

FUNKTIONSWEISE UND ANWENDUNGSMÖGLICH- KEITEN VON KÜNSTLICHER INTELLIGENZ

Vorwissenschaftliche Arbeit

eingereicht von

Max Spannring

bei

Prof. Dr. Ingo Rath

im Februar 2020

Akademisches Gymnasium Salzburg

Abstract

Künstliche Intelligenz ist eines der angesagtesten Themen der modernen Computerwissenschaft, wird aber oft als Buzzword verwendet, ohne die Technik dahinter zu verstehen. Diesen Begriff zu entmystifizieren und verständlich zu erklären, ist ein Ziel dieser Arbeit. Ich werde einen kurzen Überblick über die vielen verschiedenen Subthemen geben, und mich anschließend auf Deep Learning konzentrieren und dabei neuronale Netze, deren Aufbau und Lernweise, ausführlich erläutern.

Ein zweites Ziel ist der Sprung von der Theorie in die Praxis. Im letzten Drittel wird, mit in Experimenten gewonnenen Erkenntnissen, eine App entwickelt, die Optical Character Recognition nutzt, um alle Vokabeln eines lateinischen Textes zu übersetzen. Die App heißt Latein Pro, und ist als moderne Alternative für Schüler zum klassischen Wörterbuch gedacht. Heruntergeladen werden kann die App kostenlos im Google Play Store, der Source Code steht auf GitHub online zur Verfügung.

Die Arbeit geht nicht auf ethische oder gesellschaftliche Fragen im Zusammenhang mit dem Thema ein, Lesern wird geraten, sich über diese Themen mittels anderweitiger Literatur zu informieren.

1. Inhaltsverzeichnis

Abstract	ii
1. Inhaltsverzeichnis.....	iii
2. Einleitung.....	1
3. Überblick	1
3.1. Machine Learning.....	2
3.2. Der Turing-Test	3
3.3. Starke & schwache KI.....	3
4. Anwendungsmöglichkeiten	4
4.1. Text.....	4
4.2. Bilder	5
4.3. Audio.....	6
4.4. Video:.....	7
4.5. Analyse	7
5. Inspiriert vom menschlichen Gehirn	8
6. Allgemein technische Herangehensweise	9
6.1. Hardware.....	9
6.2. Coding	9
6.3. Daten	9
7. Aufbau von neuronalen Netzen	11
7.1. Aufbau von neuronalen Netzen, Kurzversion	11
7.2. Aufbau von neuronalen Netzen, Langversion	12
7.3. Layers	13
7.4. Warum mehrere Schichten?	13
7.5. Convolutional Layers.....	13
7.6. Pooling Layers.....	15
7.7. Flattening & Fully-Connected Layers	15
7.8. Aktivierungsfunktionen	16
7.9. Gewichte.....	17
8. Trainieren von neuronalen Netzen.....	18
8.1. Kurzfassung.....	18
8.2. Die Verlustfunktion	18

8.3.	Gradient Descent	19
8.4.	Backpropagation	21
8.5.	Der Trainingszyklus.....	22
9.	Implementierung in Real-World-Scenarios.....	23
9.1.	Machine Learning: On Device vs. Cloud-based.....	23
10.	Praxiseinsatz mit Pre-Trained Models.....	24
10.1.	Google Cloud Vision API	24
10.2.	Tesseract OCR.....	24
10.3.	Firebase ML Kit	27
11.	Praktische Anwendung: Latein Pro	29
11.1.	Die Idee	29
11.2.	Aufbau einer Android App.....	30
11.3.	Aufbau von Latein Pro.....	30
11.4.	Unterstützung bei der Entwicklung von Latein Pro.....	33
11.5.	Probleme der App	33
12.	Resümee	34
12.1.	Latein Pro.....	34
13.	Literaturverzeichnis	36
14.	Abbildungsverzeichnis.....	41

2. Einleitung

Kaum eine Technologie hat sich in den letzten Jahren so stark entwickelt, bietet so großes Potential und ist so wegweisend wie „Künstliche Intelligenz“, kurz KI genannt. Die Definition von KI selbst ist jedoch schwierig, weil der Begriff „Intelligenz“ an sich nicht klar definiert ist. Eine in der Fachwelt anerkannte Definition findet man im Collin Dictionary: „*Artificial intelligence is a type of computer technology which is concerned with making machines work in an intelligent way, similar to the way that the human mind works.*”[1]

Das Feld der AI, also „Artificial Intelligence“ (= KI), ist allerdings so umfassend, dass es natürlich unmöglich ist, alle Aspekte in einer Arbeit zu vereinen, weswegen ich mich vor allem auf den Bereich „Deep Learning“ konzentrieren werde. Dies ist ein relativ junges Teilgebiet der KI, das auf sehr vielen Gebieten, unter anderem in der Schrift/Spracherkennung, Bilderkennung, oder auch in der Bioinformatik zum Teil bessere Ergebnisse liefert als menschliche Vergleichspersonen [2].

Ziel dieser Arbeit ist es, ein Grundverständnis für die vielfältigen Anwendungsmöglichkeiten von KI zu vermitteln und die dahinterstehenden Technologien zu erklären.

Neben theoretischen Erklärungen findet sich in dieser Arbeit außerdem ein praktisches Beispiel, die App **Latein Pro**, die Deep Learning/OCR nutzt, um das Übersetzen von lateinischen Texten angenehmer zu gestalten. Die App ist seit Ende Jänner im Google Play Store verfügbar.

3. Überblick

Schon seit sehr langer Zeit beschäftigen sich Menschen mit „denkenden, menschenähnlichen Maschinen“, man denke nur an Bücher wie „Frankenstein“ oder Filme wie „Metropolis“, die schon lange vor der Entwicklung der modernen, künstlichen Intelligenz entstanden [3]. Den Startschuss für den Beginn der KI-Forschung gab die Dartmouth-Conference, die im Sommer 1956 am Dartmouth College in New Hampshire stattfand. Ziel dieser Konferenz war es, ein Konzept für denkende Maschinen zu entwickeln. [4] In den darauffolgenden Jahren wurden von den Forschern immer neue Methoden erfunden, um vermeintliche Intelligenz auf Computer zu übertragen.

Eine der ersten praktischen Anwendungen war Mycin, ein 1972 entwickeltes Programm, das zur Diagnose und Therapie von Infektionskrankheiten eingesetzt wurde [5]. Dieses sogenannte Expertensystem bestand aus 350 Bedingungen [6], die nacheinander abgearbeitet wurden und nach folgendem Prinzip funktionierten [5]:

WENN:

1. Es sich um eine bakterielle Infektion handelt
2. Keine Organismen beim Gram-Test festgestellt wurden
3. Der Patient zwischen 15 und 55 Jahre alt ist

4. Der Patient keine Kopfverletzung hat
5. Der Patient keine Gehirnhautentzündung hat

DANN:

Bei den Organismen, die die Infektion verursachen, könnte es sich um Pneumokokken oder Meningokokken handeln.

Dieses Programm erreichte zum Teil eine Genauigkeit von über 90% [5], trotzdem weisen solche Systeme entscheidende Nachteile auf: Jede Bedingung muss von einem menschlichen Experten von Hand einprogrammiert werden und das System kann daher nur so gut sein, wie der Mensch, der es programmiert, außerdem kann es nur in einem sehr schmalen Bereich eingesetzt werden. Wenn man zum Beispiel das Mycin Programm benutzen will, um Aktienkurse vorauszusagen, müsste man wieder menschliche Experten hinzuziehen, um jede einzelne Bedingung umzuschreiben und so den kompletten Code neu zu entwerfen, was einen riesigen Aufwand darstellen würde. Wäre es nicht viel effektiver, wenn der Computer diese Bedingungen selbst lernen könnte?

3.1. Machine Learning

Genau diesen Ansatz verfolgt Machine Learning, eine Subkategorie von KI, die darauf spezialisiert ist, Muster in Daten zu erkennen, um so ohne menschliche Hilfe über ein bestimmtes Thema zu lernen. Decision Trees, also Entscheidungsbäume, beziehungsweise Random Forests, eine Weiterentwicklung davon, und Support Vector Machines zählen zu den wichtigsten Algorithmen in diesem Bereich. Die Grundlage für Machine Learning bilden Trainingsdaten, aus denen relevante Merkmale extrahiert werden, um automatisierte, präzise Vorhersagen abgeben zu können. Schon in den 1960ern wurden erste Modelle, sogenannte künstliche neuronale Netze, entwickelt, die sich in ihrer Funktionsweise am menschlichen Gehirn orientieren.

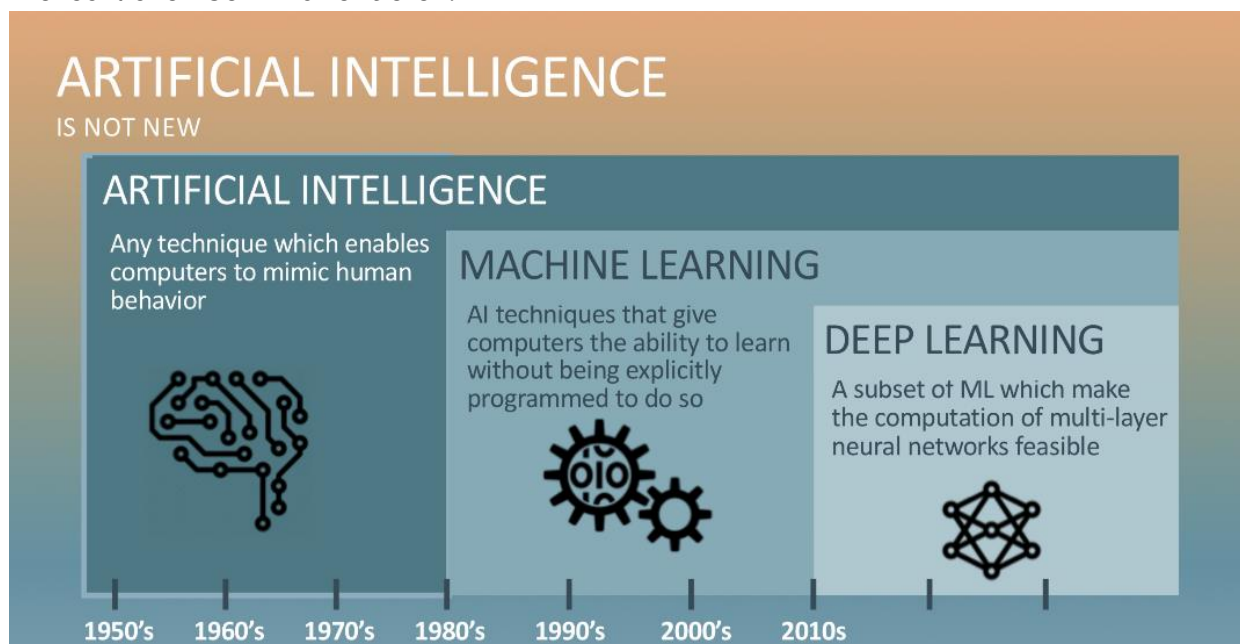


Abbildung 1: Entwicklung von KI (Quelle: <https://www.meetup.com/de-DE/SanFranciscoMachineLearning/>)

Diese wurden zwar in den nächsten Jahren von anderen, „leichter verständlichen“ Methoden wie den oben genannten Algorithmen verdrängt, erlebten in den späten 2000ern jedoch wieder einen Boom [7].

Genau in dieser Zeit wurde auch Deep Learning immer wichtiger, speziell eine Weiterentwicklung von Machine Learning, die auf mehrschichtigen neuronalen Netzwerken aufbaut. Deep Learning ist State-of-the-Art auf dem Gebiet der AI, und bietet zum Teil deutlich bessere Performance als Menschen in vielen Gebieten. Für den Erfolg von Deep Learning sind vor allem 2 Dinge besonders wichtig: Rechenleistung und riesige Datenmengen [7]. Da sich die Rechenleistung von PCs bis heute zirka alle 1.5 Jahre verdoppelt hat [8], leistungsfähige GPUs entwickelt werden und viele Firmen massenhaft Daten sammeln und viele hochqualitative Datensätze auch frei im Internet erhältlich sind, entwickelt sich die Technik auch immer noch rasant weiter.

3.2. Der Turing-Test

Eine der berühmtesten Methoden, um künstliche Intelligenz zu bestimmen, ist der nach dem britischen Mathematiker Alan Turing benannte Turing-Test. Dabei führt ein menschlicher Leiter Konversationen mit mehreren Teilnehmern, die er weder sehen und noch hören kann, über einen Computer. Für einen Roboter gilt der Test dann als bestanden, wenn der Leiter nicht erkennen kann, ob es sich bei ihm um einen Menschen oder eine Maschine handelt. Seit den 1950ern ist es schon vielen Chatbots (= auf Konversationen spezialisierte Programme) gelungen, diesen Test zu bestehen, meistens in dem die Entwickler spezialisierte Taktiken anwendeten: Eines der ersten „intelligenten“ Programme hieß ELIZA und ahmte einen Psychologen nach, in dem es die Fragen immer wieder zurück auf den Fragestellenden reflektierte und ihn somit ermutigte, mehr zu reden. Ein neueres Chatbots Beispiel namens „Eugene Goostman“ wurde als 13-jähriger ukrainischer Junge konzipiert, sodass die Testuser seine Sprachfehler und unlogische Grammatik als kulturelle Unterschiede deuteten. Wieder andere Beispiele nutzen Deep Learning und große Datenmengen, um das Führen von Konversationen selbst zu lernen [9].

Allerdings wird auch die Relevanz des Turing Testes immer mehr in Frage gestellt: Sollte man die Intelligenz von Maschinen am Vorbild des Menschen messen, und besteht diese Intelligenz nur in der Fähigkeit Gespräche zu führen? Ziel der meisten KI-Entwickler ist es nicht, den Menschen perfekt zu imitieren, sondern dort zu unterstützen, wo seine Fähigkeiten nicht mehr ausreichen [10].

3.3. Starke & schwache KI

Es ist auch wichtig zu erwähnen, dass es sich bei allen bis heute entwickelten AI-Modellen um schwache KI handelt, das heißt, dass die Modelle zwar in einem, kleinen, klar definierten Bereich „intelligentes“ Verhalten zeigen, in anderen Anwendungsgebieten jedoch komplett scheitern. Eine künstliche Intelligenz, die darauf trainiert wurde, Bilder von Katzen und Hunden zu unterscheiden, kann das zwar ganz gut, ist aber verwirrt, wenn es ein Bild von

einer Kuh sieht, genau so wenig kann sie Karten spielen oder Texte schreiben. Starke KI hingegen würde aus eigenem Antrieb, genauso bzw. besser handeln und denken als ein Mensch. Trotz intensiver Forschung gibt es auf diesem Gebiet allerdings noch keine Anwendungsbeispiele [11], [12] .

Ein weit verbreiteter Irrglaube ist auch, dass AI-Entwickler künstliche Gehirne entwickeln, denn in der Praxis handelt es sich bei KI genau so wenig um Intelligenz wie es sich bei Flugzeugen um künstliche Vögel handelt. Am Ende kann alles auf die dahinterstehende Mathematik heruntergebrochen werden, denn selbst bei den kompliziertesten Modellen handelt es sich nur um schlichte mathematische Anweisungen.

4. Anwendungsmöglichkeiten

Die Anwendungsmöglichkeiten von Deep Learning sind sehr breit gefächert und reichen von einfachen Apps in unseren Smartphones bis zu riesigen industriellen Algorithmen, die Millionen von Usern analysieren. Die wichtigsten Beispiele möchte ich hier kurz vorstellen.

4.1. Text

Ein Beispiel dafür ist der Tone Analyzer von IBM. Dieses Programm ist in der Lage, die verschiedenen Emotionen des Autors herauszufinden [13] (siehe Abbildung 2, verfügbar unter: tone-analyzer-demo.ng.bluemix.net). Neuronale Netzwerke sind nicht nur in der Lage, Texte zu verstehen, sie können auch Texte selber schreiben, die zum Teil so echt wirken, wie von Menschen geschriebene [14].

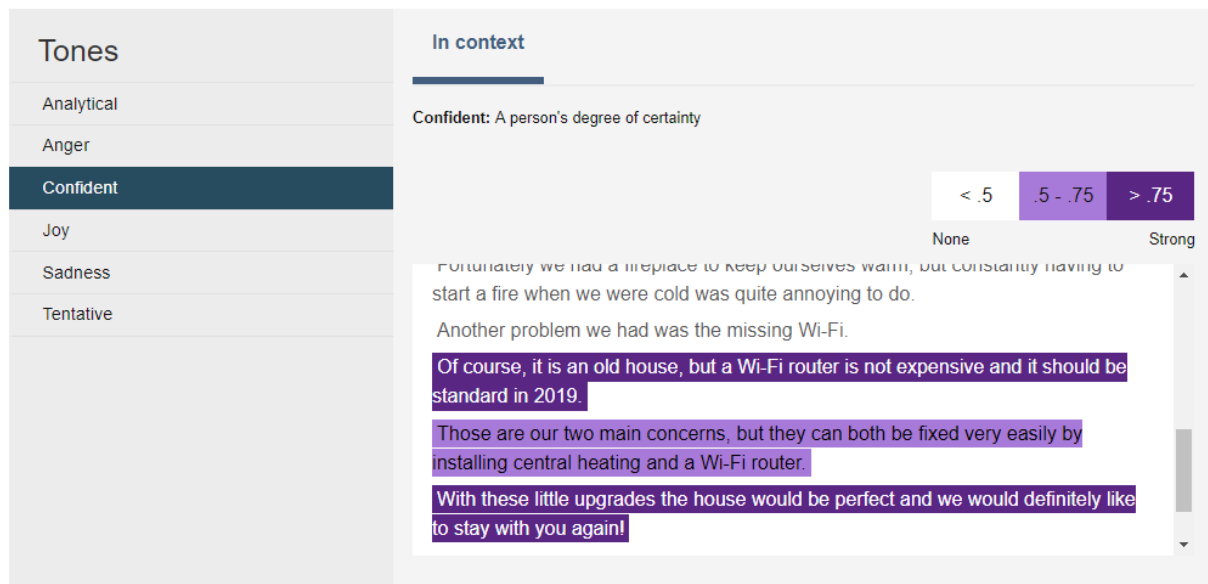


Abbildung 2: Beispiel aus einem Beschwerde-E-Mail (Quelle: <https://tone-analyzer-demo.ng.bluemix.net/>)

4.2. Bilder

Durch die Tiefe von neuronalen Netzen stellen auch Bilder kein Problem dar. Dank neuer Algorithmen können sogar verschiedene Bildinhalte exakt und genau erkannt werden [15], [16]. Ein Beispiel dafür wäre FaceID, das sich in verschiedenen Smartphones findet. Dies ist eine Technik, bei der man sein Handy mit seinem Gesicht entsperren kann [17]. Dasselbe Prinzip kann aber natürlich auch auf Überwachungskameras oder auf Kameras in Autos angewendet werden, um Fußgänger, Verkehrszeichen oder andere Fahrzeuge zu erfassen, siehe Abbildung 3.

