

**Erkennen von Objektmerkmalen in Wimmelbildern
mit Hilfe von neuronalen Netzen**

**STUDIENARBEIT
über die vierte und sechste Theoriephase**

des Studienganges Informationstechnik
der Fakultät Technik

an der Dualen Hochschule Baden-Württemberg Ravensburg
Campus Friedrichshafen

von

Mathias Brax und Nikolai Klatt

19. Juli 2021

Bearbeitungszeitraum	5. und 6. Semester
Matrikelnummer, Kurs	2340604, TIS-18 9353424, TIT-18
Betreuerin der Dualen Hochschule	Claudia Zinser

Eidesstattliche Erklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 27.07.2020:

Wir versichern hiermit, dass wir die Studienarbeit mit dem Titel:

Erkennen von Objektmerkmalen in Wimmelbildern mit Hilfe von neuronalen Netzen

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Mathias Brax

Nikolai Klatt

Kurzfassung

Die Themen künstliche Intelligenz, Machine Learning und neuronale Netze polarisieren seit über einem Jahrzehnt die Informationstechnik. Mit dem Aufkommen von mehr Anwendungen, die durch neuronale Netze besser als mit konventionellen Methoden gelöst werden können, wächst deren Adaption. Dadurch werden immer wieder neue Ansätze entwickelt, die aus der Verbesserung alter Ansätze hervorgehen. Die vorliegende Arbeit zeigt die medizinische Inspiration und die Grundlagen neuronaler Netze, sowie die Grundelemente des Machine Learning auf. Zudem wird evaluiert, inwiefern neuronale Netze für die Bilderkennung geeignet sind. Dies wird am Beispiel des Wimmelbildsuchspiels „Wo ist Walter?“ getestet. Für die Evaluation wurde das Framework YOLOv5, welches auf PyTorch basiert genutzt. Als Daten wurden Bilder aus den offiziellen „Wo ist Walter?“-Büchern genommen. Um die Größe des Datensatzes zu erhöhen, wurden mit einem Algorithmus neue Bilder erzeugt, bei denen Walter auf einen Hintergrund eingefügt wurde. Mit diesen Daten wurden verschieden große neuronale Netze mit unterschiedlichen Parametern trainiert. Zu den variablen Parametern zählen: die Image-Size mit der trainiert wurde, die Epochenzahl der Trainings, die Batch-Size und die Lernrate. Die Trainings und Tests zeigen, dass die neuronalen Netze grundsätzlich in der Lage sind Walter zu erkennen. Er wird allerdings nicht immer erkannt. Außerdem erkennen die verschiedenen trainierten neuronalen Netze Walter unterschiedlich gut und mit unterschiedlicher Sicherheit. Das Ergebnis hängt im wesentlichen von der Qualität der genutzten Trainingsdaten und gut gewählten Parametern im Training ab.

Abbildungsverzeichnis

1	Walter	2
2	Sagittalschnitt durch das Gehirn	4
3	Vereinfachte Übersicht eines Neurons	5
4	Beispiel Underfitting, gute Generalisierung und Overfitting	13
5	Exemplarisches Modell eines künstlichen Neurons	14
6	Exemplarischer Aufbau der Layer-Struktur eines neuronalen Netzes . .	16
7	Exemplarischer Aufbau der Layer-Struktur eines Convolutional Neural Network mit Feature Maps	18
8	Ablauf des Trainingsprozesses	19
9	Stochastic Gradient Descent und Auswirkung der Lernrate	20
10	Graphische Repräsentation der Intersection over Union Metrik	22
11	Workflow Trainingsprozess	32
12	Zusammengefasste Ergebnisse der Generationen	52

Abkürzungsverzeichnis

BP	Backward Propagation
CNN	Convolutional Neural Network
DL	Deep Learning
FFNN	Feed-Forward Neural Network
FP	Forward Propagation
GPU	Graphics Processing Unit
HDD	Hard Disk Drive
IoU	Intersection over Union
KI	Künstliche Intelligenz
LTD	Langzeitdepression
LTP	Langzeitpotenzierung
mAP	Mean Average Precision
ML	Machine Learning
NN	Neuronales Netz
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SL	Supervised Learning
SSD	Solid State Drive
UL	Unsupervised Learning
YOLOv5	You Only Look Once Version 5

Tabellenverzeichnis

1	Vortrainierte Models	30
2	Parameter Generation 1	33
3	Ergebnisse Generation 1	33
4	Parameter Generation 2a	35
5	Ergebnisse Generation 2a	35
6	Parameter Generation 2b	36
7	Ergebnisse Generation 2b	37
8	Parameter Generation 2c	37
9	Ergebnisse Generation 2c	38
10	Parameter Generation 3a	39
11	Ergebnisse Generation 3a	39
12	Parameter Generation 3b	40
13	Ergebnisse Generation 3b	41
14	Parameter Generation 3c	42
15	Ergebnisse Generation 3c	42
16	Parameter Generation 4a	44
17	Ergebnisse Generation 4a	44
18	Parameter Generation 4b	45
19	Ergebnisse Generation 4b	45
20	Parameter Generation 4c	46
21	Ergebnisse Generation 4c	46
22	Parameter Zusatzgeneration Lernrate erhöht	47
23	Ergebnisse Zusatzgeneration Lernrate erhöht	47
24	Parameter Zusatzgeneration Wilma	48
25	Ergebnisse Zusatzgeneration Wilma 50 Epochen	48
26	Ergebnisse Zusatzgeneration Wilma 250 Epochen	49
27	Ergebnisse Zusatzgeneration Wilma 1000 Epochen	49
28	Parameter Zusatzgeneration Epochenvergleich	50
29	Ergebnisse Zusatzgeneration Epochenvergleich - 16 Epochen	50
30	Ergebnisse Zusatzgeneration Epochenvergleich - 30 Epochen	51
31	Ergebnisse Zusatzgeneration Epochenvergleich - 50 Epochen	51

Inhaltsverzeichnis

Eidesstattliche Erklärung	I
Kurzfassung	II
Abbildungsverzeichnis	III
Abkürzungsverzeichnis	IV
Tabellenverzeichnis	V
1 Einführung	1
1.1 Motivation	1
1.2 Aufgabenstellung	2
1.3 Eingrenzung des Umfangs	2
2 Grundlagen	3
2.1 Medizinische Grundlagen	3
2.1.1 Funktionsweise des Gehirns	3
2.1.2 Lernen	6
2.2 Mathematische Grundlagen	7
2.3 Machine Learning	10
2.3.1 Lernalgorithmus	10
2.3.2 Arten des Lernens	11
2.3.3 Overfitting und Underfitting	12
2.3.4 Regularisierung	13
2.4 Neuronale Netze	14
2.4.1 Architektur	14
2.4.2 Training	19
2.4.3 Performance-Metriken	21
2.5 Datenvorbereitung und Datenaufbereitung	23
2.6 Etablierte Programmiersprachen für Neuronale Netze	25
3 Projektvorbereitung	26
3.1 Programmiersprache, Libraries und Entwicklungsumgebung	26
3.2 Konzeption und Vorgehensweise	28

4 Analyse YOLOv5	29
4.1 Ursprung	29
4.2 Verwendbarkeit	29
4.3 Architektur	30
4.4 Funktion	31
4.5 Geschwindigkeit	31
5 Training und Validierung des Neuronalen Netzes	32
5.1 Workflow	32
5.2 Generation 1	33
5.3 Generation 2	35
5.4 Generation 3	39
5.5 Generation 4	44
5.6 Zusatzgenerationen	47
5.6.1 Lernrate erhöht	47
5.6.2 Wilma	48
5.6.3 Epochen-Vergleich	50
6 Schlussbetrachtung	52
6.1 Fazit	52
6.2 Kritische Auseinandersetzung	53
6.3 Ausblick	54
Literaturverzeichnis	IX
A Labelformat YOLOv5	XVI
B Programmablaufplan Image_Label_Creator	XVII
C Programmablaufplan Train_Val_Test_randomizer	XVIII
D Bildsammlung	XIX
D.1 Walter und Wilma werden erkannt	XIX
D.2 Walter False-Positive	XIX
D.3 Walter Rotationsproblem	XIX
D.4 Walter Kontrastproblem	XX
D.5 Walter Hochkant- und gestreifte Objekte Problem	XX

E Metriken Generation 1	XXI
F Metriken Generation 2	XXII
F.1 Generation 2a	XXII
F.2 Generation 2b	XXIII
F.3 Generation 2c	XXIV
G Metriken Generation 3	XXV
G.1 Generation 3a	XXV
G.2 Generation 3b	XXVI
G.3 Generation 3c	XXVII
H Metriken Generation 4	XXVIII
H.1 Generation 4a	XXVIII
H.2 Generation 4b	XXIX
H.3 Generation 4c	XXX
I Metriken Zusatzgeneration	XXXI
I.1 Lernrate erhöht	XXXI
I.2 Walter und Wilma erkennen	XXXII
I.2.1 50 Epochen	XXXII
I.2.2 250 Epochen	XXXIII
I.2.3 1000 Epochen	XXXIV
I.3 Epochenvergleich	XXXV
I.3.1 16 Epochen	XXXV
I.3.2 30 Epochen	XXXVI
I.3.3 50 Epochen	XXXVII

1 Einführung

1.1 Motivation

Die Themen Künstliche Intelligenz (KI) und Machine Learning (ML) sind in den letzten 15 Jahren sehr populär geworden, die Idee hinter der Entwicklung einer menschenähnlich denkenden Maschine liegt jedoch sehr viel weiter zurück. KI ist definiert als Technologie bzw. Computerprogramm, das mit menschenähnlicher Intelligenz Probleme lösen kann [1]. ML ist definiert als die Fähigkeit eines KI-Systems, Wissen aus repräsentativen Daten zu erlangen [2, S. 2]. Die Idee eines Systems, welches menschliches Denken formalisieren und imitieren kann, gab es bereits in der griechischen Mythologie. Formalisiert wurde das moderne Konzept von KI 1950 durch Alan Turing [3]. Er entwickelte seinen berühmten Turing-Test, in welchem ein Mensch durch eine Reihe von Fragen bestimmen soll, ob der Antwortende, auch ein Mensch oder eine Maschine ist. Der erste KI-Algorithmus, genannt „Logic Theorist“, wurde 1956 von John McCarthy, Marvin Minsky, Nathaniel Rochester and Claude E. Shannon vorgestellt [4]. Der Logic Theorist sollte die Fähigkeiten eines Menschen bezüglich Problemlösung imitieren. Zunächst konnte sich KI nicht wirklich etablieren, dies hing vor allem damit zusammen, dass Rechenleistung sehr teuer war [5]. Mit fortschreitender Entwicklung der Technik, wurden die Computer schneller und konnten viel mehr Daten verarbeiten [6]. Diese Entwicklung beflogelte auch die KI-Technologie. Es dauerte dennoch bis 1997, dass eine KI erstmals einem Menschen überlegen war, indem der von IBM entwickelte Computer IBM Deep Blue den Schach-Weltmeister Garri Kasparow besiegte [7]. Bereits 2011 gewann der von IBM entwickelte Computer „IBM Watson“ das Spiel Jeopardy gegen die besten Spieler Ken Jennings und Brad Rutter [8]. Lediglich fünf Jahre später gewann die von Google entwickelte KI AlphaGo gegen den mehrfachen Go Weltmeister Lee Sedol [9]. Der Trend entwickelt sich in die Richtung, dass KI viele Aufgaben der Menschheit zunächst unterstützt und letztendlich vollständig übernimmt. Bisher sind KIs in vielen Feldern dem Menschen unterlegen [10]. Vor allem in der Klassifizierung und Lokalisierung von Objekten in Bildern werden allerdings immer wieder Fortschritte erzielt. Für die Erkennung von Bildern werden derzeit Netze aus künstlichen Neuronen, ein sog. Neuronales Netz (NN), genutzt [10]. Diese NNs bieten viele Möglichkeiten und es gibt viele Ansätze, um Objekte in Bildern zu erkennen. Um diese zukunftsorientierte Technologie weiter voran zu bringen, ist es allerdings notwendig diese Möglichkeiten immer wieder zu überprüfen und zu testen, um so neue Ansätze zu entdecken oder bestehende weiterzuentwickeln.

1.2 Aufgabenstellung

Wimmelbilder haben einen hohen Detailgrad, das heißt, es gibt viele unterschiedliche Figuren und Objekte, welche chaotisch, aber dennoch mit Struktur, angeordnet sind [11, S. 102]. Die Objekte in Wimmelbildern sind teilweise verdeckt, skaliert oder farblich vom Hintergrund schwer zu unterscheiden. Die Aufgabe für den Betrachter der Wimmelbilder ist es, die gesuchten Objekte zu finden, was sich durch der genannten Eigenschaften als schwierig erweisen kann. Aufgrund der Schwierigkeiten, die beim Finden der Objekte auftreten, ist das Ziel dieser Arbeit die gesuchten Objekte virtuell erkennen und visualisieren zu lassen. Als technisches Hilfsmittel wird hierbei ein NN zum Einsatz kommen. Es wird evaluiert werden, inwiefern NNs für eine solche Aufgabe geeignet sind. Der Fokus liegt dabei auf der Analyse der Reaktion des NN auf unterschiedliche Datensätze und variierende Parameter. Als Wimmelbild-Datensatz wird „Wo ist Walter?“ (Abbildung 1) genutzt [12]. Für die Evaluation werden vortrainierte Modelle mit dem Datensatz trainiert und die Ergebnisse analysiert. Anhand dieser Ergebnisse werden die Trainingsparameter und der Datensatz angepasst.



Abbildung 1: Walter

1.3 Eingrenzung des Umfangs

Die vorliegende Arbeit dient nicht dazu verschiedene Frameworks für NNs zu vergleichen bzw. zu evaluieren. Das genutzte Framework wird lediglich in seiner grundsätzlichen Funktionsweise analysiert, um dessen Funktionen zu verstehen und sie anwenden zu können. Außerdem dient die vorliegende Arbeit nicht der tiefgehenden technischen und mathematischen Ausführung der Themen KI, ML / Deep Learning (DL) und NN. Die Themen werden zwar teilweise ausführlich, teilweise gekürzt erläutert, jedoch begrenzen sie sich auf grundlegende Informationen, die zum Verständnis der Arbeit hilfreich sind. Mit dieser Arbeit soll zudem kein theoretisches Modell zur Evaluierung der Eignung von NN für bestimmte Tasks erarbeitet werden.

2 Grundlagen

Um die Entwicklung eines NN zur Verarbeitung von Bilddateien, bzw. zum Klassifizieren und Lokalisieren eines Objektes in einem Bild durchzuführen, werden im Folgenden einige Grundlagen aufgeführt. Zunächst wird dabei der medizinische bzw. neurologische Aufbau des Gehirns erläutert, sowie auf den Prozess des Lernens eingegangen. Anschließend wird der mathematische Bezug zu NNs hergestellt. Daraufhin werden die Grundlagen von ML ausgeführt, sowie die Komponenten und der Aufbau von NNs erläutert. Dann wird auf die Anforderungen und technischen Herausforderungen bei der Datenvorbereitung eingegangen. Abschließend werden einige, für NNs etablierte, Programmiersprachen aufgeführt und deren Vorteile visualisiert.

2.1 Medizinische Grundlagen

Wie viele menschliche Errungenschaften sind auch NNs biologisch inspiriert [13]. Diese Inspiration wird im folgenden erläutert, sowie der Prozess des Lernens ausgeführt.

2.1.1 Funktionsweise des Gehirns

Das Gehirn bildet die Grundlage des menschlichen Denkens, Lernens und Handelns. Es wiegt durchschnittlich 1400 Gramm und macht damit ca. 2 % des Körpergewichts aus, gleichzeitig ist es allerdings für ca. 20 % des gesamten Energieverbrauchs des Körpers verantwortlich [14, S. 7]. Laut dem amerikanischen Professor für Psychiatrie und Verhaltensforschung Allen Frances [15] ist das Gehirn das „bei Weitem komplizierteste Gebilde im Universum“. Diese weitreichende Komplexität erlaubt es nicht vollumfänglich zu erläutern, wie das Gehirn funktioniert. Daher wird in dieser Arbeit lediglich ein grundätzliches Verständnis für die Funktionsweise des Gehirns geschaffen. Abbildung 2 auf der nächsten Seite zeigt einen Sagittalschnitt, also einen vertikalen Schnitt, durch die Mittelachse des Gehirns, und gibt damit eine Übersicht über die Bereiche des Gehirns. Wichtige Bestandteile des Gehirns sind dabei das Groß-, Zwischen- und Kleinhirn, der Hirnstamm, sowie die Großhirnrinde, der sog. Kortex [16]. Das Großhirn ist mit dem Kortex im wesentlichen für Motorik, Sensorik und Sprachverständnis zuständig und beinhaltet daher auch das Seh-, Hör- und Sprachzentrum. Dem Kortex, der unter anderem auch für das typische Aussehen eines Gehirns mit den Furchen und Windungen verantwortlich ist, werden auch höhere intellektuelle Fähigkeiten, wie Denken, Planen oder das Lösen von Problemen zugeschrieben. Das Zwischenhirn beinhaltet das Gedächtnis,

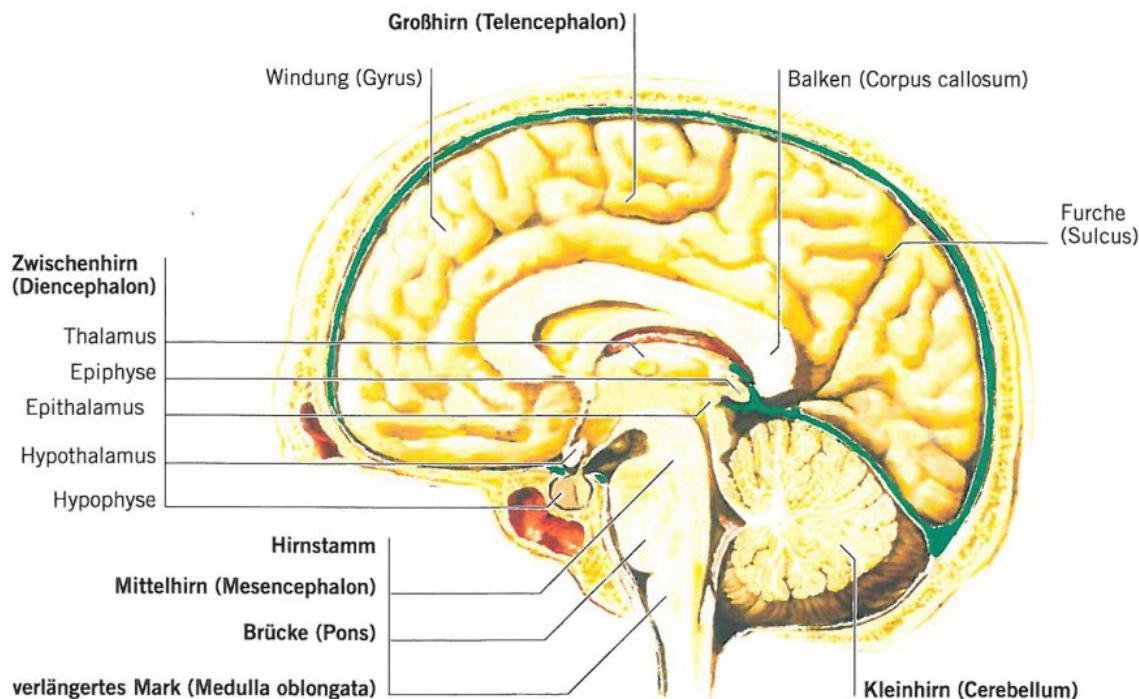


Abbildung 2: Sagittalschnitt durch das Gehirn [14, S. 11]

den Hippocampus, und bildet mit dem Thalamus und Hypothalamus eine Sammelstelle für fast alle Informationen, die in das Gehirn fließen. Außerdem werden hier vegetative Körperfunktionen, wie der Stoffwechsel oder die Verdauung gesteuert. Das Kleinhirn steuert motorische und kognitive Prozesse, bei denen ein Zeitbewusstsein benötigt wird. Der Hirnstamm unterstützt die Steuerung der vegetativen Körperfunktionen, beeinflusst den Herzrhythmus und leitet spezifische Informationen der Sinnesorgane weiter. [17] Die Informationen, die im Gehirn ankommen, sind elektrochemische Impulse. Die Übermittlung und Verarbeitung dieser Impulse wird durch das Nervensystem geregelt. Dieses besteht aus Nervenzellen, den sog. Neuronen, Synapsen und Gliazellen, wobei letztere die anderen Zellen stützen und für den Botenstoffaustausch zuständig sind. [18]

Neuronen und Synapsen

Das Gehirn besteht schätzungsweise aus 85 – 200 Milliarden ($0.85 – 2.0 \times 10^{11}$) Neuronen [18], wobei meistens eine Anzahl von 100 Milliarden Neuronen genannt wird [16], [19]. Zwischen den Neuronen gibt es Verbindungen, welche durch Synapsen abgebildet werden. Obwohl eine genaue Zahl der Synapsen bisher nicht belegt wurde, sieht die Wissenschaft

Schätzungen in Höhe von 100 Billionen – 1 Billiarde (10^{14} – 10^{15}) als realistisch [20]. Das würde bedeuten, dass jedes der Neuronen, durch die Synapsen, mit durchschnittlich 1.000 – 10.000 anderen Neuronen verbunden ist. Der Aufbau eines Neurons ist in Abbildung 3 zu sehen. Neuronen können vereinfacht in drei Regionen aufgeteilt werden.

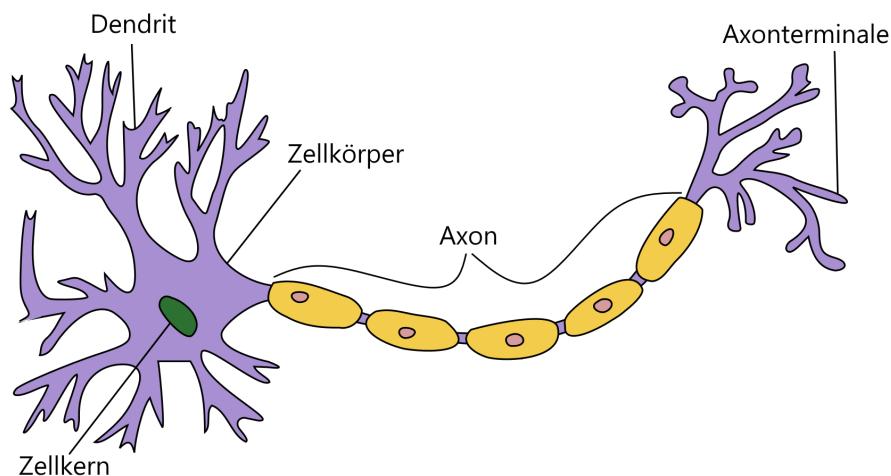


Abbildung 3: Vereinfachte Übersicht eines Neurons

Dendriten

Die Dendriten sind die Empfänger von Nervenimpulsen, die sie durch die Synapsen erhalten. Die Impulse werden über den Zellkörper weitergeleitet. [18]

Zellkörper

Der Zellkörper, auch Perikaryon, ist das Zentrum des Neurons, es enthält den Zellkern. Zusammen mit dem Zellkern ist das Perikaryon für Stoffwechselvorgänge und interne Prozesse, sowie das Weiterleiten von Botenstoffen zuständig. [18]

Axon und Axonterminale

Das Axon, auch Nervenfaser genannt, geht aus dem Perikaryon hervor und ist für die Weiterleitung und Erzeugung von Nervenimpulsen zuständig. Die Impulse werden am Ende des Axons an die sog. Axonterminale übergeben, wo sie wiederum, durch Synapsen, den Impuls an andere Neuronen übermitteln. [18]

Die Informationen fließen im Neuron typischerweise von den Synapsen auf die Dendriten über den Zellkörper durch das Axon zu den Axonterminalen und auf andere Synapsen.

2.1.2 Lernen

Neurologisch gesehen ist das Lernen ein Prozess, der sich synaptische Plastizität nennt. Diese prägt sich auf vier verschiedene Arten aus. Die Übertragung eines Signals durch die Synapse wird (1) erstmals initiiert, (2) verstärkt (Langzeitpotenzierung (LTP)), (3) geschwächt (Langzeitdepression (LTD)), oder (4) abgebrochen [16]. Bei der LTP handelt es sich um eine Erhöhung der synaptischen Übertragung. Ausgelöst wird die LTP durch eine Häufung von Potentialen innerhalb des Axons. Diese Erhöhung kann einige Stunden oder Tage andauern, um diesen Zustand jedoch über einen längeren Zeitraum zu erhalten, müssen die Potentiale immer wieder an diesem Axon gehäuft werden. Bei der LTP wird postsynaptisch die Empfindlichkeit oder Anzahl von Rezeptoren erhöht, damit eintreffende Neurotransmitter eine größere Wirkung haben [21]. Bei der LTD verhält es sich ähnlich, nur, dass hier die Empfindlichkeit bzw. Anzahl von Rezeptoren abnimmt. Dies hängt mit einem niedrigen oder ausbleibenden Potential innerhalb des Axons zusammen [21],[22]. Generell: bei der LTP wird gelernt und bei der LTD wird vergessen.

Psychologisch gesehen ist Lernen definiert als eine auf Erfahrung basierte meist permanente Änderung des Verhaltens einer Person bzw. eines Lebewesens [23]. Untermauert wird dies durch den Behaviorismus, welcher auf dem Gedanken beruht, dass jegliches Verhalten durch Konditionierung gesteuert werden kann [24]. Unterschieden wird hierbei zwischen klassischer und operanter Konditionierung. Die klassische Konditionierung zielt darauf ab, eine natürliche Reaktion mit einer neutralen Aktion zu verknüpfen. So z. B. das Klingeln einer Glocke mit dem Geruch von Nahrung (Pawlowscher Hund). Die operante Konditionierung zielt darauf ab, Reaktionen durch Belohnung hervorzurufen oder durch Bestrafung zu unterdrücken. [23] Albert Bandura legte 1962 das Fundament für eine andere Theorie über das Lernen, die sog. Social Learning Theory. Sie besagt, dass Individuen, durch Observieren und Imitieren, das Verhalten von sog. social agents übernehmen und durch Transfer, Wissen bzw. Erfahrung erlangen können [25]. Dadurch, dass Kinder noch nicht die Fähigkeiten von Erwachsenen haben, sind sie, vor allem in jungen Jahren, darauf angewiesen, durch Beobachtung und Imitation von Erwachsenen (meistens deren Eltern), zu lernen [26], [27].

Ähnlich wie in der Psychologie gibt es auch bei NNs verschiedene Modelle und Theorien, wie diese lernen. Auf Anwendungen und Unterschiede zwischen diesen wird in 2.3.2 auf Seite 11 näher eingegangen. Davor werden allerdings mathematische Grundlagen, die den Umgang mit NNs erleichtern, erläutert.

2.2 Mathematische Grundlagen

Den Themenfeldern KI, ML und NN liegen viele mathematische Konzepte zugrunde, diese sind essentiell, und können die Arbeit mit NNs wesentlich erleichtern. Im Folgenden werden lediglich die Grundlagen der einzelnen mathematischen Felder erläutert, für weiterführende Erklärungen hilft folgende Literatur [28], [29], [30].

Lineare Algebra

Für die Arbeit mit NNs und vielen ML-Algorithmen ist Lineare Algebra essentiell. Sie beinhaltet wichtige mathematische Komponenten, wie z. B. Skalare, Vektoren, Matrizen und Tensoren, sowie Funktionen, die auf diese angewendet werden.

- *Skalare* sind einzelne Zahlen [2, S. 31]. Skalare können wie folgt deklariert werden: „sei $a \in \mathbb{N}$ “ oder „sei $a \in \mathbb{R}$ “.
- Ein *Vektor* ist ein Array aus geordneten Zahlen, die durch einen Index identifiziert werden können [2, S. 32]. Ein Vektor \mathbf{v} wird wie folgt deklariert:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}; \Rightarrow \mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}. \quad (1)$$

Der Vektor \mathbf{v}^T ist der Transponierte des Vektors \mathbf{v} . Hier ändert sich die Dimension des Vektors von einer Spalte zu einer Zeile.

- Eine *Matrix* $\mathbf{A}_{n,m}$ ist ein zweidimensionales Array aus geordneten Zahlen. Jedes Element wird hier durch zwei Indizes identifiziert [2, S. 32]. Eine Matrix wird wie folgt deklariert:

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,m} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,m} \end{bmatrix}; \Rightarrow \mathbf{A}^T = \begin{bmatrix} A_{1,1} & A_{2,1} & \cdots & A_{1,n} \\ A_{1,2} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{bmatrix}. \quad (2)$$

Die Transponierte der Matrix \mathbf{A} ist \mathbf{A}^T , diese ist an der Diagonalen gespiegelt.

