Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

HAW HAMBURG

Masterarbeit

Lia Bansmann

Automatische Klassifikation von Schichtwiderständen mittels Bildverarbeitung und Deep Learning

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

Lia Bansmann

Automatische Klassifikation von Schichtwiderständen mittels Bildverarbeitung und Deep Learning

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. Sönke Appel

Eingereicht am: 18. Januar 2024

Lia Bansmann

Thema der Arbeit

Automatische Klassifikation von Schichtwiderständen mittels Bildverarbeitung und Deep Learning

Stichworte

Deep Learning, Bildverarbeitung, Widerstände, OpenCV, Keras, Tensorflow, Faltungsnetz

Kurzzusammenfassung

In dieser Arbeit wird ein Bildverarbeitungsalgorithmus unter Verwendung von OpenCV in C++ entwickelt. Dieser Algorithmus zielt darauf ab, Widerstände anhand selbst erstellter Bilder zu lokalisieren und auszuschneiden. Zusätzlich wird ein Convolutional Neural Network (CNN) in Python mithilfe von TensorFlow mit den extrahierten Widerstandsdaten trainiert. Dies ermöglicht schließlich eine Echtzeitklassifizierung von Widerständen, die vor eine Webcam gehalten werden.

Lia Bansmann

Title of Thesis

Automatic classification of film resistors using image processing and Deep Learning

Keywords

Deep Learning, Image Processing, Resistors, OpenCV, Keras, Tensorflow, Convolutional Neural Network

Abstract

This paper focuses on developing an image processing algorithm using OpenCV in C++. The primary objective of this algorithm is to locate and extract resistors from self-generated images. Furthermore, a Convolutional Neural Network (CNN) is trained in Python using TensorFlow with the extracted resistor data. Ultimately, this facilitates real-time classification of resistors held in front of a webcam.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungen

AdaGrad Adaptive Gradient Descent.

Adam Adaptive Moment Estimation.

ANN Artificial Neural Network.

BB Bounding Box.

BLOB Binary Large Object.

CNN Convolutional Neural Network.

ELU Exponential Linear Unit.

JPEG Joint Photographic Experts Group.

k-NN k Nearest Neighbor.

KI Künstliche Intelligenz.

KNN Künstliches neuronales Netz.

MSE Mean Squared Error.

NN Neural Network.

ReLU Rectified Linear Unit.

RMSprop Root Mean Square Propagation.

SELU Scaled Exponential Linear Unit.

SGD Stochastic Gradient Descent.

SVM Support Vector Maschine.

VGG Visual Geometry Group.

YOLO You only look once.

1 Einleitung

Künstliche Intelligenz ist heutzutage dank Plattformen wie ChatGPT allgegenwärtig und ein viel diskutiertes Thema. Früher war künstliche Intelligenz lediglich eine Vision aus der Science-Fiction. Heute befinden wir uns in einer Ära, in der sich ständig neue Forschungsgebiete und praktische Anwendungen eröffnen. Ein Teilgebiet der künstlichen Intelligenz ist das Machine Learning. Hierbei erlernen Systeme anhand von Rohdaten Muster zu erkennen und dieses erlangte Wissen auf neue Daten anzuwenden. Eine spezielle Methode des Machine Learnings ist das Deep Learning, bei dem komplexe künstliche neuronale Netze mit einer umfangreichen inneren Struktur gebildet werden. Insbesondere in der Bildverarbeitung kommen sogenannte Convolutional Neural Network (CNN) zum Einsatz, welche Faltungsoperationen nutzen. Die Bildverarbeitung befasst sich mit der Verarbeitung von Signalen, die Bilder repräsentieren. In diesem Zusammenhang könnten elektrische Bauteile identifiziert und klassifiziert werden, indem auf einer Platine Muster und Strukturen analysiert werden. Widerstände sind elektrische passive Bauelemente, die beispielsweise den Strom begrenzen oder verteilen können.

In dieser Arbeit werden verschiedene Ansätze zur Klassifizierung von Widerständen auf Bildern mithilfe von Deep Learning und Bildverarbeitung entwickelt und getestet. Dabei liegt der Fokus auf der Analyse und Vergleich verschiedener Methoden, um die Effizienz und Genauigkeit der Widerstandsklassifizierung zu verbessern. Die Motivation besteht darin, am Ende ein Programm zu entwickeln, das mithilfe von Bildverarbeitung und Deep Learning in der Lage ist, Widerstände, die vor die Webcam gehalten werden, zu klassifizieren.

Im Folgenden wird der Aufbau der Arbeit kurz erläutert. Kapitel 2 dient der Erarbeitung eines fundierten Wissensstands, um den Leser in die Thematik einzuführen und ein Verständnis für die Arbeit zu ermöglichen. Das Kapitel 3 wird der aktuelle Forschungsstand zur Klassifizierung von Widerständen erläutert und wichtige Forschungsarbeiten werden vorgestellt. Kapitel 4 beschäftigt sich mit der Zielsetzung, Systembeschreibung und den Anforderungen. Kapitel 5 diskutiert die Ansätze, Anforderungen und den Arbeitsablauf

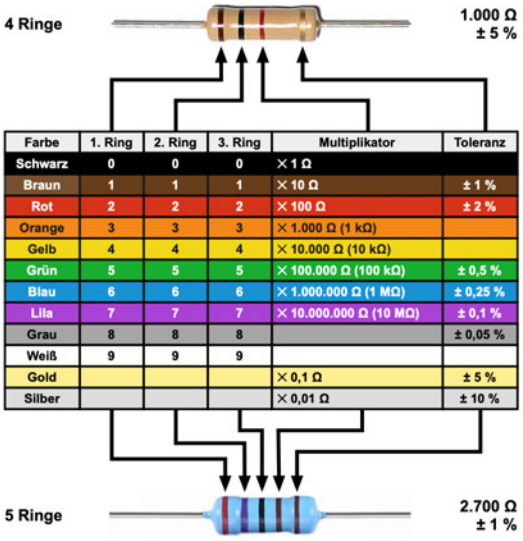
der Arbeit. Im Kapitel 6 liegt der Fokus auf der Aufnahme der Widerstandsbilder und den dabei zu berücksichtigenden Aspekten. Kapitel 7 umfasst das Design, die Implementierung und die Bewertung des Bildverarbeitungsalgorithmus zur Lokalisierung der Widerstände. Das Kapitel 8 widmet sich dem Entwurf, der Umsetzung und der Auswertung des CNN, für die Widerstandsklassifizierung. Im Kapitel 9 werden die Ergebnisse beschrieben und verifiziert. Abschließend erfolgt in Kapitel 10 eine Zusammenfassung und einen Ausblick auf mögliche Weiterentwicklungen.

2 Grundlagen

In diesem Kapitel werden die Grundlagen gelegt, welche für den Verlauf der Arbeit vorausgesetzt werden.

2.1 Widerstandsfarbcode

Es existieren verschiedene Arten von Festwiderständen, darunter Schichtwiderstände und Drahtwiderstände. Unter den Schichtwiderständen sind zwei gängige Typen zu finden: Metallschichtwiderstände, die typischerweise über 5 Ringe verfügen und oft einen hellblauen Widerstandskörper aufweisen, sowie Kohleschichtwiderstände mit 4 Ringen und einem beigen Widerstandskörper [11]. Die Farbcodierungen der Ringe entsprechen der DIN 41429-Norm und tragen unterschiedliche Kennzeichnungen. Die Funktionen und Bedeutungen der Farbringe werden in Abbildung 2.1 dargestellt.



Farbe	1. Ring	2. Ring	3. Ring	Multiplikator	Toleranz
Schwarz	0	0	0	× 1 Ω	
Braun	1	1	1	× 10 Ω	± 1 %
Rot	2	2	2	× 100 Ω	± 2 %
Orange	3	3	3	× 1.000 Ω (1 kΩ)	
Gelb	4	4	4	× 10.000 Ω (10 kΩ)	
Grün	5	5	5	× 100.000 Ω (100 kΩ)	± 0,5 %
Blau	6	6	6	× 1.000.000 Ω (1 MΩ)	± 0,25 %
Lila	7	7	7	× 10.000.000 Ω (10 MΩ)	± 0,1 %
Grau	8	8	8		± 0,05 %
Weiß	9	9	9		
Gold				× 0,1 Ω	± 5 %
Silber				× 0,01 Ω	± 10 %

Abbildung 2.1: Farbringaufschlüsselung [11]

Bei Kohleschichtwiderständen repräsentieren die ersten beiden Ringe den Wert, der dritte Ring den Multiplikator und der vierte Ring die Toleranz. Im Gegensatz dazu geben bei Metallschichtwiderständen die ersten drei Ringe den Wert an. Der vierte Ring steht für den Multiplikator und der fünfte für die Toleranz [11].

2.2 Bildverarbeitung

In der Informatik und Elektrotechnik bezieht sich Bildverarbeitung auf die Bearbeitung von Signalen, die visuelle Informationen darstellen, wie zum Beispiel Fotografien oder Einzelbilder aus Videos. Das Resultat der Bildverarbeitung kann entweder ein verändertes Bild sein oder eine Sammlung von Merkmalen, die aus dem Eingangsbild extrahiert wurden, wie es bei der Bilderkennung der Fall ist [21].

2.2.1 Kontrast

Als Kontrast wird der Unterschied zwischen hellen und dunklen Bereichen eines Bildes bezeichnet. Genauer gesagt bezeichnet der Kontrast ein Unterscheidungsmerkmal für den Helligkeitsverlauf. Im Generellen wird der Kontrast anhand der Modulationsübertragungsfunktion in Bezug auf die Auflösung beschrieben. Die mathematische Darstellung des Vergleichs zwischen dem Detailkontrast an den Kanten eines Objekts und dem Detailkontrast in seiner bildlichen Darstellung wird als Modulationsübertragungsfunktion bezeichnet. Durch die Leuchtdichten L_{max} und L_{min} wird der Kontrast definiert. Es existieren verschiedene Kontrast Definitionen. Allgemein kann der Kontrast als

$$K = L_{max} - L_{min} \tag{2.1}$$

bezeichnet werden. Der Weber-Kontrast ist wie folgt definiert [20].

$$K_w = \frac{L_{max} - L_{min}}{L_{min}} \text{ mit } 0 \leq K_w \leq \infty \tag{2.2}$$

Dieser Kontrast findet oft Anwendungen in kleinen Sehobjekten, wie Leuchtdioden in großem Umfeld. Der Michelson-Kontrast, auch als Modulation bekannt, findet Anwendung, wenn eine klare Unterscheidung zwischen Vordergrund und Hintergrund nicht eindeutig möglich ist. Dies tritt beispielsweise bei periodischen Objekten wie Gittern oder

benachbarten Sehobjekten ähnlicher Größe auf. Er wird definiert als das Verhältnis zwischen der Amplitude und dem Durchschnitt der Leuchtdichteverteilung [20]. Die Gleichung 2.3 zeigt die mathematische Definition des Michaelson-Kontrast.

$$K_m = \frac{L_{max} - L_{max}}{L_{max} + L_{min}} \text{ mit } 0 \leq K_m \leq 1 \tag{2.3}$$

Das Leuchtdichteverhältnis kann als Kontrastdefinition zur Kennzeichnung optischer Anzeigen verwendet werden [20].

$$K = \frac{L_{max}}{L_{min}} \tag{2.4}$$

Bei der Kontrastmaximierung wird das Histogramm so ausgedehnt, dass die Graustufen den Wertebereich ganz ausnutzen. Es handelt sich um eine ortsunabhängige Punktoperation. Mathematisch wird eine Funktion gesucht, die den kleinsten Grauwert g_{min} auf $g = 0$ abbildet und den größten Wert g_{max} auf $g = 2^b - 1$ [8].

$$f(g(x,y)) = (2^b - 1) \cdot \frac{g(x,y) - g_{min}}{g_{max} - g_{min}} \tag{2.5}$$

2.2.2 Morphologische Operationen

Als mathematische Morphologie wird in der Bildverarbeitung ein theoretisches Modell für digitale Bilder bezeichnet. Diese ist eng mit der Theorie der Menge verbunden [21]. Durch morphologische Operationen wird die Gestalt von Objekten verändert, indem an den Rändern Pixel entfernt oder hinzugefügt werden [8].

Bei der Erosion werden äußere Pixel am Rand eines Objektes entfernt. Dabei wird ein sogenanntes Strukturelement (auch Filtermaske), welches festlegt welche Bildpunkte der Nachbarschaft bei der Operation berücksichtigt werden, definiert. Für die Filtermaske h gilt: $h(x,y) \in \{0,1\}$. Bei der Erosion wird der Wert $g(x,y)$ durch den kleinsten Wert unter allen Pixeln in der Nachbarschaft mit $h = 1$ ersetzt. Pixel, bei denen $h = 0$ zutrifft, bleiben bei dieser Operation unverändert [8].

$$g(x,y) \ominus h(x,y) = \{\min(g(x+m, y+n)) \mid h(m,n) = 1\} \tag{2.6}$$

Die Abbildung 2.2 veranschaulicht das Verfahren.

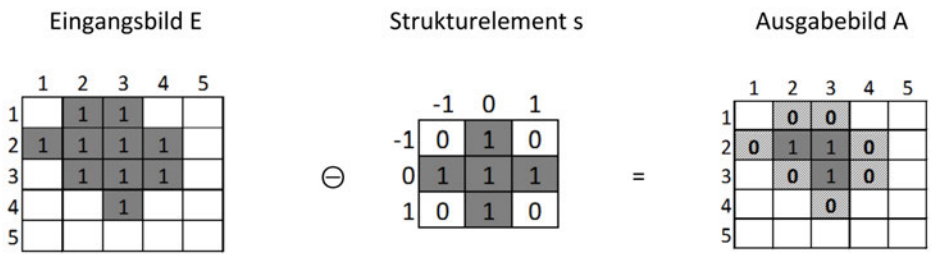


Abbildung 2.2: Erosion [3]

Im Gegensatz zur Erosion vergrößert die Dilatation die Pixel am Rand des Objekts. In diesem Zusammenhang wird der Wert $g(x, y)$ durch den größten Wert unter allen Pixeln in der Nachbarschaft mit $h = 1$ ersetzt. Pixel, bei denen $h = 0$ zutrifft, bleiben bei dieser Operation unberücksichtigt [8].

$$g(x, y) \oplus h(x, y) = \{\max(g(x + m, y + n)) \mid h(m, n) = 1\} \tag{2.7}$$

Die Abbildung 2.3 veranschaulicht die Dilatation.

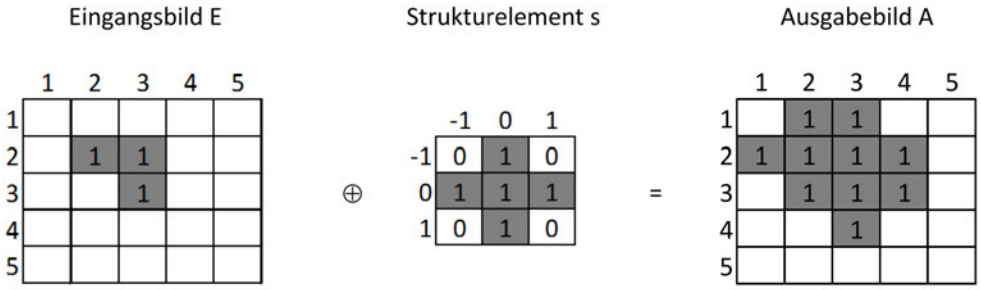


Abbildung 2.3: Dilatation [3]

Auch Verknüpfungen beider Operationen sind möglich. Beim sogenannten Opening wird zuerst eine Erosion durchgeführt und anschließend eine Dilatation angehängt. Der Vorteil ist hierbei, dass durch die Erosion störende Elemente entfernt werden und anschließend durch die Dilatation die gebliebenen Elemente wieder vergrößert werden [8].

$$g(x, y) \circ h(x, y) = (g(x, y) \ominus h(x, y)) \oplus h(x, y) \tag{2.8}$$

Beim sogenannten Closing wird zuerst eine Dilatation und anschließend eine Erosion durchgeführt. Durch diese Operation können beispielsweise Löcher durch die Dilatation gefüllt werden. Bei der anschließenden Erosion bleiben sie weiterhin gefüllt. Durch die Dilatation haben sich aber auch die Ränder vergrößert, welche durch die Erosion wieder verkleinert werden [8].

$$g(x,y) \bullet h(x,y) = (g(x,y) \oplus h(x,y)) \ominus h(x,y) \tag{2.9}$$

2.2.3 Segmentierung

Bei der Segmentierung werden Objekte voneinander getrennt. Im einfachsten Fall ist damit die Trennung von Hinter- und Vordergrund gemeint [8]. Es gibt verschiedene Verfahren zur Segmentierung. Bei allen Verfahren wird jedem Bildpunkt eine bestimmte Eigenschaft zugeordnet und mithilfe dieser Eigenschaften die Bildpunkte zu Objekten zusammengesetzt. Oft wird der Grauwert als Eigenschaft genutzt, dieser reicht vom Wert 0 (Schwarz) bis 255 (Weiß) und gibt die Helligkeit an [10].

Pixelorientierte Verfahren entscheiden für jedes Pixel einzeln, zu welchem Objekt es gehört. Ein Beispiel ist der globale Schwellwert. Dabei wird ein globaler Schwellwert τ festgelegt und je nachdem ob der Pixelwert unter oder über der Schwelle liegt, wird das Pixel zum Hinter- oder Vordergrund gezählt [10]. Für ein binäres Bild wird der Pixelwert entweder auf 0 oder 255 festgelegt. Dabei repräsentiert der Pixelwert 0 die Farbe Schwarz, während 255 die Farbe Weiß darstellt.

$$\tilde{g}(x,y) = \begin{cases} 0 & : g(x,y) \leq \tau \\ 255 & : g(x,y) > \tau \end{cases} \tag{2.10}$$

Beim adaptiven Schwellwertverfahren wird für jedes Pixel eine Nachbarschaft N betrachtet. Mithilfe dieser Nachbarschaft wird ein Schwellwert $\tau(N)$ berechnet [10]. Die Berechnungsvorschrift für jedes Pixel lautet wie folgt.

$$\tilde{g}(x,y) = \begin{cases} 0 & : g(x,y) < \tau(N(x,y)) \\ 255 & : g(x,y) \geq \tau(N(x,y)) \end{cases} \tag{2.11}$$

Kantenorientierte Verfahren suchen im Bild nach Kanten und Konturen. Ein kantenbasierter Algorithmus ist der Canny-Kantendetektor. Canny betrachtet dabei das eindi-

mensionale Signal senkrecht zur jeweils betrachteten Kante im Bild. Als Kantendetektor identifiziert er das Maximum der Ableitung des rauschgefilterten Signals. Die Umsetzung des Canny-Kantendetektors erfolgt in der Regel in einem dreistufigen Verfahren [22]:

- Glättung des Bildes mithilfe eines Gauss'schen Kerns
- Ermittlung des Gradienten durch Kantenfilterung
- Verfolgung von Kanten (Unterdrückung von Nicht-Maxima und Verfolgung mit Hysterese)

2.2.4 Regionen

In Binärbildern ist eine klare Trennung zwischen Vordergrund und Hintergrund möglich. Zusammenhängende Bereiche im Vordergrund werden als Binary Large Object (BLOB) bezeichnet. Es existieren verschiedene Techniken zur Identifizierung solcher Regionen in Bildern. Eine dieser Techniken ist die sequentielle Regionenmarkierung, auch als Region Labeling bekannt.

Dieses Verfahren besteht aus zwei essenziellen Schritten. Zunächst wird das Bild sequentiell von links oben nach rechts unten durchlaufen, wobei alle Vordergrundpixel vorläufig markiert werden. Anschließend wird in einem zweiten Schritt die Nachbarschaft der markierten Pixel analysiert. Bei der N8-Nachbarschaft werden beispielsweise alle acht umliegenden Pixel berücksichtigt. Falls das analysierte Pixel die Anforderungen der Nachbarschaft nicht erfüllt, wird es dem Hintergrund zugeordnet.

Abbildung 2.4: Nachbarschaften [8]

Eine alternative Methode ist die sogenannte Flood Fill Methode. Hierbei wird zunächst ein unmarkiertes Vordergrundpixel gesucht und die zusammenhängenden Pixel daraufhin auf ihre Nachbarschaft hin überprüft und markiert. Die Suche und Markierung erfolgen auf ähnliche Weise wie das Ausbreiten einer Flutwelle [24].

2.2.5 Momente

Als Momente werden bestimmte gewichtete Mittelwerte bezeichnet, die sich aus den Helligkeitswerten der einzelnen Pixel eines Bildes bestimmen. Momente können zur Lage und Orientierungsbeschreibung nützlich sein. Außerdem dienen sie als Merkmale zur Klassifikation von Objekten. Es gibt Flächenmomente, Linienmomente und Punktmomente, welche sich jeweils anhand der geometrischen Einteilung unterscheiden. In dieser Arbeit werden nur Flächen betrachtet, deswegen werden die Flächenmomente im Folgenden genauer beschrieben [21].

Die allgemeine Form des Moments der Ordnung p + q für die Grauwertfunktion g(x,y) ergibt sich wie folgt.

$$m_{p,q} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y) \cdot x^p y^q \tag{2.12}$$

Betrachtet man das 0te Moment wird ersichtlich, warum dies die Fläche der Funktion wiedergibt.

$$m_{0,0} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y) \cdot x^0 y^0 = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y) \tag{2.13}$$

Das Moment 1. Ordnung gibt den Schwerpunkt an [21].

$$S_x = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y) \cdot x}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y)} = \frac{m_{1,0}}{m_{0,0}} \tag{2.14}$$

$$S_y = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y) \cdot y}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y)} = \frac{m_{0,1}}{m_{0,0}} \tag{2.15}$$

In die Berechnung der Momente geht die x- und y-Position mit ein, dies ist aber nicht immer erwünscht. Die translationsinvarianten Merkmale betrachtet das Objekt unabhängig von seiner Position [3].

$$\mu_{p,q} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} g(x,y) \cdot (x - x_s)^p \cdot (y - y_s)^q \tag{2.16}$$

Die Exzentrizität gibt die Relation von der Breite des Objektes zu dessen Länge an. Der Wertebereich reicht von 0 bis 1. Bei einem Wert von 1 handelt es sich um eine Linie und

bei 0 um einen Kreis [5]. Mit folgender Formel lässt sich die Exzentrizität berechnen.

$$E = \frac{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2}{(\mu_{20} + \mu_{02})^2} \in [0, 1] \tag{2.17}$$

Ein weiteres Merkmal ist die Orientierung. Diese gibt die Richtung der Hauptachse an. Die Hauptachse bezeichnet die Gerade, die durch den Schwerpunkt entlang der größten Ausdehnung des Objektes verläuft [3].

$$\Theta = \frac{1}{2} \cdot \arctan\left(\frac{2 \cdot \mu_{11}}{\mu_{20} - \mu_{02}}\right) \tag{2.18}$$

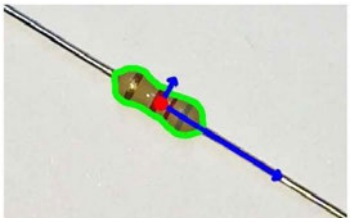


Abbildung 2.5: Orientierung und Hauptachse [13]

Abbildung 2.5 illustriert das Prinzip der Orientierung. Der rote Punkt repräsentiert den Schwerpunkt, während der lange blaue Pfeil die Richtung der Hauptachse zeigt.

2.2.6 LAB-Farbraum

Der Lab-Farbraum (auch: CIELAB) nutzt einen dreidimensionalen Farbraum, bei dem der Helligkeitswert L senkrecht auf der Farbebene (a,b) steht. Abbildung 2.6 veranschaulicht den LAB-Farbraum. Bei diesem Farbraum wird sich die Gegenfarbtheorie von Ewald Hering zur Hilfe genommen. Analog zu dieser Theorie gibt die a-Koordinate die Farbart und Farbintensität zwischen Grün und Rot an und die b-Koordinate die Farbart und die Farbintensität zwischen Blau und Gelb. Je größer die positiven a- und b-Werte und je kleiner die negativen a- und b-Werte sind, umso intensiver ist der Farbton [21].

[1] https://www.druckerei-konstanz.de/technik/farbmanagement/

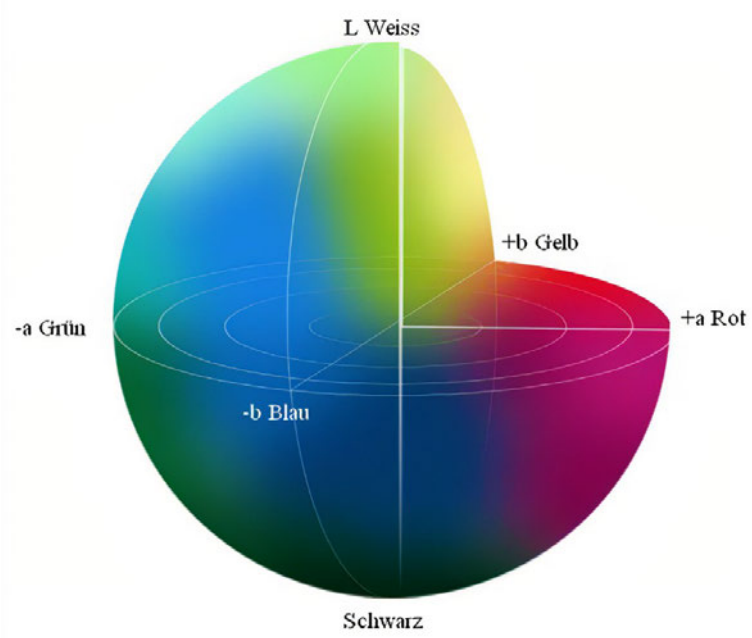


Abbildung 2.6: LAB-Farbraum [1]

2.2.7 Filter

Mithilfe von Filtern lassen sich Bilder verändern. Zu den linearen Filtern gehören beispielsweise Glättungsfilter, welche das Bild glätten.

Gaußfilter

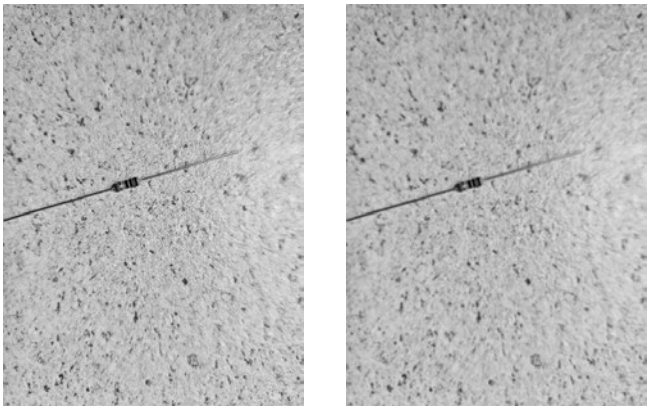
Bei Faltungsfiltern wird ein Filterkern auf jedes Pixel des Grauwertbildes einzeln angewendet, wobei die Werte der überlappenden Pixel miteinander multipliziert werden. Danach können alle erhaltenen Produkte aufaddiert werden und durch die Anzahl aller Werte des Faltungskerns geteilt werden [8].

Beim Gaußfilter oder auch Gauß'scher Weichzeichner lassen sich die 2D-Kerne durch zwei 1D-Kerne erzeugen. Die Kerne entsprechen einer abgetasteten und auf Eins normierten Normalverteilung, deren Breite über die Standardabweichung σ angepasst werden kann [8].

$$h(x,y) = \frac{1}{2\pi\sigma} \cdot e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{2.19}$$

Abbildung 2.7 zeigt die Anwendung des Gaußfilters auf ein Widerstandsbild. Die Größe des Filterkerns (K_{size}) beträgt 3x3. Wenn für den Parameter σ der OpenCV-Funktion kein Wert übergeben wird, berechnet sich dieser aus der Größe des Filterkerns wie folgt [15].

$$\sigma = 0,3 \cdot ((K_{size} - 1) \cdot 0,5 - 1) + 0,8 \tag{2.20}$$



(a) Original Bild (b) Anwendung Filter

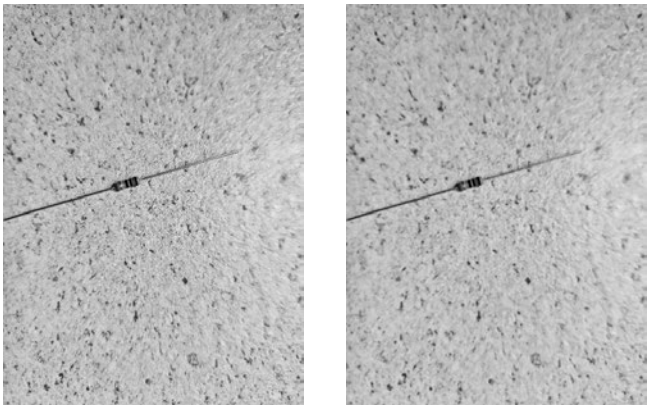
Abbildung 2.7: Gauß Filter

Median Filter

Beim Median Filter handelt es sich um einen Rangordnungsfilter. Bei dieser Art von Filtern werden die Pixelwerte einer Nachbarschaft dem Grauwert nach aufsteigend sortiert. Anschließend werden die Grauwerte in einer KxK großen Nachbarschaft betrachtet und beispielsweise durch den kleinsten Wert ersetzt (Minimumfilter). Beim Median Filter werden die Grauwerte durch den mittleren Wert ersetzt [8].

$$\text{median}_{\text{K}}(\text{g}(\text{x},\text{y})) = \text{rank}\left(g(x,y), \frac{K^2+1}{2}\right) \tag{2.21}$$

Median Filter werden oft angewendet, wenn nur einzelne Pixel stark verrauscht sind. Dies ist zum Beispiel beim sogenannten Salt & Pepper Rauschen der Fall.



(a) Original Bild (b) Anwendung Filter

Abbildung 2.8: Median Filter mit Filterkern 3x3

2.3 Deep Learning

Der Begriff Deep Learning bezeichnet tiefschichtige Neuronale Netze [18].

2.3.1 Neural Network

Der Psychologe Frank Rosenblatt wird als Schöpfer des ersten neuronalen Netzes angesehen, welches er als das Perzeptron im Jahr 1958 vorstellte. Dieses einfache neuronale Netz besteht aus einem Neuron, das Eingabewerte (Inputs) erhält und einen Ausgabewert (Output) generiert. Die Darstellung des Perzeptrons, das in Abbildung 2.9 dargestellt ist, erhält zwei Inputs, x_1 und x_2, welche mit den Gewichten w_1 und w_2 multipliziert werden. Anschließend werden diese gewichteten Summen addiert. Wenn die resultierende gewichtete Summe größer als ein definierter Schwellwert θ ist, beträgt der Output O 1, andernfalls 0 [18].

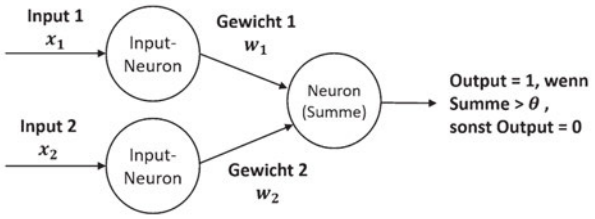


Abbildung 2.9: Erstes Perzeptron [18]

Folgende Formel beschreibt das Netz.

$$O = \begin{cases} 0 & : x_1 w_1 + x_2 w_2 \leq \theta \\ 1 & : x_1 w_1 + x_2 w_2 > \theta \end{cases} \tag{2.22}$$

Die Eingabewerte sind festgelegte numerische Werte, während die Gewichte im Netzwerk durch das Lernen angepasst werden. In realen Anwendungen besteht ein Netzwerk in der Regel aus mehreren Neuronen und Schichten. Jedes Netzwerk umfasst eine Eingabe- und Ausgabeschicht. Die dazwischenliegenden Schichten werden als *Hidden Layers* bezeichnet. Die Abbildung 2.10 zeigt einen solchen Aufbau.

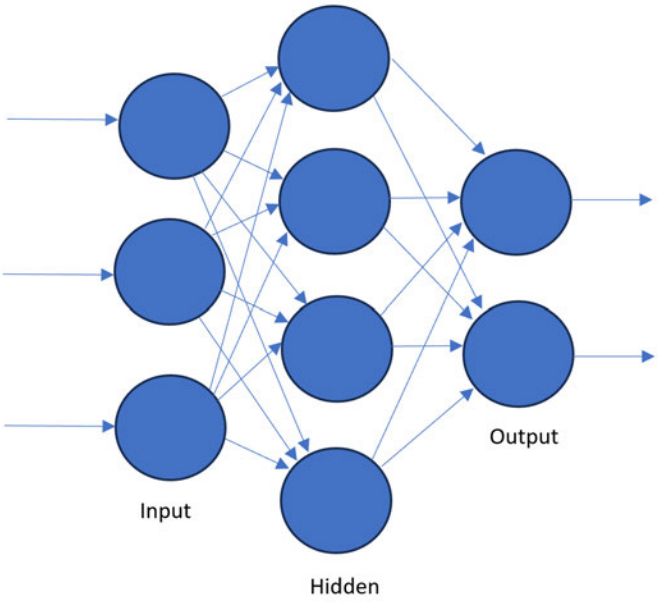


Abbildung 2.10: Aufbau neuronales Netz

Jedes Neuron außerhalb der Eingabeschicht erhält Eingaben von vorherigen Neuronen. Wie das Neuron auf das Eingangssignal $z = w \cdot x + b$ reagiert, wird durch die sogenannte Aktivierungsfunktion beschrieben. Das b steht für den sogenannten Bias, einen Verschiebungsterm, der beispielsweise verwendet wird, um sicherzustellen, dass ein Netzwerk aktiviert wird, auch wenn die Eingaben null sind. Der Bias ist der negative Schwellwert $b = -\theta$. Statt eines Schwellwertes wird nun eine stetig differenzierbare Aktivierungsfunk-

tion benötigt [18]. Mit der Aktivierungsfunktion definiert sich der Ausgang wie folgt.

$$O = f(w \cdot x + b) \text{ mit } f(z) \begin{cases} 0 & : z \leq 0 \\ 1 & : z > 0 \end{cases} \tag{2.23}$$

Im nächsten Abschnitt 2.3.2 werden die geläufigsten Aktivierungsfunktionen vorgestellt.

2.3.2 Aktivierungsfunktion

Wie bereits erwähnt, bestimmt die Aktivierungsfunktion die Reaktion des Neurons auf das Eingangssignal. Durch solche Funktionen ist es dem Netz möglich, komplexe Muster in den Daten zu erlernen. Heutzutage stehen zahlreiche Aktivierungsfunktionen zur Verfügung. Im Folgenden werden die wichtigsten Funktionen erläutert.

Sigmoid

Eine häufig verwendete Funktion ist die stetig differenzierbare Sigmoid Funktion. Die Ausgangswerte dieser Funktion liegen zwischen 0 und 1. Die Sigmoidfunktion zeigt eine Sättigung, wenn ihr Argument deutlich positiv oder negativ ist. In solchen Fällen flacht die Funktion stark ab, und geringfügige Änderungen der Eingabewerte haben nur noch geringe Auswirkungen auf die Funktionswerte [7]. Abbildung 2.11 zeigt den Verlauf der Funktion.

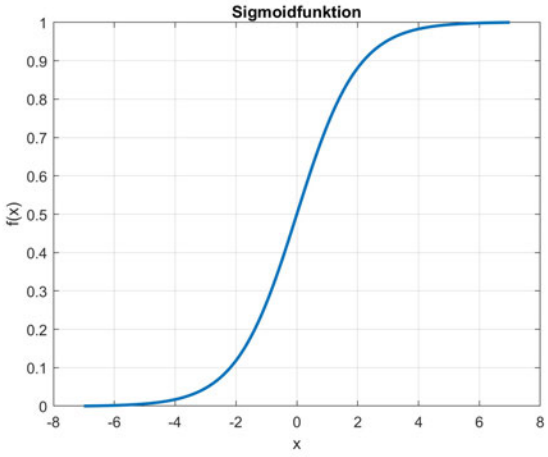


Abbildung 2.11: Sigmoid Funktion

Die Sigmoid Funktion wird wie folgt definiert.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.24}$$

Ein Nachteil der Funktion ist das Auftreten des Problems der verschwindenden Gradienten. Bei diesem Phänomen führt ein sehr kleiner oder sehr großer Wert zu extrem kleinen Ableitungen, da die Ableitung von sehr großen oder sehr kleinen Werten selbst sehr klein wird. Aufgrund dessen gilt die Sigmoid-Funktion als ungeeignet für sehr tief gestaffelte Netzwerke. Dies bezieht sich allerdings nur auf die ersten und inneren Schichten, nicht auf die Ausgabeschicht.

Tangens Hyperbolicus

Die Tangens Hyperbolicus Funktion beschreibt ebenfalls eine S-Kurve, wobei die Ausgangswerte jedoch im Bereich von -1 bis 1 liegen. Da der Tangens Hyperbolicus symmetrisch um den Ursprung ist, ist er für negative und positive Eingabewerte gleichermaßen empfindlich [6].

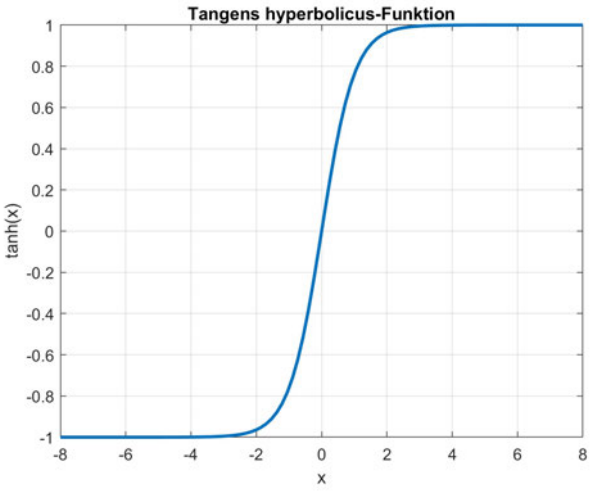


Abbildung 2.12: Tanh Funktion

Abbildung 2.12 zeigt den Verlauf der Funktion. Das Problem der Verschwindenden Gradienten, kann durch den erweiterten Wertebereich zwar gemildert werden aber bleibt

aber auch hier bestehen.

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.25}$$

Rectified Linear Unit

Die Rectified Linear Unit (ReLU) Funktion ist eine beliebte Funktion im Bereich Deep Learning. Ihr großer Vorteil liegt darin, dass sie das Problem der Sättigung bei positiven Werten nicht aufweist. Zudem ist ihre Berechnungsgeschwindigkeit schneller im Vergleich zu der Sigmoid- und Tangens Hyperbolicus-Funktion. Allerdings kann das Problem der sterbenden Gradienten auftreten. Dies tritt auf, wenn die Eingangswerte negativ sind, wodurch die Funktion inaktiv wird und den Wert null zurückgibt. Dies kann insbesondere während des Backpropagation-Prozesses zu Problemen führen [6].

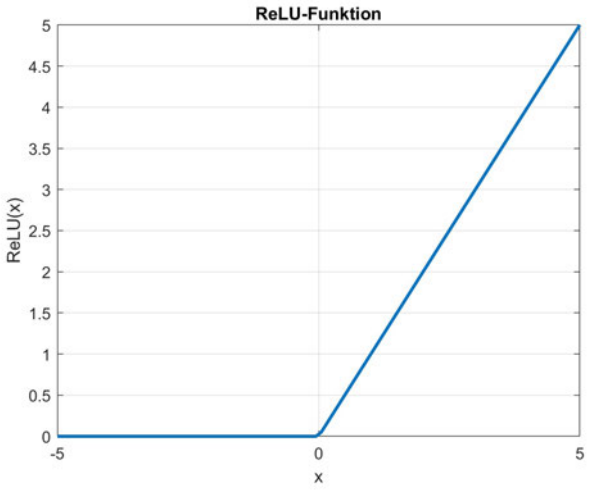


Abbildung 2.13: ReLU Funktion

In Abbildung 2.13 ist der typische Verlauf eines Gleichrichters zu sehen. Die Funktion wird wie folgt definiert.

$$\sigma(x) = \begin{cases} \max(0, x) & : z \geq 0 \\ 0 & : z < 0 \end{cases} \tag{2.26}$$

Leaky Rectified Linear Unit

Die Leaky-ReLU Funktion wurde entwickelt, um das Problem der sterbenden Gradienten zu kompensieren. Bei negativen Eingängen gibt die Funktion kleine, aber nicht null x-

Werte zurück, was dazu beiträgt, das Verschwinden von Gradienten zu verhindern und somit das Lernen in tiefen neuronalen Netzwerken zu erleichtern [18]. Abbildung 2.14 zeigt den Verlauf der Funktion mit einen a von 0,05.