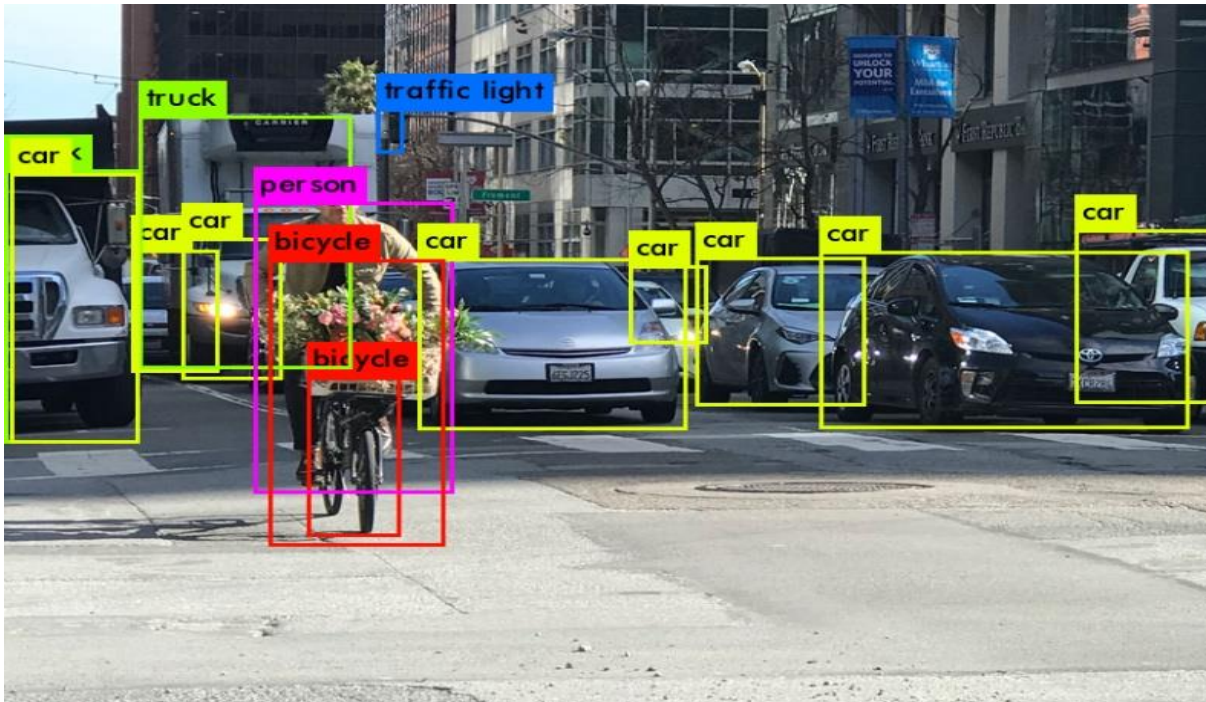


Abbildung 3: Object-Detection mit Neuronalen Netzen (Quelle: <https://mc.ai/part-2-fast-r-cnn-object-detection/>)

Generative Adversarial Networks, kurz GAN, eine Sonderform von Deep Learning Modellen, sind in der Lage, Zeichnungen, Icons, oder sogar ganze Bilder oder Fotos selbst zu erzeugen: Der Grafikkarten Hersteller NVIDIA hat ein Modell erstellt, das selbstständig Bilder von Menschen generiert, die es gar nicht gibt. Dasselbe Prinzip wurde auch angewandt, um Bilder von Katzen, Autos und Schlafzimmern zu erzeugen. Unter <https://thispersondoesnotexist.com/> kann man das auch selbst ausprobieren [18].

Ende 2018 wurde ein Prototyp namens CycleGAN veröffentlicht, der in der Lage ist, Merkmale von einem Bild auf ein anderes zu übertragen. Dabei kann man beispielsweise ein Pferd so aussehen lassen, als ob es ein Zebra wäre, siehe Abbildung 4 [19], [20].

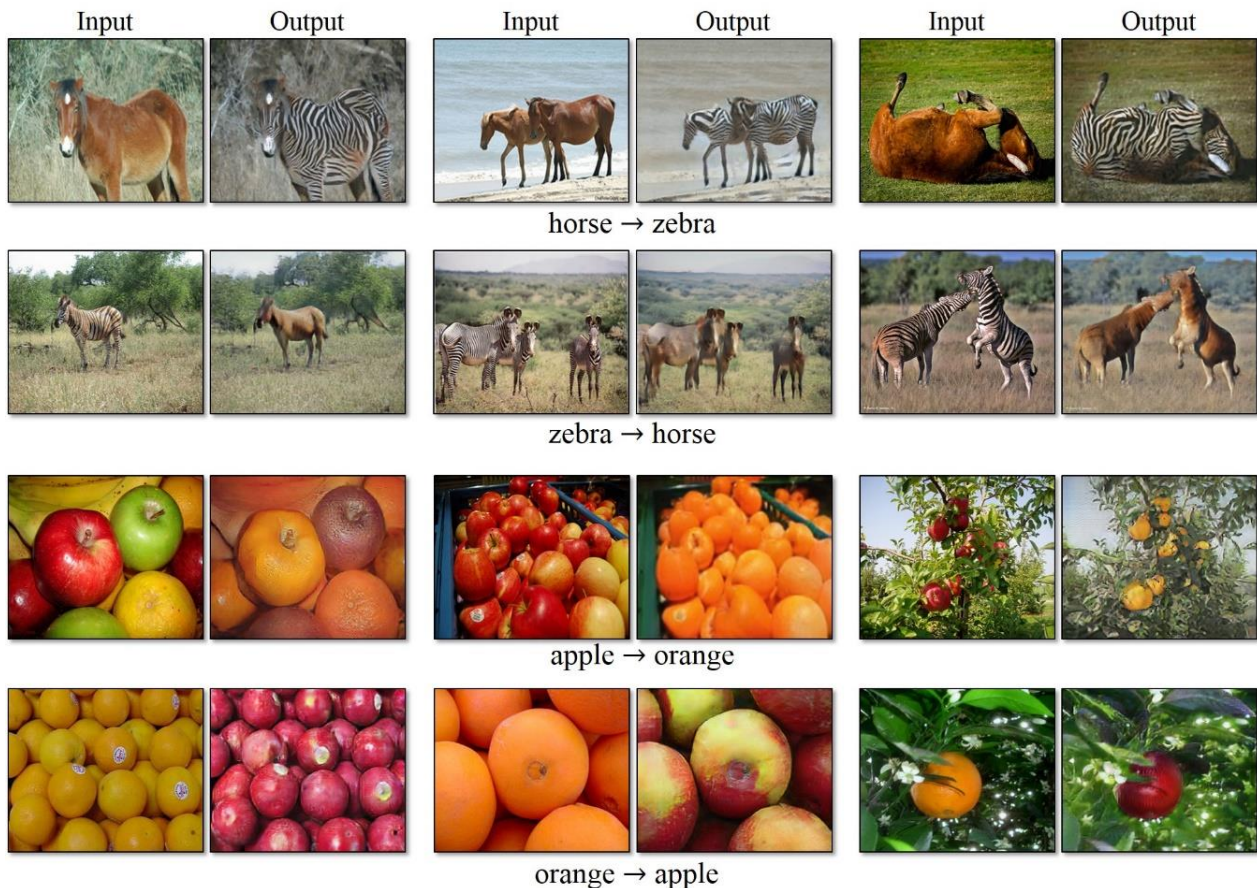


Abbildung 4: Beispiel CycleGAN (Quelle: <https://arxiv.org/pdf/1703.10593.pdf>)

4.3. Audio

Speech Recognition ist ein Untergebiet von Natural Language Processing (NLP), einer Schnittstelle zwischen Sprachwissenschaften und Informatik, die es sich zur Aufgabe gemacht hat, Computern unsere Sprache näher zu bringen. Auch hier erweisen sich neuronale Netze als sehr nützlich. Wie der Name schon sagt, fokussiert sich Speech Recognition auf gesprochene Sprache [21] und viele Anwendungsmöglichkeiten haben sich schon längst in unserem Alltag etabliert: Fast jeder hat schon einmal mit Siri, Alexa und Co gesprochen, aber auch YouTube nutzt diese Technologie, um automatisch Untertitel zu erzeugen [22]. Chatbots sind auch Teil von NLP, das sind Programme, die genauso mit Menschen chatten, als ob es sich dabei um echte Menschen handelt, oft werden sie als Kundenservice auf Webseiten eingesetzt, denn im Gegensatz zu menschlichen Angestellten sind sie rund um die Uhr wach, müssen nicht bezahlt werden und geben meist mindestens so kompetente Antworten wie menschliche Ansprechpartner [21]. Des Weiteren haben neuronale Netze auch gelernt, die menschliche Stimme zu imitieren, um so geschriebene Texte laut vorzulesen. Google bietet zum Beispiel einen online Text-to-Speech Service mit über 100 Stimmen in 20 verschiedenen Sprachen an. Zwar klingt die Ausgabe noch leicht Roboter-ähnlich, wenn man aber bedenkt, dass der Dienst erst wenige Jahre alt ist, ist das doch sehr eindrucksvoll [23], [24].

4.4. Video

Videos sind nichts anderes als eine Folge von Bildern mit hinterlegtem Ton, deshalb kann auch hier Deep Learning gut eingesetzt werden. Mit den oben erwähnten CycleGAN wurden zum Beispiel auch schon Bilder editiert [25]. Ein noch viel beeindruckenderes Beispiel liefern Forscher der University of Washington: Ihnen ist es gelungen, die Stimme und Mundbewegungen von Barak Obama komplett zu synthetisieren, dann zu verändern und wieder zurück auf Obama zu projizieren, um ihn so Dinge sagen zu lassen, die er nie gesagt hat. [26] Das Video ist auf YouTube verfügbar und wirkt täuschend echt, was natürlich auch zu einem sehr großen Problem werden kann. Solche gefälschten Videos nennt man auch Deep Fake Videos, und mit dem schnellen Fortschritt der AI Technologie werden auch sie immer besser, doch zum Glück gibt es auch schon Firmen, die sich darauf spezialisiert haben, solche Videos zu erkennen, und eine Art „Antivirenschutz für Deep Fakes“ entwickelt haben, der natürlich auch wieder mit Deep Learning arbeitet [27]. Die Technik kann nicht nur für böartige Zwecke missbraucht werden, sondern bietet auch ein riesiges Potential für die Filmindustrie [28], [29].

4.5. Analyse

Deep Learning eignet sich auch hervorragend für die Analyse von riesigen Datenmengen. YouTube nutzt diese Technik zum Beispiel, um Usern möglichst interessante Videos vorzuschlagen, genauso wie Netflix und Spotify [30], [31]. Im Marketing findet sie ebenfalls ihren Nutzen, um schneller und zielgerechter zu werben [32]. Die meisten der hier vorgestellten Technologien stecken noch in den Kinderschuhen. Die Forschung macht aufgrund der beinahe unbegrenzten Möglichkeiten rasante Fortschritte und in der Zukunft kommen wahrscheinlich noch viele große Innovationen auf uns zu.

5. Inspiriert vom menschlichen Gehirn

Bevor wir auf die technischen Feinheiten von künstlichen neuronalen Netzen (KNN) eingehen, will ich in diesem Kapitel den Vergleich zum intelligentesten Organ überhaupt herstellen: dem menschlichen Gehirn. Dort findet man bis zu 86 Milliarden Neuronen, also Nervenzellen, die über Axone/Dendriten miteinander verbunden sind und über die elektrische Impulse, sogenannte Aktionspotentiale, mit bis zu 120m/s gesendet werden. Enthält eine Zelle ein ausreichend starkes Aktionspotential oder ist die Summe der eintreffenden Aktionspotentiale groß genug, dann wird es weitergeleitet. Auf diese Weise sind fast alle Neuronen im Gehirn miteinander verbunden, und genau dieses Prinzip wird auch auf KNNs angewendet [33]. Beim Prozess des Lernens werden die einzelnen Nervenzellen stärker miteinander verbunden, sodass Aktionspotentiale leichter übertragen werden.

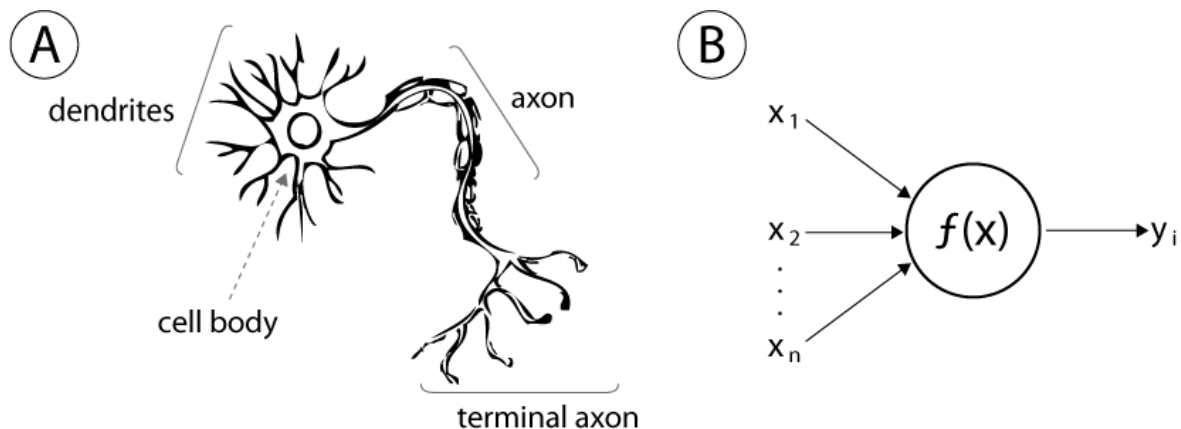


Abbildung 5: Vergleich Nervenzellen/künstliche Neuronen (Quelle: https://www.researchgate.net/figure/Model-of-a-human-brain-cell-neuron-A-and-an-artificial-neuron-B-used-in-Artificial_fig21_314761683)

Genauso werden die Verbindungen zwischen künstlichen Neuronen verstellt, damit die richtigen Neuronen aktiviert werden. In den Neuronen finden ebenfalls Rechengvorgänge statt, und sogenannte Weights, also auf Deutsch Gewichte, bilden als Pendant zu Axonen und Dendriten die Verbindungen untereinander. Wann ein Aktionspotential ausgelöst werden soll, bestimmen Aktivierungsfunktionen.

6. Allgemein technische Herangehensweise

Um ein erfolgreiches neuronales Netzwerk zu erstellen benötigt man Hintergrundwissen über die einzelnen Komponenten, wie zum Beispiel über die verschiedenen Arten von Schichten oder Aktivierungsfunktionen, die ich in diesem Kapitel näher erklären werde. Beginnen möchte ich daher mit der grundlegenden Herangehensweise: Um Deep Learning zu perfektionieren, muss man sich mit der komplexen Mathematik dahinter beschäftigen. Wenn man aber nur einfache Netze erstellen will und einen schnellen Einstieg möchte, reicht es aber auch aus, nur die grundlegenden Funktionsmechanismen zu kennen.

6.1. *Hardware*

Hardwaretechnisch genügt schon ein mittelpreisiger Laptop, um einfache Netze zu erstellen, bevorzugt mit Grafikkarte (engl.: Graphical Processing Unit = GPU). Eine Besonderheit des Deep Learning ist das „Trainieren“ des Netzes, dabei analysiert es die vorgegebenen Daten, versucht komplexe Muster zu erkennen und lernt so Zusammenhänge zu „verstehen“. Dieser Vorgang kann, abhängig von der Größe des Netzes und der Hardware, von einigen Sekunden bis hin zu mehreren Tagen dauern. Vor allem auf einem normalen Prozessor (engl.: Central Processing Unit = CPU) mit nur wenigen Kernen kann dieser Prozess sehr zeitintensiv sein, GPUs die mehrere tausend Kerne besitzen, können diese Aufgabe sehr gut parallelisieren und Netze so bis zu 10 mal schneller trainieren [34], [35]. Falls man keine GPU lokal auf seinem Rechner eingebaut hat, bietet es sich auch an, das Modell online in einer Cloud zu trainieren, um so auf große Rechenleistung zugreifen zu können. [36]

6.2. *Coding*

Die beliebteste Programmiersprache für diese Anwendung ist Python, eine Sprache, die in den letzten Jahren immer beliebter wurde und sich durch ihre gute Lesbarkeit und einfache Schreibweise auszeichnet. Dabei werden sogenannte Libraries, also Erweiterungen, genutzt, die die benötigten Funktionen zur Verfügung stellen. Drei der am häufigsten verwendeten Libraries sind: Tensorflow, Keras und PyTorch. Tensorflow wurde von Google entwickelt und für jeden frei zur Verfügung gestellt und ermöglicht die Herstellung von komplexen Netzen. Eingesetzt wird die Library unter anderem auch in Googles eigenen Produkten. Keras ist zwar ein eigenes Modul, baut aber auf Tensorflow auf und vereinfacht die Funktionen, um möglichst benutzerfreundlich zu sein und schnelles experimentieren zu ermöglichen, während PyTorch von Facebook entwickelt wurde und ebenfalls gute Performance bei hoher Flexibilität ermöglicht. Alle drei Module stehen gratis zur Verfügung und sind Open Source, das heißt der Quellcode steht online bereit und kann von jedem gelesen und verändert werden [37]–[39].

6.3. *Daten*

Essenziell für die Qualität des Deep Learning Modells ist auch die Qualität und die Quantität der Input-Daten. Wichtig ist, dass alle Daten relevant sind, wenn man also ein Netz trainieren möchte, dass Gesichtsausdrücke von Menschen erkennen soll, macht es keinen Sinn Bilder von Hunden oder Katzen in den Trainingsdaten zu haben. Besser ist es, wenn man Fotos von menschlichen Gesichtern aus verschiedenen Blickwinkeln, bei unterschiedlichen

Belichtungsverhältnissen und mit möglichst gleichmäßig aufgeteilten Bildkategorien inkludiert, denn so sorgt die Varianz dafür, dass das Netz lernt, relevantere Features zu extrahieren und verhindert das Auswendiglernen von einzelnen, unwichtigen Merkmalen [40]. Bei Deep Learning gilt auch: je mehr desto besser: Um Bildinhalte gut analysieren zu können, benötigt man mindestens einige hundert bis tausend Trainingsbeispiele. Bevor die Daten benutzt werden können, müssen sie vorher auch aufbereitet werden. Im Fall von Bildern muss meist die Auflösung auf dasselbe Format geändert werden, wobei man dabei achten muss, dass möglichst wenig verzerrt wird, außerdem muss jedes Bild in einen Array (eine Liste von Werten) umgewandelt werden. Oft fehlen im Datensatz auch einige Werte, die man richtig ersetzen muss, um spätere Fehler zu verhindern. Zum Beispiel kann man bei Tabellen mit numerischen Werten die Lücken mit dem Median oder dem Durchschnitt ausfüllen [41], [42].

Meistens müssen die Daten auch noch richtig kategorisiert werden, damit sie korrekt gelernt werden können. Bevor sie verwendet werden können, müssen die Daten noch in Trainings- und Testdaten aufgeteilt werden, damit man am Ende die Qualität des Netzes verlässlich prüfen kann. Oft wird nach dem Verhältnis 80 % Trainingsdaten / 20 % Testdaten geteilt. Unterließe man diese Unterteilung und würde die Trainingsdaten für die Beurteilung des Modells verwenden, würde man einen verfälschten Eindruck von der Leistung des Netzes bekommen. Man könnte das mit einer Mathematik Schularbeit vergleichen, die genau dieselben Aufgaben enthält, wie die, die in der Schule durchgenommen wurden: Die SchülerInnen würden besser abschneiden, aber nicht, weil sie den Stoff besser verstanden haben, sondern weil sie genau dieselben Aufgaben schon einmal gesehen haben, und genauso verhält es sich auch mit neuronalen Netzen [43].