- Ein *Tensor* $A_{i,j,k,\dots}$ ist ein mehrdimensionales Array mit mehr als zwei Dimensionen. Jedes Element kann durch seine Indizes identifiziert werden [2, S. 33].

Wichtige Funktionen auf Vektoren und Matrizen sind das Skalarprodukt und das Kreuzprodukt. Das Skalarprodukt von zwei Vektoren ist die Summe der Produkte der einzelnen indexgleichen Elemente der Vektoren [30, S. 26]. Das Ergebnis ist, namensgebend, ein Skalar. Definiert ist das Skalarprodukt wie folgt:

$$\mathbf{v} \cdot \mathbf{w} = \sum_k \mathbf{v}_k \mathbf{w}_k, \quad z. B. \text{ in } \mathbb{R}^2 : \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = v_1 w_1 + v_2 w_2. \quad (3)$$

Das Kreuzprodukt von zwei Matrizen hat als Ergebnis eine neue Matrix. Dabei ergibt sich die Dimension der neuen Matrix aus den beiden ursprünglichen Matrizen. Das Kreuzprodukt funktioniert nur, wenn die erste Matrix die selbe Anzahl an Spalten hat, wie die zweite Matrix Zeilen [2, S. 34f.]. Definiert ist das Kreuzprodukt wie folgt:

$$\mathbf{C}_{m,p} = \sum \mathbf{A}_{m,n} \mathbf{B}_{n,p}. \quad (4)$$

Formel 5 zeigt die Berechnung des Kreuzproduktes mit den Matrizen $\mathbf{A}_{2,2}$ und $\mathbf{B}_{2,2}$, welche mit exemplarischen Werten aufgefüllt wurden. Die Matrizen werden zeilen- und spaltenweise multipliziert bzw. addiert, wobei eine Zeile der Matrix \mathbf{A} mit einer Spalte der Matrix \mathbf{B} multipliziert wird.

$$\begin{aligned} \mathbf{C}_{2,2} &= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \\ &= \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix} \\ &= \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix} \end{aligned} \quad (5)$$

Die hier aufgeführten mathematischen Konzepte der linearen Algebra erleichtern den Umgang und das Rechnen mit NNs erheblich. Ein weitere mathematische Aspekte, die in NNs Anwendung finden, sind die Probabilistik und Numerik.

Probabilistik

Die meisten computerbasierten Anwendungen sind deterministisch, heißt, dass mit sehr hoher Sicherheit vorausgesagt werden kann welche Zustände und Abläufe die Anwendung durchläuft. KI, ML und NNs sind dagegen typischerweise probabilistische Systeme, heißt, dass deren Zustände und Abläufe von hoher Unsicherheit bzw. Unbestimmtheit geprägt sind. Diese Unbestimmtheit kann verschieden verursacht werden: [Vgl. 2, S. 54]

- **Unvollständige Beobachtbarkeit:** Wenn nicht alle Variablen, die das System beeinflussen, beobachtbar sind, so kann nicht angenommen werden, dass das Resultat deterministisch ist.
- **Unvollständige Modellierung:** Wenn das Modell einige beobachtete Informationen, die das System beeinflussen, ausschließt, so wird das Resultat unbestimmt.
- **Dem Modell inhärente stochastische Bedingungen:** Gegeben, dass ein Modell in seiner Anwendung mit tatsächlichem Zufall bzw. Willkür arbeitet (z. B. gemischte Karten) wohnt dem System folglich eine Unbestimmtheit inne.

Diese Unbestimmtheit ist allerdings auch nützlich, da es oft Sinn macht anstatt einer komplexen und deterministischen Regel eine zwar unbestimmte, dafür aber简plere Regel zu nutzen. Die Regel: „Die meisten Menschen können gehen“ ist sehr viel einfacher umzusetzen und auch vielseitiger als die Regel: „Menschen können gehen, allerdings erst: ab einem unbestimmten Alter, wenn sie es gelernt haben; bis zu einem unbestimmten Alter, wenn sie nicht mehr die Kraft dazu haben; wenn sie nicht temporär oder dauerhaft durch Krankheit oder Unfall eingeschränkt sind“. Diese Regel ist zwar genauer, allerdings auch schwieriger umzusetzen und hat keine Garantie jeden Fall abzudecken. [2, S. 55]

Numerik

Obwohl für NNs viele numerische Berechnungen durchgeführt werden, wird hier besonderer Fokus auf die gradient-basierte Optimierung gelegt. Das Ziel dieser Optimierung ist es mit dem Stochastic Gradient Descent (SGD)-Verfahren das globale Minimum einer ableitbaren Funktion $f(x)$, meistens Verlust- oder Kostenfunktion, zu finden [2, S. 82ff.]. Da die Funktionen eines NN äußerst komplex sind, wurde typischerweise nur ein sehr kleiner Wert für f gesucht. Globale Minima lassen sich allerdings dennoch, vor allem durch die inzwischen hohe Rechenleistung, finden. Da das Finden von globalen Minima eine komplexe Berechnung erfordert, wird hier auf folgende Artikel referenziert [31], [32].

2.3 Machine Learning

Wie in 1 auf Seite 1 bereits erwähnt, ist ML die Fähigkeit eines KI-Systems, Wissen aus repräsentativen Daten zu erlangen [2, S. 2]. Dabei handelt es sich um einen Algorithmus, der anhand gegebener Daten und spezifischer Parameter lernt. Eine spezielle Form von ML ist das sog. DL, welches mit vielschichtigen NNs arbeitet [33]. Im Folgenden wird darauf eingegangen, was ein „Lernalgorithmus“ ist und welche Aufgaben er erfüllen kann bzw. soll. Anschließend werden verschiedene Arten des Lernens erläutert und Probleme zwischen den Datensätzen und dem Modell aufgezeigt. Abschließend wird das Konzept der Regularisierung ausgeführt.

2.3.1 Lernalgorithmus

Ein Lernalgorithmus ist ein computerbasiertes Programm mit einer definierten Menge an Tasks T , deren Performance durch P gemessen wird. Das eigentliche Lernen findet dann statt, wenn der Algorithmus aus einer Erfahrung E die Performance der Tasks erhöhen kann [34, S. 2]. Formel 6 zeigt die mathematische Beschreibung, wobei die Performance P_1 sich aus dem Task T in Abhängigkeit von der Erfahrung E_0 ergibt.

$$P_1[T|E_0] < P_2[T|E_1], \text{ wobei } \lim_{n \rightarrow +\infty} P_n[T|E_{n-1}] = 1. \quad (6)$$

P_1 wird kleiner sein als P_2 , die sich aus T in Abhängigkeit von E_1 ergibt. E_1 ist die Erfahrung, die sich aus der Analyse von P_1 ergibt. P strebt gegen 1, also 100%.

Der Task T

Die meisten Tasks oder Aufgaben, die ML-Lernalgorithmen bewältigen sollen, sind komplexe Aufgaben, welche vom Menschen gar nicht oder nur mit sehr hohem Aufwand entwickelt werden können [2, S. 99]. Differenziert werden muss hierbei, dass der Task lediglich die zu bewältigende Aufgabe vorgibt und nicht den Lernprozess abbildet. Einige typische Tasks, die von ML-Algorithmen bearbeitet werden sind: [2, S. 100ff.]

- **Klassifikation:** Der Algorithmus soll anhand verschiedener Eingangsparameter bestimmen, zu welcher Klasse k das jeweilige Objekt bzw. der Input gehört.
- **Regression:** Der Algorithmus soll eine fundierte Voraussage über eine bestimmte Zahl treffen.
- **Transkription:** Der Algorithmus soll aus einem gegebenen, meist auditiven, Input einen textuellen Output erzeugen.

Die Performance P

Die Performance der Tasks eines Lernalgorithmus gibt Aufschluss darüber, wie gut der Task erfüllt werden kann. Dies kann mit verschiedenen Metriken, wie der Accuracy bzw. Genauigkeit oder der Error Rate bzw. Verlustrate abgebildet werden. Je nach Anwendung können andere Metriken aussagekräftiger sein. Wichtig ist, dass P auch mit unbekannten Daten ermittelt wird. [2, S. 103f.].

Die Erfahrung E

Die Erfahrung eines ML-Lernalgorithmus ist der zu Beginn auf T angewendete Datensatz, kombiniert mit der genutzten Lernmethode [2, S. 105]. Die Lernmethode, die im Lernalgorithmus angewendet wird, hat direkten Einfluss darauf, wie sich E entwickelt. Es gibt unterschiedliche Ansätze für die Lernmethoden, aber Grundsätzlich kann zwischen drei Arten von Lernmethoden unterschieden werden.

2.3.2 Arten des Lernens

Supervised Learning

Wenn Supervised Learning (SL) in einem Lernalgorithmus angewendet wird, besteht E aus dem Datensatz mit Merkmalen, die durch Labels markiert sind. Die Labels enthalten spezifische Informationen, die die Merkmale des Datensatzes beschreiben. Kategorisiert werden kann der Datensatz als Input und das Label als Output, der das gesuchte Ergebnis enthält. Jeder Datensatz muss je nach Task T anders gelabeled werden. Da dieser Prozess allerdings sehr aufwändig und komplex ist werden meistens menschliche „supervisors“ benötigt, um die Labels zu erstellen, es ist allerdings auch möglich diese automatisiert erstellen zu lassen. Typischerweise wird SL bei der Klassifikation oder Regression angewendet. [2, S. 105], [2, S. 139], [35].

Unsupervised Learning

Bei Unsupervised Learning (UL) besteht E lediglich aus dem Datensatz ohne beschreibende Labels. Der Algorithmus lernt die Struktur der Daten oder extrahiert Informationen aus ihnen. Vorteil von UL ist, dass es wenig menschliche Unterstützung braucht. Typische Anwendungen für UL sind Clustering und Assoziation. [2, S. 105], [2, S. 145], [36]. Bekannte UL-Algorithmen sind z. B. die Principal Component Analysis oder der k-means Clustering Algorithmus. Beide dieser Algorithmen werden zum Lernen der Darstellung von Datenmengen genutzt [2, S. 146ff.].

Reinforcement Learning

Beim Reinforcement Learning (RL) besteht E aus dem Datensatz und einer Reward / Penalty Funktion, die gute Resultate belohnt und schlechte bestraft. Das Ziel des Algorithmus ist es eine Regel zu finden, um den Reward, den er bekommt, zu maximieren. E wird hierbei stark von der Reward / Penalty Funktion beeinflusst. Typischerweise wird RL bei Entscheidungsfindungsproblemen eingesetzt. [37].

Unabhängig von der Lernmethode, ist zu beachten, wie gut das Modell die reale Anwendung in T abbildet, unbeachtet kann es zu Over- oder Underfitting kommen.

2.3.3 Overfitting und Underfitting

Die Güte der Abbildung der realen Anwendung durch das Modell spiegelt sich darin wieder, wie hoch P ist, wenn das Modell vollkommen neue und unbekannte Daten (Testdaten) als Input bekommt [2, S. 110]. Damit P mit den Testdaten maximal wird, muss das Modell einen hohen Grad an Generalisierung aufzeigen. Generalisierung heißt in diesem Kontext also, dass das Modell auch Daten, die nicht zum Trainingsdatensatz gehören, richtig zuordnen kann [34, S. 110f.]. Um eine hohe Generalisierung zu erreichen wird der gesamte Datensatz aufgeteilt in: [38]

- **60 % / 98 % Trainingsdaten:** Die Daten, mit denen das NN T trainiert.
- **20 % / 1 % Validierungsdaten:** Die Daten, mit denen das NN T validiert.
- **20 % / 1 % Testdaten:** Die Daten, mit denen T getestet wird.

Die prozentualen Werte sind bezogen auf die Größe des gesamten Datensatzes, wobei die Aufteilung 60 / 20 / 20 bei einer Größe bis 1.000.000 und die Aufteilung 98 / 1 / 1 ab 1.000.000 Daten angewendet wird. Wenn das Modell keine gute Generalisierung hat, dann ist es entweder overfitted oder underfitted. Overfitting bedeutet, dass P des Modells bei den Trainingsdaten hoch ist, aber bei den Validierungsdaten niedriger [34, S. 67]. Das Modell greift Merkmale auf, die nicht dem eigentlichen Datensatz zugrunde liegen und scheitert dann bei der Validierung [39]. Underfitting bedeutet, dass P des Modells bei den Trainingsdaten niedrig ist, aber bei den Validierungsdaten hoch [2, S. 111]. Hier scheitert das Modell daran Schlüsselmerkmale, die dem Datensatz zugrunde liegen, aufzugreifen, die Validierung ist somit ungültig [40]. Eine visuelle Repräsentation von Underfitting, Overfitting, sowie einer optimalen Generalisierung zeigt Abbildung 4 auf der nächsten Seite. Die Gerade (l) kann die Datenpunkte nicht hinreichend abbilden

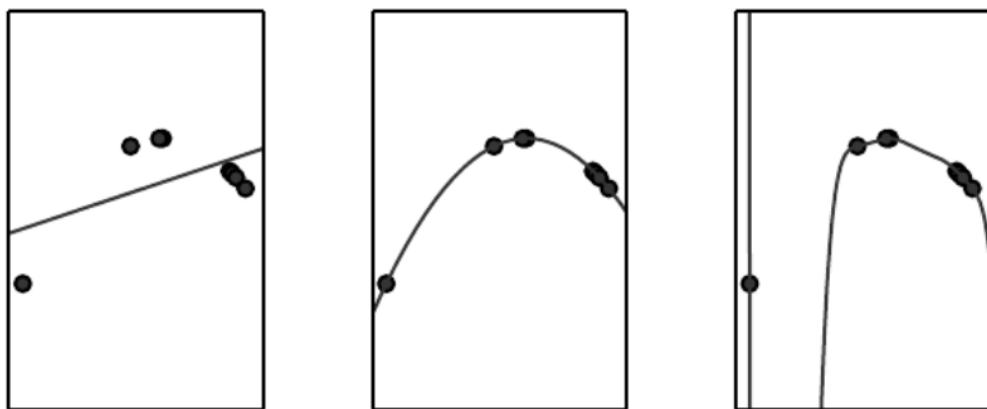


Abbildung 4: Beispiel Underfitting(l), gute Generalisierung(m) und Overfitting(r) [Entnommen aus: 2, S. 113].

und ist somit underfitted. Das Polynom neunten Grades (r) bildet die Datenpunkte zwar perfekt ab, allerdings wird hier die Struktur der Daten verzerrt, was in dem Tal nach dem ersten Punkt zu sehen ist, die Funktion ist also overfitted. Die quadratische Funktion (m) bildet die Datenpunkte hinreichend ab und es gibt kein bedenkenswertes Over- oder Underfitting.

Um Over- bzw. Underfitting entgegenzuwirken und die Lernfähigkeit des Modells zu erhöhen, wurde das Konzept der Regularisierung entwickelt.

2.3.4 Regularisierung

Die Regularisierung zielt darauf ab, durch Modifikation des Lernalgorithmus, die Generalisierung des Modells zu erhöhen. Umgesetzt wird dies meist durch eine Penalty-Funktion, welche das NN dafür bestraft sich die Daten einzuprägen, anstatt die inhärenten Muster zu erlernen. Dadurch sollen Generalisierungsfehler reduziert werden ohne das Training selbst zu beeinflussen. Implementiert werden kann die Regularisierung auf zwei verschiedene Arten: L1 und L2. Die L1 Regularisierung bezieht sich auf eine lineare Berechnung der zu bestrafenden Werte. Die L2 Regularisierung bezieht sich auf eine quadratische Berechnung der zu bestrafenden Werte. Der Vorteil von L2 Regularisierung ist, dass große Werte stärker bestraft werden als kleine Werte, weshalb die L2 Regularisierung typischerweise angewendet wird. [2, S. 118ff.], [41, S. 335]

2.4 Neuronale Netze

Der Begriff hinter NN wurde bereits oft aufgegriffen, aber um wirklich zu verstehen, was NNs sind, wird im Folgenden auf deren Architektur eingegangen. Anschließend wird der Trainingsprozess näher erläutert und die inhärenten Mechanismen ausgeführt. Abschließend werden Metriken der Performance erläutert und Möglichkeiten für Optimierung aufgezeigt.

2.4.1 Architektur

Die Architektur von NNs besteht aus Neuronen, die in Schichten miteinander vernetzt sind. Daraus ergeben sich mehrere theoretische Ansätze und Topologien.

Neuronen

Ähnlich wie in Abschnitt 3 auf Seite 5, werden Neuronen auch in künstlichen NNs modelliert. Abbildung 5 zeigt den exemplarischen Aufbau eines Neurons. Hierbei werden x_1 bis x_n als Inputwerte und w_1 bis w_n als Gewichtungen der jeweiligen Verbindungen definiert. Die Berechnung eines inneren Zwischenwerts t wird durch die in Formel 7



Abbildung 5: Exemplarisches Modell eines künstlichen Neurons [Eigene Darstellung von 41, S. 13]

abgebildete Summenfunktion ausgeführt. Der Zwischenwert wird durch die Summe aller Inputs x multipliziert mit den jeweiligen Gewichtungen w und der anschließenden Addition des Bias b berechnet. Der Bias b ist ein dem Neuron inhärenter Wert, der t positiv oder negativ beeinflussen kann. Dadurch soll das Modell die reale Anwendung dynamischer abbilden können. [41, S. 13f.]

$$t = \sum_{n=1}^N x_n \cdot w_n + b \quad (7)$$

Um den Output des Neurons Z zu berechnen, wird der temporäre Zwischenwert t dann mithilfe einer Aktivierungsfunktion beeinflusst bzw. aktiviert [41, S. 18]. Dementsprechend ergibt sich in Formel 8 [41, S. 185] die Berechnung des Outputs Z .

$$Z = f(t) = f\left(\sum_{n=1}^N x_n \cdot w_n + b\right) \quad (8)$$

Aktivierungsfunktion

Die Aktivierungsfunktion beeinflusst den Output des NN, indem sie die Werte jedes Neurons skaliert [41, S. 72]. Es gibt dabei verschiedene Aktivierungsfunktionen:

- **Die Step-Funktion:** Die Step-Aktivierungsfunktion simuliert, wann ein Neuron „feuert“. Das bedeutet, dass, wenn $t > 0$ feuert das Neuron und der Output $Z = 1$, sonst, also bei $t \leq 0$, ist $Z = 0$. Diese Aktivierungsfunktion wird eher selten genutzt. [41, S. 73].
- **Linear-Funktion:** Die lineare Aktivierungsfunktion gibt den Input ohne Veränderung weiter: $t = Z$ [41, S. 74].
- **Sigmoid-Funktion:** Die Sigmoid Aktivierungsfunktion ist eine verbesserte Variante der Step-Funktion. Sie zeigt, wie nah das Neuron am Wendepunkt von feuern zu nicht feuern war, bevor es aktiviert wurde. Die Berechnung erfolgt folgendermaßen: $Z = \frac{1}{1 + e^{-t}}$. [41, S. 75].
- **Rectified Linear-Funktion:** Die Rectified Linear Aktivierungsfunktion ist eine Mischung aus der linearen und der Step-Funktion. Für $t > 0$ gilt $Z = t$, aber für $t \leq 0$ gilt $Z = 0$. Durch diese non-lineare Eigenschaft der Aktivierungsfunktion ist sie sehr vielseitig einsetzbar, weshalb sie derzeit am verbreitetsten ist. [41, S. 76f.], [2, S. 173f.].

Typischerweise wird in einem NN nicht nur ein einzelnes Neuron, sondern viele Neuronen genutzt und in verschiedene Schichten aufgeteilt.

Schichten / Layer

Typischerweise werden NN in Schichten aufgeteilt. Die Größe der Schichten kann stark variieren. Es gibt drei Arten von Schichten in NNs, welche verschiedene Aufgaben erfüllen. Eine Übersicht mit den Schichten und Verbindungen zeigt Abbildung 6 auf der nächsten Seite. [41, S.18f.], [42].

- Das **Input Layer** enthält den Input, der durch den Datensatz in das NN gelangt, das können Pixelwerte oder andere Daten sein. Typischerweise werden die Daten vor der Einspeisung in das Netz vorbereitet
- Das **Hidden Layer** übernimmt die eigentliche „Logik“ des NN. Hier wird die Summenfunktionen, in Abhängigkeit der Verbindungen berechnet. Typischerweise gibt es mehrere Hidden Layer, durch die Inputwerte laufen. Empirische Daten belegen, dass eine höhere Zahl an Hidden Layers direkt mit der Güte der Generalisierung, bzw. P des Modells, zusammenhängt, wobei der Effekt ab 12 Layers marginal ist [2, S. 201]. In verschiedenen Ansätzen von spezialisierten NNs können den Hidden Layers spezielle Funktionen zugeordnet werden [43].
- Das **Output Layer** enthält die Endwerte des NN. Die Größe des Output Layers wird durch die Anwendung vorgegeben. Die Endwerte bewegen sich meistens im Intervall zwischen 0 und 1, sie können z. B. als Koordinaten oder Boolean-Werte interpretiert werden.

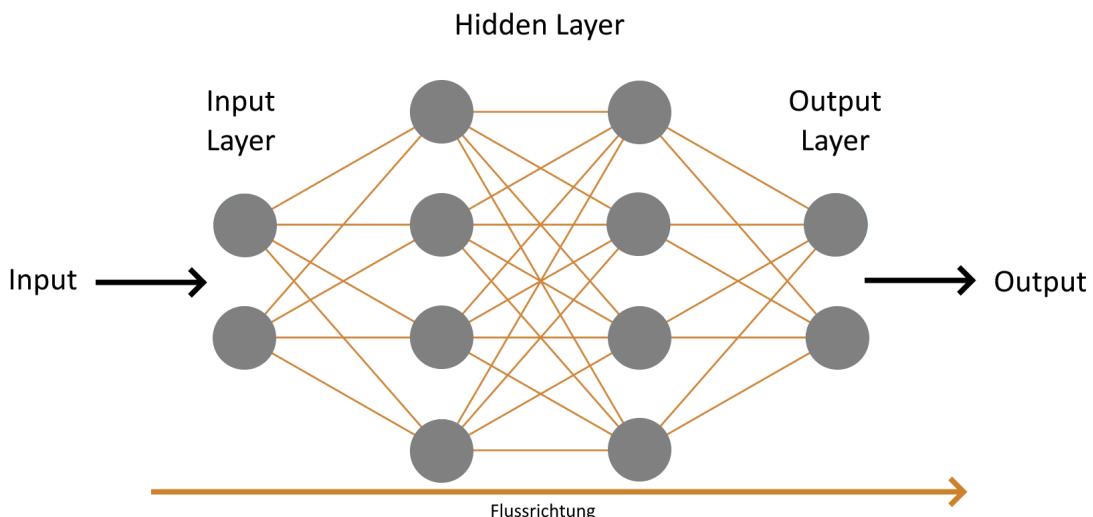


Abbildung 6: Exemplarischer Aufbau der Layer-Struktur eines neuronalen Netzes [Eigene Darstellung von 41, S. 19]

Nicht nur die Anzahl der Neuronen und Schichten hat einen direkten Einfluss auf die Performance des Modells, sondern auch die angewandten Mechanismen und die Flussrichtung der NNs. Dementsprechend wurden im Laufe der Zeit verschiedene Ansätze für spezialisierte Topologien vorgestellt.

Topologien

Um eine größere Bandbreite an Anwendungen bewältigen zu können, wurden verschiedene Topologien bzw. Arten von NNs entwickelt. Dabei wurden bestehende NNs abgeändert, um bei einer neuen Anwendung eine höhere Performance zu erzielen [43]. Unabhängig von einer verbesserten Performance ergeben sich noch weitere Vorteile. Betrachtet werden im Folgenden einige gängige Topologien für NNs, der Fokus wird dabei auf Multi-Layer Netzwerke gelegt.

Feed-Forward Neural Network (FFNN)

Ein FFNN besteht, ähnlich wie in Abbildung 6 auf der vorherigen Seite abgebildet, aus mehreren Schichten verbundener Neuronen. Der Name „Feed-Forward“ kommt daher, dass die Parameter von vorne nach hinten durch das Netz fließen. Die Entwicklung von FFNNs bildet einen wichtigen Meilenstein in der Wissenschaft, da FFNNs die vielseitigsten NNs sind, die bisher entwickelt wurden. Dadurch können sie für viele verschiedene Aufgaben eingesetzt werden. Außerdem sind viele heutzutage angewendeten NNs als Weiterentwicklung aus den FFNNs entstanden. Vorteile von FFNNs sind vor allem, dass sie mit wenig menschlicher Interaktion lernen, dazu auch noch ziemlich robust sind und eine relativ stabile Lernkurve haben. Nachteile bilden sich vor allem durch die schlechte Interpretierbarkeit der internen Prozesse und eine hohe Trainingszeit ab. [2, S. 167ff.], [44].

Convolutional Neural Network (CNN)

Ein CNN ist eine spezialisierte Art von FFNN, die optimiert auf das Arbeiten mit in Gitter strukturierten Daten, wie Zeitreihen oder Bilddateien, ist [2, S. 330]. Der hauptsächliche Unterschied zu FFNNs ist, dass CNNs die namensgebende Operation „convolution“ (Faltung) und zusätzlich noch sub-sampling (Unterabtastung) ausführen [43, S. 6]. Convolution dient dazu bestimmte Features eines Datensatzes zu erkennen, z. B. Kanten oder Formen in einem Bild. Um das zu ermöglichen, wird meist eine gewichtete Summe über einen Teil des Datensatzes gebildet und die Ergebnisse in einer Feature Map gespeichert [2, S. 331]. Die Werte in der Summe werden gewichtet, damit bestimmte Teile des Datensatzes mehr zum Ergebnis beitragen. Die Feature Map enthält dann das erkannte Feature und die ungefähre Position [43, S. 6]. Nach der Convolution wird meistens ein sub-sampling ausgeführt, welches dazu dient das Ergebnis weniger anfällig für Rauschen und Unschärfe im Datensatz zu machen. Der Ansatz dazu ist, dass die Features der Feature Map den Inhalt des Datensatzes klassifizieren können und die exakte Position der Features nicht wichtig ist bzw. sogar negativen Einfluss

auf das Ergebnis haben kann. Entsprechend der Operationen ist der spezifische Aufbau der Hidden Layer eines CNN anders als die eines FFNN. Die Hidden Layer werden unterteilt in Convolutional Layer, Sub-Sampling Layer und Fully-Connected Layer. Eine Übersicht zeigt hier Abbildung 7. Das Convolutional Layer führt die Convolution aus, das Sub-Sampling Layer führt das sub-sampling aus und das Fully-Connected Layer interpretiert die Feature Map und generiert den Output. Vorteile im Vergleich zu FFNNs sind vor allem eine kürzere Trainingszeit und eine schnellere Lernkonvergenz, sowie eine durch die Spezialisierung hervorgerufene flexible Skalierbarkeit bei den Inputdaten. [2, S. 330ff.], [43].

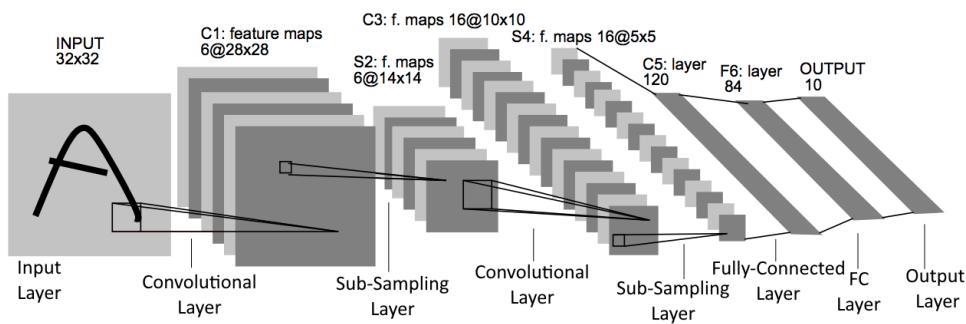


Abbildung 7: Exemplarischer Aufbau der Layer-Struktur eines Convolutional Neural Network mit Feature Maps [Eigene Darstellung von 43, S. 7]

Recurrent Neural Network (RNN)

Ein RNN ist ebenfalls ein spezialisiertes FFNN, welches auf das Arbeiten mit Datensequenzen spezialisiert ist [2, S. 373]. Spezielle Einsatzmöglichkeiten bieten sich bei Sprachmodellierung und -erkennung [45]. RNNs nutzen Loops, um den Output eines Layers wieder als Input für dasselbe Layer zu nutzen [46]. Sie nutzen sog. Recurrent Layer als Hidden Layer, dadurch können sie wiederholt dieselbe Funktion auf die Daten bzw. Werte anwenden. Um die Werte zwischenzuspeichern nutzen RNNs sog. „hidden states“, welche auf der Aktivierung der Neuronen basieren [47].

