

Bilderkennung anhand von Blättern mit faltenden Neuronalen Netzwerken

Besondere Lernleistung im Fachbereich Informatik

Schule: Gymnasium Musterschule
Betreuer: Hans Dietmar Jäger

Frederik Glitzner

18. März 2019

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Die Stellen der Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Frankfurt, 18. März 2019

Frederik Glitzner

Zusammenfassung

Nach einer Einführung in das Thema mit Grundlagen wie Maschinelles Lernen, Künstliche Intelligenz und künstliche Neuronale Netze, sowie der Erklärung der nötigen theoretischen Grundlagen für Fotos und deren Verarbeitung wird ein Fokus auf die Funktionsweise und Entwicklung faltender Neuronaler Netze gelegt. Hierbei werden über die Arbeitsweise hinweg praktische und populäre Ansätze gezeigt und verglichen. Schließlich werden eigene Ansätze und Architekturen entwickelt und verglichen, um einen selbsterstellten Bild-Datensatz bestehend aus gescannten Blättern von Pflanzen zu klassifizieren. Eine erfolgreiche Entwicklung gelingt, liefert eine gute Genauigkeit und bietet eine umfangreiche Diskussionsgrundlage.

Glossar

Abkürzung	Bedeutung
KI	Künstliche Intelligenz
NN	Künstliches Neuronales Netzwerk
CNN	Convolutional Neural Network
CL	Convolutional Layer
LR	Lernrate im Training
S	Signal
N	Neuron
V	Verbindung
Loss	(Hier:) Fehler
MSE	Mittlere quadratische Abweichung
Gradient	Richtung des steilsten Anstiegs einer Funktion

Inhaltsverzeichnis

1 Motivation und Ziele der Arbeit	2
2 Künstliche Intelligenz und Maschinelles Lernen	4
3 Künstliche Neuronale Netze	6
4 CNN und die Herangehensweise	10
5 Der Datensatz	15
5.1 Digitale Fotos	15
5.2 Kreation des Datensatzes	17
6 Das Netz	21
6.1 Die Theorie	21
6.1.1 Architektur-Theorie	24
6.1.2 Optimierung	28
6.2 Die Erwartung	31
6.3 Historische Durchbrüche und Erfolge	32
6.4 Eigene Architekturen im Vergleich	37
6.4.1 Präsentation und Auswertung verschiedener Konfigurationen .	38
6.4.2 Diskussion der Architektur-Ergebnisse	54
7 Variation im Training	56
7.1 Verwendung von SW-Bildern	56
7.2 Der Wert guter Daten	58
8 Zusammenfassung der Ergebnisse	60

Kapitel 1

Motivation und Ziele der Arbeit

Ziel dieser Arbeit ist es das in der Bearbeitungszeit selbstständig angereicherte Wissen über Convolutional Neural Networks (CNN) strukturiert und informativ anhand des produktorientierten Anwendungsfalls der Pflanzenidentifizierung mit maschinell selbsterlernter Bilderkennung von Blattabbildungen zu präsentieren. Hier stelle ich also nicht nur die Frage, ob eine solche Implementierung funktioniert und erstelle eine praktische Anwendung, sondern probiere, die Fragen wie und warum es funktioniert ebenfalls zu erklären. Es soll Licht in die abstrakte Funktionsweise der Netzwerke gebracht werden, um in diesem Bereich weiter zu forschen und eigene Entdeckungen zu gewinnen. Die Implementierung findet im Bereich des Möglichen für eine nicht-hochprofessionelle technische Ausstattung statt, da beispielsweise Rückgriffe auf extrem leistungsstarke Computer nicht möglich waren.

Die Anmeldung des Projektes war sicherlich beeinflusst von einem hohen Grad an Vertrauen in mich und die Technik, denn vor der Erarbeitung hatte ich mich nicht praktisch mit dem Thema auseinandergesetzt, sondern lediglich einige Beiträge in Fachzeitschriften und wissenschaftlichen Veröffentlichungen gelesen, die mein Interesse stark geweckt hatten. Zuvor hatte ich also nahezu keinerlei Berührung mit diesem spezifischen Themengebiet, da es nicht leicht ist, hier einen Einstieg zu bekommen. Die erste Begegnung damit hatte ich auf einer Veranstaltung eines 24-stündigen 'Hackathons' im Januar 2017, die ich verwirrt aber fasziniert verlassen habe. Immerhin wurde das Feld 'Machine Learning' (ML) erst kürzlich als Wahlmodul zu technischen Studiengängen wie Informatik an einigen Universitäten hinzugefügt, während ein großer Teil der Entwickler in diesem Bereich promoviert ist. Weder die Anwendung noch die Theorie war mir bekannt. Dazu kommt, dass weder jemand in meiner Familie noch jemand in meinem engen Bekanntenkreis einen technischen Beruf ausübt, den ich um Rat hätte fragen können.

Zu meiner persönlichen Laufbahn im Bereich Informatik lässt sich sagen, dass ich bereits ab der 5. Klasse meine ersten Webseiten mit Zugriff auf den Schulserver entwickelt habe und dieses Lernen mit dem über Server und das Internet, die zur Veröffentlichung nötig waren, verbunden habe. Mit seit jeher geweckter Neugierde habe ich mich kurz darauf auch intensiver mit den elektrotechnischen Grundlagen für Computer vertraut gemacht, was zunächst im Selbstbau eines Personal Computers gipfelte und mir auch bei diesem Projekt hier half das richtige Equipment auszuwählen und die nötigen Software und Treiber zu installieren. In den folgenden

Jahren legte ich den Fokus eher auf Webdesign und die Entwicklung von Website-Backend Anwendungen, sowie dem Erlernen von mehreren Programmiersprachen, was mir hier half das Projekt praktisch zu realisieren. In der elften Klasse machte ich ein zweiwöchiges Betriebspraktikum bei einer IT-Firma, die hauptsächlich für die Bereitstellung großer Datensätze mit Finanz- und Unternehmensdaten verantwortlich ist und dies im Auftrag unter anderem der G20 und des Financial Board of Stability ausführt. Hier lernte ich ebenfalls mich eigenständigen Projekten effizient und erfolgreich zu widmen und besonders informatische Probleme schnell und sinnvoll zu lösen, ebenso wie den Umgang mit großen Datenmengen, deren Visualisierung, Informationssammlung, -gewinnung und -verarbeitung. Die erfolgreiche Entwicklung und Präsentation meiner Arbeit dort bezüglich der Darstellung von Unternehmensbeziehungen auf mehreren Ebenen als Graph stellte einen persönlichen Meilenstein in der Welt der Informatik für mich dar.

Meine persönliche Motivation für dieses Projekt lag am Anfang und liegt nach wie vor in dem großen Reiz der selbstständigen Erarbeitung eines komplexen Themas, der zeitgenössischen wie zukünftigen Relevanz und Präsenz von Techniken des ML, zu denen Neuronale Netzwerke gehören, sowie der intensiven Auseinandersetzung mit diesem informatischen Konzept. Das dem Erlernten zugrundeliegende Wissen stammt primär aus umfangreichen Recherchen in Textbüchern, dem Internet und Forschungsarbeiten zu diesem und korrespondierenden Themengebieten.

Kapitel 2

Künstliche Intelligenz und Maschinelles Lernen

Eine Künstliche Intelligenz (KI) könnte vieles sein und es existiert keine einheitliche Definition des Begriffs, beziehungsweise was eine KI ausmachen würde, sollte sie existieren. Das Feld wird akademisch meistens der Informatik oder der Kognitionswissenschaft zugeordnet - eine Emanzipation als eigenständige Wissenschaft ist nicht vollständig eingetreten, da dieses Feld eher eine Überschneidung vieler Elementarwissenschaften zu einem bestimmten Zweck darstellt und auf deren Methoden und Wissensschatz zurückgreift. Eine KI, so die umstrittene Definition von Sharkey (2009), sei eine Maschine, die Dinge tue, die uns glauben ließen, sie sei intelligent. Dies ist eine Abweichung der bekannten Definition von Marvin Minsky, nach der eine KI Dinge tue, die, wenn von Menschen ausgeführt, Intelligenz benötigen würden (Sharkey, 2009). Generell wird eine KI wohl als komplexes System angesehen, das selbstständig Dinge tun kann, die als anspruchsvoll in einer abstrakten Art angesehen werden, wie Entscheidungen zu treffen¹ oder Arbeitsabläufe zu koordinieren.² Das Feld ist in einem Entwicklungshoch und wird auch verbunden mit dem Internet der Dinge (IoT - ein Cluster von verbundenen Geräten im alltäglichen Leben) und Robotik.

Eine von vielen denkbaren Möglichkeiten für eine Superintelligenz stellt die computergestützte Künstliche Intelligenz dar. Im Vergleich zu anderen Ansätzen sieht Bostrom (2014) diese Möglichkeit sogar als überdurchschnittlich wahrscheinlich an. Die Entwicklung eines Systems, das sich selbst durch Evolution verbessert und so eine ähnliche Entwicklung wie die der menschlichen Intelligenz erfährt, wäre der dementsprechende Weg.

Der computergestützte Anteil dieser denkbaren Künstlichen Intelligenz könnte nach heutigem Erkenntnisstand vor allem durch Maschinelles Lernen ermöglicht werden - ein Feld, welches der Informatik zugeordnet werden kann. Hier gibt es ebenfalls wieder verschiedene Methoden, die entwickelt wurden und werden. Es geht meistens um die Verarbeitung von Daten mit dem Ziel eines eintretenden 'Lernef-

¹Im Jahr 1997 hat der von IBM entwickelte Computer 'Deep Blue' im Schachspiel gegen den Meister Garry Kasparov gewonnen. Eine solche Implementierung mit überlegenen, komplexen Analysen von Spielzügen galt lange als unmögliche Aufgabe für einen Computer.

²'Watson', eine general purpose (Allzweck-)KI von IBM, wird unter anderem für Behandlungsentscheidungen für Patienten von Doktoren und Krankenschwestern genutzt.

fekts³ bei der Maschine, der sich beispielsweise durch probabilistische Voraussagen von Parametern oder Zugehörigkeiten eines bestimmten Falls im Szenario der Anwendungsentwicklung äußert. Praktisch sind solche Algorithmen unter anderem auf Online-Plattformen implementiert. Wie Armbruster und LeCun (2018) im Gespräch berichten, funktioniere die Plattform Facebook ohne Künstliche Intelligenz nicht. Auf die Frage, wo KI implementiert sei, antwortet LeCun, 'die ganzen Inhalte, die du siehst, und die ganzen Inhalte, die du nicht siehst - [...] beide sind überwiegend durch KI-Systeme bestimmt.'

Konzepte des Maschinellen Lernens können klassischerweise in zwei Gruppen aufgeteilt werden: 'supervised learning' (überwachtes Lernen) und 'unsupervised learning' (unüberwachtes Lernen). Während bei ersterem ein 'labeled' (beschrifteter) Datensatz vorliegt, bei dem einzelne Objekte bereits Kategorien oder Werten zugeordnet sind, die nach dem Lernen eigenständig von der Maschine ermittelt werden sollen, sind Datensätze für unüberwachtes Lernen oft unsortiert und für experimentellere Anwendungen geeignet. Um bei der Anwendung auf Online-Plattformen zu bleiben, könnte überwachtes Lernen eingesetzt werden, um anhand von Erfahrungswerten des Verhaltens einen Benutzer in Kategorien einzuteilen wie *Alter*, und unüberwachtes Lernen, um auf Sachkenntnis gestützte Vermutungen darüber zu treffen, an welchen Nachrichten und Veröffentlichungen ein bestimmter Benutzer interessiert sein könnte. Unüberwachtes Lernen kann vor allem auf Basis von Verhaltensdaten wie kausalen Beziehungen implementiert werden; für überwachtes Lernen kann meist ein tabellarischer Datensatz genutzt werden.

Künstliche Neuronale Netzwerke (NN) sind eine der bekanntesten Methoden überwachten Lernens, die bereits einen hohen Entwicklungsgrad aufweisen.

³Der Begriff 'Lernen' ist bei Maschinen möglicherweise irreführend. Unter 'lernen' wird hier verstanden, mit fortlaufender Zeit oder Entwicklung bessere Ergebnisse bei einer bestimmten Aufgabe erzielen zu können.

Kapitel 3

Künstliche Neuronale Netze

Zunächst sollte man sich bei der Vorstellung von NN von den grundlegenden Ideen konventioneller Computer trennen. Ein Computer kann klassisch aufgeteilt werden in verschiedene physische Bestandteile - Teile der Hardware wie Prozessor, Speicher und Eingabegerät - und unsichtbare Bestandteile wie Programme und gespeicherte Daten. Ein Programm spricht hier nach einem festen und vorprogrammierten Ablauf verschiedene Teile der Hardware an und führt diese Operationen klassischerweise seriell aus. Im Gegensatz dazu ist ein NN eher eine 'Assoziationsmaschine', wie Stanley und Bak (1991) schreiben, die 'keinen getrennten Speicher für die Aufbewahrung von Daten' besitzt, sondern als System zu sehen ist, dass gruppenweise und graphenartig organisiert ist. Im Kontrast zu einem klassischen Programm, bei dem Operationen standardmäßig hintereinander ausgeführt werden, werden bei einem Neuronalen Netzwerk meist viele Operationen gleichzeitig ausgeführt, die stets gruppierbare Zwischen- oder Gesamtergebnisse liefern, als solche aber relativ ausdruckslos sind.

Die Idee NN zu schaffen kommt von der Erkenntnis, dass der Ursprung menschlicher Intelligenz und das menschliche Denkvermögen im Gehirn verankert sind. Nach elementarer Forschung im Bereich der Neurologie - der Wissenschaft und Lehre vom Nervensystem - wurde die Funktionsweise des Gehirns als Neuronales Netzwerk modelliert. Grundlegend und stark vereinfacht kann man sich dieses als Anordnung von Neuronen vorstellen. Neuronen sind Nervenzellen, die Signale als Input erhalten und bei Übertreffen eines Schwellenpotentials ein Aktionspotential auslösen - dann feuert die Nervenzelle. Das vorhandene Potential kann anschließend übertragen werden an weitere Nervenzellen. Generell gilt, dass die ankommenden Signale in ihrem Potential abgesehen von ihrer Ursprungsstärke auch abhängig von ihrem zurückgelegten Weg zum Neuron durch die Verbindung sind. Längere Strecken führen zu einer kleineren Signalstärke. Gleichzeitig ankommende Signale können sich überlagern und so ein stärkeres Signal bewirken. Das Gehirn eines Menschen besitzt ungefähr 100 Milliarden Neuronen und zehn Millionen Milliarden Verbindungen (Stanley und Bak, 1991).

Aufgrund der Ergebnisse, die der Mensch mit diesem System Gehirn erzielen kann, soll dieses biologische und natürliche System modellhaft übernommen und mit Mitteln der Informatik simuliert werden. Hierzu wird angenommen, dass die Signale numerisch verstanden und verarbeitet werden können. In einer minimalen

Analogie zu dem biologischen System finden sich also Signale S , Neuronen N und Verbindungen V .

S können je nach Anwendungsfall verschiedene Formate sein. Hier wird von einem numerischen Datensatz ausgegangen; entsprechend sind S numerische Werte.

N sind zunächst, in der ersten Ebene des Netzwerks folgend auf die Eingangssignale, gekoppelt an die ursprünglichen S . Ihre Aktivierung (Aktionspotential) setzt sich aus einer gewichteten Synthese der eingegangenen S zusammen und wird, je nach Erfolg des zu erreichenden Schwellenpotentials, an die N der folgenden Ebene weitergegeben. Diese Weitergabe über verschiedene Ebenen hinweg kann theoretisch bis zur maximal möglichen Berechenbarkeit wiederholt werden.

V bestehen zunächst zwischen der ursprünglichen Signaleingabeschicht und der ersten Schicht N , anschließend zwischen den Ebenen von N , bei einem vorwärtsgerichteten NN jedoch nicht innerhalb einer Ebene und nicht zu vorherigen Ebenen. Sie transportieren Aktivierungen beziehungsweise S . Der Unterschied, der biologisch durch den Weg der V verursacht wird, wird hier durch eine faktorielle Gewichtung¹ der Verbindung simuliert.

Folgend kommt eine Einführung in die elementare mathematische Funktionsweise von NN auf Basis des Perzeptrons von Rosenblatt (1957).

Der ankommende Signalwert sei gegeben durch x , der Ausgabewert des Neurons sei o (hier mit dem Index $n1$ für ein Neuron 1), das Gewicht einer Verbindung sei w . Der Parameter j sei eine Variable für das spezifische Signal, da mehrere Signale parallel ankommen können.

$$o_{n1} = \sum_{j=0}^{\hat{n}} w_j * x_j$$

Da die Gewichte und ankommenden Signale auch als j -dimensionale Vektor dargestellt werden können, entspricht das auch dem Skalarprodukt

$$o_{n1} = \vec{w} \cdot \vec{x}$$

Da ein bestimmtes Schwellenpotential t erreicht werden muss, damit das Neuron feuert, ist die finale Ausgabe o abschnittsweise definiert durch

$$o = \begin{cases} 0 & \text{falls } o_{n1} \leq t \\ o_{n1} & \text{falls } o_{n1} > t \end{cases}$$

Um diese Bedingung direkt in die Funktion für das Ausgabesignal zu integrieren, wird eine Tendenz $b = -t$ eingeführt. Dementsprechend gilt für das Ausgabesignal (hier indexiert mit p in Anlehnung an das gleich vorgestellte Perzeptron-Modell):

$$o_{p1} = \sum_{j=1}^{\hat{n}} (w_j * x_j) + b$$

¹Durch die Höhe des Schwellenpotentials zur erfolgreichen Aktivierung wird ebenfalls eine Gewichtung der Signale vorgenommen, jedoch hat diese für ein jeweiliges N stets einen konstanten

Jedes N hat genau eine Tendenz b und genau so viele Gewichte w wie ankommende Signale x .

$$\Rightarrow o_p = \begin{cases} 0 & \text{falls } o_{p1} \leq 0 \\ o_{p1} & \text{falls } o_{p1} > 0 \end{cases} \quad (3.1)$$

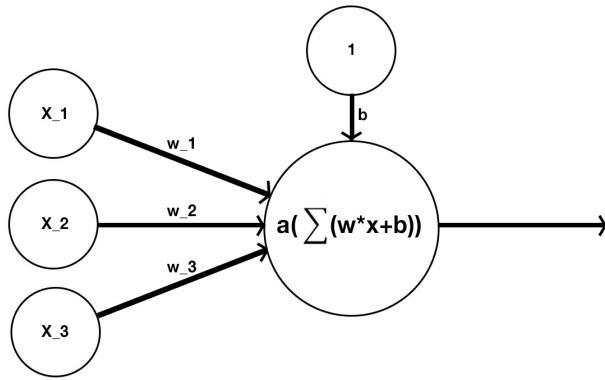


Abbildung 3.1: Perzeptron-Modell mit einem Neuron. a steht für eine beliebige Aktivierungsfunktion siehe nächster Abschnitt.

Das obige Modell nach Gleichung 3.1 basiert auf dem bekannten Perzeptron-Algorithmus von Rosenblatt (1957), der diesen für die binäre lineare Klassifikation entwickelt hat. Schichtet man diese Art Neuron vertikal in einer einzelnen Schicht und horizontal mehrere Schichten hintereinander², wobei die Aktivierungen aller Neuronen einer Schicht als Signale aller Neuronen der nächsten Schicht gelten und durch Verbindungen sequentiell verknüpft sind, so entsteht ein einfaches, strukturiertes und symmetrisches NN. Das Ziel ist es dann mit einem Lernalgorithmus die Parameter w und b anzupassen, sodass ein sinnvolles Ergebnis in der Implementierung - beispielsweise zur Klassifikation der ursprünglichen Signale - berechnet und erzielt werden kann. Die Ebenen, auch genannt Schichten, zwischen den ursprünglichen Eingangssignalen und der finalen Schicht Ausgabesignale werden im Englischen 'Hidden Layers', also verdeckte Schichten, genannt, da ihre genaue Funktionsweise aufgrund der selbsterlernten Parameter ungesteuert und uneinsichtig von menschlichen Entwicklern abläuft, die nur die Architektur des Netzwerkes steuern und entwerfen. Ein künstliches Neuronales Netzwerk mit mindestens zwei verdeckten Schichten wird auch 'Deep Neural Network', also tiefes Neuronales Netzwerk, genannt.

Da, wie später gezeigt wird, der Algorithmus zum Trainieren der Parameter w und b die Werte direkt verändert, kann es zu sehr großen Schwankungen kommen. Sollte zum Beispiel ein Gewicht erhöht, das Eingabesignal jedoch nicht normalisiert³ werden, so wird o möglicherweise direkt stark vergrößert. Als Maßnahme zur Normalisierung gibt es Aktivierungsfunktionen; für viele Anwendungsfälle hat sich

Einfluss und verändert nicht das ankommende S .

²Diese Anordnung wird nur zu Vorstellungszwecken verwendet - aufgrund der Graphen-Struktur mit stets gerichteten Kanten ist es jedoch nicht relevant, wie die Visualisierung des Layouts stattfindet

³Das Konzept der Normalisierung von Werten wird immer wieder auftreten. Eine Normalisierung bewirkt oft eine Angleichung der vorhandenen Werte aneinander, sodass die Diskrepanz

die biologisch inspirierte Sigmoidfunktion durchgesetzt, die in ihrem Verlauf etwa der Aktivierungsfunktion der menschlichen Neuronen im Gehirn gleichen soll. So wurde das 'Sigmoid-Neuron' kreiert, dessen Funktionsweise elementar wie die des Perzeptron-Neurons ist, jedoch mit der Erweiterung, dass die Ausgabe o_p vor der Weitergabe an folgende Schichten normalisiert wird.

Die logistische Sigmoidfunktion ist definiert als

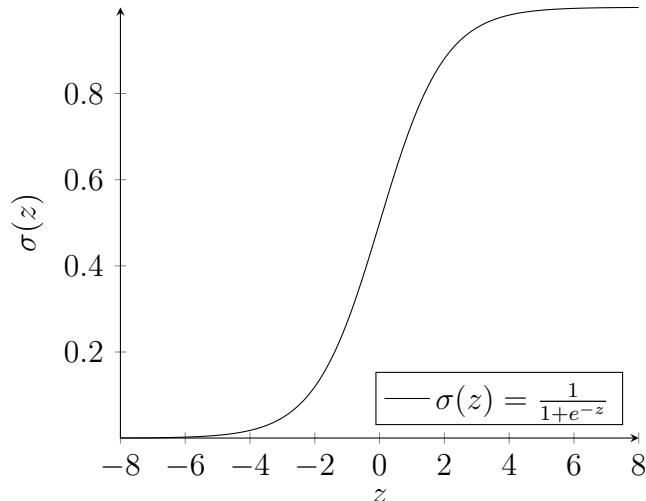
$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (3.2)$$

Die Funktion für das Ausgabesignal o_{p1} eingesetzt in Gleichung 2 ergibt

$$\sigma(o_{p1}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}} \quad (3.3)$$

Die Aktivierungsfunktion $\sigma(o_{p1})$ in Gleichung 3 strebt für

$$\lim_{o_{p1} \rightarrow -\infty} \sigma(o_{p1}) = 0 \wedge \lim_{o_{p1} \rightarrow \infty} \sigma(o_{p1}) = 1$$



In der aktuellen Praxis gibt es eine Vielzahl von Variationen dieser Funktion, die ein ähnliches Verhalten aufweisen. Dieser Funktionstyp wird nicht nur wegen des normalisierenden Verhaltens gewählt, sondern auch wegen der relativ leichten Differenzierbarkeit der Funktion.

bei großen Schwankungen, die zum Beispiel aufgrund von fehlerhaften Daten aufkommen kann, verringert wird und das Endergebnis nicht zu einem unbrauchbaren Ergebnis manipuliert. In dieser speziellen Anwendung hier werden die Ausgabesignale normalisiert, damit extreme Werte eines Neurons angepasst werden und nicht die ganze Berechnung verändern. Dies könnte auftreten, wenn ein Gewicht w im Training fehlerhaft initialisiert wurde.

Kapitel 4

CNN und die Herangehensweise

Klassische NN, beispielsweise nach dem erklärten Perzeptron-Modell, sind häufig geeignet zum Verarbeiten numerischer Daten in Tabellenform. Dies könnte eine Regression oder Klassifikation angewandt auf einen Datensatz, gespeichert in einer Tabelle einer Datenbank, sein. Denkbar und aktiv implementiert ist eine Berechnung der zu erwartenden Mietkosten einer neuen Mietwohnung als Regressionsproblem oder analog eine Einordnung dieser Wohnung in eine Preiskategorie als Klassifikationsproblem. Hierbei könnten Attribute wie *Lage*, *Entfernung*, *Zimmeranzahl*, *Groesse*, *Baujahr* eine Rolle spielen, deren Anzahl dann im System mathematisch als Dimensionen eines Vektorraumes einzuordnen wären.¹ Bei Erweiterung des Vektorraums um das Attribut *Preis* könnte der Algorithmus nun eine Hyperebene berechnen, die den bekannten Datensatz mit dem Attribut *Preis* in Bezug zu den anderen Variablen beschreibt. Idealerweise, danach wird stets gestrebt, wäre diese Ebene generalisiert², sodass sie für die potentielle Preisberechnung der neuen Wohnung geeignet wäre.