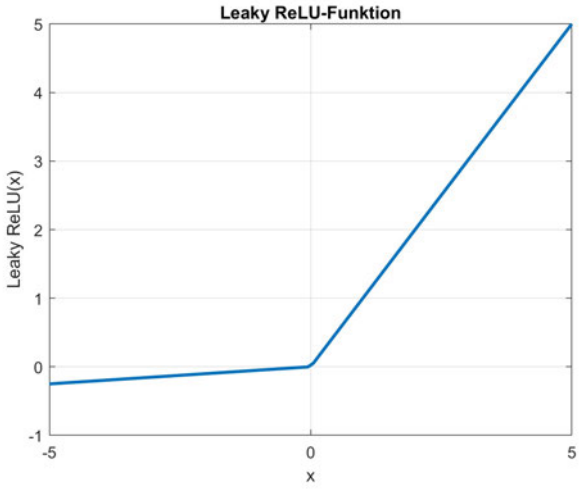


Abbildung 2.14: Leaky ReLU Funktion

Abbildung 2.14 zeigt den Verlauf der Leaky ReLU Funktion. Der Verlauf in positiver Richtung ist identisch der ReLU-Funktion. Allerdings fällt die Funktion in negative Richtung durch den Parameter a linear ab. Folgende Formel beschreibt die Funktion.

$$f(x) = \begin{cases} x & : x > 0 \\ a \cdot x & : x \leq 0 \end{cases} \tag{2.27}$$

Exponential Linear Units

Die Exponential Linear Unit (ELU) Funktion löst ebenfalls das Problem der negativen Eingänge. Im Gegensatz zur Leaky Relu Funktion ist die ELU-Funktion im negativen Bereich nicht linear, was zu einer etwas langsameren Berechnung führt.

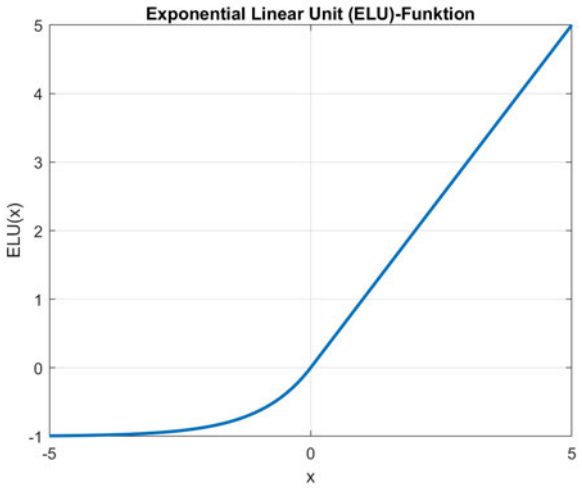


Abbildung 2.15: Elu Funktion

In Abbildung 2.15 ist der Einfluss des e-Terms sichtbar, die Formel dazu lautet wie folgt [1].

$$f(x) = \begin{cases} x & : x \geq 0 \\ a(e^x - 1) & : x < 0 \end{cases} \tag{2.28}$$

Softmax Funktion

Die Softmax-Funktion wird bei Klassifizierungsproblemen mit mehreren Klassen eingesetzt. Sie erhält als Eingabe einen Vektor, dessen einzelne Werte mithilfe der Funktion in Wahrscheinlichkeiten umgewandelt werden. Wenn der numerische Wert des Vektors hoch ist, ist auch die Wahrscheinlichkeit hoch. Die folgende Formel beschreibt die Funktion. Wobei p_i die Wahrscheinlichkeit des aktuellen Labels ist.

$$p_i(x) = \frac{e^{x_i}}{\sum_{k=0}^{K} e_k^x} \tag{2.29}$$

Durch ein Beispiel wird die Funktionsweise der Funktion verdeutlicht. Der Vektor $\mathbf{x}$ stellt den Labelvektor mit den Klassen 1,2 und 3 dar.

$$\vec{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \rightarrow p(\vec{x}) = \begin{pmatrix} \frac{1}{5} \\ \frac{2}{5} \\ \frac{3}{5} \end{pmatrix} = \begin{pmatrix} 0,166 \\ 0,33 \\ 0,5 \end{pmatrix}$$

Für die drei Klassen wird durch die Softmax Funktion die Wahrscheinlichkeit der Labels berechnet. In diesem Fall würde das Ergebnis bedeuten, dass es sich bei der vorhergesagten Klasse wahrscheinlich um die 3 handelt, da dort die Wahrscheinlichkeit am höchsten ist. Die Softmax Funktion wird häufig bei Mehrklassenproblemen eingesetzt. Ein wichtiger Aspekt ist, dass die Summe der Wahrscheinlichkeiten aller Klassen stets 1 ergibt [6].

2.3.3 Training

Das Training, der Lernprozess eines neuronalen Netzes, kann auf verschiedene Arten erfolgen. In dieser Arbeit wird das überwachte Lernen angewendet. Beim Training lernt das Netz aus den Eingabevektoren, indem es die Gewichte anpasst und die Fehlerrate reduziert. Dabei erhält das Netz als Input Trainingsdaten und die entsprechenden Labels. Labels dienen als Klassifizierungsmerkmal; zum Beispiel könnten Hunde das Label 1 und Katzen das Label 2 erhalten.

Das Ziel besteht darin, die Gewichte so anzupassen, dass das Netz in der Lage ist, eine Funktion zu approximieren, die den Eingangsvektor auf die Labels abbildet. Hierfür existieren verschiedene Ansätze. Bei der Vorwärtspropagation trifft das Netz für einen Eingabevektor eine Vorhersage und gibt einen entsprechenden Vorhersagevektor aus. In der Backpropagation werden die Gewichte basierend auf der Fehlerquote der vorherigen Iteration angepasst. Backpropagation ist eine gängige Methode in modernen neuronalen Netzen.

Das Training erfolgt in sogenannten Epochen. In jeder Epoche werden die Trainingsdaten dem Netzwerk präsentiert, um das Netzwerk zur Generalisierung zu ermutigen, d.h. es soll in der Lage sein, auf neue Daten Vorhersagen zu treffen. Am Ende jeder Epoche wird der Algorithmus mit Validierungsdaten getestet und in der folgenden Epoche verbessert. Das übergeordnete Ziel des Trainings besteht darin, die Verlustfunktion zu minimieren [18].

2.3.4 Verlustfunktion

Die Verlustfunktion, auch als Kostenfunktion bezeichnet, dient als Maß dafür, wie gut das Netz eine bestimmte Aufgabe erfüllt. Typischerweise handelt es sich dabei um eine Regression oder Klassifizierung. Die Funktion beschreibt die Diskrepanz zwischen den

Vorhersagen des Modells und den tatsächlichen Zielwerten. Während des Trainings muss die Verlustfunktion kontinuierlich verringert werden, um das Netz zu verbessern [2].

Mittlere Quadratische Fehler (MSE)

In Regressionsaufgaben wird häufig der Mean Squared Error (MSE) verwendet. Diese Funktion berechnet die quadrierten Unterschiede zwischen den Elementen des vorhergesagten Vektors $\mathbf{y}$ und dem gegebenen Label-Vektor $\hat{\mathbf{y}}$ und summiert diese auf [7].

$$MSE = \frac{1}{m}\sum_{i}(y_i - \hat{y_i})^2 \tag{2.30}$$

Kreuzentropie

In Klassifizierungsproblemen wird häufig die Kreuzentropie (Cross Entropy) als Verlustfunktion verwendet. Ähnlich wie beim MSE wird auch hier der Verlust zwischen dem Vorhersagevektor $\mathbf{y}$ und dem Label-Vektor $\hat{\mathbf{y}}$ berechnet.

$$L = -\frac{1}{N}\sum_{i=1}^{n}\sum_{j=1}^{k}\hat{y_{ij}} \cdot log(y_{ij}) \tag{2.31}$$

Durch den Logarithmus Term fließen Fehler umso stärker ein, desto kleiner der Wahrscheinlichkeitswert ist. Fehler haben somit eine größere Einwirkung auf die Verlustfunktion als bei der MSE [2]. Oft wird in diesem Zusammenhang vom One-Hot-Encoding gesprochen, dabei können die Werte nur 0 oder 1 betragen. Bei Tensorflow gibt es beispielsweise zwei Kreuzentropie Funktionen, eine fordert numerische Labelwerte die andere fordert Labelwerte im One-Hot-Encoding Format. Ein Vektor des Typs $\begin{pmatrix} Hund \\ Katze \\ Pferd \end{pmatrix}$ ergibt sich im One-Hot-Encoding Format wie folgt.

Typ	Hund One hot	Katze One hot	Pferd One hot
Hund	1	0	0
Katze	0	1	0
Pferd	0	0	1

Tabelle 2.1: One-Hot-Encoding

2.3.5 Optimierer

Optimierer sind Algorithmen oder Methoden, die dabei helfen, die Verlustfunktion zu minimieren und die Leistung des Modells zu verbessern [7]. Die Wahl des geeigneten Optimierers erfordert oft Experimentieren, da sie einen wesentlichen Beitrag zur Konvergenz des Modells und zur Steigerung seiner Leistungsfähigkeit leistet. Es existieren verschiedene Optimierer, die unterschiedliche Optimierungsalgorithmen verwenden.

Stochastischer Gradienten Abstieg

Das Stochastic Gradient Descent (SGD)-Verfahren berechnet den Gradienten der Verlustfunktion anhand eines zufälligen Teils des Datensatzes. Dieser Algorithmus eignet sich besonders gut für große Datenmengen. Die schrittweise Annäherung an den steilsten Abstieg zielt darauf ab, die Funktion zu minimieren. Bei einer Verlustfunktion $S(\theta)$ wird der Parameter θ wie folgt berechnet.

$$\theta_{new} = \theta_{old} - \eta \cdot \nabla S(\theta_{old}) \tag{2.32}$$

Der Parameter η stellt die Lernrate dar und bestimmt die Schrittweite des Algorithmus. Es ist entscheidend, einen angemessenen Wert für die Lernrate zu wählen, da eine zu kleine Lernrate dazu führen kann, dass die Funktion schnell in einem lokalen Minimum stecken bleibt. Durch eine zu hohe Lernrate besteht hingegen die Gefahr, Minima unbeabsichtigt zu übersehen. Andererseits kann eine zu niedrige Lernrate zu einem hohen Rechenaufwand führen [7]. Die Abbildung 2.16 verdeutlicht die Bedeutung der richtigen Wahl der Lernrate.

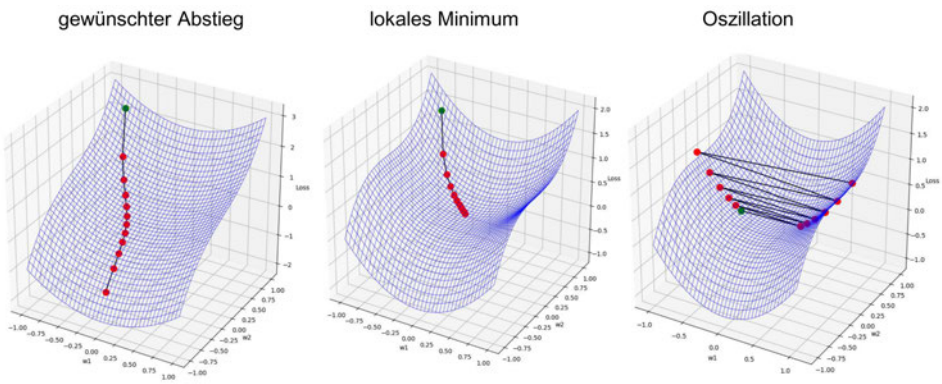


Abbildung 2.16: SGD Optimierung [3]

Beim SGD-Verfahren ist die Wahrscheinlichkeit, lokale Minima nicht zu überwinden, relativ hoch. Daher wurden einige Verbesserungen dieses Algorithmus entwickelt.

AdaGrad

Beim Adaptive Gradient Descent (AdaGrad) wird die Lernrate durch antiproportionale Skalierung zur Quadratwurzel der Summe aller historischen quadrierten Werte des Gradienten angepasst. Für einen Parameter w im Schritt t bedeutet dies mathematisch:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot \nabla w_t \tag{2.33}$$

mit der akkumulierte Summe der quadrierten Gradienten v_t:

$$v_t = v_{t-1} + (\nabla w_t)^2 \tag{2.34}$$

Die Hauptidee hinter AdaGrad ist, dass häufig aktualisierte Parameter kleine Update-Schritte erhalten, während selten aktualisierte Parameter größere Schritte erhalten. Allerdings gibt es ein Problem bei diesem Verfahren: Wenn das Training zu lange dauert, wird der Gradient durch eine sehr große Zahl geteilt, was zu äußerst kleinen Aktualisierungsschritten der Gewichte führt. Je länger das Training dauert, desto kleiner wird dieser Schritt. Im schlimmsten Fall könnte dies dazu führen, dass das Training endlos weitergeht [7].

RMSProp

Das Root Mean Square Propagation (RMSprop) soll das Problem von AdaGrad lösen, indem es eine gleitende Durchschnittsgröße für die akkumulierten Gradienten verwendet. Die akkumulierte Summe wird wie folgt berechnet.

$$v_t = \beta v_{t-1} + (1 - \beta) \cdot \nabla w_t^2 \tag{2.35}$$

Der Parameter w_t wird gemäß Formel 2.33 berechnet. Der Hyperparameter β steuert die exponentielle Abnahme der Gewichtung der vergangenen Gradienten und liegt normalerweise nahe bei 1.

Adam

Adaptive Moment Estimation (Adam) ist ein äußerst leistungsstarker Algorithmus, der die Konzepte von RMSProp und dem SGD mit Momentum kombiniert. Der Parameter w im Schritt t wird wie folgt berechnet.

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \tag{2.36}$$

Das beinhaltet den Durchschnitt der quadrierten Gradienten v_t, dem leitende Durchschnitt der Gradienten m_t, die Bias-korrigierten Schätzungen der Momente $\hat{v}_t$ und $\hat{m}_t$ und den Exponentialabnahmefaktoren der Momente β_1 und β_2.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla w_t^2 \tag{2.37}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla w_t \tag{2.38}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.39}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.40}$$

Durch die Verwendung des gleitenden Durchschnitts der quadrierten Gradienten des RMSProp und des gleitenden Durchschnitts der Gradienten (Momentum) ist Adam ein robuster und weit verbreiteter Algorithmus. Eine zusätzliche Stärke des Algorithmus liegt in der Bias-korrigierten Schätzung der Momente, die dazu beiträgt, die Anfangsphasen des Trainings zu verbessern, insbesondere wenn die Momente initialisiert werden und niedrige Schätzungen aufweisen könnten [7].

2.3.6 Qualitätsmaß

Das Qualitätsmaß dient dazu, die Modellleistung zu bewerten und zu optimieren. Die Genauigkeit (Accuracy) wird definiert als der Prozentsatz der korrekten Vorhersagen T im Verhältnis zur Gesamtanzahl der Instanzen n.

$$A_c = \frac{T}{n} \tag{2.41}$$

Statt der Genauigkeit kann auch der Fehler L zu Beurteilung dienen. Dieser ergibt sich zu:

$$L = 1 - A_c \tag{2.42}$$

Weitere Metriken sind der Recall R und die Präzision S, welche oft bei binären Problemen angewendet werden. Die Präzision gibt an, wie viele der als positiv vorhergesagten Instanzen tatsächlich positiv sind, während der Recall angibt, wie viele der tatsächlich positiven Instanzen korrekt erkannt wurden. Da die Präzision und der Recall oft widersprechende Ziele sind, wird meistens eine Kombination beider Kennzahlen verwendet, die als F1-Score bezeichnet wird [3].

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{S}} \tag{2.43}$$

2.3.7 Convolutional Neural Network

CNN sind eine spezielle Form neuronaler Netze. Diese Art von Netz wird häufig bei Bilddaten oder anderen zweidimensionalen Daten genutzt. CNNs nutzen die mathematische Funktion der Faltung. Für eine Eingabematrix I und einen Filterkern (Kernel) K definiert sich die Feature-Map S wie folgt.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.44}$$

Durch die Faltung der Eingabedaten mit Filtern werden Muster und Merkmale erkannt. So soll jedes Filter lernen bestimmte Merkmale zu erkennen und herauszufiltern, beispielsweise Kanten oder Texturdetails. Nach einer Faltungsschicht folgt in den meisten Fällen ein sogenannter Pooling-Layer. Bei dieser Schicht wird die Dimension verringert, indem sich ein Bereich angeschaut wird und nur das wichtigste Merkmal behalten wird. Beim Max-Pooling Layer ist dies beispielsweise der größte Wert. Abbildung 2.17 zeigt ein neuronales Netz mit Faltungsschichten.

Nach den abwechselnden Faltungs- und Poolingschichten erfolgt meistens ein Fully Connected Layer. Vor dem letzten Fully Connected Layer, welcher als Ausgang die Klassen hat, muss ein Flatten Layer eingefügt werden. Dieser Layer formt die multidimensionalen Daten, die aus den vorherigen Faltungsschichten stammen, in einen einzigen Vektor oder eine eindimensionale Struktur um [7].

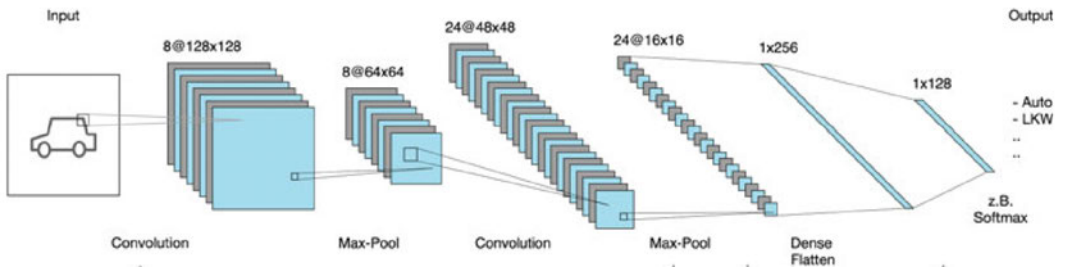


Abbildung 2.17: Convolutional Neural Network [2]

2.3.8 Über- und Unteranpassung

Hauptziel des Machine Learnings ist, dass der Algorithmus gut auf neuen unbekannten Daten funktioniert. Diese Fähigkeit wird als Generalisierung bezeichnet. Ein Algorithmus kann noch so gut auf den Trainingsdaten funktionieren, tut er dies nicht auf den Validierungsdaten kann der Algorithmus schlecht generalisieren. Deswegen ist ein Ziel während des Trainings den Trainings- und Generalisierungsfehler (auch Testfehler genannt) möglichst klein zuhalten. Ist der Abstand der beiden zu groß, wird von Überanpassung (Overfitting) gesprochen. Man könnte sagen, dass das Netz auswendig gelernt hat, da der Trainingsfehler oft niedrig ist, aber der Generalisierungfehler deutlich höher. Das Gegenteil tritt bei der Unteranpassung (Underfitting) auf, in diesem Fall ist die Modellkapazität zu klein. Dies ist häufig der Fall, wenn nicht genug Trainingsdaten zur Verfügung stehen [7].

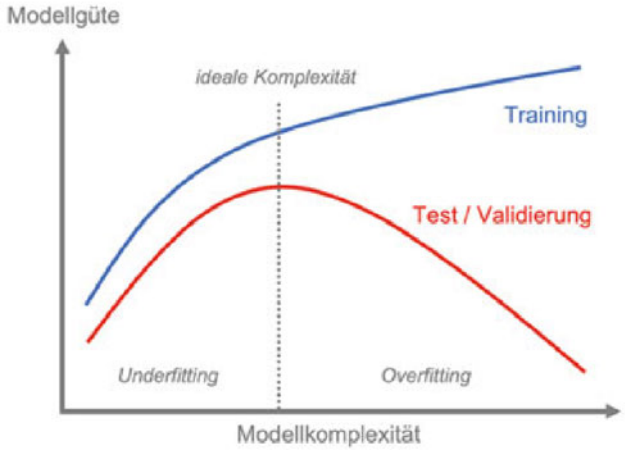


Abbildung 2.18: Über- und Unteranpassung [2]

Die Trainingsdaten und der dazugehörigen Trainingsfehler und Trainingsgenauigkeit bezieht sich auf die Trainingsdaten. Von den Trainingsdaten wird ein kleiner Teil Daten abgespalten, die Validierungsdaten, welche nicht zum Trainieren verwendet werden. Nach jeder Trainingsepoche wird der Algorithmus auf diesen unbekannten Daten getestet. Dazu gehören der Generalisierungsfehler und die Validierungsgenauigkeit. Außerdem gibt es noch einen Testdatensatz, dieser Datensatz wird erst nach Abschluss des Trainings und nach Anpassung der Hyperparameter genutzt. Die Testgenauigkeit und der Testfehler beziehen sich auf diesen einmalig getesteten Datensatz. Die Testgenauigkeit oder der Testfehler entscheiden letztendlich wie gut das angepasste Modell auf völlig unbekannten Daten funktioniert.

Abbildung 2.19: Aufteilung Daten

3 Stand der Technik

Deep Learning ist heutzutage ein weithin diskutiertes Thema mit zahlreichen Anwendungsbereichen und Forschungsarbeiten in diesem Bereich. Die Erkennung von Widerständen ist ebenfalls Gegenstand der Forschung, wobei verschiedene Ansätze verfolgt werden können. Beispielsweise klassifizierte Elias Soud in seiner Arbeit [19] Widerstände mithilfe von Deep Learning. Allerdings besteht Verbesserungspotenzial, da sein Ansatz die Widerstandsbilder ohne Vorverarbeitung und Objektdetektion verwendet.

Eine andere interessante Herangehensweise wurde von Borna Houmani-Farahani auf GitHub [9] entwickelt. Dort wird ein Haar-Classifier trainiert, um Merkmale von Widerständen zu erkennen. In jedem Bereich, in dem ein Widerstand erkannt wird, wird ein digitaler Zoom angewendet. Anschließend erfolgt die Verwendung eines adaptiven Schwellwerts, um den Hintergrund vom Widerstandskörper zu separieren. Nach der Konvertierung des Bildes in den HSV-Farbraum wird das Farbbild nach vordefinierten Farbbereichen durchsucht. Es wird eine Maske für jede Farbe erstellt, die mit dem adaptiven Schwellwert verknüpft ist. Die Gültigkeit des Farbbandes kann durch die Anwendung der resultierenden Maske bestimmt werden. Schließlich wird die Reihenfolge der Farbbänder erkannt, um den Widerstandswert zu berechnen [9].

Des Weiteren gibt es zahlreiche wissenschaftliche Arbeiten zur Segmentierung und Klassifikation von Widerständen. Im Folgenden werden einige besonders interessante und nützliche Arbeiten näher beschrieben.

3.1 Arbeit 1: Automated Resistor Classification

Bei der Arbeit „Automated Resistor Classification“ handelt es sich um eine Gruppenarbeit [17], in dem ein Algorithmus für die automatisierte Klassifizierung von Widerständen mithilfe der Bilderkennung entwickelt wurde. Dieser Algorithmus wurde in einer Android-App mit Java implementiert. Der Prozess beginnt damit, den Widerstand auf

dem erfassten Bild zu lokalisieren und dann einen Ausschnitt davon zu erstellen. Nachdem der Widerstand ausgeschnitten wurde, erfolgt die Trennung der Farbringe vom Hintergrund, wobei sie spezifischen Farben zugeordnet werden. Am Ende ist es möglich, den Widerstandswert zu ermitteln.

Der Algorithmus ist darauf ausgelegt, ein Bild mit einem oder mehreren Widerständen vor einem einheitlich grauen Hintergrund zu verarbeiten. Er besteht aus drei aufeinanderfolgenden Stufen. In der ersten Stufe wird das Bild auf eine vordefinierte Größe skaliert. Anschließend erfolgt die Lokalisierung, Rotation und Ausschnitt des Widerstands. In der zweiten Stufe werden die Farbringe identifiziert, und es wird eine Liste von Farbwerten zurückgegeben. Die letzte, dritte Stufe hat die Aufgabe, den Widerstand zu klassifizieren [17]. Für die vorliegende Arbeit ist insbesondere die erste Stufe von Interesse, da sie wertvolle Ansätze für die Bildverarbeitung der Widerstände bietet. Die Identifizierung der Widerstandswerte und die damit verbundene Erkennung der Farbringe sollen in dieser Arbeit von einem neuronalen Netz übernommen werden und erfordern keine zusätzliche Bildverarbeitung, wie sie in der beschriebenen Gruppenarbeit durchgeführt wird.

3.1.1 Resistor Localization

Die Lokalisierung des Widerstands wird mithilfe eines Zeilenerkennungsalgorithmus und einer statistischen Analyse des Bildes gelöst [17].

Um die Ergebnisse der Zeilenerkennung zu verbessern, werden verschiedene Rauschfilter auf das von der Kamera aufgenommene Bild angewendet. Zunächst werden ein Medianfilter und ein Gauß'scher Weichzeichner verwendet, um Kameraartefakte zu eliminieren und das Bildrauschen signifikant zu reduzieren.

Um die Kanten des Widerstands zu finden wird ein Kantenerkennungsalgorithmus verwendet. Für den Kantenerkennungsalgorithmus wird das Bild in ein Graustufenbild umgewandelt. Es wird mit dem Standard-Glättungsfilter von OpenCV und der morphologischen Funktion Erosion gefiltert, wobei eine Kernelgröße von 5 verwendet wird. Danach wird der Canny-Algorithmus angewendet, wobei ein unterer Schwellenwert von 15 und ein oberer Schwellenwert von 60 verwendet werden.

Um die Linien im Binärbild zu identifizieren, wird die Hough-Transformation für Linien angewendet. Dabei wird eine Auflösung von 1 Pixel durch 180 Grad und eine Schwelle

von 80 verwendet. Die OpenCV-Implementierung der Hough-Transformation gibt Linien zurück, die durch den Winkel θ ihrer Normalen in Bezug auf die x-Achse und ihrem Abstand ρ vom Ursprung in der unteren linken Ecke des Bildes parametrisiert sind. Da viele Linien zurückgegeben werden, die mehr oder weniger parallel zum Widerstand verlaufen, werden diese in Gruppen eingeteilt.

Die Gruppierung erfolgt, indem der Winkel zwischen zwei fast parallelen Linien betrachtet wird. Hierzu wird eine leere Liste von Liniengruppen erstellt. Für jede durch die Hough-Transformation gefundene Linie wird der Winkel θ mit dem durchschnittlichen Winkel $\overline{\theta}$ jeder Liniengruppe in der Liste verglichen. Wenn die Differenz zwischen den beiden Winkeln kleiner als 0,1 Bogenmaß und die jeweilige Differenz zwischen den Abständen ρ und $\overline{\rho}$ kleiner als 100 ist, wird die Linie zur Liniengruppe hinzugefügt. Wenn die Parameter einer gefundenen Zeile nicht innerhalb dieser Bereiche liegen, wird eine neue Zeilengruppe erstellt. Am Ende dieses Prozesses werden alle Linien in sehr wenigen Zeilengruppen gruppiert [17].

In Abbildung 3.1 sind die verschieden Liniengruppen in verschiedenen Farben dargestellt.

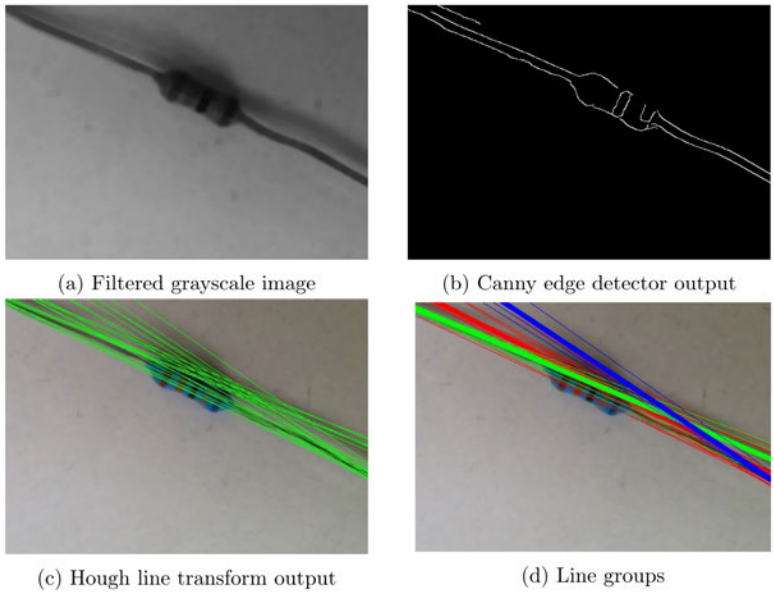


Abbildung 3.1: Diese vier Bilder veranschaulichen die einzelnen Schritte der Zeilenerkennung [17]

Nach der Hough-Transformation für Linien erfolgt die Rotation und der Zuschnitt des Bildes. Zuerst wird das Bild so gedreht, dass die Linie horizontal verläuft. Anschließend wird das Bild um die Linie herum auf einen Streifen zugeschnitten, der eine Höhe von 30 % der Breite des Originalbildes hat und als Widerstandsstreifen bezeichnet wird.

Die horizontale Position des Widerstands wird ermittelt, wobei die Tatsache genutzt wird, dass der Körper eines Widerstands der einzige farbige Teil des Widerstandsstreifens ist. Um diesen Effekt zu verstärken, wird ein Medianfilter mit einer relativ großen Blendenöffnung von 11 angewendet, und der Sättigungswert jedes Pixels wird so angepasst, dass die durchschnittliche Sättigung im gesamten Widerstandsstreifen 60 % beträgt [17].

Das Bild wird anschließend in den LAB-Farbraum konvertiert. Der LAB-Farbraum speichert die Informationen in drei Kanälen. Mithilfe der Kanäle wird die Stärke der Farbabweichung berechnet [17].

$$\sigma = k1 \cdot \sigma_L + k2 \cdot (\sigma_a + \sigma_b) \tag{3.1}$$

σ_L, σ_a und σ_b sind die Standardabweichungen der entsprechenden Farbkanäle aller Pixel. $k1$ und $k2$ sind Skalierungsfaktoren, die in der Implementierung auf 1 bzw. 10 gesetzt sind [17].

Abbildung 3.2: Jede Pixelspalte unterhalb des Widerstandsstreifens wird entsprechend der Farbabweichung der darüber liegenden Spalte eingefärbt: Je höher die Farbabweichung, desto heller die Spalte [17]

Wie in Abbildung 3.2 zu sehen ist, erfolgt die Lokalisierung des Widerstands anhand der Farbabweichung. Hierzu wird die Farbabweichung für jede Spalte berechnet. Wenn die Farbabweichung einen bestimmten Grenzwert σ_0 überschreitet, wird die entsprechende Spalte als Teil des Widerstands markiert. Falls der Abstand zwischen zwei markierten Spalten kleiner als d_0 Pixel ist, werden auch alle dazwischenliegenden Spalten als Teil des Widerstands gekennzeichnet. Dieses Verfahren wird ebenfalls für die vertikale Analyse angewandt [17].

Abbildung 3.3: Jede Pixelzeile rechts vom Widerstandsstreifen wird entsprechend der Farbabweichung der Zeile links davon eingefärbt: Je höher die Farbabweichung, desto heller die Reihe [17]

Am Ende wird der größte Widerstandsbereich ausgewählt. Wenn er mehr als h_0 Pixel hoch ist, wird er ausgeschnitten und der Liste der Widerstände hinzugefügt. In der Implementierung werden die folgenden Parameter verwendet: $\sigma_0 = 70$, $d_0 = 20$ (horizontale Analyse) oder $d_0 = 10$ (vertikale Analyse), $l_0 = h_0 = 21$. Nachdem alle Leitungen analysiert wurden, wird eine Liste von Widerstandsbildern und ihre Position im ursprünglichen Kamerabild erhalten.

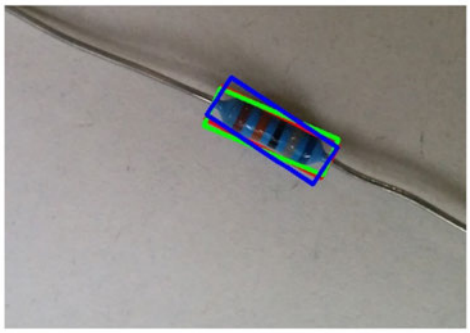


Abbildung 3.4: Der Algorithmus fand in 3.1d für jede Leitungsgruppe einen Widerstand – hier gekennzeichnet durch ein Kästchen in der jeweiligen Farbe [17]

In Abbildung 3.4 ist ein Beispielbild dargestellt, in dem der Algorithmus drei Versionen desselben Widerstands identifiziert, eine für jede Liniengruppe. Wenn sich zwei oder mehr potenzielle Widerstandsgrenzen überschneiden, wählt der Algorithmus diejenige mit der kleinsten Fläche aus und entfernt die anderen aus der Liste. Dieser Schritt soll verhindern, dass derselbe Widerstand in den folgenden rechenintensiven Schritten mehrfach analysiert wird [17].

3.1.2 Color Ring Detection

Im Anschluss wird die Gruppenarbeit [17] durch die Fortsetzung der Farbringdetektion vorangetrieben. Hierzu erfolgt die Umwandlung des Bildes in den LAB-Farbraum, welcher die Identifikation der Ränder der Farbringe ermöglicht. Anschließend wird der mittlere Farbwert eines spezifischen Spaltenbereichs in einer Reihe berechnet – dieser wird im Folgenden als Row Section Mean (RSM) bezeichnet.

Im zweiten Abschnitt der Farbring-Erkennung wird die Liste der Kanten aus dem ersten Teil genutzt, um die Farbringe zu lokalisieren. Der Algorithmus muss für jedes Widerstandsegment zwischen einem Paar aufeinanderfolgender Kanten entscheiden, ob der Abschnitt tatsächlich zu einem Farbring gehört (Actual Color Ring (ACR)) oder zum Hintergrund. Die Ergebnisse für alle Widerstandabschnitte können durch eine Bitfolge codiert werden: Eine 1 zeigt an, dass der entsprechende Abschnitt zu einem ACR gehört, während eine 0 bedeutet, dass er Teil des Hintergrunds ist. [17]. Abbildung 3.5 zeigt eine solche ACR-Hintergrund-Zuordnung.

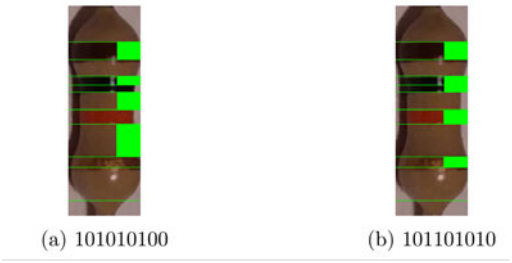


Abbildung 3.5: Die Subfigures (a) und (b) zeigen zwei verschiedene ACR-Hintergrund-Zuordnungen mit der entsprechenden Bitfolge. (b) ist die korrekte Zuordnung [17]

Ein Strafsystem (Penalty system) wird verwendet, das jede mögliche Bitfolge anhand mehrerer Kriterien bewertet. Bei der Klassifizierung vergleicht der Algorithmus die Farbmittelfarbe des ACR mit jeder Referenzfarbe [17].

Für den Verlauf dieser Arbeit haben sich interessante Ansätze für die Bildverarbeitung herausgestellt. Einige Ansätze dieser Gruppenarbeit finden sich im Kapitel 7 wieder.

3.1.3 Ergebnis

Um das entwickelte System zu testen, wurde der Algorithmus an 261 Widerständen evaluiert. Dabei ist zu beachten, dass nur die Widerstände berücksichtigt wurden, die alle vorherigen Stufen durchlaufen haben [17].

	Genauigkeit
Resistor Location	88,9 %
Edge Detection	78,4 %
Penalty system	79,1 %
Color Classification	25,0 %

Tabelle 3.1: Ergebnis Automated Resistor Classification Arbeit

In Tabelle 3.1 ist zu erkennen, dass die Widerstandslokalisierung, Kantenerkennung und das Strafsystem gute Ergebnisse erzielen, während die Farbklassifizierung hingegen weniger zufriedenstellend ausfällt.

3.2 Arbeit 2: Automatic Segmentation and Classification of Resistors in Digital Images

In der Arbeit von Mia Muminovic und Emir Sokic [13] wird eine Methode vorgestellt, die sich mit der Detektion, Segmentierung und Klassifizierung von Widerständen in digitalen Bildern befasst, basierend auf deren Nennwerten. Der Prozess umfasst die Schritte der Bildsegmentierung, morphologischen Bildverarbeitung, Objektdarstellung und -beschreibung, Merkmalsextraktion sowie der Klassifizierung der extrahierten Daten unter Einsatz von Support-Vektor-Maschinen (SVM) [13]. Abbildung 3.6 zeigt die Algorithmusschritte.

Im Prozess der Bildsegmentierung werden spezifische Elemente des Bildes isoliert, indem das Bild in Segmente unterteilt wird. Zur Segmentierung kommen Methoden wie Schwellenwertbildung, kantenbasierte und regionsbasierte Segmentierung zum Einsatz [13].

Im nächsten Schritt werden morphologische Operationen angewendet. Zuerst wird das Rauschen und die Leiterbahnen aus dem Binärbild entfernt, indem Erosion angewendet wird. Anschließend erfolgt die Dilatation, um die Objekte sichtbarer zu machen und

ihre Größe auf das Ursprüngliche zurückzuführen. In der Arbeit wurde festgestellt, dass die Größe des Kernels für die morphologischen Operatoren anhand der Nenngröße des Widerstands, der Kameradistanz von der analysierten Ebene und der Auflösung des Kamera-Sensors ausgewählt werden sollte. Sobald die Region die Kontur des Objekts gefunden hat, ist es möglich, Merkmale wie den Schwerpunkt, die Fläche, die Ausrichtung und andere ähnliche Eigenschaften der Objektform zu bestimmen [13].

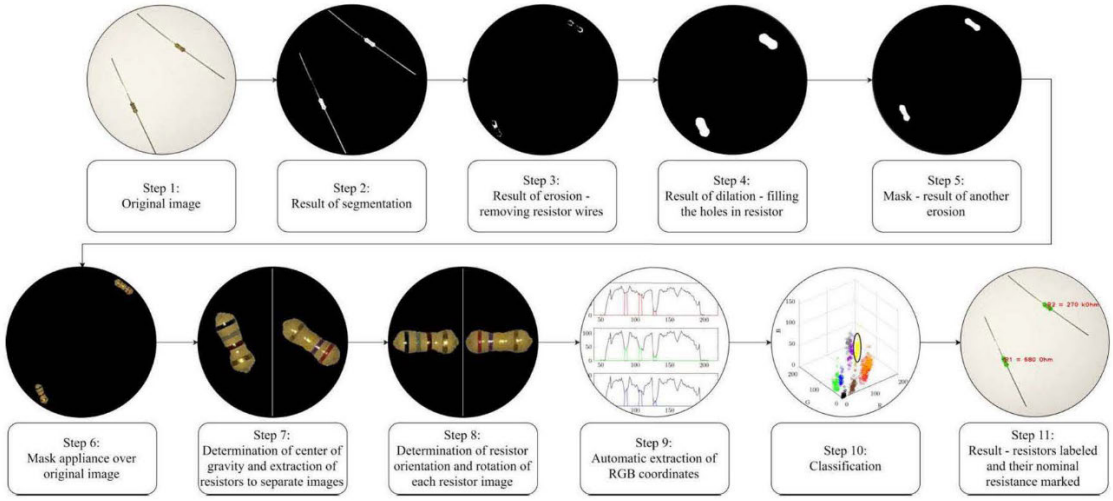


Abbildung 3.6: Algorithmusschritte zur Bestimmung des Nennwiderstands von Widerständen [13]

Mithilfe von Momenten können zwei wichtige Merkmale des gefundenen Objekts berechnet werden. Der Schwerpunkt repräsentiert das Zentrum des Objekts und bestimmt die Position des Widerstands innerhalb eines Bildes. Der zweite Deskriptor ist die Achse mit dem geringsten Trägheitsmoment (ALI), welche als Linie definiert ist. Sie minimiert das Integral der Quadrate der Abstände zu den Punkten des Formrands. Diese wird genutzt, um die Rotation des Widerstands im Bild zu erfassen und gegebenenfalls zu korrigieren [13].

Nach der korrekten Rotation des Bildes sollen die Farbringe identifiziert werden. Aufgrund geringfügiger Unterschiede in den Ringpositionen wurden statistische Analysen der Linienpositionen auf den Widerständen durchgeführt. Basierend auf den erzielten Ergebnissen genügt es, die Endpunkte des Widerstands zu bestimmen, um die Bandpositionen festzulegen. Die statistische Analyse ergab, dass die Breite des Widerstandsbands (Rings) im Durchschnitt etwa 5 % der Gesamtlänge des Widerstands beträgt [13].

