Entwicklung und Evaluation von KI-Modellen auf Basis synthetischer Daten aus digitalen Modellen zur Fehlererkennung bei Werkzeugmaschinen

1. Projektarbeit

vorgelegt am 01. September 2025

Fakultät Wirtschaft und Gesundheit

Studiengang Wirtschaftsinformatik

Kurs WWI2024F

von

Felix Wolfram

Betreuung in der Ausbildungsstätte:

TRUMPF SE + Co. KG
Tom Körner

DHBW Stuttgart:

Prof. Dr. Kai Holzweißig

Sperrvermerk

Die vorliegende Arbeit enthält firmeninterne Informationen und vertrauliche Daten der TRUMPF Gesellschaft. Sie darf aus diesem Grunde nur zu Prüfzwecken verwendet werden. Veröffentlichung oder Vervielfältigungen der Arbeit, auch nur auszugsweise, sind ohne ausdrückliche Genehmigung der TRUMPF Gesellschaft nicht gestattet. Es dürfen keinerlei Kopien oder Abschriften – auch in digitaler Form – gefertigt werden. Die Arbeit ist nur Korrektoren sowie Mitgliedern des Prüfungsausschusses zugänglich zu machen, die ihrerseits zur Geheimhaltung verpflichtet sind. Im Rahmen des Notenfindungsprozesses kann die Arbeit weiteren Personen zugänglich gemacht werden.

Stuttgart, 01.09.2025

Felix Wolfram

Inhaltsverzeichnis

Abkürzungsverzeichnis

CNN Convolutional Neural Network
AP Average Precision
CAD Computer-Aided Design
CV Computer Vision
DL Deep Learning
DSR Design Science Research
FDD Fault Detection and Diagnostics
FN False Negative
FP False Positive
GAN Generative Adversial Network
HVT Hybrid Vision Transformer
IoU Intersection over Union
KI Künstliche Intelligenz
mAP Mean Average Precision
NMS Non-Maximum Suppression
RT-DETR Real-Time Detection Transformer
TP True Positive
USD Universal Scene Description
ViT Vision Transformer
YOLO You Only Look Once

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung[1]

1.1 Problemstellung

Ein zentrales Problem bei der Automatisierung von industriellen Produktionsprozessen stellt die Fehlerekennung und -klassifikation dar.[2] Insbesondere bei geringen Losgrößen sind Fehler nicht vorhersehbar, weshalb Systeme mit regelbasierten Ansätzen an ihre Grenzen stoßen. Es werden daher KI-Modelle benötigt, um mit dieser Komplexität umgehen zu können. Die schnelle und präzise Erkennung und Klassifikation von Fehlern wird daher insbesondere bei der Fernüberwachung von Maschinen und Anlagen (*Remote Monitoring*) zunehmend relevanter. [3]

Eine zentrale Herausforderung stellt dabei die Verfügbarkeit großer und qualitativ hochwertiger Datensätze für das Training und die Evaluierung von KI-Modellen dar. Insbesondere der Mangel an Daten für seltene Fehlerfälle stellt in der Praxis oft eine erhebliche Hürde dar.[4] In den letzten Jahren hat sich die Nutzung von Simulationsumgebungen zur Generierung synthetischer Daten als ein vielversprechender Ansatz herausgestellt, um dieses Problem zu adressieren. [5]

Es können hierdurch große und vielfältige Datensätze generiert werden, was eine große Chance für die Entwicklung leistungsfähiger KI-Modelle darstellt.[6] Durch die Aktualität des Themas ist dieses Feld gleichzeitig nur unzureichend erforscht, insbesondere hinsichtlich der Einsatzmöglichkeiten bei der Fernüberwachung.

Vor diesem Hintergrund ergibt sich die zentrale Forschungsfrage dieser Arbeit: Wie können digitale Modelle zur Generierung synthetischer Daten für das Training von KI-Modellen zur Fehlererkennung an Werkzeugmaschinen genutzt werden und wie geeignet sind diese Daten für den Einsatz in realen Anwendungsfällen?

1.2 Motivation

Die Motivation für diese Arbeit ergibt sich aus der zunehmenden Bedeutung von KI-Modellen in industriellen Anwendungen, insbesondere im Bereich der Fehlererkennung. Die Fähigkeit, Fehler frühzeitig zu erkennen und zu klassifizieren, ist in diversen industriellen Anwendungsfällen von großer Bedeutung. Aufgrund hierbei auftretender erheblicher Herausforderungen, insbesondere hinsichtlich der Verfügbarkeit von qualitativ hochwertigen Datensätzen, ist die Erforschung generativer Ansätze zur Datengenerierung von hoher Relevanz.

[1] Sprachlich geglättet durch ChatGPT-5
[2] Vgl. Wu, Triebe, Sutherland 2023, S. 439
[3] Vgl. Leite et al. 2024, S. 1 f.
[4] Vgl. Urgo, Terkaj, Simonetti 2024, S. 250
[5] Vgl. Schmedemann et al. 2022, S. 1101 f.
[6] Vgl. Monnet, Petrovic, Herfs 2024, S. 768

1.3 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung und Evaluation eines pipelinebasierten Ansatzes zur Generierung synthetischer Bilddaten aus einem digitalen Modell einer Werkzeugmaschine. Diese Daten sollen anschließend genutzt werden, um KI-Modelle zu trainieren und zu evaluieren, welche in der Lage sein sollen, Fehler an Werkzeugmaschinen zu erkennen. Dabei soll die Leistung dieser Modelle auf synthetischen und realen Bilddaten untersucht und verglichen werden, um die Eignung synthetischer Daten in realen Anwendungsfällen zu bewerten.

Es handelt sich dabei um eine prototypische Umsetzung (*Proof-of-Concept*), die die Machbarkeit und das Potenzial synthetischer Daten für solche Anwendungsfälle aufzeigen soll, ohne eine umfassende, bereits praxistaugliche Lösung zu liefern. Der Schwerpunkt liegt auf der Entwicklung einer Datenpipeline, die eine Generierung umfangreicher und qualitativ hochwertiger Bilddatensätze ermöglicht, wobei Reproduzierbarkeit und Automatisierung eine zentrale Anforderung darstellen.

1.4 Forschungsmethodik

Um die Zielsetzung dieser Arbeit zu erreichen, wird im ersten Schritt eine umfassende Literaturrecherche durchgeführt, um den aktuellen Stand der Forschung in diesem Bereich zu erfassen. Anschließend soll ein Artefakt in Form einer Datenpipeline entwickelt werden, um die in Kapitel 1.1 gestellte Forschungsfrage zu adressieren. Die erzielten Ergebnisse werden anschließend evaluiert und kritisch reflektiert.

