FH Westküste
Wirtschaft und Technik

HAW HAMBURG

Masterarbeit

Tim-Gabriel Paulini

Effektivität der automatischen Variation von Kameraparametern als Data Augmentation beim Training neuronaler Netze

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Tim-Gabriel Paulini

Effektivität der automatischen Variation von Kameraparametern als Data Augmentation beim Training neuronaler Netze

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Masterstudiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Marc Hensel
Zweitgutachter: Prof. Dr.-Ing. Sönke Appel

Eingereicht am: 14. April 2025

Tim-Gabriel Paulini

Thema der Arbeit

Effektivität der automatischen Variation von Kameraparametern als Data Augmentation beim Training neuronaler Netze

Stichworte

Computer Vision, Neuronale Netze, Data Augmentation, Kameraparameter

Kurzzusammenfassung

Diese Arbeit untersucht, ob die automatische Variation von Kameraparametern mithilfe einer Sequencer-Funktion eine effektive Alternative zur softwarebasierten Data Augmentation für das Training neuronaler Netze darstellt. Ziel ist es, die Effizienz dieses Ansatzes zu bewerten und sein Potential für industrielle Anwendungen aufzuzeigen.

Tim-Gabriel Paulini

Title of Thesis

Effectiveness of automatic variation of camera parameters as data augmentation in training neural networks

Keywords

Computer vision, neural networks, data augmentation, camera parameters

Abstract

This thesis investigates whether the automatic variation of camera parameters using a sequencer function is an effective alternative to software-based data augmentation for training neural networks. The aim is to evaluate the efficiency of this approach and to demonstrate its potential for industrial applications.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

Adam Adaptive Moment Estimation.

ANN Artificial Neural Network.

API Application Programming Interface.

CNN Convolutional Neuronal Networks.

DCGAN Deep Convolutional Generative Adversarial Network.

GAN Generative Adversarial Network.

KI Künstliche Intelligenz.

MLP Multi-Layer-Perzeptron.

NST Neural Style Transfer.

RMSprop Root Mean Square Propagation.

SGD Stochastic Gradient Descent.

VAE Variational Auto-Encoder.

VGG Visual Geometry Group.

Listings

1 Einleitung

Im Zuge der Industrialisierung wurden immer wieder einfache Tätigkeiten durch Maschinen ersetzt und damit die Produktivität gesteigert. So hat sich die Technik im Laufe der Zeit weiterentwickelt, Aufgaben übernommen und neue Arbeitsplätze geschaffen.

Heute übernehmen Maschinen einen Großteil der schweren Arbeit, zum Beispiel in einer Schraubenfabrik. Die Schrauben werden aus Metallstangen hergestellt, indem diese geschnitten, geformt und mit einem Gewinde versehen werden. Dann werden sie über ein Fließband in eine Schachtel gelegt, verpackt und ausgeliefert. Dabei kann eine Sichtkontrolle durch einen Menschen oder eine Kamera erfolgen, damit falsche, zu kurze oder defekte Schrauben aussortiert werden und der Kunde sie nicht erhält.

Diese Sortieraufgabe übernimmt eine Maschine, die mit Hilfe von Bildverarbeitung die Schrauben analysiert und klassifiziert, um anschließend zu entscheiden, ob die Schraube den Qualitätsanforderungen entspricht. Die Bildverarbeitung kann dabei auf klassischen algorithmusbasierten Verfahren beruhen, die speziell für diesen Anwendungsfall entwickelt wurden. Ein solcher Algorithmus analysiert beispielsweise die Länge der Schraube, die Beschaffenheit des Gewindes und die Form des Schraubenkopfes. Anhand dieser Merkmale wird entschieden, ob die Schraube für den Verkauf geeignet ist. Dieser Prozess kann entweder durch einfache Regelabfragen erfolgen oder durch den Einsatz von maschinellem Lernen, um die Entscheidungsfindung zu optimieren.

Beim maschinellen Lernen wird ein Computer zunächst mit einer großen Datenmenge trainiert: Ihm wird gezeigt, wie eine korrekte Schraube auszusehen hat und welche Fehler akzeptabel sind. Mit diesen Daten und zusätzlichen Beispielen fehlerhafter Schrauben lernt das System dann, einen neuen Input aus der Bildverarbeitung zu klassifizieren und diese Schraube auszusortieren.

Der Schritt der Bildverarbeitung kann dabei auch teilweise oder vollständig durch ein neuronales Netz ersetzt werden, das die Aufnahmen direkt analysiert und auswertet.

Dadurch können Strukturen und Formen erkannt werden, die herkömmlichen Bildverarbeitungsmethoden oft verborgen bleiben. Ein neuronales Netz benötigt jedoch eine große Datenbasis, um effizient trainiert zu werden und vielseitig klassifizieren zu können. Im Gegensatz zur klassischen Bildverarbeitung, die auf spezifische Algorithmen angewiesen ist, kann ein gut trainiertes neuronales Netz flexibel auf verschiedene Aufgaben angewendet werden - vorausgesetzt, es wurde mit ausreichend unterschiedlichen Daten trainiert.

Wissenschaftliche Arbeiten fokussieren häufig auf den Aufbau eines universellen Netzes, das mit Aufnahmen aus dem Internet trainiert wird. Für industrielle Anwendungen ist dies jedoch meist nicht sinnvoll, da am Fließband ein neues Produkt entsteht, für das es noch keine Aufnahmen aus dem Internet gibt - ein vortrainiertes Netz ist daher nur bedingt geeignet. Vor Ort werden Aufnahmen gemacht, aus denen ein neuer, speziell auf das Problem in der Fabrik zugeschnittener Datensatz erstellt und für das Training des Netzes genutzt wird. Durch Data Augmentation, bei der künstlich weitere Daten erzeugt werden, lässt sich der Erstellungsaufwand verringern und das Netz robuster trainieren. So lernt das System, das gewünschte Objekt auch bei variierenden Lichtverhältnissen, Ausrichtungen oder optischen Veränderungen zuverlässig zu erkennen.

Diese Arbeit untersucht das Training eines neuronalen Netzes, die anschließende Analyse sowie die Evaluierung der Ergebnisse. Im Fokus steht dabei die Frage, wie sich die Data Augmentation mit Hilfe des Sequencers, der die Kameraparameter während der Aufnahme verändern kann, von rein softwarebasierten Ansätzen unterscheidet. Zudem wird analysiert, wie sich ein Datensatz mit und ohne Data Augmentation auf die Performance des neuronalen Netzes auswirkt.

Die Zusammenarbeit mit der Allied Vision Technologies GmbH aus Ahrensburg, die das Kamerasystem mit Sequencer-Funktion bereitstellt, bietet die Möglichkeit, den praktischen Nutzen dieses Ansatzes zu bewerten. Ziel ist es herauszufinden, ob dieser Ansatz das Potenzial hat, zukünftige Produktionsprozesse effizienter zu gestalten und weiterentwickelt werden sollte.

2 Grundlagen

In diesem Kapitel werden die Grundlagen vermittelt, die zum Verständnis des Kerns dieser Arbeit notwendig sind. Es werden die Themen Kameraparameter, maschinelles Lernen, neuronale Netze, Convolutional Neuronal Networks (CNN) und die dazugehörigen Optimierungsverfahren behandelt.

2.1 Kameraparameter

Diese Masterarbeit beschäftigt sich mit der Aufnahme von Datensätzen, die durch die Einstellung der Kameraparameter beeinflusst werden sollen. Daher sollen die Kameraparameter Belichtungszeit, Gain, Gamma, RGB, Farbton und Sättigung kurz erläutert werden, um ein besseres Verständnis für die spätere Verarbeitung der Datensätze zu schaffen.

Für eine Aufnahme muss ein Sensor das Licht über einen bestimmten Zeitraum - die **Belichtungszeit** - einfangen und anschließend verarbeiten. Eine lange Belichtungszeit sorgt dabei dafür, dass der Sensor lange belichtet wird und die Aufnahme dadurch heller wird. Bei Bewegungen ist eine kurze Belichtungszeit sinnvoll, da nur ein Moment der Bewegung aufgenommen werden soll. Mit einer langen Belichtungszeit wird die Bewegung mit aufgenommen und die Aufnahme wird unscharf. Eine kurze Belichtungszeit „friert“ schnelle Bewegungen ein, aber die Aufnahme ist dunkel - hier kann aber das Empfangssignal durch die Verstärkung (**Gain**) verstärkt werden. Durch Erhöhung des Verstärkungsfaktors wird das durch die einfallenden Lichtstrahlen erzeugte elektrische Signal verstärkt, was zu einer helleren Aufnahme führt. Eine zu hohe Verstärkung kann auch zu Bildrauschen führen, da besonders bei dunklen Aufnahmen, wo das Signal schwach ist, das Rauschen im Verhältnis stärker wahrgenommen wird.

Mit Hilfe des **Gamma**-Wertes kann der Helligkeitsverlauf einer Aufnahme beeinflusst werden. Die veränderbaren Werte stellen den mittleren Bereich der Helligkeitswerte dar,

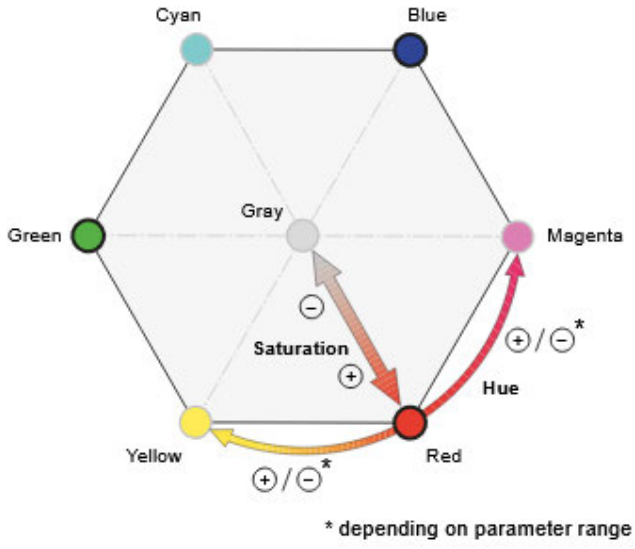


Abbildung 2.1: Darstellung von den Kameraparametern Sättigung und Farbton [3]

was zur Folge hat, dass die dunklen und hellen Bereiche unverändert bleiben. Dieser Effekt verbessert den Kontrast und die Detailgenauigkeit.

Ein Farbbild besteht aus drei Kanälen, die die Farben repräsentieren: Rot, Grün und Blau (**RGB**). Es ist möglich, einen Farbkanal durch Multiplikation mit einem hohen Faktor hervorzuheben oder einen Kanal wegzulassen (Multiplikation mit 0). Mit Hilfe des Farbtons (**Hue**) kann die dominante Farbe in der Aufnahme gesteuert werden. Dadurch ist es möglich, diese Farbe in der Aufnahme zu verschieben und eine andere Farbe dominanter zu machen (siehe Abbildung 2.1). Mögliche Farben sind Rot, Magenta, Blau, Cyan, Grün und Gelb. Mit der Sättigung (**Saturation**) kann die Farbintensität eingestellt werden. Bei einer Sättigung von 0 verschwinden alle Farben und es entsteht ein Graustufenbild, während eine hohe Sättigung die Farbintensität verstärkt. Die Kameraparameter Belichtungszeit und Gain können ausschließlich direkt an der Kamera angepasst werden, während Werte wie Gamma, Farbton, Sättigung und RGB-Matrix digital eingestellt werden können. Für diese Arbeit sind vor allem die analogen Einstellungen von Interesse, da sie in einem digitalen Datensatz nicht exakt simuliert werden können. Daher könnte es vorteilhaft sein, wenn diese Werte während der Aufnahme automatisch angepasst werden könnten.

Normalerweise werden die Kameraparameter vor einer Aufnahme festgelegt und bleiben für alle nachfolgenden Aufnahmen unverändert. Mit Hilfe eines Sequencers kann

diese Einschränkung überwunden werden. Ein Sequencer ist eine Kamerafunktion, die es ermöglicht, voreingestellte Parameter in aufeinanderfolgenden Aufnahmen automatisch zu variieren. Dadurch können mehrere Bilder mit unterschiedlichen Einstellungen aufgenommen und gespeichert werden, ohne dass manuelle Anpassungen erforderlich sind.

2.2 Maschinelles Lernen

Im Bereich des maschinellen Lernens stehen zwei Fragen im Mittelpunkt: Wie kann ein System geschaffen werden, das sich durch Erfahrung und Zeit automatisch verbessert, und was sind die Regeln für dieses Lernverhalten von System, Mensch und Computer? Maschinelles Lernen hat sich in den letzten Jahrzehnten von Laborversuchen zu kommerziellen und alltäglichen Anwendungen entwickelt. Künstliche Intelligenz (KI) ist heute der Lösungsansatz für Softwareentwicklungen im Bereich des *Computer Vision*, der Spracherkennung, der Robotersteuerung und vielen anderen Ansätzen. Bei diesem Ansatz werden Aufgaben, wie z. B. Anwendungen im Bereich der digitalen Bildverarbeitung, mit Hilfe von KI gestaltet, indem dem System das gewünschte Verhalten an den Ein- und Ausgängen gezeigt und daraus eine Funktion generiert wird, anstatt einen klassischen Bildverarbeitungsansatz zu verwenden.

Ein Problem, das durch maschinelles Lernen gelöst werden soll, ist in der Regel die Ausführung einer Aufgabe, die durch eine Art Training effizienter gemacht werden soll. Ein Beispiel wäre die Erkennung von Spam in einem E-Mail-Postfach. Dieser Prozess wird mit Hilfe von Leistungsmetriken verbessert, wobei entschieden werden muss, welche Metrik verbessert werden soll. In diesem Fall kann die Genauigkeitsmetrik des Spam Mail Identifiers verbessert werden, indem ein Training mit nach Spam und Nicht-Spam sortierten E-Mails durchgeführt wird, um ein Muster zu erkennen. Eine weitere Möglichkeit, den Algorithmus zu verbessern, wäre eine Metrik, die prüft, ob eine Mail fälschlicherweise als Spam klassifiziert wurde und diese mit einem Strafpunkt versieht. Ein solcher Strafpunkt zeigt dem Algorithmus, dass die Lösung falsch ist und er lernt daraufhin das neue Muster (nach [19]).

Dieses Erkennen eines Musters und die anschließende Zuordnung zu einer Klasse (Spam oder kein Spam) wird auch als Klassifikation bezeichnet.

Maschinelles Lernen lässt sich in verschiedene Kategorien von Lernverfahren unterteilen. Dazu gehören das *Supervised Learning*, das *Unsupervised Learning*, das *Semi-Supervised Learning* und das *Reinforcement Learning*, die im Folgenden erläutert werden.
Ein Modell ist dabei ein Konzept einer Anwendung des maschinellen Lernens, bei dem bestimmte Parameter und Hyperparameter (s. Kap. 2.4.6) zusammenwirken, um eine Aufgabe zu lösen.
Supervised Learning ist ein Verfahren des maschinellen Lernens, bei dem ein Modell mit Hilfe von Trainingsdaten trainiert wird. Ziel ist es, aus Eingabe- und Ausgabedaten eine Funktion zu erlernen, um neue Daten vorhersagen zu können. Typische Anwendungen sind die Klassifikation und die Regression[1].
Unsupervised Learning setzt auf das Lernen von Mustern und Strukturen und konstruiert daraus eine Funktion, ohne mit gelabelten Datensätzen zu arbeiten. Gelabelte Daten sind Daten, die zuvor von einem Menschen identifiziert und gespeichert wurden. Dies wird verwendet, um Merkmale zu extrahieren oder Muster in Daten zu erkennen.
Semi-supervised Learning ist eine Kombination der beiden vorhergehenden Methoden, bei der ein Modell mit einer kleinen Menge von gelabelten Daten und einer großen Menge von ungelabelten Daten trainiert wird.
Reinforcement Learning ist eine Methode des maschinellen Lernens, bei der ein Agent[2] in einer Umgebung agiert und durch Belohnung und Bestrafung lernt, eine optimale Strategie zu entwickeln. Diese Methode eignet sich besonders für die Automatisierung, die Robotik und auch für Spiele. Allerdings eignet sich diese Methode weniger für die Klassifikation als vielmehr für die Optimierung der Effizienz (nach [42]).

2.2.1 Computer Vision

Computer Vision ist eine Spezialisierung des maschinellen Lernens und bezieht sich auf das Lernen und Erkennen von Mustern in Aufnahmen. Ein solches System, das Computer Vision anwendet, muss auch verstehen, welcher Kontext in einer Aufnahme dargestellt wird. Am Beispiel eines autonomen Fahrzeugs bedeutet dies, dass das Auto verstehen muss, welches Straßenschild was bedeutet, wo die Fahrbahnbegrenzungen sind, wo die entsprechenden Mittelpunkte sind und wo sich Fußgänger befinden. All

[1] Die Regression ist ein Verfahren zur Vorhersage kontinuierlicher Werte, bei dem der Zusammenhang zwischen einer abhängigen und einer oder mehreren unabhängigen Variablen modelliert wird.
[2] Ein Agent in einem System ist ein Konzept, das einen Prozess beobachtet und auf der Grundlage seines Wissens Entscheidungen trifft.

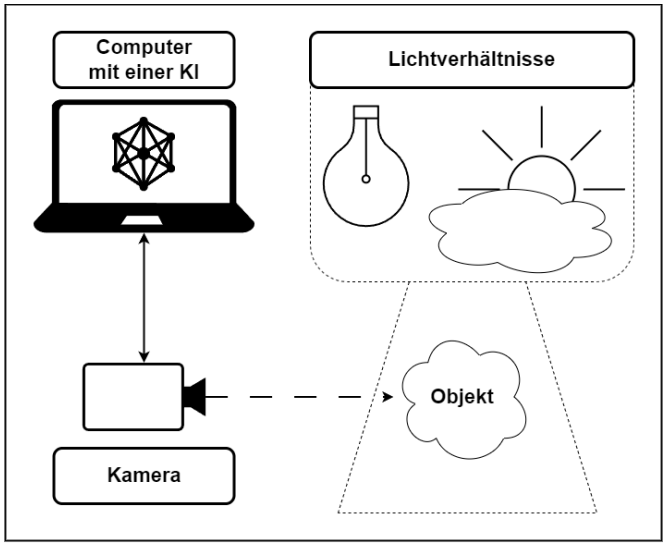


Abbildung 2.2: Komponenten, die beim maschinellen Sehen berücksichtigt werden müssen

diese Merkmale muss ein solches System mit Hilfe von Sensoren verstehen, um sich autonom im Straßenverkehr bewegen zu können.

Das Erkennen und Klassifizieren eines Objekts durch einen Computer soll auch anhand eines Menschen erklärt werden: Ein Mensch sieht mit seinen Augen - nimmt eine Aufnahme auf und kann dann aufgrund seiner Erfahrungen und Erinnerungen sagen, was genau auf dieser Aufnahme zu sehen ist und kann daraus Rückschlüsse auf die Umstände ziehen. Ein Mensch kann von seinen Erinnerungen und Erfahrungen profitieren - eine KI kann das zunächst nicht. Wird einem Menschen ein Objekt zum ersten Mal gezeigt, lernt er und kann das Gelernte später wiedergeben. Durch diese Analogie ist es auch einer KI möglich, anhand von Aufnahmen zu lernen, ein Objekt zu klassifizieren.

Die Abbildung 2.2 zeigt die Voraussetzungen für Computer Vision. Benötigt wird ein Computer mit der KI, die sehen soll - in diesem Fall mit einer Kamera[3]. Bei dieser Aufnahme eines Objektes sind dann auch die unterschiedlichen Umgebungsbedingungen (Sonneneinstrahlung, Schatten oder farbiges Licht) von Bedeutung, da diese einen Einfluss auf die spätere Klassifizierung haben können. Damit ein Algorithmus ein Objekt identifizieren kann, muss er in der Lage sein, aus Merkmalen und Mustern zu lernen. Um dies zu ermöglichen, wurde ein Konzept des menschlichen Gehirns nachgebildet: Neuro-

[3]Es muss keine Kamera vorhanden sein - es können auch Bilder aus dem Speicher des Computers verwendet werden.

nen. Neuronen sorgen im Gehirn dafür, dass Informationen verarbeitet werden können. Wenn ein Neuron aktiviert wird, sendet es ein Signal an die im Netzwerk verbundenen Neuronen und es kommt zu einer Kettenreaktion. Eine Aktivierung findet statt, wenn das Neuron genügend Eingangssignale bis zu einem bestimmten Schwellenwert erhalten hat, der auch als Aktivierungsfunktion bezeichnet wird. Eine künstliche Verknüpfung solcher Neuronen wird auch als *Artificial Neural Network (ANN)* bezeichnet (vgl. [10, S. 3ff]).

2.2.2 Klassifikation

Eine der Aufgaben des maschinellen Lernens ist die Vorhersage einer Klassifikation oder Regression mit Hilfe einer Funktion. Unter Klassifikation versteht man die Zuordnung von Daten zu einer Klasse, unter Regression die Vorhersage von Zahlenwerten. Bei der Klassifikation wird zwischen binärer und multipler Klassifikation unterschieden. Bei der binären Klassifikation gibt es zwei Klassen - meist positiv und negativ - während bei der multiplen Klassifikation mehrere Klassen existieren. Hier wird weiter unterschieden zwischen der Multiclass-Klassifikation, bei der die Aufgabe darin besteht, aus mehreren Klassen eine Klasse zu bestimmen, und der Multilabel-Klassifikation, bei der die Aufgabe darin besteht, für ein Objekt mehrere Klassen zu bestimmen (nach [42]).

In der Klassifikation im Bereich *Computer Vision* wird die *Multiclass*-Klassifikation verwendet, um die Klassen von Objekten zu bestimmen. Dabei werden die Varianten *Image Classification*, *Object Detection* und *Image Segmentation* unterschieden (siehe Abbildung 2.3). Die *Image Classification* ist die einfachste Variante, bei der eine Aufnahme einer Klasse zugeordnet wird. Die *Object Detection* ermittelt die Position und Klasse von Objekten in einer Aufnahme (meist in Form eines Rechtecks) und stellt somit eine Erweiterung der *Image Classification* dar. Eine weitere Erweiterung ist die *Image Segmentation*, bei der die Pixel einer Aufnahme einer Klasse zugeordnet werden (nach [20]). Um ein Projekt im Bereich der *Computer Vision* durchführen zu können, wird ein Datensatz benötigt, der im Falle des überwachten Lernens gelabelte Aufnahmen enthält. Ein Projekt wäre dann z. B. ein Modell, welches eine einfache Klassifikation durchführen muss.

In der Forschung wird bei der Entwicklung präziser Modelle in der Regel ein bereits vorhandener Datensatz verwendet, der als Benchmark[4] dient. Anhand dieses Bench-

[4]Ein Benchmark ist ein Vergleichsmaßstab und dient zum Vergleich zwischen mehreren Modellen.

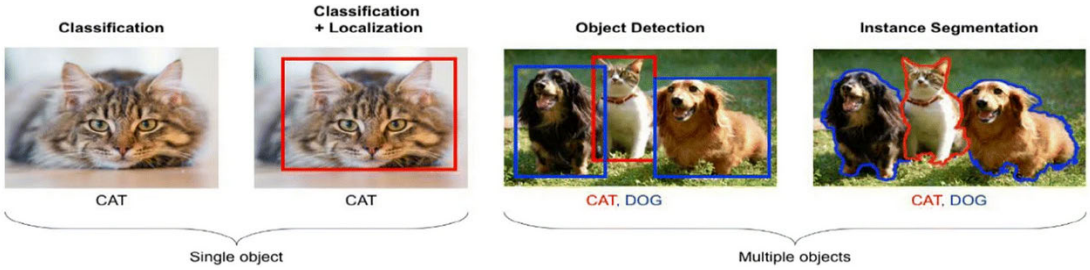


Abbildung 2.3: Klassifikationsverfahren im Bereich der *Computer Vision* [20]

marks kann dann die Güte des trainierten Modells gemessen werden und dient somit als Vergleich. Es gibt viele verschiedene Benchmarks, der bekannteste im Bereich *Computer Vision* ist der MNIST-Datensatz [26], der aus handgeschriebenen Zahlen besteht. Neuere und komplexere Datensätze sind ImageNet [8], MS COCO [30] und Pascal VOC [12]. In der Praxis kann es jedoch vorkommen, dass ein eigener Datensatz erstellt werden muss, um ein spezifisches Problem zu lösen. In diesem Fall ist es wichtig, dass der Datensatz groß genug ist und die Daten gut beschriftet sind, um ein gutes Modell zu trainieren. Es gibt viele Methoden, um einen Datensatz zu erstellen, z. B. Daten aus dem Internet sammeln, synthetische Daten erstellen oder eigene Daten erstellen (nach [20]).

2.3 Neuronale Netze

Ein neuronales Netz setzt sich aus einzelnen Neuronen zusammen. Jedes Neuron verarbeitet Eingabewerte, um mithilfe einer Aktivierungsfunktion (siehe Kapitel 2.3.1) eine Ausgabe zu erzeugen. Mit Hilfe dieser Aktivierungsfunktion wird geprüft, ob ein Schwellenwert überschritten wurde und wenn ja, sendet dieses Neuron eine Eins an die nachfolgenden Neuronen.

Diese Eingänge werden *Input Features* genannt, und ein Neuron kann mehrere davon haben. Nicht jeder Input hat das gleiche Gewicht für die Berechnung der Aktivierungsfunktion und wird gewichtet - das sind die *Connection Weights* pro *Input Feature*. Diese Gewichtung zu optimieren, damit das neuronale Netz später eine korrekte Klassifikation durchführen kann, ist Teil des Trainings.

Die Abbildung 2.4 zeigt das Neuron, dargestellt durch die Aktivierungsfunktion $f(x)$, und seine *Input Features*, bezeichnet mit x_n. Jedes dieser *Input Features* wird mit einem

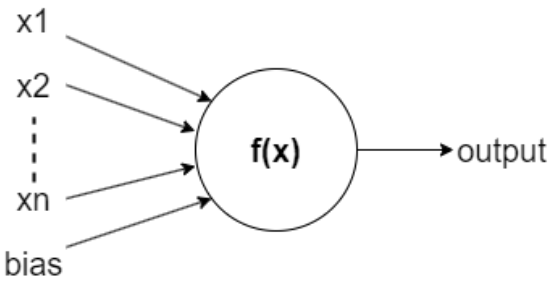


Abbildung 2.4: Darstellung eines Neurons in einem neuronalen Netz

Gewicht w_n multipliziert und anschließend wird ein *Bias* addiert, der als zusätzlicher Input des Neurons verstanden werden kann und durch das Training beeinflusst wird. Daraus ergibt sich die folgende Gleichung einer linearen Kombination z:

$$z = \sum_{n} x_n \cdot w_n + b \tag{2.1}$$

Das Neuron lernt durch *Trial and Error* und versucht bei jedem Durchlauf die entsprechenden Gewichte so anzupassen, dass die Ausgabe richtig berechnet wird. Der *Feedforward* Prozess ist der Prozess der Berechnung einer Vorhersage $\hat{y}$ aus der gewichteten Summe 2.1 durch die Aktivierungsfunktion.

$$\hat{y} = f(\mathbf{x}) \tag{2.2}$$

Dann wird berechnet, wie weit diese Vorhersage vom tatsächlichen Label abweicht. Dies geschieht mit der folgenden Funktion, die ebenfalls mit der Verlustfunktion - *Loss Function* - (siehe Kap. 2.3.2) berechnet wird.

$$\mathrm{E} = y - \hat{y} \tag{2.3}$$

Mit diesem berechneten Fehler (E) werden dann die Gewichte angepasst und der Prozess wiederholt sich.

Dieses eine Neuron ist also für die Trennung der Daten in einem linearen Zusammenhang zuständig, was nur ein Teil der Lösung für komplexere Aufgaben ist. Diese müssen daher aus mehreren Schichten von Neuronen bestehen, sogenannten *Multi-Layer-Perzeptron (MLP)* . Dabei werden mehrere Neuronen gemeinsam genutzt, um komplexere Strukturen erkennen zu können.

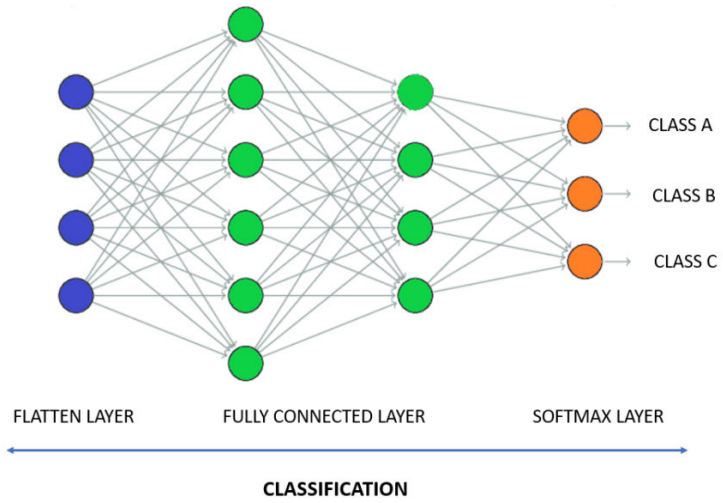


Abbildung 2.5: Darstellung der Fully Connected Layers [1]

Die Neuronen sind dabei in Spalten angeordnet, erhalten ihre Eingangsmerkmale von mehreren Neuronen aus einer vorhergehenden Spalte und geben ihre Ausgangsmerkmale an die nachfolgende Spalte ab. Diese Spalte von Neuronen wird dann auch *Hidden Layer* genannt, da diese Spalte keine direkte Ausgabe hat, die gelesen werden kann - sie ist nur für die nachfolgende Spalte von Neuronen entscheidend. Dies ist das erste wichtige Konstrukt für neuronale Netze: *Fully Connected Layers* (zu sehen in der Abbildung 2.5).

Außerdem werden die Eingänge des neuronalen Netzes als *Flatten Layer* bezeichnet, das nur Neuronen mit Ausgängen besitzt. Am Ausgang des Netzes steht dann in der Regel ein *softmax Layer* (siehe Kapitel 2.3.1), die die Klassifikation darstellt (nach [10, S. 36ff]).

Ein neuronales Netz lernt durch Wiederholung von *Feedforward* und *Backpropagation*. Bei *Feedforward* wird die gewichtete Summe mit der jeweiligen Aktivierungsfunktion multipliziert, um den Ausgangswert zu erhalten. Mit der Verlustfunktion (s. Kap. 2.3.2) wird dann berechnet, wie weit die Vorhersage vom richtigen Label abweicht. Mit diesem Fehler wird dann im Optimierungsprozess ein Gradient berechnet, der für die Anpassung des Gewichts verantwortlich ist (s. Kap. 2.3.3). Es folgt die Formel für die Gewichtsänderung als Gradientenschritt δ (mit E als Fehler aus der Verlustfunktion):

$$\delta w_n = -\alpha \frac{dE}{dw_n} \tag{2.4}$$

Hierbei wird α, die Lernrate (*learning rate*) (siehe Kapitel 2.3.3), mit der Ableitung des Fehlers nach dem jeweiligen Gewicht multipliziert. Die Lernrate bestimmt, wie stark die Gewichte in jedem Schritt angepasst werden, um den Fehler zu minimieren.

Im *Backpropagation*-Prozess wird die Gewichtungsmatrix $\mathbf{W}$ aktualisiert. Das Ergebnis ist die neue Gewichtungsmatrix:

$$\mathbf{W} = \mathbf{W} - \alpha \cdot \nabla_{\mathbf{W}} E \tag{2.5}$$

wobei $\nabla_{\mathbf{W}} E$ der Gradient des Fehlers bezüglich $\mathbf{W}$ ist.

Das *Backpropagation*-Verfahren unterscheidet sich etwas von einem vollständigen neuronalen Netz: Hier arbeitet sich ein Algorithmus vom Ausgang (*back*) zum Eingang vor und überarbeitet die Gewichte gemäß der Formel (mit t als Iterationsschritt):

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \alpha \cdot \nabla_{\mathbf{W}} E \tag{2.6}$$

Damit ist jedes Gewicht im Netz aktualisiert und der Prozess kann von neuem beginnen (nach [10, S. 86f]).

2.3.1 Aktivierungsfunktion

Es gibt eine Vielzahl von Aktivierungsfunktionen, die für jede Schicht definiert werden können. Die beschriebene Funktion 2.1 ist eine lineare Übertragungsfunktion und stellt die einfachste Variante dar.
Die *sigmoid*-Funktion der Formel 2.7 ist eine sehr gängige Methode zur Bestimmung der Wahrscheinlichkeit einer binären Klassifikation. Der Vorteil liegt darin, dass Ausreißer nicht eliminiert werden, sondern nur minimal in die Berechnung eingehen. Die Exponentialfunktion beschreibt dabei den Zusammenhang zwischen dem Eingangswert x und dem Ausgang der Aktivierungsfunktion.

$$f(x)_{sigm} = \frac{1}{1 + e^{-x}} \tag{2.7}$$

Die Softmax-Funktion wird häufig in der letzten Schicht eines neuronalen Netzes verwendet, um die Wahrscheinlichkeiten für jede Klasse zu berechnen. Sie stellt sicher,

dass die Wahrscheinlichkeiten p_i zwischen 0 und 1 liegen und sich zu 1 addieren:

$$p_i = \frac{e^{a_i}}{\sum_{n=1}^{N} e^{a_n}} \tag{2.8}$$

Dabei ist a_i der nicht normierte Ausgabewert der vorhergehenden Schicht und N die Anzahl der Klassen.
Eine häufig verwendete Aktivierungsfunktion ist die *ReLU*-Funktion, die positive Eingabewerte verwendet und daher weniger Rechenleistung benötigt (nach [10, S. 51ff], [4]).

$$f(x)_{ReLU} = \max(0, x) \tag{2.9}$$

2.3.2 Verlustfunktion

Verlustfunktionen dienen zur Korrektur der Neuronen, um die jeweiligen Gewichte anzupassen. Es wird immer der Verlustfunktion berechnet. Denn wenn der Fehler klein ist, ist auch der Fehler, den das neuronale Netz gemacht hat, klein und liefert somit eine richtige Lösung. Es gibt verschiedene Methoden, diesen Fehler zu berechnen; kleine Fehler sollten die Verlustfunktion nicht so stark beeinflussen wie große Fehler. Ein häufig verwendetes Beispiel für eine Verlustfunktion ist der *Cross-Entropy Loss*. Diese Funktion wird besonders oft bei *CNN* (siehe Kapitel 2.4) eingesetzt (vgl. [4]). Die Wahrscheinlichkeiten der Vorhersagen werden nun mit der folgenden Formel berechnet, um den Fehler zu erhalten (mit $\hat{y}$ als Vorhersage und y als korrektes Label):

$$E(\mathbf{W}, \mathbf{b}) = -\sum_{n=1}^{N} y_n \log(\hat{y}_n) \tag{2.10}$$

Diese Formel kann also auf N Klassen angewendet werden, die hier identifiziert werden sollen. Dabei ist $\mathbf{W}$ die Gewichtsmatrix aller zugehörigen Gewichte und $\mathbf{b}$ der zugehörige *Bias*-Vektor. Demnach ist $E = 1,2$ ein schlechteres Ergebnis als $E = 0$, was eine korrekte Verteilung darstellt (nach [10, S. 71f]).

Die *Negative Log Likelihood Loss* (NLLLoss) ist eine weitere Verlustfunktion, die zur Optimierung von CNNs verwendet werden kann. Sie ist mathematisch identisch zur *Cross-Entropy Loss* und berechnet den Fehler zwischen den richtigen Labels y und den Vorhersagen $\hat{y}$. Ziel ist es, den Verlustwert zu minimieren, um die Genauigkeit (*Accuracy*) des Netzes zu erhöhen. Dies wird erreicht, indem die negative logarithmische

Wahrscheinlichkeit der vorhergesagten Klasse minimiert wird. Es wird mit einem *One-Hot*-Vektor gearbeitet, der die wahre Klasse repräsentiert. Bei der *One-Hot*-Kodierung wird die Klasse, die die Aufnahme darstellt, mit 1 kodiert, während alle anderen Klassen mit 0 kodiert werden. Somit ist $y_c = 1$ für die wahre Klasse c und $y_n = 0$ für alle anderen Klassen $n \neq c$. Der folgende Fehlerterm wird berechnet (wobei c der Index der richtigen Klasse ist):

$$E(\mathbf{W}, \mathbf{b}) = -log(\hat{y}_c) \tag{2.11}$$

Hierbei tragen die falsch vorhergesagten Klassen nicht direkt zum Fehler bei, da $y_n = 0$ für $n \neq c$ gilt. Wenn $\hat{y}_c$ gegen 1 konvergiert, ist der Fehler klein, wenn $\hat{y}_c$ sehr klein ist, ist der Fehler groß (nach [57]).

Eine weitere Verlustfunktion ist die *Kullback-Leibler-Divergenz*, die häufig bei *Variational Auto-Encoder (VAE)* verwendet wird. Diese Methode zielt darauf ab, eine Verteilung und nicht einen festen Wert zu modellieren. Damit diese Verlustfunktion funktioniert, muss zunächst in der Software sichergestellt werden, dass die Summe der berechneten Wahrscheinlichkeiten 1,0 ergibt. Die Formel für diese Berechnung ist das Verhältnis der wahren Verteilung y zur vorhergesagten Verteilung $\hat{y}$ (nach [25, 45, 9, 22]):

$$E(\mathbf{W}, \mathbf{b}) = \mathcal{D}_{KL}(y|\hat{y}) = \sum_{n=1}^{N} y_n \cdot \log\left(\frac{y_n}{\hat{y}_n}\right) \tag{2.12}$$

Dabei entspricht y_n der tatsächlichen Wahrscheinlichkeit (bei *One-Hot*-Kodierung meist 0 oder 1) und $\hat{y}_n$ der vom Modell berechneten Wahrscheinlichkeit.

2.3.3 Optimierungsverfahren

Es gibt verschiedene Optimierungsverfahren, die zu einem minimalen Fehler führen. Dabei gilt: Je kleiner der Fehler, desto höher die Genauigkeit des Netzes. Bei dieser Optimierung werden die Gewichte und der *Bias* des Netzes so angepasst, dass am Ende der Fehler minimal ist.

Um dieses Optimierungsproblem zu lösen, müssen der Gradient und die Lernrate angepasst werden. Die Lernrate bestimmt die Größe der Schritte, die das Netz während der Optimierung macht. Ein zu großer Wert kann dazu führen, dass das Netz nicht konvergiert, ein zu kleiner Wert kann dazu führen, dass das Netz sehr lange braucht, um zu

konvergieren. Zu diesem Zweck wird auch ein *Scheduler* verwendet, der im Abschnitt 2.3.4 näher erläutert wird.

Ein weiterer wichtiger Hyperparameter ist die *Batch Size*. Die *Batch Size* bestimmt, wie viele Trainingsbeispiele in einem einzelnen *Batch* verarbeitet werden. Eine Epoche besteht aus mehreren *Batches*, so dass der gesamte Trainingsdatensatz einmal durch das Netz läuft. Eine kleine *Batch Size* führt zu einer häufigeren Aktualisierung der Gewichte, was zu einer schnelleren Konvergenz, aber auch zu einer größeren Varianz in den Gradienten führen kann. Eine große *Batch Size* führt zu einer stabileren Schätzung der Gradienten, benötigt aber mehr Speicher und kann die Trainingszeit verlängern. Die Wahl der *Batch Size* hängt von der verfügbaren Hardware und den spezifischen Anforderungen des Modells ab.

Die Abbildung 2.6 veranschaulicht ein solches Optimierungsproblem, wobei der Einfachheit halber nur zwei Gewichte dargestellt werden. Der Punkt A stellt den Startpunkt dar, der durch eine Initialisierung der Gewichte bestimmt wird (vgl. Kap. 2.3.5). Die Aufgabe besteht darin, das Minimum des dargestellten Graphen zu finden. Dazu sind zwei Parameter entscheidend: der Gradient und die Lernrate. Der Gradient gibt an, in welcher Richtung dieses Minimum gesucht werden soll und ist die Ableitung des Fehlers E, abhängig von der Gewichtung: $\frac{dE}{dw}$.

Die Lernrate hingegen ist die Schrittweite zum nächsten Punkt. Eine große Lernrate steht für große und eine kleine Lernrate für kleine Schritte zwischen den Punkten. Betrachtet man in der Abbildung 2.6 den Weg ab Punkt C, so fällt auf, dass dieser nicht zu einem Minimum, sondern wieder zu einem Maximum führt. Es zeigt sich also, dass ein hoher Wert für die Lernrate nicht immer eine gute Wahl ist, da ein langer Sprung den Fehler nicht immer minimiert, sondern vielleicht sogar erhöht. In diesem Beispiel wäre ein niedrigerer Wert für Punkt C besser geeignet, um das Minimum zu erreichen.

Für diese Optimierungsprobleme eignen sich Optimierer (*Optimizer*): Das sind Algorithmen, die auch mit sehr vielen Gewichten das Minimum finden können (nach [10, S. 74ff]).

Ein einfaches Verfahren ist der *Batch Gradient Descent*, der den Gradienten nach folgender Formel berechnet:

$$\mathbf{W} = \mathbf{W} - \alpha \cdot \nabla_{\mathbf{W}} E(\mathbf{W}, \mathbf{b}) \tag{2.13}$$

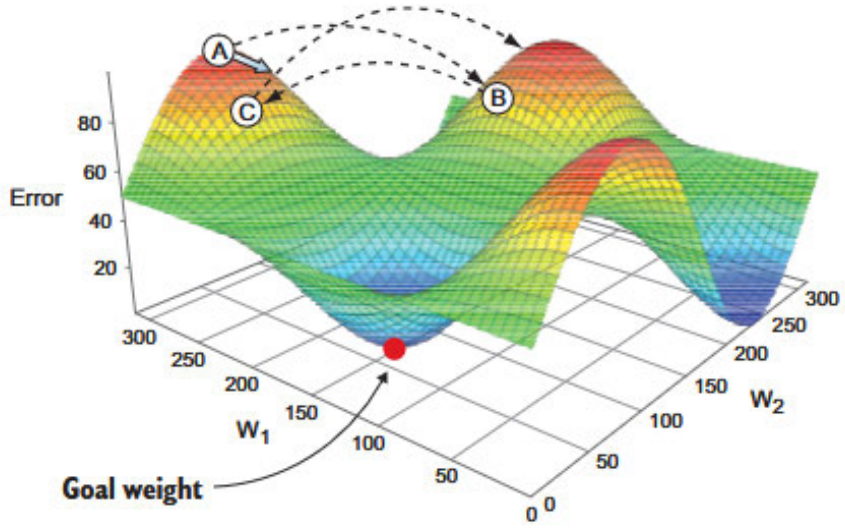


Abbildung 2.6: Grafische Darstellung einer Optimierung mit Gewichten eines neuronalen Netzes [10]

Ein wesentlicher Nachteil dieser Methode ist, dass nach jeder Epoche alle Gradienten für den gesamten Datensatz berechnet werden müssen, was sehr zeitaufwändig und rechenintensiv ist. Eine Epoche ist eine Zeiteinheit, die einen Entwicklungsschritt eines neuronalen Netzes darstellt. Die Parameter werden immer vor der Epoche berechnet und können während der Berechnung nicht angepasst werden (nach [40]).

Der *Stochastic Gradient Descent (SGD)* ist ein anderer Ansatz, der für jeden Trainingspunkt $x^{(n)}$ und das zugehörige Label $y^{(n)}$ die Parameter berechnet:

$$\mathbf{W} = \mathbf{W} - \alpha \cdot \nabla_{\mathbf{W}} E(\mathbf{W}, \mathbf{b}; x^{(n)}, y^{(n)}) \tag{2.14}$$

Da die Aktualisierungen häufiger erfolgen und die berechnete Datenmenge deutlich geringer ist als bei *Batch Gradient Descent*, ist diese Methode häufig auch schneller. Dies führt zu sehr variantenreichen Aktualisierungen, was auch in den Lernkurven sichtbar wird (nach [40]).