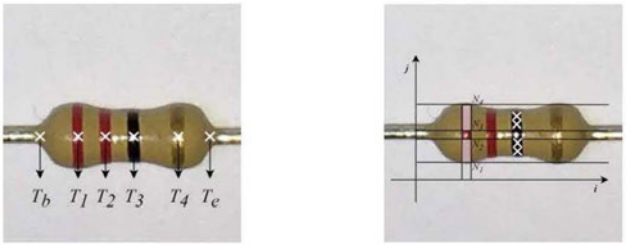


Abbildung 3.7: Manuelle Merkmalsextraktion aus dem Widerstandsbild, a) Extraktion von Pixeln zur Durchführung der Linienpositionsstatistik, b) Extraktion von RGB-Werten für schwarze Farbe [13]

Mithilfe folgender Formel kann der Farbdeskriptor berechnet werden.

$$RGB_{band-k} = \frac{\sum f(i,j)}{2\epsilon[(N_4 - N_3) + (N_2 - N_1)]} \tag{3.2}$$

Wobei $k \in 1, 2, 3, 4$ die Nummer des Bandes ist und ϵ die halbe Breite des Bandes in Pixeln. Die Funktion $f(i,j) = [R_{ij}, G_{ij}, B_{ij}]$ gibt den RGB-Tripletwert des Pixels an der Stelle (i,j) zurück. Für i gilt die Vorschrift $i \in (T_k - \epsilon, T_k + \epsilon)$ und für j die Vorschrift $j \in (N_1, N_2) \cup (N_3, N_4)$ [13].

Für die Erstellung eines Trainingsdatensatzes zur Klassifikation wurde eine Datenbank mit 60 Bildern unterschiedlicher Widerstände erstellt, die insgesamt 180 Farbbänder umfassten.

Für die Klassifizierung wird eine Support Vector Maschine (SVM) verwendet. Klassifikatoren wurden für jede der zehn Farben (schwarz, braun, rot, orange, gelb, grün, blau, lila, grau und weiß) sowie für die Gehäusefarbe des Widerstands (hellbraun) erstellt. Durch die Nutzung dieser Klassifikatoren zusammen mit der Formel 3.2 können die Farbringe identifiziert und der Widerstandswert berechnet werden [13].

3.2.1 Ergebnis

Folgendes Ergebnis wurde in der Arbeit beschrieben.

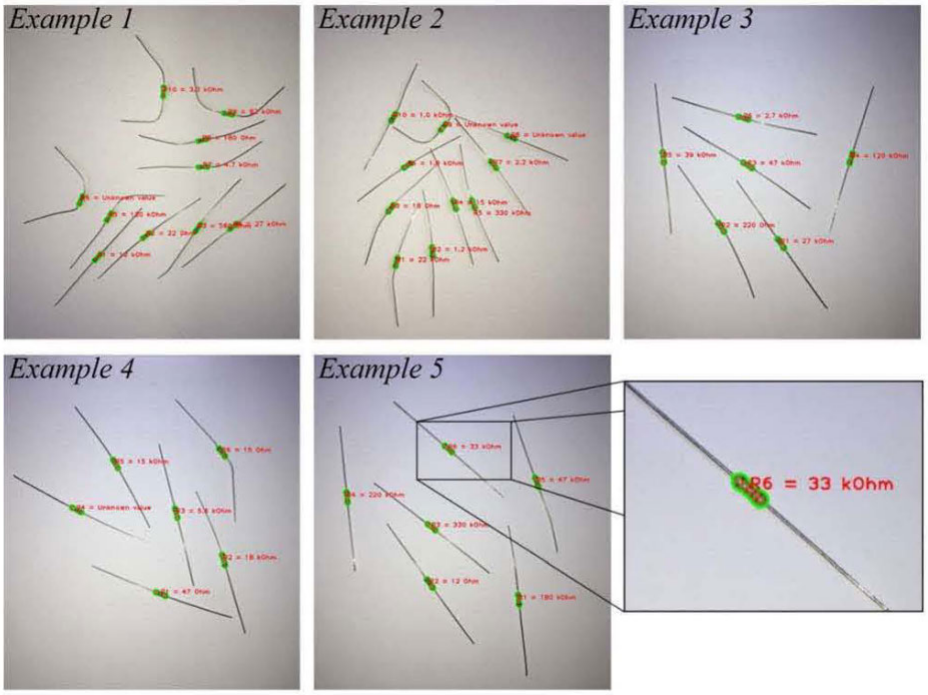


Abbildung 3.8: Experimentelle Ergebnisse - fünf Bilder mit insgesamt 38 Widerständen [13]

Abbildung 3.8 zeigt die Klassifizierung der Widerstände in fünf verschiedenen Beispielen. Die Tabelle 3.2 zeigt die Ergebnisse der Klassifizierung.

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	Genauigkeit
Example 1	✓	✓	✓	✓	✓	x	✓	x	✓	✓	80 %
Example 2	✓	✓	✓	✓	✓	✓	✓	x	x	✓	80 %
Example 3	✓	✓	✓	✓	✓	✓					100 %
Example 4	✓	✓	✓	x	✓	✓					83,33 %
Example 5	✓	✓	✓	✓	✓	✓					100 %

Tabelle 3.2: Klassifikationsergebnisse für Beispiele in Abb.3.8

3.3 Arbeit 3: An evaluation of classifiers for reading resistor colors

Die Arbeit von Yoshihiro Mitani, Wataru Yoshimura und Yoshihiko Hamamoto [12] präsentiert eine Bewertung von Klassifikatoren zur Identifizierung von Widerstandsfarben in einem RGB-Farbraum unter unterschiedlichen Beleuchtungsbedingungen. Die Studie untersucht acht verschiedene Klassifikatoren: k Nearest Neighbor (k-NN) (k=1, 3 und 5), Decision Tree (DT), SVM, Gaußian Naive Bayes (NB), Artificial Neural Network (ANN) und Random Forest (RF). Diese Klassifikatoren werden verwendet, um Widerstände in verschiedenen Beleuchtungsszenarien zu klassifizieren. In dieser Arbeit liegt der Fokus darauf zu untersuchen, welche Klassifikatoren im maschinellen Lernen effektiv in der Lage sind, Farben zu klassifizieren. In dieser Untersuchung wurde aus Gründen der Einfachheit der RGB-Farbraum als Merkmal für die Klassifizierung von Widerstandsfarben verwendet. Dadurch ergibt sich eine Gesamtanzahl von drei verwendeten Funktionen. Abbildung 3.9 zeigt Bilder von Widerstandswerten für 11 verschiedene Farben von Widerständen unter verschiedenen Beleuchtungssituationen. [12].

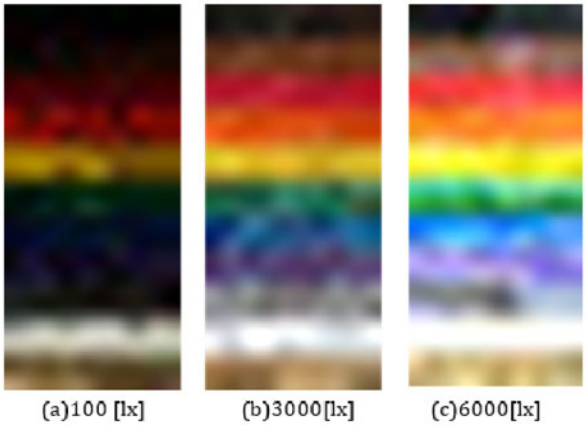


Abbildung 3.9: Bilder von Widerstandswerten für 11 verschiedene Farben von Widerständen unter verschiedenen Beleuchtungssituationen. [12]

3.3.1 KNN

In der Studie wird ein KNN mit Eingabe-, Hidden- und Ausgabeschicht eingesetzt. Die Anzahl der Neuronen in der Eingabeschicht entspricht der Anzahl der Merkmale, wobei der RGB-Merkmalsraum verwendet wird, wodurch die Anzahl der Merkmale 3 beträgt.

Die Ausgabeschicht besteht aus 11 Neuronen aufgrund der 11 Klassen (11 Farben). Für die Generalisierungsfähigkeit des KNN ist die Anzahl der Neuronen in der versteckten Schicht entscheidend. Zur Vereinfachung besteht die versteckte Schicht aus 64 Neuronen. Das KNN wird als 3-64-11 Netzwerk beschrieben, wobei die Verbindungen zwischen den Schichten vollständig vernetzt sind. Außerhalb der Ausgabeschicht werden ReLU-Funktionen als Aktivierungsfunktionen verwendet, während für die Ausgabeschicht die Softmax-Funktion eingesetzt wird. Der Adam-Optimierer wird für das Training des KNN verwendet [12].

3.3.2 Ergebnis

Das Ziel des Experiments besteht darin, die Klassifizierungsleistung jedes Klassifikators für die Trainingsstichprobengröße pro Klasse unter verschiedenen Beleuchtungssituationen zu untersuchen. Abbildung 3.10 zeigt die Ergebnisse der durchschnittlichen Fehlerquoten in Abhängigkeit von der Anzahl der Trainingsproben pro Klasse für jeden Klassifikator unter verschiedenen Beleuchtungssituationen [12].

Unter normaler Beleuchtung (3.10 b) zeigte der ANN-Klassifikator die beste Leistung bei einer Anzahl von 10 Trainingsproben pro Klasse. Ähnliche Ergebnisse waren auch bei guter Beleuchtung zu beobachten. Bei dunklen und gemischten Lichtverhältnissen erzielten SVM und der 1-k-NN die besten Ergebnisse. Die Studie kam zu dem Schluss, dass für eine effektivere Klassifizierung von Widerstandsfarben die Verwendung von Farbbildern unter hellen Bedingungen sowie eine große Anzahl von Trainingsproben von Bedeutung ist [12].

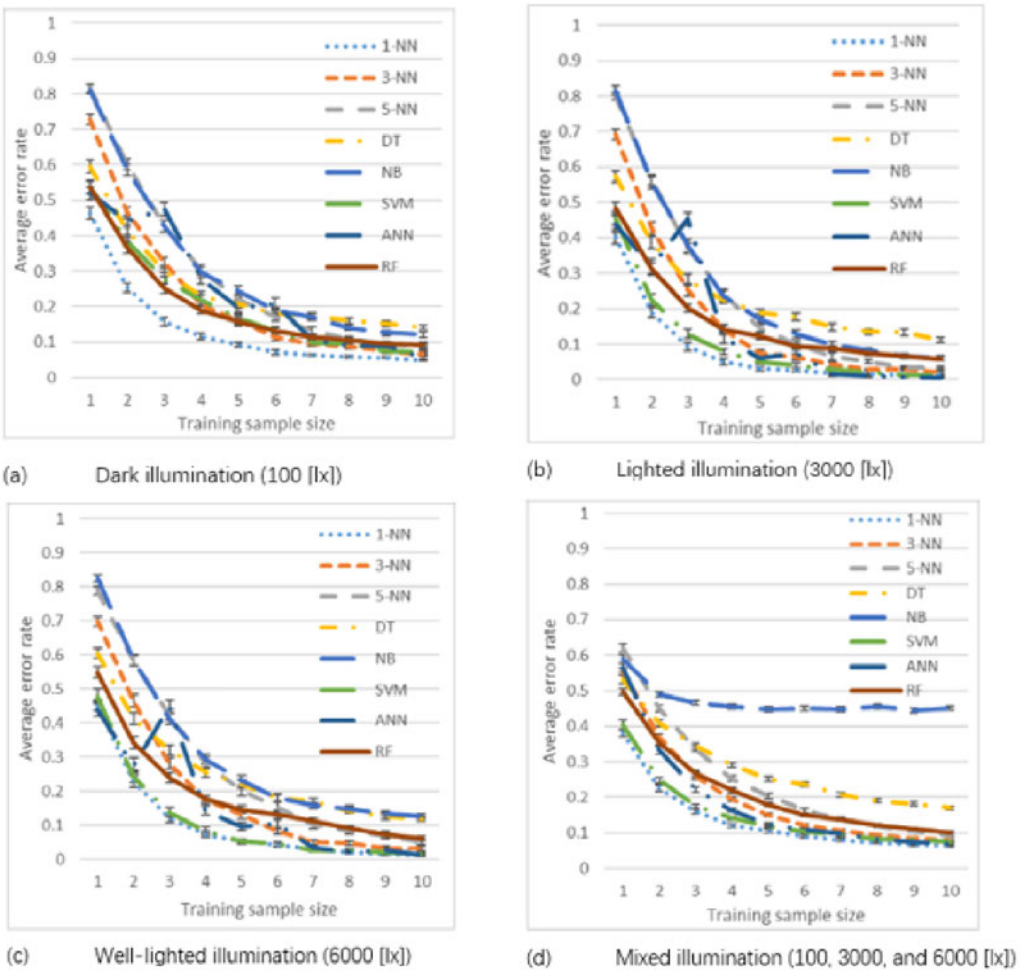


Abbildung 3.10: Der durchschnittliche Fehleranteil in Abhängigkeit von der Anzahl der Trainingsproben pro Klasse für jeden Klassifizierer unter verschiedenen Beleuchtungssituationen [12]

4 Anforderungen und Ziele

In diesem Kapitel wird das Ziel dieser Arbeit formuliert. Außerdem wird eine Anforderungsanalyse durchgeführt, diese stellt sicher, dass das Ziel in Abschnitt 4.1 umgesetzt wird. Es ist außerdem erforderlich Anforderungen zu erstellen, um die Eigenschaften und Fähigkeiten des Produktes zu definieren. Des Weiteren wird durch die Anforderungen eine Grundlage für die Verifikation der Software geschaffen.

4.1 Zielsetzung

Das Ziel dieser Arbeit besteht darin, ein System zu entwickeln, das mithilfe von Deep Learning und Bildverarbeitung Widerstände klassifiziert. Wie bereits in Kapitel 3 erläutert wurde, existieren bereits einige Forschungsarbeiten zu diesem Thema. Die vielversprechende Herangehensweise liegt in der Kombination von Bildverarbeitung und Deep Learning. Durch die Integration der Bildverarbeitungstechniken soll nicht nur die Genauigkeit der Klassifizierung erhöht werden, sondern auch die Qualität der Trainingsdaten verbessert werden, was wiederum die Robustheit und Effizienz des entwickelten Systems steigert. Außerdem soll Kapitel 2 als ein Einstieg in das Thema Deep Learning und die Bildverarbeitung dienen. Das entstehende System kann nicht nur für die vorliegende Arbeit von Nutzen sein, sondern auch als solide Grundlage für zukünftige Forschungsprojekte dienen.

4.2 Systembeschreibung

Wie schon erwähnt, besteht das Hauptziel dieser Arbeit darin, ein Programm zu entwickeln, das die Klassifizierung von Widerständen mithilfe von Deep Learning und Bildverarbeitung ermöglicht. Dies wird durch folgende Schritte erreicht:

1. Erstellung eines Trainingsdatensatzes: Es wird ein umfangreicher Datensatz von Bildern von Widerständen erstellt, der für das Training des Modells verwendet wird.

2. Bildvorverarbeitung: Die Bilder im Trainingsdatensatz werden mithilfe von Bildverarbeitungsalgorithmen vorverarbeitet, um die Qualität der Eingabedaten zu verbessern.

3. Training von Modellen: Es wird ein CNN trainiert. Dieses Modell soll die Merkmale und Muster der Widerstandsbilder aus dem Trainingsdatensatz erlernen.

4. Live-Klassifikation: Das entwickelte Programm ermöglicht die Echtzeitklassifikation von Widerständen, die vor eine Webcam gehalten werden. Das trainierte Modell wird verwendet, um die erkannten Widerstände in Kategorien zu klassifizieren.

Es soll also ein zuverlässiges und effizientes Programm oder System entwickelt werden, das mithilfe von Deep Learning die Erkennung von Widerständen ermöglicht. Dies kann in verschiedenen Anwendungsbereichen von Nutzen sein. Das zu entwickelnde System setzt sich aus zwei Softwarekomponenten zusammen, die miteinander interagieren.

- Bildverarbeitung
- Deep Learning

Der Bildverarbeitungsteil beschäftigt sich mit der Aufbereitung der Bilder wie z. B. der Kontrastmaximierung und der Objektdetektion. Der andere Teil beschäftigt sich mit dem Deep Learning. Für das Training wird ein Datensatz mit Bildern von Widerständen erstellt. Für die Klassifikation wird ein Widerstand in eine Webcam gehalten und anschließend erkannt und verarbeitet. Der entstandene Input wird in das trainierte Netz gegeben und klassifiziert. Die Abbildung 5.3 zeigt die Übersicht des Projektes.

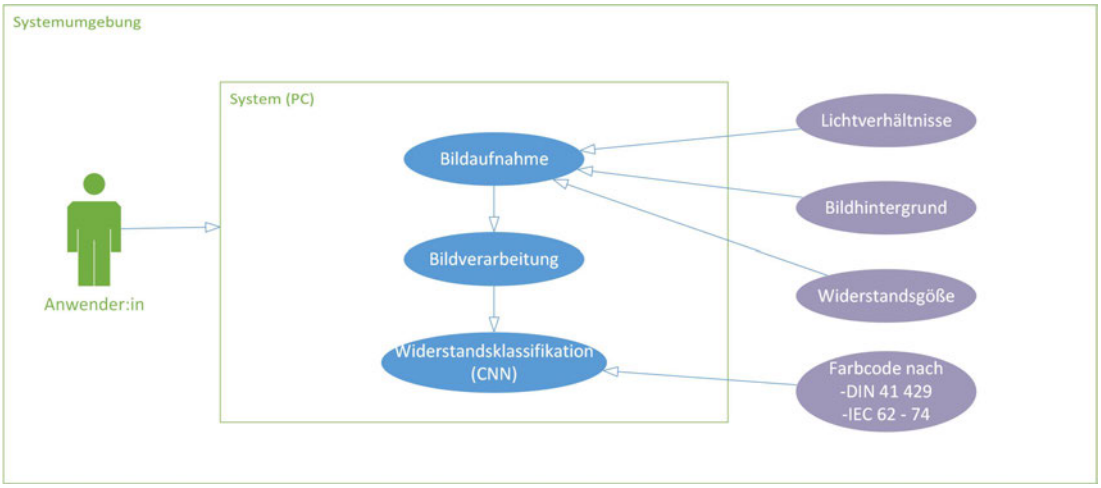


Abbildung 4.1: Systemüberblick

4.3 Hardware

Bei den Widerständen, die klassifiziert werden sollen, handelt es sich um Kohleschichtwiderstände der E-Reihe. Diese haben für die Klassifizierung vier Ringe auf der Kohleschicht des Widerstandskörpers. Unter Abschnitt 2.1 ist der Farbcode erläutert. Zusätzlich zu den Kohleschichtwiderständen sollen auch optional Metallschichtwiderstände mit fünf Ringen klassifiziert werden.

4.4 Stakeholder

Stakeholder bezeichnen Personen, Gruppen oder Institutionen, die von den Aktivitäten eines Unternehmens direkt oder indirekt betroffen sind oder ein Interesse an diesen Aktivitäten haben. Durch die Stakeholder und deren Interessen, entstehen Anforderungen, die beim Systementwurf beachtet werden müssen [4].

Anwender:in
Die Anwender können mithilfe der Software Widerstände klassifizieren. Diese Widerstände können vor die Kamera des Laptops gehalten werden. Eine Klassifizierung ist beispielsweise am Arbeitsplatz sinnvoll. Das Programm soll einer breiten Nutzungsgruppe zur Verfügung stehen und sollte somit auf mehreren Plattformen mit unterschiedlichen

Hardwareanforderungen zur Verfügung stehen. Voraussetzung ist eine Webcam, welche Bilder mit unterschiedlichen Hintergründen und Beleuchtungen aufnehmen kann.

Auftraggeber
Der betreuende Professor und Erstprüfer Herr Dr.-Ing Marc Hensel ist der Auftraggeber. Die Anforderungen wurden persönlich abgestimmt. Gefordert ist eine automatische Klassifikation von Widerständen, dabei sollen sowohl die Bildverarbeitung als auch Deep Learning zum Einsatz kommen. Die am Ende entstandene Software kann für weitere Projekte genutzt werden oder als Vergleich für bestehende Projekte dienen. Für zukünftige Abschlussarbeiten kann die Arbeit als Grundlage dienen. Durch die entstandenen Beispiele und Demonstratoren im Bereich Deep Learning und der Bildverarbeitung entsteht ein Erkenntnisgewinn. Für die weitere Verwendung und Nachvollziehbarkeit sollte eine gute Dokumentation vorhanden sein. Durch die Veröffentlichung der Masterarbeit kann diese auch für die Lehre nützlich sein.

Softwareentwicklerin
Die Softwareentwicklerin ist die Autorin dieser Arbeit. Das Interesse besteht darin eine Software zu erstellen, welche zuverlässig Widerstände klassifiziert. Auch die Methoden- und Algorithmen-Entwicklung ist von Interesse. Am Ende soll eine Software entstehen die, die geforderten Anforderungen erfüllt. Außerdem sollen das Wissen im Bereich Deep Learning und Bildverarbeitung gefestigt und erweitert werden. Die KI-Modellentwicklung ist ein weiterer Interessenpunkt.

Weiterentwicklung durch Studierende
Für die Weiterentwicklung oder einen Vergleich des Systems sollte eine gute Dokumentation vorhanden sein, damit Studenten sich gut in das Thema einarbeiten können. Die entwickelte Bildverarbeitung und Künstliche Intelligenz (KI) kann auch als Grundlage für zukünftige Arbeit dienen.

4.5 Anwendungsfälle

Nachdem die Systembeschreibung und die Interessengruppen betrachtet wurden, werden die Anwendungsfälle näher untersucht. Diese sind wichtig für das Konzept und die anschließende Validierung. Im Folgenden werden die Anwendungsfälle näher beschrieben.

4.5.1 Bildaufnahme

Der erste Schritt ist die Aufnahme eines Bildes. Dieses Bild wird als Farbbild mit drei Kanälen gespeichert und kann, wenn nötig in ein Grauwertbild oder Binärbild umgewandelt werden.

Titel	Bildaufnahme
Kurzbeschreibung	Bild eines Widerstandes wird mit der Laptop Webcam aufgenommen
Eingang	Bild von Webcam
Ausgang	Bilddatei mit 3 Farbkanälen
Ablauf	Bild wird durch den Anwender aufgenommen und weiterverarbeitet

Tabelle 4.1: Anwendungsfall Bildaufnahme

4.5.2 Objektdetektion

Ein wichtiger Punkt ist die Detektierung des Widerstandes, dieser soll erkannt und durch eine Bounding Box umrandet werden.

Titel	Objektdetektion
Kurzbeschreibung	Der Widerstand wird mithilfe eines Algorithmus detektiert und ausgeschnitten.
Eingang	Binärbild
Ausgang	Bildausschnitt des Widerstandes
Ablauf	Der Widerstand wird durch einen Algorithmus detektiert und die resultierende Bounding Box wird ausgeschnitten und angezeigt

Tabelle 4.2: Anwendungsfall Objektdetektion

4.5.3 Widerstandsklassifikation

Der Bildausschnitt des Widerstands soll klassifiziert werden. Es soll der Widerstandswert ausgegeben werden. Außerdem soll die Wahrscheinlichkeit des Klassifizierungsergebnis ausgegeben werden.

Titel	Widerstandsklassifikation
Kurzbeschreibung	Mithilfe eines CNN wird der Widerstand klassifiziert
Eingang	Bildausschnitt des Widerstandes
Ausgang	Widerstandswert und Wahrscheinlichkeit
Ablauf	Der Bildausschnitt wird in ein Faltungsnetz gegeben und klassifiziert

Tabelle 4.3: Anwendungsfall Widerstandsklassifikation

4.6 Anforderungen

Damit das System getestet und validiert werden kann, werden aus der Analyse Anforderungen abgeleitet. Auch für die Entwicklung sind Anforderungen an das System unverzichtbar. Die Anforderungen mit der Endung *opt* sind als optional zu betrachten.

SW-1 Als Programmierumgebung soll Visual Studio dienen.

SW-2 Es soll ein Trainings- und ein Testdatensatz erstellt werden.

SW-3 Der in die Kamera gehaltene Widerstand soll mit einer Bounding Box umrandet werden und die Widerstandswerte neben der Box stehen.

SW-4 Der Testdatensatz soll eine Genauigkeit von mindesten 90% vorweisen.

SW-5 Das Programm soll modular aufgebaut werden.

SW-6 Die Klassifizierung soll den Widerstandswert in Ω angeben.

SW-7 Der Widerstand darf beliebig gedreht sein.

SW-8 Es sollen mindestens zwei Ansätze entwickelt und getestet werden.

SW-9 Für die Klassifizierung soll ein CNN verwendet werden.

SW-10-opt Widerstände sollen bei wechselndem Hintergrund erkannt und klassifiziert werden.

SW-11-opt Widerstände sollen bei wechselndem Beleuchtungen erkannt und klassifiziert werden.

HW-1 Die Webcam des Laptops soll genutzt werden.

HW-2 Es sollen Kohleschichtwiderstände aus der E-Reihe klassifiziert werden.

HW-3-opt Es sollen zusätzlich auch Metallschichtwiderstände klassifiziert werden.

Die Zielsetzung einer Testgenauigkeit von 90 % beruht auf verschiedenen Überlegungen. Um ein zufriedenstellendes Ergebnis zu gewährleisten, sollte diese Marke mindestens erreicht werden. Da es sich nicht um eine sicherheitskritische Anwendung handelt, besteht keine zwingende Notwendigkeit, eine höhere Genauigkeit anzustreben. Die begrenzte Menge an verfügbaren Daten könnte zudem die Erreichung einer höheren Genauigkeit beeinträchtigen. Letztendlich spielt auch der persönliche Anspruch eine Rolle, die Testgenauigkeit auf mindestens 90 % zu erhöhen.

5 Konzept

Dieses Kapitel widmet sich der Darlegung des Konzepts und berücksichtigt dabei die Anforderungen, die in Kapitel 4 formuliert wurden.

5.1 Ansätze

Um das Ziel zu erreichen sind mehrere Konzepte möglich. Diese werden im Folgenden diskutiert und am Ende wird eine Entscheidung für das finale Konzept getroffen. Die Abbildung 5.1 zeigt mögliche Ansätze für das Konzept.

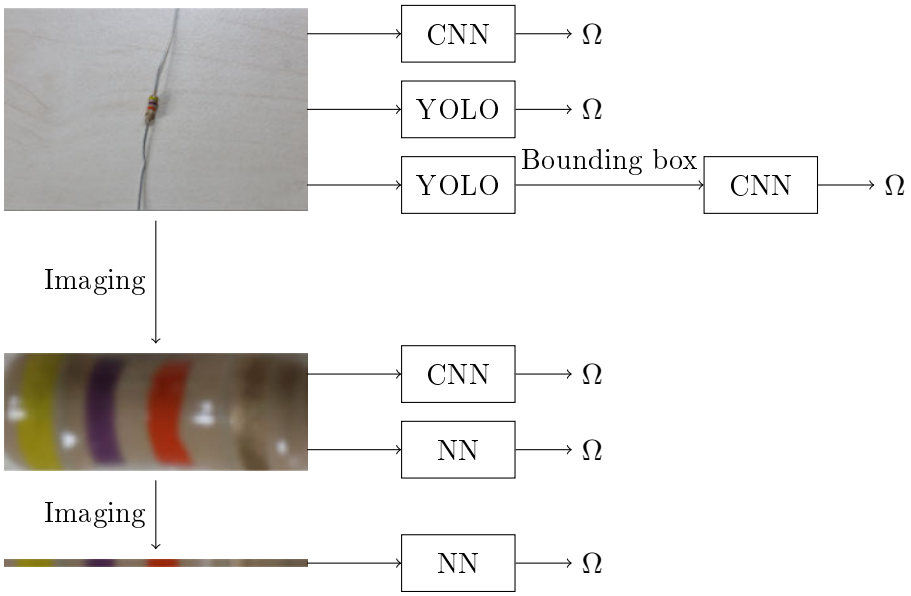


Abbildung 5.1: Konzept

5.1.1 Vollbild

Das einfachste Konzept besteht darin, die Originalbilder einem Faltungsnetz zu übergeben. Wie bereits in Abschnitt 3 erwähnt, wurde dieser Ansatz in der Arbeit von Elias Soud [19] verfolgt. Ein erheblicher Nachteil dieses Ansatzes liegt darin, dass das Netz nicht nur die Widerstände fokussiert, sondern auch den Hintergrund einbezieht. Dadurch ist die Robustheit dieses Konzepts äußerst begrenzt, weshalb es nicht in Betracht gezogen werden kann. Zusätzlich sind die benötigten Trainingsdaten vergleichsweise größer. Der einzige Vorteil besteht in einem geringeren Implementierungsaufwand.

Eine mögliche Verbesserung wäre der Einsatz eines You only look once (YOLO) Detektionsalgorithmus. Dieser könnte entweder die Widerstände erkennen und klassifizieren oder aber im Anschluss an den YOLO-Algorithmus, der lediglich die Widerstände erkennt, könnte ein Faltungsnetz zur Klassifizierung geschaltet werden.

5.1.2 Bildausschnitt

Ein vielversprechendes Konzept besteht darin, die Widerstände aus den Trainingsbildern zu detektieren und auszuschneiden. Diese ausgeschnittenen Bildbereiche werden dann einem Faltungsnetz zur Klassifizierung übergeben. Diese Herangehensweise bietet den bedeutenden Vorteil der Robustheit: Durch die vorherige Detektion der Widerstände wird das Netz nicht mit dem Hintergrund belastet. Zudem ist die Größe der benötigten Trainingsdaten geringer, was wiederum die Trainingsdauer verkürzt. Eine andere Möglichkeit ist die Verwendung von mehreren mittleren Bildzeilen, welche gemittelt werden. Dieser Bildstreifen mit den gemittelten Werten kann ebenfalls einem Faltungsnetz übergeben werden.

Im Vergleich zum vorherigen Ansatz besteht allerdings der Nachteil des höheren Umsetzungsaufwands. Es ist notwendig, einen Bildverarbeitungsalgorithmus zu entwickeln, der in der Lage ist, die Widerstände zuverlässig zu erkennen und auszuschneiden. Trotzdem überwiegen die Vorteile dieses Konzepts, da es eine präzisere und effizientere Klassifizierung ermöglicht.

5.1.3 Bildzeile

Ein weiteres Konzept besteht darin, Bildzeilen als Trainingsdaten zu verwenden. Auch hier erfolgt zunächst die Detektion des Widerstands. Anschließend wird eine Bildzeile aus dem erkannten Widerstandsausschnitt entnommen, die die Farbinformationen enthält. Diese Farbvektoren werden dann durch ein Neural Network (NN) klassifiziert. Im Gegensatz zum vorherigen Bildausschnitt-Konzept wird hier kein Faltungsnetz, sondern ein neuronales Netzwerk für eindimensionale Daten verwendet. Ein zusätzlicher Vorteil dieses Ansatzes besteht darin, dass die Größe der benötigten Trainingsdaten verringert wird.

Die Verwendung von Bildzeilen als Trainingsdaten ermöglicht eine effiziente Verarbeitung der Farbinformationen. Die Anwendung eines neuronalen Netzwerks kann eine präzise Klassifizierung der Widerstände ermöglichen.

5.1.4 Vergleich

In der Tabelle 5.1 sind die diskutierten Themen zusammengefasst.

	Vollbild	Bildausschnitt	Bildzeile
Umsetzungsaufwand	+	o	o
Robustheit	-	+[2]	o[2]
Größe Trainingsdaten	-	o	+
Trainingsdauer	-	o	+

Tabelle 5.1: Konzeptvergleich

Der Ansatz „Vollausschnitt“ wird wegen seiner geringen Robustheit nicht weiter verfolgt. Der Ansatz „Bildzeile“ ist vielversprechender aufgrund seiner geringen Trainingsdauer und gleichzeitig voraussichtlich guten Robustheit. Aus Zeitgründen konnte dieser Ansatz leider nicht mehr angewendet werden.

Das erfolgversprechendste Konzept ist das des „Bildausschnittes“. Dieser Ansatz verspricht eine hohe Robustheit gegenüber variierenden Hintergründen. Zur Evaluierung der Leistung werden verschiedene Ansätze von Faltungsnetzen verwendet, um einen Vergleich hinsichtlich ihrer Effektivität anzustellen.

[2] voraussichtlich

5.2 Ablauf

Wie vorher diskutiert, wird der Ansatz des Bildausschnittes weiterverfolgt. Die Abbildung 5.2 zeigt den konzeptionellen Ablauf.

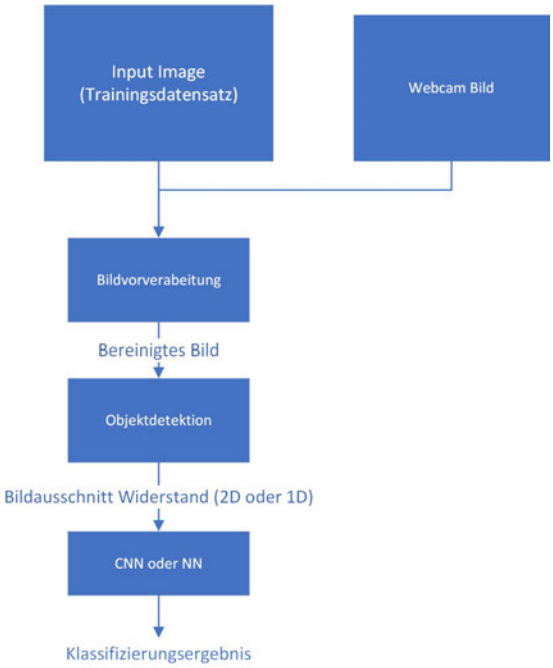


Abbildung 5.2: Grober Ablauf

Dabei ist zu beachten, dass im ersten Schritt die Trainingsdaten als Input dienen und damit das Netz trainiert wird. Das trainierte Netz wird gespeichert. Im zweiten Schritt dienen dann die Bilder der Webcam als Input. Das vorverarbeitete Webcam-Bild mit dem Widerstand wird in das trainierte Netz gegeben und klassifiziert. Abbildung 5.3 verdeutlicht den unterschiedlichen Verlauf in der Entwicklung und in der Anwendung.

Der Ablauf in der Entwicklung und Anwendung zeigt keine großen Unterschiede. Der Hauptunterschied liegt in den verwendeten Eingabedaten. Während der Entwicklung werden die aufgenommenen Bilder der Widerstände als Eingabe verwendet, während in der Anwendung die Webcam-Bilder dienen. Die Bildausschnitte werden während der Entwicklungsphase in ein nicht trainiertes Netzwerk eingegeben, während sie in der Anwendungsphase in das zuvor trainierte Netzwerk eingegeben werden, um ein Klassifizierungsergebnis zu erhalten.

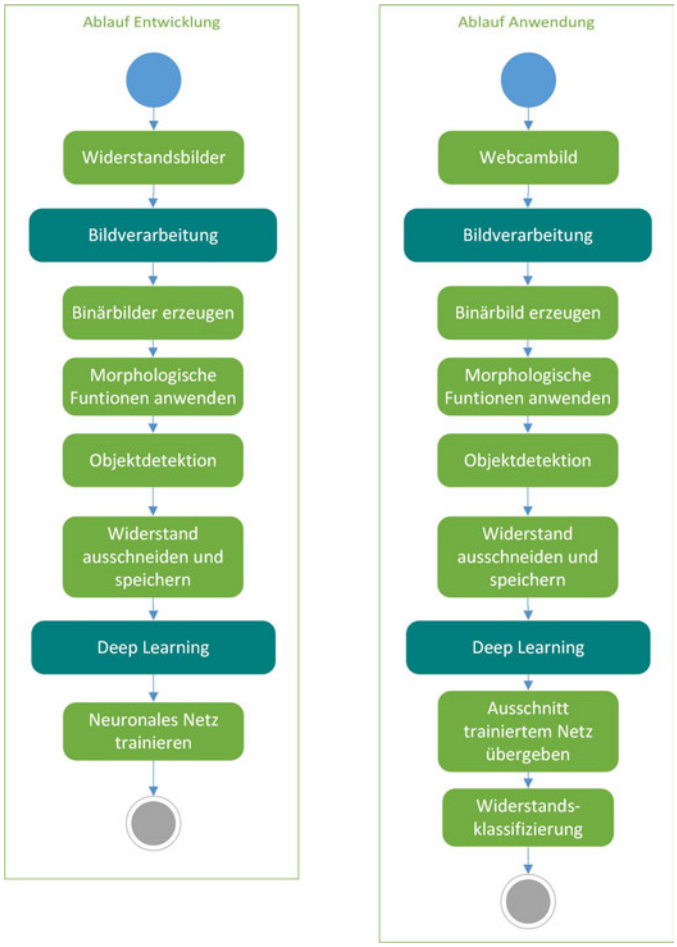


Abbildung 5.3: Ablauf

In der Entwicklungsphase besteht das Ergebnis des Netzwerks aus einer Genauigkeitsbewertung, die angibt, wie gut das Netzwerk in der Lage ist, die Widerstände zu klassifizieren. Dies ermöglicht eine Einschätzung der Leistungsfähigkeit des Netzwerks in Bezug auf die Widerstandserkennung.

6 Erzeugung der Bilddaten

Dieses Kapitel widmet sich der Generierung von Bilddaten, die für das Training eines neuronalen Netzes erforderlich sind. Um ein effektives Training zu gewährleisten, müssen ausreichend Trainingsdaten zur Verfügung stehen. In diesem Kontext stellen die Trainingsdaten Abbildungen von verschiedenen Widerständen dar. Um die Robustheit des Netzwerks zu maximieren, ist es von Bedeutung, dass die Bilddaten eine Vielzahl von Variationen in Bezug auf Lichtverhältnisse, Bildhintergrund, Ausrichtung der Widerstände und Widerstandsgrößen aufweisen. Zudem spielen sowohl die Dimension als auch die Qualität der Bilder eine entscheidende Rolle.

Es ist wichtig, darauf zu achten, dass die Bildgröße angemessen gewählt wird, um unnötige Verlängerungen der Trainingszeit und zusätzlichen Rechenaufwand zu vermeiden. Obwohl die Bilder nicht unbedingt hochauflösend sein müssen, sollten sie dennoch eine gewisse Qualität aufweisen, um die zuverlässige Darstellung der relevanten Merkmale sicherzustellen. Jeder Widerstand wird in 201 verschiedenen Bildern erfasst, und da insgesamt 20 verschiedene Widerstände betrachtet werden, ergibt sich eine Gesamtanzahl von 4020 Trainingsbildern. Die Kohleschichtwiderstände sind in Tabelle 6.1 aufgeführt. Es ist wichtig anzumerken, dass die Toleranz bei diesen Widerständen $\pm\,5\,\%$ beträgt. Um die Zuverlässigkeit und Genauigkeit des Systems weiter zu steigern, wurden zusätzliche Bilddaten integriert. Dieser Schritt ermöglicht eine Erhöhung der Trainingsdatenmenge, die von entscheidender Bedeutung ist. Die ergänzenden Bilddaten wurden von Martin Zeilinger zur Verfügung gestellt, der ebenfalls an einer Arbeit zu diesem Thema arbeitet [25].

Widerstand	Bild	Widerstand	Bild	Widerstand	Bild
1 Ω		10 Ω		100 Ω	
1 $k\Omega$		10 $k\Omega$		100 $k\Omega$	
1 $M\Omega$		150 Ω		1.5 $k\Omega$	
220 Ω		2.2 $k\Omega$		22 $k\Omega$	
220 $k\Omega$		330 Ω		33 $k\Omega$	
47 Ω		470 Ω		4.7 $k\Omega$	
47 $k\Omega$		470 $k\Omega$			

Tabelle 6.1: Widerstände

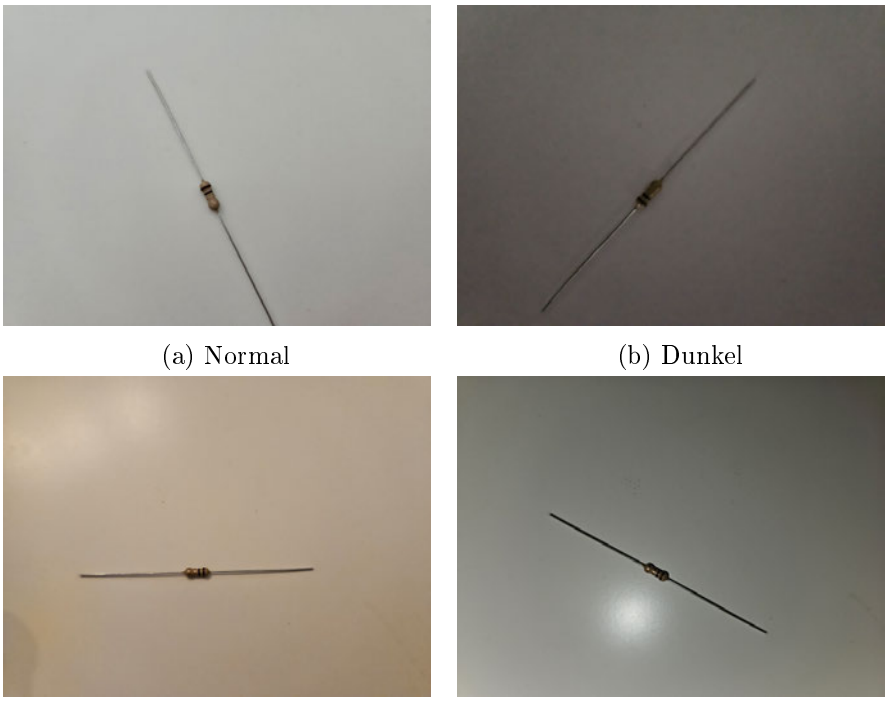
6.1 Größe und Qualität der Bilder

Die Aufnahmen stammen von einem Samsung Galaxy S23+, welches die Bilder im JPEG-Format speichert. Das Joint Photographic Experts Group (JPEG) Format bietet einen hohen Grad an kontrollierter Kompression. Benutzer können das Verhältnis zwischen Qualität und Dateigröße nach ihren eigenen Anforderungen anpassen. Dies ermöglicht sowohl kleine Dateigrößen als auch hohe Bildqualität bei hoher Komprimierung [26]. Für diese Anwendung erweist sich dieses Format als äußerst geeignet, da eine geringere Dateigröße die Leistung des Trainingsprozesses verbessert. Die Bildabmessungen von 3000x4000 Pixeln sind für diese Anwendung nicht erforderlich und daher überdimensioniert. Daher werden diese Abmessungen mithilfe der OpenCV-Funktion *resize* verkleinert.