Das in dieser Arbeit angegangene Problem der Klassifikation von Fotos mit einem NN, also der Bilderkennung mit Hilfe Maschinellen Lernens, wäre mit einem klassischen NN jedoch nicht oder nur sehr schwer möglich. Das ganze wäre in der Theorie machbar: Jeder Bildpunkt (siehe Kapitel 5.1) könnte als Attribut des Bildes gesehen werden, die gesamten Aktivierungen aller Bildpunkte könnten dementsprechend als individuelle und eindeutige Zusammensetzung von Attributwerten eines Bildes gesehen werden. Diese Verarbeitung hätte jedoch mehrere Nachteile:

1. Es wäre ein enormer Rechenaufwand nötig, um jeden Bildpunkt (im NN wiederzufinden als Eingabesignal S) mit jedem künstlichen Neuron N der nachfolgenden Ebene zu verknüpfen und alle Neuronen nachfolgender Schichten ebenfalls entsprechend zu koppeln.
2. Die Dreidimensionalität des Bildes würde verloren gehen, da die zweidimensional angeordneten Bildpunkte umstrukturiert werden müssten zu einem Feld oder Vektor.

¹Die fünf Attribute würden einen fünfdimensionalen Vektorraum ergeben, in dem Wohnungen abgebildet werden könnten.

²Mit Generalisierung, in Abgrenzung zur Normalisierung, wird der Prozess beschrieben, bei dem in diesem Fall eine Formel oder ein System möglichst allgemeingültig gemacht wird und nicht nur in einem bestimmten Fall, zum Beispiel bei einem ganz bestimmten Datensatz, funktioniert und gilt.

3. Das 'Erlernen' von Mustern in dem Bild würde sich schwierig gestalten, da das Netzwerk aufgrund der diskreten und genauen Pixelaktivierung nur wenig generalisiert wäre. Ein Datensatz mit niedriger Gleichmäßigkeit, beispielsweise häufig verschobenen Elementen im Bild (position difference) wie der Sonne im Himmel links oder rechts, wäre schwer zu klassifizieren, da die Ebene über den Attributen eine starke Angleichung an naheliegende Bildpunkte nötig hätte (faltende Neuronale Netze werden diesem Problem durch verschiedene Methoden gerecht, selbst bei weniger stark generalisierten Daten (Warden, 2017)).

Zur Lösung dieser Probleme wurde eine neue Art von NN entwickelt, zu dessen Theorie Yann LeCun maßgeblich beigetragen hat. Es nennt sich 'Convolutional Neural Network' (wörtlich übersetzt: faltendes NN) und wendet eine andere Methode zur Verarbeitung der Bildpunkte an. Es werden zur Klassifikation, wie in Kapitel 6.1 genauer erläutert, nicht mehr Gewichte und Tendenzen für jeden eingegebenen Bildpunkt berechnet; sie werden nun zu einzelnen Filtern zusammengefasst, die nicht die Dimensionen des Bildes haben, sondern, anschaulich erklärt, über das Bild geschoben werden. Hierbei herrscht 'weight-sharing', also Gewichtsteilung, vor, sodass dieselben Gewichte für den kompletten Filter, der jedoch aus mehreren Neuronen besteht, gelten. Das ist eine klare Minderung des Rechen- und Speicheraufwands.

Es gibt also keine individuelle Behandlung der Bildpunkte mehr, sondern eine Verarbeitung von Bildpunktgruppen, wobei jeder einzelne Bildpunkt natürlich weiterhin einen Effekt auf das Ergebnis hat. Diese Methode bietet für Bilderkennung eine effizientere und zugleich genauere Einordnung als das klassische NN ohne Informationen zu verlieren.

Ziel dieser Arbeit ist es, Bilder von pflanzlichen Blättern zu erkennen und diese zu klassifizieren, also die Zugehörigkeit zu einer von mehreren möglichen Kategorien maschinell zu erfassen.

Nun zur Herangehensweise. Da, wie oben erläutert, das NN gehirnähnliche Merkmale bezüglich der Entscheidungsfindung aufweist, gilt es zu erörtern, auf welcher Basis das menschliche Gehirn Blätter vermutlich einordnet. Die Vermutung ist, dass das Gehirn alle zur Verfügung stehenden Informationen kompilieren würde und auf Basis von Gewichtungen der Einzelemente einen Schluss ziehen würde. Zur Verfügung stehen hier standardmäßig die erfassbaren Daten der fünf menschlichen Sinne:

1. Auditive Wahrnehmung: Die Vermutung liegt nahe, dass das Hören bei der Blatterkennung maximal eine geringe Rolle spielt.
2. Olfaktorische Wahrnehmung: Der Geruchssinn hat vermutlich eine stärkere Funktion bei dieser Aufgabe als die auditive Wahrnehmung, jedoch sind die meisten Blattarten für Menschen nahezu bis komplett geruchslos.
3. Gustatorische Wahrnehmung: Der Geschmack fällt bei der Problemlösung komplett weg, da ein Mensch standardmäßig keine nicht-identifizierten Gewächse, was ja hier gerade die Aufgabe ist, verkostet.
4. Taktile Wahrnehmung: Dem Tastsinn fällt, meiner Annahme nach, die zweit-

größte Gewichtung zu. Vermutlich ließe sich die Aufgabe bei einer sehr selektiven Auswahl an Arten allein durch das Gefühl bewältigen. Durch die taktile Wahrnehmung kann beispielsweise die Größe, die Verzweigungen, die Form, die Struktur, die Oberfläche, die Tiefe, die Härte und die Konsistenz wahrgenommen oder geschätzt werden.

5. Visuelle Wahrnehmung: Den Informationen, die das Sehen liefert, fällt meiner Hypothese nach jedoch die größte Gewichtung zu. Hier wiederholen sich Aspekte, die möglicherweise auch bei der taktilen Wahrnehmung festgestellt werden können. Zu den visuell wahrnehmbaren oder qualitativ einschätzbaren Attributen gehört die Größe, die Verzweigung, die Form, die Struktur und die Farbe des Blattes.

Von der Annahme ausgehend, dass Blätter anhand qualitativer Einschätzungen auf der Basis visueller Wahrnehmungen von einem Gehirn eindeutig identifiziert werden können, sollte es möglich sein ein CNN zu entwickeln, welches die Aufgabe des Gehirnes imitiert und diese Entscheidungen näherungsweise gleich trifft. Bei der Nutzung eines CNN anstelle eines menschlichen Gehirnes treten jedoch einige möglicherweise schwerwiegende Nachteile auf: Ein CNN nutzt, nach heutigem Entwicklungsstand, nur eine zweidimensionale Anordnung von Bildpunkten bei Fotos als Eingabe (abgesehen von der Kanaltiefe, die durch verschiedene Farbkanäle zustande kommt). Anhand dieser Eingabe müssen alle nötigen Attributwerte festgestellt werden, die zu einer erfolgreichen Einordnung benötigt werden. Durch die Zweidimensionalität geht jedoch die Tiefe des räumlichen Wahrnehmungsvermögens verloren, wobei einige menschlich erfassbaren Daten wegfallen. Hier ist die Dicke des Blattes denkbar. Außerdem können technische Schwierigkeiten wie eine falsche Farbkalibrierung oder ein schräger Blickwinkel auf das Blatt auftreten. Des Weiteren sind an die Datenqualität enorme Anforderungen gestellt, abgesehen von der großen Datenmenge, die das Netzwerk zum Trainieren benötigt. Hinzu kommt, dass die Größe des Blattes durch die fototechnische Verarbeitung vereinheitlicht wird und deswegen eine Bewertung dieses Aspektes unmöglich ist.

Festzuhalten ist, dass an das CNN hohe Anforderungen gestellt sind und dass es die Aufgabe des menschlichen Gehirnes bei der Pflanzenidentifizierung imitieren soll durch die Abwägung und Gewichtung eigenständig extrahierter Attributwerte aus einer visuellen Datenquelle. Außerdem besteht aus informatischer Sicht eine Schwierigkeit darin, dass der Computer kein Assoziationsvermögen wie das menschliche Gehirn hat. Beispiel: Die Form wird wenn überhaupt durch 'das Blatt ähnelt in der Form einem Kreis' als durch 'das Blatt hat die Form eines Ovals' beschrieben - diese Transferleistung kann nicht leicht trainiert werden.

Die folgende Vorgehensweise ist idealisiert und läuft in einem kontrollierten Umfeld ab. Es wird nicht zwanghaft versucht die Analogie zum menschlichen Gehirn aufrecht zu erhalten, da der Großteil der NN schon lange nicht mehr mit diesem Ansatz entwickelt wird, sondern dieses Konzept als erfolgreiche Grundlage zur Implementierung in verschiedenen Fragestellungen genutzt wird. Um botanisch korrekt zu bleiben und genauere Ergebnisse zu erzielen, müsste man weitere Aspekte wie die Verzweigung der einzelnen Blätter entlang eines Blätterzweigs hinzuziehen, da allein aufgrund dieses Merkmals je nach Auswahl der Pflanzenarten die Auswahl

drastisch reduziert werden könnte. Diese Addition ergibt jedoch bei der Aufgabenstellung keinen Sinn, da dadurch der Versuchsaufbau insofern eingeschränkt wird, dass die Identifizierung nicht mehr nur anhand des einzelnen Blattes abläuft. Des Weiteren würde diese Ergänzung hier möglicherweise keinen Mehrwert beitragen, da die Auflösung (mit Auflösung bezeichnet man hier die Bildpunktzahl, oft bewertet über die Länge l und Breite b des Bildes in Pixel), also Detailgenauigkeit, der einzelnen Merkmale des Blattes unter der Skalierung stark abfallen würde. In ein Bild des vorgegebenen Pixelumfangs von $l * b$ Pixeln passen nur Details, die sich anhand von Kanten festmachen lassen. Die Kanten entstehen durch Änderungen der Farb-/Intensitätszusammensetzungen eines Pixels zum nächsten. Ist das Blatt im Zentrum dieses Bildes platziert und nimmt es einen Großteil des Pixelumfangs ein, so kann bei richtiger fotografischer Ausführung dieser Großteil des Bildes für blattspezifische Details genutzt werden. Genauer: wenn man das Blatt idealisiert als Rechteck-füllend sieht, stehen

$$(l - l_{\text{Hintergrund}}) * (b - b_{\text{Hintergrund}})$$

Pixel für die individuellen Details des Blattes zur Verfügung. Der Hintergrund ist in unserem Fall eine weiße Fläche. Man nehme jetzt drei an einem Zweig verbundenen Blätter. Die Abbildung derselben nimmt nun $(l - l_{\text{Hintergrund}}) * (b - b_{\text{Hintergrund}})$ Pixel ein, doch bleibt für die Details des einzelnen Blattes jetzt nur noch maximal ein Pixelumfang von

$$(l - l_{\text{Zweig}}) * (b - b_{\text{Zweig}})/3 - (l_{\text{Hintergrund}} * b_{\text{Hintergrund}})$$

Die Maße sind zunächst in drei Teile geteilt, da idealisiert jedem Blatt nur noch

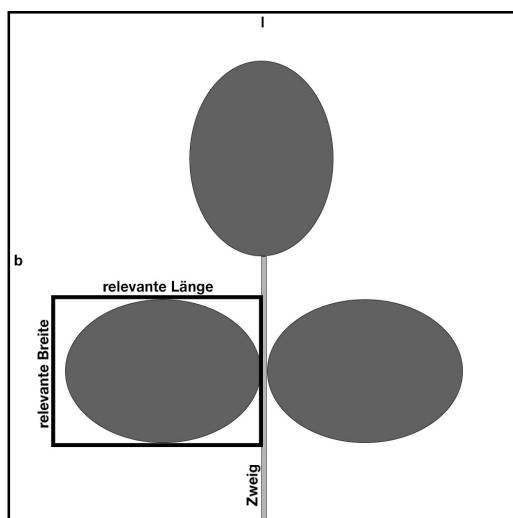


Abbildung 4.1: Zweig mit drei Blättern - eins (gerahmt) zu klassifizieren

1/3 des Bildplatzes zukommt. l_{Zweig} und b_{Zweig} sind hier die Maße des Zweigs, die Bildpunkte einnehmen. Natürlich werden hier eigentlich Teilflächen des Bildes betrachtet, die nur idealisiert mathematisch als verformte Rechtecke des Gesamtbildes betrachtet werden. Die Skalierung des einzelnen Blattes hat abgenommen, wodurch auch die Detailgenauigkeit desselben abgenommen hat. Es folgt die Annahme, dass mit einer höheren Detailgenauigkeit des Einzelblattes eine genauere Kategorisierung der Blätter möglich ist, da jede Art Blatt sich von allen anderen in Details wie der

Struktur des Blattes abhebt. Ergänzend lässt sich zu dem botanischen Einwand sagen, dass es hier um die Gewinnung informatischer Ergebnisse geht und nicht um die botanisch-korrekte Analyse eines Blattes. Zu dem kontrollierten Umfeld wie anfangs erwähnt lässt sich ergänzen, dass dieser Versuch so durchgeführt wird, dass der informatischen Umsetzung möglichst wenige Hindernisse aufgrund von fotografischen Schwierigkeiten im Weg stehen (siehe 5.2).

Kapitel 5

Der Datensatz

5.1 Digitale Fotos

Die Fotografie, so steht es in 'Meyers Universal Lexikon' aus dem Jahr 1982, ist eine 'Sammelbezeichnung für alle Verfahren, ein durch Strahlung, durch Licht des sichtbaren Spektralbereichs erzeugtes reelles Bild auf lichtempfindlichen Schichten mittels Kamera festzuhalten', wobei sich diese durch eine 'unübertroffene Wiedergabegenaugkeit und unübertroffenem Informationsgehalt' auszeichne (mey, 1982). Dieses Merkmal ist zentral in der Auswahl dieser Verfahren für die Problembewältigung. Die Fotografie ermöglicht es, Situationen, Gegebenheiten und Details festzuhalten und dabei die zeitliche Dimension der Erde zu umgehen, beziehungsweise mit ihr zu arbeiten, sodass es möglich ist Ausschnitte zeitlos für die Zukunft aufzzeichnen. Mit dem Vorläufer der Fotografie, der 'Camera Obscura', die bereits im

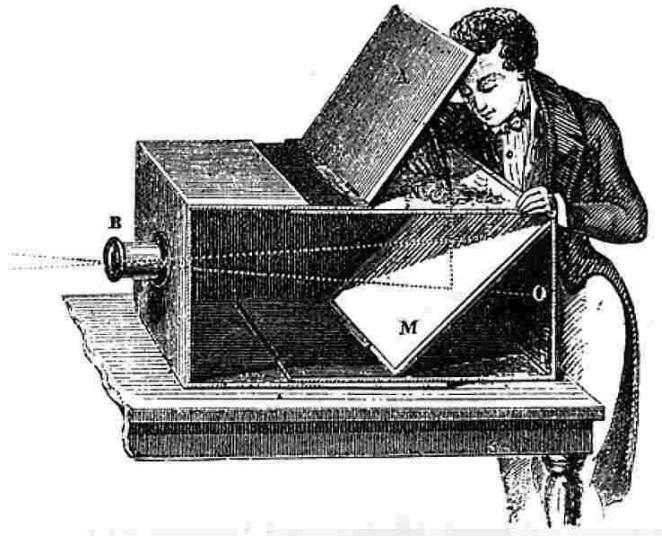


Abbildung 5.1: Künstler mit Camera Obscura, circa 1850. Quelle: 19th Century Dictionary Illustration, Unbekannter Illustrator

Mittelalter Verwendung in den Wissenschaften fand, war diese zeitlose Festigung eines Blickes noch nicht möglich. Erst in der Mitte des 19. Jahrhunderts gab es erste Durchbrüche in fotografischen Verfahren, die auf verschiedenen chemischen Prozessen basierten und mittels Linsen einen Ausschnitt zweidimensional festhalten konn-

ten. Nach den grundlegenden Entwicklungen kamen dann im 20. Jahrhundert unter anderem durch unternehmerische Pioniere in dem Bereich wie Ernst Leitz und seinen Nachfahren, die zusammen die Marke Leitz aufbauten, aus der später die Leica Gruppe mit maßgeblichen Produktionsreihen für den Weltmarkt entsprang, schnelle Fortschritte. Fast das ganze Jahrhundert hat es jedoch gedauert, bis der Prozess begonnen hat, dass die Film-betriebenen Kleinbildkameras, die auf ein physikalisches Replikationsmedium angewiesen sind, von Digitalkameras abgelöst wurden. Die erste kommerzielle Digitalkamera von 'Fairchild Imaging' aus dem Jahr 1973 nutzte die 'Charge-coupled Device' (CCD)-Technologie und hatte eine Auflösung von 100×100 Pixeln, was im Vergleich zu heutigen Kameras wie 'Digital Single-Lens Reflex' (DSLR), die Anfang der 2000er den Markt übernommen haben, eine enorm kleine Auflösung ist (die 2017 vorgestellte kommerzielle Kamera 'a7R III' von dem Hersteller Sony beispielsweise bietet eine maximale Auflösung von 7952×5304 Pixel).

Digitale Fotografie bietet sich für diesen Einsatz gut an, denn die heute verfügbare fotografische Technologie ist nicht nur preisgünstig und leicht zugänglich, sondern braucht kein permanentes physisches Speichermedium, da die Fotos auf digitalen Speichern wie Solid-State-Drives oder normalen Festplatten gespeichert werden, und ist deshalb effizient zu verarbeiten. Die digitalen Fotos sind aufgebaut in einer Anordnung sogenannter Pixel. Ein Pixel ist an sich nur ein Platzhalter für einen numerischen Wert, beispielsweise einen reellen Wert zwischen 0 und 1. Dieser Wert ist dann die Aktivierung des Pixels, wobei 1 beispielsweise weiß und 0 schwarz sein kann - die dazwischenliegenden Zahlen wären entsprechende Abstufungen in Graustufen. Das gleiche kann auch anders skaliert werden; so ist die Festlegung einer Aktivierung

$$x \in \mathbb{N} \mid 0 \leq x \leq 255$$

heute Standard¹. Die Anordnung geschieht nun als $l \times b$ -Matrix mit l -Pixeln in der Länge und b -Pixeln in der Breite. Ein Foto kann also als diskret-definierte Funktion für Pixelaktivierungen mit einem Wertebereich W , definiert wie die Aktivierung x , beschrieben werden.

Die obige Erklärung gilt für ein Bild in Graustufen ('schwarz-weiß' Foto). Ein Farbfoto hat jedoch eine sehr ähnliche Struktur: Das Farbfoto setzt sich ebenfalls aus diskreten Pixelaktivierungen zusammen, jedoch werden je drei Aktivierungen an gleicher Stelle in der Darstellung überlagert. Das liegt an der Natur eines Farbbildes in der standardmäßigen Verarbeitung im RGB-Farbraum: Die Aufnahme ist unterteilt in die Grundfarben rot, grün und blau, existiert also theoretisch drei mal unabhängig von einander. Einmal nur mit den Pixelaktivierungen für den roten Kanal, einmal mit den Pixelaktivierungen für den grünen Kanal und analog dazu für den blauen Kanal. Überlagert nach dem additiven Verfahren ergeben diese drei Kanäle eine farblich-realistische Darstellung der Aufnahme. Es existieren auch andere Farbräume; so nutzt ein Tintenstrahldrucker Gelb, Magenta und Cyan und kann mit diesem Farbtripel durch ein subtraktives Verfahren den gesamten Farbraum abdecken.

¹Dieser Wertebereich ermöglicht eine Ablage der einzelnen Aktivierungen in einem 8-Bit ($\cong 1$ Byte) Speicher.

Hier ist die Darstellung von drei Versionen eines Blattes des folgenden Datensatzes in Graustufen:



Abbildung 5.2: 256*256 Pixel

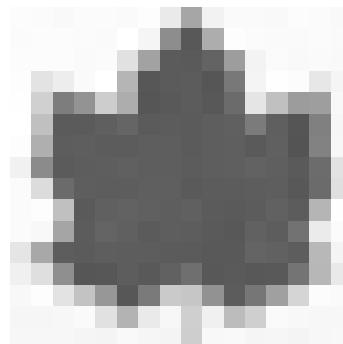


Abbildung 5.3: 16*16 Pixel

0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.6	0.9	0.9	0.9	0.9	0.9	0.9	0.9
0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.5	0.3	0.6	0.9	0.9	0.9	0.9	0.9	0.9
0.9	0.9	0.9	0.9	0.9	0.9	0.8	0.6	0.3	0.3	0.3	0.6	0.9	0.9	0.9	0.9
0.9	0.8	0.9	0.9	1.0	0.7	0.3	0.3	0.3	0.3	0.4	0.9	0.9	0.9	0.9	0.9
1.0	0.8	0.4	0.5	0.8	0.6	0.3	0.3	0.3	0.3	0.4	0.8	0.7	0.6	0.6	0.9
0.9	0.7	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.4	0.3	0.3	0.5	0.5	0.9
0.9	0.6	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.5	0.5	0.9
0.9	0.5	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.4	0.4	0.8
0.9	0.8	0.4	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.4	0.4	0.8
0.9	0.9	0.7	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.7	0.7	0.9
0.9	0.9	0.6	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.4	0.9	0.9
0.8	0.6	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.7	0.9
0.9	0.8	0.5	0.3	0.3	0.3	0.3	0.3	0.4	0.3	0.3	0.3	0.4	0.7	0.9	0.9
0.9	0.9	0.8	0.7	0.5	0.3	0.4	0.7	0.7	0.5	0.3	0.4	0.6	0.8	0.9	0.9
0.9	0.9	0.9	0.9	0.8	0.7	0.9	0.9	0.8	0.9	0.7	0.8	0.9	0.9	0.9	0.9
0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.8	0.9	0.9	0.9	0.9	0.9	0.9	0.9

Abbildung 5.4: 16*16 Pixel in Matrix-Form

5.2 Kreation des Datensatzes

Für die Lösung der Klassifikation musste ein erheblicher Datensatz bestehend aus digitalen Fotos von Blättern erstellt werden.

Der erste Ansatz den ich bei der Datenerhebung verfolgte, war das Fotografieren im Freien.

Als zugleich aufgrund der Aufnahmequalität und aufgrund der Zugänglichkeit geeignete Kamera stand eine DSLR der Marke Canon (Modell 600d) fest. Anschließend wurden verschiedene passende Objektive auf wesentliche Verhaltenseigenschaften ausgiebig getestet. Zu diesen Ergebnissen bin ich gekommen:

1. Tamron 70-300mm: Das Tele-Zoomobjektiv punktet in einer guten Vergrößerung und ausgezeichneter Schärfe, ist jedoch trotz 'Makro'-Modus aufgrund der Minimalentfernung von 95cm zum Objekt ungeeignet
2. Canon 50mm: Das Normalobjektiv hat eine konstante Brennweite, weist wenig Verzerrung auf und ist im Vergleich sehr lichtstark, jedoch etwas unscharf und aufgrund des Crop-Faktors der Kamera² zu nah am Objekt.

²Die 600d hat keinen Vollformatsensor, auf den sich der Abbildungsbereich bezieht, sondern einen kleineren, der das Foto entsprechend in der Abbildung um circa den Faktor 1,6 vergrößert.

3. Canon 28-105mm: Das Standard-Zoomobjektiv hat eine gute Abbildungsqualität und eine angenehme Schärfe, dafür aber eine deutlich erkennbare Abnahme der Schärfe an den Kanten und einen Mindestabstand von 70cm zu dem Objekt.
4. Canon 18-55mm: Das Standard-Zoomobjektiv hat eine sehr akzeptable Abbildungsqualität und Schärfe, die gerade bei 35mm gut geeignet ist für den Einsatzzweck, und bietet dazu den vergleichsweise sehr geringen Mindestabstand von 11cm.

Die Millimeter-Angabe bezieht sich stets auf die Brennweite bezogen auf 35mm-Filmkameras³ und ist eine verbreitete und standardisierte Angabe in der Fotografie. Ich habe mich also für das zuletzt aufgeführte Zoomobjektiv entschieden und es auf 35mm eingestellt. An die Ausleuchtung der Blätter war eine besondere Anforderung gestellt, da diese möglichst gleichmäßig sein muss, um als standardisiert betrachtet werden zu können. Das spätsommerliche Licht der Sonne ermöglicht bereits eine eher diffuse Ausleuchtung, die ich mit einer Aufsteck-LED der Marke Aputure am Blitzschuh der Kamera mit $18 * 11 = 198$ Einzel-LEDs gepaart habe. Vor die LED-Leuchte habe ich einen weiteren Diffusor gespannt, sodass das Licht möglichst weich strahlt und keine harten Konturen schafft oder kantige Schatten wirft. Der Weißwert ist bei dieser Lichtquelle justierbar, was eine genaue Synchronisation mit dem Sonnenlicht und der Kamera ermöglicht.