Das Verfahren ist in der Abbildung 2.7 dargestellt: die jeweiligen Punkte sind die zufällig gewählten Startpunkte, von denen aus ein Abstieg gesucht und dann das lokale Minimum bestimmt wird. Auf diese Weise werden mehrere Minima gefunden, von denen das kleinste lokale Minimum als globales Minimum bestimmt wird. Diese Methode stellt sicher, dass auch mehrere lokale Minima in der Auswahl enthalten sind - würde

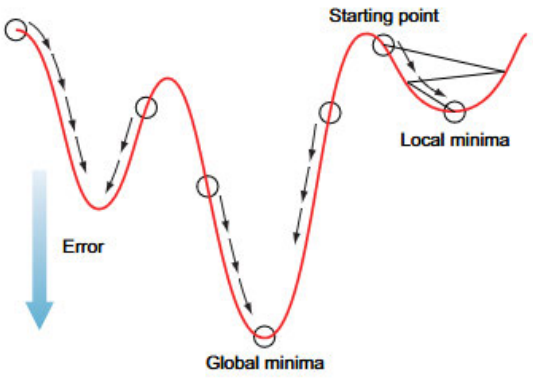


Abbildung 2.7: Optimierer *Stochastic Gradient Descent* [10]

man nur von einem Punkt ausgehen (z. B. dem rechten in der Abbildung 2.7), wäre das gefundene Minimum nur das lokale und somit ein Prozess verfälscht (nach [10, S. 83]).

Der SGD-Optimierer kann auch durch die Nestrov-Momentum-Methode verbessert werden. Diese Methode verbessert die SGD, indem sie den Gradienten in Richtung des vorherigen Schrittes anpasst. Dadurch wird sichergestellt, dass der Gradient nicht zu stark schwankt und das Netz schneller konvergiert (nach [33]).

Das *Root Mean Square Propagation (RMSprop)*-Verfahren ist ein Optimierer, der die Lernrate adaptiv anwendet und Anpassungen je nach aktuellem Zustand des Modells vornimmt.

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \tag{2.15}$$

$$E[g^2]_t = \gamma \cdot E[g^2]_{t-1} + (1 - \gamma) \cdot g_t^2 \tag{2.16}$$

Dabei ist $E[g^2]_t$ der exponentiell gleitende Mittelwert der quadrierten Gradienten zum Zeitpunkt t, $\mathbf{W}_t$ der Parameter (Gewicht oder Bias) zum Zeitpunkt t und $\mathbf{W}_{t+1}$ der Wert zum nächsten Zeitpunkt. Dabei ist g_t der Gradient der Verlustfunktion zum Zeitpunkt t, der die Richtung angibt, in der die Parameter angepasst werden müssen, um den Verlust zu minimieren. Der Term $\sqrt{E[g^2]_t}$ wird zur Normierung der Lernrate und zur Skalierung der Aktualisierungen verwendet. Der Wert $\epsilon \approx 10^{-8}$ wird hinzugefügt, um eine Division durch Null zu vermeiden. Der Glättungsfaktor γ bestimmt, wie stark die Vorwerte in die Berechnung des aktuellen Wertes eingehen (nach [40]).

Der *Adaptive Moment Estimation (Adam)* Optimierer verwendet das erste und zweite Moment des Gradienten, um die Parameter effizient zu aktualisieren. Durch die Schätzung der Varianz kann die Anpassung der Lernrate dynamisch gesteuert werden.

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \frac{\alpha}{\sqrt{v_t} + \epsilon} \cdot m_t \tag{2.17}$$

$$m_t = \frac{\beta_1 m_{t-1} + (1 - \beta_1) g_t}{1 - \beta_1^t} \tag{2.18}$$

$$v_t = \frac{\beta_2 v_{t-1} + (1 - \beta_2) g_t^2}{1 - \beta_2^t} \tag{2.19}$$

Die aktualisierten Parameter $\mathbf{W}_{t+1}$ werden aus den Momenten m_t (der Mittelwert der Gradienten) und v_t (die unzentrierte Varianz der Gradienten) berechnet, die jeweils aus den vorhergehenden Momenten berechnet werden. Der Hyperparameter (s. Kap. 2.4.6) β_1 bestimmt, wie stark frühere Gradienten gewichtet werden (typischer Wert 0,9) und β_2 bestimmt, wie stark der frühere Wert der Varianz einfließt (typischer Wert 0,999) (nach [40, 21]).

2.3.4 Scheduler

Der *Scheduler* sorgt dafür, dass die Lernrate (LR) während des Trainings des Optimierers angepasst wird. Es gibt verschiedene Methoden zur Anpassung der Lernrate, um das Modell zu verbessern. Eine Auswahl von *Scheduler*-Methoden sind:

- StepLR
- MultiStepLR
- ReduceLROnPlateau

Mit *StepLR* wird die Lernrate nach einer bestimmten Anzahl von Epochen um einen Wert reduziert. Beispielsweise kann die Lernrate alle 10 Epochen um den Wert 0,1 reduziert werden.

Bei *MultiStepLR* wird die Lernrate nach dem Erreichen von Meilensteinen reduziert, die im Voraus festgelegt werden müssen. So kann z. B. festgelegt werden, dass bei den Meilensteinen 10 und 30 die Lernrate um den Wert 0,1 reduziert wird.

ReduceLROnPlateau hingegen reduziert die Lernrate automatisch, wenn der Fehler nicht mehr sinkt. Damit kann sichergestellt werden, dass die Lernrate nur dann reduziert wird, wenn sich das Modell nicht mehr verbessert. Der Nachteil ist, dass die Lernrate erst spät reduziert wird und das Modell länger braucht, um sich zu verbessern (nach [39]).

2.3.5 Methoden der Initialisierung

Damit die Gewichte und der *Bias* im Voraus festgelegt und nicht zufällig verteilt werden, kann eine Initialisierung der Gewichte durchgeführt werden.

Gradient Vanishing ist ein Problem, das bei der Initialisierung von Gewichten in tiefen neuronalen Netzen auftreten kann. Es beschreibt den Effekt, dass die Ableitungen der Aktivierungsfunktionen in den tieferen Schichten des Netzes immer kleiner werden. Dadurch werden die Gewichte in diesen Schichten nur minimal oder gar nicht mehr aktualisiert, was das Lernen erheblich erschwert oder sogar vollständig zum Stillstand bringt (nach [13]).
Im Folgenden werden verschiedene Methoden zur Initialisierung von Gewichten vorgestellt, um dieses Problem zu vermeiden.

Ones und *Zeros* sind zwei Methoden, die ähnliche Ansätze verfolgen. Alle Gewichte werden auf 1 oder 0 gesetzt, was für einige Anwendungen geeignet sein könnte.

Die *Uniform* Initialisierung verteilt die Gewichte zufällig über eine Gleichverteilung innerhalb der Grenzen a und b (nach [37]):

$$\mathbf{W} \sim U(a, b) \tag{2.20}$$

Die *Xavier* Initialisierung versucht die Gewichte $\mathbf{W}$ so zu verteilen, dass die Forward- und Backpropagation nicht zu stark verzerrt werden. Dazu werden die Gewichte aus einer Gleichverteilung U mit folgender Ober- und Untergrenze gebildet:

$$\mathbf{W} \sim U[-\frac{\sqrt{6}}{\sqrt{f_{in} + f_{out}}}, \frac{\sqrt{6}}{\sqrt{f_{in} + f_{out}}}] \tag{2.21}$$

Dabei steht f_{in} für die Anzahl der Eingänge und f_{out} für die Anzahl der Ausgänge der Schichten. Durch diese vorgegebenen Grenzen wird sichergestellt, dass die Verteilung

nicht zu groß oder zu klein wird und sich somit jeder Schicht individuell anpasst. Dieses Verfahren eignet sich für neuronale Netze mit symmetrischen Aktivierungsfunktionen softsign und tanh (nach [31, 13]).

Für asymmetrische Aktivierungsfunktionen ist die *Kaiming* Initialisierungsmethode wie für ReLU geeignet, um die Gewichte **W** einzustellen. Bei ReLU werden negative Werte auf 0 gesetzt, so dass nur die Hälfte der Neuronen einer Schicht aktiv ist, was zu einer Asymmetrie führt. Die Eingaben der Schicht f_{in} werden betrachtet und daraus eine Normalverteilung N generiert.

$$\mathbf{W} \sim N(0, \frac{2}{f_{in}}) \tag{2.22}$$

Dieses Verfahren stellt sicher, dass die Varianz der Aktivierungsfunktionen über die Schichten konstant bleibt (nach [31, 15]).

2.4 Convolutional Neuronal Networks

Convolutional Neuronal Networks sind eine Weiterentwicklung der im Kapitel 2.3 beschriebenen neuronalen Netze, die sich besonders für die Bilderkennung eignen (nach [11]). CNNs übernehmen die Grundlagen von MLPs und bauen auf diesen auf.

Neuronale Netze verwenden als Eingangsparameter einen 1D-Eingangsvektor, der sich nur bedingt auf 2D-Bilder anwenden lässt. Daher muss ein 2D-Bild in einen 1D-Vektor umgewandelt werden, was auch als *Flattening* bezeichnet wird - das 2D-Bild wird stückweise „ausgeschnitten“ und die resultierenden Vektoren dem neuronalen Netz übergeben, das dann aus den Pixelwerten Zusammenhänge ableiten soll. Bei dieser Methode geht viel Information über das 2D-Bild verloren und es wird für ein MLP schwieriger, Strukturen zu erkennen.

CNNs hingegen benötigen keinen Vektor als Eingabe, sondern nehmen eine ganze Matrix und lernen aus den Aufnahmen. Ein MLP kann verwendet werden, um identische Strukturen zu erkennen und zu identifizieren, die gleich angeordnet sind, aber wenn eine Struktur in einem anderen Winkel liegt, versagt das MLP. Im Gegensatz dazu kann ein CNN ein Objekt besser erkennen, wenn es sich in verschiedenen Positionen und Winkeln befindet, da es die Merkmale erkennen und auch lokalisieren kann. Bei MLPs sind alle Neuronen miteinander verbunden, was dazu führt, dass die Anzahl der Verbindungen

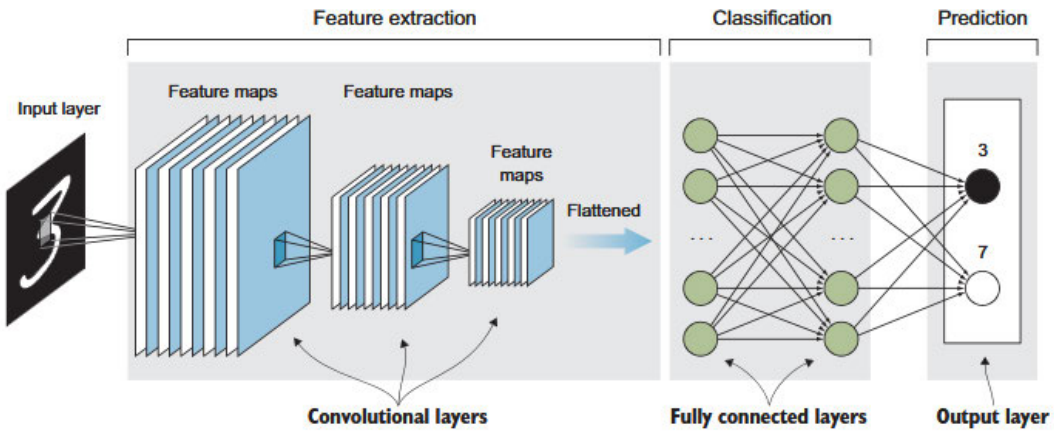


Abbildung 2.8: Übersicht einer klassischen CNN-Struktur ([10, S. 103])

exponentiell mit der Größe des Eingabebildes steigt. CNNs umgehen dieses Problem durch lokal verbundene Schichten, die nur einen Teil der Aufnahme betrachten.

CNNs setzen auf lokal verbundene Schichten, die - im Beispiel einer Aufnahme - nur einen Teil betrachten, dort nach Merkmalen suchen und ein kleines, in sich geschlossenes Netzwerk bilden. Das Ergebnis dieses Netzes wird dann als Input für das nächste Netz verwendet und so weiter. So entsteht ein CNN, das in den ersten Schichten grundlegende Strukturen wie Kanten oder Linien lernt. In der nächsten Schicht werden dann komplexere Strukturen wie geometrische Figuren erkannt, bis in den letzten Schichten komplexe Strukturen wie Gesichter, Ohren oder Fahrzeuge erkannt werden können (nach [10, S. 99ff]).

Der prinzipielle Aufbau eines CNN ist in der Abbildung 2.8 dargestellt. Es ist zu sehen, dass das Eingangsbild zunächst eine *Feature Extraction* durchläuft, die aus mehreren *Convolutional Layers* (s. Kap. 2.4.1) besteht. Hier ist zu erkennen, dass die jeweiligen *Feature Maps* (s. Kap. 2.4.2) nur einen kleinen Teil der Daten aus der vorherigen Schicht extrahieren und zur Informationsgewinnung nutzen. Das Ergebnis ist ein 1D-Vektor, der in einem *Fully Connected Layer* zur Klassifikation verwendet wird, um daraus eine Vorhersage zu generieren.

Die Hauptkomponenten eines CNNs sind also *Convolutional Layers* (Kap. 2.4.1), *Pooling Layers* (Kap. 2.4.2) und *Fully Connected Layers* (Kap. 2.4.3), die in den folgenden Unterkapiteln erläutert werden.

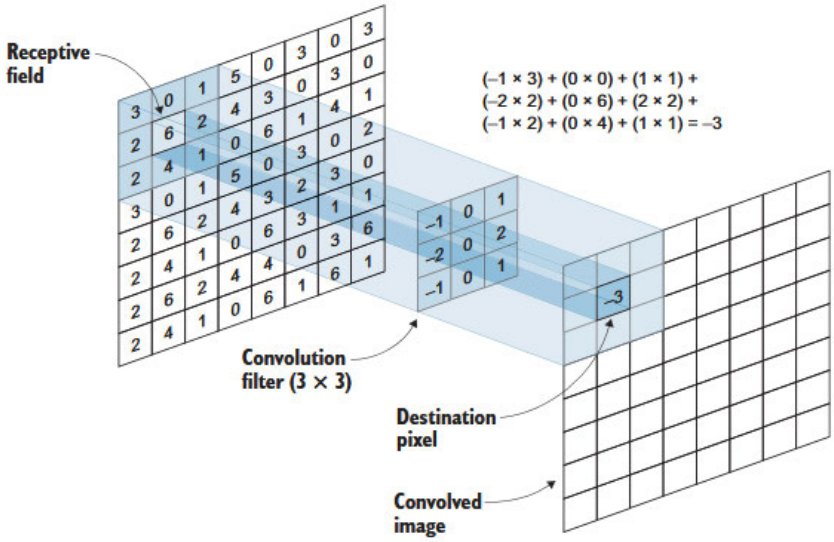


Abbildung 2.9: Darstellung eines *Convolutional Layers* ([10, S. 107])

2.4.1 Convolutional Layer

Die Abbildung 2.9 zeigt eine *Convolutional Layer*. Die linke Matrix stellt einen Bildausschnitt mit verschiedenen Pixelwerten dar. Zur Berechnung der nächsten Schicht durchläuft ein Filter den Bildausschnitt und berechnet jeweils einen neuen Wert, der dann gespeichert wird. Im Beispiel wird ein 3×3 Filter über den Bildausschnitt gelegt, der die dargestellte Berechnung durchführt. Diese jeweiligen Werte im Filter sind bei CNN die zu trainierenden **Gewichte**. Filter bestehen z. B. aus einer 3×3 Matrix und setzen sich aus Fließkommazahlen zusammen, die mit den Pixelwerten der Aufnahme multipliziert werden.

Das Ergebnis ist eine sogenannte *Feature Map*, die als Eingabe für die nächste *Convolutional Layer* dient. Die Größe der Ausgabe kann durch die Parameter *Stride* und *Padding* beeinflusst werden.

Der *Stride* gibt an, wie weit der Filter bei jedem Schritt verschoben wird. Ein *Stride* von 1 sorgt dafür, dass die Ausgabegröße der Eingabegröße ähnelt, während ein größerer *Stride* (z. B. zwei oder mehr) die Ausgabe verkleinert.

Das *Padding* legt fest, ob und wie der Rand der Eingabe mit zusätzlichen Werten (z. B. Nullen) aufgefüllt wird. Andere Methoden sind beispielsweise *Reflective Padding* oder *Replicate Padding*, die den Rand der Eingabe mit den Werten der angrenzenden Pixel

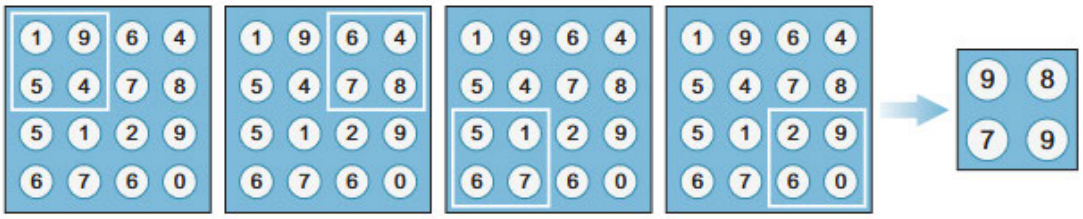


Abbildung 2.10: Darstellung eines *Pooling Layers* ([10, S. 115])

auffüllen. Dies verhindert Informationsverlust an den Rändern der Eingabe, wenn der Filter Berechnungen durchführt. Je nach Padding-Wert werden zusätzliche Zeilen oder Spalten um die Eingabe hinzugefügt (nach [10, S. 106ff], [54]).

2.4.2 Pooling Layer

Jede Schicht in den *Convolutional Layers* erhöht die Tiefe der erzeugten Ausgabe. Eine RGB-Aufnahme besteht beispielsweise aus drei Schichten, jeweils für die Farbkanäle Rot, Grün und Blau. Durch die Anwendung einer Faltungsschicht mit 4 Kernen und einem definierten *Stride* entsteht eine neue Ausgabe mit einer Dimension von $28 \times 28 \times 4$. Um die Größe dieser Ausgabe und damit die benötigte Rechenleistung zu reduzieren, werden zwischen den Faltungsschichten *Pooling Layers* eingefügt. Diese reduzieren die Dimensionen der Ausgabe, indem sie entweder den maximalen Wert (*maxPooling*) oder den Durchschnittswert (*averagePooling*) innerhalb eines definierten Fensters extrahieren.

Diese Schicht durchläuft die von der *Convolutional Layer* erzeugten Schichten und betrachtet die Werte in einem definierten Fenster und nimmt dann nur den höchsten Wert[5] und stellt diesen der nächsten Schicht zur Verfügung. In der Abbildung 2.10 ist dieser Vorgang nach dem *maxPooling* dargestellt und zeigt, dass bei dieser Methode mit einem 2×2 Fenster das Originalbild in der Dimension halbiert wird. Dadurch bleiben wichtige Merkmale der Originalschicht erhalten, aber die Aufnahme verliert an Auflösung und Größe - was auch die Komplexität des neuronalen Netzes reduziert (nach [10, S. 114ff], [54]).

[5] den höchsten Wert bei der Methode *maxPooling*, die Methode *avaragePooling* berechnet jeweils den Mittelwert

2.4.3 Fully Connected Layer

Nach einer Reihe von Kombinationen von *Convolutional Layers* und *Pooling Layers* ist das Ausgangsprodukt eine „sehr lange Matrix“ (z. B. eine $5 \times 5 \times 40$ Matrix), die dann als Eingabe für ein MLP verwendet wird, das die Klassifikation durchführt. Hier werden dann eine oder mehrere Schichten von Neuronen hinzugefügt, die schließlich bei der Klassifikation über eine *softmax*-Funktion ausgeführt werden. Der Aufbau dieser Schicht erfolgt dann entsprechend der Abbildung 2.5 (nach [10, S. 119], [54]).

Mit dem Wissen, wie ein CNN grundsätzlich aufgebaut ist, können auch verschiedene Arten von Netzwerken erstellt und getestet werden. Eine kleine Auswahl von Netzwerken wird im folgenden Kapitel 2.4.4 erläutert.

2.4.4 Netzwerkarchitekturen

Das erste CNN, das den MNIST-Datensatz erfolgreich und zuverlässig klassifizieren konnte und damit den Grundstein für dieses Thema legte, war das LeNet [4, 28]. Da dieses jedoch nur für die Auswertung der handschriftlichen Ziffern des MNIST-Datensatzes ausgelegt war und keine anderen Aufnahmetypen identifizieren konnte, wurde nach einem besseren Netzwerk gesucht.

Im Bereich der *Deep CNN Architectures* wurde dann das AlexNet [24] entwickelt, das den Vorteil hat, bessere Ergebnisse zu liefern und auch andere Arten von Eingangsdaten klassifizieren zu können. Das Netz besteht aus fünf *Convolutional Layers* mit den Kernelgrößen 11×11, 5×5 und 3×3 und jeweils einer *Pooling Layers* zur Größenbegrenzung (siehe Abbildung 2.11).

In den *Fully Connected Layers* werden die Aktivierungsfunktionen *ReLU* verwendet und anschließend mit einer *softmax*-Funktion klassifiziert. Dieses Netz hat gegenüber LeNet den Vorteil, dass die Eingangsgröße nicht festgelegt ist und trotzdem gute Ergebnisse liefert und somit auch als Vergleich für verschiedene Netze dienen kann.

AlexNet hat sich beim ILSVRC 2012 besonders gut geschlagen und mit einer Fehlerquote von 15,3 % bei fünf Versuchen deutlich besser als die anderen Teams abgeschnitten - der nächstbeste Konkurrent kam auf eine Fehlerquote von 26,17 % (nach [41, 24]).

Das *Visual Geometry Group (VGG)net* ist ein leistungsstarkes CNN, das durch seine einfache Struktur und hohe Rechenintensität besticht. Die Variante VGG16 umfasst 16

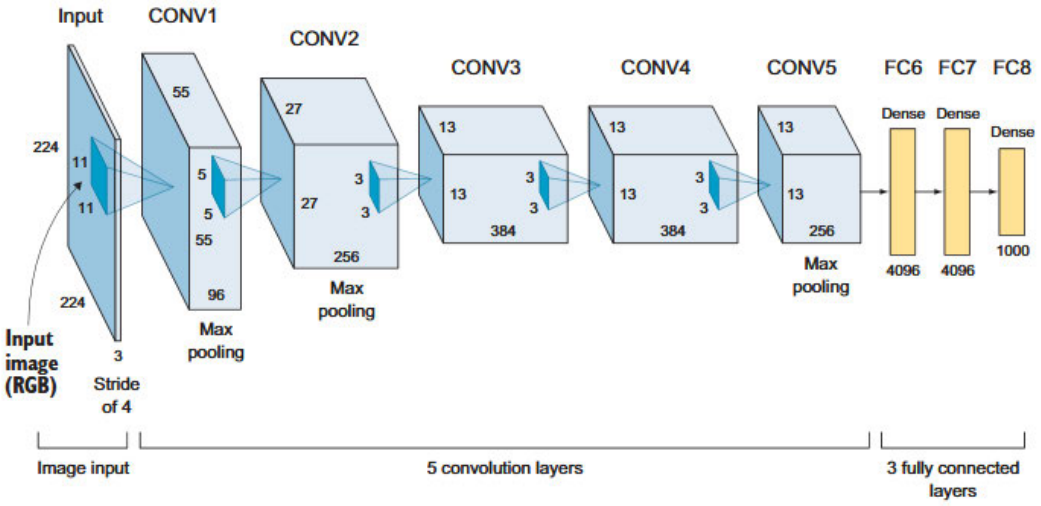


Abbildung 2.11: Darstellung der Netzwerkarchitektur AlexNet ([10, S. 204])

Schichten, bestehend aus 13 *Convolutional Layers* und 3 *Fully Connected Layers*, die in einer klaren Abfolge angeordnet sind (siehe Abbildung 2.12).

Ein wesentlicher Unterschied zwischen AlexNet und VGG16 liegt in der Verwendung kleinerer *Kernel* mit einer Größe von 3×3. Diese ermöglichen es dem VGG16, feinere Merkmale und Strukturen zu lernen. Gleichzeitig erlaubt die kleinere Kernelgröße eine größere Netzwerktiefe, wodurch das Modell detailliertere Muster erfassen kann. Zudem reduziert die kleinere Kernelgröße die Anzahl der zu berechnenden Gewichte, was die Rechenoperationen effizienter gestaltet.

Das VGG16 ist besonders geeignet für Anwendungen, die eine einfache Implementierung erfordern und dennoch eine hohe Klassifikationsgenauigkeit erzielen möchten [47].

Das VGGnet hat beim ILSVRC Wettbewerb 2014 sehr gut abgeschnitten: Das Netzwerk gewann den ersten Platz in einer Lokalisierungsaufgabe und den zweiten Platz in einer Klassifikationsaufgabe (nach [41, 47]).

2.4.5 Ablauf des Trainings

Das Training eines neuronalen Netzes kann auf zwei Arten erfolgen: Ein Training von Grund auf oder ein Training, das die Gewichte eines bereits trainierten Netzes übernimmt (*Transfer-Learning*).

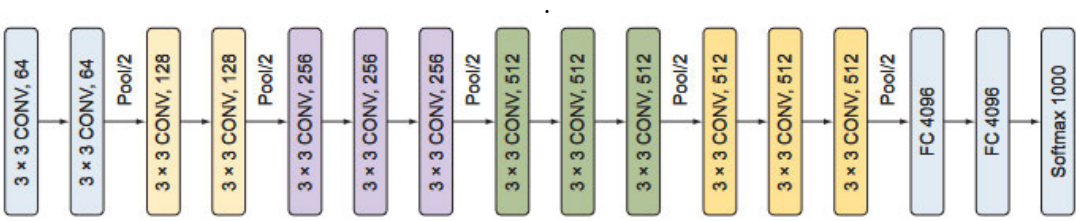


Abbildung 2.12: Darstellung der Netzwerkarchitektur VGG16 ([10, S. 213])

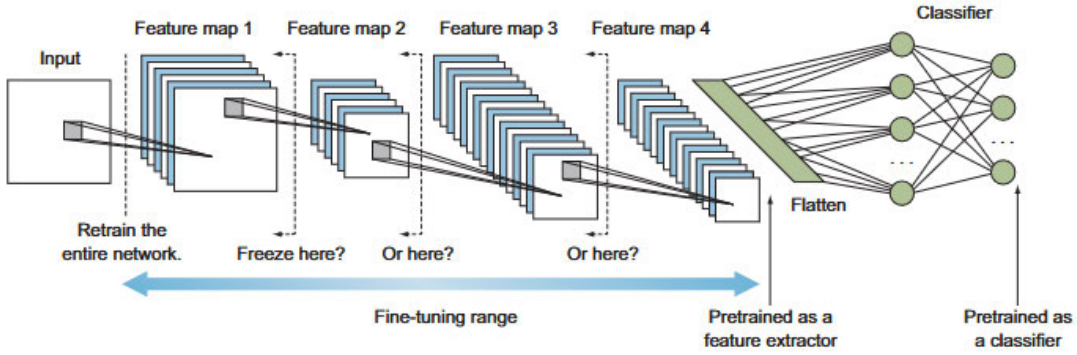


Abbildung 2.13: Darstellung von *Transfer-Learning* ([10, S. 258])

Beim *Transfer-Learning* wird der ursprüngliche Klassifikator eines vortrainierten Netzes entfernt und durch einen neuen Klassifikator ersetzt, der auf die aktuelle Aufgabe zugeschnitten ist. Abbildung 2.13 zeigt verschiedene Ansatzpunkte für die Anwendung von *Transfer-Learning*, auch bekannt als *Fine-Tuning*.

Die Abbildung illustriert, ab welcher Schicht eines neuronalen Netzes die Gewichte eingefroren werden und ab welcher Schicht neue Gewichte trainiert werden. Ein weiter links gesetzter Schnittpunkt bedeutet, dass das Netz größtenteils neu trainiert wird, ohne die vortrainierten Gewichte zu nutzen. Ein weiter rechts gesetzter Schnittpunkt hingegen erlaubt es, mehr vortrainiertes Wissen zu übernehmen, wodurch das Netz schneller und effizienter auf die neue Aufgabe angepasst werden kann. In der Regel wird ein neuer Klassifikator hinzugefügt, der speziell auf den neuen Anwendungszweck abgestimmt ist (vgl. [4] und [10, S. 254ff]).

Um ein neuronales Netz von Grund auf zu trainieren, müssen zunächst Muster und Strukturen aus Datensätzen gelernt werden. Der dafür verwendete Datensatz, der später die Grundlage für das Klassifikationsproblem bildet, wird für dieses Training aufgeteilt.

In der Vergangenheit war es üblich, den Datensatz in einen Trainingsdatensatz und einen Validierungsdatensatz aufzuteilen (nach [51]).

Der Trainingsdatensatz wird verwendet, um die Gewichte des neuronalen Netzes anzupassen, indem die Vorhersagen, die durch das Training mit den richtigen Labels erzeugt wurden, mit einer Verlustfunktion überprüft werden. Dies liefert einen Wert für die Genauigkeit und den Verlust des Netzes, anhand dessen die Gewichte angepasst werden. Um zu verhindern, dass das Netz diese Daten auswendig lernt, wird ein Validierungsdatensatz erzeugt, der nur aus Daten besteht, die nicht für das Training des Netzes verwendet wurden. Anhand dieser Metrik der resultierenden Genauigkeit und des Verlustes kann dann überprüft werden, wie gut das Netz ist. Da es auch hier vorkommen kann, dass das Netz die Daten auswendig lernt und dies zu einem *Overfitting* führen kann (siehe Kap. 2.4.6), wurde zusätzlich ein Testdatensatz als *Cross-Validation* eingeführt (nach [44]).

Für das Training wird der Datensatz dann in drei Teile aufgeteilt: Training, Validierung und Test. In der Regel ist eine Aufteilung im Verhältnis 70/15/15 denkbar, was aber wiederum von der Größe des Datensatzes selbst abhängt. Der Validierungs- und Testdatensatz werden hier klein gehalten, da das Netz mit möglichst vielen Daten trainiert werden soll.

Abbildung 2.14 illustriert den Verlauf eines Trainingsprozesses. Während des Trainings durchläuft das Modell in jeder Epoche die Schritte Training, Validierung sowie die Berechnung von Genauigkeit und Verlust. Idealerweise nähert sich die Genauigkeitskurve mit zunehmenden Epochen dem Wert 1,0 an, während die Verlustkurve gegen 0,0 konvergiert. Ein gut trainiertes Modell zeigt dabei ähnliche Fehlerkurven für Training und Validierung, die beide gegen Null streben. Dies deutet darauf hin, dass das Modell auch bei unbekannten Validierungsdaten eine vergleichbare Leistung erbringt.

Ein *Overfitting* zeigt sich daran, dass die Validierungskurve deutlich über der Trainingskurve liegt und nach einigen Epochen wieder ansteigt (siehe Abbildung 2.14, Mitte). In diesem Fall hat das Modell die Trainingsdaten auswendig gelernt und liefert bei neuen Validierungsdaten keine zufriedenstellenden Ergebnisse.

Wenn der Konvergenzwert beider Kurven deutlich über Null liegt, wird dies als *Underfitting* bezeichnet (siehe Abbildung 2.14, links). Dies deutet darauf hin, dass das Modell aufgrund unzureichender Daten oder zu geringer Komplexität nicht in der Lage

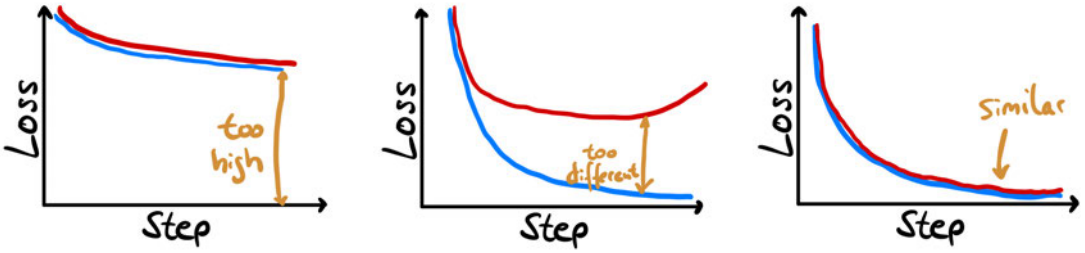


Abbildung 2.14: Darstellung der Fehlerkurve des Trainings (blau) und der Validierung (rot) für *Underfitting*, *Overfitting* und eine ideale Kurve (von links nach rechts in der Abbildung) (angepasst nach [6])

ist, Muster und Strukturen zu erkennen, die für eine korrekte Klassifikation erforderlich sind.

2.4.6 Overfitting

Wie bereits im vorherigen Unterkapitel beschrieben, kann es zu einem *Overfitting* der Netze kommen, wenn das Training nicht in der Lage ist, mit neuen Daten umzugehen. Um dies zu verhindern, gibt es verschiedene Ansätze.

Dropout ist eine Methode beim Netzaufbau, die dafür sorgt, dass in jeder Epoche immer ein zufälliges Neuron deaktiviert wird. Dies hat den Vorteil, dass sich das Netz nicht auf einzelne gelernte Strukturen und Muster verlassen kann und neue Merkmale lernen muss.

Drop-Weights ist der vorherigen Methode ähnlich, nur dass hier die Gewichte und die daraus resultierenden Verbindungen zwischen den Neuronen für eine Epoche deaktiviert werden.

Batch Normalization ist eine Methode, die die Ausgaben der Neuronen in einem *Mini-Batch* zusammenfasst und so eine Normalisierung durchführt. Dabei werden in jeder Epoche andere *Batches* zusammengefasst und daraus eine neue Normalisierung erzeugt, die einem Rauschen ähnelt. Dieses Rauschen verhindert, dass das Netz die Daten auswendig lernt. Außerdem reduziert diese Methode die Auswirkungen einer fehlerhaften Initialisierung von Gewichten und/oder Hyperparametern (nach [4]).

Data Augmentation ist eine Methode, dem *Overfitting* mit einer größeren Menge an generierten Daten entgegenzuwirken. Die verschiedenen Methoden der Data Augmentation werden im Kapitel 2.5 näher erläutert.

Das Setzen von Hyperparametern ist auch eine Methode, um *Overfitting* zu vermeiden. Hyperparameter sind Parameter, die vor einem Training eingestellt werden können[6]. Hyperparameter sind zum Beispiel: Lernrate, Anzahl an Epochen, *Batch Size* und weitere.

Im Folgenden werden zwei weitere Methoden zur Vermeidung von *Overfitting* beschrieben: *Early Stopping* und *Regularization*. *Early Stopping* ist eine Methode, bei der das Training abgebrochen wird, wenn die Genauigkeit des Validierungsdatensatzes nicht mehr zunimmt oder der Verlust nicht mehr abnimmt. Dies kann bedeuten, dass das Netz die Daten auswendig gelernt hat und keine neuen Muster und Strukturen mehr erkennt oder dass dem Netz nicht genügend Daten zur Verfügung stehen, um Muster und Strukturen zu erkennen (nach [10, S. 177], [35]).

Regularization ist eine Methode, bei der die Gewichte des Netzes durch eine zusätzliche Funktion reguliert werden. Diese Funktion wird dann in die Verlustfunktion eingebaut und sorgt dafür, dass die Gewichte nicht zu groß werden und somit das Netz nicht zu stark auf einzelne Muster und Strukturen trainiert wird. Es gibt verschiedene Arten der Regularisierung, die hier kurz erläutert werden:

- *L1-Regularization* ist eine Methode, bei der die Gewichte der Neuronen so reguliert werden, dass sie klein gehalten werden.

- *L2-Regularization* ist eine Methode, bei der die Gewichte der Neuronen so reguliert werden, dass sie nicht zu groß werden.

Der Regularisierungsterm wird in die Verlustfunktion 2.5 integriert, um die Gewichte zu regulieren. Mit λ als Regularisierungsparameter, N als Anzahl der Trainingsdaten und $\mathbf{W}$ als Gewichtsmatrix ergibt sich der neue Fehlerterm wie folgt:

$$E_{\text{neu}}(\mathbf{W}, \mathbf{b}) = E_{\text{alt}}(\mathbf{W}, \mathbf{b}) + \frac{\lambda}{2N} \sum \|\mathbf{W}\|^2 \tag{2.23}$$

[6] Nicht zu verwechseln mit Parametern, die während des Trainings vom neuronalen Netz selbst bestimmt werden.

Abbildung 2.15: Data Augmentation Methoden [48]

Durch diese Regularisierung wird das Modell vor *Overfitting* geschützt. Es ist auch möglich, verschiedene Regularisierungsmethoden zu kombinieren, um die Netzleistung weiter zu optimieren (vgl. [10, S. 177f], [7]).

2.5 Data Augmentation

Data Augmentation kann *Overfitting* verhindern, indem der Datensatz künstlich erweitert wird. Ziel ist es, den ursprünglichen Rohdatensatz zu vergrößern. In der Regel ist es nicht möglich, einen ausreichend großen Datensatz zu generieren, da es z. B. nicht möglich ist, Aufnahmen einer seltenen genetischen Mutation zu erhalten (für ein Modell, das in der medizinischen Diagnostik eingesetzt werden soll, um solche genetischen Anomalien wie z. B. seltene Hautkrankheiten zu erkennen). Daher ist die künstliche Generierung von Daten manchmal die einzige Lösung, um ein neuronales Netz mit mehr Daten zu trainieren.

Durch die Erweiterung des Datensatzes kann ein Modell robuster trainiert werden. Dabei ist es wichtig, dass die Aufnahmen das ursprüngliche Label beibehalten. Wenn z. B. Aufnahmen des Buchstabens M gemacht wurden, könnte eine Data Augmentation mit der Rotationsmethode ein W ergeben. Die Auswirkungen müssen daher bei der Wahl der Data Augmentation Methode berücksichtigt werden. Die Abbildung 2.15 zeigt eine kleine Auswahl der Methoden, die im Folgenden erläutert werden sollen.

Geometrische Transformationen sind Methoden, die die Pixelwerte des Eingabebildes verschieben, um einen neuen Datensatz zu erzeugen. Diese Methoden sind die einfachsten und stellen den Standard in der Data Augmentation dar. Beispielsweise

wurde die Methode der Spiegelung bereits bei der Entwicklung von AlexNet [24] eingesetzt.

Dabei wird die Position oder Ausrichtung einzelner oder mehrerer Pixel verändert. Dies kann bedeuten, dass die Aufnahme gedreht oder gespiegelt wird oder dass Bereiche ausgeschnitten und an anderer Stelle wieder eingefügt werden [48, 46].

Die Spiegelung (Flipping) an einer Achse ist am einfachsten zu realisieren, da hier nur die Hälfte einer Aufnahme auf den Rest der Seite kopiert wird [4].
Die Drehung (Rotating) der Aufnahme erfolgt nach folgender Formel, wobei (x, y) die Startkoordinaten und (x', y') die resultierenden Positionen sind.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.24}$$

Hier reicht ein Winkel θ von $\pm 30°$ aus, um eine neue, für ein neuronales Netz sinnvolle Aufnahme zu erzeugen, wobei auch eine Drehung von $\pm 359°$ möglich ist. Dabei muss jedoch darauf geachtet werden, dass die Aufnahme nicht verzerrt wird und nach der Rotation dem Label entspricht (nach [48, 4, 46]).

Random Cropping ist eine weitere geometrische Transformationsmethode, bei der ein zufälliger Ausschnitt aus einer Aufnahme entfernt wird. Dadurch kann ein neuronales Netz auch dann lernen, wenn das Objekt nicht vollständig sichtbar ist oder sich etwas im Sichtfeld befindet. Dies reduziert die Größe der Aufnahme, da nur der Ausschnitt für die weitere Betrachtung verwendet wird. Bei der Methode *Translation* hingegen wird die Aufnahme in eine Richtung verschoben und die durch den neuen Ausschnitt entstandene Lücke mit Nullen oder zufälligem Rauschen gefüllt [4, 46].

Noise Injection fügt dem Eingangsbild eine Matrix aus Zufallswerten einer Normalverteilung hinzu. Durch das Hinzufügen von Rauschen kann das Netz robuster trainiert werden [46, 32].

Edge Enhancement hebt das zu klassifizierende Objekt in einer Aufnahme durch Hervorhebung der Konturen hervor. Dazu wird zunächst ein Kantenfilter auf das Eingangsbild angewendet, die Aufnahme in Graustufen umgewandelt, invertiert und anschließend die Maske über das Eingangsbild gelegt [4, 46].

Photometrische Methoden sind Methoden, die die Eigenschaften der Farbkanäle verändern.

Color Space Transformation kann verwendet werden, um nur einen Farbkanal zu isolieren oder die einzelnen Farbkanäle so zu manipulieren, dass sich die Helligkeit ändert. Dazu können auch die Werte für die Farbintensität oder den Farbraum über ein Histogramm eingestellt werden (nach [4, 46]).

Coller Jittering ist eine Methode in diesem Bereich, die dafür sorgt, dass die Aufnahme in einer zufälligen Farbe oder durch einen bestimmten Filter neu eingefärbt wird [4, 46].

Fancy PCA (Principal Component Analysis) ist eine Transformation zur gezielten Erhöhung der Farbintensität bei unterschiedlichen Lichtverhältnissen und Farbvariationen. Dabei werden die RGB-Kanäle analysiert, um neue, leicht veränderte Versionen einer Aufnahme zu erzeugen.

Zusammenfassend kann gesagt werden, dass die genannten Methoden der Data Augmentation für sich genommen nur einen geringen Einfluss auf das Training eines neuronalen Netzes und damit auf die Erweiterung des Datensatzes haben. Am besten ist es jedoch, mehrere oder alle Methoden zur Anreicherung des Datensatzes zu verwenden, solange die Transformationen das Label nicht verfälschen.

2.6 Leistungsmetriken

Um neuronale Netze zu vergleichen, ist es notwendig, ihre Leistung anhand geeigneter Metriken zu bewerten. Diese Metriken ermöglichen es, den Klassifikator mit der besten Leistung auszuwählen, insbesondere wenn mehrere Modelle trainiert wurden.

Die Bewertung erfolgt in zwei Schritten: Zunächst wird die Leistung des Modells anhand der Trainingsdaten gemessen. Diese Daten dienen dazu, die Genauigkeit und den Verlust während des Trainings zu bewerten und sicherzustellen, dass das Modell die bekannten Daten korrekt verarbeitet.

Im zweiten Schritt wird die tatsächliche Leistungsfähigkeit des Modells mit Testdaten überprüft. Diese Testdaten sind dem Modell unbekannt und dienen dazu, die Fähigkeit des Modells zu bewerten, neue, bisher nicht gesehene Daten korrekt zu klassifizieren. Die Vorhersagen des Modells werden anschließend anhand der Testdaten analysiert und bewertet.

Es gibt vier grundlegende Werte: *True Positive* (TP) und *True Negative* (TN), die angeben, wie viele Daten korrekt als wahr bzw. falsch erkannt wurden. Das Gegenpaar bilden die Werte *False Positive* (FP) und *False Negative* (FN), die angeben, wie viele Daten fälschlicherweise den Kategorien Wahr und Falsch zugeordnet wurden.