Das Ziel eines NN ist es, eine bestimmte Funktion f zu finden bzw. zu lernen und damit in gewisser Weise anzunähern. Diese Funktion f soll so gut wie möglich den dem NN unterliegenden Task T abbilden. Um die Funktion zu erlernen, muss das Netz trainiert werden. Dieser Prozess wird im Folgenden erläutert.

2.4.2 Training

Der Prozess zum Trainieren eines NN läuft durch mehrere Phasen. Abbildung 8 zeigt den möglichen Ablauf eines Trainingsprozesses. Angenommen wird hier, dass die Daten bereits bereinigt und vorbereitet wurden (Abschnitt 2.5 auf Seite 23). Bevor das eigentliche Training startet, wird das NN initialisiert, meist mit zufälligen Werten [48]. Danach beginnt der Trainingsprozess mit der Forward Propagation (FP) (auch forward pass). Als FP ist der Durchlauf der Daten durch das NN in dessen Flussrichtung bezeichnet. Das bedeutet, dass die Trainingsdaten durch das Netz gespeist werden, bis am Ende der Output Z ausgegeben wird. Nach der FP beginnt die Backward Propagation (BP) mit der Berechnung des Losses. Die BP zielt darauf ab, durch

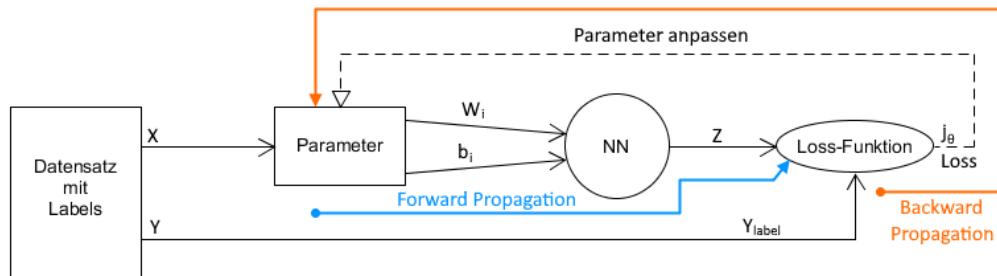


Abbildung 8: Ablauf des Trainingsprozesses [Eigene Darstellung von 48]

wiederkehrende Veränderung der Werte des NN den Loss zu minimieren, bzw. den errechneten Output an den gewünschten Output anzunähern [49]. Zunächst erzeugt die BP durch die Loss-Funktion den skalaren Wert j_θ als Verlustwert bzw. Loss. Für unterschiedliche Anwendungen gibt es verschiedene Loss-Funktionen, die angewendet werden, eine Übersicht gibt folgende Dokumentation [50]. Der Loss ist ein Indikator für die Performance des NN, da bei $j_\theta = 0$ das Netz alle Trainingsdaten korrekt labeln konnte [2, S. 203], [48]. Um nun die Optimierung der Parameter vorzunehmen, nutzt BP den SGD, welcher in Formel 9 [51] mathematisch beschrieben ist. Hier wird der SGD auf L angewendet, indem einem Gewicht W ein neuer Wert zugewiesen wird.

$$W_{neu} \leftarrow W - \eta \nabla_W L(N) \quad (9)$$

Der neue Wert errechnet sich anhand der Ableitung nach W multipliziert mit der Lernrate η . Die Lernrate ist hier entscheidend, da sie die Konvergenz „steuert“ [51]. Abbildung 9 auf der nächsten Seite zeigt einerseits den SGD und andererseits, wie die Lernrate ihn beeinflusst. Hier zu sehen ist, dass sich, selbst bei einer gut gewählten

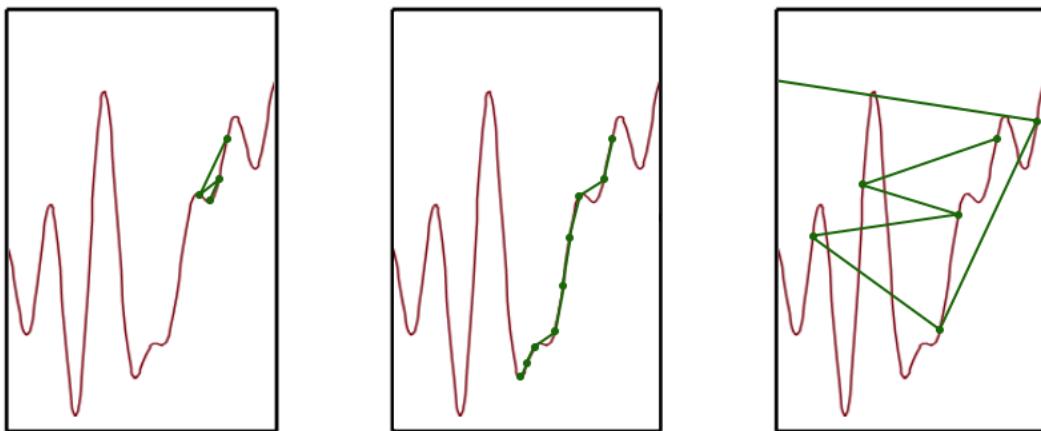


Abbildung 9: Stochastic Gradient Descent und Auswirkung der Lernrate [Eigene Darstellung von 41, S. 257ff.] Links ist η nicht optimal, sodass sich die Funktion in einem lokalen Minimum fängt. Rechts ist η zu hoch gewählt, sodass die Funktion nicht konvergiert. In der Mitte ist η gut gewählt, sodass sich die Funktion in einem tiefen lokalen Minimum fängt.

Lernrate, die Funktion nicht zwangsläufig zum globalen Minimum konvergiert. Der SGD nutzt die Lernrate, um die Abtastung der Funktion für die Optimierung zu berechnen. Dementsprechend wird bei optimaler Lernrate eine optimale Abtastung und dadurch eine möglichst optimale Konvergenz erreicht [51]. Nach der Berechnung durch den SGD wird die eigentliche BP (hier auch backward pass genannt) ausgeführt. Das bedeutet, dass das NN von hinten nach vorne durch iteriert wird und die Gewichte W durch optimierte W_{neu} ersetzt werden [48]. Hierbei werden die Gewichte, welche den größten positiven Einfluss auf j_θ priorisiert. Typischerweise werden FP und BP für einen kompletten Datensatz in einer sog. Epoche zusammengefasst [48]. Um das Modell weiter zu optimieren werden für einen Trainingsprozess mehrere Epochen durchlaufen, diese sind variabel festlegbar. Zu viele Epochen können zu Overfitting führen, zu wenige zu Underfitting. Da viele ML Modelle eine große Menge an Daten brauchen, um gute Ergebnisse zu erzielen, die Rechenleistung allerdings nicht ausreicht, um den gesamten Datensatz durch das Netz laufen zu lassen, wird dieser in Batches aufgeteilt [48]. Der Vorteil von diesen Batches ist, dass die Trainingszeit verkürzt wird, aber auch die Konvergenz schneller sein kann, sofern die BP nach jedem Batch die Gewichte optimiert [41, S. 44f.]. Die Batch-Size ist allerdings auch stark von der genutzten Hardware, speziell vom dedizierten Grafikspeicher oder dem Arbeitsspeicher des Computers abhängig [48]. Das Verändern der Lernrate, Epochenzahl oder Batch-Size bietet die Möglichkeit die Performance des NN zu optimieren.

2.4.3 Performance-Metriken

Um die Performance eines NN zu messen werden verschiedene Metriken genutzt, anhand dieser Metriken können bestimmte Optimierungsmaßnahmen eingeleitet werden [2, S. 424]. Die hier aufgeführten Metriken werden für die Klassifikation und Lokalisation genutzt, allerdings gibt es für andere Tasks auch weitere Metriken, eine Übersicht gibt es hier [52]. Für die beispielhafte Ausführung der Metriken wird hier von einem Datensatz mit 1000 Bildern ausgegangen, 100 davon enthalten einen Hund, 900 nicht. Hiervon wurden 60 als Hund erkannt (true-positive), 40 wurden nicht als Hund erkannt (false-negative), 850 wurden als nicht-Hund erkannt (true-negative) und 50 wurden nicht als nicht-Hund erkannt (false-positive).

Accuracy

Die Accuracy gibt die Richtigkeit des Modells an. Hier wird die Anzahl der richtigen Klassifizierungen in Relation mit der Gesamtzahl der Bilder gesetzt: [52]

$$\text{accuracy} = \frac{\text{true-positive} + \text{true-negative}}{900 + 100} = \frac{60 + 850}{1000} = 91\%. \quad (10)$$

Precision

Die Precision gibt die Genauigkeit des Modells an. Hier wird die Anzahl der true-positives mit der Gesamtzahl der Klassifikationen für die jeweilige Klasse in Relation gesetzt:

$$\text{precisionHund} = \frac{\text{true-positive}}{\text{true-positive} + \text{false-positive}} = \frac{60}{60 + 50} = 54.5\%. \quad (11)$$

Die Unterscheidung zwischen Accuracy und Precision ist wichtig, da die Klassen in diesem Beispiel stark voneinander abweichen. Würde das Modell jedes Element als nicht-Hund Klassifizieren, läge die Accuracy dennoch bei 90 %, obwohl das Modell nichts lernen würde. [52].

Recall

Der Recall gibt die Genauigkeit der Klassen des Modells an. Das heißt, es werden die true-positives mit der Gesamtzahl der Bilder der Klasse in Relation gesetzt: [52]

$$\text{recallHund} = \frac{\text{true-positive}}{\text{true-positive} + \text{false-negative}} = \frac{60}{60 + 40} = 60\%. \quad (12)$$

Intersection over Union (IoU)

Die IoU ist eine speziell für Lokalisation entwickelte Metrik, die die Übereinstimmung der Fläche der Labels (prediction und true) auf einem Bild ermittelt. Formel 13 zeigt die mathematische Definition und Abbildung 10 zeigt die graphische Repräsentation. Die Berechnung erfolgt über einen bestimmten Threshold (meistens 0.5). Wenn die IoU unter dem Threshold ist, wird die Lokalisation als falsch deklariert [53].

$$IoU = \frac{|A \cap B|}{|A \cup B|} = \frac{|I|}{|U|}; IoU \in [0; 1]. \quad (13)$$

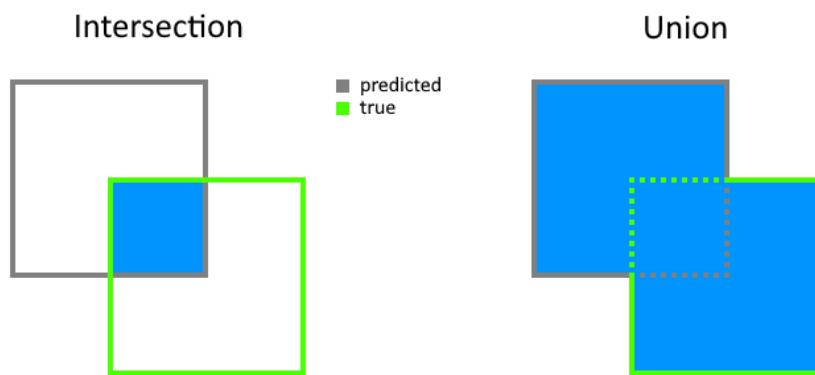


Abbildung 10: Graphische Repräsentation der Intersection over Union Metrik
 [Eigene Darstellung von 54]

Mean Average Precision (mAP)

Die mAP ist die gemittelte bedingte Genauigkeit des Modells, sie wird ermittelt, indem die bedingte Genauigkeit über alle IoU Thresholds gemittelt wird. Die bedingte Genauigkeit wird durch die Interpolierung der Precision mit dem Recall errechnet. Typischerweise werden für die IoU Thresholds elf Werte (0, 0.1, 0.2, ..., 0.9, 1) genutzt. Berechnet werden kann die mAP folgendermaßen: [54]

$$mAP = \frac{1}{11} \sum_{i=0}^{10} \frac{1}{10} \cdot Precision_{interpolated}(i). \quad (14)$$

Neben der Architektur des NN und den Trainingsparametern haben die Daten, die in das Netz eingespeist werden, einen hohen Einfluss auf die Performance. Typischerweise müssen die Daten vorbereitet, bzw. aufbereitet werden.

2.5 Datenvorbereitung und Datenaufbereitung

Im Folgenden wird darauf eingegangen, weshalb die Daten aufbereitet werden müssen, welche Schritte bei der Konstruktion des Datensatzes beachtet werden müssen und welche Möglichkeiten es gibt die Daten zu transformieren.

Bedeutung im Machine Learning

Die Daten, die für das Training von NNs genutzt werden, haben einen sehr großen Einfluss auf die Performance des Modells. Laut Google [55] sind die Datenmenge und Qualität der Daten die wichtigsten Eigenschaften des Datensatzes. Das liegt daran, dass das NN anhand der Daten, die es erhält, das Modell trainiert. Sind die Daten repräsentativ, kann das NN das Modell nahe an die reale Anwendung trainieren, sind die Daten nicht repräsentativ, ist es das Modell auch nicht. Das Arbeiten mit Daten, also Datensammlung, Datenaufbereitung und Datenpräsentation umfasst ca. 60–80% der Zeit, die für ML aufgewendet wird. [55], [56], [57].

Konstruktion des Datensatzes

Die Konstruktion des Datensatzes teilt Google [55] in drei Schritte:

- Zunächst müssen die Daten gesammelt werden [57]. Nach Google kann hier bereits auf die Qualität und Verlässlichkeit der Daten geachtet werden, sodass schlechte Daten bereits im Voraus aussortiert werden. Hierbei muss vor allem überprüft werden, ob die Daten Rauschen und Ausreißer enthalten und, dass keine doppelten Daten oder fehlende Werte enthalten sind. Um eine ausreichende Menge an Daten für den Datensatz zu sammeln ist es manchmal nötig Daten aus verschiedenen Quellen heranzuziehen. Falls dies der Fall ist, muss sichergestellt werden, dass die Daten die gleichen Formate haben.
- Der zweite Schritt der Konstruktion des Datensatzes ist, laut Google das Identifizieren der Features. Dieser Schritt ist typischerweise relativ einfach, da er direkt von der Aufgabe, die das System erfüllen soll, abhängig ist. Die Identifikation gibt am Ende die Labels für die jeweiligen Daten vor.
- Im dritten Schritt soll nach Google der Datensatz aufgeteilt werden. Die relative Aufteilung wurde bereits in Abschnitt 2.3.3 auf Seite 12. Hier gibt es die Möglichkeit die Daten strukturiert oder zufällig aufzuteilen, je nach Anwendung wird eine andere Methode genutzt.

Transformation der Daten

Datentransformation beschreibt den Prozess, Daten zu manipulieren oder zu organisieren, um diese anschließend analysieren zu können, hier in einem NN [57]. Diese Transformation ist wichtig, da so zum einen die Kompatibilität der Daten und zum anderen die Qualität der Daten sichergestellt werden kann [55]. Die Transformation der Daten kann vor dem Training stattfinden, sodass die Daten nur einmal transformiert werden müssen. Das hat allerdings zur Folge, dass bei Veränderung der Transformation der gesamte Datensatz neu generiert werden muss [55]. Die Transformation kann außerdem während des Trainings stattfinden, dadurch können, unabhängig von Änderungen an der Transformation, immer die selben Daten verwendet werden. Das wirkt sich allerdings negativ auf die Trainingszeit des Modells aus [55]. Folgende Funktionen können als Transformation auf die Daten angewendet werden:

- **Skalieren:** Typischerweise werden NN entsprechend einer fixen Größe der Daten entwickelt, dementsprechend müssen die Daten vor dem FP skaliert werden, damit das NN sie richtig nutzen kann [58].
- **Normalisieren:** Daten zu normalisieren bezieht sich darauf alle Daten auf eine gleiche Skala zu beziehen. Typischerweise wird die Skala auf $[0; 1]$ gesetzt, wobei der höchste Wert auf eins und der niedrigste auf null gesetzt wird [58]. Hier gibt es verschiedene Verfahren, die spezielle Effekte haben. Z. B. können mit „Feature Clipping“ Ausreißer reduziert werden und mit „Log Scaling“ die Skala komprimiert werden [55].
- **Diskretisieren:** Hier werden quantitative Daten in qualitative Daten transformiert, bzw. numerische Werte in kategorische Werte [59].

Trotz der ausgeführten Maßnahmen, kann es sein, dass einige fehlerhafte Daten im Datensatz erhalten sind, die Anzahl muss soweit wie möglich reduziert werden, da sonst das Modell verfälscht wird. Deshalb sollten nach der Transformierung nochmals folgende Probleme bedacht werden: (1) Fehlende Werte sollten angenähert oder die Datensätze komplett entfernt werden und (2) Rauschen im Datensatz soll durch Entfernen von Ausreißern reduziert werden. [57]

2.6 Etablierte Programmiersprachen für Neuronale Netze

Über die Zeit haben sich einige Programmiersprachen für Anwendungen bezüglich ML und NN etabliert. Einige dieser Programmiersprachen bieten entscheidende Vorteile, während andere Stärke durch ihre Vielseitigkeit und einfache Erlernbarkeit zeigen. Die Auswahl einer Programmiersprache hängt zudem stark von der jeweiligen Anwendung, die entwickelt werden soll, ab. Folglich gibt es keine „beste“ Programmiersprache, die für ML und NN eingesetzt werden soll. In einer Umfrage der DeveloperNation[60].

Python

Python hat mit 57% die höchste Adaptierung für ML Anwendungen und ist der de-facto Standard. Entscheidende Vorteile sind hier vor allem die zahlreichen und mächtigen Frameworks und Libraries, wie PyTorch, TensorFlow, Keras, Scikit-learn, oder NumPy. Sie erlauben einen schnellen Einstieg und eine gute Vereinheitlichung der Anwendungen. Zusätzliche Vorteile bietet Python durch die einfache Erlernbarkeit und den einfachen Umgang. [60], [61].

C++

C++ hat mit 44 % die zweithöchste Adaptierung für ML Anwendungen. Entscheidende Vorteile von C++ sind, dass es sehr schnell und effizient ist und eine hohe Kontrolle über die inhärenten Mechaniken der zu entwickelnden Anwendung bietet. Kritisch betrachtet werden kann C++ vor allem bezüglich seiner Komplexität, welche es schwierig macht die Sprache schnell zu lernen und gut zu beherrschen. [60], [61].

Java

Java hat mit 41 % die dritthöchste Adaptierung für ML Anwendungen. Obwohl Java eher für Netzwerkanwendungen geeignet ist, bietet es vor allem in der Datenverarbeitung für ML Anwendungen gute Leistung. Java bietet außerdem das WEKA Framework [62], welches gut zum Lernen von ML mit Java geeignet ist. Allerdings ist Java auch eine kritisierte Programmiersprache, die zu komplex ist und zu viele Richtlinien hat. Außerdem ist Java nicht für ML ausgelegt. Die Programmiersprache Skala wäre hier eine mögliche Alternative. [60], [61].

3 Projektvorbereitung

Zur Vorbereitung des Projektes wurden die Programmiersprache, genutzte Libraries und die Entwicklungsumgebung festgelegt, sowie eine Analyse des Problems durchgeführt.

3.1 Programmiersprache, Libraries und Entwicklungsumgebung

Aus der Analyse der etablierten Programmiersprachen in 2.6 auf der vorherigen Seite gehen die Vorteile von Python hervor. Eine schnelle und unkomplizierte Umsetzung, sowie die umfangreichen Frameworks und Libraries waren ausschlaggebend in der Entscheidung Python als Programmiersprache zu nutzen. Da die Neuentwicklung eines NN sehr zeitintensiv ist und viel Grundwissen voraussetzt, wurde auf ein Framework zurückgegriffen. Das genutzte Framework ist You Only Look Once Version 5 (YOLOv5), ein speziell für Bildverarbeitung und Objekterkennung entwickeltes Framework, dieses wird in Abschnitt 4 auf Seite 29 näher erläutert. Zusätzliche, nicht standardmäßig in Python enthaltene und teilweise auch für die Nutzung von YOLOv5 notwendige, Libraries sind folgende:

- **NumPy:** NumPy wird meist zur Umsetzung von mathematischen Lösungen verwendet. Vor allem für NNs stellt NumPy wichtige Werkzeuge für die Berechnung bereit und bietet außerordentliche Performance. [63]
- **CUDA:** Die CUDA Library ermöglicht es die Nvidia CUDA Technologie in Python zu nutzen und damit auch NNs auf der Graphics Processing Unit (GPU) zu trainieren. [64]
- **PyTorch:** PyTorch ist ein open-source ML Framework, das viele Funktionen für das Entwickeln, Trainieren und Optimieren von NNs enthält. [65]
- **PIL - Python-Pillow:** Python Pillow ist ein Fork von PIL und bietet Funktionen zur Bildbearbeitung in Python. Mit dieser Bibliothek können Bildeigenschaften verändert werden. [66]
- **Albumentations:** Albumentations stellt eine Bildverarbeitungsbibliothek speziell für Maschine Learning und Image Augmentation bereit. Albumentations bietet, wie Python-Pillow, Möglichkeiten für das Verändern von Bildeigenschaften wie Kontrast, Helligkeit, Größe oder Rotation. [67]

Außerdem wurden folgende, in Python standardmäßig enthaltene, Libraries genutzt:

- logging [68]
- os [69]
- sys [70]
- pathlib [71]
- datetime [72]
- shutil [73]
- random [74]

Entwicklungsumgebung

Als Entwicklungsumgebung wurde Visual Studio Code genutzt, da es leichtgewichtig aber dennoch mächtig ist und Python unterstützt. Die Hardware, die für das Training der NNs genutzt wurde ist folgende:

- Training
 - Intel i7 4790k
 - Nvidia Geforce GTX 980 4 GB
 - Nvidia Geforce GTX 1060 6 GB
 - Nvidia Geforce RTX 3090 24 GB
- Datenhaltung
 - Hard Disk Drive (HDD)
 - Solid State Drive (SSD)

Inwiefern sich die unterschiedliche Hardware auf Trainingszeit und Performance auswirken wurde dokumentiert und wird in Abschnitt 5 auf Seite 32 näher erläutert. Die Nvidia Geforce RTX 3090 ist hierbei die leistungsstärkste GPU, ausgestattet mit der 3. Generation Tensor-Recheneinheiten von Nvidia. Diese sind auf ML und DL spezialisiert und ermöglichen eine bis zu 20-fache Beschleunigung der Berechnungen im Vergleich zu älteren Modellen. Dennoch sollte hier darauf aufmerksam gemacht werden, dass die Nvidia Geforce RTX 3090 eine externe Ressource ist, die nicht dauerhaft in vollem Umfang zur Verfügung stand. Zum Labeln der Bilder wurde das Programm labellImg genutzt, dieses ermöglicht es graphisch Labels zu erstellen und diese in einem YOLOv5 kompatiblen Format auszugeben [75].

3.2 Konzeption und Vorgehensweise

Um die Problemstellung nachvollziehen zu können und ein Konzept zu entwickeln, war es zunächst wichtig die bereitgestellten Daten zu sichten. Da der initiale Datensatz¹ aus insgesamt 60 Bildern besteht, von denen nach Sichtung acht Bilder nicht für das Training genutzt werden können, ergibt sich ein Datensatz von 52 Bildern, die für den Trainingsprozess genutzt werden können. Die acht Bilder sind deshalb ungeeignet, da dort Walter entweder nicht vorhanden ist, oder auf dem Bild eine Sammlung von hunderten Walters abgebildet ist. Nachdem der Datensatz identifiziert wurde, wurden die Bilder manuell gelabelt, wie die entstandenen Labels strukturiert sind, wird in Anhang A auf Seite XVI erläutert. Da der Datensatz relativ klein ist, wurde hier bereits ein Konzept entwickelt, wie zusätzliche Trainingsdaten automatisiert erzeugt und gelabelt werden können. Zunächst wurde folgender Ansatz erarbeitet:

- (1) Es werden Walter-Bilder benötigt, auf denen Walters Gesicht bzw. ganzer Körper abgebildet ist;
- (2) Es werden Hintergrundbilder benötigt, auf die diese Walter-Bilder, in möglichst zufälligen Koordinaten, eingefügt werden können;
- (3) Die Walter Bilder werden, mittels eines Python Skripts (Anhang B auf Seite XVII), eingefügt und es werden automatisch Labels im YOLOv5-Format erstellt;
- (4) Die erstellten Bilder werden möglichst zufällig mit einem Python Skript (Anhang C auf Seite XVIII) in ca. 80 % Trainingsdaten und ca. 20 % Validierungsdaten aufgeteilt.

Generierte Bilder als Testdaten zu nutzen ist in diesem Fall nicht zielführend, da Walter in den Initialdaten erkannt werden soll. Damit die erstellten Daten nicht zu viel Speicherplatz einnehmen, wurden die Hintergründe auf die Walter eingefügt werden soll in ihrer Breite auf 2.000 Pixel beschränkt. Die Trainings wurden protokolliert und deren Ergebnisse anschließend qualitativ und quantitativ ausgewertet. Anhand der Ergebnisse wurden dann Anpassungen am Prozess und Vorgehen vorgenommen.

Um den Trainingsprozess und die Architektur von YOLOv5 nachvollziehen zu können wurde eine Analyse des Frameworks durchgeführt.

¹Der initiale Datensatz ist dieser Arbeit in digitaler Form beigefügt.

4 Analyse YOLOv5

YOLOv5 ist ein Objekterkennungs-Framework, basierend auf dem von der Firma Ultralytics in PyTorch umgesetzten YOLOv3. Aufgrund der hohen Geschwindigkeit und der Genauigkeit ist YOLOv5 eines der bekanntesten Frameworks für Echtzeit Objekterkennung. [76] Im Folgenden wird die Eignung von YOLOv5 für die Erkennung von Walter mittels eines NN ermittelt. Dazu werden Ursprung, Verwendbarkeit, Architektur, Funktion und Geschwindigkeit des Frameworks analysiert.

4.1 Ursprung

YOLOv5 ist die fünfte Version des Ursprünglich von Joseph Redmon 2015 entwickelten YOLO [77]. Nur ein Jahr später veröffentlichte Redmon YOLOv2 mit vielen Verbesserungen [78]. Mit YOLOv3 veröffentlichte Redmon 2018 dann eine dritte und nochmals verbesserte Version [79]. Die nächste Version, YOLOv4, wurde dann 2020 von Alexey Brokovskiy, Chien-Yao Wang und Hong-Yuan Liao veröffentlicht [80]. YOLOv5 ist als Weiterentwicklung der eigenen YOLOv3 erschienen [76].

4.2 Verwendbarkeit

Da YOLOv5 open-source ist und deshalb der gesamte Code und die Dokumentation auf GitHub sind, kann jeder das Framework nutzen [81]. Dies ist auch explizit gewünscht, da so das Framework kontinuierlich verbessert werden kann. Das GitHub Repository zeigt bei der Analyse eine hohe Aktivität mit beinahe täglichen Commits und Releases. Darüber hinaus bietet Ultralytics mit Competitions, welche ein Preisgeld von insgesamt 10.000 € auszahlen, der YOLOv5-Community einen deutlichen Ansporn zur Verwendung und Verbesserung des Algorithmus. Auf der Homepage und im GitHub Repository gibt es eine umfangreiche Dokumentation mit vielen Tutorials und Tipps zur Verbesserung des Trainings, lediglich technische Themen, wie die Architektur werden sehr oberflächlich behandelt. Das Framework wurde in Python entwickelt, was mit Unterstützung aktueller Python-Versionen eine deutliche Erleichterung in der Bedienung darstellt. Mit der Anbindung an WanDB oder Tensorboard bietet YOLOv5 eine übersichtliche Möglichkeit, Trainings-Metriken grafisch auszugeben.