6.2 Lichtverhältnisse

Bilder können sowohl unter- als auch überbelichtet sein. Beide Lichtverhältnisse erschweren die Segmentierung, da der Vordergrund sich nicht groß vom Hintergrund unterscheidet. Damit der Algorithmus auf beide Fälle trainiert wird, werden Bilder bei normalen, etwas helleren und etwas dunkleren Lichtverhältnissen aufgenommen. Somit wird die Robustheit gegenüber Helligkeitsschwankungen erhöht. Abbildung 6.1 zeigt eine Auswahl

der Trainingsbilder bei verschiedenen Lichtverhältnissen. Die Bilder wurden bei natürlichem Licht, künstlichen Licht, wenig Licht und mit Blitzlicht aufgenommen.



(a) Normal (b) Dunkel

(c) Künstliches Licht (d) Mit Blitz

Abbildung 6.1: Widerstand 1 Ω bei verschiedenen Lichtverhältnissen

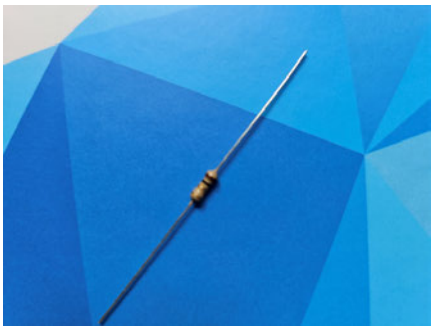
6.3 Hintergrund

Der Hintergrund trägt ebenfalls wesentlich zur Segmentierung bei. Optimal ist ein durchgängig weißer Hintergrund, da der Widerstand dadurch klar vom Hintergrund abgegrenzt werden kann. Da diese Bedingung jedoch in der Praxis selten erfüllt ist, wird die Widerstandsfähigkeit der Segmentierung durch die Berücksichtigung variierender Hintergründe gesteigert. Abbildung 6.2 veranschaulicht den Widerstand vor verschiedenen Hintergründen. Vorwiegend werden Bilder vor einheitlich weißen Hintergründen aufgenommen, da in diesen Fällen die Segmentierung am vielversprechendsten ist. Denn es ist entscheidend, ausreichend ausgeschnittene Widerstände als Trainingsdaten zu haben. Es werden auch Bilder mit Hintergründen wie Holz oder Arbeitsplatten verwendet, um die Robustheit der Segmentierung zu gewährleisten.

(a) Weiß

(b) Arbeitsplatte

(c) Fensterbank



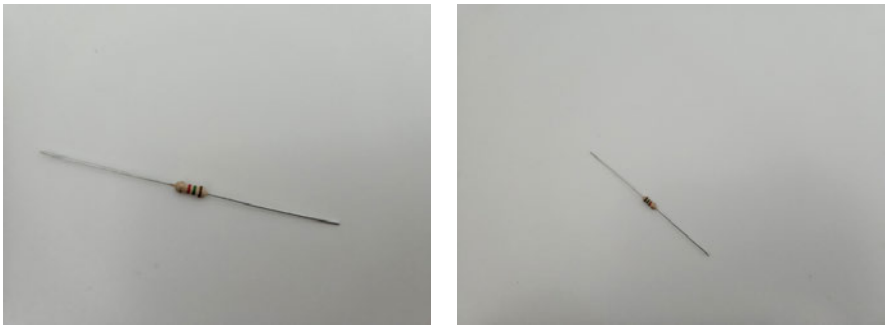
(d) Geometrisch Blau

(e) Holz

Abbildung 6.2: Widerstand 1 Ω mit verschiedenen Hintergründen

6.4 Widerstandsausrichtung und -Größe

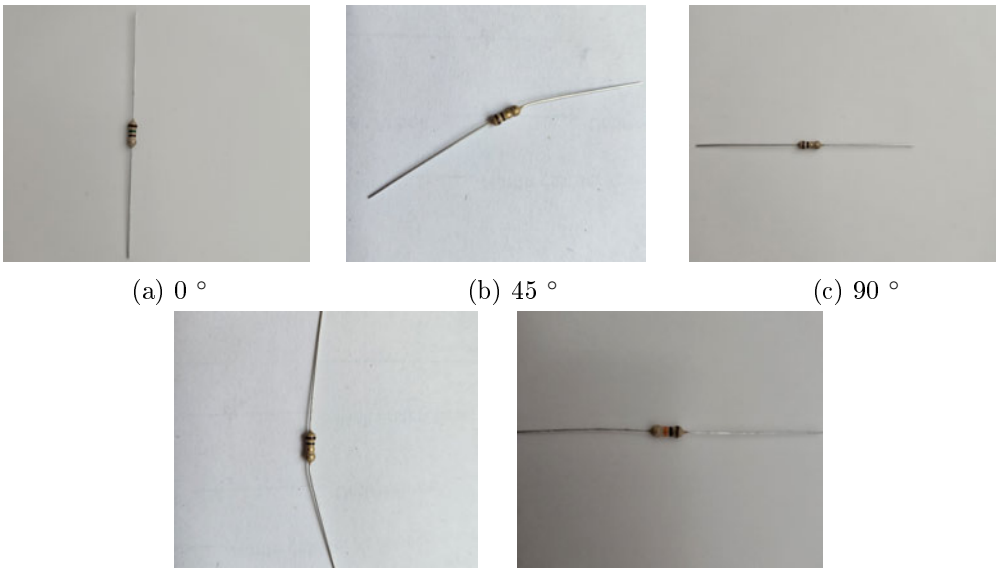
Die Ausrichtung der Widerstände spielt ebenfalls eine wichtige Rolle. Um eine zuverlässige Erkennung der Ringe zu gewährleisten, müssen die Widerstände korrekt ausgerichtet werden. Hierfür werden die Widerstände in kleinen Schritten um 360° Grad gedreht und jeweils aufgenommen. Zusätzlich wird die Größe der Widerstände variiert, um eine erhöhte Robustheit zu erzielen.



(a) Groß (b) Klein

Abbildung 6.3: Verschiedene Größen der Widerstandskörper

Abbildung 6.3 zeigt den Widerstand in großer und in kleiner Auflösung.



(a) 0 ° (b) 45 ° (c) 90 °

(d) 180 ° (e) 270 °

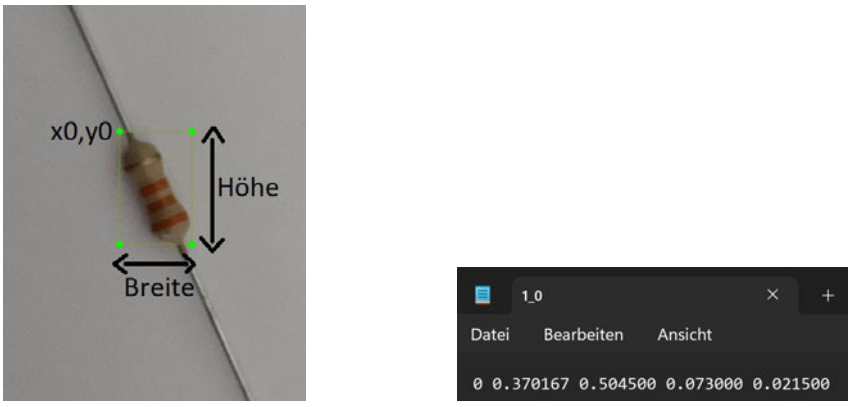
Abbildung 6.4: Verschieden Widerstandsausrichtungen

Die Abbildung 6.4 zeigt einen Widerstand mit verschiedenen Ausrichtungen.

6.5 Labeling und Bounding Boxen

Um sicherzustellen, dass die Widerstände korrekt erkannt werden, werden Referenz-Bounding-Boxen erstellt. Diese dienen später dazu, zu überprüfen, ob der Widerstand ordnungsgemäß gefunden wurde. Zur Erstellung dieser Referenzboxen wird das Programm *labelIMG*[1] verwendet. Dieses Tool ermöglicht das Labeln von Bildern und ist in Python geschrieben. Es nutzt Qt für die grafische Benutzeroberfläche. Die Daten der definierten Bounding Box (BB) werden in einer Textdatei gespeichert. Diese Daten umfassen das Label, die x_0- und y_0-Koordinaten, die Breite und die Höhe der Bounding Box.

In diesem Kontext ist das Label unwesentlich, da sich pro Bild nur ein Widerstand befindet. Die Textdateien werden in den jeweiligen Ordner der Widerstandsbilder gespeichert, am Ende entstehen somit 20 Ordner, die jeweils die Bilder eines Widerstands beinhalten und die dazugehörigen Textdateien mit den Bounding Box-Koordinaten. Die Koordinaten richten sich nach dem YOLO-Algorithmus. Dabei werden die Koordinaten nicht pixelgenau festgelegt, sondern proportional zur Bildgröße angegeben. Ein wesentlicher Vorteil dieser Methode besteht darin, dass die Bounding Box auf Bilder unterschiedlicher Größen angewendet werden kann. Abbildung 6.5 illustriert beispielhaft eine Bounding Box für einen Widerstand sowie die zugehörige Textdatei.



(a) Bounding Box eingezeichnet (b) Textdatei mit BB Koordinaten

Abbildung 6.5: Bounding Box

[1] https://pypi.org/project/labelImg/

7 Bildverarbeitung

Die Bildverarbeitung in dieser Arbeit ist für die Vorverarbeitung der Daten sowie für die nachfolgenden Schritte der Segmentierung, Detektion und Ausrichtung des Widerstands verantwortlich. Für die Bildverarbeitung sind mehrere Ansätze möglich, um das Ziel zur erreichen, den Widerstand auszuschneiden.

7.1 Design

Trotz der verschiedenen Ansätze bleibt der grobe Ablauf gleich.

Abbildung 7.1: Bildverarbeitungskette

Abbildung 7.1 veranschaulicht den gängigen Ablauf einer Bildverarbeitungskette. Der Schritt der Akquisition umfasst die Datenerfassung, welche im Kapitel 6 ausführlich behandelt wird. Das Konzept der Vorverarbeitung, Segmentierung und Merkmalsextraktion werden im Folgenden erläutert. Der Begriff „Klassifikation“ bezieht sich in diesem Zusammenhang auf die Detektion des Widerstandes als Objekt.

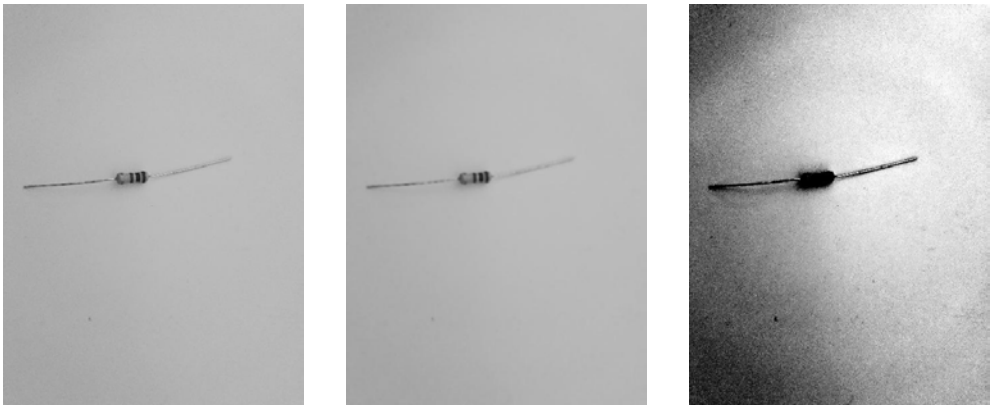
7.1.1 Entwicklungsumgebung

Als Entwicklungsumgebung wird Visual Studio 2022 in Verbindung mit der Programmiersprache C++ genutzt. C++ ist weit verbreitet und bietet zahlreiche Vorteile wie objektorientierte Programmierung, Typsicherheit und eine breite Palette an Bibliotheken, auch in Bezug auf Bildverarbeitung. Für die Bildverarbeitung wird OpenCV verwendet,

es handelt sich um eine freie Programmbibliothek für die Bildverarbeitung [16]. OpenCV ist in der Bildverarbeitung sehr verbreitet und steht für verschiedene Programmiersprachen wie C++, Python und Java zur Verfügung. Diese Bibliothek bietet eine umfangreiche Sammlung von Bildverarbeitungsalgorithmen. Für die Programmierung mit C++ und OpenCV existieren bereits Vorkenntnisse.

7.1.2 Bildvorverarbeitung

Der erste Schritt besteht darin, das Farbbild in ein Grauwertbild umzuwandeln. Um eine zuverlässige Segmentierung zu ermöglichen, kann eine Kontrastmaximierung sinnvoll sein. Bei der Kontrastmaximierung handelt es sich um eine ortsunabhängige Punktoperation, deren Abbildungsvorschrift sich aus dem Histogramm ergibt [8]. Darüber hinaus ist auch die Anwendung digitaler Filter zur Bildglättung sinnvoll, um beispielsweise das Rauschen zu reduzieren. Beispiele hierfür sind der Median-Filter und der Gauß'sche Weichzeichner. In Abbildung 7.2 und Abbildung 7.3 sind die angewendeten Methoden dargestellt.



(a) Grauwertbild (b) Gauß'sche Weichzeichner (c) Kontrastmaximierung

Abbildung 7.2: Bildvorverarbeitung

Bei Abbildung 7.3 handelt es sich um ein Histogramm. Ein Histogramm ist eine grafische Darstellung einer Häufigkeitsverteilung, oft repräsentiert es die Verteilung der Pixelwerte in einem Graustufenbild [8].

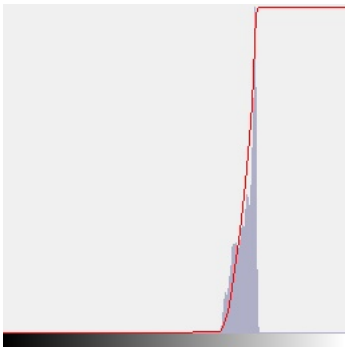


Abbildung 7.3: Histogramm (blau) und kumulatives Histogramm (rot)

7.1.3 Segmentierung

Nach Abschluss der Vorverarbeitung kann das Objekt vom Hintergrund segmentiert werden. Hierfür ist die Umwandlung des Grauwertbildes in ein Binärbild erforderlich. Es stehen verschiedene Verfahren zur Verfügung, um die Segmentierung zu realisieren.

Unter den pixelorientierten Verfahren wird für jedes Einzelpixel überprüft, ob es zum Hintergrund oder Vordergrund gehört. Ein bekanntes pixelorientiertes Verfahren ist das globale Schwellwertverfahren. Abbildung 7.4 zeigt ein exemplarisches Ergebnis.

Abbildung 7.4: Globales Schwellwertverfahren mit Schwellwert 160

Bei adaptiven Schwellwertverfahren wird für jedes Pixel ein individueller Schwellwert berechnet. Dies kann mithilfe von Durchschnittswerten oder einer Fensterberechnung, beispielsweise mit einem Gauß'schen Fenster, erfolgen, wobei die Umgebung berücksichtigt

wird. Abbildung 7.5 verdeutlicht diesen Ansatz. Im Gegensatz zum globalen Schwellwertverfahren ist der Hintergrund deutlich unruhiger, aber der Widerstandskörper ist dennoch deutlich erkennbar.

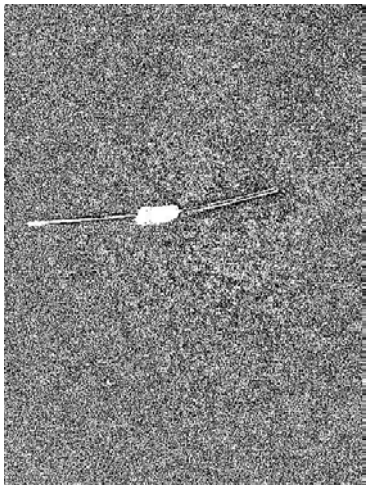


Abbildung 7.5: Adaptives Schwellwertverfahren

Kantenorientierte Verfahren zielen darauf ab, Kanten oder Übergänge in Bildern zu erkennen. Beispiele für solche Verfahren sind der einfachere Sobel-Filter oder der komplexere Canny-Algorithmus, welche beide Faltungsoperationen nutzten. In Abbildung 7.6 sind die Kanten deutlich sichtbar.

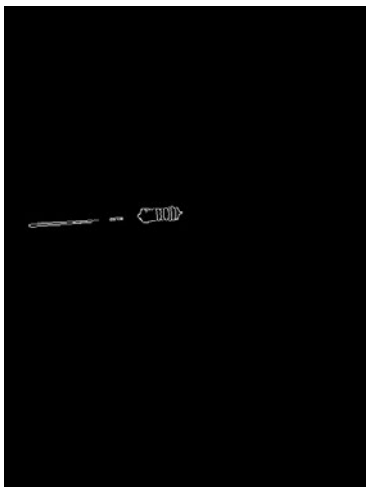


Abbildung 7.6: Canny Algorithmus

Im Gegensatz zu Pixelorientierten Verfahren berücksichtigen Region orientierte Verfahren auch benachbarte Pixel. Ein Beispiel ist das Region-Growing. Modellbasierte Verfahren wie die Hough-Transformation erfordern Vorwissen über das Bild, beispielsweise die geometrische Form der Objekte [10].

In Tabelle 7.1 werden vier verschiedene Algorithmen für die Segmentierung diskutiert und verglichen. Unter Robustheit versteht man hier die Fähigkeit, mit der die Software auch bei Veränderungen wie Helligkeitsschwankungen stabil und zuverlässig funktioniert.

Algorithmus	**Vorteile**	**Nachteile**
Fester Schwellwert	Einfachheit und kurze Rechenzeit	Empfindlich gegenüber Helligkeitsveränderungen und Rauschen, Mangelnde Anpassungsfähigkeit
Adaptiver Schwellwert	Robust gegenüber Rauschen, Anpassungsfähigkeit	Höhere Rechenzeit und Komplexität
Canny Algorithmus	Robust, Geringe Fehlerrate, Automatische Schwellenwertwahl	Rechenaufwändig
Hough-Transformation	Robust, Parameterfreie Linienentdeckung	Rechenaufwändig, nicht effektiv für kurvige Linien

Tabelle 7.1: Segmentierungsalgorithmen Vergleich

Angesichts der Variation der Lichtverhältnisse bei den verwendeten Bildern ist die Anwendung eines festen Schwellwerts nicht sinnvoll. Stattdessen wird auf eine Methode mit adaptivem Schwellwert zurückgegriffen, da diese sich bei der Verarbeitung von Widerstandsbildern als effektiv erwiesen hat. Auch der Canny-Algorithmus wird in einem Ansatz verwendet. Nach der Generierung des Binärbildes werden morphologische Operationen darauf angewendet, um die Form des erkannten Objekts zu verändern. Dadurch können unruhige Hintergründe entfernt oder die Drähte der Widerstände beseitigt werden.

Die Erosion entfernt beispielsweise die äußeren Pixel eines Objekts, wie in Abbildung 7.7 dargestellt. In den folgenden Bildern wird ein Filterkern mit einer rechteckigen Struktur

der Größe 5x5 verwendet. Diese Operationen wurden auf das Binärbild angewendet, das durch einen globalen Schwellwert erzeugt wurde.

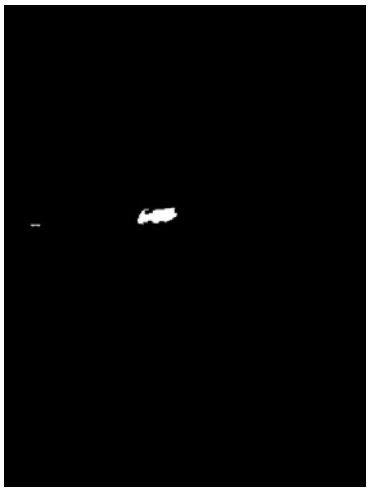


Abbildung 7.7: Erosion

Die Dilatation hingegen erweitert die Ränder eines Objekts, wie Abbildung 7.8 verdeutlicht.

Abbildung 7.8: Dilatation

Es gibt auch Kombinationen dieser beiden Operationen, die als Opening und Closing bezeichnet werden [8]. In diesem Fall erweist sich das Opening als besonders stark. Durch die Erosion werden störende Elemente wie der Draht entfernt und anschließend durch

Dilatation der Widerstandskörper wieder in seine ungefähre ursprüngliche Größe zurückgebracht. Dieser Prozess hilft dabei, das erkannte Objekt präziser zu definieren und unerwünschte Artefakte zu reduzieren.

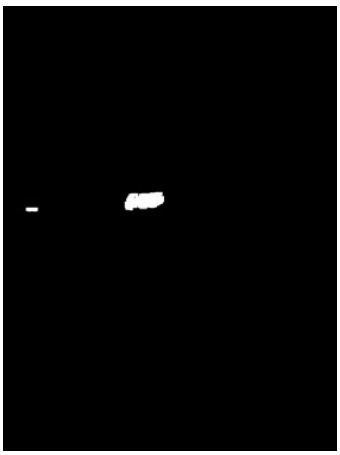


Abbildung 7.9: Opening

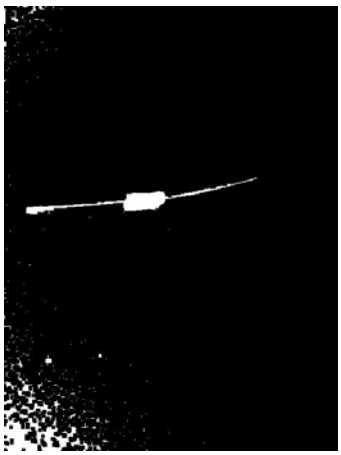


Abbildung 7.10: Closing

Abbildung 7.9 zeigt das Opening, bei dem zunächst störende Elemente entfernt und anschließend nützliche Elemente wiederhergestellt werden. Das Closing hingegen, bei dem zuerst die Dilatation und dann die Erosion angewendet wird, ist für diese Anwendung ungeeignet, wie Abbildung 7.10 zeigt.

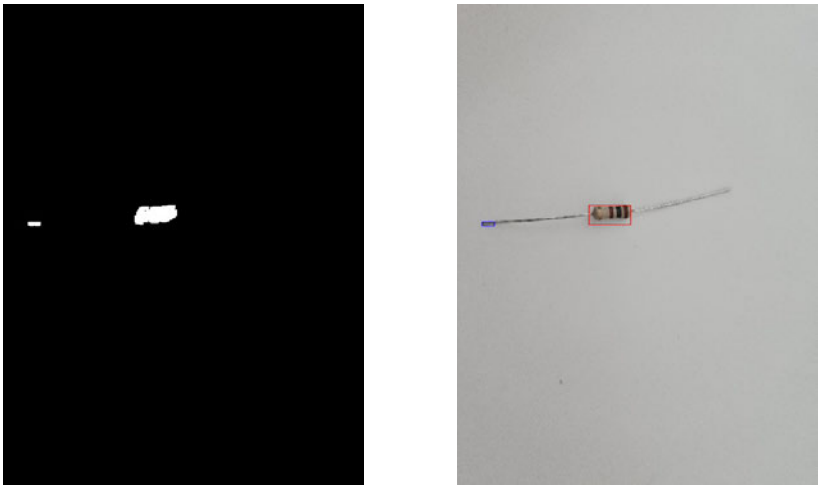
7.1.4 Objektdetektion und Merkmalsextraktion

Durch die Segmentierung werden Objekte vom Hintergrund getrennt. Anschließend muss geprüft werden, welche Pixel zusammengehören. Eine solche zusammenhängende Region wird als BLOB bezeichnet. Hierbei wird überprüft, ob die Vordergrundpixel zusammenhängen, und wenn dies der Fall ist, werden sie zu einem BLOB zusammengefügt. Diese BLOBs können dann gekennzeichnet werden. Die Bounding Box, das kleinste Rechteck das die Region umschließt, wird verwendet um, den Widerstand auszuschneiden.

Abbildung 7.11 zeigt das Bild mit den gefundenen Bounding Boxen. Die rote Bounding Box stellt den Widerstand dar, und die blaue Bounding Box repräsentiert einen Teil des Widerstandsdrahts.

Im Idealfall ist das BLOB mit der größten Fläche der Widerstand, dies ist aber nicht immer der Fall. Deswegen sollten für die Robustheit auch statistische Merkmale wie

Schwerpunkt, Orientierung oder die Exzentrizität betrachtet werden. Die Exzentrizität ist ein sinnvolles Merkmal zur Beurteilung, ob es sich um einen Widerstand handelt. Sie gibt das Verhältnis von Länge zu Breite eines Objekts wieder.



(a) Binärbild (b) Bild mit Bounding Boxen

Abbildung 7.11: Bounding Boxen

Bevor der Bildausschnitt des Widerstandes weiterverarbeitet wird, muss auch die Ausrichtung überprüft werden. Hierbei helfen erneut statistische Merkmale. Besonders hilfreich ist die Orientierung, die die Richtung des Objekts angibt. Dieser Prozess stellt sicher, dass der ausgeschnittene Bildbereich des Widerstands korrekt ausgerichtet ist und zur weiteren Verarbeitung bereitsteht. Mehr zu statistischen Merkmalen in Kapitel 2.2.5. Auch die Umwandlung des Bildes in einen anderen Farbraum kann die Widerstandssegmentierung erleichtern. Mögliche Farbräume hierfür sind der LAB-Farbraum oder der HSV-Farbraum. Der LAB-Farbraum bietet beispielsweise den Vorteil, Informationen zur Farbintensität, Farbton und Helligkeit zu liefern. Diese Informationen können eine präzise Identifikation des Widerstands ermöglichen. Weitere Einzelheiten zum LAB-Farbraum finden sich in Abschnitt 2.2.6.

7.2 Umsetzung

Die Programmierung erfolgt, wie bereits erwähnt, in der Programmiersprache C++ innerhalb der Entwicklungsumgebung Visual Studio 2022. Dabei werden zwei separate Programme entwickelt:

1. Ein Programm für die Entwicklungsphase, das die aufgenommenen Widerstände einliest, erkennt und die Bildausschnitte der Widerstände speichert. Mit diesem Programm entstehen die Trainings- und Testdaten für das neuronale Netz.

2. Ein weiteres Programm für die eigentliche Anwendung, das das Webcam-Bild einliest, den Widerstand erkennt und klassifiziert. Dies wird im Kapitel 9 erläutert.

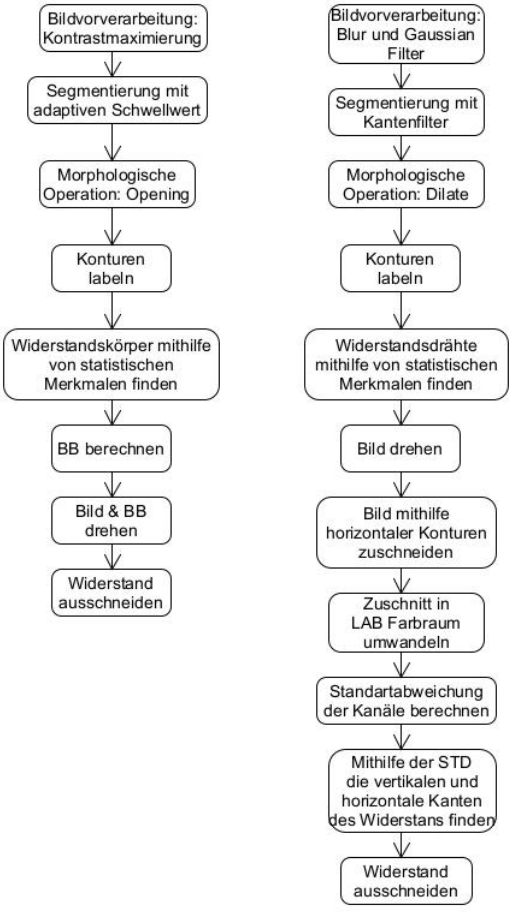


Abbildung 7.12: Ablauf Bildverarbeitungskonzepte

Zur Durchführung eines Vergleichs werden zwei Ansätze getestet, die jeweils unterschiedliche Konzepte für die Bildverarbeitung verwenden. In Abbildung 7.12 ist der Ablauf beider Konzepte dargestellt.

Im Folgenden werden die Programmabläufe der beiden Ansätze erläutert. Zunächst werden erforderliche Bibliotheken wie OpenCV eingebunden. Anschließend erfolgt die Deklaration von Konstanten, Funktionsprototypen, globalen Variablen usw. Die Hauptfunktion *main* wird daraufhin definiert und im Folgenden näher erläutert.

Funktion Main

Zu Beginn werden innerhalb einer for-Schleife die Bilder der Widerstände eingelesen. Die Dateipfade der einzulesenden Bilder, Referenzrahmen (Bounding Boxes) und die Ausschnitte der Widerstände werden durch die Verwendung der *append*-Funktion erstellt. Danach wird das Bild anhand des Dateipfades geladen, und es wird überprüft, ob es leer ist. Falls das Bild nicht leer ist, wird die entsprechende Bounding Box aus einer Textdatei gelesen und in einem Array gespeichert. Anschließend erfolgt bei Bedarf eine Skalierung des Bildes. Listing A.1 zeigt den Codeausschnitt der beschriebenen Schritte.

Wenn *ADAPT* definiert ist, wird der erste Ansatz mit adaptivem Schwellenwert verwendet (Schwellwert Ansatz). Falls dies nicht der Fall ist, erfolgt eine Kantendetektion mit dem Canny-Algorithmus (kantenbasierter Ansatz). Abbildung A.1 veranschaulicht den Ablauf der *main*-Funktion.

7.2.1 Programm Schwellwert Ansatz

Beim ersten Ansatz erfolgt nach der Skalierung eine Kontrastmaximierung. Der Kontrast des Bildes wird mithilfe der Funktion *maximizeContrast* erhöht. Danach wird das Grauwertbild durch Anwendung eines adaptiven Schwellenwerts in ein Binärbild umgewandelt. Neben dem ein- und Ausgangsbild und dem maximalen Pixelwert von 255, werden der Funktion *adaptiveThreshold* vier weitere Parameter übergeben. Als Methode kann eine gewichtete Gauß-Summe oder der Mittelwert übergeben werden, wobei die gewichtete Gauß-Summe hier bessere Ergebnisse erzielt hat. Der Schwellwerttyp gibt an, ob Pixel, die über dem Schwellwert liegen, den Wert 0 oder 255 erhalten. Beim adaptiven Schwellwert werden Nachbarpixel berücksichtigt, wofür eine Blockgröße von 7x7 übergeben wird. Außerdem kann ein Parameter C übergeben werden, der von der gewichteten

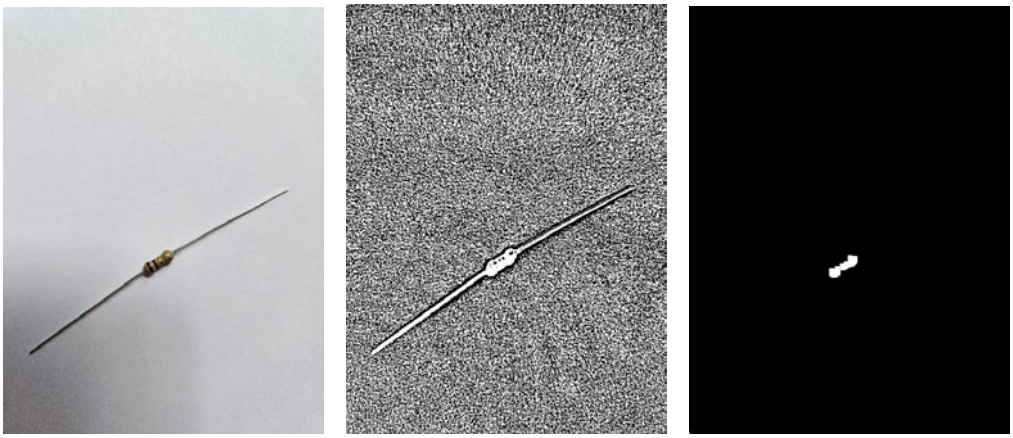
Summe subtrahiert wird. In dieser Anwendung beträgt C=0. Die mathematische Grundlage für diese Methoden wird im Kapitel 2.2 näher erläutert. Durch die Verwendung der OpenCV-Funktion *morphologyEx* mit dem Parameter *MORPH_OPEN* wird das „Opening“ auf das Binärbild angewendet. Abbildung 7.13 veranschaulicht die Anwendung des adaptiven Schwellenwerts und das anschließende „Opening“ auf das Binärbild. Der entsprechende Code ist im Listing 7.1 zu finden.

Listing 7.1: Codeausschnitt Main Funktion adaptiver Treshhold

```
//------------------adaptive Treshhold--------------------------------
    #ifdef ADAPT
    adaptiveThreshold(img, binImg, 255, ADAPTIVE_THRESH_MEAN_C, IS_INVERT_BINARY,
    7, 0);
    Mat structure = getStructuringElement(MORPH_RECT, Size(MORPH_SIZE, MORPH_SIZE)
    );
    erode(binImg, contrast, structure);
    double number_of_white_pix = sum(binImg == 255)[0] / binImg.total();
    //proof if there are not to much white pixel
    if (number_of_white_pix < 142) {
      maximizeContrast(img, contrast_max, 5);
      adaptiveThreshold(contrast_max, bin_final, 255, ADAPTIVE_THRESH_GAUSSIAN_C,
    IS_INVERT_BINARY, 7, 0);
      structure = getStructuringElement(MORPH_ELLIPSE, Size(5, 5));
    }
    else {
      maximizeContrast(img, contrast_max, 0.9);
      adaptiveThreshold(contrast_max, bin_final, 255, ADAPTIVE_THRESH_GAUSSIAN_C,
    IS_INVERT_BINARY, 7, 0);
      structure = getStructuringElement(MORPH_RECT, Size(MORPH_SIZE, MORPH_SIZE));
    }

    morphologyEx(bin_final, binErode, MORPH_OPEN, structure);
    found = findObject(image, binErode, box_ref);
    #endif
    if (found == 1){
      imwrite(outputImagePath, image);
      right_res++;
    }
    counter++;
  }
  cout << "Genauigkeit: " << (right_res / counter) * 100 << "%" << endl;
  cv::waitKey(0);
  return 0;
```

Im Anschluss daran wird die Funktion *findObject* aufgerufen, die im Abschnitt 7.2.1 näher beschrieben wird. Falls der Rückgabewert dieser Funktion 1 ist, bedeutet das, dass der Widerstand gefunden wurde. In diesem Fall wird ein Zähler für die Genauigkeit erhöht.



(a) Originales Bild (b) Binärbild (c) Binärbild mit Opening

Abbildung 7.13: Binärbilder Schwellwert Ansatz

Zusätzlich wird der Bildausschnitt des erkannten Widerstands mithilfe der OpenCV-Funktion *imwrite* im Verzeichnis für Trainings- oder Testdaten gespeichert. Auf diese Weise werden Trainings- und Testdaten für das neuronale Netz generiert.

Funktion findObject

Die Funktion *findObjekt* befindet sich in der C++-Datei „Resistor_Detection“. Sie überprüft, ob die gefundenen Konturen im übergebenen Binärbild mithilfe der Referenz-Bounding-Boxen einen Widerstand darstellen.

Zuerst werden benötigte Variablen deklariert. Mithilfe der OpenCV-Funktion *findContours* werden Konturen im übergebenen Binärbild gefunden und gespeichert. In dieser Funktion werden das Binärbild, ein leeres Array zur Speicherung der Konturen als Vektor von Punkten, der Konturabrufmodus und die Angabe, welche Konturpunkte gespeichert werden sollen, übergeben. Der Abrufmodus wird auf den Parameter „RETR_TREE“ gesetzt, was bedeutet, dass alle Konturlinien abgerufen werden und eine vollständige Hierarchie verschachtelter Konturen rekonstruiert wird. Durch den Parameter „CHAIN_APPROX_NONE“ werden alle Punkte der Konturen gespeichert.

Anschließend wird die selbst geschriebene Funktion *identifyResistor* aufgerufen, um das Konturlabel der Widerstandskontur zu ermitteln. Die Übergabeparameter sind das Array mit den Konturen, drei leere Arrays für die Momente, die Exzentrizitäten und die Orientierungen sowie das binäre Bild. Der Rückgabewert, der das Label der Kontur darstellt,

wird gespeichert. Das erhaltene Label wird überprüft. Falls es den Wert -1 hat, wurde kein Widerstand gefunden. Andernfalls handelt es sich um eine potenzielle Widerstandskontur.

Wenn ein potenzieller Widerstand gefunden wurde, wird die Bounding Box dieser Kontur mithilfe der OpenCV-Funktion *BoundingRect* ermittelt. Diese Funktion ermittelt die Bounding Box der übergebenen Kontur und gibt ein Rechteck mit den Koordinaten x_0, y_0, der Breite und der Höhe zurück. Die Parameter (Position(x_0, y_0), Breite, Höhe) der Bounding Box werden anschließend proportional zur Bildgröße berechnet und gespeichert. Die folgende Formel wird für x_0 verwendet, wobei C die Bildspalten darstellt.

$$x_{0,prop} = x_0 \cdot \frac{100}{C} \tag{7.1}$$

Die anderen Größen y_0, *width* und *height* werden äquivalent berechnet. Das Listing 7.2 zeigt den Code zu den beschriebenen Schritten.

Listing 7.2: Codeausschnitt findObject Funktion 1

```
1  findContours(bin, contours, RETR_TREE, CHAIN_APPROX_NONE);
2  //Get the moments
3  vector<Moments> mu(contours.size());
4  vector<double> exz(contours.size());
5  vector<float> ori(contours.size());
6  //proof if label match resistor contur
7  biggest_label = identifyResistor(contours, mu, exz, ori, bin);
8  if (biggest_label != -1) {
9    //Creating a Bounding Box of the Found Contur
10   Rect box = boundingRect(contours[biggest_label]);
11   x = box.x;
12   y = box.y;
13   height = box.height;
14   width = box.width;
15   //Calculate parameters proportional to size
16   x_box = x / (imag.cols / 100);
17   y_box = y / (imag.rows / 100);
18   height_box = height / (imag.rows / 100);
19   width_box = width / (imag.cols / 100);
```

Die Bounding Box wird mit der Referenz-Bounding-Box des Widerstands verglichen. Dabei wird berücksichtigt, dass die gefundenen Bounding Boxen nicht exakt mit der Referenz-Bounding Box übereinstimmen müssen, sondern Toleranzen aufweisen können. Für x_0 und y_0 hat sich eine Toleranz von $\frac{1}{30}$ der Bildspalten gut bewährt. Für die Breite und Höhe wird eine minimale Toleranz von $\frac{1}{75}$ der Bildspalten und maximale Toleranz

von $\frac{1}{45}$ der Bildspaten angewendet. Diese Toleranzen haben die besten Ergebnisse erzielt, sodass die meisten Widerstände ausgeschnitten wurden, und gleichzeitig keine unzureichenden Widerstandsausschnitte fälschlicherweise als richtig klassifiziert wurden.

Falls die ermittelte Bounding Box innerhalb der Toleranzgrenzen der Referenz-Bounding-Box liegt, werden der Schwerpunkt in x- und y-Richtung sowie die Orientierung ausgelesen. Abbildung 7.14 zeigt die Referenz-Bounding Box in Grün und die gefundene Bounding Box in Rot.