Aus diesen Werten kann die Genauigkeit ermittelt werden, die den Anteil der richtig klassifizierten Daten angibt.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.25}$$

Der *Recall*-Wert gibt an, wie gut das Modell tatsächlich korrekte Vorhersagen trifft.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.26}$$

Der Präzisionswert - *Precision* - gibt an, wie viele der als positiv klassifizierten Fälle auch richtig sind.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.27}$$

Aus den genannten Metriken kann der $F1_{Score}$ berechnet werden, der eine Zusammenfassung der beiden Metriken darstellt und je näher er bei 1,0 liegt, desto besser ist das Modell (nach [4]).

$$F1_{score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.28}$$

Die *Confusion* Matrix stellt diese Metriken TP, TN, FP und FN in einer Matrix dar, die die Leistung eines Modells visuell darstellt. Die Abbildung 2.16 zeigt ein Beispiel für einen sehr guten Fall einer solchen Matrix. Sie ist das Ergebnis eines neuronalen Netzes, das fünf Vorhersagen treffen musste und in fast allen Fällen die richtige Klasse vorhersagte. Eine Diagonale zeigt, dass das Modell den Testdatensatz perfekt vorhergesagt hat und daher das Ziel sein sollte. Im gezeigten Beispiel wurde die Klasse 8 einmal fälschlicherweise als Klasse 7 erkannt (nach [10, S. 147]).

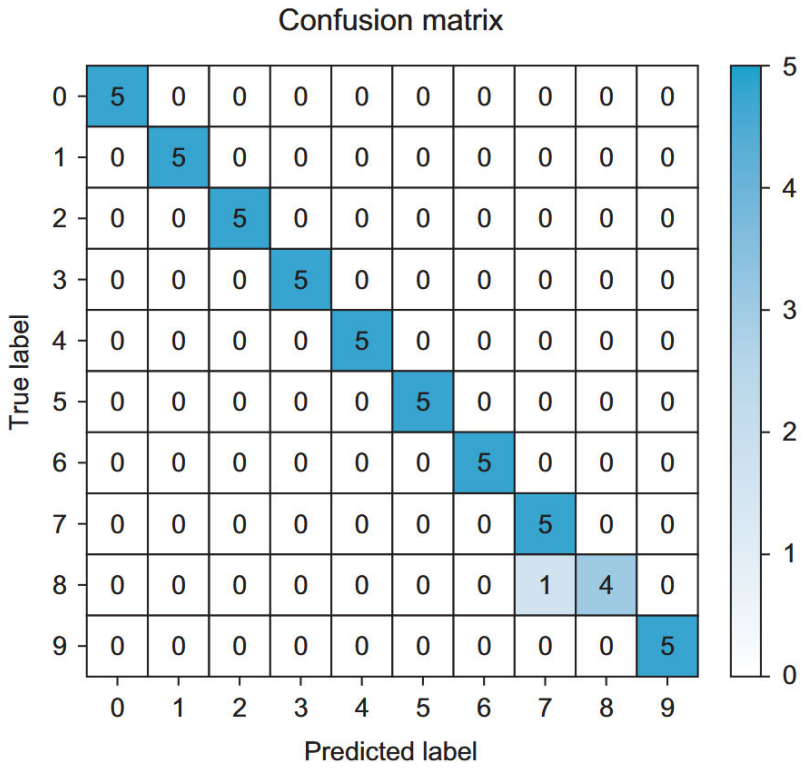


Abbildung 2.16: Beispiel einer *Confusion* Matrix mit 9 Klassen [10, S. 281]

3 Stand der Technik

Dieses Kapitel untersucht den aktuellen Stand der Technik mit Beispielen aus dem Bereich der Data Augmentation. Zuerst werden Beispiele untersucht, die sich auf die Kamera und die daraus resultierende Data Augmentation beziehen und den Vorteil gegenüber keiner Data Augmentation behandeln.

Im Anschluss werden andere Ansätze betrachtet, die der Data Augmentation über Kameraparameter ähneln, aber in andere Richtungen gehen. Die Forschung in diesem Bereich hat sich bisher auf die Erzeugung neuer Datensätze durch künstliche Bilderzeugung konzentriert. Einige Arbeiten befassen sich mit der Hardware, d. h. der Kamera selbst, um unterschiedliche Kameras bei der Klassifikation berücksichtigen zu können. Beispiele für digital erzeugte Umgebungen zur Generierung von Datensätzen werden ebenfalls untersucht.

3.1 Ansätze bei der Data Augmentation

Die Methoden der Data Augmentation sind weitreichender als im Kapitel 2.5 dargestellt und Teil der Forschung. In diesem Unterkapitel sollen die Forschungsarbeiten in dieser Richtung und die damit verbundenen Ergebnisse vorgestellt werden.

3.1.1 Verwendung unterschiedlicher Hardware

Das Gebiet der Gesichtserkennung ist in einigen Teilen der Welt ein aktuelles Forschungsthema und soll hier als Beispiel für die verwendete Hardware dienen. Gesichtserkennung ist ein Teilgebiet der Bildverarbeitung, das sich mit der Erkennung und Identifizierung von Gesichtern in Bildern oder Videos beschäftigt. Ein großes Problem bei der Gesichtserkennung ist die Verwendung unterschiedlicher Kameras. Es wird vermutet,

dass das Training neuronaler Netze mit unterschiedlicher Hardware zu unterschiedlichen Ergebnissen führt. Laut [36] gibt es Farbunterschiede zwischen Kameramodellen, die dazu führen, dass ein neuronales Netz mit den Pixelwerten nur einer Kamera lernt und dann mit anderen Kamerasensoren nicht so gut zurechtkommt.

Eine Anwendung ist die Personenerkennung: Es werden verschiedene Überwachungskameras von verschiedenen Herstellern und damit auch Kamerasensoren installiert und anhand der Daten einer Kamera soll schließlich eine Anwendung so trainiert werden, dass eine Person überall erkannt werden kann. In der wissenschaftlichen Arbeit [36] wird untersucht, ob das Farbprofil einer Kamera auf eine andere übertragen werden kann, um neuronale Netze robuster gegenüber unterschiedlicher Hardware zu machen. Dabei wird eine Art Data Augmentation von *Color Jittering* (vgl. Kap. 2.5) verwendet, welche in diesem Fall statt einer zufälligen Einfärbung nach einer Farbmatrix geschieht. Eine wichtige Erkenntnis aus dieser Arbeit ist, dass zwei Bilder von zwei Kameras auf den ersten Blick identisch aussehen, sich aber in den RGB-Werten stark unterscheiden können.

Für das Experiment wurden verschiedene Handykameras verwendet, die Bilder von einem gedruckten Farbmuster machten. Anschließend wurden diese Bilder analysiert und eine Farbmatrix erstellt, mit deren Hilfe die Kamerabilder farblich auf eine andere Hardware exportiert wurden. Dieser erzeugte Datensatz, einschließlich einer Erweiterung mit Aufnahmen unter verschiedenen Lichtverhältnissen, wurde dann verwendet, um ein neuronales Netz zu trainieren.

Das Experiment zeigte, dass der auf eine andere Hardware portierte Datensatz in allen Vergleichsmetriken besser abschnitt als ein Datensatz ohne diese Portierung der Farbwerte. Man kann also sagen, dass unterschiedliche Farbwerte bereits einen größeren Einfluss auf das Training eines neuronalen Netzes haben. Damit sollte man *Color Jittering* auf jeden Fall bei der Erstellung eines Datensatzes im Sinne der Data Augmentation verwenden. Sinnvoller wäre es, verschiedene Kameras zu betrachten, aber schon eine kleine Abweichung der Farben scheint ein Netz robuster zu machen.

3.1.2 Einfluss von automatischen Kameraparametern

Eine weitere Arbeit [55] beschäftigt sich mit den Auswirkungen automatisch generierter Einstellungen von Kameraparametern. Dabei wird zunächst untersucht, welchen Einfluss diese automatischen Einstellungen auf die Objektdetektion haben.

Dazu wird ein zuvor trainierter Algorithmus übernommen und auf einen neuen Datensatz angewendet. Untersucht werden die Kameraparameter Auslösezeit, Spannungsverstärkung (im Zusammenhang mit dem ISO-Wert[1] und dem analogen *Gain*-Wert) und die Umgebungsvariable Belichtung (vgl. Kap. 2.1). Ein größeres Problem stellen Datensätze, wie das ImageNet, MS-COCO oder Pascal-VOC aus dem Internet dar, die in der Regel mit automatischen Kameraeinstellungen erstellt wurden (vgl. Kap. 2.2.2). Dadurch entsteht ein sogenannter sensorspezifischer Bias, der das Training beeinflussen kann und dafür sorgt, dass ein so trainiertes Netz Probleme mit realen Aufnahmen bekommt. Die zugehörige Abbildung A.1 im Anhang zeigt ein Beispiel aus der Arbeit, das zeigt, dass ein Objekterkennungsalgorithmus mit automatischen Einstellungen die Flasche bei den Lichtverhältnissen nicht erkennen kann. Mit dem in der Arbeit entwickelten Algorithmus zur Einstellung von ISO-Wert und Belichtungszeit wird ein deutlich besseres Ergebnis erzielt. Die dargestellten Ergebnisse zeigen, dass die automatische Einstellung dieser Parameter zu schlechteren Ergebnissen führt als die selbst erstellte Methode zur Einstellung dieser Parameter.

3.1.3 Data Augmentation durch Simulation

Eine weitere Idee zur Data Augmentation ist die Verwendung von CAD-Modellen, die dann auch zur Generierung von Datensätzen verwendet werden. In der wissenschaftlichen Arbeit von [43] wird untersucht, welchen Einfluss synthetisch erzeugte Daten bei der Entwicklung von neuronalen Netzen haben. Dabei wurde untersucht, welche Umgebungsvariablen welchen Einfluss auf die Ergebnisse haben. Ziel war es herauszufinden, ob Fehler bei der Herstellung von Kunststoff- oder Spritzgussteilen, sogenannte *Flashes* beim Gießen des Rohmaterials, durch ein neuronales Netz binär erkannt werden können.

Bei der Herstellung von Spritzgussteilen ist das Trainieren eines Modells mit realen Testobjekten relativ aufwendig, weshalb die Erstellung über ein CAD-Programm gewählt wurde. Es wurde festgestellt, dass die Ausrichtung der Kamera in verschiedenen Orientierungen einen Vorteil gegenüber der statischen Ausrichtung von oben hat. Aufnahmen mit Rauschen zeigten ebenfalls ein deutlich robusteres Netz. Darüber hinaus führte eine Variation der Lichtverhältnisse zu einer signifikanten Verbesserung des Modells.

[1] Der ISO-Wert ist ein Wert, welcher die Lichtempfindlichkeit des Sensors angibt.

Zusammenfassend kann zu dieser Arbeit gesagt werden, dass hier keine *klassische* Data Augmentation direkt angewendet wurde, sondern in der virtuellen Umgebung. Diese Methode scheint speziell für diese Art von Projekten in der Spritzgussverarbeitung sinnvoll zu sein, da sie Kosten sparen kann, jedoch ist die angegebene Dauer der Datengenerierung (4,5 Tage für die Generierung von 20.000 Datensätzen) für Anwendungen nach dieser Ausarbeitung nicht sinnvoll. Dennoch kann gesagt werden, dass der in dieser Arbeit verfolgte Ansatz dem von [43] ähnelt, da ein Datensatz durch verschiedene Arten der Data Augmentation - virtuell und einmal direkt über Kameraparameter - erzeugt wird. Es wird auch gezeigt, dass Verbesserungen zu erwarten sind, wenn die Umgebungsvariablen Kameraposition, Licht oder Rauschen hinzugefügt werden.

3.1.4 Erweiterung des Datensatzes durch künstlich erzeugte Bilder

Eine weitere Arbeit [18] beschäftigt sich mit der künstlichen Erweiterung von Datensätzen durch die Nachbildung von Szenen mittels einer Spiele-Engine. Dabei wird das Problem adressiert, dass z. B. bei der Erkennung von Fahrspurmarkierungen bei Kraftfahrzeugen aufgrund eines Bias ein Problem mit den gelernten Mustern besteht.

Die Publikation [50] zeigt, dass große Datensätze aus dem Internet alle einen Bias haben und ein Modell, das nur einen Datensatz kennt, einen anderen Datensatz möglicherweise nicht klassifizieren kann. Ein Grund dafür ist, dass ein Datensatz wie *Caltech-101* eine sehr schlechte Grundlage für eine allgemeine Klassifikation darstellt, da alle Bilder sehr offensichtlich gestaltet sind. Die Autoren weisen darauf hin, dass z. B. Autos von Datensatz zu Datensatz unterschiedlich dargestellt werden (*Caltech-101* zeigt Autos eher von der Seite, *ImageNet* zeigt eher Rennwagen und im *LabelMe* Datensatz zeigen Bilder von Autos eher ein kleines Objekt im Bild). Wenn also ein Modell mehrere Datensätze gemeinsam klassifizieren soll, wird das Ergebnis in der Regel schlechter.

Daher ist es auch schwieriger, einen Datensatz von Fahrbahnbegrenzungen zu erzeugen, der im Prinzip die ganze Welt abdecken soll (damit das Modell sehr robust ist). Aus diesem Grund wird in der Ausarbeitung nach [18] eine Spiele-Engine zur Generierung von Datensätzen verwendet und diese dann mit den Ausgaben eines *Generative Adversarial Network (GAN)* [14] fusioniert.

Ein *GAN* erzeugt zusätzlich zum Training neue künstliche Bilder und versucht den Klassifikator davon zu überzeugen, dass das erzeugte Bild echt ist. Ein Problem mit der Nutzung von *GAN* Aufnahmen ist es, dass diese Artifakte aufweisen und ein Rauschen

zu erkennen ist - die Aufnahme kann deutlich unterschieden werden von einer echten Aufnahme und fallen dem Netz auf. Aus diesem Grund wurden die Aufnahmen mit einer Spiele-Engine gemacht und anschließend mit einem *GAN* realistisch nachgebildet. Um einen Vergleich darstellen zu können, wurde in diesem Experiment nicht auf eine Vielzahl an Daten gesetzt, sondern auf die Verwendung der synthetischen Daten, um den Bias nachzuweisen. Die Ausarbeitung zeigt, dass eine Verbesserung des $F1_{Score}$ erreicht werden kann, wenn die realen Daten eines Datensatzes durch synthetische Daten ersetzt werden.

Im Vergleich zu dieser Ausarbeitung wird dort eine Data Augmentation angewendet, um einen Bias zu minimieren, indem die ursprünglichen Aufnahmen simuliert und dem Datensatz hinzugefügt werden. Die Größe des Trainingsdatensatzes bleibt dabei unverändert. Im Gegensatz zur Data Augmentation mit den Kameraparametern kann man den Vergleich setzen, dass auch durch die Aufnahme über eine Kamera eine Art Bias entstehen kann. Unterschiedliche Kameras machen unterschiedliche Aufnahmen und somit haben diese Bilder auch unterschiedliche RGB-Werte.

Die Ausarbeitung [18] zeigt, dass dieser Bias minimiert werden kann und die Art der Data Augmentation dazu führt, dass das Modell besser klassifizieren kann. Daher kann davon ausgegangen werden, dass eine Data Augmentation mit unterschiedlichen Kameraparametern eine Auswirkung auf die Leistung eines neuronalen Netzes haben wird, im Gegensatz zur üblichen Methode der Data Augmentation. Bei dieser Methode werden alle Aufnahmen auf die gleiche Weise verändert, aber diese Parameter können die tatsächliche Aufnahme nicht beeinflussen.

3.2 Data Augmentation Beispiele

In diesem Abschnitt werden Beispiele von wissenschaftlichen Arbeiten betrachtet, die sich damit beschäftigen, inwieweit Data Augmentation Vorteile gegenüber keiner Data Augmentation bietet.

Tabelle 3.1: Ergebnisse aus den Data Augmentation Versuchen von Taylor und Nitschke mit dem Datensatz Caltech101 [48]

Methode	**Top-1 Genauigkeit (%)**	**Top-5 Genauigkeit (%)**
Baseline	48,13 ± 0,42	64,50 ± 0,65
Flipping	49,73 ± 1,13	67,36 ± 1,38
Rotating	50,80 ± 0,63	69,41 ± 0,48
Cropping	**61,95 ± 1,01**	**79,10 ± 0,80**
Color Jittering	49,57 ± 0,53	67,18 ± 0,42
Edge Enhancement	49,29 ± 1,16	66,49 ± 0,84
Fancy *PCA*	49,41 ± 0,84	67,54 ± 1,01

3.2.1 Unterschiedliche Arten von Data Augmentation im Vergleich

Eine wissenschaftliche Arbeit [48], die sich mit verschiedenen Methoden der Data Augmentation und deren Auswirkungen auf die Trainingsgenauigkeit beschäftigt, kam unter anderem zu den in der Tabelle 3.1 dargestellten Ergebnissen.

Diese Methoden sind auch im Kapitel 2.5 beschrieben. In der Ausarbeitung wurden populäre Data Augmentation Methoden (vgl. Abbildung 2.15) auf den Caltech101 Datensatz angewendet und dieser somit erweitert. Mit diesen Daten wurde ein angepasstes CNN gemäß der Ausarbeitung [56] trainiert und anschließend evaluiert.

Die Top-1-Genauigkeit gibt an, zu welchem Anteil das Modell mit der gegebenen Datenanreicherung die richtige Klasse vorhergesagt hat. Dabei wird jeweils die größte Übereinstimmung aller Klassen betrachtet. Die Top-5-Genauigkeit prüft, ob die richtige Klasse unter den fünf Klassen mit der höchsten Wahrscheinlichkeit des Modells enthalten ist.

In der Ausarbeitung wurde nur eine Auswahl der in diesem Kapitel beschriebenen Methoden verwendet. Es zeigt sich, dass die Methode des *Cropping* die größte Verbesserung gegenüber dem Rohdatensatz erzielt.

3.2.2 Erkennung von Pflanzenkrankheiten

Die wissenschaftliche Ausarbeitung [5] vergleicht die Verwendung von Data Augmentation bei der Analyse von Pflanzenkrankheiten. Es wird gezeigt, dass es schwieriger ist, Daten über solche Krankheiten zu sammeln, als gesunde Exemplare zu finden. Dies hat zur Folge, dass in großen Datenbanken grundsätzlich mehr Aufnahmen von gesunden

Pflanzen vorhanden sind und somit ein Ungleichgewicht entsteht, welches ausgeglichen werden soll (ein Beispiel ist im Anhang in der Abbildung A.2a zu sehen). Data Augmentation wird eingesetzt, um das Ungleichgewicht im Datensatz auszugleichen und Landwirte bei der automatisierten Erkennung von erkrankten Pflanzen zu unterstützen.

Die Data Augmentation Methoden (vgl. Kap. 2.5), die hier verwendet wurden, sind *Image Flipping*, *Cropping*, *Rotation*, *Color Transformation* (siehe Anhang A.2b) und *PCA* (ab hier „normale Data Augmentation“ genannt). Zusätzlich wurden Deep Convolutional Generative Adversarial Network (DCGAN) untersucht, die durch *Unsupervised Learning* Bilder erzeugen können. Diese Methode scheint den *GAN* überlegen zu sein, was zu einer Weiterentwicklung der *Wasserstein GAN* führte. Dabei wird statt einer *sigmoid* eine lineare Aktivierungsfunktion am Ausgang verwendet (siehe Anhang A.2c). Eine weitere untersuchte Methode ist der Neural Style Transfer (NST), der aus zwei Eingabebildern ein neues Ausgabebild erzeugt. Dabei wird der Stil aus dem ersten Bild extrahiert und das zweite Bild modifiziert, so dass eine neue Version entsteht (siehe Anhang A.2d).

Die generierten Datensätze wurden einzeln betrachtet und ihre Leistung mit den neuronalen Netzen VGG16, ResNet und InceptionV3 mittels *Transfer Learning* trainiert und bewertet. Das Ergebnis der Arbeit [5] zeigt, dass ein Datensatz, der aus einem ausgewogenen Set von gesunden und ungesunden Pflanzen besteht, besser abschneidet als ein unausgewogenes Set mit weniger ungesunden Beispielen. Es folgt, dass die Generierung von Daten durch *DCGAN* und *Wasserstein GAN* am besten abschneidet, gefolgt von *NST* und dann der normalen Data Augmentation. Ein weiterer Versuch zeigte, dass eine Kombination aller Methoden die besten Ergebnisse lieferte.

Für die vorliegende Ausarbeitung wird mitgenommen, dass die normale Data Augmentation Methode eine Verbesserung der Genauigkeit mit sich bringt.

3.2.3 Erkennung elektrischer Komponenten im industriellen Umfeld

Die Ausarbeitung [16] beschreibt einen Prozess für eine industrielle Anwendung, um eine Anwendung für einen Klassifikator zu erstellen. Der Anwendungsfall ist hier eine Art Förderband, an dem eine Kamera fest installiert ist und somit auch eine feste Position hat. Es soll ein Prozess entworfen werden, wie eine Datenbasis für diese Anwendung anhand einer Klassifikation von elektrischen Bauteilen erstellt werden kann.

Anschließend wird der Datensatz mit vortrainierten Netzen und einem selbst erstellten Netz ausgewertet.

Für die Bildaufnahme wurde das Modell Alvium 1800 U-050 gewählt, da die Bilder einfach über USB auf den Computer übertragen werden können. Außerdem liefert dieses Modell sehr detaillierte Bilder, die besonders für den industriellen Einsatz geeignet sind.

Als Anwendungsbeispiel wurde der Herstellungsprozess von Funkkommunikationsgeräten gewählt, wobei verschiedene elektrische Komponenten und ein Hintergrund ohne Komponenten aufgenommen wurden. Für die Anwendung wurde außerdem festgelegt, dass sich die Kamera 80 cm über dem Objekt befinden und unter konstanten Lichtbedingungen arbeiten muss. Außerdem sollten unterschiedliche Kamerahöhen und Helligkeiten berücksichtigt werden, da sich die Kamera während der Installation bewegen kann und dadurch andere Lichtverhältnisse entstehen können. Aus diesem Grund wurden die Kamerahöhen und Lichtverhältnisse variiert und jeweils in einem anderen Datensatz aufgenommen.

Zuerst wurde ein Benchmark-Modell erstellt, um zu sehen, ob der Datensatz ausreichend ist, um mit einem neuronalen Netz gute Ergebnisse zu erzielen. Ist dies der Fall, können weitere Netze entworfen werden. Für den Versuch wurde zunächst ein eigenes Netz entwickelt, welches mit Hilfe von zufälligen Data Augmentation Methoden trainiert wurde. Dabei erzielte das Netz, das nur mit Data Augmentation trainiert worden war, eine Genauigkeit von 96,5%. Gegenüber dem gleichen Netz, welches ohne Data Augmentation, aber unter anderem mit *Dropout* trainiert wurde, wurde nur eine Genauigkeit von maximal 88% erzielt.

Bei den vortrainierten Netzwerken wurde der Klassifikator durch einen eigenen ersetzt und dieser dann trainiert. Dabei ist herausgekommen, dass Data Augmentation in den Fällen keine wirkliche Verbesserung darstellt und die Genauigkeit der Netze dadurch nur minimal verbessert oder sogar verschlechtert wird. Es ist jedoch anzumerken, dass die Autoren in dem Versuch mit Data Augmentation den bestehenden Datensatz teilweise ersetzt und nicht erweitert haben[2].

Diese Arbeit zeigt, wie mit einem selbst erstellten Datensatz eine industrielle Anwendung erstellt werden kann. Andere Arbeiten beschäftigen sich nur mit der Erstellung

[2] Nach [48] sind mehrere Daten im Datensatz durch Data Augmentation vorteilhaft und sorgen für bessere Ergebnisse.

und dem Training eines Netzes und verwenden Daten aus fertigen Datensätzen, wie z. B. ImageNet. Eine veränderbare Kameraposition und konfigurierbare Umgebungsbedingungen sind bei einer solchen Erstellung hilfreich, um das Netz robuster zu machen und reale Bedingungen zu berücksichtigen. Vor der Erstellung eines komplexeren Netzes sollte zunächst geprüft werden, ob der Datensatz ausreichend ist.

Data Augmentation trägt dazu bei, die Robustheit eines neuronalen Netzes zu erhöhen. Diese Arbeit zeigt jedoch, dass Data Augmentation nur einen geringen Einfluss auf die Ergebnisse vortrainierter Netze hat. Dies lässt sich damit erklären, dass solche Netze bereits während ihres ursprünglichen Trainings umfangreiche Data Augmentation durchlaufen haben. Eine zusätzliche Anwendung dieser Methode bringt daher nur begrenzte Verbesserungen. Dennoch erreichen vortrainierte Netze, wie beispielsweise ResNet50, oft die höchste Genauigkeit, wobei ResNet50 in diesem Fall eine Genauigkeit von 98,99% erzielt.

Aus der wissenschaftlichen Ausarbeitung [16] stellt sich somit für diese Ausarbeitung heraus, dass vortrainierte Netze für den Versuch nicht geeignet sind, da bei ihnen der Effekt der Data Augmentation nur schwer nachzuweisen ist. Die Vorgehensweise zur Erzeugung des Datensatzes, wie die Verwendung unterschiedlicher Lichtverhältnisse und verschiedener Kamerapositionen[3], wird daher übernommen.

[3] Die Kamerapositionen können auch durch einen Drehteller realisiert werden, mehr dazu im Kapitel 5.1.4.

4 Anforderungsanalyse

Es soll ein System entwickelt werden, welches die Wirksamkeit der Data Augmentation durch einen Sequencer im Vergleich zur softwarebasierten Lösung darstellen kann. Die Anforderungsanalyse befasst sich mit den Anforderungen an das in dieser Arbeit zu entwickelnde System. Dabei wird auf die Stakeholderanalyse, die Anwendungsfälle und die Anforderungen an das System eingegangen.

4.1 Stakeholderanalyse

Stakeholder sind Personengruppen, die von dieser Arbeit profitieren können, sowie Normen und Gesetze, die bei der Entwicklung von Anforderungen berücksichtigt werden müssen. In diesem Abschnitt sollen die folgenden Personengruppen, die für diese Anwendung von Interesse sind, vorgestellt und ihre Motivation erläutert werden: Auftraggeber, Anwender, Entwickler und die Forschenden.

4.1.1 Auftraggeber

Auftraggeber ist die Firma Allied Vision, die Kameras für industrielle Anwendungen herstellt. Hier ist die Forschungsabteilung an dem System interessiert, da diese Sequencer-Funktion angeboten wird und als marktfähige Lösung für das Unternehmen interessant ist. Der Auftraggeber erhofft sich durch den Einsatz eines Sequencers im maschinellen Lernen Vorteile wie eine schnellere Datenerfassung mit einem daraus resultierenden robusteren neuronalen Netz für industrielle Anwendungen.

Allied Vision wird durch die Ergebnisse dieser Arbeit zunächst Fachwissen über die Erstellung von Datensätzen für neuronale Netze mit einer Kamera mit Sequencer-Funktio-nalität erwerben. Dieses Wissen kann dann für weitere Entwicklungen und Spezifikationen in Richtung Sequencer genutzt werden, so dass in Zukunft ein neues

Produkt in dieser Richtung entstehen kann oder das bestehende Produkt besser vermarktet werden kann. Durch die abschließende Analyse der Kameraparameter in dem Versuch dieser Arbeit wird auch ein Wissen darüber gewonnen, welche Parameter für Anwendungen in Frage kommen und welche noch fehlen, damit ein solches Produkt weiterentwickelt werden kann.

4.1.2 Anwender

Dieses Dokument ist eine wertvolle Entscheidungshilfe für Anwender, die selbst einen Datensatz für eine spezifische Anwendung erstellen wollen. Sie hilft zu beurteilen, ob der Einsatz einer Kamera mit Sequencer-Funktion sinnvoll und vorteilhaft ist. Für die Erstellung eines Datensatzes mit direkter Data Augmentation kann es von Vorteil sein, wenn diese Methode Zeit und Ressourcen sparen kann. Darüber hinaus kann der Anwender durch diese Ausarbeitung einen tieferen Einblick in die Konfiguration des Sequencers erhalten und eine ähnliche Anwendung entwickeln. Es wird auch erklärt, wie die Vorverarbeitung der Daten durchgeführt wird und wie diese schließlich zum Training eines neuronalen Netzes verwendet werden können. Mit dem Wissen aus dieser Ausarbeitung kann der Anwender selbst entscheiden, ob er sich für eine Kamera mit Sequencer-Funktion entscheidet und welche Vor- und Nachteile diese Wahl mit sich bringt.

4.1.3 Entwickler

Für den Entwickler dieses Systems ist der Wissensgewinn über neuronale Netze von großer Bedeutung. Die Anwendung der Application Programming Interface (API) der Kamera, die Konfiguration des Sequencers und die daraus resultierende Generierung eines Datensatzes stellt einen großen Wissensgewinn dar. So lernt der Entwickler die Grundlagen der Ansteuerung einer Kamera für den industriellen Einsatz und damit die Vorgehensweise bei der Nutzung von APIs. Der Entwickler lernt, wie ein neuronales Netz aufgebaut ist, welche mathematischen Zusammenhänge bestehen, wie ein Training abläuft und wie ein Ergebnis zu bewerten ist. Mit diesem Wissen ist es möglich, in Zukunft ähnliche Projekte zu entwerfen und auf diesem Wissen aufzubauen. Durch die Anwendung einer Klassifikation lässt sich dieses Wissen in Zukunft auch auf eine Objekterkennung oder Segmentierung anwenden. Eine strukturierte Herangehensweise

an neue Themengebiete und die Anwendung wissenschaftlicher Methoden werden durch die Ausarbeitung gefördert.

4.1.4 Forschende

Als Forschungsansatz ist diese Ausarbeitung wichtig, um ein Grundverständnis von Data Augmentation anhand von Kameraparametern zu gewinnen. Dieser Ansatz kann weiter ausgebaut werden, indem mehrere Sequencer-Sets zur Verarbeitung herangezogen oder andere Sets ausgewählt werden. Darüber hinaus bietet diese Arbeit einen Einblick in die Entwicklung eines neuronalen Netzes und stellt somit einen idealen Einstieg in das Thema dar. Es werden allgemeine Beispiele und Leistungen von neuronalen Netzen vorgestellt und gezeigt, wie eine Bewertung dieser Ergebnisse zu betrachten ist. Es wird gezeigt, welche Methoden es für die Data Augmentation gibt, wie diese das Training beeinflussen und gibt im Kapitel Stand der Technik einen Ausblick auf sehr unterschiedliche Methoden.

4.2 Anwendungsfälle

Das Ziel dieser Arbeit ist es zu zeigen, ob es sich lohnt, eine Kamera mit Sequencer-Funktion für das Training von neuronalen Netzen zu verwenden. Dies ist dann der Fall, wenn das Ergebnis zeigt, dass ein solcher Ansatz Ressourcen spart oder bessere Ergebnisse liefert. Wenn ein Kunde einen Datensatz für eine Anwendung mit neuronalen Netzen erstellen möchte, könnte ein solcher Ansatz ein gutes Argument für die Kamera von Allied Vision sein.

Dabei soll die Robustheit gegenüber Lichtverhältnissen im Vordergrund stehen, so dass der Kunde einen Vorteil bei der Erstellung des Trainingsdatensatzes hat und sich weniger um die softwaretechnische Data Augmentation kümmern muss. Hier wird nur der Sequencer betrachtet, da dieser das Kernstück dieser Arbeit darstellt und später auch für diesen Anwendungszweck beworben werden könnte. Da Allied Vision noch keine Lösungen anbietet, sondern nur Kameras verkauft, ist dies einer der ersten Schritte, um den Kunden Lösungen anbieten zu können. Zu diesem Zweck wurde das in Abbildung 4.1 dargestellte Anwendungsfalldiagramm erstellt, welches die möglichen Anwendungsfälle des Sequencers darstellt.

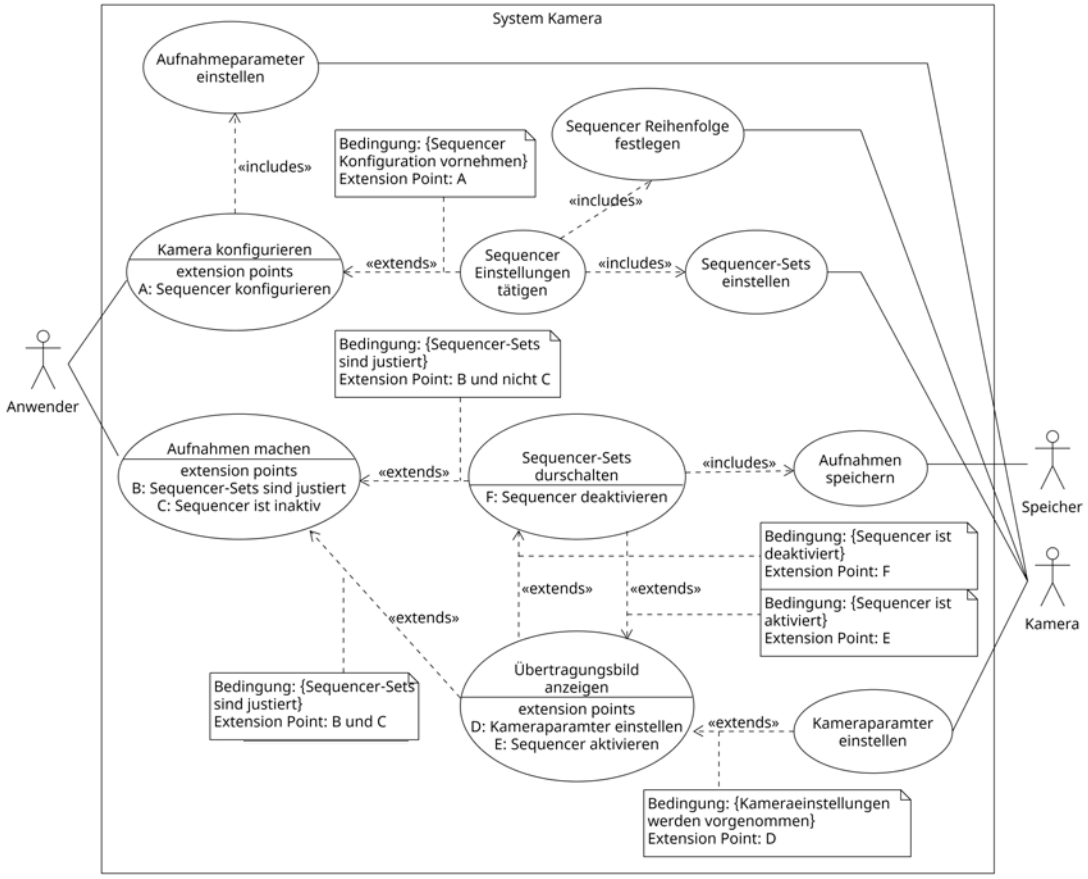


Abbildung 4.1: Anwendungsfalldiagramm

Es ist zu erkennen, dass der Anwender zwei Möglichkeiten hat, mit dem Kamerasystem zu interagieren: Konfigurieren der Kamera oder Aufnahmen machen. Die Konfiguration der Kamera umfasst die Einstellungen der Aufnahmeparameter, die unter anderem die Grundeinstellungen für die Kommunikation mit dem Interface und die Einstellungen der Kameraparameter (vgl. Kap. 2.1) beinhalten. Falls zusätzlich eine Sequencer-Funktion integriert werden soll, kann diese ebenfalls eingestellt werden. Dazu gehören auch die Einstellungen für die Reihenfolge der Sequencer-Sets, die festlegen, welches Set auf welches folgt und welcher Trigger zum Umschalten benötigt wird. Außerdem können hier die jeweiligen Sequencer-Sets genauer definiert werden, indem die Kameraparameter für das jeweilige Set eingestellt werden.

Sind die Einstellungen vorgenommen, kann auch die Kamera für Aufnahmen verwendet werden. Ist der Sequencer aktiv und sind die Sequencer-Sets eingestellt, können Aufnahmen mit dem Sequencer durchgeführt werden. Die Sequencer-Sets werden dabei

aufgezeichnet und gespeichert. Ist der Sequencer inaktiv und sind die Sequencer-Sets eingestellt, kann ein Livebild der Kamera betrachtet werden. Hier soll es möglich sein, die Kameraparameter entsprechend den aktuellen Lichtverhältnissen neu einzustellen und den Sequencer zu aktivieren. Durch Betätigen einer Taste soll es möglich sein, in eine Kameraparametereinstellung zu gelangen, die diese einstellen kann. Wenn der Sequencer aktiviert wird, wird der oben beschriebene Ablauf für den Sequencer aktiviert und umgekehrt, wenn der Sequencer deaktiviert wird.

Der Kamera-Aktor kann auf die eingestellten Kamera-Parameter sowie auf die Sequencer-Einstellungen der Sequenz und der Sequencer-Sets zugreifen. Diese Informationen werden über die Kamera-API eingestellt und die Kamera ist dann für die Ausführung der Funktionen verantwortlich. Der Speicher-Aktor hingegen greift auf die Aufzeichnungen zu und speichert diese auf der Festplatte, so dass im nächsten Schritt aus den Daten ein Datensatz erzeugt werden kann.

4.3 Anforderungsspezifikation

Um den Versuch in dieser Ausarbeitung reproduzierbar zu machen, ist es sinnvoll, einige Anforderungen an das Gesamtsystem zu stellen. Die Anforderungen sind in die Komponenten unterteilt, aus denen das Experiment besteht. Dazu gehören die Kamerasysteme mit dem Versuchsaufbau und den Objekten, die den ersten Teil des Systems bilden. Der zweite Teil bezieht sich auf die Erstellung eines neuronalen Netzes und den Plan, wie diese voneinander unterschieden werden sollen.

Das System soll entscheiden, ob die Erzeugung von Datensätzen durch Variation der Kameraparameter einer Data Augmentation vorzuziehen ist. Dazu muss ein eigener Datensatz erzeugt werden, der die Grundlage für das Training eines neuronalen Netzes darstellt, welches im Anschluss die Klassifikation durchführen soll. Um die Ergebnisse der neuronalen Netze miteinander vergleichen zu können, müssen geeignete Leistungsmetriken definiert werden.

Zunächst soll überprüft werden, ob die Data Augmentation mit Kameraparametern bessere Ergebnisse liefert als das Training mit Rohdaten. Anschließend soll ein Vergleich zwischen der Data Augmentation über Software und der Variation von Kameraparametern durchgeführt werden. Für den Vergleich sollen mehrere neuronale Netze

erstellt werden, um zunächst geeignete Hyperparameter zu finden, die die Klassifizierung durchführen können.

Durch Evaluierungsdatensätze soll das ausgewählte Prüfobjekt unter verschiedenen Umgebungsbedingungen abgebildet und gezeigt werden, welchen Einfluss die Kameraparameter auf die Genauigkeit des Netzes im Vergleich zu einer softwarebasierten Data Augmentation haben. Anschließend sollen die Kameraparameter gefunden werden, die den größten Einfluss auf die Genauigkeit eines neuronalen Netzes haben. Dazu sollen die besten Netze aus der Evaluierung mit den speziellen Datensätzen trainiert und für die Evaluierung verwendet werden.

Es folgen die Anforderungen an die Komponenten, die in dem Versuch benötigt werden, um die Frage zu beantworten, ob es einen Vorteil hat, Data Augmentation über einen Sequencer anzuwenden: Kamera, Objektauswahl, Versuchsaufbau, neuronales Netz und Vergleichskriterien.

4.3.1 Kamera

Eine Kamera muss geeignete Eigenschaften aufweisen, um für den Vergleich in diesem Projekt in Frage zu kommen. Die Anforderungen in der Tabelle 4.1 beschreiben die notwendigen Eigenschaften der Kamera, um eine einfache Steuerung und Konfiguration über eine API zu ermöglichen. Zudem wird sichergestellt, dass die Kamera in der Lage ist, sequenzielle Aufnahmen mit variierenden Kameraparametern zu erstellen, wodurch eine Data Augmentation direkt durch die Kamera ermöglicht wird.

4.3.2 Objektauswahl

Für diese Untersuchung müssen geeignete Prüfobjekte ausgewählt werden, die für die Erstellung des Datensatzes verwendet werden. Daher ist es notwendig, Anforderungen an die Prüfobjekte für den Datensatz zu formulieren. Diese sind in der Tabelle 4.2 zusammengefasst. Dies soll sicherstellen, dass die Prüfobjekte für eine Klassifikation geeignet sind und auch geeignete Eigenschaften zur Unterscheidung bieten, die ggf. die Vorteile eines Sequencers hervorheben können.

Tabelle 4.1: Anforderungen an die Kamera

Kennung	Anforderung
K-1	Die Kamera muss eine Funktion unterstützen, die eine Sequenz von Aufnahmen mit veränderten Kameraparametern ermöglicht.
K-2	Je Sequencer-Set muss mindestens ein Kameraparameter verändert werden, um einen Unterschied zwischen der Rohaufnahme darzustellen. Daraus muss eine diverse Auswahl an Sequencer-Sets entstehen, welche als Data Augmentation für einen Datensatz genutzt werden soll.
K-3	Die Kamera muss über eine USB-Schnittstelle erreichbar und über eine objektorientierte Programmiersprache programmierbar sein, so dass Konfigurationen vorgenommen, Frames empfangen und verarbeitet werden können.
K-4	Die Kamera muss mit einem Stativgewinde ausgestattet sein, um eine stabile Befestigung und Aufnahmen aus einer festen Position zu gewährleisten.

4.3.3 Versuchsaufbau

Es werden Anforderungen an den Versuchsaufbau gestellt, damit die Erzeugung der Datensätze von den Prüfobjekten reproduzierbar wiederholt werden kann. In der Tabelle 4.3 sind die Anforderungen an den Versuchsaufbau zusammengefasst.

4.3.4 Neuronales Netz

Für die Erstellung eines neuronalen Netzes zur Verarbeitung der Datensätze werden die Anforderungen in der Tabelle 4.4 aufgestellt. Die Anforderungen an das neuronale Netz sind wichtig, um sicherzustellen, dass die Ergebnisse der Evaluierung aussagekräftig sind und die Vergleichbarkeit der verschiedenen Ansätze gewährleistet ist.

4.3.5 Vergleichskriterien

Um die erstellten neuronalen Netze miteinander vergleichen zu können, werden Anforderungen an die genaue Umsetzung dieses Vergleichs mithilfe von Leistungsmetriken gestellt. Diese Anforderungen sind in der Tabelle 4.5 zusammengefasst.

Tabelle 4.2: Anforderungen an die Objektauswahl

Kennung	Anforderung
O-1	Für die Klassifikation der Prüfobjekte müssen 5 bis 10 unterschiedliche Klassen ausgewählt werden. Bei zu vielen Klassen wird die Datenerfassung schwierig, da jede Klasse ausreichend repräsentiert sein muss und bei zu wenigen Klassen ist die Aufgabe für ein neuronales Netz zu einfach. Daher ist die Modellkomplexität bei der geforderten Anzahl von Klassen überschaubar und somit für das übergeordnete Ziel, die Wirkung des Sequencers zu untersuchen, sehr gut geeignet.
O-2	Die Prüfobjekte müssen von ähnlicher Art sein. Ähnliche Objekte verhindern eine zu einfache Klassifizierungsaufgabe.
O-3	Für eine praktikable Durchführung darf ein Prüfobjekt eine Größe von 10 cm^3 nicht überschreiten. Diese Größe ermöglicht eine einfache Durchführung der Aufnahmen des Prüfobjekts und stellt sicher, dass das Prüfobjekt problemlos aus allen Richtungen aufgenommen werden kann. Zudem erlaubt diese Größenbeschränkung die Verwendung eines kompakten und transportablen Versuchsaufbaus, was die Flexibilität bei der Datenerfassung erhöht.
O-4	Die Prüfobjekte müssen optische Merkmale wie reflektierende Oberflächen oder marginale Farbunterschiede aufweisen. Solche Merkmale sollen dazu führen, dass ein neuronales Netz mit Hilfe unterschiedlicher Arten von Data Augmentation die Merkmale eindeutiger bestimmen kann.