7. Aufbau von neuronalen Netzen

Im menschlichen Gehirn gibt es ca. 10 Milliarden unterschiedliche Neuronen, die untereinander stark vernetzt sind und parallel arbeiten und damit die Grundlage für unser Nervensystem bilden. Von dieser Struktur sind neuronale Netze (NN) sehr stark inspiriert, in diesem Kapitel geht es daher um den grundlegenden technischen Aufbau von solchen Netzen. Es gibt verschieden Typen von Netzen, in meiner Arbeit konzentriere ich mich hauptsächlich auf Convolutional Neuronal Networks (CNNs), die vor allem für die Bildanalyse prädestiniert sind, sich aber auch in anderen Bereichen, wie der Textverarbeitung, als nützlich erweisen [45].

7.1. Aufbau von neuronalen Netzen, Kurzversion

Bevor ich die genaue Funktionsweise der einzelnen Komponenten erkläre, möchte ich kurz die Grundlagen der NN erläutern:

Ein neuronales Netz besteht aus einigen Schichten (Layers) und unzähligen Neuronen. In der ersten Schicht nimmt je ein Neuron die Pixelwerte des Inputbildes an und reicht diese zu den Neuronen des nächsten Layers weiter. Je nach dem um welche Schicht es sich dabei handelt, führen diese unterschiedliche mathematische Operationen durch, wobei bei jeder Schicht der Abstraktheitsgrad für den Computer zu- und für uns Menschen abnimmt. In der letzten Schicht gibt es für jede mögliche Ausgangskategorie ein dazugehöriges Neuron. Wird dieses aktiviert dann gehört das Inputbild zu jener Output Kategorie. Um dabei möglichst gute Ergebnisse zu erzielen, spielen viele verschiedenen Parameter eine Rolle: Die Anzahl und Zusammensetzung der unterschiedlichen Arten von Layers, die Wahl der richtigen Aktivierungsfunktion (siehe Kapitel Aktivierungsfunktionen), Hyperparameter und vieles mehr. Wichtig ist es auch zu erwähnen, dass es keinen festen Bauplan gibt, nach dem jedes Netz aufgebaut ist. Zwar gibt es allgemeine Tipps und Vorgehensweisen, das Programmieren besteht zu einem großen Teil aber aus experimentieren und herumprobieren.

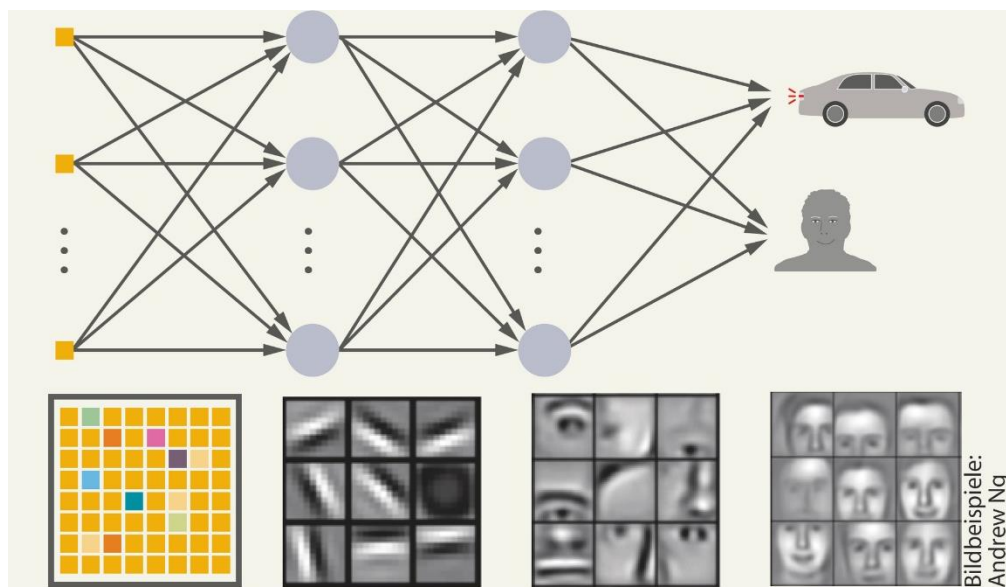


Abbildung 6: Vereinfachter Aufbau eines neuronalen Netzes (Quelle: <https://www.heise.de/select/ct/2016/6/1458191210995647>)

7.2. Aufbau von neuronalen Netzen, Langversion

Stellen wir uns vor, wir wollen ein Netz erstellen, das Bilder von handgeschriebenen Zahlen zwischen 0 und 9, mit einer Auflösung von 28x28 Pixeln, kategorisieren kann. Alle Bilder sind in schwarz-weiß und sehen so aus:

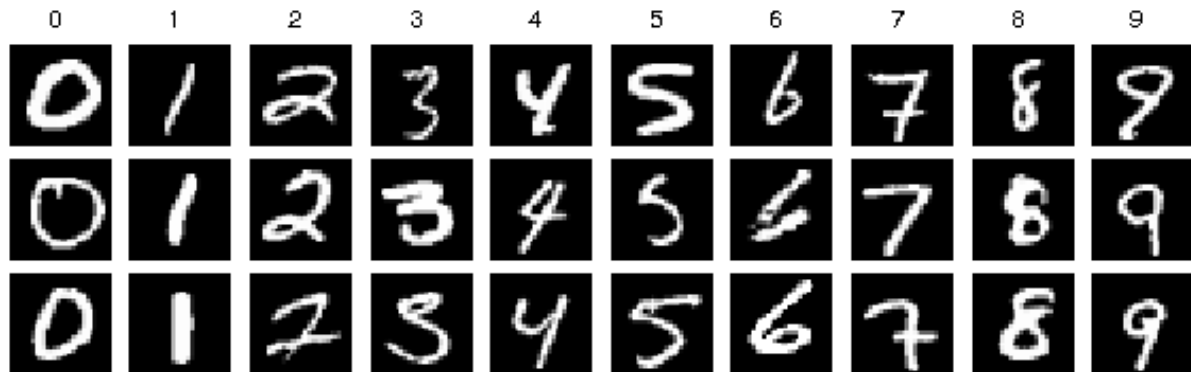


Abbildung 7: Handgeschriebene Zahlen 0-9 (Quelle: <http://programmersought.com/article/74931184251/>)

Jedes Bild kann (und muss für den Computer) als Matrix dargestellt werden, bei der jeder Wert der Helligkeit des Pixels entspricht, also 0 bei schwarz und 255 für weiß. Die Matrix für eine 5 würde dementsprechend so aussehen (Abbildung 8):

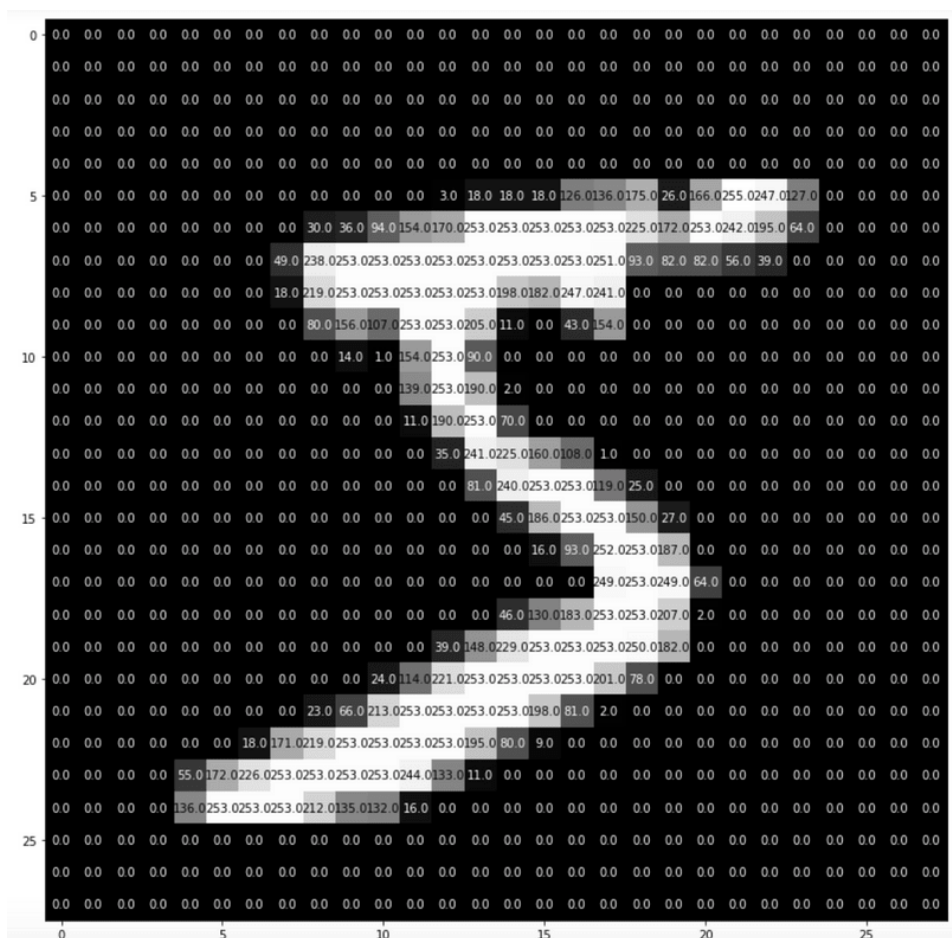


Abbildung 8: Pixelmatrix einer Fünf (Quelle: <https://mc.ai/deciphering-handwritten-numbers-with-a-neural-network/>)

Wir wissen nun also, dass wir 784 ($=28 \times 28$) Eingabewerte haben, jeder davon zwischen 0 und 255, und weil wir 10 Klassifikationsmöglichkeiten haben, gibt es 10 mögliche Ausgabewerte von 0-9.

7.3. Layers

Diese 784 Input-Werte werden dementsprechend auch als Input-Layer bezeichnet, in unserem Beispiel würde sie 784 Neuronen umfassen, diese Neuronen sind nichts anderes als Elemente, die mathematische Funktionen durchführen und das Ergebnis speichern. In der ersten Schicht führen sie aber noch keine Rechnungen durch, sondern speichern einfach die Input-Werte. Indirekt sind die Neuronen der Input-Layer auch mit den (in unserem Fall) 10 Neuronen der Output-Layer verbunden, dazwischen liegen aber Elemente, die den Erfolg von neuronalen Netzen ausmachen: die sogenannten Hidden-Layers, also versteckte Schichten [46].

7.4. Warum mehrere Schichten?

Bevor wir uns die Hidden-Layers genauer ansehen, möchte ich darauf eingehen, warum man überhaupt mehrere Schichten benutzt und nicht einfach die Input-Schicht direkt mit der Ausgabeschicht verbindet. Das menschliche Sehen funktioniert, vereinfacht gesagt so, dass wir zuerst allgemeine Merkmale, wie Kanten und Ecken, dann Details wie zum Beispiel Augen oder Kreise und schließlich komplexe Muster wie Gesichter oder Buchstaben oder Katzen erkennen. Künstliche, neuronale Netze bedienen sich des selben Ansatzes: Der Input besteht nur aus vielen Pixel-Werten, die auf jeder Schicht immer weiter verallgemeinert, kombiniert und abstrahiert werden, bis man das Endergebnis in (für den Computer) vollkommen abstrakte Kategorien einteilen kann [46], [47].

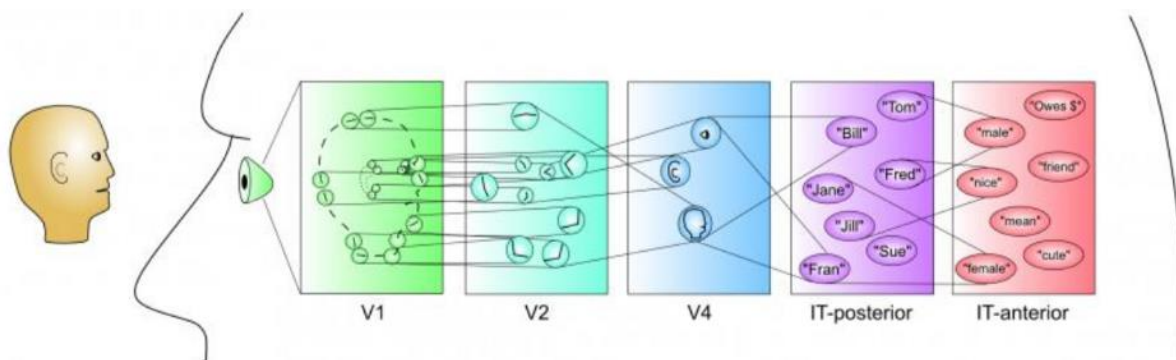


Abbildung 9: Abstraktion von Bildern im menschlichen Gehirn (Quelle: <https://hackernoon.com/mit-6-s094-deep-learning-for-self-driving-cars-2018-lecture-4-notes-computer-vision-f591f14b3b99>)

7.5. Convolutional Layers

Damit wir den Input sinnvoll abstrahieren können, benötigen wir eine Möglichkeit, grundlegende Muster zu erkennen, eine Art Filter, der horizontale, vertikale oder schräge Striche erkennen kann. Solche Filter, die auch Kernels genannt werden [48], sind Matrizen die 3×3 , 5×5 oder ähnlich viele Pixel groß sind, und nacheinander mit den Pixeln des Input Bildes multipliziert werden. Falls der analysierte Teil des Bildes das Muster des Filters enthält, ist die Summe der Ergebnismatrix dementsprechend hoch, wenn nicht, dann ist die

Summe aller Werte in der Ergebnismatrix sehr klein oder 0. Sehen wir uns ein Beispiel an (Abbildung 10):

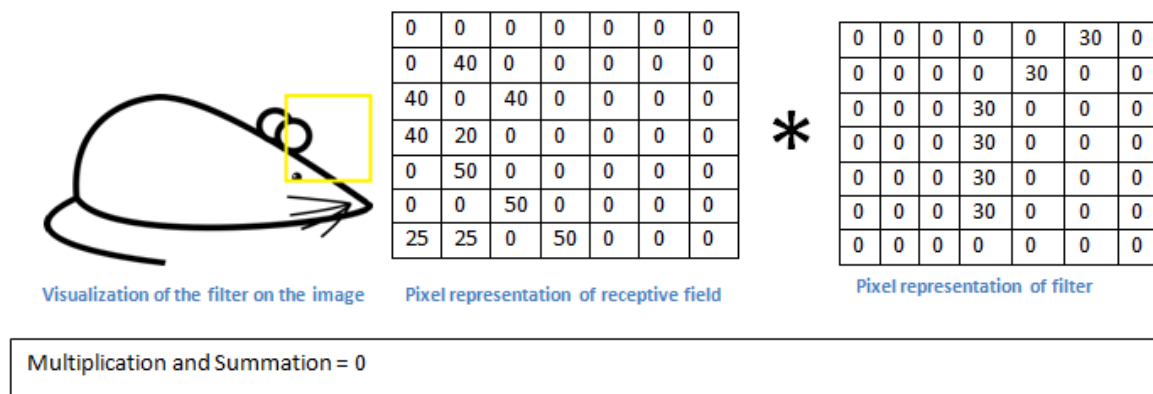


Abbildung 10: Convolution Matrix 1 (Quelle: <https://lipyeow.github.io/ics491f17/morea/deepnn/ImageClassification-CNN.pdf>)

Hier haben wir ein Bild von einer Maus, die Matrix links enthält die Pixelwerte des Bildes im Bereich des gelben Rechtecks, das ist der Bereich auf den der 7x7 Pixel große Filter (Matrix rechts) angewendet wird. Der Filter erkennt leichte Rechtskurven, man sieht, dass fast alle Werte der Filtermatrix 0 betragen, außer die, die das zu erkennende Muster darstellen. Multipliziert man beide Matrizen und summiert das Ergebnis, erhält man 0, weil in dem gelb umrahmten Bereich keine Rechtskurve zu sehen ist.

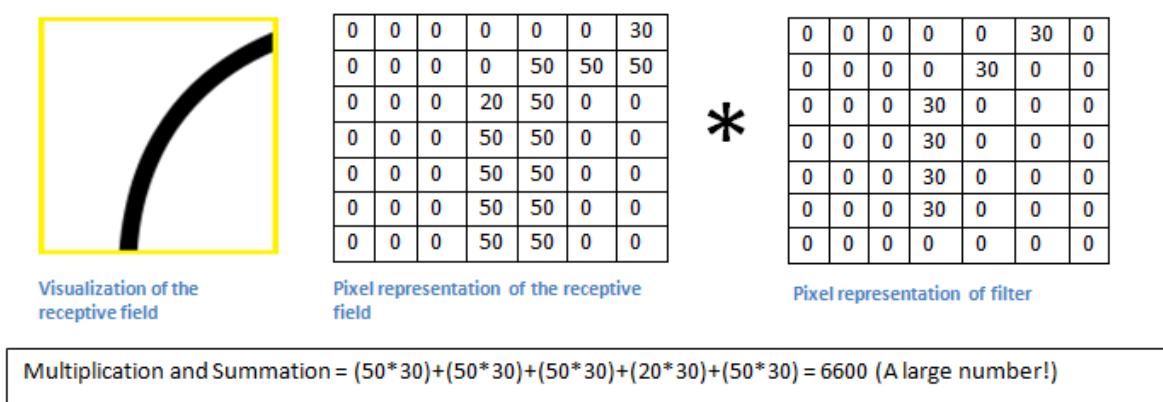


Abbildung 11: Convolution Matrix 2 (Quelle: <https://lipyeow.github.io/ics491f17/morea/deepnn/ImageClassification-CNN.pdf>)

Verschiebt man den gelben Bereich aber auf den unteren Rücken der Maus (Abbildung 11) und wiederholt dieselbe Prozedur, erhält man eine sehr hohe Zahl, 6600, weil dieser Bereich dem Muster des Filters entspricht. Denselben Filter könnte man auch bei unserem Beispiel mit handgeschriebenen Zahlen verwenden: Bei Ziffern wie 8, 9, 6, 0 oder 2 wäre der Wert im der linken oberen Teil sehr groß, dafür aber klein in anderen Bereichen, und bei Ziffern wie 4 oder 1 würde er überhaupt nicht reagieren [49], [50]. Natürlich kann man mit einem 7x7 Kernel noch viele weitere Muster repräsentieren, und wenn man dieses gelbe Rechteck und damit die Filter über das ganze Bild zieht, erhält man eine gute Darstellung davon, in welchen Bildbereichen welche Muster vorhanden sind. Weil in diesem Convolutional-Layer Werte von $7 \times 7 = 49$ Neuronen zusammengefasst werden, hat er weniger Neuronen als die Schicht davor, dafür ist aber auch der Informationsgehalt höher [46], [47].