Methodisch orientiert sich die Vorgehensweise in dieser Arbeit am Design Science Research (DSR)-Paradigma nach Hevner et al. 2004. Es wird ein iterativer Prozess verfolgt, der die Phasen der Problemanalyse, der Entwurfsentwicklung und der Evaluation umfasst.[7]

1.5 Aufbau der Arbeit

Die Arbeit ist in mehrere Kapitel gegliedert, welche den Leser im Sinne des DSR-Paradigmas von der Problemstellung bis hin zu den Ergebnissen und deren Schlussfolgerungen führen. Nach der Einleitung in Kapitel 1 folgt in Kapitel 2 die theoretische Fundierung, in welcher die relevanten Konzepte und Technologien erläutert werden. Kapitel 3 beschreibt die Methodik, die zur Entwicklung des Artefakts verwendet wird, während anschließend in Kapitel 4 das entwickelte Artefakt detailliert beschrieben wird. Die Evaluation des Artefakts wird in Kapitel 5 präsentiert, gefolgt von einer Diskussion dieser Ergebnisse in Kapitel 6. Hier werden außerdem bestehende Herausforderungen und Limitationen dieser Arbeit sowie der Bedarf weiterer Forschung in diesem Feld diskutiert.

[7] Vgl. Kapitel 3

2 Diskussion des aktuellen Stands der Forschung und Praxis[8]

2.1 Digitale Modelle in der Fertigungsindustrie

Die Entwicklung digitaler Modelle von Maschinen und Anlagen ist ein zentrales Element der Industrie 4.0. Durch die bessere Vernetzung von Systemen sowie Fortschritte in der Datenanalyse, insbesondere durch KI, steigt auch die Forschung und Nutzung an digitaler Abbildungen.[9] Im Folgenden soll näher auf den Stand der Forschung in diesem Bereich sowie Entwicklungstrends und aktuelle Herausforderungen eingegangen werden.

2.1.1 Begriffsabgrenzung: Digitales Modell, Digitaler Schatten, Digitaler Zwilling

Obwohl in der Literatur häufig pauschal von Digitalen Zwillingen die Rede ist, lässt sich eine Unterscheidung in Digitales Modell, Digitaler Schatten und Digitaler Zwilling vornehmen. Diese Begriffe werden fälschlicherweise oftmals synonym verwendet, obwohl sie einige signifikante Unterscheidungen in ihrer Systemarchitektur aufweisen.[10] Zwar existieren keine einheitlichen Definitionen,[11] jedoch unterscheidet eine gängige Klassifizierung die drei Formen der digitalen Abbildungen basierend auf dem Grad der Datenintegration zwischen dem physischen und dem digitalen System.[12]

Digitales Modell

Bei einem digitalen Modell handelt es sich um eine digitale Repräsentation eines existierenden oder geplanten Objekts. Es findet kein automatisierter Austausch von Daten statt, der Datenfluss erfolgt also in beide Richtungen, vom physischen zum digitalen Objekt und umgekehrt, ausschließlich manuell. Das Modell kann durchaus auf Basis realer Daten erstellt werden oder diese nutzen, jedoch passiert dieser Austausch nicht automatisch, wodurch die beiden Systeme keinerlei Einfluss aufeinander haben.[13]

Digitaler Schatten

Von einem digitalen Schatten ist häufig die Rede, wenn der Datenfluss einseitig automatisiert ist. Daten vom physischen Objekt fließen automatisch in die digitale Repräsentation ein, während umgekehrt der Fluss vom digitalen in das physische Objekt weiterhin ausschließlich manuell

[8] Sprachlich geglättet durch ChatGPT-5
[9] Vgl. Fuller et al. 2020, S. 108952
[10] Vgl. Kritzinger et al. 2018, S. 1017
[11] Vgl. Liu et al. 2021, S. 350; Fuller et al. 2020, S. 108953
[12] Vgl. Fuller et al. 2020, S. 108953; Kritzinger et al. 2018, S. 1017 f.
[13] Vgl. Fuller et al. 2020, S. 108953

stattfindet. Dies führt dazu, dass eine Veränderung des physischen Objekts einen direkten Einfluss auf das digitale Objekt hat, allerdings nicht umgekehrt.[14]

Digitaler Zwilling

Der Digitale Zwilling ist die Architektur mit dem höchsten Grad der Datenintegration zwischen physischem und digitalem Objekt. Hierbei findet der Datenfluss in beide Richtungen automatisiert statt, beide Systeme sind also miteinander verbunden und haben beidseitig einen direkten Einfluss aufeinander. Alle Informationen, welche von dem realen System gesammelt werden können, sollen auch in die digitale Repräsentation einfließen, wodurch diese das physische System spiegeln soll. Änderungen des physischen Objekts führen also direkt zu Änderungen des digitalen Objekts und umgekehrt.[15]

Oftmals wird fälschlicherweise pauschal von einem Digitalen Zwilling gesprochen, wobei es sich in vielen Fällen nicht um einen tatsächlichen Digitalen Zwilling mit einem bidirektionalen Fluss an Daten handelt.[16]

2.1.2 Forschungsstand und Entwicklungstrends

Frühere Arbeiten konzentrierten sich überwiegend auf die Forschung und die Entwicklung von Konzepten und weniger auf konkrete Einsatzmöglichkeiten in der Fertigung.[17] Jedoch lässt sich hier ein klarer Wandel erkennen, welcher immer mehr Forschung zum empirischen Einsatz von Digitalen Zwillingen, Modellen und Schatten aufzeigt, während auch die Forschung in der Fertigungsindustrie zunimmt.[18]

2.1.3 Rolle der KI und einhergehende Herausforderungen

KI-Anwendungen in Verbindung mit digitalen Zwillingen wurden zunächst selten erforscht und eingesetzt[19], jedoch wird dieser Trend zunehmend populärer.[20] KI stand anfangs noch weniger umfangreich am Rande des Einsatzes,[21]gewinnt jedoch mittlerweile in unterschiedlichen Einsatzfeldern, insbesondere bei der Fehlererkennung und -klassifizierung sowie vorausschauender Wartung (engl. *Predictive Maintenance*), immer mehr von Bedeutung. Damit werden auch Probleme, welche sich beim Training von KI-Modellen, wie z.B. zu wenige oder einseitige Daten, immer relevanter.[22] Insbesondere der Einsatz von Digitalen Modellen zur Generierung solcher

[14]Vgl. Kritzinger et al. 2018, S. 1017
[15]Vgl. Kritzinger et al. 2018, S. 1017; Fuller et al. 2020, S. 108963
[16]Vgl. Kritzinger et al. 2018, S. 1020
[17]Vgl. Kritzinger et al. 2018, S. 1018 ff.
[18]Vgl. Liu et al. 2021, S. 349
[19]Vgl. Liu et al. 2021, S. 352 ff.
[20]Vgl. Mikołajewska et al. 2025, S. 1
[21]Vgl. Liu et al. 2021, S. 352 ff.
[22]Vgl. Mikołajewska et al. 2025, S. 1

Trainingsdaten ist ein hochaktuelles Forschungsthema, welches sich noch in einem frühen Stadium befindet und in der aktuellen Forschung bislang unterrepräsentiert ist. Hier besteht klar ein weiterer Forschungsbedarf.