Tabelle 4.3: Anforderungen an den Versuchsaufbau

Kennung	Anforderung
A-1	Es muss ein Versuchsaufbau erstellt werden, welcher es ermöglicht, Aufnahmen reproduzierbar zu erstellen. Dafür muss eine stabile Konstruktion erstellt werden, die die Kamera und das Prüfobjekt in einer festen Position hält.
A-2	Der Versuchsaufbau muss Aufnahmen von 360° vom Prüfobjekt ermöglichen. Dies soll ein robusteres neuronales Netz ermöglichen, das auch Objekte aus mehreren Perspektiven erkennen kann.
A-3	Lichtverhältnisse sollen simuliert werden können und es müssen Aufnahmen bei farbigem Licht erstellt werden. Eine Konstruktion sollte verschiedene Jahreszeiten berücksichtigen, spezielle Räume wie z. B. in der Fotolithografie (in der Fotolithografie wird gelbliches Licht verwendet, das keine kurzwelligen Anteile im UV-Bereich hat und somit einen Fotolack nicht ungewollt belichtet [2]). Wird ein neuronales Netz mit Datensätzen trainiert, die im Sommer erstellt wurden, sind diese im Winter bei anderer Lichteinstrahlung möglicherweise nicht mehr aktuell - die Prüfobjekte reflektieren möglicherweise anders.
A-4	Der Untergrund vom Prüfobjekt muss austauschbar sein. Diese Option soll helfen, verschiedene Situationen in der Realität zu simulieren und damit das neuronale Netz robuster zu machen.
A-5	Der Versuchsaufbau muss transportabel sein, damit die Datenaufnahme unter verschiedenen Bedingungen erfolgen kann.

Tabelle 4.4: Anforderungen an das neuronale Netz

Kennung	Anforderung
N-1	Ein neuronales Netz muss klassifizieren können.
N-2	Das neuronale Netz muss bei der Validierung eine Genauigkeit von mindestens 95% erreichen, um für die Bewertung geeignet zu sein. Eine Genauigkeit von 95% stellt sicher, dass das neuronale Netz eine hohe Zuverlässigkeit bei der Klassifikation aufweist und somit die Ergebnisse der Evaluierung aussagekräftig sind. Ein niedrigerer Wert könnte darauf hindeuten, dass das Netz nicht ausreichend trainiert wurde oder die Datenqualität unzureichend ist, was die Vergleichbarkeit der verschiedenen Ansätze beeinträchtigen würde.
N-3	Neuronale Netze müssen reproduzierbar erstellt werden, um einen fairen Vergleich zu ermöglichen. Das bedeutet, dass alle Zufallsparameter, die das neuronale Netz beeinflussen, auf feste Werte gesetzt werden müssen.
N-4	Die zehn besten neuronalen Netze müssen durch eine Evaluierung erneut überprüft werden. Eine Evaluierung soll ein neuronales Netz mit neuen Daten, die andere Umweltbedingungen repräsentieren, erneut überprüfen.

Tabelle 4.5: Anforderungen an die Vergleichskriterien

Kennung	Anforderung
V-1	Die Leistungsmetriken *Precision*, *Recall* und der $F1_{Score}$ müssen für jedes erstellte neuronale Netz bestimmt werden.
V-2	Die trainierten neuronalen Netze müssen anhand der Leistungsmetriken miteinander verglichen werden.
V-3	Nachdem die zehn besten neuronalen Netze ermittelt wurden, muss durch einen Vergleich mit der Auswertung festgestellt werden, ob die Kameraparameter einen Einfluss auf die Genauigkeit des Netzes haben.
V-4	Für das neuronale Netz mit der besten Leistung in der Evaluierung muss anschließend überprüft werden, welche Kameraparameter den größten Einfluss auf die Ergebnisse in der Evaluierung haben.

5 Konzept und Design

In diesem Abschnitt werden die erarbeiteten Lösungen für die gestellten Anforderungen dargestellt und analysiert. Dabei werden verschiedene Konzepte mit ihren jeweiligen Vor- und Nachteilen beschrieben und vergleichend bewertet. Abschließend wird die ausgewählte Lösung detailliert ausgearbeitet und die Umsetzung im Design konkretisiert.

5.1 Konzept

In diesem Abschnitt wird der Versuchsaufbau beschrieben, der die Grundlage für die Erstellung eines Datensatzes bildet. Dazu gehört die Auswahl geeigneter Prüfobjekte, die für die Klassifikation verwendet werden, sowie die detaillierte Beschreibung des Aufbaus selbst.

Nach der Erstellung des Datensatzes wird ein neuronales Netz entwickelt, das die Klassifikation der Prüfobjekte übernimmt. Verschiedene Ansätze zur Netzarchitektur werden analysiert und entsprechend den Anforderungen aus dem Kapitel 4.3 ausgewählt. Darüber hinaus werden geeignete Leistungsmetriken definiert, um die Qualität der Klassifikation zu bewerten.

Abschließend wird durch eine Evaluierung untersucht, welches neuronale Netz unter realistischen und erschwerten Bedingungen die besten Ergebnisse liefert. Ziel ist es, die Leistungsfähigkeit der verschiedenen Ansätze zu vergleichen und die optimale Lösung zu identifizieren.

5.1.1 Kamera

Die Geräteauswahl wird stark von den Anforderungen K-1 und K-2 beeinflusst, die die Sequencer-Funktion beinhalten. Auf dem Markt gibt es einige Hersteller, die die

gewünschte Sequencer-Funktion anbieten, wie z. B. die CV60-USB-3.0-Serie von Zebra[1] oder die USB 3 uEye CP Rev. 2 Kamerafamilie von IDS[2].

Diese Ausarbeitung soll herausfinden, ob der Sequencer von Allied Vision eine Verbesserung der Data Augmentation ermöglicht, daher wird eine Kamera des Herstellers verwendet. Sensoren, die in Kameras von Allied Vision eingebaut sind und eine Sequencer-Funktion unterstützen, sind Sony IMX Sensoren, die in Alvium 1800 U und GigE Kameras eingebaut sind (vgl. [53]).

Eine Sequencer-Funktion ermöglicht die Aufnahme mehrerer Aufnahmeserien mit unterschiedlichen Kameraparametern. Um die Anforderung K-2 zu erfüllen, müssen möglichst viele Kameraparameter veränderbar sein. Dies schließt monochrome Kameras aus, da bei diesen die Einstellung von Farbparametern nicht möglich ist und somit weniger Data Augmentation möglich wäre.

Um die Anforderung K-3 zu erfüllen, sollte die Kamera auch über eine USB-Schnittstelle verfügen, um die Kameraparameter zu konfigurieren und die Aufnahmen zu empfangen. Kameras mit GigE-Schnittstelle (Gigabit Ethernet) sind ebenfalls möglich, jedoch ist die Übertragungsrate bei USB höher, so dass die Aufnahmen schneller empfangen werden können. Daher wird die USB-Schnittstelle gewählt, da eine Übertragung über GigE zu langsam ist.

Jede der Alvium 1800 U und GigE Kameras verfügt über die Möglichkeit, ein Stativ an der Kamera zu befestigen, wodurch die Anforderung K-4 erfüllt wird. Unter Berücksichtigung der oben genannten Anforderungen folgt eine kleine Auswahl von möglichen Modellen, die für diese Ausarbeitung in Frage kommen: Alvium 1800 U-2460c, Alvium 1800 U-240c, Alvium 1800 U-052c und Alvium 1800 U-040c.

Die Modelle Alvium 2460c (24,6 Megapixel) und Alvium 1800 U-240c (2,4 Megapixel) sind zu hoch aufgelöst und scheiden daher aus. Die zu trainierenden Netze sollten auf kleineren Aufnahmen trainiert werden, da dies schneller geht und viele Netze miteinander verglichen werden sollen. Die Unterschiede zwischen den Modellen Alvium 1800 U-052c und Alvium 1800 U-040c sind geringer. Der U-052c hat eine Auflösung von 0,5 Megapixel (816×624 Pixel) und der U-040c hat eine Auflösung von 0,4 Megapixel (728×544 Pixel). Beide Größen sind für das Training neuronaler Netze gut geeignet, so dass beide Kameramodelle in Frage kommen. Die Alvium 1800 U-052c wird für das

[1] https://www.zebra.com/de/de/products/spec-sheets/industrial-machine-vision-fixed-scanners/machine-vision-cameras/cv60-usb3.html
[2] https://de.ids-imaging.com/techtipp-details/techtip-parameter-change-use-sequencer.html

Abbildung 5.1: Produktbild der Alvium 1800 U-052c [52]

Experiment verwendet (siehe Abbildung 5.1), da sie eine höhere Auflösung und eine höhere Aufnahmerate bietet. Sie hat eine Bildrate von 688 fps, während die U-040c im Vergleich dazu nur eine Bildrate von 494 fps hat (Daten dieses Kameramodells nach [52]).

Die ausgewählte Kamera verfügt über folgende Kameraparameter, die mit dem Sequencer verändert werden können: Belichtungszeit, Gain, Gamma, Sättigung, Farbton und Farbwert der RGB-Kanäle. Eine aufgenommene Aufnahme hat eine Größe von 816x624 Pixel und wird mit 8 Bit pro Kanal gespeichert. Die Kamera verfügt über eine USB 3.0 Schnittstelle und kann auf einem Stativ montiert werden. Zur Ansteuerung der Kamera bietet Allied Vision mit Vimba X eine Reihe von APIs für verschiedene Programmiersprachen an.

5.1.2 Data Augmentation Methoden durch den Sequencer

Die durch den Sequencer erzeugten Aufnahmen sollen anschließend als Data augementation genutzt werden. Daher muss zunächst festgelegt werden, welche Kameraparameter mit dem Sequencer verändert werden sollen. Verfügbare Kameraparameter sind: Belichtungszeit, Gain, Gamma-Wert, RGB-Wert, Farbton und Sättigung. Daraus ergeben sich die folgenden Sequencer-Sets, die durchlaufen werden sollen:

1. Normale Aufnahme, ohne Veränderung der Parameter
2. Minimale Belichtungszeit und maximales Gain
3. Erhöhtes Gain
4. Niedrige Belichtungszeit
5. Minimales Gamma

6. Maximales Gamma
7. Minimale Sättigung
8. Maximale Sättigung mit maximalem Farbton
9. Minimale Sättigung mit minimalem Farbton
10. - 15. Rote, gelbe, grüne, türkise, blaue und rosa Aufnahme

Bei den Sequencer-Sets 10 bis 15 wird jeweils eine Kombination der RGB-Kanäle isoliert, so dass ein oder zwei Kanäle deaktiviert werden. Dadurch entstehen die beschriebenen Effekte, wie beispielsweise eine grüne Aufnahme, wenn der rote Kanal deaktiviert wird und nur die grünen und blauen Kanäle aktiv bleiben.

5.1.3 Objektauswahl

Ein Klassifizierungsproblem erfordert eine Auswahl von Prüfobjekten, die identifiziert werden sollen. Für die Auswahl eines geeigneten Prüfobjektes müssen zunächst die Rahmenbedingungen für die Erstellung eines solchen Trainingsdatensatzes festgelegt werden, aus denen dann eine Auswahl zu treffen ist. Wissenschaftliche Arbeiten, wie z. B. [46, 34, 29], die sich mit dem Vergleich von Data Augmentation Verfahren beschäftigen, verwenden in der Regel Datensätze wie CIFAR-10 oder MNIST, die für anwendungsspezifische Fälle nicht geeignet sind, da die Aufnahmedaten eine Zusammenstellung von Aufnahmen verschiedener Kameras sind.

Die Frage dieser Ausarbeitung ist, ob eine Kamera mit Sequencer-Funktion für Data Augmentation genutzt werden kann und das funktioniert nur mit echten Objekten, die mit dieser Kamera aufgenommen wurden. Daher muss ein eigener Datensatz erstellt werden, der eine Auswahl von Prüfobjekten erfordert. Basierend auf den gestellten Anforderungen sollen nun die folgenden Vorschläge von Prüfobjekten diskutiert werden.

Vorschlag 1 - Miniaturfiguren: Miniaturfiguren werden im Modellbau gerne zur Darstellung von Alltagssituationen verwendet. Es gibt eine scheinbar unendliche Auswahl an Figuren, die auch preiswert zu erwerben sind. Für die Anforderung O-1 werden sie in mehrere Figurenklassen eingeteilt und in der Regel gibt es viele verschiedene Figuren in einem Set. Durch die Eigenschaft einer Figur sind sich diese sehr ähnlich, da sie sich nur im Aussehen und in der Ausstattung unterscheiden. Dies erfüllt die Anforderungen O-2 und O-3. Für O-4 ist zu diskutieren, ob sich die Figuren auf der

Oberfläche ausreichend voneinander unterscheiden. Kunststoff reflektiert sehr gut, aber herkömmliches Spielzeug ist in der Regel billig hergestellt und alle Figuren haben die gleiche Textur und unterscheiden sich nur in der Farbe. Aufgrund der fehlenden Komplexität der Figuren in der Oberflächenbeschaffenheit wird dieser Ansatz nicht weiter verfolgt.

Vorschlag 2 - Klemmbausteine: Die Anforderung O-1 ist durch die Vielfalt der Bausteine erfüllt, und da alle Bausteine auch vom Aufbau her sehr ähnlich sind, ist auch die Anforderung O-2 erfüllt. Klemmbausteine sind auch klein (für Anforderung O-3) und für reproduzierbare Aufnahmen geeignet, obwohl einige Teile auch etwas größer werden können. Die Variation der Oberflächenstrukturen ist auch bei Klemmbausteinen gegeben, es gibt u.a. matte, reflektierende und farbige Oberflächen, womit die Anforderung O-4 erfüllt ist. Dieser Vorschlag erfüllt alle Anforderungen, aber es ist zu beachten, dass diese Teile in allen möglichen Positionen aufgenommen werden müssen. Da diese Teile aus allen Blickwinkeln erkannt werden müssen, muss das Klemmbauteil für die Erstellung des Datensatzes gedreht und gewendet werden. Daher ist der Aufwand für die Aufnahmen zu hoch und dieser Vorschlag wird für den Versuch nicht verwendet.

Vorschlag 3 - Elektrische Bauteile: Diese Bauteile sind in großen Mengen in Bausätzen erhältlich und stellen somit eine große Auswahl dar. Ihre Größe eignet sich sehr gut für einen solchen Test, da sie normalerweise für Leiterplatten verwendet werden, wodurch die Anforderung O-3 perfekt erfüllt wird. Zu beachten ist auch, dass diese Bauteile in der Regel aus allen Blickwinkeln gleich aussehen, was die Erstellung des Datensatzes erleichtert. Die Anforderungen O-1, O-2 und O-4 werden auch durch die große Auswahl an Prüfobjekten erfüllt, die ebenfalls unterschiedliche Eigenschaften haben und in verschiedenen Farben und Größen erhältlich sind. Insbesondere die unterschiedlichen Materialien, aus denen die elektronischen Bauteile bestehen, sind für die Variation der Oberflächen sehr vorteilhaft. Da sich die Bauteile gut voneinander unterscheiden, ist die Klassifizierung nicht zu schwierig und das resultierende Netz nicht zu komplex. Im Kapitel 3.2.3 wird ein ähnlicher Ansatz verfolgt und im Experiment ist die Vorgehensweise sehr gut beschrieben. Da der Fokus dieser Ausarbeitung auf dem Thema Data Augmentation liegt, wird die Idee aufgegriffen und die Prüfobjekte zur Klassifizierung von elektrischen Bauteilen verwendet.

Für diese Untersuchung wurden folgende elektronische Bauelemente ausgewählt (siehe auch Abbildung 5.2): Widerstand, LED, Elektrolyt-Kondensator, Keramik-Kondensator, Photoresistor, Diode und Transistor. Mit diesen sieben Klassen soll das Klassifikations-

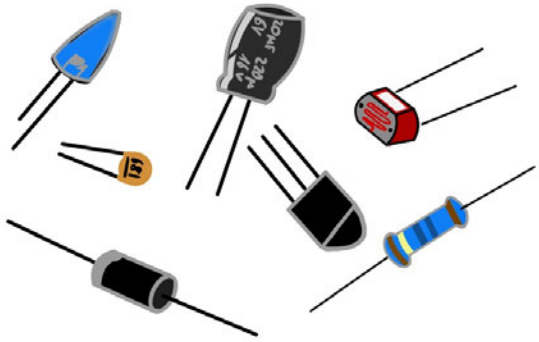


Abbildung 5.2: Für die Untersuchung dieser Ausarbeitung verwendete Prüfobjekte

problem des neuronalen Netzes trainiert werden. Die ausgewählten Prüfobjekte sind farblich in drei Klassen unterteilt: blau für LED und Widerstand, schwarz für Transistor, Diode und Elektrolytkondensator und rot für Photoresistor und Keramikkondensator. Das Netz soll die Bauelemente voneinander unterscheiden können, und durch diese Einteilung wird ein bloßes Lernen einer Farbe verhindert. Die Diode und der Widerstand sind in ihrem äußeren Aufbau und der Anzahl der Beinchen ähnlich. Diese Auswahl an Prüfobjekten wurde so gewählt, dass das neuronale Netz mehr lernen muss als nur eine Farbe, Anordnung oder Ausrichtung der Beine.

5.1.4 Versuchsaufbau

Um einen Datensatz erstellen zu können, muss eine geeignete Umgebung geschaffen werden, damit die Prüfobjekte reproduzierbar aufgenommen werden können. Außerdem ist es wichtig, die Umgebungsbedingungen so zu konfigurieren, dass verschiedene Fälle von Aufnahmeorten abgedeckt werden können.

Vorschlag 1: Platzierung des Prüfobjektes auf einem Tisch, der von einer Schreibtischlampe beleuchtet wird. Dieser Vorschlag erfüllt die Anforderungen A-2, A-4 und A-5, da ein Prüfling aus allen möglichen Richtungen aufgenommen werden kann, die Tischoberfläche variabel gestaltet werden kann und der Aufbau transportabel ist. Der große Nachteil ist, dass eine reproduzierbare Aufnahme nicht gewährleistet ist, da die Position der Kamera nicht fixiert ist, was die Anforderung A-1 nicht erfüllt. Somit muss der Anwender die Kamera um das Prüfobjekt herum bewegen oder das Prüfobjekt selbst drehen. Diese Option stellt keine reproduzierbare Methode dar und es ist auch schwer

vorstellbar, dass ein Anwender dies im industriellen Umfeld so reproduzieren würde. Daher wird diese Option nicht weiter verfolgt.

Vorschlag 2: Ein Aufbau aus einem Lichtlabor, der eine gleichmäßige Ausleuchtung des Prüfobjektes ermöglicht. Ein solcher Aufbau erfüllt die Anforderungen A-1 und A-3, da eine gleichmäßige Ausleuchtung des Prüfobjektes gewährleistet ist, unterschiedliche Lichtverhältnisse erzeugt werden können und die Kamera auf einem Stativ befestigt werden kann. Wird zusätzlich ein Drehteller verwendet, so kann jedes Prüfobjekt unter gleichen Bedingungen aufgenommen werden, womit die Anforderung A-2 erfüllt ist. Auf den Drehteller können verschiedenfarbige Folien gelegt werden, um die Anforderung A-4 zu erfüllen. Der Nachteil ist, dass der Aufbau nicht transportabel ist, da er nur an einem festen Ort eingesetzt werden kann und somit die Anforderung A-5 nicht erfüllt ist. Ein Transport dieses Aufbaus würde es ermöglichen, ihn einem potentiellen Kunden vorzuführen. Ein solcher Aufbau müsste dann auch beim Kunden eingesetzt werden, was zusätzliche Kosten zur Kamera bedeuten würde. Da diese Anforderung nicht erfüllt ist und ein solcher Aufbau auch sehr teuer ist, wird dieser Vorschlag nicht weiter verfolgt.

Vorschlag 3: Eine Schiene mit einer Kamera, die um den Prüfling herum montiert ist, und einer Lichtquelle. Dieser Vorschlag erfüllt die Anforderungen A-1, A-2 und A-3, da die Kamera um den Prüfling herum aufgebaut ist, eine reproduzierbare Aufnahme gewährleistet ist und die Kamera auf einem Stativ befestigt werden kann. Die Aufnahmen können unter gleichen Bedingungen aufgenommen werden, wodurch die Anforderung A-1 erfüllt ist. Durch die Schiene kann die Kamera um den Prüfling bewegt werden, womit die Anforderung A-2 erfüllt ist. Durch verschiedene Einstellungen der Lichtquelle können unterschiedliche Lichtverhältnisse simuliert werden, womit die Anforderung A-3 erfüllt ist. Der Prüfaufbau ist transportabel und leicht aufzubauen, wodurch die Anforderung A-5 erfüllt ist. Da eine Schiene nur in einer Ebene um den Prüfling fahren kann, wird der Untergrund des Prüflings nicht erfasst, was die Anforderung A-4 nicht erfüllt. Außerdem ist dieser Vorschlag nicht für einen marktfähigen Aufbau geeignet, da ein Aufbau mit Schienen nicht praktisch zu verkaufen ist. Da dieser Vorschlag die Anforderung A-4 nicht erfüllt, wird dieser Vorschlag nicht weiter verfolgt.

Vorschlag 4: Ein Aufbau mit Schreibtischlampe, Drehteller, Stativ und Farbfolien. Die Anforderung A-1 ist hier erfüllt, da das Stativ reproduzierbare Aufnahmen ermöglicht. Der Drehteller stellt sicher, dass jedes Prüfobjekt in jeder Orientierung aufgenommen werden kann, wodurch die Anforderung A-2 erfüllt ist. Die Schreibtischlampe ermöglicht eine einfache Ausleuchtung des Prüfobjektes, was der Anforderung A-3 entspricht.

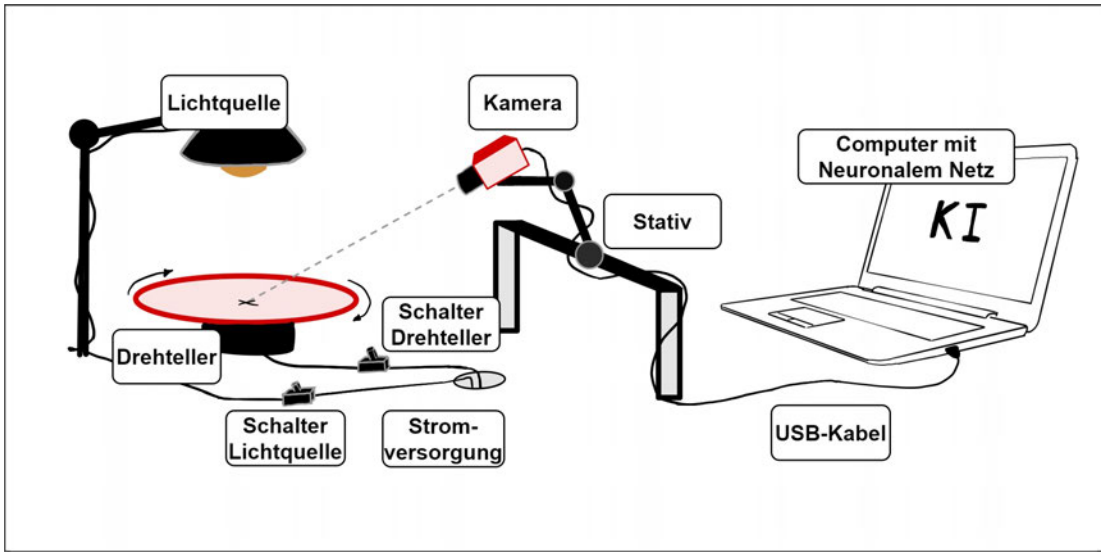


Abbildung 5.3: Versuchsaufbau

Es ist auch möglich, farbige Folie oder Papier an der Lichtquelle zu befestigen, um verschiedene Lichtverhältnisse zu simulieren. Auf dem Drehteller können verschiedenfarbige Folien angebracht werden, um die Anforderung A-4 zu erfüllen. Da die gesamte Konstruktion einfach aufzubauen und leicht zu transportieren ist, ist die Anforderung A-5 erfüllt.

Dieser Vorschlag erfüllt alle Anforderungen, ist zudem kostengünstig und wird daher für den Versuch verwendet. Der geplante Aufbau ist in der Abbildung 5.3 dargestellt.

5.1.5 Neuronales Netz

Die Data Augmentation wird für das Training eines neuronalen Netzes verwendet, daher muss ein solches Netz entwickelt werden. Es gibt verschiedene Ansätze, ein solches Netz für die Aufgabenstellung dieser Arbeit zu entwerfen, die hier diskutiert werden sollen.

Zunächst müssen die Anforderungen an das neuronale Netz aus der Anforderungsspezifikation erfüllt werden. Die Anforderung N-1 besagt, dass das Netz nur eine Klassifikation durchführen soll. Daher kommen für diese Ausarbeitung CNN-Modelle in Frage, die üblicherweise zur Klassifikation von Aufnahmen verwendet werden (vgl. [10, S. 92]). Es gibt verschiedene Arten von CNN-Modellen, die zur Klassifikation von Aufnahmen verwendet werden können.

Es sollte ein Netz gewählt werden, das nicht zu komplex, aber auch nicht zu einfach ist, da die Genauigkeit des Netzes nicht perfekt sein muss. Daher bietet sich ein AlexNet oder ein VGG16 Modell an, da diese relativ einfach aufgebaut sind und einen guten Vergleich darstellen (vgl. Kap. 2.4.4). Diese Modelle werden häufig als Grundlage für Modelle in der Aufnahmenkennung verwendet (vgl. [11]) und sind relativ einfach aufgebaut, weshalb diese Modelle für den Versuch verwendet werden.

Eine weitere Arbeit [49] vergleicht verschiedene Arten der Data Augmentation und verwendet ein CNN nach Krizhevsky (s. [23]), das ursprünglich sehr gute Ergebnisse bei der Klassifikation des ImageNet2012-Datensatzes erzielt hatte. Dieses „TaylorNitschkeNet“ wurde aus [56] übernommen und die Idee dahinter in der Ausarbeitung [49] weitergeführt (vgl. Kap. 3.2.1). Dort wurde dieses CNN-Modell ausgewählt, da es eine gute Genauigkeit bei der Klassifikation von Aufnahmen erreicht und auch relativ einfach aufgebaut ist und somit auch wenig Rechenzeit benötigt, was für einen Vergleich verschiedener Data Augmentationsverfahren wichtig ist. Daher wird dieses Netz auch in diesem Experiment zu den zu untersuchenden Modellen hinzugefügt.

Die neuronalen Netze müssen gemäß der Anforderung N-2 eine Genauigkeit von mindestens 95% erreichen, um eine zuverlässige Klassifikation zu gewährleisten. Durch die Anpassung von Hyperparametern wie Lernrate, *Batch Size* und Anzahl der Epochen kann das Netz optimal konfiguriert werden, um diese Anforderung zu erfüllen. Eine systematische Optimierung der Hyperparameter erfolgt mittels einem ausführlichen *Grid Search*[3]. Die besten Modelle werden anschließend basierend auf ihrer Performance ausgewählt. Alle Netze, die trainiert werden, müssen gemäß der Anforderung N-3 unter den gleichen Bedingungen trainiert werden, um die Vergleichbarkeit zu gewährleisten. Dies bedeutet, dass alle Zufallsparameter, wie z. B. der *Seed* für Zufallsberechnungen, für alle Netze gleich sein müssen. Durch eine anschließende Evaluierung nach Anforderung N-4 müssen die Leistungsmetriken erneut berechnet werden.

Zusätzlich zu den Anforderungen N-1 bis N-4 stellt sich die Frage, ob ein neuronales Netz komplett neu trainiert werden soll oder ob vortrainierte Gewichte verwendet werden sollen.

[3] *Grid Search* ist eine Methode zur Optimierung von Hyperparametern in KI-Modellen. Dabei werden für jeden Hyperparameter diskrete Wertebereiche definiert und alle möglichen Kombinationen getestet, um die Konfiguration mit der besten Modellleistung zu finden (vgl. *Beratung-KI: Grid Search*, verfügbar unter https://beratung-ki.de/glossar-eintrag/grid-search/, zuletzt abgerufen am 14.03.2025).

Vorschlag 1: Es wird ein vortrainiertes Netz verwendet, das z. B. auf dem ImageNet-Datensatz trainiert wurde. Diese Methode ermöglicht es, ein Netz zu verwenden, das bereits viele verschiedene Arten von Aufnahmen gesehen hat und daher eine gute Genauigkeit bei der Klassifikation von Aufnahmen erreichen kann. Beim *Transfer-Learning* wird schließlich nur der Klassifikator neu konfiguriert (vgl. Kap. 2.4.5). Dies hat den Vorteil, dass das Netz nur noch an die gewünschten Klassen angepasst werden muss und dann direkt eingesetzt werden kann. Ein großer Nachteil ist jedoch, dass das Netzwerk bereits viele verschiedene Arten von Data Augmentation gesehen hat und somit nicht mehr von einer neuen Data Augmentation profitieren kann. Darüber hinaus kann eine potentielle Verbesserung der Ergebnisse durch einen Sequencer nicht korrekt nachgewiesen werden, da ein solches Netzwerk nicht von neuen Datensätzen profitieren kann. Somit wird auch der Effekt des Sequencers nicht korrekt nachgewiesen.

Vorschlag 2: *Transfer-Learning* wird teilweise verwendet, um ein Netz zu trainieren. Hierbei wird ein Netz verwendet, das bereits an einem Datensatz trainiert wurde und dann an die gewünschten Klassen angepasst wird. Dabei wird nur eine bestimmte Anzahl von Schichten des neuronalen Netzes trainiert, die restlichen Schichten werden übernommen. Hier wäre der Vorschlag, dass z. B. beim VGG16 Netz die ersten Schichten übernommen werden, die für die Erkennung einfacher Formen zuständig sind (je tiefer die Schicht, desto komplexere Formen und Zusammenhänge können erkannt werden). Diese Methode hat den Vorteil, dass das Netz bereits einfache Strukturen gelernt hat, was das Training erleichtert. Ein Nachteil ist jedoch, dass das Netz bereits auf einem Datensatz trainiert wurde und somit nicht mehr von einem neuen Datenzuwachs profitieren kann. Da auch hier der Effekt des Sequencers nicht vollständig nachgewiesen werden kann, wird dieser Vorschlag nicht weiter verfolgt.

Vorschlag 3: Ein neuronales Netz wird komplett neu trainiert. Dieser Ansatz ist der zeitaufwendigste, da das Netz von Grund auf neu trainiert werden muss und grundlegende Strukturen erst erlernen muss. Dabei ist dieses Netz auf einen neuen Datensatz angewiesen und profitiert daher am meisten von der Data Augmentation, da diese Methode den Datensatz vergrößert und das Netz somit besser trainiert werden kann. Da hier ausgeschlossen werden kann, dass das Netz keinen anderen Einflüssen ausgesetzt war, ist dieser Ansatz für diese Ausarbeitung am besten geeignet. Damit die Gewichte des Netzes nicht zufällig sind und das Training reproduzierbar bleibt, muss hier eine Methode zur Initialisierung der Modelle gewählt werden.

Da der Schwerpunkt der Ausarbeitung auf der Untersuchung von Data Augmentationsmethoden liegt, wird der dritte Vorschlag gewählt und ein neuronales Netz komplett neu trainiert.

5.1.6 Vergleichskriterien

Jedes Netz, das durch die Hyperparameter-Kombinationen erstellt und trainiert worden ist, muss durch einen Test Leistungsmetriken erstellen und speichern. Diese sind für den Vergleich zwischen den Netzen wichtig, um festzustellen, wie gut sie sind und ob es sich lohnt, mit ihnen in die Bewertung zu gehen.

Die Anforderung V-1 besagt, dass jedes trainierte neuronale Netz die Leistungsmetriken *Precision*, *Recall* und den $F1_{Score}$ erzeugen soll (vgl. Kap. 2.6). Um die Anforderung V-2 zu erfüllen, werden die neuronalen Netze anhand der definierten Leistungsmetriken mit den Roh-, Sequencer- und Data Augmentation Datensätzen verglichen.

Im Rahmen einer Evaluierung sollen die zehn besten neuronalen Netze mit einem neuen Datensatz erneut trainiert werden. Dieser stellt erschwerte Bedingungen dar und soll sicherstellen, dass der Effekt des Sequencers deutlich wird. Dieser Vergleich ist gemäß der Anforderung V-3 durchzuführen und verwendet wiederum die gewonnenen Leistungsmetriken.

Schließlich werden die Sequencer-Sets ermittelt, die den größten Einfluss auf diese Metriken haben. Dieser Vergleich ist gemäß Anforderung V-4 durchzuführen und verwendet die Leistungsmetriken. Um später eine visuelle Darstellung der Ergebnisse zu haben, wird für jedes neuronale Netz eine *Confusion* Matrix erstellt.

Es soll nun diskutiert werden, mit welcher Genauigkeit ein Vergleich und ein daraus resultierendes *Ranking* der Netze durchgeführt werden soll. Mit den vorhandenen Metriken *Precision*, *Recall* und dem $F1_{Score}$ soll ein gemeinsamer Wert gefunden werden, der die Leistungsfähigkeit des Netzes widerspiegelt. Alle Anteile, die zu diesem Wert beitragen, sollten zusammen 100% ergeben. Netze mit einer hohen Genauigkeit, d. h. einem hohen Anteil richtig klassifizierter Aufnahmen im Verhältnis zu allen falsch klassifizierten Aufnahmen, sollten bevorzugt werden. Daher geht dieser Faktor mit einer hohen Gewichtung in die Bewertung ein: 40%.

Der *Recall*-Wert ist in diesem Fall nicht so wichtig, da er keine Bedeutung hat, wenn eine Komponente falsch klassifiziert wird. Dennoch wird dieser Wert einen geringen Einfluss auf die Bewertung haben: 10%.

Die Validierungsgenauigkeit wird mit 30% gewichtet, da eine hohe Genauigkeit wünschenswert ist.
Ein gutes neuronales Netz sollte auch einen geringen Validierungsverlust haben, daher wird dieser Wert mit 20% gewichtet.

Der Wert $F1_{Score}$ entfällt, da dieser aus *Precision* und *Recall* gebildet wird und diese beiden Werte bereits in der folgenden Gleichung mit einer anderen Gewichtung enthalten sind. Mit diesen Gewichtungen kann dann eine Rangfolge der Netze gebildet werden, die die besten Netze für die Forschungsfrage darstellt. Die Formel zur Berechnung des *Rankings* lautet:

$$\text{Ranking} = 0,4 \cdot \text{Precision} + 0,1 \cdot \text{Recall} + 0,3 \cdot \text{Accuracy} + 0,2 \cdot (1,0 - \text{Loss}) \quad (5.1)$$

5.2 Design

Das Design beschreibt die Softwarestruktur, die zugrunde liegenden Prozesse und die Implementierungsansätze zur Erfüllung der Anforderungen. Es umfasst die Darstellung der Kamerasteuerung, die Verarbeitung der gewonnenen Daten, das Training der neuronalen Netze sowie den anschließenden Vergleich der Ergebnisse. Abschließend wird die gesamte Softwarearchitektur zur Erstellung und zum Training der neuronalen Netze erläutert.

5.2.1 Abläufe der Software

Die Software gliedert sich in zwei Teile: Im ersten Teil wird die Kamerasteuerung separat betrachtet, im zweiten Teil erfolgt die Erstellung und der Vergleich der neuronalen Netze.

Bei der Ansteuerung der Kamera wird beschrieben, wie ein Anwender mit der Kamera mit Sequencer-Funktion Aufnahmen erstellen kann und wie der interne Ablauf

durchgeführt wird. Auf die genaue Konfiguration des Sequencers wird hier nicht weiter eingegangen (im Anhang A.4 ist jedoch eine grobe Vorgehensweise zur Konfiguration beschrieben).

Der Prozess zur Erstellung neuronaler Netze beginnt mit der Generierung eines Datensatzes aus den Sequencer-Aufnahmen. Mit diesem Datensatz wird dann ein neuronales Netz trainiert. Danach erfolgt ein Vergleich der trainierten Netzwerke, gefolgt von einer Evaluierung. Ziel ist es, herauszufinden, welcher Ansatz zur Datenerfassung die beste Performance liefert und welcher Kameraparameter den größten Einfluss auf die Ergebnisse hat.

Ansteuerung der Kamera

Für die Umsetzung der Anforderungen K-1 bis K-4 wird die Sequencer-Funktion der Kamera Alvium 1800 U-052c verwendet (vgl. Kap. 5.1.1). Die Abbildung 5.4 zeigt den Ablauf der Kameraansteuerung.

Der Ablauf ist wie folgt: Zunächst wird die Software für die Aufnahme des Datensatzes gestartet und der Benutzer kann die Anzahl m der Aufnahmen und den Namen des Prüfobjekts angeben, die aufgenommen werden sollen.

Die Software verbindet sich dann automatisch über die USB-Schnittstelle mit der Kamera (siehe Anforderung K-3) und richtet den Sequencer ein. Gemäß Anforderung K-4 ist die Kamera bereits stabil auf einem Aufbau montiert (siehe Abbildung 5.3) und muss nur noch auf das Prüfobjekt ausgerichtet werden. Sobald dies geschehen ist, zeigt die Software den aktuellen Kamerastream an[4] und der Benutzer kann gegebenenfalls die Kamera neu ausrichten, falls sich das Prüfobjekt nicht im Sichtfeld befindet. Wenn der Ausschnitt nicht scharf eingestellt ist, kann der Benutzer den Fokus der Kamera über das Objektiv einstellen und dann erneut prüfen, ob das Prüfobjekt scharf im Bild ist.

Im nächsten Schritt können die Kameraparameter an die aktuellen Umgebungsbedingungen angepasst werden. Dazu wird ein Impuls an die Kamera gesendet, der die Parameter Belichtungszeit, Gain und Weißabgleich einmalig automatisch zur aktuellen Umgebungsbedingung einstellt. Ist der Kamerastream anschließend nicht in Ordnung - die Farben stimmen nicht oder die Aufnahme ist zu hell oder zu dunkel - kann der

[4] Der Stream bezieht sich auf die von der Kamera zum Computer übertragenen Bilder.

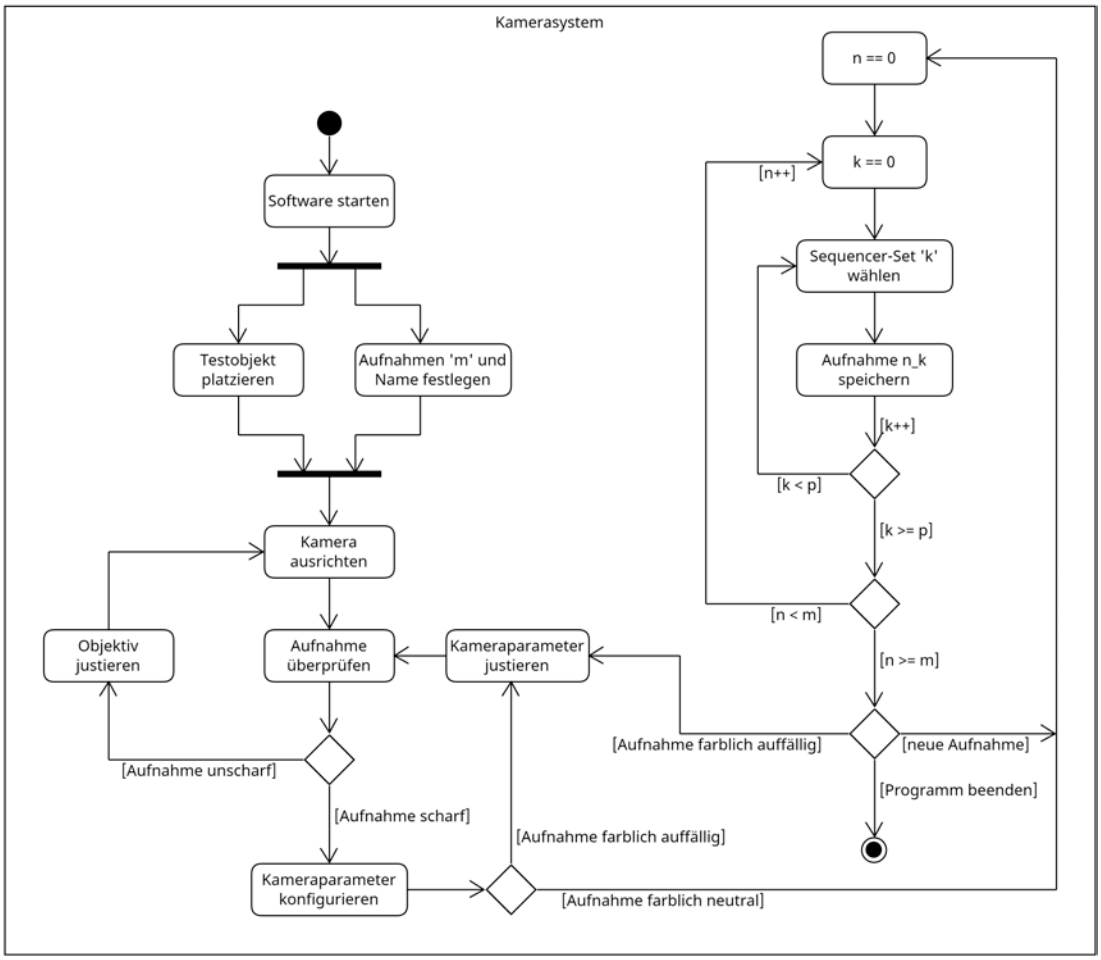


Abbildung 5.4: Aktivitätsdiagramm Ablauf der Kameraansteuerung

Benutzer die Parameter jeweils erneut automatisch einstellen lassen. Ist der Kamerastream dann korrekt, kann der Benutzer die Aufnahme starten.

Es werden die Aufnahmen n_k gemacht, solange die eingestellte Anzahl an Aufnahmen m noch nicht erreicht ist. Hierbei stellt n die aktuelle Anzahl der Aufnahmen und k das aktuelle Set dar - somit lassen sich die Sequencer-Sets im Nachhinein voneinander unterscheiden. Diese Aufnahmen werden dann im Prozess jeweils auf dem festgelegten Speicherort gespeichert. Eine Aufnahme gliedert sich dabei in p Sequencer-Sets, welche durch die Konfiguration des Sequencers festgelegt werden. Pro Sequencer-Set wird ein Kameraparameter verändert (siehe Kapitel 5.1.2).

Ist die Anzahl der Aufnahmen m erreicht, hat der Benutzer die Wahl, eine neue Aufnahme mit den gleichen Konfigurationen zu starten, die Aufnahmeparameter zu ändern oder das Programm zu beenden.

Ablauf vom Training der neuronalen Netze

Um den Einfluss der Kameraparameter bestimmen zu können, muss zunächst ein neuronales Netz entwickelt werden, das in der Lage ist, eine grundlegende Klassifikation durchzuführen. Dazu muss zunächst eine Auswahl an neuronalen Netzen gefunden werden, die für das Experiment in Frage kommen.

Dies geschieht durch ein ausführliches *Grid Search*, welches alle logischen Kombinationen von Hyperparametern anwendet und dann je ein neuronales Netz trainiert. Ziel ist es, eine Kombination zu finden, die zu einem neuronalen Netz führt, das gute und vergleichbare Klassifikationsergebnisse liefert. Dieser Vorgang ist in der Abbildung 5.5 dargestellt und soll im Folgenden näher erläutert werden.

Zunächst werden die zuvor generierten Daten als Datensatz geladen und auf eine einheitliche Größe skaliert. Anschließend erfolgt eine Standardisierung der Pixelwerte, um das Training des neuronalen Netzes zu stabilisieren. Dabei wird von jedem Pixelwert x der Mittelwert μ aller Aufnahmen eines Datensatzes abgezogen und durch die Standardabweichung σ aller Aufnahmen eines Datensatzes geteilt:

$$x' = \frac{x - \mu}{\sigma} \tag{5.2}$$

Dies stellt sicher, dass die Pixelwerte des Datensatzes eine gleichmäßige Verteilung mit einem Mittelwert von 0 und einer Standardabweichung von 1 haben. Dadurch kann das Netz effizienter lernen, da es nicht durch große Helligkeitsunterschiede beeinflusst wird. Ohne Normalisierung können helle Bildbereiche dunklere dominieren, was zu falschen Lernmustern und Stagnation des Trainings führen kann (nach [17, 27]).