7.6. Pooling Layers

Ein Problem von Convolutional Layers ist, dass sie sehr sensibel auf die Positionen von verschiedenen Features (also Merkmalen) reagieren. Eine gute Methode um das Netz hierfür robuster machen sind Pooling Layers, die in zwei Unterkategorien eingeteilt werden: Max Pooling und Average Pooling. Sie funktionieren folgendermaßen: In einem meist 2x2 großen Filterbereich werden alle Werte zum Maximalwert (bei Max Pooling) bzw. zum Durchschnittswert (bei Average Pooling) zusammengefasst. Das hat zur Folge, dass die Höhe und die Breite jeweils um die Hälfte reduziert werden, d.h. ein 18x18 = 324 Pixel großes Bild wird zu einem 9x9 = 81 Pixel großen Bild. Ähnlich wie beim Convolutional Layer werden also größere Bereiche zu einem kleineren zusammengefasst, was den Vorteil bringt, dass die Convolutional Layers des Netzes Muster immer noch erkennen, wenn sie ein paar Pixel verschoben sind und das Netz damit robuster wird. In den meisten konvolutionellen Netzwerken folgt daher ein Pooling nach einer Convolutional Layer [51], [52].

7.7. Flattening & Fully-Connected Layers

Flattening und Fully-Connected (FC) Layers kommen hauptsächlich im letzten Teil eines neuronalen Netzes vor. Flattening Layers wandeln den mehrdimensionalen Output von z.B. Pooling Layers in einen langen, eindimensionalen Vektor um, der von FC Layers weiterverarbeitet werden kann. Diese Schichten, bei denen, wie der Name schon sagt, alle Neuronen miteinander verbunden sind, helfen dem Netzwerk, die nichtlinearen Zusammenhänge zwischen den Mustern zu lernen [53], [54].

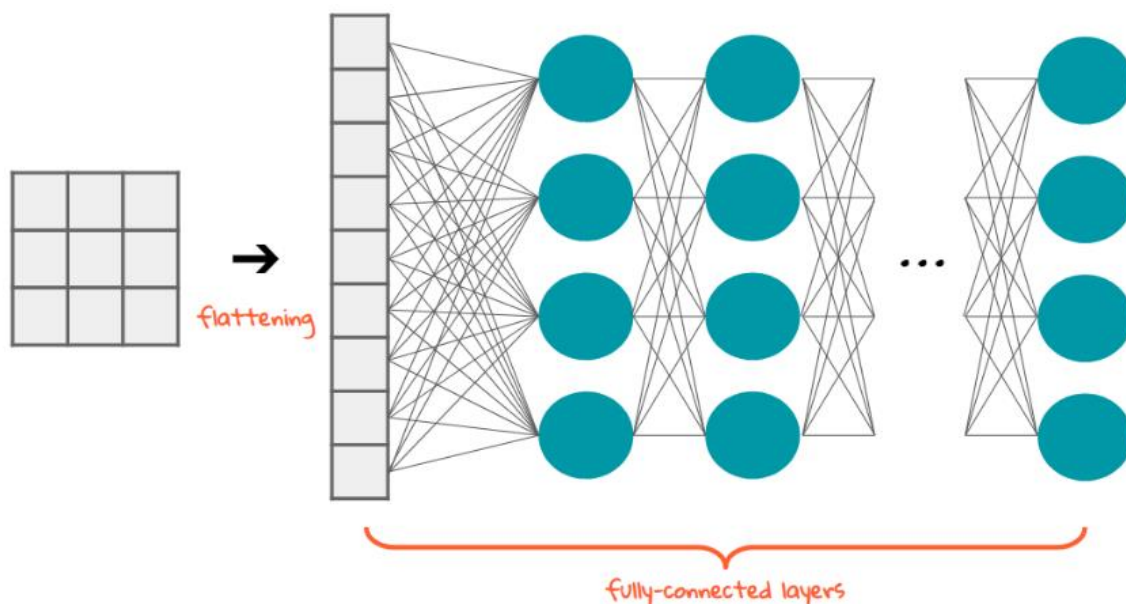


Abbildung 12: Flattening + FC Layer (Quelle: <https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480>)

7.8. Aktivierungsfunktionen

Jedes Neuron enthält eine Aktivierungsfunktion und diese kann eine riesige Auswirkung auf das Verhalten des Netzes haben. Rein theoretisch könnte ein Neuron Werte von $-\infty$ bis $+\infty$ haben, und eine Aktivierungsfunktion bestimmt, wie der Name schon sagt, wann dieses Neuron aktiviert wird. Ein einfaches Beispiel dafür wäre eine Stufenfunktion, die das Neuron aktiviert (also den Wert 1 gibt), wenn der Inputwert über einem bestimmten Schwellwert, z.B. 0, liegt, und nicht aktiviert (also den Wert 0 gibt), wenn sich der Wert darunter befindet. Als Graph würde das folgendermaßen aussehen (Abbildung 13):

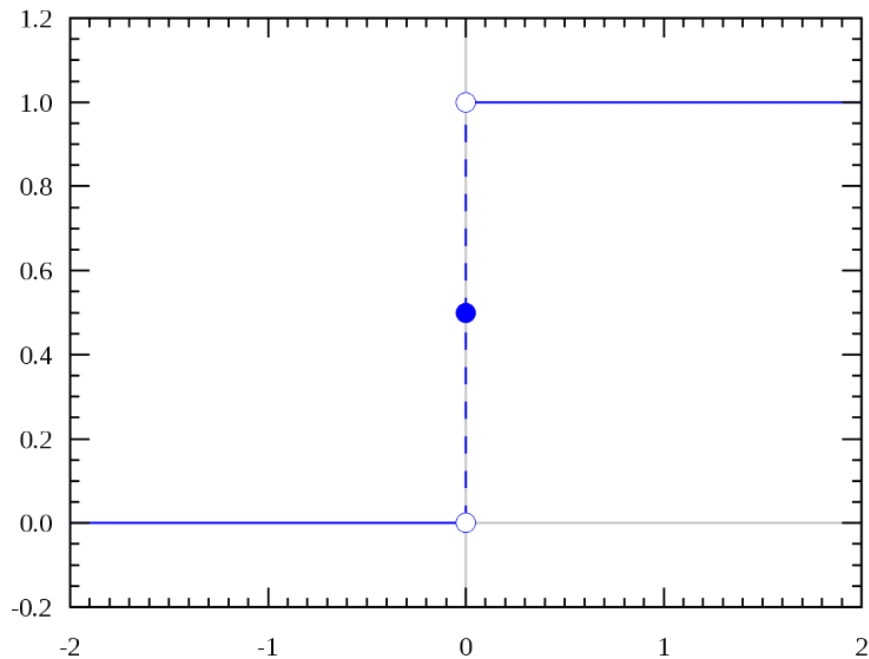


Abbildung 13: Stufenfunktion (Quelle:
<https://sites.google.com/site/calcoloamo1976/assignments>)

So eine Funktion hat aber nur 2 mögliche Ausgabewerte, aktiviert oder nicht aktiviert, 0 oder 1, und das kann einige Probleme mit sich bringen, deswegen wird diese Variante auch nicht eingesetzt [55]. ReLU (Rectified Linear Unit) ist ebenfalls eine Aktivierungsfunktion und wird sehr häufig verwendet und funktioniert folgendermaßen: Falls der Input größer als 0 ist, bleibt er unverändert, wenn dagegen kleiner als 0 ist, wird er auf 0 gesetzt. Daneben gibt es noch viele weitere Aktivierungsfunktionen, aber ReLU bietet einige Vorteile: Es benötigt nicht so viel Rechenleistung wie andere Funktionen und sorgt dadurch dafür, dass das Netz schneller trainiert, Probleme, die bei anderen Funktionen häufig auftreten, treten bei ReLU nicht auf, außerdem hilft das lineare Verhalten der Funktion, das Netzwerk besser zu verstehen und zu optimieren. Generell sollte man ReLU als Standard-Aktivierungsfunktion für Hidden Layers verwenden. [55] Für die letzte Schicht ist sie jedoch ungeeignet, hier wird meist die sogenannte Softmax Funktion verwendet, die alle Werte in verschiedenen Wahrscheinlichkeiten zwischen 0 und 1 aufteilt, deren Summe 1 ergibt. So erhalten wir, wenn wir wieder an unser Zahlenbeispiel denken, für jede Ziffer eine bestimmte Vorhersagewahrscheinlichkeit anstatt schwer verständlichen Zahlen [58].

7.9. Gewichte

Bis jetzt haben wir über Neuronen, Schichten und Aktivierungsfunktionen gesprochen, einen wichtigen Teil haben wir aber noch gar nicht erwähnt: Gewichte, englisch Weights. Gewichte sind die Verbindungen zwischen den einzelnen Neuronen.

Dabei repräsentiert jedes Gewicht die Relevanz des vorhergehenden Neurons, jedes Gewicht besitzt nämlich einen Koeffizienten, mit dem der Wert des Neurons multipliziert wird. Ist dieser Koeffizient hoch, erhöht sich auch der Wert des Neurons, ist er klein, sinkt der Wert des Neurons. So weit so gut, aber wie erkennt das neuronale Netzwerk welche Neuronen wichtig sind und welche nicht? Das passiert während des Trainings, indem die Weights mit komplizierten Algorithmen schrittweise angepasst werden bis das Netz möglichst gute Ergebnisse liefert. Zu Beginn sind die Werte noch komplett zufällig, und so auch die Deutung des Netzes, dazu aber später mehr [46], [59]. Wichtig ist anzumerken, dass es sich bei den Gewichten nicht um Teile der Neuronen an sich handelt, sondern um die Verbindungen zwischen ihnen. Bei der Berechnung des Neuronen Wertes wird außerdem noch ein sogenannter Bias hinzugefügt, der den Wert nach unten oder nach oben verschiebt und oft essenziell für erfolgreiches Lernen ist [60]. Damit ergeben sich folgende Formeln:

$$Y = \sum (weight * input) + bias$$

Y ist also die Summe aller Gewichte multipliziert mit deren Input Werten, plus dem Bias (in Abbildung 14 graphisch dargestellt). Y ist aber noch nicht der endgültige Ausgabewert, man muss noch die Aktivierungsfunktion miteinbeziehen, zum Beispiel ReLU, um diesen zu erhalten, also $ReLU(Y)$.

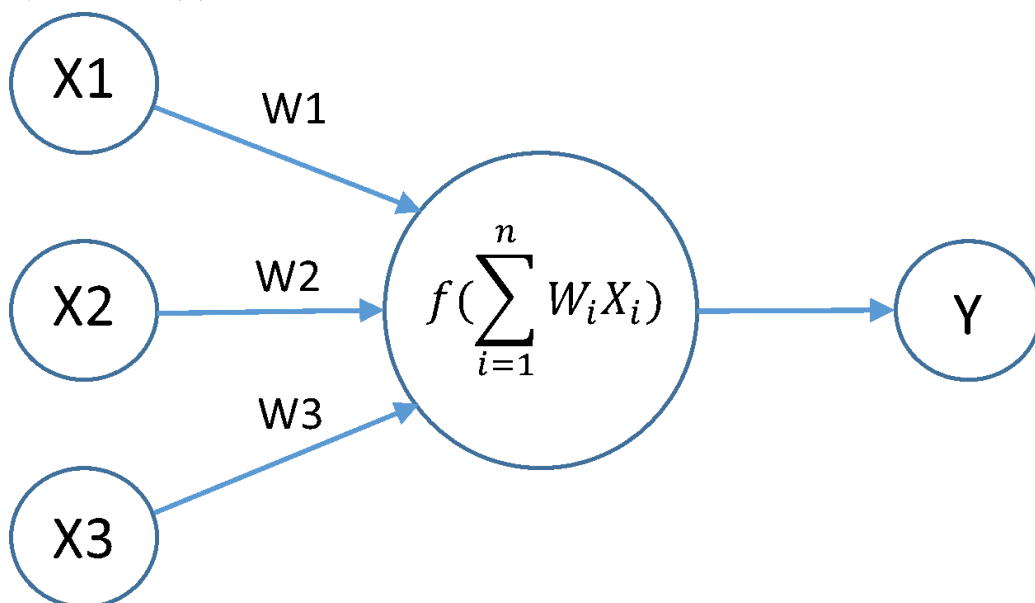


Abbildung 14: Berechnung des Neuronen Wertes (Quelle: <https://medium.com/@abhismatrix/neural-network-model-training-tricks-61254a2a1f6b>)

8. Trainieren von neuronalen Netzen

8.1. *Kurzfassung*

Nach der Erklärung über den Aufbau von NN konzentriere ich mich im folgenden Abschnitt auf das Training von NN, denn ohne Training ist das Netz nutzlos, da wir die Gewichte noch so adjustieren müssen damit die richtigen Neuronen angesprochen werden. Rein theoretisch könnten wir das manuell für jede einzelne Verbindung machen, bei Netzen mit mehreren hunderttausend Neuronen kann das aber anstrengend werden. Hier kommen die Trainingsdaten ins Spiel, wie bereits in vorherigen Abschnitten erwähnt. Dem Netz werden Daten, in unserem Fall Bilder gezeigt, die es analysieren soll. Eine Verlustfunktion (engl. loss function) berechnet, wie weit die Einschätzung des Künstlichen NN von der korrekten Klassifikation entfernt liegt. Das Ziel ist es, den Wert dieser Funktion möglichst nahe an 0 zu bringen. Jedes einzelne Gewicht hat Auswirkungen auf das Ergebnis, und um dieses hochdimensionale Optimierungsproblem zu lösen, wird beispielsweise das sogenannte Gradientenverfahren (engl. Gradient Descent) zusammen mit dem Backpropagation-Algorithmus eingesetzt, der die Gewichte schrittweise so anpasst, sodass die Verlustfunktion minimiert wird. Dieser Vorgang wird mehrfach wiederholt, was viel Rechenleistung und Zeit in Anspruch nehmen kann, und nach mehreren Durchgängen ist das Netz bereit zum Einsatz [61].

8.2. *Die Verlustfunktion*

Jedes Mal, wenn wir unser Netz mit Input füttern, erhalten wir Output. Bei unserem Beispiel mit den handgeschriebenen Zahlen werden also die Neuronen der letzten Schicht aktiviert, jeder mit einem bestimmten Wert. Diese Werte stehen oft für die verschiedenen Wahrscheinlichkeiten. Wenn also das Neuron, das für die Ziffer „3“ steht, den Wert 0.55 annimmt, das, welches die Ziffer „5“ symbolisiert, 0.25 und das für „9“ den Wert 0.1 erhält, dann ergibt sich folgendes: Das Inputbild zeigt mit einer Wahrscheinlichkeit von 55% eine 3, mit 25% eine 5 und mit 10% eine 9. Weil die Wahrscheinlichkeit für eine 3 am höchsten ist, ist diese Ziffer auch die endgültige Antwort des Netzes. Gerade am Beginn des Trainings, an dem die Gewichte nur zufällige Werte haben, liegt das neuronale Netz aber oft daneben, so könnte es sein, dass auf dem Bild statt einer 3 tatsächlich eine 5 zu sehen war. Um etwas „dazuzulernen“, müssen die Neuronen-Werte für 3 und 9 gesenkt werden, während der bei 5 erhöht wird, wobei der von 3 selbstverständlich um einiges stärker gesenkt werden muss als bei 9. Der erste Schritt dieses Lernprozesses besteht darin, zu erkennen, wie groß die Fehler des Netzes tatsächlich sind und diese Aufgabe übernimmt die Verlustfunktion (engl. loss function) [62].

Es gibt einige verschiedene Arten von loss functions, ich will nur die wichtigsten anführen. Eine davon nennt sich Mean Squared Error (MSE), zu Deutsch mittlere quadratische Abweichung. Dabei wird der Vorhersagewert des Netzes vom eigentlich richtigen Wert subtrahiert, zum Quadrat genommen und davon wird schließlich der Durchschnitt berechnet. Hier die Formel [63]:

$$Loss = \frac{1}{n} * \sum_{i=0}^n (Y_i - Y_{pred_i})^2$$

Neben MSE gibt es noch eine Variante, die anstatt mit dem Quadrat mit dem Absolutwert der Vorhersagedifferenz rechnet, was sie robuster gegen Ausreißer macht.

In einem perfekten Netz beträgt der Loss 0, die Vorhersage des Netzes stimmt also mit dem richtigen Ergebnis überein. Bis es aber soweit ist, gibt uns die Verlustfunktion aber Auskunft darüber wie weit wir unsere Parameter noch optimieren müssen. Eine populäre Optimierungsstrategie nennt sich Stochastic Gradient Descent und wird im nächsten Abschnitt behandelt.

8.3. Gradient Descent

Das Justieren aller Parameter stellt ein Optimierungsproblem dar, das es zu lösen gilt, indem wir die Verlustfunktion minimieren. Sehen wir uns dazu an, wie man so ein Problem in 2 Dimensionen lösen würde: Hierfür gibt es rechnerische Methoden, man muss jene Stelle finden, an der die erste Ableitung der Funktion, also die Steigung, 0 entspricht und 2. Ableitung ungleich 0 ist. Alternativ könnte man auch folgende Strategie anwenden: Man nimmt einen Punkt auf der Funktion und verschiebt ihn immer in die Richtung, in die die Steigung abnimmt, wie ein Ball der einen Hang hinabrollt. Irgendwann würde man so auf ein (lokales) Minimum stoßen.