4.3 Architektur

Die Architektur von YOLOv5 hat sich im Vergleich zur eigenen YOLOv3 nur wenig geändert. YOLO an sich basiert seit der ersten Version auf CNNs. YOLOv5 nutzt auch CNNs, allerdings ist das gesamte Framework in PyTorch umgesetzt worden. Die originalen Versionen 1-4 basieren auf dem DarkNet. Aufgrund einer fehlenden technischen Architekturbeschreibung ist es hier nicht möglich exakte Aussagen über die Architektur des NN zu machen. Folgendes geht aus einer Analyse des Quellcodes heraus. Das CNN von YOLOv5 besteht aus mehreren Komponenten. (1) Ankerpunkte werden genutzt, um die Detektierung der Klassen zu optimieren. (2) Ein Backbone wird für die Feature-Extraktion genutzt, hier werden alternierend CNNs verschiedener Größe und Sub-Sampling Layers genutzt. (3) Der sog. Head berechnet den Output und gibt ihn am Ende aus, er enthält die Fully-Connected Layers. Die exakte Größe des Netze, also die Anzahl der Layers ist nicht ersichtlich. Allerdings ist YOLOv5 mit mehreren vortrainierten Models ausgestattet, deren Größe unterschiedlich ist. Eine Übersicht gibt hier Tabelle 1. Hier ist zu sehen, wie sich die verschiedenen Modellgrößen auswirken. YOLOv5s ist dabei das kleinste und schnellste Modell und YOLOv5x ist das größte und genauste. Die Größen sind dabei anhand von YOLOv5l als Basis skaliert worden. Sie haben einen Einfluss darauf, wie schnell das Netz trainiert werden kann und wie hoch die Genauigkeit ist. Die Unterschied bei den Modellen mit dem Postfix 6 ist, dass diese mit einer Bildgröße von 1280, anstatt 640 Pixeln vortrainiert wurden.

Tabelle 1: Vortrainierte Models

Model	Dimension	Image-Size	Geschwindigkeit (ms)	Parameter (mio)
YOLOv5s	0.33x0.5	640 px	2.0	7.3
YOLOv5m	0.67x0.75	640 px	2.7	21.4
YOLOv5l	1.0x1.0	640 px	3.8	47.0
YOLOv5x	1.33x1.25	640 px	6.1	87.7
YOLOv5s6	0.33x0.5	1280 px	4.3	12.7
YOLOv5m6	0.67x0.75	1280 px	8.4	35.9
YOLOv5l6	1.0x1.0	1280 px	12.3	77.2
YOLOv5x6	1.33x1.25	1280 px	22.4	141.8

Hierbei sollte noch beachtet werden, dass die Geschwindigkeit in Tabelle 1 pro Bild angegeben ist und mit einer Nvidia V100 GPU berechnet wurde. Die V100 ist eine auf ML und DL hochspezialisierte GPU mit eigenem Tensor Core.

4.4 Funktion

Das YOLOv5-Framework ist sehr groß und bietet viele Funktionen. Der Fokus soll im Folgenden auf der 'train.py' und der 'detect.py' liegen. Diese werden allerdings nicht tiefgehend dargestellt, sondern lediglich deren Funktion und übergebene Parameter ausgeführt. Hierbei ist es wichtig zu erläutern, dass im Voraus die 'dataset.yaml' angepasst werden muss. Hier müssen die Pfade zu den Trainings- und Validierungsdaten, die Anzahl der Klassen und der Name der Klassen angegeben werden. Der Trainingsbefehl 'train.py' dient dazu das Netz zu trainieren. Als Parameter können ihm z. B. das Modell, die Daten (als .yaml oder als Pfad), die Epochenzahl, die Batch-Size, die Bildgröße oder das für das Training zu verwendende Gerät übergeben werden. Der Detektionsbefehl 'detect.py' dient dazu Daten erkennen zu lassen, die dem Netz übergeben werden. Als Parameter können ihm z. B. das Modell, die Daten (als Pfad), die Bildgröße oder ebenfalls das zu verwendende Gerät übergeben werden. YOLOv5 bietet außerdem viele gängigen Funktionen zur Optimierung der Objekterkennung, darunter folgende:

- **Hyperparameter Evolution:** Automatische Optimierung von Hyperparametern
- **Test-Time-Augmentation:** Gewichtete Performance über gleiche Bilder mit unterschiedlicher Augmentation
- **Frozen Layers:** Temporäres oder dauerhaftes einfrieren von bestimmten Layers, sodass sich deren Gewichte nicht mehr ändern
- **Model Ensembling:** Vereinen von verschiedenen Modellen
- **Model Pruning:** Optimieren des NNs durch entfernen von Neuronen oder Verknüpfungen

4.5 Geschwindigkeit

YOLOv5 unterstützt Nvidia-CUDA, wodurch es möglich ist das NN auf der GPU zu trainieren. Dies erhöht die Geschwindigkeit erheblich, was in Abschnitt 5 auf der nächsten Seite nochmals ausgeführt wird. Darüber hinaus bietet YOLOv5 zahlreiche Schnittstellen und Tutorials, um das NN in der Cloud trainieren zu können. Im direkten Vergleich mit YOLOv4 ist nicht klar, welche der beiden Frameworks schneller und genauer ist, dies hängt auch von der Anwendung und den Daten ab. Dennoch wird YOLOv5 stetig weiterentwickelt und es werden immer wieder neue Funktionen hinzugefügt, welche das Framework beschleunigen und genauer machen.

5 Training und Validierung des Neuronalen Netzes

Für das Training der NNs wurde ein Workflow entwickelt, der den gesamten Trainingsprozess inklusive der Erzeugung der Trainingsdateien abbildet.

5.1 Workflow

Der in Abbildung 11 dargestellte Trainingsprozess ist als Schleife entwickelt worden. Hierbei ist jede Iteration eine Möglichkeit zur Verbesserung der Trainingsbedingungen. Zu Beginn wird überprüft, ob die vorhandenen Trainingsbilder ausreichend sind, bei der ersten Iteration ist diese Abfrage hinfällig. Sollten die Trainingsbilder nicht ausreichend sein, ist es notwendig neue Walter-Bilder und Hintergrundbilder für die Generierung hinzuzufügen. Daraufhin wird ein vortrainiertes Modell ausgewählt, das zum Trainieren genutzt wird. Diesem Modell werden dann die Parameter übergeben und das Training mit 'train.py' gestartet. Nachdem das Training des Modells abgeschlossen ist, wird mit 'detect.py' das Training evaluiert. Hier wird das Modell auf die Testdaten angewendet und evaluiert, wie viele true-positives und wie viele false-positives erkannt wurden. Anhand der Evaluation und einer Analyse der Metriken wird dann entschieden, ob die Parameter des Trainings angepasst werden müssen, oder, ob noch mehr und qualitativ hochwertigere Trainingsbilder erzeugt werden müssen. Falls große Veränderungen der Parameter oder Trainingsbilder vorgenommen wurden, wurde dies in dieser Arbeit als neue Generation deklariert.

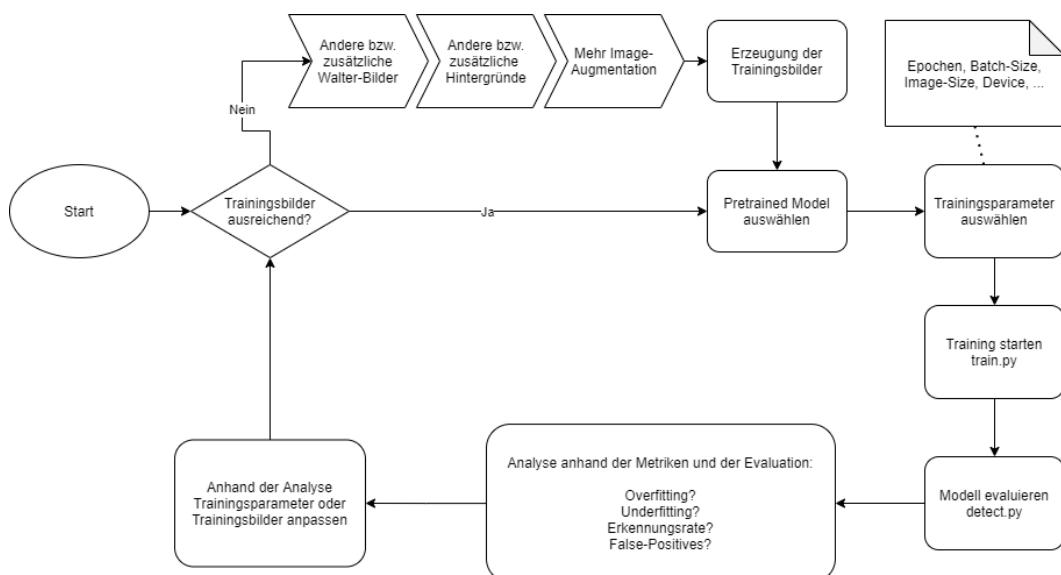


Abbildung 11: Workflow Trainingsprozess

5.2 Generation 1

Vorgehensweise

In dieser ersten Generation wurde zunächst analysiert, wie ein Modell trainiert und welche Ergebnisse ausschließlich mit dem initialen Datensatz erzielt werden können. Die verwendeten Parameter sind Tabelle 2 zu entnehmen. Es wurde YOLOv5s als Modell genutzt, da dies das schnellste Modell ist und so schnell Ergebnisse erzielt werden können. Der initiale Datensatz wurde aufgeteilt, wobei 29 Trainingsbilder und sechs Validierungsbilder verwendet wurden. Der Rest wurde zum Testen genutzt. Als Epochenzahl wurde 16 gewählt, da ab einer Epochenzahl von 25 die Wahrscheinlichkeit von Over- bzw Underfitting stark ansteigt. Dieses Modell wurde auf dem Intel Prozessor trainiert.

Tabelle 2: Parameter Generation 1

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	29	6	16	12	640 px

Ergebnis

Die Ergebnisse der ersten Generation sind Tabelle 3 zu entnehmen. Die Metriken der ersten Generation sind im Anhang E auf Seite XXI abgebildet. Mit einer max. Precision von 0.19 und einem max. Recall von 0.33 ist das Modell nicht im Stande Walter zu erkennen. Dennoch ist in der Recall- und Precision-Metrik ein nicht monotoner Aufwärtstrend zu erkennen. Auch mAP steigt ab der 13. Epoche harmonisch an. Der Abwärtstrend der Loss-Metrik ist harmonisch und die Trainings- und Validierungswerte scheinen zu konvergieren, was auf eine grundsätzlich hohe Generalisierung schließen lässt.

Tabelle 3: Ergebnisse Generation 1

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/640px	detect@0,5/1280px	true-pos	false-pos	true-pos	false-pos
ca. 2 min	0,19	0,33	0	0	0	0	0	0

Probleme

Das schwerwiegendste Problem in dieser Generation ist, dass viel zu wenig Daten vorhanden sind. Dadurch kann das Modell nicht hinreichend trainiert werden. Obwohl ein Trend im Loss zu erkennen ist, ist das Modell nicht gut genug. Zudem ist die Aufteilung des Datensatzes nicht optimal, diese wurde allerdings so gewählt, dass evaluiert werden kann, wie die grundsätzliche Performance des Systems ist. Ein weiteres Problem ist, dass das Modell sehr langsam ist. Für 35 Bilder im Trainingsprozess ca. 2 min Trainingszeit pro Epoche ist sehr viel. Denn bei 35.000 Bildern skaliert sich die Trainingszeit auf 2.000 min, also über 33 h pro Epoche. Ein weiterer limitierender Faktor ist der Arbeitsspeicher. Dadurch, dass die für das Training genutzten Bilder und ein Abbild der NN im Speicher liegen, limitiert die Image-Size die Batch-Size. Eine kleinere Batch-Size reduziert wiederum die Geschwindigkeit des Trainings.

Lösung

Um wesentlich mehr Daten zum Trainieren nutzen zu können, wurde der in Abschnitt 3.2 auf Seite 28 vorgestellte Algorithmus implementiert. Hierzu wurden zunächst Walter-Köpfe ausgeschnitten und „Realwelt“ Hintergründe gesammelt. Diese wurde dann anhand des in Anhang B auf Seite XVII ausgeführten Algorithmus zusammengeführt und gelabelt. Dadurch konnten sehr viele zusätzliche Daten generiert werden. Durch die größere Menge an Daten können diese auch viel sinniger aufgeteilt werden, wodurch das Modell effizienter trainiert werden kann. Um die Geschwindigkeit zu verbessern, wurde vom Prozessor zur GPU gewechselt. Hierbei ist klar, dass unterschiedliche GPUs verwendet werden, wodurch Trainingszeiten variieren können. Die Problematik mit der Batch-Size wird durch den Wechsel zur GPU teilweise noch verschlimmert, da die GPUs eher weniger Speicher haben, als der Arbeitsspeicher des Computers. Allerdings muss angemerkt werden, dass auch der Arbeitsspeicher als Auslagerungsspeicher genutzt wird, wodurch sich die Problematik wiederum abschwächt.

Durch diese vielen Änderungen, wird die nächste Generation initiiert.

5.3 Generation 2

2a

Diese Generation wurde auf einem Leistungsstarken Computer mit 64 GB Arbeitsspeicher und der Nvidia RTX 3090/24 GB als GPU trainiert.

Vorgehensweise

Um die Trainingsdaten zu erzeugen, wurde Walter aus einem Bild des Initialdatensatzes ausgeschnitten. Dieser Walter wurde in verschiedenen Größen und Positionen auf einen weißen Hintergrund eingefügt. Durch das Skript wurden ca. 400.000 Bilder erzeugt, wovon allerdings nur 750 für das Training und 100 für die Validierung genutzt wurden. Aufgrund der Leistung des genutzten Rechners wurde die Epochenzahl und die Batch-Size erhöht. Außerdem wurde auch hier wieder YOLOv5s als vortrainiertes Modell genutzt. Die Parameter sind Tabelle 4 zu entnehmen.

Tabelle 4: Parameter Generation 2a

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	750	100	50	64	640 px

Ergebnis

In Tabelle 5 sind die Ergebnisse dieses Trainingslaufs aufgeführt. Zu erkennen ist hierbei, dass die Precision und der Recall bei einem Wert von 1 liegen, was bedeutet, dass jedes Validierungsbild zu 100% erkannt wird. Aufgrund der Einfachheit der Bilder können diese Werte nachvollzogen werden. Denn Walter hebt sich hier sehr deutlich vom Hintergrund ab. Allerdings wird Walter in den Testbildern nicht erkannt. Die Metriken in Anhang F.1 auf Seite XXII sind zwar viel besser als die der letzten Generation, allerdings fällt auf, dass die stärkste Verbesserung bis zur Epoche 11 stattfindet.

Tabelle 5: Ergebnisse Generation 2a

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/640px	detect@0,5/1280px
ca. 30 s	1	1	0 %	0 %

Probleme

Das Skript erzeugte einen Datenpool von über 400.000 Bildern, wovon nur ein Bruchteil für das Training verwendet wurde. Aufgrund der Datenmenge und Bildgröße wird sehr viel Speicherplatz für das Erzeugen der Daten benötigt. Für die zukünftige Erzeugung wurde dies als Problem gesehen, welches immer schwerwiegender werden würde. Die schlechte Erkennung der Wimmelbilder beruht wahrscheinlich darauf, dass Hintergründe nur einfarbig in weiß gewählt wurden. Diese sind dementsprechend nicht repräsentativ im Hinblick auf die tatsächlichen Testdaten. Denn dort besteht, durch den hohen Detailgrad, viel „Rauschen“, wodurch sich Walter nicht mehr so stark vom Hintergrund abhebt.

Lösung

Als Lösungsansatz für die Reduzierung des Speicherbedarfs wurden die Parameter des Skriptes angepasst. Dadurch können die Zahl der Positions- und Größenvariationen verringert werden. Dadurch werden pro Erzeugungsdurchlauf weniger Daten erzeugt und somit weniger Speicherplatz benötigt. Das Verringern des Datenpools kann aufgrund der weiterhin vielen Bildvariationen durchgeführt werden, ohne die Variationsvielfalt negativ zu beeinflussen. Um die Erkennung der Wimmelbilder zu verbessern werden im nächsten Trainingslauf weitere Hintergründe integriert. Diese sollen die Repräsentanz der Daten im Bezug auf die Testdaten erhöhen.

2b

Vorgehensweise

In diesem Trainingslauf wurden einige Parameter des Skriptes angepasst und mehrere Hintergrundbilder hinzugefügt. Somit wurden ca. 100.000 Bilder erzeugt, von diesen wurden ca. 1.700 für den Trainingsprozess genutzt. Die Parameter dieses Laufs sind Tabelle 6 zu entnehmen. Hier wurde die Image-Size erhöht, wodurch die Batch-Size reduziert werden musste. Die generierten Daten haben Hintergründe, welche reale Umgebungen, wie Parks, Innenstätte oder Gebäude abbilden.

Tabelle 6: Parameter Generation 2b

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	1500	200	50	24	1280 px

Ergebnis

Trotz der guten Ergebnisse aus Tabelle 7 wurde Walter nicht erkannt. Gegenüber der Generation 2a hat sich die Erkennung von erzeugten Daten also nicht geändert. Auch die Erhöhung der Image-Size hat in diesem Fall nichts geändert, wobei nicht klar ist, ob das Problem an der Image-Size per se liegt. Die mAP- und Loss-Metriken im Anhang F.2 auf Seite XXIII sind nicht monoton steigend, bzw. fallend, was darauf hindeutet, dass das Modell nicht viel mit den Daten anfangen kann und letztendlich nicht viel lernt.

Tabelle 7: Ergebnisse Generation 2b

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/640px true-pos	detect@0,5/1280px false-pos	detect@0,5/1280px true-pos	detect@0,5/1280px false-pos
ca. 1 min	1	1	0 %	0 %	0 %	0 %

Probleme

Das Hauptproblem in diesem Lauf ist, dass Walter immer noch nicht erkannt wird. Zusätzlich dazu ist die Varianz der generierten Bilder noch zu niedrig.

Lösung

Die Ergebnisse dieses Trainingslaufs lassen darauf schließen, dass die Variationen und das genaue Aussehen des Gesichtes durch Image-Augmentation verändert werden muss. Mit Änderung von Bildeigenschaften wie Kontrast, Farben, Helligkeit und Drehung werden Gesichtsvariationen generiert.

2c

Vorgehensweise

In diesem Lauf wurden Image-Augmentations auf die generierten Bilder angewendet, um die Varianz zu erhöhen. Das Walter-Gesicht wurde auf verschiedenen Hintergründen mit Veränderung von Rotation, Helligkeit und Kontrast eingefügt und teilweise vertikal gespiegelt. Die Parameter dieses Laufs sind Tabelle 8 zu entnehmen.

Tabelle 8: Parameter Generation 2c

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	1500	200	50	24	1280 px

Ergebnis

Trotz der vorgenommenen Änderungen wurde Walter auch in diesem Lauf nicht erkannt. Auch die Metriken im Anhang F.3 auf Seite XXIV zeigen ein sehr unstimmiges Bild. Hier fällt vor allem der unregelmäßige Verlauf der mAP und Loss-Metrik auf. Dies lässt auf qualitativ schlechte Trainings- und Valiierungsdaten schließen.

Tabelle 9: Ergebnisse Generation 2c

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/640px true-pos	detect@0,5/640px false-pos	detect@0,5/1280px true-pos	detect@0,5/1280px false-pos
ca. 9,5 sec	1	1	0 %	0 %	0 %	0 %

Probleme

Trotz Image Augmentation besteht weiterhin das Problem, dass Walter nicht erkannt wird. Zusätzlich dazu zeigen die Metriken ein unstimmiges Bild.

Lösung

Entsprechend der Problematiken in diesem Lauf, wird der Generierungsprozess überarbeitet und angepasst. Außerdem werden mehr Walter-Köpfe und Walter-Körper ausgeschnitten, sowie vielfältigere vor allem Comic-Hintergründe gesammelt. Die Comic-hintergründe heben sich strukturell und farblich nicht sehr von den Walter-Bildern ab, wodurch das Modell besser lernen könnte. Diese Maßnahmen werden die Qualität der generierten Daten erhöhen. Zusätzlich dazu werden mehr Daten für den Trainingsprozess genutzt.

5.4 Generation 3

3a

Vorgehensweise

Die neuen Lösungsansätze aus Generation 2 wurden hier umgesetzt. Für die Trainingsdatengenerierung wurden speziell verschiedene Comic-Hintergründe herausgesucht. Außerdem wurde ein großer Datensatz an Walter Bildern angelegt, hier sind Walters Gesicht und auch die ganze Figur abgebildet. Insgesamt wurden so ca 120.000 Trainingsbilder aus 55 Walter-Bildern und 18 Hintergründen erzeugt. Für die Image-Augmentation wurde die Rotationsfunktion deaktiviert, weil diese Probleme verursacht hat. Von den Trainingsbildern wurden 80 % entfernt und im Verhältnis 85 % Training / 15 % Validierung aufgeteilt. Die Parameter sind Tabelle 10 zu entnehmen.

Tabelle 10: Parameter Generation 3a

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	20883	3456	16	12	640 px

Ergebnis

Die Ergebnisse des ersten Trainings dieser Generation sind Tabelle 11 zu entnehmen. Der Trainingslauf war ursprünglich auf 16 Epochen angesetzt, wurde allerdings schon nach acht Epochen abgebrochen, da die Trainingszeit, mit 3,5 h pro Epoche zu lang ist. Mit einer max. Precision von 0.989 und einem max. Recall von 0.807 ist das Modell im Stande Walter teilweise zu erkennen. Die Erkennungsrate ist allerdings nicht sehr hoch und es werden viele false-positives (Anhang D.2 auf Seite XIX) markiert, außerdem gibt es auch viele Bilder auf denen nichts erkannt wird. Die Metriken, im Anhang G.1 auf Seite XXV, zeigen hier ein nochmals stimmigeres Bild als noch in der vorherigen Generation. Nicht nur die Werte, sondern auch die Trends zeigen, dass das NN lernt. Der Abwärtstrend der Loss-Metrik ist zwar harmonisch, allerdings scheinen die Trainings- und Validierungswerte eher parallel zu laufen, was auf ein Underfitting hinweist.

Tabelle 11: Ergebnisse Generation 3a

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/640px	detect@0,5/1280px
ca. 3,5 h	0,989	0,807	13 %	46 %

Probleme

Obwohl die Ergebnisse dieses Trainings vielversprechend sind, gibt es hier wieder einige Probleme. Zunächst ist die sehr hohe Trainingszeit ein großes Problem, da so keine schnellen Erfolge erzielt werden können. Durch den Wechsel zur GPU wurde hier auf ein anderes Problem aufmerksam gemacht, die HDD. Durch den geringen Datendurchsatz der Festplatte, konnten die Bilder nicht schnell genug geladen werden, wodurch das Training unnötig verzögert wird. Ein anderes Problem ist, dass die Rotationsfunktion der Image Augmentation fehlerhaft ist und Walter dann doppelt anzeigt (zu sehen in Anhang D.3 auf Seite XIX). Zusätzlich dazu ist die Menge an Daten, von denen dann wieder 80 % gelöscht wird, zu hoch. Bei Nutzung aller Daten hätte eine Epoche mind. 12 h gedauert.

Lösung

Damit die Trainingsläufe schneller werden, wird von einer HDD zu einer SSD gewechselt. Außerdem wurde davon abgesehen die Rotationsfunktion im weiteren Verlauf zu nutzen. Die Walter und die Hintergründe werden für das nächste Training reduziert. Da die Erkennungsrate dennoch niedrig ist, wird die Trainingsbildgröße erhöht.

3b

Vorgehensweise

Im zweiten Trainingslauf wurden neue Trainingsdaten generiert, diese wurden auf der SSD gespeichert. Mit 47 Walter-Bildern und 17 Hintergründen wurden ca. 64.000 Bilder erzeugt, von diesen wurden dann 66 % entfernt. Die übrigen Bilder wurden wieder im Verhältnis 85 % Training / 15 % Validierung aufgeteilt. Die Rotationsfunktion wurde deaktiviert. In diesem Trainingslauf wurde zudem die Image-Size auf 1024 erhöht, um zu testen, welche Auswirkung dies auf die Ergebnisse hat. Dies hat zur Folge, dass die Batch-Size reduziert werden muss. Die Parameter sind Tabelle 12 zu entnehmen.

Tabelle 12: Parameter Generation 3b

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	17755	3281	16	6	1024 px

Ergebnis

Die Ergebnisse dieses Trainingslaufs sind Tabelle 13 zu entnehmen. Hier wird deutlich, dass der Wechsel zur SSD sehr positive Auswirkungen auf die Trainingszeit hat. Mit einer max. Precision von 0.999 und einem max. Recall von 0.984 ist das Modell im Stande Walter besser zu erkennen. Die Erkennungsrate ist zwar höher als im letzten Trainingslauf, allerdings werden immer noch viele false-positives markiert. Außerdem gibt es auch hier viele Bilder auf denen nichts erkannt wird. Die Metriken, im Anhang G.2 auf Seite XXVI, zeigen hier eine deutliche Verbesserung im Vergleich zum letzten Trainingslauf. Auffallend ist, dass mAP und der Recall sehr stark gestiegen sind. Außerdem ist zu erkennen, dass der Loss wieder leicht konvergiert. Anzunehmen ist, dass sich die Erhöhung der Image-Size positiv auf die Performance auswirkt. Das Training wurde nach 12 Epochen abgebrochen, um einen weiteren optimierten Trainingslauf zu starten.

Tabelle 13: Ergebnisse Generation 3b

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/640px true-pos	detect@0,67/1024px false-pos	detect@0,67/1024px true-pos	detect@0,67/1024px false-pos
ca. 0,3 h	0,999	0,984	27 %	33 %	40 %	60 %

Probleme

Obwohl die Ergebnisse dieses Trainings etwas besser sind als im letzten Lauf, gibt es hier wieder einige Probleme und Überlegungen. Ein Problem ist die immer noch relativ schlechte Erkennungsrate, sowie hohe false-positives Rate. Außerdem werden noch viele der Bilder verworfen. Zusätzlich dazu ist ein Fehler bei der Image-Augmentation aufgetreten, der durch das Verändern des Kontrast der Bilder hervorgerufen wird. Die Auswirkung ist im Anhang D.4 auf Seite XX zu sehen. Die Walter, die auf den Hintergrund eingefügt werden, sind hier nur als Schatten zu sehen, wodurch das NN falsche Parameter lernt.

Lösung

Als Test und um die Erkennungsrate zu verbessern, wird die Image-Size weiter erhöht, was womöglich die Batch-Size wieder verringert. Außerdem werden die generierten Daten nicht mehr verworfen, dadurch wird sich die Epochenzeit womöglich erhöhen. Zusätzlich dazu werden die Image-Augmentation Parameter angepasst, damit die Probleme und Auswirkungen verringert werden.

3c

Vorgehensweise

Im dritten Trainingslauf wurden mit den neuen Parametern neue Trainingsdaten generiert. Mit 47 Walter-Bilder und 17 Hintergründen wurden ca. 45.000 Bilder erzeugt. Diese wurden wieder im Verhältnis 85 % Training / 15 % Validierung aufgeteilt. In diesem Trainingslauf wurde die Image-Size auf 1440 erhöht. Dies hat zur Folge, dass die Batch-Size auf 2 reduziert werden muss. Dies wird die Epochenzzeit stark beeinflussen. Die Parameter sind Tabelle 14 zu entnehmen.

Tabelle 14: Parameter Generation 3c

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	36975	6540	16	2	1440 px

Ergebnis

Die Ergebnisse dieses Trainingslaufs sind Tabelle 15 zu entnehmen. Mit einer max. Precision von 0.999 und einem max. Recall von 0.993 ist das Modell im Stande Walter viel besser zu erkennen. Die Erkennungsrate ist deutlich besser als zu Beginn dieser Generation, allerdings werden weiterhin viele false-positives markiert. Die Metriken, im Anhang G.3 auf Seite XXVII, zeigen erneut eine Verbesserung im Vergleich zum letzten Trainingslauf. Auffallend ist, dass die Werte nach der ersten Epoche bereits gut sind und sich im Lauf des Trainings stark verbessern. Hier wird bestätigt, dass die Image-Size die Performance beeinflusst, allerdings könnte dies auch an der Erhöhung der Bildmenge liegen. Es ist auch auffallend, dass die Epochenzzeit mit 1.25 h stark gestiegen ist.

Tabelle 15: Ergebnisse Generation 3c

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,75/1280px	detect@0,8/2048px
ca. 1,25 h	0,999	0,993	33 %	60 %

Probleme

Auch wenn die Erkennungsraten gestiegen sind, ist auffällig, dass sich die false-positives ebenfalls erhöhen. Der Trend, dass eine höhere Image-Size bei der Detektion weniger false-positives hervorruft, sollte überprüft werden. Zusätzlich dazu tritt ein weiteres Problem auf, welches mit der Erkennung von Walter zusammenhängt. Teilweise werden nicht nur Menschen, sondern auch längliche Objekte und gestreifte Objekte vom NN als Walter erkannt. Eine visuelle Repräsentation des Problems kann in Anhang D.5 auf Seite XX eingesehen werden.

