

Projektarbeit

Cand.-Ing.: Sebastian Quaisser

Matr.-Nr.: 415536

Cand.-Ing.: Lars Alsbach

Matr.-Nr.: 377537

Cand.-Ing.: Ziya Valiyev

Matr.-Nr.: 417775

Kurzthema: Anwendung von Computer Vision in der kollaborativen Montage

Betreuende Assistentin: Minh Trinh

Aachen, den 18. April 2023

Diese Arbeit wurde vorgelegt am Werkzeugmaschinenlabor WZL,
Lehrstuhl für Fertigungsmesstechnik und Qualitätsmanagement.

Inhalt und Ergebnis dieser Arbeit sind ausschließlich zum internen Gebrauch bestimmt. Alle Urheberrechte liegen bei der RWTH Aachen. Ohne ausdrückliche Genehmigung des betreuenden Lehrstuhls ist es nicht gestattet, diese Arbeit oder Teile daraus an Dritte weiterzugeben.

I Inhaltsverzeichnis

I Inhaltsverzeichnis	i
II Abbildungsverzeichnis	iii
III Tabellenverzeichnis	iv
1 Einleitung	<i>Sebastian Quaisser</i> 1
2 Stand der Technik	2
2.1 Computer Vision	<i>Lars Alsbach</i> 2
2.1.1 Metriken	<i>Ziya Valiyev</i> 3
2.1.2 Neuronale Netze	<i>Lars Alsbach</i> 5
2.1.3 Convolutional Neural Networks	<i>Lars Alsbach</i> 6
2.2 Hyperparameteroptimierung	<i>Sebastian Quaisser</i> 8
2.2.1 Grid Search	8
2.2.2 Genetische Algorithmen	9
2.3 ROS	<i>Sebastian Quaisser</i> 11
2.4 CLAIX-2018	<i>Sebastian Quaisser</i> 12
2.5 Google Colab	<i>Ziya Valiyev</i> 12
3 YOLOv5	<i>Lars Alsbach</i> 13
3.1 YOLOv5	13
3.2 Umwandlung der Bilddaten	14
3.3 Architektur	15
4 Generierung der Datensätze	<i>Lars Alsbach</i> 17
4.1 Erstellung von Trainingsbildern	17
4.2 Labeling	18
4.3 Generierung von Datensätzen in Roboflow	19
5 Training	21
5.1 Trainingsumgebung	<i>Ziya Valiyev</i> 21
5.2 YOLOv5 Modelle	<i>Ziya Valiyev</i> 21
5.3 Vorgehen	<i>Ziya Valiyev</i> 22
5.4 Trainingsparameter	<i>Ziya Valiyev</i> 22
5.5 Trainingsergebnisse	<i>Ziya Valiyev</i> 23
5.6 Optimierung der Hyperparameter	<i>Sebastian Quaisser</i> 24
6 Integration	<i>Sebastian Quaisser</i> 25
6.1 Zielsetzung	25
6.2 Implementierung	25

7 Evaluation	<i>Sebastian Quaisser</i>	27
7.1 Erkennungszeit		27
7.2 Optimierung der Hyperparameter		27
7.3 Anwendung im Montageprozess		29
8 Ausblick	<i>Lars Alsbach</i>	31
8.1 Integration in den gesamten Montageprozess		31
8.2 YOLOv8		31
Literaturverzeichnis		33

II Abbildungsverzeichnis

Abbildung 2.1: Intersection over Union (IoU) [BAEL22]	4
Abbildung 2.2: Zwei-Schichten-Netzwerk [BISH06]	6
Abbildung 2.3: Rekombination (crossover) und Mutation zweier binärer Strings [PADR95]	10
Abbildung 3.1: RGB-Matrix [MHOA19]	14
Abbildung 3.2: YOLOv5-Architektur [KATS22]	15
Abbildung 4.1: Kameraperspektive	17
Abbildung 4.2: Klasse screw	18
Abbildung 4.3: Klasse passive-screw	18
Abbildung 5.1: Erkennung der Schrauben	23
Abbildung 7.1: Fitness der besten Individuen jeder Generation	28
Abbildung 7.2: Vergleich zweier Trainings mit und ohne Hyperparameteroptimierung	29
Abbildung 7.3: Schraubenerkennung mit verschiedenen neuronalen Netzen	30

III Tabellenverzeichnis

Tabelle 5.1: YOLOv5 Modelle	21
Tabelle 5.2: Trainingsbedingungen	23
Tabelle 5.3: Trainingsergebnisse	24
Tabelle 7.1: Vergleich der Standard- und optimierten Werte der Hyperparameter	28

1 Einleitung

Diese Projektarbeit gliedert sich in das Projekt *Cobo Trees* des Werkzeugmaschinenlabors WZL ein. In diesem wird ein kollaborativer Montageprozess behandelt, bei dem Roboter und Mensch zusammenarbeiten, um die Vorteile beider Akteure zu vereinen. Eine vollständig automatisierte Montage ist zwar aufgrund von Effizienz und Lohnkostenersparnis erstrebenswert, allerdings mangelt es dabei an Flexibilität und es ist ein hohes Maß an Expertise erforderlich, um diese einzurichten.

Im behandelten Montageprozess einer Schreibtischlampe soll der Roboter die vom Menschen eingelegten Schrauben automatisch anfahren und festziehen. Die Position, in der sich die Werkstücke und Schrauben dabei befinden, soll flexibel sein. Damit diese unterschiedlichen Positionen erkannt und die entsprechenden Koordinaten an den Roboter gesendet werden können, wird auf Objekterkennung durch Computer Vision zurückgegriffen.

Eine vorherige Projektarbeit [JOEC22] befasste sich bereits mit der Eingliederung eines neuronalen Netzes zur Schraubenerkennung in das Projekt. Ziel dieser Arbeit ist insbesondere die Verbesserung der erreichten Ergebnisse hinsichtlich Genauigkeit und Geschwindigkeit der Auswertung durch die Verwendung eines neuen Netzes und eines erweiterten Trainingsdatensatzes.

In Kapitel 2 werden zunächst die grundlegenden Konzepte sowie die verwendeten Frameworks und Plattformen erläutert. Außerdem wird in Kapitel 3 das in dieser Arbeit verwendete neuronale Netz detailliert beschrieben. Der Kern der Projektarbeit gliedert sich in die Generierung von Datensätzen (Kap. 4), das Trainieren des Netzes (Kap. 5) und die Integration in den Montageprozess (Kap. 6). Abschließend werden in Kapitel 7 die Ergebnisse der Arbeit untersucht und mit den Ergebnissen der Vorgängerarbeit [JOEC22] verglichen, sowie in Kapitel 8 eine Einordnung und mögliche Verbesserungen aufgezeigt.

2 Stand der Technik

In diesem Kapitel werden komprimiert die im Rahmen dieser Projektarbeit verwendeten Technologien und Konzepte dargelegt.

2.1 Computer Vision

Während es für Menschen relativ einfach ist, auf einem gegebenen Bild bestimmte Objekte und deren genaue Position auf diesem zu erkennen, stellt sich diese Aufgabe für Maschinen respektive Computer komplizierter dar. [THUA21] Die wissenschaftliche Disziplin, welche sich mit diesem Sachverhalt beschäftigt, wird als Computer Vision bezeichnet.

Unter Computer Vision versteht man das Interpretieren und Weiterverarbeiten von Bildern und Videos. Diese liegen in Form eines Arrays vor, wobei für jedes Pixel eines Bildes die drei RGB-Werte Rot, Grün und Blau hinterlegt sind. [RASC19] Dabei bieten sich je nach Zielsetzung verschiedene Möglichkeiten an, wie mit diesem Input verfahren werden kann, wie die Klassifizierung des Bildes in vorgegebene Kategorien (Bildklassifizierung) oder das Erkennen vorgegebener Objekte im Bild (Objekterkennung). [RASC19] Die Objekterkennung unterteilt sich wiederum in das Erkennen und Lokalisieren einzelner meist vorher definierte Objekte im Bild, sowie die Zuordnung einzelner Bildbereiche in bestimmte Objektkategorien (Bildsegmentierung). Hierbei wird jedem Pixelbereich im Bild eine bestimmte Kategorie zugeordnet. [LONG14]

Im Zuge dieser Projektarbeit steht das Erkennen und Klassifizieren eingelegter bzw. unbefestigter Schrauben auf und in einem Bauteil, anhand eines Kamerabildes, welches über eine *Intel RealSense* Kamera bereitgestellt wird, sowie die positionelle Lokalisierung dieser im Vordergrund. Daher wird vor allem der Lokalisierungsaspekt als auch der Klassifizierungsaspekt der Computer Vision in Betracht gezogen. Darüber hinaus ist die Zuordnung weiterer Bildaspekte abseits der Schrauben in Objektkategorien wie Montagefläche oder etwaige Personen für das spätere Anfahren durch den Roboter wenig zielführend, sodass der Bildsegmentierungsansatz der Computer Vision nicht betrachtet wird.

Um die Objekterkennung beziehungsweise das Training eines entsprechenden Algorithmus nun konkret durchzuführen, existieren sowohl Ansätze auf der Basis eines neuronalen Netzes, als auch Ansätze, welche kein neuronales Netz verwenden, wie der Ansatz von Viola et al. [VIOL01]. Der Vorteil bei der Verwendung von Ansätzen auf der Basis eines neuronalen Netzes liegt in der zumeist höheren Genauigkeit der Erkennung, auch wenn die Anzahl der zu optimierenden Parameter steigt. [VIOL01] Der in dieser Arbeit verwendete YOLOv5-Algorithmus ist ein Ansatz auf Basis eines neuronalen Netzes, wie noch in Kapitel 3 diskutiert wird.

2.1.1 Metriken

Es existieren diverse Metriken, um ein Computer Vision Modell zu bewerten. Im Folgenden sind die wichtigsten Hilfsmerkmale aufgelistet: [ANWA22]

1. True Positive (TP) beschreibt, wie oft das Modell ein positiv definiertes Muster korrektweise als positiv identifiziert.
2. False Negative (FN) beschreibt, wie oft das Modell ein positiv definiertes Muster fälschlicherweise als negativ identifiziert.
3. False Positive (FP) beschreibt, wie oft das Modell ein negativ definiertes Muster fälschlicherweise als positiv identifiziert.
4. True Negative (TN) beschreibt, wie oft das Modell ein negativ definiertes Muster korrektweise als negativ identifiziert.

