Technische Universität Hamburg-Harburg
Vision Systems

Prof. Dr.-Ing. R.-R. Grigat

Optimierung der Topologie Neuronaler Netzwerke unter Verwendung eines evolutionären Algorithmus zur Detektion von DataMatrix-Codes in Druckerzeugnissen

Master Thesis

von Lukas Petersen

3. Dezember 2015

Technische Universität Hamburg-Harburg

Erklärung

Ich, Lukas Petersen, geboren am 30.11.1989 in Hamburg, versichere hiermit an Eides statt, dass ich diese von mir bei der Technischen Universität Hamburg-Harburg (TUHH) vorgelegte Master Thesis selbstständig verfasst habe. Ich habe ausschließlich die angegebenen Quellen und Hilfsmittel benutzt.

Ort, Datum	Unterschrift

Danksagung

Ich möchte mich zunächst ganz herzlich bei Professor Grigat bedanken, der diese Arbeit als Erstprüfer betreut und unterstützt hat. Weiterer Dank gilt Professor Schlaefer, als Zweitprüfer dieser Arbeit.

Des Weiteren möchte ich mich bei der EyeC GmbH sowie sämtlichen Mitarbeitern, insbesondere bei Dr.-Ing. Marcus Rosenstiel, für die umfangreiche Unterstützung bedanken. Dr.-Ing. Rosenstiel hat mich bei EyeC betreut, motiviert und hat mir auch wertvolle Hinweise gegeben und somit einen großen Teil für den erfolgreichen Abschluss dieser Arbeit beigetragen.

Ein weiteren Dank gilt an dieser Stelle meinen Freunden und meiner Familie, die mich nicht nur bei dieser Arbeit, sondern auch während des gesamten Studiums stets unterstützt und motiviert haben. Ohne euch hätte ich das nicht geschafft.

Inhaltsverzeichnis

Abbildungsverzeichnis **vi**

Tabellenverzeichnis **x**

Symbolverzeichnis **xii**

1 Einleitung **1**
1.1 Problembeschreibung 2
1.2 2D-Codes allgemeinen 2
1.2.1 DataMatrix-Codes 2
1.3 Stand der Technik 3
1.3.1 Abgrenzung dieser Arbeit gegenüber dem Stand der Technik . . 5
1.4 Anwendungsgebiete Neuronaler Netzwerke 5

2 Theoretischer Hintergrund **7**
2.1 Analyse der Datensätze 7
2.1.1 Synthetischer Datensatz 7
2.1.2 Kundendatensatz 10
2.1.2.1 Druck- und Scanprozess 11
2.1.3 Konfidenzanalyse 13
2.1.3.1 Synthetischer Datensatz 14
2.1.3.2 Kundendatensatz 15
2.1.3.3 Extraktion der Lerndaten 15
2.2 Neuronale Netzwerke 15
2.2.1 Neuron 16
2.2.2 Netzwerk Architektur 18
2.2.3 Training 20
2.3 Evolutionärer Algorithmus 22
2.3.1 Ablauf 22
2.3.2 Initial Population 23
2.3.3 Fortpflanzung 24

2.3.4 Rangordnug und Auswahl 24
2.4 Optimierung der Netztopologie mit Hilfe eines evolutionären Algorithmus 25

3 Umsetzung 27
3.1 Neuronales Netzwerk 27
3.1.1 Netzwerkschichten 28
3.1.1.1 Schichttypen 28
3.1.1.1.1 Dense Layer 29
3.1.1.1.2 Convolutional Layer 29
3.1.1.1.3 Maxpooling Layer 30
3.1.1.1.4 Dropout Layer 31
3.1.1.2 Aktivierungsfunktion 32
3.1.1.2.1 Sigmoid Funktion 33
3.1.1.2.2 Softmax 33
3.1.1.2.3 Tangens Hyperbolicus (tanh) 34
3.1.1.2.4 Lineare Funktion 35
3.1.1.2.5 Rectifed Linear Function 36
3.1.1.2.6 Leaky Rectifed Linear Function 37
3.1.1.2.7 Zusammenfassung 38
3.1.1.3 Initialisierung der Gewichte 39
3.1.2 Backpropagation 41
3.1.2.1 Stochatisches Gradienten Abstiegsverfahren und Momentum 41
3.1.2.2 Vanishing Gradient Problem 42
3.1.2.3 Early Stopping 42
3.1.3 Evaluation 43
3.2 Evolutionärer Algorithmus 44
3.2.1 Genom 44
3.2.2 Details 46
3.2.3 Konfiguration 46
3.3 Detektion 47
3.4 Python/C++-Schnittstelle 52

4 Auswertungen und Ergebnisse 55
4.1 Rauschanalyse des Kundendatensatzes 55
4.2 Evolutionärer Algorithmus 57
4.2.1 Festlegung der Hyperparameter 57
4.2.2 Bestimmung der finalen Netztopologie 59
4.2.2.1 Hyperparametergruppe 1 60
4.2.2.2 Hyperparametergruppe 2 61
4.2.2.3 Hyperparametergruppe 3 62

4.2.2.4 Zusammenfassung, Ergebnisse und Bestimmung des finalen Hyperparametersatzes . . . 62
4.2.2.5 Ergebnisse des finalen Hyperparametersatzes . . . 63
4.2.2.6 Weitere Erkenntnisse . . . 64
4.3 Neuronales Netzwerk . . . 65
4.3.1 Lernprozesse und Klassifizierungsleistung . . . 65
4.4 Auswertung Gesamtbilder . . . 69
4.4.1 Synthetischer Datensatz . . . 70
4.4.2 Kundendatensatz . . . 73
4.4.3 Schrittweite . . . 76
4.4.4 Zusammenfassung . . . 78

5 Fazit und Ausblick 79

A Evolutionärer Algorithmus 80
A.1 Allgemein . . . 81
A.2 Hyperparametergruppe 1 . . . 86
A.3 Hyperparametergruppe 2 . . . 89
A.4 Hyperparametergruppe 3 . . . 92
A.5 Finaler Hyperparametersatz . . . 95

B Klassifizierungsergebnisse der finalen Neuronalen Netzwerke 95

C Detektionsalgorithmus 100

Literaturverzeichnis 102

Abbildungsverzeichnis

1.1 Ein typischer DataMatrix-Code. Zu sehen ist das durchgängige und das alternierende Muster an jeweils zwei aneinanderliegenden Rändern, welche die Informationsmodule umschließen. 3

2.1 Beispielbild aus dem synthetischen Datensatz; ein DataMatrix-Code, ein Code128 sowie sechs Texte mit unterschiedlichen Orientierungen sind zu sehen. Des Weiteren ist im Hintergrund ein Polygon und das Bild besitzt Neigung von unter 5°. 9
2.2 Beispielbild aus dem Datensatz aus Kundendaten der EyeC GmbH; zu sehen ist eine Musterverpackung der EyeC GmbH mit einem DataMatrix-Code und einem QR-Code. 10
2.3 Darstellung das Ablaufs vom Erstellen des Bildes bis zur Digitalisierung zur Weiterverarbeitung. 11
2.4 Schematische Darstellung eines Neurons (menschliche Gehirnzelle) sowie Benennung der einzelnen Komponenten. 16
2.5 Künstliches Neuron basierend auf dem Vorbild aus der Natur. 18
2.6 Schematische Darstellung eines dichten 3-4-4-2 künstlichen Neuronalen Netzes; die Eingangsschicht ist in blau dargestellt, die verdeckten Schichten sind grau und die Ausgabeschicht ist grün. Im Rahmen dieser Arbeit werden Eingangs- und Ausgabeschicht bei der Tiefe des Netzwerkes nicht berücksichtigt; dieses Netzwerk hat die Tiefe 2. 19
2.7 Flussdiagramm zur Visualisierung des Ablaufs eines Evolutionären Algorithmus. Dieser ist in zwei Phasen eingeteilt: Initialisierung und Hauptschleife. Während der Ausführung der Hauptschleife werden neue Individuen erstellt und evaluiert sowie ggf. zur Population zu gunsten eines anderen Individuums hinzugefügt. Wird die Hauptschleife beendet definiert das fitteste Individuum das Ergebnis des evolutionären Algorithmus. 23

3.1 Klassendiagramm des Neuronalen Netzwerk Trainers mit den wichtigsten Variablen und sämtlichen Funktionen. 28

3.2 Schematische Darstellung eines künstlichen Neuronalen Netzes. Die Eingangsdaten haben ein Bild zum Inhalt. Gefolgt von der Eingangsschicht ist eine Faltungsschicht mit mehreren Filterkernen zu sehen. Daraufhin findet ein Subsampling statt. Diese Daten werden in zwei weitere versteckte Schichten geführt, gefolgt von der Ausgabeschicht. . 30
3.3 Oben ist ein 3-4-4-2-Neuronales Netzwerk ohne die Anwendung von Dropout Layern zu sehen und unten ist das selbe Netzwerk unter Verwendung von Dropout Layern dargestellt. Einzelne Neuronen samt deren Verbindungen werden hierbei zufällig ausgesetzt. 32
3.4 Sigmoid Funktion sowie die Ableitung der Sigmoid Funktion. 33
3.5 Tangens Hyperbolicus Funktion und die Ableitung der Tangens Hyperbolicus Funktion. 34
3.6 Lineare Funktion und die Ableitung mit $m = 1$. 35
3.7 ReL-Funktion und die Ableitung der ReL-Funktion. 36
3.8 LReL-Funktion sowie die Ableitung; $m = 0{,}01$. 38
3.9 Idealisierte Lernkurven. Die blaue Kurve zeigt die Entwicklung des Trainingsfehlers über die Iteration, die grüne Kurve zeigt die Entwicklung des Validierungsfehlers über die Iterationen. Mit hoher Anzahl an Iterationen können die Kurven auseinander laufen; es kommt zu Overfitting. In rot markiert ist der minimale Validierungsfehler. 43
3.10 Ablaufdiagramm der Detektion von DataMatrix-Codes mittels künstlichen Neuronalen Netzwerks. 48
3.11 Heatmap des in Abbildung 2.2 Beispielbildes. Fenstergröße 32 px × 32 px, Schrittweite 8 px, 8 px. Die Skala der Heatmap ist für alle weiteren Heatmaps identisch. 49
3.12 Zusammenfügen einzelner Heatmaps zu einem Gesamtbild. Fenstergröße 32 px × 32 px, Schrittweite 8 px, 8 px. 50
3.13 Binäre Heatmap. Fenstergröße 32 px × 32 px, Schrittweite 8 px, 8 px, Weichzeichnungsradius 9 px, Schwellwert 0,5. Alle Werte größer oder gleich dem Schwellwert sind rot dargestellt. 51
3.14 Klassenhierachie des C++ Interfaces. 53

4.1 Verteilung der Grauwertabweichung vom Mittelwert von Flächen ohne Schriften und Grafiken der gescannten Druckerzeugnisse. Es wurde für jedes Element aus dem Datensatz ein 32 px × 32 px großes Quadrat ausgewählt, das in einer Fläche liegt, die keine Schriften oder andere Grafiken enthält, woraufhin der mittlere Grauwert dieser Fläche berechnet wurde und die Abweichung von diesem in dieser Abbildung dargestellt wurde. Die dargestellten Einteilungen zwischen a und b sind von der Form $[a;b)$. 56

4.2 Lernkurven der finalen Neuronalen Netzwerke für den synthetischen Datensatz und den Kundendatensatz. Zu sehen ist der Trainingsfehler und der Validierungsfehler sowie die Iteration, die den kleinsten Validierungsfehler liefert. 66
4.3 Bildausschnitte, die die nicht detektierten DataMatrix-Codes beinhalten. 71
4.4 Verteilung der Größenabweichung des Detektionsergebnisses vom tatsächlichen DataMatrix-Code. 72
4.5 Verteilung der Größenabweichung des Detektionsergebnisses vom tatsächlichen DataMatrix-Code. 73
4.6 Bildausschnitte, die die nicht detektierten DataMatrix-Codes beinhalten. 74
4.7 Bildausschnitte, die die falsch detektierten DataMatrix-Codes beinhalten. 75
4.8 Testbild Nr. 59 - Kundendatensatz. 76

A.1 Verteilung der Auswahl der Individuen zur Reproduktion. 82
A.2 Differenz aus Validierungs- und Fitnessfehler während des evolutionären Algorithmus zur Visualisierung des overfittings. 84
A.3 Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke. . . 86
A.4 Entwicklung der Fehler der Validierungs- und Fitnessmenge. 87
A.5 Verteilung der Anzahl an verworfenen Individuen am Stück pro Iteration. 88
A.6 Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke. . . 89
A.7 Entwicklung der Fehler der Validierungs- und Fitnessmenge. 90
A.8 Verteilung der Anzahl an verworfenen Individuen am Stück pro Iteration. 91
A.9 Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke. . . 92
A.10 Entwicklung der Fehler der Validierungs- und Fitnessmenge. 93
A.11 Verteilung der Anzahl an verworfenen Individuen am Stück pro Iteration. 94
A.12 Ergebnisse des evolutionären Algorithmus mit dem finalen Hyperparametersatz. 95

B.1 Falschdetektionen beim synthetischen Datensatz; p gibt den Ausgabewert des Netzwerks an. Alle Bildausschnitte wurden fälschlicherweise als DataMatrix-Codes erkannt. 97
B.2 Falschdetektionen beim synthetischen Datensatz; p gibt den Ausgabewert des Netzwerks an. Alle Bildausschnitte wurden fälschlicherweise nicht als DataMatrix-Codes erkannt. 97
B.3 Falschdetektionen beim Kundendatensatz; p gibt den Ausgabewert des Netzwerks an. Alle Bildausschnitte wurden fälschlicherweise als DataMatrix-Codes erkannt. 98
B.4 Falschdetektionen beim Kundendatensatz; p gibt den Ausgabewert des Netzwerks an. Alle Bildausschnitte wurden fälschlicherweise nicht als DataMatrix-Codes erkannt. 99

C.1 Bildausschnitte, die die gesondert zu behandelnden DataMatrix-Codes beinhalten. 101

Tabellenverzeichnis

2.1 Beispiel einer Population P mit $|P| = 5$, geordnet nach Fitnesswerten f_j der jeweiligen Individuen. 24
2.2 Beispiel einer Population P mit $|P| = 5$, geordnet nach Fitnesswerten f_j der jeweiligen Individuen mit den entsprechenden Auswahlwahrscheinlichkeiten. 25

3.1 Übersicht aller in dieser Arbeit verwendeten Aktivierungsfunktionen. . 38
3.2 Beispiel eines Genoms mit 3 aktiven Schichten und 4 maximal möglichen aktiven Schichten. 45
3.3 Zusammenfassung der untersuchten Hyperparameter des evolutionären Algorithmus. 47

4.1 Untersuchte Hyperparametersätze; es wird jeweils nur ein Parameter geändert, um die Auswirkung dieses Parameters analysieren zu können. . 57
4.2 Konfiguration der Neuronalen Netzwerke für den evolutionären Algorithmus. 58
4.3 Finaler Hyperparametersatz zur Entwicklung der finalen Netztopologie. 63
4.4 Finales Genom mit 4 aktiven Schichten und 5 maximal möglichen aktiven Schichten. 64
4.5 Verteilung der Schichttypen über die Schichten im evolutionären Algorithmus über alle Hyperparametersätze. 65
4.6 Verteilung der Aktivierungsfunktionen über die Schichten im evolutionären Algorithmus über alle Hyperparametersätze. 65
4.7 Klassifizierungsergebnisse des Neuronalen Netzwerkes der Testdaten (Bildausschnitte). Dieses Neuronale Netzwerk wurde auf dem synthetischen Datensatz angelernt und ausgewertet. 66
4.8 Klassifizierungsergebnisse des Neuronalen Netzwerkes der Testdaten (Bildausschnitte). Dieses Neuronale Netzwerk wurde auf dem Kundendatensatz angelernt und ausgewertet. 68
4.9 Parameter des Detektionsalgorithmus zur Detektion von DataMatrix-Codes. 70

A.1 Ergebnisse der Untersuchten Hyperparametersätze. Zu sehen sind die Informationen der letzten Iteration. 85

Symbolverzeichnis

$1-\alpha$	Konfidenzniveau eines Datensatzes.
$\bar{X}$	Wahrscheinlichkeit für das Auftreten eines DataMatrix-Codes in den Testsets der Datensätze.
η	Lernrate eines Gradientenabstiegsverfahrens; $\eta > 0$.
γ	Gewichtung des Nesterov Momentum Terms; $\gamma \in [0;1]$.
λ	Wellenlänge in nm.
$\|P\|$	Gibt die Anzahl der Individuen der Population P an.
$\mathbb{D}$	Definitionsbereich einer Funktion.
$\mathbb{R}$	Menge der reellen Zahlen.
$\mathbb{W}$	Wertebereich einer Funktion.
$\mathcal{U}(b_\mathrm{u}, b_\mathrm{o})$	Gleichverteilung mit der unteren Grenze b_u und der oberen Grenze b_o.
K_p	Kofindenzintervall für Erfolgswahrscheinlichkeit p.
∇	Nabla-Operator zur Kennzeichnung eines Gradienten.
$\mathrm{red}(S), \mathrm{green}(S), \mathrm{blue}(S)$	Funktionen, die den jeweiligen Rot-, Grün- oder Blauwert eines Bildes S zurückgeben.
θ	Typischerweise nichtlineare Aktivierungsfunktion eines mathematischen Neurons.
a_x, a_y	x- und y-Koordinaten des Ursprungs eines Filterkerns $K^{u\times v}(m_x, m_y)$.
b	Das Bias b wird als Schwellwert für die nichtlineare Funktion θ verwendet und direkt mit den anderen Gewichten eines mathematischen Neurons mit angelernt.

E	Fehlerfunktion eines Neuronalen Netzwerkes.
e	Eulersche Zahl.
$E[\cdot]$	Erwartungswert.
f_j	Fitness des Individuums j einer Population P.
h_s	Höhe eines kontinuierlichen Bildes in mm.
i_{es}	Sobald der Zähler vom Early Stopping diese Anzahl an Iterationen erreicht wird das Training abgebrochen.
$K(m_x, m_y)$	Filterkern für eine Faltungsoperation.
L	Negative Log-Likelihood-Funktion.
M	Größe des Trainingsdatensatzes.
N	Anzahl an Eingangswerten eines mathematischen Neurons.
N_x, N_y	Maximale Anzahl an Pixeln eines digitalen Bildes S in x-, respektive y-Richtung; es gilt $n_x \in [0; N_x - 1]$ und $n_y \in [0; N_y - 1]$; der Ursprung des Bildes ist in der oberen linken Ecke.
n_x, n_y	Diskrete Koordinaten eines Bildes in Pixeln, wobei N_x und N_y die entsprechenden Anzahlen von Zeilen und Spalten sind.
N_{Neuronen}	Anzahl an Neuronen eines Neuronalen Netzwerkes.
o_*	Ausgabewert einer mathematischen Funktion; $*$ kennzeichnet die entsprechende Funktion passend zu der Ausgabe.
p	Detektionsergebnis des DataMatrix-Code Detektors.
$p(\cdot)$	Wahrscheinlichkeit von $\cdot$.
P_i	Populations zur Iteration i. Mit $i = 0$ ist die initiale Population gemeint.
p_{drop}	Wahrscheinlichkeit für das Auslassen eines Neurons während der Lernphase; $p_{\text{drop}} \in [0; 1]$.
p_{m}	Mutationswahrscheinlichkeit; Wahrscheinlichkeit, mit welcher eine Eigenschaft eines Individuums nicht von einen Elternteil übernommen wird, sondern zufällig vergeben wird.

$p_{\text{rank}}(j)$	Auswahlwahrscheinlichkeit eines Individuums j bei der Rangordnung basierender Auswahl.
r_j	Fitnessrang des Individuums j einer Population P.
$r_{\text{error},j}$	Rang nach dem Fehler E eines Individuums j.
$r_{\text{size},j}$	Rang nach der Anzahl an Parametern eines Individuums j.
s	Größe der Stichprobe.
$S(n_x, n_y)$	Diskrete, quantisierte Amplitude an der Stelle n_x, n_y eines digitalen Bildes S.
$s(x, y, \lambda)$	Kontinuierliche Amplitude eines kontinuierlichen Bildes s an den Koordinaten x und y und der Wellenlänge λ.
s_p	Skalierungsfaktor $s_p \in [0; 1]$ des evolutionären Algorithmus zur Bestimmung des Fitnesswertes f_j eines Individuums j.
s_{batch}	Größe eines Batches.
t	Iterationsschritt eines Gradientenabstiegsverfahren.
t_m	Erwarteter Ergebnisvektor des korrespondierenden Eingangsvektor $x_{\text{in},m}$
W	Gewichtungsmatrix; die Spalten von W entsprechen den einzelnen Gewichtungsvektoren w einer Netzwerkschicht.
w	Zusammenfassung der N Gewichte w_i eines mathematischen Neurons.
W^*	Sämtliche Gewichte eines Neuronalen Netzwerkes
w_i	Gewichtung des Eingangswertes x_i eines mathematischen Neurons.
w_s	Breite eines kontinuierlichen Bildes in mm.
x, y	Kontinuierliche horizontale bzw. vertikale Koordinaten eines Kontinuierlichen Bildes s in mm.
x_i	Eingangswert i eines mathematischen Neurons.
$x_{\text{in},m}$	Eingangsvektor mit Lerndaten für ein Neuronales Netzwerk; die dazugehörigen Ausgabedaten sind in t_m zu finden.
x_{input}	Zusammenfassung der N Eingangswerte x_i eines mathematischen Neurons.

$z^*_{1-\frac{\alpha}{2}}$	$(1-\frac{\alpha}{2})$-Quantil der Standardnormalverteilung.
w'	Erweiterter Gewichtungsvektor w das Bias b zur Biasmodellierung.
x'_{input}	Erweiterter Eingangsvektor x_{input} um einen Wert 1 zur Biasmodellierung.

Kapitel 1

Einleitung

Das Finden von Mustern und Bildmerkmalen ist eine Problematik des maschinellen Sehens mit diversen Lösungsansätzen. In der Industrie wird maschinelles Sehen unter anderem zur Schrifterkennung (Handschriftenerkennung, OCR[1]), Erkennung von Fehlern auf Druckerzeugnissen, Gesten- und Gesichtserkennung sowie in der Medizintechnik zur Unterstützung von Diagnosen verwendet. Diese Arbeit beschäftigt sich mit der Thematik der Detektion von DataMatrix-Codes in Druckerzeugnissen. DataMatrix-Codes sind zweidimensionale Codes, welche in der Industrie und vor allem im pharmazeutischen Umfeld zum Einsatz kommen.

Die Arbeit ist wie folgt aufgebaut: In Kapitel 1 ist zu Beginn die Problembeschreibung zu finden. Außerdem gibt es eine kurze Einführung in 2D-Codes sowie eine Übersicht zu gängigen Verfahren, um DataMatrix-Codes in Bildern zu detektieren. Daraufhin ist in Abschnitt 1.3.1 beschrieben, inwiefern sich diese Arbeit von den in Abschnitt 1.3 vorgestellten Verfahren unterscheidet.

Kapitel 2 vermittelt die für die weiteren Kapitel notwendige Theorie. Es werden zum Anfang die beiden vorhandenen Datensätze vorgestellt sowie eine Konfidenzanalyse der Datensätze vorgenommen. Daraufhin wird der theoretische Hintergrund von Neuronalen Netzwerken aufgearbeitet. Abgeschlossen wird das Kapitel mit Informationen zu evolutionären Algorithmen sowie dem Zusammenspiel eines evolutionären Algorithmus und den Neuronalen Netzwerken.

Kapitel 3 handelt von der Umsetzung eines Neuronalen Netzwerkes sowie des evolutionären Algorithmus. Des Weiteren beinhaltet dieses Kapitel auch die finale Umsetzung des Neuronalen Netzwerkes zur Detektion der DataMatrix-Codes.

Im Kapitel 4 sind die quantitativen Auswertungen dieser Arbeit zu finden. Zum einen wird der evolutionäre Algorithmus evaluiert, um festzustellen, welche Hyperparameter welchen Einfluss auf die Entwicklung der Population haben, zum anderen werden die Netzwerke mit der finalen Netztopologie, also dem Ergebnis des evolutionären Al-

[1] **o**ptical **c**haracter **r**ecognition - Optische Zeichenerkennung

gorithmus, ausgewertet. Zum Schluss werden noch die Gesamtbilder der entsprechenden Testmenge ausgewertet.

Im letzten Kapitel, Kapitel 5, findet sich abschließend eine Zusammenfassung und das Fazit dieser Arbeit sowie einen Ausblick auf mögliches weiteres Vorgehen.

1.1 Problembeschreibung

Ziel dieser Arbeit ist es DataMatrix-Codes in Scans von Druckerzeugnissen zu finden. Es sollen die Eigenschaften der Druckerzeugnisse berücksichtigt werden, um eine sinnvolle Problemlösung finden zu können.

Gegeben sind zwei Datensätze, die analysiert werden sollen. Die Datensätze sind in Abschnitt 2.1 beschrieben.

1.2 2D-Codes allgemeinen

GS1[2] definiert zwei Arten von 2D-Codes: QR-Codes[3] und DataMatrix-Codes.

Vorteile von 2D-Codes gegenüber klassischen Barcodes oder auch Strichcodes sind, dass sie typischerweise mehr Informationen beinhalten können, als 1D-Codes. Der GS1-128 Code beispielsweise ist ein eindimensionaler Barcode, welcher maximal 48 alphanumerische Zeichen enthalten kann; ein GS1-DataMatrix-Code mit 132×132 Modulen kann bis zu 2335 alphanumerische Daten enthalten.

Des Weiteren besitzen 2D-Codes typischerweise eine Fehlererkennung und -korrektur. Im Fall des GS1-128 Codes ist nur eine Checksumme vorhanden, welche zur Fehlererkennung verwendet wird, nicht aber zur Fehlerkorrektur. Die GS1 QR-Codes besitzen, wie auch die DataMatrix-Codes, sowohl eine Fehlererkennung als auch eine Fehlerkorrektur.

1.2.1 DataMatrix-Codes

DataMatrix-Codes sind kompakte 2D-Codes, die häufig in der Industrie eingesetzt werden. Die ISO[4] definiert in [1] DataMatrix-Codes als normalerweise quadratische Module, die mit einem Suchmuster umgeben sind. Dieses Suchmuster besteht zum einen aus zwei aneinanderliegenden Seiten, die durchgängig eingefärbt sind und den anderen beiden Seiten, die ein alternierendes Muster besitzen (siehe Abbildung 1.1).

[2] **G**lobal **S**tandards One (**1**)
[3] **Q**uick **R**esponse Codes
[4] International Organization for Standardization

Abbildung 1.1: Ein typischer DataMatrix-Code. Zu sehen ist das durchgängige und das alternierende Muster an jeweils zwei aneinanderliegenden Rändern, welche die Informationsmodule umschließen.

Ein weiterer Standard bezüglich DataMatrix-Codes wird von GS1 definiert, welcher auf [1] basiert. GS1 gibt dabei in [2] eine Übersicht über den Aufbau, die verschiedenen Formen und zur Interpretation sowie weitere Details von GS1 DataMatrix-Codes.

Informationen bezüglich der Kodierung der Daten in den DataMatrix-Codes sowie Details bezüglich der Fehlerbehandlung sind sowohl in [1] als auch in [2] zu finden.

1.3 Stand der Technik

In [1] wird ein Referenzalgorithmus angegeben, um DataMatrix-Codes in einem Bild zu finden. Der Referenzalgorithmus basiert hierbei auf zeilen- und spaltenweise Durchsuchen des gegebenen Bildes, um die äußeren Kanten des DataMatrix-Codes zu finden.

Eine verbreitete Open Source Implementierung zur Detektion und Dekodierung von verschiedenen 1D- und 2D-Codes[5] ist ZXing [3]. Laut Google Playstore [4] wurde das Programm zum Detektieren und Dekodieren von Barcodes des ZXing Teams zwischen 100 und 500 Millionen mal installiert. Aus dem frei zugänglichen Open Source Quelltext wird folgende Lokalisation von Codes ersichtlich[6]: Ausgehend von der Mitte wird ein Suchfeld definiert. In diesem Suchfeld werden auf Ebene von Pixeln Zeile für Zeile bzw. Spalte für Spalte nach dem entsprechenden Muster gesucht, welches dekodiert werden soll (z.B. das Suchmuster von DataMatrix-Codes). Wenn es nicht gefunden wird, wird das Suchfeld vergrößert und erneut geprüft, bis ein entsprechender 1D- oder 2D-Code gefunden wurde oder das Suchfeld das ganze Bild einnimmt und somit der Algorithmus zum Schluss kommt, dass kein Code vorliegt. Diese und weitere Arten von Verfahren werden auch in [5] beschrieben.

Ein weiterer Ansatz aus [5] ist es nach der hohen Konzentration von Kantenstrukturen in verschiedenen Orientierungen zu suchen. Dieser Ansatz wird dabei sowohl für

[5] Seit Version 1.0 von 2008 auch DataMatrix-Codes

[6] Stand ZXing Version 3.2.0 vom 15.02.2015.

1D- als auch für 2D-Codes verwendet. Das Bild wird in Bereiche unterteilt; jeder Bereich bekommt einen Wert zugewiesen, der die Wahrscheinlichkeit des Vorhandenseins eines Barcodes angibt. Je höher die Konzentration an Kantenstrukturen in dem jeweiligen Bereich ist, desto höher ist die zugewiesene Wahrscheinlichkeit. Um Invarianz bezüglich der Skalierung zu erlangen wird das Verfahren auf unterschiedlich große Bereiche ausgeführt. Final werden die Bereiche zusammengefasst und eine Region zurückgegeben, in welcher ein Barcode liegen sollte (ROI[7]). Weiter gehen die Autoren in [5] noch auf Ansätze mit morphologischen Operatoren ein, wobei hier auch erwähnt wird, dass diese Probleme bezüglich der Detektion haben und öfter Textbereiche als Teile von 1D- oder 2D-Codes klassifizieren.

In [6] wird auch mit morphologischen Operatoren gearbeitet. Hierfür wird das zu untersuchende Bild vorverarbeitet (verringern des Rauschens mittels eines Gauß-Filter), die Kanten verstärkt und ein Binärbild berechnet. Dieses Binärbild wird dann mit morphologischen Operatoren (Dilatation und Erosion) bearbeitet, um so ROIs festzulegen, in welchen sich ein Barcode befinden sollte.

Eine etwas andere Herangehensweise ist in [7] zu finden. Hierbei wird auf Pixelebene nach den zwei durchgehenden und den zwei gestrichelten Linien gesucht. Hierfür wird zur Vorverarbeitung ein Canny Filter verwendet und dann sämtliche Linien extrahiert. Anschließend werden die Linien verworfen, welche sich nicht in einer L-Form oder in einem entsprechenden Winkel zu einander befinden. Nun wird anhand der übrig gebliebenen Linien nach den beiden gestrichelten Linien gesucht und somit die ROI des DataMatrix-Codes zurückgegeben.

[8] stellt einen Ansatz zur Lokalisierung von QR-Codes vor, welcher auf tiefen neuronalen Netzwerken basiert. Das entsprechende Bild wird dabei in Blöcke unterteilt, welche durch ein neuronales Netzwerk klassifiziert werden. Jeder Block bekommt einen Wert zugewiesen, welcher repräsentiert, wie hoch die Wahrscheinlichkeit ist, ob der Block zu einem QR-Code gehört oder nicht. Am Ende werden diese Blöcke zusammengefasst und so eine ROI definiert, in welcher sich der QR-Code befinden sollte. Des Weiteren wird in [8] noch mit verschiedenen Möglichkeiten der Vorverarbeitung experimentiert. Zum einen werden die Pixel ohne Vorverarbeitung beim Klassifizierer angelernt sowie ausgewertet und zum anderen mittels DCT[8] und DFT[9] vorverarbeitet. Außerdem wird mittels Sobel Filtern das Bild bearbeitet und diese Daten zum Lernen und Detektieren verwendet, wobei in diesem Fall die besten Ergebnisse erzielt wurden.

Zusammenfassend ist festzustellen, dass häufig auf Binärbildern und mit Finden der eindeutigen Suchmustern gearbeitet wird oder mit Klassifizierern, welche meistens mit Vorverarbeitung und Extraktion von besonderen Merkmalen, wie Ecken und Kanten, ein besseres Ergebnis erzielen, als mit den Rohdaten. Neuronale Netze werden im allgemei-

[7] **R**egion **o**f **I**nterest - Region von Interesse

[8] **D**iscrete **C**osine **T**ransform - Diskrete Cosinustrainsformation

[9] **D**iscrete **F**ourier **T**ransform - Diskrete Fouriertransformation

nen zur Objekterkennung verwendet; der Bereich der Lokalisation und Klassifizierung von 2D-Codes, insbesondere von DataMatrix-Codes, mittels Neuronalen Netzwerken ist zum jetzigen Zeitpunkt noch nicht weit verbreitet.

1.3.1 Abgrenzung dieser Arbeit gegenüber dem Stand der Technik

Die in [3,5,7,8] vorgestellten Verfahren beziehen sich stets auf typische Fotoaufnahmen, welche nicht die Einschränkungen (siehe Abschnitt 2.1.2.1) haben, wie sie in dieser Arbeit auf Grund der Umfeldes der Druckerzeugnisse angenommen werden können. Die Hauptunterschiede sind vor allem die verschiedenen Orientierungen und Verzerrungen der 2D-Codes in typischen Fotoaufnahmen, respektive den in [8] vorgestellten synthetischen Bildern. Des Weiteren wird teilweise von starkem Rauschen ausgegangen sowie Blurring[10] und niedrigen Auflösungen, was bei Scanns von Druckerzeugnissen nicht zu erwarten ist.

Auch unterscheidet sich diese Arbeit von den oben genannten durch das Vorhandensein von deutlich mehr Text und somit deutlich mehr Kanten und Ecken, wodurch einige Ansätze, wie zum Beispiel [5] oder [6], nicht die selben guten Ergebnisse erzielen könnten. Auch Rauschen wird in dieser Arbeit eine untergeordnete Rolle übernehmen, wodurch die zum Beispiel in [9] vorgeschlagenen Gegenmaßnahmen zur Rauschunterdrückung in diesen Umfang nicht von Nöten sein werden.

Somit grenzt sich diese Arbeit thematisch gegenüber den Arbeiten [3, 5, 7, 8] durch die Betrachtung von Druckerzeugnissen deutlich ab.

Neu in dieser Arbeit und damit im Unterschied zum Stand der Technik ist die Verwendung eines evolutionären Algorithmus zur Bestimmung der Netztopologie zur Detektion von 2D-Maschinencodes, welcher in den beschriebenen Verfahren aus Abschnitt 1.3, insbesondere in [8], nicht zum Einsatz kommt. Die Netztopologie ist ein entscheidendes Kriterium für die Leistung eines Neuronalen Netzwerkes; wird eine Netztopologie einfach angenommen und gegebenenfalls nur im kleinen Bereich modifiziert ist die Wahrscheinlichkeit hoch, dass die Netztopologie nicht optimal ist.

1.4 Anwendungsgebiete Neuronaler Netzwerke

Wie in Abschnitt 1.3 schon erwähnt, werden Neuronale Netzwerke unter anderem zu Objektdetektion verwendet. In [10] wird auch auf andere Anwendungsgebiete eingegangen. Dabei wird auf den Einsatz von Neuronalen Netzwerken in der Spracherkennung und zur Lokalisation von Geräuschen hingewiesen. Auch können Neuronale Netzwerke im Allgemeinen zur Approximation von Funktionen eingesetzt werden. Weiter werden

[10] Verwischung

Neuronale Netzwerke auch in der Genforschung und zur 3D-Rekonstruktion aus Videodaten verwendet.

Kapitel 2

Theoretischer Hintergrund

Dieses Kapitel bildet die Basis der Arbeit. In Abschnitt 2.1 werden die Datensätze formal behandelt und analysiert. Daraufhin wird in Abschnitt 2.2 auf den theoretischen Hintergrund der Neuronalen Netzwerke eingegangen. Weiter wird in Abschnitt 2.3 evolutionäre Algorithmen beschrieben, wie sie in dieser Arbeit zum Einsatz kommen. Das Kapitel endet mit Abschnitt 2.4, in dem beschrieben wird, wie der evolutionäre Algorithmus die Netztopologie Neuronaler Netzwerke optimieren kann.

2.1 Analyse der Datensätze

Diese Arbeit basiert auf zwei verschiedenen Datensätzen. Der erste Datensatz beinhaltet synthetisch erstellte Bilder, welche in 2.1.1 näher beschrieben werden.

Der zweite Datensatz setzt sich aus Kundendaten der Firma EyeC GmbH zusammen und besteht vorwiegend aus Scans von Pharmazieerzeugnissen.

2.1.1 Synthetischer Datensatz

Der synthetische Datensatz ist wie folgt charakterisiert:

- 10000 Bilder (5000 Bilder mit vorhandenen DataMatrix-Code, 5000 Bilder ohne DataMatrix-Code)
- 8 Bit Graustufenbild
- Zufällige Hintergrundfarbe
- 1000 px[1] $\times$ 1000 px Größe

[1] **Pix**el

- Rotation des Gesamtbildes sowie einzelner Elemente in 90° Schritten begleitet von kleinen Rotationen $\leq \pm 5°$
- Zwischen 3 und 10 Textblöcken
  - zufällige Textgröße zwischen 10 px und 25 px
  - zufällige Schriftart
  - zufällig zwischen 20 und 50 Sätzen, welche nicht komplett im Bild vorhanden sein müssen
  - zufällige Orientierung in 90° Schritten
- Maximal einen 1D-Barcode vom Typ Code128 ist mit 50 % Wahrscheinlichkeit vorhanden, welcher eine zufällige alphanumerische Zeichenketten kodiert
- Maximal einen DataMatrix-Code ist in 50 % der Daten vorhanden, welcher zufällige alphanumerische Zeichenketten kodiert
- Zwischen 0 und 3 Polygone im Hintergrund des Bildes mit 3 bis 12 zufälligen Punkten und zufälligen Farben
- Zusätzlich leichtes Rauschen (AWGN[2])[3]

In Abbildung 2.1 ist ein solches Bild zu sehen.