Lösung

Um die Erkennungsrate zu verbessern und die false-positives zu reduzieren, wurde als Lösungsansatz der Wechsel der Hintergründe vorgeschlagen. Hier werden alle bestehenden Hintergründe durch Bildausschnitte des Initialdatensatzes ersetzt. Dadurch unterscheiden sich die Walter-Bilder und die Hintergrundbilder farblich und strukturell nicht mehr so stark. Außerdem wird dabei darauf geachtet, dass die Hintergründe richtig skaliert sind, da so eventuell dem Problem mit den länglichen Objekten vorgebeugt werden kann. Zusätzlich dazu werden dem Trainings- und Validierungsdatensatz mehr Bilder des originalen Datensatzes beigefügt, sodass beim Training mehr reale Daten verwendet werden.

Vor Umsetzung der Lösungsvorschläge, wurde ein weiterer Trainingslauf gestartet mit einer Image-Size von 2048 px. Da die Epochenzzeit, aufgrund reduzierter Batch-Size, allerdings über acht Stunden betrug, wurde hier von einem Fortfahren des Trainings abgesehen. Nach nur einer Epoche wurde das Training deshalb abgebrochen. Die jeweiligen Metriken und die Performance sind daher ungültig.

Die Änderungen durch den letzten Trainingslauf initiieren die nächste Generation.

5.5 Generation 4

4a

Vorgehensweise

Die neuen Lösungsansätze aus Generation 3 wurden hier umgesetzt. Mit 47 Walter-Bildern und 19 Walter-Comic-Hintergründen, wurden über 65.000 neue Trainingsprozessbilder erzeugt. Von diesen wurden 25 % ausgesondert, sodass noch ca. 48.750 Bilder für den Trainingsprozess übrig waren. Diese wurden wieder im Verhältnis 85 % / 15 % aufgeteilt. Zusätzlich wurden noch mehr Bilder vom Initialdatensatz in den Trainingsprozess aufgenommen. Die Parameter sind Tabelle 16 zu entnehmen.

Tabelle 16: Parameter Generation 4a

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	42849	7142	16	2	1440 px

Ergebnis

Die Ergebnisse dieser Generation sind Tabelle 17 zu entnehmen. Die Metriken dieser Generation sind im Anhang H.1 auf Seite XXVIII abgebildet. Mit einem Recall und einer Precision nahe 1 ist die Performance dieses Modells sehr hoch. Dies bildet sich auch in der Erkennungsrate ab. Hier wurden mit einer relativ hohen Konfidenz von 0.75 und 0.8 die Detektion ausgeführt. Zu sehen ist, dass die Erkennungsrate zwar ähnlich geblieben ist, die Anzahl an false-positives allerdings stark zurückgegangen ist. In der Loss-Metrik ist zu sehen, dass die Loss-Werte stark konvergieren, auch mAP ist im Vergleich zur letzten Generation wieder gestiegen.

Tabelle 17: Ergebnisse Generation 4a

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/2048px true-pos	detect@0,8/2048px false-pos	detect@0,8/2048px true-pos	detect@0,8/2048px false-pos
ca. 2:30 h	0,999	0,999	53 %	20 %	40 %	7 %

Probleme

Obwohl die false-positives reduziert werden konnten, ist die Erkennungsrate noch nicht hoch genug. Bei der Detektion mit anderen und niedrigeren Konfidenzen können zwar unterschiedliche Ergebnisse erzielt werden, dennoch sollte Walter öfter erkannt werden.

Lösung

Eine Möglichkeit die Erkennungsrate des Modells zu erhöhen wäre es, das Anfangsmodell zu wechseln. Entsprechend dessen wird im nächsten Lauf YOLOv5m genutzt.

4b

Vorgehensweise

Um bessere Ergebnisse zu erzielen wird in diesem Lauf von YOLOv5s zu YOLOv5m gewechselt. YOLOv5m ist ein größeres NN und kann somit besser auf die Erkennung von Walter trainiert werden. Die Trainingsbilder wurden beibehalten. Ursprünglich wurde die Image-Size 1472 px eingestellt. Auch hier war die GPU die Limitierung, weshalb eine Image-Size von 1024 px genutzt werden musste.

Tabelle 18: Parameter Generation 4b

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5m.pt	42849	7142	16	2	1024px

Ergebnis

Die Ergebnisse dieses Trainingslaufs sind Tabelle 19 zu entnehmen. Die Metriken des Laufs sind im Anhang H.2 auf Seite XXIX abgebildet. Mit einer Precision und einem Recall von nahe 1, ist dieses Modell in der Lage Walter zu erkennen. Die Ergebnisse sind allerdings eher dürftig, vor allem im Vergleich mit dem letzten Lauf. Dies dürfte wohl auf die Epochenzahl zurückzuführen sein. Da YOLOv5m größer ist, ist es auch träger und braucht länger, um die Gewichte anzupassen. Dennoch zeigt sich vor allem in der Loss-Metrik, dass das Modell sehr gut konvergiert.

Tabelle 19: Ergebnisse Generation 4b

Trainingszeit	max.	max.	detect@0,5/1024px		detect@0,5/2048px	
pro Epoche	Precision	Recall	true-pos	false-pos	true-pos	false-pos
ca. 2,5 h	1	0,999	47 %	40 %	26 %	40 %

Probleme

Probleme bereitet dieses Modell vor allem in der tatsächlichen Fähigkeit Walter zu erkennen, hier sollte überprüft werden, ob mit zusätzlichen Epochen nachgebessert werden kann.

Lösung

Im nächsten Trainingslauf wird das Modell 25 Epochen lang trainiert.

4c

Vorgehensweise

Um zu evaluieren, ob mit höherer Epochenzahl in diesem Fall bessere Ergebnisse erzielt werden können, wird in diesem Lauf die Epochenzahl auf 25 erhöht. Die Trainingsbilder wurden beibehalten. Die Parameter sind Tabelle 20 zu entnehmen.

Tabelle 20: Parameter Generation 4c

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5m.pt	42849	7142	25	2	1024px

Ergebnis

Tabelle 21 zeigt die Ergebnisse dieses Laufs. Die Metriken des Laufs sind im Anhang H.3 auf Seite XXX abgebildet. Mit einer hohen Erkennungsrate und wenigen false-positives sind die Ergebnisse dieses Laufs sehr gut. Auch die Metriken liefern ein ähnliches Bild. Hier zeigt sich, dass sich die Metriken, im Vergleich zum letzten Lauf, verbessert haben.

Tabelle 21: Ergebnisse Generation 4c

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/1024px true-pos	detect@0,5/2048px false-pos	true-pos	false-pos
ca. 2,5 h	1	1	60 %	20 %	66 %	13 %

Probleme und mögliche Lösungen

Probleme sind in diesem Lauf nicht aufgetreten, allerdings ist zu erkennen, dass eine Erhöhung der Epochenzahl über 25 nicht viel bringen würde, da die Loss-Metrik bereits gesättigt ist.

5.6 Zusatzgenerationen

Die nachfolgend aufgeführten Trainingsläufe wurden durchgeführt, um Thesen zu überprüfen und Möglichkeiten zur Optimierung herauszufinden.

5.6.1 Lernrate erhöht

Vorgehensweise

Um zu testen, wie das NN auf eine wesentlich höhere Lernrate reagiert, wurde die Lernrate von ursprünglich 0.01 auf 0.15 angehoben. Eine zu hohe Lernrate kann, wie in Abschnitt 2.4.2 auf Seite 19 bereits ausgeführt, zu Overfitting führen. Die Parameter des Tests sind in Tabelle 22 aufgeführt.

Tabelle 22: Parameter Zusatzgeneration Lernrate erhöht

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	42849	7142	12	2	1440px

Ergebnisse

Die Ergebnisse dieses Testlaufs sind in Tabelle 23 aufgeführt. Trotz der hohen Lernrate sind die Ergebnisse vielversprechend. Mit einer Precision und einem Recall nahe 1 ist das Modell gut in der Lage Walter zu erkennen. Die Erkennungsraten sind besser als bei Generation 4b und fast so gut wie Generation 4c. Die Metriken in Anhang I.1 auf Seite XXXI wirken etwas wirr, dies kann mitunter an der reduzierten Epochenzahl liegen. Hier schön zu sehen ist, dass die Loss-Metriken zu konvergieren scheinen.

Tabelle 23: Ergebnisse Zusatzgeneration Lernrate erhöht

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,75/1440px true-pos	detect@0,8/2048px false-pos	true-pos	false-pos
ca. 2,5 h	0,999	0,999	46 %	33 %	53 %	20 %

Probleme und mögliche Lösungen

Probleme sind in diesem Lauf nicht aufgetreten. Allerdings kann mit einer höheren Epochenzahl möglicherweise das Ergebnis noch deutlich verbessert werden.

5.6.2 Wilma

Vorgehensweise

Um das Netz zu erweitern und sowohl Walter, als auch Wilma erkennen zu lassen, wurde der Initialdatensatz erneut gelabelt. Dieses Mal wurden Walter und Wilma gelabelt. Dadurch musste auch das Netz in der dataset.yaml um eine Klasse erweitert werden. Zunächst wurden mit 50 Epochen, dann mit 250 und schließlich mit 1.000 Epochen trainiert. Da die Daten, mit denen trainiert wurde, nur aus dem Initialdatensatz stammen, wird das Ergebnis nicht sehr vielversprechend sein. Die Parameter sind Tabelle 24 zu entnehmen.

Tabelle 24: Parameter Zusatzgeneration Wilma

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	35	6	50 / 250 / 1000	6	1472px

Ergebnisse

Die Ergebnisse des ersten Laufs mit 50 Epochen überzeugen nicht, diese sind Tabelle 25 zu entnehmen. Hier wird weder Walter noch Wilma erkannt. Auch den Metriken kann man nicht viel entnommen werden, diese sind im Anhang I.2.1 auf Seite XXXII zu sehen. Es fällt lediglich auf, dass mAP gegen Ende leicht steigt und, dass die Loss-Metrik relativ parallel verläuft. Ebenso wenig kann der Lauf mit 250 Epochen, dessen Ergebnisse in Tabelle 26 auf der nächsten Seite zu sehen sind, überzeugen. Auch hier werden weder Walter noch Wilma erkannt. Die Metriken (Anhang I.2.2 auf Seite XXXIII) geben Aufschluss darüber, weshalb das so sein könnte. Die mAP steigt zwar, fluktuiert allerdings immer wieder stark. Das selbe gilt für die Precision, die immer wieder von 1 auf 0 fällt, nur um dann wieder auf 1 zu steigen. Der Recall ist ebenfalls sehr unruhig. Lediglich die Loss-Metrik, welche auch teilweise unruhig ist, zeigt, dass das Netz immer noch versucht zu lernen. Erkannt werden Wilma und Walter lediglich im dritten Lauf, zu sehen in Anhang D.1 auf Seite XIX. Nach 1.000 Epochen werden zwar nur wenige

Tabelle 25: Ergebnisse Zusatzgeneration Wilma 50 Epochen

Trainingszeit pro Epoche	max. Precision	max. Recall	Walter@0,67/1472px true-pos	Wilma@0,8/1472px false-pos	true-pos	false-pos
ca. 1 min	0,25	0,07	0 %	0 %	0 %	0 %

Tabelle 26: Ergebnisse Zusatzgeneration Wilma 250 Epochen

Trainingszeit pro Epoche	max. Precision	max. Recall	Walter@0,5/1472px true-pos	Wilma@0,8/1472px false-pos	true-pos	false-pos
ca. 1 min	1	0,58	0 %	0 %	0 %	0 %

Tabelle 27: Ergebnisse Zusatzgeneration Wilma 1000 Epochen

Trainingszeit pro Epoche	max. Precision	max. Recall	Walter@0,5/1472px true-pos	Wilma@0,8/1472px false-pos	true-pos	false-pos
ca. 1 min	1	0,58	13 %	0 %	13 %	0 %

erkannt, dafür gibt es jedoch auch keine false-positives. Die Ergebnisse sind Tabelle 27 zu entnehmen. Die Metriken in Anhang I.2.3 auf Seite XXXIV zeigen, dass der fluktuierende Trend fortgesetzt wird. Es sind dennoch klare Aufwärtstrends in mAP und Precision zu erkennen. Außerdem fällt auf, dass der Validierungs-Loss gesättigt ist und das Modell zum Overfitting tendiert.

Probleme

Das Hauptproblem in diesen Läufen ist, dass zu wenig Daten vorhanden sind. Wenn hier mehr Bilder mit Walter und Wilma genutzt werden, sollte die Performance stark zunehmen. Des Weiteren zeigt sich gegen Ende des dritten Laufs, dass das Modell anfängt zu overfitten, was schlecht ist. Dies könnte zwar auch mit der geringen Anzahl an Daten zusammenhängen, wird aber vor allem durch die Epochenzahl verursacht.

Lösung

Um die Probleme dieser Läufe zu lösen, sollte die Anzahl an Daten im Trainingsprozess stark erhöht werden. Gleichzeitig kann mit zunehmenden Daten die Epochenzahl reduziert werden, um Overfitting zu verhindern, bzw. diesem vorzubeugen.

5.6.3 Epochen-Vergleich

In diesem Training wurde die Theorie überprüft, ab welcher Anzahl von Epochen das Modell „gesättigt“ ist bzw. keine bedeutenden Verbesserungen mehr erreicht oder sogar Verschlechterungen hervorgerufen werden.

Vorgehensweise

Um eine Überprüfung durchzuführen wurden drei Trainings mit identischen Parametern durchgeführt, diese sind Tabelle 28 zu entnehmen. Lediglich die Epochen wurden schrittweise von 16 über 30 auf 50 erhöht. In diesem Training wurde die Vorgehensweise von Generation 3 umgesetzt.

Tabelle 28: Parameter Zusatzgeneration Epochenvergleich

Model	TrainImg	ValidateImg	Epochen	Batch-Size	Image-Size
yolov5s.pt	600	60	16 / 30 / 50	6	2560px

Ergebnisse

Die Metriken der verschiedenen Trainingsläufe (Anhang I.3.1 auf Seite XXXV bis Anhang I.3.3 auf Seite XXXVII) zeigen nur marginale Unterschiede. Dementsprechend sind auch die max. Precision und der max. Recall in Tabelle 29 bis Tabelle 31 auf der nächsten Seite durchgängig gleich. Lediglich in den Erkennungsraten zeigen sich Unterschiede, denn diese verbessert sich zunächst, verringert sich dann allerdings wieder. Wobei hier mit verbesserter Erkennungsrate auch die false-positives ansteigen. Für eine gute Erkennung müssen also anstatt der Erhöhung der Epochenzahl, andere Optimierungen durchgeführt werden. Die dargestellten Ergebnisse zeigen, dass eine Epochenzahl von 16 (möglicherweise bis unter 30) für das Training aussagekräftig genug ist und somit für weitere Trainings nicht verändert werden muss.

Tabelle 29: Ergebnisse Zusatzgeneration Epochenvergleich - 16 Epochen

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/2560px	detect@0,8/2560px
ca. 9 min	1	0,9983	33 %	27 %

Tabelle 30: Ergebnisse Zusatzgeneration Epochenvergleich - 30 Epochen

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/2560px		detect@0,8/2560px	
ca. 9 min	1	0,9983	true-pos	false-pos	true-pos	false-pos
			60 %	40 %	47 %	20 %

Tabelle 31: Ergebnisse Zusatzgeneration Epochenvergleich - 50 Epochen

Trainingszeit pro Epoche	max. Precision	max. Recall	detect@0,5/2560px		detect@0,8/2560px	
ca. 9 min	1	0,9983	true-pos	false-pos	true-pos	false-pos
			47 %	33 %	40 %	20 %

Kritische Betrachtung

Generell zeigt der Test, dass mehr Epochen nicht gleich bessere Ergebnisse bedeuten. Hier muss differenziert werden, dass die Epochenzahl keine grundsätzliche Aussage für eine höhere Performance eines Trainings ist. Allerdings wird der Test bei der Anwendung weiterer Generation wiederholt werden, um individuell eine optimale Epochenzahl zu erhalten. YOLOv5 speichert nach jeder Epoche die Gewichte des Modells als Trainingssoutput ab. Hier gibt es eine 'last.pt' und 'best.pt' Datei. Das bedeutet, dass bei zu vielen Epochen, wenn 'last.pt' schlechter ist als 'best.pt', weiterhin das beste Training verwendet werden kann. Die 'best.pt' wird nur dann überschrieben, wenn 'last.pt' besser ist. Alternativ bietet YOLOv5 auch die Option, bei geringen oder ausbleibenden Lernerfolgen das Training zu stoppen (Early-Stopp).

Es sei angemerkt, dass wesentlich mehr Trainings ausgeführt wurden. Teilweise um Theorien im Zusammenhang mit den hier aufgeführten Trainings zu überprüfen. Teilweise aber auch, um den Umgang mit dem Framework zu lernen und zu testen, wie es auf verschiedene Parameter reagiert. Die Erkenntnisse dieser Trainings sind in die Ausführungen der Arbeit mit eingeflossen, würden aber den Umfang der Arbeit wesentlich erweitern.

6 Schlussbetrachtung

6.1 Fazit

Es gibt viele Anwendungen bei denen NN bessere Performance liefern, als konventionelle Ansätze es könnten. Die Objekterkennung im Hinblick auf die Anwendung am Datensatz von „Wo ist Walter?“ gehört hier auch dazu. Die Tests haben gezeigt, dass die verschiedenen trainierten und optimierten NNs eine sehr hohe Performance liefern. Zu den Optimierungen des NN zählen unter anderem die Anpassung des Datensatzes, der Epochenzahl, der Image-Size, der Batch-Size und die Veränderung der Größe des NN. Die Anpassung des Datensatzes hat die größte Auswirkung auf die Performance des Trainings. Beachtet werden müssen hier die Menge und Aufteilung der Daten, sowie die qualitative Repräsentation der Daten in Bezug auf die reale Anwendung. Eine zu geringe oder zu große Epochenzahl hat negative Auswirkung auf die Performance. Beachtet werden muss hierbei jedoch, dass bei einem größeren NN mehr Epochen für das Training benötigt werden. Eine höhere Image-Size wirkt sich positiv auf die Erkennungsrate des NN aus, wobei nicht sicher ist, ob das in jedem Fall zutrifft. Abbildung 12 zeigt den Verlauf der gemittelten Detektionen der einzelnen Trainings über die verschiedenen Generationen. Der Verlauf zeigt, dass sich mit fortlaufenden Generationen die Erkennung erhöht und die false-positives verringern. Mit jeder neuen Generation kamen zwar auch neue Hürden und andere Problemaspekte hinzu, allerdings konnten diese durch verschiedene Ansätze und Best-Practices gelöst werden. Entsprechend dessen kann, im Rahmen der Tests, attestiert werden, dass NNs für die Erkennung von Walter geeignet sind.

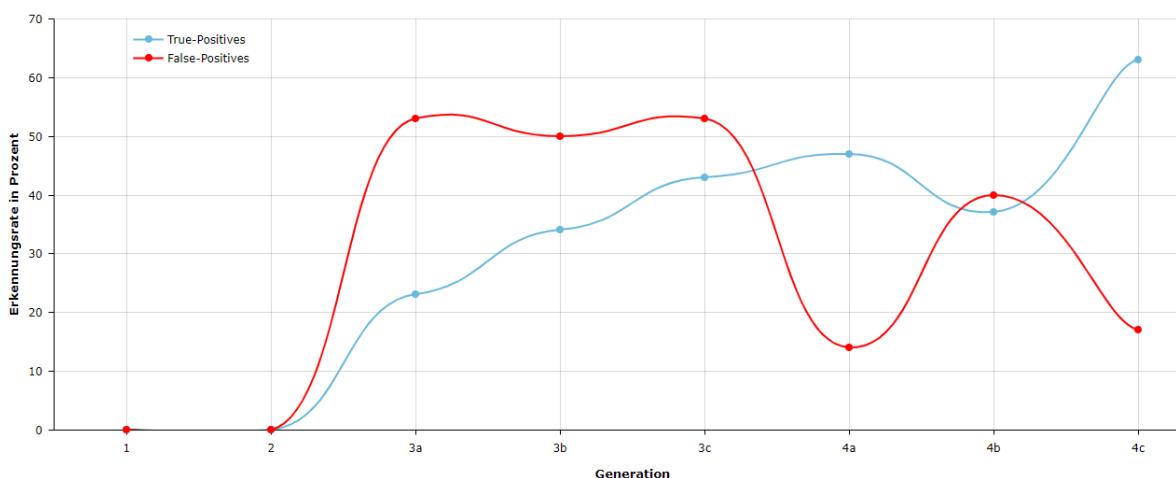


Abbildung 12: Zusammengefasste Ergebnisse der Generationen

6.2 Kritische Auseinandersetzung

Obwohl die Ergebnisse der Tests in dieser Arbeit vielversprechend sind, wirkten sich einige der Vorbedingungen der Arbeit einschränkend aus. Falls diese verändert werden können, hätte dies großen Einfluss auf die Performance und damit die Ergebnisse. Zunächst ist der Initialdatensatz, mit 52 Bildern, relativ klein. Dementsprechend mussten sehr viele zusätzliche Daten generiert werden. Obwohl hier sehr viel Wert auf eine gute Repräsentanz der generierten Daten im Hinblick auf die Initialdaten gelegt wurde, können die erzeugten Daten die Initialdaten nie zu 100 % repräsentieren. Dies führt dazu, dass das Netz viele false-positives erzeugt, da eine gewisse Unschärfe im Prozess vorhanden ist. Gut zu erkennen ist dies an den Ergebnissen der Tabelle 27 auf Seite 49. Hier wurden ausschließlich Bilder des Initialdatensatzes verwendet und es werden keine false-positives erzeugt. Hier muss es sich nicht um einen kausalen Zusammenhang handeln, allerdings gibt die Korrelation einen Hinweis darauf. Des Weiteren ist durch die Menge an Initialdaten der Testdatensatz relativ klein. Dass einige Bilder des Initialdatensatzes für bessere Erkennung im Trainings- und Validierungsdatensatz vorkommen, verringert den Testdatensatz erneut. Dementsprechend ist die Erkennungsrate anhand einer sehr geringen Menge an Testdaten evaluiert worden. Dies stellt die Aussagekraft der Erkennungsrate, in Bezug auf die statistische Auswertung, in Frage. Bei einer geringen Anzahl an Daten zum Testen fallen Anomalien viel stärker ins Gewicht, so z. B. sehr schwer erkennbare Walter, die sehr klein oder verdeckt sind. Dies könnte mit mehr Daten ebenfalls relativiert oder verbessert werden. Dennoch wurde durch Sichten der Bilder versucht, diese anhand der Schwierigkeit so gleichmäßig wie möglich zu verteilen. Weitere Einschränkungen ergeben sich durch die Hardware-Limitierung der GPUs. Hier kann der Trainingsprozess durch leistungsstarke Hardware signifikant beschleunigt werden, wodurch wiederum mehr und vor allem unterschiedliche Parameter getestet werden können. Dadurch könnten sich die Ergebnisse wesentlich diversifizieren lassen. Außerdem könnten so auch wesentlich größere NNs trainiert werden.

6.3 Ausblick

Diese Arbeit hat gezeigt, dass Walter, sowie Wilma, mit dem NN erkannt werden. Entsprechend dessen ist es möglich beliebig viele zusätzliche Objekte erkennen zu lassen. Hierfür müssen, nach derzeitiger Strategie, folgende Schritte durchgeführt werden:

1. Im Initialdatensatz müssen alle zu findenden Objekte gelabelt werden.
2. Um die Erkennung zu erhöhen, müssen zusätzliche Daten mit den Objekten, anhand der Image-Augmentation, erzeugt werden.
3. Die Objekte müssen als Klassen in der dataset.yaml und im Modell angegeben werden.

Nach diesen Schritten kann das Netz trainiert werden. Zusätzlich zu diesen Änderungen können auch andere Strategien verfolgt werden, um das Netz weiter zu optimieren. Hierbei können größere NNs genutzt werden oder die Menge der Trainingsprozessdaten erhöht werden. Zusätzlich können weitere Image-Augmentation-Schritte eingeführt oder die Image-Size stark erhöht werden. Anhand der Metriken und Ergebnisse müssen dann die Parameter der einzelnen Trainings abgestimmt werden. Des Weiteren kann durch integrieren von nicht gelabelten Daten versucht werden, die Performance des NN positiv zu beeinflussen. Eine weitere Möglichkeit ist, dass ein komplett eigenes NN, mit eigener Architektur, entwickelt wird. Hier sind einige Designaspekte, wie die Anzahl der Neuronen und Schichten, die Funktion der Schichten, sowie die Art und Umsetzung der BP oder der Loss-Funktion zu beachten. Ein weiterer wichtiger Aspekt der beim Arbeiten mit NNs hilft, ist die Aufbereitung und das Verwalten von durchgeführten Trainings, sowie das Dokumentieren der Resultate. Hier ist es wichtig bereits im Voraus eine Strategie zu entwickeln, wie und wo die Trainings gespeichert werden. Außerdem müssen die genutzten Parameter und Ergebnisse richtig zugeordnet werden. Mit zunehmender Anzahl an Trainings wird dieses Problem noch verstärkt. Hier kann durch MLOps, bzw. ML Operations, entgegengewirkt werden. MLOps fungiert hierbei als eine Automatisierungsmöglichkeit für die Analyse und Verwaltung von Trainings. Die Nutzung eines solchen Tools, wie z. B. MLflow [82], könnte die Verwaltung der Trainings und den Zeitaufwand für die Analyse dieser optimieren.

Abschließend sei angemerkt, dass an das Thema der Objekterkennung mittels NNs auf sehr verschiedene Arten und Weisen herangegangen werden kann und jede Möglichkeit unterschiedliche Vor- und Nachteile hat. Vor allem ohne Vorwissen bleibt meist nur die Möglichkeit per „Trial and Error“ zu testen und dadurch Erfahrung zu sammeln.

Literaturverzeichnis

- [1] Cambridge Advanced Learner's Dictionary & Thesaurus. „artificial intelligence.“ (2021), Adresse: <https://dictionary.cambridge.org/dictionary/english/artificial-intelligence> (besucht am 14.07.2021).
- [2] Y. Bengio, A. Courville und I. Goodfellow, *Deep Learning*. Cambridge: MIT Press, 2016.
- [3] A. Turing, „Computing Machinery and Intelligence,“ *Mind*, Jg. 49, 236 Okt. 1950. DOI: 10.1093/mind/LIX.236.433.
- [4] J. McCarthy, M. Minsky, N. Rochester und C. E. Shannon, „A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence,“ *AI Magazine*, Jg. 27, 4 31. Aug. 1955.
- [5] R. Garner. „Early Popular Computers, 1950 - 1970.“ (2021), Adresse: https://ethw.org/Early_Popular_Computers,_1950_-_1970 (besucht am 14.07.2021).
- [6] J. Koomey, S. Berard, M. Sanchez und H. Wong, „Computer nach dem Vorbild des Gehirns? Wie es gelingen kann, das menschliche Gehirn und seine Arbeitsweise nachzuahmen,“ *IEEE Annals of the History of Computing*, Jg. 33, 3 3. März 2011. DOI: 10.1109/MAHC.2010.28.
- [7] IBM Corporation. „Deep Blue.“ (7. März 2012), Adresse: <https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/> (besucht am 14.07.2021).
- [8] ——, „A Computer Called Watson.“ (7. März 2012), Adresse: <https://www.ibm.com/ibm/history/ibm100/us/en/icons/watson/> (besucht am 14.07.2021).
- [9] DeepMind. „Alpha Go.“ (2021), Adresse: <https://deepmind.com/research/case-studies/alphago-the-story-so-far> (besucht am 14.07.2021).
- [10] Y. Bengio, G. Hinton und Y. Lecun, „Deep Learning for AI,“ *Communications of the ACM*, Jg. 64, 7 Juli 2021. DOI: 10.1145/3448250.
- [11] N. Büttner, *Hieronymus Bosch*. Dortmund: C.H.Beck, 2012.
- [12] „Waldo Wiki.“ (2021), Adresse: https://waldo.fandom.com/wiki/Waldo_Wiki (besucht am 14.07.2021).
- [13] K. Meier, „Computer nach dem Vorbild des Gehirns? Wie es gelingen kann, das menschliche Gehirn und seine Arbeitsweise nachzuahmen,“ *Ruperto Carola*, Jg. 1, 2007. Adresse: <https://www.uni-heidelberg.de/presse/ruca/ruca07-1/vorbild.html>.