So habe ich mit dieser auf einem Dreibeinstativ montierten Ausrüstung auf umliegenden Grünflächen in Frankfurt am Main und Umgebung (unter anderem Stadtwald, Holzhausenpark und Grüneburgpark) Expeditionen durchgeführt, die man sich so vorstellen kann:

1. Aussuchen eines geeigneten Baums
2. Observieren der Blätter und Entscheidung für bestimmte Äste fällen
3. Kamera montieren und geeignete Belichtungseinstellungen wählen
4. Ein auf ein Klemmbrett-montiertes rotes, später weißes Blatt Papier im Hintergrund des einzelnen Blattes positionieren
5. Die Kamera auslösen und das Foto aufnehmen
6. Ein neues Blatt wählen und alle Schritte ab Schritt 3 wiederholen, bis mindestens 30 einzelne Blätter fotografiert sind

Dieser Prozess wurde wiederholt für alle einzelnen Baumarten. Die Überlegung mit dem roten Blatt Papier als den kompletten Hintergrund abdeckende Kulisse basiert auf der Idee und Technik der farbbasierten Bildfreistellung (Englisch: 'chroma keying'), die vor allem bei Film- und Fernsehproduktionen häufig eingesetzt wird. Hierbei wird eine Farbe, die möglichst komplementär zu der des zu fotografierenden Objektes ist, als Farbfläche im Hintergrund des Objektes gewählt in der Absicht, diese in späterer Bildbearbeitung technisch zu ersetzen durch einen neutralen, entweder weißen oder transparenten Hintergrund.

³Hier wäre der Crop-Faktor 1, die Brennweite also unverfälscht.

Schon relativ am Ende der Datenerfassung stellte sich dieser Prozess jedoch als zu kompliziert in der Nachbearbeitung heraus. Während des Archivierens und Bearbeitens wurde schnell klar, dass diese Lösung

1. zu ineffizient ist, da die Selektion der Blätter und das manuelle digitale Herausschneiden derselben bei den nötigen Datenmengen illusorisch ist und
2. die Fotos selber trotz der beschriebenen Maßnahmen ungleichmäßig wurden, beispielsweise bezüglich der doch sehr unterschiedlichen Lichteinstrahlung zwischen Wald und Park.

Es musste also ein neuer Ansatz gefunden werden. Dieser Ansatz wurde der zunächst aus ökologischen Gründen abgelehnte des Scannens mit einem handelsüblichen Scanner. Hierzu mussten die Blätter von ihren Ästen gelöst werden, doch wurde dies aufgrund des sowieso annähernden Blattverlustes durch den Herbst eingegangen und die Blätter wurden möglichst pflanzenschonend abgetrennt – direkt bei der Abzweigung und unter Benutzung einer Vielzahl von verschiedenen Bäumen der gleichen Art, also einer Art Diversifizierung des Schadens.

Die Blätter wurden dann in mehreren Ladungen eingescannt.

Im Anschluss mussten mehrere zeitaufwendige manuelle Anpassungen gemacht wer-



Abbildung 5.5: Blättersammlung (Auszug)

den, angefangen mit Erstellen von $\sum_{j=1}^{\hat{n}} x_j \approx 365$ digitalen Kopien (mit x Anzahl der Blätter auf Scan j) gefolgt von dem Zuschneiden der Kopien, sodass das Längen-/Breitenverhältnis 1 : 1 beträgt, im Zentrum jeweils nur ein Blatt vorhanden ist und dieses Blatt einen zu den Seitenkanten des Bildes möglichst parallel verlaufenden Querschnitt aufweist. Dieser Schritt benötigt keine weitere Ausführung, da er auf dem analogen Weg zum früheren physischen Ausschneiden nun mit der Foto-bearbeitungssoftware Adobe Lightroom per Hand durchgeführt wurde. Diese Ausschnitte wiesen jedoch Überlappungen der Blätter auf, sodass die hier ungewollten, abgeschnittenen Blattinhalte entfernt werden mussten. Außerdem lag zu diesem Zeitpunkt aufgrund des ungleichmäßigen Zuschneidens eine ungleiche Auflösung bei den Blattaufnahmen vor. Nun wurden die Ausschnitte in die Bearbeitungssoftware Adobe Photoshop geladen, die vorrangig zur Bildmanipulation genutzt wird, und sowohl von den ungewollten Inhalten bereinigt als auch von der Größe her standardisiert. Dieser Schritt musste ebenfalls für jedes Blatt individuell durchgeführt werden und konnte aufgrund der individuellen Bildeigenschaften und Dateieigenschaften nicht



Abbildung 5.6: Einscannen



Abbildung 5.7: Zugeschnitten



Abbildung 5.8: Fertig bearbeitet

automatisiert werden.

Es entstand eine Sammlung von standardisierten Blattabbildungen von guter Qualität. Würde man dieses Projekt skalieren, müsste man eine andere, automatisierte Methode für diese Erfassung nutzen. Denkbar ist hier ein Abtrennen der Blätter und ein Fotografieren unter Studio-Bedingungen mit konstantem Hintergrund, begradigten Blättern und perfekt-kalibrierter Kamera.

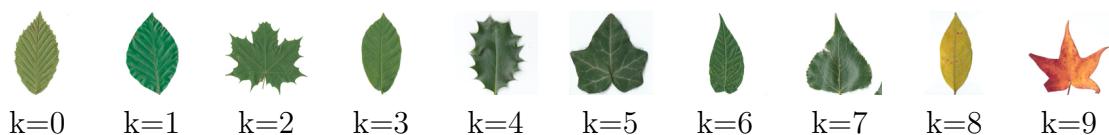


Abbildung 5.9: Ein Bild aus der jeweiligen Kategorie

Kapitel 6

Das Netz

6.1 Die Theorie

Wie beschrieben wird die Aufgabe der Klassifizierung mit einem vorwärtsgekoppelten (feedforward) NN bewältigt. Allgemein gebe es die Eingabe-Matrix (Input) eines digitalen Fotos I und den Ausgabe-Vektor (Output) O , wobei O eine eindeutige Klassifizierung der Eingabe in die gegebenen Klassen, aus mathematischer Sicht Dimensionen, zulassen soll¹.

Gesucht werden nun geeignete Zwischenschritte, die von $I \rightarrow O$ führen.

$$\begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 2 \\ 1 & 1 & 0 \end{bmatrix} \dots \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Abbildung 6.1: Stark vereinfacht wird eine Eingabematrix I durch Operationen '...' umgeformt zu einem Ausgabevektor O

Sei E alle Eingaben e , hier Bilder, der Form I mit der Größe $m * m$ und K alle Klassen k . Eine intuitive Herangehensweise ist es, in I Muster zu finden, die alle Eingaben $e_k \in E$ der Klasse $k \in K$ ebenfalls aufweisen. Diese Muster können erkannt werden durch die Faltungsmatrizen $f \in F$ der Maße

$$n * n \mid n = 2d - 1 ; d \in \mathbb{N} \setminus \{0\}$$

die auf I angewandt werden können.

F werden im Folgenden Filter genannt, wobei anzumerken ist, dass Filter in der Literatur auch Kern, Filteroperator, Filtermaske oder Faltungskern genannt werden. Bei der besagten Anwendung auf I handelt es sich um eine diskrete Faltung.

Eine Faltung ist in der mathematischen Funktionsanalysis allgemein ein Produkt zweier Funktionen; sie liefert also bei zwei Funktionen r und f eine dritte Funktion $r * f$. Da r hier die Eingabematrix I und f hier die Faltungsmatrix f , die Funktionen also Matrizeelemente als Werte liefern, haben sie einen diskreten Definitionsbereich. Eine diskrete Faltung wird deshalb nicht wie eine Faltung zweier

¹Wie folgend gezeigt, entspricht die Koordinate, dessen Position im Vektorraum von O darge-

reell-definierten Funktionen durch ein Integral kalkuliert, sondern durch eine Summenfunktion.

I^* ist die gefaltete Funktion; die Werte x und y geben die Position des gesuchten Elements in der Ergebnis-Matrix an und a gibt die Zeile und Spalte (Zeilenzahl=Spaltenzahl, da quadratisch) des mittleren Elements des Filters $a = (n + 1)/2$ an.

Die gefaltete Funktion für ein Foto mit einem Kanal (Grautöne) ist gegeben und sieht aus wie folgt:

$$I^*(x, y) = \sum_{i=1}^{\hat{n}} \sum_{j=1}^{\hat{m}} I(x - i + a, y - j + a) * f(i, j) \mid x, y, a \in \mathbb{N}^2$$

Je höher der berechnete Wert, desto höher die Korrelation des unterliegenden Bildausschnitts mit dem Filter.

Die Faltungs-Operation arbeitet also keineswegs wie eine klassische Matrizenmultiplikation, sondern ist eine Konvolution der Input-Matrix und des Filters. Aufgrund der Anwendung dieser Operation wird der in diesem Projekt verwendete Netzwerk-Typ im Englischen 'Convolutional Neural Network' - kurz CNN - genannt.

Aus praktischen Gründen und analog zu dem in Kapitel 3 vorgestellten Perzeptron wird die Faltung in der Implementierung ergänzt um die Tendenz b (englisch: 'bias'), einen numerischen Wert ($b \in \mathbb{N}$), der zu dem Ergebniswert der Faltung addiert wird, um eine Aktivierungsschwelle herzustellen und kategorisch bestimmte Filter stärker zu gewichten als andere. Als Ausgabe liefert diese Operation einen so-genannten Convolutional Layer (CL), der sich nach dem Neuronen-Modell aus den durch $I^*(x, y)$ zu berechnenden Neuronenaktivierungen zusammensetzt. Eine solche Berechnung ist hier beispielhaft und auszugsweise berechnet für $I^*(1, 1)$ auf Basis von zwei willkürlich gewählten Matrizen:

$$I = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}, f = \begin{bmatrix} 17 & 18 & 19 \\ 20 & 21 & 22 \\ 23 & 24 & 25 \end{bmatrix}, b = 0$$

$$I^*(x, y) = \left(\sum_{i=1}^3 \sum_{j=1}^3 I(x - i + 2, y - j + 2) * f(i, j) \right) + b ; 0 \leq x, y \leq 5$$

stellt wird, den probabilistischen Werten der Klassenzugehörigkeit des Bildes I .

²Da $I_{min} = I(1, 1)$, muss $x, y \geq 2 - a$ und da $I_{max} = I(m, m)$ muss $x, y \leq m + 0, 5n - 0, 5$.

$$\Rightarrow I^* = \begin{bmatrix} I^*(0, 0) & I^*(0, 1) & I^*(0, 2) & I^*(0, 3) & I^*(0, 4) & I^*(0, 5) \\ I^*(1, 0) & I^*(1, 1) & I^*(1, 2) & I^*(1, 3) & I^*(1, 4) & I^*(1, 5) \\ I^*(2, 0) & I^*(2, 1) & I^*(2, 2) & I^*(2, 3) & I^*(2, 4) & I^*(2, 5) \\ I^*(3, 0) & I^*(3, 1) & I^*(3, 2) & I^*(3, 3) & I^*(3, 4) & I^*(3, 5) \\ I^*(4, 0) & I^*(4, 1) & I^*(4, 2) & I^*(4, 3) & I^*(4, 4) & I^*(4, 5) \\ I^*(5, 0) & I^*(5, 1) & I^*(5, 2) & I^*(5, 3) & I^*(5, 4) & I^*(5, 5) \end{bmatrix}$$

Es ist festzustellen, dass bei der Operation die Filtermatrix in an beiden Achsen gespiegelter Form mit dem unterliegenden Ausschnitt gleicher Größe der Input-Matrix komponentenweise, also als Hadamard-Produkt, multipliziert wird. Außerdem ist zu sehen, dass sich Länge und Breite von I^* im Vergleich zu I um zwei vergrößert haben. Dies liegt an der Natur des Definitionsbereiches ($0 \leq x, y \leq 5$), der theoretisch nach Belieben und je nach den Zieldimensionen der gefalteten Funktion gewählt werden kann. Hier ist ein Bereich gewählt, der die Ränder von I mit einschließt und dementsprechend auch nicht vorhandene Werte (hier gewählt 0) beinhaltet. Es wird also definiert, dass Werte außerhalb des echten Definitionsbereiches von I als 0 definiert sind.³

Nun soll aber das komplette Potential der Fotos des Datensatzes genutzt werden. Hierzu zählt die drei vorhandenen Farbkanäle rot, grün und blau und nicht nur die Luminanz (Pixelaktivierung) im Schwarz-Weiß-Bereich zu verarbeiten. Dies ermöglicht eine komplexere Filteranwendung, die auch auf bestimmte Farbmuster und Farbanordnungen oder Farbtöne anspringen kann und somit zu einer besseren, schnelleren und möglicherweise effizienteren Erkennung beitragen kann.

Nun hat I nicht mehr die Dimensionen $m * m$, sondern auch die Tiefe 3, folglich: $m * m * 3$.

Es liegen also drei Input-Matrizen vor. Entsprechend erweitern sich auch die Dimensionen des Filters f zu $n * n * 3$.

Die gefaltete Funktion könnte nun entweder so angepasst werden, dass auch drei Ergebnis-Matrizen kalkuliert werden, oder so, dass eine Ergebnis-Matrix der Tiefe 1 kalkuliert wird. Es wird der zweite Ansatz gewählt, da nur so die erwähnten Farbanordnungen berücksichtigt werden.

³Dies könnte auch so gewählt werden, wie es in der Praxis oft der Fall ist, dass die Werte außerhalb des eigentlichen Definitionsbereiches die Randwerte wiederholen (beispielsweise $I(0,0) = I(1,1)$). Die Methoden für die Definition der Randwerte werden auch 'padding' genannt.

Folglich wird die Formel der Funktion $I^*(x, y)$ so angepasst, dass eine Tiefe 1 entsteht:

$$I^*(x, y) = \left(\sum_{t=1}^3 \sum_{i=1}^3 \sum_{j=1}^3 I_t(x - i + 2, y - j + 2) * f(i, j) \right) + b \quad (6.1)$$

Die Tiefe $t = 3$ gilt nur für die direkt auf die Eingabeschicht folgende erste CL-Schicht. In folgenden Schichten ist $t = \#I^*$, da die Tiefe der Eingangsschicht aufgrund der sequenziellen Struktur durch die Anzahl der Ergebnismatrizen der vorherigen Schicht bestimmt wird. Gleichung 6.1 ist die in diesem Fall zunächst vollständige Faltungsoperation in den folgenden CNN.

Der iterative Ablauf der Berechnung für alle Ergebniswerte der möglichen Faltungsoperationen scheint sehr abstrakt und streng mathematisch, schließt jedoch eine intuitive Vorstellung des Prozesses nicht aus. Eine bildliche Vorstellung könnte so aussehen: Man möchte auf einem Foto von einem Apfelbaum die Positionen der Äpfel erkennen. Eine transparente Schablone, die kleiner ist als das Foto, mit einem Kreis in der Mitte wird in kleinen, sich überschneidenden Bildausschnitten über eben dieses Abbild eines Apfelbaums gezogen. In jedem Schritt wird die Korrelation des Kreises mit dem unterliegenden Bildausschnitts ermittelt - sollte dieser nun einen mittig-platzierten Apfel zeigen, so wird dieser Korrelationswert zwischen dem Kreis und dem Apfel entsprechend groß sein. Am Ende ergibt sich aus den Korrelationswerten eine Matrix, die genau in den Positionen der Äpfel hohe Werte hält, in allen anderen, für die Suche irrelevanten, Orten jedoch geringe Werte, die nur durch Rauschen zustandekommen. In der erweiterten Variante ergänzt man diesen Prozess insoweit, dass die Korrelation zwischen der Schablone und dem Foto nur berechnet wird, wenn das abgebildete Objekt unter der Schablone rot ist. So wird sogar das Rauschen minimiert und die Positionserfassung wird genauer.

6.1.1 Architektur-Theorie

Die am Anfang der Theorie erwähnten Zwischenschritte, zwischen *Input* und *Output*, müssen sinnvoll strukturiert werden. Es reicht nicht, eine Konvolutionsoperation zwischen den beiden Ebenen zu haben; allein von den Dimensionsveränderungen würde dies nicht passen. Außerdem ist es absolut unrealistisch eine sinnvolle Segmentierung der Bildbereiche und daraufhin eine eindeutige Klassifikation mit einem einzigen Filter zu erreichen. Ziel ist es nun also in der weiteren Entwicklung, eine 'Architektur' zu entwerfen - sorgfältig Schichten wie ein Architekt hintereinander zu positionieren und eine Struktur für einen Graphen-artigen Fluss von Informationen zu generieren, sodass genügend Schritte ablaufen, um eine genaue Klassifikation zu ermöglichen, und möglichst wenige, sodass die Klassifikation möglichst ressourcen- und recheneffizient abläuft. So viele Schritte wie nötig, so wenig Schritte wie möglich.

Für die Kontinuität mit der Implementierung des Projekts muss zunächst dem CL eine weitere Funktion hinzugefügt werden: eine Aktivierungsfunktion. Je nach Einsatzgebiet des NN gibt es hier verschiedene etablierte Möglichkeiten. In Regressionsanwendungen wird häufig die erwähnte Sigmoidfunktion verwendet, die eine Normalisierung und Angleichung der Werte vollzieht. So kann es sinnvoll sein, Neuronenaktivierungen zu normalisieren, sodass es keine starken Ausreißer unter

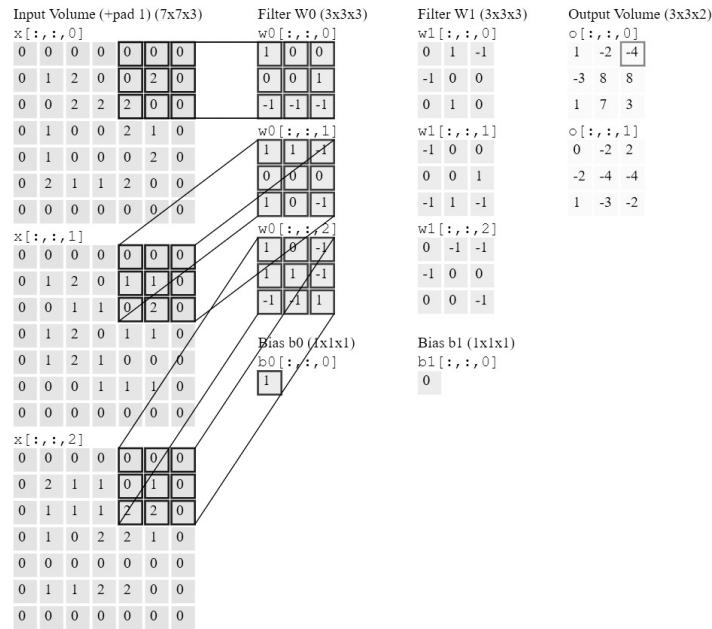


Abbildung 6.2: Anschauliche Berechnung der diskreten Faltung: Bildpunkte (Input-Matrizen) werden stets kollektiv und nicht individuell verarbeitet. **Quelle:** Karpathy (2018)

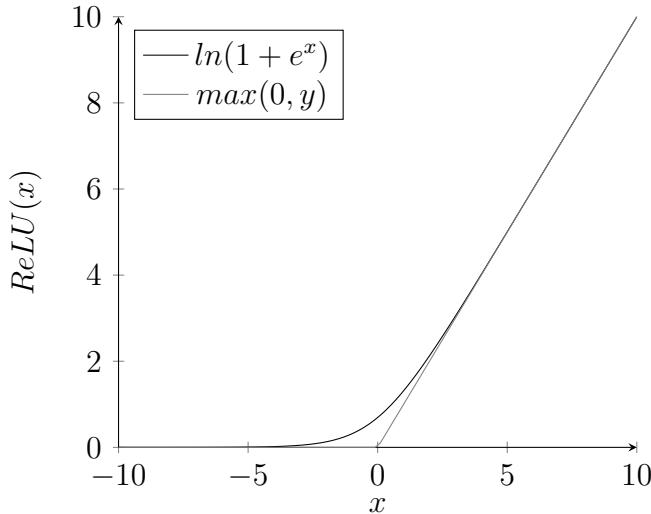
den Filtern gibt. Bei Klassifikation mit CNN ist diese Normalisierung jedoch nicht nötig, da die Werte von Anfang auf den standardisierten Wertebereich der einzelnen Pixelaktivierungen zwischen 0 und 255 komprimiert sind. Es hat sich deshalb eine andere bewährt, die eher einen logischen praktischen Verwendungsgrund hat. Es kann nämlich bei der Faltungsoperation nach obiger Ausführung vorkommen, dass eine Aktivierung $x < 0$ berechnet wird. Im Sachzusammenhang hieße dies jedoch, dass eine negative Korrelation des Bildausschnitts mit dem Filter vorliegen würde. Obwohl dies zwar als hemmende Wirkung vorkommen könnte, soll dies ausgeschlossen werden, da maximal keine Korrelation erkannt werden soll. Diese Aufgabe übernehmen die 'Rectified Linear Units' - kurz ReLU - , eine Art Gleichrichter. Die ReLU-Funktion ist definiert als

$$\text{ReLU}(x) = \max(0, x)$$

Werte $x > 0$ werden also identisch übernommen, Werte $x < 0$ werden ersetzt durch 0. Während dies programmiertechnisch nicht schwer umzusetzen ist ⁴, ist eine mathematisch-korrekte differenzierbare Formulierung, wie sie wie später erläutert für das Gradientenabstiegverfahren benötigt wird, nicht so einfach. Deshalb wird eine logarithmische Approximation $\text{ReLU}(x) = \ln(1 + e^x)$ verwendet.

⁴beispielsweise mit einer einfachen Entscheidungsregel eingebettet in eine Python-Funktion

```
def ReLU(x):
    if (x<1):
        x=0
    return (x)
```



Wie auf dem Graphen zu sehen ist, sind erhebliche Abweichung des Rückgabewertes der Funktion für $-2 \leq x \leq 2$ vorhanden, doch kann man diese Bedenken entkräften, da sich diese stets auf Nachkommastellen beschränken, die bei der Verwendung von Integerzahlen, also ganzzahligen Werten mit abgeschnittenen Nachkommastellen, automatisch vernachlässigt werden. Analog zu der vorherigen Notation kann man die ReLu-Funktion schreiben als

$$\text{ReLU}(x, y) = \max(0, I^*(x, y)) \approx \ln(1 + e^{I^*(x, y)}).$$

ReLU, so Singh (2016), ist außerdem erheblich schneller zu berechnen während der Trainings-Phase als Sigmoid-ähnliche Funktionen (Vergleich Kapitel 3).

Nun gibt es nicht nur CL, die ein nach heutigen Maßstäben sinnvolles CNN bilden. Würde man nur CL hintereinander schichten, so würde die benötigte Speicher- und Verarbeitungskapazität schnell zunehmen und in den Bereich des Unberechenbaren abdriften. Deshalb gibt es eine weitere in der Literatur etablierte Schicht: den 'Max-Pooling Layer' (PL). Je nach Konfiguration können Datenreduktionen zu beispielsweise 1/4 stattfinden.

Der Sinn dahinter ist nicht nur eine Verkleinerung der Datenmenge und Effizienzsteigerung der Berechnung, sondern auch das Entfernen von Ballast, von unnötig gespeicherten Daten. Bei einer neuen beliebigen Matrix

$$I = \begin{bmatrix} 1 & 3 & 3 & 0 \\ 5 & 1 & 7 & 8 \\ 9 & 1 & 4 & 5 \\ 14 & 12 & 11 & 10 \end{bmatrix}$$

würde die Matrix nach einem PL mit $2 * 2$ Quadranten so aussehen (P für pooled):

$$I = \left[\begin{array}{cc|cc} 1 & 3 & 3 & 0 \\ 5 & 1 & 7 & 8 \\ \hline 9 & 1 & 4 & 5 \\ 14 & 12 & 11 & 10 \end{array} \right] \Rightarrow P(I) = \begin{bmatrix} 5 & 8 \\ 14 & 11 \end{bmatrix}$$

Die Veränderung ist ersichtlich: aus den $2 * 2$ Quadranten der Matrix I wurden nur die höchsten Werte (deshalb Max-Pooling) übernommen. Das ergibt Sinn, denn es

werden folgend nur die relevanten und höchsten Aktivierungen betrachtet! Es gibt hier keine Überlappung der Quadrate und keine trainierbaren Parameter oder Variablen in dieser Art von Schicht. Außerdem kann hiermit einer Positionsveränderung eines Elements von einem Foto zum nächsten entgegengewirkt werden. Es werden nur die relevanten Kanten aus einem bestimmten Bildbereich übernommen. Bei dem Beispiel mit dem Apfelbaum würden so die geringen durch Rauschen verursachten Aktivierungen vernachlässigt und nur die relevanten Aktivierungen der hohen Korrelationen mit den vorhandenen Äpfeln übernommen werden.

Wie festgestellt, muss noch eine Dimensionsreduzierung stattfinden, um aus den Ergebnis-Filtermatrizen folgend aus den Faltungsoperationen einen Vektor zu formen. An diesem Punkt sind zum Beispiel bereits mehrere CL-Schichten durchlaufen und es liegen nach mehrfachen PL r Filterergebnismatrizen der Größe $4 * 4$ vor. Die Dimensionen sind dementsprechend $4 * 4 * r$, wobei r analog zu den Farbkanälen die Tiefe ist. Zur Dimensionsreduzierung kann nun eine planierende Schicht in die Architektur inseriert werden (Englisch: 'Flatten Layer'), die alle Neuronenaktivierungen übernimmt. Es folgt ein Vektor mit $4 * 4 * r$ -Dimensionen. Dies wird dadurch erreicht, dass jedes Element der vorhandenen Ergebnismatrizen am Ende der Faltungsoperationen als einzelnes Neuron betrachtet wird und die Neuronenaktivierungen als Werte in einem einspaltigen Vektor übernommen werden.⁵ Gewonnen wird hierdurch ein Vektorformat. Sobald entweder die Filterdimensionen oder die Anzahl der Filter groß werden, werden die Vektordimensionen entsprechend sehr groß. Trainierbare Parameter gibt es hier keine, auch hier liegt eine reine Transformationsfunktion vor, welche die weitere Verarbeitung vereinfacht.