[2] **A**dditive **W**hite **G**aussian **N**oise - Additives weißes gaußsches Rauschen

[3] Das Bild wurde normiert ($[0;1]$) und mit AWGN überlagert. Das AWGN besitzt eine zufälligen Varianz aus dem Bereich $[0;10^{-4}]$. Danach wurde das Bild zurück in ein 8 Bit Graustufenbild gewandelt.

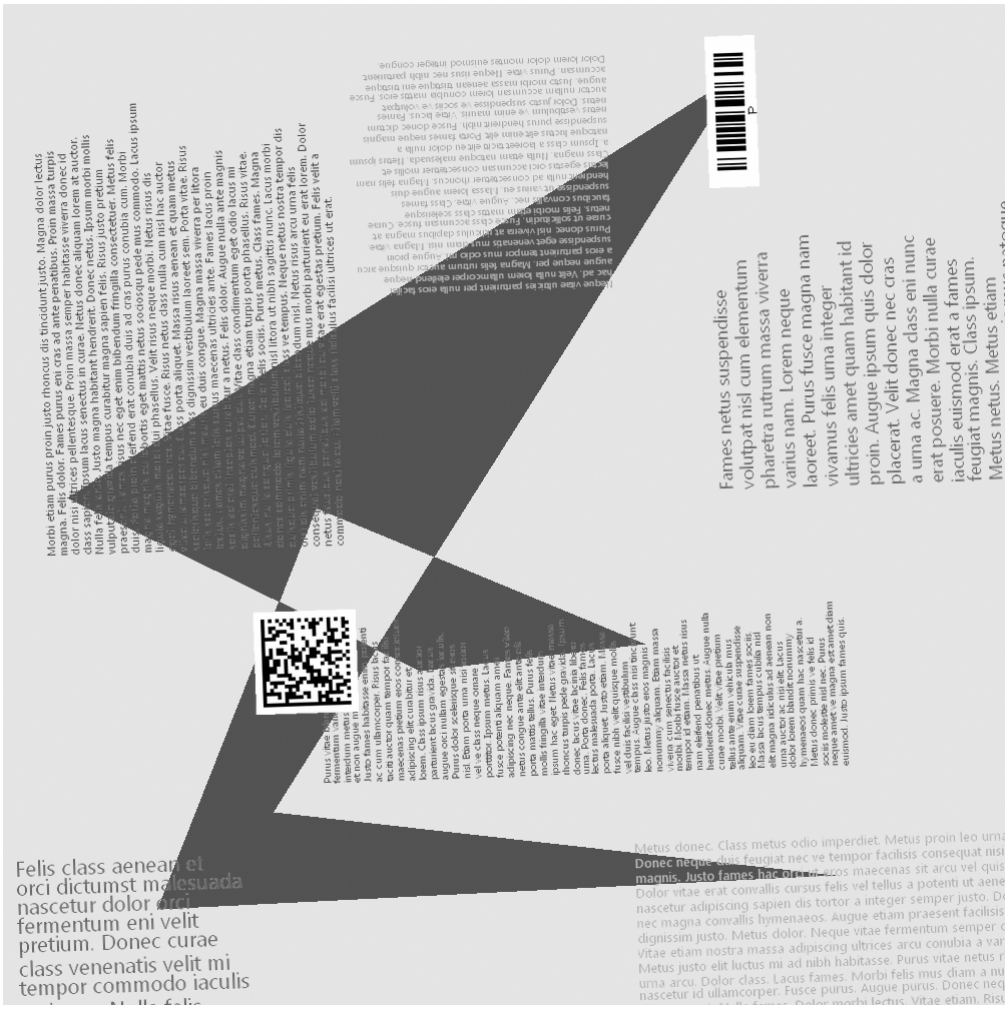


Abbildung 2.1: Beispielbild aus dem synthetischen Datensatz; ein DataMatrix-Code, ein Code128 sowie sechs Texte mit unterschiedlichen Orientierungen sind zu sehen. Des Weiteren ist im Hintergrund ein Polygon und das Bild besitzt Neigung von unter 5°.

Die DataMatrix-Codes haben zufällige Größen; es wurde beim erstellen des Datensatzes keine Rücksicht auf die Lesbarkeit beziehungsweise Dekodierbarkeit genommen. Es kann also vorkommen, dass die Modulgröße der DataMatrix-Codes zu gering ist, um sie dekodieren zu können. Die Eigenschaften des synthetischen Datensatzes entsprechen den in Abschnitt 2.1.2.1 Eigenschaften von Druckerzeugnissen.

2.1.2 Kundendatensatz

Der Datensatz aus Kundendaten besteht aus eingescannten Faltschachteln und Etiketten von Erzeugnissen vorwiegend aus dem pharmazeutischen Bereich. Vorhanden sind 588 Bilder von Faltschachteln, wobei 326 Bilder mindestens einen DataMatrix-Code enthalten, also 262 Bilder von Faltschachteln ohne DataMatrix-Codes vorhanden sind. Die 326 Bilder enthalten insgesamt 773 DataMatrix-Codes.

Abbildung 2.2: Beispielbild aus dem Datensatz aus Kundendaten der EyeC GmbH; zu sehen ist eine Musterverpackung der EyeC GmbH mit einem DataMatrix-Code und einem QR-Code.

In Abbildung 2.2 ist ein Scan eines Musters der EyeC GmbH zu sehen. Die Bil-

der $S(n_x, n_y)$ liegen mit Farbinformationen vor, werden aber in 8 Bit Graustufenbilder $S'(n_x, n_y)$ mittels ITU-R 601-2 luma Transformation (siehe [11])

$$S'(n_x, n_y) = 0{,}299 \cdot \text{red}(S(n_x, n_y)) + 0{,}587 \cdot \text{green}(S(n_x, n_y)) + 0{,}114 \cdot \text{blue}(S(n_x, n_y)) \tag{2.1}$$

transformiert.

2.1.2.1 Druck- und Scanprozess

Die aufgenommenen, nicht synthetischen Bilder haben alle eingescannte Aufnahmen (Flachbettscanner) von Druckerzeugnissen zum Inhalt. Der Ablauf des Druck- und Scanprozesses ist in Abbildung 2.3 dargestellt.

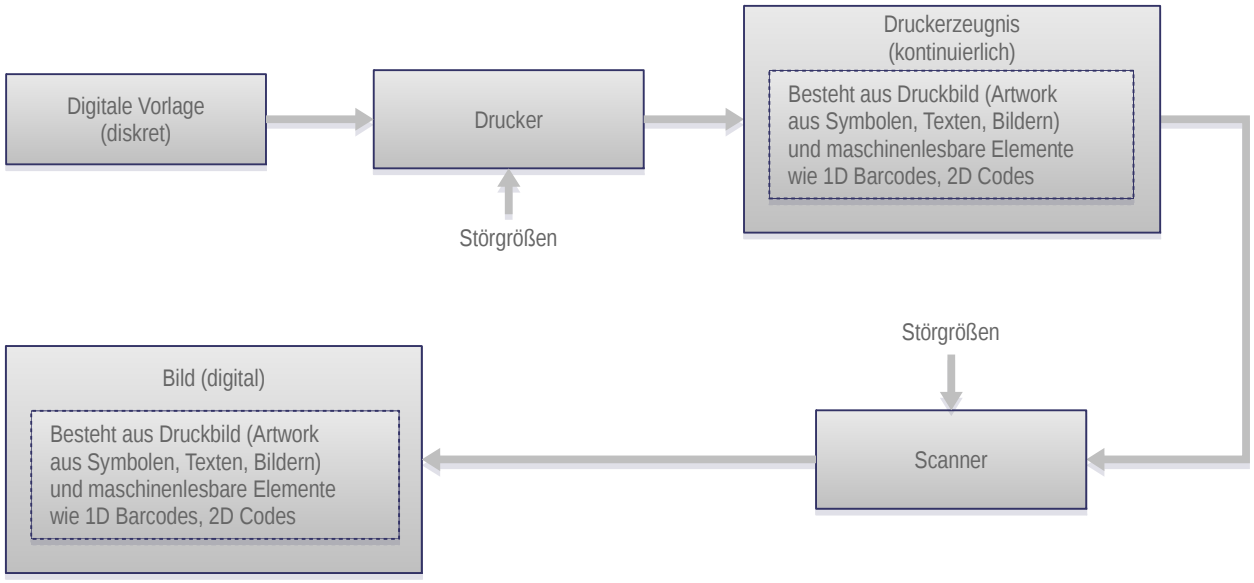


Abbildung 2.3: Darstellung das Ablaufs vom Erstellen des Bildes bis zur Digitalisierung zur Weiterverarbeitung.

Es existiert eine digitale Vorlage $S(n_x, n_y)$, welche ausgedruckt und damit in ein kontinuierliches Bild $s(x, y, \lambda)$ umgewandelt wird. Dieses Druckerzeugnis s wird nun wieder mit Hilfe eines Flachbettscanners digitalisiert und es entsteht ein weiteres digitales Bild $S'(n_x, n_y)$, wobei S' auf Grund des Druck- und Scannprozesses typischerweise nicht S entspricht.

Ein Vorschlag zur Modellierung des Druck- und Scanprozesses ist in [12] beschrieben: Gründe für Störungen im Druck- und Scanprozess auf Pixelebene werden durch Beleuchtung, Kontraständerungen, Gammakorrektur, Chrominanz-Variationen sowie

Blurring von aneinander liegenden Pixeln verursacht. Des Weiteren können sich die geometrischen Eigenschaften für das wieder eingescannte Druckerzeugnis verändern. Hierbei handelt es sich um Rotations-, Skalierungs-, Translations- und Cropping[4]-Effekte.

In [12] wird weiter beschrieben, dass der Druckprozess eine Tiefpasscharakteristik für das menschliche Auge darstellt. Die Pixel von S werden hierbei als einzelne Punkte auf das Druckmedium aufgetragen, wobei die Farbinformationen typischerweise subtraktive gemischt werden (CMYK[5]). Allerdings sind auch Vollton- oder andere Prozessfarben möglich. Durch die Tiefpasscharakteristik des menschlichen Auges sind die einzelnen Punkte ab einem gewissen Abstand, respektive ab einer gewissen Druckauflösung nicht mehr als Punkte, sondern als Flächen, zu erkennen.

Eine weitere Eigenschaft des Druckers (und auch des Scanners) ist seine Abtastrate, die typischerweise in DPI[6] angegeben wird. Auch diese hat einen Einfluss auf das Druckergebnis. Insgesamt wird die Umwandlung des digitalen Bildes S in das kontinuierliche Druckerzeugnis s als eine mit Dithering[7] und adaptiven Rauschen behaftete Abbildung bezeichnet. Weiter wird in [2] auf die Druckqualität von DataMatrix-Codes eingegangen. Es kann zu Problemen kommen, wenn beim Druck zu viel oder zu wenig Farbe auf das Druckmedium aufgetragen wird. Die zu druckenden Strukturen werden dabei entsprechend ausgedünnt oder die Strukturen werden dicker, als in der Vorlage vorgesehen.

Der Scanprozess wird in [12] so beschrieben, dass das Ergebnis S' maßgeblich von der Systemantwort des Detektors sowie von der Veränderung der geometrischen Eigenschaften abhängt. Auf Grund der translatorischen Bewegung des Scannermotors, welcher den Zeilendetektor bewegt, kann es während des Scanprozesses zu einen Bewegungsjitter kommen. Dieser Effekt kann als Hochpass modelliert werden, der in Bewegungsrichtung des Zeilendetektors stärker ausfällt als orthogonal zu dieser. Des Weiteren wird von AWGN ausgegangen. Auch findet während des Prozesses eine Gammakorrektur statt. Es wird auch noch auf andere Effekte, wie thermisches Rauschen und durch Dunkelstrom verursachtes Rauschen eingegangen.

Die geometrischen Störungen während des Druck- und Scanprozesses bezüglich Rotation, Translation, Skalierung und Cropping, wie sie weiter oben schon genannt wurden, können für den Scanvorgang noch weiter spezifiziert und in Bezug auf diese Arbeit eingeschränkt werden. Rotationen kommen beim Scannen typischerweise in Schritten von 90° vor, begleitet von kleinen Orientierungsänderungen[8], auf Grund der Genauigkeit bei der Handhabung der Scanvorrichtung. Translationen sind für diese Arbeit nicht relevant, da stets das komplette Bild abgesucht wird und davon auszugehen ist, dass das

[4] Zuschneidung

[5] CMYK Farbmodell: **c**yan, **m**agenta, **y**ellow und **k**ey (schwarz) mehr in [13].

[6] **D**ots **P**er **I**nch - Punkte pro Zoll

[7] Fehlerdiffusion

[8] In [12] wird von Winkeln $< \pm 5^\circ$ ausgegangen.

Druckerzeugnis komplett eingescannt ist und somit keine Informationen fehlen. Gleiches gilt für das Cropping. Skalierung hingegen ist äußerst relevant für diese Arbeit, da die zu findenden Merkmale in unterschiedlichen Größen auftreten können und die ausgewählten Methoden unterschiedliche Skalierungen behandeln müssen.

Eine andere Eigenschaft mit Bedeutung für diese Arbeit, die der Druck und Scanprozess laut [12] mit sich bringen kann, ist zero padding[9]. Druckerzeugnisse müssen nicht immer eine rechteckige Form haben (Beispiel Abbildung 4.4.3), wodurch beim Scanprozess zusätzliche Bereiche hinzugenommen werden müssen, welche nicht vom Druckerzeugnis abgedeckt sind, um rechteckiges Bild zu generieren.

Zusammenfassend ist zu sagen, dass das Ausgangsbild S sich von dem eingescannten Bild S' in dieser Arbeit geometrisch in der Orientierung und in der Skalierung unterscheidet. Mit weiteren affinen Transformationen ist nicht zu rechnen. Auch Rauschen und gegebenenfalls Unterschiede in der Farbdarstellung, unter anderem durch die entsprechende Belichtung im Scanner, kommen hinzu, wobei das Rauschen in den in 2.1.2 vorgestellten Kundendatensatz vernachlässigbar gering ist (vgl. Abschnitt 4.1).

2.1.3 Konfidenzanalyse

Um die Detektionsleistung des finalen Neuronalen Netzes bewerten zu können müssen die Datensätze einer Konfidenzanalyse unterzogen werden. Die Datensätzen repräsentieren nur eine endliche Anzahl an Stichproben der allgemeinen Gesamtheit. Die Konfidenzanalyse wird verwendet, um eine Aussage zur allgemeine Detektionsleistung treffen zu können. Zur Bestimmung des geforderten Stichprobenumfanges müssen zunächst die Forderungen definiert werden. Das Ziel dieser Arbeit soll dabei mit einem Konfidenzniveau von 95 % bei der Klassifizierung für weitere Stichproben liegen, wobei ein Schätzfehler von 5 % akzeptabel ist. Es existieren nur zwei Klassen (DataMatrix-Code/kein DataMatrix-Code); diese werden im Stichprobenumfang gleichverteilt sein.

In [14] findet man hierzu eine Herleitung für den Ausdruck

$$\mathrm{K}_p = \left[\bar{X} - z^*_{1-\frac{\alpha}{2}} \cdot \sqrt{\frac{\bar{X}\cdot(1-\bar{X})}{s}}; \bar{X} + z^*_{1-\frac{\alpha}{2}} \cdot \sqrt{\frac{\bar{X}\cdot(1-\bar{X})}{s}}\right] \tag{2.2}$$

für ein zweiseitiges Konfidenzintervall K_p für den Anteil p bei Binomialverteilung und großem Stichprobenumfang, wobei p in diesem Zusammenhang dem Detektionsergebnis des DataMatrix-Code Detektors entspricht. Es wird die Annahme gemacht, dass sich die Abweichung von dem gesetzten Ziel (Konfidenzniveau von 95 % und Schätzfehler von 5 %) binomial verteilt. $\bar{X}$ definiert dabei die Wahrscheinlichkeit des zu untersuchenden Ereignisses, wobei in dieser Arbeit wie schon erwähnt von einer Gleichverteilung der Klassen ausgegangen wird, wodurch

$$\bar{X} = (1 - \bar{X}) = 0{,}5 \tag{2.3}$$

[9] Zusätzlicher Hintergrund, welcher beim scannen entsteht und in dem finalen Bild S' vorhanden ist.

gilt. Die Variable s bezeichnet die Größe der Stichprobe und ist in diesem Fall der zu bestimmende Wert. $1-\alpha$ entspricht dem Konfidenzniveau und $z^*_{1-\frac{\alpha}{2}}$ ist das $(1-\frac{\alpha}{2})$-Quantil der Standardnormalverteilung. Aus

$$1-\alpha = 0{,}95 \tag{2.4}$$

folgt

$$\frac{\alpha}{2} = 0{,}025 \tag{2.5}$$

und aus der Tabelle aus [14] für die Standardnormalverteilung folgt

$$z^*_{0{,}975} = 1{,}96 \tag{2.6}$$

wodurch sich die Gleichung für die untere Intervallgrenze

$$\bar{X} - z^*_{0{,}975} \cdot \sqrt{\frac{\bar{X}\cdot(1-\bar{X})}{s}} = \bar{X} - 0{,}05 \tag{2.7}$$

beziehungsweise für die obere Intervallgrenze

$$\bar{X} + z^*_{0{,}975} \cdot \sqrt{\frac{\bar{X}\cdot(1-\bar{X})}{s}} = \bar{X} + 0{,}05 \tag{2.8}$$

nach s auflösen lässt und man auf das Ergebnis

$$s \approx 385 \tag{2.9}$$

kommt. Also muss der Umfang des Testsets in dieser Arbeit mindestens 385 Elemente betragen, um die Aussage treffen zu können, dass für weitere gleichverteilte Testsätze ein Detektionsergebnis erzielt wird, welches sich mit einer Wahrscheinlichkeit von 95 % um maximal 5 % von dem Ergebnis dieser Arbeit unterscheidet.

2.1.3.1 Synthetischer Datensatz

Von den 10000 Bildern werden 20 % zum späteren Testen des Neuronalen Netzwerks verwendet. Die 80 % werden wiederum in Trainings- (80 %) und Validierungsset (20 %) eingeteilt.

Mit einem Testset von 2000 Bildern liegt man deutlich über den in 2.1.3 ausgerechneten 385 Elementen und somit innerhalb des angegebenen Konfidenzintervalls.

2.1.3.2 Kundendatensatz

Ähnlich der Betrachtung des synthetischen Datensatzes muss auch für den nicht synthetischen Datensatz eine Testmengengröße gewählt werden, um im Konfidenzintervall zu liegen. Von den 773 DataMatrix-Codes werden 580 ($\approx 75\,\%$) zum anlernen (80 % der 580 Bildausschnitte mit DataMatrix-Codes) und validieren (20 % der 580 Bildausschnitte mit DataMatrix-Codes) verwendet. Die restlichen Bilder mit 193 DataMatrix-Codes werden zum Testen verwendet. Die Bilder ohne DataMatrix-Codes werden so aufgeteilt, dass 193 Bilder zum Testen und die restlichen 69 Bilder noch zum lernen respektive validieren verwendet werden.

Die Scans stammen zum Großteil von Faltschachteln und Labels aus dem pharmazeutischen Bereich; teilweise sind auch Validierungsbögen, Musterverpackungen und ähnliches in den Scans zu finden.

Mit dieser Aufteilung sind in den Testbildern insgesamt 193 DataMatrix-Codes vorhanden und zusätzlich noch 193 Bilder ohne DataMatrix-Codes. Dementsprechend kann von einem Stichprobenumfang von 386 ausgegangen werden, wodurch die in Abschnitt 2.1.3 Bedingungen für das geforderte Konfidenzintervall erfüllt werden.

2.1.3.3 Extraktion der Lerndaten

Das Anlernen der Neuronalen Netzwerke wird mit Bildausschnitten arbeiten. Es werden also typischerweise keine kompletten DataMatrix-Codes zum Anlernen verwendet sondern nur Teile derer.

Bildausschnitte zum Anlernen von nicht-DataMatrix-Code Daten können auch aus den Bildern mit DataMatrix-Codes extrahiert werden. Somit kann gewährleistet werden, dass in den Lern- und Validierungsdaten Bildausschnitte mit und ohne DataMatrix-Codes gleichermaßen vorhanden sind. Des Weiteren kommt hinzu, dass jeder DataMatrix-Code typischerweise in mehrere Ausschnitte unterteilt wird, somit stehen zum Anlernen der Neuronalen Netzwerke mehr Ausschnitte mit DataMatrix-Codes zur Verfügung als einzelne DataMatrix-Codes in den Trainings- und Validierungsdaten vorhanden sind. Für das Anlernen der Netzwerke ist somit für beide Datensätze die in Abschnitt 2.1.3 gestellte Forderung von $s \geq 385$ erfüllt.

2.2 Neuronale Netzwerke

Die Idee das menschliche Gehirn, beziehungsweise dessen Funktionalität auf den Computer abzubilden, existierte schon 1943. Hier wurde in [15] der Begriff des Neuronalen Netzwerkes eingeführt.

Bei den Neuronalen Netzwerken, welche in dieser Arbeit zum Einsatz kommen, handelt es sich um Feedforward Netzwerke, also Netzwerke ohne Rückkopplungen. Die mathematische Modellierung einer Nervenzelle ist in Abschnitt 2.2.1 zu finden. Weiter

ist in Abschnitt 2.2.2 die Verknüpfung mehrerer Nervenzellen beschrieben. Damit ein Neuronales Netzwerk als Klassifizierer eingesetzt werden kann, muss es angelernt werden. Wie das Anlernen funktioniert ist in Abschnitt 2.2.3 erläutert.

Die Umsetzung der Neuronalen Netzwerke in dieser Arbeit ist in Kapitel 3 zu finden.

2.2.1 Neuron

Wie eine menschliche Nervenzelle aufgebaut ist, ist in Abbildung 2.4 zu finden. Das Neuron sammelt Signale von anderen Neuronen, wobei die Axonterminals der anderen Neuronen über Synapsen mit den Dendriten des betrachteten Neurons verbunden sind. Also ist mit jedem Dendriten ein weiteres Neuron angeschlossen. Die Signale von den anderen Neuronen werden unterschiedlich stark an das betrachtete Neuron weitergeleitet. Somit existiert eine Verkettung von Neuronen, an deren Enden erneut Synapsen stehen, welche bestimmte Ereignisse, wie zum Beispiel eine Bewegung, auslösen. Die Verkettung wird in Abbildung 2.6 noch einmal verdeutlicht.

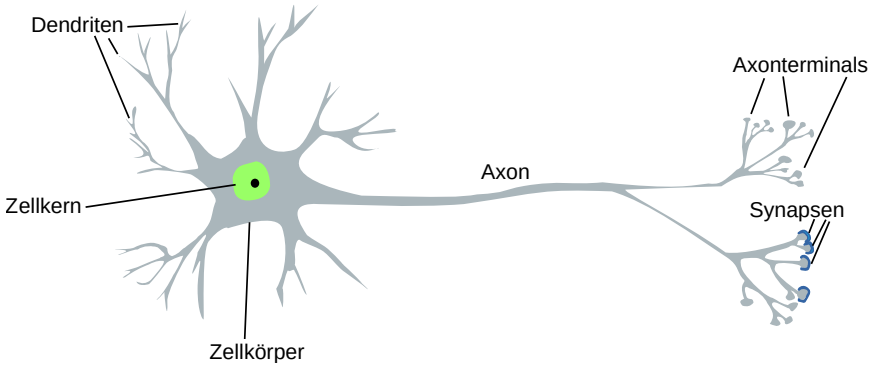


Abbildung 2.4: Schematische Darstellung eines Neurons (menschliche Gehirnzelle) sowie Benennung der einzelnen Komponenten.

Dieses Verhalten eines Neurons wird nun mathematisch abgebildet. Eine analoge Darstellung zu Abbildung 2.4 ist in Abbildung 2.5 zu finden. Die N Eingänge des mathematischen Neurons werden mit x_i bezeichnet, beziehungsweise zu einem Vektor

$$x_{\text{input}} = \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} \tag{2.10}$$

zusammengefasst. Analog hierzu werden die Synapsen als Gewichte w_i modelliert und zu einem Vektor

$$w = \begin{pmatrix} w_0 \\ \vdots \\ w_{N-1} \end{pmatrix} \tag{2.11}$$

zusammengefasst. Hinzu kommt noch ein weiterer Wert, b, welcher ein Bias repräsentiert. Dieses Bias wird für die Schwellwertverschiebung der Aktivierungsfunktion θ verwendet. Fügt man dieses Bias zu w und x_{input} hinzu erhält man

$$x'_{\text{input}} = \begin{pmatrix} x_{\text{input}} \\ 1 \end{pmatrix} \tag{2.12}$$

und

$$w' = \begin{pmatrix} w \\ b \end{pmatrix} \tag{2.13}$$

wodurch das Bias nun Bestandteil der Gewichtung geworden ist.

Wie schon erwähnt handelt es sich bei θ um die Aktivierungsfunktion, welche typischerweise nichtlinear ist. Wäre die Funktion linear, würde ein hintereinander schalten von Netzwerkschichten keinen weiteren Vorteil mit sich bringen, da die lineare Aktivierungsfunktion dazu führen würde, dass die Schichten zu einer einzelnen Schicht auf Grund der linearen Abhängigkeit kollabieren würde. Welche Aktivierungsfunktionen in dieser Arbeit zum Einsatz kommen ist in Abschnitt 3.1.1.2 zu finden.

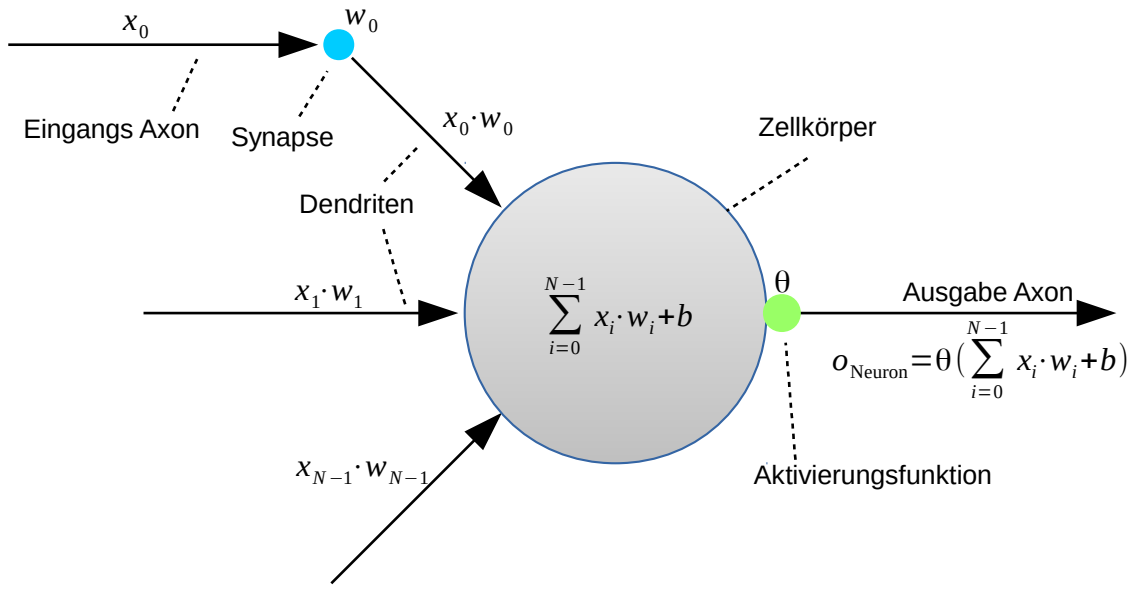


Abbildung 2.5: Künstliches Neuron basierend auf dem Vorbild aus der Natur.

Zusammenfassend ist zu sagen, dass alle Eingangswerte x'_{input} des Neurons mit den entsprechenden Gewichten w' multipliziert werden; die gewichteten Eingangswerte werden akkumuliert und daraufhin die Aktivierungsfunktion auf die Summe angewendet. Dies ist in Formel 2.14 zu sehen.

$$o_{\text{Neuron}} = \theta\left(x'^{T}_{\text{input}} w'\right) \tag{2.14}$$

In [16] findet man eine noch detailliertere Darstellung des biologischen Neurons und dem Übergang zum mathematischen Modell.

2.2.2 Netzwerk Architektur

Die in Abschnitt 2.2.1 beschriebenen mathematischen Neuronen werden zu Netzen verknüpft. Der Ausgang der einen Neuronenschicht ist mit dem Eingang der nächsten Neuronen verbunden. In Abbildung 2.6 ist dies schematisch dargestellt. In diesem Fall gibt es 3 Eingangsgrößen, welche mit den jeweiligen Neuronen der folgenden Schicht verbunden sind. Jede Verbindung impliziert ein Gewicht w_i für die jeweiligen Neuronen. Fasst man die Gewichtsvektoren w für alle Neuronen einer Schicht zu einer Gewichtungsmatrix

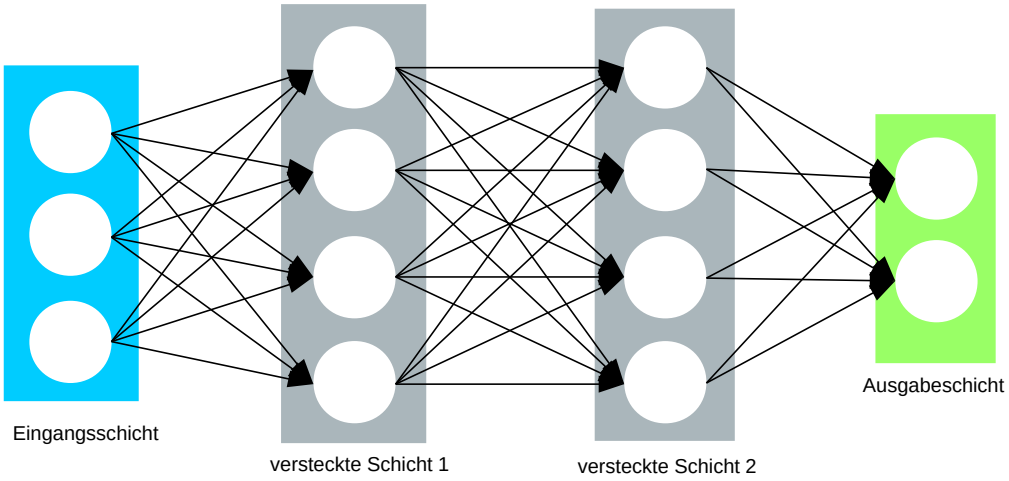


Abbildung 2.6: Schematische Darstellung eines dichten 3-4-4-2 künstlichen Neuronalen Netzes; die Eingangsschicht ist in blau dargestellt, die verdeckten Schichten sind grau und die Ausgabeschicht ist grün. Im Rahmen dieser Arbeit werden Eingangs- und Ausgabeschicht bei der Tiefe des Netzwerkes nicht berücksichtigt; dieses Netzwerk hat die Tiefe 2.

$$W = \begin{pmatrix} | & | & | & | \\ w_{\text{Neuron}_0} & \cdots & w_{\text{Neuron}_{N-1}} & b \\ | & | & | & | \end{pmatrix} \tag{2.15}$$

zusammen, kann man mit dem Ausdruck

$$o_{\text{layer}} = \theta\left(W^T x'_{\text{input}}\right) \tag{2.16}$$

die Ausgabe für eine komplette Netzwerkschicht beschreiben. Es folgt eine weitere verdeckte Schicht und eine Ausgabeschicht. Die finale Ausgabewerte können wie in Formel 2.17[10] beschrieben werden.

$$o(x'_{\text{input}}) = o_{\text{Ausgabeschicht}}(x'_{\text{input}}) = \theta_3\left(W_3^T \theta_2\left(W_2^T \theta_1\left(W_1^T \theta_0\left(W_0^T x'_{\text{input}}\right)\right)\right)\right) \tag{2.17}$$

[10] Die Indizes $[0;3]$ sind entsprechend Eingangsschicht, versteckte Schicht 1, versteckte Schicht 2, Ausgabeschicht zugeordnet.

Bei den in diesem Abschnitt beschriebenen Schichten handelt es sich stets um dichte Schichten, also alle Ausgaben der vorangehenden Schicht sind mit den entsprechenden Eingängen der nachfolgenden Schicht verbunden. In Abschnitt 3.1.1 wird noch auf eine weitere Schichtenart eingegangen.

2.2.3 Training

Um ein Neuronales Netzwerk als Klassifizierer verwenden zu können, muss es zunächst trainiert werden. Hierfür existiert ein Datensatz, in welchem zu jedem Eingangsvektor $x_{\text{in},m}$ ein korrespondierender Ausgabevektor t_m bekannt ist. Es existieren insgesamt M Vektorpaare, welche den Datensatz für das Training definieren. Wird eine solche Konfiguration zum Trainieren verwendet, spricht man vom überwachten Lernen. Sämtliche Gewichte des Neuronalen Netzwerkes werden dabei zu einer Variablen W^* zusammengefasst.

Das Training des Neuronalen Netzwerkes findet statt, in dem die Fehlerfunktion

$$E(W^*) = \frac{1}{2} \sum_{m=0}^{M-1} (o(x_m, W^*) - t_m)^2 \tag{2.18}$$

minimiert wird. Der Faktor $\frac{1}{2}$ dient hierbei der Berechenbarkeit, da er beim differenzieren von $E(W^*)$ wegfällt. Im Allgemeinen kann die Fehlerfunktion auch anders als hier nicht den quadratischen Fehler berechnen, sondern beispielsweise eine negative log likelihood Funktion oder im Fall einer mehrklassen Klassifizierung eine Kreuz-Entropie Fehlerfunktion sein.

Typischerweise ist die Fehlerfunktion nicht konvex (vgl. [17]); es kann also vorkommen, dass man mit klassischen Gradientenabstiegsverfahren in einem lokalen Minimum endet.

Um den Ausdruck 2.18 zu minimieren, kommt ein Backpropagation Algorithmus zum Einsatz, also wie schon erwähnt ein Gradientenabstiegsverfahren. Dieses wird in [18] in seinen Grundzügen beschrieben.

Die zentrale Formel dabei gibt an, dass die für jeden Iterationsschritt $(t+1)$ die Gewichte so gewählt werden, dass diese sich in Richtung des negativen Gradienten $\nabla E(W_t^*)$ bewegen. Dieser wird mit der Lernrate $\eta > 0$ multipliziert.

$$W_{t+1}^* = W_t^* - \eta \nabla E(W_t^*) \tag{2.19}$$

In jeder Iteration werden also die Gewichte W_{t+1}^* basierend auf den Gewichten W_t^* der vorangegangenen Iteration gewählt, wobei die Lernrate η mit den negativen Gradienten der Fehlerfunktion $\nabla E(W_t^*)$ multipliziert wird.

In der Praxis kommen typischerweise Variationen des oben genannten Gradientenabstiegsverfahrens zum Einsatz. In dieser Arbeit wird beispielsweise das SGD[11]-Verfahren verwendet. Dieses arbeitet nur auf einem Teil der Daten in jeder Iteration (Batch) und nicht auf allen M Trainingsdaten, (siehe Gleichung 2.18). Die Daten für einen Batch werden zu jeder Iteration zufällig ausgewählt. Dies hat mehrere Vorteile. Wie in Formel 2.20 zu sehen, wird die Summe nur noch über einen Teil der Daten berechnet.

$$E(W^*,k) = \frac{1}{2} \sum_{\text{batch}} (o(x_m, W^*) - t_m)^2 \tag{2.20}$$

Somit ist der Rechenaufwand geringer gegenüber dem klassischen Gradientenabstiegsverfahren. Des Weiteren ist der Gradient, welcher nur auf einem Teil der Daten berechnet wird im Vergleich mit den Gradienten aller Daten mit Rauschen behaftet, wodurch das SGD-Verfahren dazu neigt kleine lokale Minima zu vermeiden und nicht in ihnen zu verweilen. Die Größe der Batches s_{batch} wird als Kompromiss zwischen Genauigkeit des Gradienten bei jeder Iteration und Rechengeschwindigkeit ausgewählt.

Das SGD-Verfahren wird in dieser Arbeit mit dem Nesterov Momentum kombiniert, um die Stabilität und Konvergenz der Optimierung zu erhöhen. Untersuchungen zum SGD-Verfahren in Kombination mit weiteren Modifikationen, wie das Nesterov Momentum, sind in [19] zu finden.

Jeder Iterationsschritt wird in zwei Schritten berechnet:

$$v_{t+1} = \gamma v_t + \eta \nabla E(W^* + \gamma v_t, k_t) \tag{2.21}$$

$$W^*_{t+1} = W^*_t - v_{t+1} \tag{2.22}$$

Die Idee hinter einem Momentum ist die letzten Gradienten für den nächsten Iterationsschritt mit einzubeziehen. Im Fall des Nesterov Momentums wird der letzte Iterationsschritt zum einen direkt berücksichtigt (γv_t) und zum anderen auch noch indirekt über die Verwendung bei der erneuten Fehlerfunktion ($E(W^* + \gamma v_t, k_t)$). Der Parameter γ gibt dabei die Updaterate an. In dieser Arbeit wird

$$\gamma = 1 - \eta \tag{2.23}$$

gewählt, wie es in [20] empfohlen wird.