Aus den oben aufgelisteten Hilfsmerkmalen werden für die Bewertung des Modells wichtige Metriken abgeleitet, welche während der Validierung des Trainings berechnet und gewertet werden:

1. Genauigkeit (engl.: Precision)

Genauigkeit ist das Verhältnis der Anzahl von richtig vorhergesagten Objekten zur Anzahl der gesamten Vorhersagen. Hierbei entspricht die Anzahl von richtig vorhergesagten Objekten True Positive (TP) und die Anzahl der gesamten Vorhersagen True Positive (TP) + False Positive (FP). Die negativ definierten Muster werden dabei nicht berücksichtigt.

$$p = \frac{TP}{TP + FP} \quad (2.1)$$

2. Recall

Recall ist das Verhältnis der Anzahl der richtig vorhergesagten Objekten (TP) zur Summe von True Positive (TP) und False Negative (FN):

$$r = \frac{TP}{TP + FN} \quad (2.2)$$

3. Intersection over Union (IoU)

IoU ist ein Wert zwischen 0 und 1, der die Ähnlichkeit von zwei Begrenzungsrahmen misst. Dieser wird durch das Verhältnis der Fläche des Schnitts der beiden Begrenzungsrahmen zur Fläche der Vereinigung der beiden Begrenzungsrahmen berechnet. Ein IoU-Wert von

1 bedeutet eine perfekte Übereinstimmung, während ein IoU-Wert von 0 keine Übereinstimmung zwischen den Begrenzungsrahmen anzeigt.

$$IoU = \frac{TP}{TP + FP + FN} \quad (2.3)$$

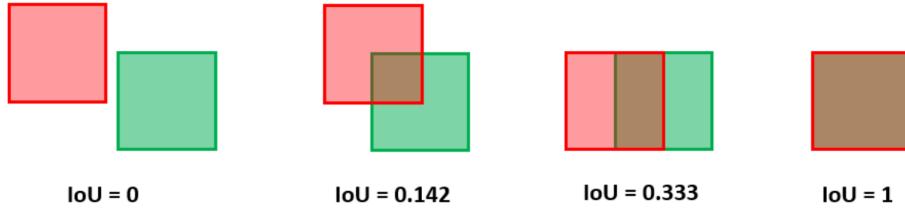


Abbildung 2.1: Intersection over Union (IoU) [BAEL22]

4. Average Precision

Bei den meisten Objekterkennungsalgorithmen wird während des Trainings der Zusammenhang zwischen Genauigkeit und Recall analysiert und daraus ein Graph in der Genauigkeit-Recall Ebene erstellt. Die Fläche dieses Graphen entspricht der Average Precision (AP).

$$AP = \int_0^1 p(r) dr \quad (2.4)$$

5. mAP

mAP ist eine Abkürzung für Mean Average Precision und ist in der Objekterkennung einer der wichtigsten Metriken um das Training zu bewerten. Hierbei werden alle APs addiert und durch den Stichprobenumfang geteilt. mAP@0.5 bezeichnet den mAP-Wert bei einer IoU-Schwelle von 0.5 und mAP@0.5:0.95 bezeichnet den durchschnittlichen mAP-Wert bei einer IoU-Schwelle von 0.5 bis 0.95 mit einem Intervall von 0.05. [HAST22]

$$mAP = \frac{1}{n} \sum_{i=1}^n AP_i \quad (2.5)$$

6. Inference Time

Die Computer Vision Inference beschreibt den Vorgang des Einspeisens von Live-Datenpunkten in einen Computer Vision Algorithmus, um eine Ausgabe, beispielsweise in Form eines einzelnen numerischen Wertes, zu erzeugen. Dieser Prozess ist ein zentraler Bestandteil von Machine Learning Anwendungen und wird in der wissenschaftlichen Forschung sowie in der Praxis eingesetzt, um Datenanalysen und Vorhersagemodelle zu generieren. Die Inference Time bezieht sich auf die Zeit, die benötigt wird, um einen einzelnen Computer

Vision Inference durchzuführen. Es handelt sich um die Dauer, die ein Modell benötigt, um die Eingangsdaten zu verarbeiten und eine Vorhersage zu generieren. Diese Zeit kann je nach Größe und Komplexität des Modells, der Menge an Daten und der verfügbaren Rechenleistung variieren. [HAZE22]

7. Confidence Score

Ein Confidence Score ist eine Zahl zwischen 0 und 1, die die Wahrscheinlichkeit repräsentiert, dass die Ausgabe eines Computer Vision Modells korrekt ist und den Anforderungen des Benutzers entspricht. Der Konfidenzwert wird oft als Maß für das Vertrauen in die Vorhersage oder das Ergebnis eines Modells verwendet. Ein höherer Confidence Score bedeutet, dass das Modell zuversichtlicher in seine Vorhersage ist, während ein niedriger Konfidenzwert darauf hinweisen kann, dass die Vorhersage ungenau sein könnte. Ein Confidence Score von über 0,7 kann als Indikator für ein zuverlässiges Computer Vision Modell angesehen werden. [TONY21]

2.1.2 Neuronale Netze

Um die grundsätzlich Funktionsweise des YOLOv5-Algorithmus zu verstehen, wird kurz auf die Grundidee eines neuronalen Netzes eingegangen. Diese liegt darin, eine Diskriminanzfunktion

$$y(x, w) = f\left(\sum_{j=0}^M w_j^{(2)} \cdot \phi_j(x)\right), \quad (2.6)$$

welche für eine gegebene Eingabe $x \in \mathbb{R}^D$ einen skalaren Ausgang y definiert, dahin zu erweitern, dass nun auch die Basisfunktionen $\phi : \mathbb{R}^n \mapsto \mathbb{R}^M$ angelernt werden. Dabei ist $f : \mathbb{R} \mapsto \mathbb{R}$ eine Aktivierungsfunktion und die Elemente von $w^{(2)} \in \mathbb{R}^M$ sind die zu trainierenden Gewichte in der zweiten Schicht (respektive $w^{(1)} \in \mathbb{R}^M$ in der ersten Schicht des beispielhaften Netzes).

Bei einer Wahl der Basisfunktionen ϕ_j mit einer linearen Abhängigkeit vom Input x , d.h.

$$\phi_j(x) = f\left(\sum_{i=0}^D w_{ji}^{(1)} \cdot x_i\right), \quad (2.7)$$

und dem darauffolgenden Einsetzen in 2.6, ergibt sich der folgende Ausdruck:

$$y(x, w) = f\left(\sum_{j=0}^M w_j^{(2)} \cdot f\left(\sum_{i=0}^D w_{ji}^{(1)} \cdot x_i\right)\right) \quad (2.8)$$

Eine grafische Darstellung dieses Vorgehens ergibt die untenstehende neuronale Netzstruktur:

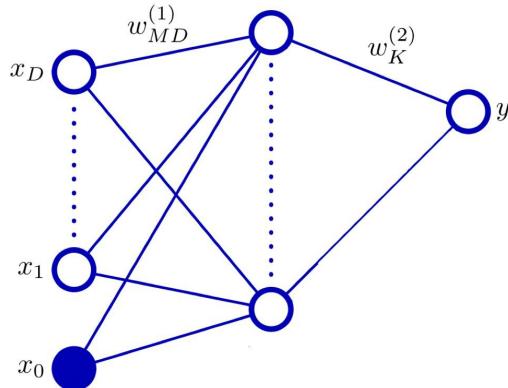


Abbildung 2.2: Zwei-Schichten-Netzwerk [BISH06]

Dieses Vorgehen wird beliebig fortgesetzt; dabei wird mit jeder neu definierten Menge an Basisfunktionen eine neue Schicht im vorliegenden Netz definiert. [BISH06].

Um die Gewichte für ein gegebenes Problem zu optimieren, wird mithilfe eines Trainingsdatensatzes bestehend aus Eingaben $x_i \in \mathbb{R}$, sowie zugehörigen Ausgaben $t_i \in \mathbb{R}$ ein Fehler $E(w)$ zwischen der Vorhersage des neuronalen Netzes y_i und des korrekten Wertes t_i bestimmt (beispielsweise der Mean-Squared-Error), und daraufhin versucht, die Gewichte w des Netzes so zu optimieren, dass dieser Fehler $E(w)$ minimal wird.

Ein häufiges Verfahren, diese optimalen Gewichte zu finden, ist das Gradientenverfahren (engl. gradient descent). [BISH06] Um hierbei auf effiziente Art und Weise die Gradienten für jedes einzelne Gewicht zu finden, bietet sich der von Rumelhart et al. vorgeschlagene Backpropagation-Algorithmus an. [RUME86]

2.1.3 Convolutional Neural Networks

Einen wichtigen Baustein des in dieser Projektarbeit verwendeten YOLOv5-Algorithmuses bilden die sogenannten Convolutional Neural Networks, beziehungsweise die darin enthaltenen Faltungsschichten (engl.: Convolutional Layers), deren Funktion nun kurz dargelegt wird.

Um Bilder als Matrixeingabe für ein neuronales Netz zu verwenden, wäre bei der Verwendung eines Fully Connected Networks mit der gleichen Anzahl an versteckten Neuronen (engl.: Hidden-Units) ein Verbindungsgewicht von jeder einzelnen Matrixzelle zu jedem einzelnen versteckten Neuron vonnöten. Für ein beispielhaftes Eingabebild einer Dimension von 1000×1000 Pixeln, unter der Betrachtung nur einer Farbdimension, ergäbe sich die folgende Anzahl an Eingabewerten (#inputs): [KHAN20]

$$\#inputs = 1000 \cdot 1000 = 10^6 \quad (2.9)$$

Für eine erste Schicht mit der identischen Anzahl an versteckten Neuronen (`#hiddenUnits`) berechnete sich die Anzahl der zu optimierenden Parameter (`#weights`), aufgrund der Verbindung von jeder Eingabe zu jedem versteckten Neuron, wie folgt:

$$\#weights = \#hiddenUnits \cdot \#inputs = 10^6 \cdot 10^6 = 10^{12} \quad (2.10)$$

Damit wären bei einem gebräuchlichen Bildformat von 1000×1000 Pixel die beachtliche Anzahl von einer Billion Parametern zu optimieren.

Dieses Wachstum der Parameterzahl in Abhängigkeit der Eingangsdimension wird auch als Fluch der Dimension bezeichnet. [BISH06]

Um die Anzahl der Parameter von der Größe des Eingangsbildes zu entkoppeln, wird auf die Idee der bereits angesprochenen Convolutional Neural Networks zurückgegriffen, welche im Folgenden kurz erläutert wird. Diese von Fukushima et al. vorgestellte Idee eines neuronalen Netzes profitiert von der paarweisen Anwendung von Faltungs- (engl.: convolutional layer) und Pooling-Schichten. [FUKU80]

Wie in 3.2 beschrieben, besteht der vorliegende Inputtensor aus drei Farbmatrizen F_{rot} , F_{gruen} und F_{blau} , welche als Input für das Netzwerk dienen. In der Faltungsschicht wird eine Dimensionsreduzierung der Eingabematrix erreicht, indem eine $H_{color} \in \mathbb{R}^{j \times k}$ Faltungsmatrix für jede einzelne der drei Farbdimension über die Eingabematrix F_{color} geschoben und dabei eine diskrete Faltung berechnet wird: [SKAL19]

$$C_{color}[m, n] = \sum_j \sum_k H_{color}[j, k] \cdot F_{color}[m - j, n - k] \quad (2.11)$$

Hierbei ist C_{color} die Ausgabematrix für eine bestimmte Farbdimension und $C_{color}[m, n]$ der Output für den Eintrag an der Stelle $[m, n]$.