2.1.4 Nutzen und Anwendungsfelder in der Fertigung

Der Einsatz digitaler Abbildungen in der Fertigung bietet vielfältige Vorteile. Durch diese Modelle ergeben sich Möglichkeiten wie die Echtzeit-Überwachung und der damit einhergehenden schnellen Reaktionsfähigkeit auf Anomalien, der Engpasskontrolle oder der Produktionsplanung und -steuerung.[23] Außerdem ist es möglich, Simulation von Maschinen oder Anlagen vor ihrem tatsächlichen physischen Einsatz durchzuführen.[24] Sie ermöglichen des Weiteren die risikofreie Durchführung gefährlicher oder kostenintensiver Prozesse.[25] Digitale Repräsentationen leisten somit einen wesentlichen Beitrag zur Erhöhung von Sicherheit, Qualität, Flexibilität und Effizienz in der industriellen Fertigung.

2.2 Synthetische Bilddaten für das Training von KI-Modellen

Durch das Wachstum von Deep Learning (DL) und Computer Vision (CV) steigt auch der Bedarf an großen Mengen von annotierten und qualitativ hochwertigen Datensätzen.[26] Die Beschaffung solcher Datensätze bringt jedoch insbesondere im industriellen Umfeld eine Vielzahl von Herausforderungen mit sich. Das Erstellen umfangreich und diverser Datensätze ist oftmals ressourcen- und zeitaufwendig sowie kostspielig und zudem menschlichem Fehler unterlegen.[27] Primär im industriellen Umfeld fehlt es außerdem häufig an ausreichend Bildern mit tatsächlichen Fehlerfällen, da diese in der Realität vergleichsweise selten auftreten.[28] Im Folgenden werden daher mögliche Lösungsansätze für diese Problematik durch die Nutzung synthetischer Daten vorgestellt und diskutiert.

2.2.1 Potenziale synthetischer Daten

Um die herausgestellten Probleme zu adressieren, rückt das Konzept der Generierung synthetischer Daten immer mehr in den Fokus der Forschung. Das Ziel ist es hierbei, durch unterschiedliche Ansätze schnell und kostengünstig möglichst realitätsnahe und umfangreiche Datensätze in praktisch unbegrenztem Umfang zu generieren, welche anschließend für das Training von KI Modellen verwendet werden können. Auch eine präzise, pixelgenaue Annotation wird hierdurch ermöglicht.[29] Ein weiterer Vorteil ist die Möglichkeit, schon vor dem Start der Produktion Bild-

[23] Vgl. Liu et al. 2021, S. 352 ff. Alfaro-Viquez et al. 2025, S. 23
[24] Vgl. Alfaro-Viquez et al. 2025, S. 7
[25] Vgl. Zaripov, Kulshin, Sidorov 2025, S. 6
[26] Vgl. Monnet, Petrovic, Herfs 2024, S. 767
[27] Vgl. Urgo, Terkaj, Simonetti 2024, S. 250
[28] Vgl. Monnet, Petrovic, Herfs 2024, S. 768
[29] Vgl. Monnet, Petrovic, Herfs 2024, S. 768; Urgo, Terkaj, Simonetti 2024, S. 250

daten generieren und Modelle für den konkreten Anwendungsfall trainieren zu können.[30]

2.2.2 Herausforderungen beim Einsatz synthetischer Daten

Es ergeben sich bei der Nutzung synthetischer Daten jedoch dennoch diverse Probleme, von denen insbesondere die Herausforderung des *Domain Gaps*, im Mittelpunkt der Forschung steht. Bei dem *Domain Gap* handelt es sich um das Problem, dass Modelle, welche auf synthetischen Daten trainiert wurden, Schwierigkeiten bei der Übertragbarkeit auf reale Anwendungsfälle aufzeigen.[31] Ein Grund hierfür kann ein unzureichender Realismus der synthetischen Daten sein, welcher die Komplexität der realen Welt nicht ausreichend widerspiegelt, was durch die Arbeit von Boikov et al. 2021 zu sehen ist. In ihrer Studie wurden Modelle auf die Erkennung von Stahlfehlern trainiert. Diese erzielten auf den synthetischen Daten gute Ergebnisse, die Genauigkeit nahm bei Tests auf realen Daten jedoch merklich ab.[32] Es ist dabei allerdings zu berücksichtigen, dass dieses Problem maßgeblich vom Anwendungsfall und der für diesen erforderlichen Detaillierungsgrad abhängt. In Fällen, wie beispielsweise der Erkennung von Objektrotationen, zeigen Manettas, Nikolakis, Alexopoulos 2021, dass auch mit weniger realistischen synthetischen Daten ein erfolgreiches Modell trainiert werden kann.[33] Es kann also festgestellt werden, dass es in diesem Bereich weiterer Forschung bedarf.[34]

Ein weiteres Problem bei der Nutzung synthetischer Daten stellt außerdem das sogenannte *Over-Labelling* dar. Hierbei werden in den Trainingsbildern Merkmale als Fehler markiert, welche durch einen menschlichen Inspektor gar nicht erkannt werden könnten, beispielsweise aufgrund des Blickwinkels oder einer geringer Sichtbarkeit des Fehlers. Dies führt dazu, dass das Modell später auch unkritische Merkmale als Fehler klassifiziert und somit die Anzahl an False Positive (FP) erhöht wird.[35]

2.2.3 Methoden zur Generierung synthetischer Daten

Um solche synthetischen Daten zu generieren, haben sich verschiedene Ansätze entwickelt, welche sich in verschiedene Kategorien einteilen lassen, auf welche im Folgenden näher eingegangen wird:[36]