[11] **S**tochastic **G**radient **D**escent - Stochastischer Gradientenabstieg

2.3 Evolutionärer Algorithmus

Evolutionäre Algorithmen werden in [21] als eine Form von Evolution auf Computern beschrieben, welche als sich als Suchmethodik eignet, um zum einen Probleme zu lösen und zum anderen evolutionäre Systeme zu modellieren. Bezüglich der zu lösenden Probleme benötigt ein evolutionärer Algorithmus keine problemspezifischen Informationen; lediglich das Problem an sich muss bekannt sein und es muss eine Fitnessfunktion existieren, um eine Lösung bewerten zu können.

Evolutionäre Algorithmen basieren dabei stets auf der Evolutionstheorie von Charles Darwin [22], welche besagt, dass natürliche Selektion einer Population dafür sorgt, dass stets die am besten angepassten Individuen auf lange Sicht überleben. Weniger gut an die Umwelt angepasste Individuen haben eine geringere Überlebenschance. Die einzelnen Individuen einer Population unterscheiden sich von einander und diese Unterschiede werden vererbt. Bei der Vererbung kann es auch zu einer Mutation kommen, wodurch bei der Fortpflanzung die neu entstandenen Gene, welche auf den Genen der Eltern basieren, teilweise modifiziert werden.

Die in der Evolutionstheorie erwähnten Gene werden auch auf den Computer abgebildet und definieren ein Individuum. Es existiert eine initiale Population, welche sich durch Fortpflanzung mit steigender Anzahl an Durchläufen des Algorithmus immer besser an die Umwelt anpasst. Dadurch dass mit jeder Fortpflanzung das Verwerfen eines Individuums einhergeht, bleibt die Größe der Population konstant ($|P_0| = |P_i| \forall i$). Die Umwelt ist in diesem Fall abgebildet mit einem Fitnesswert, welcher definiert, wie gut ein Individuum angepasst ist beziehungsweise mit welcher Wahrscheinlichkeit ein Individuum zur Reproduktion eingesetzt wird; je fitter ein Individuum, desto größer ist die Wahrscheinlichkeit, dass es zur Reproduktion eingesetzt wird.

Ein Vorteil eines evolutionären Algorithmus als Suchheuristik gegenüber klassischen Optimierungsverfahren ist laut [23], dass sie nicht die klassischen Statiblitätsprobleme, wie zum Beispiel finden einer geeigneten Lernrate, besitzen.

Allerdings gibt es auch bei evolutionären Algorithmen einige Parameter, welche geeignet gewählt werden müssen, damit die Algorithmen entsprechend optimale Ergebnisse für die jeweiligen Problematiken liefern.

2.3.1 Ablauf

Der grundsätzliche Ablauf eines Evolutionären Algorithmus ist in Abbildung 2.7 visualisiert.

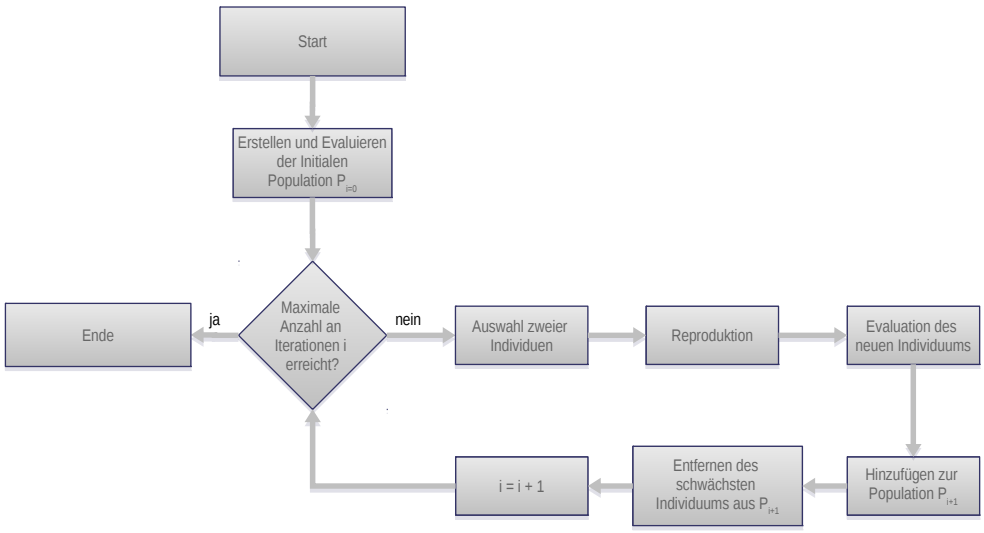


Abbildung 2.7: Flussdiagramm zur Visualisierung des Ablaufs eines Evolutionären Algorithmus. Dieser ist in zwei Phasen eingeteilt: Initialisierung und Hauptschleife. Während der Ausführung der Hauptschleife werden neue Individuen erstellt und evaluiert sowie ggf. zur Population zu gunsten eines anderen Individuums hinzugefügt. Wird die Hauptschleife beendet definiert das fitteste Individuum das Ergebnis des evolutionären Algorithmus.

Eine initiale Population P_0 wird zufällig erstellt und daraufhin evaluiert. Danach beginnt die Hauptschleife, in der jeweils zwei Individuen anhand des Fitnesswertes ausgewählt werden. Die ausgewählte Individuen werden zur Reproduktion verwendet. Während der Reproduktion findet die Kreuzung der Gene statt sowie die Mutation. Das neue Individuum wird evaluiert und, vorausgesetzt das Individuum ist besser als ein Individuum der aktuellen Population, zur Population hinzugefügt. Das schwächste Individuum der Population wird daraufhin verworfen. Wenn das neue Individuum hingegen ein niedrigeres Fitnesslevel aufweist, als alle anderen Individuen der Population, wird es direkt verworfen. Danach wird die Hauptschleife wieder von vorne durchlaufen, bis die Abbruchbedingung, in diesem Fall eine vorgegebene maximale Anzahl an Iterationen, erreicht wird. Die maximale Anzahl an Iterationen ist dabei problemspezifisch. Wie die Anzahl in dieser Arbeit gewählt wurde ist in Abschnitt 4.2.1 zu finden.

Das Ergebnis ist eine nach Fitnesslevel geordnete Population, wobei das Individuum, welches das höchste Fitnesslevel aufweist in der Population an erster Stelle steht.

2.3.2 Initial Population

Die initiale Population P_0 wird, wie in 2.3.1 schon beschrieben, mit zufälligen Individuen beziehungsweise mit Individuen mit zufälligen Eigenschaften erzeugt.

Die Größe der initialen Population ist allerdings einer der Hyperparameter, welcher in 4.2.1 behandelt wird. Die Größe hat in Kombination mit einem weiteren Hyperparameter, Mutationswahrscheinlichkeit p_{m} (siehe Abschnitt 2.3.3), einen Einfluss auf die Vielfalt der Population über die Laufzeit des Algorithmus. Wird die Population zu klein, im Extremfall mit zwei Individuen, so sind die weiteren Populationen P_i stets Kombinationen der ersten zwei Individuen von P_0. An dieser Stelle bekommt die Mutationswahrscheinlichkeit p_{m} einen größeren Einfluss auf die Variationen der jeweiligen Population P_i.

2.3.3 Fortpflanzung

Die Kreuzung zweier Individuen sieht wie folgt aus: Die Eigenschaften der zwei Individuen werden mit jeweils 50% Wahrscheinlichkeit in das neue Individuum übertragen. Somit ist das neue Individuum eine Kombination der beiden Eltern-Individuen.

Um eine Variation in die Fortpflanzung zu bringen, besteht die Möglichkeit einer Mutation. Hierfür wird der Hyperparameter p_{m} als Wahrscheinlichkeit, mit welcher eine Eigenschaft des aus Kombination zweier Eltern Individuen erzeugtes Individuum zufällig erzeugt wird, definiert. Jedes neue Individuum enthält somit mit der Wahrscheinlichkeit p_m einen neuen, zufälligen Wert, welcher nicht von einem Elternteil abhängt.

2.3.4 Rangordnug und Auswahl

In 2.3.3 wird der Fortpflanzungsmechanismus beschrieben, bei dem zwei Elternteile ein neues Individuum schaffen. Um die zwei Elternteile aus einer Population mit $|P| > 2$ auszuwählen, wird ein fitnessbasierter Ansatz ausgewählt. Jedes Individuum bekommt einen Fitnesswert f_j zugewiesen.

Sortiert man alle Individuen einer Population P nach den entsprechenden Fitnesswerten erhält man eine Rangordnung, wobei jedes Individuum einen Rang r_j erhält. Dies ist in Tabelle 2.1 dargestellt.

Rang r_j	**ID der Individuen** (j)	**Fitnesswert** f_j
1	1	55
2	2	43
3	0	20
4	3	6
5	4	3

Tabelle 2.1: Beispiel einer Population P mit $|P| = 5$, geordnet nach Fitnesswerten f_j der jeweiligen Individuen.

Zur Auswahl der entsprechenden zwei Elemente, welche zur Fortpflanzung verwendet werden sollen, werden zwei Ansätze in [24] beschrieben: Der erste Ansatz basiert

auf den Fitnesswert der Individuen und der zweite Ansatz basiert auf den Rängen der jeweiligen Individuen. In [24] wird außerdem noch aufgezeigt, dass eine auf der Rangordnung basierende Auswahl sich besser eignet, als eine Auswahl, die auf den Fitnesswerten basiert. Des Weiteren hat ein auf dem Fitnessrang basierendes Verfahren den weiteren Vorteil, dass durch die fixe Populationsgröße automatisch beim Erstellen der Rangordnung eine Normierung einher geht.

Die Wahrscheinlichkeit $p_{\text{rank}}(j)$ der Auswahl eines Individuums j wird dabei wie folgt berechnet:

$$p_{\text{rank}}(j) = \frac{|P| - r_j + 1}{\sum_{l=1}^{|P|} l} = 2\frac{|P| - r_j + 1}{|P|^2 + |P|} \tag{2.24}$$

Tabelle 2.1 kann somit um die entsprechende Auswahlwahrscheinlichkeit $p_{\text{rank}}(j)$ erweitert werden (vgl. Tabelle 2.2).

Rang r_j	**ID der Individuen** (j)	**Fitnesswert** f_j	$p_{\textbf{rank}}(j)$
1	1	55	$\frac{1}{3}$
2	2	43	$\frac{4}{15}$
3	0	20	$\frac{1}{5}$
4	3	6	$\frac{2}{15}$
5	4	3	$\frac{1}{15}$

Tabelle 2.2: Beispiel einer Population P mit $|P| = 5$, geordnet nach Fitnesswerten f_j der jeweiligen Individuen mit den entsprechenden Auswahlwahrscheinlichkeiten.

Es wird zu jedem Iterationsschritt zwei Individuen mit der Wahrscheinlichkeit $p_{\text{rank}}(j)$ für die Fortpflanzung ausgewählt.

2.4 Optimierung der Netztopologie mit Hilfe eines evolutionären Algorithmus

Zur Bestimmung der Netztopologie von Neuronalen Netzwerken existieren zur Zeit keine allgemeingültigen Regeln (vgl. [25]). Typischerweise wird sich einer optimalen Netztopologie mit der Versuch und Irrtum Methode genähert.

In dieser Arbeit wird dazu allerdings ein evolutionärer Algorithmus verwendet. Ein evolutionärer Algorithmus wird auf Grund der in Abschnitt 2.3 genannten Vorteile in

dieser Arbeit verwendet. Da es keine allgemeinen Regeln gibt, um eine Netzwerktopologie festlegen zu können aber eine Fitnessfunktion beschrieben werden kann, um eine Netzwerktopologie bewerten zu können, eignet sich der evolutionäre Algorithmus zur Lösung dieses Problems.

Die Individuen werden mehrere Parameter haben. Diese definieren die Netztopologie sowie weitere Parameter, wie zum Beispiel die Lernrate η oder den Schichttyp.

Es wird somit global mittels evolutionären Algorithmus nach dem optimalen Neuronalen Netzwerk gesucht, wobei die einzelnen Netzwerke mittels SGD-Verfahren mit Nesterov Momentum angelernt werden.

Für die Wahl der Hyperparameter wird der evolutionäre Algorithmus mit verschiedenen Parametersätzen ausgeführt, die Ergebnisse analysiert und daraufhin ein finaler Parametersatz bestimmt. Mit diesem wird erneut der evolutionäre Algorithmus ausgeführt, um die finale Netztopologie zu bestimmen.

Kapitel 3

Umsetzung

Dieses Kapitel beschreibt die Umsetzung der Neuronalen Netzwerke und des evolutionären Algorithmus sowie die Verwendung der Neuronalen Netzwerke zur Detektion von DataMatrix-Codes.

In Abschnitt 3.1 wird die Implementierung der Neuronalen Netzwerke im Detail beschrieben. Des Weiteren wird noch auf die Eigenschaften der Neuronalen Netzwerke und deren Schichten und Neuronen sowie den Lernprozess eingegangen. In Abschnitt 3.2 wird die Umsetzung des evolutionären Algorithmus dargestellt und in Abschnitt 3.3 wird aufgezeigt, wie man mit Hilfe des Neuronalen Netzwerkes DataMatrix-Codes in Druckerzeugnissen detektieren kann.

In dieser Arbeit wurde Größtenteils mit Python in der Version 2.7.9.4 gearbeitet. Die Neuronalen Netzwerke und der evolutionäre Algorithmus wurden in Python implementiert und es wurden zusätzliche Bibliotheken verwendet. Lasagne [20] Version 0.7 mit Theano [26] Version 0.7.0 bilden hierbei die Kernkomponenten. Theano ist eine Bibliothek, die mathematische Funktionalitäten liefert, insbesondere mathematische Operationen auf mehrdimensionalen Arrays. Auch wird die transparente Nutzung der GPU von Theano unterstützt. Lasagne basiert auf Theano und bietet ein Interface für Neuronale Netzwerke. Des Weiteren wurde die Python/C API verwendet, um eine C++ Integration der Neuronalen Netzwerke zu liefern. Hierfür wurde ein Minimalbeispiel implementiert.

3.1 Neuronales Netzwerk

Wie eingangs erwähnt, wurde das Neuronale Netzwerk mittels Lasagne und Theano implementiert. Es unterstützt die in den folgenden Abschnitten dargestellten Aktivierungsfunktionen und Schichttypen sowie das beschriebene Backpropagationverfahren, um das Netzwerk anzulernen. Das Klassendiagramm des Neuronalen Netzwerk Trainers ist in Abbildung 3.1 dargestellt.

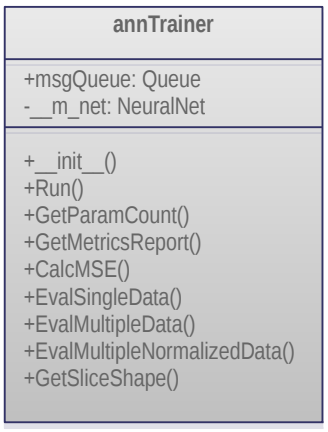


Abbildung 3.1: Klassendiagramm des Neuronalen Netzwerk Trainers mit den wichtigsten Variablen und sämtlichen Funktionen.

Die Queue wird zur Interprozesskommunikation verwendet und liefert unter anderem die Zwischenergebnisse der Lernprozesse sowie Fehlermeldungen an das GUI[1]. Die zentrale Variable ist `__m_net`. Diese beinhaltet das Neuronale Netzwerk, das über die `__init__()`-Funktion initialisiert wird. Welche Eigenschaften das Neuronale Netzwerk besitzt wird in den weiteren Abschnitten deutlich.

3.1.1 Netzwerkschichten

Künstliche Neuronalen Netzwerke werden maßgeblich über ihre Netzwerktopologien definiert. Die Netzwerktopologie besteht aus der Anzahl der Verbindungen und damit einhergehend der Anzahl der Parameter sowie der Art der Verbindungen. Die Schichttypen oder auch Layer eines Netzwerkes sowie die Aktivierungsfunktionen der einzelnen Schichten, die in dieser Arbeit verwendet werden, werden in den folgenden Unterabschnitten beschrieben.

3.1.1.1 Schichttypen

Im folgenden Abschnitt werden die in dieser Arbeit zum Einsatz kommenden Schichttypen beschrieben und verglichen. Welche Kombination aus Schichten im finalen Neuronalen Netzwerk zum Einsatz kommen werden, wird der evolutionäre Algorithmus zeigen (vgl. Abschnitt 4.2.2).

[1] **G**raphical **U**ser **I**nterface - Graphische Benutzeroberfläche

3.1.1.1.1 Dense Layer

Ein Dense Layer[2] ist eine Schichttopologie, bei der jedes Neuron einer Schicht mit sämtlichen Neuronen der darauf folgenden Schicht verknüpft ist. Neuronale Netzwerke, die nur aus Dense Layers bestehen werden auch als voll verbundenen Netzwerken bezeichnet. Ein Neuron der nachfolgenden Schicht hat somit genau so viele Eingänge und damit auch Gewichte (unter Vernachlässigung des Bias b), wie die Anzahl der Neuronen der vorherigen Schicht. Grafisch dargestellt wurde dies schon in Kapitel 2 in Abbildung 2.6. Dichte Schichten spiegeln die klassische Struktur von Neuronalen Netzwerken wieder. Diese sind in [18] näher beschrieben.

3.1.1.1.2 Convolutional Layer

Ein Convolutional Layer[3] führt eine Faltung auf den Eingangsdaten aus. Die Eingangsdaten sind stets Bilddaten $S(n_x, n_y)$, welche mit einem Filterkern $K^{u \times v}(m_x, m_y)$ gefaltet werden. Das Ergebnis dieser Faltung ist wieder ein Bild $S'(n_x, n_y)$. Der Ursprung des Filterkerns a_x, a_y ist beispielsweise für Filterkerne $K^{3\times 3}(m_x, m_y)$: $a_x = a_y = 2$, also das jeweils mittlere Element der Filtermatrix.

$$S'(n_x, n_y) = S(n_x, n_y) * K(m_x, m_y) \tag{3.1}$$

$$S'(n_x, n_y) = \sum_{w_x=1}^{u} \sum_{w_y=1}^{v} S(n_x + w_x - a_x, n_y + w_y - a_y) \cdot K(w_x, w_y) \tag{3.2}$$

Die zu Grunde liegende Idee eines Convolutional Layers ist es die Vorverarbeitung mittels Filtern in das Neuronale Netzwerk zu transferieren und somit während des Lernprozesses mit anzulernen. Ursprünglich entwickelt wurden Convolutional Layers aus der Beobachtung visueller Mechanismen von Lebewesen und daraufhin in [27] mathematisch Abgebildet.

Ein Netzwerk mit einer Faltungsschicht und darauffolgender Poolingschicht ist in Abbildung 3.2 zu sehen.

Es können mehrere Filterkerne pro Convolutional Layer eingesetzt werden; je Filterkern kommt ein weiteres Bild $S'(n_x, n_y)$ als Ergebnis der Faltungsschicht hinzu. Die Größe des Bildes $S'(n_x, n_y)$ ist dabei von der Größe des Eingangsbildes $S(n_x, n_y)$ sowie von der Größe des Filterkerns $K(m_x, m_y)$ abhängig. Hat das Eingangsbild beispielsweise eine Größe von 32 px $\times$ 32 px und der Filterkern eine Größe von 3 px $\times$ 3 px so hat das Ausgabebild eine Größe von 32 px $-$ 3 px $+$ 1 px $=$ 30 px[4]. Es werden also die Ränder

[2] Dichte Schicht

[3] Faltende Schicht

[4] Eingangsgröße - Filterkerngröße + 1 = Ausgangsgröße

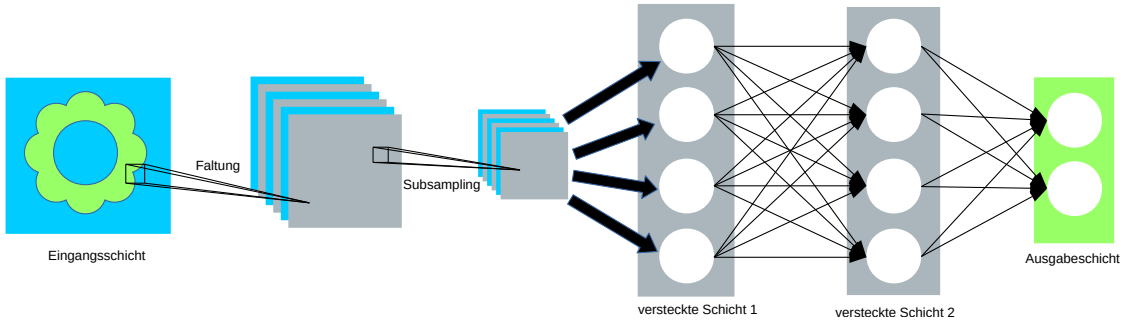


Abbildung 3.2: Schematische Darstellung eines künstlichen Neuronalen Netzes. Die Eingangsdaten haben ein Bild zum Inhalt. Gefolgt von der Eingangsschicht ist eine Faltungsschicht mit mehreren Filterkernen zu sehen. Daraufhin findet ein Subsampling statt. Diese Daten werden in zwei weitere versteckte Schichten geführt, gefolgt von der Ausgabeschicht.

nicht gesondert behandelt (es wird kein Padding oder ähnliches vorgenommen). In dieser Arbeit wird auf Graustufenbildern gearbeitet; liegen Farbinformationen vor könnte man dieses Modell erweitern und die Eingangsschicht hätte einen Tensor statt einer Matrix, welche die Informationen hält.

Bei einer Faltung werden nicht alle Pixel der Eingangsschicht mit allen Neuronen der folgenden Schicht verbunden, wie es bei einem Dense Layer der Fall ist (vgl. Abschnitt 3.1.1.1.1). Es wird von einer lokalen Konnektivität gesprochen, durch die sich die Anzahl an Parametern, im Vergleich zu beispielsweise dichten Schichten, bei gleicher Anzahl an Neuronen verringert.

3.1.1.1.3 Maxpooling Layer

Maxpooling Layer werden typischerweise nach Convolutional Layer eingesetzt. Die Anwendung des Maxpooling geht mit einem Subsampling einher. Das Bild $S(n_x, n_y)$ wird in nicht überlappende Rechtecke unterteilt. Von jeden Rechteck wird der maximale Wert verwendet und ein neues Bild $S'(n_x, n_y)$ geschaffen, welches entsprechend der Größe der Rechtecke eine niedrigere Auflösung besitzt. Liegt $S(n_x, n_y)$ zum Beispiel in der Auflösung 32 px $\times$ 32 px vor und es wird ein Maxpooling mit einem Rechteck der Größe 2 px $\times$ 2 px vorgenommen, wird $S'(n_x, n_y)$ die Größe 16 px $\times$ 16 px haben und aus den jeweiligen Maxima der 2 px $\times$ 2 px Blöcke aus $S(n_x, n_y)$ bestehen. Wird ein Bild $S(n_x, n_y)$ (N_x px $\times$ N_y px) mit einem Block der Größe k_x px $\times$ k_y px verarbeitet entsteht ein neues, verkleinertes Bild $S'(n_x, n_y)$ der Größe $\frac{N_x}{k_x}$ px $\times$ $\frac{N_y}{k_y}$ px.

Maxpooling wird verwendet, um die Berechnungskosten für die darauffolgenden Schichten zu reduzieren. In der Blockgröße geht mit dem Maxpooling auch eine gewisse

Translationsinvarianz einher. Eine Verschiebung um einen Pixel innerhalb eines 2 px × 2 px Blocks hätte auf das Ergebnis des Maxpoolings keine Auswirkung.

Grundsätzlich sind auch andere Poolingtechniken, neben Maxpooling, gebräuchlich. Eine Variante wäre es den Mittelwert des Blockes zu bilden, um die Dimension zu reduzieren. In [28] wird allerdings gezeigt, dass das Maxpooling in der Praxis bessere Ergebnisse liefert als andere Downsampling Methoden.

Eine Besonderheit bei Maxpooling Schichten im Rahmen des Backpropagation Algorithmus muss noch erwähnt werden: Während der Fehlerrückführung muss entsprechend komplementär zum Downsampling mittels Maxpooling ein Upsampling stattfinden. Hierfür wird der Wert vor dem Upsampling Schritt auf ein entsprechend der Blockgröße der Schicht großen Block repliziert.

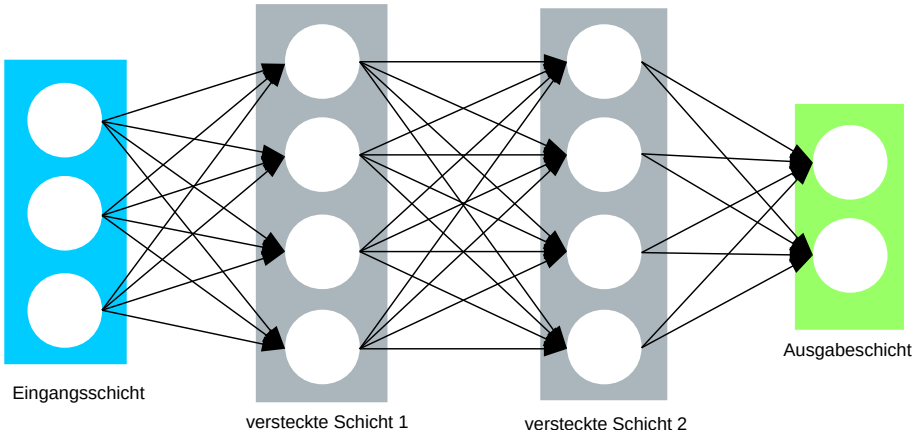
3.1.1.1.4 Dropout Layer

Dropout Layer werden zur Prävention von Overfitting[5] eingesetzt (vgl. [29]). Dabei werden einzelne Neuronen samt ihrer Verbindungen zu den vorherigen und nachfolgenden Schichten während des Trainings zufällig Ausgesetzt und beim Training nicht verwendet. Dieses Verhalten ist in Abbildung 3.3.2 dargestellt.

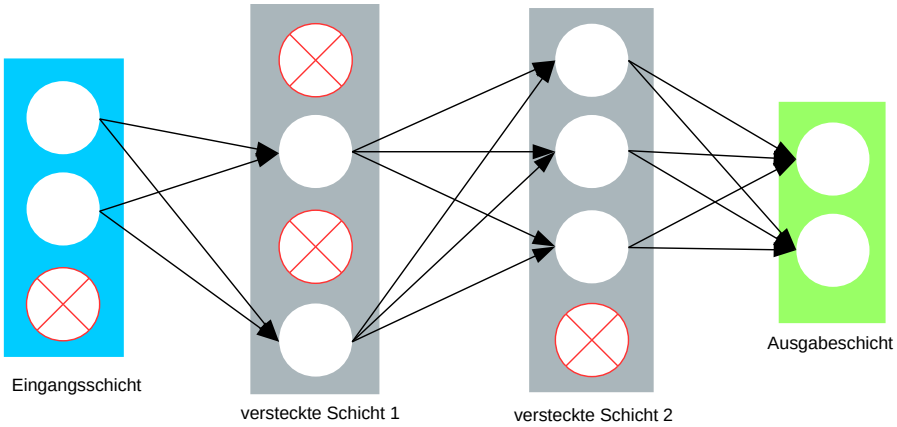
Dropout Schichten sollen laut [29] Overfitting verhindern, da eine zu hohe Spezialisierung des Modells nicht ohne weiteres möglich ist, da sich das Modell in jeder Lerniteration verändert. Die Auswertung hingegen findet mit dem kompletten Netzwerk statt. Die grundsätzliche Idee ist, dass sich ein Netzwerk mit Dropout Schichten wie viele einzelne Netzwerke ohne Dropout Schichten verhält, welche sich Gewichte teilen. Ein Netzwerk mit N_{Neuronen} Neuronen kann als Ansammlung von bis zu $2^{N_{\text{Neuronen}}}$ möglichen ausgedünnten Netzwerken angesehen werden[6], welche alle Gewichte teilen. Zur Testzeit werden allerdings nicht bis zu $2^{N_{\text{Neuronen}}}$ Netzwerke ausgewertet sondern nur eines, welches nicht ausgedünnt ist.

[5] Überanpassung

[6] Zwei mögliche Zustände (Neuron wird verwendet/Neuron wird nicht verwendet) für alle N_{Neuronen} Neuronen.



3.3.1 Ein 3-4-4-2-Neuronales Netzwerk ohne die Anwendung von Dropout Layern.



3.3.2 Ein 3-4-4-2-Neuronales Netzwerk mit der Anwendung von Dropout Layern.

Abbildung 3.3: Oben ist ein 3-4-4-2-Neuronales Netzwerk ohne die Anwendung von Dropout Layern zu sehen und unten ist das selbe Netzwerk unter Verwendung von Dropout Layern dargestellt. Einzelne Neuronen samt deren Verbindungen werden hierbei zufällig ausgesetzt.

Die Dropout Wahrscheinlichkeit p_{drop} wird für die inneren Schichten in [29] mit $p_{\text{drop}} = 0{,}5$ angegeben und für die Eingangsschicht wird $p_{\text{drop}} \in [0;0{,}5]$ empfohlen.

3.1.1.2 Aktivierungsfunktion

Wie in 2.2 schon erwähnt wird nach jeder Gewichtung der Neuronen eine typischerweise nichtlineare Funktion θ auf die Ausgangswerte ausgeführt (vgl. Gleichung 2.20). Es existieren mehrere verschiedene Aktivierungsfunktionen, welche im Rahmen dieser Arbeit zum Einsatz kommen. Diese werden in den folgenden Abschnitten vorgestellt

und verglichen. Sämtliche Aktivierungsfunktionen müssen für den Backpropagation-Algorithmus differenzierbar bzw. stückweise differenzierbar sein.

3.1.1.2.1 Sigmoid Funktion

Die Sigmoid-Funktion ist die klassische Aktivierungsfunktion, die aus den Anfängen der Neuronalen Netzwerke stammt. Sie ist, wie die Neuronalen Netzwerke an sich, inspiriert durch die Natur. Die Sigmoid-Funktion wird in [30] näher vorgestellt und ist in Abbildung 3.4 dargestellt.

$$f_{\text{sigmoid}}(x) = \frac{1}{1+e^{-x}} \tag{3.3}$$

$$\frac{df_{\text{sigmoid}}(x)}{dx} = \left[1 - f_{\text{sigmoid}}(x)\right] \cdot f_{\text{sigmoid}}(x) \tag{3.4}$$

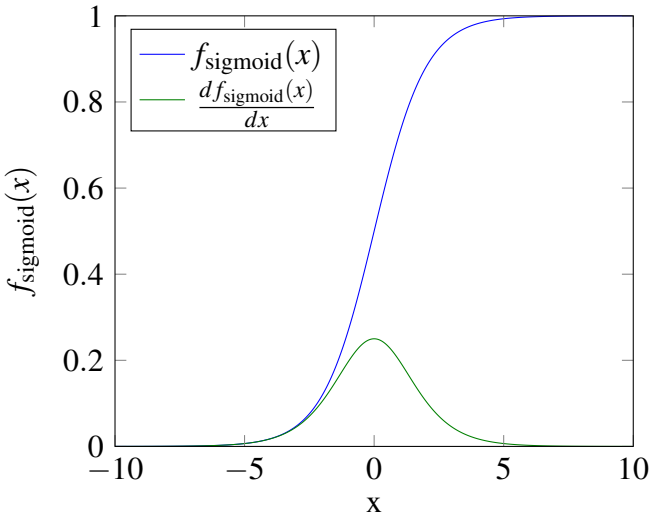


Abbildung 3.4: Sigmoid Funktion sowie die Ableitung der Sigmoid Funktion.

Der Definitionsbereich $\mathbb{D}_{\text{sigmoid}}$ der Funktion f_{sigmoid} ist $\mathbb{D}_{\text{sigmoid}} = \mathbb{R}$ und der Wertebereich $\mathbb{W}_{\text{sigmoid}}$ ist $\mathbb{W}_{\text{sigmoid}} = [0;1]$.

3.1.1.2.2 Softmax

Die Softmax-Funktion oder auch normalisierte Exponentialfunktion bildet einen K-dimensionalen Vektor z auf einen K-dimensionalen Vektor $f_{\text{softmax}}(z)$ ab. Der Definitionsbereich $\mathbb{D}_{\text{softmax}}$ von $f_{\text{softmax}}(z)$ ist $\mathbb{D}_{\text{softmax}} = \mathbb{R}$; der Wertebereich $\mathbb{W}_{\text{softmax}}$

ist $\mathbb{W}_{\text{softmax}} = [0;1]$ und die Summe über alle Elemente ergibt 1. Die jeweilige j-te Komponente des Eingangsvektors v berechnet sich wie folgt:

$$f_{\text{softmax}}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{3.5}$$

Diese nichtlineare Funktion wird im Kontext der Neuronalen Netzwerke häufig bei der Ausgabeschicht eingesetzt. Soll das Neuronale Netzwerk beispielsweise eine Klassifikationsaufgabe lösen kann man das Ergebnis der Ausgabeschicht als Wahrscheinlichkeit einer Klassenzugehörigkeit interpretieren. In dieser Arbeit existieren, wie in Abschnitt 2.1 schon erwähnt, zwei Klassen. Mit einer Ausgabeschicht mit der Softmax-Funktion kann somit die Wahrscheinlichkeit eines vorhandenen DataMatrix-Codes angegeben werden.

3.1.1.2.3 Tangens Hyperbolicus (tanh)

Bei der Tangens Hyperbolicus Aktivierungsfunktion handelt es sich um eine Funktion, die Ähnlichkeiten zur Sigmoid-Aktivierungsfunktion aufweist. Wie man in Abbildung 3.5 sehen kann ist die Form der Funktion um den Nullpunkt ähnlich. Beide Funktionen haben den Definitionsbereich $\mathbb{D}_{\text{tanh}} = \mathbb{D}_{\text{sigmoid}} = \mathbb{R}$ unterscheiden sich aber im Wertebereich $\mathbb{W}_{\text{tanh}} = [-1;1]$.

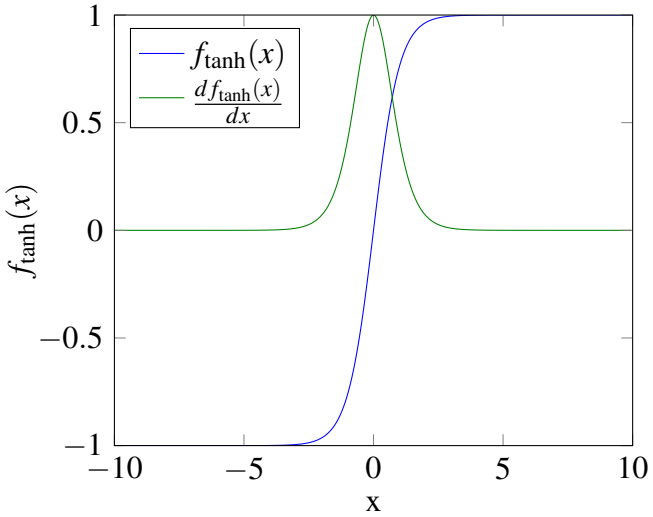


Abbildung 3.5: Tangens Hyperbolicus Funktion und die Ableitung der Tangens Hyperbolicus Funktion.

$$f_{\tanh}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.6}$$

$$\frac{df_{\tanh}(x)}{dx} = 1 - \tanh^2(x) = 1 - \frac{\left(e^x - e^{-x}\right)^2}{\left(e^x + e^{-x}\right)^2} \tag{3.7}$$

In [31] wird darauf eingegangen, dass die Sigmoid-Funktion auf Grund des von 0 verschiedenen Mittelwertes das Lernen mittels Gradientenabstiegsverfahrens verlangsamt. Dieses Verhalten tritt bei der Tangens Hyperbolicus Aktivierungsfunktion nicht auf, da der Funktionswert im Mittel über den Definitionsbereich 0 beträgt.

3.1.1.2.4 Lineare Funktion

Eine lineare Funktion $f_{\text{linear}(x)}$ sowie die Ableitung der linearen Funktion $\frac{df_{\text{linear}}(x)}{dx}$ lassen sich typischerweise in weniger Rechenschritten berechnen als die oben vorgestellten nichtlinearen Funktionen. Sie ist in Abbildung 3.6 visualisiert. Des Weiteren existiert bei einer linearen Aktivierungsfunktion nicht das "Vanishing Gradient Problem" (siehe Abschnitt 3.1.2.2).

$$f_{\text{linear}}(x) = mx + c \tag{3.8}$$

$$\frac{df_{\text{linear}}(x)}{dx} = m \tag{3.9}$$

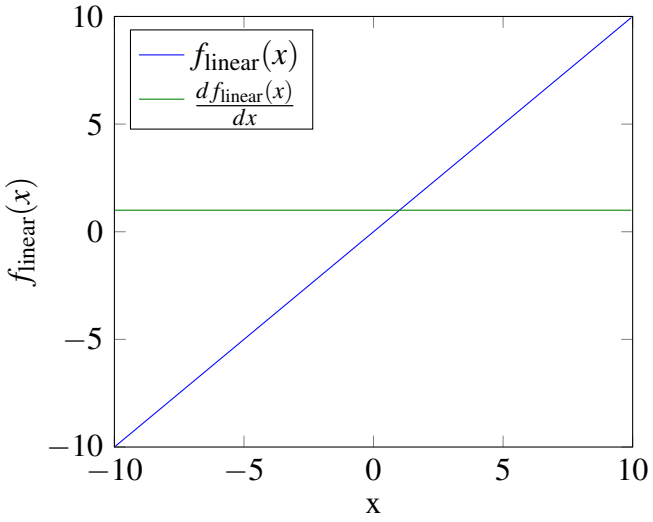


Abbildung 3.6: Lineare Funktion und die Ableitung mit $m = 1$.