Um nun die komplette Ausgabematrix C zu erhalten, werden die entsprechenden Matrizen C_{color} addiert:

$$C = \sum_{color \in Colors} C_{color} \quad (2.12)$$

Die in den Faltungsmatrizen H_{color} verwendeten Gewichte sind örtlich konstant, d.h. auch für die positionell folgende Faltung $C[m + 1, n + 1]$, welche aus den einzelnen Faltungsmatrizen für jede Farbe H_{color} und den Eingabematrizen für jede Farbe F_{color} nach obiger Vorschrift berechnet wird, werden dieselben Gewichte verwendet, sodass die Anzahl der Parameter nicht mehr mit der Größe des Bildinputs skaliert. Damit hängt die Anzahl der zu optimierenden Parameter nur von der Dimension der verwendeten Faltungsmatrizen H_{color} ab, und skaliert nicht mehr, wie es noch beim Fully-Connected-Network der Fall war, exponentiell mit der Größe

des Eingabeinputs, wodurch eine deutliche Komplexitätsreduktion des Optimierungsproblems sowie ein Laufzeitgewinn, da nun weniger Parameter optimiert werden müssen, einhergehen. Angelernt werden diese entsprechenden Gewichte analog zu dem in Kapitel 2.1.2 beschriebenen Vorgehen.

Auch in den Pooling-Schichten wird eine Dimensionsreduzierung vorgenommen, allerdings ohne dass dabei explizit eigene Gewichte angelernt werden. Stattdessen wird ein bestimmter Wert in einem vorher definierten Matrixbereich extrahiert. Dies kann ja nach Art des Poolings der Maximalwert (engl.: Max-Pooling) oder der Durchschnittswert sein (engl.: Average-Pooling). [FUKU80]

Durch das Einbringen dieser beiden Schichten ergeben sich die folgenden zwei Vorteile: zum einen wird die bereits angesprochene Entkoppelung der Komplexität des Netzwerkes von der Eingabegröße erreicht, da nun die Anzahl der von zu optimierenden Gewichte unabhängig von der Größe der Eingabe ist; zum anderen wird auf die Hierarchie des Eingabebildes eingegangen, indem zunächst die bedeutesten Bildmerkmale (engl.: high-level features) des Eingabebildes extrahiert werden, welche darauf folgend in stetig schrumpfende Bildmerkmale zerlegt werden.

Angewendet auf das vorliegende Problem der Schraubenerkennung bedeutete dies zunächst eine Extraktion der Montagefläche aus dem Eingabebild, darauf folgend die Extraktion des Bauteils, gefolgt von der eigentlichen Schraube sowie der abschließenden Klassifizierung.

2.2 Hyperparameteroptimierung

Ein Computer Vision Modell wie YOLOv5 umfasst eine Reihe an Hyperparametern. Diese sind Parameter, die nicht durch das maschinelle Lernen bestimmt werden, sondern vor Trainingsbeginn festgelegt werden und das Verhalten des Modells beeinflussen. [CLAE15] YOLOv5 beinhaltet ca. 30 dieser Parameter, beispielsweise die *learning rate*, welche bestimmt, wie stark die Gewichte in jeder Iteration geändert werden. Werden diese nicht durch den Nutzer spezifiziert, wird auf Standardwerte zurückgegriffen, welche bereits für das Training von Grund auf mit dem *Common Objects in Context (COCO)* Datensatz optimiert sind. [GITH20] Da für jeden Datensatz das Training jedoch unterschiedlich abläuft, unterscheiden sich die optimalen Werte der Hyperparameter. Diese zu bestimmen erweist sich allerdings als sehr komplex. [CLAE15]

2.2.1 Grid Search

Eine Möglichkeit zur Bestimmung geeigneter Parameterwerte ist die sogenannte Grid Search. Dabei wird für jeden zu optimierenden Hyperparameter eine Anzahl an Werten festgelegt, welche evaluiert werden sollen. Dadurch ergibt sich ein Gitter aus unterschiedlichen Kombinationen der Werte mit einer Dimensionalität in Höhe der Anzahl Parameter. Zur Bestimmung dieser Parameter wird für jeden Gitterpunkt, also jede Kombination der zulässigen Werte, ein Training

durchgeführt. Anschließend wird bewertet, welche Einstellung optimal ist. Als Metrik dient dazu eine *score function*, beispielsweise die mean Average Precision (mAP, siehe Kapitel 2.1.1). [SCIK]

Die Bestimmung geeigneter Parameter ist auf diese Weise extrem aufwändig, da jedes einzelne Training bereits eine beträchtliche Rechenkapazität in Anspruch nimmt, welche mit der Anzahl der durchgeführten Trainings skaliert. Letztere wird durch die Größen und Abstufungen der Suchintervalle, vor allem aber durch die Anzahl der Parameter bestimmt. Für 30 verschiedene Hyperparameter, wie sie bei YOLOv5 vorhanden sind, ergeben sich bereits bei einer Auswahl von zwei Werten pro Parameter $2^{30} \approx 10^9$ Kombinationen. Aufgrund dieser extrem hohen Dimensionalität ist die Anwendung der Grid Search nicht praktikabel.

2.2.2 Genetische Algorithmen

Eine für hohe Parameterzahlen geeignetere Methode ist die in YOLOv5 implementierte Hyperparameterevolution, welche auf einem Genetischen Algorithmus (GA) basiert. Bei GA handelt es sich um Algorithmen, welche die Entwicklung realer Lebensformen und das Prinzip der natürlichen Selektion imitieren. Im Gegensatz zu herkömmlichen Verfahren wird nicht durch eine determinierte Transition zwischen festen Zuständen (beispielsweise den Gitterpunkten in Kap. 2.2.1) gewechselt; ausgehend von einer Basispopulation mehrerer Zustände werden durch probabilistische Transitionsregeln neue Populationen generiert. Diese neuen Generationen erhalten Eigenschaften der „Eltern“ sowie Mutationen. In jeder Generation werden die einzelnen Individuen (Zustände) durch eine Fitness-Funktion evaluiert, welche das zu optimierende Gütemaß darstellt. Die Eigenschaften der besser bewerteten („fitteren“) Individuen werden mit einer höheren Wahrscheinlichkeit weitervererbt. [GOLD89]

Zunächst wird eine Chiffrierung der Parameter vorgenommen, welche mehr oder weniger intuitiv sein kann. Für ein einfaches Problem mit dichotomen Variablen kann beispielsweise eine Binärcodierung verwendet werden, bei der jedes Bit einer Variable mit den möglichen Zuständen 0 und 1 entspricht. In komplexeren Fällen ist die Chiffrierung hingegen weniger offensichtlich und nachvollziehbar. Anschließend wird mit diesen Zeichenketten (*Strings*) gearbeitet, nicht mit den Parametern selbst, wodurch sich die GA von klassischen Herangehensweisen unterscheiden. Die Manipulation der Strings geschieht durch drei Operatoren: Reproduktion, Rekombination und Mutation. Durch die Reproduktion werden Nachkommen eines Strings in der Folgegeneration erzeugt. Die zur Fitness der Individuen proportionale Wahrscheinlichkeit der Reproduktion entspricht einer künstlichen Nachahmung der natürlichen Selektion („Survival of the fittest“). Bei der Rekombination wird ein zufällig bestimmter Anteil zweier zuvor ausgewählter Strings vertauscht. Dieser Prozess ist eine offensichtliche Imitation der natürlichen Zeugung von Nachkommen. Zusätzlich werden mit einer geringen Wahrscheinlichkeit Mutationen, Änderungen an einzelnen Stellen der Strings, vorgenommen. Dadurch wird sichergestellt, dass die Optimierung

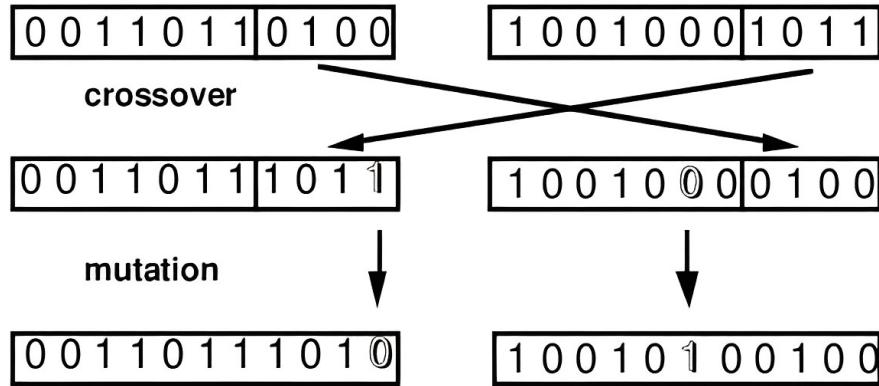


Abbildung 2.3: Rekombination (crossover) und Mutation zweier binärer Strings [PADR95]

nicht auf die Merkmale der Startpopulation begrenzt sind, sondern der gesamte Parameterbereich in Betracht gezogen wird. [GOLD89]

In Abbildung 2.3 ist eine einfache Rekombination dargestellt sowie eine Flip Mutation, wie sie häufig bei Strings in Binärcodierung zum Einsatz kommt. Andere Möglichkeiten der Mutation sind beispielsweise Random Resetting, Swap Mutation und Scramble Mutation. [TUTO23]

Ein Optimierungsalgorithmus basierend auf diesen simplen Operatoren erweist sich als überraschend effizient. Außerdem wird durch die Betrachtung von Populationen anstatt einzelner Individuen eine Robustheit gewährleistet, über welche klassische Optimierungsmethoden nicht verfügen. Letztere benötigen zusätzlich Hilfsinformationen wie z.B. Ableitungen zur analytischen Bestimmung der optimalen Parameter, welche bei der Anwendung eines GA nicht notwendig sind. [GOLD89]

2.3 ROS

ROS (Robot Operating System) ist ein open-source Framework zur Robotersteuerung, welches eine breite Kompatibilität für unterschiedlichste Hardware verspricht. [OPENa] Diese wird durch verschiedene Prinzipien gewährleistet, auf die das Design von ROS ausgelegt ist:

- **Peer-To-Peer**

Ein durch ROS aufgebautes System kann als Graph dargestellt werden. Die Knoten des Graphen bilden *Nodes*, einzelne Prozesse, die auf verschiedenen Hosts laufen können. Diese Nodes kommunizieren miteinander über *Topics* nach dem Peer-to-Peer Prinzip, das heißt alle Nodes sind gleichberechtigt. Es können jeweils mehrere Nodes einem Topic Nachrichten senden (*publish*) und diese auslesen (*subscribe*). Die Nachrichten werden als *Messages* übermittelt, welche Datenstrukturen mit einem definierten Typ sind. ROS stellt eine Reihe an häufig verwendeten Datentypen im Package *std_msg* zur Verfügung; darüber hinaus können nutzerdefinierte Messages implementiert werden. [QUIG09]

- **Thin**

Um die Wiederverwendung von bereits geschriebenem Code zu erleichtern, wird die „thin“ Ideologie verwendet. Das Build-System von ROS erzeugt eine modulare Struktur aus Bibliotheken, welche selbst unabhängig von ROS implementiert werden können und nur durch „dünne“ ausführbare Dateien mit klar definierten Schnittstellen mit ROS interagieren. Dies ermöglicht eine einfache Extraktion des essenziellen Codes aus den einzelnen Bibliotheken. [QUIG09]

- **Sprachunabhängig**

Das Design des ROS Frameworks ist sprachunabhängig, sodass die Sprache des Quellcodes frei nach Präferenz des Nutzers gewählt werden kann. Die Implementierungen in Python, C++ und Lisp sind bereits abgeschlossen, während an weiteren Sprachen wie z.B. Java noch gearbeitet wird. [OPENa]

- **Open-source**

Der gesamte Quellcode von ROS ist frei zugänglich und unter BSD-Lizenz sowohl nicht-kommerziell als auch kommerziell zu nutzen und zu verändern. [QUIG09]

ROS ist außerdem durch seine Modularität und integrierte Testframeworks einfach zu debuggen und für großskalige Anwendungen praktikabel. Die einzelnen Prozesse können in *packages* organisiert werden. Diese werden durch den in ROS standardmäßig genutzten Build-Prozess catkin [ROBOOb] erstellt, welcher auf CMake [KITW] basiert. [OPENa, OPENb]

2.4 CLAIX-2018

CLAIX-2018 (Cluster Aix-la-Chapelle) ist die zweite und aktuelle Version des High-Performance-Computing Clusters der RWTH Aachen. Mit einer Peak Performance von 4,112 PFlop/s liegt CLAIX momentan auf Platz 286 in der *Top500* Liste der leistungsstärksten Supercomputer der Welt. [TOP5] Für Studenten der RWTH Aachen steht ein monatliches Pensum von 500 Core-hours frei zur Verfügung, darüber hinaus können Projekte mit höheren Ressourcenkontingenten beantragt werden.