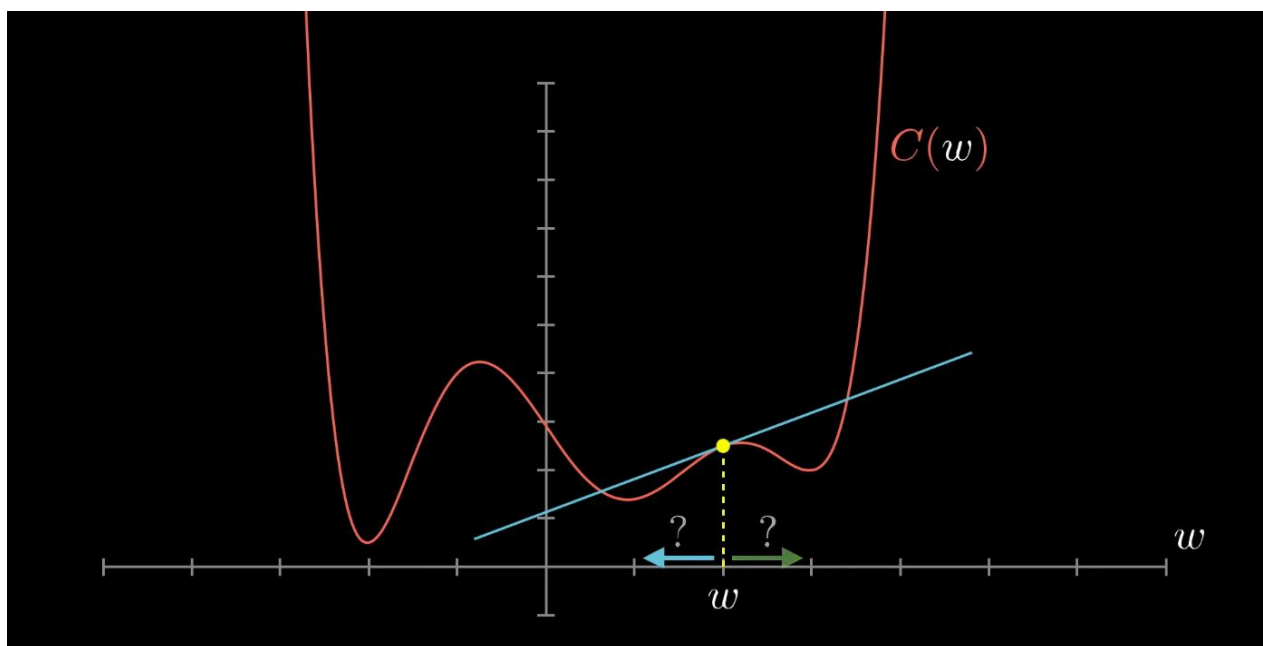


Abbildung 15: Funktion mit mehreren Minimumstellen (Quelle: https://proglib.io/p/neural-network-course/?fbclid=IwAR0tRdSKzsSdhg8zUkZhQ2TknzwHi_10FcccBXLqvB0bbUQCU0erRMu1k5g)

Um auf das globale Minimum zu kommen, und nicht in einem lokalen Minimum stecken zu bleiben, können wir denselben Prozess mit verschiedenen Startwerten wiederholen, sodass auch möglichst jedes Minimum erkannt wird. Genau dasselbe Prinzip kann auch für Funktionen mit 2 Inputvariablen genutzt werden, wie folgende Abbildung (Abbildung 16) zeigt. Sie verdeutlicht auch, wieso es wichtig ist, die Berechnung mit verschiedenen Startwerten zu wiederholen. [62]

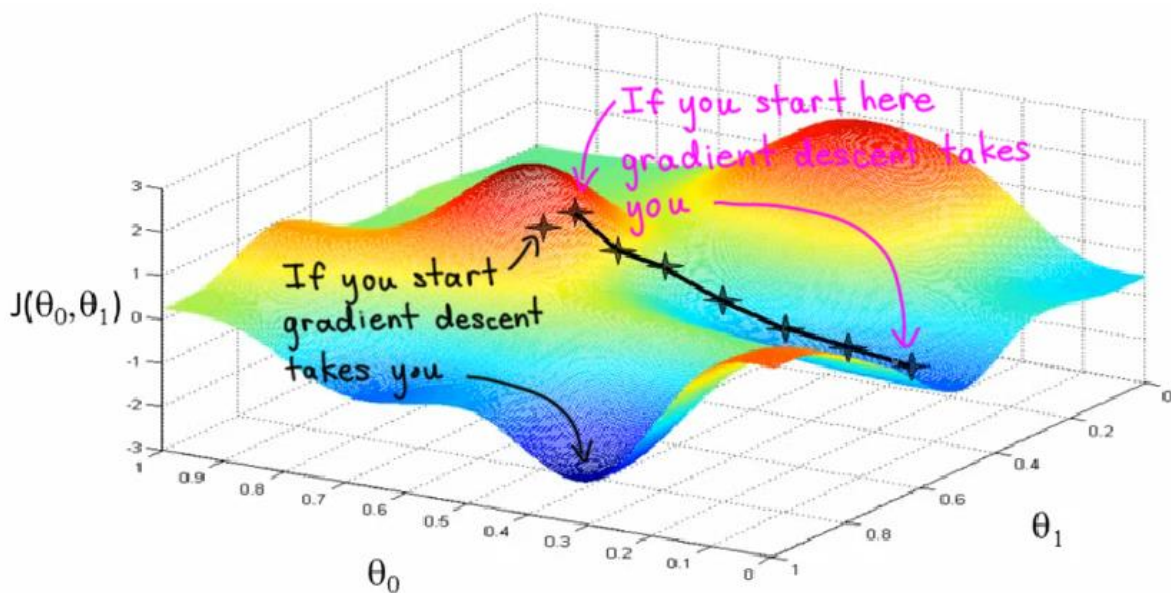


Abbildung 16: Gradienten Verfahren in 3D (Quelle: <https://github.com/RNogales94/ISI-Tensorflow/blob/master/README.md>)

Genau so funktioniert das Gradienten Verfahren, also Gradient Descent auf Englisch. Ein Gradient ist vergleichbar mit der Ableitung einer Funktion im 2-Dimensionalen, und gibt die Richtung des steilsten Anstieges an [64]. Man muss sich also immer in die Richtung des negativen Gradienten bewegen, um zum Minimum zu gelangen, daher der Name. In der Praxis werden zu dem weitere Tricks genutzt, um den Prozess effizienter zu gestalten. Zum Beispiel wird die Schrittweite, mit der man sich dem Minimum nähert, variabel angepasst, sodass diese immer kleiner wird, je näher man dem Punkt kommt, um möglichst hohe Genauigkeit zu garantieren.

Bei einem neuronalen Netz stellt jedes Neuron einen Parameter in der zu minimierenden Funktion dar. Wenn man Gradient Descent aber in dieser Form anwenden würde, würde es bei so vielen verschiedenen Werten ewig lange dauern, das Optimum zu berechnen, deswegen kommt meist eine abgewandelte Version davon zum Einsatz, das sogenannte *Stochastic* Gradient Descent-Verfahren. Dabei werden nicht die gesamten Trainingsdaten auf einmal verwendet, sondern kleinere Portionen, sogenannte „Mini-Batches“ nacheinander. Damit geht zwar ein kleiner Genauigkeitsverlust einher, dafür verringert sich die Trainingsdauer massiv.

Noch einmal kurz zusammengefasst: Die Verlustfunktion erkennt, wie viel das Netz noch lernen muss (wie sehr die einzelnen Gewichte noch adjustiert werden müssen), mit dem Gradientenverfahren findet es heraus, worauf es sich beim Lernen konzentrieren muss (mit welchen Gewicht-Werten es zum optimalen Ergebnis kommt). Die einzelnen Weights und Bias werden schließlich mit Backpropagation angepasst, die im nächsten Abschnitt erklärt werden.

8.4. Backpropagation

Denken wir wieder an das Beispiel zurück, das im Abschnitt Die Verlustfunktion erwähnt wurde: Die Output Wert für den Neuronen, der eine 5 symbolisiert, muss erhöht werden während alle anderen verringert werden sollen. Zur Erinnerung hier noch einmal die Formel für den Wert Y eines Neurons in einem Netz, in dem die ReLU Aktivierungsfunktion genutzt wird:

$$Y = \text{ReLU} \left(\sum (\text{weight} * \text{input}) + \text{bias} \right)$$

Wir haben also 3 Möglichkeiten, Y zu verändern:

1. Den Wert der Gewichte verändern
2. Den Wert der Neuronen in der Schicht davor anpassen
3. Den Bias erhöhen bzw. vermindern

Dank Gradient Descent weiß der Computer, wie er Gewichte und Bias Werte anpassen muss, um die Verlustfunktion zu minimieren. Auf den Wert des vorhergehenden Neurons können wir aber nicht direkt zugreifen, dieser besteht ja auch wiederum aus der Summe der Gewichte multipliziert mit den vorhergehenden Neuronen plus einem Bias in Bezug auf eine Aktivierungsfunktion. Dementsprechend gehen wir eine Schicht von hinten nach vorne zurück (deswegen der Name „Backpropagation“) und wiederholen den Prozess, passen Gewichte und Bias an, um den Wert des Neurons zu ändern. Dieser Vorgang wird so lange fortgesetzt, bis wir bei der allerersten Inputschicht ankommen, dort können wir logischerweise nichts verändern, da wir sonst die Eingabe in das Netz verfälschen würden [66].

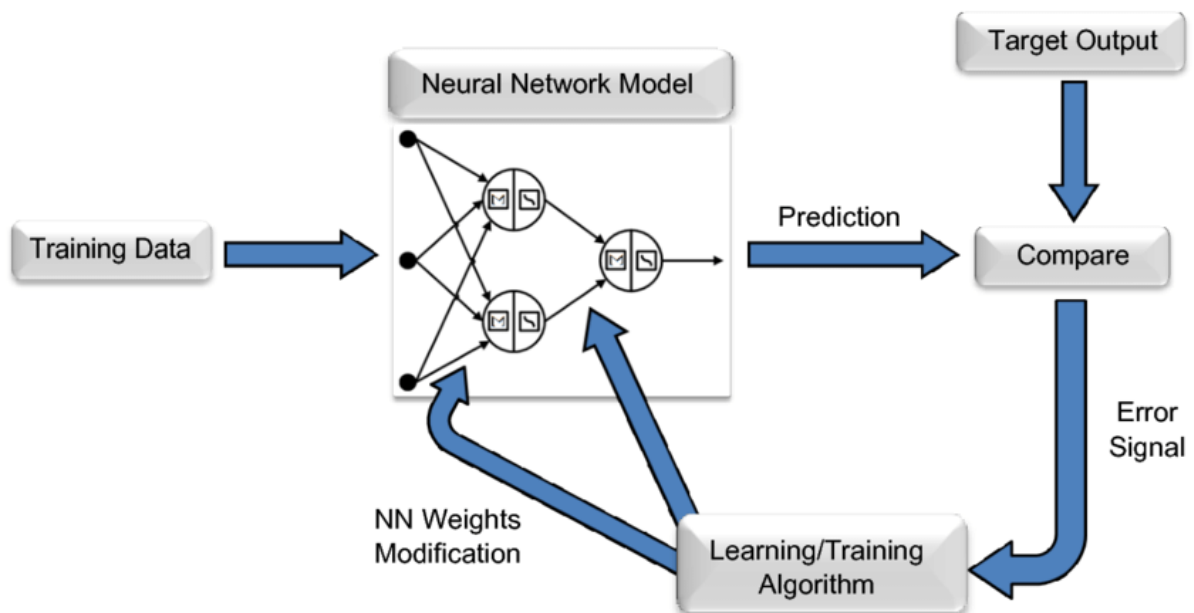


Abbildung 17: Übersicht über den Trainingsablauf (Quelle: https://www.researchgate.net/figure/The-representation-of-neural-network-training-process_fig5_299390844)

8.5. Der Trainingszyklus

Ein neuronales Netz wird in mehreren Durchläufen trainiert. Ein Durchgang, bei dem der ganze Trainingsdatensatz verwendet wird, nennt man auch Epoche, weil der gesamte Datensatz oft zu groß ist, um ihn in den Arbeitsspeicher zu laden, wird, er in Batches, also kleinere Portionen aufgeteilt. Abbildung 17 zeigt den schematischen Ablauf so einer Epoche, der gesamte Trainingsablauf beinhaltet oft bis zu mehrere hundert Durchgänge. Abbildung 18 zeigt den Verlauf der Erkennungsgenauigkeit und der Verlustfunktion eines Beispiel-Netzes über den Verlauf von 200 Epochen, jeweils bei den eigentlichen Trainingsdaten und den Testdaten, die nicht zum Trainieren verwendet werden.

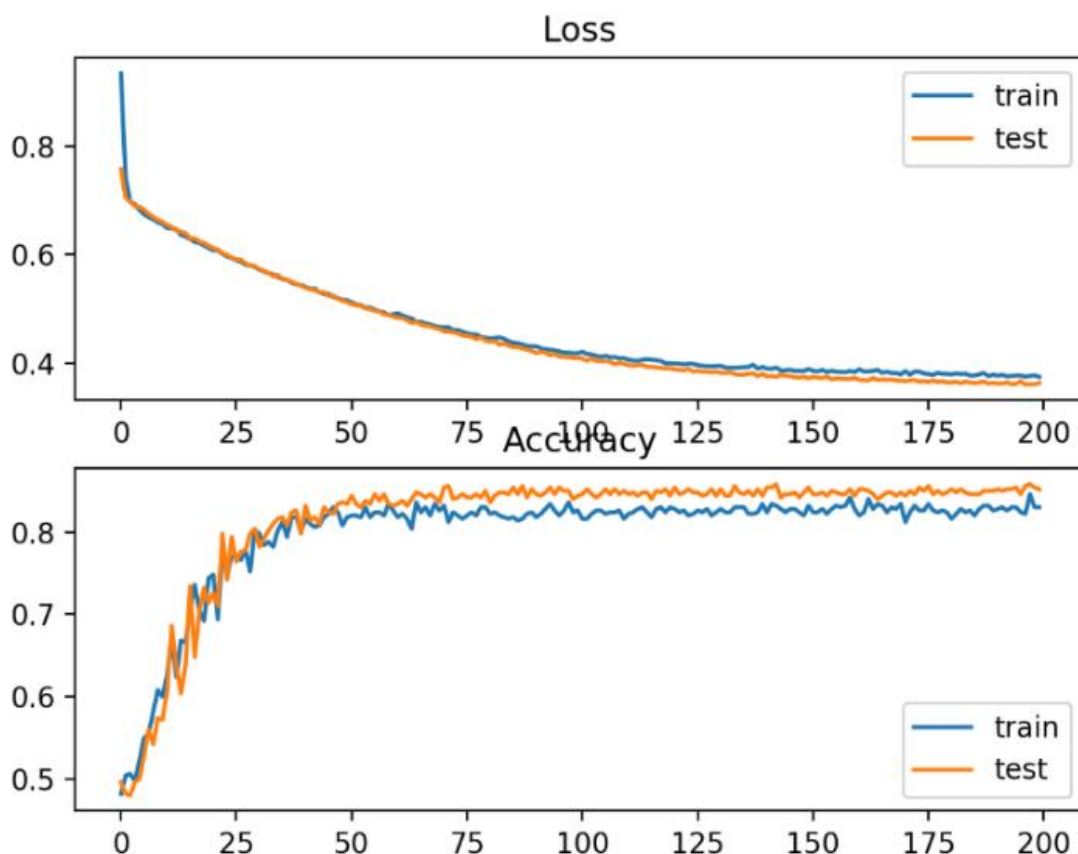


Abbildung 18: Fortschrittsverlauf während dem Training (Quelle: <https://medium.com/@pratyush057/elements-of-machine-learning-e09ebf16af19>)

9. Implementierung in Real-World-Scenarios

Sobald das neuronale Netzwerk soweit trainiert ist, dass es eine annehmbare Performance bietet, kann es gespeichert werden und steht für die weitere Nutzung bereit. Meist wird es als .h5 File gespeichert, in einem Dateiformat, dass vor allem für große technische oder wissenschaftliche Datensätze ausgelegt ist [67]. Diese Datei kann dann zum Beispiel auf einem Webserver in eine Internetseite implementiert werden, oder Teil einer Smartphone App werden.

Um eine App bzw. eine Website oder ähnliches mit Machine Learning auszustatten, gibt es prinzipiell zwei Möglichkeiten:

1. Das Modell selbst trainieren
2. Auf ein vortrainiertes Modell oder eine andere vorgefertigte Lösung zurückgreifen

Die erste Möglichkeit kann sehr aufwendig werden, vor allem wenn der Anwendungsbereich sehr exotisch ist. Die erste Herausforderung dabei ist das Sammeln der Daten, die dann, wie in den vorherigen Kapiteln beschrieben, verwendet werden, um das Model von Grund auf zu erstellen und zu trainieren.

Gerade für häufige Aufgabenstellungen, wie Texterkennung (engl. Optical Character Recognition, kurz OCR), Gesichts- oder Objekterkennung gibt es viele Möglichkeiten, diese zu lösen, ohne selbst ein eigenes Netz zu designen [69]. Zum Teil handelt es sich dabei um fertige Libarys, die man in ein Projekt einbindet, um auf die ML-Features zugreifen zu können, andere sind Cloud-based, funktionieren also über das Internet.

9.1. Machine Learning: On Device vs. Cloud-based

Es ist eine wichtige Entscheidung, ob das fertige Model über die Cloud oder direkt auf dem Gerät laufen soll, beides bietet unterschiedliche Vor- und Nachteile.

Bei Cloud-Based Machine Learning wird das Model auf einem Server gespeichert. Die Clients, also jene Geräte, auf denen die Software läuft, verbinden sich über eine API, also eine Schnittstelle mit dem neuronalen Netzwerk auf dem Server. Der Vorteil dabei ist, dass man so leistungsfähigere Modelle nutzen kann, die unabhängig von der Rechenleistung des Clients arbeiten. Dafür benötigt man aber eine funktionierende Internetverbindung. Denn wenn große Datenmengen wie Fotos oder Videos verschickt werden, kann das schnell Bandbreite verbrauchen.

Andererseits kann das Model auch direkt auf dem Gerät des Anwenders gespeichert werden, was gerade bei komplizierteren Netzen einiges an Speicherplatz kosten kann. Dafür ist der Nutzer aber auch nicht von seiner Internetverbindung abhängig, und die Anwendung funktioniert schneller, da die Daten nicht übers Internet verschickt werden müssen. On-Device Lösungen sind meist weniger leistungsstark als solche, die über die Cloud funktionieren.

Natürlich kann man sowohl eigene, selbst trainierte Netze über die Cloud nutzen, als auch vorgefertigte Kits [68].

10. Praxiseinsatz mit Pre-Trained Models

Den Aufbau und die Funktionsweise von neuronalen Netzen haben wir in den vorherigen Kapiteln schon ausführlich besprochen. Nun werden wir uns ansehen, welche Möglichkeiten man als Entwickler hat, in den Genuss von künstlicher Intelligenz zu kommen, ohne dafür massenhaft Daten zu sammeln und umständlich ein Netz erstellen zu müssen. Danach werden wir uns ein Praxisbeispiel ansehen, das zeigt wie eine Android App vorgestellte Techniken für OCR (Optical Character Recognition, also Texterkennung) nutzen kann.

10.1. Google Cloud Vision API

Alphabet bzw. Google ist sehr stark im ML-Sektor engagiert und bietet neben Frameworks wie Tensorflow, spezieller Hardware und online Trainingskursen auch KI-Produkte an [69]. Eines davon ist die Vision API von Google Cloud. Durch diese API (engl. Application Programming Interface, eine Schnittstelle zwischen 2 Programmen) erhält man Zugang zu leistungsfähigen, vorabtrainierten neuronalen Netzen, die auf Google-Servern laufen. Diese eignen sich für verschiedene Anwendungen, z.B. Gesichts-/Texterkennung oder für die Angabe von Bildattributen. Die Kosten werden in Blöcken zu je 1000 Anfragen verrechnet, wobei der erste Block kostenlos ist, danach fallen pro Block Gebühren von 0,60 \$ bis 3,50 \$ an [70].

10.2. Tesseract OCR

Tesseract ist eine freie Software zur Texterkennung (OCR) [71]. Das Projekt wurde bereits 1985 von Hewlett-Packard gestartet, inzwischen wurde es aber in ein Open-Source Projekt umgewandelt, das heißt der Quellcode wurde öffentlich verfügbar gemacht. Seit 2006 wird Tesseract von Google weiter betreut. Die Software benutzt LSTM Netze statt den hier besprochenen konvolutionellen Netzen, und ist in der Lage mehr als 100 Sprachen zu unterscheiden, kann aber auch auf weitere, bisher nicht unterstützte Sprachen übertragen werden. Tesseract arbeitet direkt auf dem Gerät, auf dem es installiert ist, also nicht über die Cloud, dafür aber gratis. Da ich für meine eigene App, auf die ich später noch genauer eingehen werde, auch auf OCR setzen will, habe ich beschlossen, in einem kleinen Testbeispiel herauszufinden ob Tesseract dafür geeignet wäre. Mit Hilfe eines Online Tutorials [72] habe ich eine kleine Android-App geschrieben, die den Text auf einem Bild erkennen soll. Das gewünschte Ergebnis ist auf der nächsten Seite zu sehen (Abbildung 19,20).