Bildet man allerdings ein mehrschichtiges Netzwerk, welches $f_{\text{linear}}(x)$ als Aktivierungsfunktionen verwendet, ist die Ausgabe eine lineare Kombination der Eingangswerte und könnte somit auch mit einer einzigen linearen Schicht abgebildet werden. Sowohl der Definitionsbereich $\mathbb{D}_{\text{linear}}$ als auch der Wertebereich $\mathbb{W}_{\text{linear}}$ der Funktion $f_{\text{linear}(x)}$ ist $\mathbb{D}_{\text{linear}} = \mathbb{W}_{\text{linear}} = \mathbb{R}$.

3.1.1.2.5 Rectifed Linear Function

Eine ReL-Funktion[7] vereint die Vorteile der linearen Aktivierungsfunktion $f_{\text{linear}}(x)$ mit denen der Nichtlinearität (vgl. [32]).

$$f_{\text{ReL}}(x) = max(0,x) = \begin{cases} x & \text{für } x > 0 \\ 0 & \text{sonst} \end{cases} \tag{3.10}$$

$$\frac{df_{\text{ReL}}(x)}{dx} = \begin{cases} 0 & \text{für } n < 0 \\ 1 & \text{für } n > 0 \end{cases} \tag{3.11}$$

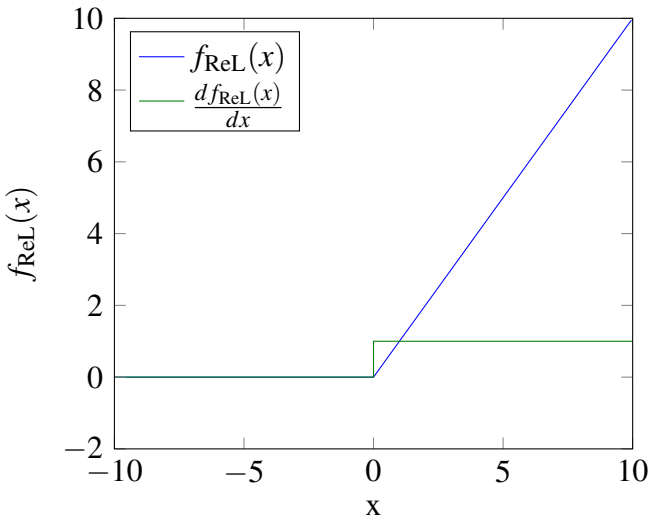


Abbildung 3.7: ReL-Funktion und die Ableitung der ReL-Funktion.

In Abbildung 3.7 ist die ReL-Funktion und die Ableitung der ReL-Funktion dargestellt. Auffallend ist, dass die Ableitung $\frac{df_{\text{ReL}}(x)}{dx}$ für $x = 0$ nicht definiert ist; in der

[7] **Re**ctifed **L**inear Function - Berichtigte Lineare Funktion

Praxis wird für den Fall $x = 0$ allerdings der rechtsseitige oder linksseitige Grenzwert verwendet. Im Rahmen dieser Arbeit wird für $x = 0$ der linksseitige Grenzwert $\lim_{x \to 0^-} \frac{df_{\text{ReL}}(x)}{dx} = 0$ der Ableitung hergenommen.

Ein Nachteil der ReL-Aktivierungsfunktion innerhalb eines Netzwerkes ist, dass der Gradient für $\frac{df_{\text{ReL}}(x<0)}{dx}$ konstant 0 ist und somit bei einer ungünstigen Initialisierung es dazu kommen kann, dass ein Neuron nie aktiv wird und dadurch keinen Beitrag zur Ausgabe des Gesamtnetzwerkes leistet.

Der Definitionsbereich $\mathbb{D}_{\text{ReL}}$ der Funktion $f_{\text{ReL}}(x)$ ist $\mathbb{D}_{\text{ReL}} = \mathbb{R}$ und der Wertebereich $\mathbb{W}_{\text{ReL}}$ ist $\mathbb{W}_{\text{ReL}} = [0;\infty)$.

3.1.1.2.6 Leaky Rectifed Linear Function

LReL-Funktionen[8] sind ähnlich aufgebaut wie ReL-Funktionen, nur dass sie im Fall $x \leq 0$ nicht den Wert 0 annehmen sondern auch einen linearen Verlauf haben (vgl. [32]).

$$f_{\text{LReL}}(x) = \begin{cases} x & \text{für } x > 0 \\ mx & \text{sonst} \end{cases} \tag{3.12}$$

$$\frac{df_{\text{LReL}}(x)}{dx} = \begin{cases} m & \text{für } n < 0 \\ 1 & \text{für } n > 0 \end{cases} \tag{3.13}$$

Der Parameter m nimmt dabei typischerweise kleine Werte an und wird innerhalb dieser Arbeit, wie in [32], zu $m = 0{,}01$ gesetzt. Daraus resultiert die in Abbildung 3.8 dargestellte Funktion.

[8] **L**eaky **Re**ctifed **L**inear Function - Leckende Berichtigte Lineare Funktion

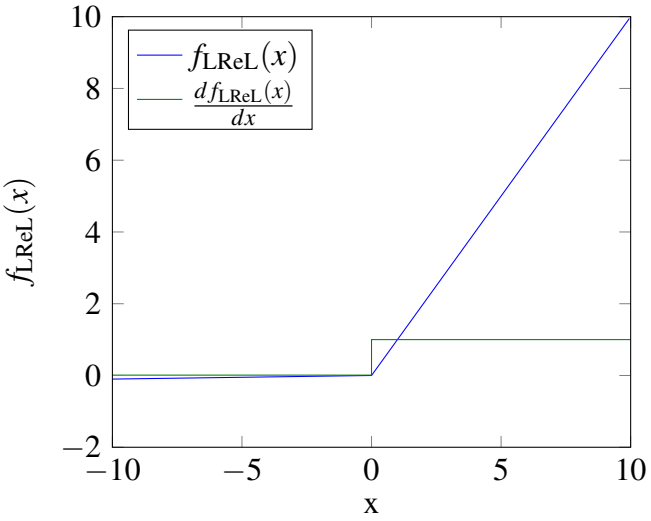


Abbildung 3.8: LReL-Funktion sowie die Ableitung; $m = 0{,}01$.

Wie man sieht unterscheidet sich die Funktion nicht viel von der in Abbildung 3.7 dargestellten Funktion. Die leichte positive Steigung von $m = 0{,}01$ sorgt allerdings für einen von 0 verschiedenen Gradienten und ist somit robuster während des Lernprozesses (siehe [32]). Des Weiteren ist der Definitionsbereich $\mathbb{D}_{\mathrm{LReL}}$ der Funktion $f_{\mathrm{LReL}}(x)$ $\mathbb{D}_{\mathrm{LReL}} = \mathbb{R}$ und auf Grund der Steigung $m < 0$ ist der Wertebereich $\mathbb{W}_{\mathrm{LReL}}$ der Funktion $f_{\mathrm{LReL}}(x)$ auch $\mathbb{W}_{\mathrm{LReL}} = \mathbb{R}$.

3.1.1.2.7 Zusammenfassung

In Tabelle 3.1 sind alle Aktivierungsfunktionen, die in dieser Arbeit verwendet werden, zusammengefasst.

Aktivierungsfunktion	**Differenzierbar**	$\mathbb{D}$	$\mathbb{W}$
Sigmoid	ja	$\mathbb{R}$	$[0;1]$
Softmax	ja	$\mathbb{R}$	$[0;1]$
Tangens Hyperbolicus	ja	$\mathbb{R}$	$[-1;1]$
Lineare Funktion	ja	$\mathbb{R}$	$\mathbb{R}$
ReL-Funktion	stückweise	$\mathbb{R}$	$[0;\infty)$
LReL-Funktion	stückweise	$\mathbb{R}$	$\mathbb{R}$

Tabelle 3.1: Übersicht aller in dieser Arbeit verwendeten Aktivierungsfunktionen.

Im Vergleich der Aktivierungsfunktionen kann man festhalten, dass die Sigmoid Funktion, die Softmax Funktion sowie die Tangens Hyperbolicus differenzierbar und

nichtlinear sind. Im Gegensatz dazu steht die ReL-Funktion und LReL-Funktion. Diese sind nur stückweise differenzierbar. Vorteil der ReL- und LReL-Funktion sowie der linearen Funktion, dass diese nicht vom "Vanishing Gradient Problem" betroffen sind und dass die Berechnung der Werte für diese Funktion und die entsprechenden Ableitungen weniger Rechenschritte benötigen, als für die anderen Funktionen. Alle Funktionen haben außerdem gemeinsam, dass sie monoton steigend sind.

3.1.1.3 Initialisierung der Gewichte

Die Initialisierung der einzelnen Gewichte wird nach dem in [31] vorgestellten Verfahren realisiert. Hierbei werden die Gewichte zufällig um den Wert 0 initialisiert.

Damit das Eingangssignal während der Vorwärts- und der Rückwärtsausbreitung des Netzwerkes nicht zu klein respektive zum Initialisierungszeitpunkt zu groß wird, wird die Varianz für die einzelnen Layer so festgelegt, dass ein solches Verhalten nicht auftritt.

Bei den verschiedenen Schichttypen, die in dieser Arbeit verwendet werden, kommt es nicht immer zu einer Sättigung, weswegen ein zu großes Signal gegebenenfalls kleine Gewichte nach sich zieht, um dieses zu kompensieren. Um zu den kleinen Gewichten zu kommen, wären, je nach Konfiguration und Art des Lernalgorithmus, viele Iterationen von Nöten; die Anlernzeit verlängert sich. Analog zu der Argumentation bei großen Signalen kann man auch für zu kleine Signale argumentieren. Ist ein Signal zu klein muss es gegebenenfalls sehr verstärkt werden, damit das Signal zur Ausgabe des Neuronalen Netzwerkes beiträgt; auch hierfür könnten viele Iterationsschritte nötig sein.

Die Wahl der Varianz für eine einzelne Schicht wird, wie in [33] beschrieben, von der Anzahl an Eingangs- und Ausgangsverbindungen fest gemacht.

Sei $W_{\text{Schicht},j} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ die Gewichtungsmatrix der Schicht j, so besitzt diese d_{in} Eingangsverbindungen und d_{out} Ausgangsverbindungen. Die Varianz wird zu

$$\text{Var}(W_{\text{Schicht},j}) = \frac{2}{d_{\text{in}} + d_{\text{out}}} \tag{3.14}$$

gewählt. Dies wird als Kompromiss zwischen Vorwärts- und Rückwärtsausbreitung verwendet. Die Idee hinter dieser Auswahl der Varianz ist folgende:

Sei x'_{input} der Eingangsvektor zur oben beschriebenen Schicht j und $o_{\text{Schicht},j}$ der entsprechende Ergebnisvektor. Es wird zunächst nur die Vorwärtsausbreitung betrachtet. Des Weiteren wird die Aktivierungsfunktion vernachlässigt und die Annahme gemacht, dass die Ein- und Ausgangsvektoren unabhängig von einander und gleichverteilt sind. Weiter kann man $o_{\text{Schicht},j}$ schreiben als

$$o_{\text{Schicht},j} = w_1 x_1 + w_2 x_2 + \cdots + w_{d_{\text{in}}} x_{d_{\text{in}}} \tag{3.15}$$

so lässt sich die Varianz der Elemente des Ergebnisvektors y mit

$$\text{Var}(w_i x_i) = E[x_i]^2 \text{Var}(w_i) + E[w_i]^2 \text{Var}(x_i) + \text{Var}(w_i)\text{Var}(x_i) \tag{3.16}$$

berechnen. Wie schon beschrieben, kommt eine mittelwertsfreie Verteilung zum Einsatz, wodurch die Therme mit dem Erwartungswert wegfallen.

$$\mathrm{Var}(w_i x_i) = \mathrm{Var}(w_i)\mathrm{Var}(x_i) \tag{3.17}$$

Aus dieser Formel für ein einzelnes Element i kann

$$\mathrm{Var}(o_{\mathrm{Schicht},j}) = \mathrm{Var}\left(\sum_{i=1}^{d_{\mathrm{in}}} w_i x_i\right) = d_{\mathrm{in}}\mathrm{Var}(w_i)\mathrm{Var}(x_i) \tag{3.18}$$

gefolgert werden. Soll die Varianz der Eingangswerte der Varianz der Ausgangswerte entsprechen, sollte $d_{\mathrm{in}}\mathrm{Var}(w_i) = 1$ gewählt werden. Für die Gewichte bedeutet dies, dass

$$\mathrm{Var}(w_i) = \frac{1}{d_{\mathrm{in}}} \tag{3.19}$$

gewählt wird. Analog zu dieser Herleitung kommt entsprechend zu der Rückwärtsausbreitung des Signals

$$\mathrm{Var}(w_i) = \frac{1}{d_{\mathrm{out}}} \tag{3.20}$$

heraus. Da häufig nicht $d_{\mathrm{in}} = d_{\mathrm{out}}$ gilt, werden die Werte, wie in Formel 3.14 beschrieben, initialisiert und ist somit ein Kompromiss zwischen der Varianz der Vorwärts- und der Rückwärtsausbreitung. In [31] wird empirisch gezeigt, dass diese Initialisierung zu einem schnelleren Konvergenzverhalten führt, als eine klassische mittelwertsfreie Gleichverteilung.

Im Rahmen dieser Arbeit wird eine Gleichverteilung $\mathcal{U}(b_{\mathrm{u}}, b_{\mathrm{o}})$ verwendet. Die Varianz einer Gleichverteilung lässt sich wie folgt berechnen:

$$\mathrm{Var}(\mathcal{U}(b_{\mathrm{u}}, b_{\mathrm{o}})) = \frac{1}{12}(b_{\mathrm{o}} - b_{\mathrm{u}})^2 \tag{3.21}$$

Mit $b_{\mathrm{u}} = -b_{\mathrm{o}} = b_{\mathrm{s}}$ (Symmetrie um Mittelwert 0) und der Forderung $\mathrm{Var}(\mathcal{U}) = \frac{2}{d_{\mathrm{in}}d_{\mathrm{out}}}$ folgt:

$$\mathrm{Var}(\mathcal{U}(-b_{\mathrm{s}}, b_{\mathrm{s}})) = \frac{b_{\mathrm{s}}^2}{3} = \frac{2}{d_{\mathrm{in}}d_{\mathrm{out}}} \tag{3.22}$$

Somit wird

$$b_{\mathrm{s}} = \sqrt{\frac{6}{d_{\mathrm{in}}d_{\mathrm{out}}}} \tag{3.23}$$

gewählt.

3.1.2 Backpropagation

Wie in Abschnitt 2.2.3 schon beschrieben kommt ein SGD-Verfahren mit Nesterov Momentum zum Einsatz. Dieser Abschnitt geht noch auf Details und Parameter des SGD-Verfahrens ein, welche in Abschnitt 2.2.3 nicht beschrieben wurden oder sich in der Umsetzung unterscheiden.

3.1.2.1 Stochatisches Gradienten Abstiegsverfahren und Momentum

Dieser Abschnitt beschreibt die Konfiguration des SGD-Verfahrens. Der Parameter der Updaterate γ wie beschrieben zu $\gamma = 1 - \eta$ gesetzt. Neu ist, dass die Lernrate η von dem Iterationsschritt t abhängig ist. Daraus folgt, dass auch die Updaterate γ vom Iterationsschritt abhängig ist und somit stets neu gesetzt wird:

$$\gamma_t = 1 - \eta_t \tag{3.24}$$

Die Lernrate η_t wird allerdings nicht zu jeder Iteration t verändert sondern nur, wenn seit mindestens fünf Iterationen keine Verbesserung des Fehlers E_t stattgefunden hat. Ist dieser Fall eingetreten lautet die Updateregel wie folgt:

$$\eta_{t+1} = 0{,}5\eta_t \tag{3.25}$$

Diese Updateregel ist in [17] zu finden; der Faktor $0,5$ entspricht dabei der Empfehlung. Die einzige Änderung ist, dass das Update nicht stets nach fünf Iterationen stattfindet sondern erst nachdem sich der Fehler für mehr als fünf Iterationen nicht mehr verbessert hat. Dadurch kann mit einer größeren Lernrate η begonnen werden, wodurch das Training schneller voranschreitet.

Ein weiterer Parameter ist die Batchgröße s_{batch}. In dieser Arbeit wird mit einer Batchgröße von 128 Daten gerechnet. Diese Batchgröße wurde als Kompromiss zwischen Anzahl an benötigten Rechenoperationen und Stabilität des Gradienten gewählt. Wird die Batchgröße zu groß werden viele Rechenoperationen benötigt; wird die Batchgröße zu klein ist der berechnete Gradient nicht repräsentativ für den Datensatz.

Der Zähler für das Early Stopping, welches in Abschnitt 3.1.2.3 erklärt wird, ist ein weiterer Parameter des Lernprozesses. Sobald es seit i_{es} Iterationen keine Verbesserung im Fehler der Validierungsmenge E_{valid} gab, wird das Training abgebrochen und die Gewichte der Iteration mit dem kleinsten Validerungsfehler E_{valid} verwendet. Die Schranke i_{es} wurde im Rahmen dieser Arbeit auf $i_{\text{es}} = 25$ gesetzt, um ggf. auftretendes Overfitting zu verhindern.

Ein weiterer Parameter, den es zu setzen gibt, ist die maximale Anzahl an Iterationsschritten. Diese wurde auf 500 gesetzt; mit der Early Stopping Schranke $i_{\text{es}} = 25$ wurde die maximale Anzahl an Iterationen zu keinem Zeitpunkt erreicht. Diese Werte wurden ähnlich den Werten in [34] gewählt.

3.1.2.2 Vanishing Gradient Problem

Das "Vanishing Gradient Problem" ist ein Problem des Backpropagation Algorithmus. Die Sigmoid Funktion hat beispielsweise einen Gradienten mit einem kleinen Wertebereich (vgl. Abbildung 3.4). Bei einer tiefen Netztopologie mit der Sigmoid Funktion als Aktivierungsfunktion für alle Schichten kommt es bei der Fehlerrückführung zu einer Multiplikation vieler Werte kleiner oder gleich 0,25 ($\max\left(\frac{df_{\text{sigmoid}(x)}}{dx}\right) = 0{,}25$). Dies hat zur Folge, dass der Gradient, je weiter er in Richtung des Eingangs des Neuronalen Netzwerkes fortschreitet, typischerweise kleiner wird, da die Gewichte typischerweise um den Wert 0 initialisiert werden. Ein Lösungsansatz für dieses Problem ist es beispielsweise ReL Aktivierungsfunktionen zu verwenden, da diese prinzipbedingt nicht vom "Vanishing Gradient Problem" betroffen sind. Des Weiteren verlangsamt dieses Problem zwar das Lernen, allerdings ist aktuelle Hardware leistungsfähig genug, um trotzdem in einer annehmbaren Zeit ein Neuronales Netzwerk anlernen zu können. Außerdem steuert eine gezielte Initialisierung der Gewichte, wie in Abschnitt 3.1.1.3 beschrieben, dem Problem zusätzlich entgegen, da dies zumindest zum Initialisierungszeitpunk dafür sorgt, dass die Ausbreitung des Eingangssignals in Richtung der tieferen Schichten nicht zu weit abgeschwächt wird. Das "Vanishing Gradient Problem" und die Lösungsansätze werden in [35] noch genauer erläutert.

3.1.2.3 Early Stopping

Eine zusätzliche Regularisierungsmaßnahme zur Verhinderung von Overfitting ist das Early Stopping[9]. Hierfür relevant ist der Fehler der Validierungsmenge. Beim Training wird nach jeder Iteration überprüft, ob sich der Fehler verringert hat. Ist dies der Fall wird ein Zähler auf 0 gesetzt. Ist der Validierungsfehler allerdings gleich geblieben oder hat sich verschlechtert wird der Zähler um 1 erhöht.

[9] Frühzeitiger Abbruch des Trainings.

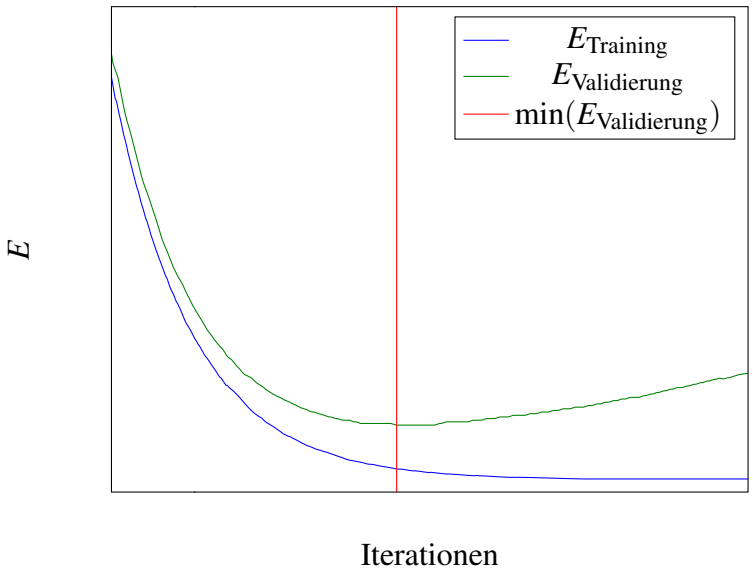


Abbildung 3.9: Idealisierte Lernkurven. Die blaue Kurve zeigt die Entwicklung des Trainingsfehlers über die Iteration, die grüne Kurve zeigt die Entwicklung des Validierungsfehlers über die Iterationen. Mit hoher Anzahl an Iterationen können die Kurven auseinander laufen; es kommt zu Overfitting. In rot markiert ist der minimale Validierungsfehler.

Erreicht der Zähler die vorher definierte Schranke i_{es} wird das Training abgebrochen und die Gewichte von dem Iterationsschitt genommen, an dem der Validierungsfehler minimal war. Dieser Iterationsschritt ist in Abbildung 3.9 rot markiert. Das Verfahren wird in [18] beschrieben.

3.1.3 Evaluation

Zur Evaluation des Netzwerkes wird eine Validierungsmenge aus den Datensätzen verwendet. Wie diese aussieht wurde in Abschnitt 2.1.2 und Abschnitt 2.1.1 beschrieben. Diese Validierungsmenge ist unabhängig von der Trainings- und Testmenge. Es kann somit auch eine Aussage über gegebenenfalls stattgefundenes Overfitting getroffen werden. Zur Messung des Fehlers während des Anlernprozesses wird die negative Log-Likelihood-Funktion L verwendet[10].

[10] $L = -\text{mean}(\log(o_{\text{output}}(x_m, t_m)) \forall m \in \text{batch})$; $\text{mean}(\cdot)$ bildet den Mittelwert über alle Elemente.

3.2 Evolutionärer Algorithmus

Der evolutionäre Algorithmus wurde so implementiert, wie es in Abschnitt 2.3.1 beschrieben wurde. Die Individuen werden durch Gene spezifiziert, welche im folgenden Abschnitt beschrieben werden. Weiter wird in den folgenden Unterabschnitten auf Details der Umsetzung sowie der Konfiguration des evolutionären Algorithmus eingegangen.

Im Allgemeinen kann man festhalten, dass in dieser Arbeit maßgeblich zwei verschiedene Eigenschaften der Individuen beziehungsweise Neuronalen Netzwerke optimiert werden. Die Anzahl an Parametern sowie der Fehler in der Klassifizierung werden minimiert.

3.2.1 Genom

Das Genom, welches die wesentlichen Erbinformationen eines Neuronalen Netzwerkes enthält, besitzt folgende Einträge:

1. Anzahl aktiver Schichten
2. Anzahl der Einheiten dichter Schichten
3. Anzahl der Einheiten der Faltungsschichten
4. Arten der Schichten
5. Dropout Wahrscheinlichkeit der Eingangsschicht
6. Filtergrößen der Faltungsschichten
7. Poolinggrößen der Faltungsschichten
8. Aktivierungsfunktionen
9. Lernrate

Der erste Punkt, Anzahl aktiver Schichten, gibt die Tiefe des Netzwerkes an. Hierbei werden Eingangs- und Ausgangsschicht nicht mitgezählt, da diese identisch für alle Individuen einer Population P sind. Des Weiteren folgt auf jede Faltungsschicht eine Maxpooling Schicht; diese Kombination wird als eine Schicht gewertet, da die Maxpooling Schicht keine eigenen Gewichte besitzt und somit keinen direkten Einfluss auf die Anzahl an Parametern hat.

Bei der Anzahl an Einheiten wird zwischen Faltungsschichten und dichter Schicht unterschieden; die gleiche Anzahl an Einheiten bei einer Faltungsschicht und einer

dichten Schicht würden auf Grund der lokalen Konnektivität der Faltungsschicht unterschiedliche Anzahl an Parametern beziehungsweise Gewichten liefern.

Der vierte Punkt, Arten der Schichten, kodiert, ob es sich bei einer Schicht um eine Faltungsschicht oder um eine Schicht mit einer dichten Struktur handelt. Handelt es sich beispielsweise um eine dichte Struktur werden entsprechend des Eintrages Anzahl der Einheiten dichter Schichten die Anzahl der Einheiten für die entsprechende Schicht gesetzt und der Punkt Anzahl der Einheiten der Faltungsschichten ignoriert.

Wie in Abschnitt 3.1.1.1.4 erwähnt, sind die Dropout Wahrscheinlichkeiten p_{drop} der inneren Schichten sowohl für den Fall einer Faltungsschicht als auch für den Fall einer dichten Struktur stets mit 50 % definiert. Nur für die Eingangsschicht ist dieser Wert abweichend und unter diesem Punkt gespeichert.

Die Filtergrößen und die Poolinggrößen der Faltungsschicht sind in zwei weiteren Punkten festgehalten. In diesen beiden Einträgen des Genoms werden die entsprechenden Dimensionen des Filterkerns respektive des Poolingkerns gespeichert.

Die Aktivierungsfunktionen werden auch Schichtenweise verwendet. Jede Schicht hat in dem Genom eine der in Abschnitt 3.1.1.2 vorgestellten Aktivierungsfunktionen kodiert.

Des Weiteren besitzt jedes Genom einen Eintrag für die Lernrate η.

Ein Genom könnte beispielsweise, wie in Tabelle 3.2 dargestellt, aussehen.

Eigenschaft	**Werte**
Anzahl aktiver Schichten	3
Anzahl der Einheiten dichter Schichten	200, 300, 100, 50
Anzahl der Einheiten der Faltungsschichten	1000, 700, 30, 10
Arten der Schichten	Conv, Dense, Dense, Dense
Dropout Wahrscheinlichkeit der Eingangsschicht	0,75
Filtergrößen der Faltungsschichten	$3\times3, 2\times1, 2\times2, 1\times1$
Poolinggrößen der Faltungsschichten	$2\times2, 1\times1, 1\times1, 1\times2$
Aktivierungsfunktionen	Tanh, Sigmoid, ReL, LReL
Lernrate	0,05

Tabelle 3.2: Beispiel eines Genoms mit 3 aktiven Schichten und 4 maximal möglichen aktiven Schichten.

Was bei diesen Genom auffällt ist, dass es obwohl nur drei Schichten aktiv sind vier Schichten innerhalb des Genoms beschrieben werden. Durch eine Mutation oder Fortpflanzung kann es passieren, dass vor einigen Iterationen des evolutionären Algorithmus ein Genom vorhanden war, das vier aktive Schichten besaß, welches bei der Erstellung des in Tabelle 3.2 beteiligt war. Somit sind die Informationen über die vierte, in diesem Genom nicht aktive, Schicht nicht verloren sondern werden mit vererbt. Die maximale Anzahl an aktiven Schichten bestimmt dabei, wie viele Werte ein Genom halten kann, um ein Neuronales Netzwerk definieren zu können (vgl. Abschnitt 3.2.3).

3.2.2 Details

Dieser Abschnitt wird auf die Details und Einschränkungen der Umsetzung eingehen. Zur Berechnung des Fitnesswertes wird ein Validierungsdatensatz benötigt. Um das Training beobachten zu können und auch Rückschlüsse auf ein eventuell auftretendes Overfitting ziehen zu können, wurde ein zweiter Validierungsdatensatz eingeführt. Im Rahmen des evolutionären Algorithmus existieren somit vier Untermengen: Eine Trainings- und Testmenge, die während des Anlernens der Neuronalen Netzwerke verwendet wird, eine Validierungsmenge, die verwendet wird, um die Fitness der Individuen berechnen zu können sowie eine weitere Validierungsmenge, um die Performance der Individuen bewerten zu können. Die Validierungsmenge zur Bestimmung der Fitnesswerte, wird im Folgenden Fitnessmenge genannt. Die Fitnessmenge entspricht 20 % der Daten, die ohne der Validierungsmenge sonst zum Training und Testen verwendet wurde.

Eine Einschränkung, die der evolutionäre Algorithmus in der Umsetzung besitzt, ist, dass auf eine dichte Schicht keine Faltungsschicht folgen kann. Diese Einschränkung ist dem Grund geschuldet, dass bei dem Übergang von einer Faltungsschicht auf eine dichte Schicht sämtliche Ausgaben der Faltungsschicht (typischerweise zweidimensionale Bilder) zu einem Vektor transformiert werden. Die Ausgabe der dichten Schicht ist erneut ein Vektor. Sollte daraufhin wieder eine Faltungsschicht mit zweidimensionalen Filterkern folgen, müsste der Vektor in eine Matrix transformiert werden. Diese Transformation wäre willkürlich, da die Positionsinformationen der Pixel beim durchlaufen der dichten Schicht auf Grund der dichten Vernetzung der Eingänge verloren gehen. Eine weitere Besonderheit der Faltungsschicht ist, dass nach einer Faltungsschicht innerhalb des evolutionären Algorithmus stets ein Maxpooling stattfindet. Dieses Maxpooling kann sich allerdings so entwickeln, dass es wirkungslos ist (Maxpooling mit $1\,\text{px} \times 1\,\text{px}$ Blockgröße).

Des Weiteren existieren zwischen den einzelnen Schichten jeweils eine Dropoutschicht mit $p_{\text{drop}} = 0{,}5$, um Overfitting zu verhindern. Diese Dropoutschichten werden zur Anzahl an Schichten nicht hinzugezählt.

3.2.3 Konfiguration

Wie in Kapitel 2.3 schon beschrieben existieren mehrere Hyperparameter, die in Tabelle 3.3 kurz zusammengefasst sind.

Die Größe der Population und die Mutationswahrscheinlichkeit haben eine direkte Auswirkung auf die Vielfalt der Individuen. Die Auswirkungen dieser beiden Hyperparameter wurden bereits in Abschnitt 2.3.2 beschrieben. Hinzu kommt der Hyperparameter s_p. Dieser Skalierungsfaktor wird zur Berechnung des Fitnesswertes f_j der einzelnen

Hyperparameter	Kurzbeschreibung
$\|P\|$	Größe der Population P.
p_m	Mutationswahrscheinlichkeit während der Reproduktion.
s_p	Skalierungsfaktor zur Bestimmung des Fitnesswertes.

Tabelle 3.3: Zusammenfassung der untersuchten Hyperparameter des evolutionären Algorithmus.

Individuen verwendet. Der Fitnesswert berechnet sich dabei wie folgt:

$$f_j = s_p(|P| - r_{\text{error},j}) + (1 - s_p)(|P| - r_{\text{size},j}) \tag{3.26}$$

Dabei wird zwischen $r_{\text{error},j}$, dem Rang nach Fehler sortiert, und $r_{\text{size},j}$, dem Rang nach der Anzahl an Parametern des Neuronalen Netzwerkes, gewichtet. Es gilt: je weniger Parameter ein Neuronales Netzwerk aufzeigt, desto besser ist der Größenrang und je kleiner der Fehler eines Neuronalen Netzwerkes ist desto besser ist der Fehlerrang. Der Skalierungsfaktor s_p kann also so gewählt werden, dass entweder das Hauptaugenmerk auf der Größe des Netzwerkes und damit auf der Schnelligkeit der Berechnungen liegt oder auf dem Fehler des Netzwerkes und damit auf der Genauigkeit der Klassifizierung des Netzwerkes. Je größer s_p ist, desto entscheidender ist der Fehlerrang des Individuums; je kleiner s_p ist, desto entscheidender ist der Rang im Vergleich der Anzahl an Parametern der Individuen. Anhand des Fitnesswertes werden die Individuen sortiert und für die Reproduktion, wie in Abschnitt 2.3.4 beschrieben, ausgewählt.

3.3 Detektion

Die Detektion der DataMatrix-Codes innerhalb von Bildern wird ähnlich der Detektion von QR-Codes in [8] realisiert. Die einzelnen Schritte des Detektionsalgorithmus sind in Abbildung 3.10 visualisiert.

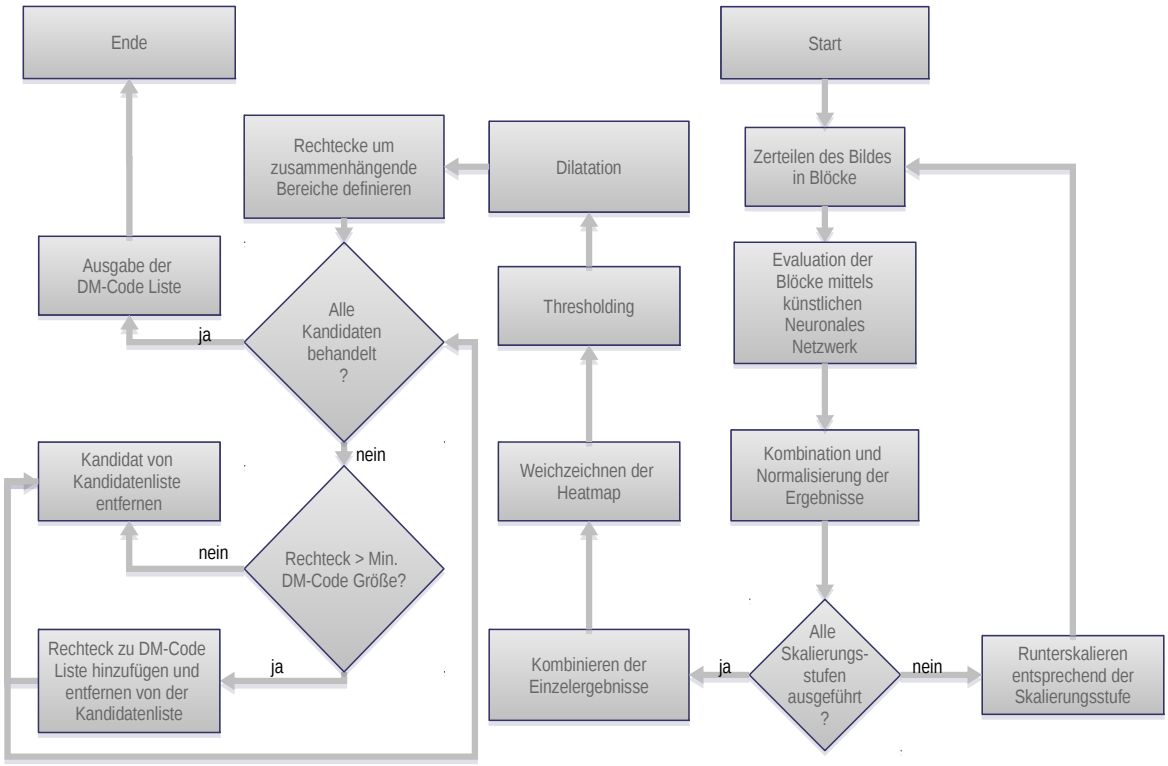


Abbildung 3.10: Ablaufdiagramm der Detektion von DataMatrix-Codes mittels künstlichen Neuronalen Netzwerks.

Das Bild wird anfangs in Blöcke mit 32 px × 32 px zerteilt, welche daraufhin mittels Neuronalem Netzwerk ausgewertet werden. Für jeden Block gibt das Neuronale Netzwerk eine Wahrscheinlichkeit wieder, mit welcher dieser Block Teil eines DataMatrix-Codes ist. In [8] wird gezeigt, dass die Blockgröße bei einem ähnlichen Verfahren für QR-Codes ein unkritischer Parameter ist. Von 30 px × 30 px bis 90 px × 90 px wurden gute Ergebnisse erzielt, wobei die zu suchenden QR-Codes ca. 180 px × 180 px groß waren. Bei dieser Arbeit sind die DataMatrix-Codes der Datensätze im Mittel ca. 90 px × 90 px groß[11]. Somit ist die Fenstergröße von 32 px × 32 px in einem ähnlichen Verhältnis zur erwarteten Codegröße gewählt, wie in [8]. Bei der Zerteilung des Bildes kann eine Schrittweite gewählt werden. Entspricht die Schrittweite der Fenstergröße werden die Blöcke ohne Überlappung erstellt. Wird die Schrittweite hingegen kleiner als

[11] Standardabweichung von ca. 43 px in x- und y-Richtung.

das Fenster gewählt kommt es zur Überlappung. Die Auswirkung verschiedener Schrittweiten wird noch im Kapitel 4 behandelt. Die Teilergebnisse werden zu einem Gesamtbild zusammengefasst; jeder Pixel dieses Gesamtbildes spiegelt die Wahrscheinlichkeit des Vorhandenseins eines DataMatrix-Codes im Originalbild wider. Dieses Wahrscheinlichkeitsbild in Falschfarbendarstellung ist beispielhaft in Abbildung 3.11 zu sehen.