2.5 Google Colab

Colab ist ein Cloud Computing Dienst angeboten von Google, auf welchem Jupyter Notebooks ausgeführt werden können. Dabei werden seitens Google bei kostenfreier Nutzung 12 Stunden Laufzeit pro Notebook zugelassen. Colab stellt mehrere Intel Xeon CPUs mit einer Frequenz von 2.2 GHz, 13 GB RAM, Tesla K80 Beschleuniger und 12 GB GDDR5 VRAM zur Verfügung. [DAS22]

3 YOLOv5

3.1 YOLOv5

YOLOv5 (englisches Akronym für You only look once) ist ein trainierbares neuronales Netz für Computer Vision, welches auf den bereits diskutierten Convolutional Neural Networks (CNNs) basiert. [ULTRb, THUA21] Es wurde von Ultralytics LLc. entwickelt und ist der Nachfolger von YOLOv4. Im Vergleich zu seinem Vorgänger ist es in Python geschrieben anstatt in C, was die Integration in bestehende Systeme, wie im Zuge dieser Projektarbeit als verbessertes Werkzeug zur Schraubenerkennung, welche als Grundlage für die anschließende Anfahrt durch den Montageroboter dient, bedeutend vereinfacht. [THUA21]

Die Grundidee von YOLOv5 besteht nun darin, für ein gegebenes Eingabebild, welches nach der in 3.2 beschrieben Vorgehensweise dem neuronalen Netz bereitgestellt wird, für einzelne im Bild befindliche Objekte rechteckige Objektkästen zu optimieren, welche im Ausgabebild einmal die Position des Objektes definieren, sowie über eine farbliche Kennzeichnung des Objektkastens eben jenes in eine vorher definierte Objektklasse einklassifizieren.

Dabei wird ein Gitter über das Eingabebild gelegt, welches dieses in gleich große Bereiche einteilt. Zusätzlich besteht jede dieser Zellen aus einer bestimmten Anzahl von Bounding-Boxes B_i , deren Position über einen Zellmittelwert c_i , sowie eine Höhe $height_i$ und Breite $width_i$ definiert wird. Aufgrund der Rechtecksgestalt der Objektkästen sind diese drei Parameter für die Eindeutigkeit jeder Bounding-Box ausreichend. Darauf hinaus bestimmt der Confidence-Wert $conf_{ij}$ für jede Bounding Box B_i und jedes Objekt j , die Wahrscheinlichkeit, dass besagte Bounding-Box das entsprechende Objekt enthält. Dieser Wert wird wie folgt berechnet: [MENE18, THAT20]

$$conf_{ij} = P(Object_j) \cdot IoU \quad (3.1)$$

Dabei ist $P(Object_j)$ die Wahrscheinlichkeit, dass sich das entsprechende Objekt in dieser Bounding-Box befindet, und IoU die *Intersection over Union*. Dieser Wert misst, inwiefern sich die Bounding-Box, welche im Zuge des Trainings optimiert wird, sowie die Ziel-Bounding-Box, welche als Vorhersagewert aus dem Trainingsdatensatz entnommen wird, überlappen (siehe auch Kapitel 2.1.1) [MALL22, MENE18].

Mit diesem Confidence-Wert $conf_{ij}$ ergibt sich zum einen ein quantitativer Wert dafür, inwiefern die zu optimierende Bounding-Box bereits mit der korrekten Ziel-Box des entsprechenden Objektes übereinstimmt, sowie einen Wert dafür, mit welcher Sicherheit in entsprechender Box das besagte Objekt liegt.

Diese Confidence-Werte für jede einzelne Klasse sind für die Bounding-Boxes in einer Grid-Cell identisch. Im Zuge des Trainings wird nun versucht die fünf Parameter jeder Bounding-Box (zwei

für die x-y-Koordinate des Zentrums, ein Wert für die Höhe, ein Wert für die Breite, sowie der Confidence-Wert für jede Klasse), nach dem in 2.1.2 beschrieben Vorgehen zu optimieren, um somit final die Objektkästen der entsprechenden Objekte zu erhalten.

Um mit den unterschiedlichen Bilddimensionen in der Eingabe umzugehen, sowie die Hierarchie eines Bildes mit in Betracht zu ziehen, werden die in Kapitel 2.1.3 beschriebenen Netze verwendet. Vorher wird noch kurz auf die Umwandlung des Bildinputs, in eine für ein neuronales Netz verwendbares Format eingegangen.

3.2 Umwandlung der Bilddaten

Für die Verwendung der Bilddateien als Eingabedaten für ein zu trainierendes neuronales Netz, ist vorher notwendig, diese in ein für das neuronale Netz verwendbares Format zu bringen.

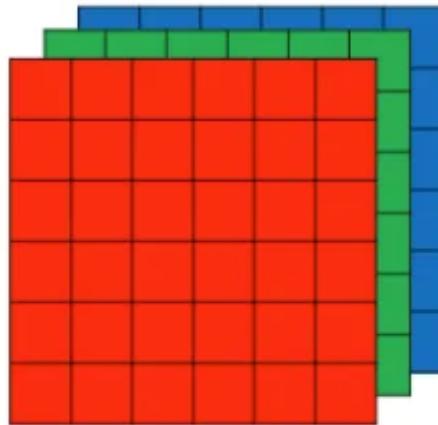


Abbildung 3.1: RGB-Matrix [MHOA19]

Hierfür werden die Bilddateien in ihre Rot-, Grün- und Blauwerte (RGB-Werte) unterteilt, und diese numerischen Werte in drei Farbmatrizen F_{rot} , F_{gruen} und F_{blau} , welche die Dimension der Pixelhöhe multipliziert mit der Pixelbreite der Bilddateien haben, hinterlegt. Mit steigender Auflösung unsere Bilder steigt demnach die Dimension der verwendeten Matrix, was zu Komplexitätsproblemen führen wird, wie in Kapitel 2.1.3 beschrieben.

Auf diese Art und Weise wird demnach ein Tensor der Dimension 3 x Bildbreite x Bildhöhe definiert, welcher als Input für unser neuronales Netz verwendet wird.

3.3 Architektur

Nachdem in den Kapiteln 2.1.3 und 3.1 auf die wesentlichen Elemente von YOLOv5 eingegangen wurde, wird nun die konkrete Architektur dargelegt.

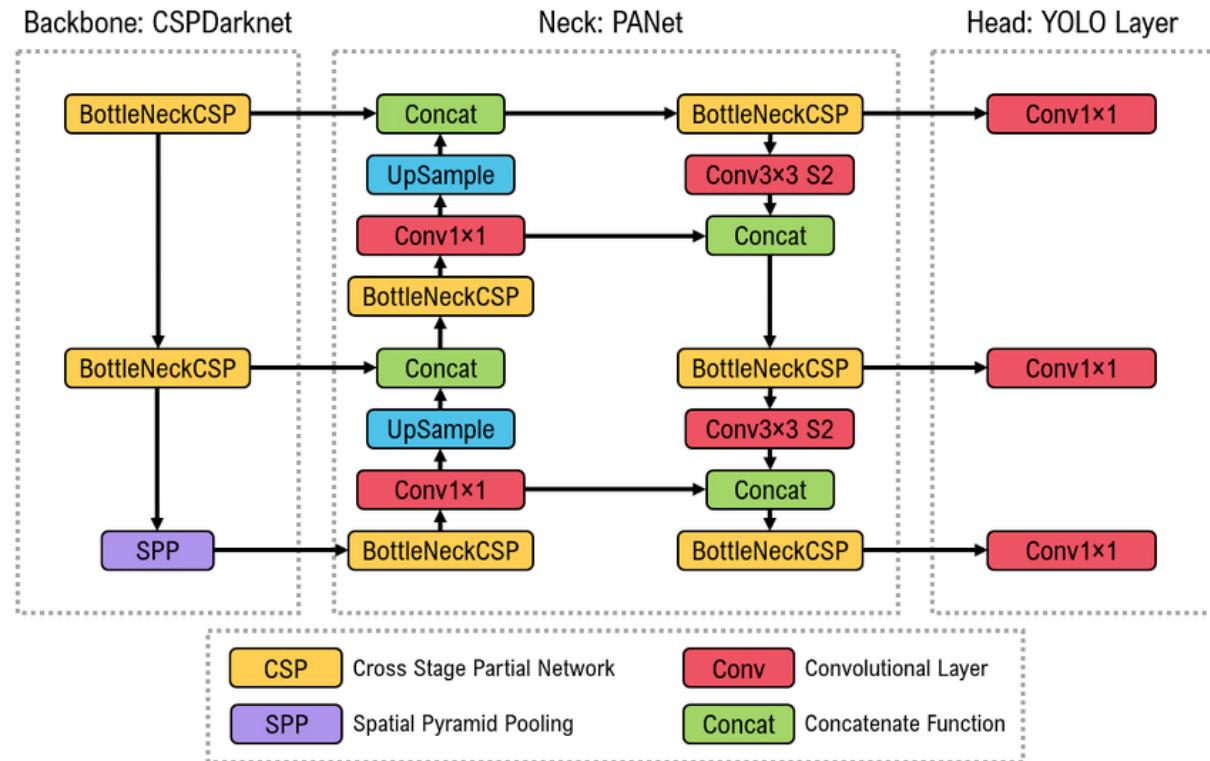


Abbildung 3.2: YOLOv5-Architektur [KATS22]

In der untenstehenden Abbildung ist der Aufbau des YOLOv5-Netzes grafisch dargestellt. Es besteht aus einem Backbone, einem Neck, und einem Head, deren Funktionen nun kurz dargelegt werden.

1. Backbone

Im Backbone- (dt. Rückgrat) Teil von YOLOv5 wird mithilfe der in Kapitel 2.1.3 beschriebenen Convolutional Neural Networks die Extraktion der den Bildcharakter dominierenden Eigenschaften (engl.: Features) vorgenommen, und in Zuge dessen die bereits erwähnte Dimensionsreduzierung des Eingabebildes erreicht. In YOLOv5 wird dafür ein CSPDarknet53 verwendet. Näheres zu diesem Aufbau findet sich bei Wang et al. [WANG19].