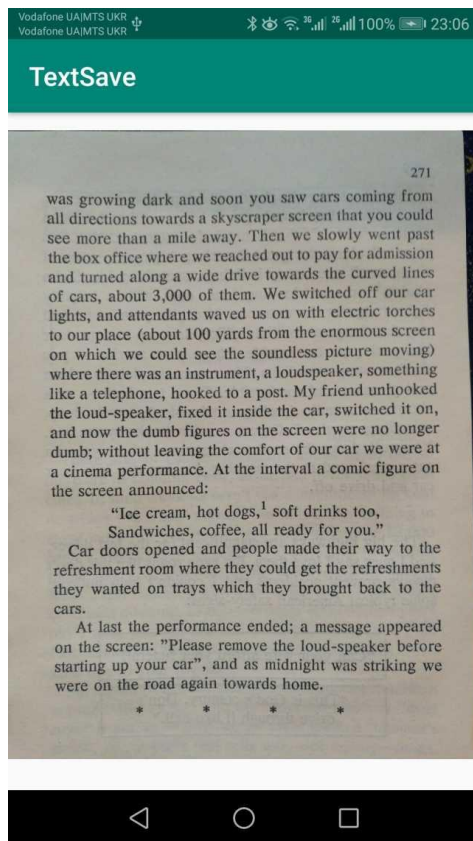


Abbildung 20: Tesseract Beispiel Input (Quelle: <https://www.codeproject.com/Articles/1275580/Android-OCR-Application-Based-on-Tesseract>)

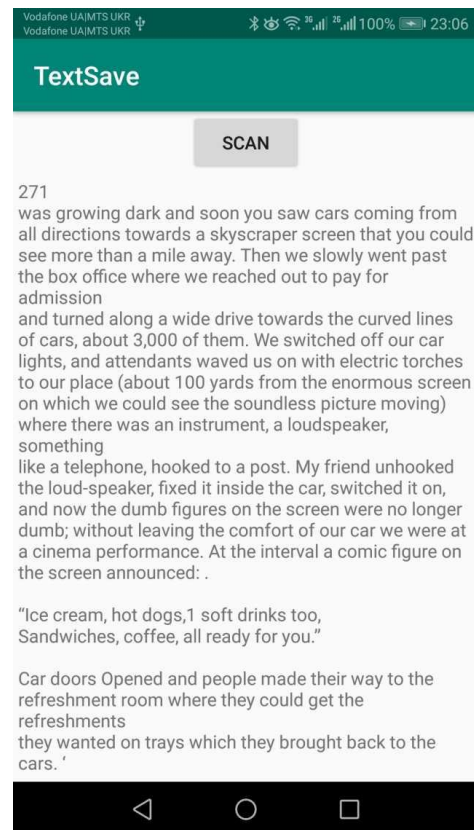


Abbildung 19: Tesseract Beispiel Output (Quelle: <https://www.codeproject.com/Articles/1275580/Android-OCR-Application-Based-on-Tesseract>)

Kurze Zeit später war ich mit meinem Prototyp fertig, hier das Ergebnis:

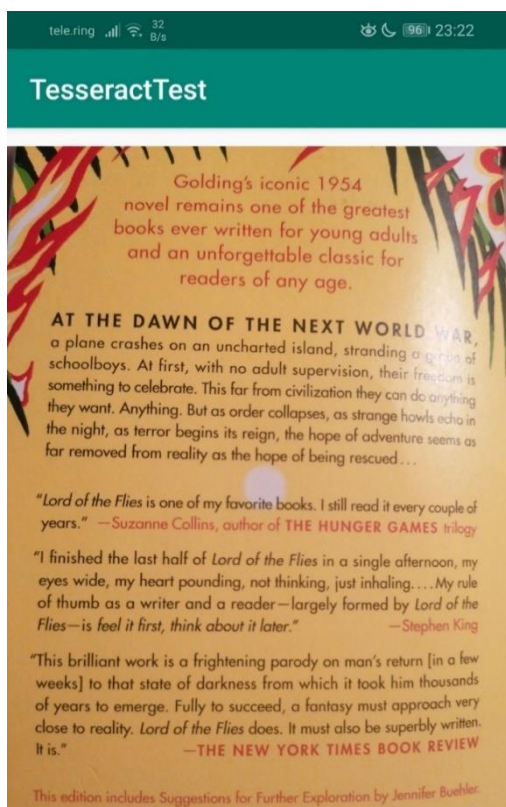


Abbildung 22: Mein Tesseract Test Input (Quelle: Selbst erstellt)

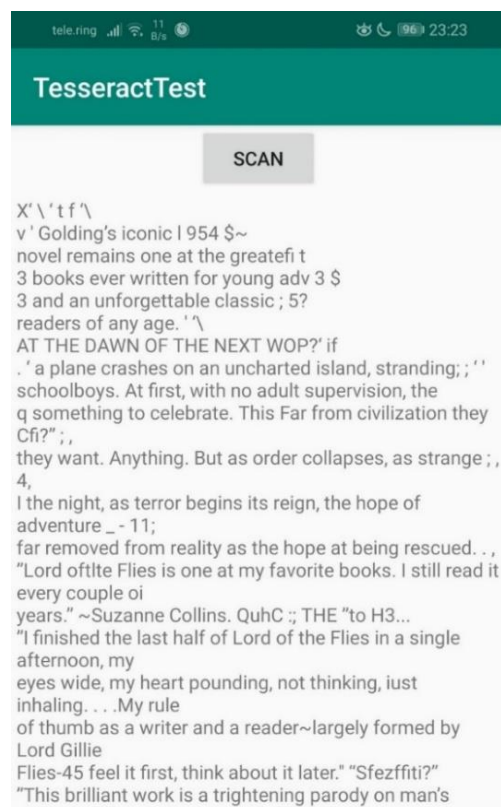


Abbildung 21: Mein Tesseract Test Output (Quelle: Selbst erstellt)

Das Verarbeiten des Bildes dauerte circa 30 Sekunden, viel zu lang für eine App, die man alltäglich nutzen können sollte. Das Ergebnis ließ leider ebenfalls zu wünschen übrig.

Woran die schlechte Performance liegt, ist für mich offen, da es genug Beispiele gibt, bei denen Tesseract besser funktioniert. Tesseract so zu konfigurieren, dass es nur alphanumerische Zeichen erkennt, hätte wahrscheinlich geholfen, trotzdem scheint das Modell nur auf einzelne Zeichen zu achten, ohne zu bedenken, ob diese als Wörter zusammengehängt Sinn ergeben. Allerdings gibt es auch im Google Playstore Apps, die Tesseract OCR verwenden, die zwar besser als mein Test funktionieren, aber immer noch weit weg vom Ideal sind. Die App, die ich getestet habe („tess-two example Tesseract OCR“ von Yuliia Ashomok [73]), benötigt mit 27,3 MB circa ein Zehntel des Speicherplatzes von meiner App und liefert die Ergebnisse bedeutend schneller (in ungefähr 2-3 Sekunden). Die Genauigkeit reicht aber immer noch nicht aus, um die Software sinnvoll anzuwenden. Das Ergebnis ist in der folgenden Abbildung zu sehen.

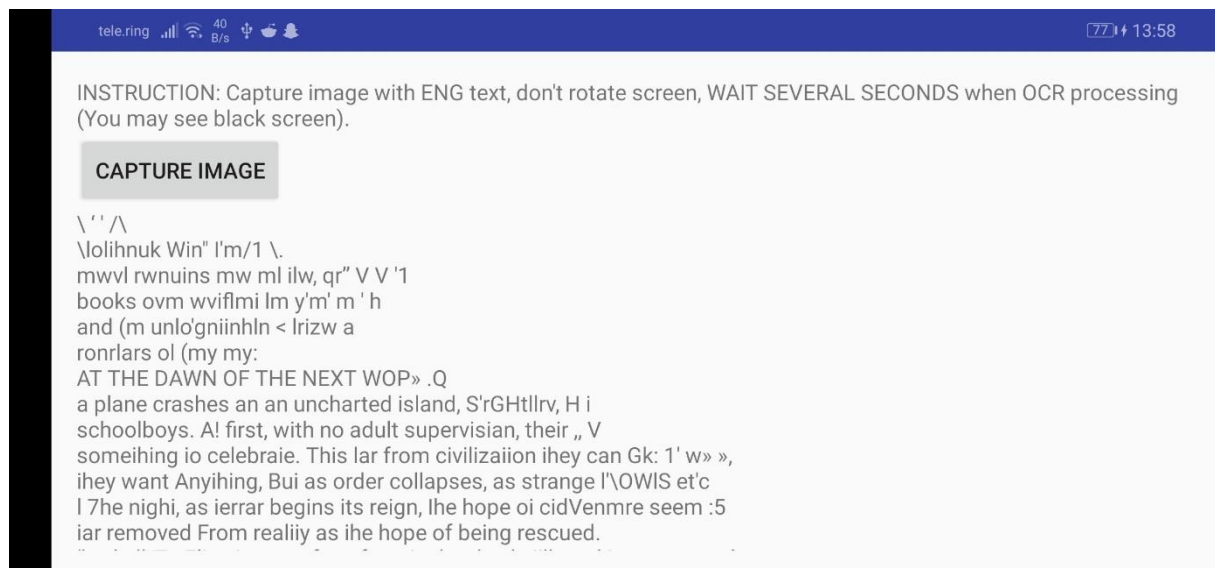


Abbildung 23: „tess-two example Tesseract OCR“ von Yuliia Ashomok (Quelle: Selbst erstellt)

Vor allem als Anfänger erscheint die Libraries ziemlich überwältigend. Es gibt nicht nur über 650 frei konfigurierbare Parameter, [74] sondern neben den über 100 manuell herunterzuladenden Sprachpaketen auch noch verschiedene Versionen für unterschiedliche Situationen. Pytesseract nennt sich beispielsweise die Version für die Programmiersprache Python, während Tess4J ausschließlich für Java funktioniert. Tess-two ist eine Variante, die speziell für Android gedacht ist.

10.3. *Firebase ML Kit*

Firebase ist eine Entwicklungsplattform für Apps und Webanwendungen, ein sogenanntes Backend-as-a-Service, kurz BaaS. Um die Entwicklung leichter zu gestalten, bietet Firebase einige nützliche Features an, darunter eine Cloud-Datenbank, Features zur leichteren Userregistrierung und Möglichkeiten, Werbung in Apps oder auf Webseiten leichter zu verwalten. Firebase wurde 2011 gegründet, 2014 für (vermutlich) 80 Millionen Dollar von Google übernommen [75] und wird derzeit von circa 1.5 Millionen Apps verwendet [76]. Ein praktisches Feature ist das ML-Kit, das Cloud-APIs und vortrainierte Modelle für gebräuchliche Anwendungen beinhaltet. Die APIs entsprechen denen von Google Cloud und sind demnach kostenpflichtig, während alle Funktionen, die direkt auf dem Gerät laufen, gratis sind. Die Anleitung ist leicht verständlich und ermöglicht einen schnellen Einstieg. Das Feature, das für meine App am interessantesten sein wird, ist die Text Recognition Funktion des Firebase ML Kits. Im Gegensatz zu Tesseract müssen hier die einzelnen Sprachpakete nicht extra heruntergeladen werden, sobald die App das erste Mal gestartet wird, verbindet sie sich mit den Firebase Servern und lädt das Modell selbständig herunter.

Nachdem Tesseract bei meinem Test hinter den Erwartungen zurückgeblieben ist, wollte ich herausfinden, ob das Firebase ML Kit besser funktioniert. Vergleiche von anderen Websites zeigen, dass Firebase besser abschneidet, wenn auch nur knapp [77]. Bei meinem eigenen Versuch war der Unterschied aber deutlicher:

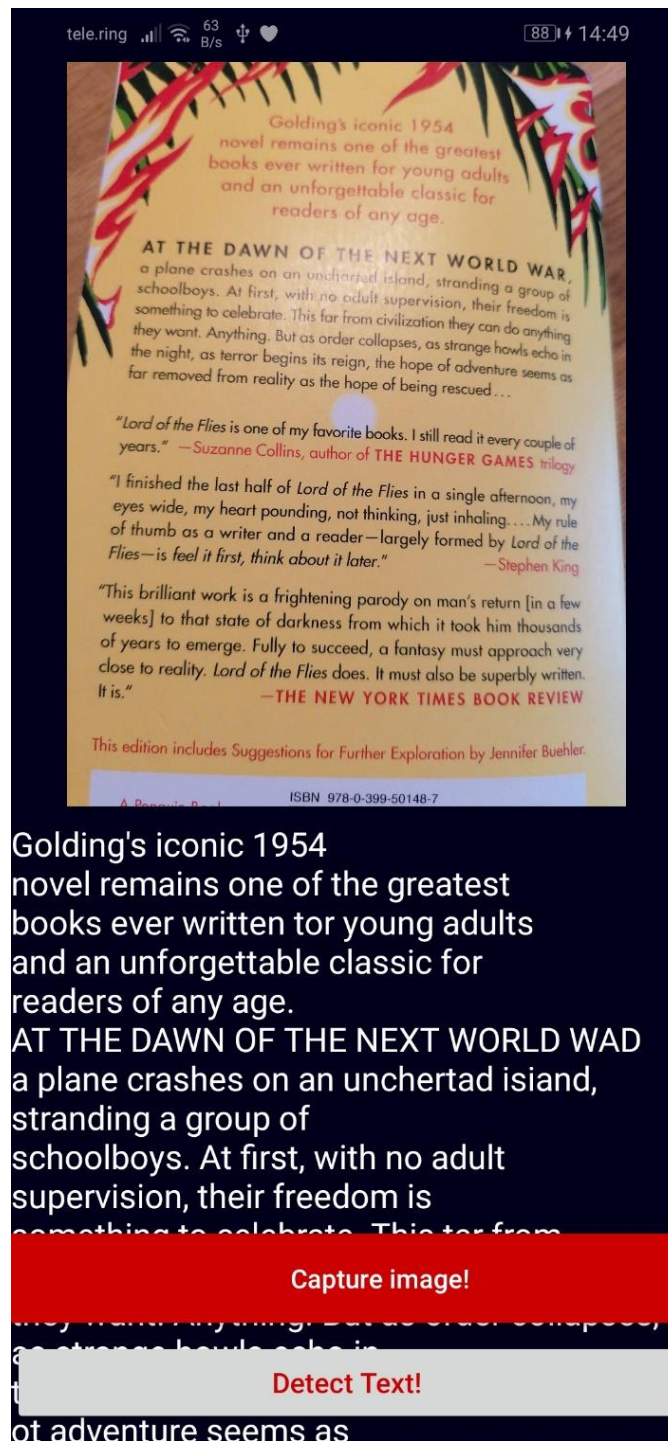


Abbildung 24: Ergebnis meines Firebase Tests (Quelle: Selbst erstellt)

Fast alle Wörter wurden fehlerfrei erkannt und dass nicht nur auf Englisch, sondern auch bei deutschen und lateinischen Texten.

Mit 147 MB war meine Firebase-Demo kleiner als mein Tesseract-Test und mit circa 1-2 Sekunden Ladezeit deutlich schneller. Meinem Empfinden nach, ist das ML Kit auch um einiges benutzerfreundlicher, statt unübersichtlich vielen Varianten gibt es nur eine einzige Version und die ist auch logisch aufgebaut und leicht handzuhaben. Damit war die Entscheidung, für meine App Firebase zu verwenden, klar.

11. Praktische Anwendung: Latein Pro

Mein Ziel bei dieser Arbeit ist es nicht nur, Machine Learning in der Theorie zu verstehen, sondern ich will auch ein praktisches Beispiel dazu liefern – der Titel meiner Arbeit heißt nicht umsonst „Funktionsweise und Anwendungsmöglichkeiten von Künstlicher Intelligenz“. Die Experimente mit Tesseract und Firebase im vorherigen Kapitel waren die Vorbereitung dafür, auf den nächsten Seiten werde ich ausführlich beschreiben, wie ich **Latein Pro** entwickelt habe, eine App, die es Schülern möglich macht, alle Vokabeln eines lateinischen Textes per Foto zu übersetzen. Der Code ist Open-Source und steht online auf GitHub zur Verfügung (unter <https://github.com/maxspanning/LateinPro>) [78], die fertige App selbst kann man im Google Play Store [79] gratis herunterladen.

11.1. Die Idee

Als Schüler eines humanistischen Gymnasiums lerne ich seit der 2. Klasse Latein, eine Sprache, bei der man (in der Oberstufe) fast immer auf ein Wörterbuch angewiesen ist. Die meisten Latein-Apps funktionieren zwar gut und liefern zuverlässige Übersetzungen, allerdings muss man jedes Wort einzeln eintippen, wodurch man gegenüber einem herkömmlichen Latein Wörterbuch kaum Zeit spart. Die Idee von Latein Pro ist es, OCR Technologie zu nutzen, um Wörter im Text zu scannen und zu übersetzen, sodass man eine Alternative zu Stowasser und Co schafft, die genau so zuverlässig, leichter, schneller, innovativer und effizienter funktioniert.

Die Zielgruppe ist groß, denn es gibt immer noch viele österreichische Schulen bei denen Latein ein Hauptpflichtfach ist. Somit könnte die App für eine hohe Anzahl von Schüler & Schülerinnen interessant sein und würde sich im Schulalltag von vielen jungen Menschen als nützlich erweisen. Die App wäre also kein reines, nur für meine VWA konzipiertes Demoobjekt, sondern könnte tatsächlich Schülern wie mir und vielen anderen helfen.

Daneben eignet sich Optical Character Recognition perfekt als Demonstrationsbeispiel für die Leistungsfähigkeit von Machine Learning. Wie bereits im vorhergehenden Kapitel gezeigt, erkennt das Firebase ML Kit zuverlässig Wörter.

Da Latein eine tote Sprache ist, entwickelt sich der Wortschatz nicht mehr weiter, was es leichter macht, die Wörterbuch-Datenbank zu pflegen. Im Gegensatz zu Sprachen wie Deutsch oder Französisch kommen kaum ungewöhnliche/schwer zu erkennende Buchstaben (z.B. ß, ê, æ, ü, oder œ) vor, und beim herkömmlichen Übersetzen ist fast immer ein Wörterbuch dabei, was das Anwendungspotential weiter erhöht. Das macht Latein zur perfekten Sprache für eine derartige App.

Die Benutzung sollte so simpel wie möglich sein: Direkt nach dem Öffnen der App soll der Nutzer sofort die Möglichkeit haben, ein Foto aufzunehmen, das er dann zuschneiden soll, um möglichst nur den gewünschten Text zu übersetzen. Anschließend werden alle Wörter in verschiedenen Farben dargestellt, je nachdem ob es sich bei dem Wort um ein Nomen, Verb oder Adjektiv handelt. Beim Click auf das jeweilige Wort erhält der Nutzer die Übersetzung. Falls der User ein einzelnes Wort übersetzen möchte, soll er auch dazu die

Möglichkeit haben, indem er das Wort einfach in eine Suchleiste eingibt. Eine detaillierte technische Beschreibung zur Funktionsweise und den verwendeten Ressourcen folgt in den nächsten Abschnitten, davor noch aber eine kurze Erläuterung über Android Apps im Allgemeinen.

11.2. Aufbau einer Android App

Bevor ich den Aufbau von Latein Pro erläutere, möchte ich davor erklären, wie Android Apps im Allgemeinen funktionieren. Die offiziellen Programmiersprachen für Android sind Java und Kotlin, Android Studio ist der offizielle von Google angebotene Editor, der es ermöglicht, Code schneller und effizient zu schreiben. Das Design der App wird über XML Files definiert, Android Studio bietet aber auch einen eingebauten Design Editor, um den Prozess zu vereinfachen. Eine wichtige Datei in jeder Android App ist außerdem AndroidManifest.xml, eine Datei, die nicht für das Aussehen der App verantwortlich ist, sondern wichtige Metainformationen enthält (z.B. der App-Name, Mindestanforderungen, App-Version etc.) [80]. Über sogenannte Gradle Scripts ist es außerdem möglich, zusätzliche Libarys zu laden, wie zum Beispiel Firebase, um den Entwicklungsprozess angenehmer zu gestalten.