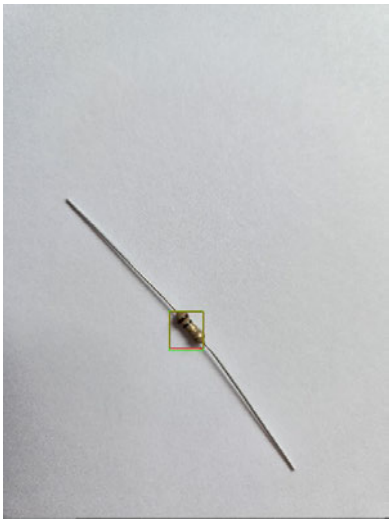


Abbildung 7.14: Bounding Boxen des Widerstands

Es sind leichte Unterschiede zwischen den beiden Boxen zu erkennen, aber beide umschließen den Widerstand ausreichend. Die Toleranzen sind wie folgt aufgeführt (h=Höhe, w=Weite, C=Bildspalten).

$$x_{0,ref} - \frac{C}{30} \leq x_0 \leq x_{0,ref} + \frac{C}{30} \tag{7.2}$$

$$y_{0,ref} - \frac{C}{30} \leq y_0 \leq y_{0,ref} + \frac{C}{30} \tag{7.3}$$

$$h_{0,ref} - \frac{C}{75} \leq h \leq h_{0,ref} + \frac{C}{45} \tag{7.4}$$

$$w_{0,ref} - \frac{C}{75} \leq w \leq w_{0,ref} + \frac{C}{45} \tag{7.5}$$

Diese Toleranzen wurden so gewählt, dass sie eine ausreichende Abdeckung des Widerstands ermöglichen, während gleichzeitig unerwünschte Ergebnisse minimiert werden.

Danach wird der Winkel der Orientierung überprüft. Wenn dieser größer als 90° ist, wird die Funktion *getRotationMatrix2D* mit dem Parameter -1 aufgerufen, um das Bild gegen den Uhrzeigersinn zu drehen. Wenn der Winkel kleiner ist, wird der Parameter 1 verwendet, und das Bild wird im Uhrzeigersinn gedreht. Diese Funktion verformt das Pixelraster und bildet es auf das Zielbild ab. Die resultierende Rotationsmatrix wird dann der OpenCV-Funktion *warpAffine* übergeben, um das Bild entsprechend der Rotationsmatrix zu drehen. Listing 7.3 zeigt den Code der beschriebenen Schritte.

Listing 7.3: Codeausschnitt findObject Funktion 2

```
1  if ((x_box <= box_ref[1] + tolerance_min_x_y && x_box >= box_ref[1] -
       tolerance_min_x_y) && (width_box <= box_ref[3] + tolerance_heigth_width_max &&
        width_box >= box_ref[3] - tolerance_heigth_width_min) && (y_box <= box_ref[2]
        + tolerance_min_x_y && y_box >= box_ref[2] - tolerance_min_x_y) && (
       height_box <= box_ref[4] + tolerance_heigth_width_max && height_box >= box_ref
       [4] - tolerance_heigth_width_min)) {
2    //calculate centroid of contur
3    cogx = (mu[biggest_label].m10 / mu[biggest_label].m00);
4    cogy = (mu[biggest_label].m01 / mu[biggest_label].m00);
5    angle = abs(90 - ori[biggest_label]);
6    //calculate rotation Matrix
7    if (ori[biggest_label] > 90)
8      rot = getRotationMatrix2D(Point2f(cogx, cogy), angle * (-1), 1);
9    else
10     rot = getRotationMatrix2D(Point2f(cogx, cogy), angle, 1);
11   //calculate new Bounding Box Parameter
12   cv::RotatedRect rotatedRectangle(Point2f(cogx, cogy), Size2f(box.width, box.
       height), angle);
13   Point2f vertices[4];
14   rotatedRectangle.points(vertices);
15   Rect rotate_rect = rotatedRectangle.boundingRect();
16   //rotate Picture
17   warpAffine(imag, rotate, rot, imag.size());
18   warpAffine(bin, bin_rotate, rot, bin.size());
```

Das Ergebnis ist ein gedrehtes Bild, bei dem der Widerstand horizontal richtig ausgerichtet ist. Abbildung 7.15 zeigt das Ergebnis dieser Funktion.

Nachdem das Bild korrekt gedreht wurde, ist es notwendig, die Bounding Box zu drehen. Dafür müssen der wert y_0 und die Höhe der Bounding Box neu berechnet werden.

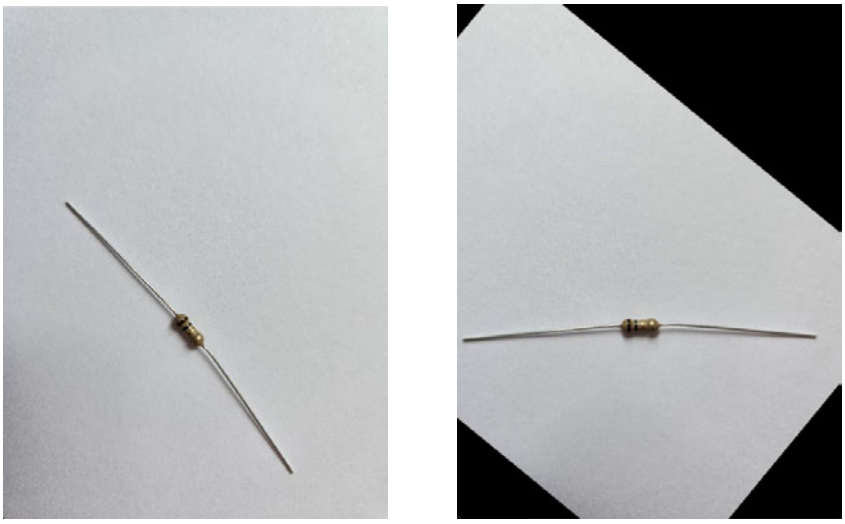
$$y_0 = S_y - \frac{C}{37} \tag{7.6}$$

Die Höhe wird auf einen Wert von $\frac{1}{27}$ der Bildreihen gesetzt. Diese Werte haben sich als am besten für den Widerstandsausschnitt erwiesen. Anschließend wird mithilfe der Bounding Box der Widerstand aus dem Bild ausgeschnitten. Dies wird erreicht, indem das Originalbild auf die Größe der ermittelten Bounding Box zugeschnitten wird. Wenn die Detektion erfolgreich war, wird eine 1 zurückgegeben, ansonsten eine 0. Listing 7.4 zeigt den zugehörigen Codeausschnitt.

Listing 7.4: Codeausschnitt findObject Funktion 3

```
cv::Rect roi;
roi.x = rotate_rect.x;
roi.y = cogy - imag.cols/37;
roi.width = rotate_rect.width;
roi.height = imag.rows/27;
bin_rotate = bin_rotate(roi);

imag = rotate(roi);
return 1;
```



(a) Originales Bild (b) Rotationsbild

Abbildung 7.15: Rotationsbilder Schwellwert Ansatz

Abbildung 7.16 zeigt den erfolgreichen Ausschnitt des Widerstands, bei dem der Algorithmus präzise die relevanten Teile des Widerstands erfasst hat. Dieser sorgfältige Ausschnitt des Widerstands ist ein entscheidender Schritt zur Gewährleistung einer zuverlässigen Erkennung und Klassifizierung in der weiteren Verarbeitung des Bildmaterials.

Abbildung 7.16: Widerstandsausschnitt Schwellwert Ansatz

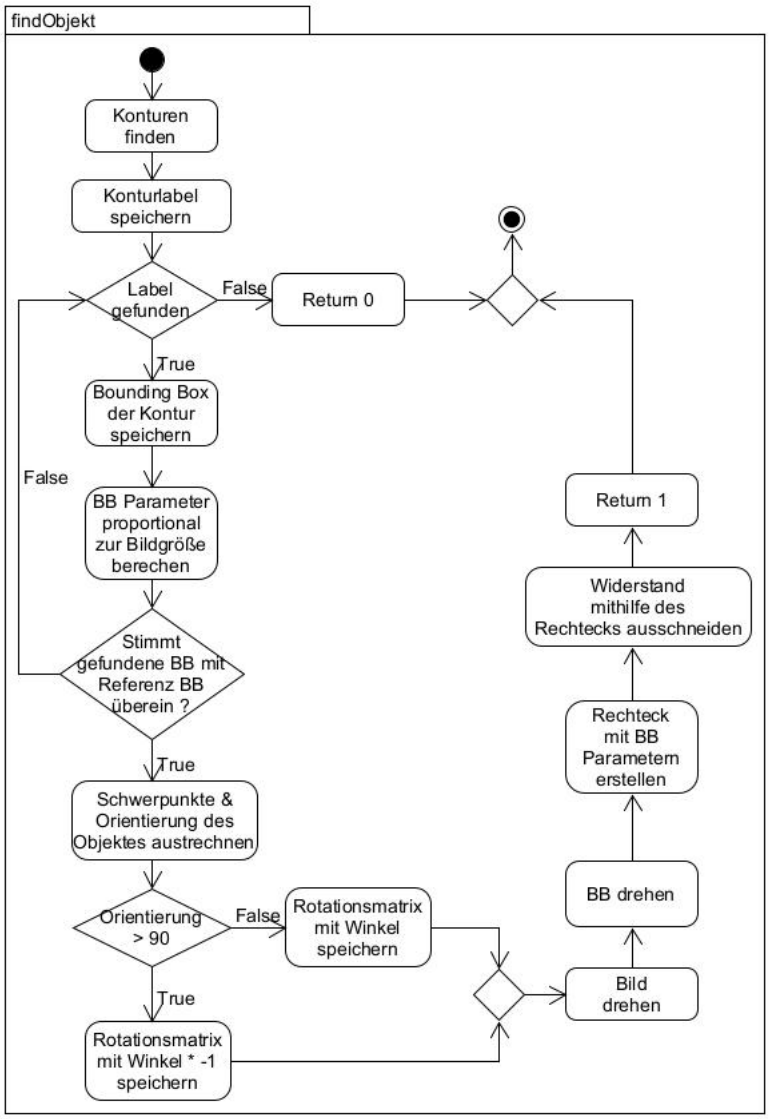


Abbildung 7.17: Ablauf findObject Funktion

Abbildung 7.17 zeigt den Ablauf der beschriebenen *findObject* Funktion, die im Algorithmus verwendet wird. Diese Funktion ist ein zentraler Bestandteil des Bildverarbei-

tungsprozesses und spielt eine wichtige Rolle bei der Identifizierung und Extraktion von Widerständen in den Bildern. Sie trägt dazu bei, die genaue Position und Struktur des Widerstands zu bestimmen, indem sie die erkannten Konturen und Muster analysiert. Die Schritte, die in dieser Funktion dargestellt sind, sind von entscheidender Bedeutung für die Gesamtleistung des Algorithmus bei der Erkennung und dem Ausschnitt der Widerstände in den Bildern.

Funktion identifyResistor

Die *identifyResistor* befindet sich ebenfalls in der C++ Datei „Resistor_Detektion“. Die Funktion prüft die gefundenen Konturen auf verschiedene Merkmale.

In einer Schleife werden alle gefundenen Konturen durchlaufen, und ihre Momente werden mithilfe der OpenCV-Funktion *moments* berechnet und in ein Array gespeichert. Die Formeln hierfür finden sich im Abschnitt 2.2.5. Nachdem die Schleife durchlaufen wurde, wird in einer zweiten Schleife iteriert. In dieser Schleife werden mithilfe der Momente die Schwerpunkte, die Orientierung und die Exzentrizitäten berechnet und in Arrays gespeichert. Die Formeln hierfür sind ebenfalls im Abschnitt 2.2.5 zu finden.

In einer if-Bedingung wird geprüft, ob die aktuelle Kontur die Eigenschaften eines Widerstands aufweist. Es wird geprüft, ob die aktuelle Größe der Kontur, größer ist als die vorherige, und die Exzentrizität zwischen 0,3 und 0,87 liegt. Die Exzentrizität gibt die Relation der Länge eines Objekts zu dessen Breite an. Wenn dieser Wert 0 beträgt, ist das Objekt rund, bei einem Wert von 1 handelt es sich um eine Linie. Im Fall eines Widerstandskörpers muss also ein Bereich angegeben werden. Ein Bereich von 0,3 bis 0,87 hat sich als beste Lösung erwiesen.

Es ist außerdem aufgefallen, dass sich am Rand des Bilds oft Störungen befinden. Dadurch wird die Kontur einer Störung oft als Widerstand klassifiziert. Um dies zu vermeiden wurde zusätzlich zur Überprüfung der Exzentrizität die Prüfung der Schwerpunkte des Objektes in die if-Bedingung aufgenommen. Befindet sich das Objekt außerhalb der folgenden Bereiche, wird es verworfen. Im Folgenden sind die Toleranzen aufgeführt (R=Bildzeilenanzahl).

$$C - \frac{C}{9} \leq S_x \leq \frac{C}{9} \tag{7.7}$$

$$R - \frac{C}{9} \leq S_y \leq \frac{C}{9} \tag{7.8}$$

Diese Toleranzen helfen, unerwünschte Störungen oder Randeffekte zu reduzieren und die Zuverlässigkeit der Widerstandserkennung zu verbessern.

Listing 7.5: Code identifyResistor Funktion

```
int identifyResistor(std::vector<std::vector<cv::Point> > contours, vector<Moments
    >& mu, vector<double>& exz, vector<float>& ori, Mat& binImg) {

  int biggest = 0;
  int biggest_label = -1;
  double a;
  double b;
  vector<double> cogx(contours.size());
  vector<double> cogy(contours.size());
  int x = 0;
  int y = 0;
  int width = 0;
  int height = 0;
  //calculate moments for the conturs
  for (int i = 0; i < contours.size(); i++)
  {
    mu[i] = moments(contours[i], false);
    a = mu[i].mu20 - mu[i].mu02;
    b = mu[i].mu20 + mu[i].mu02;
    exz[i] = (pow(a, 2) + (4 * pow(mu[i].mu11, 2))) / (pow(b, 2));
    ori[i] = ((0.5 * atan2(mu[i].mu20 - mu[i].mu02, 2 * mu[i].mu11)) * (180 / M_PI
    )) + 45;
    cogx[i] = (mu[i].m10 / mu[i].m00);
    cogy[i] = (mu[i].m01 / mu[i].m00);
  }
  int rows = binImg.rows;
  int cols = binImg.cols;
  int tol_edge = cols/9;

  for (int label = 0; label < contours.size(); label++) {
    //poof if the contur could be a resistor
    if (contours[label].size() > biggest && exz[label] > 0.30 && exz[label] < 0.87
     && cogx[label] < (cols - tol_edge) && cogx[label] > (tol_edge) && cogy[label]
     < (rows - tol_edge) && cogy[label] > (tol_edge)) {
      //save the contur label
      biggest = contours[label].size();
      biggest_label = label;
    }
  }
  return biggest_label;
}
```

Wenn die Struktur die erforderlichen Bedingungen erfüllt, wird das Label gespeichert, und die Schleife wird fortgesetzt. Nach Abschluss der Schleifen wird das vielversprechendste

Label als Rückgabewert zurückgegeben. Wenn kein passendes Label gefunden wurde, wird -1 zurückgegeben. Listing 7.5 zeigt den Code der Funktion.

Abbildung 7.18 veranschaulicht den Ablauf der Funktion. Auch diese Funktion spielt eine essenzielle Rolle im Verarbeitungsprozess der Bilddaten. Sie ist dafür zuständig die richtige Widerstandskontur zu finden. Der in Abbildung 7.18 gezeigte Ablauf verdeutlicht die Schritte und Prozesse, die in dieser Funktion ausgeführt werden, um die gewünschten Ziele zu erreichen. Die korrekte Umsetzung und Ausführung dieser Funktionen sind von grundlegender Bedeutung, um die Gesamtleistung des Algorithmus sicherzustellen.

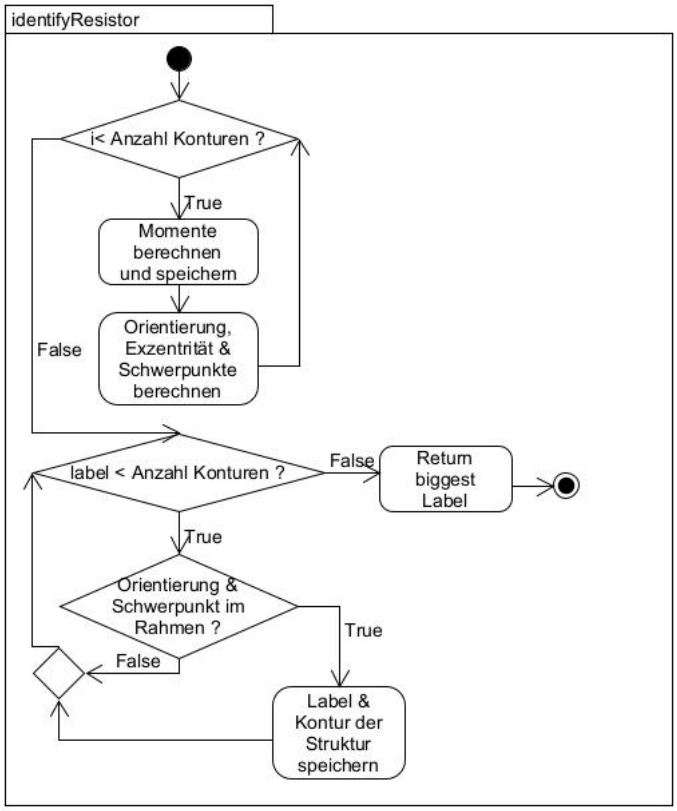


Abbildung 7.18: Ablauf identifyResistor Funktion

7.2.2 Programm kantenbasierter Ansatz

Im zweiten Ansatz erfolgt nach der Skalierung der Bilder eine Glättung zur Rauschreduktion. Zuerst wird ein Median-Filter und anschließend ein Gauß'schen Weichzeichner

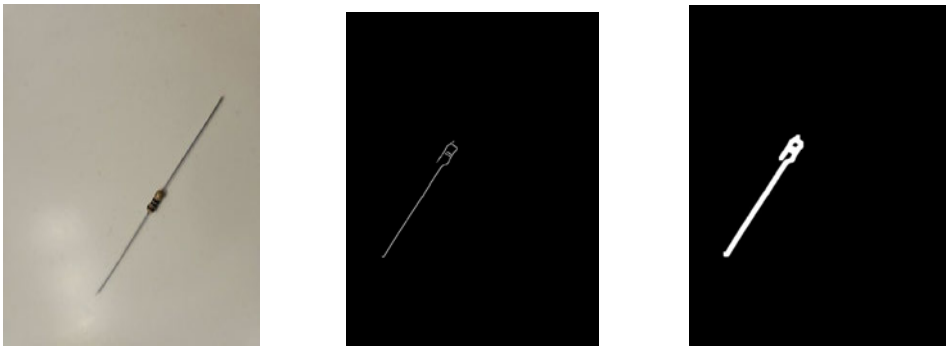
angewendet. Besonders bei unruhigem Hintergrund sind Glättungen hilfreich. Die Anwendung von zwei Filtern hintereinander hat die Genauigkeit des Algorithmus erhöht. Weitere Informationen zu Filtern finden sich in Abschnitt 2.2.7.

Anschließend wird die Anzahl der weißen Pixel berechnet und geprüft, ob die Anzahl zwischen 0,4 und 0,81 liegt. Dieser Bereich hat sich als beste Lösung für die Segmentierung von Widerstand und Hintergrund herausgestellt. Diese Bedingung wird in einer while Schleife geprüft. Ist die Bedingung nicht erfüllt, wird die OpenCV Funktion *canny* aufgerufen, dieser Funktion werden das Eingangs- und Ausgangsbild, sowie die beiden Schwellwerte übergeben. Der Anfangsschwellwert beträgt 700, und der zweite Schwellwert ergibt sich durch Division des ersten Schwellwerts durch zwei, da dies die besten Ergebnisse erzielt hat. Anschließend wird das Verhältnis der Anzahl der weißen Pixel wie folgt berechnet (P=Pixel).

$$P_{verhaeltnis} = \frac{\sum_{n=0}^{N} P}{N} \tag{7.9}$$

Diese Bedingung und die Anwendung von Canny-Filtern dienen dazu, die Widerstandserkennung zu verfeinern und sicherzustellen, dass das gefundene Objekt ausreichend von seinem Hintergrund abgegrenzt ist.

Im Anschluss wird der Schwellwert um 10 verringert. Wenn die Bedingung der while Schleife erfüllt ist, wird die morphologische Operation *dilate* angewendet, um die Drahtstrukturen der Widerstände zu vergrößern. Anschließend wird die Funktion *findObject_canny* aufgerufen, diese gibt bei erfolgreicher Detektion eine 1 zurück. Die nachfolgenden Schritte zur Berechnung der Genauigkeit und zum Ausschneiden des Widerstandsbereichs entsprechen denen des ersten Ansatzes. Abbildung 7.19 zeigt die Umwandlung des Bildes. Nachfolgend ist der Code im Listing 7.6 zu finden.



(a) Originales Bild (b) Binärbild (c) Binärbild mit Dilatation

Abbildung 7.19: Binärbilder kantenbasierter Ansatz

Listing 7.6: Codeausschnitt Main Funktion Canny Algorithmus

```
//------Edge Filter-----------------------------------------
  #ifndef ADAPT
  int low_tresh = 700;
  int ratio = 3;
  double number_of_white_pix = 8;

  GaussianBlur(img, blur, Size(3,3), BORDER_DEFAULT);
  Mat structure = getStructuringElement(MORPH_RECT, Size(MORPH_SIZE, MORPH_SIZE));
  //proof if there is the right number of white pixel
  while ((number_of_white_pix >= 0.81 || number_of_white_pix<0.4) ) {
    Canny(img, canny_edges, low_tresh, low_tresh / ratio);
    number_of_white_pix = sum(canny_edges == 255)[0] / canny_edges.total();
    low_tresh -= 10;
    if (low_tresh <= 0){
      break;
    }
  }
  dilate(canny_edges, binErode, structure);
  found = findObject_canny(image, binErode,box_ref);
  #endif
//-----------------------------------------------------------
  if (found == 1){
    imwrite(outputImagePath, image);
    right_res++;
  }
  counter++;
}
cout << "Genauigkeit: " << (right_res / counter) * 100 << "%" << endl;
cv::waitKey(0);
return 0;
```

Funktion findObject_canny

Die Funktion *findObject_ canny* befindet sich ebenfalls in der C++-Datei „Resistor_Detection“. Diese Funktion überprüft, ob die im übergebenen Binärbild gefundenen Konturen tatsächlich den Draht des Widerstands anhand von Referenz-Bounding-Boxen repräsentieren. Der Ablauf ist ähnlich wie in der *findObject*-Funktion. Zunächst werden Konturen im Bild gesucht. Anschließend werden diese mit der Funktion *identifyResistorWire* überprüft, die das Label des Drahtes zurückgibt.

Danach wird das Bild mithilfe des Schwerpunktes und der Orientierung gedreht. Dieser Vorgang ist in der Funktion *findObject* bereits beschrieben. Beim zweiten Ansatz erfolgt eine Drehung des Bildes um seinen Mittelpunkt, während im ersten Ansatz die Drehung um den Schwerpunkt des erkannten Widerstandes erfolgt. Dies führt dazu, dass der Widerstand bei einer Drehung an eine andere Position gelangt, was besonders deutlich bei Objekten am Rand des Bildes sichtbar wird. Aufgrund dieser Verschiebung des Objektes sind die Parameter x_0 und y_0 nicht mehr direkt mit den Werten der Referenzbox vergleichbar und müssen neu berechnet werden. Abbildung 7.20 veranschaulicht dieses Problem.

(a) Original (b) Rotationsbild

Abbildung 7.20: Rotationsbild kantenbasierter Ansatz

Um die neue Position der Referenzbox zu bestimmen, wird die Rotationsmatrix der *getRotationMatrix2D* Funktion verwendet. Folgende Matrix gibt die Funktion zurück.

$$M = \begin{bmatrix} \alpha & \beta & (1-\alpha) \cdot S_x - \beta \cdot S_y \\ -\beta & \alpha & \beta \cdot S_x + (1-\alpha) \cdot S_y \end{bmatrix}$$

Wobei α und β wie folgt definiert sind. Der Parameter s steht für die Skalierung, in dieser Anwendung beträgt er 1, hat also keine Auswirkungen.

$$\alpha = s \cdot \cos(\theta) \tag{7.10}$$

$$\beta = s \cdot \sin(\theta) \tag{7.11}$$

Mithilfe dieser Formeln können die neuen Koordinaten berechnet werden [14].

$$x' = \alpha \cdot x + \beta \cdot y + (1 - \alpha) \cdot S_x - \beta \cdot S_y \tag{7.12}$$

$$y' = -\beta \cdot x + \alpha \cdot y + \beta \cdot S_x + (1 - \alpha) \cdot S_y \tag{7.13}$$

Der Codeausschnitt ist im Listing 7.7 zu sehen. Anschließend werden erneut Konturen im gedrehten Bild gesucht und ausgewertet. Durch das gedrehte Bild und die gefundenen Drähte kann der Widerstand an den horizontalen Linien schmaler zugeschnitten werden, damit die Berechnung der Standardabweichung präziser ist. Die Parameter y_0 und *height* des Ausschnitts können mithilfe der Drahtkontur und Angabe einer Toleranz (Listing 7.8 Z.10-11) bestimmt werden. Der Wert für x_0 wird zunächst auf 0 und für *width* auf die gesamte Länge des Bildes gesetzt. Abbildung 7.21 zeigt einen beispielhaften Widerstandsausschnitt.

Abbildung 7.21: Bildausschnitt an horizontalen Linien (kantenbasierter Ansatz)

Durch das Zuschneiden des Bildes in horizontale Richtung muss auch die y-Koordinate der Referenzbox neu berechnet werden. Im Listing 7.8 ist die Berechnung in Zeile 24 und 25 gezeigt.

Listing 7.7: Codeausschnitt findObject_canny 1

```
findContours(bin, contours, hierarchy, RETR_TREE, CHAIN_APPROX_NONE);
//Get the moments
vector<Moments> mu(contours.size());
vector<double> exz(contours.size());
vector<float> ori(contours.size());
//proof if label match resistor contur
biggest_label = identifyResistorWire(contours, mu, exz, ori, bin);
//if a label was found
if (biggest_label != -1) {
  //calculate centroid of picture
  cogx = imag.cols / 2;
  cogy = imag.rows / 2;
  angle = abs(90 - ori[biggest_label]);
  //get rotation Matrix
  if (ori[biggest_label] > 90){
    rot = getRotationMatrix2D(Point2f(cogx, cogy), angle * (-1), 1);
    angle = angle * (-1);
  }
  else {
    rot = getRotationMatrix2D(Point2f(cogx, cogy), angle, 1);
  }
  //rotate picture
  warpAffine(imag, rotate, rot, imag.size());
  warpAffine(bin, bin_rotate, rot, bin.size());
  //calculate new reference Bounding Box
  int x_pix = box_ref[1] * (orig.cols / 100.0);
  int y_pix = box_ref[2] * (orig.rows / 100.0);
  int width_pix = box_ref[3] * (orig.cols / 100.0);
  int height_pix = box_ref[4] * (orig.rows / 100.0);
  int x_new = abs(rot.at<double>(0, 0) * x_pix + rot.at<double>(0, 1) * y_pix +
    rot.at<double>(0, 2));
  int y_new = abs(rot.at<double>(1, 0) * x_pix + rot.at<double>(1, 1) * y_pix +
    rot.at<double>(1, 2)) ;
```

Listing 7.8: Codeausschnitt findObject_canny 2

```
findContours(bin_rotate, contours_rotate, hierarchy, RETR_TREE, CHAIN_APPROX_NONE);
vector<Moments> mu_rot(contours_rotate.size());
vector<double> exz_rot(contours_rotate.size());
vector<float> ori_rot(contours_rotate.size());
//proof if contur could be resistor wires
biggest_label = identifyResistorWire(contours_rotate, mu_rot, exz_rot, ori_rot, bin_rotate);

if (biggest_label != -1) {
  Rect box_rotate = boundingRect(contours_rotate[biggest_label]);
  y = box_rotate.y - 20;
  height = box_rotate.height + 50;

  try {
    y_box = (y / (imag.rows / 100.0));
    height_box = height / (imag.rows / 100.0);
    //parameter for crop picture horizontal
    cv::Rect roi;
    roi.x = 0;
    roi.y = y;
    roi.width = bin_rotate.cols;
    roi.height =  height;
    imag = rotate(roi);
    y_new =y_new-y;
    x_new = (x_new * (100.0 / imag.cols));
    y_new = (y_new * (100.0 / imag.rows)) -box_ref[4] / 2;
```

Anschließend wird der präzise Bildausschnitt entlang der horizontalen Linien erstellt. Dafür erfolgt die Konvertierung des Bildausschnitts in den LAB-Farbraum. Der LAB-Farbraum verfügt über drei Kanäle und basiert auf der Gegenfarbtheorie. Der L-Kanal repräsentiert die Helligkeitsachse, der A-Kanal die Rot-Grün-Achse und der B-Kanal die Blau-Gelb-Achse [21]. In Abbildung 7.22 ist das Bild in den drei Kanälen dargestellt. In allen drei Kanälen ist der Widerstand erkennbar, wobei er im L-Kanal und B-Kanal am deutlichsten zu erkennen ist.

Um den Widerstand zu segmentieren, wird die Farbabweichung σ_c berechnet. Das Bild wird mithilfe von zwei verschachtelten for-Schleifen Zeilen- und Spaltenweise durchlaufen. Dabei wird die Farbabweichung für alle Bildspalten berechnet. Wenn die Farbabweichung einen bestimmten Schwellenwert überschreitet, ist höchstwahrscheinlich eine Spalte zu finden, in der der Widerstand liegt.

(a) Kanal L

(b) Kanal A

(c) Kanal B

Abbildung 7.22: Konvertierung in LAB-Farbraum

Um die Farbabweichung zu berechnen, müssen zunächst die Summe, der Mittelwert $\overline{x}$, die Varianz σ^2 und die Standardabweichung σ der drei Farbkanäle ermittelt werden. Dabei beziehen sich die Pixelwerte immer auf eine bestimmte Spalte im Bild. Folgende Formeln werden verwendet.

$$\overline{x} = \frac{\sum_{n=0}^{N} P}{N} \tag{7.14}$$

$$\sigma^2 = \frac{\sum_{n=0}^{N} (P - \overline{x})^2}{N} \tag{7.15}$$

$$\sigma = \sqrt{\sigma^2} \tag{7.16}$$

Nachdem die stochastischen Maße für jeden Kanal ausgerechnet wurden, kann die Farbabweichung mithilfe der folgenden Formel ermittelt werden.

$$\sigma_c = k1 \cdot \sigma_L + k2 \cdot (\sigma_A + \sigma_B) \tag{7.17}$$

Mithilfe dieser Formel erhält man die Farbabweichung für jede einzelne Spalte. Die Werte $k1$ und $k2$ sind Skalierungsfaktoren und betragen 1 und 10, wie im Paper der Technischen Hochschule Zürich [17] angegeben, an dem sich in diesem Ansatz orientiert wurde. Die Abweichung wird anschließend geprüft. Wenn der Wert einen Schwellenwert überschreitet, sich nicht zu weit links im Bild befindet und die Variable *found* den Wert 0 hat, dann wurde voraussichtlich der Anfang des Widerstandskörpers gefunden und die aktuelle Spalte wird in einer Variable gespeichert. Die zweite if-Bedingung prüft, ob die Abweichung den Schwellenwert unterschreitet und die Variable *found* den Wert 1 hat.

In diesem Fall wird die aktuelle Spalte gespeichert, und die Schleife wird mithilfe eines „break“-Befehls verlassen. Der Schwellenwert beträgt 70, da sich dieser Wert in der Praxis bewährt hat. Wenn x_1 ungleich 0 ist, wird das Bild mithilfe von x_0 und x_1 ausgeschnitten. Abbildung 7.23 visualisiert die Abweichungen der Bildspalten. Eine hohe Abweichung bedeutet eine helle Farbe. Im Listing A.2 ist der Code zu finden.

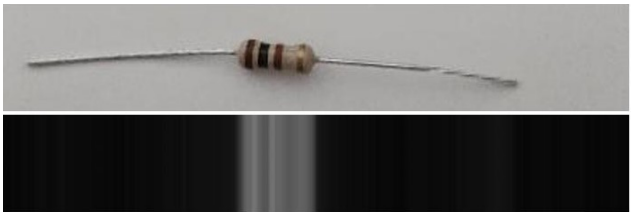


Abbildung 7.23: Standartabweichung Bildspalten

Anschließend wird wieder durch eine verschachtelte Schleife iteriert. Dabei werden die Formeln statt auf die Spalten auf die Reihen des Bildes angewandt. Es wird wieder die Abweichung ausgerechnet, wobei der Schwellwert in diesem Fall 40 beträgt. Mit diesen Werten können die Bounding-Box-Parameter y_0 und *height* festgelegt werden. Listing A.3 zeigt den beschriebenen Codeausschnitt.

Abbildung 7.24 visualisiert die Abweichungen der Bildzeilen.

Abbildung 7.24: Standartabweichungen Bildzeilen

Anschließend wird die gefundenen Bounding Box mit der Referenz Bounding Box verglichen und bei erfolgreichem Vergleich wird der Widerstand ausgeschnitten. Die Funktion gibt dann eine 1 als Rückgabewert zurück. Der Codeausschnitt ist im Listing 7.9 zu finden.

Die Abbildung 7.25 zeigt einen beispielhaften Ausschnitt des Widerstandes, welcher mithilfe des Algorithmus aus dem Gesamtbild ausgeschnitten wurde.

Abbildung 7.25: Widerstandsausschnitt kantenbasierter Ansatz

Abbildung A.2 zeigt den Ablauf der findObject_canny Funktion.

Listing 7.9: Codeausschnitt findObject_canny 5

```
1  //calculate Bounding Box for resistor
2  roi.x = x_0;
3  roi.y = y_0;
4  roi.width = x_1 - x_0;
5  roi.height = y_1 - y_0;
6  //proof if the found Bounding Box match with refernce Bounding Box
7  if (y_1 != 0 &&(x_box <= x_new + tolerance_min_x && x_box >= x_new -
       tolerance_min_x) && (y_box <= y_new + tolerance_min_y && y_box >= y_new-
       tolerance_min_y) && (width_box <= 5 + tolerance_heigth_width_max && width_box
      >= 5 - tolerance_heigth_width_min) && (height_box >= box_ref[4]&&height_box<=
       60))
8  {
9    imag = imag(roi);
10   return 1;
11 }
```

Funktion identifyResistorWire

Die Funktion *findObject_canny* soll im Gegensatz zur *identifyResistor*-Funktion den Draht der Widerstände erkennen und nicht den Körper des Widerstands. Daher ändert sich lediglich die Bedingung, unter der angenommen wird, dass eine Struktur wahrscheinlich den Draht des Widerstands darstellt. In diesem Fall wird die Exzentrizität überprüft, und wenn sie über 0,93 liegt, handelt es sich sehr wahrscheinlich um den Draht des Widerstands. Da der Draht des Widerstandes eine Linie bilden sollte und somit bei fast 1 liegt, hat sich hier ein Wert von 0,93 für die Exzentrizität als beste Lösung herausgestellt. Der Code der Funktion ist im Listing 7.10 zu finden.

Abbildung 7.26 veranschaulicht den Ablauf der Funktion, die im Algorithmus verwendet wird. Diese Funktion ist speziell darauf ausgerichtet, zu überprüfen, ob die übergebene Kontur tatsächlich den Widerstandsdraht darstellt, und entsprechend darauf zu reagieren. Ihr dargestellter Ablauf verdeutlicht die Schritte und Entscheidungsprozesse, die in dieser Funktion durchgeführt werden, um sicherzustellen, dass nur relevante Konturen als Widerstandsdraht erkannt werden.

Listing 7.10: identifyResistorWire Funktion

```
int identifyResistorWire(std::vector<std::vector<cv::Point> > contours, vector<
    Moments>& mu, vector<double>& exz, vector<float>& ori, Mat& binImg) {

  int biggest_label = -1;
  double a;
  double b;
  vector<double> cogx(contours.size());
  vector<double> cogy(contours.size());
  //calculate moments for conturs
  for (int i = 0; i < contours.size(); i++)
  {
    mu[i] = moments(contours[i], false);
    a = mu[i].mu20 - mu[i].mu02;
    b = mu[i].mu20 + mu[i].mu02;
    exz[i] = (pow(a, 2) + (4 * pow(mu[i].mu11, 2))) / (pow(b, 2));
    ori[i] = ((0.5 * atan2(mu[i].mu20 - mu[i].mu02, 2 * mu[i].mu11)) * (180 / M_PI
    )) + 45;
    cogx[i] = (mu[i].m10 / mu[i].m00);
    cogy[i] = (mu[i].m01 / mu[i].m00);
  }
  //proof if contur could be resistor wire
  for (int label = 0; label < contours.size(); label++) {
    //if yes save label and return it
    if (exz[label] > 0.93) {
      biggest_label = label;
    }
  }
  return biggest_label;
}
```

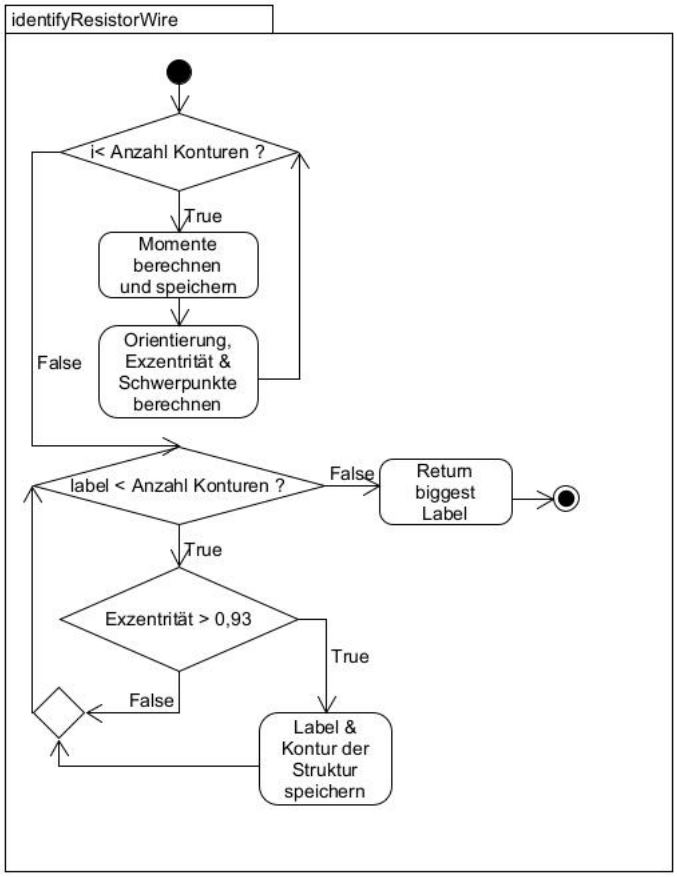


Abbildung 7.26: Ablauf identifyResistorWire Funktion

7.3 Auswertung

Die Trainingsdaten werden durch die korrekte Angabe der Bildpfade geladen. Jeder Datensatz der Widerstände wird mit einer entsprechenden Genauigkeitsbewertung versehen. Die ermittelten Genauigkeiten sind in Tabelle 7.2 aufgeführt. Der „gemischte Datensatz“ bezieht sich auf den in Kapitel 6 selbst erstellten Datensatz. Der „weiße Datensatz“ hingegen umfasst Widerstände, die nur vor einem weißen Hintergrund und unter weitgehend konstanter Beleuchtung aufgenommen wurden.

Schwellwert Ansatz

Für den Schwellwert Ansatz wurden folgende Parameter verwendet.