2. Neck

Der Neck (dt. Nacken) dient als Schnittstelle zwischen Backbone und Head und hat die Aufgabe, eine Feature Aggregation durchzuführen. [THUA21] Dies bedeutet, dass der dimensionsreduzierte Input, bzw. die erhaltenen Features, welche wir aus unserem Backbone erhalten, gemischt und auf verschiedene Arten kombiniert werden, um die Robustheit der

extrahierten Features zu verstrken. [SOLA22] Die auf diese Weise aggregierte Feature-Map dient anschlieend als Eingabe fr die Head-Schicht. In YOLOv5 wird dafr die von Liu et. al verwendete Architektur des PAN-Netzes (Path-Aggregation Network) verwendet. [LIU18]

3. Head

Im Head (dt. Kopf) findet die in Abschnitt 3.1 bereits beschriebene Lokalisierung und Optimierung der einzelnen Bounding-Boxes B_i statt. Diese beiden Aufgaben werden im Head bei YOLOv5 simultan ausgefhrt. [THUA21]

Die beschriebene Struktur stellt die Architektur von YOLOv5 dar, welche sich durch evolutionre Verbesserungen aus den Vorgngern YOLOv1 - YOLOv4 ergeben hat. Dabei ist die Grundstruktur identisch geblieben, und es wurden in jedem Entwicklungsschritt kleinere Anpassungen zur Verbesserung der Leistungsfhigkeit vorgenommen. Zwei bedeutende Entwicklungsschritte sollen kurz dargelegt werden:

1. Verwendung von Batch Normalization seit YOLOv2

Bei der Batch Normalisation werden die Eingabewerte fr jede Schicht des Netzes auf einen einheitlichen Erwartungswert und Standardabweichung normiert, um die Stabilitt zu verbessern. [THUA21]

2. Einbringen von Residual Neural Networks in YOLOv3

Residual Neural Networks sind durch das Einbringen sogenannter skip-connections zwischen einzelnen Schichten gekennzeichnet. Diese haben die Aufgabe wrend des Updates der Gewichte im Zuge des Backpropagation-Algorhitmuses, einen stabileren Fluss der Gewichtsgradienten zu gewhrleisten, um damit das Problem der Verschwindenden Gradienten (engl.: Vanishing Gradients) zu dmpfen. [HE15]

In der Zwischenzeit wurden von Ultralytics LLC. weitere verbesserte Versionen von YOLO verffentlicht. Die neuste stellt dabei YOLOv8 dar, welche im Januar 2023 verffentlicht wurde. [SOLA23]

Da dieses Datum nach dem Beginn der vorliegenden Projektarbeit liegt, wird YOLOv8 in dieser Projektarbeit nicht verwendet und auf dessen Verwendungsmglichkeiten im Zuge der gegebenen Problemstellung nur in Kapitel 8 eingegangen.

4 Generierung der Datensätze

4.1 Erstellung von Trainingsbildern

Um den Erfolg des Trainings und der anschließenden Verwendung des erstellten Modells sicherzustellen, ist eine sorgsame und vielseitige Generierung von Trainingsbildern notwendig. In dieser Arbeit werden Bilder verwendet, welche mit zwei *Intel RealSense* Kameras aufgenommen wurden.

Diese haben eine fixe Position, und erstellen dadurch Bilder, welche stets die selbe Perspektive auf die Montageflächen bieten.

Zusätzlich wird auf 100 bereits erstellte Bilder aus einer vorigen Projektarbeit zurückgegriffen [JOEC22], um eine höhere Varianz in den Trainingsbildern zu erreichen.



Abbildung 4.1: Kameraperspektive

Im Zuge der Bilderaufnahme wird versucht, verschiedene Szenarien im Montageprozess des Bauteils abzubilden, indem die Position des Bauteils, die Anzahl der bereits montierten Schrauben (eine, zwei oder drei eingelegte Schrauben) sowie die Farbe des Bauteils (rot oder grau) variiert wird. Zusätzlich werden sowohl auf der Montagefläche als auch auf dem Bauteil an sich, weitere, unbefestigte Schrauben platziert, um die Varianz der Trainingsbilder zu erhöhen, und um eine höhere Robustheit gegenüber unbefestigten Schrauben zu gewährleisten.

Insgesamt wurden auf diese Art und Weise 401 Trainingsbilder erstellt, welche im nachfolgenden Schritt annotiert werden. Zusammen mit den 100 Trainingsbildern der vorigen Kamera ergibt sich somit eine Gesamtanzahl von 501 Trainingsbildern.

Um einen anfänglichen Eindruck des Erkennungsverhaltens des YOLOv5-Algorhitmus zu erhalten, wurde für die ersten Trainingsanläufe auf 100 Trainingsbilder aus einer vorigen Projektarbeit zurückgegriffen. Für einumfassenderes Training wird dann der vollumfängliche Trainingsdatensatz von 501 Bildern verwendet. Entsprechendes Vorgehen wird in Kapitel 5 dargelegt.

4.2 Labeling

Da es sich bei der Objekterkennung um eine Form des überwachten Lernens handelt, ist es nötig, vor dem eigentlichen Training die erstellten Bilder zu labeln, das heißt im Bild selbst den Pixelbereich zu markieren, in welchem sich die zu erkennende Schrauben befindet. [RASC19] Nur auf diese Weise ist es für YOLOv5 möglich, die charakterisierenden Eigenschaften des zu erkennenden Objekts zu lernen, um somit in der eigentlichen Verwendung des trainierten Modells eigenständig Objekte erkennen zu können.

Für die einfache und unkomplizierte Umsetzung dieses Schrittes bietet sich das Annotation-Tool Roboflow an. [ROBOa] Dieses bietet die Möglichkeit, eigens erstellt Bilder hochzuladen und auf diesen den Pixelbereich zu markieren, welcher von YOLOv5 erkannt werden soll. Um die Unterscheidbarkeit unterschiedlicher Objekte zu gewährleisten, bietet Roboflow darüber hinaus an, verschiedene Objektklassen zu definieren.

Die im Zuge dieser Arbeit erstellten 501 Trainingsbilder werden in Roboflow eingebunden, und auf diesen die entsprechenden Schrauben im Bauteil sowie die unbefestigten Schrauben, welche auf der Montagfläche verteilt liegen, mithilfe rechteckiger Objektkästen markiert.

Dabei werden die Klassen "screw" für die zu erkennenden Schrauben, sowie die Klasse "passive-screw" für die Schrauben, welche unbefestigt auf der Montagefläche sowie dem Bauteil selbst platziert wurden, verwendet.



Abbildung 4.3: Klasse passive-screw

Abbildung 4.2: Klasse screw

Roboflow bietet zusätzlich die Möglichkeit, polygone Objektkästen zu verwenden. Da der Fokus auf dem grundsätzlichen Erkennen von im Bauteil eingelegten Schrauben, und weniger auf der konkreten Form dieser liegt, wird für beide Klassen nicht auf diese Option zurückgegriffen.

Mit dieser Aufteilung in zwei Schraubenklassen soll das Lernen der unterschiedlichen Merkmale beider Klassen in den Trainingsbildern forciert werden, sodass im Verwendungszustand eine robuste Unterscheidung zwischen korrekt befestigten Schrauben, sowie Schrauben, welche un-

befestigt auf der Montagefläche und dem Bauteil vorhanden sind, und damit für das Anfahren des Roboters im nächsten Schritt keine Relevanz haben, ermöglicht wird.

4.3 Generierung von Datensätzen in Roboflow

Aus den aufgenommenen sowie im vorigen Schritt annotierten Bildern werden nun unterschiedliche Trainingsdatensätze generiert, mit welchen im nachfolgenden Trainingsschritt (Kap. 5) YOLOv5 trainiert wird.

Die Generierung in Roboflow besteht aus folgenden Schritten: [ROBOa]

1. Aufteilung in Trainings-, Validierungs- und Testdaten

In diesem Schritt wird die Aufteilung des Datensatzes in Trainingsdaten, welche für das eigentliche Training verwendet werden, Validierungsdaten sowie Testdaten vorgenommen.

Validierungsdaten werden dazu verwendet, im Laufe des Trainings ein Overfitting des trainierenden Netzes zu vermeiden. Unter Overfitting versteht man ein „Überlernen“ von bestimmten Mustern und Beziehungen im Trainingsdatensatz, welche zwar für den Trainingsdatensatz ihre Gültigkeit haben, aber im Generalisierungsschritt, d.h. bei der Anwendung des trainierten Netzes auf einen noch unbekannten Datensatz, zumeist schlechter arbeiten.

Die Einteilung der in Roboflow generierten Daten in Trainings-, Validierungs- und Testdaten erfolgt fix (Hold-Out-Methode) [ROBOa], d.h. eine Unterteilung des gesamten Trainingsdatensatzes in Trainings- und Validierungsdaten, in einzelne kleinere Datensets, wie es zum Beispiel bei k-fold Cross-Validation der Fall ist, erfolgt hier nicht.

2. Preprocessing

In diesem Schritt können verschiedene Preprocessing-Optionen, wie das Verändern der Bildgröße, das Isolieren von Objekten etc., ausgewählt werden, welche auf sämtlichen Bildern vor dem eigentlichen Training angewendet werden.

Mit Preprocessing (dt: Vorverarbeitung) bezeichnet man das Manipulieren des Datensatzes vor der eigentlichen Verwendung im anstehenden Training des Netzes. Dies kann unter anderem deshalb sinnvoll sein, um verwendetet Daten auf eine normierte Bildgröße zu bringen oder bestimmte Daten vom Training auszuschließen.

Roboflow bietet unter anderem die Möglichkeiten diverse Objekte im Bild zu isolieren, das Verändern der Bildgröße sowie weitere Optionen an.

3. Augmentation

Durch Augmentierung (dt. Vermehrung) können auf Basis der erstellten Bilder weitere Bilder generiert werden, um die Anzahl sowie die Varianz der Bilder im finalen Datensatz zu erhöhen. Dabei werden durch Roboflow auf der Basis der bereits bestehenden Bilder künstlich zusätzliche Bilder generiert. Hierfür bietet Roboflow Optionen wie das Variieren der Helligkeit, das Drehen des Bildes um einen bestimmten Winkel, das künstliche Hinzufügen von Rauschen in den Bildern sowie weitere Optionen an. Mit dem Variieren der Helligkeit können verschiedene Tageslichtszenarien abgebildet werden, durch das Drehen der Trainingsbilder können verschiedene Bauteilpositionen modelliert werden. Durch das Hinzufügen von Rauschen können Unsicherheiten im Bezug auf die genaue Schraubenposition dargestellt werden.

4. Generate

Im finalen Schritt besteht die Möglichkeit, die Anzahl der Bilder auszuwählen, welchen man Generieren möchte. Aus diesem Schritt resultiert somit das finale Trainingsdatenset. Bis zu einer Größe von 1197 Bildern ist dieser Schritt in der Basisversion von Roboflow möglich.

Mit welchen Optionen die verwendeten Trainingsdatensets im einzelnen erstellt wurden, wird in Kapitel 5 beschrieben.