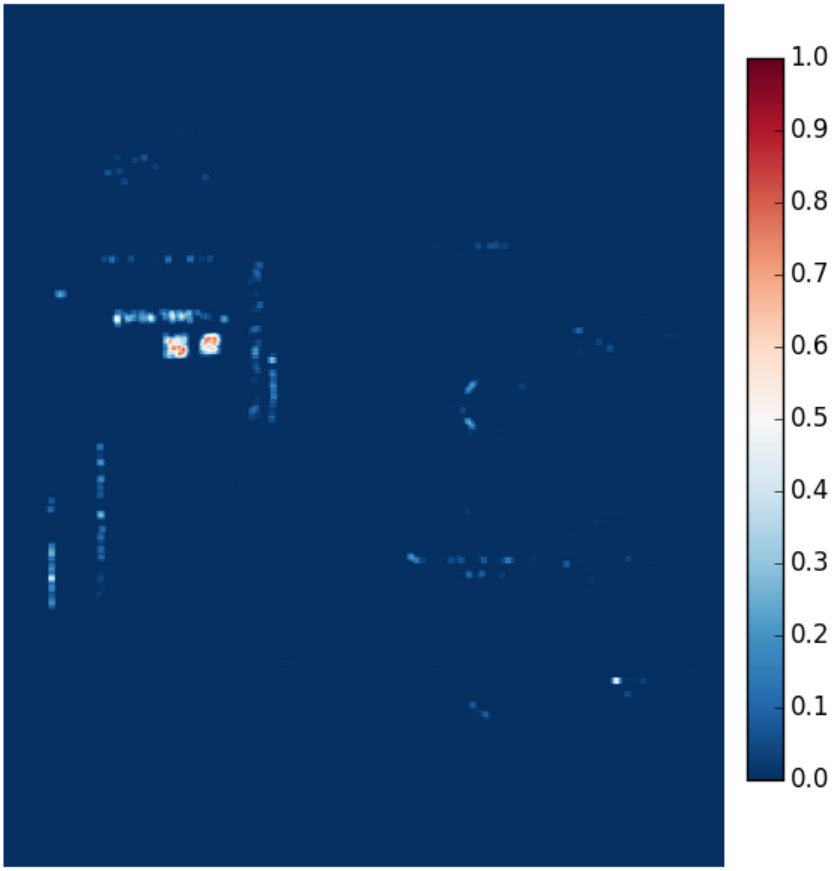


Abbildung 3.11: Heatmap des in Abbildung 2.2 Beispielbildes. Fenstergröße 32 px × 32 px, Schrittweite 8 px, 8 px. Die Skala der Heatmap ist für alle weiteren Heatmaps identisch.

Um eine gewisse Invarianz bezüglich der Größe der DataMatrix-Codes zu bekommen, kann das Originalbild herunter skaliert und erneut evaluiert werden. Dieser gesamte Vorgang wird je nach gewünschten Skalierungsstufen wiederholt. Dieses Vorgehen wird unter anderem in [5] praktiziert, um eine Invarianz bezüglich der zu detektierenden Maschinencodes zu erhalten.

Die dabei entstehenden Heatmaps werden nun zu einer Gesamtheatmap kombiniert. Dafür werden die Heatmaps alle auf die größte Skalierungsstufe hoch skaliert.

Danach werden die Wahrscheinlichkeitswerte der Zwischenergebnisse addiert und normiert. Dies ist in Abbildung 3.12 visualisiert.

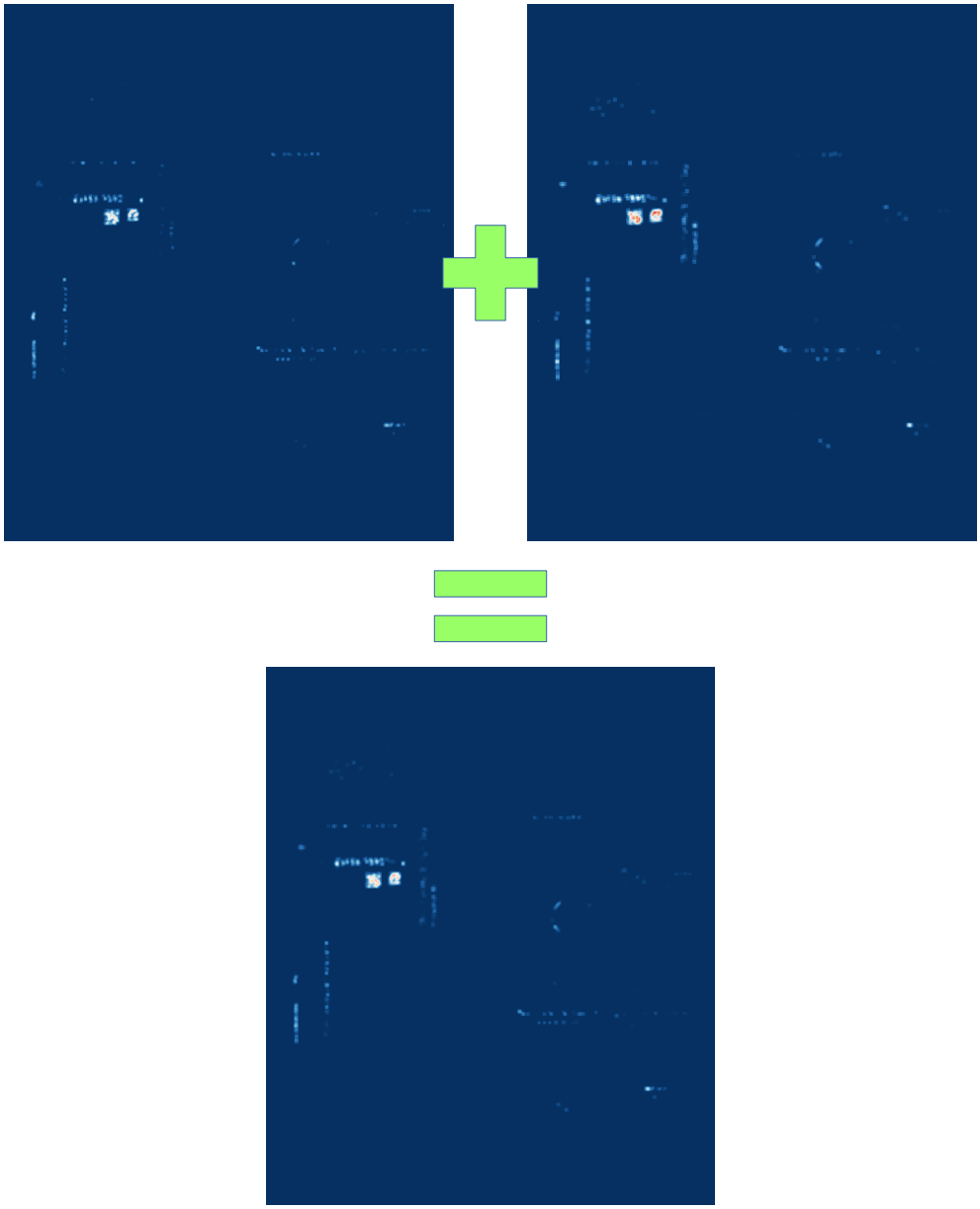


Abbildung 3.12: Zusammenfügen einzelner Heatmaps zu einem Gesamtbild. Fenstergröße 32 px $\times$ 32 px, Schrittweite 8 px, 8 px.

Daraufhin wird diese Gesamtheatmap noch weich gezeichnet[12] und über ein Threshold[13] zu ein Binärbild gewandelt. Dieses Binärbild ist in Abbildung 3.13 zu sehen.

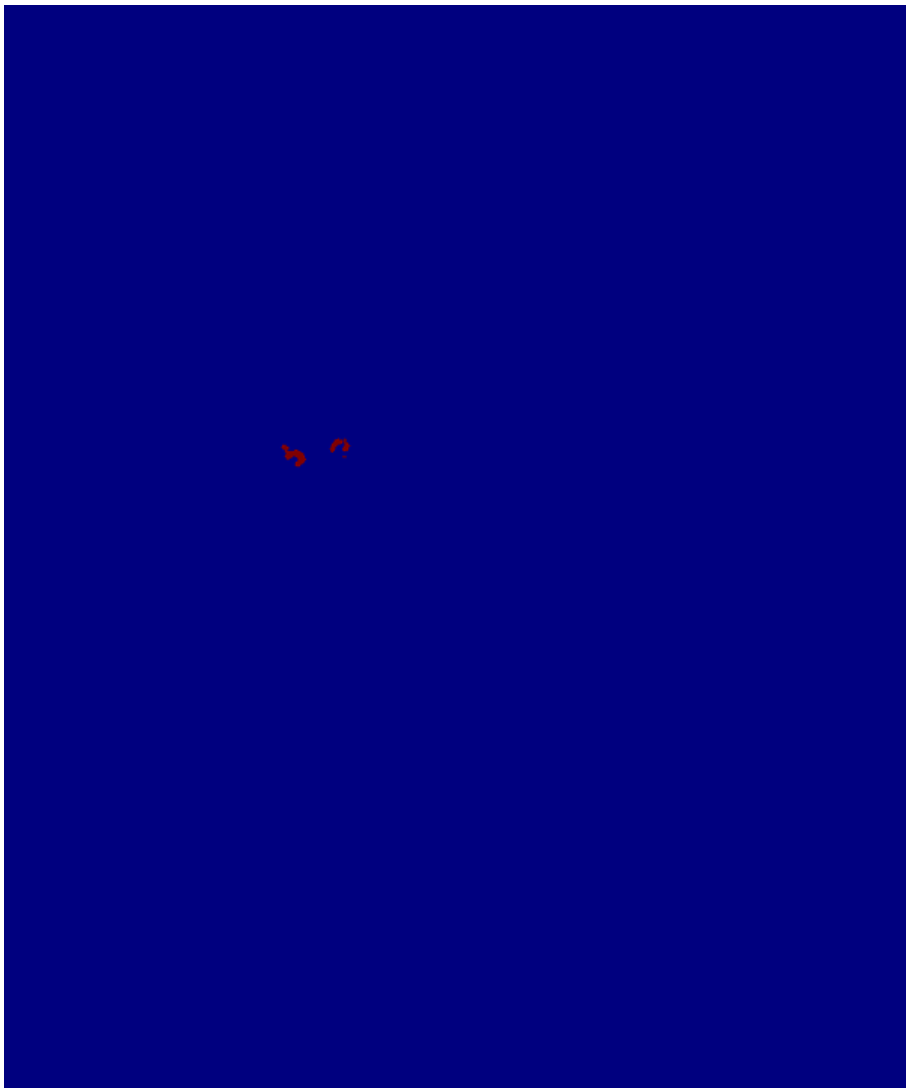


Abbildung 3.13: Binäre Heatmap. Fenstergröße 32 px × 32 px, Schrittweite 8 px, 8 px, Weichzeichnungsradius 9 px, Schwellwert 0,5. Alle Werte größer oder gleich dem Schwellwert sind rot dargestellt.

Nun werden Rechtecke definiert, die die verbleibenden Bereiche umfassen. Diese Rechtecke werden alle in einer Kandidatenliste zusammengefasst. Für alle Rechtecke

[12] Lokaler Mittelwert in einer Umgebung von r px × r px, wobei r als Weichzeichnungsradius definiert wird.

[13] Schwellwert

der Kandidatenliste wird geprüft, ob diese größer als eine minimale DataMatrix-Code Größe sind. Ist dies der Fall, wird das Rechteck zur DataMatrix-Code Liste hinzugefügt. Ist der Bereich zu klein, wird dieser verworfen. Laut [2] besteht der kleinste DataMatrix-Code aus 8×8 Informationsmodulen. Hinzu kommt noch das Suchmuster, bestehend aus einem umlaufenden Rahmen, wodurch der kleinst mögliche DataMatrix-Code aus 10×10 Modulen besteht (unter Vernachlässigung der Ruhezone um den DataMatrix-Code). Des Weiteren wird in [2] spezifiziert, dass ein Modul mindestens 0,254 mm (0,0100 inches) misst[14]. Bei einem Scanner mit 240 DPI folgt eine Mindestgröße von 24 px $\times$ 24 px. Dies ist der Wert, welcher in dieser Arbeit als minimale DataMatrix-Code Größe verwendet wird. Dieser Wert deckt sich auch mit den DataMatrix-Codes aus den Datensätzen; die kleinsten DataMatrix-Codes haben in jeder Dimension bis auf drei Ausnahmen mehr als 31 px; die Ausnahmen sind drei abgeschnittene DataMatrix-Codes. Wurden alle Kandidaten der Kandidatenliste behandelt werden die DataMatrix-Code Regionen ausgegeben und der Algorithmus ist damit abgeschlossen.

3.4 Python/C++-Schnittstelle

Zusätzlich zum Python Programm, welches in vorherigem Abschnitt vorgestellt wurde, wurde eine C++ Anbindung des Klassifizierers implementiert. Hierzu kam die Python/C-Schnittstelle zum Einsatz, die Python besitzt.

Die Python/C-Schnittstelle wird in [36] beschrieben. Des Weiteren kommt die Numpy C-Schnittstelle zum Einsatz (siehe [37]), um die eingelesenen Bilddaten in Python verwenden zu können. Unter Verwendung der Numpy C-Schnittstelle kann der selbe Speicherbereich in Python verwendet werden, der auf der C++ Seite allokiert wurde. Gerade bei größeren Bilddaten ist es sinnvoll keinen zusätzlichen Speicher für die Python Seite allokieren zu müssen.

Für die Implementierung des C++ Interfaces wurde die in Abbildung 3.14 dargestellte Klassenhierarchie verwendet.

[14] Typischerweise 0,3 mm (0,0118 inches)

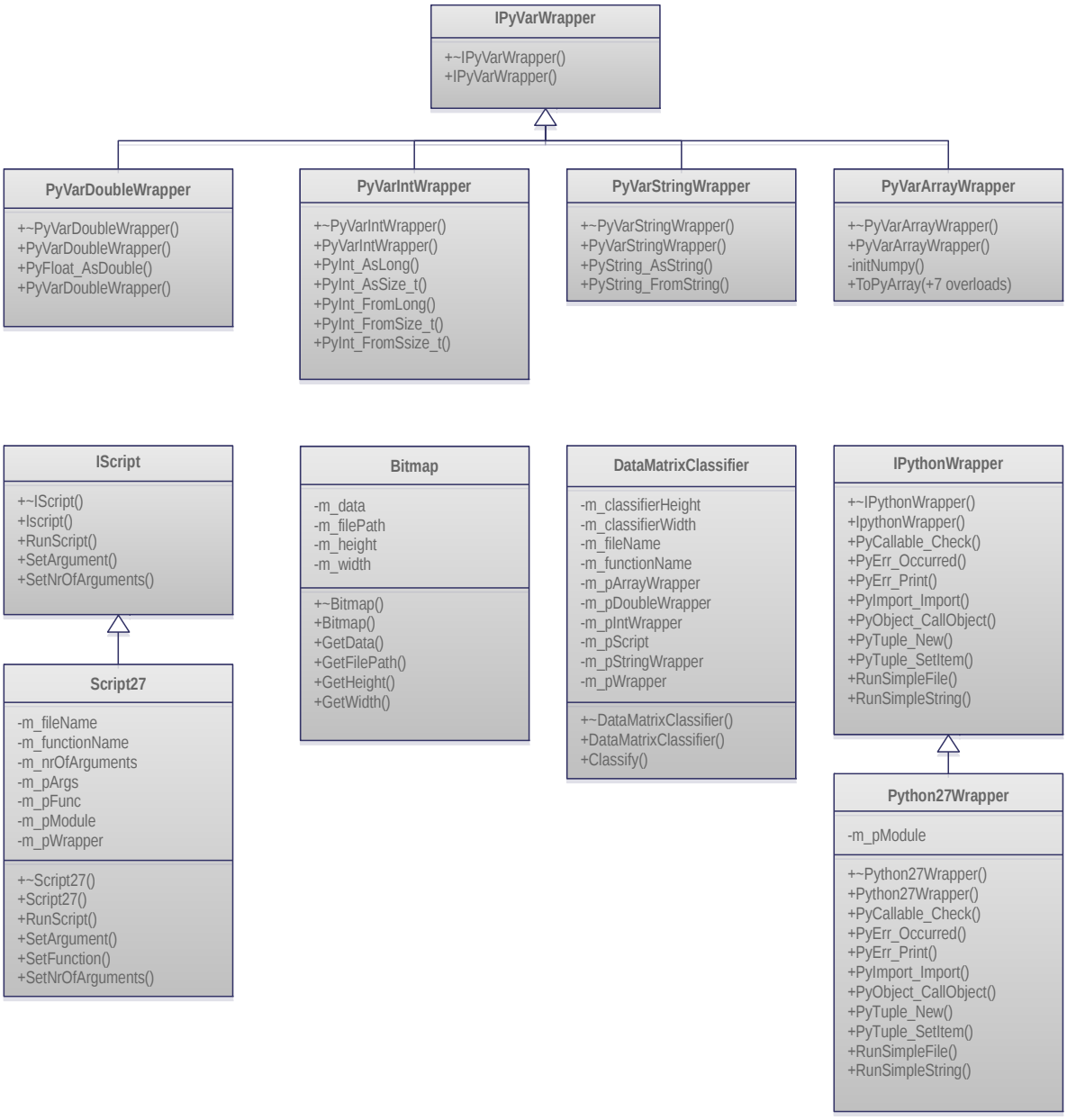


Abbildung 3.14: Klassenhierachie des C++ Interfaces.

Dies hat den Vorteil, dass bei einem Versionswechsel von Python (beispielsweise von der verwendeten Version 2.7.9.4 auf eine 3.X Version) viele Codefragmente wiederverwendet werden können. Die Funktionalität ist in eigenen Klassen gekapselt, die

leicht austauschbar sind. Des Weiteren wurde die Funktionalität erweitert, um beliebige Python Skripte sowie Python Befehle direkt aus C++ ausführen zu können. Die Funktionalitäten werden in dem mitgelieferten Beispielprojekt demonstriert.

Ziel dieser Schnittstelle ist es eine Anbindung an C++ bieten zu können, um Teile des Detektionsalgorithmus in künftigen Arbeiten performant umsetzen zu können.

Kapitel 4

Auswertungen und Ergebnisse

Dieses Kapitel ist in vier Teile aufgeteilt. Anfangs wird der Kundendatensatz bezüglich des Rauschens untersucht. Der zweite Teil beschäftigt sich mit dem evolutionären Algorithmus zur Netztopologie-Bestimmung, der dritte Teile beschäftigt sich mit dem finalen Neuronalen Netz und im vierten Teil ist die Auswertung der Gesamtbilder zu finden.

4.1 Rauschanalyse des Kundendatensatzes

Der Kundendatensatz wurde bezüglich des Rauschens untersucht. Hierfür wurden $32\,\text{px} \times 32\,\text{px}$ große Quadrate aus jedem Bild des Kundendatensatz ausgewählt, deren Verteilung der Amplituden der Graustufenbilder betrachtet wurde. Hierfür wurden die Amplituden für jedes Quadrat vom Mittelwert befreit und dann die entsprechende Abweichung vom Mittelwert betrachtet. Dies ist in Abbildung 4.1 zu sehen.

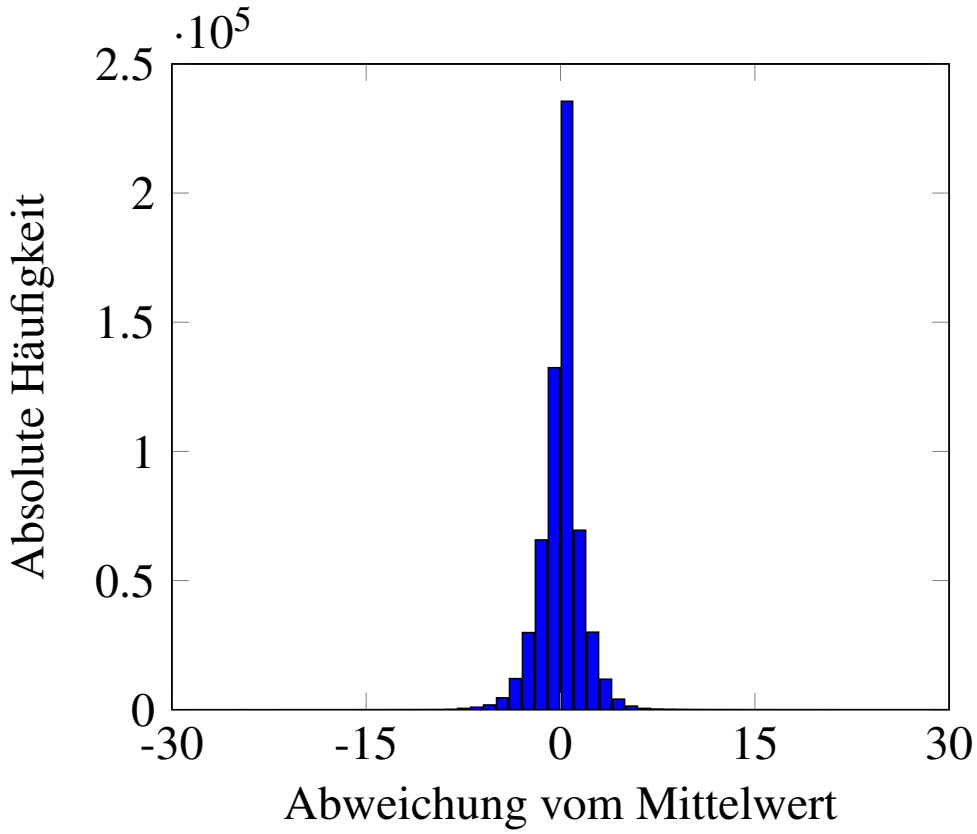


Abbildung 4.1: Verteilung der Grauwertabweichung vom Mittelwert von Flächen ohne Schriften und Grafiken der gescannten Druckerzeugnisse. Es wurde für jedes Element aus dem Datensatz ein $32\,\text{px} \times 32\,\text{px}$ großes Quadrat ausgewählt, das in einer Fläche liegt, die keine Schriften oder andere Grafiken enthält, woraufhin der mittlere Grauwert dieser Fläche berechnet wurde und die Abweichung von diesem in dieser Abbildung dargestellt wurde. Die dargestellten Einteilungen zwischen a und b sind von der Form $[a;b)$.

Es ist zu sehen, dass ein Großteil der Daten (ca. 73 %) maximal 2 Graustufen vom Mittelwert abweichen. Des Weiteren ist anzumerken, dass bei dem Inhalt der Quadrate unter anderem Papier und Karton als Untergrund dienen. Somit bringt das eingescannte Material teilweise eine Grundvarianz mit, welche in dieser Betrachtung nicht vom Rauschen unterscheidbar ist. Man kann allerdings festhalten, dass die Kundendatensätze ein geringes Maß an Varianz in den Flächen ohne grafischen Elementen aufweisen und somit von einem geringen Rauschanteil ausgegangen werden darf.

4.2 Evolutionärer Algorithmus

Dieser Abschnitt wird sich mit der Evaluierung des evolutionären Algorithmus beschäftigen. Erst wird in 4.2.1 der evolutionäre Algorithmus mit einer Auswahl an Hyperparametern durchlaufen, um die Auswirkungen der einzelnen Parameter zu bestimmen. Dann wird ein Hyperparametersatz festgelegt, mit welchem die finale Netztopologie bestimmt wird. Die entsprechenden Ergebnisse sind im Abschnitt 4.2.2 zu finden.

4.2.1 Festlegung der Hyperparameter

Zur Bestimmung der Optimalen Netztopologie wurde der evolutionäre Algorithmus mit verschiedenen Hyperparametersätzen ausgeführt. Die folgenden Ergebnisse und die entsprechenden Rückschlüsse basieren auf jeweils einem Lauf pro Hyperparametersatz. Die Ergebnisse und Rückschlüsse sind also nicht im Allgemeinen gültig. Des Weiteren wurde der evolutionäre Algorithmus nur mit dem synthetischen Datensatz ausgeführt. Die finale Netztopologie wurde daraufhin allerdings sowohl mit dem synthetischen als auch mit dem nicht synthetischen Datensatz getestet. Die Anzahl an Iterationen des evolutionären Algorithmus beträgt 200. Die Anzahl an Iterationen ist ein kritischer Faktor für das Ergebnis des evolutionären Algorithmus. Wird diese Anzahl zu klein gewählt haben die Individuen, je nach Populationsgröße $|P|$, eine geringe Wahrscheinlichkeit aus mehrfacher Reproduktion zu entstehen. In Abschnitt 4.2.2.1 sieht man zum Beispiel, dass für $|P| = 30$ die Anzahl von 200 Iterationen zu gering gewählt wurde, da das fitteste Individuum aus der Initialpopulation stammt.

Die Hyperparameter, die in Abschnitt 3.2.3 eingeführt und erklärt wurden, wurden dabei mit den Konfigurationen aus Tabelle 4.1 evaluiert.

Durchlauf (Run)	**Populationsgröße** $\|P\|$	**Mutations WK** p_m	**Fitnessskalierung** s_p
1	10	0,1	0,6
2	20	0,1	0,6
3	30	0,1	0,6
4	10	0,2	0,6
5	10	0,3	0,6
6	10	0,1	0,75
7	10	0,1	0,9

Tabelle 4.1: Untersuchte Hyperparametersätze; es wird jeweils nur ein Parameter geändert, um die Auswirkung dieses Parameters analysieren zu können.

Wie man in Tabelle 4.1 sehen kann, wurde stets nur ein Hyperparameter von Lauf zu Lauf geändert, um die entsprechenden Auswirkungen sehen zu können. Die Popula-

tionsgröße $|P|$ befindet sich im Bereich von 10 bis 30 Individuen. Dieser Bereich wurde ausgewählt, um die Bedeutung der Vielfalt der Initialpopulation P_0 zu untersuchen. Bei einer Anzahl von 200 Iterationen des evolutionären Algorithmus sollte dieser Parameter nicht zu groß gewählt werden, da es sonst unwahrscheinlicher ist, dass Reproduktion Generationen von Individuen entstehen lässt, deren Stammbaum eine gewisse Tiefe mit sich bringt. Würden bei 200 Iterationen die Anzahl an Individuen $|P|$ beispielsweise $|P| = 1000$ betragen, wäre die Wahrscheinlichkeit gering, dass zwei Individuen, die selbst aus einer Reproduktion stammen (nicht aus der Initialpopulation P_0), sich erneut reproduzieren.

Die Mutationswahrscheinlichkeit p_m wird im Bereich von 0,1 bis 0,3 evaluiert. Dieser Wertebereich wurde gewählt, um den Einfluss der Reproduktion hinreichend groß zu belassen. Des Weiteren sind das typische Werte für die Mutationswahrscheinlichkeit, wie sie zum Beispiel in [38] zu finden sind.

Die Fitnessskalierung s_p wird im Bereich von 0,6 bis 0,9 untersucht, also stets mit einer größeren Bedeutung der Klassifizierungsleistung der Neuronalen Netzwerke als auf der Größe und damit auf der Rechengeschwindigkeit; je kleiner ein Netzwerk ist, desto kleiner sind die Gewichtungsmatrizen W der Schichten und desto weniger Rechenoperationen sind notwendig, um eine Klassifizierung mit dem Neuronalen Netzwerk durchzuführen.

Die weiteren Parameter für die einzelnen Individuen des Neuronalen Netzwerkes sind in Tabelle 4.2 zu sehen.

Eigenschaft	**Wert**
Anzahl an Schichten	$[2;5]$
Mögliche Schichttypen	{dichte Schicht, Faltungsschicht}
Anzahl an Einheiten pro dichter Schicht	$[500;3000]$
Anzahl an Ausgängen pro Faltungsschicht	$[10000;20000]$
Faltungsschicht Filtergröße	$[1;3]$
Faltungsschicht Poolinggröße	$[1;2]$
p_{drop} Eingangsschicht	$[0;0,5]$
Aktivierungsfunktionen	alle in Abschnitt 3.1.1.2 aufgeführten
Lernrate η	$[0,002;0,2]$
Maximale Anzahl an Lerniterationen	500
Early Stopping Schranke i_{es}	25

Tabelle 4.2: Konfiguration der Neuronalen Netzwerke für den evolutionären Algorithmus.

Die Neuronalen Netzwerke sollen also zwischen 2 und 5 Schichten haben; Dropout-,

Eingangs-, Ausgangs- und Maxpoolingschichten werden nicht hinzugezählt. Es kommen sowohl dichte Schichten als auch Faltungsschichten zum Einsatz. Die dichten Schichten sollen zwischen 500 und 3000 Einheiten besitzen. Die Anzahl an Parametern werden über die Anzahl an Einheiten nur indirekt beeinflusst, da die Anzahl an Einheiten nur einen Einfluss auf die Anzahl an ausgehenden Verbindungen dieser Schicht hat aber nicht an der Anzahl der eingehenden Verbindungen in diese Schicht.

Die Anzahl an Einheiten pro Faltungsschicht bestimmt im Zusammenhang mit den Filtergrößen die Topologie der Faltungsschicht. In diesem Fall können Filterkerne $K^{u\times v}$ Größen von $u \in \{1,2,3\}$ und $v \in \{1,2,3\}$ annehmen. Es muss nicht gelten $u = v$.

Um zu zeigen, wie die Topologie einer Faltungsschicht aus diesen beiden Parametern entsteht folgt ein Beispiel:

Sei $u = v = 3$, die Anzahl an Ausgängen pro Faltungsschicht 10000, die vorherige Schicht die Eingangsschicht mit $32\,\text{px} \times 32\,\text{px}$ Bildausschnitten und eine Poolinggröße von $2\,\text{px} \times 2\,\text{px}$. Die Anzahl an Filterkernen n_f beträgt

$$\mathrm{n}_f = \left\lfloor \frac{10000}{(32-3+1)*(32-3+1)} + 0{,}5 \right\rfloor = 11 \tag{4.1}$$

Sollte n_f kleiner als 1 sein, wird $\mathrm{n}_f = 1$ gesetzt. Die Anzahl an Einheiten wird also somit über die Größe der Ausgabeschicht ($30\,\text{px} \times 30\,\text{px}$) und der angegebenen Anzahl an Einheiten (10000) berechnet. Danach findet ein Maxpooling mit der Poolinggröße von $2\,\text{px} \times 2\,\text{px}$ statt; die Ausgabe der Faltungsschicht/Maxpoolingschicht-Kombination sind somit elf $15\,\text{px} \times 15\,\text{px}$ große, mit den Filterkernen $K_{\mathrm{n}_f}^{3\times 3}, \mathrm{n}_f \in [1;11]$ gefilterte Bilddaten. Des Weiteren wird beim Maxpooling darauf geachtet, dass diese Operation durchführbar ist. Sollte beispielsweise ein Maxpooling mit $2\,\text{px} \times 2\,\text{px}$ auf Bilddaten mit ungerader Anzahl an Pixeln stattfinden, wird das Maxpooling um einen verringert, damit diese Operation ausgeführt werden kann. Daraus folgt, dass ein Maxpooling mit $1\,\text{px} \times 1\,\text{px}$ stattfinden würde, was kein Maxpooling entspricht.

Die Dropoutwahrscheinlichkeit p_{drop} der Eingangsschicht wird im Bereich von $p_{\text{dropout}} \in [0;0{,}5]$ liegen, so wie es in Abschnitt 3.1.1.1.4 beschrieben ist. Die Aktivierungsfunktionen aus Abschnitt 3.1.1.2 werden alle für den evolutionären Algorithmus verwendet. Die Ausgangsschicht besitzt dabei die Softmax Aktivierungsfunktion.

Die Lernrate η liegt im Bereich $\eta \in [0{,}002;0{,}2]$. Dieser Bereich umfasst die in [39] genannten initialen Lernrate. Die Wahl der maximalen Iterationen sowie der Early Stopping Schranke i_{es} ist in Abschnitt 3.1.2.1 begründet.

4.2.2 Bestimmung der finalen Netztopologie

Dieser Abschnitt geht auf die Ergebnisse des evolutionären Algorithmus ein. Wie in Abschnitt 4.2.1 erwähnt sind diese Ergebnisse nicht allgemeingültig, da zu viele nichtdeterministische Vorgänge beteiligt sind, wie zum Beispiel die Auswahl der Individuen

zur Reproduktion, die Wahl der Eigenschaften der Individuen bei einer Reproduktion, die mutiert werden sollen sowie die Initialisierung der Gewichte der einzelnen Neuronalen Netzwerke und der Population.

Die in Tabelle 4.1 Hyperparametersätze werden in drei Gruppen unterteilt. Die erste Gruppe (Hyperparametergruppe 1) besteht aus den Durchläufen 1, 2 und 3 und wird verwendet, um die Auswirkung der Populationsgröße $|P|$ auf das Ergebnis zu untersuchen.

Hyperparametergruppe 2 besteht aus den Durchläufen 1, 4 und 5 und dient zur Untersuchung der Mutationswahrscheinlichkeit p_m.

Die dritte Hyperparametergruppe beinhaltet die Durchläufe 1, 6 und 7. Mit diesen Hyperparametersätzen wird die Fitnessskalierung s_p untersucht.

In den folgenden Unterabschnitten sind folgende Grafiken zu sehen:

1. Die Fehlerentwicklung der Neuronalen Netzwerke der Validierungsmenge
2. Die Fehlerentwicklung der Neuronalen Netzwerke der Fitnessmenge
3. Die Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke
4. Die Anzahl an verworfenen Individuen pro Iteration
5. Die Verteilung der Auswahl der Individuen zur Reproduktion

Für die Punkte 1. bis 3. werden jeweils die mittleren, minimalen und maximalen Werte pro Iteration angegeben[1] sowie die Werte des fittesten Individuums[2].

4.2.2.1 Hyperparametergruppe 1

Mit der Hyperparametergruppe 1 werden zunächst die Auswirkungen der Populationsgröße auf die Entwicklung der Neuronalen Netzwerke untersucht. Hierfür wird zu Anfang die Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke in Abbildung A.3 betrachtet.

Zu beobachten ist, dass sich die mittlere Größe nach 200 Iterationen in etwa gleich entwickelt (vgl. Tabelle A.1). Die Größen der jeweils fittesten Netzwerke sind allerdings unterschiedlich. Das kleinste fitteste Netzwerk entsteht bei Run 3, also der Konfiguration mit $|P| = 30$. Allerdings ist auch zu sehen, dass während der Entwicklung beim ersten und zweiten Hyperparametersatz bis kurz vor Ende der 200 Iterationen ein kleineres Netzwerk den besten Fitnesswert liefert. Bei Run 3 hingegen stammt das fitteste Individuum aus der initialen Population P_0 und nicht aus der Evolution.

[1] Markierung mittels "mean", "min" und "max" im Diagramm.

[2] Markierung mittels "best" im Diagramm.

In Abbildung A.4 ist die Entwicklung der MSE[3] der Fitness- und Validierungsmenge zu sehen. Der MSE sowohl in der Validierungs- als auch in der Fitnessmenge sind für das jeweils fitteste Individuum sehr ähnlich. Stark unterscheiden sich allerdings die mittleren Fehler der unterschiedlichen Hyperparametersätze. Sowohl in der Fitnessmenge als auch in der Validierungsmenge fällt auf, dass mit zunehmender Größe $|P|$ der Populationen der mittlere Fehler größer ist. Allerdings ist auch zu sehen, dass die Tendenz bei allen Hyperparametersätzen in Richtung kleinerer Netzwerke geht. Würde man folglich mehr als 200 Iterationen zulassen würden die entsprechenden mittleren Fehler wahrscheinlich auch bei den größeren Populationsmengen $|P| > 10$ sinken.

In Abbildung A.5 sind die Anzahl an verworfenen Individuen am Stück zu sehen. Wurde ein Individuum in die Population aufgenommen, wird der Zähler für die Anzahl an verworfenen Individuen am Stück erneut zurückgesetzt. Anhand einer solchen Grafik kann man eine Art Konvergenz des evolutionären Algorithmus erahnen. Je mehr Individuen am Stück verworfen werden, desto weniger der trainierten Individuen werden in die Population aufgenommen. Je weniger neue Individuen aufgenommen werden, desto potentiell besser sind die bereits vorhandenen; eine Verbesserung der Individuen findet nur noch seltener statt.

Im Lauf des ersten Hyperparametersatzes fällt auf, dass zu fortschreitender Iteration die Anzahl an am Stück verworfener Individuen steigt. Ein solches Verhalten ist in den anderen beiden Hyperparametersätzen nicht zu beobachten. Besonders beim zweiten Hyperparametersatz fällt auf, dass recht regelmäßig, auch bis zur letzten Iteration, eine Verbesserung der Populationsmenge stattfindet.

4.2.2.2 Hyperparametergruppe 2

Mit der Hyperparametergruppe 2 werden die Auswirkungen der Mutationswahrscheinlichkeit p_m evaluiert.

Die Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke ist in Abbildung A.6 visualisiert.