- [14] L. Kowal-Summek, *Neurowissenschaften und Musikpädagogik, Klärungsversuche und Praxisbezüge*, 2. Aufl. Wiesbaden: Springer, 2018.
- [15] A. Frances, *NORMAL, Gegen die Inflation psychiatrischer Diagnosen*. Köln: DuMont, 2013.
- [16] Max-Planck-Gesellschaft. „Das Gehirn.“ (2021), Adresse: <https://www.mpg.de/gehirn> (besucht am 14.07.2021).
- [17] SfN, „Das Gehirn, Eine kurze Zusammenfassung über das Gehirn und das Nervensystem,“ The Society for Neuroscience, Broschüre, 2010, Deutsche Übersetzung.
- [18] S. Erulkar und T. Lenz, *Nervous System*, in *Encyclopedia Britannica*, K. Rogers und G. Young, Hrsg., 10. Nov. 2020. Adresse: <https://www.britannica.com/science/nervous-system> (besucht am 12.03.2021).
- [19] F. Azevedo, L. Carvalho und L. Grinberg, „Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain,“ *Comparative Neurology*, Jg. 513, 5 10. Apr. 2009. DOI: 10.1002/cne.21974.
- [20] S. DeWeerdt, „Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain,“ *Nature*, Jg. 571, 24. Juli 2019. DOI: 10.1038/d41586-019-02208-0.
- [21] T. Bonhoeffer und V. Nägerl, „Morphologische Plastizität in Neuronen und ihre Konkurrenz um synaptische Proteine,“ Max-Planck-Institut für Neurobiologie, Forschungsbericht, 2005.
- [22] G. Collingridge, J. Howland, S. Peineau und Y. T. Wang, „Long-term depression in the CNS,“ *Nature Reviews Neuroscience*, Jg. 11, Juli 2010. DOI: 10.1038/nrn2867.
- [23] K. Cherry. „The Psychology of Learning.“ (31. März 2020), Adresse: <https://www.verywellmind.com/learning-study-guide-2795698> (besucht am 14.07.2021).
- [24] ——, „History and Key Concepts of Behavioral Psychology.“ (20. Feb. 2021), Adresse: <https://www.verywellmind.com/behavioral-psychology-4157183> (besucht am 14.07.2021).
- [25] A. Bandura, „Social learning through imitation,“ *Nebraska Symposium of Motivation*, 1962.
- [26] A. Meltzoff, „Born to Learn, What Infants Learn from Watching Us,“ Department of Psychology, University of Washington, 2010.

- [27] A. Meltzoff, V. Jaswal und R. Williamson, „Learning the rules, observation and imitation of a sorting strategy by 36-month-old children,“ *Developmental Psychology*, Jg. 46, 1 Jan. 2010. DOI: 10.1037/a0017473.
- [28] M. Cohen, *Linear Algebra: Theory, Intuition, Code*. Sincxpress Bv, 2021.
- [29] E. Jaynes, *Probability Theory, The Logic of Science*. Cambridge: Cambridge University Press, 2003.
- [30] P. Furlan, *Das gelbe Rechenbuch 1, Lineare Algebra und Differentialrechnung*. Dortmund: Verlag Martina Furlan, 2012.
- [31] S. Du, J. Lee, H. Li, L. Wang und X. Zhai, „Gradient Descent Finds Global Minima of Deep Neural Networks,“ *ArXiv*, Jg. 1811.03804, v4 28. Mai 2019. Adresse: <https://arxiv.org/abs/1811.03804v4>.
- [32] K. Kawaguchi und J. Huang, „Gradient Descent Finds Global Minima for Generalizable Deep Neural Networks of Practical Sizes,“ in *57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2019, S. 92–9. DOI: 10.1109/ALLERTON.2019.8919696.
- [33] Y. LeCun, Y. Bengio und G. Hinton, „Deep Learning,“ *Nature*, Jg. 521, 27. Mai 2015. DOI: 10.1038/nature14539.
- [34] T. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.
- [35] „Supervised Learning,“ in *Encyclopedia of Machine Learning*, C. Sammut und G. Webb, Hrsg. Boston, MA: Springer US, 2011, S. 941. DOI: 10.1007/978-0-387-30164-8_803.
- [36] „Unsupervised Learning,“ in *Encyclopedia of Machine Learning*, C. Sammut und G. Webb, Hrsg. Boston, MA: Springer US, 2011, S. 1009. DOI: 10.1007/978-0-387-30164-8_867.
- [37] P. Stone, „Reinforcement Learning,“ in *Encyclopedia of Machine Learning*, C. Sammut und G. Webb, Hrsg. Boston, MA: Springer US, 2011, S. 849–51. DOI: 10.1007/978-0-387-30164-8_714.
- [38] S. Kumar. „Data splitting technique to fit any Machine Learning Model.“ (2020), Adresse: <https://towardsdatascience.com/data-splitting-technique-to-fit-any-machine-learning-model-c0d7f3f1c790> (besucht am 14.07.2021).

- [39] G. Webb, „Overfitting,“ in *Encyclopedia of Machine Learning*, C. Sammut und G. Webb, Hrsg. Boston, MA: Springer US, 2011, S. 744. DOI: 10.1007/978-0-387-30164-8_623.
- [40] H. Jabbar und R. Khan, „Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study),“ *Computer Science, Communication & Instrumentation Devices*, 27. Dez. 2014. DOI: 10.3850/978-981-09-5247-1_017.
- [41] H. Kinsley und D. Kukieła, *Neural Networks from Scratch in Python*. selfpublished, 2020.
- [42] J. Khaw, B. Lim und L. Lim, „Optimal design of neural networks using the Taguchi method,“ *Neurocomputing*, Jg. 7, 3 1995. DOI: 10.1016/0925-2312(94)00013-I.
- [43] Y. LeCun, L. Bottou, Y. Bengio und P. Haffner, „Gradient-Based Learning Applied to Document Recognition,“ *Proceedings of the IEEE*, Jg. 86, 11 Nov. 1998. DOI: 10.1109/5.726791.
- [44] D. Svozil, V. Kvasnička und J. Pospíchal, „Introduction to multi-layer feed-forward neural networks,“ *Chemometrics and Intelligent Laboratory Systems*, Jg. 39, 1997. DOI: 10.1016/S0169-7439(97)00061-0.
- [45] W. Zarema, I. Sutskever und O. Vinyals, „Recurrent Neural Network Regularization,“ *ArXiv*, Jg. 1409.2329, 5 19. Feb. 2015. Adresse: <https://arxiv.org/abs/1409.2329v5>.
- [46] A. Subasi, „Chapter 3 - Machine learning techniques,“ in *Practical Machine Learning for Data Analysis Using Python*, Academic Press, 2020, S. 91–202. DOI: 10.1016/B978-0-12-821379-7.00003-5.
- [47] S. Amidi. „CS 230 – Deep Learning, Recurrent Neural Networks.“ (20. Mai 2020), Adresse: <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks> (besucht am 14. 07. 2021).
- [48] J. Torres. „Learning Process of a Deep Neural Network.“ (21. Apr. 2020), Adresse: <https://towardsdatascience.com/learning-process-of-a-deep-neural-network-5a9768d7a651> (besucht am 14. 07. 2021).
- [49] D. Rumelhart, G. Hinton und R. Williams, „Learning representations by back-propagating errors,“ *Nature*, Jg. 323, 9. Okt. 1986. DOI: 10.1038/323533a0.

- [50] Keras Team. „Keras Documentation: Losses.“ (2021), Adresse: <https://keras.io/api/losses/> (besucht am 14.07.2021).
- [51] C. Wick, „Deep Learning,“ *Informatik Spektrum*, Jg. 40, Feb. 2017. doi: 10.1007/s00287-016-1013-2.
- [52] S. Minaee. „20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics.“ (Okt. 2019), Adresse: <https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce> (besucht am 14.07.2021).
- [53] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid und S. Savarese, „Generalized Intersection over Union, A Metric and A Loss for Bounding Box Regression,“ *The IEEE Conference on Computer Vision and Pattern Recognition*, Juni 2019.
- [54] S. Yohanandan. „mAP (mean Average Precision) might confuse you!“ (9. Juni 2020), Adresse: <https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2> (besucht am 14.07.2021).
- [55] „Data Preparation and Feature Engineering for Machine Learning,“ Google. (2021), Adresse: <https://developers.google.com/machine-learning/data-prep/> (besucht am 14.07.2021).
- [56] „Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says,“ Forbes. (März 2016), Adresse: <https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/> (besucht am 14.07.2021).
- [57] G. Webb, „Data Preperation,“ in *Encyclopedia of Machine Learning*, C. Sammut und G. Webb, Hrsg. Boston, MA: Springer US, 2011, S. 259–60. doi: 10.1007/978-0-387-30164-8_194.
- [58] J. Brownlee, *Machine Learning Mastery With Python, Understand Your Data, Create Accurate Models, and Work Projects End-to-End*. Machine Learning Mastery, 2016.
- [59] Y. Yang, „Discretization,“ in *Encyclopedia of Machine Learning*, C. Sammut und G. Webb, Hrsg. Boston, MA: Springer US, 2011, S. 287–8. doi: 10.1007/978-0-387-30164-8_221.
- [60] C. Voskolou, M. Wilcox, S. Schuermans und A. Sobolevski, „State of the Developer Nation Q1 2017,“ Vision Mobile, Report, März 2017.

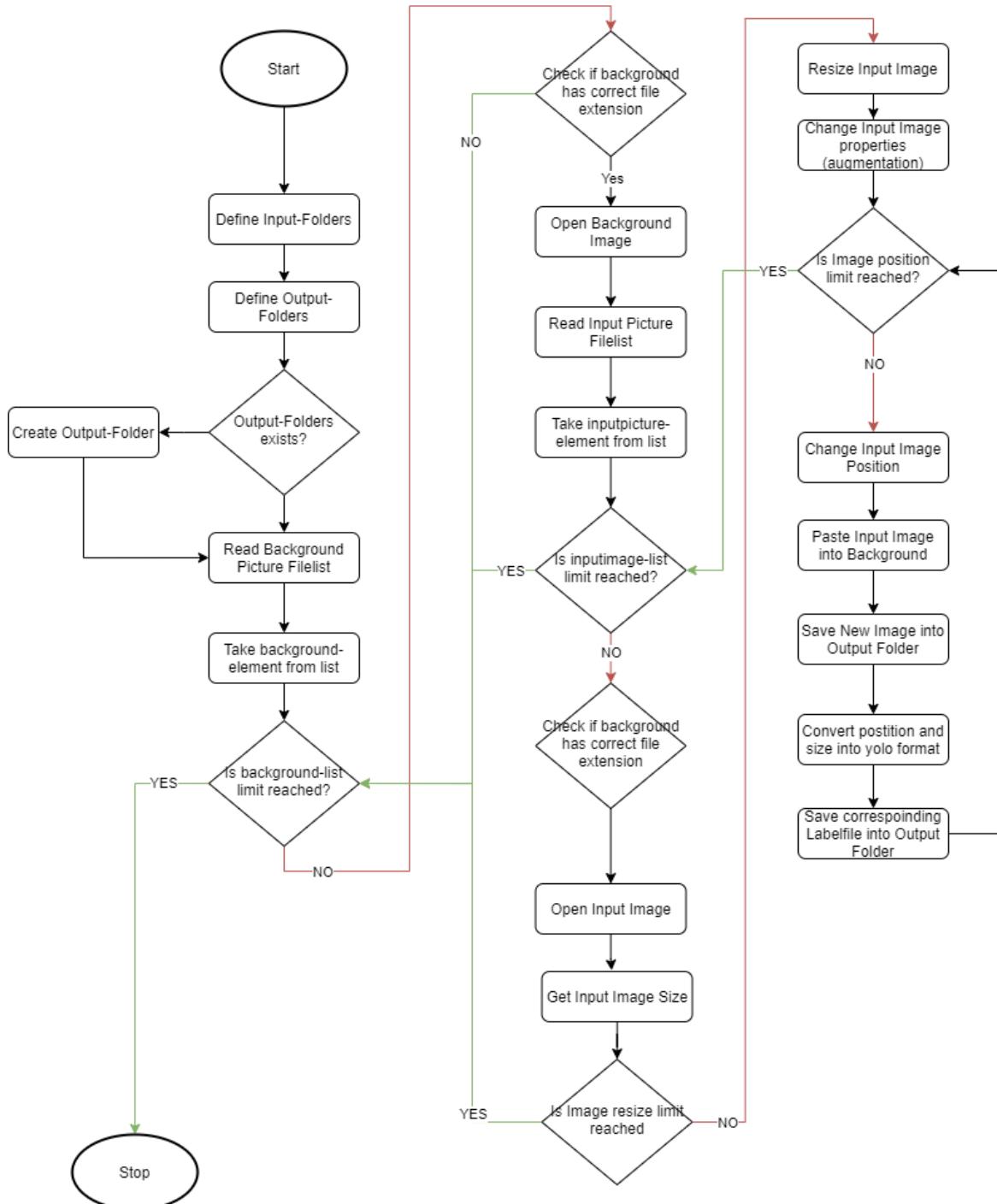
- [61] D. Karczewski. „What Is The Best Language For Machine Learning In 2021?“ (18. Juni 2020), Adresse: <https://www.ideamotive.co/blog/what-is-the-best-language-for-machine-learning> (besucht am 14.07.2021).
- [62] E. Frank, M. Hall und I. Witten. „The WEKA Workbench, Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques".“ (2016), Adresse: https://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf (besucht am 14.07.2021).
- [63] NumPy. „NumPy, The fundamental package for scientific computing with Python.“ (2021), Adresse: <https://numpy.org/> (besucht am 14.07.2021).
- [64] NVIDIA Corporation. „About CUDA.“ (2021), Adresse: <https://developer.nvidia.com/about-cuda> (besucht am 14.07.2021).
- [65] PyTorch. „PyTorch.“ (2021), Adresse: <https://pytorch.org/> (besucht am 14.07.2021).
- [66] A. Clark. „Pillow, Python Imaging Library (PIL Fork).“ (8. Juli 2021), Adresse: <https://python-pillow.org/> (besucht am 14.07.2021).
- [67] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin und A. A. Kalinin, „Albumentations, Fast and Flexible Image Augmentations,“ *Information*, Jg. 11, 2 2020. DOI: 10.3390/info11020125.
- [68] V. Sajip. „Logging HOWTO.“ (8. Juli 2021), Adresse: <https://docs.python.org/3/howto/logging.html> (besucht am 14.07.2021).
- [69] P. S. Foundation. „os, Miscellaneous operating system interfaces.“ (8. Juli 2021), Adresse: <https://docs.python.org/3/library/os.html> (besucht am 14.07.2021).
- [70] ——, „sys, System-specific parameters and functions.“ (8. Juli 2021), Adresse: <https://docs.python.org/3/library/sys.html> (besucht am 14.07.2021).
- [71] ——, „pathlib, Object-oriented filesystem paths.“ (8. Juli 2021), Adresse: <https://docs.python.org/3/library/pathlib.html> (besucht am 14.07.2021).
- [72] ——, „datetime, Basic date and time types.“ (8. Juli 2021), Adresse: <https://docs.python.org/3/library/datetime.html> (besucht am 14.07.2021).
- [73] ——, „shutil, High-level file operations.“ (8. Juli 2021), Adresse: <https://docs.python.org/3/library/shutil.html> (besucht am 14.07.2021).

- [74] ——, „random, Generate pseudo-random numbers.“ (8. Juli 2021), Adresse: <https://docs.python.org/3/library/random.html> (besucht am 14.07.2021).
- [75] T. Lin. „labelImg.“ (2021), Adresse: [https://github.com/tzutalin\(labelImg](https://github.com/tzutalin(labelImg) (besucht am 14.07.2021).
- [76] G. Jocher. „YOLOv5 Documentation.“ (7. Juli 2021), Adresse: <https://docs.ultralytics.com/> (besucht am 14.07.2020).
- [77] J. Redmon, S. Divvala, R. Girshick und A. Farhadi, „You Only Look Once, Unified, Real-Time Object Detection,“ *ArXiv*, Jg. 1506.02640, v5 8. Juni 2015. Adresse: <https://arxiv.org/abs/1506.02640v5>.
- [78] J. Redmon und A. Farhadi, „YOLO9000, Better, Faster, Stronger,“ *ArXiv*, Jg. 1612.08242, v1 25. Dez. 2016. Adresse: <https://arxiv.org/abs/1612.08242v1>.
- [79] ——, „YOLOv3, An Incremental Improvement,“ *ArXiv*, Jg. 1804.02767, v1 8. Apr. 2018. Adresse: <https://arxiv.org/abs/1804.02767v1>.
- [80] A. Brokovskiy, C.-Y. Wang und H.-Y. Liao, „YOLOv4, Optimal Speed and Accuracy of Object Detection,“ *ArXiv*, Jg. 2004.10934, v1 23. Apr. 2020. Adresse: <https://arxiv.org/abs/2004.10934v1>.
- [81] G. Jocher. „YOLOv5.“ (Juli 2020), Adresse: <https://github.com/ultralytics/yolov5> (besucht am 14.07.2021).
- [82] MLflow. „MLflow, A platform for the machine learning lifecycle.“ (2021), Adresse: <https://mlflow.org/> (besucht am 14.07.2021).

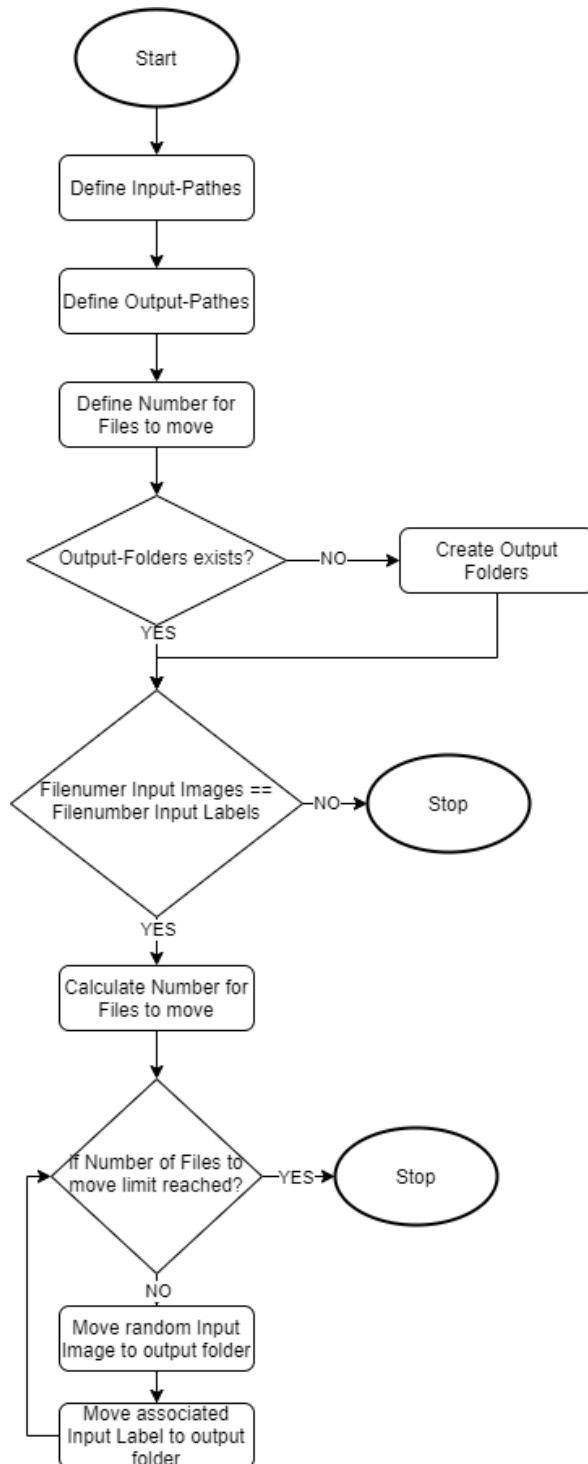
A Labelformat YOLOv5

```
0  0.480109 0.631250 0.692969 0.713278
0  0.740125 0.523659 0.694587 0.932658
0  0.875692 0.754148 0.215964 0.269874
1  0.369584 0.265987 0.125487 0.245896
```