5 Training

Als Training wird das Anlernen von YOLOv5 im Rahmen der gegebenen Problemstellung der Schraubenerkennung bezeichnet. Da das Training das Kernelement des späteren Erfolgs des integrierten Modells darstellt, ist eine sorgsame und umfassende Vorgehensweise notwendig. Über diese geben die nachfolgenden Abschnitte einen detaillierten Überblick.

5.1 Trainingsumgebung

Trainiert wurde auf drei verschiedenen Plattformen: CLAIX-2018 (Cluster Aix-la-Chapelle) angeboten von der RWTH Aachen, Google Colab und auf dem lokalen Ubuntu 20.04 Rechner. Die verwendete Umgebung hat keinen Einfluss auf die Ergebnisse der Trainings, allerdings ist durch die sehr viel höhere Rechenleistung auf dem Cluster und auf Google Colab das gleiche Training bedeutend schneller. Da die Trainingszeiten im Bereich von Stunden liegen, ist dies von hoher Relevanz. CLAIX-2018 bietet außerdem den Vorteil, dass für Studierende der RWTH Aachen kostenfreie ein Pensum von 500 Corehours pro Monat zur Verfügung steht. Bei Verwendung der kostenfreien Version von Google Colab werden nur 12 Stunden Zeit pro Training zur Verfügung gestellt, was aber für die Durchführung der Trainings ausreichend war.

5.2 YOLOv5 Modelle

YOLOv5 stellt fünf verschiedene Modelle zur Auswahl, die für einen Trade-Off zwischen der Erkennungszeit und Genauigkeit des trainierten Modells sorgen. In der Tabelle 5.1 sind die unterschiedlichen Laufzeiten und Präzisionen bei einer einheitlichen Bildgröße von 640x640 dargestellt. [ULTRb]

Als bester Kompromiss hat sich das Modell YOLOv5s ergeben, das sowohl sehr schnell auf dem Rechner läuft, als auch die Schrauben relativ zuverlässig erkennt. Außerdem hat sich YOLOv5s Modell durch eine vergleichsweise schnelle Trainingszeit ausgezeichnet.

Tabelle 5.1: YOLOv5 Modelle

Modell	mAP@0.5-0.95	mAP@0.5	Erkennungszeit (in ms)	FLOPs
YOLOv5n	28.0	45.7	45	4.5
YOLOv5s	37.4	56.8	98	16.5
YOLOv5m	45.4	64.1	224	49.0
YOLOv5l	49.0	67.3	430	109.1
YOLOv5x	50.7	68.9	766	205.7

5.3 Vorgehen

Nachdem die Datensätze alle gelabelt und augmentiert wurden, kommt YOLOv5 zum Einsatz. Zunächst wird der Datensatz mithilfe von API Key mit folgendem Codeausschnitt direkt aus Roboflow importiert:

```
1 from roboflow import Roboflow
2 rf = Roboflow(model_format="yolov5", notebook="ultralytics")
3 rf = Roboflow(api_key="YOUR API KEY HERE")
4 project = rf.workspace().project("YOUR PROJECT")
5 dataset = project.version("YOUR VERSION").download("yolov5")
```

Auflistung 5.1: Datensatz importieren

Im Anschluss daran kann das Training mit gewünschten Hyperparametern begonnen werden. Im folgenden Codeausschnitt sieht man den Trainingsbefehl, wobei das Trainingsskript von YOLOv5 bereitgestellt wird.

```
1 !python train.py --img 640 --batch 32 --epochs 100 --data {dataset.location}/data.yaml weights yolov5s.pt
```

Auflistung 5.2: Training

Hier bietet es sich an, die vorgegebenen Gewichte [yolov5s.pt], welche als Initialisierungsgewichte von YOLOv5 bereitgestellt werden, um einen schnellen Trainingserfolg zu ermöglichen.

Nachdem das Training erfolgreich abgeschlossen ist, wird ein .pt-Datei generiert, in welche die Gewichte des Trainings gespeichert sind. Mithilfe von Tensorboard kann der Trainingsverlauf visualisiert werden. Hierfür verwendet man die untenstehenden Command Line Argumente:

```
1 %load_ext tensorboard
2 %tensorboard --logdir runs
```

Auflistung 5.3: Codeausschnitt Tensorboard

Schließlich werden die entsprechenden Modelle abhängig von deren Parametern und Hyperparametern gewertet.

5.4 Trainingsparameter

Die Trainingsparameter wurden je nach der Leistung der Trainingsplattform entschieden. Es wurden zehn Trainings mit verschiedenen Parametern, mit zwei verschiedenen Datensätzen und auf drei verschiedenen Plattformen durchgeführt. Tabelle 5.2 beschreibt alle Bedingungen der Trainings, die durchgeführt wurden:

Tabelle 5.2: Trainingsbedingungen

	Plattform	Datensatz	Epochen	Batchgröße	YOLOv5-Modell
1	Linux (lokal)	1	100	6	YOLOv5s
2	Google Colab	1	150	16	YOLOv5s
3	Google Colab	1 (3*augmentiert)	150	16	YOLOv5x
4	Google Colab	2 (3*augmentiert)	100	16	YOLOv5s
5	Google Colab	2 (2*augmentiert)	150	32	YOLOv5m
6	Claix-2018	1	50	16	YOLOv5s
7	Google Colab	2	171	32	YOLOv5s
8	Claix-2018	2 (3*augmentiert)	98	32	YOLOv5m
9	Claix-2018	2 (3*augmentiert)	50	32	YOLOv5s
10	Claix-2018	2 (3*augmentiert)	50	32	YOLOv5s

5.5 Trainingsergebnisse

Abbildung 5.1 veranschaulicht die visuelle Repräsentation eines Computer-Vision-Inference-Prozesses. Die Tabelle 5.3 beschreibt alle Trainingsergebnisse, die durchgeführt wurden. Die Ergebnisse werden in Kapitel 7 interpretiert und ausgewertet.



(a) Training #2



(b) Training #4

Abbildung 5.1: Erkennung der Schrauben

Tabelle 5.3: Trainingsergebnisse

	Trainingszeit	Erkennungszeit	Durchschnittlicher Confidence Score
1	80 Minuten	70 ms	0.8
2	5 Minuten	50 ms	0.75
3	45 Minuten	400 ms	0.8
4	3 Stunden	70 ms	0.7
5	40 Minuten	140 ms	0.6
6	3 Stunden	40 ms	0.6
7	30 Minuten	70 ms	0.6
8	10 Stunden	140 ms	0.6
9	1 Stunde	70 ms	0.6
10	1 Stunde	70 ms	0.7

5.6 Optimierung der Hyperparameter

Um das trainierte neuronale Netz weiter zu verbessern, wird eine Optimierung der Hyperparameter vorgenommen. YOLOv5 bietet dafür das Argument `--evolve` an, welches eine Evolution der Parameter basierend einem genetischen Algorithmus wie in Kapitel 2.2 beschrieben durchführt. Die zu maximierende Fitness-Metrik wird dabei als Kombination aus den Maßen Precision, Recall, mAP@0.5 und mAP@0.5:0.95 (siehe Kapitel 2.1.1) mit nutzerdefinierten Gewichten gebildet. Die empfohlenen Standardwerte der Gewichte sind [0, 0, 0.1, 0.9] [GITH20], dementsprechend ist der mAP@0.5:0.95 Wert am stärksten ausschlaggebend.

Es wird eine Evolution über 50 Generationen durchgeführt, wobei jede Generation 10 Epochen lang trainiert wird. Anschließend wird mit den Parametern, die während dieser Evolution die beste Fitness vorweisen, ein vollständiges Training über 50 Epochen durchgeführt.

6 Integration

6.1 Zielsetzung

Um das kontinuierliche Ablaufen des Montageprozesses zu gewährleisten, soll die Schraubenerkennung in die Robotersteuerung eingegliedert werden. Die am Gehäuse des Roboters montierten Kameras nehmen regelmäßig Bilder auf, welche anschließend vom neuronalen Netz ausgewertet werden sollen. Dieses sendet die Koordinaten der erkannten Schrauben wiederum an die Robotersteuerung, sodass diese angefahren werden können.

6.2 Implementierung

Bei der Implementierung in Python wird dafür eine neue ROS Node *yolov5node* angelegt, wobei sich an der bisher genutzten Node orientiert wird, welche das von Facebook AI Research veröffentlichte *Detectron2* [WU19] Netz verwendet. Die Node gliedert sich in das System ein, indem sie zu dem Topic subscribed, in welchem die Kamerabilder veröffentlicht werden. Außerdem werden zwei neue Topics erstellt, in denen die ausgewerteten Bilder mit Markierungen der erkannten Schrauben und die Koordinaten veröffentlicht werden. Das neuronale Netz wird bei der Initialisierung der Node geladen. Dabei ist ein Austauschen des Netzes einfach über die Angabe der PyTorch Model Datei (.pt) möglich, in welcher die Gewichte des zu verwendenden Netzes gespeichert sind.

In Auflistung 6.1 ist ein Teil des Python-Codes der zuvor beschriebenen Node gegeben, welcher die Grundstruktur des Ablaufs zeigt. Zunächst wird das Bild, welches in Form des ROS Datentyps *sensor_msgs/Image* vorliegt, in ein numpy-Array konvertiert, das vom yolo-Netz verarbeitet werden kann. Nach der Auswertung des Bildes werden aus der *results*-Struktur die „aktiven“ Schrauben gefiltert und deren Mittelpunkt bestimmt, indem die Koordinaten der Bounding Boxes arithmetisch gemittelt werden. Diese werden anschließend in den Datentypen *std_msgs/Float32MultiArray* konvertiert und an das ROS topic veröffentlicht. Durch die aus der *threading* Bibliothek eingebundene Klasse *Lock* wird gewährleistet, dass während der Auswertung eines Bildes durch die Node kein neues erfasst wird. Die Rate der verarbeiteten Bilder hängt deshalb direkt von der Geschwindigkeit des neuronalen Netzes ab.

Idealerweise wird die neu erstellte Python-Datei mit in die catkin packages eingebunden. Da allerdings momentan der Kompilierprozess mit catkin make nicht funktioniert, wird die bestehende Version des Behaviour Trees gestartet und anschließend separat die *yolov5node* ausgeführt. An der Behebung des Problems wird nicht gearbeitet, da die Software ohnehin für die Kompatibilität mit der neuen Version ROS 2 überarbeitet werden muss.

```
1 def run(self):
2     rate = rospy.Rate(100)
3     while not rospy.is_shutdown():
4         if self._msg_lock.acquire(False):
5             img_msg = self._last_msg
6             self._last_msg = None
7             self._msg_lock.release()
8         else:
9             rate.sleep()
10            continue
11
12     if img_msg is not None:
13
14         img_yolo = self.convert_to_cv_image(img_msg)
15
16         results = self._model(img_yolo)
17         result_msg = self.getResult(results)
18         self._result_pub.publish(result_msg)
```

Auflistung 6.1: Codeausschnitt yolov5node.py

Die Tests und Ergebnisse der Verwendung von YOLOv5 für die Schraubenerkennung sowie der Integration in ROS sind in Kapitel 7 aufgeführt.