Nach Belieben kann auch dieser Vektor weiterverarbeitet werden; die Möglichkeiten sind zahlreich. Hier kann man auf Techniken klassischer NN zurückgreifen, die ja komplett vernetzt sind - alle Neuronen der vorherigen Schicht also direkt verknüpft sind mit allen Neuronen der ihr direkt folgenden Schicht. Die dichte Schicht (Englisch: 'Dense Layer') ist wieder mit ihren gewichteten Verknüpfungen der Neuronen trainierbar und hat entsprechende Parameter, wie bei den klassischen NN dargestellt.

Am Ende der CL muss mindestens eine dichte Schicht stehen, da sie die planierende Schicht mit, um bei dem Beispiel oben zu bleiben, $4 * 4 * r$ Dimensionen reduziert auf einen k -dimensionalen Vektor, wobei k die Kategorien sind. Am Ende muss jeder Kategorie ein Neuron zugewiesen sein, dessen Aktivierung die Wahrscheinlichkeit angibt, dass das Bild dieser Kategorie zugehörig ist. Bei dem erstellten Blätter-Datensatz ist das ein zehn-dimensionaler Vektor, da es zehn Kategorien von Blättern gibt. Dieser Vektor ist eine Wahrscheinlichkeitsverteilung. Auch hier wird wieder eine Aktivierungsfunktion benutzt. Da der Vektor eine Wahrscheinlichkeitsverteilung ist, sollten die Einzelwahrscheinlichkeiten des Vektors $\sum_{k=0}^9 P(k) = 100\%$. ReLU arbeitet unabhängig von den anderen Aktivierungen der Neuronen einer Schicht und wird dieser Anforderung dementsprechend von Natur aus nicht gerecht. Es bietet sich die Softmax-Funktion an, da sie eine Wahrscheinlichkeitsverteilung berechnet, die summiert stets 100% ergibt. Das interessante ist, dass Softmax nicht

⁵Wie folgend bei der Präsentation der eigenen Architekturen klar werden sollte, würde aus 256 Ergebnismatrizen der Maße $8 * 8$ Werte ein einspaltiger Vektor mit $8 * 8 * 256 = 16384$ Zeilen entstehen (Vergleich Flatten-Operation bei der Architektur a_0).

nur die Werte in den Bereich zwischen 0 und 1 normalisiert, sondern auch die Unterschiede in der Verteilung vergrößert. Die Funktion ist folgendermaßen definiert:

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

Es ergibt eine kategoriale Wahrscheinlichkeitsverteilung über K -möglichen Ausgängen (Finkernagel, 2018).

Eine besondere Herausforderung stellt die Generalisierung. Es muss sichergestellt werden, dass die Einordnung der aktivierten Filter nicht nur auf das Trainingsdatenset ausgerichtet ist, sondern auch abweichende Datensätze auf Basis der zuvor erstellten Filter korrekt klassifizieren kann. Auch hier gibt es wieder verschiedene Möglichkeiten. Srivastava et al. (2014) stellt Dropout vor - eine Methode, die nicht nur Generalisierung optimiert, sondern auch unter anderem die Performanz des NN deutlich verbessern kann. Die Idee ist, dass alle Neuronen eine einheitlich-definierte Wahrscheinlichkeit im Training zugeteilt bekommen, zu der ihre Aktivierung per Zufallsprinzip nicht in der weiteren Berechnung berücksichtigt wird, sondern ihre Aktivierung kategorisch von diesem Trainingsvorgang ausgeschlossen wird. Das betrifft nicht die Aktivierung im eigentlichen Einsatz zum Klassifizieren eines neuen Datensatzes, bei dem stets alle Neuronen aktiv sind und in der Verarbeitung berücksichtigt werden. Srivastava et al. (2014) verwendet Raten von bis zu 0,8 und kann eine direkte Fehlerreduktion bei der Klassifikation von 1,6% zu 1,35% nachweisen - eine Optimierung von 15,625%.

6.1.2 Optimierung

Die Aufgabe der Optimierung des Netzwerks ist die Berechnung geeigneter Parameter w und b zum möglichst korrekten Klassifizieren einzelner Bilder eines Bild-Datensatzes.

Das Lernen selber ist überwacht. Es gibt einen bereits klassifizierten Datensatz, mit dem die Parameter erlernt werden sollen. Um später die Genauigkeit des Netzwerks testen zu können, wird dieser Datensatz zunächst in zwei Subsätze ohne Überschneidung geteilt: Menge Train (Tr) und Menge Test (Te). Das Verhältnis kann von dem Entwickler gewählt werden - hier wurde in der späteren Implementierung ein Verhältnis $\frac{\#Tr}{\#Te} = \frac{4}{1}$ gewählt.

Zunächst werden alle Parameter zufällig initialisiert, um eine optimierbare Basis zu schaffen. Um die Parameter zu optimieren, muss ein Maß für den Fehler eingeführt werden, der minimiert werden soll. Für jedes Bild kann bereits mit den zufälligen Parametern eine Wahrscheinlichkeitsverteilung p mit Prognosen berechnet werden. Diese wird wahrscheinlich genauso zufällig sein wie die Parameter und natürlich keine sinnvollen Ergebnisse liefern. Jedem Bild kann aber eine genaue Wahrscheinlichkeitsverteilung zugeordnet werden, nämlich nur eine einzige Aktivierung von 1 in der korrekten Kategorie und 0 in allen anderen. Der Fehler kann nun durch einen Abgleich der korrekten Verteilung mit der berechneten Verteilung als Diskrepanz kalkuliert werden, zum Beispiel mit der mittleren quadratischen Abweichung (MSE von 'mean-squared-error'):

$$MSE = \frac{1}{2} \sum_k (p_k^{korrekt} - p_k^{berechnet})^2$$

MSE ist natürlich ein Funktionsterm abhängig von den Parametern der Architektur und den Eingabewerten des Netzwerks, da sich $p_k^{berechnet}$ ja die Ausgabe O des Netzwerks ist. Das Quadrat der absoluten Fehler hat einen weiteren Aspekt, der hier hilfreich sein kann. So sind große Fehler stärker gewichtet als geringe Fehler, was eine Setzung der Prioritäten in der Optimierung bewirkt. Die Fehlerfunktion, hier MSE , wird auch 'Loss-Funktion' genannt, wobei die deutsche Übersetzung von 'Loss' (Verlust) eher unpassend ist.

Anteilig der Gewichtung der vorherigen Neuronen, die mit dem fehlerhaften Neuron verbunden sind, wird dieser Fehler an die vorherige Schicht zurückgegeben. Das Neuron in der vorherigen Schicht bekommt also einen Teil des Fehlers eines darauf folgenden Neurons zugeteilt. Dieser anteilige Fehler wird jetzt wieder relativ zu den Gewichtungen an die Neuronen der vorherigen Schicht zurückgegeben. Dieser iterative Prozess wird bis zur ersten Schicht wiederholt und zwar nicht nur für jeweils ein Neuron, sondern parallel für alle Neuronenverbindungen des gesamten Netzwerks. Wird dieser Fehler bei allen Bildern von Tr bei allen Neuronen gemittelt, so kann von Beginn an eine Anpassung der Gewichte und Tendenzen stattfinden, der den jeweils daraus resultierenden Fehler in den darauffolgenden Schichten reduziert. Dieser Prozess basiert auf dem 'Backpropagation'-Algorithmus (Fehlerrückführung) nach LeCun et al. (1998b).

Da normalerweise ein sehr großer Datensatz Tr vorhanden ist, die Ermittlung der Anpassungen jedoch rechenintensiv ist, gibt es die häufig implementierte Methode des 'Stochastic Gradient Descent' (stochastischer Gradientenabstieg: SGD), bei dem nur ein Element, also ein Bild für die Aktualisierung genutzt wird und die Parameter in kleinen, einzelnen Schritten durch stochastische Erwartungswerte angepasst werden. Außerdem kann 'batch learning' verwendet werden, bei dem Tr weiter unterteilt wird in zufällig zusammengesetzte 'Batches', kleineren Subsätzen, die so iterativ verarbeitet werden können und deren Änderungen nach und nach durchgeführt werden und nicht direkt auf einmal. So verlaufen die Änderungen nicht perfekt, jedoch deutlich effizienter als mit dem ganzen Datensatz in der Berechnung ab. Nach einem Anpassungsprozess über den kompletten Datensatz Tr ist eine 'Epoche' vorbei (Chollet et al., 2015). Das Training kann sich je nach Anzahl der trainierbaren Parameter und Schrittgröße in der Anpassung über viele Epochen ziehen.

Gradientenabstiegsverfahren

Mathematisch ist dieser Prozess gradientenbasiert.⁶ Es folgt ein kleiner Abschnitt über Grundlagen der Parameteraktualisierung durch Gradientenberechnung auf Basis von Karpathy (2018), Kafunah (2018) und LeCun et al. (1998b).

Die trainierbaren Parameter⁷ befinden sich in einem Raum mit $\#w + \#b$ Dimensionen - unvorstellbar für Menschen, aber kein Problem für die Mathematik (außer der damit aufkommenden Rechenaufwand). Mit zufällig-initialisierten Parametern ent-

⁶Der Gradient einer Funktion ist generell die Richtung des steilsten Anstiegs. Bei dem Einsetzen des Parametersets kann, wie gleich zu sehen, die Richtung berechnet werden, in die die Parameter verändert werden sollten, um diesem Anstieg zu folgen. Entsprechend kann hiermit berechnet werden, wie die Parameter geändert werden müssen, um den Fehler MSE zu minimieren.

⁷Generell wird ein Parameter als konstanter Wert angesehen und eine Variable als veränderbarer Wert einer beliebigen Funktion. Hier werden Parameter jedoch auch in ihrem Wesen als variabel

steht ein Punkt in diesem Raum. Durch die Änderung der Parameter ist es möglich, sich auf dieser Hyperebene zu bewegen. Die Kostenfunktion MSE bildet eine Hyperebene - auf dieser nicht-linearen Hyperebene ein globales Minimum zu finden würde bedeuten, den optimalen Punkt im Raum bezogen auf den kleinsten Fehler gefunden zu haben. Die hier geforderte Rechenleistung ist in höheren Dimensionen, wie später bei der Betrachtung von Architekturen klar wird, für normale Computer zu groß, da die Kostenfunktion theoretisch zunächst für jedes Bild von Tr für jeden Punkt im Raum berechnet werden müsste. Deshalb wird mit dem Gradientenabstiegsverfahren ein möglichst gutes Minimum gesucht. Mit Hilfe der multivariablen Infinitesimalrechnung kann der Funktionsterm von MSE partiell abgeleitet werden in Bezug auf die Parameter, die den Fehler ausmachen. Durch minimale Änderungen der Parameter wird das darauffolgende relative Verhalten der Kostenfunktion betrachtet. Die dadurch resultierende berechnete Steigung der Kostenfunktion in eine bestimmte Richtung (Tangente) bei Einsetzen des Punkts kann nun als Grundlage für die Aktualisierung der Parameter-Position genutzt werden, die entweder entlang oder entgegen der Tangentenrichtung aktualisiert wird.

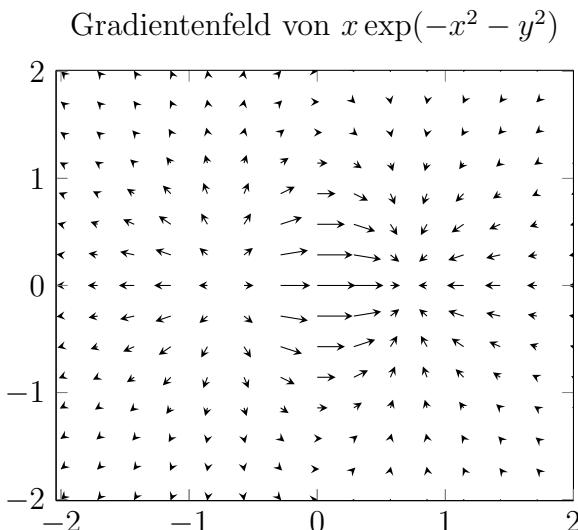


Abbildung 6.3: Ein Beispiel. Die Pfeile zeigen die Tangentenrichtung an, wobei die Pfeile rechts in die Richtung des Maximum-Extremums zeigen und die Pfeile links weg von dem Minimum zeigen. Je länger der Pfeil, desto größer der An- beziehungsweise Abstieg. Da die Funktion MSE aber minimiert werden soll, würde die inverse Richtung der Pfeile in der Aktualisierung der Gewichte verfolgt werden.

Der Gradient ∇MSE ist ein Vektor aller $\#w + \#b$ möglichen partiellen Ableitungen (in der Leibniz-Notation) von MSE , zum Beispiel

$$\nabla MSE = \begin{pmatrix} \frac{\partial MSE}{\partial w_1} \\ \vdots \\ \frac{\partial MSE}{\partial w_j} \end{pmatrix}$$

verstanden, da es das Ziel des Trainings ist, die Parameter so zu variiieren, dass sie die tatsächlichen Eingaben (\equiv Variable Werte des finalen Netzwerks mit konstanten Parametern) erfolgreich und in dem gewünschten Weg verrechnen können.

Die Gewichte w können so nach der Delta-Regel $\Delta w = w_{neu} - w_{alt}$ (Swingler, 2012) iterativ aktualisiert werden mit

$$w_{neu} = w_{alt} - \Delta w = w_{alt} - \eta \frac{\partial MSE_{alt}}{\partial w_{alt}} \quad (6.2)$$

wobei η die sogenannte 'learning rate' (Lernrate: LR), eine skalare Konstante, ist. Die LR wird normalerweise von dem Entwickler gewählt und beeinflusst das Training maßgeblich. Ein hoher Wert verursacht große Sprünge und eine schnelle Anpassung, jedoch können so gute Minima übergangen werden falls die Schritte zu groß waren.

Ein Problem stellen lokale Minima dar. So kann es aufgrund des stochastischen Aspekts der Berechnung oder aufgrund einer ungünstigen Ausgangslage der zufällig initialisierten Parameter dazu kommen, dass in der Optimierung ein Minimum gefunden wird, welches jedoch lokal und nicht global ist. Diese Situation ist sogar am wahrscheinlichsten, da es viele lokale Minima geben kann. Dadurch entsteht jedoch eine Inperfektion und zwangsläufig ein Fehler in der Klassifizierung. Die Parameter haben sich stabilisiert und kommen nicht aus dem lokalen Tiefpunkt heraus, um zu einem besseren Tiefpunkt zu gelangen. Das kann in manchen Fällen durch eine Anpassung der LR abgefangen werden, die es ermöglicht in einem weiteren Umkreis nach geeigneteren Abstiegen zu finden. Wie oben erklärt, ist eine zu große Lernrate jedoch meistens nicht wünschenswert. Deshalb wurden spezielle Algorithmen entwickelt, die mit einer adaptiven LR arbeiten, die LR also je nach Stadium in der Optimierung anpassen. So kann beispielsweise am Anfang mit größeren Schritten gearbeitet werden, da es wahrscheinlich ist, dass die zufälligen Parameter nicht nah an einem Minimum liegen. Mit einem abnehmenden Klassifizierungsfehler und einem zunehmenden Optimierungsstadium kann diese adaptive LR verringert werden, um nicht ein gutes Minimum zu überspringen. Wenn sich der Fehlerwert und damit die Optimierung stabilisiert hat, kann erneut ein weiterer Radius getestet werden, um zu schauen, ob in einem größeren Umkreis ein besseres Minimum vorliegt. 'Adam' ('Adaptive Moment Estimation') ist ein solcher Optimierungsalgorithmus, der SGD in der Implementierung ablösen kann. Zusätzlich zu der möglichen Verbesserung der Ergebnisse, erspart der Algorithmus dem Entwickler ein aufwendiges Testen verschiedener LR (Lau, 2017).

Immer wieder wird bei der Entwicklung die Generalisierung betont. Das liegt daran, dass jeder Datensatz Rauschen beinhaltet - sei es durch fehlerhafte Aufnahmen oder andere Faktoren. Wichtig ist deshalb, dass eine längere Trainingsphase deshalb oft nicht nur keine Verbesserung bringt, sondern mit ausreichend trainierbaren Parametern sogar eine genaue Spezialisierung des Netzwerks auf den Datensatz Tr inklusive seines Rauschens, was zu Problemen in der Klassifizierung von dem eigentlich wichtigen Set Te führt. Ein solcher Prozess wird auch als 'Overtraining' (Übertrainierung) bezeichnet (LeCun et al., 1998b).

6.2 Die Erwartung

Nach der theoretischen Beschreibung geht es nun an die praktische Implementierung eines solchen Netzes zur Bewältigung der Klassifikation des Blätter-Datensatzes in zehn Kategorien. Hierfür werden zunächst Netzwerke vorgestellt, die durch ihre

Funktionsweise oder ihren Erfolg neue Maßstäbe in der Technologie gesetzt haben, und anschließend eigene Konfigurationen anhand des gesammelten Datensatzes von Blättern getestet und verglichen.

Die eigenen Architekturen sollen anhand von Feststellungen und Beobachtungen variiert werden, sodass grundlegend unterschiedliche Konfigurationen verglichen werden können. Besonders die Menge an nötiger Komplexität des Netzwerks ist interessant, da sie auch der entscheidende Faktor in der Rechen- und Speichereffizienz ist. Zu erwarten ist eine größere Genauigkeit mit steigender Filteranzahl sowie eine erhöhte Genauigkeit durch genauere Filter. Am Ende soll eine Architektur gefunden worden sein, die für diese Aufgabe die Methode CNN erfolgreich implementiert und eine gute Klassifikation ermöglicht.

6.3 Historische Durchbrüche und Erfolge

Wie Markoff (2012) schreibt, haben Wissenschaftler die Datenbank ImageNet, bestehend aus menschlich kategorisierten Bildern von Objekten, im Jahr 2009 mit der Absicht veröffentlicht, neue Algorithmen für die Bilderkennung zu entwickeln und zu testen. Das ultimative Ziel der Professorin Fei-Fei Li, die an der Entstehung des Projektes beteiligt war, sei, ein Visionssystem zu entwickeln, dass die Welt erkennt wie Menschen. Zu dem Näherkommen dieses Ziels fehlten jedoch passend-skalierte Datenbanken. Nach eigenen Angaben besteht der Datensatz zur Zeit aus über 14 Millionen Bildern in 21841 Kategorien (ImageNet, 2010). Um die Forschung mit ImageNet voranzubringen, ist die Datenbank frei verfügbar. Außerdem gibt es seit 2010 eine Ausschreibung, die ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Jedes Jahr können Entwickler und Forscher auf Basis der Ausschreibung ihre besten Algorithmen einsenden, die auf bekannte Kategorien, jedoch natürlich mit neuen Fotos, getestet werden. So ist es möglich, dass in nur wenigen Jahren viele Entwicklungen in dem Bereich stattgefunden haben und eine enorme Genauigkeitsverbesserung durch kreative und innovative Ansätze unter Wissenschaftlern weltweit erzielt werden konnte, sodass, wie Aron (2015) berichtet, im Jahr 2015 Ansätze bereits die Genauigkeit von Menschen übertreffen konnte. 2017, so Gershgorin (2017), hatten nur neun von 38 angetretenen Teams einen Fehler von mindestens 5%. Da besonders hier Fortschritte gemacht wurden, werden hier einige bahnbrechende und innovative Einsendungen auf Basis der Veröffentlichungen von Singh (2016) und Das (2017) vorgestellt. Die Ergebnisse des Wettbewerbs basieren auf der Veröffentlichung von Russakovsky et al. (2015).

LeNet-5: Ist ein wichtiger Schritt unter faltenden neuronalen Netzwerken gewesen. Die von LeCun et al. (1998a) vorgestellte Architektur wurde zu dem Zweck der Zahlenerkennung entwickelt. Dieses heute häufig angeführte Beispiel für Bilderkennung ermöglichte mit einem Datensatz von $28 * 28$ Pixel Scans handgeschriebener Zahlen und sieben konsekutiven Schichten von 'Sigmoid-Neuronen' eine Genauigkeit

⁸Bedeutet: Die Merkmale, die erkannt werden sollen, passen von den Maßen in eine $5 * 5$ -Matrix. Was im Folgenden nicht immer dazu gesagt wird, ist der Abstand zwischen den Anwendungen der Filtermatrix auf das Bild. Hier werden die Werte von x, y jeweils um eins verändert. Dieser Abstand hängt einerseits davon ab, wie genau man aneinanderliegende Bildabschnitte analysieren möchte und andererseits, welche resultierenden Maße für die Ergebnismatrix erreichen werden sollen.

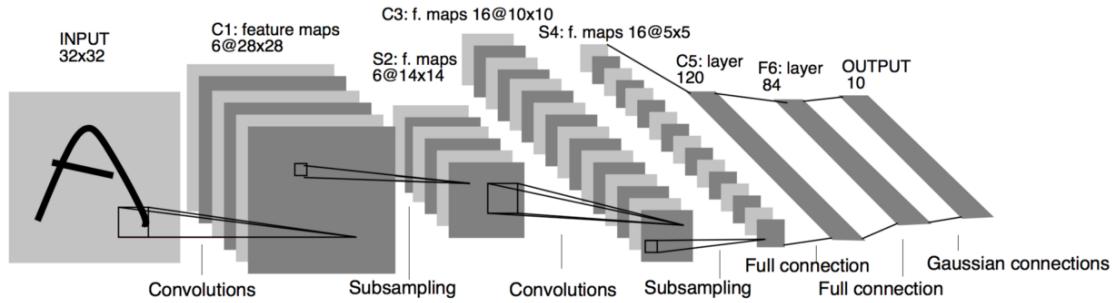


Abbildung 6.4: LeNet, Quelle: LeCun et al. (1998a)

von 0,992 unter Verwendung von Data Augmentation und einem hierbei resultierenden Umfang des Trainingsdatensatzes von 600.000 Beispielen. Alle drei vollen CL haben eine Filterdimension $n*n$; $n = 5$.⁸ Auf die Input Schicht der Größe $32*32$ Pixel (an die ursprüngliche Skalierung der Scans wurde ein weißer Rahmen gelegt, um einen Abstand der Zahlen zu dem Rand des Bildes sicherzustellen) folgen zunächst sechs Filter, die Ergebnis-Matrizen der Größe $28 * 28$ kalkulieren. Nach einer Dimensionsreduzierung um $1/4$ (die Operation liefert die Schicht $S2$ mit den Maßen $14*14$) durch einen $2*2$ PL folgt eine weitere Schicht von 16 CL, die $10*10$ Ergebnis-Matrizen berechnen. Auch diese Schicht wird reduziert zu $16 5*5$ (also wieder eine Reduktion zu $1/4$) Matrizen. Nach zwei dichten Schichten folgt die Ausgabeschicht mit einem zehndimensionalen Vektor (stehend für die zu erkennenden Zahlen 0 bis 9). Die PL hier sind keine wie in der Theorie erläuterten Max-Pooling Schichten. Sie haben ebenfalls trainierbare Parameter; die einzelnen Aktivierungen der zu reduzierenden $2*2$ -Felder werden addiert, mit einem trainierbaren Faktor multipliziert und zu einer ebenfalls trainierbaren Gewichtung addiert. Der Faktor und die Gewichtung sind einheitlich für jede Ergebnis-Matrix; es gibt also 6 individuell- trainierbare Faktoren und 6 individuell-trainierbare Gewichtungen. Gesamt führt diese Architektur zu genau 60.000 trainierbaren Parametern (kumulierte Faktoren, Gewichtungen und Tendenzen).

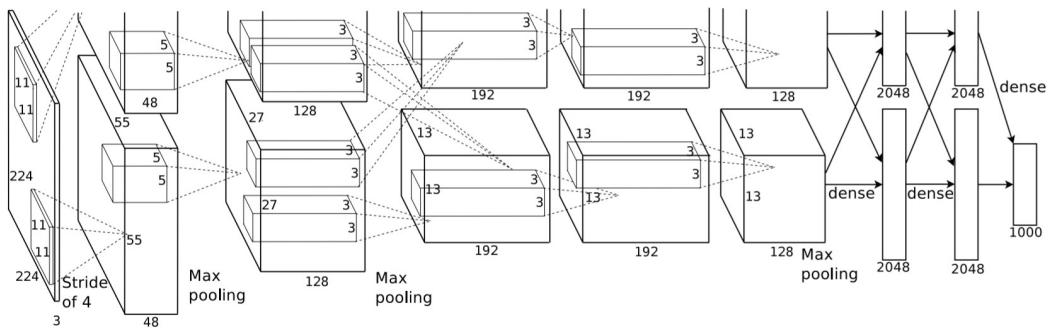


Abbildung 6.5: AlexNet, Quelle: Krizhevsky et al. (2012)

AlexNet: Knapp 14 Jahre später erreichte diese Architektur von Krizhevsky et al. (2012) bei der ILSVRC im Jahr 2012 den ersten Platz. Mit 1.000 mal so vielen trainierbaren Parametern (60 Millionen) und einer erheblich komplexeren Struktur auf der erkennbaren Grundlage von LeNet-5, nach Singh (2016) relativ gesehen

jedoch einem simplen Layout, wurden neue Techniken implementiert, die so 1998 noch nicht entwickelt waren. Auf Basis experimenteller Ergebnisse entschieden sich Krizhevsky et al. (2012) für fünf CL und zwei anschließende dichte Schichten, die, aus Gründen der Effizienz, parallel auf zwei Grafikkarten berechnet werden (deshalb die parallele und vernetzte Struktur). Der durch die Bildgröße von $224 * 224$ Pixeln in Farbe zustandekommende Dimensionalität von 150.528 in der Eingabeschicht ist bereits ein Indikator für die fortlaufende Größe der Architektur. Insgesamt haben die CL 6.144 Filter plus die trainierbare Vernetzung in den vier 2.048-dimensionalen dichten Schichten. Bei dieser Konfiguration ist Maxpooling jeweils nach den ersten beiden und dem letzten CL implementiert. Die Filter haben zunächst die Maße $n * n; n = 11$, in dem folgenden CL von $5 * 5$ und bei den letzten drei CL von $3 * 3$. Herausstechend ist diese deutliche Verkleinerung des Rezeptionsfeldes. Krizhevsky et al. (2012) liefern keine Erklärung für dieses Vorgehen, was auf erhebliche Rechenanstrengungen in der Entwicklung und empirischen Optimierungen schließen lässt. Wie bei LeNet-5 wurde auch hier im Training Data Augmentation genutzt. Darüber hinaus gibt es viele Verbesserung hinsichtlich des Lern-Algorithmus, Dropout und eine Implementierung der ReLU Aktivierungsfunktion nach jedem CL und jeder dichten Schicht.