Die Anzahl an Parametern der jeweiligen Netztopologien variiert mit unterschiedlichen Mutationswahrscheinlichkeiten p_m. Die mittlere Parameteranzahl ist beim ersten Hyperparametersatz am kleinsten und beim vierten am größten. Die geringste Parameteranzahl des fittesten Individuums ist hingegen beim fünften Hyperparametersatz zu finden; am schlechtesten schneidet der vierte Satz an Hyperparametern ab (siehe Tabelle A.1).

Bei der Fehlerentwicklung ist sowohl bei der Validierungs- als auch bei der Fitnessmenge keine große Varianz zu finden. Die Werte liegen alle nah beisammen. Von der Entwicklung während der Ausführung der Iterationen (siehe Abbildung A.7) entwickelt sich der fünfte Hyperparametersatz am besten; er erreicht einen Fehlerbereich von unter

[3] **m**ean **s**quared **e**rror - Mittlerer quadratischer Fehler.

10^{-2} sowohl im Mittel als auch im schlechtesten Fehler, im Vergleich der in diesem Abschnitt untersuchten Hyperparametersätze, am schnellsten.

In Abbildung A.8 ist die Verteilung der Anzahl an verworfenen Individuen am Stück zu finden. Die untersuchten Hyperparametersätze verhalten sich ähnlich. Anfangs finden sehr viele Verbesserungen statt, insbesondere beim fünften Parametersatz; gegen Ende der 200 Iterationen immer weniger.

4.2.2.3 Hyperparametergruppe 3

Mit der Hyperparametergruppe 3 werden die Auswirkungen der Fitnessskalierung s_p evaluiert. Diese ist notwendig, um zu unterscheiden, wie stark die Gewichtung der Optimierung der Anzahl an Parametern beziehungsweise der Klassifikationsperformance ist.

In Abbildung A.9 ist die Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke dargestellt.

Wie zu erwarten war wächst bei steigender Fitnessskalierung s_p die Anzahl an Parametern sowohl im Mittel als auch beim fittesten Individuum (vgl. Tabelle A.1). Betrachtet man die Entwicklung der Anzahl an Parametern so sieht man, dass bei den sechsten und siebten Hyperparametersätzen im Mittel eine leicht steigende Tendenz festzustellen ist; Netzwerke mit mehr Parametern scheinen eine bessere Fehlerentwicklung zu bieten. Um die Fehlerentwicklung zu betrachten, wurde diese in Abbildung A.10 visualisiert.

Zu sehen ist die Entwicklung der Fehler der Fitness- und Validierungsmenge. Sowohl die mittleren als auch die besten Fehler bewegen sich nach 200 Iterationen auf einem Niveau (vgl. Tabelle A.1). Auffallend ist, dass je größer der Skalierungsfaktor s_p ist, desto schneller die Fehler fallen.

In Abbildung A.11 ist die Anzahl an verworfenen Individuen am Stück zu sehen. Anfangs finden, wie in den anderen Hyperparametergruppen, stets schnell Verbesserungen statt; mit steigender Iterationszahl werden Verbesserungen seltener. Insgesamt lässt sich jedoch keine klare Tendenz feststellen, welcher Parametersatz bezüglich der kontinuierlichen Verbesserung der Individuen am geeignetsten ist.

4.2.2.4 Zusammenfassung, Ergebnisse und Bestimmung des finalen Hyperparametersatzes

Zusammenfassend ist festzustellen, dass die Populationsgröße $|P|$ und die Mutationswahrscheinlichkeit p_m maßgeblich die Vielfalt der Population steuern. Je größer die Populationsgröße $|P|$ ist, desto größer ist die mögliche Vielfalt in der Initalpopulation P_0. Je größer der Wert der Mutationswahrscheinlichkeit p_m, desto wahrscheinlicher ist es, dass ein Individuum, welches aus einer Reproduktion entsteht, weniger mit den Eltern gemeinsam hat; die Vielfalt der Population steigt. Die Fitnessskalierung s_p gibt dient

zur Gewichtung zwischen den Zielen schnelle Berechenbarkeit und Klassifikationsergebnisse. Für den finalen Hyperparametersatz wurden auf Grund der Ergebnisse der Untersuchungen der Auswirkung der einzelnen Hyperparameter die Werte aus Tabelle 4.3 verwendet.

Durchlauf (Run)	**Populationsgröße** $\|P\|$	**Mutations WK** p_m	**Fitnessskalierung** s_p
8	10	0,3	0,75

Tabelle 4.3: Finaler Hyperparametersatz zur Entwicklung der finalen Netztopologie.

Wie schon erwähnt bestimmt die Kombination aus Populationsgröße $|P|$ und Mutationswahrscheinlichkeit p_m die Vielfalt der Population. Bei einer kleinen Populationsgröße $|P|$ wurde eine große Mutationswahrscheinlichkeit p_m gewählt, um eine große Vielfalt gewährleisten zu können. Des Weiteren hat die Untersuchung aus Abschnitt 4.2.2.1 gezeigt, dass bei einer Populationsgröße von $|P| = 30$ das fitteste Netzwerk aus der Initialpopulation P_0 stammt und somit keiner Evolution entstammt. Um das zu verhindern wurde $|P| = 10$ gewählt. Die Fitnessskalierung s_p wurde mit $s_p = 0{,}75$ festgelegt. Dieser Wert wurde als Kompromiss zwischen schneller Fehleroptimierung und damit besseren Klassifizierungsergebnissen und Anzahl an Parametern gewählt. Der Unterschied der Anzahl an Parametern des besten Netzwerkes zwischen $s_p = 0{,}6$ und $s_p = 0{,}75$ ist dabei gering, im Vergleich der Anzahl an Parametern der besten Netzwerke zwischen $s_p = 0{,}75$ und $s_p = 0{,}9$ (vgl. Tabelle A.1).

4.2.2.5 Ergebnisse des finalen Hyperparametersatzes

Mit dem finalen Hyperparametersatz wurde der evolutionäre Algorithmus erneut ausgeführt. Die Ergebnisse sind in Abbildung A.12 zu finden.

Wie man in Abbildung A.12.1 sehen kann entspricht das fitteste Netzwerk nach 200 Iterationen dem kleinsten Netzwerk. Die Entwicklung der mittleren Anzahl an Parametern ist stets um $1{,}1 \cdot 10^7$ zu finden; das kleinste und fitteste Netzwerk hat ca. $4{,}7 \cdot 10^6$ Parameter. In Abbildung A.12.2 ist die Entwicklung der Anzahl an verworfenen Individuen am Stück zu sehen. Auch hier ist das bekannte Muster zu erkennen, dass anfangs viele Verbesserungen der Fitness stattfinden und mit Fortschreiten des evolutionären Algorithmus die Verbesserungen seltener werden. Abbildung A.12.4 und Abbildung A.12.3 zeigen die Entwicklungen der Fehler der Validierungs- und Fitnessmenge. Anfangs entspricht das fitteste Netzwerk dem Netzwerk mit dem kleinsten Fehlern, ab Iteration 100 entsteht eine Netztopologie, welche ein fitteres Individuum zur Folge hat aber eine schlechtere Leistung in der Klassifizierung besitzt.

Die Eigenschaften des fittesten Individuums des finalen Hyperparametersatzes sind in Tabelle 4.4 zu finden.

Eigenschaft	Werte
Anzahl aktiver Schichten	4
Anzahl der Einheiten dichter Schichten	4800, 3248, 989, 506, 3606
Anzahl der Einheiten der Faltungsschichten	24872, 20239, 15646, 8141, 13628
Arten der Schichten	Conv, Dense, Dense, Dense, Dense
Dropout Wahrscheinlichkeit der Eingangsschicht	0,230189701777
Filtergrößen der Faltungsschichten	$3 \times 3, 3 \times 3, 3 \times 3, 1 \times 1, 3 \times 3$
Poolinggrößen der Faltungsschichten	$2 \times 2, 2 \times 2, 2 \times 2, 2 \times 2, 2 \times 1$
Aktivierungsfunktionen	Tanh, ReL, ReL, LReL, ReL
Lernrate	0,0778735639298

Tabelle 4.4: Finales Genom mit 4 aktiven Schichten und 5 maximal möglichen aktiven Schichten.

Dieses Individuum wird nun für die zwei Datensätze angelernt, um damit die Auswertung der Gesamtbilder machen zu können. Die Lernprozesse sind in Abschnitt 4.3 zu finden.

4.2.2.6 Weitere Erkenntnisse

Im Allgemeinen lässt sich feststellen, dass es zur Ausführung des evolutionären Algorithmus zu einem geringen Overfitting des Klassifizierungsfehlers kommt. Dies ist im Vergleich zwischen der Fehlerentwicklung des Validierungs- und Fitnessfehlers zu sehen; der Validierungsfehler liegt typischerweise über dem Fitnessfehler. In Abbildung A.2 sind die Differenzen der Fehler der Fitnessmenge gegenüber den Fehlern der Validierungsmenge dargestellt. Diese Differenz ist sowohl für die mittleren Fehler als auch für die Fehler der fittesten Individuen über die meisten Iterationen hin leicht negativ; es kommt also zu einem leichten Overfitting. Auf Grund der Overfitting Minimierungsmaßnahmen, Dropoutschichten und Early Stopping, kommt es nur zu wenig Overfitting.

Es kristallisierte sich auch ein Muster bei der Verteilung der Schichttypen heraus. Hierfür wurden alle finalen Ergebnisse der Durchläufe des evolutionären Algorithmus betrachtet. In Tabelle 4.5 ist zu sehen, dass es sich bei der ersten Schicht typischerweise um eine Faltungsschicht handelt; erst ab der zweiten Schicht übernehmen die dichten Schichten die Überhand. Der Fakt, dass bei den späteren Schichten wenig Faltungsschichten zum Einsatz kommen, ist auch dem Algorithmus an sich anzulasten. In Abschnitt 3.2.2 ist beschrieben, dass nach einer dichten Schicht keine Faltungsschicht mehr kommen kann.

Bei einer weiteren Analyse über sämtliche Läufe des evolutionären Algorithmus wurden die Aktivierungsfunktionen der einzelnen Schichten betrachtet. Es ist keine Tendenz zu erkennen, welche Schichtnummer eine gewisse Aktivierungsfunktion bevorzugt; dennoch kann man sehen, dass ReL, Sigmoid, Tanh und LReL Aktivierungs-

Schichtnummer	Faltungsschicht	Dichte Schicht
1	84	26
2	11	99
3	5	105
4	1	109
5	0	110

Tabelle 4.5: Verteilung der Schichttypen über die Schichten im evolutionären Algorithmus über alle Hyperparametersätze.

funktionen ein Großteil der eingesetzten Aktivierungsfunktionen ausmachen (siehe Tabelle 4.6).

Schichtnummer	ReL	LReL	Linear	Sigmoid	Softmax	Tanh
1	45	19	0	18	0	28
2	26	27	1	38	0	18
3	24	24	2	31	0	28
4	41	16	7	22	4	20
5	25	20	3	39	8	15
$\sum$	162	106	13	148	12	109

Tabelle 4.6: Verteilung der Aktivierungsfunktionen über die Schichten im evolutionären Algorithmus über alle Hyperparametersätze.

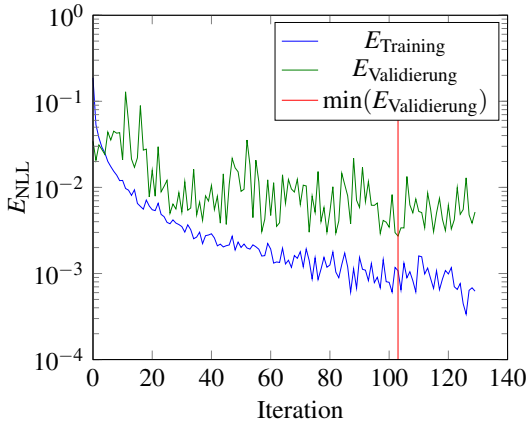
Weitere Erkenntnisse sind, dass im Rahmen dieser Arbeit eine mittlere Lernrate von $\text{mean}(\eta_{\forall \text{ Paramametersätze}}) \approx 0{,}072$ zu beobachten ist, die mittlere Dropout Wahrscheinlichkeit $\text{mean}(p_{\text{drop}_{\forall \text{ Paramametersätze}}}) \approx 0{,}093$ betrug und die mittlere Anzahl an Schichten ca. 2,96 entsprach.

4.3 Neuronales Netzwerk

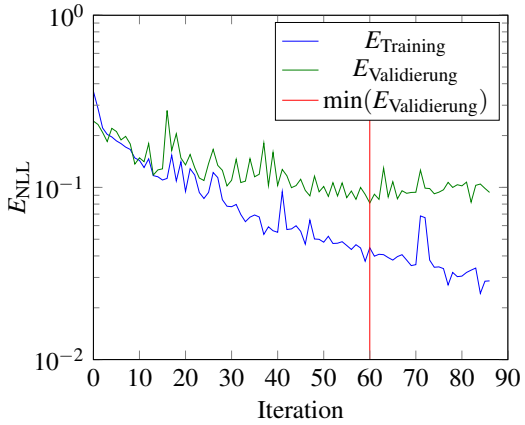
Das Individuum mit der finalen Netztopologie aus Abschnitt 4.2.2.5 wurde zum einen mit dem synthetischen Datensatz und zum anderen mit dem Kundendatensatz angelernt. Dieser Abschnitt beinhaltet die Klassifikationsleistung der Neuronalen Netzwerke, die die finale Netztopologie vom evolutionären Algorithmus erhalten haben.

4.3.1 Lernprozesse und Klassifizierungsleistung

In den Abbildung 4.2 sind die entsprechenden Lernkurven der Neuronalen Netzwerke für den synthetischen Datensatz und den Kundendatensatz zu sehen.



4.2.1 Lernkurve synthetischer Datensatz.



4.2.2 Lernkurve Kundendatensatz.

Abbildung 4.2: Lernkurven der finalen Neuronalen Netzwerke für den synthetischen Datensatz und den Kundendatensatz. Zu sehen ist der Trainingsfehler und der Validierungsfehler sowie die Iteration, die den kleinsten Validierungsfehler liefert.

Wie man sieht kommt es auch hier bei beiden Neuronalen Netzwerken zu einem Overfitting und auch das Early Stopping wird vor Erreichen der maximal möglichen Anzahl an Iteration ausgeführt. Die Leistungsfähigkeiten der Neuronalen Netzwerke als reine Klassifizierer von Einzeldaten aus den entsprechen Testmengen ist in den Tabellen 4.7 und 4.8 zu sehen.

echter Typ \ klassifizierter Typ	DataMatrix-Code	kein DataMatrix-Code
DataMatrix-Code	10111	2
kein DataMatrix-Code	33	10142

Tabelle 4.7: Klassifizierungsergebnisse des Neuronalen Netzwerkes der Testdaten (Bildausschnitte). Dieses Neuronale Netzwerk wurde auf dem synthetischen Datensatz angelernt und ausgewertet.

Aus Tabelle 4.7 lassen sich folgende Werte berechnen:

$$p(\text{richtig klassifiziert}) = \frac{10111+10142}{10111+10142+33+2} \approx 0{,}9983 \tag{4.2}$$

$$p(\text{falsch klassifiziert}) = \frac{2+33}{10111+10142+33+2} = 1 - p(\text{richtig klassifiziert}) \approx 0{,}0017 \tag{4.3}$$

$$p(\text{DMC}^4\text{richtig klassifiziert}|\text{DMC klassifiziert}) = \frac{10111}{10111+2} \approx 0{,}9998 \tag{4.4}$$

$$p(\text{DMC richtig klassifiziert}|\text{DMC}) = \frac{10111}{10111+33} \approx 0{,}9967 \tag{4.5}$$

Die falsch erkannten Testdaten sind in den Abbildungen B.1 und B.2 zu sehen.

In Abbildung B.1 sieht man die DataMatrix-Codes, die fälschlicherweise nicht als DataMatrix-Codes klassifiziert wurden. Ein Großteil der DataMatrix-Codes besitzt eine geringe Modulgröße, wodurch diese nicht mehr als DataMatrix-Codes identifiziert werden konnten.

Weiter existieren Ausschnitte, die von großen Flächen dominiert werden; es sind wenige Hinweise auf einen DataMatrix-Code zu erkennen. Betrachtet man die Bildausschnitte die Werte nah an der Entscheidungsschwelle von 0,5 als Ausgabe vom Neuronalen Netzwerk erhalten sind deutlich DataMatrix-Codes zu erkennen. An dieser Stelle versagt der Klassifizierer.

In Abbildung B.2 sieht man zwei Bildausschnitte, die Schrift mit hohen Kontrast zur Umgebung beinhalten. Diese wurden wahrscheinlich auf Grund des hohen Kontrastes und den häufigen Wechsel zwischen hellen und dunklen Stellen fälschlicherweise als DataMatrix-Codes klassifiziert.

Die gleichen Berechnungen können für die Werte aus Tabelle 4.8 durchgeführt werden.

[4] **D**ata**M**atrix-**C**ode

echter Typ \ klassifizierter Typ	DataMatrix-Code	kein DataMatrix-Code
DataMatrix-Code	611	17
kein DataMatrix-Code	11	669

Tabelle 4.8: Klassifizierungsergebnisse des Neuronalen Netzwerkes der Testdaten (Bildausschnitte). Dieses Neuronale Netzwerk wurde auf dem Kundendatensatz angelernt und ausgewertet.

$$p(\text{richtig klassifiziert}) = \frac{611+669}{611+669+11+17} \approx 0{,}9786 \tag{4.6}$$

$$p(\text{falsch klassifiziert}) = \frac{11+17}{611+669+11+17} = 1 - p(\text{richtig klassifiziert}) \approx 0{,}0214 \tag{4.7}$$

$$p(\text{DMC richtig klassifiziert}|\text{DMC klassifiziert}) = \frac{611}{611+17} \approx 0{,}9729 \tag{4.8}$$

$$p(\text{DMC richtig klassifiziert}|\text{DMC}) = \frac{611}{611+11} \approx 0{,}9823 \tag{4.9}$$

Die falsch erkannten Testdaten des Kundendatensatzes sind in den Abbildungen B.3 und B.4 dargestellt.

Abbildung B.3 zeigt die Bildausschnitte, die fälschlicherweise als DataMatrix-Code erkannt wurden.

Die Abbildungen B.3.2, B.3.6, B.3.7, B.3.11, B.3.16 und B.3.17 zeigen alle horizontale, beziehungsweise vertikale Linien auf, die potentiell zu einem DataMatrix-Code gehören könnten.

Abbildung B.3.5 zeigt ein kontraststarkes Herstellerlogo, das harte Übergänge von dunklen zu hellen Flächen beinhaltet.

In Abbildung B.3.1 und in Abbildung B.3.4 erkennt man jeweils ein Teil eines Buchstabens. Das Neuronale Netzwerk gibt einen Wert von $p = 0{,}51$ für den Bildausschnitt von Abbildung B.3.1 aus, welcher nah an der Entscheidungsschwelle von 0,5 ist. Die große dunkle Fläche in Kombination mit dem Teil eines hellen Buchstabens wurde fälschlicherweise als DataMatrix-Code interpretiert. Analog dazu sieht es in Abbildung B.3.4 aus, nur dass der Wert mit $p = 0{,}63$ etwas weiter von der Entscheidungsschwelle ist.

Die Abbildungen B.3.3, B.3.8, B.3.9, B.3.12, B.3.13 und B.3.14 zeigen jeweils Teile kontraststarker Schriftzüge. Diese wurden auch fälschlicherweise als DataMatrix-Code klassifiziert. Dies könnte mit der hohen Anzahl an Kanten und Ecken sowie dem hohen Kontrast zusammenhängen.

Abbildung B.3.15 beinhaltet zwei parallele Linien, die als DataMatrix-Code identifiziert wurden. Dies könnte im Zusammenhang mit der Form der Module der DataMatrix-Codes stehen.

In Abbildung B.3.10 sieht man einen weiteren Bildausschnitt, der fälschlicherweise als DataMatrix-Code klassifiziert wurde. In diesem Fall handelt es sich um einen Ausschnitt eines QR-Codes, der auf Grund seines Aufbaus, DataMatrix-Codes ähnlich ist.

In Abbildung B.4 sieht man Bildausschnitte mit DataMatrix-Codes, die nicht als DataMatrix-Codes erkannt wurden.

Die Abbildungen B.4.2, B.4.3, B.4.9 und B.4.8 haben einen vergleichsweise hohen Anteil an einfarbiger Fläche, was möglicherweise dazu geführt hat, dass diese nicht als DataMatrix-Code identifiziert werden konnten.

Abbildung B.4.1 könnte auch als Buchstabe interpretiert werden. Es sind wenig Hinweise auf einen DataMatrix-Code vorhanden.

Abbildungen B.4.2, B.4.3, B.4.4 und B.4.5 haben vergleichsweise große Modulgrößen. Diese Bildausschnitte haben möglicherweise zu wenig der entsprechenden DataMatrix-Codes beinhalten, um als solche erkannt zu werden.

In den Abbildungen B.4.6, B.4.7 und B.4.10 ist der Kontrast im Vergleich zu den anderen Bildausschnitten niedrig, was zu einer niedrigeren DataMatrix-Code Wahrscheinlichkeit geführt haben könnte.

Abbildung B.4.11 ist mit $p = 0{,}454$ nah an der DataMatrix-Code Schwelle von 0,5. Es dominiert ein dunkler Bereich; der Ausschnitt zeigt gegebenenfalls zu wenig Informationen, um als DataMatrix-Code erkannt zu werden.

4.4 Auswertung Gesamtbilder

Dieser Abschnitt befasst sich mit der Detektionsleistung von DataMatrix-Codes in Bilder mittels den in Abschnitt 3.3 vorgestellten Algorithmus.

Die Konfiguration entspricht dem Beispiel aus Abschnitt 3.3:

Beschreibung	**Werte**
Schrittweite	8 px, 8 px
Minimale DMC Größe	24 px, 24 px
Weichzeichnungsradius r	9 px, 9 px
Entscheidungsschwelle	0,5
Anfangsskalierung	1
Endskalierung	0,75
Skalierungsschritt	0,25

Tabelle 4.9: Parameter des Detektionsalgorithmus zur Detektion von DataMatrix-Codes.

Mit der Konfiguration aus Tabelle 4.9 für den Detektionsalgorithmus wurden die Testbilder des synthetischen Datensatzes und die des Kundendatensatzes analysiert. Die Ergebnisse sind dabei in den folgenden Unterabschnitten getrennt nach Datensätzen zu finden.

Als gefundener DataMatrix-Code gilt ein Code, der komplett von einer ROI umgeben ist beziehungsweise in dem ein ROI liegt. Über die Genauigkeit der Lokalisierung wird in den weiteren Abschnitten eingegangen.

4.4.1 Synthetischer Datensatz

Der synthetische Datensatz wurde mit der in Abschnitt 4.2.2.5 vorgestellten finalen Netztopologie ausgewertet. Der Lernprozess des finalen Netzwerkes sowie die Klassifizierungsleistung einzelnen Bildausschnitten für den synthetischen Datensatz ist in Abschnitt 4.3.1 beschrieben.

Von den 2000 Testbildern mit insgesamt 1000 DataMatrix-Codes sind die Testergebnisse wie folgt:

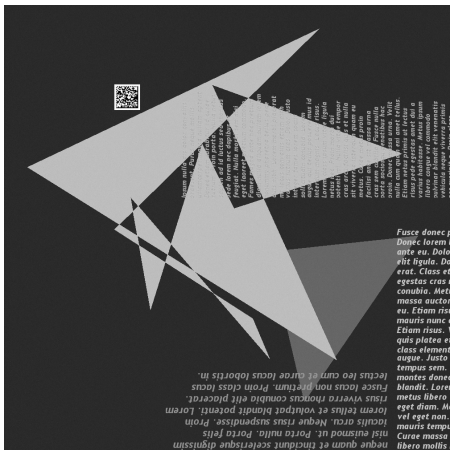
Es wurden insgesamt 999 DataMatrix-Codes korrekt detektiert. Bei einem Bild konnte der vorhandene DataMatrix-Code nicht gefunden werden. Es gab keine Falschdetektionen. Daraus ergibt sich:

$$p(\text{richtig klassifiziert}) = \frac{999 + 1000}{20000} = 0{,}9995 \tag{4.10}$$

$$\tag{4.11}$$

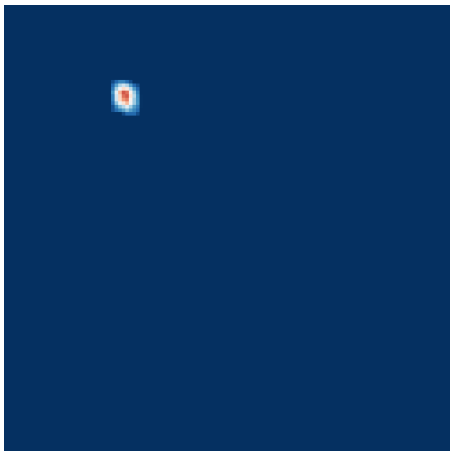
Das Testbild, bei dem der DataMatrix-Code nicht erkannt wurde, ist in Abbildung 4.3 zu sehen. Auf den entsprechenden Heatmaps erkennt man, dass man diesen Code mit einer niedrigeren Schwelle problemlos detektieren könnte, da in der Umgebung keine weiteren potentiellen DataMatrix-Codes sind. Allerdings ist die DataMatrix-Code Wahrscheinlichkeit nicht sehr hoch; dies könnte mit der geringen Modulgröße von ca.

2 px zusammenhängen. Dies würde auch erklären, warum die Heatmap für den Skalierungsfaktor von 100 % eine deutlich höhere DataMatrix-Code Wahrscheinlichkeit aufweist als mit dem Skalierungsfaktor 75 % (vgl. Abbildung 4.3.3 und 4.3.4).

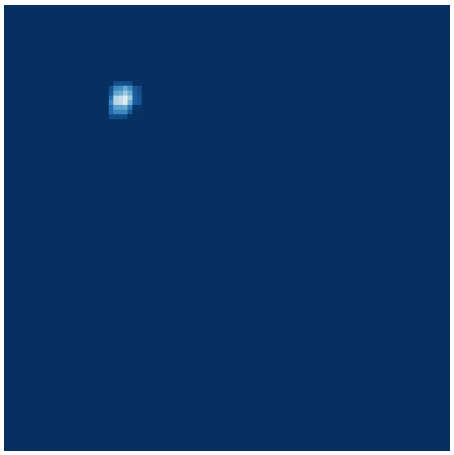


4.3.1 Testbild Nr. 899 - nicht detektierter DataMatrix-Code.

4.3.2 Testbild Nr. 899 - Heatmap gesamt.



4.3.3 Testbild Nr. 899 - Heatmap 100 % skaliert.



4.3.4 Testbild Nr. 899 - Heatmap 75 % skaliert.

Abbildung 4.3: Bildausschnitte, die die nicht detektierten DataMatrix-Codes beinhalten.

Die Ergebnisse des Detektionsalgorithmus wurden auch bezüglich der Lokalisationsgenauigkeit analysiert. Es wurden die jeweiligen Abweichungen in der Größe der dargestellten ROI mit den eigentlichen Rand des DataMatrix-Codes verglichen. Der Rand des DataMatrix-Codes wird in diesem Zusammenhang als das Suchmuster definiert, das die Informationsmodule umschließt; die Ruhezone um den DataMatrix-Code wird vernachlässigt. Es wurde außerdem nur die fehlende Anzahl an Pixeln betrachtet,

da zu große ROIs typischerweise noch innerhalb der Ruhezone der DataMatrix-Codes sind.

Wichtig ist an dieser Stelle, dass die Anzahl an Pixeln nur ein Hinweis auf die Güte der Lokalisation bietet. Die DataMatrix-Codes haben unterschiedliche Größen und auch unterschiedliche Modulgrößen; um eine Allgemeingültige Aussage für diesen Datensatz machen zu können müsste man die fehlende Anzahl an Pixeln beispielsweise auf die Größe eines Moduls normieren. Des Weiteren wurde die Auswertung händisch vorgenommen und kann somit fehlerbehaftet sein. Außerdem ist beispielsweise eine Abweichung von 5 px bei einer Modulgröße von 10 px unkritisch, da der DataMatrix-Code trotzdem noch vollständig dekodiert werden könnte; bei einer Modulgröße von 3 px würde hingegen mindestens eine Modulreihe fehlen.

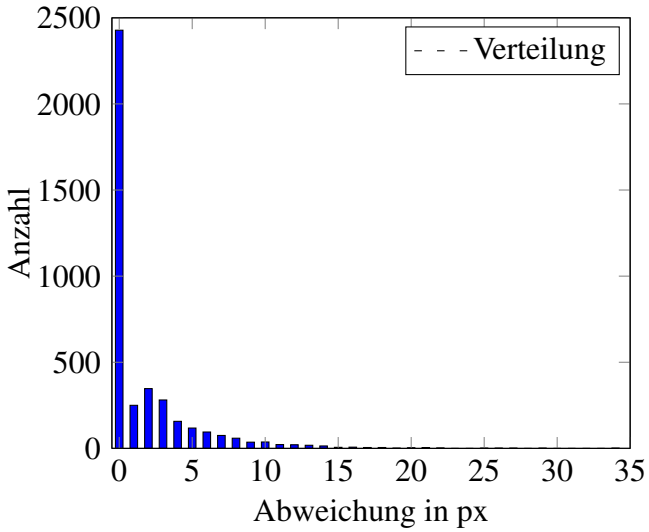


Abbildung 4.4: Verteilung der Größenabweichung des Detektionsergebnisses vom tatsächlichen DataMatrix-Code.

Das Ergebnis dieser Abweichung ist in Abbildung 4.4 zu sehen. Im Mittel existiert eine Abweichung in jede Richtung von ca. 1,7 px. Um die Lokalisation zu verbessern gäbe es mehrere Ansätze, die man verfolgen könnte. Zum einen könnte die Schrittweite verringert werden, um eine bessere Aussage treffen zu können, wo sich ein DataMatrix-Code befindet. Dies wird in Abschnitt 4.4.3 noch weiter ausgeführt.

Zum anderen könnte das Neuronale Netzwerk unter anderem auch mit unvollständigen DataMatrix-Code Bildausschnitten angelernt werden; also Bildausschnitten mit beispielsweise 50 % eines DataMatrix-Codes. Dieses Vorgehen wurde in [8] erfolgreich eingesetzt.

4.4.2 Kundendatensatz

In der Testmenge des Kundendatensatzes sind, wie in Abschnitt 2.1.3.2 zu sehen, 193 DataMatrix-Codes vorhanden. Diese sind in 95 Bildern verteilt. Des Weiteren existieren 193 Bilder ohne DataMatrix-Codes. Die Analyse dieser Testdaten ergab folgende Ergebnisse:

- 174 DataMatrix-Codes wurden detektiert
- 13 DataMatrix-Codes müssen gesondert behandelt werden
- 6 DataMatrix-Codes wurden nicht detektiert
- 2 Falschdetektionen gab es

Bei den 173 Detektierten DataMatrix-Codes wurden, analog zu Abschnitt 4.4.1, die Lokalisation überprüft. Die Ergebnisse sind in Abbildung 4.5 zu sehen. Daraus ergibt sich im Mittel eine Abweichung von 7 px in jede Richtung.

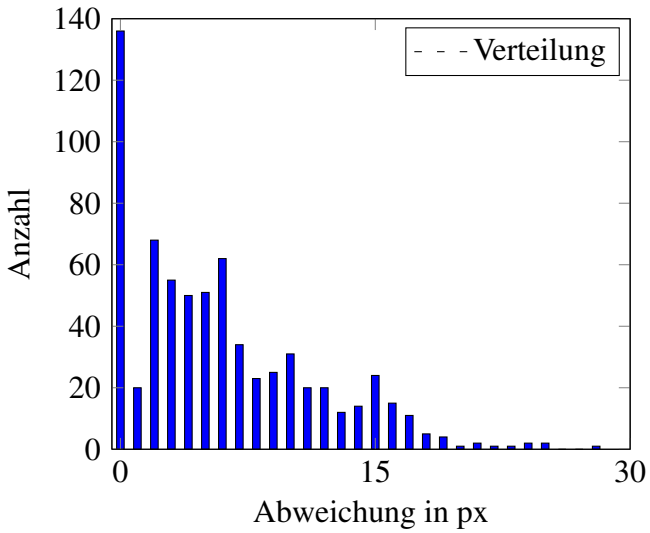
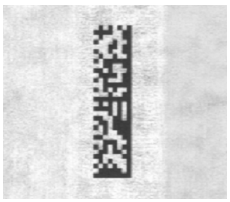
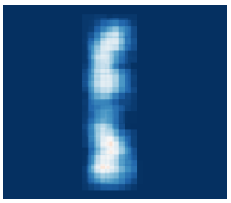


Abbildung 4.5: Verteilung der Größenabweichung des Detektionsergebnisses vom tatsächlichen DataMatrix-Code.

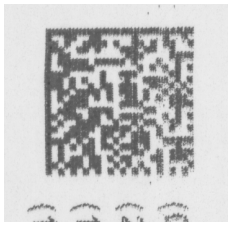
Die 6 nicht detektierten DataMatrix-Codes sind in Abbildung 4.6 zu sehen.



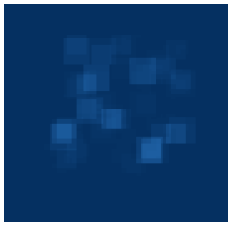
4.6.1 Testbild Nr. 39 - kritischer DataMatrix-Code.



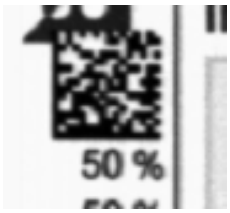
4.6.2 Testbild Nr. 39 - Heatmap.



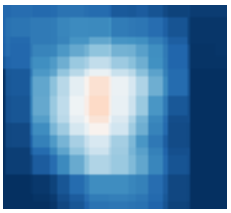
4.6.3 Testbild Nr. 43 - kritischer DataMatrix-Code.



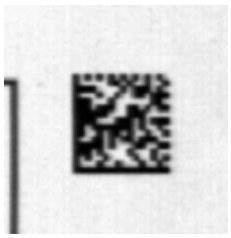
4.6.4 Testbild Nr. 43 - Heatmap.



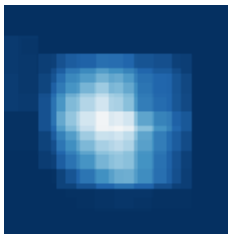
4.6.5 Testbild Nr. 63 - kritischer DataMatrix-Code.



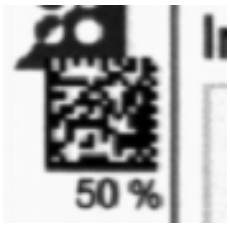
4.6.6 Testbild Nr. 63 - Heatmap.



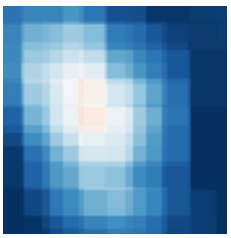
4.6.7 Testbild Nr. 63 (2) - kritischer DataMatrix-Code.



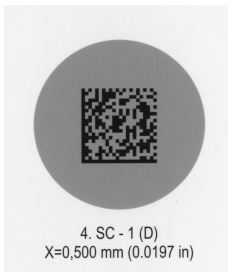
4.6.8 Testbild Nr. 63 (2) - Heatmap.



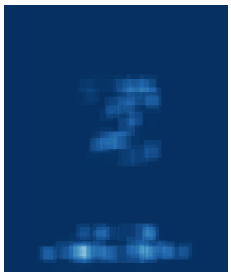
4.6.9 Testbild Nr. 63 (3) - kritischer DataMatrix-Code.



4.6.10 Testbild Nr. 63 (3) - Heatmap.



4.6.11 Testbild Nr. 93 - kritischer DataMatrix-Code.



4.6.12 Testbild Nr. 93 - Heatmap.

Abbildung 4.6: Bildausschnitte, die die nicht detektierten DataMatrix-Codes beinhalten.

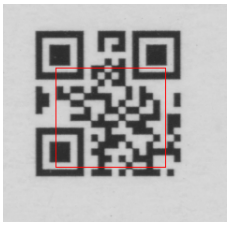
In Abbildung 4.6.1 ist ein DataMatrix-Code zu sehen, dessen Hintergrund stark verschmiert ist, wodurch zum einen der Kontrast niedrig wird und zum anderen Bildstörungen hinzukommen. In der passenden Heatmap erkennt man, dass kein hinreichend großer und starker Bereich zusammen gekommen ist, der für eine DataMatrix-Code Detektion ausreicht. Dieses Problem mit schwachkonstrastigen DataMatrix-Codes könnte man entgegenwirken, indem mehr schwachkonstrastige DataMatrix-Codes in die Lernmenge mit aufgenommen werden.

In Abbildung 4.6.3 sieht man einen weiteren nicht detektierten DataMatrix-Code. Auch hier ist der Kontrast schwach sowie das Druckbild verwaschen. In der Heatmap

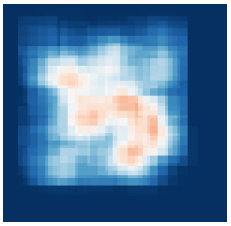
sind wenig Hinweise auf einen DataMatrix-Code zu sehen.

Die Abbildungen 4.6.5, 4.6.7 und 4.6.9 stammen alle aus dem selben Bild. Alle drei DataMatrix-Codes sind nicht scharf; die einzelnen Module sind kaum als solche zu erkennen. Dennoch sieht man in den entsprechenden Heatmaps jeweils einen Hinweis auf einen DataMatrix-Code. Hier könnte man gegebenenfalls dem Schwellwert des Algorithmus etwas verringern, um diese DataMatrix-Codes noch detektieren zu können. Allerdings muss man beobachten, ob die Fehldetektionsrate dadurch steigt.