7 Evaluation

Zur Bewertung der Performance des neuronales Netzes werden die Erkennungszeit und die Genauigkeit der Schraubenerkennung untersucht. Die Erkennungszeit wird sowohl bei isolierter Anwendung des Netzes als auch im vollständigen System integriert untersucht.

7.1 Erkennungszeit

Bei Verwendung von YOLOv5s beträgt die Erkennungszeit ca. 70 Millisekunden, während YOLOv5m mit 140 Millisekunden doppelt so lange benötigt. Diese Zeit ist insbesondere von der verwendeten Netzwerkarchitektur abhängig, allerdings zeigen sich bei unterschiedlichen Trainings Abweichungen, sodass Training #6 (siehe Tab. 5.2 bei Verwendung der gleichen Architektur eine Erkennungszeit von nur 40 ms aufweist.

Nach dem Einbinden in ROS kann mit YOLOv5s eine kontinuierliche Bilderkennungsrate von ca. 3 Bildern pro Sekunde erreicht werden, mit YOLOv5m sind es ca. 1,3 Bilder pro Sekunde. Diese Werte sind eine extreme Verbesserung zum vorher verwendeten Detectron2 Netzwerk, welches nur ca. 0,1 Bilder pro Sekunde verarbeiten konnte.

7.2 Optimierung der Hyperparameter

Abbildung 7.1 zeigt die Verbesserung des Training durch die Optimierung der Hyperparameter (Kap. 5.6). Dargestellt ist die aus mAP@0.5 und mAP@0.95 zusammengesetzte Fitness-Metrik der jeweils besten Individuen der einzelnen Generationen.

Neben den Schwankungen ist eine generelle Zunahme der Fitness zu erkennen. Außerdem wird der Maximalwert in der letzten Generation erreicht, was diese sowohl als geeignetste auszeichnet, als auch darauf schließen lässt, dass ein Fortsetzen der Evolution über weitere Generationen wahrscheinlich ein noch besseres Ergebnis mit sich führen würde. Da diese Vorgehensweise allerdings hohe Rechenkapazitäten erfordert, werden keine weiteren Hyperparameterevolutionen durchgeführt und stattdessen die ermittelten Parameter aus der letzten Generation für ein vollständiges Training über 50 Epochen verwendet. Tabelle 7.1 zeigt einen Vergleich der Hyperparameter, wie sie standardmäßig in YOLOv5 verwendet werden, mit den neu bestimmten Werten.

Der Vergleich eines Training mit den Standardparametern mit dem neuen Training zeigt den Erfolg der Hyperparameteroptimierung (Abb. 7.2). Es ist eine immense Steigerung der mAP@0.5:0.95 Metrik über das gesamte Training zu verzeichnen, mit einem um den Faktor 2 höheren Endwert nach 50 Epochen.

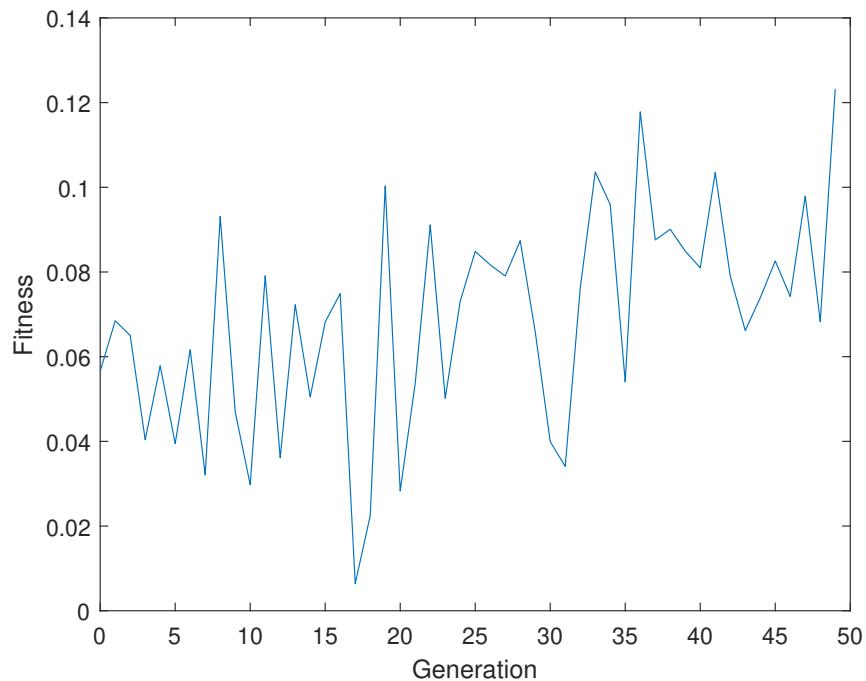


Abbildung 7.1: Fitness der besten Individuen jeder Generation

Tabelle 7.1: Vergleich der Standard- und optimierten Werte der Hyperparameter

Parameter	Standard	Optimiert	Parameter	Standard	Optimiert
lr0	0.01	0.0141	hsv_h	0.015	0.01193
lrf	0.01	0.01	hsv_s	0.7	0.86493
momentum	0.937	0.94026	hsv_v	0.4	0.56457
weight_decay	0.0005	0.00047	degrees	0.0	0.0
warmup_epochs	3.0	2.2493	translate	0.1	0.07606
warmup_momentum	0.8	0.74818	scale	0.5	0.46868
warmup_bias_lr	0.1	0.08071	shear	0.0	0.0
box	0.05	0.0474	perspective	0.0	0.0
cls	0.5	0.57119	flipud	0.0	0.0
cls_pw	1.0	1.2535	fliplr	0.5	0.5
obj	1.0	1.474	mosaic	1.0	0.69556
obj_pw	1.0	0.89154	mixup	0.0	0.0
iou_t	0.2	0.2	copy_paste	0.0	0.0
anchor_t	4.0	4.6263	anchors	-	3.6475
fl_gamma	0.0	0.0			

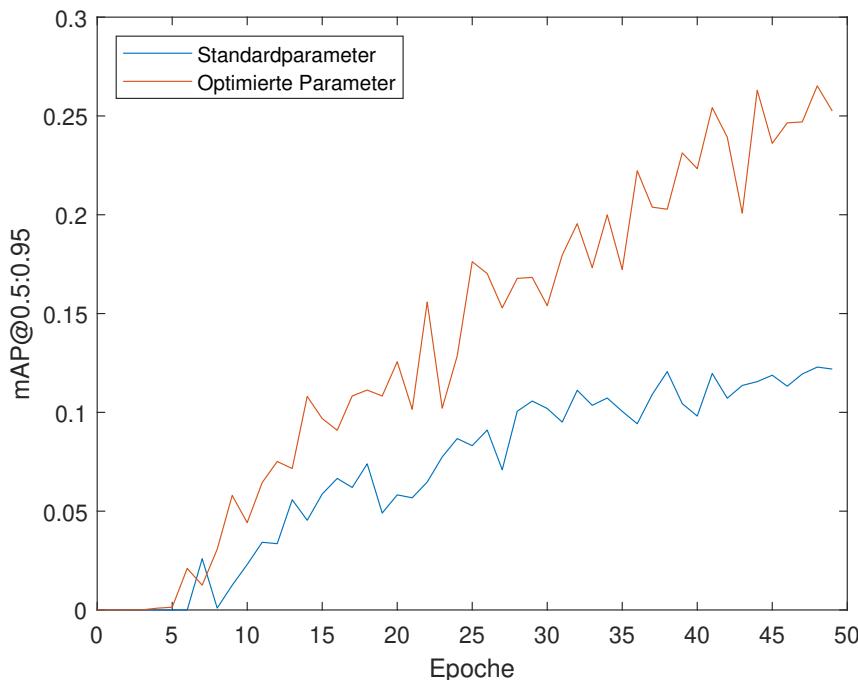


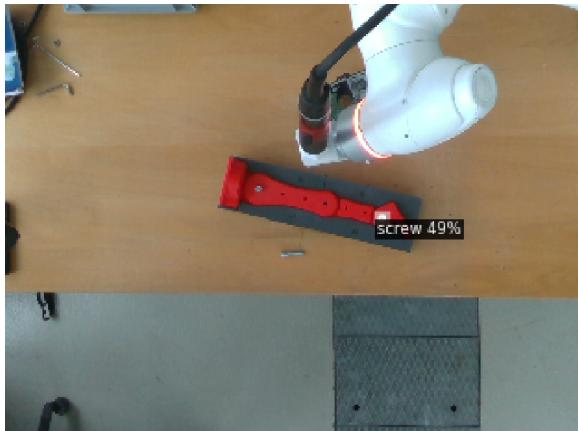
Abbildung 7.2: Vergleich zweier Trainings mit und ohne Hyperparameteroptimierung

7.3 Anwendung im Montageprozess

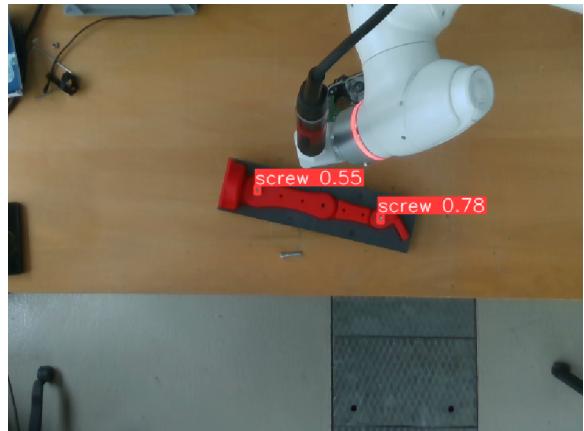
In Abbildung 7.3 ist die Anwendung verschiedener neuronaler Netze auf die Schraubenerkennung dargestellt, wie sie im Montageprozess stattfindet. 7.3a zeigt die Auswertung mit dem vorherigen Detectron2 Netzwerk, wobei zu sehen ist, dass nur eine der beiden in das Bauteil eingelegten Schrauben erkannt wird. Die abgebildete Ziffer beschreibt dabei den Confidence Score. In 7.3b bis 7.3d werden verschiedene trainierte Versionen von YOLOv5s verwendet, welche beide Schrauben korrekt erkennen.

Es ist zu beachten, dass die beim Training verwendeten Metriken wie mAP nicht alleine als Indikator für die Performance unter realen Bedingungen aussagekräftig sind, da diese lediglich auf dem Validierungsteil des Datensatzes ermittelt werden. Ein gutes Erkennen der Schrauben auf den Bildern des Datensatzes lässt nicht zwangsläufig auf ein gutes Erkennen bei der Anwendung im Montageprozess schließen. Bei einer deutlichen Diskrepanz liegt ein Overfitting zum Datensatz vor, wie es beim Detectron2 Netz der Fall ist. Durch die Erweiterung des Datensatzes ist dieses Overfitting in den neuen Netzen reduziert. Diese sind deshalb robuster, sodass sowohl die eingelegten Schrauben konsistent erkannt werden, als auch seltener eine falsche Erkennung einer nicht eingelegten Schrauben auftritt.