Um die Leistung des Sequencers mit der normalerweise verwendeten Methode der Data Augmentation vergleichen zu können, muss diese zuerst durchgeführt werden. Dazu werden die Rohdaten aus dem Sequencer verwendet und durch Data Augmentation verändert. Dies geschieht vor dem gesamten Prozess, damit beide Varianten die gleichen Startbedingungen haben. Hierbei werden die Data Augmentation Datensätze nach den Sequencer-Sets verändert, so dass beide Varianten für einen Betrachter gleich aussehen.

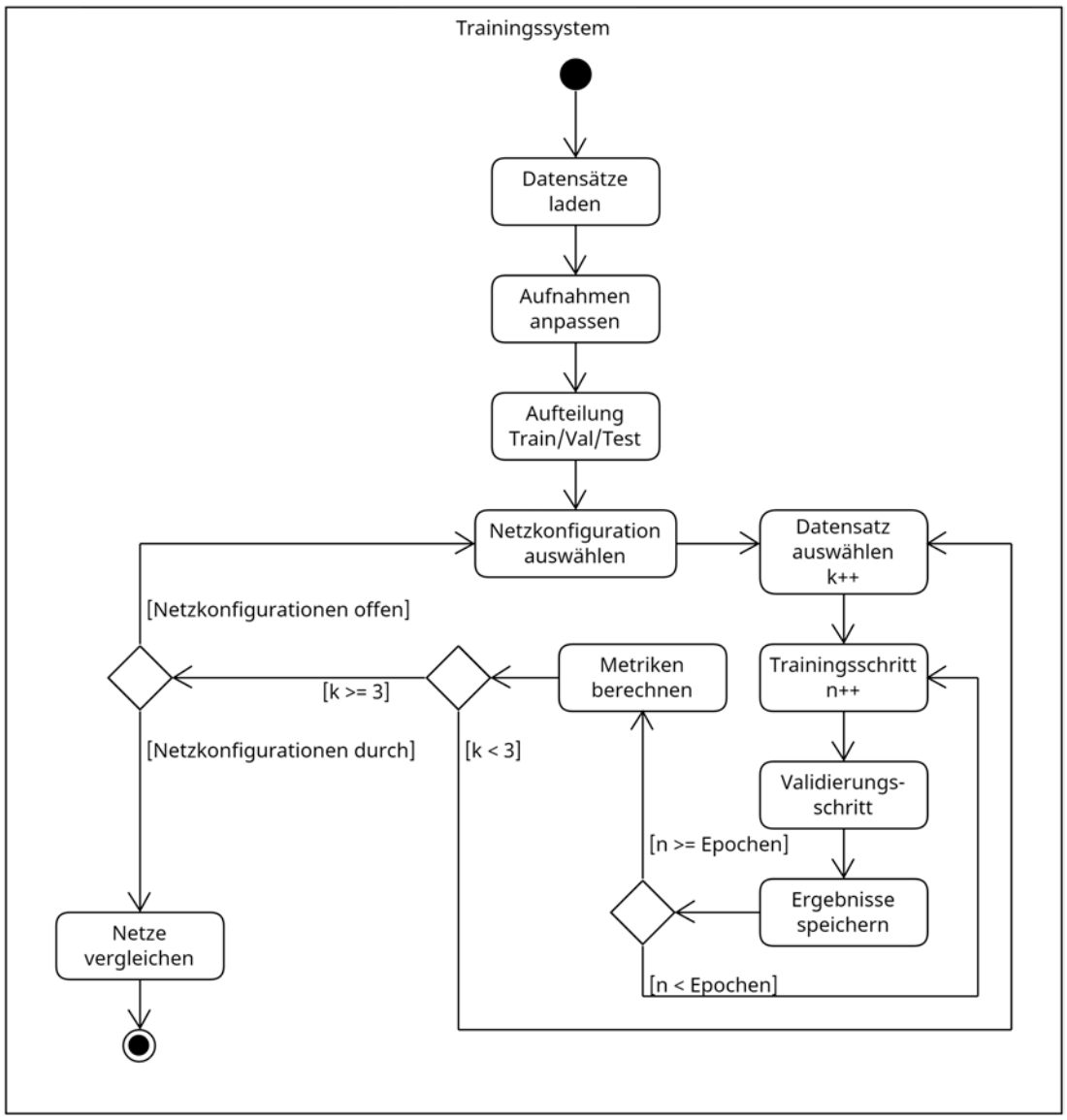


Abbildung 5.5: Aktivitätsdiagramm Ablauf des Trainings der neuronalen Netze

Damit soll sichergestellt werden, dass beide Ansätze die gleichen Startpositionen haben und ob der Datensatz mit direktem Einfluss der Kameraparameter unterschiedliche Leistungen erbringen kann.

Sobald alle Datensätze erstellt sind, werden sie in Trainings-, Validierungs- und Testdatensätze unterteilt. Um die Anforderung N-3 zu gewährleisten, werden die Datensätze für alle Modelle nur einmal erstellt, so dass auch die Aufteilung in Trainings-,

Validierungs- und Testdatensätze gleich ist. Außerdem werden alle Zufallsparameter im Programm auf eine feste Zahl gesetzt, so dass ein weiterer Durchlauf mit den gleichen Bedingungen durchgeführt werden kann.

Es folgt die Netzkonfiguration des zu trainierenden neuronalen Netzes. Hier werden die gewünschten Hyperparameter Modell, *Batch Size*, Initialisierungsmethode, Optimierer, Verlustfunktion, Lernrate, Scheduler und die Anzahl der Epochen durchlaufen. Für jede Kombination wird ein Modell trainiert und die Ergebnisse gespeichert. Die Aufgabe nach der N-1-Anforderung eines solchen Modells ist es, eine Klassifizierung anhand der elektrischen Komponenten aus dem Kapitel 5.1.3 durchzuführen.

Anschließend werden die oben genannten Datensätze *Rohdatensatz* (raw), *Data Augmentation Datensatz* (dA) und *Sequencerdatensatz* (seq) geladen, der Trainingsschritt und anschließend der Validierungsschritt durchgeführt. Danach werden die Zwischenergebnisse gespeichert und die nächste Epoche beginnt. Dies wiederholt sich, bis die festgelegte Anzahl von Epochen erreicht ist oder das Training durch andere Bedingungen unterbrochen wird. Eine Unterbrechung kann durch das Ausbleiben einer Verbesserung der Zwischenergebnisse der Validierung erfolgen.

Nachdem die Anzahl der Epochen erreicht wurde, werden die Leistungsmetriken *Precision*, *Recall*, $F1_{Score}$ und *Confusion* Matrix aus den Ergebnissen mit den Testdatensatz gemäß der Anforderung V-1 berechnet und gespeichert (vgl. Kap. 2.6). Dieser Prozess wird dann pro Datensatz k (*raw*, *dA* und *seq*) für ein Modell durchgeführt und anschließend ein weiteres Modell mit einer anderen Kombination von Hyperparametern trainiert. Nachdem alle Kombinationen durchlaufen wurden, können die Netze miteinander verglichen werden.

Ablauf vom Vergleich der Netze

Um zunächst eine Auswahl an Netzen finden zu können, mit welchen eine Evaluierung stattfinden soll, müssen zunächst die auf Basis der Anforderung V-1 trainierten Netze miteinander verglichen werden. Die folgende Abbildung 5.6 zeigt den Ablauf des Vergleichs der neuronalen Netze.

Im ersten Schritt werden die berechneten Leistungsmetriken aufgerufen und daraus ein *Ranking* nach der Formel 5.1.6 erstellt. Das Ergebnis ist eine nach diesem Wert sortierte Liste, die für die Auswahl der zehn besten Netzwerkkonfigurationen verwendet wird.

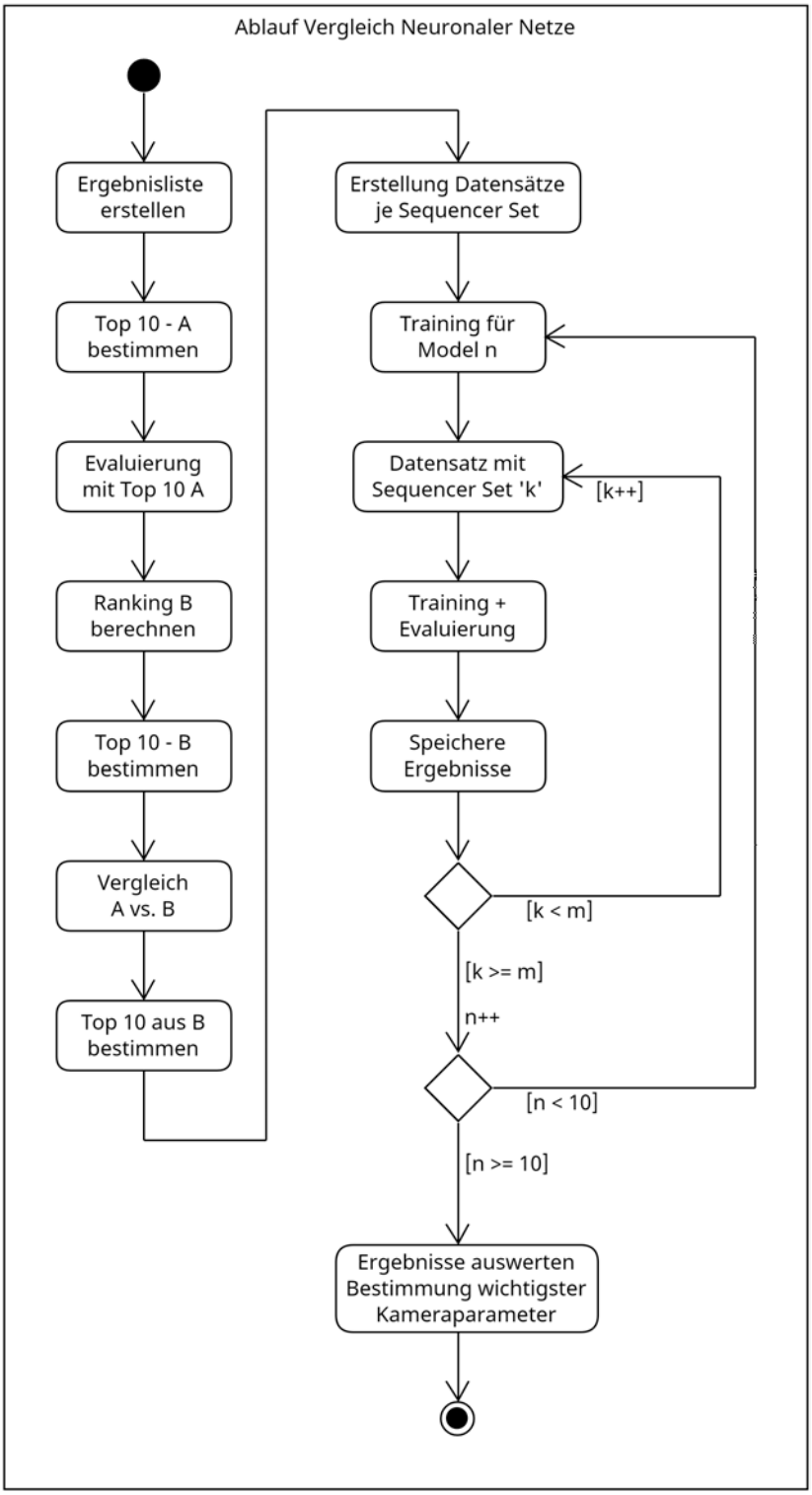


Abbildung 5.6: Aktivitätsdiagramm Ablauf des Vergleichs neuronaler Netze

Es wird geprüft, welche Konfiguration von Hyperparametern nach dem *Ranking* die besten Ergebnisse hat - hier wird der Datensatz nicht berücksichtigt sondern das beste Ergebnis aus den drei Datensätzen ausgewählt. Die resultierende Liste enthält nur die Top 10 der besten Hyperparameter-Kombinationen. Bei dieser Bewertung werden alle Netzkonfigurationen ausgeschlossen, die ungültige Ergebnisse liefern, sowie Netze mit einer Genauigkeit von weniger als 95 % (dies entspricht der Anforderung N-2).

Gemäß der Anforderung V-2 werden für die folgende Evaluierung auch die komplementären Modelle mit anderen Datensätzen hinzugefügt, da ein Vergleich angestrebt wird. Die Evaluierung erfolgt somit mit 30 Modellen (trainiert mit drei Datensätzen pro Modell). Die Evaluierung besteht darin, die gleichen Kombinationen von Hyperparametern erneut zu trainieren und dann mit einer größeren Anzahl von Daten zu testen. Dieser neue Evaluierungsdatensatz besteht aus Aufnahmen, die unter anderen Umgebungsbedingungen wie einer anderen Beleuchtungsfarbe oder -intensität oder einer anderen Hintergrundfarbe aufgenommen worden sind. Dieser Datensatz ist wesentlich größer als der Testdatensatz aus dem vorherigen Schritt und simuliert eine Anwendung unter realen Bedingungen. Aus den Ergebnissen wird wiederum eine Top-10-Liste (Version B) erstellt und die beiden Ranglisten werden gemäß Anforderung V-3 miteinander verglichen.

Der Vergleich dieser beiden Listen soll zeigen, ob die Modelle auch bei unterschiedlichen Lichtverhältnissen vergleichbare Ergebnisse liefern. Ähnliche Ergebnisse würden darauf schließen lassen, dass die Modelle robust gegenüber unterschiedlichen Lichtverhältnissen sind.

Aus den Modellen, die bei der Evaluierung die besten Ergebnisse erzielt haben, werden alle Sequencer-Modelle n zur Beantwortung der Frage dieser Ausarbeitung der Anforderung V-4 verwendet. Dazu wird zunächst ein Modell ausgewählt und jeweils nur mit dem Rohdatensatz einschließlich *eines* Sequencer-Sets k trainiert und evaluiert (m als Gesamtzahl an Sequencer-Sets). Da jedes Sequencer-Set einen Kameraparameter hervorhebt, kann durch Isolierung des Sequencer-Sets der Einfluss auf die Ergebnisse ermittelt werden. Hier kann ein Vergleich mit dem Rohdatensatz erfolgen, der die Verbesserung der Ergebnisse eines Kameraparameters darstellt. Ein weiterer Vergleich erfolgt zwischen den isoliert trainierten Netzen mit anderen Kameraparametern.

Für jedes der zehn Modelle werden Ergebnisse generiert, die anschließend analysiert werden, um den einflussreichsten Kameraparameter zu identifizieren. Der Vergleich der

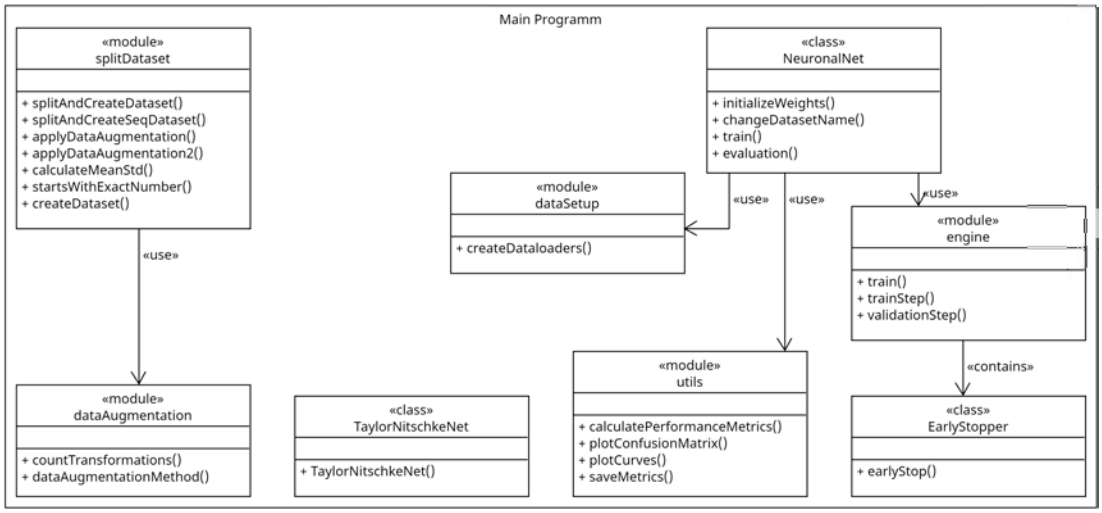


Abbildung 5.7: Klassendiagramm der Software

Ergebnisse zwischen den Modellen dient dazu, die Aussagekraft der Analyse weiter zu untermauern.

5.2.2 Struktur der Software

In diesem Abschnitt soll die Software für das Training eines neuronalen Netzes strukturiert werden. Dazu wird ein Klassendiagramm, siehe Abbildung 5.7, erstellt, das die Klassen und ihre Beziehungen untereinander darstellt. Dadurch soll gezeigt werden, wie die Software aufgebaut ist und wie die unterschiedlichen Methoden untereinander kommunizieren und funktionieren.
Die Software ist in der Programmiersprache Python unter Verwendung des Frameworks PyTorch geschrieben, das die Erstellung und das Training von neuronalen Netzen ermöglicht. Zusätzlich werden die Bibliotheken NumPy, Matplotlib und Pandas zur Datenverarbeitung und Visualisierung der Ergebnisse verwendet.

Das *main*-Programm enthält so viele Instanzen der Klasse *NeuronalNet*, wie es Kombinationen von Hyperparametern gibt, jedoch mindestens eine. Für die Aufgaben Training, Evaluation und spätere Analyse der besten Sequencer-Sets existieren jeweils leicht angepasste Versionen des *main*-Programms.

Modul	Methode	Beschreibung
splitDataset	splitAndCreateDataset()	Teilt den Datensatz auf in raw, seq und dA
	splitAndCreateSeqDataset()	Erstellt Datensatz zur Analyse der Sequencer-Sets
	applyDataAugmentation()	Wendet Data Augmentation auf den dA Datensatz an
	applyDataAugmentation2()	Wendet erweiterte Data Augmentation an
	calculateMeanStd()	Berechnet den Mittelwert und Standardabweichung des Datensatzes
	startsWithExactNumber()	Hilfsfunktion zur Erkennung ob Datenname mit Nummer x beginnt
	createDataset()	Erstellt den Datensatz für raw, seq und dA
applyDataAugmentation	countTransformations()	Zählt die Anzahl an verfügbaren Transformationen
	dataAugmentationMethod()	Wendet die gewünschte Data Augmentation Methode an
TaylorNitschkeNet		Initialisiert ein TaylorNitschkeNet
NeuronalNet	initializeWeights()	Initialisiert die Gewichte und Bias eines neuronalen Netzes
	changeDataSetName()	Ändert die Beziehung des Datensatzes für die Sequencer-Sets
	train()	Trainiert und Validiert ein neuronales Netz
	evaluation()	Evaluiert ein neuronales Netz mit neuen Datensätzen
dataSetup	createDataloaders()	Erstellt Dataloaders aus den übergebenden Datensätzen
utils	calculatePerformanceMetrics()	Berechnet die Leistungsmetriken aus den Ergebnissen
	plotConfusionMatrix()	Erstellt eine Confusionsmatrix aus den Ergebnissen
	plotCurves()	Erstellt Genauigkeits- und Verlustgraphen
	saveMetrics()	Speichert die Leistungsmetriken und Informationen über das neuronale Netz ab
engine	train()	Startet das Training eines neuronalen Netzes
	trainStep()	Ein Trainingsschritt einer Epoche
	validationStep()	Ein Validierungsschritt einer Epoche
EarlyStopper	earlyStop()	Überprüft, ob die Bedingungen für einen frühen Stopp gegeben sind

Tabelle 5.1: Auflistung und Kurzbeschreibung der Module des Programms

Die Tabelle 5.1 gibt einen Überblick über die verwendeten Module und ihre Funktionen. Im Hauptprogramm werden die zu testenden Hyperparameter in Listen definiert und mit Hilfe von Schleifen systematisch durchlaufen und getestet.

Mit dem Modul *splitDataset* wird der vorhandene Datensatz aufgeteilt und für das neuronale Netz vorbereitet. Hierbei entstehen die Datensätze Roh-, Sequencer- und Data Augmentation Datensatz. Durch das dazugehörige Modul *dataAugmentation* wird der Data Augmentation Datensatz verändert, sodass die optischen Effekte des Sequencers nachgebildet werden können. Die genutzten Modelle *AlexNet* und *VGG16* werden di-

rekt über PyTorch geladen und das *TaylorNitschkeNet* wird durch die gleichnahmige Klasse erstellt.

Über die *NeuronalNet* Klasse werden die Hyperparameter in ein Modell umgesetzt, sodass dieses trainiert werden kann. Mit *dataSetup* werden die Datensätze in Dataloaders umgewandelt, sodass PyTorch damit arbeiten kann. Über das Modul *engine* wird dann schließlich das neuronale Netz anhand der Datensätze trainiert und durch die Klasse *EarlyStopper* überwacht, sodass bei den eingestellten Bedingungen das Training abgebrochen werden kann.

In der Funktion *splitAndCreateDataset* im Modul *splitDataset* sollen die Daten für den Dataloader vorbereitet werden.
Der Sequencer erzeugt die Aufnahmen und speichert sie mit den jeweiligen Labels im Ordner *data* unter *source*. Ein Sequencer-Set besteht aus mehreren Aufnahmen mit unterschiedlichen Kameraparametern, wobei die erste Aufnahme stets die unveränderte Version ist. Anschließend werden die Aufnahmen im Verhältnis 70/15/15 in Trainings-, Validierungs- und Testdatensätze aufgeteilt. Diese Aufteilung wird in die entsprechenden Unterordner für die Datensätze *raw*, *seq* und *dA* kopiert. Nur der Sequencer enthält alle Aufnahmen eines Sequencer-Sets, während die anderen zunächst nur die ersten (unveränderten) Aufnahmen eines Sets enthalten. Die *dA* Aufnahmen durchlaufen anschließend eine Data Augmentation über das *dataAugmentation* Modul, so dass die Anzahl der Aufnahmen letztendlich der des Sequencerdatensatzes entspricht.

Die Methoden *splitAndCreateSeqDataset()* und *changeDataSetName()* sind Funktionen, die speziell für die Untersuchung der wichtigsten Kameraparameter erstellt wurden. Sie sorgen dafür, dass das Programm zunächst einen Datensatz pro Sequencer-Set anlegt und diesen dann einmal pro Durchlauf wechselt.

Die Methode *applyDataAugmentation2()* hingegen ist eine Abänderung der ursprünglichen Methode, welche in diesem Fall die neuen Arten der Data Augmentation anwendet, die für einen späteren Versuch gebraucht werden. Der Versuch soll eine vielseitigere Data Augmentation betrachten und mit den anderen Modellen vergleichen.

6 Umsetzung

Das Kapitel über die Umsetzung soll zeigen, wie das zuvor erstellte Konzept umgesetzt wurde. Es wird beschrieben, wie die Versuchsanordnung aufgebaut ist, worauf zu achten ist und wie die Umgebungsbedingungen variiert wurden. Um einen Überblick zu schaffen, wie ein Sequencer konfiguriert wird und wie diese Aufnahmen dann für das Training bereitgestellt werden.

Für die Datensätze wird dann gezeigt, wie diese für den Vergleichsdatensatz verändert werden und welche Methode hier angewendet wird. Es wird darauf eingegangen, inwieweit sich der Evaluierungsdatensatz von den anderen unterscheidet und anschließend werden alle Aufnahmen für das Training vorbereitet.

Anschließend wird beschrieben, wie die Suche nach den am besten geeigneten Hyperparametern für diesen Versuch durchgeführt wird. Dazu gehört auch eine Beschreibung, wie das Training, die Validierung, der Test und die Evaluierung umgesetzt werden und wie die jeweiligen Ergebnisse im Anschluss dargestellt werden. Abschließend wird gezeigt, wie die besten Kameraparameter für diesen Versuch gefunden werden können.

6.1 Versuchsaufbau und Datensatzgenerierung

In diesem Abschnitt werden der Versuchsaufbau und die Generierung der Versuchsdatensätze beschrieben. Dabei werden die einzelnen Komponenten des Versuchsaufbaus im Detail vorgestellt und ihre jeweilige Funktion erläutert.

Zusätzlich wird auf die Anpassungen des Versuchsaufbaus für die Evaluierung eingegangen, um die Unterschiede zum Trainingsaufbau hervorzuheben. Außerdem wird beschrieben, wie die Durchführung erfolgte und welche Änderungen vorgenommen wurden, um die gewünschten Ergebnisse zu erzielen.

Abbildung 6.1: Auswahl an Aufnahmen aus mehreren Perspektiven

6.1.1 Kamera

Die Kamera Alvium 1800 U-052c wird auf einer Konstruktion gemäß Abbildung 5.3 befestigt. Diese Konstruktion kann auch durch ein Stativ ersetzt werden, jedoch erweist sich diese Methode im Versuch als vorteilhaft, da die Position leichter verändert werden kann. Durch das Stativgewinde ist es dann auch möglich, dass diese Kamera fest installiert werden kann und sich somit nicht bewegen kann.

Die angebrachten Teleskopstangen haben gezeigt, dass eine veränderbare Position der Kamera wichtig ist, um die kleinen Prüfobjekte zu erfassen.

6.1.2 Drehteller

Der Drehteller sorgt dafür, dass das Prüfobjekt aus mehreren Perspektiven aufgenommen werden kann. Durch die langsame Drehgeschwindigkeit ist es außerdem möglich, alle Sequencer-Sets nacheinander aufzunehmen, ohne dass sich das Objekt in den Aufnahmen zu sehr bewegt.

Als Trainingshintergrund wurde weißes Papier verwendet, da es einen neutralen Hintergrund darstellt. Ziel ist es, das Netz so zu trainieren, dass der Hintergrund im Idealfall ignoriert wird und das Netz nur das Objekt wahrnimmt.

In der Abbildung 6.1 sind Aufnahmen eines Widerstandes zu sehen, welcher durch den Drehteller aus mehreren Perspektiven aufgenommen wurde.

6.1.3 Beleuchtung

Für die Beleuchtung ist es zunächst wichtig, dass der Aufbau in einem dunklen Raum stattfindet. Dadurch ist sichergestellt, dass die aufgestellte Schreibtischleuchte die einzige Lichtquelle im Raum ist und alle Aufnahmen unter gleichen Bedingungen stattfinden können.

Diese Erkenntnis kam durch die ersten Aufnahmen im Winter, welche durch kurze Zeiten mit Sonneneinstrahlung die Belichtung am Arbeitsplatz mit einem Fenster keine gleichen Bedingungen pro Aufnahme gewährleisten. Hierbei sind durch die Deckenbeleuchtung in Kombination mit der Schreibtischlampe Überbelichtungen entstanden und diese Aufnahmen konnten dann nicht verwendet werden.

Im dunklen Raum ist es dann auch möglich, ein farbiges Licht für die Evaluierung zu nutzen, da dort sichergestellt werden kann, dass das Prüfobjekt nur von einer Lichtquelle bestrahlt wird.

6.1.4 Prüfobjekt

Für den Versuch wurden die Prüfobjekte in Form von elektrischen Bauteilen ausgewählt. Diese Objekte zeichneten sich durch ihre geringe Größe und die unterschiedliche Oberflächenbeschaffenheit aus. Außerdem müssen sie nicht umgedreht werden, da sie eine zylindrische Form haben und von allen Seiten gleich aussehen (mit Ausnahme des Transistors, der im Experiment nur in einer Position getestet wurde). Die Klassen für das Experiment sind in der Abbildung 6.2 dargestellt. Hier sind die Aufnahmen zu sehen, welche im Anschluss auch für das Training verwendet werden sollen.

Jede Klasse der elektrischen Bauteile wird 100-mal aufgenommen, während der Drehteller sich kontinuierlich dreht. Zwischen den Aufnahmen wird ein Intervall von einer Sekunde eingehalten, das durch die manuelle Konfiguration der Kameraeinstellungen festgelegt wurde. Während dieses Intervalls werden die 15 Sequencer-Sets automatisch durchgeschaltet, um die Effekte der Data Augmentation zu untersuchen.

Für den Trainingsdatensatz werden die Aufnahmen ohne Veränderung der Umgebungsvariablen gemacht: Hierbei dreht sich der Drehteller und es werden 100 Aufnahmen je Klasse gemacht.

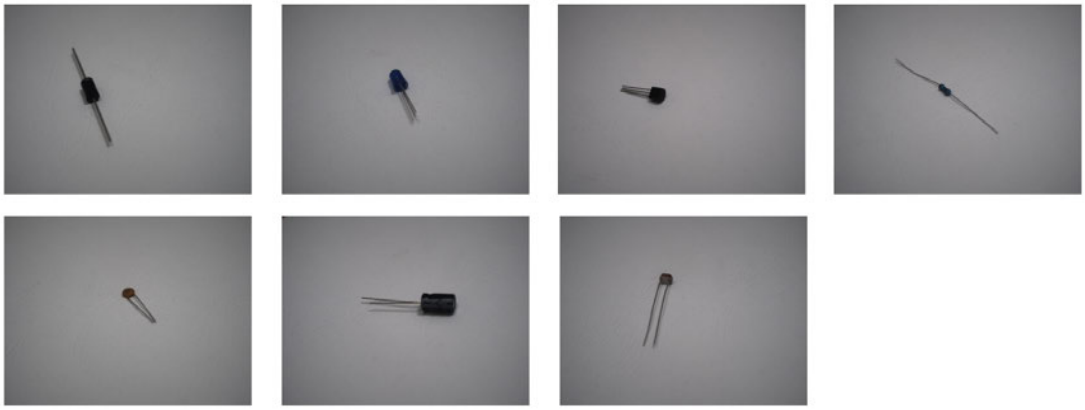


Abbildung 6.2: Auswahl an Aufnahmen der verschiedenen Klassen für den Versuch

Position	Winkel	Lichtfarbe
40 cm	0°	Weiß
40 cm	0°	Gelb
40 cm	0°	Blau
15 cm	0°	Weiß
15 cm	0°	Gelb
15 cm	0°	Blau
15 cm	50°	Weiß
15 cm	50°	Gelb
15 cm	50°	Blau

Tabelle 6.1: Kombinationen Versuchsaufbau für die Evaluierung

6.1.5 Änderungen für die Evaluierung

Für die Evaluierung wird der Versuchsaufbau angepasst: Der Drehteller wird zusätzlich mit schwarzem Papier und der roten Oberfläche des Drehtellers ausgestattet. Die Schreibtischlampe wird mit gelber und blauer Folie abgedeckt, und ihre Position wird vom Anwender umgestellt.

Für die Aufnahmen des Evaluierungsdatensatzes werden je 10 Aufnahmen pro Prüfobjekt in den in der Tabelle 6.1 zu findenden Kombinationen gemacht, jeweils mit den Untergründen des Prüfobjektes weiß, rot und schwarz.

Die Lichtintensität und der Schattenwurf wurden durch verschiedene Winkel und Entfernungen variiert, um das neuronale Netz unter schwierigeren Bedingungen zu testen.

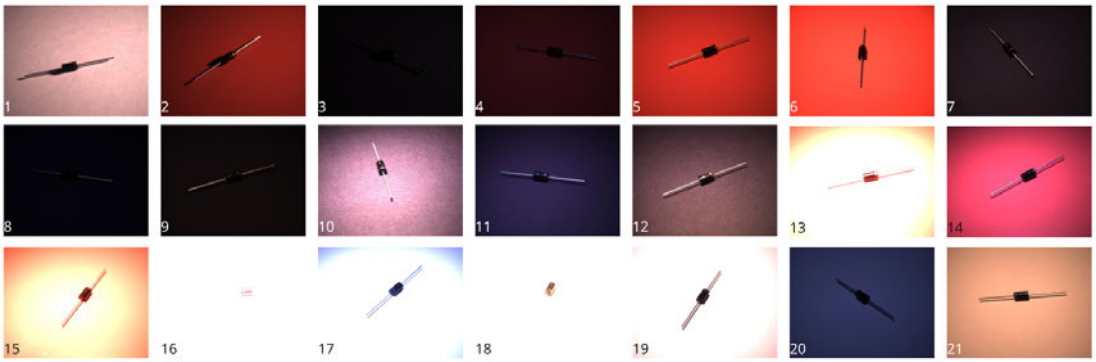


Abbildung 6.3: Auswahl an Evaluierungsaufnahmen einer Diode für den Versuch

Die neuen Lichtfarben sorgen außerdem für extreme Beispiele der Belichtung und der Hintergrund simuliert drei verschiedene Situationen.

Zunächst wurde auch eine Entfernung von 40 cm mit einem Winkel von 50° untersucht, aber es hat sich herausgestellt, dass diese erweiterte Entfernung keinen Unterschied bei den Aufnahmen erzielt hat. Daher sind je 10 Aufnahmen mit dieser Einstellung nicht im Evaluierungsdatensatz enthalten.

In der Abbildung 6.3 sind die Aufnahmen einer Diode unter den in der Tabelle 6.1 genannten Bedingungen zu sehen. Die Aufnahmen 1 bis 3 zeigen seitlich beleuchtete Szenen. Dabei weist die Aufnahme 3 einen schwarzen Hintergrund auf, wodurch die Diode kaum sichtbar wird. Die Aufnahmen 4-9 und 19-21 zeigen die Aufnahmen, bei denen die Lichtquelle einen Abstand von 40 cm hat, mit den jeweiligen zu erkennenden Untergründen.

Die restlichen Aufnahmen 10-18 zeigen die Aufnahmen mit der Lichtquelle im Abstand von 15 cm. Bei den Aufnahmen 5, 8, 11, 14, 17 und 20 wurde jeweils eine blaue Folie verwendet, während bei den Aufnahmen 6, 9, 12, 15, 18 und 21 jeweils eine gelbe Folie eingesetzt wurde.

Es ist zu erkennen, dass diese Aufnahmen alle Extremfälle darstellen, die vorkommen können. Dies ist wichtig, um die Robustheit der Modelle unter schwierigen Bedingungen zu testen. Durch die blaue Folie werden die Aufnahmen sehr dunkel und das Prüfobjekt ist schwer zu erkennen. Im Gegenzug sorgt die gelbe Folie dafür, dass die Aufnahmen deutlich heller wirken, genauso wie die nähere Einstellung der Lichtquelle.

Diese verschiedenen Situationen, in denen das Prüfobjekt zusätzlich auch noch in unterschiedlichen Positionen dargestellt wird, sollen für die Evaluierung reale Bedingungen wiedergeben und vor allem auch Extreme darstellen. Daher werden die trainierten Netze sehr stark beansprucht und es ist zu erwarten, dass sie mit dem Evaluierungsdatensatz keine guten Ergebnisse erzielen werden. Die entscheidende Frage ist, ob der Sequencerdatensatz bessere Ergebnisse liefert als die Methode der softwarebasierten Data Augmentation.

6.2 Erstellung und Aufteilung der Datensätze

Dieser Abschnitt beschreibt die Erstellung und Verteilung der Datensätze für das Experiment. Zunächst wird die Konfiguration des Sequencers erläutert, der für die automatisierte Aufnahme von Sequencer-Sets verantwortlich ist. Anschließend wird die Aufteilung der Daten in Trainings-, Validierungs- und Testdatensätze beschrieben.

Daraufhin wird die Anwendung von Data Augmentation vorgestellt, um den Vergleichsdatensatz zu erstellen. Hierbei wird die Methode so gewählt, dass sie den Sequencer-Aufnahmen möglichst ähnlich ist, um eine faire Vergleichsbasis zu schaffen. Abschließend werden die Evaluierungsdatensätze beschrieben, die unter verschiedenen Bedingungen aufgenommen wurden, um die Robustheit der Modelle zu testen.

Zusätzlich wird die Standardisierung der Datensätze erläutert, um die Pixelwerte in einen normierten Bereich zu transformieren, was die Konvergenz des Modells beschleunigt. Ebenso wird die Transformation der Datensätze durch den Dataloader beschrieben, der die Daten für das Training vorbereitet, indem er sie skaliert, standardisiert und in Tensoren umwandelt.

Konfiguration des Sequencers

Um die Data Augmentation mit dem Sequencer durchführen zu können, muss dieser zunächst konfiguriert werden. Ein Sequencer-Set besteht aus einer Zusammenstellung aus Kameraparametern und einem nächsten Set, welches darauf folgt.

Daher soll ein Prozess entstehen, welcher aus unterschiedlichen Sets besteht, welche nacheinander geschaltet werden können, wenn eine Aufnahme getätigt wird. Als Alternative könnte auch ein manuelles Durchschalten der Sets erfolgen. Diese Methode stellt

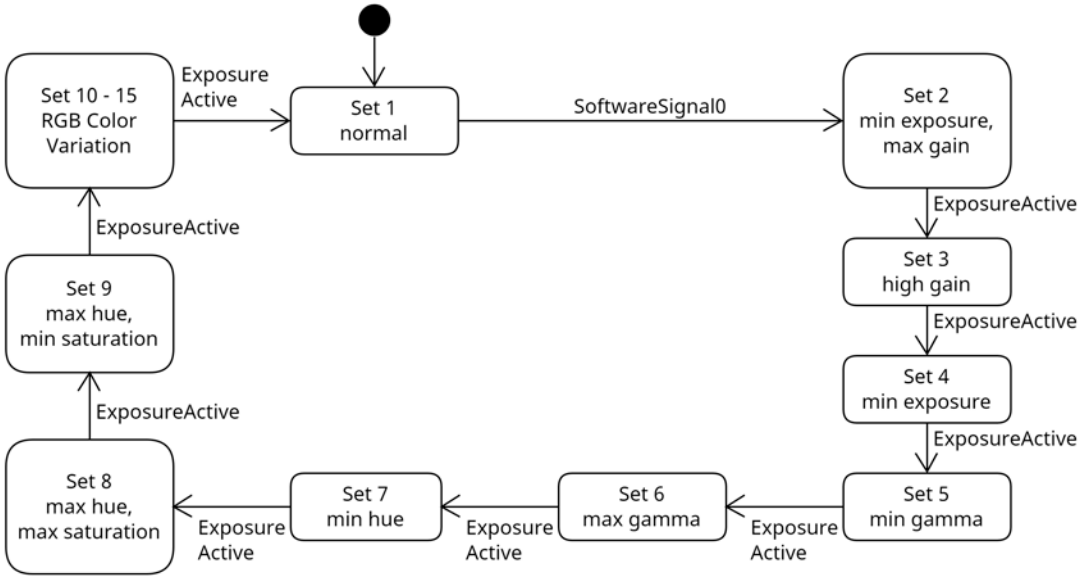


Abbildung 6.4: Zustandsdiagramm der Sequencer-Sets

sich aber als sehr aufwendig heraus, da 100 Aufnahmen pro Prüfobjekt mit jeweils 15 Sequencer-Sets aufgenommen werden müssen. Daher wird sich für ein automatisches Durchschalten der Sets entschieden.

In der Abbildung 6.4 ist das Zustandsdiagramm der Sequencer-Sets dargestellt. Dabei ist zu sehen, dass alle Sets nacheinander ablaufen, solange *ExposureActive* aktiv ist. Das heißt, dass die Kamera einen Frame empfangen hat und sorgt dafür, dass das nächste Sequencer-Set aktiviert werden kann. Wird das Set 1 erreicht, so kann der Prozess nur wiederholt werden, wenn das *SoftwareSignal0* gesetzt wird. Dieses erfolgt durch die Verwendung der Software, welche direkt auf die Kamera zugreift.

Die verwendeten Sequencer-Sets sind in Abbildung 6.5 am Beispiel einer Diode dargestellt. Das erste Set (oben links) zeigt eine Aufnahme ohne veränderte Parameter. Im zweiten Set wird der *Gain* bei minimaler Belichtungszeit maximiert. Das Ergebnis sind sichtbare Artefakte - Bereiche, in denen die Pixel zu hohe Werte aufweisen, da durch die geringe Lichtaufnahme weniger Informationen an den Sensor gelangen.

Das dritte und vierte Set veranschaulichen die Auswirkungen von Über- und Unterbelichtung. Bei maximalem *Gain* führt die Überbelichtung zu stark aufgehellten Bereichen, während die Unterbelichtung durch die minimale Belichtungszeit deutlich dunklere Aufnahmen erzeugt.

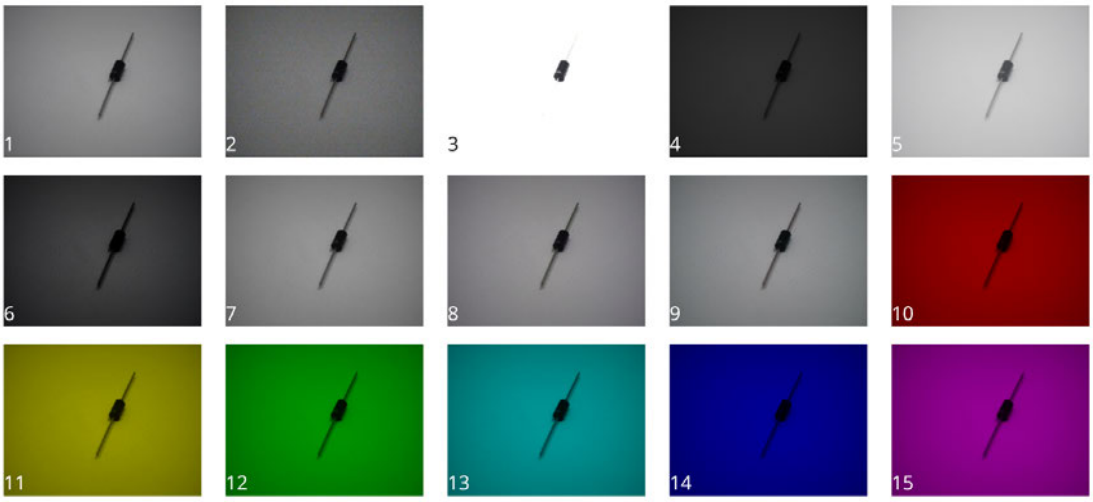


Abbildung 6.5: Auswahl an Aufnahmen der Sequencer-Sets für den Versuch

Die fünfte und sechste Aufnahme illustrieren den Einfluss des Gamma-Werts: Minimales Gamma erzeugt blasse, kontrastarme Bilder, während maximales Gamma zu intensiveren und kontrastreicheren Aufnahmen führt.

In der siebten Aufnahme wird ein Graustufenbild bei minimaler Sättigung erzeugt. Die achte und neunte Aufnahme zeigen die Auswirkungen von maximaler bzw. minimaler Farbtoneinstellung. Während die achte Aufnahme ein leicht pinkes Farbprofil aufweist, erscheint die neunte mit einem grünlichen Farbton, auch wenn die Unterschiede auf den ersten Blick nicht groß erscheinen.

Die übrigen Aufnahmen stellen jeweils einen oder zwei Farbkanäle dar, welche einzeln oder in Kombination wiedergegeben werden. Eine detaillierte Konfiguration des Sequencers ist im Anhang A.3 zu finden.

6.2.1 Aufteilung der Daten

Ein neuronales Netz wird mit einem Trainings-, Validierungs- und Testdatensatz trainiert und evaluiert. Dazu muss der Eingangsdatensatz in diese aufgeteilt werden. Es wird eine Aufteilung von 70/15/15 entschieden, sodass 100 Rohdaten pro Klasse für das Training, 15 für die Validierung und 15 Aufnahmen für den Test verwendet werden. Für den Sequencerdatensatz *seq* und den Vergleichsdatensatz Data Augmentation *dA* ist die jeweilige Anzahl 15-mal so hoch (durch die Sequencer-Sets).