- Kernel Größe: 7
- Strukturelement für das Opening: Rechteck
- Ignorierte Pixel bei Kontrastmaximierung: 0,9 %

Widerstand	Genauigkeit Datensatz gemischt	Genauigkeit Datensatz weiß
1 Ω	73,1 %	100 %
10 Ω	75,6 %	100 %
47 Ω	78,6 %	97,5 %
100 Ω	69,2 %	99 %
150 Ω	64,7 %	97,5 %
220 Ω	77,6 %	98,5 %
330 Ω	76,1 %	98 %
470 Ω	69,2 %	96,5 %
1k Ω	75,1 %	100 %
1,5 kΩ	75,6 %	94,5 %
2,2 kΩ	80,6 %	98 %
4,7 kΩ	76,6 %	95 %
10 kΩ	76,6 %	97,5 %
22 kΩ	79,1 %	97,5 %
33 kΩ	77,6 %	98,5 %
47 kΩ	72,6 %	90,5 %
100 kΩ	74,1 %	94,5 %
220 kΩ	76,1 %	91,5 %
470 kΩ	67,1 %	90,5 %
1 MΩ	69,1 %	97 %

Tabelle 7.2: Genauigkeit Widerstandsdetektion beider Datensätze Schwellwert Ansatz

Wie aus Tabelle 7.2 ersichtlich ist, erweist sich der Algorithmus als äußerst effektiv bei der Verarbeitung von Daten mit weißen Hintergründen. In einigen Fällen werden sämtliche Widerstände erfolgreich erkannt und isoliert. Tabelle 7.3 zeigt die Genauigkeiten des

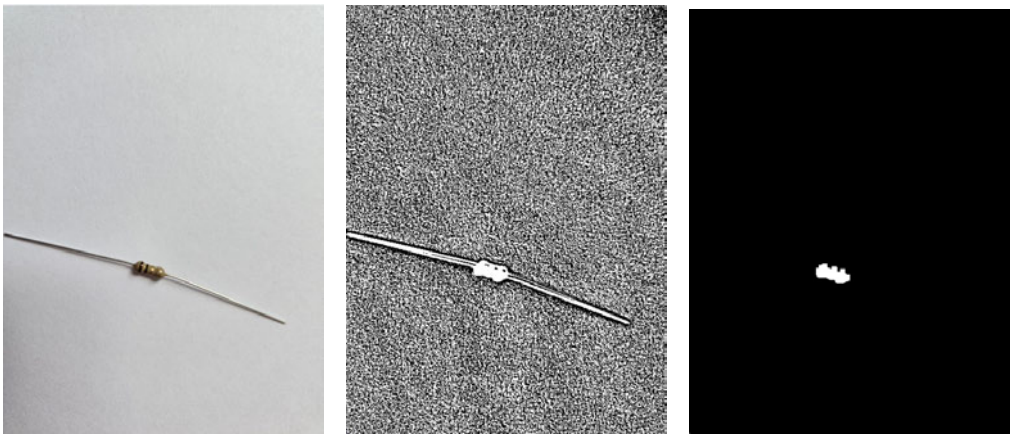
erstellten Datensatzes nochmal detaillierter. Für eine zielführende Analyse sollten die verschiedenen Eigenschaften der Bilder wie der Hintergrund einzeln betrachtet werden.

Widerstand	**Hintergrund gemischt**	**Hintergrund weiß**	**Blitz**	**Dunkel**
1 Ω	9,5 %	95 %	7,1 %	85,7 %
10 Ω	29,7 %	97,2 %	50 %	87,5 %
47 Ω	27 %	94,3 %	37,5 %	0 %
100 Ω	14,9 %	98,4 %	11,8 %	100 %
150 Ω	26,7 %	98 %	0 %	0 %
220 Ω	26,4 %	92,2 %	21,4 %	100 %
330 Ω	41,4 %	93 %	93,3 %	100 %
470 Ω	6,2 %	96,7 %	69,6 %	100 %
1k Ω	24,3 %	97,7 %	73,3 %	14,3 %
1,5 kΩ	27,7 %	88 %	53,3 %	92,9 %
2,2 kΩ	19 %	98,6 %	64,3 %	20 %
4,7 kΩ	20 %	95,5 %	87,5 %	50 %
10 kΩ	44,4 %	96,3 %	86,7 %	16,7 %
22 kΩ	25 %	99,3 %	40 %	100 %
33 kΩ	31,3 %	97,9 %	53 %	100 %
47 kΩ	24,3 %	98,1 %	46,7 %	70 %
100 kΩ	26,7 %	96,3 %	86,7 %	0 %
220 kΩ	32,4 %	96,4 %	46,7 %	100 %
470 kΩ	17,2 %	99,1 %	56,3 %	100 %
1 MΩ	25,4 %	94,6 %	71,4 %	100 %

Tabelle 7.3: Genauigkeit Widerstandsdetektion erstellter Datensatz Schwellwert Ansatz

Der Algorithmus zeigt auf dem Datensatz mit den verschiedenen Hintergründen und Beleuchtungsverhältnissen wie erwartet eine eingeschränkte Leistung. Die Ursachen hierfür erfordern eine gründliche Untersuchung, und im Abschnitt 7.3.1 wird eine umfassende Fehleranalyse vorgenommen.

Abbildung 7.27 veranschaulicht ein Originalbild sowie das entsprechende Binärbild sowohl vor als auch nach Anwendung des Opening-Verfahrens bei einer erfolgreichen Erkennung.



(a) Original (b) Binärbild (c) Binärbild mit Opening

Abbildung 7.27: Erfolgreiche Widerstandsdetektion Schwellwert Ansatz

Kantenbasierter Ansatz

Für den kantenbasierten Ansatz sind die Genauigkeiten in Tabelle 7.5 aufgelistet. Wie zu erwarten läuft der Algorithmus auf den Bildern mit weißem Hintergrund zuverlässiger als auf den Bildern mit gemischten Hintergrund. Im Gegensatz zum Schwellwert Algorithmus weist er allerdings auch auf den Bildern mit weißen Hintergründen einige Schwächen auf. Auch bei Bildern mit gemischten Hintergründen und unterschiedlichen Lichtverhältnissen zeigt der kantenbasierte Algorithmus nicht die gleiche Leistung wie Schwellwert Algorithmus.

Widerstand	Genauigkeit Datensatz gemischt	Genauigkeit Datensatz weiß
1 Ω	72,1 %	96 %
10 Ω	67,6 %	95 %
47 Ω	74,1 %	88,1 %
100 Ω	76,1 %	89,1 %
150 Ω	66,7 %	94,5 %
220 Ω	79,1 %	65,7 %
330 Ω	70,6 %	91,5 %
470 Ω	67,2 %	94 %
1k Ω	70,6 %	86,6 %
1,5 kΩ	76,1 %	94 %
2,2 kΩ	79,6 %	95 %
4,7 kΩ	74,1 %	76,1 %
10 kΩ	79,6 %	93,5 %
22 kΩ	73,6 %	87,1 %
33 kΩ	73,6 %	87,6 %
47 kΩ	73,6 %	88,6 %
100 kΩ	75,6 %	95,5 %
220 kΩ	73,6 %	86,6 %
470 kΩ	72,6 %	84,1 %
1 MΩ	65,2 %	61,2 %

Tabelle 7.4: Genauigkeit Widerstandsdetektion beider Datensätze kantenbasierter Ansatz

Auch bei diesem Ansatz wird der selbst erstellte Datensatz aus Kapitel 6 genauer betrachtet. Tabelle 7.5 verdeutlicht, dass der kantenbasierte Algorithmus ebenfalls zuverlässig auf Bildern mit weißem Hintergrund arbeitet. Hingegen zeigt er in der Regel eine etwas geringere Zuverlässigkeit auf dunklen Bildern, da die Farb- und Helligkeitsunterschiede nicht so ausgeprägt sind wie bei den besser beleuchteten Bildern. Im Vergleich zum ersten Ansatz erzielt der zweite Algorithmus jedoch eine bessere Identifikation bei Bildern mit Blitz und gemischtem Hintergrund.

Widerstand	Hintergrund gemischt	Hintergrund weiß	Blitz	Dunkel
1 Ω	45,2 %	87 %	79 %	14,3 %
10 Ω	39,1 %	97,2 %	75 %	50 %
47 Ω	48,1 %	99,2 %	81,3 %	12,5 %
100 Ω	38,3 %	97,7 %	94,1 %	100 %
150 Ω	50,7 %	98 %	100 %	13,3 %
220 Ω	51 %	96,9 %	83,3 %	100 %
330 Ω	44,8 %	95,6 %	40 %	100 %
470 Ω	33,8 %	100 %	69,6 %	100 %
1k Ω	43,2 %	88,4 %	86,7 %	14,3 %
1,5 kΩ	29,8 %	95,2 %	73,3 %	92,9 %
2,2 kΩ	43,2 %	99,3 %	85,7 %	60 %
4,7 kΩ	51,1 %	98,5 %	37,5 %	12,5 %
10 kΩ	42,2 %	98,1 %	46,7 %	66,7 %
22 kΩ	56,8 %	88,8 %	100 %	100 %
33 kΩ	50 %	100 %	88,2 %	88,9 %
47 kΩ	47,1 %	99,1 %	80 %	100 %
100 kΩ	51,1 %	98,5 %	66,7 %	14,3 %
220 kΩ	72,1 %	99,1 %	73,3 %	100 %
470 kΩ	43,7 %	100 %	81,3 %	100 %
1 MΩ	41,8 %	100 %	71,4 %	44,4 %

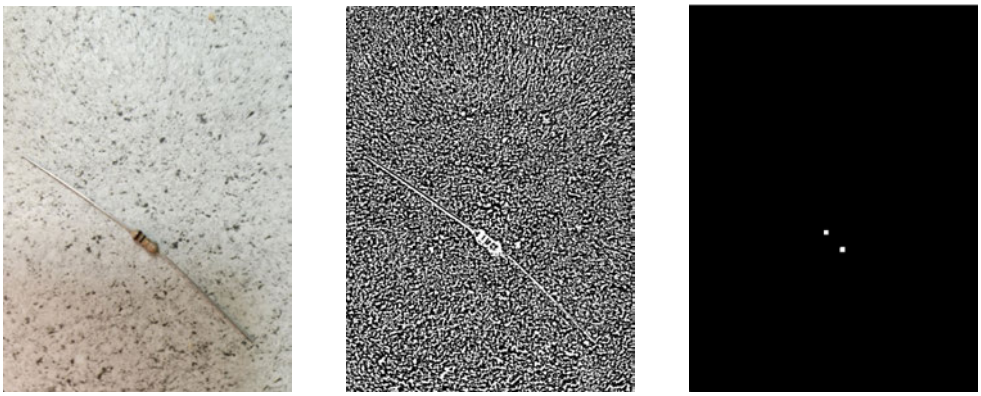
Tabelle 7.5: Genauigkeit Widerstandsdetektion erstellter Datensatz kantenbasierter Ansatz

7.3.1 Fehleranalyse

Um Algorithmen effektiv zu optimieren und ihre Leistung zu steigern, erweisen sich umfassende Fehleranalysen als äußerst hilfreich. Diese Analysen ermöglichen es, Schwachstellen zu identifizieren und gezielte Verbesserungen vorzunehmen, um die Effizienz und Genauigkeit der Algorithmen zu steigern.

Schwellwert Ansatz

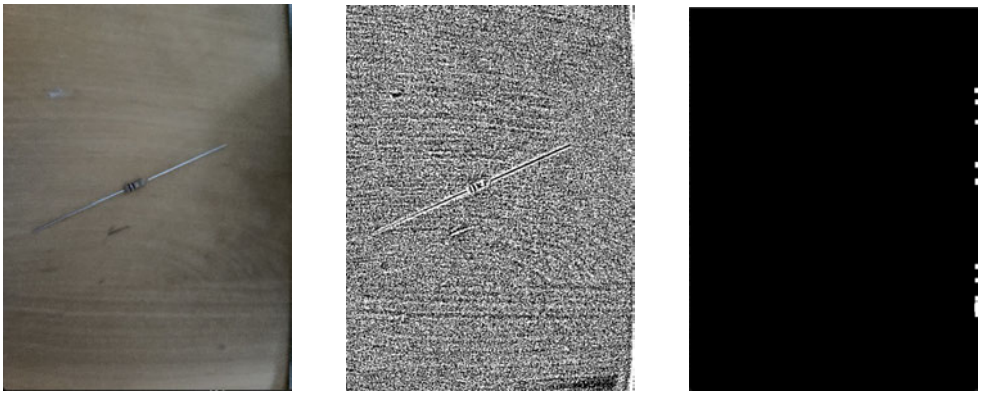
Bei der Betrachtung der Bilder, bei denen der Widerstand nicht korrekt erkannt wird, wird deutlich, dass es sich hauptsächlich um Aufnahmen mit unruhigem Hintergrund und ungünstigen Beleuchtungsverhältnissen handelt. Ein Beispiel für eine misslungene Erkennung wird in Abbildung 7.28 präsentiert. In diesem Fall ist der Widerstand im Binärbild nicht deutlich abgegrenzt, und nach der Anwendung des Opening-Verfahrens sind lediglich zwei weiße Punkte erkennbar.



(a) Original (b) Binärbild (c) Binärbild mit Opening

Abbildung 7.28: Gescheiterte Detektion bei unruhigem Hintergrund (Schwellwert Ansatz)

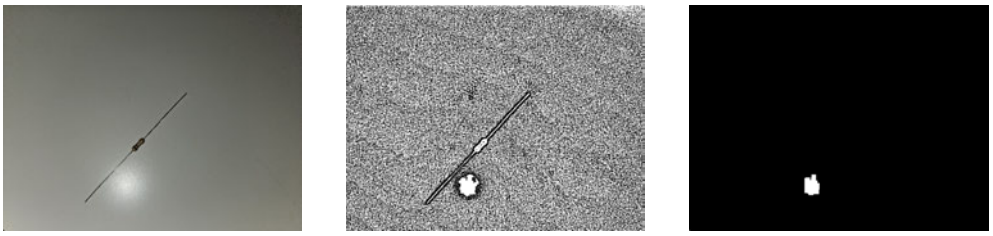
Abbildung 7.29 zeigt ein weiteres Beispiel für eine fehlgeschlagene Erkennung. Das Originalbild weist eine ungünstige Beleuchtung auf und der Hintergrund ist unruhig. In Anbetracht dieser Umstände ist das Ergebnis der Binärbilder nicht überraschend.



(a) Original (b) Binärbild (c) Binärbild mit Opening

Abbildung 7.29: Gescheiterte Detektion bei unruhigem Hintergrund und schlechter Beleuchtung (Schwellwert Ansatz)

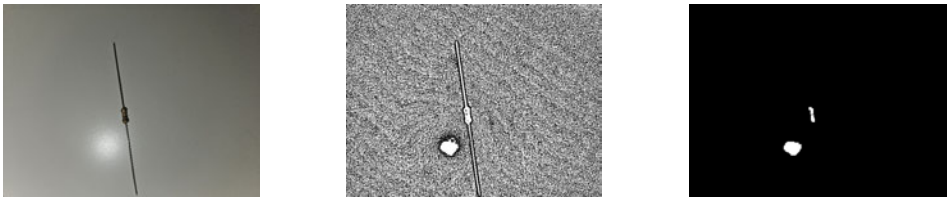
Abbildung 7.30 verdeutlicht ein weiteres Problem bei der Erkennung. Bei den Bildern, die mit einem Blitz aufgenommen wurden, ist die Detektion lediglich teilweise erfolgreich. Bei genauer Betrachtung der Abbildung wird auch schnell deutlich, warum dies der Fall ist: Der helle weiße Kreis resultiert aus dem Blitzlicht.



(a) Original (b) Binärbild (c) Binärbild mit Opening

Abbildung 7.30: Gescheiterte Detektion mit Blitzlicht (Schwellwert Ansatz)

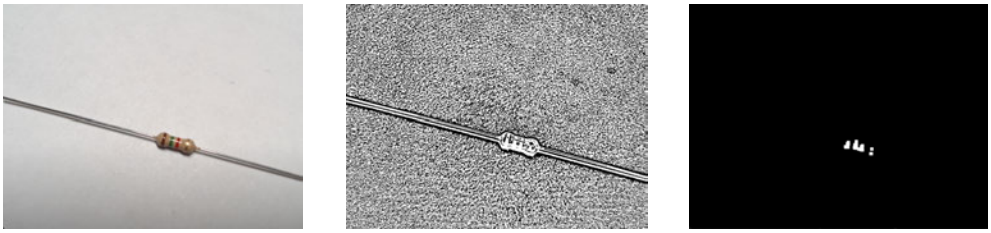
Es gibt aber auch erfolgreiche Detektionen mit Blitzlicht, wie Abbildung 7.31 zeigt.



(a) Original (b) Binärbild (c) Binärbild mit Opening

Abbildung 7.31: Erfolgreiche Detektion mit Blitzlicht (Schwellwert Ansatz)

Die Widerstände auf weißen Hintergrund, die nicht richtig detektiert werden, liegen oft zu weit am Rand. Der Algorithmus ignoriert Konturen, die zu nah am Rand liegen, da sich gezeigt hat, dass am Rand oft Störungen auftreten. In der Anwendung sollte der Widerstand deswegen relativ mittig in die Kamera gehalten werden. Ein anderes Problem was selten auftritt ist, in Abbildung 7.32 dargestellt.



(a) Original (b) Binärbild (c) Binärbild mit Opening

Abbildung 7.32: Gescheiterte Detektion bei weißem Hintergrund (Schwellwert Ansatz)

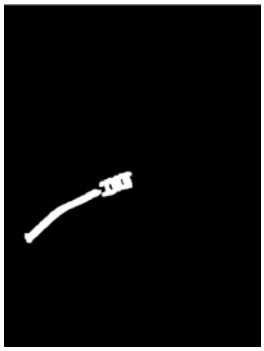
Der Widerstand ist nach dem Opening keine zusammenhängende Struktur mehr. Dies liegt an der Größe des Filterkerns, der der morphologischen Funktion übergeben wird.

Kantenbasierter Ansatz

Im zweiten Ansatz wird ebenfalls eine Fehleranalyse durchgeführt. Ein Problem, das während der Entwicklung aufgetreten ist, betrifft die Bildrotation. Die Rotation des Bildes basiert auf der Erkennung der Widerstandsdrähte, die in den meisten Fällen genauso gerade wie der Widerstandskörper selbst verlaufen. Es gibt jedoch Bilder, auf denen der Draht nicht gerade verläuft, was zu einer fehlerhaften Bildrotation führt. Ein Beispiel für dieses Problem ist in Abbildung 7.33 dargestellt.

Die Drehung des Bildes aufgrund des nicht geraden Widerstandsdrahts ist nicht vollkommen korrekt, jedoch nicht in gravierendem Maße falsch. Ein weiteres Rotationsproblem tritt auf, wenn der Widerstandsdraht fehlerhaft erkannt wird. Aufgrund dieser falsch erkannten Struktur wird das Bild in der Regel fehlerhaft gedreht. Dies tritt häufig auf, wenn am Bildrand eine Kante im Hintergrund sichtbar ist. Das Problem wird in Abbildung 7.34 veranschaulicht.

(a) Original



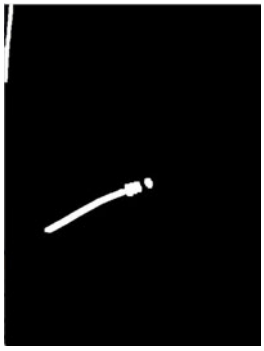
(b) Binärbild

(c) Bildausschnitt

(d) Gedrehtes Bild

Abbildung 7.33: Gescheiterte Rotation (kantenbasierter Ansatz)

(a) Original



(b) Binärbild

(c) Gedrehtes Bild

Abbildung 7.34: Gescheiterte Identifizierung des Widerstandsdrahts (kantenbasierter Ansatz)

Eine mögliche Lösung könnte darin bestehen, Konturen, die sich zu nah am Bildrand befinden, zu ignorieren, um die Fehler bei der Bildrotation zu minimieren. Eine zusätzliche Schwäche des Algorithmus zeigt sich darin, dass die Widerstände gelegentlich zu knapp ausgeschnitten werden, wie es in Abbildung 7.35 deutlich wird. Dies könnte durch eine Anpassung des Ausschnittsprozesses verbessert werden, um sicherzustellen, dass die erkannten Widerstände vollständig erfasst werden.

Abbildung 7.35: Knapper Widerstandsausschnitt (kantenbasierter Ansatz)

7.3.2 Optimierung

In der Praxis hat sich gezeigt, dass andere Parameter beim Schwellwert Ansatz bessere Ergebnisse auf den Datensatz mit den verschiedenen Hintergründen hat. Diese lauten wie folgt.

- Kernel Größe: 5
- Strukturelement für das Opening: Ellipse
- Ignorierte Pixel bei Kontrastmaximierung : 5 %

Wendet man diese auch auf den Datensatz mit dem weißen Hintergrund an, verschlechtert sich das Ergebnis. Deswegen muss eine Lösung gefunden werden, die je nach Bild andere Parameter verwendet. In der Praxis hat sich gezeigt, dass die Binärbilder verschiedene Weißanteile aufweisen. Es wird eine Bedingung implementiert, die den Mittelwert des Weißanteils des Binärbilds überprüft. Ist dieser kleiner als 142 werden die obigen Parameter genutzt, ansonsten die Parameter aus Punkt 7.3. Der Weißanteil von 142 Pixeln hat sich hierbei am besten bewährt. Auch die Widerstände auf den schlecht belichteten Bilder werden dadurch besser segmentiert. Tabelle 7.6 zeigt die Genauigkeiten des optimierten Algorithmus.

7.3.3 Vergleich

Wenn man die Tabellen 7.2 und 7.4 vergleicht, fällt auf, dass die Genauigkeiten auf dem gemischten Datensatz beider Algorithmen fast identisch sind. Der Schwellwert Ansatz erreicht im Durchschnitt eine Genauigkeit von 74,2 %, während der kantenbasierte Ansatz eine Genauigkeit von 73,1 % aufweist. Der Schwellwert Ansatz erweist sich minimal besser auf dem gemischten Datensatz. Ebenso zeigt der Schwellwert Algorithmus auf Bildern mit weißem Hintergrund eine bessere Leistung. Hingegen schneidet der kantenbasierte Algorithmus besser auf Bildern ab, die mit Blitzlicht aufgenommen wurden.

Nach der Optimierung des Schwellwert Ansatzes erweist er sich als noch zuverlässiger und erreicht eine höhere Genauigkeit als der kantenbasierte Ansatz, auch bei Bildern mit

unruhigem Hintergrund. Es wurde auf eine Optimierung des kantenbasierten Ansatzes verzichtet, da die ausgeschnittenen Widerstände teilweise unzureichend waren und der Algorithmus zu viele Schwächen aufgewiesen hat.

Widerstand	**Optimierter Algorithmus**	**Alter Algorithmus**
1 Ω	77,1 %	73,1 %
10 Ω	86,6 %	75,6 %
47 Ω	84,6 %	78,6 %
100 Ω	84,1 %	69,2 %
150 Ω	78,6 %	64,7 %
220 Ω	78,6 %	77,6 %
330 Ω	87,1 %	76,1 %
470 Ω	72,6 %	69,2 %
1k Ω	86,1 %	75,1 %
1,5 kΩ	81,1 %	75,6 %
2,2 kΩ	88,1 %	80,6 %
4,7 kΩ	85,6 %	76,6 %
10 kΩ	88,6 %	76,6 %
22 kΩ	86,6 %	79,1 %
33 kΩ	78,6 %	77,6 %
47 kΩ	80,1 %	72,6 %
100 kΩ	87,1 %	74,1 %
220 kΩ	81,1 %	76,1 %
470 kΩ	79,6 %	67,1 %
1 MΩ	78,1 %	69,1 %

Tabelle 7.6: Genauigkeit Widerstandsdetektion gemischter Datensatz mit optimierten Algorithmus

Am Ende sind mithilfe des erstellten Algorithmus für den Schwellwert Ansatz insgesamt 7148 Trainingsdaten und 200 Testdaten entstanden.

8 Deep Learning

Auch im Bereich des Deep Learning existieren zahlreiche Wege, um das angestrebte Ziel zu erreichen. Es gibt eine Vielzahl von Konzepten, die erforscht und analysiert werden müssen. In Abbildung 8.1 werden für die letzten vier Schritte verschiedene Ansätze getestet und gründlich analysiert. Ein entscheidender Aspekt ist beispielsweise die Data Augmentation, bei der vorhandene Daten künstlich verändert werden, um neue Datenpunkte zu generieren. Bei der Auswahl der Netzwerkarchitektur stehen zahlreiche Optionen zur Verfügung, da die verschiedenen Schichten des Netzes frei gewählt werden können. Auch das Qualitätsmaß spielt eine wichtige Rolle. Beim Training sind ebenfalls verschiedene Einstellungen möglich. Um am Ende ein bestmögliches Ergebnis zu erzielen, ist eine Hyperparametersuche erforderlich.

8.1 Design

Die entstandenen Bildausschnitte der Widerstände werden mithilfe von Deep Learning weiterverarbeitet. Abbildung 8.1 zeigt den Ablauf der Deep Learning Anwendung. Die Datenbeschaffung und Datenaufbereitung sind schon durch die Kapitel 6 und 7 abgedeckt.

Abbildung 8.1: Deep Learning Ablauf

8.1.1 Entwicklungsumgebung

Die neuronalen Netze werden in der universellen und gut lesbaren Programmiersprache Python trainiert. Da bereits Vorkenntnisse in der Webanwendung JupyterLab mit der

Distribution Anaconda existieren, erfolgt die Programmierung in dieser Umgebung. Auch für die Open Source Deep-Learning-Bibliothek Keras mit Python existieren Vorkenntnisse. Keras bietet eine schnelle Implementierung neuronaler Netzwerke für Anwendungen des Deep Learnings. Die Bibliothek ist in Python geschrieben und bietet unter anderem die Verwendung des weit verbreiteten Frameworks TensorFlow.

8.1.2 Data Augmentation

Bei der Data Augmentation wird der Datensatz vergrößert, indem die vorhandenen Daten künstlich verändert werden. Ein sinnvoller Schritt ist hier die Bilder um 180° zu drehen. Im Bildverarbeitungsprozess wurden die Bilder horizontal ausgerichtet, die Richtung war allerdings Winkel abhängig. Deswegen sind die Bilder sowohl so ausgerichtet, dass der goldene Toleranzstreifen des Widerstandes sich sowohl rechts als auch links im Bild befindet. Für das Netz soll die horizontale Ausrichtung unerheblich sein. Durch diesen Prozess kann die Anzahl der Trainingsdaten erhöht werden und somit die Chance auf eine bessere Genauigkeit gesteigert werden. Auch andere Methoden wären denkbar. Die Lichtverhältnisse könnten beispielsweise geändert werden oder Rauschen zu den Bildern hinzugefügt werden.

8.1.3 Trainings-, Validierungs- und Testdaten

Die Trainingsdaten werden für das Trainieren des Netzes verwendet. Aus diesen Daten lernt das Netz bestimmte Aufgaben zu erfüllen. Neben den Trainingsdaten gibt es noch die Validierungsdaten, diese stammen häufig aus der Trainingsdatenmenge, werden aber nicht fürs Training genutzt. Durch die Validierungsdaten kann die Leistung des Modells während des Trainings überwacht werden. Nach jeder Trainingsepoche wird das Modell auf den Validierungsdaten getestet um zu prüfen wie gut es generalisiert. Außerdem gibt es noch die Testdaten, bei diesen handelt es sich um einen unabhängigen Datensatz. Diese Daten werden erst am Ende verwendet, um die Endleistung des Modells zu testen [7].

8.1.4 Netzwerkarchitektur

Die Netzwerkarchitektur richtete sich nach den Eingangsdaten. Der Eingangslayer ist abhängig von den Eingangsdaten. Eine gängige Praxis von Bildern als Eingangsdaten ist die

Skalierung. Hierbei werden die Pixel auf den Bereich 0 bis 1 skaliert. Dies bringt zahlreiche Vorteile mit sich, darunter die Normalisierung der Daten, was in vielen Netzwerken von Nutzen ist, außerdem kann damit das Problem der verschwindenden Gradienten reduziert werden. Zusätzlich ermöglicht die Skalierung eine bessere Initialisierung der Gewichtungen.

Nach der Anpassung des Eingangslayers können verschiedene Schichten dem Modell hinzugefügt werden. Im Folgenden werden die verschiedenen Schichttypen kurz erklärt, wobei eine detaillierte Erläuterung in Abschnitt 2.3 zu finden ist.

Ein Convolutional Layer führt Faltungsvorgänge auf Bildern durch, um unterschiedliche Merkmale der Bilder zu extrahieren. Hierbei werden verschiedene Filterkerne mit vordefinierter Filtergröße auf die Eingangsdaten angewendet [6]. Zudem wird eine Aktivierungsfunktion auf die Ausgabewerte angewendet, mehr zu Aktivierungsfunktionen im Kapitel 2.3.2.

Ein Max-Pooling Layer führt Max-Pooling-Operationen auf den Daten durch, um die räumliche Dimension der Ausgabe zu reduzieren. Dabei wird der maximale Wert in jedem Pooling-Fenster ausgewählt. Im Gegensatz dazu wählt der Average-Pooling Layer den Durchschnittswert in jedem Pooling-Fenster aus [6].

Der Flatten Layer konvertiert die 2D-Ausgabe der vorherigen Schicht in einen flachen Vektor. Dies ist beispielsweise vor dem Ausgabelayer erforderlich [6].

Der Dropout Layer deaktiviert während des Trainings zufällig eine bestimmte Anzahl von Neuronen. Dies bedeutet, dass die Ausgaben der deaktivierten Einheiten auf null gesetzt werden. Durch diese zufällige Deaktivierung wird das Netz dazu gezwungen redundante Wege und breite Merkmalskombinationen zu entwickeln und eine bessere Generalisierung zu erzielen. Dadurch kann Überanpassung, also das auswendig Lernen eines Netzes, vermieden werden [6].

Ein Batch Normalization Layer kann in tiefen Convolutional Neural Networks und Deep Learning-Modellen eine entscheidende Rolle spielen. Hierbei werden die Eingaben normalisiert, indem der Durchschnitt und die Standardabweichung der Aktivierungen in der aktuellen Minibatch (eine Teilmenge der Trainingsdaten) berechnet werden. Batch Normalization kann das Training effizienter gestalten, das Problem des Verschwindens von Gradienten verringern und zu einer verbesserten Generalisierung führen [6].

Der Ausgabelayer ist in der Regel ein Fully-Connected Layer. Die Anzahl der Neuronen entspricht immer der Anzahl der Klassen. Dieser Layer generiert die Klassenausgabe für das Klassifikationsproblem. Die angegebene Aktivierungsfunktion berechnet für jede Klasse die Wahrscheinlichkeit [6].

Es gibt noch viele andere Schichttypen, die je nach Anwendung verwendet werden können. Für diese spezielle Anwendung sind die beschriebenen Schichten ausreichend.

Es gibt zahlreiche berühmte CNN-Architekturen. Eines davon ist das Visual Geometry Group (VGG)-Netz, dieses wird häufig für Bildklassifizierungsaufgaben verwendet. Es wurde 2014 von Karen Simonyan und Andrew Zisserman entwickelt. Ein typisches Merkmal ist die Einfachheit und Tiefe des Netzes, welches durch die Abfolge von Convolutional-Schichten mit kleinen 3x3 Filtern und Max-Pooling-Schichten (2x2) zur Dimensionsreduzierung und Stapelung mehrerer Schichten erreicht wird. Es gibt mehrere Versionen des VGG-Netzes, so besitzt das VGG16 Netz 16 Schichten und das VGG19 Netz 19 Schichten [23]. Abbildung 8.2 zeigt die Architektur eines VGG-Netzes.

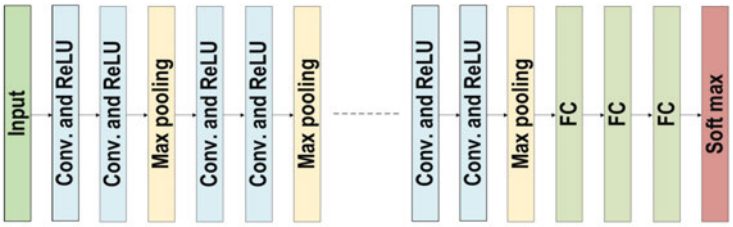


Abbildung 8.2: Architektur VGG-Netz [23]

8.1.5 Training

Nach Festlegung der Netzwerkarchitektur und des Qualitätsmaßes kann das Training gestartet werden. Hierbei wird das Modell mit den Trainingsdaten gefüttert und so optimiert, dass die Verlustfunktion minimiert wird. Dies erfolgt durch Vorwärts- und Rückwärtsdurchläufe, bei denen die Gewichtungen des Modells aktualisiert werden. Die Anzahl der Epochen gibt an, wie oft der Trainingsprozess durchlaufen wird, genauer gesagt wie oft die Trainingsdaten zum Training verwendet werden. Dieser Parameter beeinflusst ebenfalls die Genauigkeit des Modells [7]. Eine Möglichkeit, den Trainingsprozess vorzeitig zu beenden, ist das sogenannte „Early-Stopping“. Dabei wird das Training gestoppt, wenn sich die Leistung auf dem Validierungsdatensatz verschlechtert oder nicht mehr

verbessert, um Überanpassung zu vermeiden. Die Überwachung des Trainingsprozesses ist entscheidend, um das Modell entsprechend anzupassen.

8.1.6 Hyperparameter

Hyperparameter sind Einstellparameter, die das Verhalten des Algorithmus steuern können. Die Auswahl der Hyperparameter kann wesentlich zur Leistungsfähigkeit des neuronalen Netzes beitragen. Anders als die vom Algorithmus gesteuerten Parameter müssen Hyperparameter manuell festgelegt werden. Oft ist eine iterative Suche nach den optimalen Hyperparametern erforderlich, um am Ende die bestmögliche Leistung des Modells zu erreichen. Ein Beispiel für einen solchen Hyperparameter ist die Lernrate, die im Gradientenabstiegsverfahren verwendet wird. Die Lernrate ist ein positiver Skalar, der die Schrittweite bei der Aktualisierung der Gewichtungen bestimmt (weitere Informationen finden sich in Abschnitt 2.3.5). Andere Hyperparameter umfassen beispielsweise die Anzahl der Schichten, die Batch-Größe oder auch die Anzahl der Neuronen in einem neuronalen Netz [7]. Die Aktivierungsfunktionen sind ebenfalls Hyperparameter. Beispiele hierfür sind ReLU, ELU oder Scaled Exponential Linear Unit (SELU).

8.2 Umsetzung

In diesem Abschnitt wird der Grundstein für das Deep Learning Modell gelegt. Es werden das Vorgehen und die Umsetzung ausführlich beschrieben. Dieses Modell wird anschließend im nächsten Abschnitt 8.3 unter verschiedenen Gesichtspunkten verbessert und bewertet.

8.2.1 Datensatz einlesen

Durch Angabe des Pfades der Trainings- und Testdaten, kann die Keras Methode *image_dataset_from_directory* die Bilder aus dem gegebenen Pfad einlesen. Listing 8.1 zeigt den zugehörigen Code.

Nach dem Einlesen der Daten werden diese zufällig in sogenannte Stapel (Batches) zusammen mit den zugehörigen Labels konvertiert. Außerdem müssen die Validierungsdaten abgezweigt werden. Listing 8.2 zeigt den zugehörigen Code.

Listing 8.1: Daten einlesen

```
archive_train="../../Bilder/Datensatz/Train/"
archive_test="../../Bilder/Datensatz/Test/"
data_dir=pathlib.Path(archive_train).with_suffix('')
data_test_dir=pathlib.Path(archive_test).with_suffix('')
image_count = len(list(data_dir.glob('*/*.jpg')))
one = list(data_dir.glob('1/*'))
```

Listing 8.2: Datenstapel erstellen

```
batch_size = 32
img_height = 40
img_width = 70

train_ds,val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="both",
  label_mode='int',
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
class_names = train_ds.class_names

test_ds = tf.keras.utils.image_dataset_from_directory(
  data_test_dir,
  label_mode='int',
  shuffle=False,
  image_size=(img_height, img_width),
  batch_size=batch_size)

print(class_names)
```

Mithilfe der Keras Funktion wird ein Datenset erstellt, welches Stapel von Bildern mit den zugehörigen Labeln enthält. Der Parameter *validation_split* spaltet 20 % des Trainingsdatensatzes ab, um daraus einen Validierungsdatensatz zu erstellen. Der Parameter *seed* bezeichnet ein optionalen zufälligen Startwert, welcher für das Mischen und Transformationen notwendig ist. Die Größe der Bilder wird auf 70x40 festgelegt. Die Batch-Size gibt die Größe des Stapels an. Zunächst wird hier der Standardwert von 32 verwendet. Die Abbildung 8.3 zeigt die Konsolenausgabe.

```
Found 7148 files belonging to 20 classes.
Using 5719 files for training.
Using 1429 files for validation.
['1', '10', '100', '1000', '10000', '100000', '1000000', '150', '1500', '220', '2200', '22000', '220000', '3
30', '33000', '47', '470', '4700', '47000', '470000']
```

Abbildung 8.3: Konsolenausgabe Datenstapel

Es ist sinnvoll eine kleine Teilmenge der Trainingsdaten zu betrachten, um diese visuell zu überprüfen. Abbildung 8.4 zeigt die Visualisierung der ersten neun zufälligen Bilder des ersten Batchs.

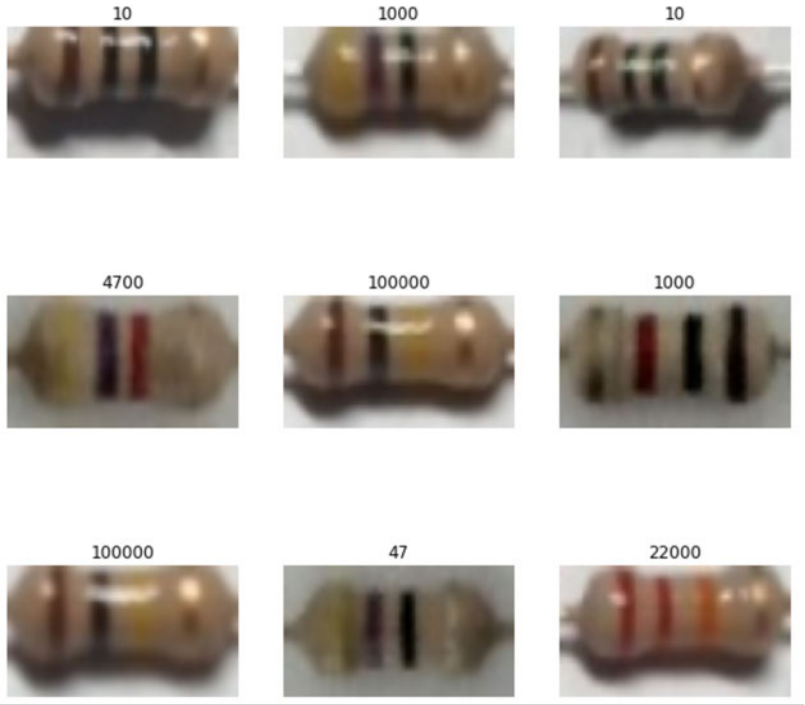


Abbildung 8.4: Ausgabe Trainingsdaten des ersten Batchs

8.2.2 Netzwerkarchitektur und Qualitätsmaß definieren

Mithilfe von Keras wird ein sequentielles Modell erstellt. So ein Modell eignet sich für einen einfachen Stapel von Layern, bei dem jeder Layer genau einen Eingabetensor und einen Ausgabetensor aufweist. Dem Modell können die verschiedenen Schichten per Befehl hinzugefügt werden. Die Netzwerkarchitektur im Listing 8.3 richtet sich nach der

Architektur eines VGG Netzes. Es mussten allerdings ein paar Änderungen vorgenommen werden, da das klassische VGG-Netz für Bilder der Größe 224x224 ausgelegt ist [23]. Deswegen besitzt das verwendete Netz nur 14 Schichten. Bei mehr als 14 Schichten werden Werte von Berechnungen negativ und werfen dadurch Fehler. Listing 8.3 zeigt das Basisnetzwerk.

Listing 8.3: Basisnetzwerk

```
num_classes=20

model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(256, 3, activation='relu'),
    tf.keras.layers.Conv2D(256, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

optimizer = tf.keras.optimizers.Adam()
loss=tf.keras.losses.SparseCategoricalCrossentropy()
model.compile(optimizer=optimizer,
              loss=loss,
              metrics=['accuracy'])
```

Der Eingabelayer ist ein Rescaling-Layer, dieser skaliert die Pixelwerte von dem Bereich 0 bis 255 auf den Bereich 0 bis 1. Anschließend folgen zwei Faltungsschichten mit 64 Filtern der Größe 3x3. Als Aktivierungsfunktion wird die ReLU gewählt. Der Max-Pooling Layer verringert die Dimension, indem er aus jedem Pooling-Fenster den maximalen Wert wählt. Anschließend folgen wieder zwei Faltungsschichten mit 128 Filtern. Oft wird die Filteranzahl mit jeder Schicht erhöht, dadurch soll das Netzwerk in der Lage sein, auf verschiedenen Ebenen unterschiedlich komplexe Merkmale zu extrahieren. Dadurch kann das Netzwerk lernen von einfachen zu komplexen Merkmalen zu gelangen [7]. Nach der sechsten Faltungsschicht folgt ein Flatten-Layer, der notwendig ist, um die Daten von den Convolutional-Schichten in die Dense-Schichten weiterzuleiten. Die Dense-Schicht dient

dazu, komplexe nichtlineare Abbildungen zu lernen. Der Ausgangslayer ist ein Fully-Connected Layer, der die endgültigen Klassenwahrscheinlichkeiten generiert.

Nachdem das Modell definiert ist, wird das Qualitätsmaß festgelegt. Dies geschieht mit der Methode *model.compile*. Als Optimierer wird Adam gewählt. Adam ist ein viel genutzter Optimierer, der oft sehr gute Ergebnisse erzielt. Die Kreuzentropie wird als Verlustfunktion gewählt, da diese bei Klassifizierungsproblemen mit mehreren Klassen mit am häufigsten verwendet wird. Als Metrik zur Bewertung des Modells dient die häufig genutzte Genauigkeit (Accuracy).