B Programmablaufplan Image_Label_Creator



C Programmablaufplan Train_Val_Test_randomizer



D Bildsammlung

D.1 Walter und Wilma werden erkannt



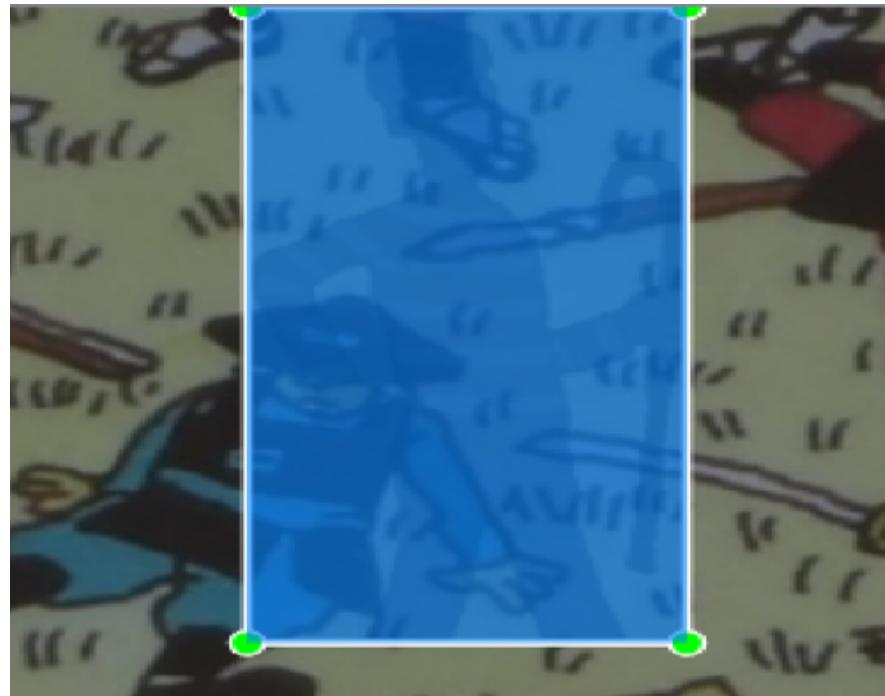
D.2 Walter False-Positive



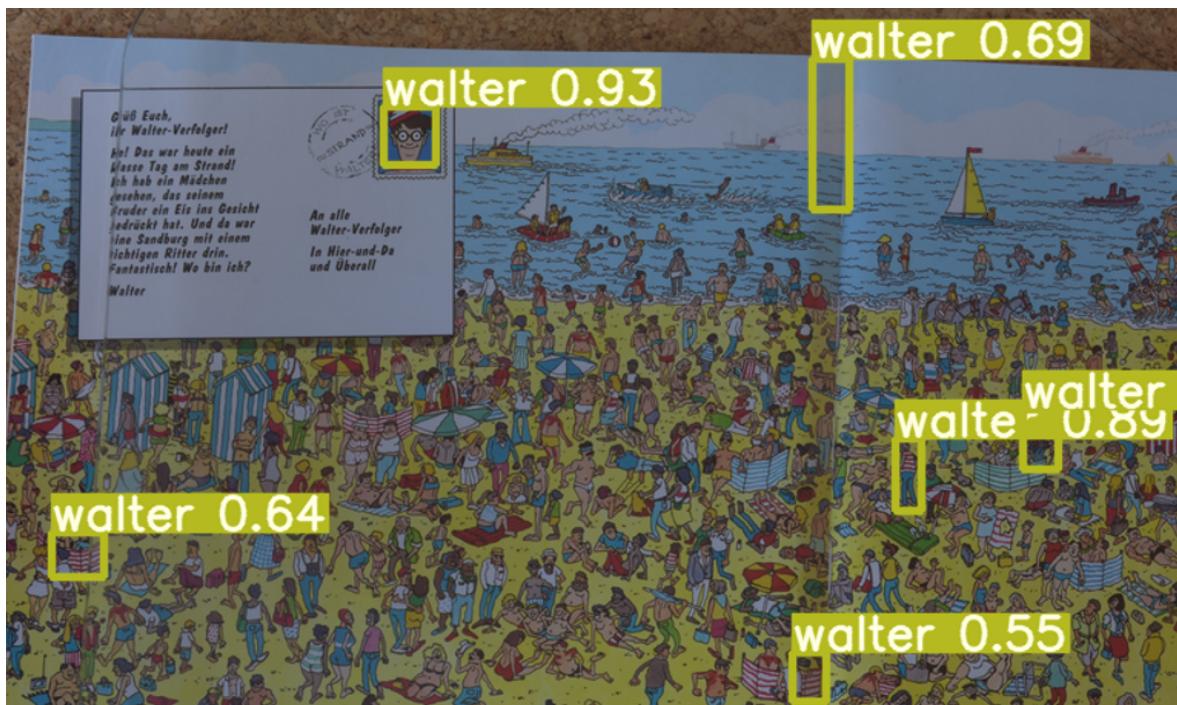
D.3 Walter Rotationsproblem



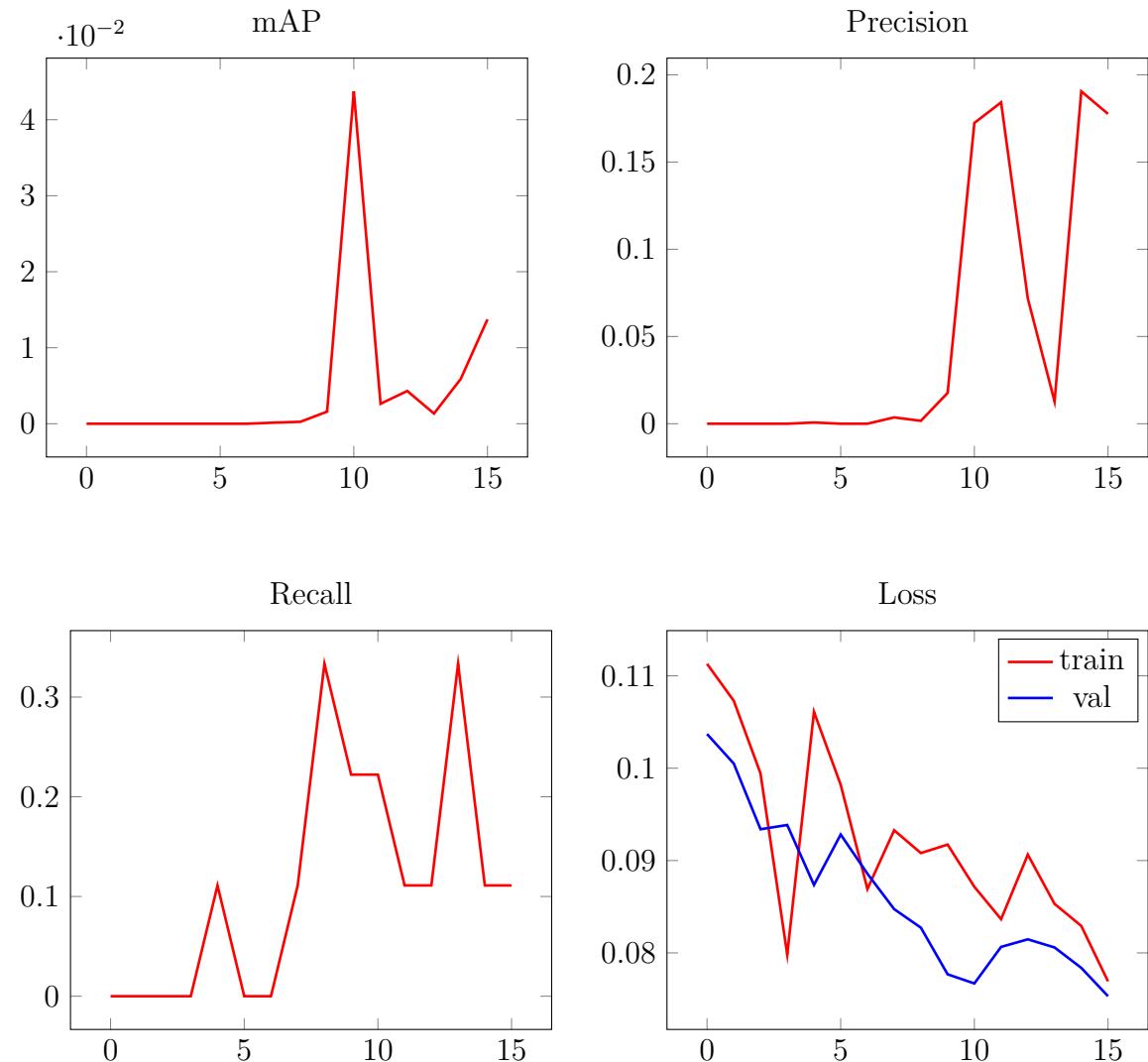
D.4 Walter Kontrastproblem



D.5 Walter Hochkant- und gestreifte Objekte Problem

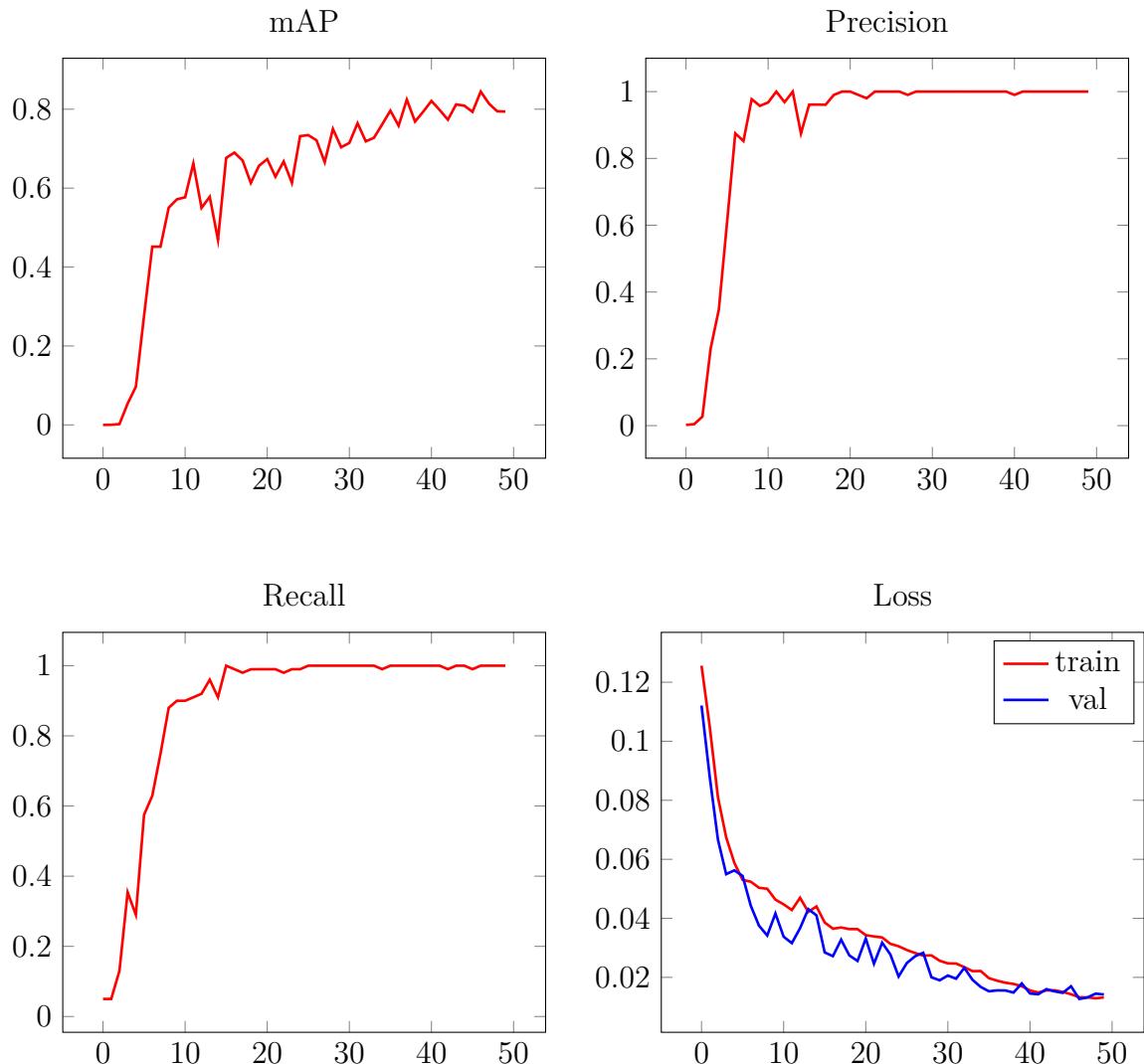


E Metriken Generation 1

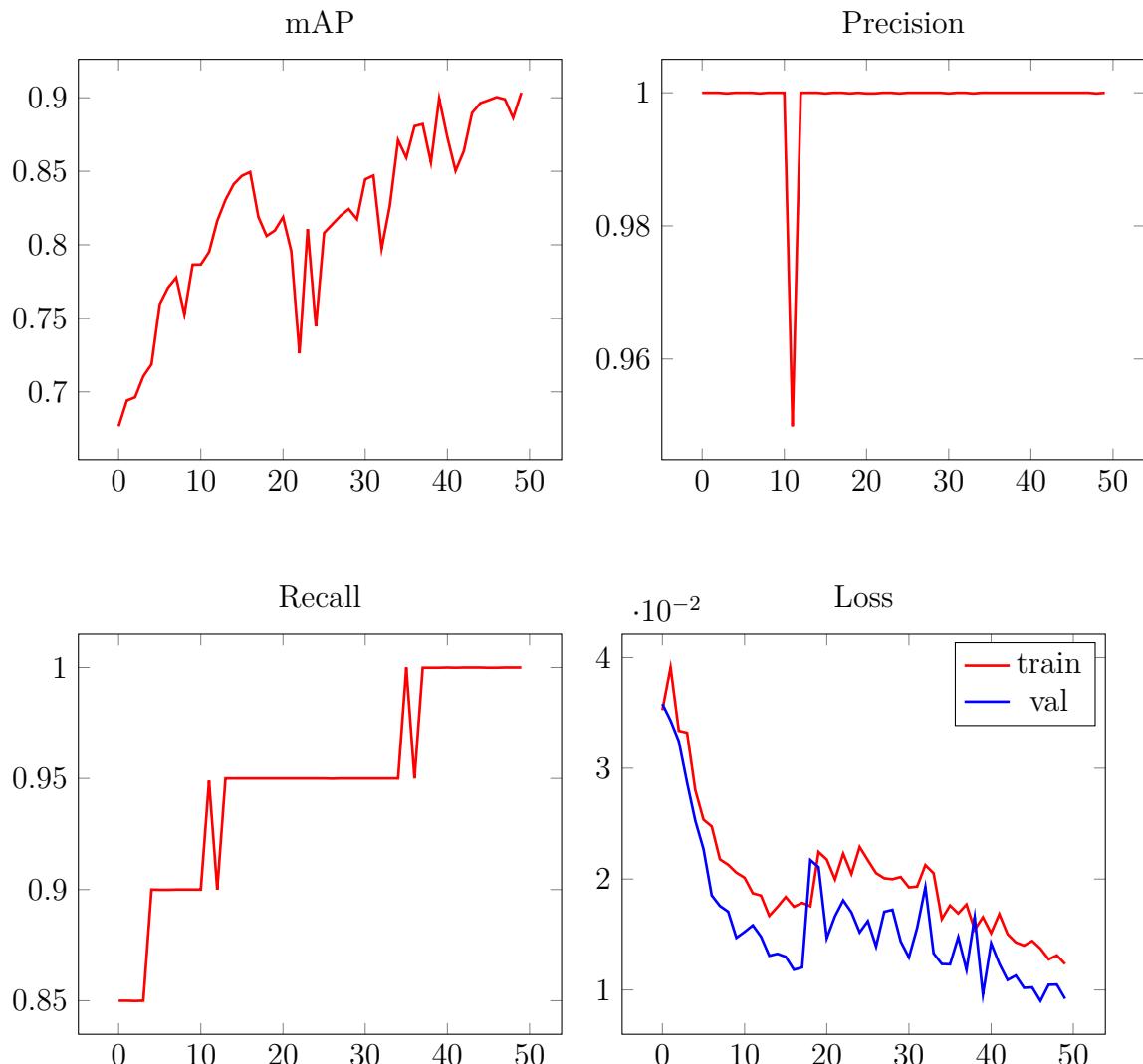


F Metriken Generation 2

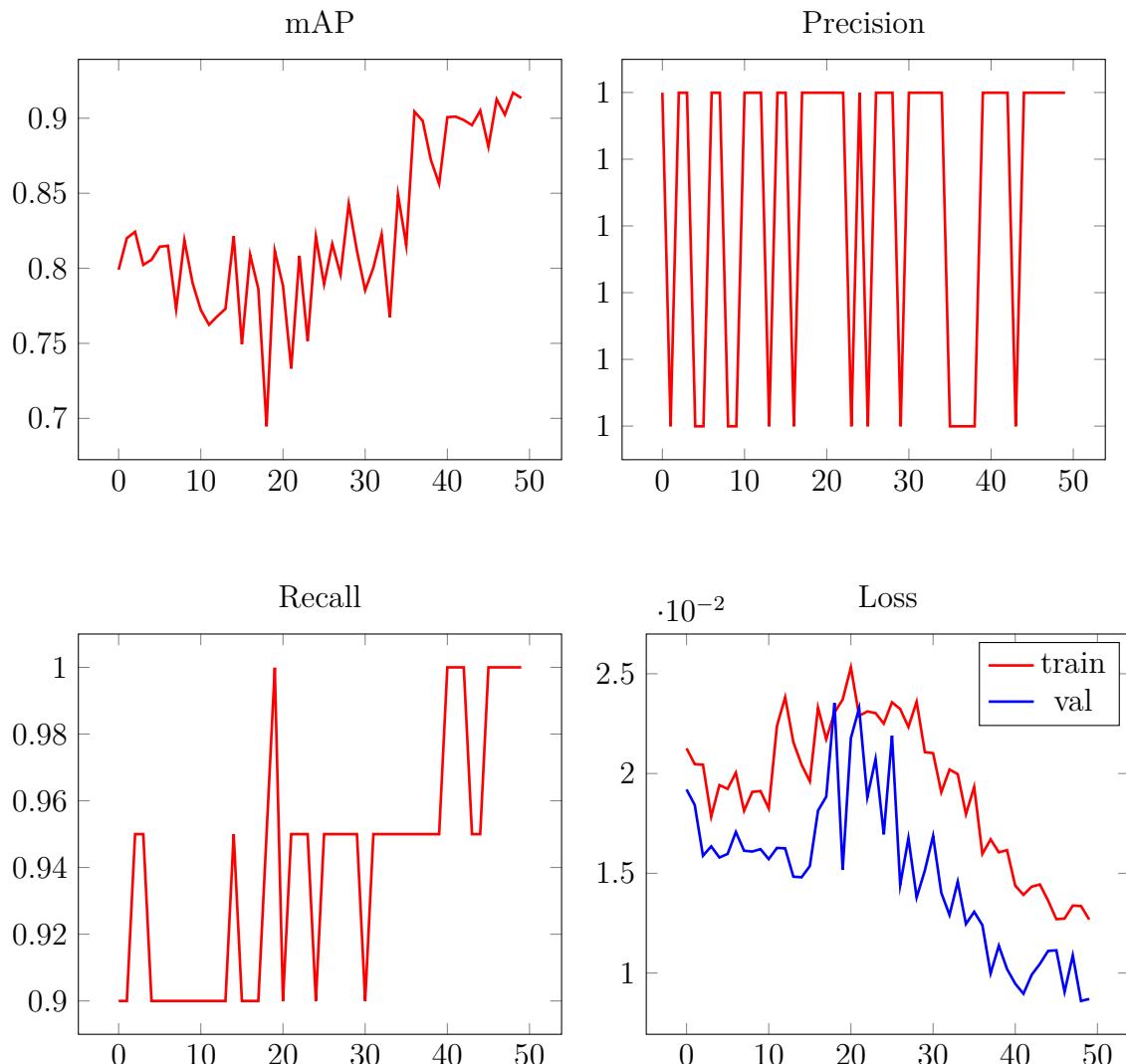
F.1 Generation 2a



F.2 Generation 2b

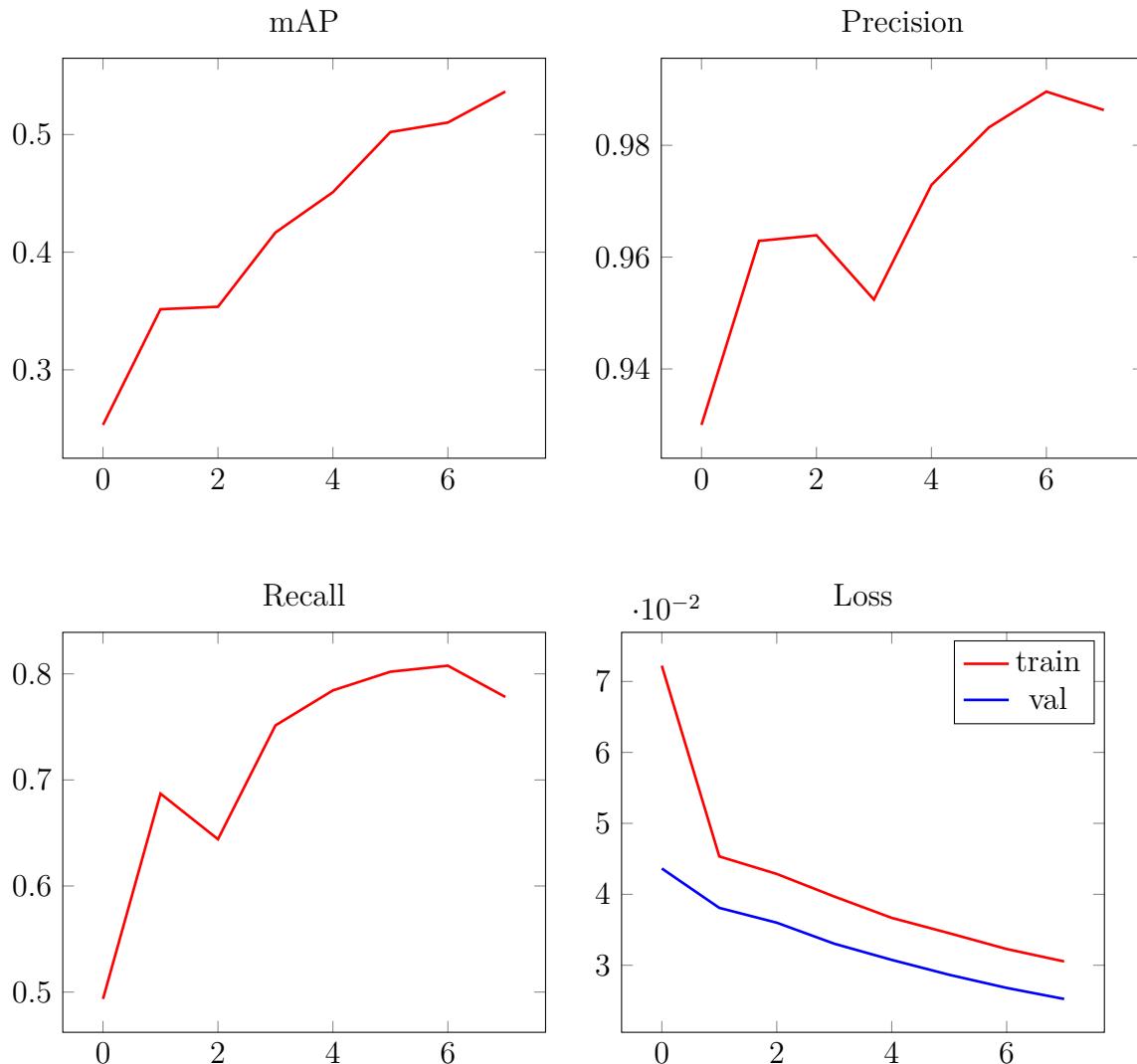


F.3 Generation 2c

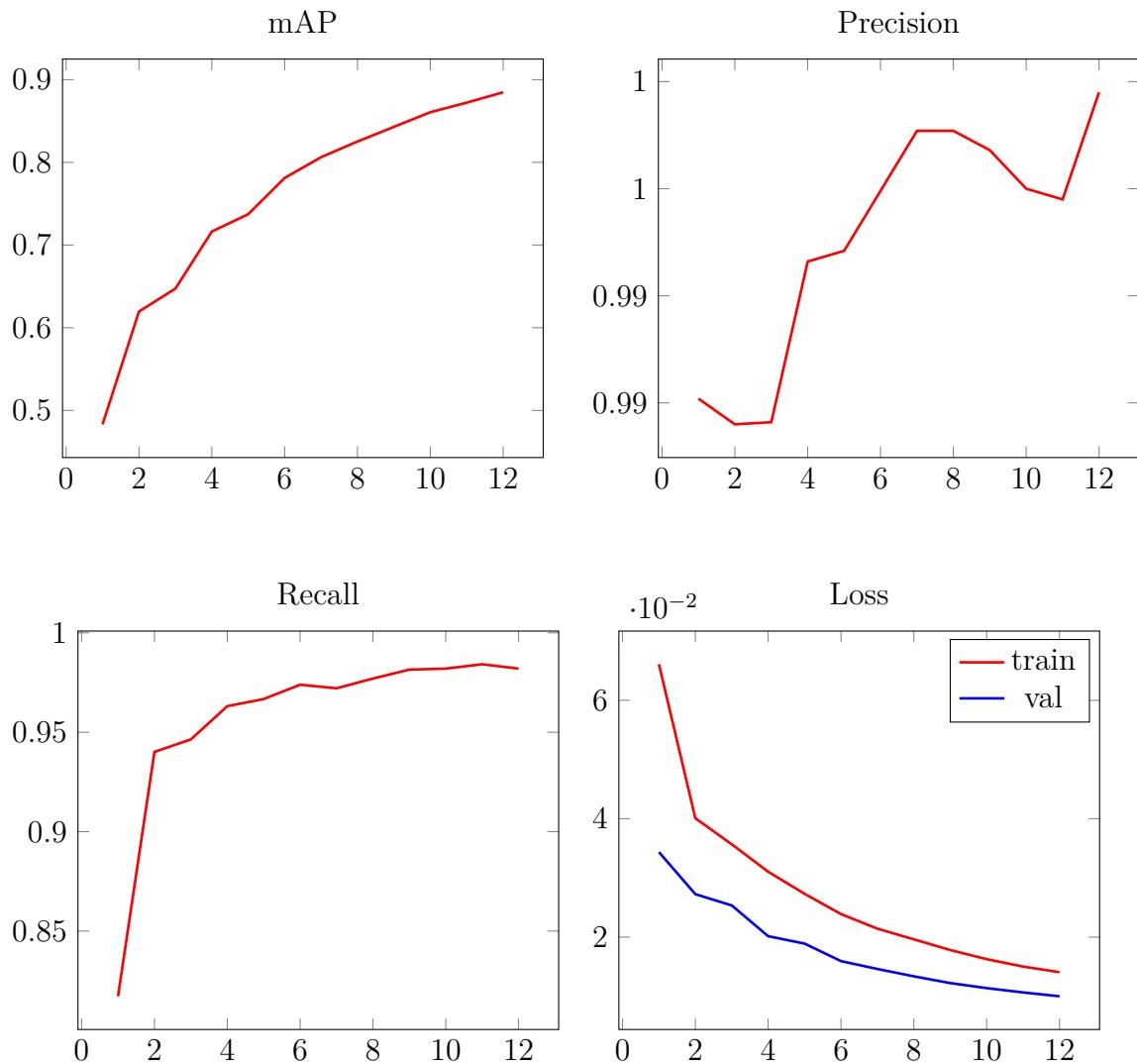


G Metriken Generation 3