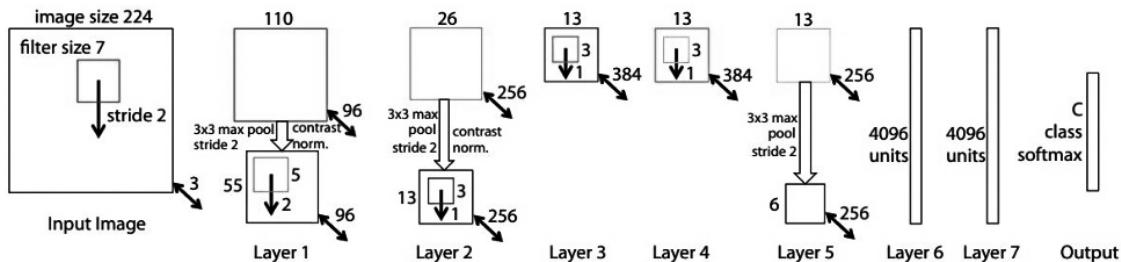


Abbildung 6.6: ZFNet, Quelle: Zeiler und Fergus (2013)

ZFNet: Diese Architektur von Zeiler und Fergus (2013) basiert auf der Alex-Net Struktur, ist jedoch leicht modifiziert und konnte deshalb im Folgejahr 2013 den ersten Platz in dem Wettbewerb erreichen. Technisch ist hier keine Innovation vorhanden. Sieben Schichten führen zum Ausgabe-Vektor, auf manche CL folgt eine $3 * 3$ Maxpool-Schicht. Zwei interessante Unterschiede bestehen jedoch darin, dass ZFNet die maximalen Filterdimensionen auf $7 * 7$ Pixel reduziert hat mit der Begründung, mehr Pixelinformationen mit zunehmender Netzwerktiefe zu erhalten (Singh, 2016), und dass der Datensatz weniger als 10% der Einzelbilder von AlexNet enthielt.

GoogLeNet: Mit neun sogenannten 'Inception Modules' postulieren Szegedy et al. (2014) einen von der streng-konsekutiven Struktur aufeinanderfolgender CL abweichenden Aufbau. Sie verschachteln mehrere Filter unterschiedlicher Dimensionen, die parallel kalkuliert werden und durch eine Konkatenation zusammengefasst werden. Durch Dimensionsreduzierung mit average-pooling (es werden Durchschnittswerte für bestimmte Bereiche berechnet) fallen dichte Schichten komplett weg. Was die Tiefe des Netzes betrifft, setzt sich diese hier offiziell in der Fassung der Wettbewerbseinsendung aus 22 Schichten zusammen. Diese trainierte Architektur passt von der Konfiguration nicht in das Bild der klassischen CNN, hat jedoch

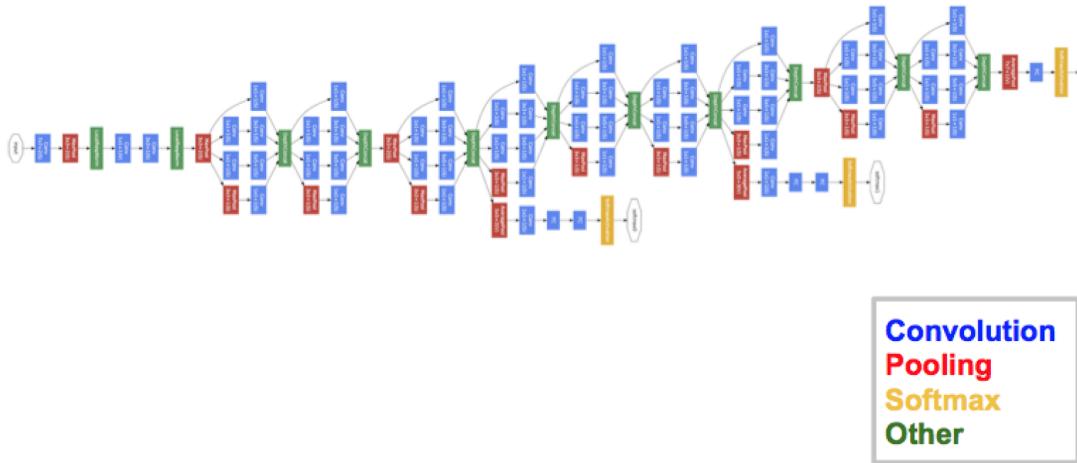


Abbildung 6.7: GoogLeNet, Quelle: Das (2017)

im Jahr 2014 den ersten Platz des Wettbewerbs belegt. Betreffend der Anzahl trainierbarer Parameter ist hier ebenfalls eine Überraschung: das Netz hat nur 4 Millionen trainierbare Parameter. Szegedy et al. (2014) haben möglicherweise eine neue Forschungsrichtung für Architekturen eingeleitet, die nicht mehr klassischen CNN entsprechen, sondern mit ihnen nur noch die mathematische Faltungsoperation gemeinsam haben.

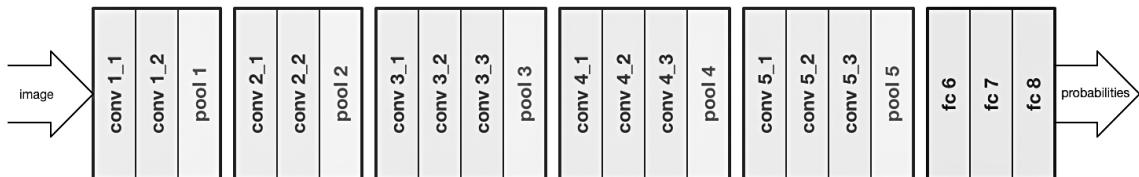


Abbildung 6.8: VGGNet, Quelle: Das (2017)

VGGNet: Platz Nummer zwei im Jahr 2014 wurde von der VGGNet-Architektur nach Simonyan und Zisserman (2015) belegt. Die Struktur ist tief aber simpel. Die gewinnende Konfiguration bestand aus 21 Schichten ohne Eingabe und Ausgabe, davon 13 CL und 3 dichte Schichten. Die CL sind in fünf Blöcke eingeteilt, die je von einem PL gefolgt werden. Alle Filter haben die Maße $n * n$ von $3 * 3$, was einen erheblichen Unterschied zu den vorherigen Ansätzen darstellt. Außerdem wird die Tiefe eines jeden Ergebnis-Matrix-Blocks, also CL, von Block zu Block (außer bei den letzten beiden) verdoppelt, sodass in den letzten beiden Blöcken in jeder Schicht 512 Faltungs-Ergebnis-Matrizen vorliegen. Die drei dichten Schichten folgen konsekutiv auf den letzten PL mit einer Dimensionalität in der letzten dichten Schicht von der Ausgabeschicht von 1000. Das (2017) merkt jedoch an, dass die resultierenden 138 Millionen trainierbaren Parameter in der Verwendung eine Herausforderung darstellen können. Davon zeugt auch die reine Trainingszeit des Netzwerkes in der Entwicklung von zwei bis drei Wochen mit Hilfe von vier GPUs (Graphics Processing Unit - Grafikprozessor) (Singh, 2016) - einem Rekord in dieser Liste in der Länge.

ResNet: Auch 2015 hat ein CNN die ILSVR Challenge gewonnen. He et al.

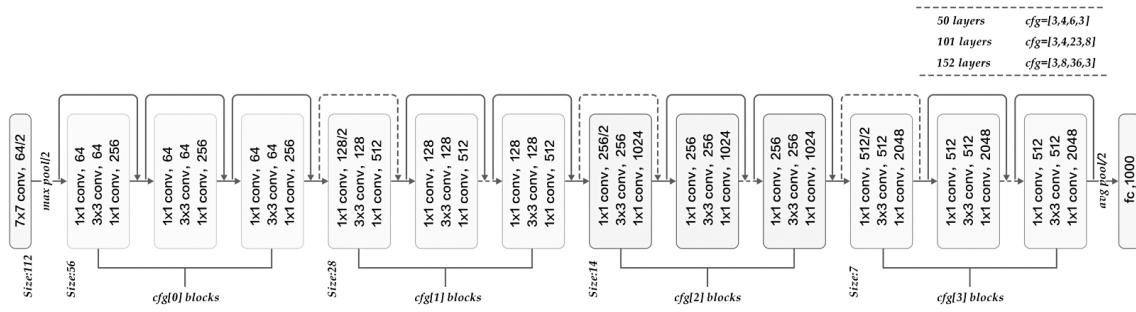


Abbildung 6.9: ResNet, Quelle: Das (2017)

(2015) von Microsoft Research haben dieses Netzwerk ähnlich wie RNN (Recurrent Neural Networks - Wiederkehrende Neuronale Netzwerke), eine Form von künstlichen NN, die ebenfalls erfolgreich entwickelt wird, eingerichtet (Das, 2017) und damit eine ähnlich komplexe Struktur wie GoogLeNet implementiert. Die Aktivierungen einer Schicht werden nicht nur direkt an die nächstfolgende Schicht weitergegeben, sondern auch über Schichten hinweg verarbeitet.⁹ Die Struktur mit Residual-(Restwert)-Blocks schafft es so die menschliche Erkennungs- und Klassifizierungsfähigkeit zu übertreffen (Das, 2017). He et al. (2015) zeigen, dass schichtweise Erweiterungen von 34 zu 152 Schichten weiterhin Leistungsverbesserungen liefern, was bei streng-konsekutiven Strukturen nicht funktioniert. Durch die groß skalierte Architektur geht hier jedoch auch eine rekordbrechende Trainingszeit in dieser Auflistung von zwei bis drei Wochen auf einer mit acht GPUs ausgestatteten Maschine einher.

Jahr	Name	# train. Schichten	# train. Parameter
1998	LeNet-5	6	60.000
2012	AlexNet	7	60.000.000
2013	ZFNet	7	
2014	GoogLeNet	22	40.000.000
2014	VGGNet	16	138.000.000
2015	ResNet	152	60.200.000

Abbildung 6.10: Überblick über die Netze, 'train.' für 'trainierbare'

Generell lässt sich feststellen, dass die Anzahl der trainierbaren Schichten eine ansteigende Tendenz aufweist, womit logischerweise auch die Anzahl der trainierbaren Parameter ansteigt. Dies lässt sich sicher auch auf das kontinuierliche Wachstum der möglichen Rechenleistung zurückführen. Amodei und Hernandez (2018) führt konkret algorithmische Entwicklung, verfügbare Daten und Rechenleistung als Motor für die voranschreitende Entwicklung im Bereich Künstliche Intelligenz auf. Im Gegensatz zu Moore's Gesetz, veröffentlicht 1965 nach David House's Voraussage, nach dem sich die Rechenleistung hardwaretechnisch alle 18 Monate verdoppeln soll-

⁹Wie auf der Abbildung zu sehen, werden Zwischenergebnisse nicht nur von der direkt darauf folgenden Schicht weiter verarbeitet wie bei klassischen CNN, sondern auch an Schichten weitergegeben, die sowohl diese Zwischenwerte als auch die Werte der Verarbeitung zwischen diesen beiden Schichten verarbeitet.

te, zeigen Amodei und Hernandez (2018), dass sich die benutzte Rechenleistung für hochentwickelte Systeme der Künstliche Intelligenz seit 2012 alle 3,5 Monate verdoppelt.

In den Folgejahren 2016 und 2017 haben ebenfalls Systeme gewonnen, die CNN implementieren. Die beiden Teams CUIImage und BDAT (in der Reihenfolge der Jahre) haben beide komplexe Systeme entwickelt, die eine Mehrzahl von Algorithmen für bestmögliche Effizienz und Schnelligkeit nutzen.

6.4 Eigene Architekturen im Vergleich

Die Implementierung erfolgte hier in der Programmiersprache Python. Python ist seit 1991 aktiv in der Entwicklung und eine interpretierte höhere Programmiersprache. Für dieses Projekt ist sie nicht nur durch ihre Effizienz, Plattformunabhängigkeit, klare und generell gut strukturierbare Syntax geeignet, sondern auch aufgrund der vielen Erweiterungsmöglichkeiten, die durch das offene und gemeinschaftsbasierte Entwicklungsmodell bestehen. Hier hätte alternativ natürlich auch mit C++ oder Java gearbeitet werden können. Mit Python steht das TensorFlow Framework zur Verfügung, das in Python und C++ implementiert ist. Es wurde von dem Google Brain Team entwickelt und 2015 veröffentlicht und ermöglicht eine Implementierung vieler Methoden des Maschinellen Lernens mit bereits teilweise eingerichteter Optimierung, um solche Techniken auf handelsüblichen Rechenmaschinen überhaupt erst zu ermöglichen. In der Implementierung wurde aus Effizienzgründen auf den Optimierungsalgorithmus Adam zurückgegriffen, um nicht unnötig viel Zeit mit dem Testen verschiedener LR zu verwenden, die Ergebnisse zu verbessern und die Rechenzeit zu verringern. Schichten neuronaler Netze wie oben dargestellt werden als Graphen behandelt, deren Operationen sequentiell ablaufen können. Als zusätzliche Erweiterung wurde Keras benutzt, das als Schnittstelle zwischen Backend-Frameworks wie TensorFlow und Entwicklern erstellt und von Francois Chollet 2015 erstmals vorgestellt wurde. Keras ist in Python geschrieben und ermöglicht darüber hinaus für die Berechnung der Operationen eine Verwendung der GPU statt CPU (Central Processing Unit - Prozessor), was bei dieser Anwendung schnell eine Geschwindigkeitsoptimierung im Training um Faktor 10 bedeuten kann (Lazorenko, 2017).

Um einen Mindestgrad von Reproduzierbarkeit zu ermöglichen und einen kleinen Einblick in die ständigen Probleme der Projektumsetzung zu geben, wird im Folgenden der Auswahlprozess der benutzten Hard- und Software beschrieben.

Die in der Primärliteratur oft beinahe lapidar erwähnte Implementierung stellte mich gerade am Anfang vor große, weitgehend technische Hindernisse, die unnötig viel Zeit in Anspruch nahmen. Es wurde davon ausgegangen, dass ein Tower-PC mit einer E3 Server-CPU von Intel gepaart mit einer GeForce GTX-650 Ti von NVIDIA mit einer Linux Distribution der Aufgabe ausreichend gewachsen wäre und ein effizientes Arbeiten ermöglichen würde. Nach zahlreichen Fehlschlägen bei der Treiberinstallation der Grafikkarte, die für die Berechnung der Algorithmen eine CUDA Grundlage erfordert (Die Compute Unified Device Architecture wurde von Nvidia entwickelt und ist eine Programmier-Technik, die es ermöglicht parallelisier-

bare Programmabläufe effizienzverbessernd auf der GPU berechnen zu lassen), also einer vermeintlichen Treiber-Inkompatibilität zwischen den verschiedenen beteiligten Software, wurde das Betriebssystem von Debian auf Ubuntu gewechselt, was jedoch ohne Überraschung keine Besserung brachte, da Ubuntu eine, wenn auch eigenständige, auf Debian basierende Distribution ist. Es wurde anschließend auf ein zuvor aus Gründen der Effizienz und Minimierung abgelehntes Windows 10 Betriebssystem gewechselt, welches leichtere Treiberinstallationen versprach, jedoch Komplikationen in Bezug auf die Programmierung brachte, da Python, im Gegensatz zu der problemlosen Kompatibilität mit Linux-Systemen, nicht nativ mit einem Windows-System entwickelt und ausgeführt werden kann.¹⁰ Die bei Linux auftretenden Fehler in diesem Bereich sind nach Internetrecherchen aufgrund der vielen verschiedenen inkompatiblen Software Versionen eine Häufigkeit. Die Treiberinstallation erfolgte erfolgreicher bei Windows, wobei auch hier einige Fehler auftraten, die jedoch nach ausgiebiger Fehlerbehebung größtenteils behoben werden konnten. Bei dem Testen eines Beispielprojekts stellte sich heraus, dass die Grafikkarte den Anforderungen von Tensorflow aufgrund einer CUDA Computability (Größe zum Messen der Rechenleistung der GPU von Nvidia) von 3.0 nicht gewachsen ist und somit per Definition als funktionierende Peripherie ausgeschlossen ist. Für die Entwicklung wurde schließlich ein Laptop mit Intel i7 CPU und einer NVIDIA GeForce GTX950M Grafikkarte, die eine CUDA Computability von 5.0 besitzt, gewählt. Außerdem läuft das System auf 16 Gigabyte Arbeitsspeicher und einer 5400rpm HDD. Die Inkompatibilitäten konnten letztendlich alle beseitigt werden und das Sample-Projekt konnte erfolgreich gestartet werden, was den Weg für die Entwicklung des eigentlichen Projektes ebnete.

6.4.1 Präsentation und Auswertung verschiedener Konfigurationen

Insgesamt wurden für diesen Datensatz 12 Modelle entwickelt und getestet, die in vier grundlegende Architekturen eingeteilt werden können. Implementiert wurde das ganze mit dem gesammelten Datensatz der pflanzlichen Blätter, der eine grob gleich-verteilte Anzahl Bilder für jede der zehn Kategorien hat.¹¹ Da das Verhältnis $\frac{\#Tr}{\#Te} = \frac{4}{1}$, werden die ≈ 365 Bilder des kompletten Datensatzes zufällig aufgeteilt in 292 Bilder für das Training aus allen Kategorien und 73 Bilder für das Testen. Die Reihenfolge der Aufführung der folgenden Architekturen entspricht ungefähr der des Testens. Modifizierte Konfigurationen entstanden vor allem aus Inspirationen der Literatur, deren Übertragbarkeit hier getestet wird, und experimenteller Feststellungen, die hier während des Prozesses getroffen werden konnten. Benennung der Tabellen:

Name der Schicht	Dimensionen der Ausgabe-Matrix	# trainierbarer Parameter
------------------	--------------------------------	---------------------------

Nicht eingeschlossen ist die Eingabeschicht des ursprünglichen Bildes.

¹⁰Python ist plattformunabhängig, kann aber auf Linuxsystemen über die Eingabeaufforderungen ('Terminal') sowohl geschrieben als auch direkt ausgeführt werden. Diese Möglichkeit gibt es bei Windows nur über erweiternde Software wie 'Anaconda'.

¹¹Leichte Variation ist hier vernachlässigt. Größere Unterschiede würden zu Problemen führen, da logische Fehler auftreten könnten. Dies könnte sich so äußern, dass sich die Wahrscheinlichkeit

Benennung der Schichten:

Conv: CL, MaxPool: MP, Flatten: Totale Verknüpfung aller Neuronen zur Reduzierung der Tiefe, Dicht: Trainierbare totale Verknüpfung aller Neuronen, Dropout : DO

Alle praktischen Tests wurden mehrfach durchgeführt. Wo nicht mehrere Ergebnisse graphisch dargestellt sind, wurden diese gewählt, die im Sachzusammenhang von der größten Aussagekraft zeugen, beziehungsweise repräsentativ sind.

ein Bild der Kategorie 1 vorliegen zu haben erhöht, wenn die meisten Bilder des Trainingssatzes aus der Kategorie 1 sind - dabei sollen keine statistischen Auftrittswerte für die Klassifikation genutzt werden (da sie nur fälschlicherweise vorliegen), sondern Muster erkannt werden.

Architektur a_0 : VGG

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*64	1.792^{12}
MaxPool	32*32*64	0
Conv	32*32*128	73.856
MaxPool	16*16*128	0
Conv	16*16*256	295.168
Conv	16*16*256	590.080
MaxPool	8*8*256	0
Flatten	16.384	0
Dicht	4.096	67.112.960
Dropout	4.096	0
Dicht	4.096	16.781.312
Dropout	4.096	0
Dicht	1.000	4.097.000
Dicht	10	10.010
		$\sum 88.962.178$

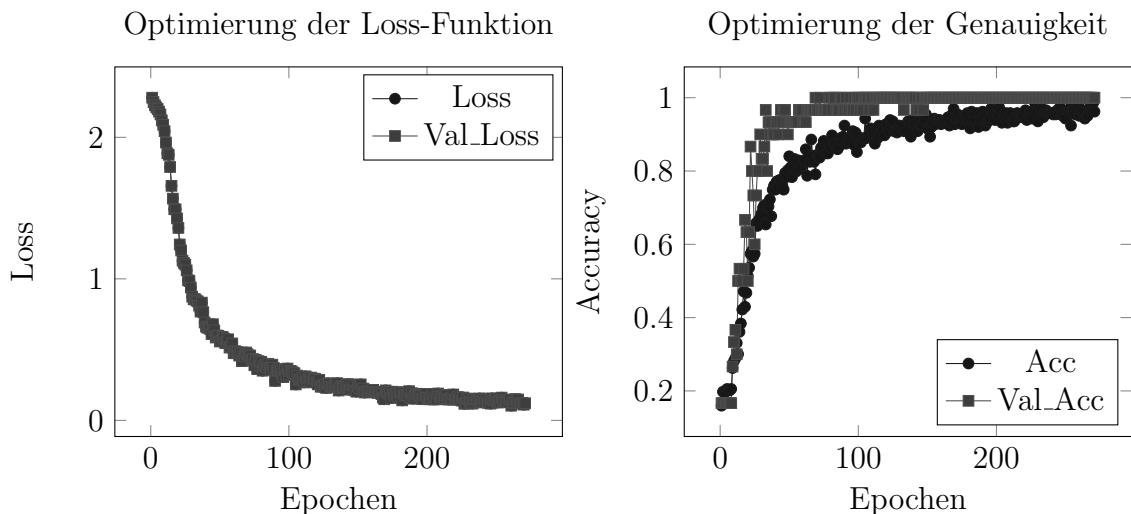


Abbildung 6.11: Für alle folgenden vergleichbaren Grafiken: 'Val' ist der direkte Abgleich beim Training mit den aktuellen Parametern mit einem Validierungssatz, der keine Überschneidung mit Tr hat.

Genauigkeit: 0,9459 ¹³

Zuerst sollte eine VGG-Architektur getestet werden. Wie an den Schichten zu sehen, ist dies nicht die Wettbewerbs-Konfiguration D, sondern die in Bezug auf Schichten und Parameter reduzierte Konfiguration A in einer 3-Conv-Block-Version. Entsprechend gibt es vier CL mit abnehmenden Größen der Ergebnis-Matrizen und

¹²Zustandekommen: $3 \times 3 \times 3$ -Filterdimension, 64 Filter, 64 Tendenzen $\Rightarrow 3 \times 3 \times 3 \times 64 + 64 = 1792$.

¹³Das ist die finale Genauigkeit, die nach dem Training mit den berechneten Parametern erzielt wird.

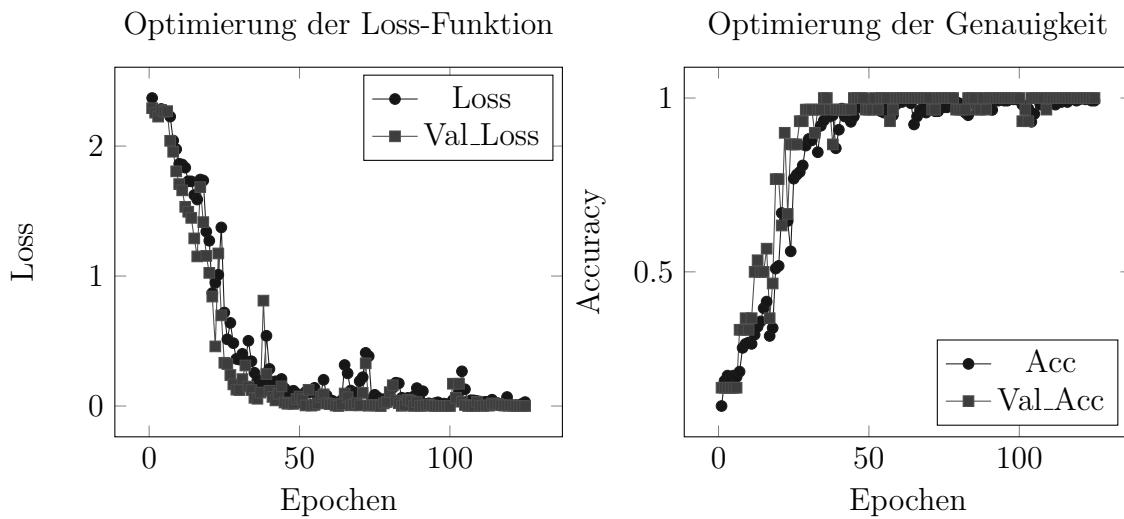
einer zunehmenden Anzahl an Filtern. Zusätzlich wurde hier im Training Dropout benutzt. Festzustellen ist die Tiefe des Netzes mit acht trainierbaren Schichten und die im Vergleich zu anderen ILSVRC-Netzen hohe Anzahl an Parametern. Die letzte Schicht ist natürlich - wie auch in allen späteren Architekturen - eine dichte Schicht, die zu den zehn möglichen Kategorien des Datensatzes führt.