8.2.3 Training und Ergebnis

Sobald das Modell vollständig definiert ist, kann mit dem Training begonnen werden. Dies passiert mit der Methode *model.fit*. Dieser Methode werden der Trainings- und Validierungsdatensatz übergeben. Außerdem muss die Anzahl der Epochen angegeben werden, in diesem Fall sind es 20. Eine Epoche von 20 ist groß genug, damit das Netz die Merkmale zuverlässig lernt und klein genug damit das Training nicht zu lange dauert. Unter Verwendung des Laptops Yoga 9 14IAP7 mit einem Intel Core i7 Prozessor dauerte das Training über 20 Epochen etwa 17 Minuten. Beim finalen Netz sollte die Epochenanzahl erhöht werden. Listing 8.4 zeigt den Code zum Training des Modells und der anschließend grafischen Darstellung der Genauigkeit.

Listing 8.4: Code Training

```
1  history = model.fit(train_ds, epochs=20,
2                      validation_data=(val_ds))
3
4  plt.plot(history.history['accuracy'], label='accuracy')
5  plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
6  plt.xlabel('Epoch')
7  plt.ylabel('Accuracy')
8  plt.ylim([0, 1])
9  plt.legend(loc='lower right')
10 plt.savefig("ac1")
```

Abbildung 8.5 zeigt übersichtlich den Trainingsfortschritt bis Epoche 8. Bei Epoche 20 wird eine Validierungs-Genauigkeit von 94 % erreicht und der Wert der Verlustfunktion beträgt 0,22.

```
Epoch 1/20
179/179 ━━━━━━━━━━━━━━━━━━━━ 61s 298ms/step - accuracy: 0.0773 - loss: 2.8702 - val_accuracy: 0.3870 - val_l
oss: 1.6144
Epoch 2/20
179/179 ━━━━━━━━━━━━━━━━━━━━ 51s 282ms/step - accuracy: 0.4371 - loss: 1.5185 - val_accuracy: 0.7236 - val_l
oss: 0.8194
Epoch 3/20
179/179 ━━━━━━━━━━━━━━━━━━━━ 52s 287ms/step - accuracy: 0.7876 - loss: 0.6554 - val_accuracy: 0.8593 - val_l
oss: 0.4305
Epoch 4/20
179/179 ━━━━━━━━━━━━━━━━━━━━ 51s 286ms/step - accuracy: 0.8835 - loss: 0.3621 - val_accuracy: 0.9020 - val_l
oss: 0.3111
Epoch 5/20
179/179 ━━━━━━━━━━━━━━━━━━━━ 51s 283ms/step - accuracy: 0.9189 - loss: 0.2451 - val_accuracy: 0.9041 - val_l
oss: 0.3794
Epoch 6/20
179/179 ━━━━━━━━━━━━━━━━━━━━ 51s 283ms/step - accuracy: 0.9384 - loss: 0.1858 - val_accuracy: 0.9272 - val_l
oss: 0.2442
Epoch 7/20
179/179 ━━━━━━━━━━━━━━━━━━━━ 51s 286ms/step - accuracy: 0.9575 - loss: 0.1311 - val_accuracy: 0.9461 - val_l
oss: 0.2018
Epoch 8/20
179/179 ━━━━━━━━━━━━━━━━━━━━ 51s 285ms/step - accuracy: 0.9489 - loss: 0.1387 - val_accuracy: 0.9363 - val_l
oss: 0.2238
```

Abbildung 8.5: Trainingsfortschritt

Die fit-Methode gibt ein sogenanntes History Objekt zurück. Dieses Objekt enthält ein Datensatz von Trainingsverlustwerten und Metrikwerten in aufeinanderfolgenden Epochen sowie von Validierungsverlustwerten und Validierungsmetrikwerten. Diese Parameter können grafisch angezeigt werden. Abbildung 8.6 zeigt den Verlauf der Genauigkeiten. Der Verlauf ist typisch für ein Netz, am Anfang steigen die Genauigkeiten stark an, haben diese eine Schwelle überschritten verbessern sie sich nur noch langsam oder verschlechtern sich minimal. Die Trainingsgenauigkeit ist größer als die Validierungsgenauigkeit.

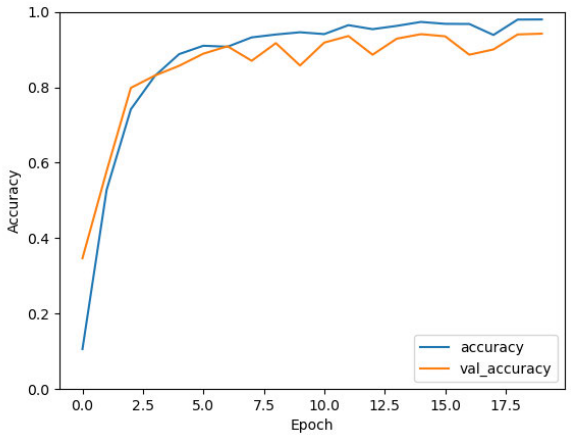


Abbildung 8.6: Genauigkeit Training & Validierung

Um zu prüfen, wie das Netz auf neue unbekannte Daten reagiert, muss es evaluiert werden. Die Methode *model.evaluate* evaluiert das Model mit den Testdaten, Listing 8.5 zeigt den dazugehörigen Code.

Listing 8.5: Code Model Evaluation

```
1 test_loss, test_acc = model.evaluate(test_ds, verbose=2)
2 print("Genauigkeit: ")
3 print(test_acc)
```

Die Genauigkeit des Models beträgt 81 % auf den Testdaten, dies ist für den ersten Versuch nicht schlecht aber definitiv verbesserungswürdig. Abbildung 8.7 zeigt das Ergebnis der Evaluierung auf der Konsole.

```
7/7 - 1s - loss: 0.7855 - accuracy: 0.8150 - 901ms/epoch - 129ms/step
Genauigkeit:
0.8149999976158142
```

Abbildung 8.7: Konsolenausgabe Evaluation

8.3 Optimierung

Im vorherigen Abschnitt wurde ein Basisnetzwerk festgelegt, welches im Folgenden durch verschiedene Methoden optimiert wird.

8.3.1 Data Augmentation

Bei der Data Augmentation werden die Trainingsdaten durch künstliche Veränderungen erweitert, um die Leistungsfähigkeit von Modellen zu verbessern. Ein gängiges Verfahren ist das horizontale Spiegeln von Bildern. In Keras ermöglicht die Schicht

```
tf.keras.layers.RandomFlip("horizontal")
```

während des Trainings eine zufällige horizontale Spiegelung der Bilder. Hierbei wird für jedes Bild im Trainingsdatensatz mit einer spezifizierten Wahrscheinlichkeit entschieden, ob es horizontal gespiegelt wird. Durch diese Methode entstehen zusätzliche Trainingsdaten, indem vorhandene Bilder horizontal gespiegelt werden. Dies trägt zur Robustheit und Vielfalt der Daten bei, wodurch das Modell besser generalisieren kann. In Abbildung

8.8 ist zu sehen, dass beide Grafen deutlich konstanter verlaufen als in Abbildung 8.6, die Data Augmentation verbessert das Modell und wird beibehalten.

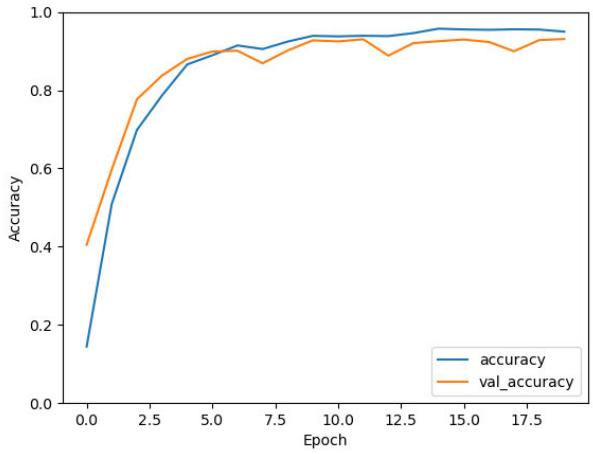


Abbildung 8.8: Genauigkeit Training & Validierung Data Augmentation

Die Testgenauigkeit beträgt 83 %.

Auch die Veränderung der Beleuchtung könnte eine sinnvolle Methode sein. Folgender Layer wird dafür eingefügt.

```
tf.keras.layers.RandomBrightness(factor=0.2)
```

Beim Training wurde jedoch festgestellt, dass die Genauigkeit nicht über 1 % steigt. Es besteht die Möglichkeit, dass der Algorithmus in einem lokalen Minimum stecken bleibt und Schwierigkeiten hat, dieses zu verlassen.

8.3.2 Netzwerkarchitektur ändern

Die Struktur eines neuronalen Netzwerks bietet eine Fülle von Anpassungsmöglichkeiten. Dabei soll die grundlegende Struktur des Netzes beibehalten werden und nur sinnvolle Änderungen in Betracht gezogen werden. Anstelle von Max Pooling-Schichten können beispielsweise Average-Pooling-Schichten oder eine Kombination aus beiden verwendet

werden, um unterschiedliche Merkmale zu erfassen. Zusätzlich ist es möglich, das Netzwerk zu vertiefen oder zu „verflachen“, indem Schichten hinzugefügt oder entfernt werden. Die Integration neuer Schichten wie beispielsweise eines Batch-Normalization-Layers kann die Modellleistung verbessern, indem es zu stabileren und schnelleren Konvergenzen während des Trainings führt. Diese Vielfalt an Anpassungsmöglichkeiten ermöglicht es, das Netzwerk an spezifische Daten oder Anforderungen anzupassen und die Leistungsfähigkeit des Modells zu optimieren.

Anstatt Max-Pooling Schichten hinter den Faltungsschichten einzubauen sind auch Average-Pooling Schichten denkbar. In Abbildung 8.9 ist der Genauigkeitsverlauf bei Average-Pooling Schichten statt Max-Pooling Schichten abgebildet.

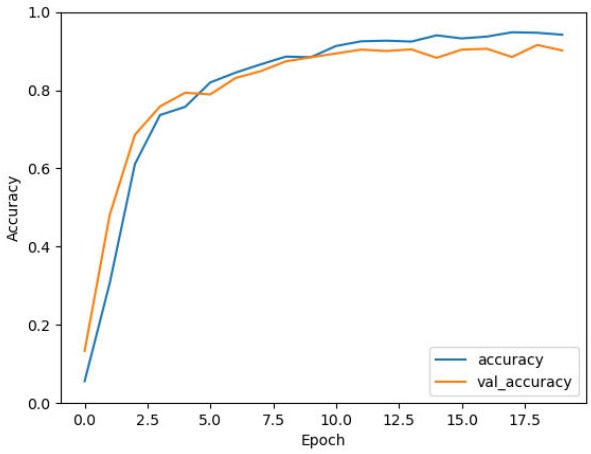


Abbildung 8.9: Genauigkeit Training & Validierung Average Pooling

Die Testgenauigkeit betrug nur 72 %, das Ergebnis hat sich verschlechtert und kommt nicht infrage. Auch eine flachere Hierarchie ist denkbar, da das Netz ggf. zu komplex ist. Dafür wurden die Faltungsschichten mit den 256 Neuronen entfernt, das Ergebnis ist in Abbildung 8.10 abgebildet. Der Verlauf sieht vielversprechend aus, allerdings hat sich das Ergebnis auf den Testdaten verschlechtert. Die Testgenauigkeit betrug 80 %, auch dies ist ein schlechteres Ergebnis.

Durch Batch-Normalization werden die Eingaben normalisiert. Dafür wird ein Batch-Normalization Layer eingefügt.

```
tf.keras.layers.BatchNormalization(epsilon=0.001, momentum=0.99)
```

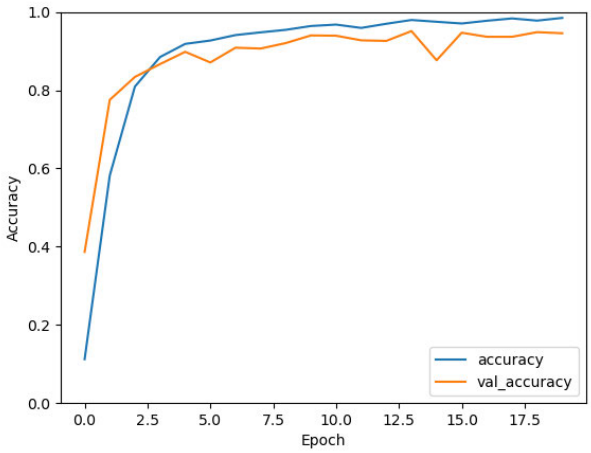


Abbildung 8.10: Genauigkeit Training & Validierung bei flacher Hierarchie

Das Ergebnis durch den zusätzlichen Layer ist in Abbildung 8.11 zu sehen.

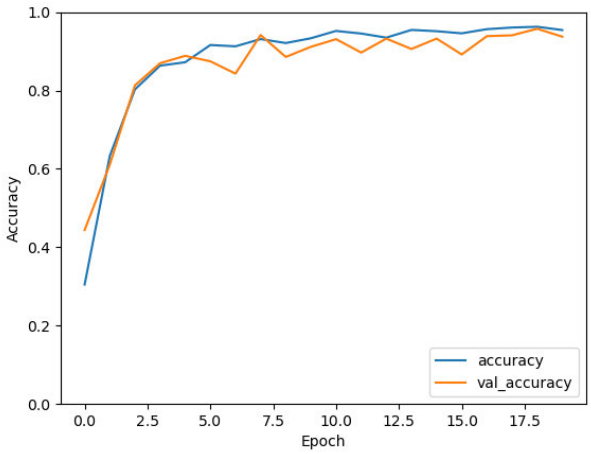


Abbildung 8.11: Genauigkeit Training & Validierung Batch-Normalization

Der Verlauf des Grafen der Validierungsgenauigkeit ist wieder unregelmäßiger als ohne Batch-Normalization, die Testgenauigkeit allerdings besser. Ein weiterer vorteilhafter Punkt ist, dass durch die Gewichtsnormierung beide Genauigkeiten zu Beginn höher ausfallen als ohne Normalisierung. Die Testgenauigkeit betrug hierbei 84 %, das Ergebnis ist im Vergleich zu den 83 % aus Abschnitt 8.3.1 besser.

Um Overfitting zu verhindern, sollten Dropout Schichten ins Netz eingebaut werden. Folgende Schicht wurde vor den letzten Dense Layer eingebaut.

```
tf.keras.layers.Dropout(0.3)
```

Abbildung 8.12 zeigt den Genauigkeitsverlauf bei einem zusätzlichen Dropout Layer. Die beiden Grafen sind nun deutlich näher beieinander, was idealerweise auch der Zweck eines Dropout Layers ist

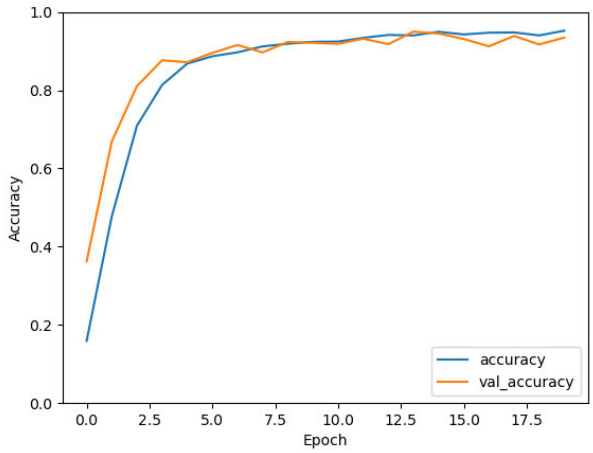


Abbildung 8.12: Genauigkeit Training & Validierung Dropout

Die Testgenauigkeit betrug 82 %. Gaussian Dropout ist eine Dropout-Variante, bei der die ausgeworfenen Gewichte einer Schicht stochastisch durch eine Gaussian-Verteilung (Normalverteilung) modelliert werden. Abbildung 8.13 zeigt das Ergebnis bei einem Dropout Layer mit einer Ausfallrate von 0,3. Auch hier sind die Grafen bis auf eine Ausnahme im Verlauf der Validierungsgenauigkeit dichter beisammen.

Die Testgenauigkeit betrug 84 %, dies ist ein gutes Ergebnis und wird beibehalten. Denkbar wären auch zwei Dropout Schichten.

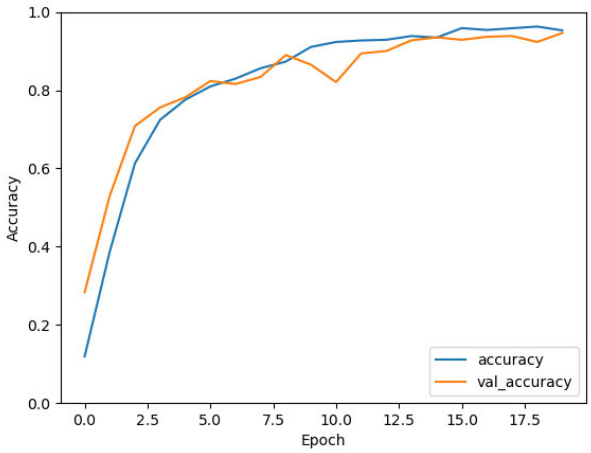


Abbildung 8.13: Genauigkeit Training & Validierung Gaussian Dropout

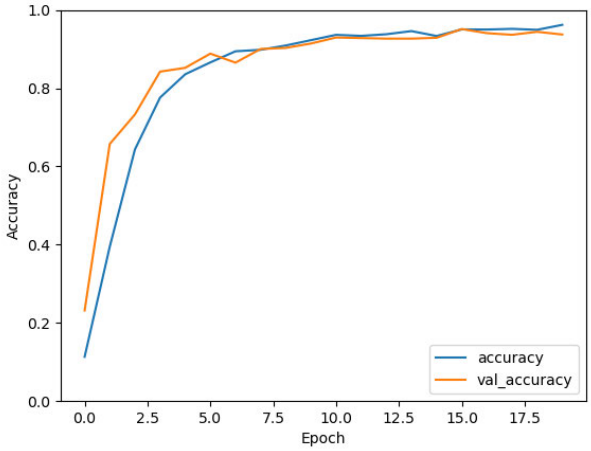


Abbildung 8.14: Genauigkeit Training & Validierung zwei Gaussian Dropout

Abbildung 8.14 zeigt das Ergebnis bei zwei Gaussian Dropout Schichten. Die Testgenauigkeit betrug allerdings nur 80 %. Die Tabelle 8.1 fasst die Testgenauigkeiten verschiedener Modifikationen der Netzwerkarchitekturen zusammen.

Modifikation	Testgenauigkeit
Data Augmentation	83 %
Average-Pooling	72 %
Flache Hierarchie	80 %
Batch-Normalization	84 %
Dropout	83 %
Gaussian Dropout	84 %
Zwei Gaussian Dropout	80 %

Tabelle 8.1: Netzwerkarchitektur Modifikationen

Als besonders gut stellt sich die Gaussian Dropout Schicht, Data Augmentation und die Batch-Normalization heraus. Die anderen Veränderungen verschlechtern dagegen das Ergebnis.

8.3.3 Qualitätsmaß

Als bestes Netzwerk hat sich das Netz in Listing 8.6 herausgestellt.

Listing 8.6: beste Netzwerkarchitektur

```
model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    tf.keras.layers.BatchNormalization(epsilon=0.001, momentum=0.99),
    tf.keras.layers.RandomFlip("horizontal"),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(256, 3, activation='relu'),
    tf.keras.layers.Conv2D(256, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.GaussianDropout(0.3),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

Die Abbildung 8.15 zeigt den Verlauf des Netzwerkes aus Listing 8.6. Beide Grafen sind dicht beieinander und weisen eine größtenteils gleichmäßigen Verlauf auf.

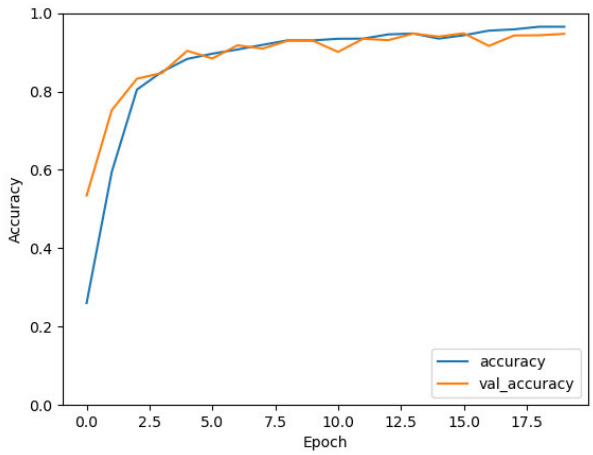


Abbildung 8.15: Genauigkeit Training & Validierung beste Netzwerkarchitektur

Die Testgenauigkeit betrug bei diesem Netzwerk 85 %.

Unter Verwendung dieses Netzwerks wurden verschiedene Optimierer und Verlustfunktionen getestet. In Tabelle 8.2 sind die entsprechenden Genauigkeiten aufgeführt, wobei sich die Trainings- und Validierungsgenauigkeit auf die zwanzigste Epoche beziehen.

	Testgenauigkeit	Trainingsgenauigkeit	Validierungsgenauigkeit
Adam	85 %	96 %	95 %
SGD	80 %	94 %	93 %
RMSprop	84 %	95 %	94 %
Adagrad	41 %	76 %	81 %

Tabelle 8.2: Optimierer

Alle Optimierer wurden mit den Standardwerten initialisiert, wobei die Standardlernrate bei allen Optimierern 0,001 beträgt. Während des Vergleichs hat sich der Adam-Optimierer als die beste Wahl erwiesen. Adagrad hat für dieses spezifische Netzwerk ungeeignete Ergebnisse geliefert. Ein Grund dafür könnte die schlechte Anpassung der Lernrate sein. Bei Adagrad kann es sein, dass die Lernrate über die Zeit zu stark reduziert wird. Ein anderes Problem ist die Akkumulation der quadrierten Gradienten im Nenner

der Lernratenformel (siehe Formel 2.33). Es kann dabei zu einer zu starken Anpassung der Lernrate kommen, wodurch die Konvergenz des Modells beeinträchtigt werden kann. Beim SGD und Adagrad ist außerdem aufgefallen, dass sich während des Trainings die Trainings- und Validierungsgenauigkeit nur langsam verbessert hat.

Im nächsten Schritt werden verschiedene Verlustfunktionen untersucht, wobei der Optimierer Adam und das Netz aus Listing 8.6 verwendet werden. Es werden ausschließlich probabilistische Verlustfunktionen in Betracht gezogen, welche die Differenzen zwischen den vorhergesagten Wahrscheinlichkeitsverteilungen und den tatsächlichen Verteilungen bewerten. Die MSE zählt zu den Regressionsverlustfunktionen, die kontinuierliche Werte vorhersagen, beispielsweise Temperaturen. In diesem Anwendungsfall ist die Verwendung der MSE nicht sinnvoll.

Die Wahl der Verlustfunktion hängt von der Art der Labels ab. Bei der „SparseCategoricalCrossentropy“ werden die Labels als Integer-Werte erwartet, während bei anderen Funktionen die Labels im One-Hot-Encoding-Format vorliegen sollten. Dies wird durch den Parameter „categorical“ im *label_mode* berücksichtigt.

Tabelle 8.3 zeigt die Genauigkeiten bei Verwendung verschiedener Verlustfunktionen.

	Testgenauigkeit	Trainingsgenauigkeit	Validierungsgenauigkeit
Kreuzentropie	85 %	96 %	95 %
Kullback-Leibler Divergenz	82 %	96 %	91 %

Tabelle 8.3: Verlustfunktion

Die Kreuzentropie erweist sich als beste Verlustfunktion auf diesem Modell.

8.3.4 Trainingsprozess optimieren

Es gibt in Keras verschiedene Callbacks, die den Trainingsprozess verbessern können. Diese Funktionen werden während des Trainings aufgerufen, um dynamische Anpassungen vorzunehmen, den Verlauf zu überwachen oder spezifische Aktionen auszuführen.

Das Konzept des Early Stoppings unterbricht das Training vorzeitig unter bestimmten Bedingungen. Eine dieser Bedingungen ist das Absinken der Verlustfunktion. Der Parameter *patience* gibt an, wie viele Epochen ohne Verbesserung abgewartet werden, bevor das Training gestoppt wird. Ein weiterer nützlicher Parameter ist *restore_best_weights*.

Wenn dieser auf True gesetzt wird, speichert das Netzwerk die Gewichte der Epoche mit dem besten Ergebnis, anstatt die Gewichte der letzten Epoche zu behalten. Der folgende Code in Listing 8.7 zeigt, wie dies implementiert werden kann.

Listing 8.7: Early Stopping

```
1 callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min',
      patience=5)
2 history = model.fit(train_ds, epochs=20,
3                     validation_data=(val_ds),callbacks=callback)
```

Beim Early Stopping wurde eine Testgenauigkeit von 85 % erreicht. Das Training hat bei Epoche 17 gestoppt. Abbildung 8.16 zeigt den Genauigkeitsverlauf.

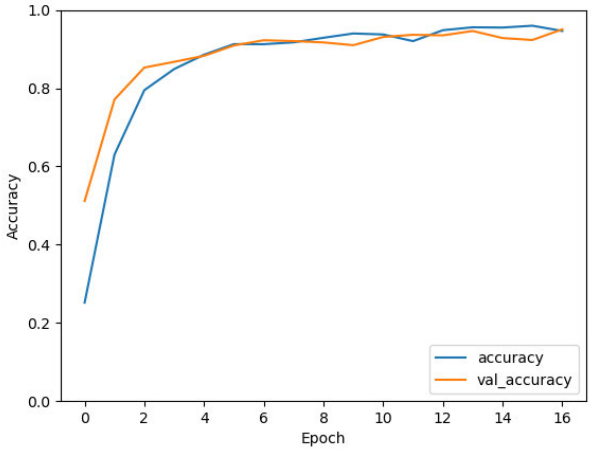


Abbildung 8.16: Genauigkeit Training & Validierung Early-Stopping

8.3.5 Hyperparameter Suche

Hyperparameter sind anpassbare Parameter, die die Genauigkeit des Modells beeinflussen können. Die Aktivierungsfunktionen sind solche veränderbaren Parameter. Im Folgenden wurden einige Aktivierungsfunktionen in der Tabelle 8.4 getestet. Diese Funktionen beziehen sich auf die Faltungsschichten sowie die vorletzte Dense-Schicht. In der letzten Schicht wird die Softmax-Funktion verwendet, um die Wahrscheinlichkeiten der Klassen als Ausgabe zu generieren. Die Aktivierungsfunktionen wurden mit dem Netz aus Listing 8.6 und Early-Stopping getestet.

	Testgenauigkeit	Trainingsgenauigkeit	Validierungsgenauigkeit
ReLU	85 %	96 %	95 %
ELU	81 %	91 %	88 %
SELU	73 %	90 %	84 %
Leaky ReLU	70 %	89 %	90 %

Tabelle 8.4: Aktivierungsfunktionen

Die ReLU erweist sich als beste Lösung für das Modell. Besonders SELU und Leaky ReLu sind für dieses Modell ungeeignet.

Ein weiterer wichtiger Parameter ist die Batch-Size, die angibt, wie viele Daten in einem Stapel verarbeitet werden. Ein Test mit einer Batch-Size von 64 ergab eine Testgenauigkeit von 83 %, während eine Batch-Size von 32 zu besseren Ergebnissen führte. Abbildung 8.17 zeigt den Verlauf für eine Batch-Size von 64, dieser zeigt bis auf eine Ausnahme bei der Validierungsgenauigkeit einen gleichmäßigen Verlauf auf.

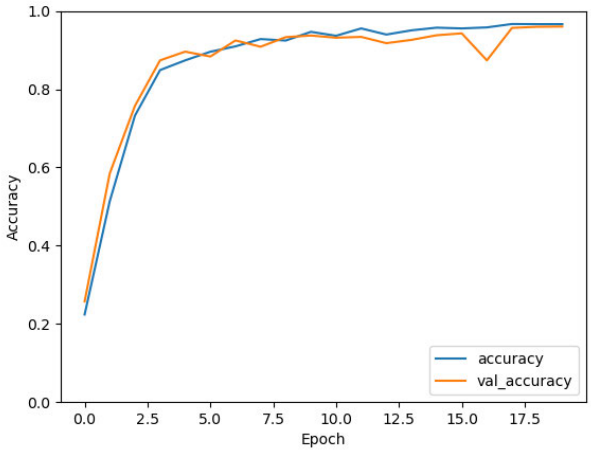


Abbildung 8.17: Genauigkeit Training & Validierung Batch-Size = 64

Die Lernrate ist ein entscheidender Hyperparameter der Optimierer. Zur Optimierung dieses Parameters wird die Keras Hyperparameter Tuning Bibliothek *KerasTuner* verwendet. Dabei wird das Netz in einer Funktion definiert, wobei spezifiziert wird, welche Hyperparameter getestet werden sollen. Anschließend wird die Methode des Hyperparameter-Tunings festgelegt, beispielsweise kann die Random Search verwendet werden.

Mit der Tuner-Klasse werden Parameter wie die Anzahl der Versuche oder die Anzahl der Ausführungen pro Hyperparameter festgelegt. Durch die Funktion *tuner.search* wird das Netz trainiert.

Im Listing A.4 wird das Hyperparameter-Tuning für die Lernrate des Adam-Optimierers dargestellt. In Zeile 25 wird die Lernrate als Hyperparameter definiert, der im Bereich von 0,001 bis 0,009 in Schritten von 0,001 variiert. Die Lernrate wurde in verschiedenen Bereichen getestet, wobei sich herausstellte, dass eine Lernrate von 0,01 und größer als zu groß angesehen wurde und keine zufriedenstellenden Ergebnisse erzielte.

Nachdem das Hyperparameter-Tuning abgeschlossen ist, können die Ergebnisse mithilfe von *tuner.result_summary* ausgegeben werden. Diese Funktion bietet eine Übersicht über die getesteten Hyperparameter-Kombinationen sowie die zugehörigen Ergebnisse.

Listing 8.8: Ergebnis Hyperparameter Tuning ausgeben

```
1 tuner.results_summary()
```

In Abbildung 8.18 sind die drei besten Ergebnisse aus den zwei am besten performenden Bereichen dargestellt.

```
Results summary
Results in result\best_lr
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 00 summary
Hyperparameters:
learning rate: 0.001
Score: 0.9351012110710144

Trial 03 summary
Hyperparameters:
learning rate: 0.002
Score: 0.9057920575141907

Trial 08 summary
Hyperparameters:
learning rate: 0.003
Score: 0.8339148759841919
```

(a) Bereich 0,001 - 0,009

```
Results summary
Results in result\best_lr1
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 05 summary
Hyperparameters:
learning rate: 0.0004
Score: 0.9630146622657776

Trial 00 summary
Hyperparameters:
learning rate: 0.0003
Score: 0.9560362696647644

Trial 03 summary
Hyperparameters:
learning rate: 0.0002
Score: 0.951151430606842
```

(b) Bereich 0,0001 - 0,0009

Abbildung 8.18: Hyperparameter Tuning Lernrate

Als nächster Hyperparameter wird die Anzahl der Neuronen in den Schichten untersucht. Im vorliegenden Fall wurde die Anzahl der Neuronen im vorletzten Dense-Layer getestet.

Das beste Ergebnis wurde bei einer Anzahl von 672 Neuronen erzielt, wie Abbildung 8.19 zeigt.

```
Results summary
Results in result\best_units
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 12 summary
Hyperparameters:
units: 672
Score: 0.9609211683273315

Trial 06 summary
Hyperparameters:
units: 736
Score: 0.959525465965271

Trial 20 summary
Hyperparameters:
units: 608
Score: 0.959525465965271
```

Abbildung 8.19: Hyperparameter Tuning Neuronenanzahl

Die Dropoutrate muss ebenfalls getestet werden. Hier wurde ein Bereich von 0,1 bis 0,9 in 0,1 Schrittweite gewählt. Abbildung 8.20 zeigt das sich eine Rate von 0,5 als am besten herausgestellt hat.

```
Results summary
Results in result\best_drop
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 8 summary
Hyperparameters:
units: 0.1
Score: 0.9581298232078552

Trial 4 summary
Hyperparameters:
units: 0.5
Score: 0.9560362696647644

Trial 1 summary
Hyperparameters:
units: 0.30000000000000004
Score: 0.9518492817878723
```

Abbildung 8.20: Hyperparameter Tuning Dropout Rate

Die drei besten Dropoutraten wurden zusätzlich auf den Testdaten getestet.

Dropoutrate	Testgenauigkeit
0,1	87 %
0,3	89 %
0,5	85 %

Tabelle 8.5: Dropoutrate

Die Dropoutrate von 0,3 stellt sich am besten heraus. Im Folgenden wird ein Netz mit den am besten gefundenen Hyperparametern trainiert. Die Lernrate des Adam Optimierers beträgt 0,0004, die Anzahl der Neuronen im vorletzten Layer beträgt 672 und die Dropoutrate wird auf 0,3 eingestellt. Die Epochenanzahl wird auf 50 erhöht. Abbildung 8.21 zeigt, dass beide Genauigkeiten ziemlich nah an 100 % kommen.

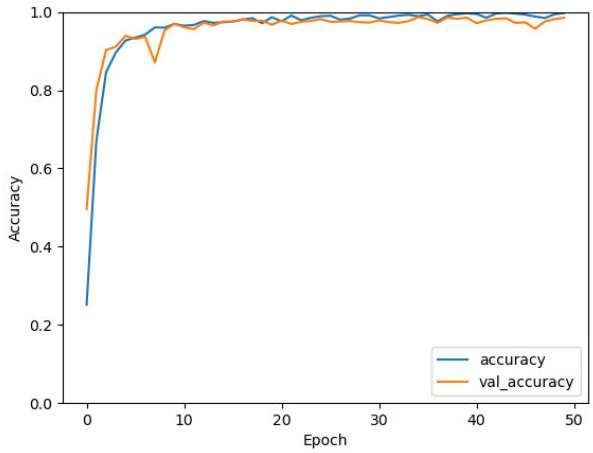


Abbildung 8.21: Genauigkeit Training & Validierung Hyperparameter Tuning

Die Testgenauigkeit beträgt 92 %. Die Genauigkeit hat sich durch das Hyperparameter Tuning verbessert.

8.3.6 Eingangsdaten optimieren

Eine weitere Möglichkeit das Modell zu verbessern ist die Eingangsdaten zu optimieren. Wie in Kapitel 5 bereits beschrieben, wäre eine Möglichkeit die mittleren Bildzeilen zu

mitteln. Dadurch würde das Netz nur schmale Bildstreifen bekommen, welche im Idealfall keinen Hintergrund oder andere Störungen enthalten.

Dabei sind die Bildzeilen identisch, weshalb theoretisch auch nur eine Bildzeile gespeichert werden könnte. Allerdings würde dies zu einer Verkleinerung des Bildes führen, und die Struktur des CNN müsste dementsprechend stärker angepasst werden. Hierbei besteht auch die Möglichkeit, die gemittelten Werte nicht in Form eines Bildes, sondern als Zahlenvektor zu speichern und diesen einem KNN-Algorithmus zu übergeben. Dadurch würde eine Annäherung an das Konzept der „Bildzeilen“ aus Kapitel 5 erfolgen.

Listing 8.9: Codeausschnitt Trainingsdaten optimieren

```
1       int numLines = 8;
2       int startRow = (Imag.rows - numLines) / 2;
3       int endRow = startRow + numLines;
4
5       // Create a matrix to save the new image
6       Mat averagedLines = Mat::zeros(numLines, Imag.cols, Imag.type());
7       for (int j = 0; j < Imag.cols; ++j) {
8         Vec3d sum = Vec3d(0, 0, 0);
9         for (int i = startRow; i < endRow; ++i) {
10          Vec3b pixel = Imag.at<Vec3b>(i, j);
11          sum += Vec3d(pixel[0], pixel[1], pixel[2]);
12        }
13        Vec3b averagedPixel = Vec3b(sum[0] / numLines, sum[1] / numLines, sum[2] /
              numLines);
14        averagedLines.col(j) = averagedPixel;
15      }
16      Imag = averagedLines;
```

Listing 8.9 zeigt den zugehörigen Code im Bildverarbeitungsprogramm zu der Erstellung der Bildzeilen. Dabei werden die acht mittleren Zeilen im Bild gemittelt und als neues Bild gespeichert. Abbildung 8.22 zeigt ein solches Bild.

Abbildung 8.22: Widerstandszeilen

Das Netz wird mit diesen neuen Daten trainiert. Abbildung 8.23 zeigt einen gleichmäßigen Verlauf beider Grafen auf.

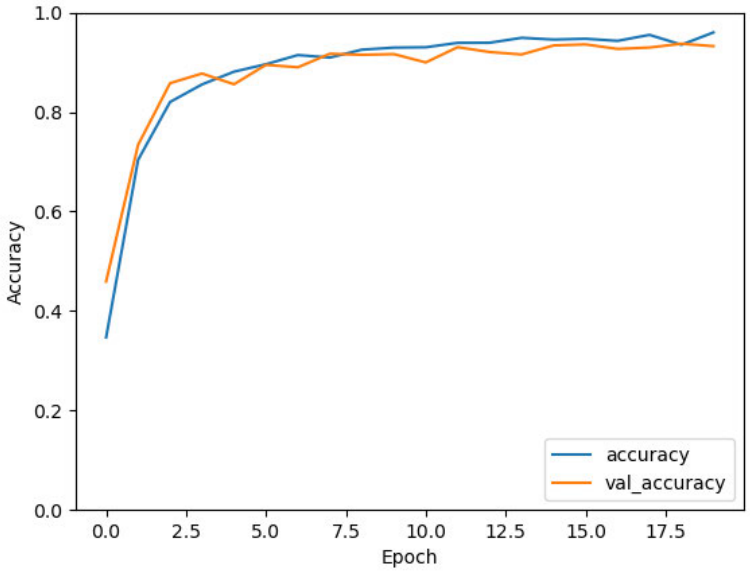


Abbildung 8.23: Genauigkeit Training & Validierung Widerstandszeilen

Die Testgenauigkeit beträgt 88 %. Im Gegensatz zu den 92 % ist dies ein schlechteres Ergebnis. Dies könnte auch daher kommen, dass die Widerstandszeilen nicht immer optimal zusammengefasst werden. Oft wird beispielsweise auch der Hintergrund einbezogen, was dazu führt, dass die Widerstandszeilen heller erscheinen, wie Abbildung 8.24 zeigt. Aus Zeitgründen kann der Algorithmus hierfür allerdings nicht weiter verfolgt werden.

Abbildung 8.24: Fehlerhafte Widerstandszeilen

9 Verifikation und Ergebnisse

In diesem Kapitel werden die Ergebnisse aus Kapitel 7 und 8 zusammengefasst und analysiert. Außerdem wird geprüft, inwiefern die Anforderungen aus Kapitel 4 erfüllt wurden. Zudem erfolgt eine Beschreibung des Programms, das das neuronale Netzwerk aus Kapitel 8 lädt und die Echtzeitklassifizierung eines Widerstands ermöglicht, der vor eine Webcam gehalten wird.

9.1 Bildverarbeitung

9.1.1 Ergebnisse

Der Bildverarbeitungsalgorithmus aus Kapitel 7 lokalisiert und isoliert Widerstände mit einer durchschnittlichen Genauigkeit von 74,2 % auf einem Datensatz mit verschiedenen Hintergründen und Lichtverhältnissen. Bei Verwendung eines Datensatzes mit weißem Hintergrund steigt die durchschnittliche Genauigkeit des Algorithmus auf 96,6 % an.

9.1.2 Verifikation

In diesem Abschnitt erfolgt die Überprüfung der Anforderungen aus Kapitel 4. In der „Status“-Spalte bedeutet ein ✓, dass die Anforderung vollständig erfüllt ist, ein o zeigt an, dass die Anforderung größtenteils erfüllt ist, und ein x bedeutet, dass die Anforderung nicht erfüllt ist. Tabelle 9.1 überprüft die Anforderungen, die zur Bildverarbeitung gehören.

Anforderung	Kurze Beschreibung	Status
SW-1	Visual Studio	✓
SW-2	Trainings- und Testdatensatz	✓
SW-5	Modularer Aufbau	✓
SW-7	Widerstand beliebige Rotation	✓
SW-8	Mindestens zwei Ansätze	✓
SW-10-opt	Widerstände bei wechselnden Hintergrund erkennen	o
SW-11-opt	Widerstände bei wechselnden Lichtverhältnissen erkennen	o
HW-2	Kohleschichtwiderstände	✓
HW-3-opt	Metallschichtwiderstände	x

Tabelle 9.1: Anforderungen Bildverarbeitung Überprüfung

Die ersten fünf Software Anforderungen gemäß Tabelle 9.1 wurden erfüllt. Die beiden optionalen Anforderungen wurden teilweise erfüllt. Widerstände werden unter verschiedenen Hintergründen und Lichtverhältnissen erkannt, jedoch nicht so zuverlässig wie unter optimalen Bedingungen mit weißem gut beleuchtetem Hintergrund. Aus Zeitgründen war es nicht möglich, zusätzlich zu den Kohleschichtwiderständen auch Metallschichtwiderstände zu klassifizieren. Diese Anforderung wäre jedoch ohnehin optional.

9.2 Deep Learning

9.2.1 Ergebnisse

Das beste Netz aus Kapitel 8 erreichte eine Genauigkeit von 92 %. Es wurde mit dem Befehl