Generative Modelle

Zu einem populären Ansatz zur Erzeugung synthetischer Daten mit generativen Modellen gehören Generative Adversial Networks (GANs). GANs bestehen aus einem Generator und einem Diskriminator, wobei der Generator synthetische Daten erzeugt und der Diskriminator versucht,

[30] Vgl. Monnet, Petrovic, Herfs 2024, S. 768
[31] Vgl. Monnet, Petrovic, Herfs 2024, S. 768
[32] Vgl. Boikov et al. 2021, S. 8
[33] Vgl. Manettas, Nikolakis, Alexopoulos 2021, S. 241 f.
[34] Vgl. Monnet, Petrovic, Herfs 2024, S. 772
[35] Vgl. Fulir et al. 2023, S. 4431
[36] Vgl. Fulir et al. 2023, S. 4426 f.

zwischen echten und generierten Daten zu unterscheiden. Beide Netzwerke lernen dabei gegenseitig voneinander und werden gemeinsam optimiert, wodurch der Generator zunehmend realistischere Daten erzeugt.[37] Allerdings ist das Problem hierbei, dass für das effektive Training von solchen GAN-Modellen immer noch eine signifikante Menge an echten Trainingsdaten erforderlich ist.[38]

Computergrafik (Rendering-basierte Methode)

Bei rendering-basierten Methoden wird eine Simulationsumgebung entwickelt, welche die realen Abläufe und Zustände widerspiegelt. Der Vorteil ist hierbei, dass einfach verschiedene Parameter, wie beispielsweise Lichtverhältnisse oder Kamerawinkel angepasst werden können, um das Modell robuster zu machen.[39] Dies hat den Vorteil, dass auf Änderungen der Umwelt oder des Fertigungsprozesses schnell reagiert und kostengünstig große Datensätze produziert werden können. Der größte Nachteil stellt der Modellierungsaufwand dar, welcher unternommen werden muss, um die Simulationsumgebung zu erstellen.[40]

Transfer Learning

Bei der Methode des Transferlernens (engl. *Transfer Learning*) werden bereits vorhandene Modelle für die Objekterkennung verwendet, welche auf realen Daten trainiert wurden. Die Schichten, welche für das Extrahieren von Merkmalen zuständig sind, bleiben unberührt und eingefroren, während nur die letzten Schichten neu trainiert werden.[41] Die Menge an benötigten Daten wird dadurch verringert, während gleichzeitig bessere Ergebnisse als bei einem Training auf rein synthetischen Daten erzielt werden können.[42] Das Problem der Übertragbarkeit auf reale Anwendungsfälle stellt allerdings dennoch eine Herausforderung dar.[43]

2.3 KI-Modelle in der Bildverarbeitung

Fortschritte in DL hat die Bildverarbeitung maßgeblich vorangetrieben, insbesondere durch die Möglichkeit, Merkmale der Datensätze effizient und automatisiert erfassen zu können.[44] Im Folgenden werden daher verschiedene relevante DL-Architekturen zur Bildverarbeitung vorgestellt und diese verglichen.

[37] Vgl. Jain et al. 2022, S. 1010
[38] Vgl. Monnet, Petrovic, Herfs 2024, S. 768
[39] Vgl. Manettas, Nikolakis, Alexopoulos 2021, S. 239
[40] Vgl. Monnet, Petrovic, Herfs 2024, S. 769
[41] Vgl. Urgo, Terkaj, Simonetti 2024, S. 250
[42] Vgl. Urgo, Terkaj, Simonetti 2024, S. 251
[43] Vgl. Urgo, Terkaj, Simonetti 2024, S. 251
[44] Vgl. Chai et al. 2021, S. 1 f.

2.3.1 Convolutional Neural Networks (CNNs)

Insbesondere die Architektur der CNNs, hat große Fortschritte bei der Bildverarbeitung erzielt und wird vielseitig für einen Großteil von CV-Anwendungen in verschiedensten Aufgabenbereichen, wie Bildklassifikation, Bildsegmentierung, Bilderkennung und Bildwiedererkennung erfolgreich eingesetzt.[45]

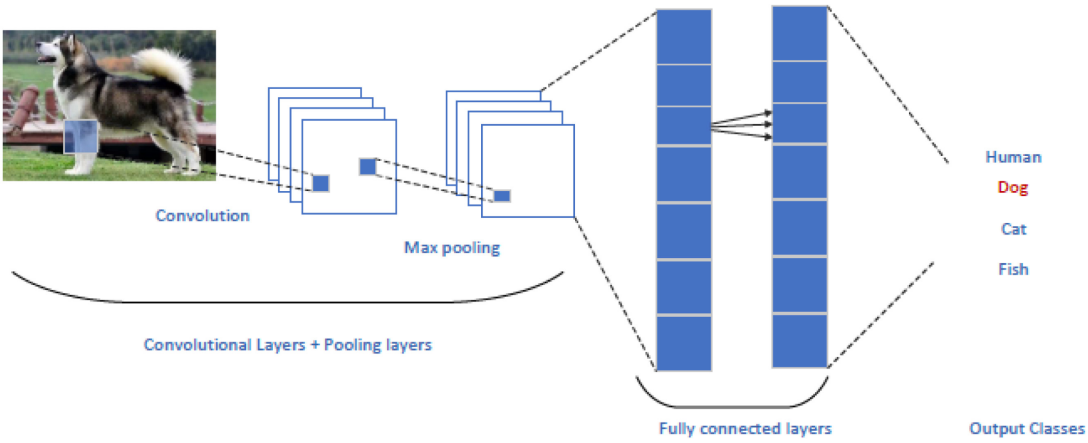


Abb. 1: Schematischer Aufbau einer Convolutional Neural Network Architektur[46]

Wie in Abbildung 1 dargestellt sind CNNs hierarchisch aufgebaut, mit mehreren *Convolution Layers*, welche sogenannte *Feature Maps* generieren. Diese enthalten extrahierte Merkmale des Eingabebildes, wie beispielsweise Kanten, Formen oder Texturen.[47] Hierbei steigt die Abstraktion in zunehmenden Schichten. In früheren Schichten erkennt das Netzwerk einfachere Muster, in tieferen Schichten komplexere Muster und Zusammenhänge. Dies ermöglicht es CNNs, automatisiert von Datensätzen zu lernen und Merkmale aus den Bildern zu extrahieren, wodurch diese nicht mehr von Hand vorgegeben werden müssen.[48] Außerdem existieren bei CNNs *Pooling Schichten*, um die Dimensionen der *Feature Maps* zu reduzieren. Am Ende befinden sich *Fully Connected Layers*, welche schlussendlich eine Klassifikation des Bildes zur Folge haben.[49]