Der Vergleichsdatensatz dA erhält zunächst nur die Rohdaten *raw* aus dem Sequencerdatensatz, wird dann aber durch Data Augmentation verändert, so dass die Sequencer-Sets nachgebildet werden können.

Für die Auswertungsdaten werden die Rohdaten aus dem Teil des Experiments übernommen und alle unverändert in die entsprechenden Ordner kopiert.

6.2.2 Anwendung von Data Augmentation

Der Vergleichsdatensatz wird die Effekte des Sequencerdatensatzes durch Data Augmentation optisch nachbilden. Dazu müssen die Methoden visuell und von den Pixelwerten her dem Sequencer-Verfahren ähnlich sein. Somit ist ein idealer Vergleich möglich, um diese beiden Methoden im Training und in der Evaluierung miteinander zu vergleichen. Hierbei wird die Data Augmentation vorerst direkt an den Datensätzen angewendet, damit gleiche Startbedingungen für beide Datensätze ermöglicht werden.

Es stehen zwei Methoden zur Data Augmentation zur Verfügung: eine direkte Veränderung durch PyTorch und einmal durch die Bibliothek OpenCV.

Zunächst soll OpenCV als mögliche Methode betrachtet werden, die Funktionen zur digitalen Bildverarbeitung bietet. Eine Funktion, um den HSV-Raum[1] zu verändern, ist in dem Listing 6.1 zu finden. Dieser zeigt beispielhaft, wie eine Konvertierung mit OpenCV aussehen kann. Hierbei wird das Eingangsbild mit Hilfe der gewünschten Werte so verändert, dass es der Aufnahme des Sequencers ähnelt.

Das folgende Listing 6.2 zeigt die Vorgehensweise über PyTorch, welches direkt Funktionen speziell für die Data Augmentation anbietet. Dabei ist zu beachten, dass die Funktionen je nach Verwendungszweck ausgewählt und später im fertigen Programm auch so verwendet werden müssen. In der Regel werden die gezeigten Funktionen zusammen und zufällig ausgeführt und nicht als Vorverarbeitung verwendet. Sie werden in der Transformation durch den *Dataloader* (siehe Kapitel 6.2.5) zusammen mit der Standardisierung und Skalierung verwendet und auch nicht gespeichert. In dieser Version für den Vergleich werden diese Methoden extrahiert und einzeln auf die Aufnahmen angewendet, damit ein Vergleich zwischen den Methoden fair bleibt. Daher stellen die Transformationen der Aufnahmen auch die gleichen optischen Versionen der Sequencer-Aufnahmen dar.

[1] Hue (Farbwert), Saturation (Farbsättigung) und Value (Helligkeit)

```python
def change_hsv_values(img, hue, saturation, value):
    # Convert to HSV
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    h = (h + hue) % 180  # Adjust hue

    # Adjust saturation and clip to valid range
    s = cv2.add(s, saturation)
    s = cv2.min(s, 255)
    s = cv2.max(s, 0)
    v = cv2.add(v, value)  # Adjust value
    v = cv2.min(v, 255)  # Ensure the values stay within the valid range

    # Merge the channels
    hsv_mod = cv2.merge([h, s, v])

    # Convert back to RGB
    img = cv2.cvtColor(hsv_mod, cv2.COLOR_HSV2RGB)

    return img
```

Listing 6.1: Data Augmentation durch OpenCV

```python
import torchvision.transforms.functional as F

# adjust brightness by 5
F.adjust_brightness(image, brightness_factor=5)

# adjust gamma by 0.4
F.adjust_gamma(image, gamma=0.4)

# convert to grayscale
F.to_grayscale(image, num_output_channels=3)

# adjust saturation by 0.09 and hue by 1.8
F.adjust_saturation(F.adjust_hue(image, hue_factor=0.09), saturation_factor=1.8)

# image to only red channel
F.to_pil_image(F.to_tensor(image) * torch.tensor([1, 0, 0]).view(3, 1, 1))
```

Listing 6.2: Data Augmentation mit PyTorch

Um die geeignete Methode für die Data Augmentation auszuwählen, wurden die Histogramme der Farbkanäle der Sequencer-Aufnahmen mit denen der Data Augmentation-Methoden von OpenCV und PyTorch verglichen. Die Analyse zeigt, dass die Data Augmentation mit PyTorch den Sequencer-Aufnahmen deutlich ähnlicher ist als die Methode mit OpenCV.

Abbildung 6.6 vergleicht die Histogramme einer Aufnahme, die mit verschiedenen Data Augmentation-Methoden verarbeitet wurde. Gezeigt wird das Sequencer-Set 8 (maxi-

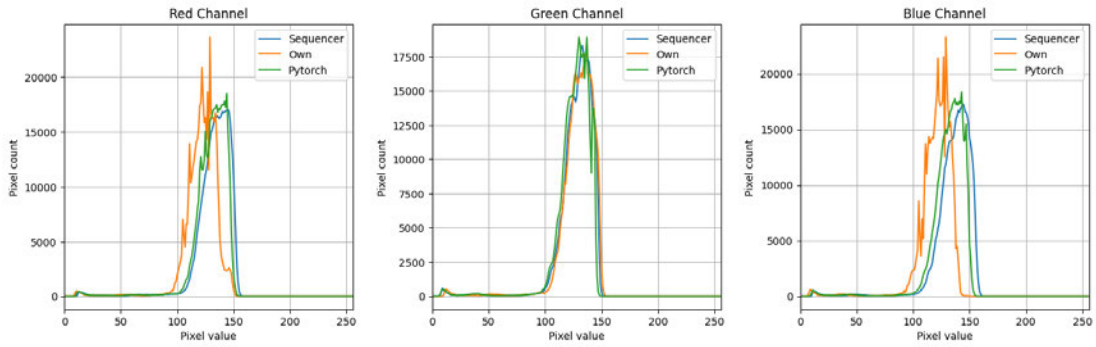


Abbildung 6.6: Histogrammvergleich: Sequencer, OpenCV und PyTorch Data Augmentation

male Sättigung und maximaler Farbton), da es die größte Variation aufweist und die Unterschiede zwischen den Methoden am besten sichtbar macht.

Die blaue Kurve repräsentiert die Aufnahme durch die Kamera mit dem Sequencer, die orange Kurve zeigt die Ergebnisse der Data Augmentation mit OpenCV und die grüne Kurve stellt die Ergebnisse mit PyTorch dar. Es ist erkennbar, dass die PyTorch-Methode deutlich näher an die Sequencer-Aufnahmen kommt. Daher wird PyTorch für die Data Augmentation verwendet. Die Ergebnisse mit OpenCV weichen stärker ab und könnten im Vergleich zu verfälschten Ergebnissen führen.

6.2.3 Evaluierungsdatensätze

Die Evaluierungsdatensätze werden gemäß den im Kapitel 6.1.5 beschriebenen Bedingungen erstellt. Diese stellen die Prüfobjekte mit anderen Umgebungsbedingungen dar und simulieren somit auch eine Anwendung unter Realbedingungen. Je Klasse werden 210 Aufnahmen erzeugt, was im Verhältnis zu den Datensätzen für Training und Test sehr groß ist. Daher wird auch erwartet, dass diese Aufnahmen mit den unterschiedlichen Belichtungen nicht sehr gut abschneiden, da diese Situationen dem Netz nicht bekannt sein sollten.

Der Datensatz besteht nur aus den Rohdaten und wird nicht zusätzlich durch Sequencer-Aufnahmen erweitert. Dieser Ansatz könnte für Sequencer-Modelle von Vorteil sein, da sie genau auf diese Art von Daten trainiert wurden.

```
def calculateMeanStd(path, datasetName):
    [...]

    image_paths = glob.glob(os.path.join(trainPath, "**", "*.png"), recursive=True)

    for imgPath in image_paths:
        img = Image.open(imgPath).convert("RGB")
        img = np.array(img, dtype=np.float32) / 255.0
        mean += img.mean(axis=(0, 1))
        std += img.std(axis=(0, 1))
        numberOfImages += 1

    mean /= numberOfImages
    std /= numberOfImages

    return mean, std
```

Listing 6.3: Berechnung des Mittelwertes und der Varianz

6.2.4 Standardisierung der Datensätze

Die Werte der Pixel einer Aufnahme bewegen sich auf den Kanälen Rot, Gelb und Blau je von 0 bis 255 (siehe auch Abbildung 6.6). Durch eine Standardisierung werden die Datensätze mit den Aufnahmen in einen Wertebereich umgewandelt, der das schnellere konvergieren des Modells ermöglich (vgl. Kapitel 5.2.1).

Das Listing 6.3 enthält die Funktion für diese Berechnung für den gesamten Datensatz. Diese Methode wird dann für jeden Datensatz *dA* (Vergleichsdatensatz Data Augmentation), *raw* (Rohdatensatz) und *seq* (Sequencerdatensatz) einmal durchgeführt, so dass jeder Datensatz entsprechend der jeweiligen Gewichtung der Pixelwerte normiert wird.

Es ist darauf zu achten, dass die Werte *std* und *mean* nur aus dem Trainingsdatensatz berechnet werden. Da das Netz über den Trainingsdatensatz lernt, sollten auch die neuen Daten über die gleiche Standardisierung verwendet werden. Im Idealfall sind diese Daten dem Netz nicht bekannt, daher ist es wichtig, dass auch diese Daten auf den Trainingsdatensatz normiert werden.

Das gleiche gilt dann auch für den Evaluierungsdatensatz, der ebenfalls durch die Standardisierung des Trainingsdatensatzes transformiert wird.

```
transform = transforms.Compose(
    [
        Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=mean, std=std),
    ]
)
```

Listing 6.4: Transformation der Datensätze

6.2.5 Transformation der Datensätze und der Dataloader

Der *Dataloader* ist eine Klasse in PyTorch, welche die Datensätze in den Speicher lädt und dann in *Batches* an das Modell übergibt. Dabei werden die jeweiligen Speicherorte der Datensätze angegeben, die dann auch als Label für die Klassen dienen. Dies ermöglicht ein einfaches Laden der Datensätze in den Speicher.

Durch eine Transformation des *Dataloader* kann man mit PyTorch eine Data Augmentation, Skalierung und auch Standardisierung durchführen.

Da sich diese Ausarbeitung auf die Data Augmentation konzentriert, wird die Data Augmentation manuell auf die Datensätze angewendet. Lediglich die Standardisierung mit den zuvor berechneten Werten aus dem Listing 6.3 und die Skalierung werden, wie im Listing 6.4 zu sehen, durchgeführt. Dabei werden alle Datensätze auf eine Größe von $224 \times 224 \times 3$ Pixel skaliert, um die Rechenleistung des neuronalen Netzes zu reduzieren.

Außerdem müssen die Datensätze in *Tensor* umgewandelt werden, damit sie von PyTorch verarbeitet werden können. Ein Tensor ist ein multidimensionales Array, das Daten, Modellparameter und Zwischenergebnisse speichert. Ein Farbbild kann als *3D-Tensor* mit den Dimensionen Höhe, Breite und Farbkanäle dargestellt werden. Diese Verwendung von *Tensors* ermöglicht eine effiziente Berechnung von Operationen auf den Daten und eignet sich daher für den Einsatz in neuronalen Netzen (nach [38]).

6.3 Modellierung und Training

Dieser Abschnitt beschreibt die Entwicklung und das Training eines neuronalen Netzes für das Experiment. Die notwendigen Schritte werden detailliert erläutert und anhand von Codebeispielen veranschaulicht.

Ziel ist es, ein neuronales Netz zu entwickeln, das eine Klassifikationsgenauigkeit von mindestens 95% erreicht. Diese hohe Genauigkeit ist notwendig, da der Evaluierungsdatensatz extreme Aufnahmebedingungen simuliert. Nur die leistungsfähigsten Netze haben unter diesen Bedingungen eine hohe Erfolgswahrscheinlichkeit.

Anhand des *TaylorNitschkeNet* wird die Vorgehensweise bei der Entwicklung eines neuronalen Netzes beschrieben. Die Netzarchitektur wird mit PyTorch implementiert, wobei die Konfiguration flexibel gestaltet ist, um verschiedene Eingabevariablen und Klassenanzahlen zu unterstützen. Details zur Modellierung und Implementierung werden im entsprechenden Abschnitt beschrieben.

Um geeignete Netze zu identifizieren, werden zahlreiche Kombinationen von Hyperparametern getestet. Dazu gehören unter anderem die Wahl der *Batch-Size*, des Optimierers, der Lernrate und der Initialisierungsmethoden. Die Auswahl dieser Hyperparameter basiert auf den Erkenntnissen aus dem Grundlagenkapitel 2.

Der Trainingsprozess umfasst mehrere Schritte, darunter die Vorbereitung der Datensätze, die Implementierung von Trainings- und Validierungsschritten sowie die Anwendung von Techniken wie *Early Stopping* zur Steigerung der Effizienz.

6.3.1 Hyperparameterauswahl

Für eine Auswahl von Hyperparametern werden die Erkenntnisse aus dem Grundlagenkapitel 2 genutzt. Es wurden Hyperparameter ausgewählt, welche für Aufgaben einer Klassifikation mit mehreren Klassen ausgelegt sind. Zu diesen Hyperparametern kommt die Auswahl der Netze und der Initialisierungsmethoden hinzu. Folgende Kombinationen von Hyperparametern werden untersucht:

- **Batch Sizes:** 16, 32
- **Modelle:** TaylorNitschkeNet, AlexNet, VGG16
- **Initialisierungsmethoden:** Xavier, Kaiming, Ones, Zeros, Uniform
- **Optimierer:** Adam, SGD, RMSprop
- **Lernraten:** 0,001, 0,0001
- **Scheduler:** MultiStepLR, ReduceLROnPlateau, StepLR

- **Verlustfunktionen:** KLDivLoss, NLLLoss, CrossEntropyLoss

Im ersten Durchlauf werden 70% der verfügbaren Daten verwendet, um die Rechenzeit zu reduzieren. Ziel ist es, zunächst eine Auswahl funktionierender Netzarchitekturen und Hyperparameter-Kombinationen zu identifizieren. Die Anzahl der Epochen wird auf 5 begrenzt, da erwartet wird, dass ein neuronales Netz bei dieser Datenmenge innerhalb dieser Zeitspanne eine präzise Klassifikation des Testdatensatzes erreichen kann. Zeigt ein Netz in diesem Zeitraum keine signifikanten Fortschritte, wird davon ausgegangen, dass die gewählte Kombination ungeeignet ist.

Um Netze auszuschließen, die bereits nach der ersten Epoche eine Genauigkeit von 1,0 erreichen, wird eine Abbruchbedingung implementiert. Diese verhindert, dass solche Netze weiter berücksichtigt werden, da sie auf *Overfitting* hinweisen.

Bei der Implementierung lieferten die Initialisierungsmethoden *Ones*, *Zeros* und *Uniform* keine brauchbaren Ergebnisse. Die initialisierten Gewichte lagen zu weit von den optimalen Werten entfernt, so dass kein effektiver Lernprozess stattfinden konnte. Auch das Modell *VGG16* zeigte innerhalb der 5 Epochen keine signifikante Verbesserung, da es im Vergleich zu den anderen untersuchten Modellen deutlich komplexer ist. Aus diesen Gründen werden die genannten Initialisierungsmethoden sowie das Modell *VGG16* in den weiteren Analysen nicht berücksichtigt.

6.3.2 Optimierer und Scheduler

Für das *TaylorNitschkeNet* wird der Optimierer *SGD* basierend auf den Empfehlungen aus [48] eingesetzt. Die Konfiguration umfasst ein Momentum von 0,9, die Nesterov-Variante und eine L2-Regularisierung mit einem Wert von 5×10^{-4}. Diese Einstellungen haben sich in der referenzierten Arbeit als effektiv erwiesen und werden daher übernommen.

Andere Optimierer wurden gemäß den Standardeinstellungen von PyTorch konfiguriert und mit den jeweiligen Lernraten für jedes Modell getestet.

Für die Scheduler werden die folgenden Einstellungen verwendet:
ReduceLROnPlateau: Die Lernrate wird um den Faktor 0,1 reduziert, wenn sich der Validierungsverlust über 10 Epochen nicht verbessert. Der Schwellenwert für die Verbesserung beträgt 10^{-4}.

StepLR: Die Lernrate wird ebenfalls um den Faktor 0,1 reduziert, wobei die Reduktion alle 10 Epochen erfolgt.

MultiStepLR: Die Lernrate wird bei definierten Meilensteinen (hier: nach 10 Epochen) um den Faktor 0,1 reduziert.

Diese Scheduler-Einstellungen wurden gewählt, um die Lernrate dynamisch anzupassen, falls das Modell keine weiteren Fortschritte erzielt.

6.3.3 TaylorNitschkeNet

Das *TaylorNitschkeNet* nach dem Kapitel 5.1.5 wird für diese Ausarbeitung als Modell verwendet. Dieses wird entsprechend der Abbildung 6.7 mit Hilfe von PyTorch implementiert, was im Listing 6.5 zu sehen ist. Die Abbildung zeigt, wie dieses Netz intern aufgebaut ist und kann im Code nachvollzogen werden.

Das neuronale Netz wird durch die Initialisierungsmethode der Klasse erstellt:
Es werden die Schichten *Convolutional Layers (conv)* und die *Pooling Layers (pool)* mit den folgenden *Fully Connected Layers (fc)* definiert (siehe Kapitel 2.4.1, 2.4.2 und 2.4.3). Diese werden mit den in der Abbildung 6.7 definierten Werten erzeugt. Der *Conv1 Layer* wird hier durch das Eingabebild der Größe $224 \times 224 \times 3$ bestimmt, gibt eine Größe von 30 aus (30@ 110×110), durchläuft das Eingabebild mit einem 6×6 Filter, der einen *Stride* von 2 und einem *Padding* von 1 hat. Alle Daten sind in der Abbildung enthalten und werden dann mit PyTorch erzeugt.

Dabei kann die Eingangsgröße in diesem Fall frei gewählt und als Input mit *in_channels* als Anzahl der Farbkanäle und *size* als Größe der Aufnahme festgelegt werden. Die Methode *get_conv_output* bestimmt dann dynamisch, wie viele Eingänge die Schicht *Fully Connected Layer* erwarten soll. Daher wird hier ein Beispiel-Tensor erzeugt und berechnet, wie viele Ausgänge dieser erzeugt.

Dadurch entstehen die letzten zwei Schichten, welche dann die Klassifikation mit der definierten Anzahl an Klassen als Ausgang vornehmen. Die Methode *Forward* definiert, wie die Eingabedaten durch das Netz fließen, um eine Vorhersage zu erzeugen.

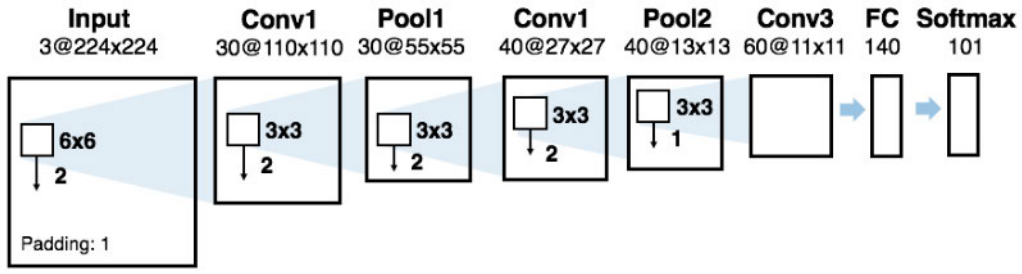


Abbildung 6.7: Darstellung der Netzarchitektur TaylorNitschkeNet nach [48]

```python
class TaylorNitschkeNet(nn.Module):
    def __init__(self, num_classes: int, in_channels: int, size: int):
        super(TaylorNitschkeNet, self).__init__()
        # First layer
        self.conv1 = nn.Conv2d(in_channels=in_channels, out_channels=30,
                               kernel_size=6, stride=2, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2)

        # Second layer
        self.conv2 = nn.Conv2d(in_channels=30, out_channels=40,
                               kernel_size=3, stride=2, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=3, stride=2)

        # Third layer
        self.conv3 = nn.Conv2d(in_channels=40, out_channels=60,
                               kernel_size=3, stride=1, padding=1)

        # Calculate the size of the feature map after the last pooling layer
        self._to_linear = None
        self._get_conv_output(in_channels, size)

        # Fully connected layer
        self.fc1 = nn.Linear(in_features=self._to_linear, out_features=140)

        # Output layer
        self.fc2 = nn.Linear(in_features=140, out_features=num_classes)

    def _get_conv_output(self, in_channels, size):
        # Create a dummy input tensor with the same dimensions as your input images
        x = torch.randn(1, in_channels, size, size)
        x = self.conv1(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.pool2(x)
        x = self.conv3(x)
        self._to_linear = x.numel()

    def forward(self, x: Tensor) -> Tensor:
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = F.relu(self.conv3(x))
        x = x.view(x.size(0), -1)  # Flatten the feature map
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Listing 6.5: Nachbildung des TaylorNitschkeNet

6.3.4 Ablauf des Trainings

Das Training des neuronalen Netzes erfolgt in mehreren Schritten, die hier erläutert werden. Die im Listing 6.6 zu findende Methode *train* der Klasse *NeuronalNetClass* verwendet die Initialisierungs-Hyperparameter, um ein Neuronalnetz zu trainieren. Vor dem Training wird sichergestellt, dass Zufallswerte zurückgesetzt werden und dann werden alle Gewichte gemäß der Initialisierungsmethode gesetzt.

Dann werden die *Dataloader* für das Training, die Validierung und den Test mit den dazugehörigen Parametern der *Batch Size* und der Transformation mit Standardisierung und Skalierung erstellt. Mit diesen wird dann das Training gestartet, welches dann Ergebnisse liefert, die dann für die Auswertung und Erstellung der Leistungsmetriken genutzt werden. Daraus wird dann das *Ranking* (vgl. Gleichung 5.1.6) berechnet, anhand dessen im Folgenden alle Modelle bewertet werden. Dabei werden immer die letzten Werte aus der Liste der Validierungsergebnisse für Genauigkeit und Verlust verwendet. Abschließend werden alle Daten gespeichert, visualisiert und ausgegeben.

Innerhalb der Methode *engine.train* befindet sich nach dem Listing 6.7 die Funktion, die die einzelnen Trainings- und Validierungsschritte pro Epoche durchführt.

Zuerst wird der Prozess auf die Grafikkarte geladen, da diese Prozesse schneller verarbeiten kann. Es wird die Funktion *earlyStopper* initialisiert, die das Training abbricht, wenn sich der Validierungsverlust über die eingestellte Anzahl von Epochen nicht außerhalb der Grenzen verändert hat (mehr dazu im Kapitel 6.3.5).

Für jede Epoche wird der Trainingsschritt mit dem zugehörigen *dataloader* und den Hyperparametern gestartet. Dazu gehören u. a. das Modell und die Verlustfunktion sowie der Optimierer. Anschließend wird der Validierungsschritt mit dem Validierungsdatensatz und den Hyperparametern gestartet.

Da im Experiment zwei Arten von *Schedulers* verwendet werden, muss hier unterschieden werden und somit die Anpassung der Lernrate angepasst werden.

Anschließend werden die Ergebnisse zwischengespeichert und überprüft, ob die Bedingungen des *EarlyStoppers* einen vorzeitigen Abbruch zulassen.

```python
def train(self, log: logging.Logger, dataSet: str) -> List[Dict[str, List[float]]]:
    torch.manual_seed(42)

    # reset the weights for the current training
    self.initializeWeights()

    # prepare the dataloaders with the transformation and batch size
    (trainDataLoader, valDataLoader, testDataLoader, self.classNames) =
        createDataLoaders(...)

    result = {}
    runName = f"{self.name}_{dataSet}"

    # create a SummaryWriter for logging the progress
    writer = SummaryWriter(self.writerDir)

    # train the model with all the hyperparameters
    result = engine.train(...)

    # calculate the performance metrics with the trained model
    precision, recall, f1Score, confusionMatrix = calculatePerformanceMetrics(...)

    # calculate the ranking from the values from the training and test
    ranking = (
        0.4 * precision + 0.1 * recall + 0.3 * result["validationAcc"][-1]
        + 0.2 * (1.0 - result["validationLoss"][-1])
    )

    [...]   # save all the data into results

    # plot the confusion matric and the curves from the results
    plotConfusionMatrix(...)
    plotCurves(...)

    # save the results into a csv for late usage
    saveMetrics(...)

    self.results.append(result)
    writer.close()

    return self.results
```

Listing 6.6: Trainingsmethode der Klasse NeuronalNet

6.3.5 EarlyStopper

Die Klasse *EarlyStopper* befindet sich im Listing 6.8. Sie wird mit den globalen Einstellungen für das *delta* aus der Änderung des Validierungsverlustes über die Epochen initialisiert. Ist der aktuelle Wert größer als der zum *delta* addierte alte Wert, wird ein *counter* inkrementiert. Wenn die eingestellte *patience* erreicht ist, gibt die Funktion *earlyStop* ein *true* zurück, das dafür sorgt, dass das Training abgebrochen wird. Dadurch wird Rechenzeit gespart. Zusätzlich soll ein Training auch abgebrochen werden können, wenn für die eingestellte Anzahl von Epochen ein perfekter Verlust von 0,0

```python
def train(...) -> Dict[str, List]:
    torch.manual_seed(42)
    # Put model on target device
    model = model.to(device)
    # Setup early stopping
    earlyStopper = EarlyStopper(...)

    for epoch in range(epochs):
        # training with model and train dataloader
        trainLoss, trainAcc = trainStep(lossFn=lossFn, optimizer=optimizer, ...)

        # validation with model and validation dataloader
        validationLoss, validationAcc = validationStep(lossFn=lossFn, ...)

        # if ReduceLROnPlateau, step with validation loss
        if isinstance(scheduler, torch.optim.lr_scheduler.ReduceLROnPlateau):
            scheduler.step(validationLoss)
        else:   # step with epoch
            scheduler.step()

        # add the results to the list for plotting
        results["trainLoss"].append(trainLoss)
        [...]   # for trainAcc, valLoss and valAcc

        [...]   # add the results to the tensorboard's SummaryWriter

        # early stopping function
        if earlyStopper.earlyStop(validationLoss):
            break

    return results
```

Listing 6.7: Trainingsprozess

erreicht wird. Wenn dieser Wert erreicht ist, wird das Training abgebrochen, aber wenn sich der Verlust danach verschlechtert, wird dieser *counter* wieder zurückgesetzt.

6.3.6 Trainingsschritt

Ein Trainingsschritt wird im folgenden Listing 6.9 dargestellt. In diesem Schritt wird das Modell in den Trainingsmodus versetzt und die Datensätze werden durchlaufen. Die Daten werden auf die Grafikkarte geladen und dann wird ein *Forward Pass* durch das Modell durchgeführt. Dieser sorgt dafür, dass die Daten durch das Modell fließen und eine Vorhersage getroffen wird.

Anschließend wird die Verlustfunktion berechnet und die Gewichte des Modells angepasst. Dabei wird unterschieden, welche Verlustfunktion verwendet wird, da die Varianten *NLLLoss* und *KLDivLoss* eine $log\_softmax$ Funktion benötigen, um die Verlustfunktion korrekt zu berechnen. Bei der *KLDivLoss* muss zusätzlich ein *one-hot encoding*

```python
class EarlyStopper:
    def __init__(self, patience=1, minDelta=0, maxPerfect=2):
        self.patience = patience
        self.minDelta = minDelta
        self.counter = 0
        self.perfectCounter = 0
        self.minValidationLoss = float("inf")
        self.maxPerfect = maxPerfect

    def earlyStop(self, validationLoss):
        if validationLoss < self.minValidationLoss:
            self.minValidationLoss = validationLoss
            self.counter = 0
            self.perfectCounter = 0
        elif validationLoss >= (self.minValidationLoss + self.minDelta):
            self.counter += 1
            if self.counter >= self.patience:
                return True
        # if the validation loss is 0.0 for more than maxPerfect epochs, stop
        if validationLoss == 0.0:
            self.perfectCounter += 1
            if self.perfectCounter >= self.maxPerfect:
                return True
        return False
```

Listing 6.8: Klasse EarlyStopper

verwendet werden, um die Daten in die richtige Form zu bringen. Diese wandelt die Daten in eine binäre Form um, um die Differenz zwischen den beiden Verteilungen zu berechnen. Diese muss nach der Berechnung des Verlustes wieder in die ursprüngliche Form gebracht werden, um die Genauigkeit zu berechnen.

Anschließend wird der Schritt *Backward* durchgeführt, um die Gradienten zu berechnen und die Gewichte des Modells anzupassen. Schließlich wird die Genauigkeit berechnet und die Ergebnisse werden in einer Liste gespeichert.

Das Modell wird auf die Grafikkarte geladen und die Daten werden durch das Modell geloopt. Anschließend wird die Verlustfunktion berechnet und die Modellgewichte werden angepasst. Schließlich wird die Genauigkeit berechnet und die Ergebnisse werden in einer Liste gespeichert.

Die Ergebnisse werden dann auf die Anzahl der *Batches* angepasst, um den Durchschnitt der Verluste und Genauigkeiten pro *Batch* zu erhalten.

```python
def trainStep(...) -> Tuple[float, float]:
    model.train()   # Put model in train mode

    for batch, (X, y) in enumerate(dataLoader): # Loop through data loader data
        batches
        X, y = X.to(device), y.to(device)   # Send data to target device

        yPred = model(X)     # Forward pass

        # Apply log_softmax if loss function is NLLLoss or KLDivLoss
        if isinstance(lossFn, torch.nn.NLLLoss):
            yPred = F.log_softmax(yPred, dim=1)
        elif isinstance(lossFn, torch.nn.KLDivLoss):
            yPred = F.log_softmax(yPred, dim=1)
            y = F.one_hot(y, num_classes=yPred.size(1)).float()

        # Calculate  and accumulate loss
        loss = lossFn(yPred, y)
        trainLoss += loss.item()

        optimizer.zero_grad()   # Optimizer zero grad

        loss.backward()     # Loss backward

        optimizer.step()    # Optimizer step

        # Calculate and accumulate accuracy
        yPredClass = torch.argmax(yPred, dim=1)

        # Ensure y is in the correct shape for accuracy calculation
        if isinstance(lossFn, torch.nn.KLDivLoss):
            y = torch.argmax(y, dim=1)

        trainAcc += (yPredClass == y).sum().item() / len(yPred)

    # Adjust metrics to get average loss and accuracy per batch
    trainLoss = trainLoss / len(dataLoader)
    trainAcc = trainAcc / len(dataLoader)

    return trainLoss, trainAcc
```

Listing 6.9: Trainingsschritt einer Epoche

6.3.7 Validierungsschritt

Der Validierungsschritt, der in jeder Epoche durchgeführt wird, ist im folgenden Listing 6.10 dargestellt. Das Modell wird in den Evaluierungsmodus versetzt und die Datensätze werden durchlaufen.

Es erfolgt ein *Forward Pass* durch das Modell und die Verlustfunktion wird berechnet. Hierbei wird auch durch die Art der Verlustfunktion unterschieden, ob ein *one-hot encoding* durchgeführt werden muss. Anschließend wird die Genauigkeit berechnet und die Ergebnisse werden in einer Liste gespeichert.

```python
def validationStep(...) -> Tuple[float, float]:
    model.eval()    # Put model in eval mode

    # Turn on inference context manager
    with torch.inference_mode():
        for batch, (X, y) in enumerate(dataLoader): # Loop through DataLoader batches
            # Send data to target device
            X, y = X.to(device), y.to(device)

            yPred = model(X)    # Forward pass

            # Apply log_softmax if loss function is NLLLoss or KLDivLoss
            if isinstance(lossFn, torch.nn.NLLLoss):
                yPred = F.log_softmax(yPred, dim=1)
            elif isinstance(lossFn, torch.nn.KLDivLoss):
                yPred = F.log_softmax(yPred, dim=1)
                y = F.one_hot(y, num_classes=yPred.size(1)).float()

            # Calculate and accumulate loss
            loss = lossFn(yPred, y)
            validationLoss += loss.item()

            # Calculate and accumulate accuracy
            yPredClass = torch.argmax(yPred, dim=1)

            # Ensure y is in the correct shape for accuracy calculation
            if isinstance(lossFn, torch.nn.KLDivLoss):
                y = torch.argmax(y, dim=1)

            validationAcc += (yPredClass == y).sum().item() / len(yPred)

    # Adjust metrics to get average loss and accuracy per batch
    validationLoss = validationLoss / len(dataLoader)
    validationAcc = validationAcc / len(dataLoader)

    return validationLoss, validationAcc
```

Listing 6.10: Validierungsschritt einer Epoche

Der Unterschied zum Trainingsschritt besteht darin, dass hier keine Gewichte angepasst werden, sondern nur die Genauigkeit und der Verlust berechnet werden.

6.4 Auswertung der Ergebnisse

Die Ergebnisse des Trainings werden anhand von Leistungsmetriken wie Genauigkeit, Verlust, *Precision* und *Recall* analysiert. Diese Metriken werden verwendet, um die Qualität der trainierten Modelle und ihre Eignung für die jeweilige Aufgabe zu bewerten.

Im Anschluss erfolgt die Evaluierung der besten Modelle, die auf Basis eines Rankings ausgewählt werden. Diese Modelle werden mit einem speziellen Evaluationsdatensatz getestet, der reale und extreme Bedingungen simuliert. Ziel ist es, die Robustheit und Generalisierbarkeit der Modelle unter verschiedenen Szenarien zu überprüfen.

Schließlich wird untersucht, welche Kameraparameter den größten Einfluss auf den *Ranking*-Wert haben. Diese Analyse hilft, die wichtigsten Faktoren zu identifizieren, die die Leistung der Modelle beeinflussen und liefert wertvolle Erkenntnisse für zukünftige Optimierungen.

6.4.1 Test des Modells

Zum Testen eines neuronalen Netzes wird ein Testdatensatz benötigt. Dieser wird im Listing 6.11 verwendet, um zu überprüfen, ob das Modell unbekannte Datensätze korrekt klassifizieren kann. Dies geschieht mit einem *Forward Pass* durch das Modell, der eine Vorhersage erzeugt. Anschließend werden die wahren und die vorhergesagten Werte in Listen gespeichert.

Diese Listen werden dann in Tensoren umgewandelt, um zunächst die True Positives (TP), False Positives (FP) und False Negatives (FN) zu berechnen. Daraus ergeben sich die Metriken *precision*, *recall* und $F1_{Score}$. Anschließend wird die *Confusion* Matrix berechnet, um die Ergebnisse zu visualisieren.

6.4.2 Evaluierung der neuronalen Netze

Die Evaluierung basiert auf den zehn besten Hyperparameter-Kombinationen aus dem Training. Dabei werden alle verfügbaren Daten berücksichtigt, einschließlich der Trainings-, Validierungs-, Test- und Evaluierungsdatensätze.

Zunächst werden die zehn besten neuronalen Netze aus einer Datei geladen, in der sie zwischengespeichert wurden. Für jede dieser Hyperparameter-Kombinationen wird ein neues Netz erstellt, das mit den entsprechenden Datensätzen trainiert wird.

Die Datensätze werden gemäß Kapitel 6.2 vorbereitet, wobei zusätzlich der Evaluierungsdatensatz geladen und ohne Data Augmentation verarbeitet wird.

```python
def calculatePerformanceMetrics(...) -> Dict[str, float]:
    # iterate over the test dataloader and make predictions
    for images, labels in testDataloader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)     # Forward pass
        _, predicted = torch.max(outputs, 1)

        # extend the lists with the true and predicted values
        yTrue.extend(labels.cpu().numpy())
        yPred.extend(predicted.cpu().numpy())

    # convert lists to tensors for calculation
    yTrueTensor = torch.tensor(yTrue)
    yPredTensor = torch.tensor(yPred)

    for cls in range(numClasses):   # calculate precision, recall, f1 score
        TP = ((yPredTensor == cls) & (yTrueTensor == cls)).sum().item()
        FP = ((yPredTensor == cls) & (yTrueTensor != cls)).sum().item()
        FN = ((yPredTensor != cls) & (yTrueTensor == cls)).sum().item()

        clsPrecision = TP / (TP + FP) if TP + FP > 0 else 0
        clsRecall = TP / (TP + FN) if TP + FN > 0 else 0
        clsF1 = (2 * (clsPrecision * clsRecall) / (clsPrecision + clsRecall)
            if (clsPrecision + clsRecall) > 0 else 0
        )

        precision.append(clsPrecision)
        recall.append(clsRecall)
        f1.append(clsF1)

    # average metrics across all classes
    precision = np.mean(precision)
    recall = np.mean(recall)
    f1 = np.mean(f1)

    # calculate confusion matrix
    confusionMatrix = torch.zeros(numClasses, numClasses)
    for t, p in zip(yTrueTensor, yPredTensor):
        confusionMatrix[t, p] += 1

    return (precision, recall, f1, confusionMatrix)
```

Listing 6.11: Berechnung der Leistungsmetriken

Nach Abschluss des Trainings wird die Evaluierungsmethode aus Listing 6.12 aufgerufen. Dabei werden die Evaluierungsdaten in den *dataloader* geladen und an die Methode *calculatePerformanceMetrics* übergeben.

Die daraus resultierenden Leistungsmetriken werden mit den Validierungsdaten aus dem Training kombiniert, um ein neues *Ranking* zu berechnen. Abschließend werden die Ergebnisse, einschließlich der *Confusion* Matrix, gespeichert und für den Vergleich bereitgestellt.