In Abbildung 4.6.11 sieht man einen DataMatrix-Code von einem Validierungsmuster. Auf Grund des niedrigen Kontrastes in der Umgebung des DataMatrix-Codes wird dieser nicht erkannt. Es gibt keine nennenswerten Hinweise auf der passenden Heatmap.

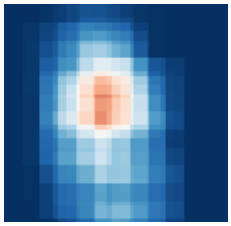


4.7.1 Testbild Nr. 29 - Fehldetektion.



4.7.2 Testbild Nr. 29 - Heatmap.

4.7.3 Testbild Nr. 63 - Fehldetektion.



4.7.4 Testbild Nr. 63 - Heatmap.

Abbildung 4.7: Bildausschnitte, die die falsch detektierten DataMatrix-Codes beinhalten.

In Abbildung 4.7 sind die Fehldetektionen zu sehen.

Bei Abbildung 4.7.1 handelt es sich um einen QR-Code, der im Informationsbereich, auf Grund des ähnlichen Aufbaus mittels Modulblöcken, fälschlicherweise als DataMatrix-Code erkannt wurde.

Abbildung 4.7.3 zeigt den Buchstaben "M"; die entsprechende Heatmap zeigt eine hohe Wahrscheinlichkeit für das Vorhandensein eines DataMatrix-Codes auf. An dieser Stelle versagt das Neuronale Netzwerk als Klassifizierer. Man könnte gegebenenfalls das Neuronale Netzwerk mit mehr Daten anlernen, um eine bessere Klassifizierungsleistung zu bekommen. Diese Daten müssten mehr Buchstaben und Sätze in verschiedenen Orientierungen enthalten, um eine solche Fehldetektion ausschließen zu können. Allerdings sollte dann auch die Anzahl an Ausschnitten von DataMatrix-Codes erhöht werden und überprüft werden, dass kein Overfitting stattfindet.

In Abbildung C.1 sind alle gesondert behandelten DataMatrix-Codes zu sehen. Hierbei handelt es sich um längliche DataMatrix-Codes, die nicht als ein kompletter DataMatrix-Code detektiert wurden, sondern als mindestens zwei getrennte DataMatrix-Codes. All diese Bildausschnitte stammen aus dem selben Testbild. Um diese DataMatrix-Codes als komplette DataMatrix-Codes detektieren zu können, könnte man beispielsweise eine Annahme über die örtliche Verteilung von DataMatrix-Codes in Drucker-

zeugnissen machen. Liegen zwei ROIs dicht genug bei einander könnten diese zusammengefasst werden. Außerdem könnte man auch den Weichzeichnungsradius r anpassen, um größere Bereiche zusammenfassen zu können.

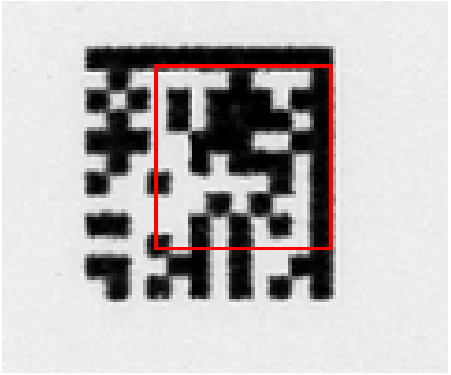
4.4.3 Schrittweite

In diesem Abschnitt soll die Auswirkung der Schrittweite auf die Lokalisationsgenauigkeit der DataMatrix-Codes exemplarisch ausgewertet werden. Mit Verringerung der Schrittweite steigt die Anzahl an zu evaluierenden Daten drastisch. Halbiert man die Schrittweite vervierfacht sich die Anzahl an Bildausschnitten (unter Vernachlässigung der Ränder).

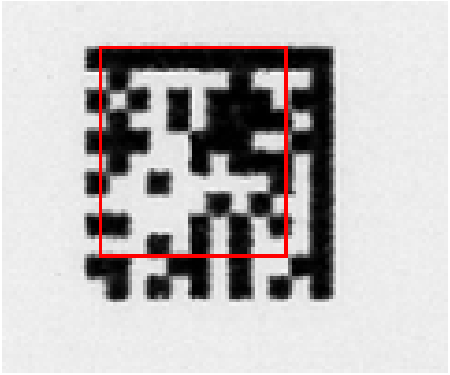
Hierfür wurde der Scan aus Abbildung 4.8 betrachtet.

Abbildung 4.8: Testbild Nr. 59 - Kundendatensatz.

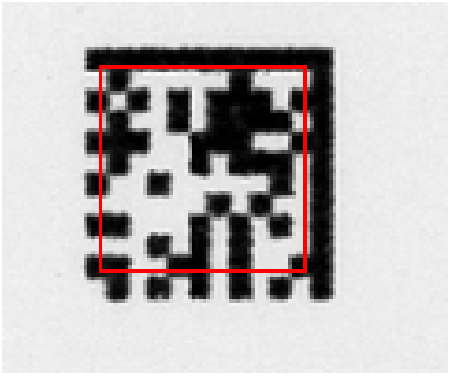
In Abbildung 4.4.3 wurden unterschiedliche Schrittweiten evaluiert.



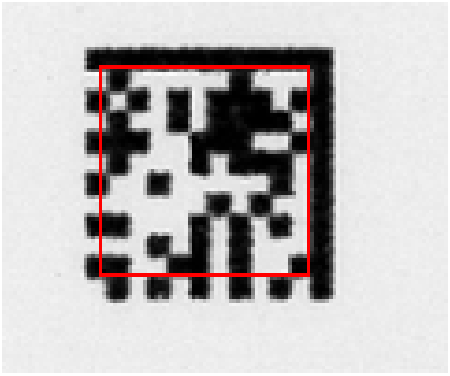
4.9.1 Testbild Nr. 59 - Ausschnitt DataMatrix-Code; Schrittweite 32 px, 32 px.



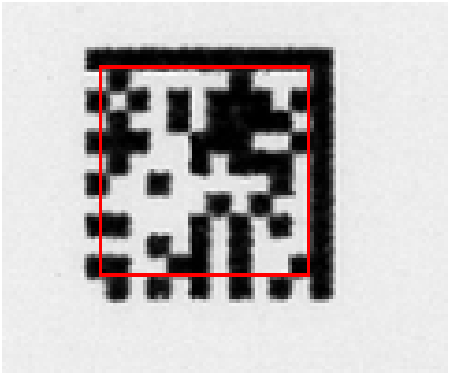
4.9.2 Testbild Nr. 59 - Ausschnitt DataMatrix-Code; Schrittweite 16 px, 16 px.



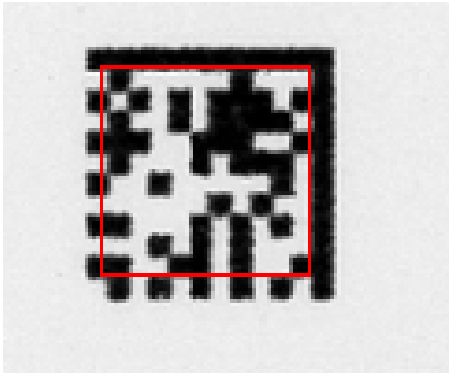
4.9.3 Testbild Nr. 59 - Ausschnitt DataMatrix-Code; Schrittweite 8 px, 8 px.



4.9.4 Testbild Nr. 59 - Ausschnitt DataMatrix-Code; Schrittweite 4 px, 4 px.



4.9.5 Testbild Nr. 59 - Ausschnitt DataMatrix-Code; Schrittweite 2 px, 2 px.



4.9.6 Testbild Nr. 59 - Ausschnitt DataMatrix-Code; Schrittweite 1 px, 1 px.

Wie man in Abbildung 4.4.3 sehen kann ist bei einer Schrittweite, die der Fensterweite entspricht die Lokalisation am schlechtesten. Die Lokalisation der DataMatrix-

Codes, die der Detektionsalgorithmus ermittelt hat, ist dabei mit einem roten Rechteck markiert. Bis zu einer Schrittweite von 4 px, 4 px findet jeweils eine Verbesserung statt; danach stagniert es.

Mit einer kleineren Schrittweite konnte somit bei diesem Bild eine bessere Lokalisation erzielt werden, die allerdings selbst bei einer Schrittweite von 1 px, 1 px nicht das erwartete Ergebnis, eine Umschließung des kompletten DataMatrix-Codes, erreicht.

4.4.4 Zusammenfassung

Es wurde ein evolutionärer Algorithmus verwendet, um eine optimale Netztopologie zu bestimmen. Diese finale Netztopologie hat sich basierend auf dem synthetischen Datensatz entwickelt. Sie ließ sich allerdings auch auf den Kundendatensatz übertragen. Die Evaluierung der Bildausschnitte ergab, dass mit 99,83 % und 97,86 % (synthetischer und nicht synthetischer Datensatz) die Klassifizierung mittels Neuronalem Netzwerk zuverlässig arbeitet.

Die Auswertung der Gesamtbilder mittels dem im Abschnitt 3.3 vorgestellten Algorithmus liefert gemischte Ergebnisse. Im synthetischen Datensatz wurden alle DataMatrix-Codes, bis auf einen, gefunden. Im Kundendatensatz konnten 6 DataMatrix-Codes nicht gefunden werden und es gab 2 Falschdetektionen. Bei beiden Ergebnissen ist festzustellen, dass die Lokalisationsgenauigkeit der DataMatrix-Codes nicht optimal ist; häufig werden die DataMatrix-Codes als zu klein angenommen.

Kapitel 5

Fazit und Ausblick

Ziel dieser Arbeit war DataMatrix-Codes in Druckerzeugnissen zu finden. Es standen dafür zum einen ein Datensatz mit synthetischen Bildern und zum anderen einen Datensatz mit Bildern von Kunden der EyeC GmbH zur Verfügung.

Es wurde sich an der in [8] vorgestellten Methodik orientiert. Neu hinzugekommen ist, dass die Netzwerktopologie mittels evolutionärem Algorithmus bestimmt werden konnte. Des Weiteren wurde gezeigt, dass die Methodik aus [8] sich auch auf DataMatrix-Codes anwenden lässt.

Der umgesetzte evolutionäre Algorithmus wurde erfolgreich eingesetzt, um eine Topologie für ein Neuronales Netzwerk zu erstellen, die die in Abschnitt 4.3.1 aufgezeigte Leistungsfähigkeit besitzt. Hierfür wurde der evolutionäre Algorithmus mit verschiedenen Parametersätzen evaluiert und die Ergebnisse analysiert. Aus den Ergebnissen konnte ein finaler Parametersatz abgeleitet und evaluiert werden. Bei der Evaluation der verschiedenen Parametersätze für den evolutionären Algorithmus konnten außerdem Regelmäßigkeiten bezüglich der entstandenen Netzwerktopologien festgestellt werden, wie zum Beispiel die Bevorzugung einer Faltungsschicht direkt nach der Eingangsschicht.

Die Evaluation des evolutionären Algorithmus wurde dabei nur mit dem synthetischen Datensatz durchgeführt; die dabei entstandene finale Netztopologie konnte daraufhin sowohl mit dem synthetischen Datensatz als auch mit dem Kundendatensatz erfolgreich getestet werden.

Es wurde also die Problematik der Auswahl einer Topologie für ein Neuronales Netzwerk mittels evolutionären Algorithmus gelöst.

Hinzu kommt, dass der Kundendatensatz bezüglich Rauschen untersucht wurde. Dabei ist herausgekommen, dass das Rauschen in den Scans von Druckerzeugnissen eine geringe Varianz hat und somit im Rahmen dieser Arbeit das Rauschen vernachlässigt werden konnte; dies Unterscheidet diese Arbeit von den meisten Verfahren, die in Abschnitt 1.3 vorgestellt wurden. Ein weiterer Unterschied ist, dass DataMatrix-Codes in Scans von Druckerzeugnissen typischerweise nur Rotationen in 90° Schritten so-

wie weitere Rotation mit kleinen Winkeln haben (vgl. Abschnitt 2.1.2.1), wodurch der Detektionsalgorithmus nicht DataMatrix-Codes mit beliebigen Rotationen detektieren muss.

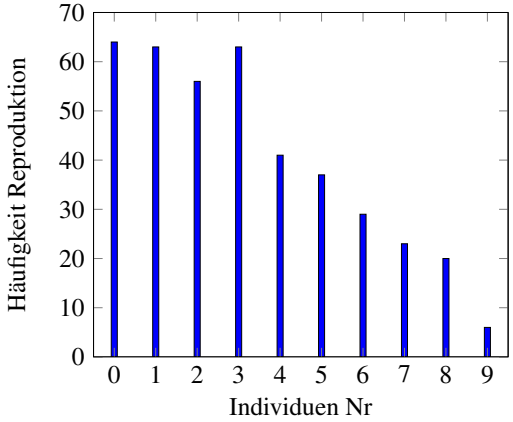
Bezüglich der Leistung des Detektionsalgorithmus muss man festhalten, dass die Lokalisation der DataMatrix-Codes in einer zukünftigen Arbeit noch Verbesserungspotential hat. Hierfür könnte man die in Abschnitt 1.3 vorgestellten Verfahren mit dem Neuronalen Netzwerk aus dieser Arbeit kombinieren. Es könnten zum Beispiel Gebiete mit hohen Kontrasten oder vielen Ecken als zu untersuchende Bereiche definiert werden, die daraufhin mittels Neuronalem Netzwerk klassifiziert werden. Hierzu wurde eine Schnittstelle für C++ implementiert, um weitere Forschung und Entwicklung zu vereinfachen.

Abschließend kann man hervorheben, dass ein "State of the Art" Neuronales Netzwerk implementiert wurde, dessen Topologie ein evolutionärer Algorithmus festgelegt hat. Der evolutionäre Algorithmus wurde implementiert und daraufhin empirisch analysiert; es konnte ein Parametersatz gefunden werden, um erfolgreich eine Netztopologie zur Klassifizierung von DataMatrix-Codes zu erstellen. Daraufhin wurde gezeigt, dass man mittels Neuronalem Netzwerk DataMatrix-Codes in Druckerzeugnissen detektieren kann.

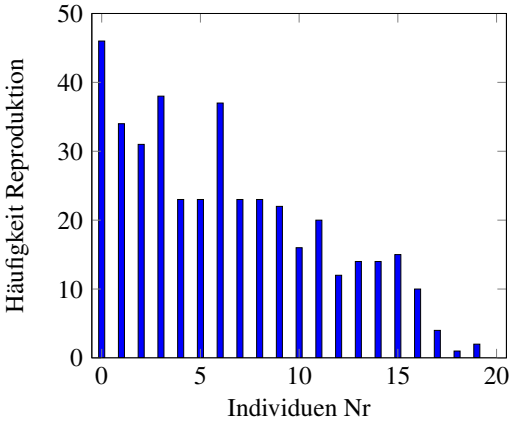
Anhang A

Evolutionärer Algorithmus

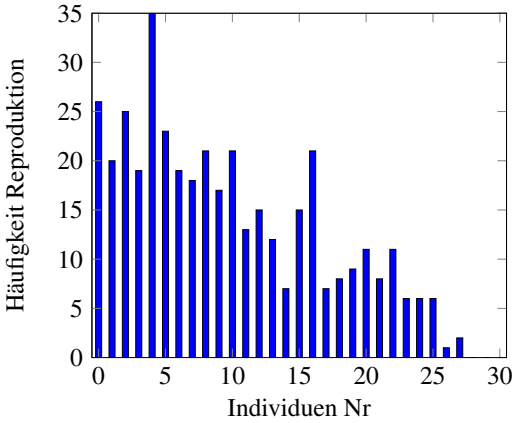
A.1 Allgemein



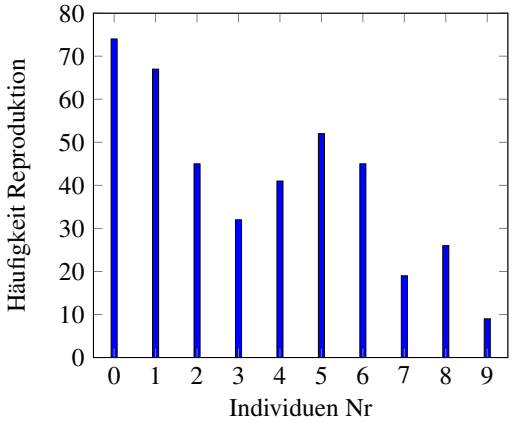
A.1.1 Run 1



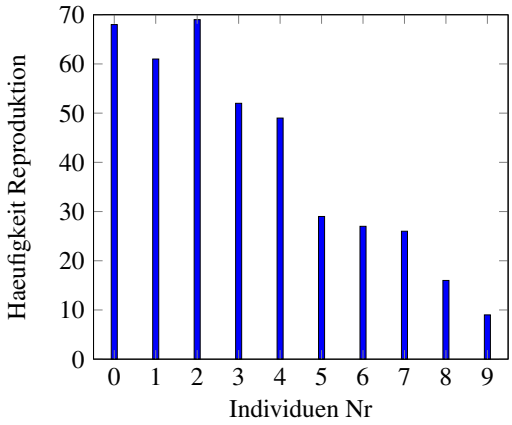
A.1.2 Run 2



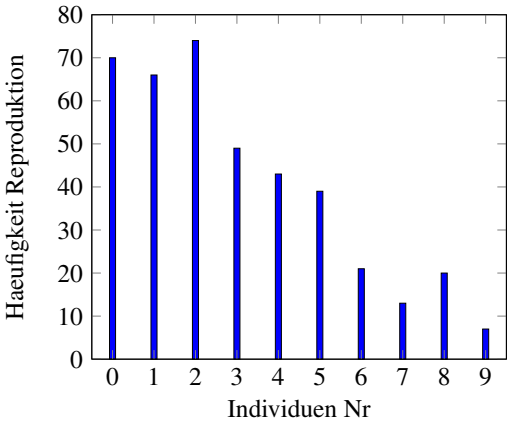
A.1.3 Run 3



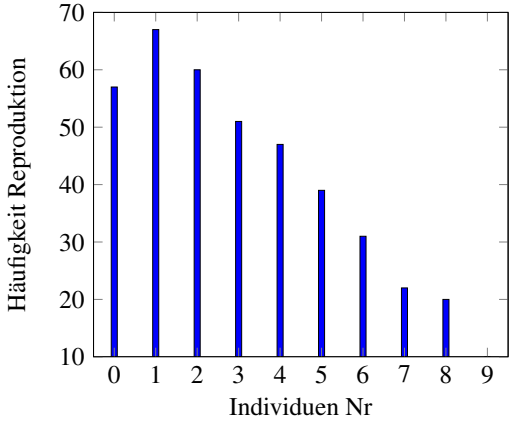
A.1.4 Run 4



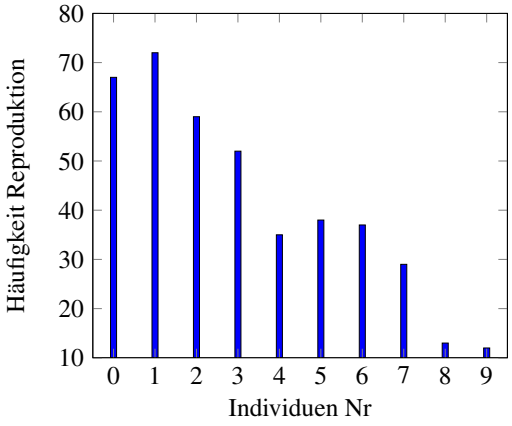
A.1.5 Run 5



A.1.6 Run 6

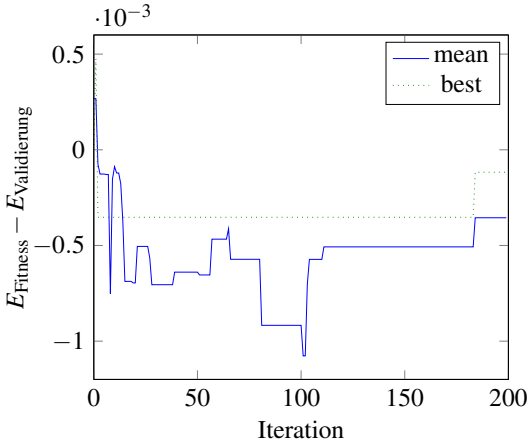


A.1.7 Run 7

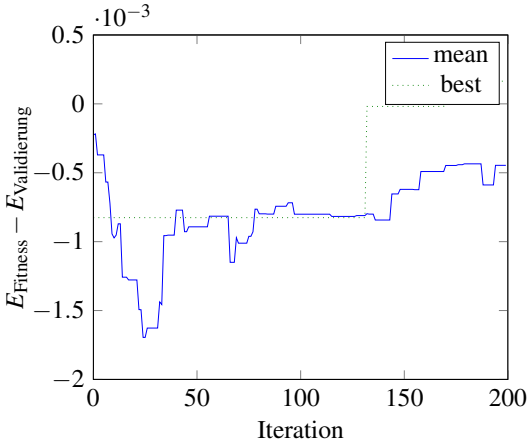


A.1.8 Run 8

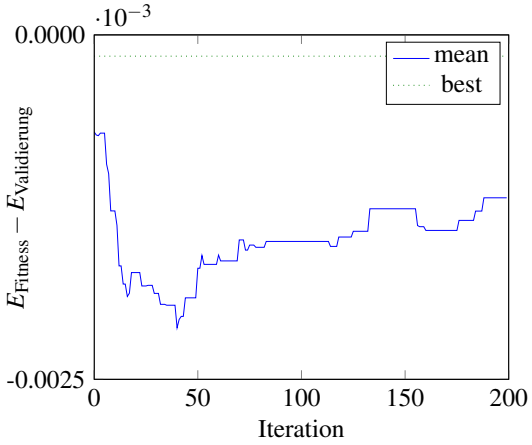
Abbildung A.1: Verteilung der Auswahl der Individuen zur Reproduktion.



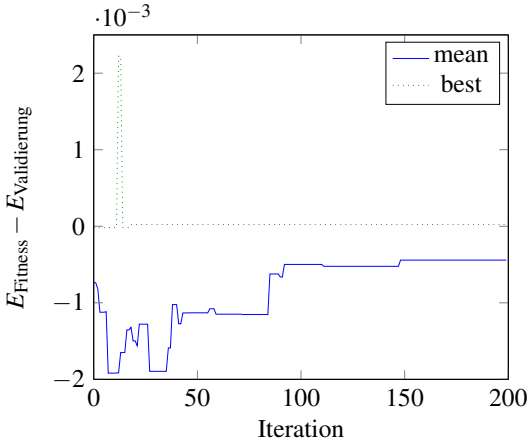
A.2.1 Run 1



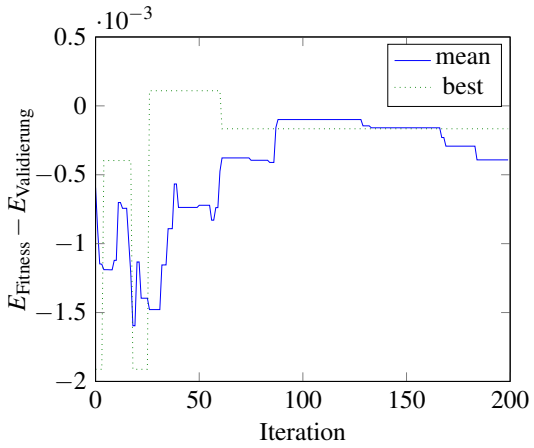
A.2.2 Run 2



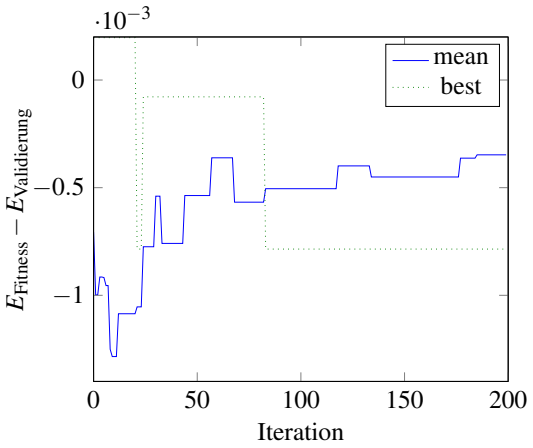
A.2.3 Run 3



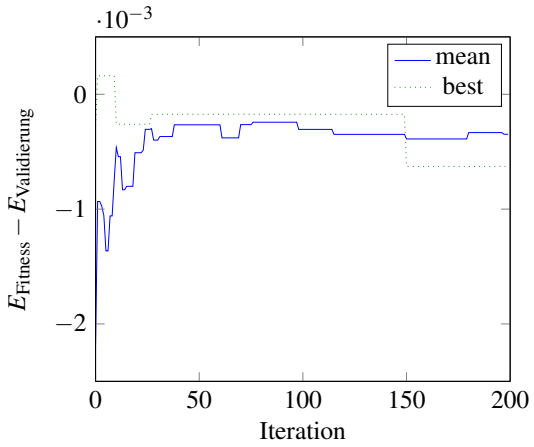
A.2.4 Run 4



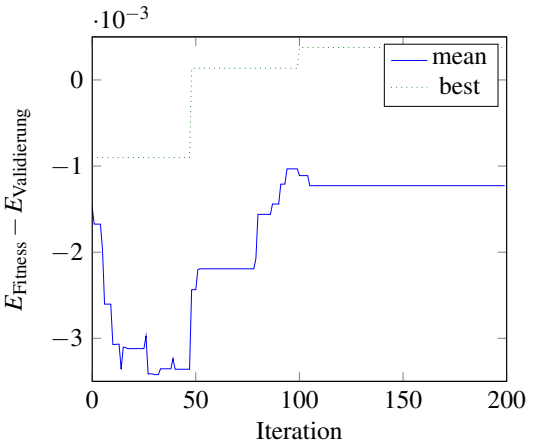
A.2.5 Run 5



A.2.6 Run 6



A.2.7 Run 7



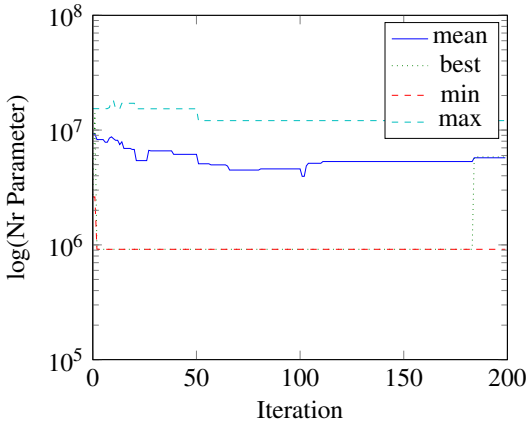
A.2.8 Run 8

Abbildung A.2: Differenz aus Validierungs- und Fitnessfehler während des evolutionären Algorithmus zur Visualisierung des overfittings.

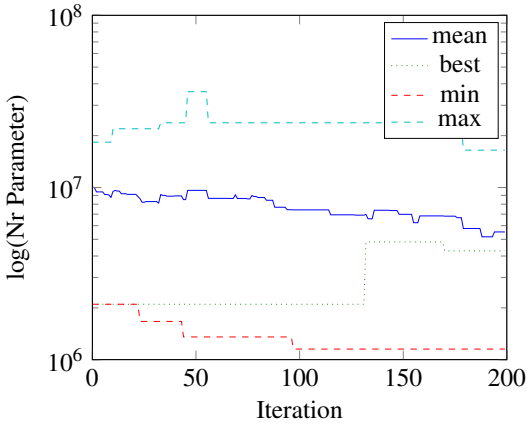
Run	1	2	3	4	5	6	7
s_p	0,6	0,6	0,6	0,6	0,6	0,75	0,9
p_m	0,1	0,1	0,1	0,2	0,3	0,1	0,1
$\|P\|$	10	20	30	10	10	10	10
bester $E_{\textbf{Fitness}}$ (10^{-4})	1,3267717	1,1759912	1,9326422	0,9616586	1,0208799	1,2559703	1,3558809
mittlerer $E_{\textbf{Fitness}}$ (10^{-4})	1,8458507	11,5930506	65,0865787	1,9434296	1,9108862	1,7670204	1,8771353
bester $E_{\textbf{Validierung}}$ (10^{-4})	1,4440179	1,0114728	2,0871137	0,9375716	1,1870976	2,0412082	1,9840382
mittlerer $E_{\textbf{Validierung}}$ (10^{-4})	2,2010243	12,0392795	66,2689688	2,3857971	2,3029355	2,114507	2,2251658
# Parameter best	5831127	4284658	2827420	13411776	2855729	7448770	19240998
# Parameter mean	5735825,5	5519138,15	5692094,4	10985660,6	7256376,2	7393359,7	13624841,4

Tabelle A.1: Ergebnisse der Untersuchten Hyperparametersätze. Zu sehen sind die Informationen der letzten Iteration.

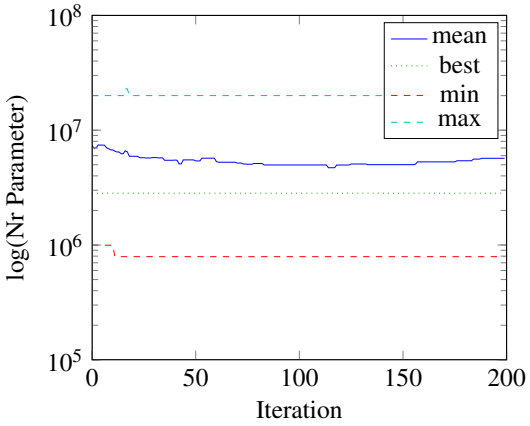
A.2 Hyperparametergruppe 1



A.3.1 Run 1 - Anzahl an Parametern.

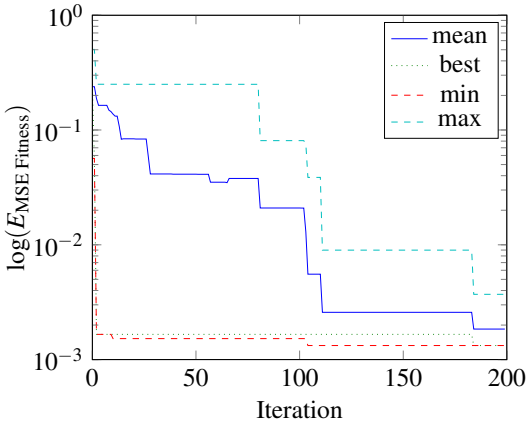


A.3.2 Run 2 - Anzahl an Parametern.

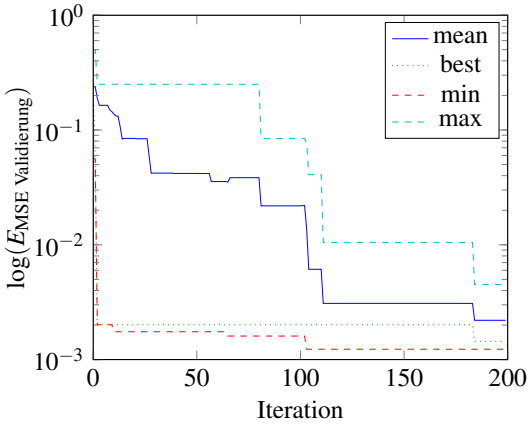


A.3.3 Run 3 - Anzahl an Parametern.

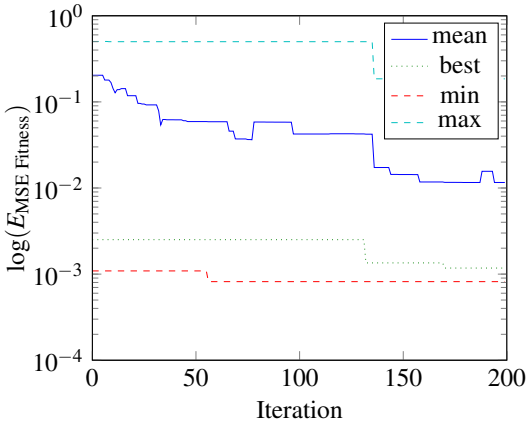
Abbildung A.3: Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke.



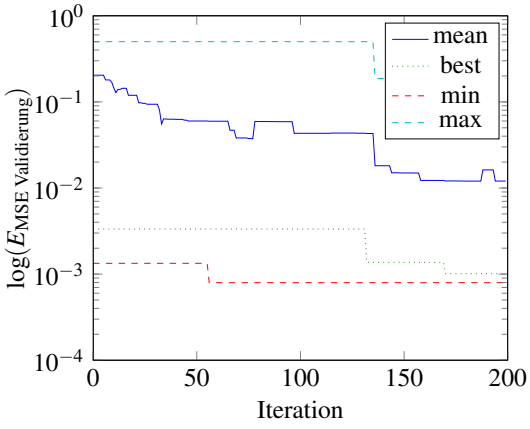
A.4.1 Run 1 - MSE Fitnessmenge.



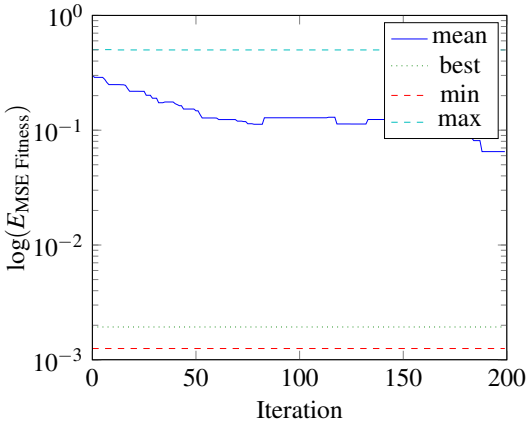
A.4.2 Run 1 - MSE Validierungsmenge.



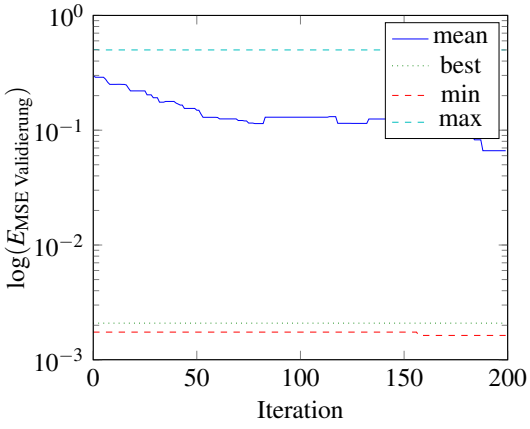
A.4.3 Run 2 - MSE Fitnessmenge.



A.4.4 Run 2 - MSE Validierungsmenge.

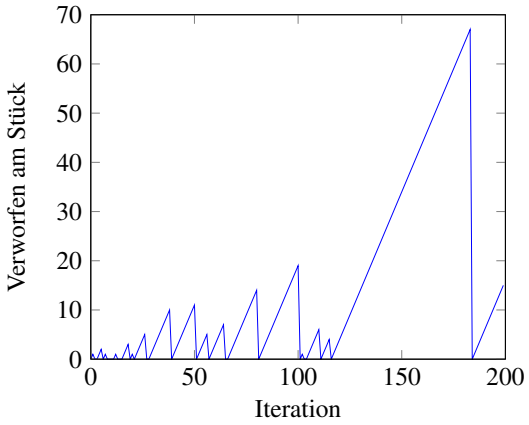


A.4.5 Run 3 - MSE Fitnessmenge.

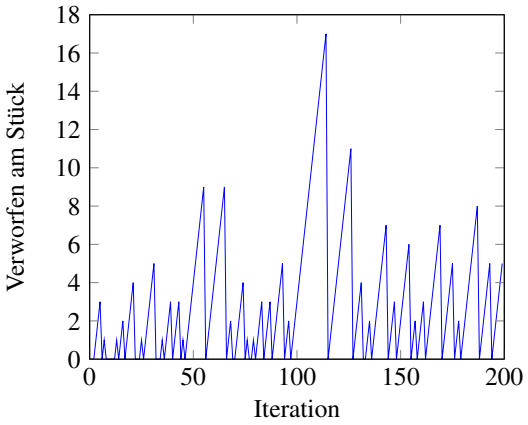


A.4.6 Run 3 - MSE Validierungsmenge.

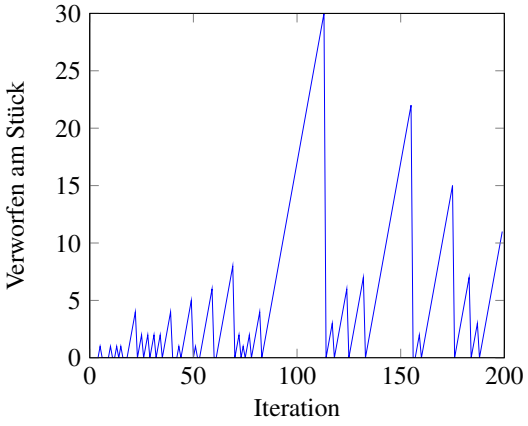
Abbildung A.4: Entwicklung der Fehler der Validierungs- und Fitnessmenge.



A.5.1 Run 1 - Verworfene Individuen.



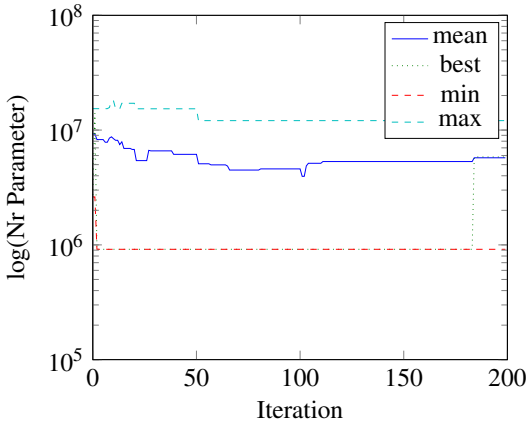
A.5.2 Run 2 - Verworfene Individuen.