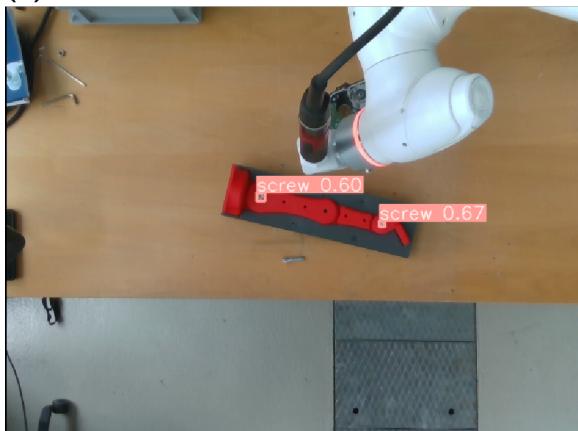
Die Erkennung der passiven Schrauben findet hingegen nicht konsistent statt. Dies kann zum einen an einer Unterrepräsentation dieser Klasse im Datensatz liegen, zum anderen daran, dass



(a) Detectron2



(b) Training #4



(c) Training #9



(d) Training #10

Abbildung 7.3: Schraubenerkennung mit verschiedenen neuronalen Netzen

bei den passiven Schrauben eine sehr viel höhere Varianz in Position, Ausrichtung und Hintergrund vorliegt als bei den „aktiven“ Schrauben, die immer in das gleiche Bauteil eingelegt sind.

8 Ausblick

Ziel der vorliegenden Projektarbeit war es, aufbauend auf der Vorarbeit einer vorangegangenen Projektarbeit [JOEC22], eine Verbesserung der Schraubenerkennung im Hinblick auf Genauigkeit und Geschwindigkeit zu erreichen, indem das dortig verwendete Netz (Detectron2) ausgetauscht und mit dem trainierten Netz YOLOv5 ersetzt wurde. Zusätzlich wurde zur Erreichung dieses Ziels auf einen selbsterstellten erweiterten Trainingsdatensatz und eine Optimierung der Hyperparameter zurückgegriffen. Letztere bietet die Möglichkeit für eine weitere Verbesserung des Netzes durch eine ausführlichere Optimierung über eine größere Anzahl von Generationen.

Nachdem das eigens trainierte YOLOv5 nach dem in Kapitel 6 beschriebenen Vorgehen in die gegebene Montageroboterinfrastruktur integriert wurde, konnte, wie in Kapitel 7 diskutiert, eine Verbesserung der Erkennungsgeschwindigkeit sowie der Genauigkeit der Schraubenerkennung im Vergleich zum vorher verwendeten Modell festgestellt werden.

8.1 Integration in den gesamten Montageprozess

Der durch YOLOv5 umgesetzte Vorgang der Schraubenerkennung stellt nur einen Bestandteil des kompletten Montageprozesse der Schrauben dar.

Der Schraubenerkennung folgt zum einen das Anfahren des Roboters, sowie das Anziehen der Schrauben, durch das am Montageroboter befestigte Schraubwerkzeug. Mit der bereits dargelegten Verbesserung der Geschwindigkeit der Schraubenerkennung selbst durch YOLOv5, ist daher auch von einer Verbesserung der Geschwindigkeit des gesamten Montevorgangs auszugehen. Für diese Verbesserung ist entscheidend, wie schnell die erkannten Schraubenpositionen von ROS weiterverarbeitet werden, und wie schnell diese im Anfahren sowie im Anziehen der Schrauben umgesetzt werden.

8.2 YOLOv8

Wie bereits in Kapitel 3 angesprochen, wurde im Januar 2023 die verbesserte Version YOLOv8 von Ultralytics LLC. vorgestellt. [SOLA23]

Eine Neuerung welche sich in hierbei in YOLOv8 wiederfindet, ist seine Erweiterbarkeit, indem es eine Unterstützung aller vorangegangenen YOLO-Modelle bietet, wodurch ein einfacher Wechsel und im Zuge dessen ein Vergleich der Performance zwischen verschiedenen YOLO-Modellen ermöglicht wird. [ULTRa] Durch diesen Flexibilitätsgewinn lässt sich YOLOv8 noch besser auf gegebene Problemstellungen anpassen.

Dadurch, dass YOLOv8 einen zu YOLOv5 verwandten technischen Aufbau besitzt, wird sich eine Integration in die gegebene Systeminfrastruktur des Montageroboters mutmaßlich als un-

kompliziert erweisen. Aufgrund der verbesserte Leistungsfähigkeit von YOLOv8 ist ebenfalls eine erhöhte Geschwindigkeit sowie Genauigkeit der Schraubenerkennung denkbar.

Die Untersuchung dieser Fragestellung bietet sich für weitergehende wissenschaftliche Arbeiten an.

Literaturverzeichnis

- [ANWA22] Anwar, A.: What is Average Precision in Object Detection & Localization Algorithms and how to calculate it? In: Towards Data Science, 13.05.2022
- [BAEL22] Baeldung: Intersection over Union for Object Detection. In: Baeldung on Computer Science, 20.04.2022
- [BISH06] Bishop, C. M.: Pattern recognition and machine learning. (Reihe: Computer science). New York, NY: Springer, 2006. ISBN: 978-0387-31073-2. URL: <http://www.loc.gov/catdir/enhancements/fy0818/2006922522-d.html>
- [CLAE15] Claesen, M.; Moor, B. D.: Hyperparameter Search in Machine Learning, 2015. Firmenschrift
- [DAS22] Das, T.: Google Colab: Everything you Need to Know. In: Geekflare, 16.08.2022
- [FUKU80] Fukushima, K.: Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. In: Biological cybernetics. 36. Jg., 1980, Nr. 4, S. 193–202
- [GITH20] GitHub: Hyperparameter Evolution · Issue #607 · ultralytics/yolov5. 2020. URL: <https://github.com/ultralytics/yolov5/issues/607> [Stand: 19.01.2023]
- [GOLD89] Goldberg, D. E.: Genetic Algorithms in Search, Optimization and Machine Learning. 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675
- [HAST22] hasty.ai: mean Average Precision (mAP) | Hasty.ai. 2022. URL: <https://hasty.ai/docs/mp-wiki/metrics/map-mean-average-precision> [Stand: 30.03.2023]
- [HAZE22] Hazelcast: Machine Learning Inference. 2022. URL: <https://hazelcast.com/glossary/machine-learning-inference/> [Stand: 30.03.2023]
- [HE15] He, K.; Zhang, X.; Ren, S.; Sun, J.: Deep Residual Learning for Image Recognition, 2015
- [JOEC22] Joecks, P.; Mitri, A.: Mensch-Roboter-Kollaboration in der Montage: Computer Vision, 2022. Projektarbeit am Werkzeugmaschinenlabor WZL der RWTH Aachen
- [KATS22] Katsamanis, I.; Karolou, E.; Davradou, A.; Protopapadakis, E.; Doulamis, A.; Doulamis, N.; Kalogerias, D.: TraCon: A novel dataset for real-time traffic cones detection using deep learning, 2022. Firmenschrift
- [KHAN20] Khanna, C.: Number of parameters in a feed-forward neural network, 2020. Firmenschrift
- [KITW] Kitware: CMake. URL: <https://cmake.org/> [Stand: 28.03.2023]
- [LIU18] Liu, S.; Qi, L.; Qin, H.; Shi, J.; Jia, J.: Path Aggregation Network for Instance Segmentation, 2018
- [LONG14] Long, J.; Shelhamer, E.; Darrell, T.: Fully Convolutional Networks for Semantic Segmentation, 2014

- [MALL22] Mallick, S.: Intersection over Union (IoU) in Object Detection & Segmentation. In: LearnOpenCV, 28.06.2022
- [MENE18] Menegaz, M.: Understanding YOLO. 16.03.2018. URL: <https://hackernoon.com/understanding-yolo-f5a74bbc7967> [Stand: 28.03.2023]
- [MHOA19] Mhoassin: Easy way to understand Convolutional Neural Network: It's Easy!!! 19.04.2019. URL: <https://medium.com/@moazzemhossain/easy-way-to-understand-convolutional-neural-network-its-easy-d9b7c0c5adb3> [Stand: 30.03.2023]
- [OPENa] Open Robotics: ROS/Introduction - ROS Wiki. URL: <http://wiki.ros.org/ROS/Introduction> [Stand: 17.02.2023]
- [OPENb] Open Robotics: ROS/Tutorials - ROS Wiki. URL: <http://wiki.ros.org/ROS/Tutorials> [Stand: 17.02.2023]
- [PADR95] Padró, F.; Ozón, J.; Aplicada, D.: Graph Coloring Algorithms for Assignment Problems in Radio Networks, 1995
- [QUIG09] Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software. 3. Jg., 01/2009
- [RASC19] Rasche, C.: Computer Vision. 10/2019
- [ROBOa] Roboflow: Give your software the power to see objects in images and video. URL: <https://roboflow.com/> [Stand: 28.03.2023]
- [ROBOb] Robotics, O.: catkin/conceptual_overview - ROS Wiki. URL: http://wiki.ros.org/catkin/conceptual_overview [Stand: 28.03.2023]
- [RUME86] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.: Learning representations by back-propagating errors. In: Nature. 323. Jg., 10/1986, Nr. 6088, S. 533–536
- [SCIK] scikit-learn: 3.2. Tuning the hyper-parameters of an estimator. URL: https://scikit-learn.org/stable/modules/grid_search.html [Stand: 26.01.2023]
- [SKAL19] Skalski, P.: Gentle Dive into Math Behind Convolutional Neural Networks. In: Towards Data Science, 12.04.2019
- [SOLA22] Solawetz, J.: Yolov4 - an explanation of how it works. 11/2022. URL: <https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/>
- [SOLA23] Solawetz, J.: What is Yolov8? the ultimate guide. 01/2023. URL: <https://blog.roboflow.com/whats-new-in-yolov8/>
- [THAT20] Thatte, A. V.: Evolution of YOLO — YOLO version 1 - Towards Data Science. 21.05.2020. URL: <https://towardsdatascience.com/evolution-of-yolo-yolo-version-1-afb8af302bd2> [Stand: 30.03.2023]
- [THUA21] Thuan, D.: Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detection algorithm, 2021

- [TONY21] Tonye, G.: Machine Learning Confidence Scores — All You Need to Know as a Conversation Designer. In: Voice Tech Global, 25.08.2021
- [TOP5] TOP500: November 2022. URL: <https://www.top500.org/lists/top500/2022/11/> [Stand: 28.03.2023]
- [TUTO23] tutorialspoint: Genetic Algorithms Tutorial. 2023. URL: https://www.tutorialspoint.com/genetic_algorithms/ [Stand: 06.03.2023]
- [ULTRa] Ultralytics: Ultralytics/ultralytics: New - yolov8 in PyTorch > ONNX > CoreML > TFLite. URL: <https://github.com/ultralytics/ultralytics>
- [ULTRb] Ultralytics: Ultralytics/yolov5: Yolov5 in PyTorch > ONNX > CoreML > TFLite. URL: <https://github.com/ultralytics/yolov5>
- [VIOL01] Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features, 2001, S. I
- [WANG19] Wang, C.-Y.; Liao, H.-Y. M.; Yeh, I.-H.; Wu, Y.-H.; Chen, P.-Y.; Hsieh, J.-W.: CSPNet: A New Backbone that can Enhance Learning Capability of CNN, 2019
- [WU19] Wu, Y.; Kirillov, A.; Massa, F.; Lo, W.-Y.; Girshick, R.: Detectron2, 2019