): *Neuronale Netze - Grundlagen, Anwendungen, Beispiele* (2. Auflage). R. Oldenbourg Verlag München. https://books.google.de/books?hl=en&lr=&id=UBd5DwAAQBAJ&oi=fnd&pg=PA6&dq=Kinnebrock,+W.+(1992):+Neuronale+Netze+-+Grundlagen,+Anwendungen,+Beispiele.+Walter+de+Gruyter+GmbH+%26+Co+KG.&ots=SUSsK5xwAI&sig=fnASSlb18eO-ENmTZcCJpiKl2E4&redir_esc=y#v=onepage&q=

Kok, J. N., Boers, E. J. W., Kosters, W. A., Putten, P. Van Der, & Poel, M. (2010): Knowledge for sustainable development: an insight into the Encyclopedia of life support systems::::::Artificial Intelligence: Definition, Trends, Techniques and Cases. *Encyclopedia of Life Support Systems (EOLSS)*, 1096–1097. https://www.eolss.net/Sample-Chapters/C15/E6-44.pdf

Lammers, D., & Wachenfeld, S. (2003): Objekterkennung in Bilddaten. *Internal Report Uni Münster*. https://www.uni-muenster.de/Informatik/u/lammers/EDU/ws03/Landminen/Abgaben/Gruppe9/Thema09-ObjekterkennungInBilddaten-ChristianGrosseLordemann-MartinLambers.pdf

Lecun, Y., Bengio, Y., & Hinton, G. (2015): Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Li, H., Zhang, L., Zhou, X., & Huang, B. (2017): Cost-sensitive sequential three-way decision modeling using a deep neural network. *International Journal of Approximate Reasoning*, *85*, 68–78. https://doi.org/10.1016/j.ijar.2017.03.008

Lüdi, G. (2007): *Künstliches Neuronales Netz Abbildung*. http://www.natur-struktur.ch/ai/neuronale.html

Michelucci, U. (2019): Advanced Applied Deep Learning. In *Advanced Applied Deep Learning*. https://doi.org/10.1007/978-1-4842-4976-5

MITX PC. (n.d.): *Difference between Training and Inference of Deep Learning Frameworks*. https://mitxpc.com/pages/ai-inference-applying-deep-neural-network-training

Munoz, A. (2015): Online Learning and Optimization. *Courant Institute of Mathematical Sciences, New York, NY.*, 1–8. https://doi.org/10.1007/978-3-642-27848-8_265-2

MxNet. (n.d.): *Optimizers*. https://mxnet.apache.org/api/python/docs/tutorials/packages/optimizer/index.html

Nandakumar, S. R., Boybat, I., Joshi, V., Piveteau, C., Le Gallo, M., Rajendran, B., Sebastian, A., & Eleftheriou, E. (2019): Phase-Change Memory Models for Deep Learning Training and Inference. *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 727–730. https://doi.org/10.1109/ICECS46596.2019.8964852

Ongsulee, P. (2018): Artificial intelligence, machine learning and deep learning. In *International Conference on ICT and Knowledge Engineering* (pp. 1–6). https://doi.org/10.1109/ICTKE.2017.8259629

Park, L. A. F. (2011): Bootstrap confidence intervals for mean average precision. *Proceedings of the Fourth ASEARC Conference*, *February*, 51–54. http://ro.uow.edu.au/asearc/22/

Peterwitz, J. (2006): Grundlagen: Bildverarbeitung / Objekterkennung. In *Technische Universität München*. http://www9.in.tum.de/seminare/hs.SS06.EAMA/material/01_ausarbeitung.pdf

Ruder, S. (2016): An overview of gradient descent optimization algorithms. *Insight Centre for Data Analytics*, 1–14. http://arxiv.org/abs/1609.04747

Samek, W., Wiegand, T., & Müller, K.-R. (2017): *Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models*. http://arxiv.org/abs/1708.08296

Smith, L. N. (2017): Cyclical Learning Rates for Training Neural Networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 464–472. https://doi.org/10.1109/WACV.2017.58

Smith, L. N. (2018): A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay. *US Naval Research Laboratory Technical Report*, 1–21. http://arxiv.org/abs/1803.09820

Strecker, S. (1997): Künstliche Neuronale Netze – Aufbau und Funktionsweise. *Arbeitspapiere WI*, *10*(10), 1–32. http://geb.uni-giessen.de/geb/volltexte/2004/1697/pdf/Apap_WI_1997_10.pdf

Süße, H., & Rodner, E. (2014): Bildverarbeitung und Objekterkennung. In *Bildverarbeitung und Objekterkennung*. https://doi.org/10.1007/978-3-8348-2606-0

Taulli, T. (2004): Artificial intelligence Basics - A Non-Technical Introduction. In *International Carpet Bulletin* (Issue 3). Artificial intelligence Basics - A Non-Technical Introduction

Tawil, M. (1999): *Künstliche Neuronale Netze - Methode und Anwendung*. 4. https://d-nb.info/1038247624/34#page=72

Traeger, M., Eberhart, A., Geldner, G., Morin, A. M., Putzke, C., Wulf, H., & Eberhart, L. H. J. (2003): Vorhersage von Übelkeit und Erbrechen in der postoperativen Phase durch ein künstliches neuronales Netz. *Anaesthesist*, *52*(12), 1132–1138. https://doi.org/10.1007/s00101-003-0575-y

Traeger, M., Eberhart, A., Geldner, G., Morin, A. M., Putzke, C., Wulf, H., & Eberhart, L. H. J. (2003): Künstliche neuronale Netze: Theorie und Anwendungen in der Anästhesie, Intensiv- und Notfallmedizin. *Anaesthesist*, *52*(11), 1055–1061. https://doi.org/10.1007/s00101-003-0576-x

Walch Kathleen. (2019): Rethinking Weak Vs. Strong AI. *Forbes - COGNITIVE WORLD*, 1. https://www.forbes.com/sites/cognitiveworld/2019/10/04/rethinking-weak-vs-strong-ai/#14d014fb6da3

Wilson, D. R., & Martinez, T. R. (2001): The need for small learning rates on large problems. *Proceedings of the International Joint Conference on Neural Networks*, *1*, 115–119. https://doi.org/10.1109/ijcnn.2001.939002

Wittpahl, V. (2018): Künstliche Intelligenz. In *Insitut für Innovation und Technik* (Vol. 73, Issue 4). https://doi.org/10.3790/dbw.60.1.14

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018): Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, *9*(4), 611–629. https://doi.org/10.1007/s13244-018-0639-9

Erklärung

Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema: „Optimierung verschiedener Hyperparameter einer künstlichen Intelligenz für Objekterkennung, genutzt im Rahmen eines Museums Showcase“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Zürich, den 28.08.20 L. Weißbeck

(Ort, Datum) (Unterschrift)