Sobald der Code fertig geschrieben ist, wird er kompiliert, das heißt der Sourcecode wird in eine ausführbare App umgewandelt. Im Falle von Android entstehen dabei .apk Dateien. Das sind die Apps, die man am Smartphone benutzen kann. APK (kurz für Android Package) Files sind technisch gesehen ZIP Ordner, die alle für die App benötigten Ressourcen (z.B. Bilder, Texte) und den Binärcode beinhalten. Bevor die App öffentlich im Play Store angeboten werden darf, muss sie außerdem noch geprüft und signiert werden, das heißt die APK wird mit einem einzigartigen Schlüssel versehen, damit Nutzer sicher sein können, dass die App vom rechtmäßigen Entwickler stammt und keine Malware beinhaltet. Zudem läuft jede App in einer isolierten Umgebung und standardmäßig ist es nicht möglich, unsignierte Apps herunterzuladen, das macht Android (und auch IOS) zu einem sehr sicheren Betriebssystem [81].

11.3. Aufbau von Latein Pro

Auf der nachfolgenden Seite findet sich eine Skizze des Programmflusses für die App. Der grün umrandete Bereich zeigt die beiden Aktivitäten, die der Nutzer hauptsächlich benutzen wird, der gelb umrandete Bereich umfasst die Hintergrundaktivitäten, die der User zwar nicht direkt zu Gesicht bekommt, die aber für die Funktionsfähigkeit der App unerlässlich sind.

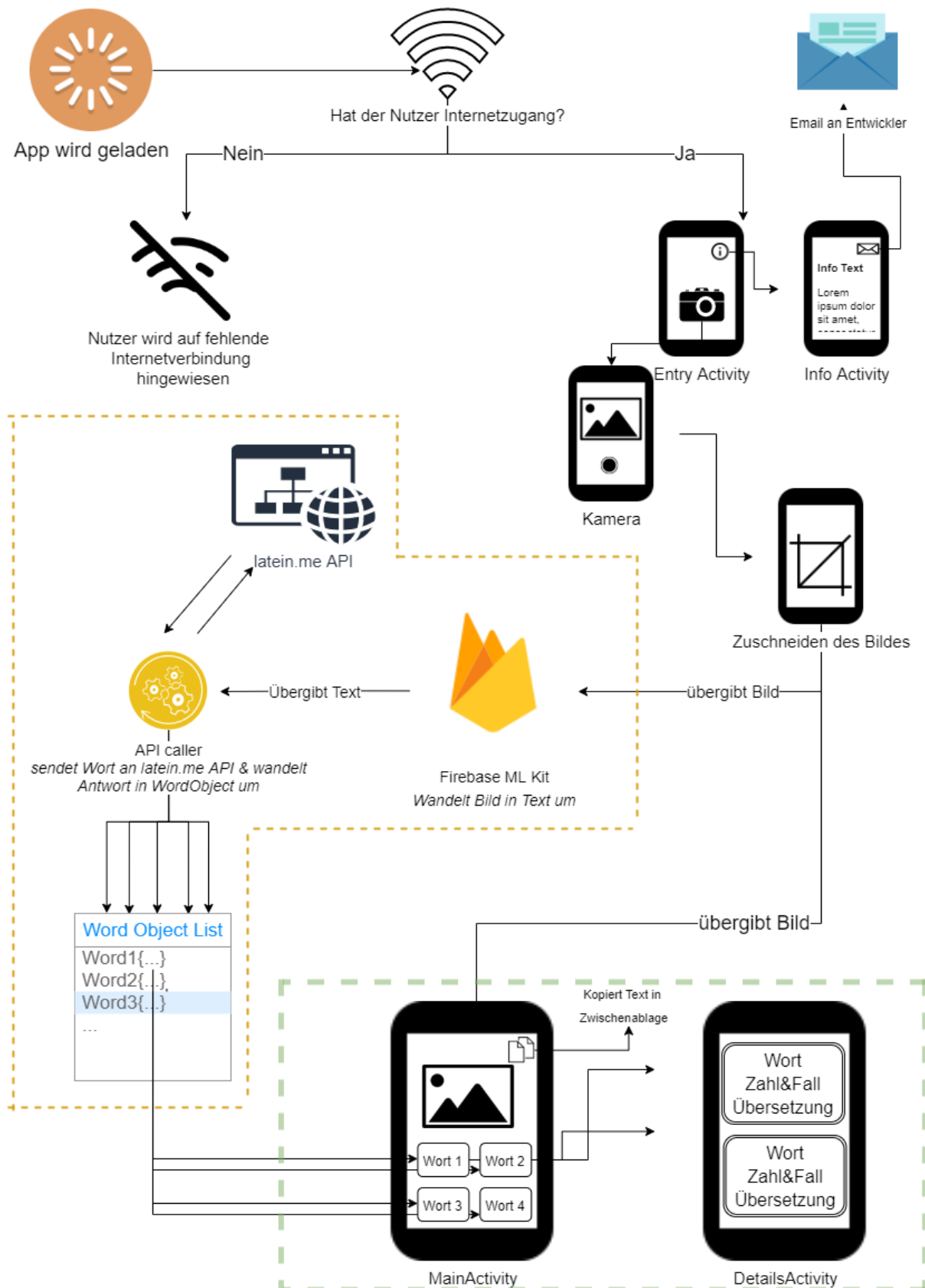


Abbildung 25: Programmfluss von Latein Pro (Quelle: Selbst erstellt)

Sofort nach dem die App geladen wird, wird geprüft, ob der User Zugang zum Internet hat, indem im Hintergrund eine Website aufgerufen wird. Falls das nicht funktioniert, stoppt die App und weist auf das Problem hin. Ansonsten wird der Startbildschirm geladen, im Sourcecode heißt dieser Teil der App *EntryActivity.java*. Durch den Klick auf einen Info Button im rechten oberen Eck erscheint eine Seite mit Bedienungsanleitung sowie Informationen zur Entwicklung und zu Kontaktmöglichkeiten, ein Email Button auf dieser Seite ermöglicht außerdem, das rasche Senden von Feedback-E-mails.

Sollte man sich dazu entscheiden, auf den großen Kamera-Button zu drücken, wird die Kamera geöffnet und der/die Nutzer/in kann ein Bild des Textes aufnehmen, den er/sie übersetzen will. Das Bild kann dann so zugeschnitten werden, dass nur der gewünschte Text zu sehen ist, so kann man die Qualität der Übersetzung verbessern. Zum Zuschneiden wird wieder eine externe Library verwendet, und zwar *Android-Image-Cropper* von Arthur, der Code ist verfügbar auf GitHub [82].

Das zugeschnittene Bild wird schließlich dem *FirebaseVisionTextRecognizer* übergeben, ein Objekt, das im ML Kit enthalten ist. Dieser Teil bildet das Herzstück der Software, denn hier wird das Bild rekursiv zuerst in Textblöcke aufgeteilt, dann in einzelne Zeilen und schlussendlich in Wörter. Die App merkt sich außerdem, wie viele Wörter in jeder einzelnen Zeile vorkommen, damit gewährleistet ist, dass die Wörter später genau so dargestellt werden, wie sie auf dem Bild zu sehen sind. Das sorgt für eine angenehmere User Experience.

Jedes einzelne Wort wird anschließend der *ApiCaller.java* Klasse übergeben, eine Datei die ein lateinisches Wort als Input nimmt und ein *WordObject* als Output zurückgibt. Dieses *WordObject* ist eigens für Latein Pro definiertes Objekt und beinhaltet alle benötigten Informationen zu einem Wort, nämlich

- das eigentliche Wort nach dem gesucht wurde
- die deutschen Übersetzungen
- die Nennform des Wortes
- die Wortart
- den Fall und das Geschlecht des Wortes

Diese Informationen erhält der *ApiCaller* über die API der Website *latein.me*. Als HTTP Client verwendet Latein Pro *OkHttp*, eine Library die im Vergleich zu Standard Java HTTP Client benutzerfreundlicher und effizienter ist [83]. Die Anfragen an die API geschehen parallel, sodass alle Daten, je nach Größe des Textes und der Qualität der Internetverbindung, in 4 bis 6 Sekunden geladen sind. Um diese Zeit zu überbrücken, wird an dieser Stelle eine Werbeanzeige geschaltet. Geregelt werden diese über das Google Admob Netzwerk. Sobald das *WordObject* erzeugt wurde, wird es zu einer List hinzugefügt, die alle *WordObjects* für alle Wörter beinhaltet.

Die *MainActivity* ist der Bereich der App, in dem sich der Nutzer am häufigsten aufhält. In der oberen Hälfte wird das zugeschnittene Bild eingefügt, darunter wird für jedes *WordObject* ein Button erstellt, der je nach Wortart des *WordObjects* eine andere Textfarbe bekommt. Sobald der Nutzer auf so einen Button klickt, gelangt er zur *DetailsActivity* Seite, auf der die

Daten, die im *WordObject* enthalten sind, übersichtlich aufbereitet dargestellt werden. Im rechten oberen Eck bietet ein Copy-Button außerdem die Möglichkeit, den gesamten Text in die Zwischenablage zu kopieren.

Eine Suchleiste ermöglicht es, von fast jedem Teil der App Wörter direkt zu suchen, dabei wird der Suchbegriff direkt an den *ApiCaller* weitergegeben, das zurückgegebene *WordObject* wiederum wird direkt in der *DetailsActivity* angezeigt. Rechts direkt neben der Suchleiste kann man mit einem Switch zwischen einem dunklen und einem hellem Erscheinungsbild wechseln, der Dark-Mode ist besonders nützlich, wenn man die App im Dunkeln benutzt und die Augen schonen möchte oder wenn man Akkustrom sparen will.

11.4. Unterstützung bei der Entwicklung von Latein Pro

Die Entwicklung von Latein Pro, vom Beginn bis zum Launch im Google Play Store, dauerte circa einen Monat. Firebase bietet nicht nur ML-Unterstützung an, sondern auch Möglichkeiten zur App-Verteilung an Beta-Tester, die mir während des Entwicklungsprozesses Feedback geben konnten, um Latein Pro noch weiter zu verbessern. Besonders hilfreiche Tester waren Felix Sax sowie Julian Flavio Müller, die mich beim Launch unterstützt haben, indem sie das Logo entworfen und den Beschreibungstext für den Playstore geschrieben haben, außerdem haben die beiden mit ihrem Engagement viele Personen von der App überzeugt und so für mehr Downloads gesorgt, daher werden sie auch am Ladescreen von Latein Pro namentlich erwähnt. An der eigentlichen Entwicklung, der Idee oder dem Code waren beide jedoch nicht beteiligt.

11.5. Probleme der App

Trotz größtmöglicher Sorgfalt beinhaltet Latein Pro immer noch einige Bugs und Probleme, die meistens auf die relativ schnelle Entwicklungszeit und hauptsächlich den Mangel an Erfahrung zurückzuführen sind. Hier sind die Wichtigsten aufgelistet:

- Probleme beim Dark Mode: Wenn ein User in der *Main Activity* in den dunklen bzw. hellen Modus wechselt, ändert sich zwar die Hintergrund Farbe, nicht aber die Farbe der Wörterbuttons. Ein ähnliches Problem besteht in der *Details Activity*, wo sich nur die Farbe des letzten Eintrages ändert und alle anderen Einträge fast unsichtbar werden. Mit mehr Recherche könnte man dieses Problem lösen, was aber mehr Zeit in Anspruch nehmen würde.
- Bei sehr alten Android Geräten stürzt die App ab, sobald man ein Foto aufgenommen hat, allerdings nur, wenn dort Text zu sehen ist, sonst nicht. Aufgrund der Seltenheit des Bugs liegen nicht mehr Informationen vor, was es schwer macht diesen zu lokalisieren und zu beheben. Meiner Vermutung nach tritt der Fehler bei den Android Versionen 5.1 und geringer auf (die aktuelle Android Version ist 10 [84]). Eine Lösungsmöglichkeit wäre es, die Mindestanforderungen zu erhöhen. Das würde aber nur die Symptome bekämpfen, nicht aber die Ursache.
- Beim Auslösen eines geheimen Eastereggs stürzt die App auf manchen Handys ab. Das Easteregg ist aber schwer zu finden und kaum bekannt, deswegen wird es auch sehr unregelmäßig ausgelöst.

Alle Bugs treten entweder sehr selten auf oder haben nur geringe Auswirkungen auf die Funktionalität der App, weswegen sie noch nicht gefixt wurden. Ein weiterer, großer Problempunkt von Latein Pro ist die Android-Exklusivität, wodurch die Hälfte aller potenziellen Nutzer verloren geht. Aufgrund des großen Aufwandes und der Kosten, die eine Portierung auf IOS mit sich bringen würde, wird es diese auch nie geben. Alle anderen Bugs werden aber nach der Reihe in zukünftigen Updates behoben.

12. Resümee

Ziel dieser Arbeit war es, Künstliche Intelligenz sowohl in der Theorie anschaulich zu erklären, als auch die Umsetzung in die Praxis zu demonstrieren. In den ersten 9 Kapiteln wurde versucht, Basiswissen über Machine Learning, neuronale Netze, ihren Aufbau und den Lernvorgang zu vermitteln. Natürlich ist das nur ein kleiner Ausschnitt aus der sich rasend schnell verändernden Welt der Artificial Intelligence, alles andere hätte auch den Rahmen dieser Arbeit bei weitem gesprengt. Auf kaum einem anderen Gebiet der Technik werden so häufig neue Innovationen publiziert, deswegen werden wir wahrscheinlich auch in Zukunft viel Interessantes von diesem Fachgebiet hören.

12.1. *Latein Pro*

Latein Pro ist der praktische, angewandte Teil meiner VWA. Die Vorbereitungen für die Entwicklung wurden bereits in Kapitel 10 getroffen, wo ich durch Experimente mit verschiedenen Modellen das passendste herausfinden konnte. Das darauffolgende Kapitel beschreibt schließlich die eigentliche Entwicklung der App.

Ziel von Latein Pro ist es, ein anschauliches Demoobjekt für Machine Learning bzw. OCR zu sein, das auch im Unterricht von Schülern als effizientes Hilfsmittel eingesetzt werden kann. Ich denke, dass beide Punkte erfüllt wurden. Die Funktionsweise wurde im Text ausführlich beschrieben und die App zeigt somit, was technisch möglich ist. Der erste Release im Google Play Store fand am 27. Jänner 2020 statt, und 2 Wochen später ist Latein Pro schon bei über 50 Downloads angelangt, mit einer perfekten 5-Sterne Bewertung. Außerdem zeigen einzelne Tests im Unterricht auch, dass die App sehr gut als eine benutzerfreundlichere Alternative zum Stowasser-Wörterbuch verwendet werden kann.

Der Sourcecode der App ist auf GitHub zu finden, dieser ist auch auf der nächsten Seite mittels QR-Code verlinkt. Jeder ist dazu eingeladen, den Code zu lesen, zu benutzen, oder auch zu verändern.



13. Literaturverzeichnis

- [1] „Artificial intelligence Definition und Bedeutung | Collins Wörterbuch“. [Online]. Verfügbar unter: <https://www.collinsdictionary.com/de/worterbuch/englisch/artificial-intelligence>. [Zugegriffen: 16-Nov-2019].
- [2] „Deep learning“, *Wikipedia*. 24-Juli-2019.
- [3] Singularity Prosperity, *The History of AI (What Is Artificial Intelligence)*. 2018.
- [4] „Dartmouth Artificial Intelligence (AI) Conference“. [Online]. Verfügbar unter: https://www.livinginternet.com/i/ii_ai.htm. [Zugegriffen: 20-Aug-2019].
- [5] Nihilist, *what are rule based expert systems and how do they work*. 2016.
- [6] W. van Melle, „MYCIN: a knowledge-based consultation program for infectious disease diagnosis“, *International Journal of Man-Machine Studies*, Bd. 10, Nr. 3, S. 313–322, Mai 1978, doi: 10.1016/S0020-7373(78)80049-2.
- [7] D. Osinga, *Deep Learning Kochbuch: Praxisrezepte für einen schnellen Einstieg*, 1. Aufl. Heidelberg: O'Reilly, 2019.
- [8] K. Manhart, „Moore's Law am Ende? - Grenzen der Prozessor-Technik 2020 erreicht - PC Magazin“, *pc-magazin*. [Online]. Verfügbar unter: <https://www.pc-magazin.de/ratgeber/moore-law-report-ende-2020-1938131.html>. [Zugegriffen: 20-Aug-2019].
- [9] „The Turing test: Can a computer pass for a human? - Alex Gendler - YouTube“. [Online]. Verfügbar unter: <https://www.youtube.com/watch?v=3wLqsRLvV-c>. [Zugegriffen: 31-Okt-2019].
- [10] „Why Is Turing Test Irrelevant? - the ML blog - Medium“. [Online]. Verfügbar unter: <https://medium.com/themlblog/why-is-turing-test-irrelevant-5e749c8a385a>. [Zugegriffen: 31-Okt-2019].
- [11] „Starke KI, schwache KI - Was kann künstliche Intelligenz?“, *JAAI.de*, 27-Sep-2017. [Online]. Verfügbar unter: <https://jaai.de/starke-ki-schwache-ki-was-kann-kuenstliche-intelligenz-261/>. [Zugegriffen: 20-Aug-2019].
- [12] C. K. GN, „Artificial Intelligence: Definition, Types, Examples, Technologies“, *Medium*, 17-Juni-2019. [Online]. Verfügbar unter: <https://medium.com/@chethankumargn/artificial-intelligence-definition-types-examples-technologies-962ea75c7b9b>. [Zugegriffen: 20-Aug-2019].
- [13] „IBM Watson Tone Analyzer“. [Online]. Verfügbar unter: tone-analyzer-demo.ng.bluemix.net. [Zugegriffen: 22-Aug-2019].
- [14] „Better Language Models and Their Implications“, *OpenAI*, 14-Feb-2019. [Online]. Verfügbar unter: <https://openai.com/blog/better-language-models/>. [Zugegriffen: 23-Aug-2019].
- [15] J. Brownlee, „How to Perform Object Detection With YOLOv3 in Keras“, *Machine Learning Mastery*, 26-Mai-2019. [Online]. Verfügbar unter: <https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/>. [Zugegriffen: 23-Aug-2019].
- [16] J. Redmon und A. Farhadi, „YOLOv3: An Incremental Improvement“, S. 6.
- [17] N. D. Palo, „How I implemented iPhone X's FaceID using Deep Learning in Python.“, *Medium*, 05-Aug-2018. [Online]. Verfügbar unter: <https://towardsdatascience.com/how-i-implemented-iphone-xs-faceid-using-deep-learning-in-python-d5dbaa128e1d>. [Zugegriffen: 23-Aug-2019].