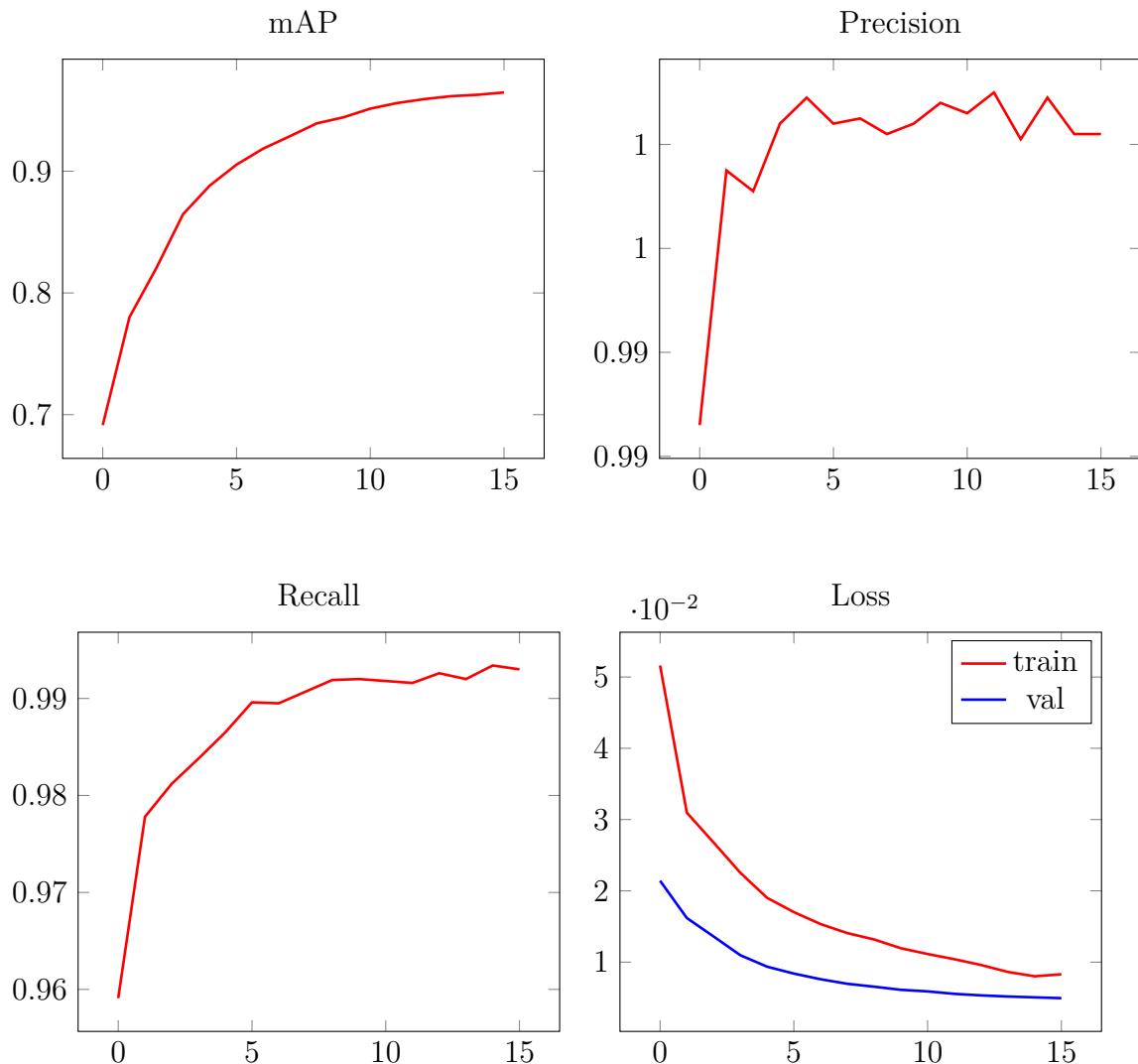
G.1 Generation 3a



G.2 Generation 3b

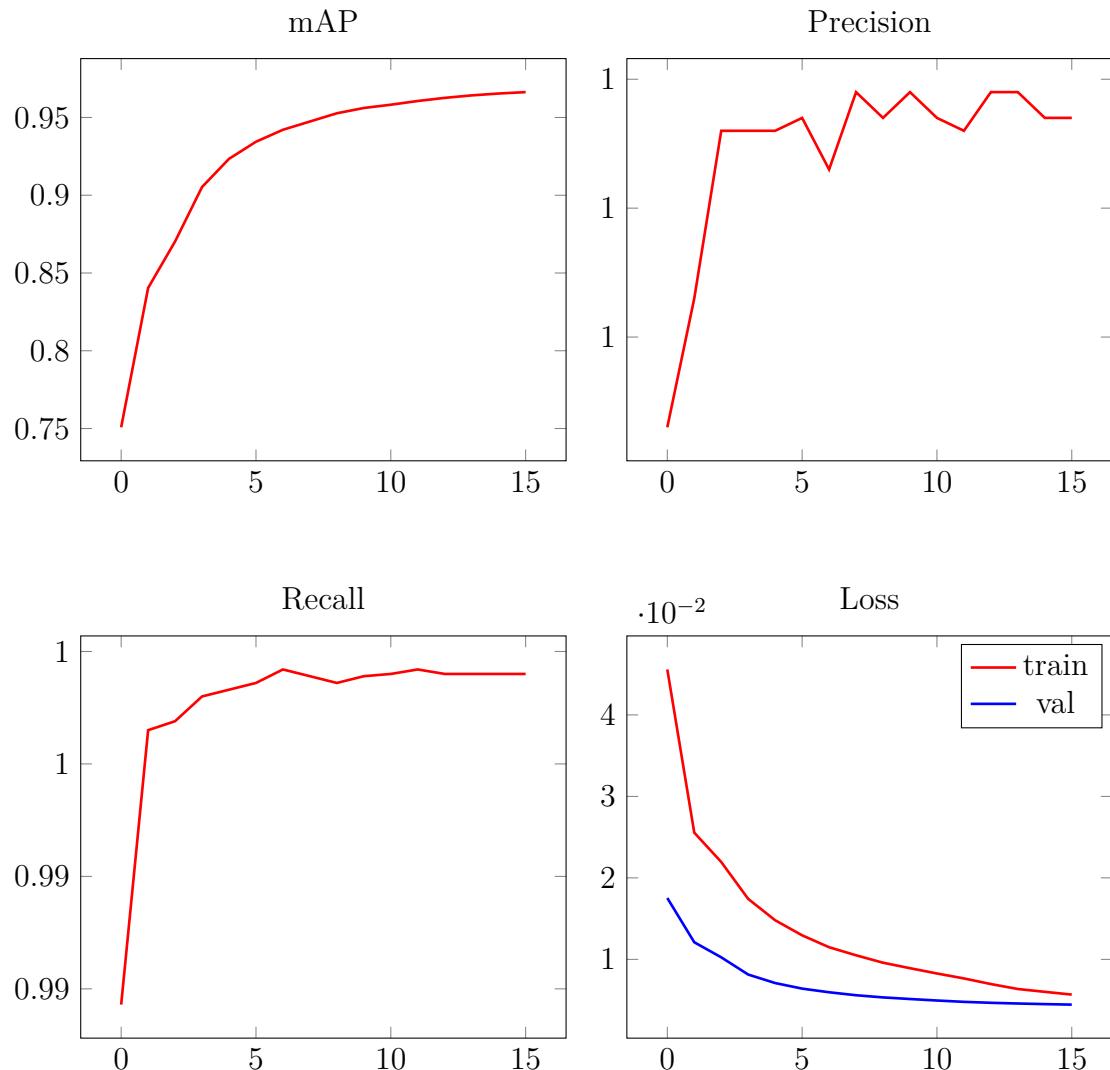


G.3 Generation 3c

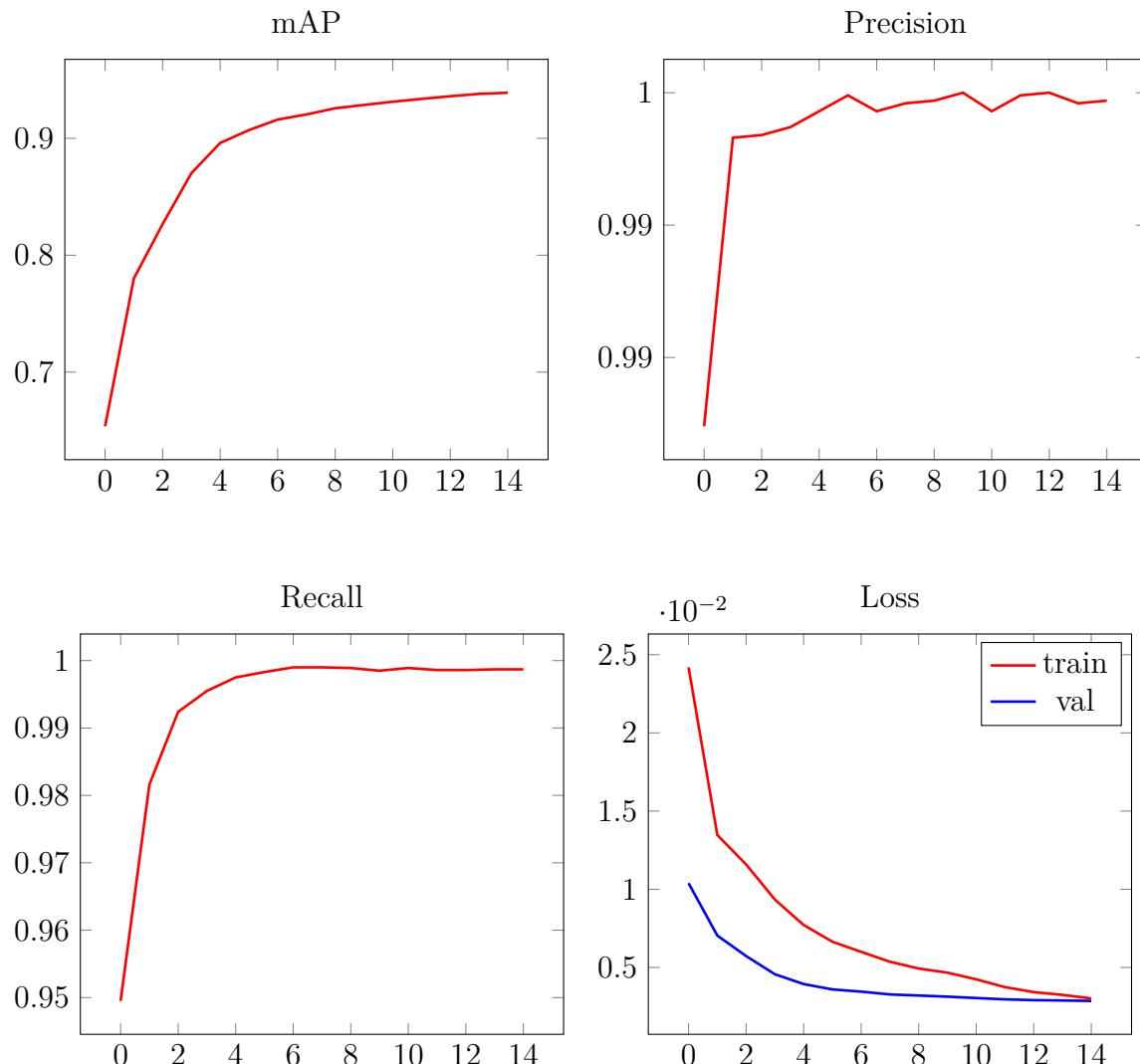


H Metriken Generation 4

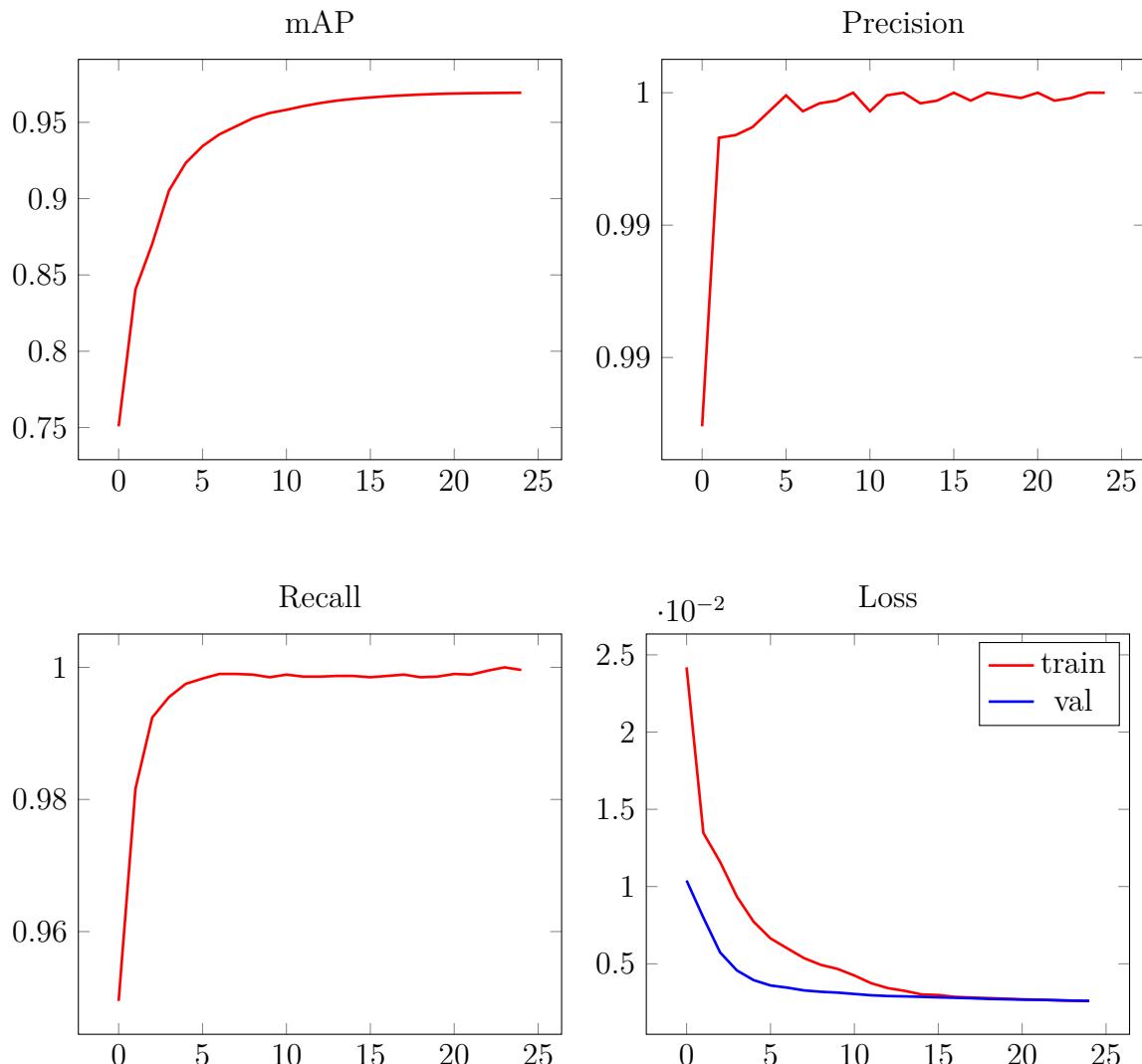
H.1 Generation 4a



H.2 Generation 4b

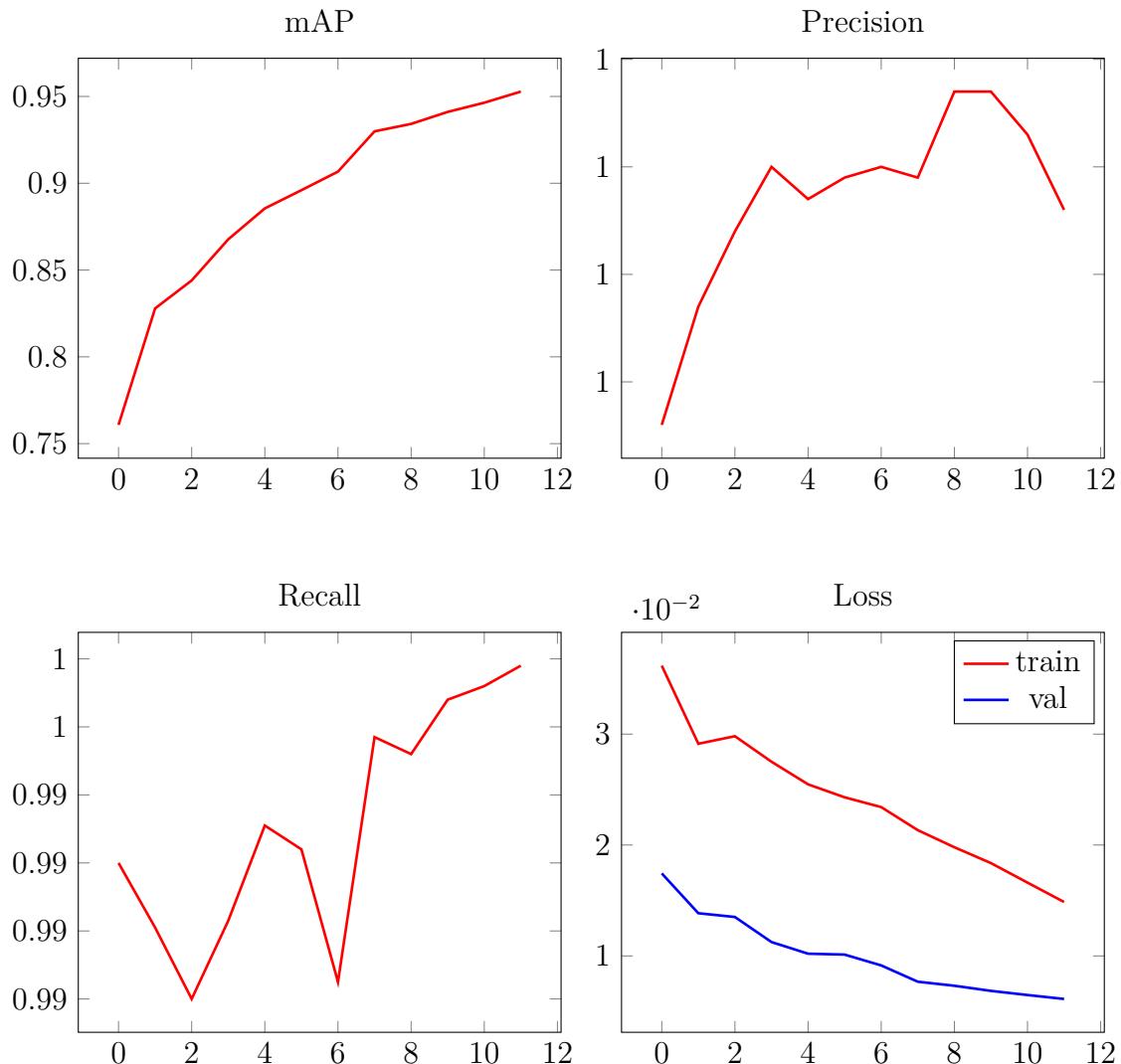


H.3 Generation 4c



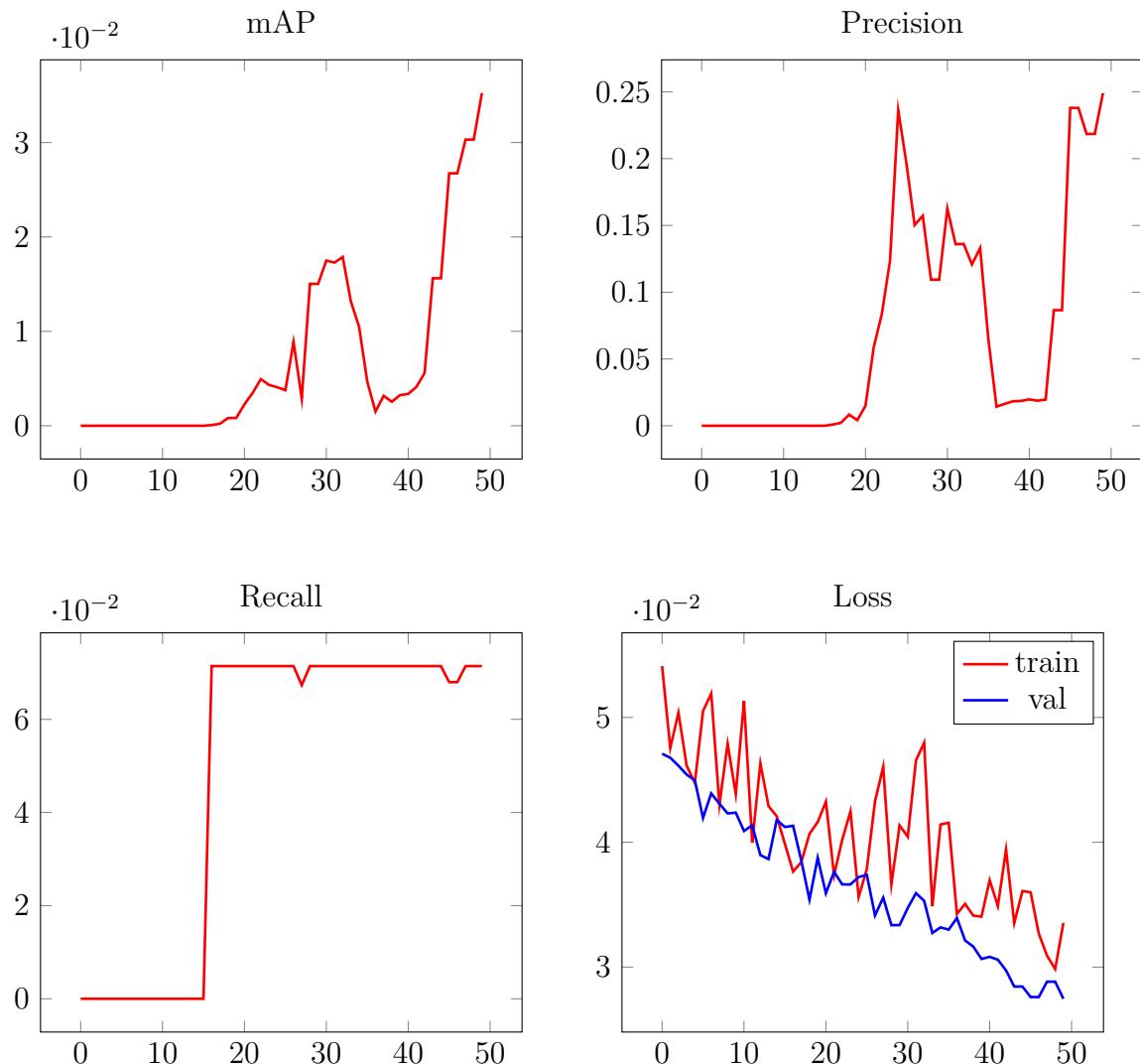
I Metriken Zusatzgeneration

I.1 Lernrate erhöht

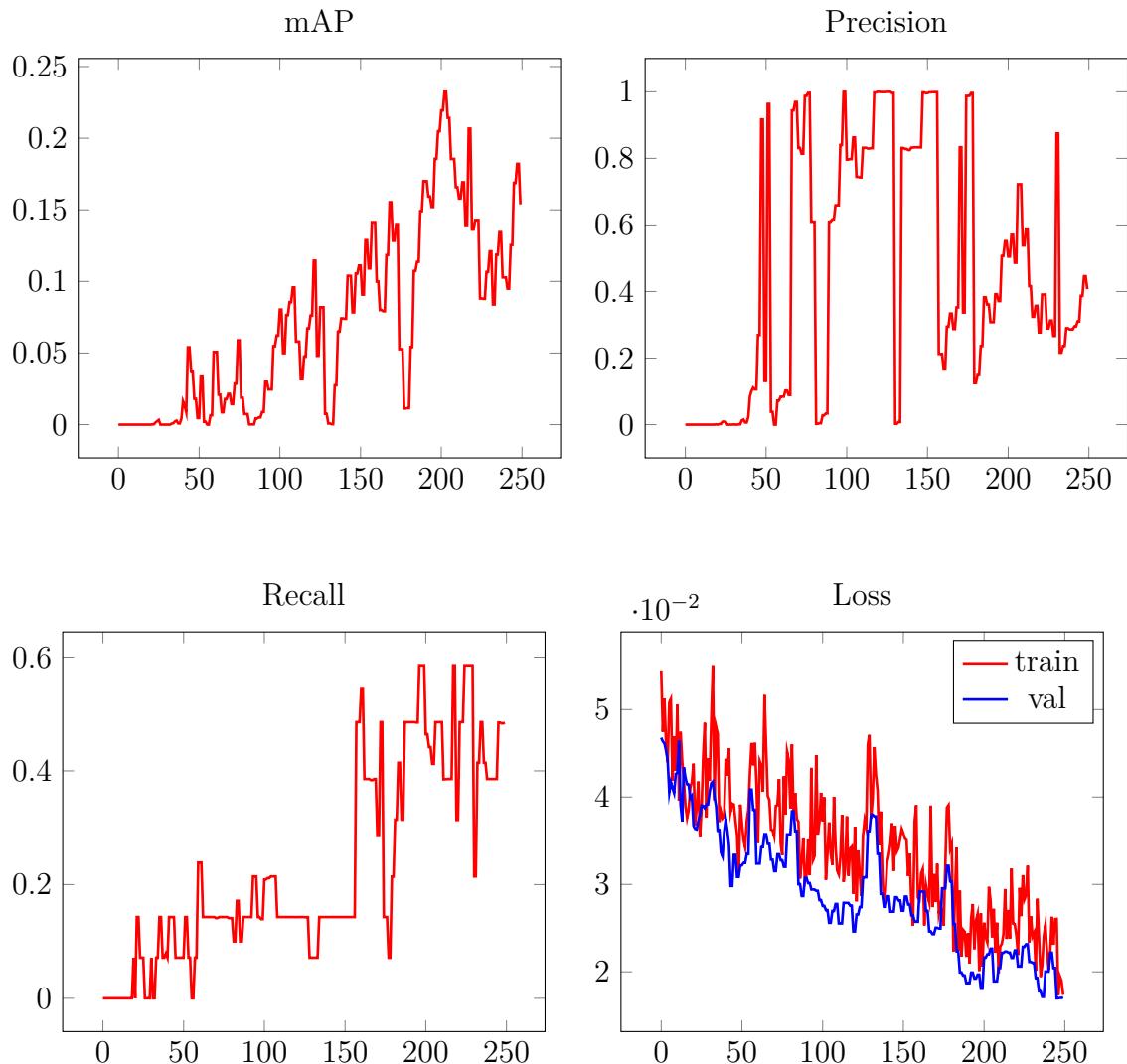


I.2 Walter und Wilma erkennen

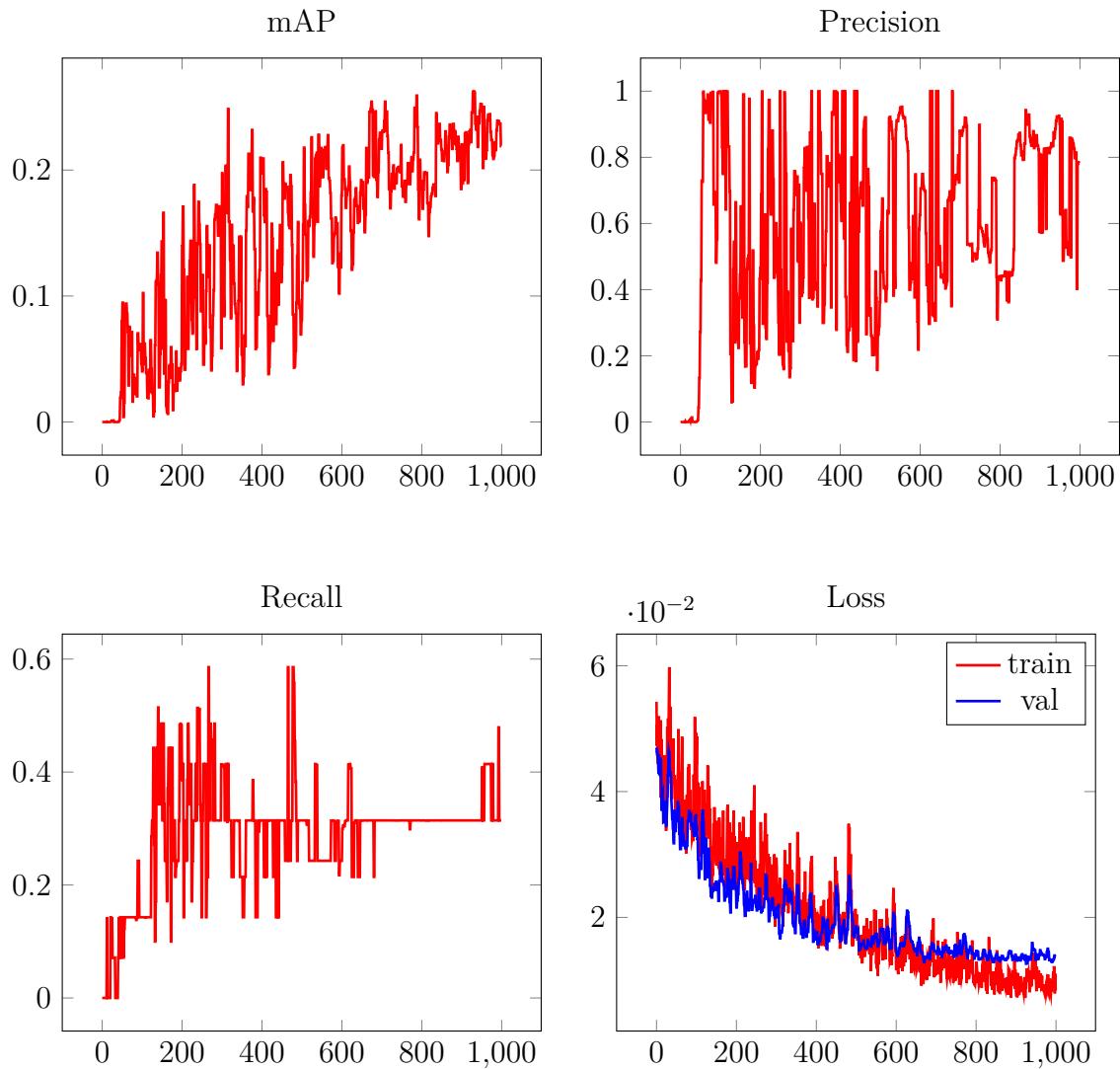
I.2.1 50 Epochen



1.2.2 250 Epochen

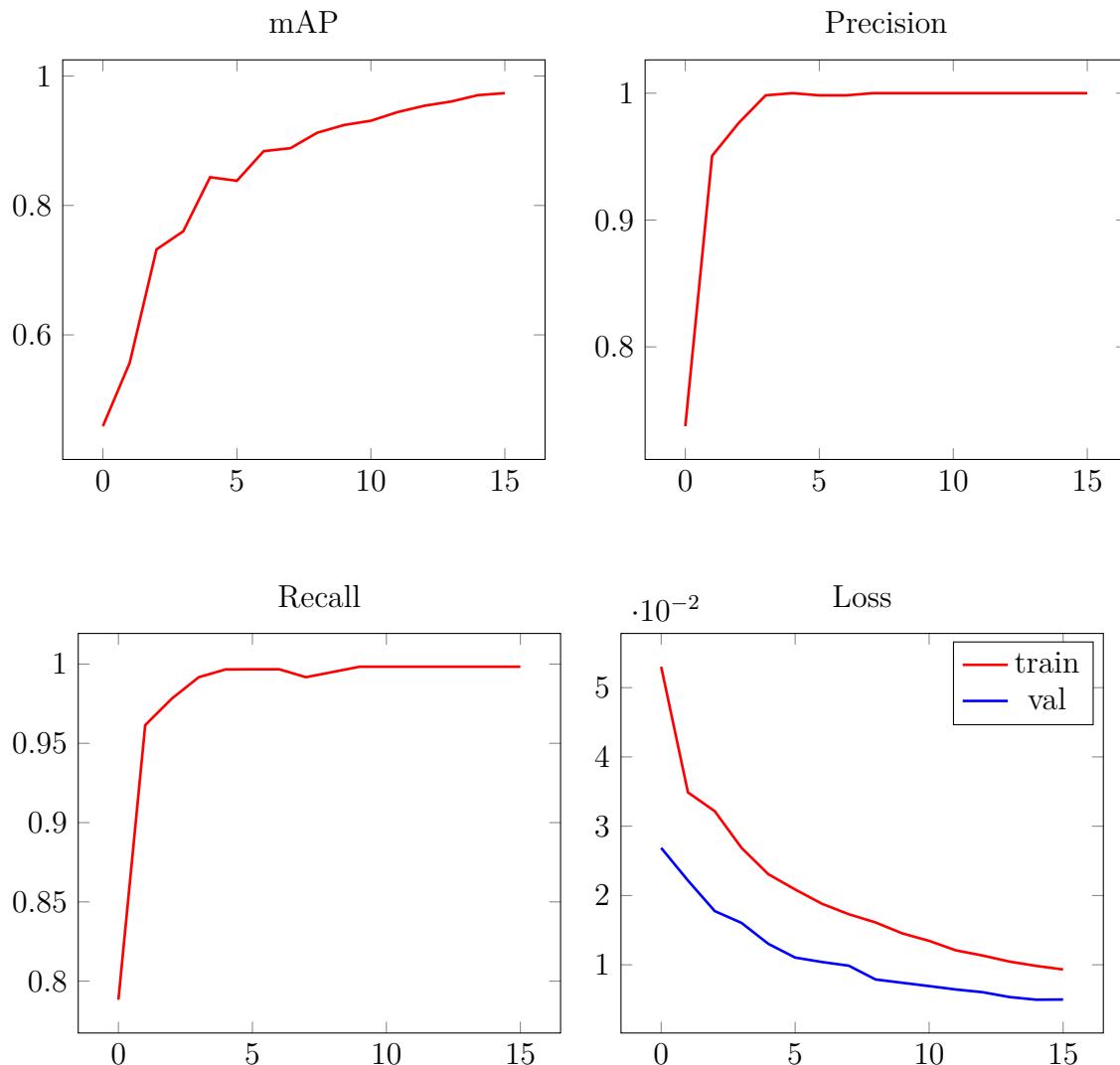


I.2.3 1000 Epochen

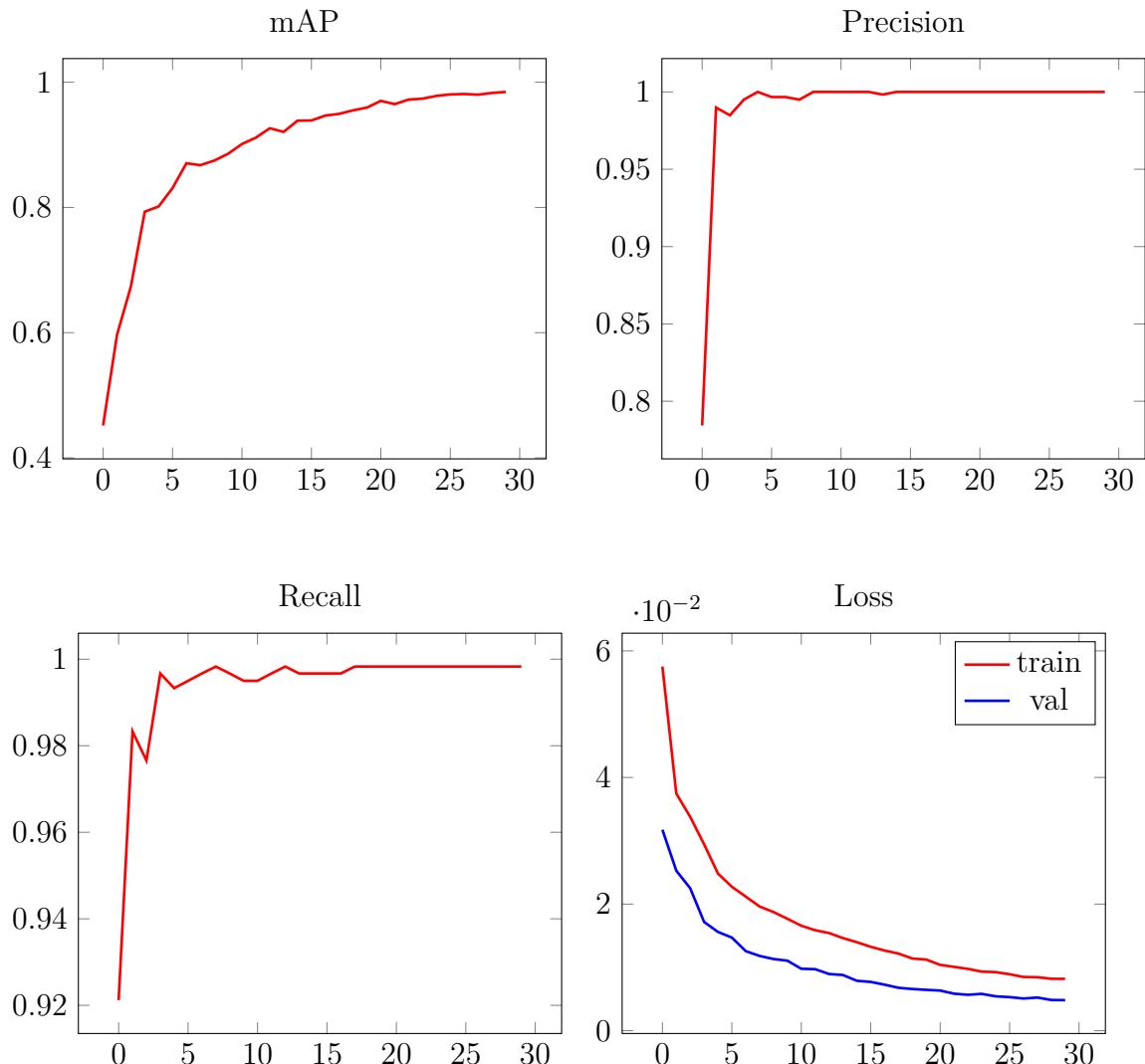


I.3 Epochenvergleich

I.3.1 16 Epochen



I.3.2 30 Epochen



I.3.3 50 Epochen