Durch diese Eigenschaften sind CNNs äußerst populär für Applikationen in der Bildverarbeitung in verschiedensten Bereichen und führten durch ihre hohe Leistung im Vergleich zu anderen Ansätzen zu Durchbrüchen in diversen Feldern.[50]

Die Eignung von CNNs für industrielle Anwendungen wird in der aktuellen Literatur differenziert bewertet. Jing et al. 2017 zeigen, dass CNNs bei der Fehlererkennung und Klassifikation von Getriebestörungen eine hohe Leistungsfähigkeit aufweisen können.[51] Allerdings wird in der Literatur betont, dass generisch trainierte Modelle in industriellen Szenarien häufig an ihre Grenzen

[45] Vgl. Bhatt et al. 2021, S. 2
[46] Entnommen aus: Singh, Sabrol 2021, S. 2
[47] Vgl. Khanam et al. 2024, S. 94251
[48] Vgl. Khanam et al. 2024, S. 94251
[49] Vgl. Chai et al. 2021, S. 2
[50] Vgl. Singh, Sabrol 2021, S. 4762 ff.
[51] Vgl. Jing et al. 2017

stoßen. So weisen Khanam et al. 2024 darauf hin, dass CNNs nur dann effektiv einsetzbar sind, wenn sie gezielt auf konkrete Anwendungsfälle zugeschnitten werden, da die Modelle ansonsten häufig ineffektiv sind.[52]

2.3.2 Vision Transformers (ViTs)

Aktuelle Entwicklungen zeigen, dass Vision Transformers (ViTs) zunehmend an Bedeutung gewinnen und eine Alternative zu CNN-Architekturen darstellen, da sie einige Limitationen überwinden können, welche solche mit sich bringen.

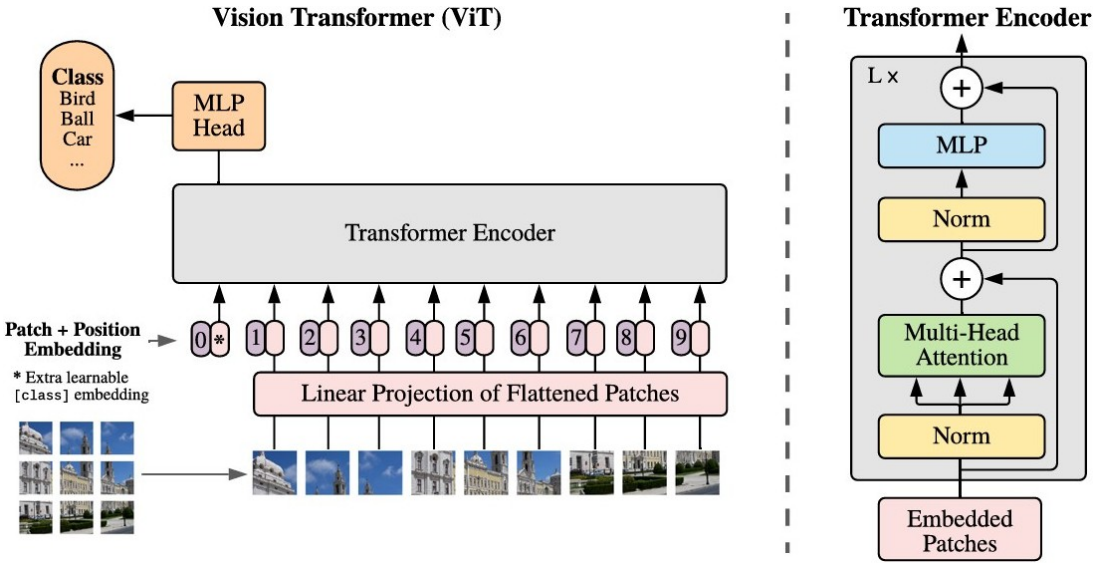


Abb. 2: Schematischer Aufbau einer Vision Transformer Architektur[53]

Architektonisch wird wie in Abbildung 2 zu sehen ist bei einem ViT das Eingabebild in mehrere Patches zerlegt und diese mittels trainierbarer linearer Projektion in einen Vektor umgewandelt. Dieser Vektor wird um eine Positionsinformation ergänzt und anschließend an den *Transformer Encoder* übergegeben[54] Durch den Einsatz von *Multi-Head Self-Attention* kann dieser *Encoder* globale Abhängigkeiten zwischen allen Patches modellieren und somit Kontext effektiv bewahren. Durch ein neuronales Netz findet anschließend eine Klassifikation statt.[55] Der *Transformer Encoder* ist dabei nicht spezifisch auf Bilddaten ausgelegt, sondern könnte auch für andere Sequenzdaten, wie *Text-Embeddings*, verwendet werden.[56]

Beispielsweise können Vision Transformers (ViTs) im Gegensatz zu CNNs besser globale Bildmerkmale sowie Zusammenhänge verschiedener Elemente bewahren[57], wodurch sie in der Praxis

[52] Vgl. Khanam et al. 2024, S. 9425
[53] Entnommen aus: Dosovitskiy et al. 2020, S. 3
[54] Vgl. Dosovitskiy et al. 2020, S. 3
[55] Vgl. Dosovitskiy et al. 2020, S. 3
[56] Vgl. Dosovitskiy et al. 2020, S. 3
[57] Vgl. Berroukham, Housni, Lahraichi 2023, S. 206

schon erfolgreich Anwendung finden und in Fällen, wie im medizinischen Umfeld zur Erkennung von bestimmten Krankheiten, CNNs übertreffen.[58]

Trotz ihrer Erfolge sind ViTs mit einer Reihe technischer Herausforderungen verbunden. Hierzu gehören neben höheren Rechenkosten durch eine große Anzahl an Parametern auch die Notwendigkeit von größeren Datensätzen, da diese ansonsten oftmals schlechter als CNN-Architekturen abschneiden.[59] ViTs sind außerdem stärker davon abhängig, dass Trainingsdaten nicht in schlechter Qualität oder verzerrt vorliegen.[60] Die Arbeit von Hütten, Meyes, Meisen 2022 zeigt allerdings, dass auch solche Herausforderungen mit geeigneten Transformer-Architekturen adressiert werden können. In ihrer Arbeit wurden verschiedene Modelle zur Anwendbarkeit von ViTs in der industriellen Inspektion am Beispiel von Güterwagen getestet. Dabei wurden Transformer-Architekturen mit CNN-basierten Ansätzen verglichen, indem sie auf dem gleichen, kleinen Datensatz trainiert und evaluiert wurden. Die Transfomer-Modelle erzielten hierbei bessere Ergebnisse als CNNs, ohne signifikante Unterschiede in der Trainingsgeschwindigkeit aufzuweisen, was die Potenziale von ViTs auch in datenlimitierten industriellen Umgebungen deutlich unterstreicht.[61]