```
model.save('resistors.keras')
```

gespeichert. Im nächsten Schritt werden die falsch klassifizierten Widerstände analysiert. Hierfür bietet sich eine Konfusionsmatrix an. Eine solche Matrix zeigt die Vorhersagen der Widerstandswerte in Bezug auf die tatsächlichen Widerstandswerte. Im Idealfall zeigt sich dabei eine deutlich ausgeprägte Hauptdiagonale.

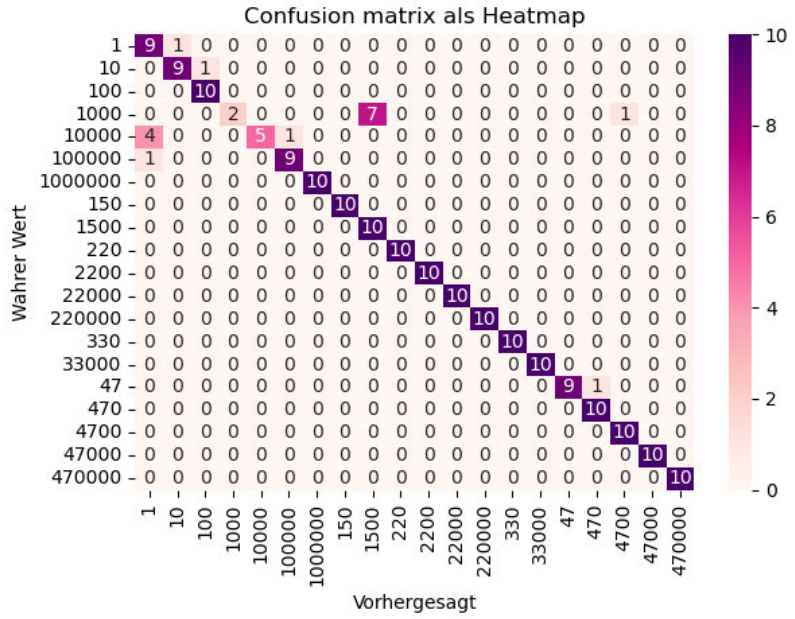


Abbildung 9.1: Confusion Matrix

Abbildung 9.1 zeigt die Konfusionsmatrix des besten Netzes. Es fällt auf, dass die 1 $k\Omega$ Widerstände größtenteils falsch klassifiziert werden. Lediglich die Hälfte der 10 $k\Omega$ Widerstände wird korrekt klassifiziert. Die anderen Widerstände werden fast immer richtig klassifiziert.

Es wäre zudem interessant zu erfahren, welche Widerstandswerte auf dem zweiten und dritten Platz der Vorhersagen landen. Tabelle 9.2 listet alle falsch klassifizierten Labels auf und zeigt die Vorhersagen für den zweiten und dritten Platz.

In 14 von 17 Fällen landet das tatsächliche Label auf dem zweiten oder dritten Platz der Vorhersagen. In drei Fällen jedoch erfolgt eine vollständig falsche Klassifizierung.

9.2.2 Verifikation

Tabelle 9.3 zeigt die Anforderungen aus Kapitel 4, die im Kontext zum Deep Learning stehen.

Wahres Label	Zweite Wahl	Dritte Wahl
1 Ω	1 Ω	100 $k\Omega$
10 Ω	10 Ω	150 Ω
1 $k\Omega$	1 $k\Omega$	10 $k\Omega$
1 $k\Omega$	1 $k\Omega$	10 $k\Omega$
1 $k\Omega$	1 $k\Omega$	1,5 $k\Omega$
1 $k\Omega$	10 $k\Omega$	4,7 Ω
1 $k\Omega$	100 Ω	10 $k\Omega$
1 $k\Omega$	1 $k\Omega$	10 $k\Omega$
1 $k\Omega$	10 $k\Omega$	1 $k\Omega$
1 $k\Omega$	1 $k\Omega$	4,7 $k\Omega$
10 $k\Omega$	10 $k\Omega$	100 $k\Omega$
10 $k\Omega$	10 $k\Omega$	100 $k\Omega$
10 $k\Omega$	10 $k\Omega$	1 $k\Omega$
10 $k\Omega$	100 $k\Omega$	1 $M\Omega$
10 $k\Omega$	1 $k\Omega$	10 $k\Omega$
100 $k\Omega$	100 $k\Omega$	47 Ω
47 Ω	47 Ω	10 Ω

Tabelle 9.2: Falsch klassifizierte Widerstände

Anforderung	Kurze Beschreibung	Status
SW-4	Testgenauigkeit $\geq 90\,\%$	✓
SW-6	Widerstandswert in Ω	✓
SW-9	Verwendung CNN	✓
SW-10-opt	Widerstände bei wechselnden Hintergrund klassifiziert	o
SW-11-opt	Widerstände bei wechselnden Lichtverhältnissen klassifiziert	o

Tabelle 9.3: Anforderungen Deep Learning Überprüfung

Die ersten drei Softwareanforderungen wurden erfüllt. Die beiden optionalen Anforderungen wurden, wie in Tabelle 9.1 teilweise erfüllt.

9.3 Webcam Widerstandsklassifizierung

Im letzten Schritt wird ein Python Programm erstellt, welches das gespeicherte Netz lädt, um den Widerstand in der Webcam live zu lokalisieren und zu klassifizieren. Hierbei wurde der Bildverarbeitungsalgorithmus von C++ in Python umgewandelt. Hier

zeigt sich ein großer Vorteil von OpenCV, da es sowohl für C++ als auch für Python zur Verfügung steht, wodurch die Portierung des C++-Programms in Python problemlos möglich war. Ursprünglich war die Idee, das gespeicherte neuronale Netz in das C++-Bildverarbeitungsprogramm zu integrieren und den lokalisierten Widerstand zu klassifizieren. Es gab jedoch Schwierigkeiten beim Laden des Netzes in C++, mehr dazu im Ausblick in Kapitel 10.

Es wird eine Funktion erstellt, die die Webcam des Laptops öffnet. Das Öffnen und der Zugriff auf die Frames der Webcam erfolgen mithilfe der OpenCV-Klasse *VideoCapture*, die eine Python API zum Lesen von Bildsequenzen bereitstellt. Auf das Kamerabild wird die Schwellwert-Methode mit einem Schwellwert von 80 angewendet. Im Gegensatz zum Programm aus Kapitel 7 wird hier ein fester Schwellwert verwendet, da der adaptive Schwellwert schlechte Ergebnisse geliefert hat. Je nach Lichtverhältnissen muss möglicherweise der feste Schwellenwert angepasst werden. Anschließend wird das Opening auf das Binärbild angewendet, und das resultierende Bild wird der Funktion *live_find_object* übergeben. Das Listing 9.1 zeigt den Code.

Die Funktion *live_find_object* sucht im übergebenen Bild nach Konturen und berechnet deren Momente. Anschließend wird die Funktion *identify_resistor* aufgerufen welche prüft, ob es sich bei der Kontur um einen Widerstand handelt. Der entsprechende Code findet sich im Listing 9.3. Handelt es sich um einen Widerstand, wird die Bounding Box berechnet und im Kamerabild angezeigt. Der zugeschnittene Bereich des Widerstands wird skaliert und dem neuronalen Netzwerk übergeben. Mithilfe des Befehls *model.predict* wird eine Vorhersage für das Widerstandslabel getroffen. Die drei wahrscheinlichsten Labels werden neben der Bounding Box angezeigt. Der entsprechende Code ist im Listing 9.2 zu finden.

Die Abbildung 9.2 zeigt eine Live Klassifizierung eines 1 Ω Widerstandes. Sie veranschaulicht ein beispielhaftes Kamerabild der Live-Klassifizierung.

Listing 9.1: capture_camera Funktion

```
model = tf.keras.models.load_model('resistors.keras')
img_height = 40
img_width = 70
class_names=['1', '10', '100', '1000', '10000', '100000', '1000000', '150', '1500', '220', '2200', '22000', '220000', '330', '33000', '47', '470', '4700', '47000', '470000']
def capture_camera():
    #open webcam
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("ERROR: Cannot open camera")
        return

    found = 0
    morph_size = 7
    scale_percent = 15
    is_invert_binary = True
    while True:
        ret, frame = cap.read()

        if not ret:
            print("ERROR: Cannot read frame")
            break
        resistor_roi = np.empty(shape=(frame.shape[0], frame.shape[1], 3), dtype=np.uint8)
        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
      #convert frame to binary with treshhold
        _, frame_bin = cv2.threshold(frame_gray, 80, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV)
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (morph_size, morph_size))
        frame_bin = cv2.morphologyEx(frame_bin, cv2.MORPH_OPEN, kernel)
      #open function to find the resistor
        found = live_find_object(frame, frame_bin,resistor_roi)
        cv2.imshow("Camera [press any key to quit]", frame)
        cv2.imwrite("outcome.jpg",frame)
        if cv2.waitKey(30) >= 0:
            break

    cap.release()
    cv2.destroyAllWindows()

    return resistor_roi

captured_frame = capture_camera()
```

Listing 9.2: live_find_object Funktion

```
def live_find_object(img, bin_img,resistor_roi):
    contours, _ = cv2.findContours(bin_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    biggest_label = -1
    exz = []
    ori = []

    try:
        for contour in contours:
        # calculate Moments
            moments = cv2.moments(contour)
            a = moments['mu20'] - moments['mu02']
            b = moments['mu20'] + moments['mu02']
            exz_value = (a**2 + (4 * moments['mu11']**2)) / (b**2)
            exz.append(exz_value)
            orientation = ((0.5 * math.atan2(moments['mu20'] - moments['mu02'], 2 * moments['mu11'])) * (180 / np.pi)) + 45
            ori.append(orientation)
        #proof if contur could be resistor
        biggest_label = identify_resistor(contours,moments, exz, ori, bin_img)
        if biggest_label != -1:
            #calculate Bounding Box from contur
            box = cv2.boundingRect(contours[biggest_label])
            x, y, w, h = cv2.boundingRect(contours[biggest_label])
            resistor_roi = img[y:y+h, x:x+w]
      #resize resistor cutout for the net
            img_res = cv2.resize(resistor_roi, ( img_width,img_height))
            cv2.imwrite("webcam.png", img_res)
            cv2.rectangle(img, box, (0, 0, 255), 1)
            img_array = tf.expand_dims(img_res, 0)
      #make prediction for the resistor cutout
            predictions = model.predict(img_array, verbose=0)
            predicted_class = tf.argmax(predictions, axis=1).numpy()
            predicted_class_name = class_names[predicted_class[0]]
            predicted_classes = np.argsort(-predictions)[0][:3]
            predicted_class_name_two = class_names[predicted_classes[1]]
            predicted_class_name_three = class_names[predicted_classes[2]]
            omega="␣Omega"
      #show the predicted results beside the Bounding Box
            cv2.putText(img,"1.␣"+predicted_class_name +omega, (x, y - 80), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
            cv2.putText(img,"2.␣"+ predicted_class_name_two+omega, (x, y - 40), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
            cv2.putText(img,"3.␣"+ predicted_class_name_three+omega, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
    except Exception as e:
        return 0

    return 1 if biggest_label != -1 else 0
```

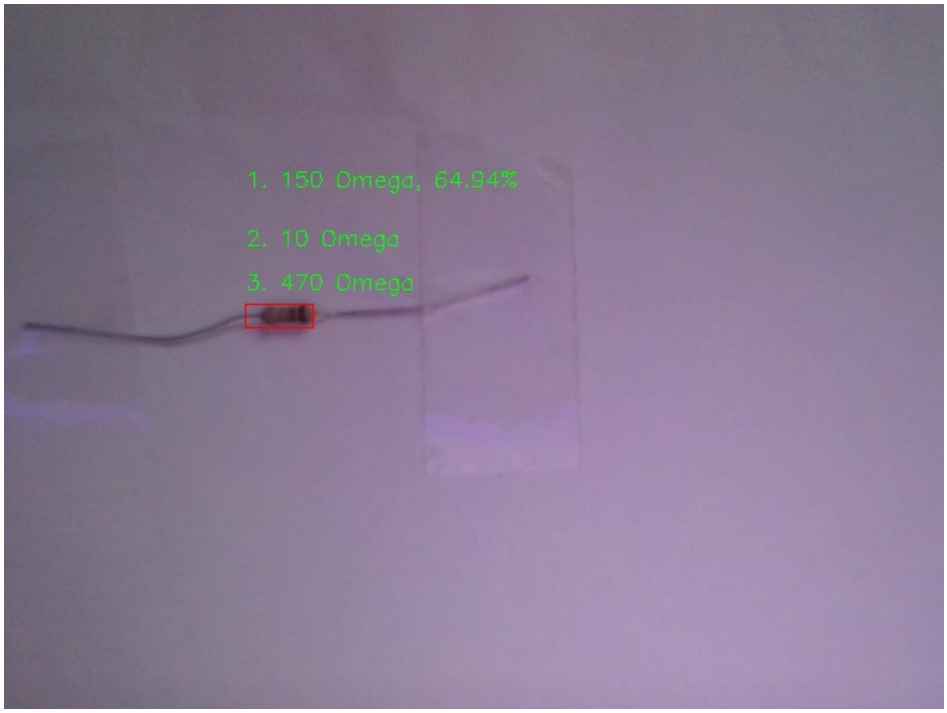


Abbildung 9.2: Live Klassifizierung

Der Widerstand wird mit einer 64,94 % Wahrscheinlichkeit als 150 Ω klassifiziert. Dies ist leider ein falsches Ergebnis. Betrachtete man den Widerstandsausschnitt, der in das Netz gegeben wird, fällt sofort die schlechte Qualität auf. Die Webcam besitzt ein Auflösung von 640x480, der Ausschnitt wird auf 70x40 skaliert. Abbildung 9.3 zeigt einen solchen Widerstandsausschnitt.

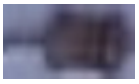


Abbildung 9.3: Widerstandsausschnitt Live Klassifizierung

Betrachtet man Abbildung 9.3 ist die falsche Klassifizierung nicht verwunderlich. Es ist schwierig die Farbstreifen des Widerstandes zu erkennen.

Listing 9.3: identify_resistor Funktion

```
def identify_resistor(contours, mu, exz, ori, bin_img):
    cogx = [0] * len(contours)
    cogy = [0] * len(contours)
    #calculate moments
    for i in range(len(contours)):
        moments = cv2.moments(contours[i], False)
        a = moments['mu20'] - moments['mu02']
        b = moments['mu20'] + moments['mu02']
        exz[i] = (a**2 + (4 * moments['mu11']**2)) / (b**2)
        ori[i] = ((0.5 * math.atan2(moments['mu20'] - moments['mu02'], 2 * moments
    ['mu11'])) * (180 / math.pi)) + 45
        cogx[i] = moments['m10'] / moments['m00'] if moments['m00'] != 0 else 0
        cogy[i] = moments['m01'] / moments['m00'] if moments['m00'] != 0 else 0

    rows, cols = bin_img.shape[:2]
    tol_edge = cols / 9  # Colum tolerance
    biggest = 0
    biggest_label = -1

    for label in range(len(contours)):
  #proof if contur could be resistor
        if len(contours[label]) > biggest and 0.30 < exz[label] < 0.87 \
            and tol_edge < cogx[label] < (cols - tol_edge) \
            and tol_edge < cogy[label] < (rows - tol_edge)
      #save and return label from resistor contur
            biggest = len(contours[label])
            biggest_label = label

    return biggest_label
```

9.3.1 Verifikation

Auch wenn die Live Klassifizierung am Ende ein falsches Ergebnis liefert, sind die Anforderungen erfüllt. Die Tabelle 9.4 zeigt die Anforderungen, welche zur Live Klassifizierung gehören.

Anforderung	Kurze Beschreibung	Status
SW-3	BB um Widerstand und Widerstandswert daneben	✓
HW-1	Nutzung Webcam	✓

Tabelle 9.4: Anforderungen Live Klassifizierung Überprüfung

Die Nutzung der Webcam und Lokalisierung des Widerstands funktionieren. Die Klassifizierung ist aufgrund der schlechten Qualität des Widerstandsausschnitts unzuverlässig.

10 Fazit und Ausblick

Das Konzept eines Bildverarbeitungsprogrammes, welches Widerstände lokalisiert und ausschneidet, wurde erfolgreich umgesetzt. Dabei wurden mehrere Ansätze untersucht und getestet, um ein zuverlässiges Verfahren zu identifizieren. Durch die Erstellung eines Datensatzes mit Widerstandsbildern konnte der Bildverarbeitungsalgorithmus getestet werden, um sowohl Trainings- als auch Testdaten für das CNN zu generieren. Das CNN wurde mithilfe der TensorFlow-Bibliothek aufgebaut, trainiert und optimiert, um präzise und effiziente Ergebnisse zu erzielen. Das gespeicherte Netzwerk kann in Python geladen werden und ermöglicht es, mithilfe der Webcam eine Live Klassifizierung durchzuführen.

Am Ende zeigt das Live-Klassifizierungssystem leider ungenaue Vorhersagen. Um diese Probleme anzugehen, sind mehrere mögliche Verbesserungen identifiziert worden. Zunächst sollte die Qualität der extrahierten Widerstandssegmente verbessert werden, um genauere und präzisere Klassifizierungen zu ermöglichen. Dies könnte durch Bildverarbeitungsalgorithmen geschehen, die die Qualität der Ausschnitte verbessern. Ein entscheidender Schritt besteht darin zu überprüfen, welche Farbringe tatsächlich erkannt werden. Dies könnte durch die Anwendung eines Bildverarbeitungsalgorithmus geschehen. Dies würde eine detaillierte Nachverfolgung der Reaktion des Netzwerks auf die Eingabedaten ermöglichen. Eine weitere Möglichkeit zur Verbesserung besteht darin, die Erkennung und Darstellung der Farbringe der Widerstände zu optimieren. Dadurch kann überprüft werden, welche Farbringe erkannt werden und wie diese Information genutzt werden kann, um die Entscheidungen des neuronalen Netzes besser nachvollziehbar zu machen. Es besteht auch Potenzial, den Bildverarbeitungsalgorithmus weiter zu verbessern, um seine Genauigkeit bei der Identifizierung von Widerständen in störenden oder unruhigen Umgebungen zu steigern.

Ein weiteres Problem bestand darin, das Netzwerk in ein C++-Programm zu integrieren. OpenCV bietet eine Klasse, die die Einbindung neuronaler Netze im ONNX- oder PB-Format ermöglicht. Jedoch stellte sich in der Praxis heraus, dass die Umwandlung

des Keras-Modells in ein solches Format herausfordernd war. Ebenso gestaltete sich die Einbindung von TensorFlow in C++ als problematisch. Diese Herausforderungen könnten in zukünftigen Arbeiten angegangen und implementiert werden, um eine nahtlose Integration der Modelle in C++ zu gewährleisten.

Trotz der Herausforderungen führte die Arbeit letztlich zu einem zufriedenstellenden Ergebnis. Die definierten Anforderungen wurden erfüllt und zahlreiche wertvolle Erkenntnisse wurden gewonnen. Diese Arbeit legt eine solide Grundlage für künftige Forschungen und Entwicklungen auf diesem Gebiet. Die Veröffentlichung dieser Arbeit könnte darüber hinaus wertvolle Informationen für den Lehrbereich bereitstellen, die als Grundlage für zukünftige Lerninhalte dienen könnten.

Literaturverzeichnis

[1] Burak Çatalbas, Ömer M.: Deep learning with ExtendeD Exponential Linear Unit (DELU). In: *Springer* (2023). – URL https://link.springer.com/article/10.1007/s00521-023-08932-z

[2] Carsten Lanquillon, Sigurd S.: *Knowledge Science – Grundlagen*. 1. Ausgabe. Springer Vieweg Wiesbaden, 2023

[3] Dahlkemper, Jörg: *Deep Learning in der Bildverarbeitung, Folien*. Folien zur Lehrveranstaltung, HAW Hamburg, 2020. – http://www.haw-hamburg.de/joerg-dahlkemper

[4] Fleig, Dr. J.: *Was sind Stakeholder und was bedeutet der Stakeholder-Ansatz?*. – URL https://www.business-wissen.de/hb/was-sind-stakeholder-und-was-bedeutet-der-stakeholder-ansatz/. – Zugriffsdatum: 12.05.2023

[5] Franz, M. O.: *Merkmale von Bildregionen, Einführung in Spektraltechniken*. 12.12.2007. – URL http://www-home.fh-konstanz.de/~mfranz/ibv_files/lect09_spectr.pdf. – Zugriffsdatum: 21.06.2023

[6] Frochte, Jörg: *Maschinelles Lernen: Grundlagen und Algorithmen in Python*. 2. Ausgabe. Carl Hanser Verlag, 2019

[7] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning. Das umfassende Handbuch - Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze*. 1. Ausgabe. mitp Verlags GmbH & Company KG, 2018. – ISBN 978-3-958-45701-0

[8] Hensel, Marc: *Digitale Bildverarbeitung*. Skript zur Lehrveranstaltung, HAW Hamburg, 2023. – http://www.haw-hamburg.de/marc-hensel

[9] Houmani-Farahan, Borna: *OhmVision*. – URL https://github.com/dishonesthips/OhmVision. – Zugriffsdatum: 21.06.2023

[10] Jens Kürbig, Martina S.: *Seminar: Bildsegmentierung und Computer Vision*. 2005/2006. – URL https://www.mathematik.uni-ulm.de/stochastik/lehre/ws05_06/seminar/ausarbeitung_sauter.pdf. – Zugriffsdatum: 06.07.2023

[11] Kompendium, Elektronik: *Kennzeichnung von Widerständen mit Widerstandsfarbcode*. – URL https://www.elektronik-kompendium.de/sites/bau/1109051.htm. – Zugriffsdatum: 23.11.2023

[12] Mitani, Yoshihiro ; Yoshimura, Wataru ; Hamamoto, Yoshihiko: An evaluation of classifiers for reading resistor colors. In: *Proceedings of the 2022 5th Artificial Intelligence and Cloud Computing Conference* (2022). – URL https://api.semanticscholar.org/CorpusID:258240373

[13] Muminovic, Mia ; Sokic, Emir: Automatic Segmentation and Classification of Resistors in Digital Images. In: *2019 XXVII International Conference on Information, Communication and Automation Technologies (ICAT)*, URL https://ieeexplore.ieee.org/document/8939034, 2019, S. 1–6

[14] OpenCV: *Geometric Transformations of Images*. – URL https://docs.opencv.org/3.4/da/d6e/tutorial_py_geometric_transformations.html. – Zugriffsdatum: 31.10.2023

[15] OpenCV: *Image Filtering*. – URL https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gac05a120c1ae92a6060dd0db190a61afa. – Zugriffsdatum: 12.01.2024

[16] OpenCV: *OpenCV About*. – URL https://opencv.org/about/. – Zugriffsdatum: 01.06.2023

[17] Pascal Niklaus, Gian U.: Automated Resistor Classification. In: *Eidgenössische Technische Hochschule Zürich* (2015). – URL https://pub.tik.ee.ethz.ch/students/2014-HS/GA-2014-02.pdf

[18] Sonnet, Daniel: *Neuronale Netze kompakt - Vom Perceptron zum Deep Learning*. 1. Ausgabe. Springer Fachmedien Wiesbaden, 2022. – ISBN 978-3-658-29080-1

[19] Soud, Elias: *Classification of Electrical Resistors by a Deep LearningModel embedded in an Application for Mobile Devices*. 18.07.2022

[20] Spektrum: *KontrastT*. – URL https://www.spektrum.de/lexikon/optik/kontrast/1639. – Zugriffsdatum: 03.11.2023

[21] Süsse, Herbert ; Rodner, Erik: *Bildverarbeitung und Objekterkennung*. 1. Ausgabe. Springer Vieweg, 2014. – ISBN 978-3-8348-2606-0

[22] u.a, Alfred N.: *Bildverarbeitung: Band II des Standardwerks Computergrafik und Bildverarbeitung*. 4. Ausgabe. Springer Vieweg Wiesbaden, 2019

[23] u.a, Laith A.: Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. In: *Springer Open*. – URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8

[24] W. Burger, M. J. B.: *Regionen in Binärbildern*. In: *Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java*, Springer, 2015. – URL https://link.springer.com/content/pdf/10.1007/978-3-642-04604-9_10.pdf/

[25] Zeilinger, Martin: *Kamerabasierte Klassifizierung von Schichtwiderständen mittels Bildverarbeitung und Deep Learning*, HAW Hamburg, Masterarbeit, 2024

[26] ZenBusiness: *Advantages and disadvantages of JPEG format [Infographic]*. – URL https://www.zenbusiness.com/blog/jpeg/. – Zugriffsdatum: 07.09.2023

A Anhang

A.1 Code

Listing A.1: Codeausschnitt Main Funktion

```
for (int u = 0; u <= 200; u++) {
  // Load image from file
  string inputImagePath = "../../../Bilder/Datensatz/";
  string inputDateiPath = "../../../Bilder/Datensatz/";
  string outputImagePath = "../../../Bilder/Datensatz/";

  inputDateiPath.append(res);
  inputDateiPath.append("/");
  inputImagePath.append(res);
  inputImagePath.append("/");
  outputImagePath.append(res);
  outputImagePath.append("/Res/");
  inputImagePath.append(res);
  inputImagePath.append("_");
  inputImagePath.append(to_string(u));
  inputDateiPath.append(res);
  inputDateiPath.append("_");
  inputDateiPath.append(to_string(u));
  inputDateiPath.append(".txt");
  inputImagePath.append(".jpg");
  outputImagePath.append(res);
  outputImagePath.append("_");
  outputImagePath.append(to_string(u));
  outputImagePath.append(".jpg");

  cout << "Load : " << u << endl;
  image = cv::imread(inputImagePath, cv::IMREAD_COLOR);
  img = cv::imread(inputImagePath, cv::IMREAD_GRAYSCALE);

  if (image.empty()) {
    cout << "[ERROR] Cannot open image: " << inputImagePath << endl;
    return 0;
  }
  string filename(inputDateiPath);
  ifstream input_file(filename);
```

```
36      //open txt File
37      if (!input_file.is_open()) {
38        cerr << "Could not open the file - '"
39          << filename << "'" << endl;
40        return EXIT_FAILURE;
41      }
42      while (input_file >> number) {
43        box_ref[n] = number * 100;
44        n++;
45      }
46      n = 0;
47      input_file.close();
48      width = int(img.cols * scale_percent / 100);
49      height = int(img.rows * scale_percent / 100);
50      //if picture to big resize it
51      if (image.cols > 1000 || image.rows > 1000) {
52        resize(img, img, Size(width, height));
53        resize(image, image, Size(width, height));
54      }
55    }
```

Listing A.2: Codeausschnitt findObject_canny 3

```
1  //convert to LAB colors
2  Mat blur = imag.clone();
3  medianBlur(imag, blur, 11);
4  Mat lab = imag.clone();
5  cvtColor(imag, lab, COLOR_BGR2Lab);
6  Mat vertical_imag = lab.clone();
7  //calculate deviations from columns
8  for (int colum = 0; colum < lab.cols; colum++) {
9    for (int row = 0; row < (lab.rows); row++) {
10     sum1_L += lab.at<Vec3b>(row, colum)[0];
11     sum1_a += lab.at<Vec3b>(row, colum)[1];
12     sum1_b += lab.at<Vec3b>(row, colum)[2];
13   }
14   mean_L = sum1_L / lab.rows;
15   mean_a = sum1_a / lab.rows;
16   mean_b = sum1_b / lab.rows;
17   for (int i = 0; i < lab.rows; i++)
18   {
19     sum2_L += (lab.at<Vec3b>(i, colum)[0] - mean_L) * (lab.at<Vec3b>(i, colum)[0]
       - mean_L);
20     sum2_a += (lab.at<Vec3b>(i, colum)[1] - mean_a) * (lab.at<Vec3b>(i, colum)[1]
       - mean_a);
21     sum2_b += (lab.at<Vec3b>(i, colum)[2] - mean_b) * (lab.at<Vec3b>(i, colum)[2]
       - mean_b);
22   }
23   var_L = sum2_L / lab.rows;
24   std_L = sqrt(var_L);
25   var_a = sum2_a / lab.rows;
26   std_a = sqrt(var_a);
27   var_b = sum2_b / lab.rows;
28   std_b = sqrt(var_b);
29   sum1_L = 0;
30   sum2_L = 0;
31   sum1_a = 0;
32   sum2_a = 0;
33   sum1_b = 0;
34   sum2_b = 0;
35   deviation = 1 * std_L + 10 * (std_a + std_b);
36   //proof deviations
37   if (deviation > 70 && found == 0 && colum > lab.cols / 3) {
38     x_0 = colum;
39     found = 1;
40   }
41   if (deviation < 40 && found == 1 && (colum - x_0 > lab.cols / 15)) {
42     x_1 = colum;
43     found = 0;
44     break;
45   }
46 }
```

Listing A.3: Codeausschnitt findObject_canny 4

```
1  //calculate deviations from rows
2  if (x_1 != 0) {
3    Rect vertical;
4    vertical.x = x_0;
5    vertical.y = 0;
6    vertical.width = x_1 - x_0;
7    vertical.height = lab.rows;
8    vertical_imag = vertical_imag(vertical);
9    Mat devi = vertical_imag.clone();
10   std::vector<cv::Mat> lab_planes(3);
11   cv::split(lab, lab_planes);
12   deviation = 0;
13   for (int row = 0; row < vertical_imag.rows; row++) {
14     for (int colum = 0; colum < (vertical_imag.cols); colum++) {
15       sum1_L += vertical_imag.at<Vec3b>(row, colum)[0];
16       sum1_a += vertical_imag.at<Vec3b>(row, colum)[1];
17       sum1_b += vertical_imag.at<Vec3b>(row, colum)[2];
18     }
19     mean_L = sum1_L / vertical_imag.cols;
20     mean_a = sum1_a / vertical_imag.cols;
21     mean_b = sum1_b / vertical_imag.cols;
22     for (int i = 0; i < vertical_imag.cols; i++)
23     {
24       sum2_L += (vertical_imag.at<Vec3b>(row, i)[0] - mean_L) * vertical_imag.at<
       Vec3b>(row, i)[0] - mean_L);
25       sum2_a += (vertical_imag.at<Vec3b>(row, i)[1] - mean_a) * (vertical_imag.at<
       Vec3b>(row, i)[1] - mean_a);
26       sum2_b += (vertical_imag.at<Vec3b>(row, i)[2] - mean_b) * (vertical_imag.at<
       Vec3b>(row, i)[2] - mean_b);
27     }
28     var_L = sum2_L / vertical_imag.cols;
29     std_L = sqrt(var_L);
30     var_a = sum2_a / vertical_imag.cols;
31     std_a = sqrt(var_a);
32     var_b = sum2_b / vertical_imag.cols;
33     std_b = sqrt(var_b);
34     sum1_L = 0;
35     sum2_L = 0;
36     sum1_a = 0;
37     sum2_a = 0;
38     sum1_b = 0;
39     sum2_b = 0;
40     deviation = 1 * std_L + 10 * (std_a + std_b);

42     if (deviation > 40 && found == 0) {
43       y_0 = row;
44       found = 1;
45     }
46     if (deviation <= 40 && found == 1 && row - y_0 > 5) {
```

```
47          y_1 = row;
48          break;
49        }
50      }
51      width = (x_1 - x_0);
52      height = (y_1 - y_0);
```

Listing A.4: Keras Hyperparameter Tuning

```
def build_model(hp):

    num_classes=20
    img_height = 40
    img_width = 70
    model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    tf.keras.layers.BatchNormalization(epsilon=0.001, momentum=0.99),
    tf.keras.layers.RandomFlip("horizontal"),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(256, 3, activation='relu'),
    tf.keras.layers.Conv2D(256, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.GaussianDropout(0.3),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
    optimizer = tf.keras.optimizers.Adam(learning_rate=hp.Float("learning_rate",
    min_value=0.001, max_value=0.009,           step=0.001))
    loss=tf.keras.losses.SparseCategoricalCrossentropy()
    model.compile(optimizer=optimizer,
                  loss=loss,
                  metrics=['accuracy'])

    return model

build_model(keras_tuner.HyperParameters())
tuner = keras_tuner.RandomSearch(
    hypermodel=build_model,
    objective="val_accuracy",
    max_trials=10,
    executions_per_trial=1,
    overwrite=True,
    directory="result",
    project_name="best_units",
)
tuner.search_space_summary()
tuner.search(train_ds, epochs=8,
                      validation_data=(val_ds))
```

A.2 Abbildungen

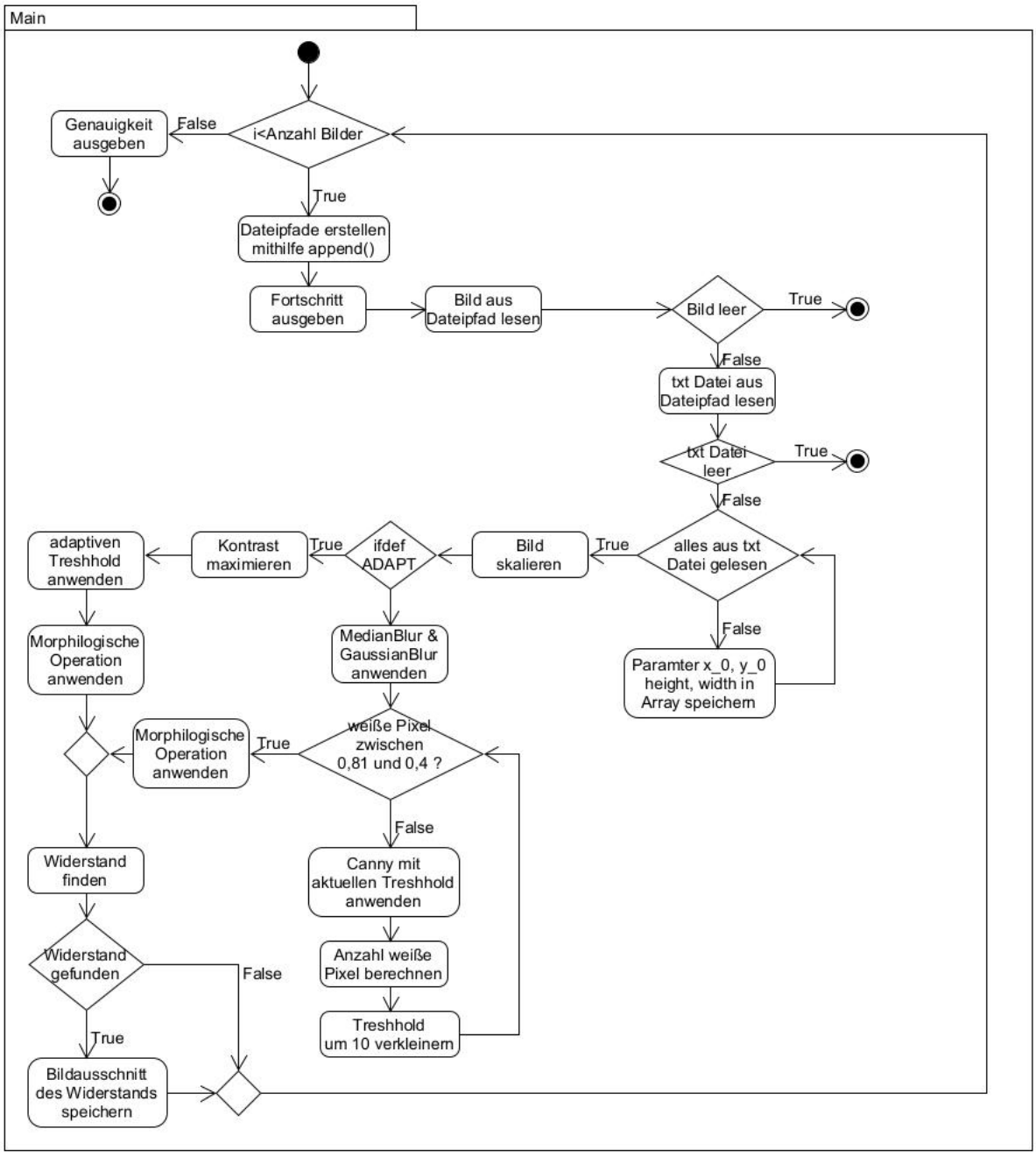


Abbildung A.1: Ablauf Main Funktion

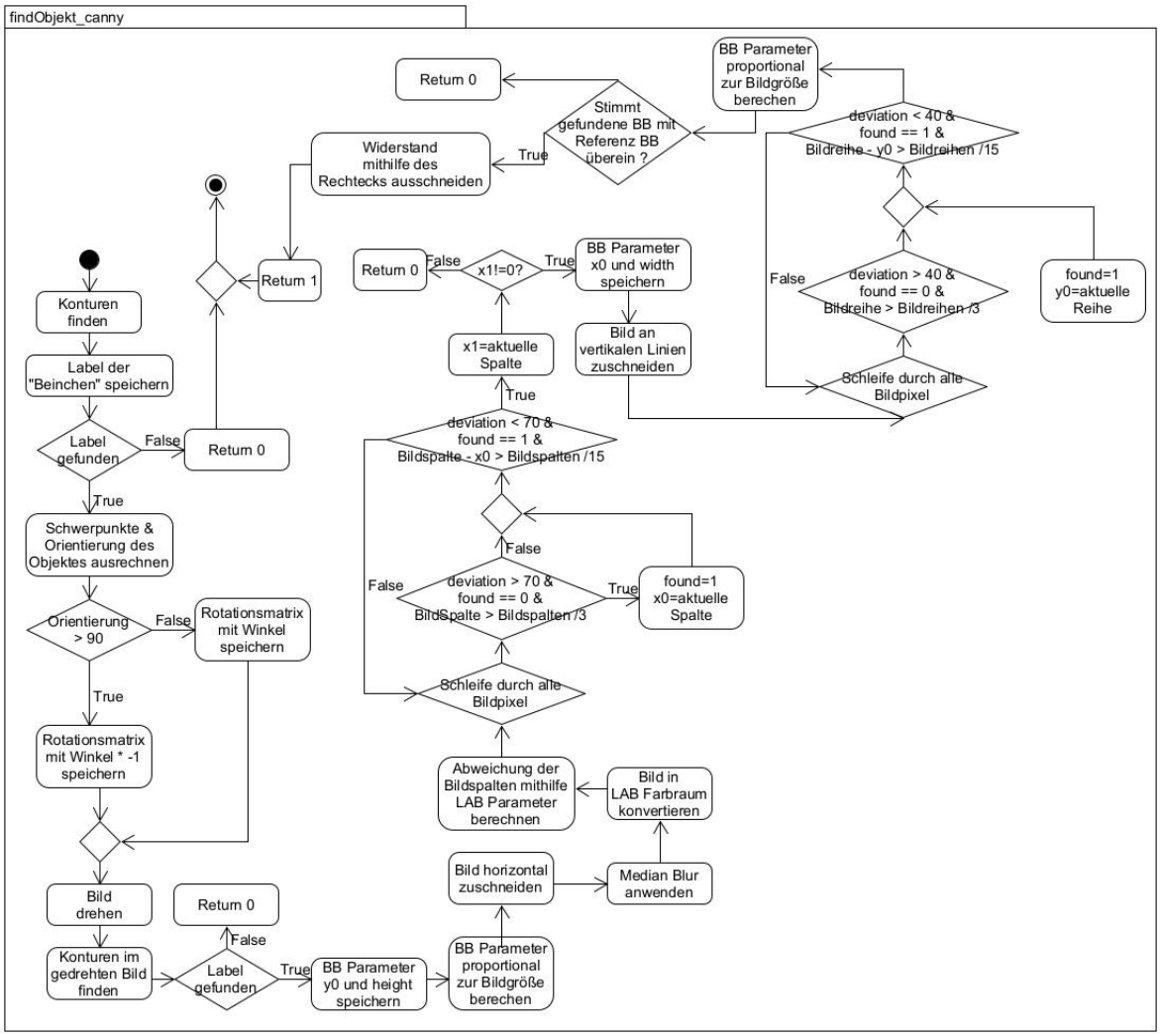


Abbildung A.2: Ablauf findObject_canny Funktion

Erklärung zur selbstständigen 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