Die Optimierung der Loss-Funktion erfolgt in einer Form ähnlich einer Hyperbel und scheint sich mit Blick auf die Validation Accuracy - der erreichten Genauigkeit auf dem Trainingsset, nach dem die Gewichte und Tendenzen optimiert werden - bereits nach circa 100 Trainingsepochen (kompletten Durchläufen) stabilisiert zu haben. Anhand der tatsächlichen Accuracy (Genauigkeit), die als kleiner Referenz Datensatz nach jedem Trainingslauf getestet wird, ist jedoch auch nach 100 Epochen noch eine konstante Optimierung zu erkennen. Nach Beendigung des Trainings mit 250 Epochen¹⁴ wird im Test - beim Abgleich des Algorithmus mit dem Test-Datensatz - eine Genauigkeit von knapp 95% berechnet. Angesichts der Tiefe des Netzes, der dementsprechend hohen Anzahl trainierbarer Parameter und zeitaufwändigen Berechnung scheint dieses Netzwerk für diesen Datensatz nicht sehr geeignet zu sein, kann aber aufgrund der bereits guten erzielten Genauigkeit als Basis für weitere Architekturen genommen werden.

¹⁴In diesem Training ist der komplette Datensatz $\#Tr$ 250-mal durch den Algorithmus zur Anpassung der Parameter gelaufen.

Architektur a_1 : Nguyen64

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*64	1.792
Conv	64*64*64	36.928
Conv	64*64*64	36.928
MaxPool	32*32*64	0
Dropout	32*32*64	0
Conv	32*32*128	73.856
Conv	32*32*128	147.584
MaxPool	16*16*128	0
Dropout	16*16*128	0
Conv	16*16*256	295.168
MaxPool	8*8*256	0
Dropout	8*8*256	0
Flatten	16.384	0
Dicht	256	4.194.560
Dropout	256	0
Dicht	10	2.570
		$\sum 4.789.386$



Genauigkeit: 0,9865

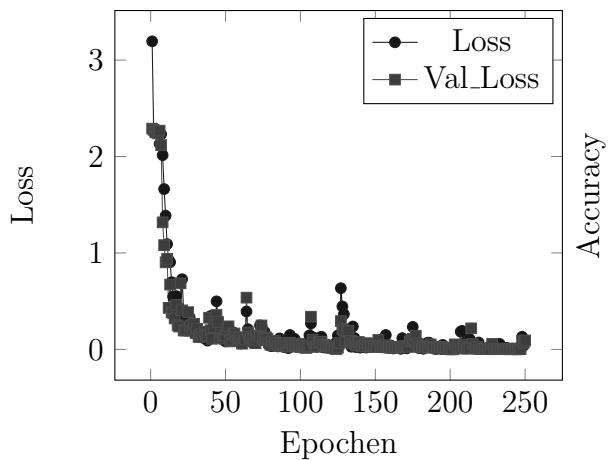
Dieses Netzwerk wurde von Nguyen und Zeigermann (2018) zur Klassifizierung von Straßenschilder-Fotos in sechs Kategorien entwickelt. Die Entwickler geben an, dieses Netzwerk auf Basis der VGG-Architektur entworfen zu haben, was in dieser Weiterentwicklung ebenfalls Sinn ergeben kann. Im Vergleich zu der VGG-Architektur beträgt die Parameteranzahl hier aufgrund der reduzierten Anzahl an komplexen dichten Schichten nur 5,38% der vorherigen.

Durch die Vermehrung der Filter und der Reduzierung der dichten Schichten scheint eine bessere Klassifizierung möglich. Die Fehlerwahrscheinlichkeit von unter 2% zeigt eine deutliche Verbesserung - und das bei stark reduzierter Trainingszeit.

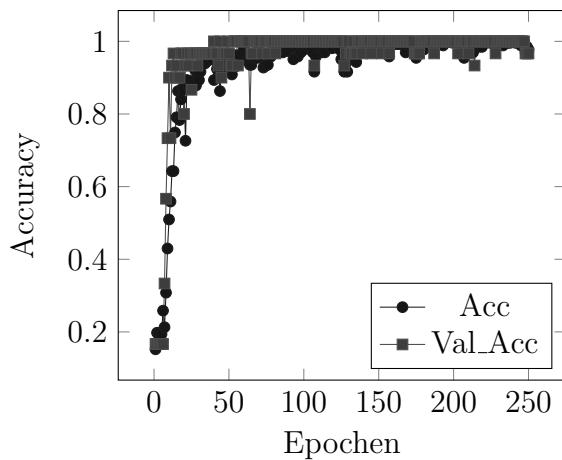
Architektur a_2 : Nguyen128

Schicht	Dim. Ausgabe	Parameter
Conv	128*128*128	3.584
Conv	128*128*128	147.584
Conv	128*128*128	147.584
MaxPool	64*64*128	0
Dropout	64*64*128	0
Conv	64*64*256	295.168
Conv	64*64*256	590.080
MaxPool	32*32*256	0
Dropout	32*32*256	0
Conv	32*32*512	1.180.160
MaxPool	16*16*512	0
Dropout	16*16*512	0
Flatten	131.072	0
Dicht	512	67.109.376
Dropout	512	0
Dicht	10	5.130
		$\sum 69.478.666$

Optimierung der Loss-Funktion



Optimierung der Genauigkeit



Genauigkeit: 0,9730

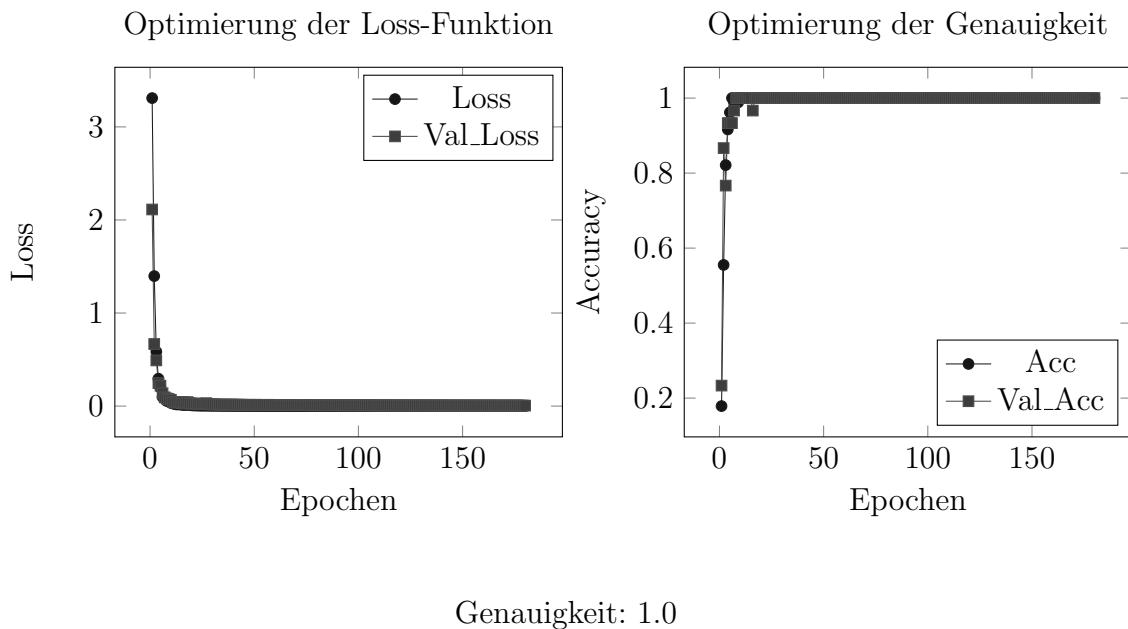
Die Bilder des Datensatzes haben eine maximale Pixelauflösung von $512 * 512$ Pixel. Da in den vorherigen Architekturen nur eine skalierte Auflösung von $64 * 64$ Pixel verarbeitet wurde, wurden vorhandenen Informationen der Bilder reduziert - die Genauigkeit innerhalb des Bildes hat sich dadurch aufgrund von Detailverlust reduziert. Hier sollte nun getestet werden, ob bei gleicher Architektur, aber in einer Version mit vierfacher Auflösung von $128 * 128$ Pixel eine bessere Genauigkeit in der Klassifizierung zu erzielen ist. Aus Gründen der Einheitlichkeit in den Dimensionen der einzelnen Schichten und aufgrund der erhöhten Anzahl an Details innerhalb der Bilder ist die Anzahl der Filter in den CL proportional zu den Außenmaßen der

Bilder gewachsen.

Sichtbar wird, dass auch hier eine gute Optimierung funktioniert. Die Stabilisierung setzt jedoch erst etwas später ein; vermutlich, da mehr Parameter optimiert werden müssen. Festzustellen ist jedoch auch, dass die Genauigkeit trotz stark erhöhter Anzahl an Parametern und einer Erhöhung der Filteranzahl abnimmt im Gegensatz zu der skalierten Version der Architektur vorher. Das ist entgegen der Erwartung, dass mit mehr Details in den Bildern eine höhere Klassifizierungsgenauigkeit erzielt werden kann. So scheint es, dass die Vergrößerung der Maße keinen positiven Beitrag leistet, sondern entweder Details hinzugibt, die widersprüchlich auf den Algorithmus wirken und möglicherweise verwirren, oder die erhöhte Anzahl der Parameter eine Optimierung erschwert. Die zweite Erklärung würde auch zu der Beobachtung passen, die im Vergleich der ersten beiden Architekturen zu sehen ist.

Architektur b_0 : 5f

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*5	140
Flatten	20.480	0
Dicht	10	204.810
		$\sum 204.950$

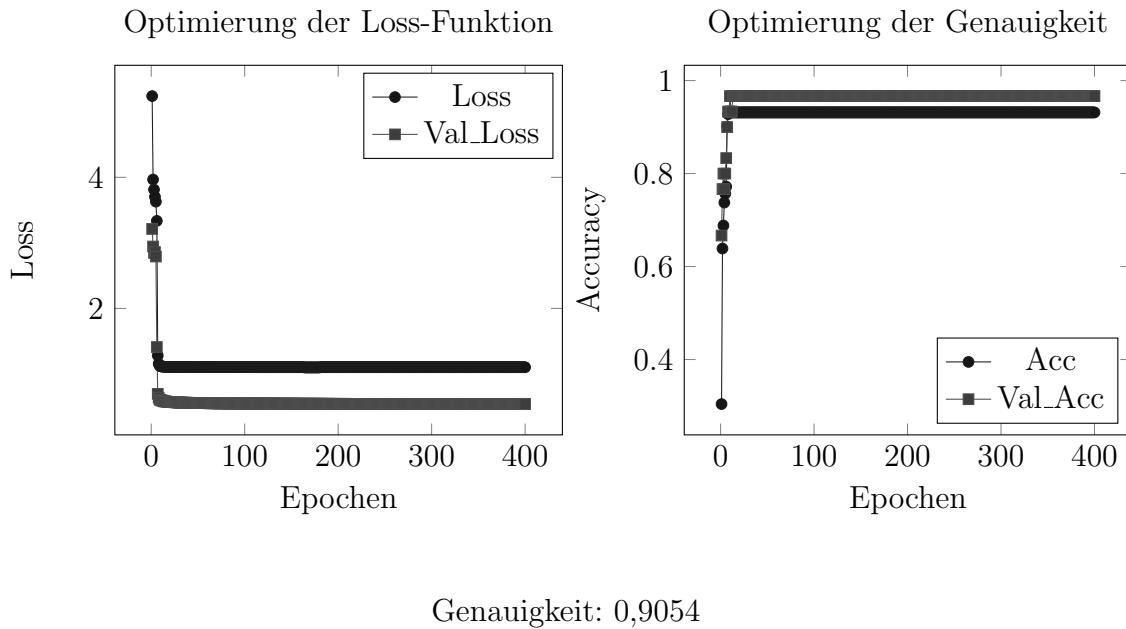


Diese total simple Architektur wurde aus der vorherigen Feststellung heraus entworfen und sollte ein absurd-simpler Test sein, was der Algorithmus mit nur einem CL, fünf Filtern und ohne Erweiterungen wie Dropout, PL oder dichten Schichten erreichen kann.

Das Ergebnis ist umwerfend und überraschend: ein perfektes Ergebnis wird erzielt. Aufgrund der kleinen Anzahl an Parametern ist die Optimierung nach weniger als 50 Epochen abgeschlossen; bei dem Testen des Algorithmus mit einem neuen Datenset werden 100% der Bilder richtig klassifiziert. Diese Beobachtung ist entgegen aller Erwartungen und stützt die These weiter, dass weniger Parameter eine bessere Optimierung ermöglichen. Es ist stark zu erwarten, dass der Erfolg dieser Architektur ein Zufallstreffer ist, da ein ähnliches Ergebnis zum Beispiel bei der originalen VGG-16-Architektur nicht beobachtet werden konnte. Der Erfolg der verschiedenen Konfigurationen korreliert dort nicht mit der Einfachheit des Netzwerks, sondern gerade mit dem Gegenteil, weshalb eine komplexere Version den Siegesplatz der ILSVRC belegt hat.

Architektur b_1 : 10f

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*10	280
Flatten	40.960	0
Dicht	10	409.610
		$\sum 409.890$

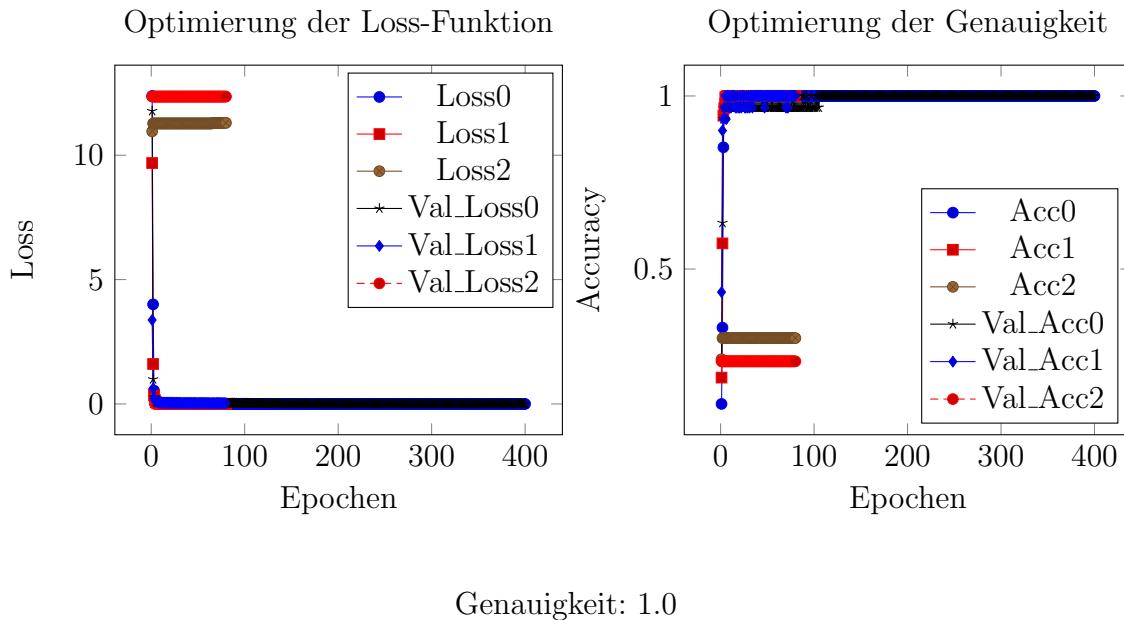


Nun sollte geschaut werden, ob das gute Ergebnis aus der vorherigen Architektur mit einer skalierten Version mit zehn Filtern ebenfalls solch ein gutes Ergebnis liefern würde. Die Genauigkeit von 1.0 kann natürlich nicht verbessert werden, dennoch sollte dieser Versuch Ergebnisse über das Verhalten mit mehr Parametern liefern und prüfen, ob die simple Architektur das gute Ergebnis bedingt hat, oder ob es ein Zufallstreffer war.

Tatsächlich ist dieses Ergebnis wieder überraschend. Mit nahezu verdoppelten Parametern erzielt das trainierte Netzwerk eine erheblich niedrigere Genauigkeit von nur knapp 90%. Weder die Validierungsgenauigkeit noch die Test-Genauigkeit werden annähernd perfekt optimiert - nicht einmal der Trainingsdatensatz Tr kann mit 100% Genauigkeit klassifiziert werden. Dies lässt darauf schließen, dass das vorherige Ergebnis ein Zufallstreffer war, dessen Wert jedoch nicht relativiert werden sollte.

Architektur b_2 : 64f

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*64	1.792
Flatten	262.144	0
Dicht	10	2.621.450
		$\sum 2.623.242$



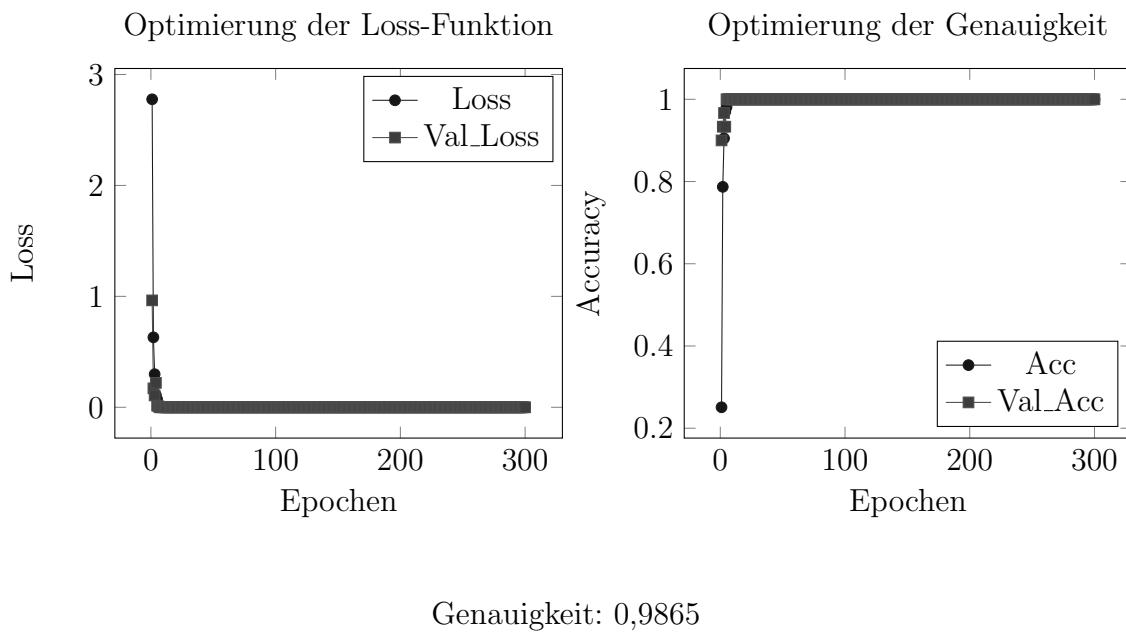
Genauigkeit: 1.0

Im Zuge der minimalen 1-Block-Architektur mit einem Block von CL, der hier aus einem CL besteht, sollte nun eine weitere Version mit 64 Filtern getestet werden, um weitere Möglichkeiten zu testen und das Verhalten der Optimierung und der Genauigkeit in Hinblick auf die getestete Architektur zu erforschen - alle weiteren Einstellungen wie in den vorherigen beiden Konfigurationen.

Zur Veranschaulichung und für spätere Diskussionen sind hier drei verschiedene Trainingsläufe abgebildet. Es wird wieder eine Genauigkeit von 1.0 erreicht - ein perfektes Ergebnis. Die Optimierung verläuft schnell und gut in zwei von den drei abgebildeten Trainingsprozessen. Der dritte Durchlauf (Beziffert 2) wird abgebrochen, da keine weitere Optimierung erfolgt, obwohl das Ergebnis nicht zufriedenstellend ist. Nach dem Erreichen einer Accuracy von knapp über 20% muss wohl ein lokales Minimum durch den Optimierungsalgorithmus gefunden worden sein, welches vermutlich an der ungünstigen Initialisierung der Parameter lag und eine weitere Optimierung unmöglich machte. Dass dies die Ausnahme ist, zeigt die starke Korrelation der Graphen der ersten beiden Durchläufe (0 und 1), die sehr ähnliche Optimierungsverhalten aufweisen und beide zu einem guten Ergebnis in der Klassifizierung führen. Interessanterweise kann hier festgestellt werden, dass die perfekte Genauigkeit bei 5f und bei 64f erzielt wird, jedoch nicht bei 10f. Eine logische Erklärung hierzu scheint es nicht zu geben. Festzuhalten gilt jedoch, dass durch diese drei Architekturen gezeigt wurde, dass in diesem Anwendungsfall der Klassifizierung von Blättern eine simple Architektur mit verhältnismäßig wenigen trainierbaren Parametern und Schichten zum Ziel führen kann.

Architektur c_0 : 3*3

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*64	1.792
Conv	64*64*64	36.928
Conv	64*64*64	36.928
Flatten	262.144	0
Dicht	10	2.621.450
		$\sum 2.697.098$

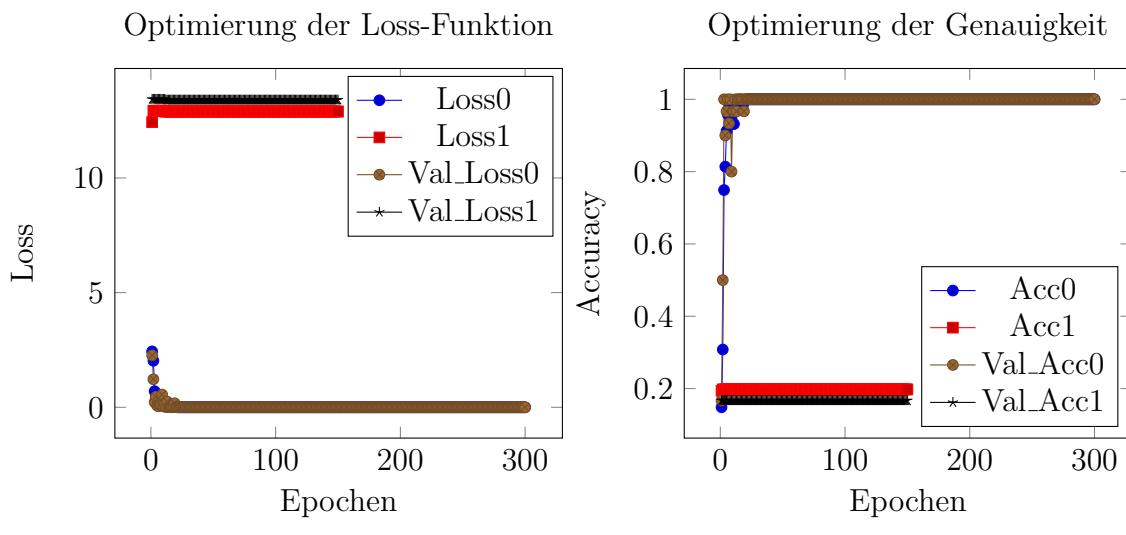


Die Entwickler der VGG-Architektur (Simonyan und Zisserman, 2015) stellten die These auf, dass Filter der Größe $n * n$ in einem tiefen Netzwerk am besten mit den Maßen $3 * 3$ funktionieren. So sollte in diesem und dem folgenden Netzwerk getestet werden, ob diese Annahme auch bei diesem Datensatz zutrifft und bessere Ergebnisse liefert. Anzumerken ist, dass kleinere Maße der Filter bei gleicher Anzahl an Filtern natürlich weniger trainierbare Parameter ergibt und somit Effizienter in der Berechnung sein sollte, jedoch vielleicht auch abstraktere Muster erkennen muss. Die 1-Block-Architektur mit drei CL dient hier als Beispiel für den Test: es ist die minimale Konfiguration eines tiefen NN und hat 192 Ergebnis-Matrizen der Filter.

Diese Architektur erreicht sehr schnell - bereits nach wenigen Epochen - ihr Endergebnis in der Optimierung, welches mit knapp 99% Genauigkeit bereits gut ist. Es hat durch Zufall die gleiche Genauigkeit der Architektur a_1 ; interessanter ist jedoch, dass die Ergebnisse in einer ähnlichen Größenordnung liegen, obwohl die Anzahl der trainierbaren Parameter deutlich kleiner ist.