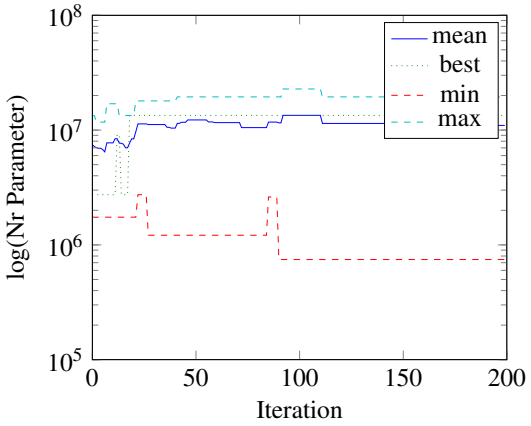
A.5.3 Run 3 - Verworfene Individuen.

Abbildung A.5: Verteilung der Anzahl an verworfenen Individuen am Stück pro Iteration.

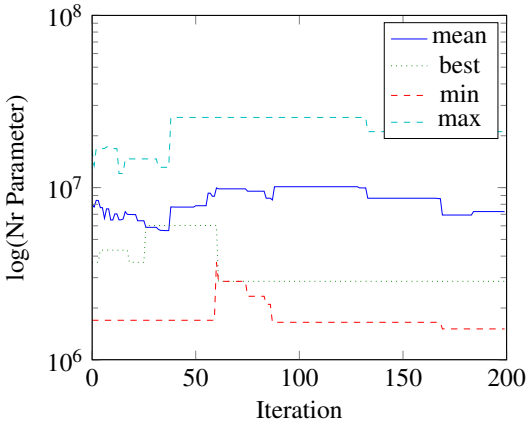
A.3 Hyperparametergruppe 2



A.6.1 Run 1 - Anzahl an Parametern.

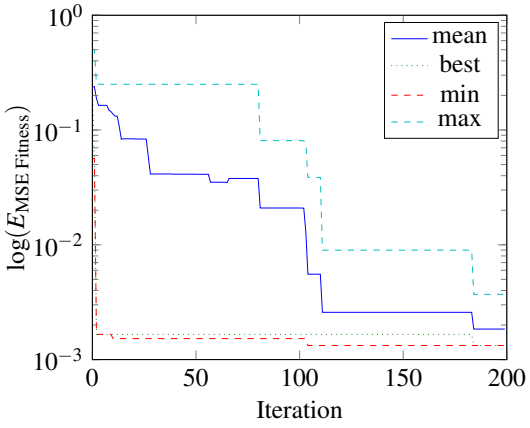


A.6.2 Run 4 - Anzahl an Parametern.

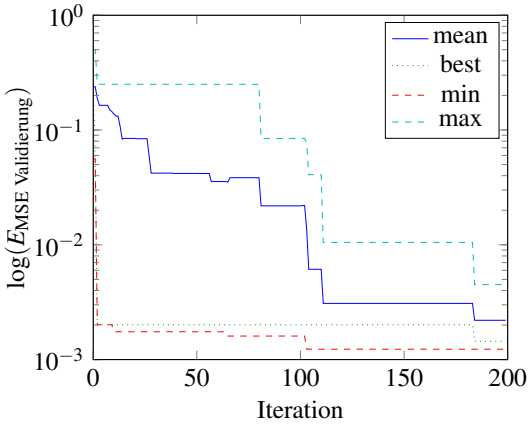


A.6.3 Run 5 - Anzahl an Parametern.

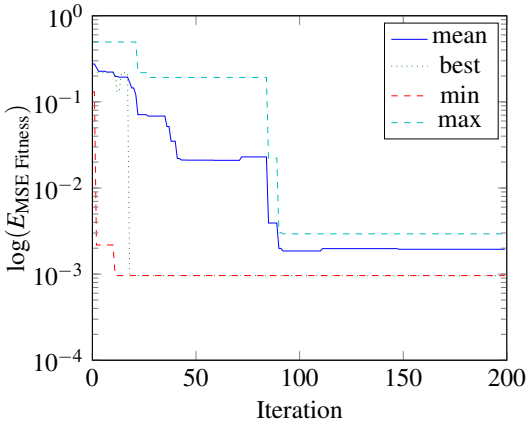
Abbildung A.6: Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke.



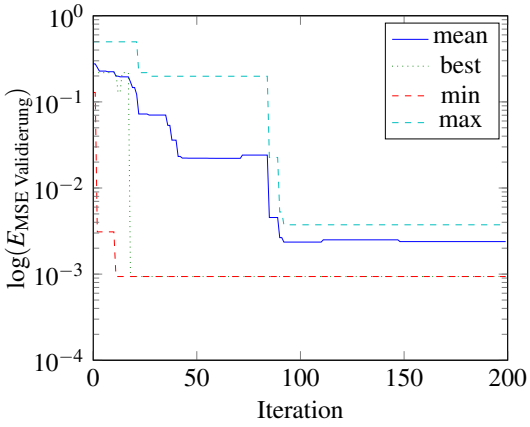
A.7.1 Run 1 - MSE Fitnessmenge.



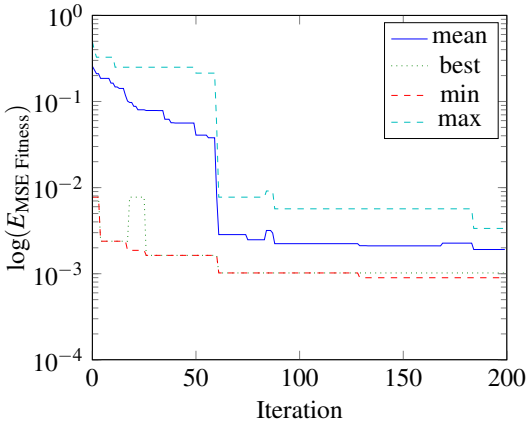
A.7.2 Run 1 - MSE Validierungsmenge.



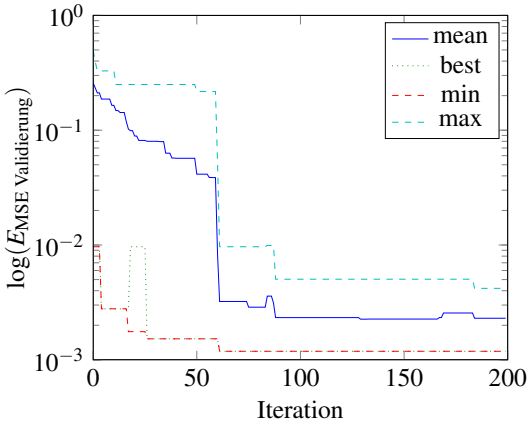
A.7.3 Run 4 - MSE Fitnessmenge.



A.7.4 Run 4 - MSE Validierungsmenge.

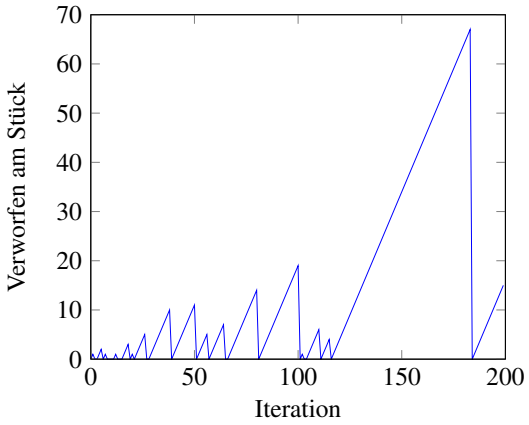


A.7.5 Run 5 - MSE Fitnessmenge.

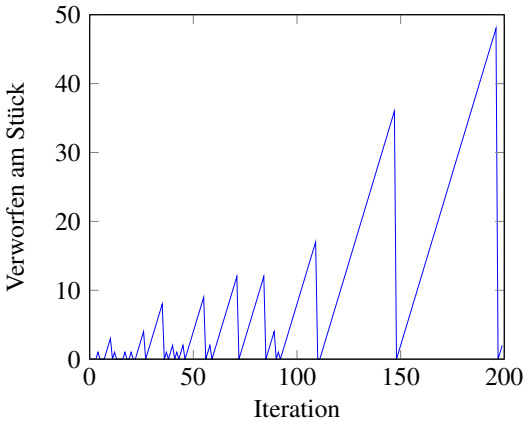


A.7.6 Run 5 - MSE Validierungsmenge.

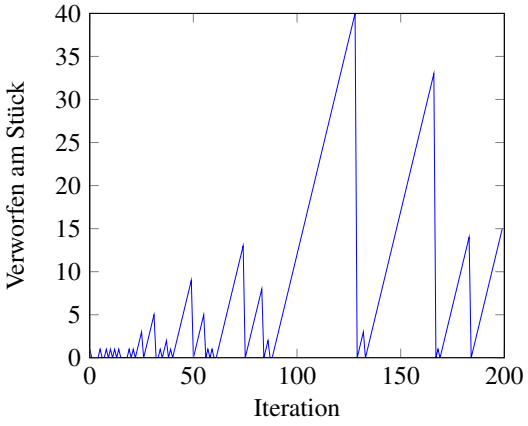
Abbildung A.7: Entwicklung der Fehler der Validierungs- und Fitnessmenge.



A.8.1 Run 1 - Verworfene Individuen.



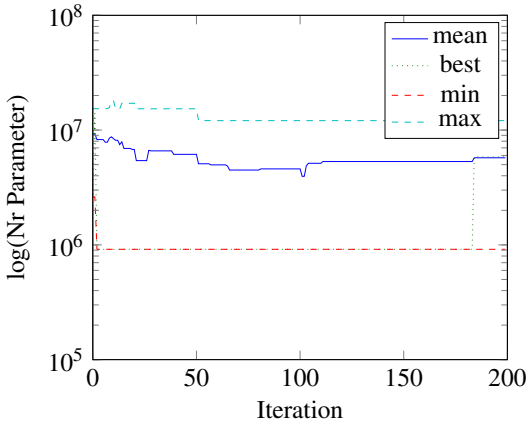
A.8.2 Run 4 - Verworfene Individuen.



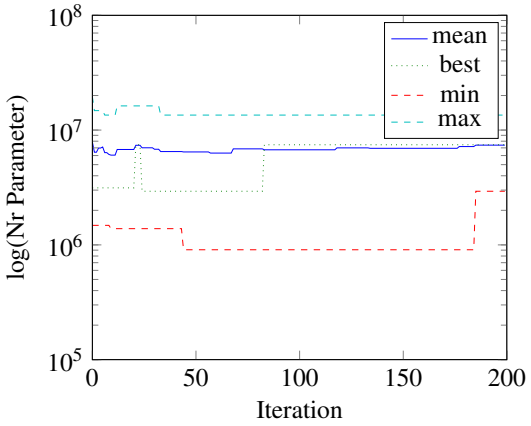
A.8.3 Run 5 - Verworfene Individuen.

Abbildung A.8: Verteilung der Anzahl an verworfenen Individuen am Stück pro Iteration.

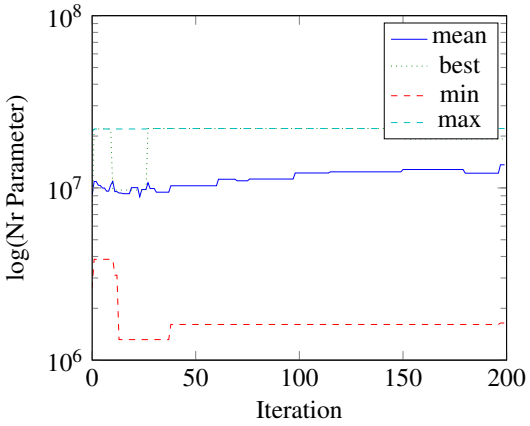
A.4 Hyperparametergruppe 3



A.9.1 Run 1 - Anzahl an Parametern.

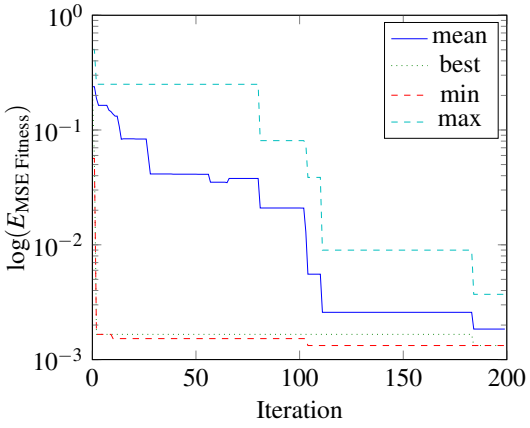


A.9.2 Run 6 - Anzahl an Parametern.

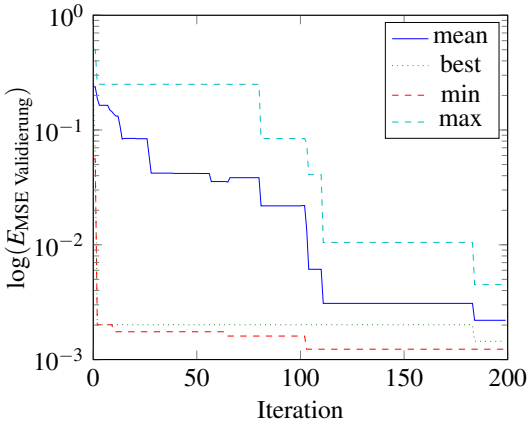


A.9.3 Run 7 - Anzahl an Parametern.

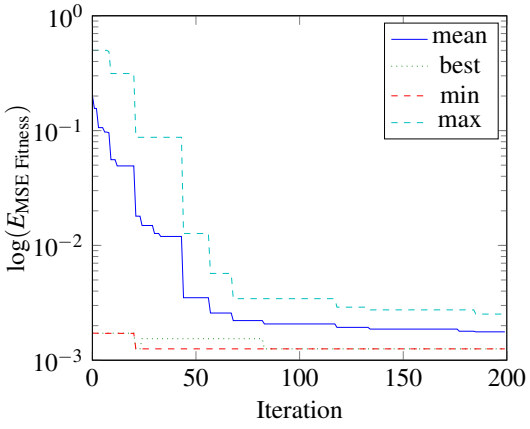
Abbildung A.9: Entwicklung der Anzahl an Parametern der Neuronalen Netzwerke.



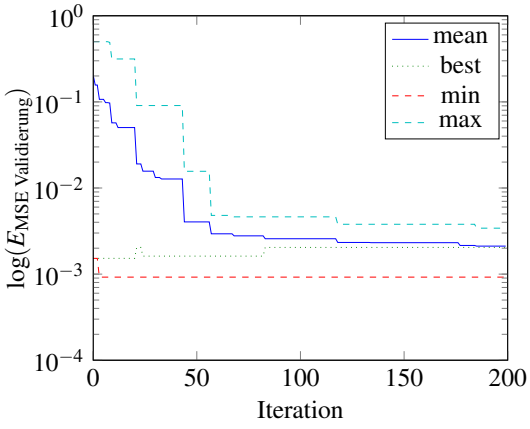
A.10.1 Run 1 - MSE Fitnessmenge.



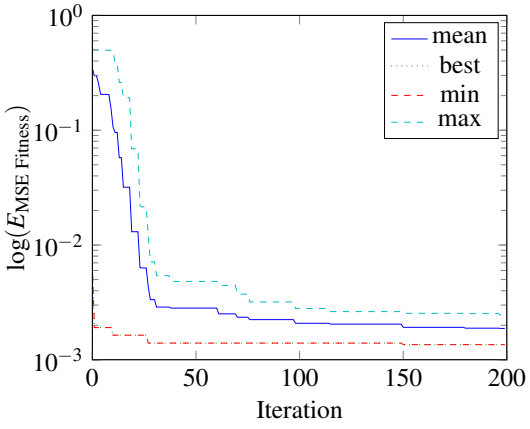
A.10.2 Run 1 - MSE Validierungsmenge.



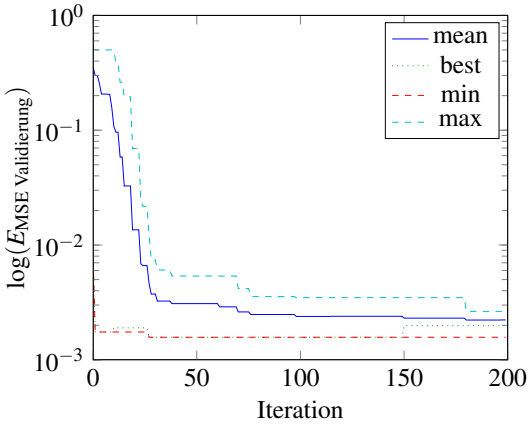
A.10.3 Run 6 - MSE Fitnessmenge.



A.10.4 Run 6 - MSE Validierungsmenge.

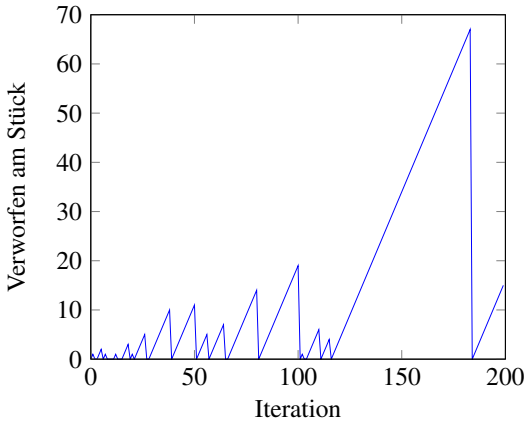


A.10.5 Run 7 - MSE Fitnessmenge.

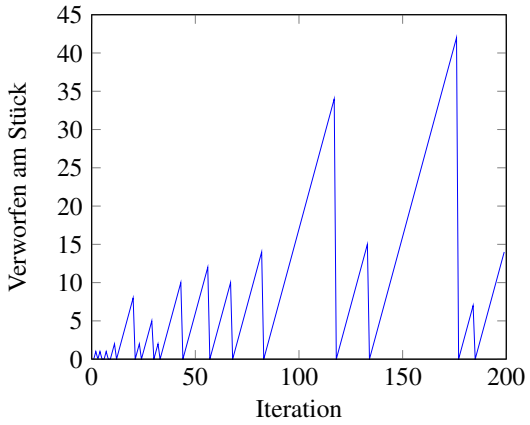


A.10.6 Run 7 - MSE Validierungsmenge.

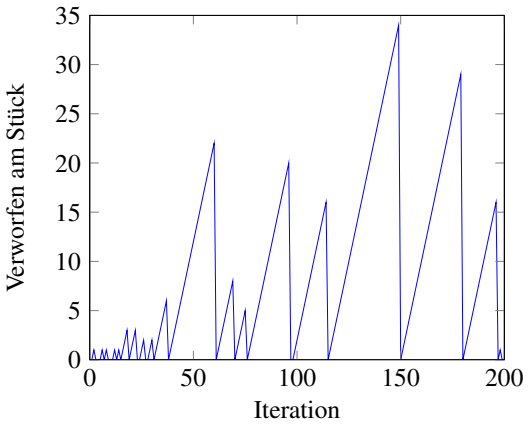
Abbildung A.10: Entwicklung der Fehler der Validierungs- und Fitnessmenge.



A.11.1 Run 1 - Verworfene Individuen.



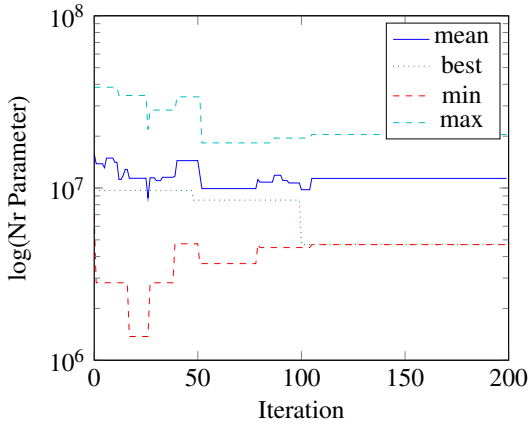
A.11.2 Run 6 - Verworfene Individuen.



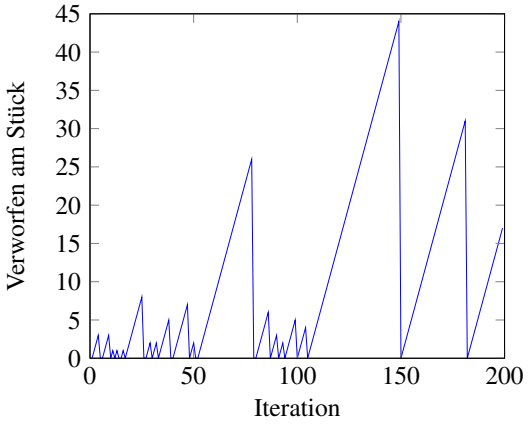
A.11.3 Run 7 - Verworfene Individuen.

Abbildung A.11: Verteilung der Anzahl an verworfenen Individuen am Stück pro Iteration.

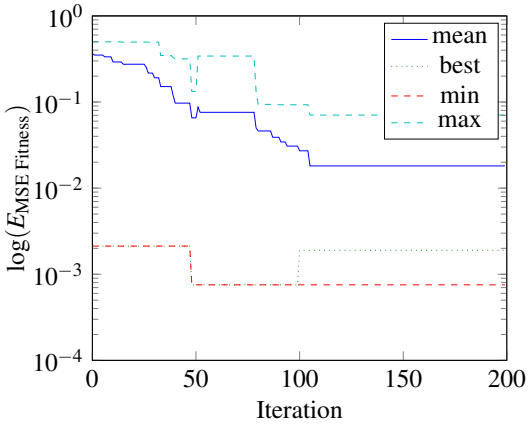
A.5 Finaler Hyperparametersatz



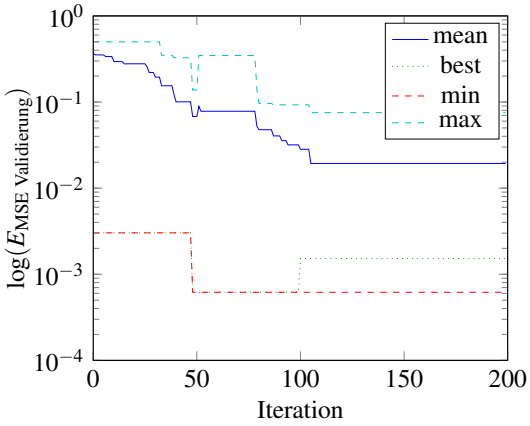
A.12.1 Run 8 - Anzahl an Parametern.



A.12.2 Run 8 - Verworfene Individuen.



A.12.3 Run 8 - MSE Fitnessmenge.



A.12.4 Run 8 - MSE Validierungssmenge.

Abbildung A.12: Ergebnisse des evolutionären Algorithmus mit dem finalen Hyperparametersatz.

Anhang B

Klassifizierungsergebnisse der finalen Neuronalen Netzwerke

B.1.1
$p = 0{,}0422602$

B.1.2
$p = 0{,}0490644$

B.1.3
$p = 0{,}0689936$

B.1.4
$p = 0{,}107405$

B.1.5
$p = 0{,}110463$

B.1.6
$p = 0{,}115113$

B.1.7
$p = 0{,}12994$

B.1.8
$p = 0{,}140034$

B.1.9
$p = 0{,}148764$

B.1.10
$p = 0{,}157531$

B.1.11
$p = 0{,}167656$

B.1.12
$p = 0{,}187715$

B.1.13
$p = 0{,}191121$

B.1.14
$p = 0{,}200197$

B.1.15
$p = 0{,}200989$

B.1.16
$p = 0{,}209651$

B.1.17
$p = 0{,}241209$

B.1.18
$p = 0{,}25298$

B.1.19
$p = 0{,}264622$

B.1.20
$p = 0{,}285391$

B.1.21
$p = 0{,}312325$

B.1.22
$p = 0{,}33487$

B.1.23
$p = 0{,}360503$

B.1.24
$p = 0{,}371881$

B.1.25
$p = 0{,}373508$

B.1.26
p = 0,399251

B.1.27
p = 0,428321

B.1.28
p = 0,435738

B.1.29
p = 0,469294

B.1.30
p = 0,473614

B.1.31
p = 0,477887

B.1.32
p = 0,480465

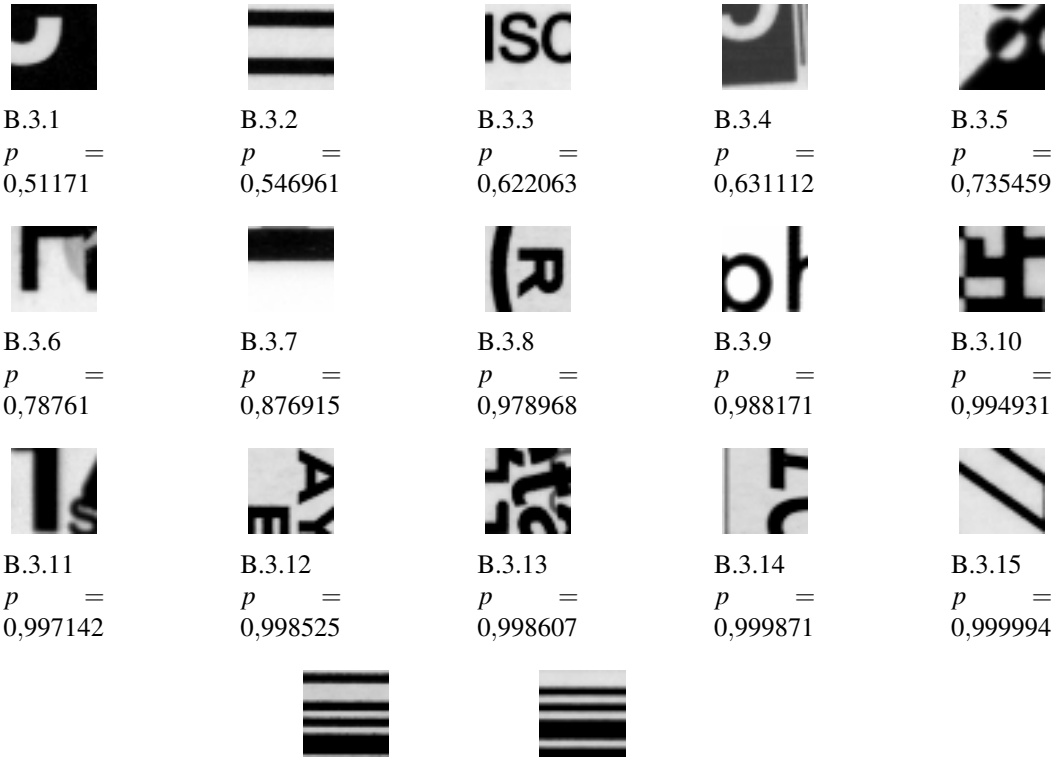
B.1.33
p = 0,482565

Abbildung B.1: Falschdetektionen beim synthetischen Datensatz; p gibt den Ausgabewert des Netzwerks an. Alle Bildausschnitte wurden fälschlicherweise als DataMatrix-Codes erkannt.

B.2.1
p = 0,720756

B.2.2
p = 0,968923

Abbildung B.2: Falschdetektionen beim synthetischen Datensatz; p gibt den Ausgabewert des Netzwerks an. Alle Bildausschnitte wurden fälschlicherweise nicht als DataMatrix-Codes erkannt.



B.3.1 $p = 0{,}51171$ | B.3.2 $p = 0{,}546961$ | B.3.3 $p = 0{,}622063$ | B.3.4 $p = 0{,}631112$ | B.3.5 $p = 0{,}735459$

B.3.6 $p = 0{,}78761$ | B.3.7 $p = 0{,}876915$ | B.3.8 $p = 0{,}978968$ | B.3.9 $p = 0{,}988171$ | B.3.10 $p = 0{,}994931$

B.3.11 $p = 0{,}997142$ | B.3.12 $p = 0{,}998525$ | B.3.13 $p = 0{,}998607$ | B.3.14 $p = 0{,}999871$ | B.3.15 $p = 0{,}999994$

B.3.16 $p = 0{,}999998$ | B.3.17 $p = 1{,}0$

Abbildung B.3: Falschdetektionen beim Kundendatensatz; p gibt den Ausgabewert des Netzwerks an. Alle Bildausschnitte wurden fälschlicherweise als DataMatrix-Codes erkannt.

B.4.1
p = 0,0144562

B.4.2
p = 0,0169206

B.4.3
p = 0,0406004

B.4.4
p = 0,0786238

B.4.5
p = 0,112864

B.4.6
p = 0,197883

B.4.7
p = 0,214204

B.4.8
p = 0,234971

B.4.9
p = 0,366414

B.4.10
p = 0,398865

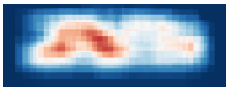
B.4.11
p = 0,454434

Abbildung B.4: Falschdetektionen beim Kundendatensatz; p gibt den Ausgabewert des Netzwerks an. Alle Bildausschnitte wurden fälschlicherweise nicht als DataMatrix-Codes erkannt.

Anhang C

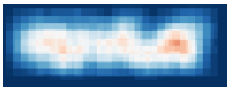
Detektionsalgorithmus

C.1.1 Testbild Nr. 39 (1) - DMC.



C.1.2 Testbild Nr. 39 (1) - Heatmap.

C.1.3 Testbild Nr. 39 (2) - DMC.

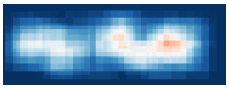


C.1.4 Testbild Nr. 39 (2) - Heatmap.

C.1.5 Testbild Nr. 39 (3) - DMC.

C.1.6 Testbild Nr. 39 (3) - Heatmap.

C.1.7 Testbild Nr. 39 (4) - DMC.



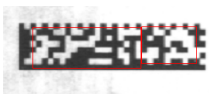
C.1.8 Testbild Nr. 39 (4) - Heatmap.

C.1.9 Testbild Nr. 39 (5) - DMC.

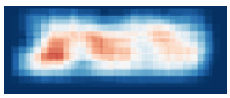
C.1.10 Testbild Nr. 39 (5) - Heatmap.

C.1.11 Testbild Nr. 39 (6) - DMC.

C.1.12 Testbild Nr. 39 (6) - Heatmap.

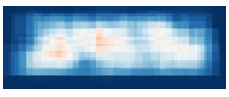


C.1.13 Testbild Nr. 39 (7) - DMC.



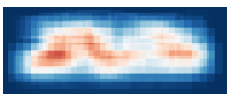
C.1.14 Testbild Nr. 39 (7) - Heatmap.

C.1.15 Testbild Nr. 39 (8) - DMC.

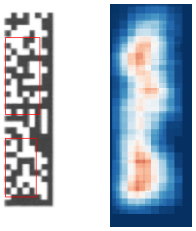


C.1.16 Testbild Nr. 39 (8) - Heatmap.

C.1.17 Testbild Nr. 39 (9) - DMC.



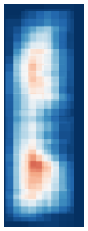
C.1.18 Testbild Nr. 39 (9) - Heatmap.



C.1.19 Testbild Nr. 39 (10) - DMC.

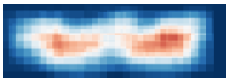
C.1.20 Testbild Nr. 39 (10) - Heatmap.

C.1.21 Testbild Nr. 39 (11) - DMC.



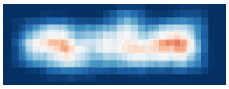
C.1.22 Testbild Nr. 39 (11) - Heatmap.

C.1.23 Testbild Nr. 39 (12) - DMC.



C.1.24 Testbild Nr. 39 (12) - Heatmap.

C.1.25 Testbild Nr. 39 (13) - DMC.



C.1.26 Testbild Nr. 39 (13) - Heatmap.

Abbildung C.1: Bildausschnitte, die die gesondert zu behandelnden DataMatrix-Codes beinhalten.

Literaturverzeichnis

[1] International Organization for Standardization. ISO/IEC 16022:2006: Information technology - Automatic identification and data capture techniques - Data Matrix bar code symbology specification. https://www.iso.org/obp/ui/#iso:std:44230:en, 2006.

[2] Ilteris Oney. GS1 DataMatrix: An Introduction and Implementation Guideline. 2015.

[3] Sean Owen. ZXing: ZXing ("zebra crossing") is an open-source, multi-format 1D/2D barcode image processing library implemented in Java, with ports to other languages. https://github.com/zxing, 2007.

[4] Google. Barcode Scanner. https://play.google.com/store/apps/details?id=com.google.zxing.client.android&hl=de, 2015.

[5] Gábor Sörös and Christian Flörkemeier. Blur-resistant joint 1D and 2D barcode localization for smartphones. In Matthias Kranz, Kåre Synnes, Sebastian Boring, and Kristof van Laerhoven, editors, *the 12th International Conference*, pages 1–8, 2013.

[6] M. Katona and L. G. Nyul. A Novel Method for Accurate and Efficient Barcode Detection with Morphological Operations. In *2012 Eighth International Conference on Signal-Image Technology & Internet-Based Systems (SITIS 2012)*, pages 307–314.

[7] Qiang Huang, Wen-Sheng Chen, Xiao-Yan Huang, and Ying-Ying Zhu. Data Matrix Code Location Based on Finder Pattern Detection and Bar Code Border Fitting. *Mathematical Problems in Engineering*, 2012:1–13, 2012.

[8] Tamás Grósz, Péter Bodnár, László Tóth, László G. Nyúl. QR Code Localization Using Deep Neural Networks. 2014.

[9] P. Bodnar and L. G. Nyul. Improving Barcode Detection with Combination of Simple Detectors. In *2012 Eighth International Conference on Signal-Image Technology & Internet-Based Systems (SITIS 2012)*, pages 300–306.

[10] Jan Klass. Neuronale Netze der 3. Generation und Anwendungsgebiete. 2010.

[11] Fredrik Lundh. Python Imaging Library. http://effbot.org/.

[12] Ching-Yung Lin, Shih-Fu Chang. Distortion Modeling and Invariant Extraction for Digital Image Print-and-Scan Process. 1999.

[13] Kristian Bredies and Dirk Lorenz. *Mathematische Bildverarbeitung: Einführung in Grundlagen und moderne Theorie*. Studium. Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden, Wiesbaden, Wiesbaden, 2011.

[14] Ulrike Genschel and Claudia Becker. *Schließende Statistik: Grundlegende Methoden*. EMILA-stat. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2005.

[15] Warren S. McCulloch and Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[16] David Kriesel. Ein kleiner Überblick über Neuronale Netze. http://www.dkriesel.com, 2007.

[17] L. Fei-Fei, Andrej Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.stanford.edu/, 2015.

[18] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, New York, 2006.

[19] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *ICASSP 2013 - 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8624–8628, 2013.

[20] Lasagne contributors. Lasagne Dokumentation: Release 0.2.dev1. http://lasagne.readthedocs.org, 2015.

[21] S. Forrest. Genetic algorithms: Principles of natural selection applied to computation. *Science*, 261(5123):872–878, 1993.

[22] C. Darwin. *On the Origin of Species by Means of Natural Selection, Or, The Preservation of Favoured Races in the Struggle for Life*. 1859.

[23] David White and Panos Ligomenides. GANNet: A genetic algorithm for optimizing topology and weights in neural network design. In José Mira, Joan Cabestany, and Alberto Prieto, editors, *New Trends in Neural Computation*, volume 686 of *Lecture Notes in Computer Science*, pages 322–327. Springer Berlin Heidelberg, 1993.

[24] Darrell Whitley. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, 1989.

[25] A. Blanco, M. Delgado, and M. C. Pegalajar. A genetic algorithm to obtain the optimal recurrent neural network. *International Journal of Approximate Reasoning*, 23(1):67–83, 2000.

[26] LISA laboratory, University of Montreal. Theano. `http://deeplearning.net/software/theano/#`, 2015.

[27] Les E. Atlas, Toshiteru Homma, Robert J. Marks II. An Artificial Neural Network for Spatio-Temporal Bipolar Patterns: Application to Phoneme Classification. *Neural Information Processing Systems Conference (AIP)*, 1987.

[28] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis, editors, *Artificial Neural Networks – ICANN 2010*, volume 6354 of *Lecture Notes in Computer Science*, pages 92–101. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15 (2014)*, 2014.

[30] Fabian Beck, Günter Daniel Rey. Neuronale Netzwerke - Eine Einführung. `http://www.neuronalesnetz.de/`.

[31] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *IN PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND STATISTICS (AISTATS'10). SOCIETY FOR ARTIFICIAL INTELLIGENCE AND STATISTICS*, 2010.

[32] Andrew L Maas, Awni Y Hannun, Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. *Proc. ICML*, (30), 2013.

[33] Andrew Jones. An Explanation of Xavier Initialization. `http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization`, 2015.

[34] Daniel Nouri. Using convolutional neural nets to detect facial keypoints tutorial. http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/, 2014.

[35] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. 2012.

[36] Python Software Foundation. Python 2.7.10 documentation. https://docs.python.org/2/index.html.

[37] The Scipy community. NumPy v1.10 Manual. http://docs.scipy.org/doc/numpy/index.html.

[38] R. N. Greenwell, J. E. Angus, and M. Finck. Optimal mutation probability for genetic algorithms. *Mathematical and Computer Modelling*, 21(8):1–11, 1995.

[39] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. http://caffe.berkeleyvision.org/, 2014.