```python
def evaluation(self, dataSet: str) -> List[Dict[str, List[float]]]:
    [...] # Setup the dir files
    evalData = datasets.ImageFolder(evalDir, transform=self.transform)

    evalDataloader = DataLoader(...)

    precision, recall, f1Score, confusionMatrix = calculatePerformanceMetrics(
        testDataloader=evalDataloader,
        ...
    )

    evalRanking = (     # calculate new ranking
        0.4 * precision + 0.1 * recall + 0.3 * existingResult["validationAcc"][-1]
        + 0.2 * (1.0 - existingResult["validationLoss"][-1])
    )

    [...] # save the new evaluation results

    plotConfusionMatrix(...)

    self.results.append(existingResult)

    return self.results
```

Listing 6.12: Evaluierungsmethode der Klasse NeuronalNet

6.4.3 Auswertung der Kameraparameter für die Data Augmentation

Um die Kameraparameter zu identifizieren, die den größten Einfluss auf die Leistungsfähigkeit des neuronalen Netzes haben, wird für jedes Sequencer-Set ein separater Datensatz erstellt. Jeder dieser Datensätze enthält sowohl die unveränderten Aufnahmen als auch die Aufnahmen, die mit den jeweiligen Kameraparametern modifiziert wurden. Die Validierungs-, Test- und Bewertungsdatensätze bleiben unverändert und bestehen ausschließlich aus den Rohdaten.

Die Analyse erfolgt mit den zehn besten Modellen, die nach der Evaluierung das höchste *Ranking* erzielt haben. Für jedes zu untersuchende Kameraparameter-Set wird ein eigener Datensatz erstellt, der entsprechend standardisiert und transformiert wird.

Im Gegensatz zu den vorherigen Experimenten wird hier ausschließlich ein Datensatz transformiert, ohne einen Vergleich zu den Rohdaten oder einer alternativen Data-Augmentation-Methode durchzuführen.

Die Ergebnisse dieser Analyse werden abschließend in einem Diagramm visualisiert, das die *Ranking*-Werte der Modelle nach der Evaluierung für die verschiedenen Kameraparameter darstellt.

7 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Experimente vorgestellt und analysiert. Zunächst werden die Ergebnisse des ersten Experiments zur Bestimmung der optimalen Hyperparameter vorgestellt, die als Grundlage für die weiteren Experimente dienen. Anschließend erfolgt eine detaillierte Analyse der Auswerteergebnisse einschließlich eines Vergleichs mit den vorherigen Experimenten, um die gewonnenen Erkenntnisse zu diskutieren.

Darüber hinaus werden die effektivsten Kameraparameter identifiziert und Empfehlungen für Parameter gegeben, die keinen signifikanten Einfluss auf die Ergebnisse haben. Abschließend wird überprüft, inwieweit die gestellten Anforderungen erfüllt wurden und Abweichungen oder alternative Umsetzungen werden erläutert.

7.1 Ergebnisse der neuronalen Netze

In diesem Abschnitt werden die Ergebnisse der Experimente dargestellt und diskutiert.

Für die Experimente stehen die Datensätze Data Augmentation (dA) und Sequencer (seq) zur Verfügung, die jeweils 7350 Aufnahmen im Trainingssplit enthalten, wobei die Data Augmentation bereits integriert ist. Zusätzlich umfassen beide Datensätze 105 Aufnahmen für die Validierung und den Test sowie 1463 Aufnahmen für die Evaluation.

Zum Vergleich wird der Rohdatensatz (raw) herangezogen, der im Trainingssplit 490 Aufnahmen enthält. Die Aufteilung für Validierung (105), Test (105) und Evaluation (1463) ist identisch zu den anderen Datensätzen.

Hyperparameter	Werte
dataSets	raw, seq, dA
learningRates	0,001, 0,0001
batchSizes	16, 32, 64
models	TaylorNitschkeNet, alexnet, vgg16
lossFunctions	KLDivLoss, NLLLoss, CrossEntropyLoss
optimizers	Adam, SGD, RMSprop
schedulers	MultiStepLR, ReduceLROnPlateau, StepLR
initializationMethods	xavier, kaiming

Tabelle 7.1: Zusammenfassung der Hyperparameter und ihrer Werte

7.1.1 Ermittlung der leistungsfähigsten neuronalen Netze

Dieser Teil des Experiments sollte zunächst zeigen, welche Kombinationen von Hyperparametern überhaupt verwendet werden können. Daher wurde ein ausführlicher *Grid Search* aller vorgeschlagenen Hyperparameter durchgeführt.

Im ersten Teil des Experiments wurde das Training auf 70% der verfügbaren Daten und 5 Epochen beschränkt, um die Rechenzeit signifikant zu reduzieren. Diese Einschränkung verringert die Anzahl der pro Epoche verarbeiteten Trainingsbeispiele und verkürzt somit die Rechenzeit pro Epoche.

Da der Scheduler erst ab der zehnten Epoche aktiv wird, konnte er in diesem verkürzten Training nicht genutzt werden. Stattdessen wurde mit einer festen Lernrate gearbeitet. Es zeigte sich jedoch, dass diese Einschränkung nur das erste Training betraf.

Nach der Identifikation der besten Modelle wurde das Training mit dem vollständigen Datensatz und einer erhöhten Epochenanzahl von 20 wiederholt. Dadurch konnten die Scheduler-Einstellungen korrekt angewendet werden, was zu einer verbesserten Optimierung der Modelle führte.

Im Vergleich zu einem vollständigen Training mit 100% der Daten und 20 Epochen konnte die Rechenzeit auf ca. 25% reduziert werden, was eine effiziente Vorauswahl der besten Hyperparameter-Kombinationen ermöglicht. Um die Anzahl der möglichen Kombinationen zu berechnen, die untersucht wurden, müssen zunächst die Hyperparameter betrachtet werden. Diese können der Tabelle 7.1 entnommen werden.

Daraus ergibt sich eine kombinierte Anzahl von Netzen von 972 Kombinationen, die trainiert werden müssen. Die Rechenzeit wird auch durch die Wahl der Hardware beeinflusst, die für diesen Versuch eine NVIDIA GeForce RTX 2070 war.

Von den 972 getesteten Kombinationen wurden 531 Modelle ausgeschlossen, da sie entweder nach der ersten Epoche eine ungewöhnlich hohe Genauigkeit von 1,0 erreichten oder einen zu hohen Verlust aufwiesen. Modelle mit einer Genauigkeit von 1,0 wurden aufgrund der definierten Abbruchbedingung vorzeitig abgebrochen, was in den meisten Fällen auf ein starkes *Overfitting* hindeutet. Selbst wenn es sich nicht um Overfitting handelt, sind solche Modelle für diesen Versuch ungeeignet, da sie keinen aussagekräftigen Vergleich zwischen den Methoden der Data Augmentation ermöglichen.

Darüber hinaus führten einige Hyperparameter-Kombinationen zu sehr hohen Verlusten, was darauf schließen lässt, dass das Modell die Daten nicht erfolgreich lernen konnte. Mögliche Ursachen für diese Ergebnisse könnten numerische Instabilitäten, ungeeignete Hyperparameter-Kombinationen oder eine zu hohe Lernrate sein.

Da diese Ergebnisse nicht plausibel sind und auf Probleme mit den jeweiligen Hyperparameter-Kombinationen hindeuten, wurden diese Modelle von der weiteren Analyse ausgeschlossen. Unter den 531 verworfenen Modellen befinden sich auch solche, die nach fünf Epochen ein negatives *Ranking* aufweisen.

Die Abbildung 7.1 zeigt die Verteilung der *Ranking*-Werte für die verschiedenen Hyperparameter als Boxplot. Es wird deutlich, dass das Modell *TaylorNitschkeNet* im Durchschnitt die besten *Ranking*-Ergebnisse erzielt.

Besonders hervorzuheben sind die Optimierer *SGD*, eine *Batch Size* von 32 sowie eine Lernrate von 0,001, die im Durchschnitt stabilere und konsistentere Ergebnisse liefern. Zudem zeigt sich, dass der Sequencerdatensatz *seq* tendenziell bessere *Ranking*-Werte aufweist, während die anderen Datensätze häufiger Ausreißer mit negativen *Ranking*-Werten enthalten.

Nach der Ermittlung der zehn besten Modellkonfigurationen wurden diese mit dem vollständigen Trainingsdatensatz und einer Epochenanzahl von 20 erneut trainiert. Dadurch ist ein späterer Vergleich zur Evaluierung möglich, da diese mit den gleichen Parametern trainieren soll.

Die Tabelle 7.2 zeigt die besten Kombinationen von Hyperparametern aus dem Experiment. Der Index ist eine fortlaufende Nummer, die dem Netz im ersten Durchlauf

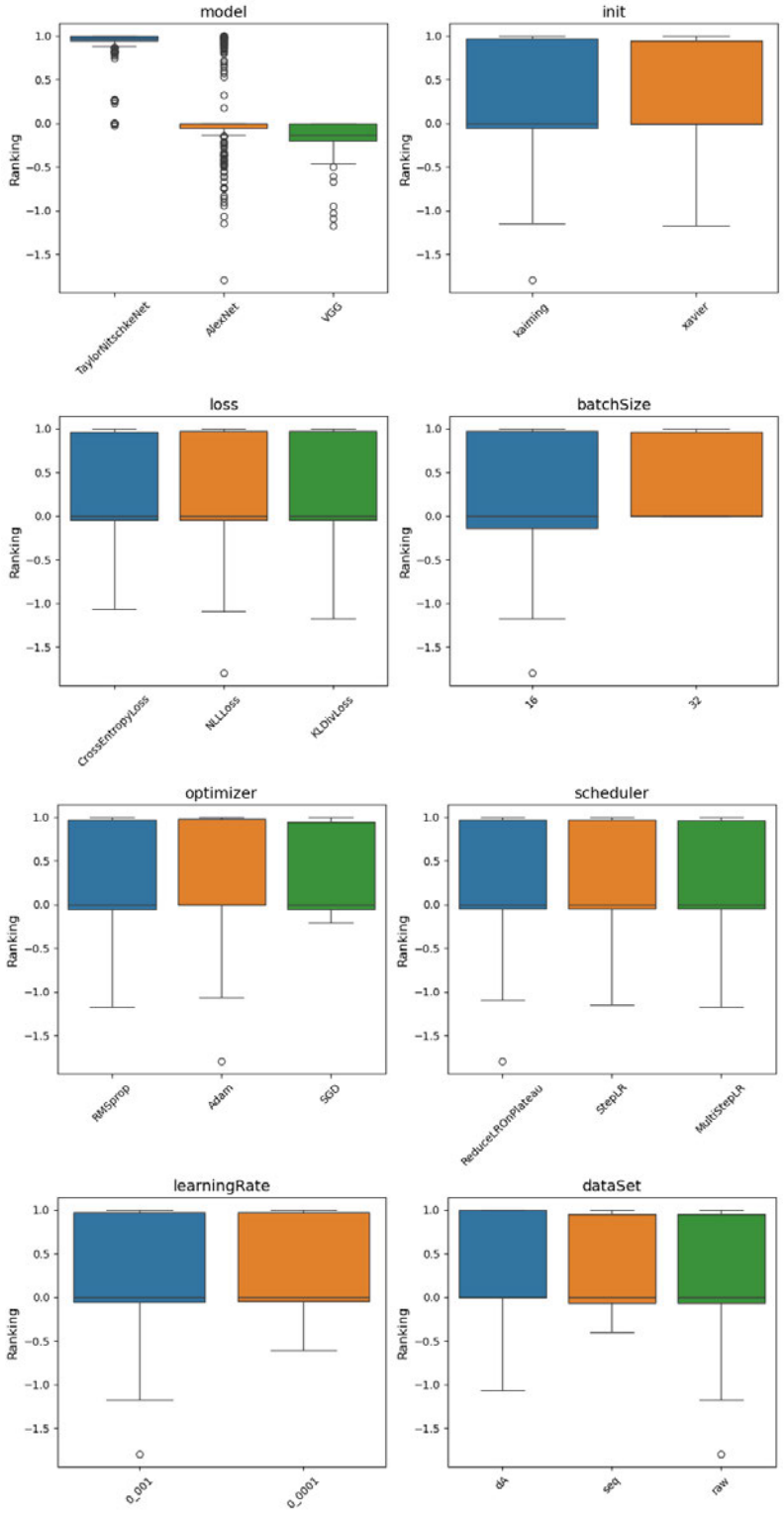


Abbildung 7.1: Hyperparameter und ihr *Ranking* im Versuch 1

#	Model	Init	Optimizer	Scheduler	Loss	BatchSize	LearningRate	DataSet	Ranking	Index
1	AlexNet	xavier	RMSprop	MultiStepLR	KLDivLoss	32	0,0001	dA	1,0	227
2	TaylorNitschkeNet	kaiming	Adam	ReduceLROnPlateau	KLDivLoss	32	0,001	seq	1,0	354
3	TaylorNitschkeNet	kaiming	RMSprop	StepLR	CrossEntropyLoss	16	0,001	dA	1,0	419
4	TaylorNitschkeNet	kaiming	RMSprop	ReduceLROnPlateau	NLLLoss	32	0,001	seq	1,0	415
5	TaylorNitschkeNet	kaiming	RMSprop	ReduceLROnPlateau	CrossEntropyLoss	16	0,001	seq	1,0	400
6	TaylorNitschkeNet	kaiming	RMSprop	MultiStepLR	NLLLoss	32	0,001	seq	1,0	396
7	TaylorNitschkeNet	kaiming	Adam	ReduceLROnPlateau	NLLLoss	16	0,001	dA	1,0	358
8	TaylorNitschkeNet	kaiming	Adam	ReduceLROnPlateau	KLDivLoss	16	0,001	dA	1,0	352
9	TaylorNitschkeNet	kaiming	Adam	StepLR	NLLLoss	16	0,001	dA	1,0	376
10	AlexNet	xavier	RMSprop	StepLR	KLDivLoss	32	0,001	seq	0,99	264

Tabelle 7.2: Darstellung der Ergebnisse des *Rankings* mit den zehn besten Kombinationen an Hyperparametern nach dem Training

zugewiesen wurde und von nun an in den folgenden Tabellen und Abbildungen zur Identifikation des Netzes dient. Die Tabelle zeigt, welcher Datensatz in der Kombination den besten Wert erzielt hat. Es stellt sich heraus, dass der Rohdatensatz *raw* nicht unter den besten Ergebnissen vertreten ist, was darauf hindeutet, dass die Data Augmentation mittels Sequencer der Verwendung von Rohdaten überlegen ist.

Die Abbildung 7.2 zeigt die *Ranking*-Werte im Vergleich zwischen den Datensätzen. Es ist zu erkennen, dass der Sequencerdatensatz in vielen Fällen minimal schlechter abschneidet. Es ist jedoch zu beachten, dass hier nur die Trainingsergebnisse dargestellt sind. In der folgenden Evaluierung werden diese Netzkonfigurationen unter schwierigen Bedingungen getestet, um zu untersuchen, ob der Sequencer eine effektivere Data Augmentation erzeugen kann.

Daher kann in dieser Phase noch keine endgültige Aussage über die Überlegenheit des Sequencers getroffen werden.

Abbildung 7.3 zeigt die Trainingsverläufe des Modells 227 für die drei Datensätze. Dieses Modell erzielte in Tabelle 7.2 die besten Ergebnisse und dient daher als Beispiel.

Links in der Abbildung ist der Verlauf für den *raw*-Datensatz dargestellt. Hier benötigt das Modell bis zur 17. Epoche, um keine Verschlechterung mehr zu zeigen. Das Training wird anschließend abgebrochen, da sich der Validierungsverlust über das festgelegte Delta hinaus verschlechtert hat.

In der Mitte ist der Verlauf für den *dA*-Datensatz zu sehen. Das Modell erreicht bereits nach der zweiten Epoche eine Genauigkeit von 1,0, woraufhin das Training vorzeitig beendet wird.

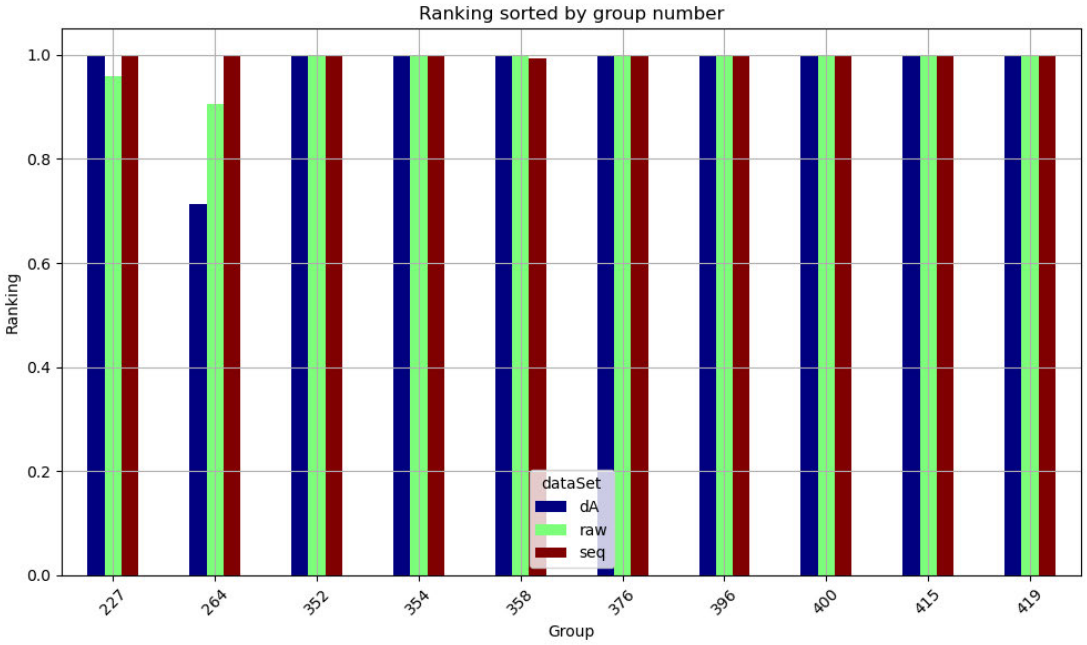


Abbildung 7.2: Darstellung der *Ranking*-Werte der Tabelle 7.2

Rechts zeigt der Verlauf für den *seq*-Datensatz, dass das Modell über alle 20 Epochen trainiert wurde. Es erreicht jedoch schon früh, ab der vierten Epoche, eine hohe Genauigkeit und konvergiert anschließend langsam zu 1,0.

Die Ergebnisse verdeutlichen, dass beide Ansätze der Data Augmentation (*dA* und *seq*) eine hohe Genauigkeit auf den Trainingsdaten erzielen. Während das *dA*-Modell bereits nach der zweiten Epoche eine Genauigkeit von 1,0 erreicht, benötigt das *seq*-Modell dafür bis zur neunten Epoche.

Damit konnte gezeigt werden, dass die Data Augmentation mit dem Sequencer eine ähnliche Leistung wie die Software-gestützte Data Augmentation erzielt.

Ein weiterer wichtiger Aspekt ist die zeitliche Betrachtung der Trainingszeiten. Abbildung 7.4 zeigt die Trainingszeiten der Modelle als Balkendiagramm. Es fällt auf, dass die Modelle, die mit dem Rohdatensatz trainiert wurden, erwartungsgemäß die kürzesten Trainingszeiten aufweisen, da pro Epoche weniger Daten geladen und verarbeitet werden müssen.

Ein Vergleich der Modelle, die mit den Datensätzen *dA* und *seq* trainiert wurden, zeigt eine klare Tendenz: Bei den zehn besten Modellen ist das Training mit Sequencer-

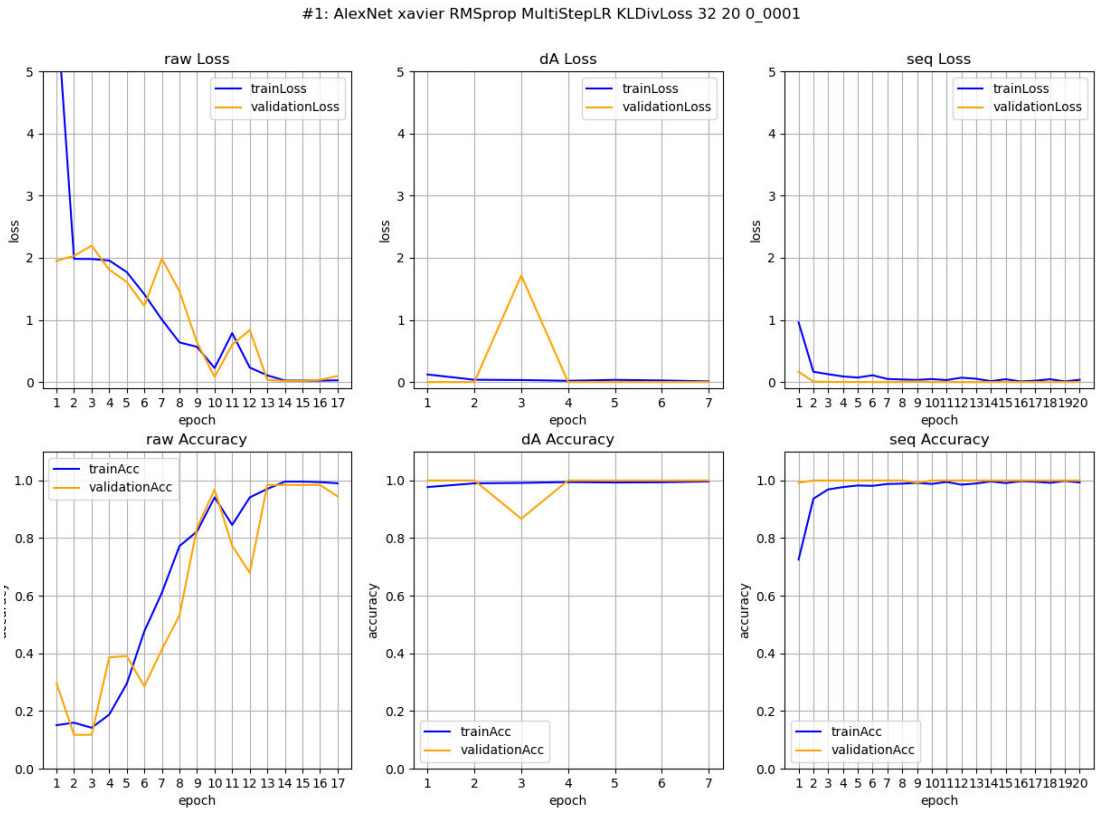


Abbildung 7.3: Trainingsverlauf des AlexNet nach dem Training

Daten stets schneller als das Training mit der PyTorch-Data-Augmentation-Methode. Ein Modell, das mit dem *dA*-Datensatz trainiert wurde, benötigte im Durchschnitt 60,07 Minuten, während das Modell mit dem *seq*-Datensatz nur 55,95 Minuten benötigte. Dadurch konnte im Durchschnitt eine Zeitersparnis von 4,12 Minuten pro Modell erzielt werden.

Während dieser Unterschied bei einem vergleichsweise kleinen Trainingsdatensatz von 7350 Aufnahmen noch moderat erscheint, wird der Zeitvorteil bei größeren Datensätzen wie ImageNet, das über 14,1 Millionen Aufnahmen umfasst, erheblich[1]. In solchen Fällen kann die Verwendung des *seq*-Datensatzes zu einer signifikanten Reduktion der Trainingszeit führen, was insbesondere bei ressourcenintensiven Projekten von großer Bedeutung ist.

[1] siehe https://paperswithcode.com/dataset/imagenet

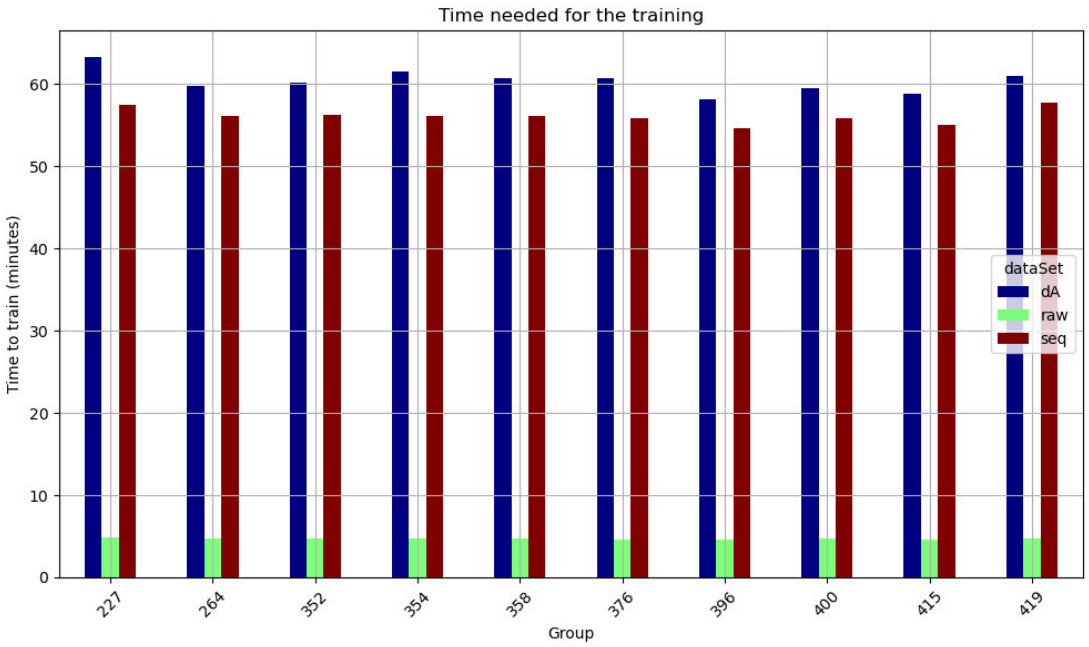


Abbildung 7.4: Vergleich der Trainingszeiten der zehn besten Modelle

7.1.2 Bewertung nach der Evaluierung

Für die Evaluierung wurden die Hyperparameter-Kombinationen aus Tabelle 7.2 erneut trainiert und mit den Evaluationsdaten bewertet. Die Ergebnisse sind in Tabelle 7.3 zusammengefasst, eine detailliertere Übersicht findet sich in Tabelle A.1.

Das beste *Ranking* nach der Evaluierung wurde von Modell 400 mit einem Wert von 0,6311 erzielt, welches im Training noch ein perfektes *Ranking* von 1,0 erreicht hatte. Der Vergleich der Tabellen 7.2 und 7.3 zeigt, dass die Modelle nach der Evaluierung durchweg schlechtere Ergebnisse erzielen, wobei die Rangfolge der Modelle weitgehend stabil bleibt.

Auffällig ist, dass der Sequencer-Datensatz (*seq*) nach der Evaluierung häufiger vertreten ist als der *dA*-Datensatz. Dies deutet darauf hin, dass Modelle, die mit dem *seq*-Datensatz trainiert wurden, tendenziell konsistentere und robustere Ergebnisse liefern.

Die Abbildung 7.5 stellt das *Ranking* aller Datensätze der Evaluierung visuell dar. Diese Darstellung entspricht dem Mittelwert von jeweils drei Versuchsdurchführungen, da sich die Ergebnisse der einzelnen Versuche leicht unterscheiden.

#	Model	Init	Optimizer	Scheduler	Loss	BatchSize	LearningRate	DataSet	Ranking	Index
1	TaylorNitschkeNet	kaiming	RMSprop	ReduceLROnPlateau	CrossEntropyLoss	16	0,001	seq	0,6311	400
2	TaylorNitschkeNet	kaiming	RMSprop	StepLR	CrossEntropyLoss	16	0,001	seq	0,6306	419
3	AlexNet	xavier	RMSprop	MultiStepLR	KLDivLoss	32	0,0001	dA	0,6271	227
4	TaylorNitschkeNet	kaiming	Adam	ReduceLROnPlateau	KLDivLoss	32	0,001	seq	0,6261	354
5	TaylorNitschkeNet	kaiming	RMSprop	ReduceLROnPlateau	NLLLoss	32	0,001	seq	0,6259	415
6	TaylorNitschkeNet	kaiming	RMSprop	MultiStepLR	NLLLoss	32	0,001	seq	0,6240	396
7	TaylorNitschkeNet	kaiming	Adam	ReduceLROnPlateau	NLLLoss	16	0,001	seq	0,6208	358
8	TaylorNitschkeNet	kaiming	Adam	StepLR	NLLLoss	16	0,001	dA	0,6189	376
9	TaylorNitschkeNet	kaiming	Adam	ReduceLROnPlateau	KLDivLoss	16	0,001	seq	0,6156	352
10	AlexNet	xavier	RMSprop	StepLR	KLDivLoss	32	0,001	raw	0,3702	264

Tabelle 7.3: Darstellung der Ergebnisse des *Rankings* mit den zehn besten Kombinationen an Hyperparametern nach der Evaluierung

Hier werden die Unterschiede zwischen den Datensätzen deutlicher. Es fällt auf, dass der Rohdatensatz im Vergleich zu den Datensätzen mit Data Augmentation deutlich schlechtere Ergebnisse liefert.

Insbesondere beim Modell 264 zeigt sich, dass die Evaluierung nicht erfolgreich bewältigt wurde. Während der Rohdatensatz noch einen *Ranking*-Wert von etwa 0,38 erreichte, erzielten die beiden anderen Datensätze mit Data Augmentation sogar negative *Ranking*-Werte von bis zu -0,1. Dies deutet darauf hin, dass Modell 264 Schwierigkeiten hatte, die Evaluierungsdaten korrekt zu klassifizieren, was auf eine unzureichende Anpassungsfähigkeit des Modells hindeutet. Mögliche Ursachen könnten ein *Overfitting* an die Trainingsdaten oder eine unzureichende Diversität der Trainingsdaten sein, wodurch das Modell nicht in der Lage war, die Evaluierungsdaten korrekt zu verarbeiten.

Die Ergebnisse zeigen, dass die Unterschiede zwischen den Datensätzen *seq* und *dA* minimal sind, jedoch der *seq*-Datensatz in den meisten Fällen leicht bessere Ergebnisse erzielt. Wie in Abbildung 7.5 zu sehen ist, übertrifft der *seq*-Datensatz den *dA*-Datensatz in sieben von zehn Fällen, während der *dA*-Datensatz nur in einem Fall ein besseres *Ranking* erreicht. Dies deutet darauf hin, dass der *seq*-Datensatz insgesamt konsistenter und effektiver ist, auch wenn die Unterschiede in den Zahlen nur geringfügig ausfallen.

Die folgenden Abbildungen zeigen die *Confusions* Matrizen des Modells 354 im Vergleich. In den Spalten sind die Vorhersagen des Modells dargestellt, während die Zeilen die tatsächlichen Labels repräsentieren. Eine perfekte Klassifikation würde sich durch eine dunkelblaue Diagonale mit Werten von 1,0 auszeichnen, was bedeutet, dass jede Klasse korrekt vorhergesagt wurde. Abweichungen von der Diagonale zeigen Fehlklassifikationen an.

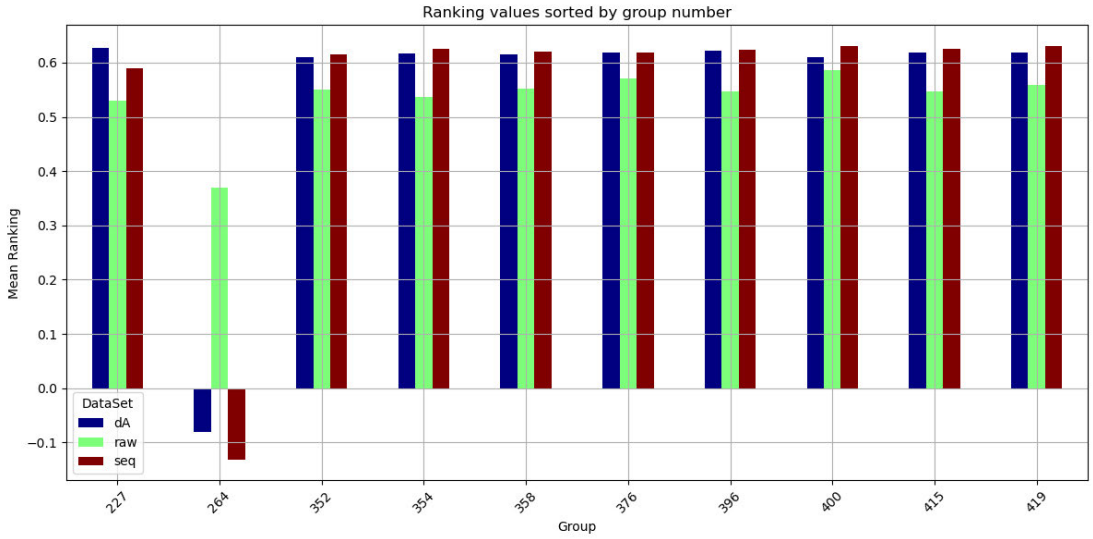


Abbildung 7.5: Darstellung der *Ranking*-Werte der Tabelle 7.3 als Mittelwert aus drei Berechnungen

Abbildung 7.6 zeigt die *Confusion* Matrix für den *raw*-Datensatz. Hier wird deutlich, dass das Modell lediglich zwischen den Klassen Diode und Keramikkondensator unterscheidet, während die übrigen Klassen nahezu vollständig ignoriert werden.

Die *Confusion* Matrix des *dA*-Datensatzes ist in Abbildung 7.7 dargestellt. Im Vergleich zum *raw*-Datensatz zeigt das Modell hier eine bessere Fähigkeit, zwischen den Klassen zu unterscheiden. Dennoch bleibt die Zuordnung der Klassen fehlerhaft, was auf eine begrenzte Genauigkeit hinweist.

Abbildung 7.8 zeigt die *Confusion* Matrix für den *seq*-Datensatz. Hier wird ersichtlich, dass das Modell in der Lage ist, die Klassen besser zu unterscheiden. Allerdings konzentriert es sich hauptsächlich auf die Klassen Photowiderstand und Widerstand, während andere Klassen weniger berücksichtigt werden.

Ein Vergleich der Ergebnisse zeigt, dass das Modell 354 mit dem *raw*-Datensatz die schlechteste Leistung erbringt, da es nur zwei Klassen unterscheidet und die übrigen ignoriert. Der *dA*-Datensatz verbessert die Klassifikationsfähigkeit durch Data Augmentation, erreicht jedoch nur eine begrenzte Genauigkeit. Der *seq*-Datensatz zeigt ebenfalls Fortschritte durch die Sequencer-basierte Data Augmentation, jedoch fokussiert sich das Modell stark auf zwei Klassen, was die Gesamtleistung einschränkt.

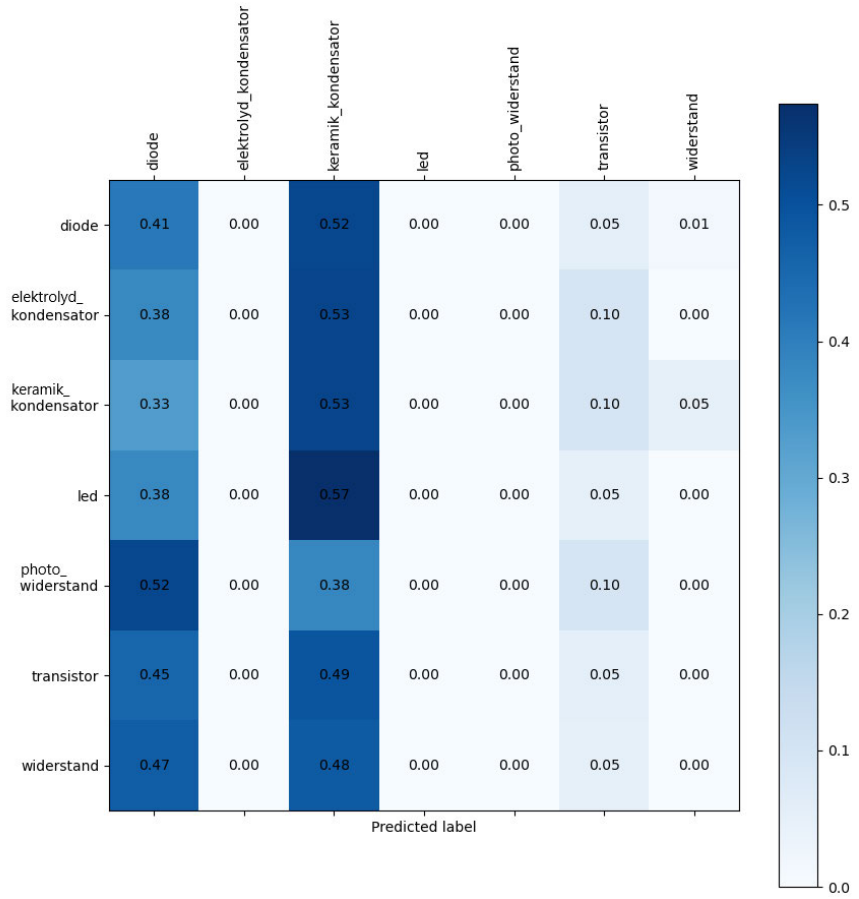


Abbildung 7.6: *Confusion* Matrix des *raw*-Datensatzes nach der Evaluierung des Modells 354 (siehe Tabelle 7.3)

7.1.3 Beurteilung der wichtigen Kameraparameter

In diesem Abschnitt werden die besten neuronalen Netze aus den vorherigen Experimenten erneut trainiert, wobei ausschließlich Sequencer-Daten als Trainingsdatensatz verwendet werden. Jeder Sequenzerdatensatz repräsentiert dabei eine bestimmte Einstellung eines Kameraparameters (siehe Kapitel 5.1.2), so dass insgesamt 14 verschiedene Datensätze entstehen.

Nach dem Training erfolgt eine Evaluierung der Modelle, um ihre Leistungsfähigkeit zu bewerten. Die Ergebnisse werden mit Hilfe des *Rankings* analysiert, wobei sowohl die Trainings- als auch die Evaluierungsergebnisse berücksichtigt werden. Ziel ist es, die

Abbildung 7.7: *Confusion* Matrix des *dA*-Datensatzes nach der Evaluierung des Modells 354 (siehe Tabelle 7.3)

Bedeutung der einzelnen Kameraparameter für die Modellleistung zu bewerten und die effektivsten Einstellungen zu identifizieren.

Abbildung 7.9 zeigt die *Ranking*-Werte der verschiedenen Sequencer-Sets, die in den vorherigen Experimenten verwendet wurden. Die Werte sind als Boxplot dargestellt, wobei auf der x-Achse die einzelnen Sequencer-Sets abgebildet sind. Um die Ergebnisse zu stabilisieren, wurde der Median der *Ranking*-Werte aus zwei Experimenten berechnet, da die Experimente nicht vollständig reproduzierbar sind.

Die Tabelle 7.4 fasst die Median-*Ranking*-Werte der Sequencer-Sets zusammen. Diese Werte basieren auf den in Abbildung 7.9 dargestellten Ergebnissen. Ausreißer wurden aus Gründen der Übersichtlichkeit nicht berücksichtigt.

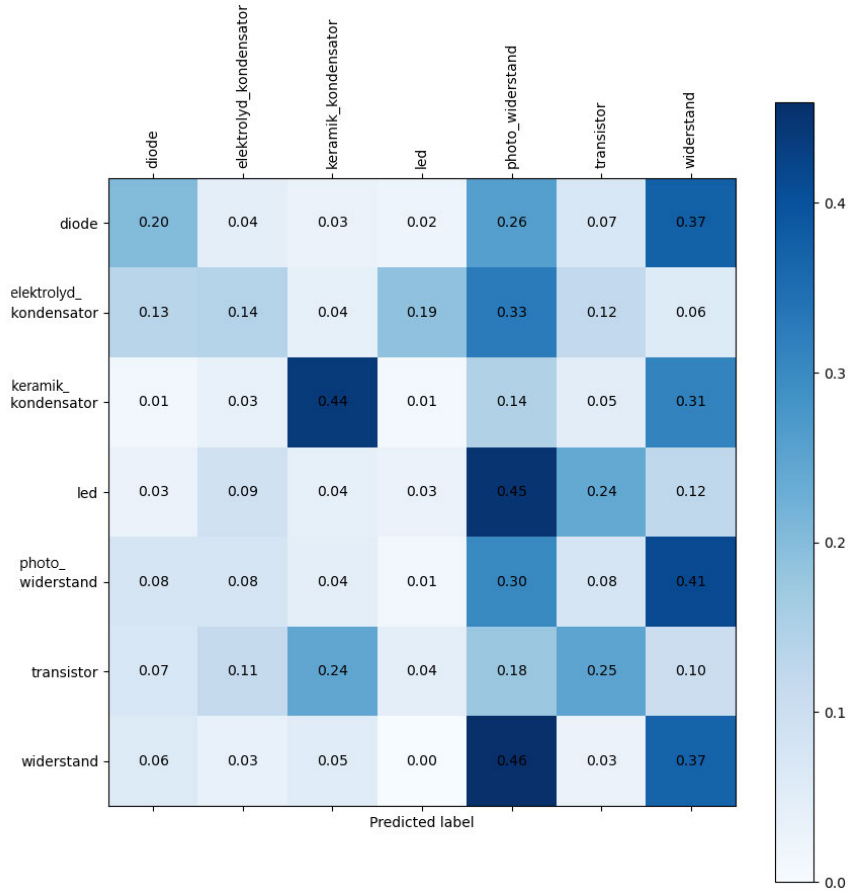


Abbildung 7.8: *Confusion* Matrix des *seq*-Datensatzes nach der Evaluierung des Modells 354 (siehe Tabelle 7.3)

Die Ergebnisse zeigen, dass Sequencer-Set 4 und Sequencer-Set 9 die besten Median-*Ranking*-Werte von 0,603 bzw. 0,606 erzielen. Besonders hervorzuheben ist, dass Sequencer-Set 4 über alle Modelle hinweg konstant gute Ergebnisse liefert, was auf die Bedeutung des Gamma-Werts hinweist. Sequencer-Set 9 zeigt ebenfalls hohe Leistungen, jedoch mit einer größeren Spannweite der *Ranking*-Werte, was darauf hindeutet, dass es nicht für alle Modelle gleichermaßen geeignet ist.

Im Gegensatz dazu weist Sequencer-Set 12 mit einem Minimum von 0,5 und einem Maximum von 0,6 die größte Spannweite auf, was darauf hindeutet, dass eine türkise Aufnahme für diesen Anwendungsfall weniger geeignet ist. Die anderen Sets weisen moderate Spannweiten auf und können als Ergänzung zu den besten Sets verwendet werden.

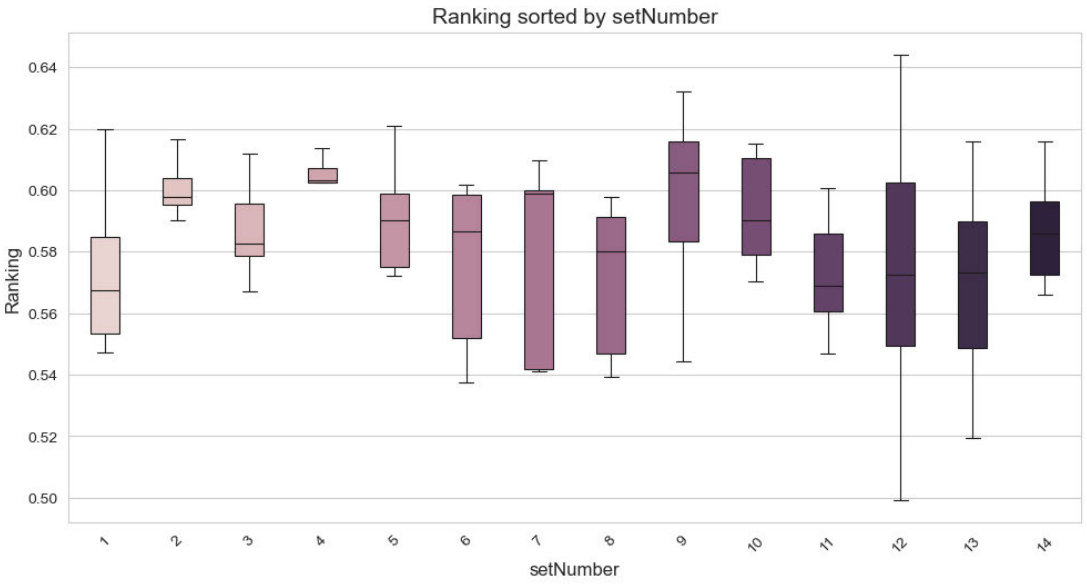


Abbildung 7.9: *Ranking*-Werte der Sequencer-Sets als Boxplot

Set	Sequencer-Set	Ranking (Median)
1	Minimale Belichtungszeit und maximales Gain	0,567
2	Erhöhtes Gain	0,598
3	Niedrige Belichtungszeit	0,583
4	Minimales Gamma	0,603
5	Maximales Gamma	0,590
6	Minimale Sättigung	0,587
7	Maximale Sättigung mit maximalem Farbton	0,599
8	Minimale Sättigung mit minimalem Farbton	0,580
9	Rote Aufnahme	0,606
10	Gelbe Aufnahme	0,590
11	Grüne Aufnahme	0,569
12	Türkise Aufnahme	0,573
13	Blaue Aufnahme	0,573
14	Rosa Aufnahme	0,586

Tabelle 7.4: Median-*Ranking*-Werte der Sequencer-Sets

Zusammenfassend zeigt dieser Versuch, dass Sequencer-Set 4 mit einem minimalen Gamma-Wert die robustesten Ergebnisse liefert. Sequencer-Set 9 erzielt ebenfalls hohe Leistungen, jedoch mit einer größeren Variabilität. Die anderen Sets können als Ergänzung verwendet werden, um die Leistung des Modells weiter zu verbessern. Es wird

Methode	Beschreibung
RandomHorizontalFlip	Zufälliges horizontales Spiegeln des Bildes.
RandomVerticalFlip	Zufälliges vertikales Spiegeln des Bildes.
RandomRotation	Zufällige Rotation des Bildes um bis zu 10 Grad.
RandomCrop	Zufälliges Zuschneiden des Bildes mit optionalem Padding.
ColorJitter	Anpassung von Helligkeit, Kontrast, Sättigung und Farbton.
RandomPerspective	Perspektivische Verzerrung des Bildes.
RandomAffine	Kombination von Translation, Skalierung und Scherung.
ElasticTransform	Elastische Transformation des Bildes.
GaussianBlur	Anwendung eines Gaußschen Weichzeichners.
RandomInvert	Invertierung der Pixelwerte des Bildes.
RandomPosterize	Reduktion der Farbtiefe des Bildes.
RandomSolarize	Invertierung der Pixelwerte oberhalb eines bestimmten Schwellenwerts.
RandomAdjustSharpness	Anpassung der Bildschärfe.
RandomAutocontrast	Automatische Kontrasteinstellung.
RandomEqualize	Histogramm-Ausgleich des Bildes.
RandomErasing	Zufälliges Löschen eines rechteckigen Bereichs im Bild.

Tabelle 7.5: Zusätzliche Standard-Data-Augmentation-Methoden

somit empfohlen, den Gamma-Wert als zentralen Parameter zu priorisieren und die anderen Sequencer-Sets unterstützend einzusetzen.

7.1.4 Vergleich mit zusätzlicher Data Augmentation

In diesem Versuch wird untersucht, wie sich eine erweiterte Data Augmentation auf die Robustheit der Modelle auswirkt. Hierzu wurde der bestehende *dA*-Datensatz um zusätzliche Standard-Data-Augmentation-Methoden erweitert, die in Tabelle 7.5 aufgeführt sind. Die Anzahl der angewendeten Transformationen pro Aufnahme erhöhte sich dadurch von 14 auf 30, was eine größere Vielfalt an Bildvariationen ermöglicht.

Die Ergebnisse der erweiterten Data Augmentation sind in Abbildung 7.10 dargestellt. Der neue Datensatz, als *dA2* markiert, zeigt in fast allen Fällen höhere oder vergleichbare *Ranking*-Werte im Vergleich zu den bisherigen Datensätzen. Dies deutet darauf hin, dass die zusätzlichen Methoden entweder durch die größere Vielfalt an Transformationen oder durch die erhöhte Anzahl an Trainingsbeispielen zu einer verbesserten Modellleistung beitragen.

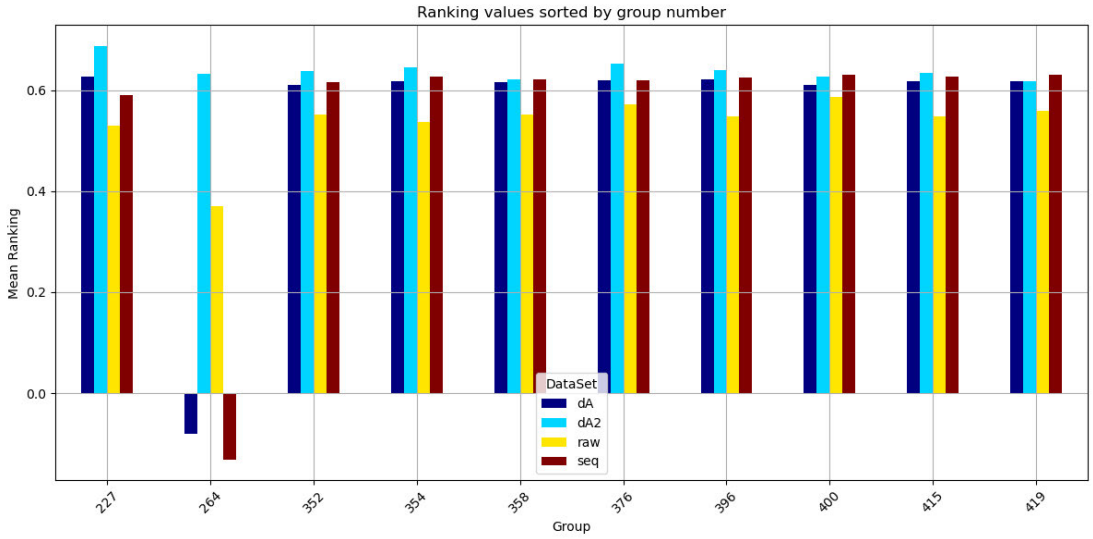


Abbildung 7.10: Darstellung der *Ranking*-Werte der Sequencer-Sets und zusätzlich von der neuen Data Augmentation als Balkendiagramm

Das Modell 227 erzielte mit der erweiterten Data Augmentation den höchsten *Ranking*-Wert in der Evaluierung. Auch das Modell 264, das mit den bisherigen Datensätzen keine zufriedenstellenden Ergebnisse lieferte, profitierte deutlich von der neuen Methode und erreichte *Ranking*-Werte auf einem vergleichbaren Niveau wie die anderen Modelle. Dies zeigt, dass die erweiterten Methoden einen positiven Einfluss auf die Robustheit und die allgemeine Leistungsfähigkeit der Modelle haben.

Zusammenfassend lässt sich feststellen, dass die erweiterte Data Augmentation die Ergebnisse der meisten Modelle verbessert und somit eine vielversprechende Methode zur Steigerung der Modellrobustheit darstellt.

7.2 Bewertung der Anforderungen

Dieses Kapitel soll einen Überblick über die gestellten Anforderungen geben und diese auf Vollständigkeit prüfen. Dazu werden die Anforderungen aus dem Kapitel 4.3 betrachtet und überprüft, ob diese eingehalten und umgesetzt wurden.

Anforderung	Erfüllung
K-1	**Erfüllt** durch den Einsatz der Kamera Alvium 1800 U-052c, die eine Sequencer-Funktion unterstützt.
K-2	**Erfüllt** durch die Anwendung der Sequencer-Sets mit einer Auswahl an ausschlaggebenden Parametern für eine Aufnahme. Die dadurch entstandenen Aufnahmen eignen sich für die Data Augmentation und konnten dort zum Training genutzt werden. Durch die verfügbaren Farbkanäle in Kombination mit den anderen verfügbaren Sequencer-Parametern konnten 15 verschiedene Sets erstellt werden.
K-3	**Erfüllt** durch die verfügbare USB-Schnittstelle der Alvium 1800 U-052c. Durch die Vimba X API war es einfach, die Kamera über Python anzusteuern und somit den Sequencer zu konfigurieren und die Aufnahmen auf den Computer zu kopieren.
K-4	**Erfüllt**, da die verwendete Kamera über ein Stativgewinde verfügt und somit fest im Versuchsaufbau installiert werden konnte.

Tabelle 7.6: Bewertung der Anforderungen an die Kamera

Anforderung	Erfüllung
O-1	**Erfüllt** mit einer Auswahl von 7 verschiedenen Klassen: Widerstand, LED, Elektrolytkondensator, Keramikkondensator, Photoresistor, Diode und Transistor.
O-2	**Erfüllt** durch die Verwendung von elektrischen Bauelementen, die in Größe und Form ähnlich sind. Beispielsweise haben Dioden die gleiche Form wie Widerstände, und auch die Farbe wurde berücksichtigt, indem blaue Dioden und blaue Widerstände verwendet wurden. Diese Vorkehrungen ermöglichen es dem neuronalen Netz, robuster zu sein, da die Klassifizierung nicht nur auf Form oder Farbe basiert.
O-3	**Erfüllt** mit den elektrischen Bauteilen, die eine Abmessung von $7\,\text{cm} \times 9\,\text{cm} \times 1\,\text{cm}$ einhalten.
O-4	**Erfüllt** mit den verschiedenen elektrischen Bauteilen, die sich optisch unterscheiden, reflektierende Oberflächen haben und sich farblich unterscheiden.

Tabelle 7.7: Bewertung der Anforderungen an die Objektauswahl

Anforderung	Erfüllung
A-1	**Erfüllt** durch den beschriebenen Aufbau einer Konstruktion, die die Kamera in einer Position hält. Dadurch wurde es möglich, dass die Aufnahmen auch an mehreren Tagen reproduzierbar aufgenommen werden konnten.
A-2	**Erfüllt** mit dem Drehteller, der es ermöglicht, die Objekte in 360° aufzunehmen.
A-3	**Erfüllt** durch das gelbe und blaue Stück Folie, welche für die Aufnahmen während der Evaluierung vor der Schreibtischlampe gehalten worden sind. Durch den Aufbau in einem dunklen Raum können so die Lichtverhältnisse direkt bestimmt werden.
A-4	**Erfüllt** mit einem weißen und einem schwarzen Blatt Papier, die auf den roten Drehteller gelegt werden können. So erhält man drei verschiedene Versionen für den Hintergrund.
A-5	**Erfüllt** durch einen Aufbau, der aus einer Kamera, einer Konstruktion für die Kamera, einer Schreibtischlampe und einem Drehteller besteht. Dieser Aufbau ist leicht zu transportieren, so dass das Experiment in einem anderen Raum durchgeführt werden kann.

Tabelle 7.8: Bewertung der Anforderungen an den Versuchsaufbau

Anforderung	Erfüllung
N-1	**Erfüllt** durch die erfüllte Aufgabe des Netzes, die eine Klassifikation ist.
N-2	**Erfüllt** für die im Training verwendeten Netze. Dort ist eine Genauigkeit von 95% erreicht worden und die Netze sind für die Evaluierung genutzt worden.
N-3	**Nicht erfüllt**, da das Training nicht vollständig reproduzierbar ist. Trotz der Maßnahmen, Zufallswerte für verschiedene Teile des Programms zu definieren, konnten leichte Abweichungen zwischen den Durchläufen nicht vollständig eliminiert werden. Dies liegt an der stochastischen Natur des Trainingsprozesses, einschließlich zufälliger Initialisierungen und der Zusammenstellung von *Batches*, die zu Variationen im Endergebnis führen können.
N-4	**Erfüllt** durch die Aufnahmen, welche unter extremen Bedingungen aufgenommen wurden. Diese stellen seltene Situationen dar, die in der Praxis auftreten können (z. B. sehr geringe Beleuchtung und Überbeleuchtung). Dies wurde getan, um zu sehen, ob die Netzwerke in diesen Situationen einen Vorteil aus dem Datensatz ziehen können.

Tabelle 7.9: Bewertung der Anforderungen an das neuronale Netz

Vergleichskriterium	Erfüllung
V-1	**Erfüllt**, wobei alle diese Metriken für jedes Netzwerk berechnet wurden, aber der F1-Wert für die Berechnung der Rangliste verworfen wurde. Dieser Wert ist schon in dem folgenden *Ranking*-Wert inhalten, da dieser auch aus den Metriken *Recall* und *Precision* besteht.
V-2	**Erfüllt**, da alle Netze, die durch Training und Evaluierung erzeugt wurden, nachträglich anhand der aus den Leistungsmetriken resultierenden *Ranking*-Werte verglichen wurden.
V-3	**Erfüllt** Die Evaluierung hat gezeigt, dass die Data Augmentation mit dem Sequencer keine signifikante Verbesserung der Ergebnisse bewirkt hat. Die *Confusions* Matrizen verdeutlichen, dass das Modell mit dem Sequencerdatensatz zwar konsistentere Vorhersagen liefert, diese jedoch häufig falsch sind. Im Gegensatz dazu zeigt der Datensatz mit der PyTorch-Data-Augmentation-Methode, dass die Modelle tendenziell gleichmäßig auf alle Klassen raten. Dennoch ist hervorzuheben, dass das Training mit den Sequencer-Daten effizienter und schneller durchgeführt werden konnte.
V-4	**Erfüllt** mit dem Überprüfen der besten zehn Netzen mit jeweils nur einem Sequencer-Set. Dabei hat sich herausgestellt, dass je nach Kombination an Hyperparametern das eine oder andere Sequencer-Set die besten Leistungen erbringt.

Tabelle 7.10: Bewertung der Vergleichskriterien

8 Fazit und Ausblick

In diesem Kapitel werden die zentralen Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf mögliche zukünftige Entwicklungen und Forschungsansätze gegeben.

8.1 Fazit

Diese Arbeit zeigt, dass der Einsatz einer Kamera mit Sequencer für die Data Augmentation eine ebenso effektive Methode darstellt wie herkömmliche softwarebasierte Ansätze. Bei der Evaluierung mit einem Datensatz, der auch Extremfälle abdeckt, lieferten beide Verfahren signifikant bessere Ergebnisse als der Rohdatensatz. Die Ergebnisse variierten jedoch je nach Kombination der Hyperparameter, so dass keine Methode durchgängig überlegen war. Der Sequencer erwies sich in allen Tests als zeitlich effizienter.

Ein limitierender Faktor war die eingeschränkte Reproduzierbarkeit der Experimente, die zu leichten Abweichungen zwischen den Durchgängen führte. Diese Abweichungen hatten jedoch keinen wesentlichen Einfluss auf die Aussagekraft der Ergebnisse. Die Experimente lieferten zudem wertvolle Erkenntnisse über den Einfluss verschiedener Kameraparameter auf die Data Augmentation. Trotz der begrenzten Anzahl an Experimenten deutet vieles darauf hin, dass eine Kombination aller verfügbaren Kameraparameter eine effektive Data Augmentation ermöglicht.

Eine weitere Einschränkung stellte die begrenzte Rechenleistung dar. Die Suche nach geeigneten Hyperparametern dauerte mit der vorhandenen Hardware mehrere Wochen bis Monate. Für zukünftige Projekte bieten Cloud-Dienste wie Google Colab oder AWS eine sinnvolle Alternative. Langfristig könnte die Anschaffung moderner Hardware, wie leistungsfähiger Grafikprozessoren, eine nachhaltigere Lösung darstellen.

Auch die Arbeitsweise kann optimiert werden. Eine gröbere *Grid-Search*, wie sie in der Industrie üblich ist, hätte die Suche nach den besten Hyperparametern beschleunigen

können. Eine effizientere Datenvorverarbeitung, z. B. durch einmalige Generierung von Validierungs-, Test- und Auswertungsdaten, könnte redundante Daten vermeiden.

Positiv hervorzuheben ist die gründliche Recherche, die sowohl durch Fachliteratur [10] als auch durch Online-Ressourcen wie *Learnpytorch*[1] unterstützt wurde. Diese Quellen lieferten theoretische Grundlagen und praktische Ansätze zur Softwareentwicklung. Besonders hilfreich war die Website[2], die eine anschauliche Darstellung von CNNs bietet. Die erfolgreiche Einarbeitung in die API der Kamera von Allied Vision, unterstützt durch die Zusammenarbeit mit Kollegen, trug ebenfalls wesentlich zum Erfolg der Arbeit bei.

Die Arbeit mit dem Sequencer und die Entwicklung der zugehörigen Software verliefen besonders erfolgreich. Die Software ermöglicht eine vollständige Konfiguration aller Sequencer-Sets sowie die automatische Speicherung der Aufnahmen. Trotz zeitaufwändiger Tests aller Hyperparameter-Kombinationen und wiederholter Hardwareprobleme wurden alle gesteckten Ziele erreicht. In einigen Fällen erzeugte der Sequencer-Datensatz eine höhere Varianz in den Trainingsdaten, was die Performance verbesserte. Im Vergleich zum Rohdatensatz und zum softwarebasierten Data Augmentation-Datensatz zeigte der Sequencer sowohl einen leichten Performance-Vorteil als auch eine deutliche Zeitersparnis.

Zusammenfassend lässt sich sagen, dass der Sequencer eine vielversprechende Methode zur Data Augmentation darstellt. Dieser Ansatz erzielte vergleichbare Ergebnisse wie die softwaregestützte Data Augmentation und erwies sich als robust und zeitsparend. Die Kombination aller verfügbaren Kameraparameter sowie die Erweiterung um zusätzliche Augmentationsmethoden könnten die Ergebnisse weiter verbessern. Die Arbeit bietet eine solide Grundlage für zukünftige Entwicklungen im Bereich der Data Augmentation und des maschinellen Lernens.

8.2 Ausblick

Die Ergebnisse dieser Arbeit eröffnen vielfältige Perspektiven für zukünftige Projekte. Im Anhang wird eine Vorgehensweise zur Konfiguration des Sequencers beschrieben, die als Grundlage für weitere Arbeiten dienen kann. Im Bereich der Data Augmentation

[1] https://www.learnpytorch.io
[2] https://poloclub.github.io/cnn-explainer/

könnte untersucht werden, ob zufällig gewählte Kombinationen von Kameraparametern zu besseren Ergebnissen führen. Eine solche zufällige Kombination könnte die Varianz der Trainingsdaten erhöhen und die Robustheit des neuronalen Netzes weiter verbessern. Zudem sollten die genannten weiteren Methoden der Data Augmentation genauer untersucht werden. Eine getrennte Betrachtung der Methoden könnte deren Effektivität besser darstellen und als Grundlage für weitere Experimente dienen.

Darüber hinaus bietet diese Arbeit einen Einstiegspunkt für die Entwicklung eigener neuronaler Netze mit geeigneter Data Augmentation. Der Vergleich zwischen Data Augmentation und Rohdatensätzen verdeutlicht die Vorteile der Augmentation und liefert wertvolle Erkenntnisse für zukünftige Anwendungen.

Da die Ergebnisse der Sequencer-basierten und der Software-basierten Data Augmentation ähnlich sind, könnte geprüft werden, ob zusätzliche Augmentationsmethoden direkt in die Kamera integriert werden sollten. Dies könnte insbesondere bei größeren Datensätzen zu einer erheblichen Zeitersparnis führen, da die Augmentierung bereits während der Aufnahme erfolgt und nachträgliche Verarbeitungsschritte reduziert werden.

Beispielsweise könnten Methoden wie das Spiegeln oder zufällige Rotieren von Aufnahmen direkt in die Kamera eingebunden werden. Diese Erweiterungen könnten untersuchen, wie ein neuronales Netz mit solchen integrierten Methoden im Vergleich zu einer PyTorch-basierten Data Augmentation abschneidet. Ein weiterer Ansatz wäre, die Sequencer Data Augmentation gezielt zur Unterstützung von rechenintensiven PyTorch-Methoden einzusetzen, um die Gesamtverarbeitungszeit zu reduzieren.

Zukünftige Experimente könnten sich darauf konzentrieren, die Effizienz und Effektivität dieser integrierten Ansätze zu evaluieren. Es wäre interessant zu analysieren, ob die Kombination von Sequencer- und softwarebasierten Augmentierungsmethoden einen positiven Effekt sowohl auf die Trainingszeit als auch auf die Modellleistung haben kann. Die Integration zusätzlicher Augmentierungsmethoden in die Kamera erfordert jedoch Hardwareanpassungen und Entwicklungsaufwand. Eine genaue Analyse der technischen Machbarkeit und der potentiellen Kosten wäre daher ein wichtiger Bestandteil zukünftiger Arbeiten.

Literaturverzeichnis

[1] Fully connected layers in convolutional neural networks. https://evbn.org/fully-connected-layers-in-convolutional-neural-networks-1677988375/. Zugriff am 30. Januar 2025.

[2] Fotolithographie: Led-gelblichtlösung für reinräume und halbleiterfertigung, 2025. Abgerufen am: 10.01.2025.

[3] Basler AG. Color adjustment, 2025. Zugriff am: 01.04.2025.

[4] L Alzubaidi, J Zhang, AJ Humaidi, A Al-Dujaili, Y Duan, O Al-Shamma, J Santamaría, MA Fadhel, M Al-Amidie, and L Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):53, 2021. Epub 2021 Mar 31.

[5] J Arun Pandian, G Geetharamani, and B Annette. Data augmentation on plant leaf disease image dataset using image manipulation and deep learning techniques. In *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, pages 199–204, 2019.

[6] Daniel Bourke. Zero to mastery learn pytorch for deep learning - custom datasets, 2024. Zugriff am: 03.01.2025.

[7] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels, 2012.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database, 2009.

[9] Carl Doersch. Tutorial on variational autoencoders, 2021.

[10] M. Elgendy. *Deep Learning for Vision Systems*. Manning, 2020.

[11] Omar Elharrouss, Younes Akbari, Noor Almadeed, and Somaya Al-Maadeed. Backbones-review: Feature extractor networks for deep learning and deep reinforcement learning approaches in computer vision. *Computer Science Review*, 53:100645, August 2024.

[12] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge, 2010.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks, 2010.

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[16] Stanisław Hożyń. Convolutional neural networks for classifying electronic components in industrial applications. *Energies*, 16(2), 2023.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[18] Nikita Jaipuria, Xianling Zhang, Rohan Bhasin, Mayar Arafa, Punarjay Chakravarty, Shubham Shrivastava, Sagar Manglani, and Vidya N. Murali. Deflating dataset bias using synthetic data augmentation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3344–3353, 2020.

[19] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[20] J. Kaur and W. Singh. Tools, techniques, datasets and application areas for object detection in an image: a review. *Multimedia Tools and Applications*, 81:38297–38351, 2022.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90, 2012.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[25] Will Kurt. Kullback-leibler divergence explained, 2017. Zugriff am: 18. Februar 2025.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition, 1998.

[27] Yann Lecun, Leon Bottou, Genevieve Orr, and Klaus-Robert Müller. Efficient backprop, 08 2000.

[28] Yann Lecun, Larry Jackel, Corinna Cortes, John Denker, Harris Drucker, Isabelle Guyon, Urs Muller, Eduard Sackinger, Patrice Simard, and Vladimir Vapnik. Learning algorithms for classification: A comparison on handwritten digit recognition. *The Statistical Mechanics Perspective*, 07 2000.

[29] Dominik Lewy and Jacek Mańdziuk. An overview of mixing augmentation methods and augmentation strategies. *Artificial Intelligence Review*, 56(3):2111–2169, 2023.

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[31] Zimeng Lyu, AbdElRahman ElSaid, Joshua Karns, Mohamed Mkaouer, and Travis Desell. An experimental study of weight initialization and weight inheritance effects on neuroevolution, 2020.

[32] Francisco J. Moreno-Barea, Fiammetta Strazzera, José M. Jerez, Daniel Urda, and Leonardo Franco. Forward noise adjustment scheme for data augmentation. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 728–734, 2018.

[33] Yu. E. Nesterov. A method of solving a convex programming problem with convergence rate $o\left(\frac{1}{k^2}\right)$, 1983.

[34] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

[35] Lutz Prechelt. Early stopping - but when?, 03 2000.

[36] Chanachai Puttaruksa and Pinyo Taeprasartsit. Color data augmentation through learning color-mapping parameters between cameras. In *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–6, 2018.

[37] PyTorch. pytorch.org initialisierungsmethoden, 2025. Zugriff am: 19.02.2025.

[38] PyTorch. Tensors: A deeper look, 2025. Zugriff am: 26.02.2025.

[39] PyTorch. *torch.optim*, 2025. Zugriff am: 05.03.2025.

[40] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[42] I. H. Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3):160, 2021. Epub 2021 Mar 22.

[43] Dominik Schraml and Gunther Notni. Synthetic training data in ai-driven quality inspection: The significance of camera, lighting, and noise parameters. *Sensors (Basel)*, 24(2):649, Jan 19 2024.

[44] Mohamed A. Shahin, Holger R. Maier, and Mark B. Jaksa. Predicting settlement of shallow foundations using neural networks. *Journal of Geotechnical and Geoenvironmental Engineering*, 128(9):785–793, 2002.

[45] Jonathon Shlens. Notes on kullback-leibler divergence and likelihood, 2014.

[46] Khoshgoftaar Shorten. A survey on image data augmentation for deep learning. *Energies*, 2019.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[48] Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation. *CoRR*, abs/1708.06020, 2017.

[49] Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation, 2017.

[50] Antonio Torralba and Alexei A. Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528, 2011.

[51] J.M. Twomey and A.E. Smith. Performance measures, consistency, and power for artificial neural network models. *Mathematical and Computer Modelling*, 21(1):243–258, 1995.

[52] Allied Vision. Alvium 1800 u-500c datasheet, 2024.

[53] Allied Vision. *Alvium Features Reference*, 2024.

[54] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Emmanouil Protopapadakis. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, 2018:7068349, 2018. Epub 2018 Feb 1.

[55] Yulong Wu and John Tsotsos. Active control of camera parameters for object detection algorithms. *arXiv preprint arXiv:1705.05685*, 2017.

[56] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013.

[57] Donglai Zhu, Hengshuai Yao, Bei Jiang, and Peng Yu. Negative log likelihood ratio loss for deep neural network classification, 2018.

A Anhang

A.1 Zu dem Stand der Technik

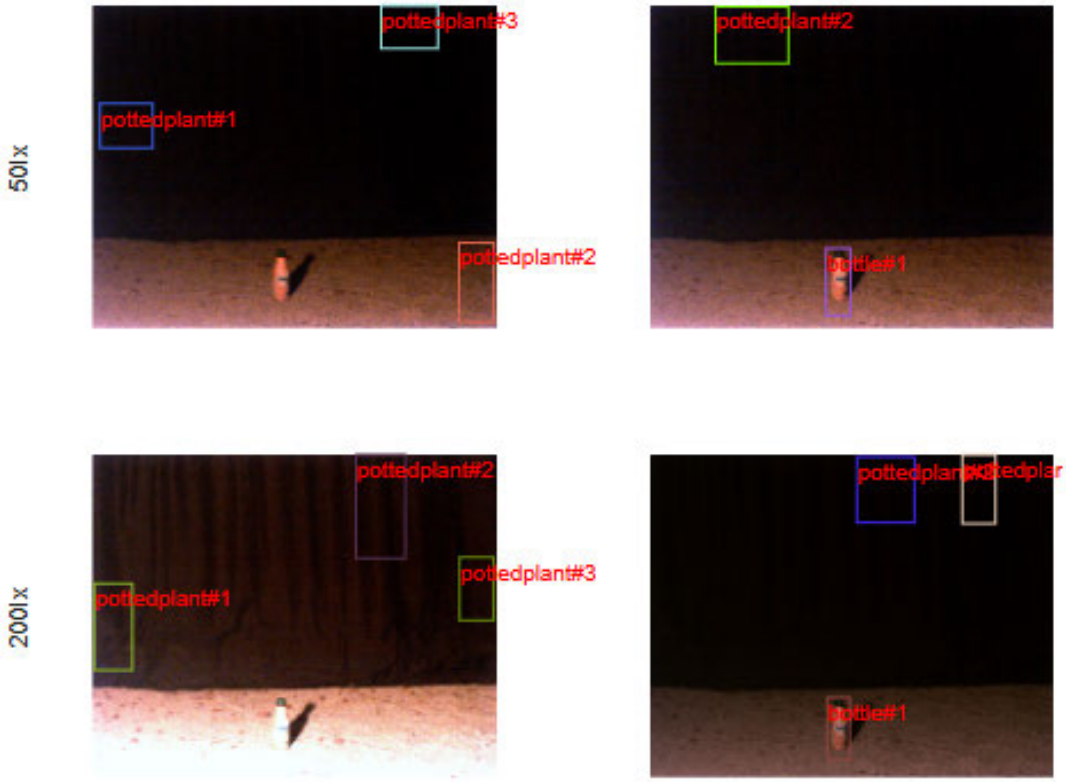
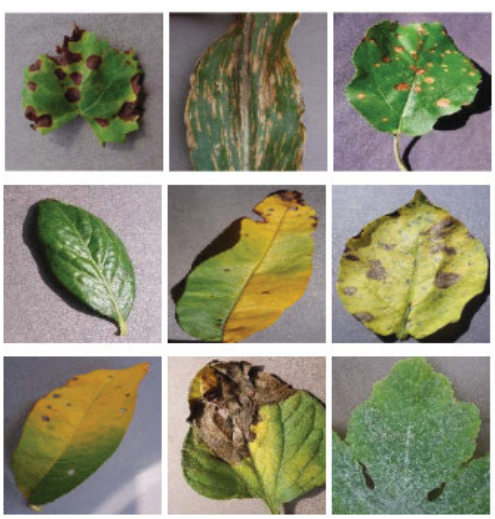
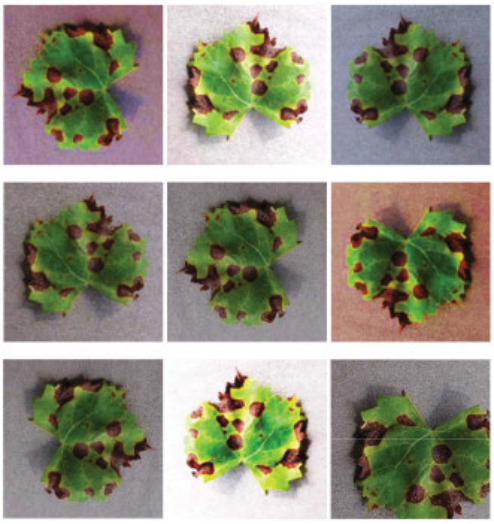


Abbildung A.1: Beispiel für sensorspezifischen Bias bei Objekterkennung unter 50 und 200 Lux (links: Autofokus, rechts: Algorithmus aus [55]).

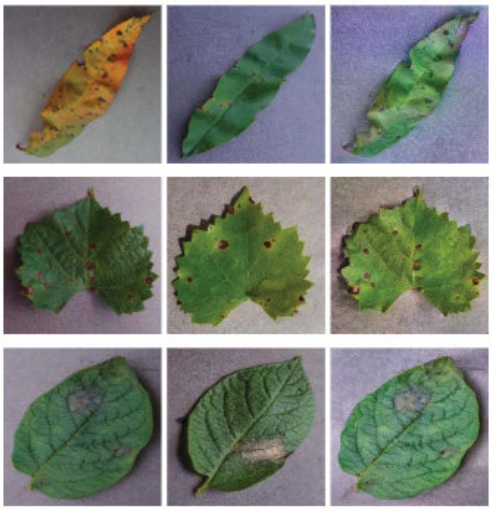


(a) Unveränderte Aufnahmen



(b) Normale Data Augmentation

(c) Änderung: DCGANs und WGANs



(d) Änderung: NST

Abbildung A.2: Beispiele für Data Augmentation bei der Erkennung von Pflanzenkrankheiten aus [5]

A.2 Zu den Versuchsergebnissen

#	Index	Ranking	DataSet	TrainLoss	TrainAcc	ValidationLoss	ValidationAcc	Precision	Recall	F1Score
1	400	0,6284	seq	0,000083	1,0	0,000000	1,0	0,2552	0,2632	0,2500
2	227	0,6227	dA	0,005170	0,9985	0,019632	0,9922	0,2542	0,2727	0,2466
3	354	0,6219	seq	0,000003	1,0	0,000000	1,0	0,2394	0,2618	0,2382
4	419	0,6203	seq	0,000006	1,0	0,000000	1,0	0,2395	0,2447	0,2323
5	415	0,6198	seq	0,000000	1,0	0,000000	1,0	0,2336	0,2638	0,2394
6	358	0,6187	seq	0,000002	1,0	0,000000	1,0	0,2355	0,2454	0,2318
7	396	0,6187	seq	0,030844	0,9963	0,000000	1,0	0,2370	0,2392	0,2288
8	396	0,6180	dA	0,033551	0,9943	0,000000	1,0	0,2332	0,2468	0,2270
9	376	0,6148	dA	0,000018	1,0	0,000000	1,0	0,2260	0,2440	0,2270
10	354	0,6135	dA	0,000005	1,0	0,000000	1,0	0,2202	0,2536	0,2293
11	352	0,6128	seq	0,000044	1,0	0,000000	1,0	0,2241	0,2317	0,2250
12	415	0,6116	dA	0,025964	0,9936	0,000000	1,0	0,2193	0,2392	0,2193
13	419	0,6103	dA	0,000000	1,0	0,000000	1,0	0,2165	0,2372	0,2169
14	376	0,6072	seq	0,000045	1,0	0,000000	1,0	0,2122	0,2228	0,2122
15	358	0,6068	dA	0,080622	0,9882	0,000000	1,0	0,2110	0,2235	0,2129
16	352	0,6063	dA	0,000002	1,0	0,000000	1,0	0,2060	0,2386	0,2184
17	400	0,6044	dA	0,061664	0,9936	0,000000	1,0	0,2059	0,2201	0,2085
18	376	0,5519	raw	0,000013	1,0	0,000063	1,0	0,0940	0,1435	0,0821
19	352	0,5500	raw	0,000002	1,0	0,000012	1,0	0,0886	0,1456	0,0882
20	358	0,5493	raw	0,000002	1,0	0,000012	1,0	0,0865	0,1470	0,0904
21	415	0,5476	raw	0,000010	1,0	0,000220	1,0	0,0802	0,1552	0,0874
22	396	0,5476	raw	0,000010	1,0	0,000220	1,0	0,0802	0,1552	0,0874
23	419	0,5412	raw	0,000022	1,0	0,001828	1,0	0,0669	0,1483	0,0834
24	400	0,5394	raw	0,000001	1,0	0,000040	1,0	0,0623	0,1449	0,0823
25	354	0,5368	raw	0,000034	1,0	0,000343	1,0	0,0568	0,1415	0,0733
26	227	0,5227	seq	0,049570	0,9878	0,277337	0,8828	0,2249	0,2331	0,1844
27	227	0,5062	raw	0,041859	0,9840	0,153927	0,9531	0,0887	0,1558	0,0888
28	264	0,1791	raw	1,931883	0,5797	1,370354	0,7335	0,0450	0,1511	0,0692
29	264	-0,1316	dA	1,946328	0,1352	1,946241	0,1172	0,0204	0,1429	0,0357
30	264	-0,1319	seq	1,946655	0,1295	1,947325	0,1172	0,0204	0,1429	0,0357

Tabelle A.1: Darstellung der Ergebnisse der Top 30 Modelle nach der Evaluierung, sortiert nach Ranking

A.3 Zusatz zur Konfiguration des Sequencers

Dieses Kapitel des Anhangs behandelt die Konfiguration des Sequencers. Zur Konfiguration des Sequencers muss ein aktueller Stream der Kamera und ein evtl. vorhandener Sequencer-Modus deaktiviert und der Sequencer-Konfigurationsmodus aktiviert werden. Anschließend können die Sequencer-Sets entsprechend der aktuellen Belichtungszeit der Kamera konfiguriert werden (siehe Listing A.1).

Für die Konfiguration der Sequencer-Sets gibt es eine globale Konfigurationsdatei, die die Parameter für die Sets enthält. Diese wird als Funktion aufgerufen und erhält als

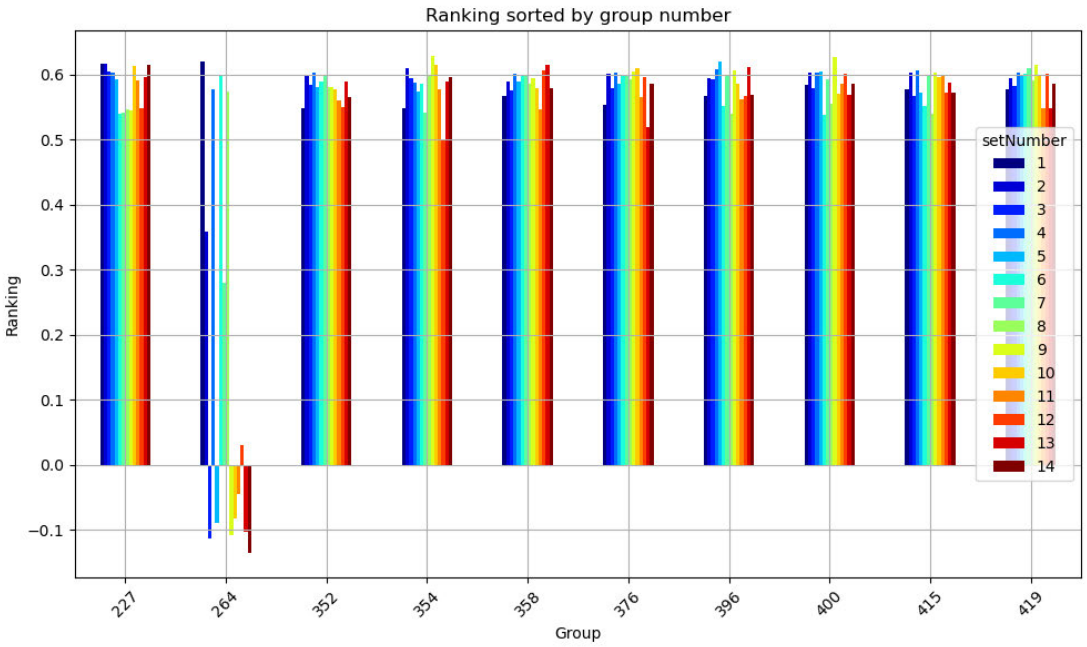


Abbildung A.3: Darstellung der Rankingwerte der Sequencer-Sets als Balkendiagramm