- [18] R. Horev, „Style-based GANs – Generating and Tuning Realistic Artificial Faces“, *Lyrn.AI*, 26-Dez-2018. [Online]. Verfügbar unter: <https://www.lyrn.ai/2018/12/26/a-style-based-generator-architecture-for-generative-adversarial-networks/>. [Zugegriffen: 24-Aug-2019].
- [19] J.-Y. Zhu, *Image-to-Image Translation in PyTorch. Contribute to junyanz/pytorch-CycleGAN-and-pix2pix development by creating an account on GitHub*. 2019.
- [20] „CycleGAN“. [Online]. Verfügbar unter: <https://hardikbansal.github.io/CycleGANBlog/>. [Zugegriffen: 25-Aug-2019].
- [21] CrashCourse, *Natural Language Processing: Crash Course Computer Science #36*, [Online]. Verfügbar unter: <https://www.youtube.com/watch?v=fOvTtapxa9c> [Zugegriffen: 24-Aug-2019]. 2017.
- [22] „Untertitel automatisch erstellen - YouTube-Hilfe“. [Online]. Verfügbar unter: <https://support.google.com/youtube/answer/6373554?hl=de>. [Zugegriffen: 24-Aug-2019].
- [23] „Cloud Text-to-Speech – Sprachsynthese | Cloud Text-to-Speech API“, *Google Cloud*. [Online]. Verfügbar unter: <https://cloud.google.com/text-to-speech/?hl=de>. [Zugegriffen: 25-Aug-2019].
- [24] „Google Text-to-Speech“, *Wikipedia*. 15-Aug-2019.
- [25] Jun-Yan Zhu, *Turning a horse video into a zebra video (by CycleGAN)*. 2017.
- [26] „Dreht Obama durch? Dieses Video zeigt eine der grössten Gefahren für Demokratien“, *bz - Zeitung für die Region Basel*. [Online]. Verfügbar unter: <https://www.aargauerzeitung.ch/leben/digital/dreht-obama-durch-dieses-video-zeigt-eine-der-groessten-gefahren-fuer-demokratien-132450970>. [Zugegriffen: 25-Aug-2019].
- [27] „Deeptrace | The antivirus for deepfakes“, *Deeptrace*. [Online]. Verfügbar unter: <https://www.deeptracelabs.com>. [Zugegriffen: 25-Aug-2019].
- [28] L. Kemp, „In the age of deepfakes, could virtual actors put humans out of business?“, *The Guardian*, 08-Juli-2019.
- [29] „Deepfake And What It Could Mean For The Film Industry“, *VFXwire*, 09-Juli-2019. [Online]. Verfügbar unter: <http://www.vfxwire.com/deepfake-and-what-it-could-mean-for-the-film-industry/>. [Zugegriffen: 25-Aug-2019].
- [30] P. Covington, J. Adams, und E. Sargin, „Deep Neural Networks for YouTube Recommendations“, in *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.
- [31] A. Yu, „How Netflix Uses AI, Data Science, and Machine Learning — From A Product Perspective“, *Medium*, 01-März-2019. [Online]. Verfügbar unter: <https://becominghuman.ai/how-netflix-uses-ai-and-machine-learning-a087614630fe>. [Zugegriffen: 11-Sep-2019].
- [32] K. Schmid, „Künstliche Intelligenz im Marketing - Warum und wozu?“, *Planet of Users*, 23-Mai-2018. [Online]. Verfügbar unter: <https://imbstudent.donau-uni.ac.at/kunde-4-0/kuenstliche-intelligenz-im-marketing/>. [Zugegriffen: 11-Sep-2019].
- [33] „Signalübertragung zwischen Nervenzellen: von Synapsen, Aktionspotenzial & Neurotransmitter“. [Online]. Verfügbar unter: <https://www.dasgehirn.info/grundlagen/kommunikation-der-zellen/nervenzellen-im-gespraech>. [Zugegriffen: 31-Okt-2019].

- [34] A. Paruthi, „Artificial Intelligence Hardware“, *Medium*, 18-Dez-2018. [Online]. Verfügbar unter: <https://becominghuman.ai/artificial-intelligence-hardware-76fa88581e53>. [Zugegriffen: 11-Sep-2019].
- [35] G. Baltazar, „CPU vs GPU in Machine Learning“. [Online]. Verfügbar unter: <https://www.datascience.com/blog/cpu-gpu-machine-learning>. [Zugegriffen: 11-Sep-2019].
- [36] J. Hale, „List of Deep Learning Cloud Service Providers“, *Medium*, 22-Okt-2018. [Online]. Verfügbar unter: <https://towardsdatascience.com/list-of-deep-learning-cloud-service-providers-579f2c769ed6>. [Zugegriffen: 15-Sep-2019].
- [37] „Home - Keras Documentation“. [Online]. Verfügbar unter: <https://keras.io/>. [Zugegriffen: 12-Sep-2019].
- [38] „Keras vs TensorFlow vs PyTorch | Deep Learning Frameworks“, *Edureka*, 05-Dez-2018. [Online]. Verfügbar unter: <https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/>. [Zugegriffen: 12-Sep-2019].
- [39] „What is TensorFlow? Introduction, Architecture & Example“. [Online]. Verfügbar unter: <https://www.guru99.com/what-is-tensorflow.html>. [Zugegriffen: 12-Sep-2019].
- [40] „Data for Deep Learning“, *SkyMind*. [Online]. Verfügbar unter: <http://skymind.ai/wiki/data-for-deep-learning>. [Zugegriffen: 15-Sep-2019].
- [41] S. Ghoneim, „5 Steps to correctly prepare your data for your machine learning model.“, *Medium*, 08-Apr-2019. [Online]. Verfügbar unter: <https://towardsdatascience.com/5-steps-to-correctly-prep-your-data-for-your-machine-learning-model-c06c24762b73>. [Zugegriffen: 15-Sep-2019].
- [42] „Data Preparation and Feature Engineering in ML | Data Preparation and Feature Engineering for Machine Learning“, *Google Developers*. [Online]. Verfügbar unter: <https://developers.google.com/machine-learning/data-prep/>. [Zugegriffen: 15-Sep-2019].
- [43] „Training and Test Sets: Splitting Data | Machine Learning Crash Course“, *Google Developers*. [Online]. Verfügbar unter: <https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data>. [Zugegriffen: 15-Sep-2019].
- [44] „Frölich - Einführung in Neuronale Netze.pdf“. .
- [45] „Convolutional Neural Networks - Aufbau, Funktion und Anwendungsgebiete“, *JAAI.de*, 06-Feb-2018. [Online]. Verfügbar unter: <https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/>. [Zugegriffen: 23-Sep-2019].
- [46] „Aber was *ist* nun ein neuronales Netzwerk? | Teil 1, Deep Learning [online] Verfügbar unter: https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi [Zugegriffen: 23-Sep-2019]. .
- [47] „Convolutional Neural Networks - The Math of Intelligence (Week 4) - YouTube“. [Online]. Verfügbar unter: <https://www.youtube.com/watch?v=FTr3n7uBluE>. [Zugegriffen: 24-Sep-2019].
- [48] P. Ganesh, „Types of Convolution Kernels: Simplified“, *Medium*, 18-Okt-2019. [Online]. Verfügbar unter: <https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37>. [Zugegriffen: 14-Feb-2020].

- [49] A. Escontrela, „Convolutional Neural Networks from the ground up“, *Medium*, 17-Juni-2018. [Online]. Verfügbar unter: <https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1>. [Zugegriffen: 24-Sep-2019].
- [50] blake west, „Building a Deep Neural Net In Google Sheets“, *Medium*, 11-Feb-2018. [Online]. Verfügbar unter: <https://towardsdatascience.com/building-a-deep-neural-net-in-google-sheets-49cdaf466da0>. [Zugegriffen: 24-Sep-2019].
- [51] J. Brownlee, „A Gentle Introduction to Pooling Layers for Convolutional Neural Networks“, *Machine Learning Mastery*, 21-Apr-2019. [Online]. Verfügbar unter: <https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/>. [Zugegriffen: 06-Okt-2019].
- [52] Packt_Pub, „Short Introduction to Convolutions and Pooling: Deep Learning 101!“, *Medium*, 27-Sep-2018. [Online]. Verfügbar unter: <https://medium.com/analytics-vidhya/deep-learning-methods-1700548a3093>. [Zugegriffen: 06-Okt-2019].
- [53] J. Jeong, „The Most Intuitive and Easiest Guide for Convolutional Neural Network“, *Medium*, 17-Juli-2019. [Online]. Verfügbar unter: <https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480>. [Zugegriffen: 06-Okt-2019].
- [54] „neural networks - What do the fully connected layers do in CNNs?“, *Cross Validated*. [Online]. Verfügbar unter: <https://stats.stackexchange.com/questions/182102/what-do-the-fully-connected-layers-do-in-cnns/182122>. [Zugegriffen: 06-Okt-2019].
- [55] A. S. V, „Understanding Activation Functions in Neural Networks“, *Medium*, 30-März-2017. [Online]. Verfügbar unter: <https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0>. [Zugegriffen: 05-Okt-2019].
- [56] J. Brownlee, „A Gentle Introduction to the Rectified Linear Unit (ReLU)“, *Machine Learning Mastery*, 08-Jan-2019. [Online]. Verfügbar unter: <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>. [Zugegriffen: 05-Okt-2019].
- [57] *Which Activation Function Should I Use?* <https://www.youtube.com/watch?v=-7scQpJT7uo>. .
- [58] H. Mahmood, „The Softmax Function, Simplified“, *Medium*, 26-Nov-2018. [Online]. Verfügbar unter: <https://towardsdatascience.com/softmax-function-simplified-714068bf8156>. [Zugegriffen: 05-Okt-2019].
- [59] F. Malik, „Neural Networks Bias And Weights“, *Medium*, 18-Mai-2019. [Online]. Verfügbar unter: <https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da>. [Zugegriffen: 06-Okt-2019].
- [60] „artificial intelligence - Role of Bias in Neural Networks“, *Stack Overflow*. [Online]. Verfügbar unter: <https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks>. [Zugegriffen: 06-Okt-2019].
- [61] T. Rashid, *Neuronale Netze selbst programmieren: Ein verständlicher Einstieg mit Python*. Heidelberg: O'Reilly, 2017.
- [62] *Gradient descent, how neural networks learn | Deep learning, chapter 2.* .
- [63] H. Bommana, „Loss Functions Explained“, *Medium*, 30-Sep-2019. [Online]. Verfügbar unter: <https://medium.com/deep-learning-demystified/loss-functions-explained-3098e8ff2b27>. [Zugegriffen: 02-Dez-2019].

- [64] „Gradient, Gradientenvektor, Schreibweise, Ableitung, mehrdimensionale Analysis - YouTube“. [Online]. Verfügbar unter:
<https://www.youtube.com/watch?v=sDXlFuma7CY>. [Zugegriffen: 06-Dez-2019].
- [65] *Was macht Backpropagation wirklich? | Kapitel 3, „Deep Learning“ [online]*
<https://www.youtube.com/watch?v=llg3gGewQ5U> [Zugegriffen: 02-Dez-2019]. .
- [66] A. MOAWAD, „Neural networks and backpropagation explained in a simple way“, *Medium*, 08-Okt-2019. [Online]. Verfügbar unter:
<https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e>. [Zugegriffen: 07-Dez-2019].
- [67] „H5 File Extension - What is an .h5 file and how do I open it?“ [Online]. Verfügbar unter: <https://fileinfo.com/extension/h5>. [Zugegriffen: 28-Dez-2019].
- [68] D. Mesquita, „Machine learning on mobile devices: 3 steps for deploying ML in your apps“, *Medium*, 25-Sep-2019. [Online]. Verfügbar unter:
<https://heartbeat.fritz.ai/machine-learning-on-mobile-devices-3-steps-for-deploying-it-in-your-apps-48a0a24364a8>. [Zugegriffen: 28-Dez-2019].
- [69] „Alphabet In AI: How Google Went From A Search Engine To An \$800B Global AI Powerhouse“, *CB Insights Research*. [Online]. Verfügbar unter:
<https://www.cbinsights.com/research/report/alphabet-google-artificial-intelligence/>. [Zugegriffen: 31-Dez-2019].
- [70] „Preise | Cloud Vision API-Dokumentation“, *Google Cloud*. [Online]. Verfügbar unter:
<https://cloud.google.com/vision/pricing?hl=de>. [Zugegriffen: 31-Dez-2019].
- [71] „Tesseract (Software)“, *Wikipedia*. 03-Dez-2019.
- [72] „Android OCR Application Based on Tesseract“. [Online]. Verfügbar unter:
<https://www.codeproject.com/Articles/1275580/Android-OCR-Application-Based-on-Tesseract>. [Zugegriffen: 01-Jan-2020].
- [73] „tess-two example Tesseract OCR - Apps on Google Play“. [Online]. Verfügbar unter:
https://play.google.com/store/apps/details?id=com.ashomok.tesseractsample&hl=en_US. [Zugegriffen: 04-Jan-2020].
- [74] „How to find parameters supported in Tesseract OCR config file“, *Stack Overflow*. [Online]. Verfügbar unter: <https://stackoverflow.com/questions/13007245/how-to-find-parameters-supported-in-tesseract-ocr-config-file>. [Zugegriffen: 01-Jan-2020].
- [75] S. Costa, „Why Firebase sucks“, *Medium*, 12-Okt-2018. [Online]. Verfügbar unter:
<https://medium.com/@scosta/why-firebase-sucks-ce5d2302eb20>. [Zugegriffen: 04-Jan-2020].
- [76] „What’s new at Firebase Summit 2018“, *The Firebase Blog*. [Online]. Verfügbar unter:
<http://firebase.googleblog.com/2018/10/whats-new-at-firebase-summit-2018.html>. [Zugegriffen: 04-Jan-2020].
- [77] Z. Sajjad, „Choose the Right On-Device Text Recognition (OCR) SDK on Android Using DeltaML“, *Medium*, 26-Sep-2019. [Online]. Verfügbar unter:
<https://heartbeat.fritz.ai/choose-the-right-on-device-text-recognition-sdk-on-android-using-deltaml-9b4b3e409b6e>. [Zugegriffen: 04-Jan-2020].
- [78] M. Spannring, <https://github.com/maxspannring/LateinPro>. 2020.
- [79] „Latein Pro - Latein Foto Übersetzer – Apps bei Google Play“. [Online]. Verfügbar unter:
<https://play.google.com/store/apps/details?id=com.lateinPro.lateinPro&hl=de>. [Zugegriffen: 05-Feb-2020].

- [80] „App Manifest Overview | Android-Entwickler“, *Android Developers*. [Online].
Verfügbar unter: <https://developer.android.com/guide/topics/manifest/manifest-intro?hl=de>. [Zugegriffen: 03-Feb-2020].
- [81] „Application Fundamentals | Android-Entwickler | Android Developers“. [Online].
Verfügbar unter:
<https://developer.android.com/guide/components/fundamentals?hl=de>.
[Zugegriffen: 03-Feb-2020].
- [82] „GitHub - ArthurHub/Android-Image-Cropper: Image Cropping Library for Android, optimized for Camera / Gallery.“ [Online]. Verfügbar unter:
<https://github.com/ArthurHub/Android-Image-Cropper>. [Zugegriffen: 03-Feb-2020].
- [83] „OkHttp“. [Online]. Verfügbar unter: <https://square.github.io/okhttp/>. [Zugegriffen: 05-Feb-2020].
- [84] „Alle Android-Versionen im Überblick“. [Online]. Verfügbar unter:
https://praxistipps.chip.de/alle-android-versionen-im-ueberblick_40765.
[Zugegriffen: 12-Feb-2020].

14. Abbildungsverzeichnis


Abbildung 1: Entwicklung von KI (Quelle: https://www.meetup.com/de-DE/SanFranciscoMachineLearning/)	2
Abbildung 2: Beispiel aus einem Beschwerde-E-Mail (Quelle: https://tone-analyzer-demo.ng.bluemix.net/)	4
Abbildung 3: Object-Detection mit Neuronalen Netzen (Quelle: https://mc.ai/part-2-fast-r-cnn-object-detection/)	5
Abbildung 4: Beispiel CycleGAN (Quelle: https://arxiv.org/pdf/1703.10593.pdf)	6
Abbildung 5: Vergleich Nervenzellen/künstliche Neuronen (Quelle: https://www.researchgate.net/figure/Model-of-a-human-brain-cell-neuron-A-and-an-artificial-neuron-B-used-in-Artificial_fig21_314761683)	8
Abbildung 6: Vereinfachter Aufbau eines neuronalen Netzes (Quelle: https://www.heise.de/select/ct/2016/6/1458191210995647)	11
Abbildung 7: Handgeschriebene Zahlen 0-9 (Quelle: http://programmersought.com/article/74931184251/)	12
Abbildung 8: Pixelmatrix einer Fünf (Quelle: https://mc.ai/deciphering-handwritten-numbers-with-a-neural-network/)	12
Abbildung 9: Abstraktion von Bildern im menschlichen Gehirn (Quelle: https://hackernoon.com/mit-6-s094-deep-learning-for-self-driving-cars-2018-lecture-4-notes-computer-vision-f591f14b3b99)	13
Abbildung 10: Convolution Matrix 1 (Quelle: https://lipyeow.github.io/ics491f17/morea/deepnn/ImageClassification-CNN.pdf)	14
Abbildung 11: Convolution Matrix 2 (Quelle: https://lipyeow.github.io/ics491f17/morea/deepnn/ImageClassification-CNN.pdf)	14
Abbildung 12: Flattening + FC Layer (Quelle: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480)	15

Abbildung 13:Stufenfunktion (Quelle: https://sites.google.com/site/calcoloamo1976/assignments)	16
Abbildung 14: Berechnung des Neuronen Wertes (Quelle: https://medium.com/@abhismatrix/neural-network-model-training-tricks-61254a2a1f6b)	17
Abbildung 15: Funktion mit mehreren Minimumstellen (Quelle: https://proglib.io/p/neural-network-course/?fbclid=IwAR0tRdSKzsSdhg8zUkZhQ2TknzwHi_10FcccBXLqvB0bbUQCU0erRMu1k5g).....	19
Abbildung 16: Gradienten Verfahren in 3D (Quelle: https://github.com/RNogales94/ISI-Tensorflow/blob/master/README.md)	20
Abbildung 17: Übersicht über den Trainingsablauf (Quelle: https://www.researchgate.net/figure/The-representation-of-neural-network-training-process_fig5_299390844)	21
Abbildung 18: Fortschrittsverlauf während dem Training (Quelle: https://medium.com/@pratyush057/elements-of-machine-learning-e09ebf16af19)	22
Abbildung 19: Tesseract Beispiel Output (Quelle: https://www.codeproject.com/Articles/1275580/Android-OCR-Application-Based-on-Tesseract).....	25
Abbildung 20: Tesseract Beispiel Input (Quelle: https://www.codeproject.com/Articles/1275580/Android-OCR-Application-Based-on-Tesseract).....	25
Abbildung 21: Mein Tesseract Test Output (Quelle: Selbst erstellt)	25
Abbildung 22: Mein Tesseract Test Input (Quelle: Selbst erstellt)	25
Abbildung 23: „tess-two example Tesseract OCR“ von Yuliia Ashomok (Quelle: Selbst erstellt)	26
Abbildung 24: Ergebnis meines Firebase Tests (Quelle: Selbst erstellt)	28
Abbildung 25: Programmfluss von Latein Pro (Quelle: Selbst erstellt)	31

Selbstständigkeitserklärung

Ich erkläre, dass ich diese Vorwissenschaftliche Arbeit eigenständig und ausschließlich unter Zuhilfenahme der im Literaturverzeichnis angeführten Fachliteratur verfasst habe.

Salzburg, am 18. Februar 2020

Unterschrift:  M. G. Springer