2.4 Fehlererkennung und -klassifikation in der Fertigung

Die Fehlererkennung und -klassifikation (engl. *Fault Detection and Diagnostics (FDD)*) ist keine neue Thematik, sondern wird bereits seit Anfang der 70er Jahre untersucht.[62] Aufgrund der neuen vielversprechenden technischen Entwicklungen, insbesondere im Bereich des DL, wird diese Thematik jedoch immer relevanter, insbesondere mit Fokus auf die Implementierung von KI-Technologien zur Unterstützung dieses Prozesses.[63] Neben der reinen Fehlervermeidung bietet FDD durch ein schnelles Erkennen und korrektes Reagieren auf Fehler erhebliche wirtschaftliche Vorteile, wie die Reduktion von Prozesskosten sowie die Steigerung der Produktivität und Qualität in der Fertigung.[64]

2.4.1 Begriffsdefinition und -abgrenzung

Die Fehlererkennung (engl. *Fault Detection*) konzentriert sich darauf, festzustellen, ob ein Fehler oder eine Anomalie vorliegt. Anschließend zielt die Fehlerdiagnose (engl. *Fault Diagnostics*) darauf ab, den Fehler in Kategorien bekannter Fehler einzuordnen und somit zu klassifizieren.[65] Dazu gehört außerdem möglichst viel über den Fehler und dessen Ursprung in Erfahrung zu

[58] Vgl. Berroukham, Housni, Lahraichi 2023, S. 208
[59] Vgl. Jamil, Piran, Kwon 2022, S. 8
[60] Vgl. Berroukham, Housni, Lahraichi 2023, S. 209
[61] Vgl. Hütten, Meyes, Meisen 2022
[62] Vgl. Mercorelli 2024, S. 4
[63] Vgl. Seid Ahmed et al. 2025, S. 1
[64] Vgl. Seid Ahmed et al. 2025, S. 4
[65] Vgl. Wu, Triebe, Sutherland 2023, S. 442

bringen, weshalb man die Fehlerdiagnose üblicherweise als einen mehrstufigen Prozess betrachten kann, welcher die Erkennung, Isolation, Identifizierung sowieso Klassifizierung und Bewertung umfasst.[66]

2.4.2 Kategorien von FDD Methoden

Es wurden viele verschiedene FDD-Methoden entwickelt, welche sich in 3 Gruppen einteilen lassen: *Data-driven*, *Model-based* und *Knowledge-based* Methoden. Im Folgenden sollen diese Methoden unterschieden werden[67]

Datengetriebene Methoden (Data-driven)

Datengetriebene Methoden nutzen historische Daten, um durch statistische Techniken oder *Machine Learning*-Algorithmen charakteristische Muster und Abweichungen zu erkennen und dadurch Anomalien und Fehler erkennen und klassifizieren zu können. Diese Verfahren sind deutlich anpassungsfähiger als andere Methoden und können auch komplexe Prozesse überwachen. Sie sind insbesondere dann nützlich, wenn umfangreiche Daten bereits vorliegen. Allerdings zeigt sich hier auch der Nachteil solcher Verfahren, da große, repräsentative Datensätze für das Training solcher Modelle erforderlich sind.[68]

Modellbasierte Methoden (Model-based)

Diese Verfahren stützen sich auf ein physikalisches oder mathematisches Modell des Systems, aufgrund welchem das erwartete Verhalten berechnet werden kann. Eine Abweichung des tatsächlichen vom erwarteten Verhalten weist auf einen potenziellen Fehler hin und kann zur dessen Identifikation und Klassifikation herangezogen werden. Die größten Herausforderungen sind hierbei der Rechenaufwand sowie das nötige Verständnis über das System. Sind diese Faktoren jedoch gegeben, kann diese Methode sehr präzise sein.[69]

Wissensbasierte Methoden (Knowledge-based)

Wissensbasierte Verfahren basieren auf kodiertem Expertenwissen und sind dadurch besonders nützlich, wenn fundiertes Fachwissen vorliegt. Ein wesentlicher Vorteil liegt neben einer hohen Verarbeitungsgeschwindigkeit in der Nachvollziehbarkeit der Entscheidungen des Systems. Bei bislang unbekannten Fehlern sowie unvollständigem oder falschem Expertenwissen stoßen diese Methoden jedoch an ihre Grenzen.[70]

Hybride Methoden

Es existieren außerdem hybride Verfahren, bei welchen Elemente aus *model-based*, *data-based* und *knowledge-based* Ansätzen kombiniert werden, um jeweilige Stärken zu nutzen und bestehende

[66] Vgl. Mercorelli 2024, S. 12; Seid Ahmed et al. 2025, S. 16
[67] Vgl. Seid Ahmed et al. 2025, S. 4
[68] Vgl. Mercorelli 2024, S. 16 ff.
[69] Vgl. Mercorelli 2024, S. 21 ff.
[70] Vgl. Mercorelli 2024, S. 22 f.

Schwächen auszugleichen. Dadurch wird das System robuster und flexibler, jedoch kann hier die Integration der verschiedenen Methoden ineinander eine Herausforderung darstellen.[71]

2.4.3 Aktuelle Trends

Aktuelle Trends befinden sich zurzeit vor allem im Bereich der künstlichen Intelligenz. DL steht im Bereich der FDD immer mehr im Mittelpunkt, um Merkmale aus Rohdaten automatisiert zu extrahieren und die Fehlererkennung und Diagnose voranzutreiben.[72] Zu den populärsten Anwendungen von DL in der FDD zählen immer noch CNNs, um Bilddaten zu analysieren und Degradationsmuster aus Sensordaten zu extrahieren.[73] Allerdings zeigt die Forschung aktuell, wie bereits in 2.3 erläutert, ein steigendes Interesse an der Nutzung von ViTs in der Fertigungsindustrie, welche die Leistung von CNNs übertreffen können. ViTs bieten unter anderem durch ihr Aufmerksamkeitsmechanismen ein großes Potenzial beim Umgang mit FDD-Daten.[74]Ein wesentlicher Vorteil bei der Nutzung von Transformern besteht zudem in der Möglichkeit zur Neuheitserkennung (engl. *Novel Detection*). Das bedeutet, dass das Modell neue oder unbekannte Fehler identifizieren kann, welche vorher noch nicht beobachtet wurden.