```
[...]
with cam:
    # stop the stream
    cam.stop_streaming()
    # first, turn off the sequencer mode
    cam.SequencerMode.set('Off')
    # turn on the configuration mode
    cam.SequencerConfigurationMode.set('On')

    # set the config sets for the set values
    CONFIG_SETS = global_settings.initialize_config_sets(cam.ExposureTime.get())
```

Listing A.1: Setup Konfiguration Sequencer

Übergabeparameter die Belichtungszeit und als Rückgabewert ein Dictionary mit den Sets (siehe Listing A.2). Mit der Belichtungszeit wird die aktuelle Einstellung der Kamera verwendet, so dass die Sequencer-Sets an die Umgebung angepasst werden können. Mit diesen Dictionaries von Sequencer Sets wird dann die Konfiguration der Sequencer-Sets durchgeführt (siehe Listing A.3). Es gibt das Default Dictionary, das die Standardwerte für die Sets enthält. Diese Werte sind die Werte, die die Kamera standardmäßig verwendet, wenn keine anderen Werte gesetzt sind und entsprechen den Werten aus dem Tutorial nach [53] und der aktuellen Belichtungszeit und dem Gain-Wert. Mit dieser Wertekombination wird dann jedes Sequencer-Set konfiguriert.

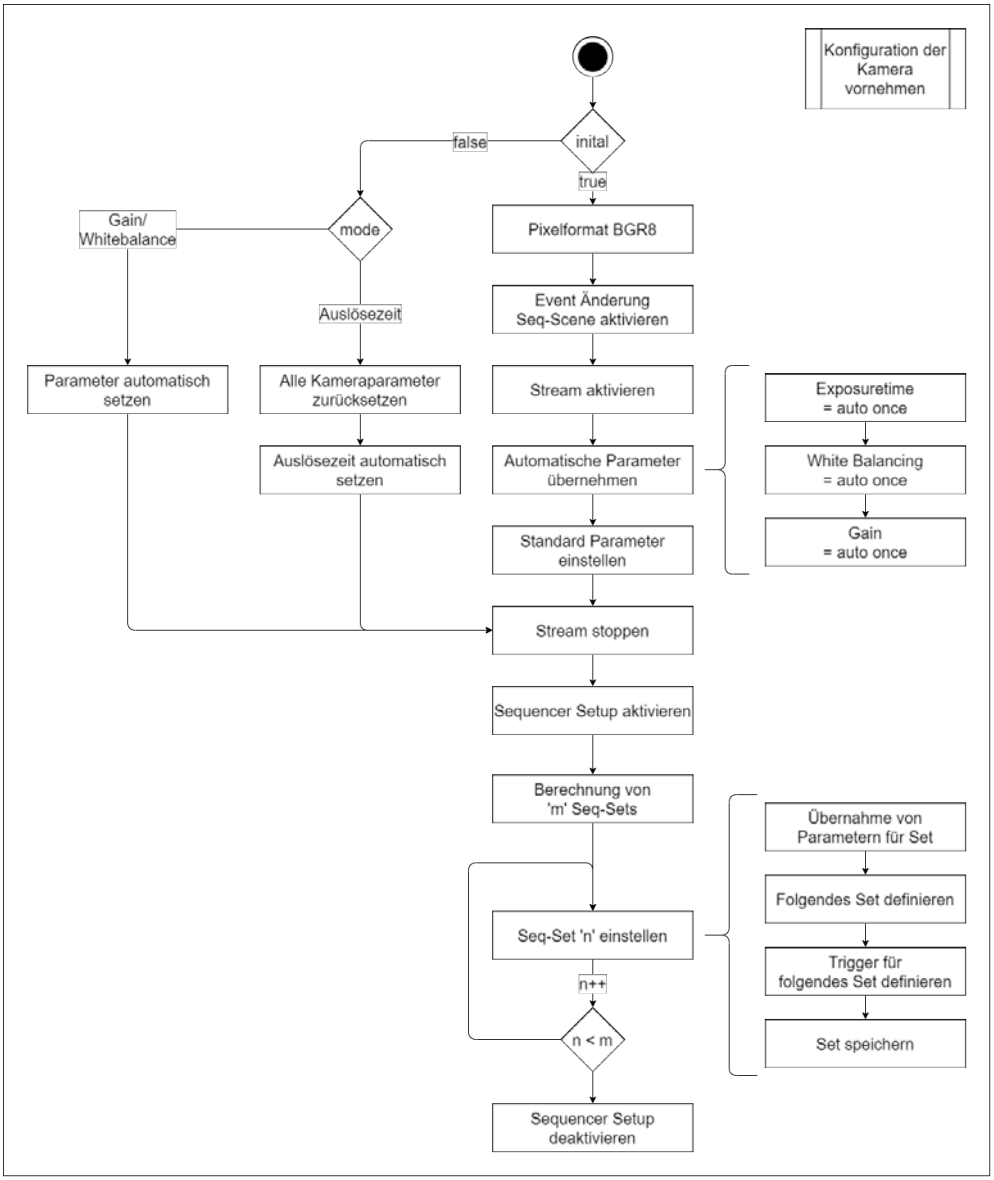


Abbildung A.4: Ablauf der Konfiguration des Sequencers

Zur Konfiguration jedes Sets wird der Code aus der Liste A.4 verwendet, der die entsprechenden Parameter setzt. Zuerst wird die Set-Nummer gesetzt, um das aktuelle Set zu konfigurieren. Danach werden Belichtungszeit, Gain, Gamma, Hue und Saturation entsprechend den Einstellungen im Dictionary gesetzt (siehe A.3). Für die verfügbaren RGB-Kanäle gibt es jeweils drei separate Gain-Werte, die in einem Dictionary gespeichert sind. Diese werden in der folgenden Schleife durchlaufen und die Werte für die

```python
def initialize_config_sets(exposureTime_env: float) -> dict:
    CONFIG_SETS = {
        # Set 0
        'crisp': {
            'Exposure': exposureTime_env, #environment value
            'Gain': 0,  #no gain
        },
        # Set 1
        'salt_pepper': {
            'Exposure': 42.0, #min ExposureTime
            'Gain': 45.0, #max Gain
        },
        [...]
    }
```

Listing A.2: Initialisierung der Sequencer-Sets

```python
# Apply the configuration sets
k = 0
for name, config in CONFIG_SETS.items():
    # use the default config for the other configurations
    full_config = {**DEFAULT_VALUES, **config}

    # save the config
    all_configs[name] = full_config

    sequencerSetConfigurator(
        cam, k=k, set_name=name, **full_config, inital_setup=inital_setup
    )
    k += 1
```

Listing A.3: Schleife zur Vergabe der Sequencer-Sets

Verstärkung der RGB-Kanäle gesetzt. Nun muss festgelegt werden, welcher Pfad für das nächste Set verwendet werden soll. Jedes Set kann mehrere Pfade mit unterschiedlichen Triggern zum Verlassen des aktuellen Sets haben; in diesem Fall wird pro Set nur ein Pfad verwendet (siehe Abbildung 6.4), der mit einer positiven Flanke des Triggers „ExposureActive“aktiviert wird. Danach wird pro Set das nächste Set gesetzt, das bei Aktivierung des Triggers aktiviert wird. Ist das letzte Set erreicht, wird hier festgelegt, dass das nächste Set das erste Set ist und der Trigger „SoftwareSignal0“gesetzt wird. Wichtig ist hier noch zu beachten, dass das Sequencer-Set anschließend gespeichert werden muss, damit die Konfiguration nicht verloren geht. Zum Schluss, bevor der Stream wieder gestartet werden kann, muss der Konfigurationsmodus des Sequencers wieder deaktiviert werden. Damit Aufnahmen mit dem Sequencer gemacht und gespeichert werden können, muss ein Event-Handler für Sequencer-Änderungen erstellt werden. Dieser Prozess läuft im Hintergrund und wird aktiviert, wenn ein Sequencer-Set geändert wird. Bei einem Setwechsel wird das aktuelle Bild mit den geänderten Kameraparametern in

```
with cam:
    cam.SequencerSetSelector.set(k)  # set the current set to k

    # configurate the new parameters
    cam.ExposureTime.set(Exposure)
    cam.Gain.set(Gain)
    cam.Gamma.set(Gamma)
    cam.Hue.set(Hue)
    cam.Saturation.set(Saturation)

    # Loop through the dictionary and set values
    for gain_selector, gain_value in gain_values.items():
        cam.ColorTransformationValueSelector.set(gain_selector)
        cam.ColorTransformationValue.set(gain_value)

    # configurate the path for the set k
    cam.SequencerPathSelector.set(0)  # path0 for set k
    cam.SequencerTriggerActivation.set(
        'RisingEdge'
    )  # activate path k with an rising edge of any source

    if k < utils.NUMBER_SETS:
        # the next set for path0 is set k+1
        cam.SequencerSetNext.set(k + 1)
        cam.SequencerTriggerSource.set(
            'ExposureActive'
        )  # path k is activated with an active exposure
    elif k == utils.NUMBER_SETS:  # for setX the next set is 0
        cam.SequencerSetNext.set(0)
        cam.SequencerTriggerSource.set(
            'SoftwareSignal0'
        )  # path k is activated with SoftwareSignal0

    cam.SequencerSetSave.run()  # save the current set to the memory
```

Listing A.4: Vergabe von den Sequencer-Parametern für ein Set

Abhängigkeit vom Set genommen und mit der korrekten Nummerierung gespeichert. Um diesen Nebenprozess mit dem Hauptprogramm zu synchronisieren, wird das Modul utils verwendet, das die Variable set_done enthält. Diese sorgt dafür, dass das Hauptprogramm keine weitere Sequencer-Aufnahme starten kann, bis alle Aufnahmen des aktuellen Sets gemacht wurden.

Bei dieser Umsetzung gab es Probleme mit der Größe des Kamerapuffers, da dieser nach einer gewissen Zeit immer voll war und somit keine weiteren Aufnahmen mehr gemacht werden konnten. Dies führte dazu, dass keine neuen Aufnahmen mehr gemacht werden konnten und somit die Sequencer-Sets nicht mehr durchgeschaltet werden konnten und somit die Aufnahme der Datensätze nicht erfolgreich war. Um dieses Problem zu lösen, wurde das Intervall der Sequencer-Aufnahmen verkürzt, so dass der Sequencer die Aufnahmen schneller abarbeiten konnte und somit der Puffer nicht mehr voll war.

```python
# event handler for incoming sequence changes
def feature_changed_handler(feature, handler):
    frame = handler.get_image()

    # check if frame is aviable and this is not set0
    if frame is not None and utils.set_number > 0 and not utils.set_done:
        if utils.picture_number == utils.NUMBER_SETS:
            utils.set_done = True

        path = (
            utils.directonary_path + f'/{utils.set_number}_{utils.picture_number}.png'
        )

        # for numbering the pictures for each set
        cv2.imwrite(path, frame)
        utils.picture_number += 1
```

Listing A.5: Sequencer Event Handler

Zusätzlich wurde eine flush_queue Funktion erstellt, die den Puffer der Kamera leert, wenn er voll ist. Diese Maßnahmen führten dazu, dass die Aufzeichnung der Datensätze erfolgreich war und somit die Sequencer-Sets erfolgreich konfiguriert werden konnten.

Ursprünglich war geplant, dass die Kamerasoftware die Aufnahmen für den Sequencer und anschließend auch für die Datenanreicherung übernimmt und diese in getrennten Ordnern abspeichert. Diese Vorgehensweise wurde jedoch verworfen, da eine Data Augmentation in der Kamerasoftware zu Zeitproblemen bei der Speicherung und Verarbeitung der Bilder führte.

Index

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original