Architektur c_1 : 5*5

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*64	4.864
Conv	64*64*64	102.464
Conv	64*64*64	102.464
Flatten	262.144	0
Dicht	10	2.621.450
		$\sum 2.831.242$



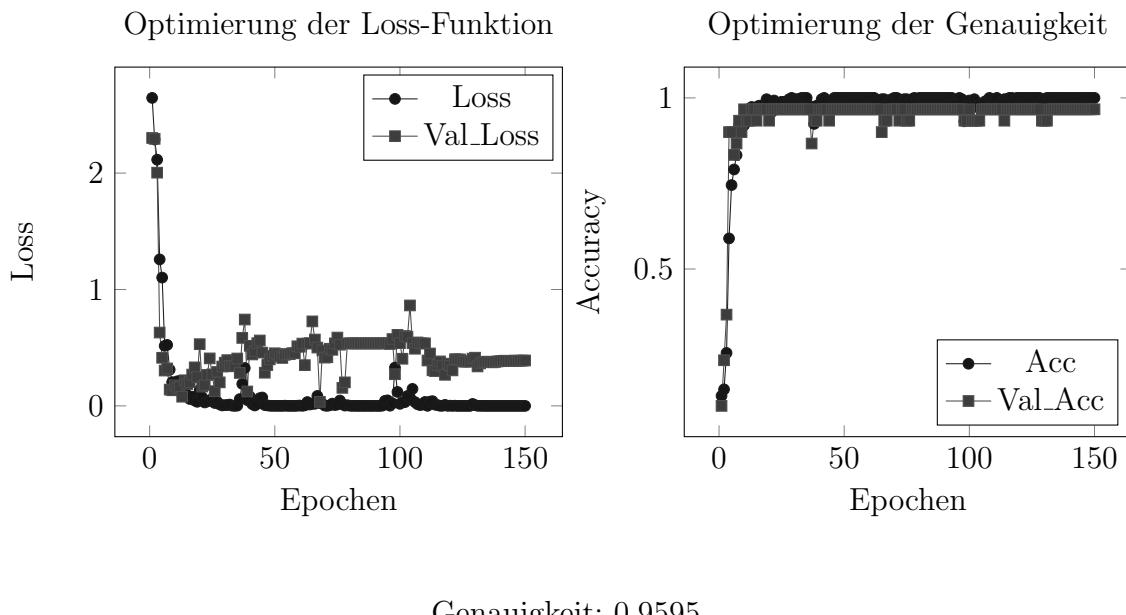
Genauigkeit: 0,9595

Hier also die gleiche Architektur, jedoch mit Filtern der Größe 5×5 . Entsprechend gibt es etwas mehr trainierbare Parameter, da die Anzahl der Filter gleich bleibt, die Dimensionen (und damit Parameter) jedoch vergrößert werden.

Auch hier noch ein mal ein Beispiel für das Erreichen eines lokalen Minimums in der Optimierung - Versuch Nummer zwei (Acc1) schafft es nicht die 20% Marke in der Genauigkeit zu erreichen. Bei Versuch 1 funktioniert die Optimierung richtig. Im Training wird ein sehr gutes Ergebnis erzielt, wie man am Graph von Acc0 sehen kann. Im Test wird jedoch trotz der erhöhten Anzahl an Parametern mit knapp 96% ein schlechteres Ergebnis erzielt. Es kann wohl sowohl aus dem Ziel der Genauigkeit, als auch aus dem Ziel der Effizienz gesagt werden, dass ein weiteres Vorgehen mit 3×3 Filtern sinnvoll ist.

Architektur c_2 : 5*5 DO

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*64	4.864
Conv	64*64*64	102.464
Conv	64*64*64	102.464
Dropout	64*64*64	0
Flatten	262.144	0
Dicht	10	2.621.450
		$\sum 2.831.242$

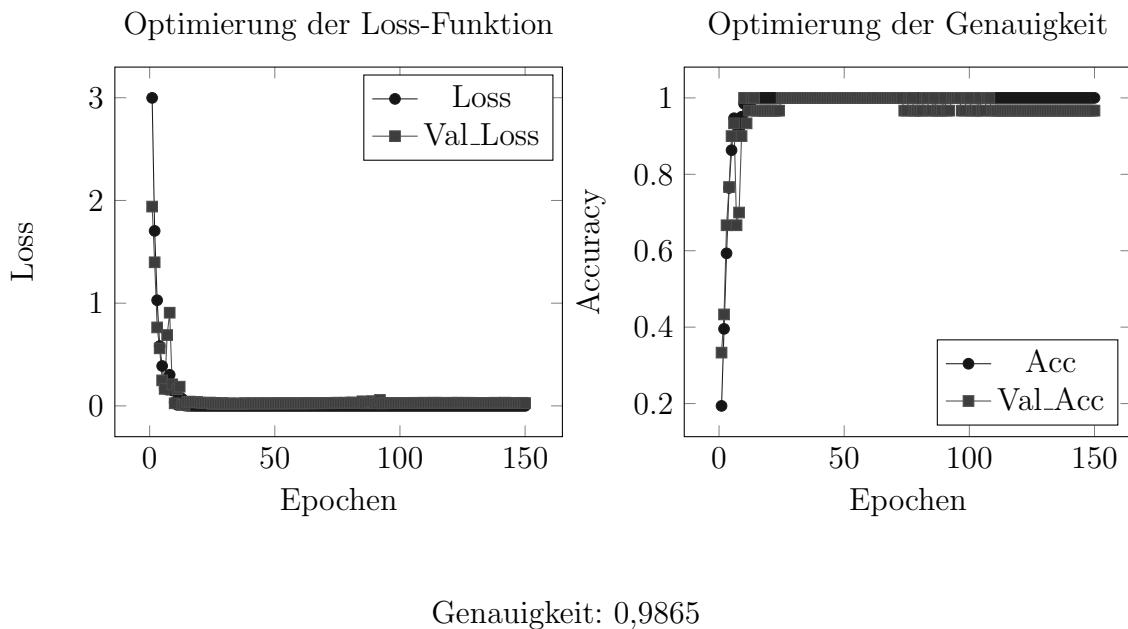


Der vermutete, teils bewiesene Wert von Dropout wurde ja bereits aufgezeigt. Um eine mögliche Verbesserung der Ergebnisse bei Einsatz dieser Methode im Training zu erreichen, wurde diese bei gleichbleibenden sonstigen Einstellungen im Vergleich zu der vorherigen Architektur getestet.

Das Ergebnis ist interessant. Die Genauigkeit bleibt genau gleich - möglicherweise wurde genau derselbe Satz an Parametern von dem Optimierungsalgorithmus gefunden. Am abgebildeten Graphen ist jedoch genau zu erkennen, wie Dropout im Training die Ergebnisse verzerrt und scheinbar schlechtere Ergebnisse erzielt. Eine Notwendigkeit die Dropout-Technik einzusetzen besteht also nicht, wenn der gleiche Satz Gewichte und Tendenzen gefunden werden kann. Möglicherweise würde er sogar von dem bestmöglichen Satz ablenken, oder das Training könnte an einer ungünstigen zeitlichen Position gestoppt werden.

Architektur c_3 : 3*3 Dicht

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*64	1.792
Conv	64*64*64	36.928
Conv	64*64*64	36.928
Flatten	262.144	0
Dicht	30	7.864.350
Dicht	10	310
		$\sum 7.940.308$

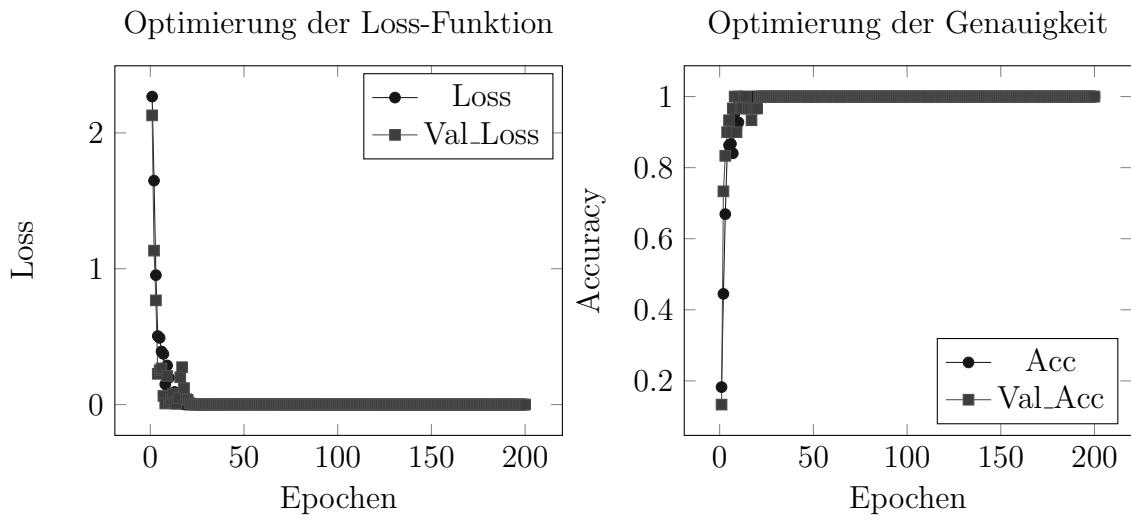


Ein letztes Experiment sollte an diesem minimalen Beispiel für ein tiefes NN noch durchgeführt werden: das Einfügen einer dichten Schicht nach dem einen Block von 3 CL. Natürlich erhöht diese Schicht die Anzahl trainierbarer Parameter stark (wie sichtbar nimmt diese Schicht 7.864.350 der 7.940.308 Parameter ein).

Auch hier ist eine vergleichsweise schnelle Optimierung mit einem guten Ergebnis möglich. Auf dem Test-Datensatz wird eine Genauigkeit von fast 99% erreicht - wohl durch Zufall dieselbe Genauigkeit wie bei der reinen CL-Architektur c_0 . Auch hier werden wohl die gleichen Filter gefunden worden sein, wobei die dichte Schicht scheinbar weder einen Vorteil noch einen Nachteil bringt. Dann kann diese obsoleten Schicht aus Effizienzgründen in dem Training in dieser Architektur wohl auch weggelassen werden.

Architektur d_0 : 2-Block

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*64	1.792
Conv	64*64*64	36.928
Conv	64*64*64	36.928
MaxPool	32*32*64	0
Conv	32*32*64	36.928
Conv	32*32*64	36.928
Conv	32*32*64	36.928
MaxPool	16*16*64	0
Flatten	16.384	0
Dicht	30	491.550
Dicht	10	310
		$\sum 678.292$



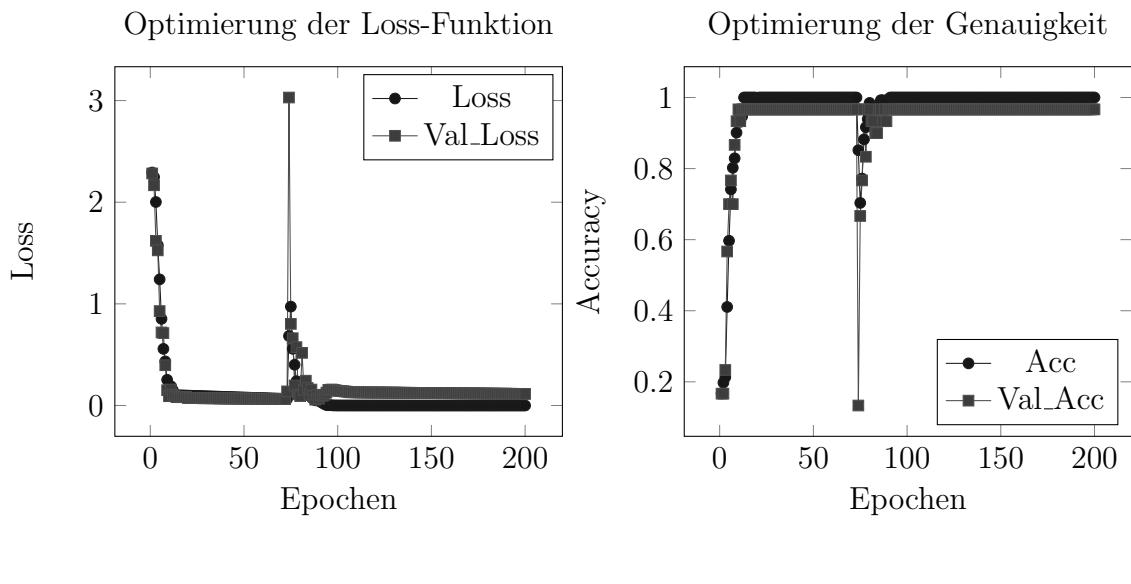
Genauigkeit: 0,9730

Da komplexe Architekturen tiefer CNN mit mehreren CL-Blöcken in der Literatur meistens vielversprechend dargestellt werden und auch in der ILSVRC wie gezeigt erfolgversprechend eingesetzt werden, sollte auch hier noch ein mal eine Synthese der bisherigen Beobachtungen getestet werden. Die Maße der Bilder sind mit $64 * 64$ Pixel genau genug für die Erkennung von Formen, enthalten aber vermutlich nicht zu viele Details, die bei der Klassifizierung ablenken könnten. Viele aufeinanderfolgende Filter in mehreren Schichten und zwischengeschobene PL sollen das nötige Abstraktionsvermögen bringen, ohne die Effizienz im Training und die Komplexität in der Speicherung und Berechnung mit zu vielen Parametern unnötig zu erhöhen. Bei der letzten dichten Schicht wurde ein Kompromiss gewählt; die Schicht dient als Puffer zwischen der 16384-dimensionalen Flatten Schicht und der 10-dimensionalen Ausgabe Schicht.

Die Optimierung erfolgt schnell und ohne besondere Ereignisse. Eine Genauigkeit von über 97% wird erreicht - gut, aber nicht annähernd perfekt.

Architektur d_1 : 2-Block DO

Schicht	Dim. Ausgabe	Parameter
Conv	64*64*64	1.792
Conv	64*64*64	36.928
Conv	64*64*64	36.928
MaxPool	32*32*64	0
Conv	32*32*64	36.928
Conv	32*32*64	36.928
Conv	32*32*64	36.928
MaxPool	16*16*64	0
Dropout	16*16*64	0
Flatten	16.384	0
Dicht	30	491.550
Dicht	10	310
		$\sum 678.292$



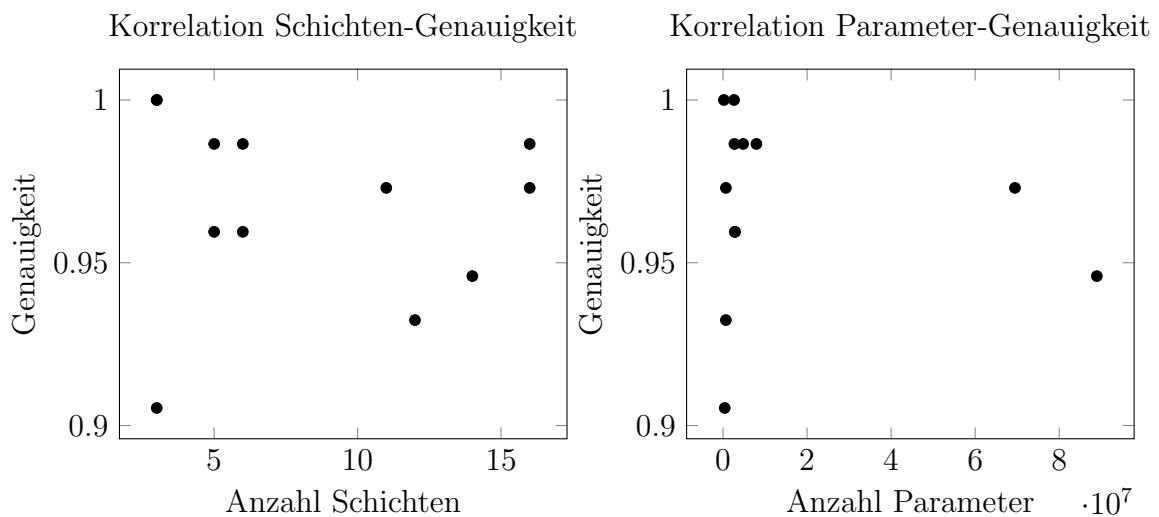
Die gleiche Architektur nur mit Dropout Methodik zwischen dem letzten PL und der Flatten Schicht.

Die Optimierung leidet unter der eingefügten Schicht und erreicht eine Genauigkeit von unter 94%. Auch hier ist kein Vorteil aus der möglichen Generalisierungsmethode zu ziehen. Außerdem fällt bei Betrachtung der Graphen auf, dass die Genauigkeit circa bei Epoche 75 kurzzeitig eingebrochen ist. Dieser Fehler konnte von dem Algorithmus jedoch in wenigen Epochen wieder korrigiert werden und ermöglichte sogar eine leichte sprunghafte Verbesserung der Genauigkeit.

6.4.2 Diskussion der Architektur-Ergebnisse

Bezeichnung	#Schichten	# train. Parameter	Genauigkeit
a_0 VGG	14	88.962.178	0,9459
a_1 Nguyen64	16	4.789.386	0,9865
a_2 Nguyen128	16	69.478.666	0,9730
b_0 5f	3	204.950	1,0
b_1 10f	3	409.890	0,9054
b_2 64f	3	2.623.242	1,0
c_0 3*3	5	2.697.098	0,9865
c_1 5*5	5	2.831.242	0,9595
c_2 5*5 DO	6	2.831.242	0,9595
c_3 3*3 Dicht	6	7.940.308	0,9865
d_0 2-Block	11	678.292	0,9730
d_1 2-Block DO	12	678.292	0,9324

Abbildung 6.12: Überblick über die implementierten Netze, 'train.' für 'trainierbare'



Während es in vielen Anwendungen von CNN, wie zum Beispiel der ILSVRC, einen Trend zu mehr Schichten und teilweise auch zu mehr Parametern gibt, zeigt der Ansatz der Komplexitätserhöhung bei Betrachtung der Graphen absolut keinen Vorteil in der Implementierung zur Erkennung und Klassifizierung von Blattabbildungen. Das CNN ist klassischerweise als tiefes NN gedacht - deshalb wurden vor allem auch Netzwerke wie a , das VGG-inspirierte und die Nguyen-Netzwerke, getestet, doch zeigen gerade die simplen Konfigurationen wie b , dass weniger hier mehr bringt. Die besten Ergebnisse konnten im Test von den Netzen mit drei Schichten erzielt werden, was zwar bereits als tiefes NN gilt, jedoch im Vergleich zu anderen Netzwerken noch relativ flach ist. Auch bei den Parametern ist eine solche Beobachtung festzustellen - die Genauigkeit fällt mit zunehmenden trainierbaren Parametern. Kurz gesagt passt die Konfiguration b_0 nicht in die Erwartungen und ist

in der Performance eine Überraschung. Die reduzierte Anzahl an Schichten und Parametern führt zu einer höheren Effizienz und einer höheren Genauigkeit. Weiter ist zu sehen, dass kleinere Filterdimensionen (getestet mit den Konfigurationen c) hier zu besseren Ergebnissen führen - dies stützt das Postulat von Simonyan und Zisserman (2015), die jedoch darauf beruht, dass sehr viele Schichten hintereinander mit kleinen Filtergrößen folgen. Scheinbar gilt diese Annahme in diesem Fall auch in einem reduzierten Netzwerkumfang.

Interessant ist außerdem, dass - je nach Anzahl trainierbarer Parameter - teilweise schon nach wenigen Epochen, maximal aber bis 100 Epochen eine Stabilisierung der Optimierung einsetzt. Bei den Architekturen b_2 und c_1 wurden außerdem Testläufe in die abgebildeten Graphen inkludiert, bei denen bereits nach sehr wenigen Epochen ein lokales Minimum in der Parameterebene gefunden wurde, aus dem der Trainingsalgorithmus in vielen darauffolgenden Epochen keinen Ausweg gefunden hat. Das war zu erwarten und zeigt ein weiteres Defizit dieser Methode. In einem hier vorliegenden Umfang mag das kein großes Problem sein, da einige weitere Testläufe genügen um zu zeigen, dass es kein Fehler der Architektur ist, doch kann diese Unsicherheit zu großen Fehlern führen, wenn die Parameter sehr komplexer Architekturen trainiert werden und das Training wie bei VGGNet zwei bis drei Wochen dauert.

Final lässt sich hier noch anmerken, dass generell ein erfolgreiches Training mit allen Konfigurationen stattgefunden hat. Aufgrund der 10 möglichen Kategorien, würde sich nach dem Gesetz der großen Zahlen die Trefferquote bei zufälligem Wählen bei 10% stabilisieren - hier wurde mit allen erfolgreich-trainierten Netzwerken eine Genauigkeit von über 90% erzielt. Ein Ansatz die Anzahl der zu trainierenden Parameter zu reduzieren ist die Verwendung von Eingabebildern der Tiefe 1 - also Bildern bestehend aus reinen Luminanzwerten ohne Farbinformationen. Im folgenden Kapitel werden dieser Ansatz und ein weiterer Versuch bezüglich der Generalisierung der trainierten Architekturen untersucht, wenn es darum geht das Training zu variieren.

Kapitel 7

Variation im Training

7.1 Verwendung von SW-Bildern

Mit dem vorliegenden Datensatz und den verwendeten Algorithmen und Architekturen konnte also ein erfolgreiches Klassifikationssystem implementiert werden. Es wurde auch die Auflösung variiert und entsprechend mit der Detailgenauigkeit experimentiert, was bei den Netzwerken a_1 und a_2 interessanterweise bei einer Verdopplung der Maße und einer entsprechenden Vervierfachung der Pixelanzahl keine Verbesserung, sondern eine Verschlechterung der Genauigkeit brachte. Eine verbesserte Bildqualität sollte jedoch eigentlich zu besseren Ergebnissen führen - die vorliegende Beobachtung liegt wohl an der erhöhten Komplexität, die durch die erhöhte Anzahl Parameter und Variablen im System vorhanden sind.

Als weiterführendes Experiment könnte eine Umwandlung der Bilder des Datensatzes von Farbe in schwarz-weiß interessant sein. Hierbei werden, wie vorher schon angeschnitten als es darum ging die Faltung von schwarz-weiß zu Farbkanälen zu erweitern, die vorhandenen drei Farbkanäle rot, grün und blau reduziert auf einen einzigen Luminanzkanal auf einer Skala von schwarz zu weiß reduziert. Formal wird hierbei die Tiefe 3 auf die Tiefe 1 reduziert - implementiert werden kann das unterschiedlich. Die einfachste Möglichkeit ist für jeden vorhandenen Pixel das arithmetische Mittel der zugehörigen Aktivierungen der Farbkanäle zu berechnen, also Aktivierung a von Pixel l, b Nach Helland (2011) unterscheidet sich das berechnete

$$a(l, b) = (a_r(l, b) + a_g(l, b) + a_b(l, b)) / 3 \quad (7.1)$$

Abbildung 7.1: Die Werte der drei Matrizen an gleichen Positionen werden addiert und anschließend durch die Anzahl der Aktivierungen geteilt - der Mittelwert der Aktivierungen an der jeweiligen Position der Matrizen wird für die neue Luminanzmatrix gebildet. Alle drei Farbaktivierungen werden berücksichtigt.

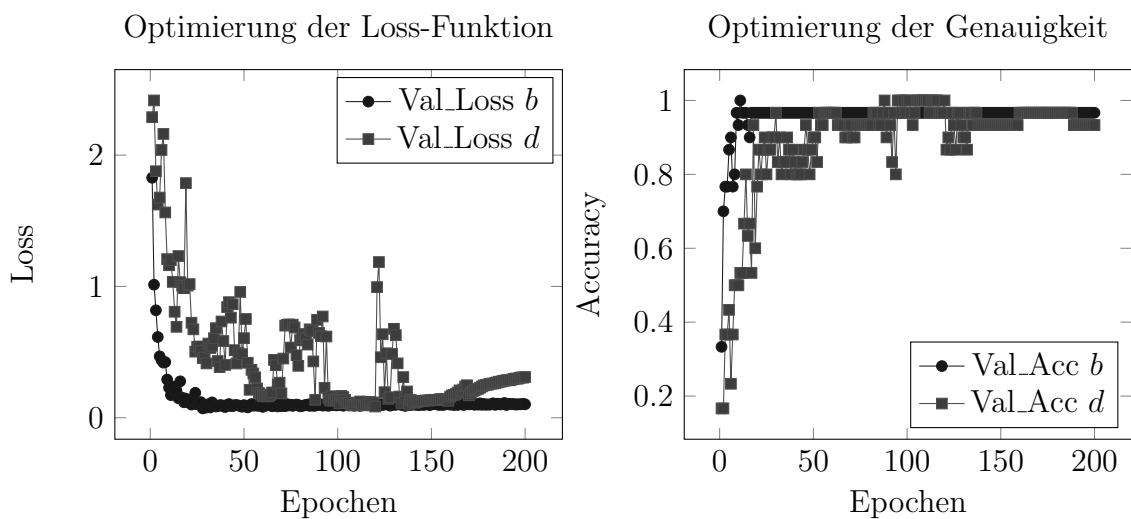
Ergebnis in den Graustufen jedoch stark von der Weise, wie Menschen Luminanzdifferenzen betrachten. Es gibt also weitere Funktionen, die die Graustufen besser berechnen für die Betrachtung mit dem menschlichen Auge. Eine weitere solche Funktion ist deshalb das arithmetische Mittel der größten und kleinsten Aktivierung des Pixels

Getestet wurden nun die Architekturen b_0 und d_0 mit demselben Blätter Daten-

$$a(l, b) = (\max(a_r(l, b), a_g(l, b), a_b(l, b)) + \min(a_r(l, b), a_g(l, b), a_b(l, b)))/2 \quad (7.2)$$

Abbildung 7.2: An der gleichen Position werden der größte und der kleinste Wert der drei vorliegenden Matrizen addiert und dividiert. Hierbei unterscheiden sich die Quellen zwangsläufig, da ein Element einer Matrix nicht zugleich den kleinsten und größten Wert der drei Matrizen besitzen kann. Die Aktivierung, die zwischen den beiden anderen liegt, wird nicht berücksichtigt in der Berechnung.

satz, bei dem jedoch vor dem Training jedes Bild in Graustufen umgewandelt wird mit der Funktion $a(l, b) = 0.2125 * a_r(l, b) + 0.7154 * a_g(l, b) + 0.0721 * a_b(l, b)$, einer genormten Gewichtung, die auf das menschliche Auge besonders natürlich wirken soll (Poynton, 1997).



b_0^{bw} : Die Werte der Genauigkeit sind ab Epoche 21 stabil mit einer Test Genauigkeit (Val_{Acc}) von 0,9595. Dieser Wert wird auch bei mehrfacher Wiederholung nicht übertroffen. Es ist also in der Tat eine Verschlechterung der Performance, die von 100% auf die reduzierten 95,95% erheblich ist, jedoch trotzdem ein ordentliches Ergebnis und eine Lösung des Optimierungsproblems liefert.

d_0^{bw} : Diese Architektur verhält sich im Training ähnlich wie im Training mit Farbbildern, nämlich zunächst etwas volatil. Außerdem braucht die Stabilisierung erheblich länger als die bei b_0^{bw} - offensichtlich bei dem komplexeren Training. Die Test-Genauigkeit beläuft sich hier nach 200 Epochen auf 0,9054 - kein wünschenswertes Ergebnis und die erneute Bestätigung, dass diese Architektur schlechter funktioniert hier als die simplere und dass die Verarbeitung von Bildern in Graustufen im Vergleich zu der Verarbeitung von den gleichen Bildern in Farbe die Genauigkeit betreffend schlechter abschneidet.