2.4.4 Herausforderungen und Forschungsbedarf

Trotz der erzielten Fortschritte bestehen mehrere zentrale Herausforderungen. Zu diesen gehören wie schon in 2.2 beschrieben, die Notwendigkeit von großen Mengen an annotierten, vielfältigen Daten, was insbesondere im Fertigungsbereich eine aktuelle Problematik darstellt und durch die Generierung von synthetischen Daten in dieser Arbeit adressiert werden soll. Des Weiteren stellt die Komplexität und Interpretierbarkeit von Modellen eine aktuelle Herausforderung dar. DL Modelle werden oftmals als "Black Boxes"betrachtet, da ihre Entscheidungslogik nur schwer nachvollzogen werden kann.[75] Eine weitere Herausforderung stellt die Generalisierung solcher Modelle dar. Treten neue Fehler auf, welche beim Training nicht vorhanden waren, werden diese vom Modell inkorrekt klassifiziert.[76] Aus der Literatur lässt sich damit ein dringender Bedarf an leistungsfähigen FDD-Methoden ableiten, insbesondere für eine zuverlässige und echtzeitfähige Fehlerdiagnose in großskaligen Produktionssystemen, sodass weitere Forschung und Entwicklung in diesem Bereich unerlässlich ist.[77]

[71] Vgl. Mercorelli 2024, S. 22 f.
[72] Vgl. Wen et al. 2018, S. 5991
[73] Vgl. Wu, Triebe, Sutherland 2023; Wen et al. 2018, S. 5991
[74] Vgl. Wu, Triebe, Sutherland 2023, S. 440
[75] Vgl. Chai et al. 2021, S. 10
[76] Vgl. Wu, Triebe, Sutherland 2023, S. 440
[77] Vgl. Seid Ahmed et al. 2025, S. 4; Wu, Triebe, Sutherland 2023, S. 442

2.5 Remote Monitoring

2.5.1 Begriffsabgrenzung: Remote Services, Remote Operations, Remote Monitoring

Der Begriff Remote Services dient als Sammelbegriff für verschiedene Formen der ortsunabhängigen Unterstützung, unter welchen unter anderem die Serviceformen Remote Repair, Remote Maintenance und Remote Operations fallen.[78] Unter dem Begriff der Remote Operations versteht man die aktive Fernüberwachung, Ferndiagnose sowie den Fernbetrieb einer Anlage. Remote Monitoring bezeichnet die Fernüberwachung und -diagnose und kann somit als Teil der Remote Operations eingeordnet werden.[79]

2.5.2 Remote Monitoring in der Fertigung

Durch Remote Monitoring wird eine kontinuierliche Überwachung und Analyse der Fertigungsanlage gewährleistet, wodurch Fehlerfälle schneller erkannt und klassifiziert werden können. Dies erleichtert die Fehlerbehebung in den nachfolgenden Schritten, wodurch die Effizienz und Auslastung der Anlage erhöht als auch Kosten eingespart werden können.[80] Darüber hinaus schafft Remote Monitoring die Grundlage für ferngesteuerten Reparaturen und Instandhaltungen im Rahmen von Remote Operations, wodurch Kosten reduziert werden und das Personal effizienter ausgelastet werden kann.[81] Remote Services gewinnen zunehmend an Bedeutung, weshalb dieses Themengebiet einen wichtigen Forschungszweig darstellt.[82] Auch bei TRUMPF kommt Remote Monitoring zum Einsatz, um Fehler bei Stillständen von Maschinen aus der Ferne identifizieren zu können, um diese anschließend per Fernzugriff zu entstören.

2.5.3 Voraussetzungen für Remote Monitoring

Für die Umsetzung eines effektiven Remote Monitoring Systems ist es nötig, die menschlichen Sinne und menschliche Intelligenz nachzubilden, um Zustände korrekt wahrzunehmen und zu interpretieren.[83] Zur Nachbildung der menschlichen Sinne kommen je nach Anwendungsfall verschiedene Sensoren, wie beispielsweise Kameras, Mikrofone, Temperatur- oder Gassensoren, zum Einsatz.[84] Um menschliche Intelligenz nachzubilden eignen sich verschiedene Methoden der künstlichen Intelligenz. Die Auswahl des geeigneten Algorithmus richtet sich hierbei ebenfalls nach den spezifischen Anforderungen des Anwendungsfalls.Trunzer et al. 2024, S. 597

[78] Vgl. Holtbrügge, Holzmüller, Von Wangenheim 2007, S. 5
[79] Trunzer et al. 2024, S. 592 f.
[80] Vgl. Holtbrügge, Holzmüller, Von Wangenheim 2007, S. 110
[81] Vgl. Holtbrügge, Holzmüller, Von Wangenheim 2007, S. 5, 110
[82] Vgl. Holtbrügge, Holzmüller, Von Wangenheim 2007, S. 5
[83] Vgl. Trunzer et al. 2024, S. 596
[84] Vgl. Trunzer et al. 2024, S. 596 f.

2.6 Evaluationsmethoden für CV-Modelle

2.6.1 Grundlagen der Modellevaluation

KI Modelle zu evaluieren ist entscheidend, um ihre Leistung, Zuverlässigkeit und Eignung für den jeweiligen Anwendungsfall beurteilen und verschiedene Modelle vergleichen zu können. Sie ermöglicht zudem den Vergleich unterschiedlicher Modellarchitekturen, wie beispielsweise CNNs und ViTs.[85]

Ziel der Modellevaluation ist es, die Leistungsfähigkeit eines Modells quantitativ zu erfassen. Neben der reinen Genauigkeit (engl. *Accuracy*) spielen außerdem weitere Kennzahlen wie Präzision (engl. *Precision*), Sensitivität (engl. *Recall*) und der harmonische Mittelwert dieser beiden, der $F1$-*Score*, eine zentrale Rolle, insbesondere bei unausgeglichenen Datensätzen[86]. Des Weiteren ist die Robustheit von Modellen, also die Fähigkeit, unter variierenden Eingabebedingungen (z.B. Rauschen, Helligkeit, Rotation, Bildqualität, geometrische Unterschiede) konsistente Ergebnisse zu liefern, ebenfalls entscheidend.[87] Diese kann durch Tests unter veränderten Bedingungen ermittelt werden.[88]