7.2 Der Wert guter Daten

Entscheidend für ein erfolgreiches Maschinelles Lernen mit künstlichen Neuronalen Netzwerken ist der benutzte Datensatz. Bei CNN ist es deshalb generell besonders wichtig das Training mit einer möglichst umfangreichen Bilderbibliothek durchzuführen, die auch möglichst in der ein oder anderen Weise - beispielsweise in Teilen über mehrere Bilder hinweg - alle fotografischen Umstände abdeckt, die später von dem trainierten Netzwerk klassifiziert werden sollen. Abgesehen von Mustern der abgebildeten Objekte sind hier unter anderem eben auch die Gegebenheiten der Beleuchtungen wichtig, die von einem CNN nicht künstlich ausgeglichen werden können, wie das menschliche Gehirn das kann. Deshalb sind populäre Datensätze wie CIFAR und ImageNet eben auch möglichst umfangreich (CIFAR: 10 Kategorien, 60000 Bilder; ImageNet: >20.000 Kategorien, >14 Millionen Bilder), decken möglichst viele unterschiedliche Fälle ab (Platzierung der zu klassifizierenden Objekte in verschiedenen Situationen und zum Beispiel mit anderen störenden Objekten im Hintergrund) und möglichst eindeutig in der Klassifizierung des Trainingssets (CIFAR hat beispielsweise keine Überlappung der Subsätze PKW und LKW - die entsprechenden Bilder sind immer eindeutig und auch kontinuierlich entweder der einen oder der anderen Kategorie zugeordnet). Aus diesem Grund wird immer häufiger 'Transfer-Learning' (Transferlernen) im Training von CNN eingesetzt.

Zunächst wird für Transferlernen die Architektur für einen bestimmten Anwendungsfall wie der Klassifizierung von Bäumen entworfen. Die Parameter dieser Architektur werden jetzt jedoch nicht direkt mit dem Datensatz der Baum-Bilder trainiert, sondern zuerst auf einem großen, umfangreichen und möglicherweise etablierterfolgversprechenden Datensatz trainiert. Erst im darauffolgenden Schritt werden diese vortrainierten Parameter weiter angepasst an die konkreten Anforderungen des Baum-Datensatzes. Das wird in der Erwartung durchgeführt, sinnvolle und eher allgemeine Filter zu erstellen, die ein besonders breites Feld an Varietäten der Baum-Bilder später erkennen sollen. Diese allgemeinen Filter werden dann weiter präzisiert und auf den vorliegenden Datensatz zugeschnitten. Von der VGG-16 Architektur wird auf der Plattform Kaggle so eine Datei kostenlos angeboten, die die trainierten Parameter w und b der Architektur zur Verwendung mit dem ImageNet Datensatz beinhaltet.

Betrachtet man das Bild eines kranken und verfärbten Ahornblattes, so hat man unter den möglichen zehn Kategorien des Datensatzes als Mensch wohl kein großes Problem, dieses Blatt in die Kategorie der weiteren, jedoch gesunden, Ahornblätter einzuführen. Schaut man jedoch auf den Graphen, so sieht man, dass das Netzwerk der b_2 Architektur große Schwierigkeiten hat, dieses korrekt einzuführen. Das Netzwerk mit seinen 64 Filtern hat eine 100% Genauigkeit erzielt, sowohl im Training als auch im Test, doch scheint dieses Bild durch die Andersartigkeit zu viel Störsignale und störende Merkmale zu besitzen. Die Farben passen nicht und ergeben sogar innerhalb des Blattes eigene Muster und die Kanten sind teilweise unklar. Die Antwort des Netzwerks ist klar: Das Bild muss zur Kategorie 8 gehören. Doch im Sachzusammenhang stimmt das nicht! Das Blatt wäre eigentlich ein Bild der Kategorie 2, doch diese Wahrscheinlichkeit liegt laut Netzwerk bei 0,113%. Diese Antwort ist jedoch gar nicht so abwegig, wenn man den Trainingsdatensatz kennt,

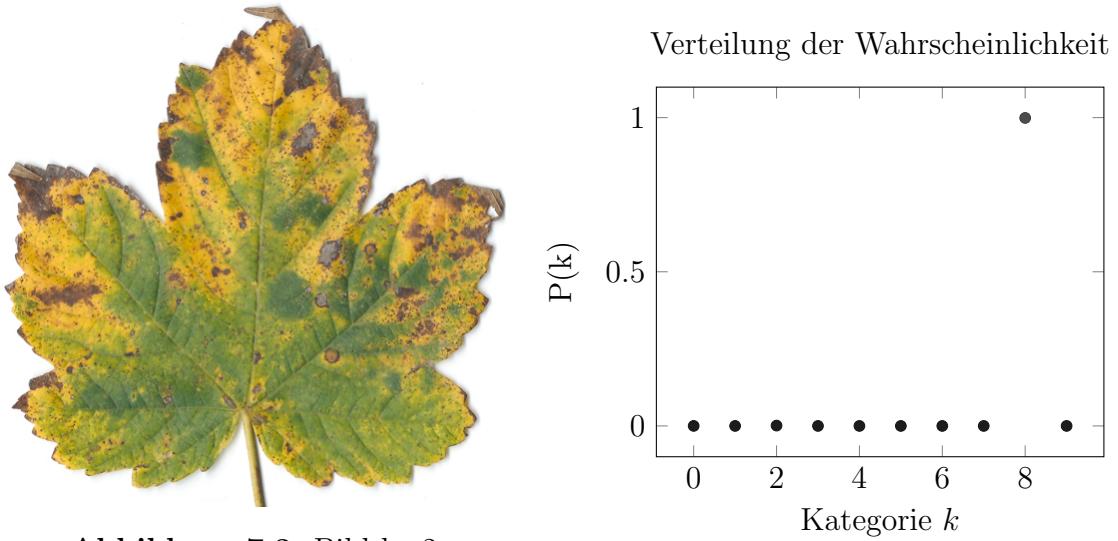


Abbildung 7.3: Bild $k=2$

denn es gibt nur zwei Kategorien, die Bilder von Blättern mit mindestens teilweiser gelber Farbe haben. Dass dieser Gelbstich des kranken Blatts als eindeutiges Zuordnungskriterium verwendet wird, ist möglich. Um auch Bilder dieser Art von diesem Algorithmus klassifizieren zu können, hätte man bereits im Training einen Datensatz haben müssen, der Instanzen solcher Art und mit solchen Mängeln beinhaltet. Hier ist eine klare Limitation dieser Methode zu erkennen.

Kapitel 8

Zusammenfassung der Ergebnisse

Von der Idee ausgehend, sich mit der Funktionsweise faltender NN strukturiert und von Grundlagen ausgehend differenziert zu beschäftigen, wurden zunächst die theoretischen Grundlagen erarbeitet und anschließend alle nötigen Arbeitsschritte hin zu einer praktischen Implementierung eines CNN in einem realen Szenario dokumentiert und kritisch betrachtet. So wurden zunächst frühe Ansätze in dem Bereich und bedeutende historische Konzepte wie das Perzepron dargestellt, Analogien zur Neurobiologie aufgezeigt und die Bedeutung dieser in der heutigen Entwicklung von NN klargestellt. Aufbauend auf die Grundstruktur NN wurde das Modell von CNN entwickelt und die Vorteile für die Verarbeitung von Bild-Datensätzen hervorgehoben. Daraufhin wurde der Aufbau von Fotos im historischen und informatischen Kontext aufgezeigt und ein eigener geordneter Datensatz mit Bedacht entwickelt, der aus zehn Kategorien mit durchschnittlich je 37 Fotos von Blättern besteht. Die theoretische Auseinandersetzung mit Möglichkeiten in der Architekturentwicklung und den Optimierungsgrundlagen für die Parameterfindung ermöglichte eine genaue Zielentwicklung für die praktische Entwicklung eines CNN zur selbsterlernten Klassifizierung von Bildern dieses Datensatzes, die im Folgenden nach dem Erörtern und Vergleichen von bedeutenden Architekturen der letzten Jahrzehnte und unter Betrachtung von realen Einsatzszenarien in verschiedenen Konfigurationen unter Variation der möglichen Bauarten und Parametern vollzogen wurde. Die direkten Ergebnisse der Optimierung und Genauigkeit auf einem zuvor unverarbeiteten Test-Datensatz ließen teils überraschende Schlüsse zu, wie der, dass weniger Parameter und simplere Architekturen mit kleinen Filtern bei der Implementierung in Szenarien wie diesen eine erheblich bessere und effizientere Performance liefern. Schließlich war es möglich, mit einer vergleichsweise simplen Architektur, bestehend aus drei Schichten, davon zwei trainierbaren (eine faltende Schicht und eine dichte Schicht), und knapp über 200.000 trainierbaren Parametern in einer relativ kurzen Trainingsphase ein perfektes Ergebnis von 100% Genauigkeit auf dem Trainings- und dem Testdatensatz zu erzielen.

Die Feststellung, dass kleiner dimensionierte CNN für relativ simple Einsätze wie diesen hier gut geeignet sind, lässt sich nicht für komplexe Anwendungen allgemeinern, aber stellt möglicherweise einen Gegenpol zu den immer komplexer und tiefer werdenden Netzwerken dar, die aktuell entwickelt werden. So scheint es möglich, dass CNN auch beispielsweise in der Automatisierung in der Industrie in Fertigungsprozessen eingesetzt werden könnten, um eine Überwachung und Kontrol-

le effizient und kostengünstig durchzuführen.

Der Algorithmus kam an seine Grenzen als Bilder getestet wurden, die semantisch deutlich anders sind und nicht in die Muster der bekannten Bildkategorien passen. Sie korrekt zu klassifizieren war nicht möglich, was eine Einschränkung in realen Einsatzszenarien wäre, da dadurch für die sinnvolle Nutzung ein sehr großer Aufwand im Vorhinein in die Datensammlung und Erstellung der Datensätze gesteckt werden muss.¹ Ein möglicher Lösungsansatz für Probleme wie diese könnte 'Data Augmentation' (Daten Vergrößerung) darstellen. Diese Methode lässt zu, den gesammelten Datensatz um ein Vielfaches zu vergrößern, indem Manipulationen durchgeführt werden wie Spiegelungen, Vergrößerungen, Verkleinerungen oder Farbänderungen der abgebildeten Objekte. Diese Methode wird für das Training vieler Netze bereits genutzt und könnte hier weiterführend auf Erfolgssteigerung getestet werden. Außerdem könnte weiterführend die erwähnte Methode des 'Transfer Learning' ausprobiert werden, bei dem das Netzwerk auf großen Datensätzen umfangreich trainiert wird und dann im nächsten Schritt an die Anforderungen des speziellen Datensatzes angepasst wird. Weiter könnte eine Visualisierung der Filter der hier trainierten Netzwerke von Interesse sein, wie sie beispielsweise von Krizhevsky et al. (2012) grafisch dargestellt wurden. So könnten möglicherweise Schlüsse auf die genaue Arbeitsweise der hier verwendeten Filter gezogen werden, auch wenn sie vermutlich für den menschlichen Betrachter unbrauchbar scheinen würden.

An dieses Dokument angehängt gehört eine Sammlung eigener Programmierungsarbeiten und berechnete Modelle einschließlich zugehöriger erlernter Gewichte und Tendenzen.

Persönlich war dieses Projekt eines, das mir über die komplette Dauer hinweg viel Freude bereitet hat, mich gefesselt hat und mich sowohl akademisch als auch persönlich immer wieder aufs neue herausgefordert hat. Akademisch aufgrund des großen Rechercheaufwands, der langen Einarbeitungszeit, den komplizierten Veröffentlichungen, Darstellungen und Erklärungen, der Entwicklung eigener Ansätze und Thesen und der kritischen und zielführenden Auseinandersetzung mit meinen Fehlern, Beobachtungen und Feststellungen. Persönlich, da diese zeitaufwändige und konzentrationsbedürftige Beschäftigung neben den anderen Schularbeiten während der Abitur-Phase viel Motivation, Eigeninitiative und Verzicht gefordert hat. Die Herbst- und Winter-'Ferien' wurden so eher Zeiten der Vertiefung und parallel zu den Klausurenphasen bedurfte es einer strukturierten Zeiteinteilung.

¹Dieses Problem der Beschaffung hochqualitativer Daten ist eines der größten Probleme der Industrie und Forschung, die auf maschinellem Lernen und insbesondere Bilderkennung aufbaut.

Literaturverzeichnis

- Meyers Universal Lexikon.* VEB Bibliographisches Institut, 1982.
- D. Amodei und D. Hernandez. Ai and compute, May 2018. URL <https://blog.openai.com/ai-and-compute/>.
- A. Armbruster und Y. LeCun. Ohne Künstliche Intelligenz funktioniert Facebook nicht. *Frankfurter Allgemeine Zeitung*, page 19, Nov. 2018.
- J. Aron. Forget the turing test – there are better ways of judging ai, Sept. 2015. URL <https://www.newscientist.com/article/dn28206-forget-the-turing-test-there-are-better-ways-of-judging-ai/>.
- N. Bostrom. *Superintelligence*. Oxford University Press, 2014.
- F. Chollet et al. Keras. <https://keras.io>, 2015.
- S. Das. Cnn architectures: Lenet, alexnet, vgg, googlenet, resnet and more, Nov. 2017. URL <https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5>.
- N. Finkernagel. 7.9: Tensorflow.js color classifier: Softmax and cross entropy, July 2018. URL <https://www.youtube.com/watch?v=r0QvaEra0og>.
- D. Gershgorin. The quartz guide to artificial intelligence: What is it, why is it important, and should we be afraid?, Sept. 2017. URL <https://qz.com/1046350/the-quartz-guide-to-artificial-intelligence-what-is-it-why-is-it-important-and-should-we-be-afraid/>.
- K. He, X. Zhang, S. Ren, und J. Sun. Deep residual learning for image recognition. Dec. 2015.
- T. Helland. Seven grayscale conversion algorithms (with pseudocode and vb6 source code), Oct. 2011. URL <http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/>.
- ImageNet. Summary and statistics, Apr. 2010. URL <http://image-net.org/about-stats>.
- J. Kafunah. Backpropagation in convolutional neural networks, May 2018. URL <https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/>.
- A. Karpathy. Cs231n: Convolutional neural networks for visual recognition, 2018. URL <http://cs231n.github.io/>.

- A. Krizhevsky, I. Sutskever, und G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, Jan. 2012.
- S. Lau. Learning rate schedules and adaptive learning rate methods for deep learning, July 2017. URL <https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1>.
- A. Lazorenko. Tensorflow performance test: Cpu vs gpu, Dec. 2017. URL <https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fc39170c>.
- Y. LeCun, L. Bottou, Y. Bengio, und P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, Nov. 1998a.
- Y. LeCun, L. Bottou, G. B. Orr, und K.-R. Müller. Efficient backprop. 1998b.
- J. Markoff. Seeking a better way to find web images, Nov. 2012. URL <https://www.nytimes.com/2012/11/20/science/for-web-images-creating-new-technology-to-seek-and-find.html>.
- C. N. Nguyen und O. Zeigermann. *Machine Learning – kurz & gut*. O'REILLY, 2018.
- C. Poynton. Frequently asked questions about color, Mar. 1997. URL <http://poynton.ca/PDFs/ColorFAQ.pdf>.
- F. Rosenblatt. The Perceptron—a perceiving and recognizing automaton. *Cornell Aeronautical Laboratory*, 85-460-1, 1957.
- O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, und L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- N. Sharkey. Artificial intelligence is a dream we shouldn't be having, Sept. 2009. URL <https://www.computerweekly.com/feature/AI-is-a-dream-we-shouldnt-be-having>.
- K. Simonyan und A. Zisserman. Very deep convolutional networks for large-scale image recognition. Apr. 2015.
- R. V. Singh. Imagenet winning cnn architectures – a review, 2016. URL http://rajatvikramsingh.github.io/media/DeepLearning_ImageNetWinners.pdf.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, und R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- J. Stanley und E. Bak. *Neuronale Netze*. Systhema Verlag GmbH, 1991.
- K. Swingler. Itnp4b lecture notes 2012: Delta rule, 2012. URL <http://www.cs.surrey.ac.uk/courses/ITNP4B/lectures/kms/3-DeltaRule.pdf>.

- C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, und A. Rabinovich. Going deeper with convolutions. Sept. 2014.
- P. Warden. How do cnns deal with position differences?, Oct. 2017. URL <https://petewarden.com/2017/10/29/how-do-cnns-deal-with-position-differences/>.
- M. D. Zeiler und R. Fergus. Visualizing and understanding convolutional networks. Nov. 2013.

Arbeitsbuch

Datum	Tätigkeit	Stunden
6.-24.8.18	Quellensammlung und Einlesen	10
25.8.18	Datenerfassung Holzhausenpark	3
27.8.18	Datenerfassung Grüneburgpark	3
28.8.18	Outline, Einleitung schreiben	2
1.9.18	Datenerfassung, Quality Control	2
2.9.18	Ansatz Blatterkennung schreiben	2
9.9.18	Datensammlung Stadtwald, da neuer Ansatz	3
10.9.18	Scannen der gesammelten Blätter	2
12.9.18	Ansatz Blackbox, Historie lesen	2
22.9.18	Studieren Touring-These, Recherche Implementierung mit Tensorflow und Keras in Python	4
23.9.18	Sammeln und Scannen von weiteren Blättern	3
24.9.18	Installieren von Linux Distribution und Treibern	6
28.9.18	Treiberinkompatibilitäten lösen	3
29.9.18	Übergehen auf Windows, Neuinstallation, TF einrichten	10
8.10.18	Installieren und Einarbeiten in Keras, Implementierung Beispielprojekt	10
9.10.18	Entwickeln von Version 0 der Klassifizierung, Netzwerk Training & Testen	6
10.10.18	Einzelne Bilder testen, Coding	5
11.10.18	Coding, Code & Architektur dynamisch gestalten, Logging	6
12.10.18	Coding, Setup Dropout Test, Logging	10
16.10.18	Coding, Dropout Test, schreiben	3
21.10.18	Datensammlung, Blattstudien	2
22.10.18	Mathematik: Faltung, diskrete Faltung	3
23.10.18	Diskrete Faltung schreiben, Visualisierung	3
26.10.18	Datenverarbeitung	6
27.10.18	Datenordnung, Anpassung des Lernalgorithmus an neue Daten	5
28.10.18	Neuprogrammieren der Einzelblatt-Klassifizierung, lesen Back-propagation Theorie	5
30.10.18	Architektur Studien, ImageNet Recherche, Architekturen entwerfen, Implementierung	8
31.10.18	Auswertung der Architekturen, Einarbeiten in und Installieren von LaTeX	8
1.11.18	Gradientenabstiegsverfahren Theorie lesen	1
5.11.18	Erstellung vorläufiger Bibliografie und Sortierung der Dokumente	2
6.11.18	Vorbereitung auf das Gespräch mit Herrn Jäger	1
7.11.18	Gespräch mit Herrn Jäger	1

19.11.18	Netzwerktheorie schreiben	2
20.11.18	Herangehensweise schreiben	2
11.12.18	Digitale Fotos schreiben, Datensatz schreiben	5
12.12.18	Architekturtheorie schreiben	5
31.12.18	Historische Durchbrüche & eigene Architekturen schreiben	2
1.1.19	Verschiedene bahnbrechende Architekturen lesen und schreiben	5
2.1.19	Eigene Architekturen mit Graphen und Tabellen ergänzen	10
4.1.19	Eigene Architekturen schreiben & überarbeiten	3
5.1.19	Eigene Architekturen im Vergleich schreiben	2
9.1.19	Motivation schreiben	2
10.1.19	Diskussion der Architektur-Ergebnisse schreiben	2
11.1.19	KI & ML recherchieren & schreiben	3
12.1.19	SW-Fotos programmieren, testen, schreiben	3
22.1.19	NN Theorie Outline, Geschichte Rosenblatt (Perzepron) Recherche & schreiben	2
23.1.19	NN Theorie schreiben, Überarbeitung Code LaTeX	2
24.1.19	Wert guter Daten testen, Daten visualisieren, schreiben	2
26.1.19	Grafiken, Funktionen einfügen, Erwartung und Zusammenfassung schreiben, generelle Überarbeitung	8
27.1.19	Theorie der Optimierung Recherche, schreiben, generelle Überarbeitung	9
28.1.19	Theorie Optimierung nachvollziehen & Überarbeiten, generelle Überarbeitung	1
29.1.19	Arbeitsbuch schreiben, LaTeX Code Überarbeitung	2
9.2.19	Überarbeitung Formulierungen	2
15.2.19	Gespräch mit Herrn Jäger	1
16.2.19	Überarbeitung nach Anregungen vom 15.2.19	3
18.2.19	Überarbeitung der Grafiken und Formulierungen	3
20.2.19	Überarbeitung der Mathematik, Einfügen von Fußnoten	4
21.2.19	Überarbeitung	4
24.2.19	Formatierung	2
25.2.19	Überarbeitung	2
1.3.19	Informieren über Druckmöglichkeiten	2
14.3.19	Python Code überarbeiten	2
15.3.19	Python Code überarbeiten	3
16.3.19	Zusammenstellung des digitalen Anhangs	2
17.3.19	Finales überarbeiten und formatieren der Arbeit	7
18.3.19	Zusammensetzung & kopieren des Anhangs, Druck	4
\sum Stunden		253

In der Kumulation sind die unzählbaren Minuten und Stunden des Kontemplierens über das Recherchierte und die Ergebnisse natürlich nicht berücksichtigt. Der zeitliche Aufwand hätte durch eine konkretere Struktur und Planung mit Sicherheit etwas verringert werden können, doch hätte diese den Aktionsspielraum verringert, der nötig war, um Beobachtungen und Feststellungen in dem Projekt zu verarbeiten und neugewonnene Erkenntnisse in die Entwicklung einfließen zu lassen.

Skripte im Anhang

```
/fglitzner_leaf_cnn
└── /backup           Protokollierung
    ├── /img_pixel_grafik :   Die Daten der Abbildungen 5.2-5.4
    │   └── ...
    └── settings.doc_2.txt :  Protokoll für die Trainingsprozesse der einzelnen Modelle.

└── /models :          Die trainierten Modelle der entworfenen Architekturen.
    └── ...

└── /scan_clean :      Der gesamte verwendete Datensatz (Bildmaße 512 * 512 Pixel).
    └── ...

└── /training_scripts : Skripte zum trainieren von Modellen der Architekturen.
    └── ...

└── leaf_cnn_7.py :     Der Trainingsaufbau der Architektur  $a_1$ .
└── leaf_cnn_7_tmp.py : Temporäre Datei für denselben Aufbau.
└── test_dropout.py :   Iteratives Testen von Dropout Raten auf leaf_cnn_7_tmp.py
└── predict_single.py : Klassifizieren eines einzelnen Bildes mit trainiertem Modell.
└── variation01.jpg :   Ein Bild, das stark von den anderen derselben Kategorie abweicht.
└── variation02.jpg :   Ein weiteres solches variierendes Bild, Abbildung 7.3
```

Zum Ausführen der Python Skripte wird eine aktuelle Version (Stand 18. März 2019) von Python 3 benötigt, sowie die in den Import-Sektionen im Kopfteil der Programmierungen genannten Erweiterungen (unter anderem TensorFlow und Keras, sowie mathematische Pakete wie NumPy).