2.6.2 Zentrale Evaluationsmetriken für die Objekterkennung

Während die Genauigkeit angibt, welcher Anteil aller Vorhersagen korrekt ist, misst die Präzision (P) den Anteil der als positiv klassifizierten Vorhersagen, welche auch korrekterweise positiv (engl. *True Positive (TP)*) sind.[89]

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Die Sensitivität (R) hingegen misst den Anteil der korrekt erkannten positiven Instanzen an allen tatsächlich positiven Instanzen, also True Positive (TP) + False Negative (FN).[90]

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Für Objekterkennungsaufgaben werden typischerweise spezifische Metriken eingesetzt, die neben der Klassifikationsleistung auch die Genauigkeit der Lokalisierung berücksichtigen, wie etwa die Average Precision (AP) oder die daraus abgeleitete Mean Average Precision (mAP).[91] Grundlage der Bewertung, ob eine Detektion als korrekt gilt, ist die Intersection over Union (IoU), welche den Überlappungsgrad zwischen vorhergesagter und tatsächlicher *Bounding Box* misst.[92] Ein

[85] Vgl. Ali et al. 2024, S. 105281
[86] Vgl. Arslanoglu, Albayrak, Acar 2025, S. 65240
[87] Vgl. Arslanoglu, Albayrak, Acar 2025, S. 65234
[88] Vgl. „Computer Vision Overview“ 2023, S. 323
[89] Vgl. Arslanoglu, Albayrak, Acar 2025, S. 65240
[90] Vgl. Arslanoglu, Albayrak, Acar 2025, S. 65240
[91] Vgl. Khanam et al. 2024, S. 94272
[92] Vgl. Khanam et al. 2024, S. 94272

gängiger Schwellenwert hierfür ist 0.5, wonach die Detektion als erfolgreich klassifiziert wird, wenn die vorhergesagte und tatsächliche *Bounding Box* um mindestens 50% übereinstimmen.[93]

Die AP entspricht der Fläche unter der Precision–Recall-Kurve bei einem festgelegten IoU-Schwellenwert (z. B. 0.5. für AP50).[94]

$$\mathrm{AP} = \int_0^1 p(r)\, dr$$

Die mAP ist der Mittelwert der AP-Werte über alle Klassen.[95] Je nach Evaluationsprotokoll kann zudem über mehrere IoU-Schwellenwerte gemittelt werden. In der Praxis wird häufig mAP50 verwendet, d. h. der Mittelwert der AP über alle Klassen bei IoU-Schwelle 0.5.[96]

In diesem Zusammenhang wird häufig der mAP50 verwendet, welcher die mAP-Metrik mit Verwendung der IoU-Schwelle 0.5 berechnet.[97]

2.6.3 Vergleich und Auswahl von Metriken

Die Auswahl geeigneter Evaluationsmetriken hängt maßgeblich von den Datencharakteristika, der Aufgabenstellung und den Prioritäten des Anwendungsfalls ab. In der Objekterkennung hat sich insbesondere die mAP etabliert, da sie sowohl Klassifikations- als auch Lokalisierungsfehler berücksichtigt und damit ein differenziertes Bild der Modellleistung liefert.[98]

Insbesondere in sicherheitskritischen Bereichen ist es darüber hinaus sinnvoll, die Robustheit eines Modells gegenüber veränderten Eingabebedingungen mit zu berücksichtigen, um ein umfassenderes Bewertungsbild zu erhalten.[99] Häufig empfiehlt sich daher eine kombinierte Evaluation, bei der mehrere Kennzahlen einbezogen werden, um unterschiedliche Leistungsaspekte des Modells abzudecken.

[93] Vgl. Khanam et al. 2024, S. 94272
[94] Vgl. Zaripov, Kulshin, Sidorov 2025, S. 13
[95] Vgl. Zaripov, Kulshin, Sidorov 2025, S. 14
[96] Vgl. Zaripov, Kulshin, Sidorov 2025, S. 13 f.
[97] Vgl. Zaripov, Kulshin, Sidorov 2025, S. 13
[98] Vgl. Arslanoglu, Albayrak, Acar 2025, S. 65240; Khanam et al. 2024, S. 94272
[99] Vgl. Arslanoglu, Albayrak, Acar 2025, S. 65234; Ali et al. 2024, S. 105282

3 Zielspezifikation und Darlegung des Forschungsdesigns[100]

3.1 Zielsetzung

Ziel dieser Arbeit ist die prototypische Entwicklung und Evaluation eines IT-Artefakts in Form einer Datenpipeline zur Generierung synthetischer Bilddaten aus einem digitalen Modell einer Laserschneidmaschine in einer Simulationsumgebung. Die Pipeline soll automatisiert umfangreiche und qualitativ hochwertige Bilddatensätze erzeugen, die spezifische Fehlerzustände der Maschine abbilden. Dabei stehen die Reproduzierbarkeit des Prozesses, die Flexibilität durch Parametrisierung sowie die einfache Erweiterbarkeit um zusätzliche Fehlerzustände im Vordergrund.

Zur Validierung der Pipeline werden die erzeugten Daten exemplarisch für das Training und die Evaluation von KI-Modellen eingesetzt. Diese Modelle bilden nicht den Hauptfokus der Arbeit, sondern dienen als Mittel, um die Eignung und Konsistenz der generierten synthetischen Daten nachzuweisen. Die Evaluation der Modelle erfolgt dabei sowohl auf synthetischen als auch realen Bilddaten. Die Arbeit versteht sich somit als *Proof-of-Concept*, der die Machbarkeit und das Potenzial pipelinebasierter Datengenerierung für die KI-gestützte Fehlererkennung in der Fertigungsindustrie aufzeigt. Erzielte Ergebnisse sollen dabei auch auf andere Domänen übertragbar sein, da der Mangel an ausreichenden und umfassenden Bilddaten für solche Anwendungen nicht ausschließlich in der Fertigungsindustrie eine Problematik darstellt.[101]

Exemplarisch wird die Pipeline in einem konkreten Anwendungsfall eingesetzt, bei welchem erkannt werden soll, ob ein ausgeschnittenes Teil während des Betriebs einer Laserschneidmaschine noch im Blech vorhanden ist. Mit einer solchen Information kann nachgelagerten Prozessen der konkrete Fehlerfall klassifiziert und im Idealfall autonom behoben werden. Dieser Anwendungsfall dient als Demonstrator, um die Funktionsfähigkeit und den praktischen Nutzen der Pipeline im Rahmen eines *Proof-of-Concepts* zu veranschaulichen.

3.2 Forschungsmethodik

Zur Erreichung der in Abschnitt 3.1 beschriebenen Zielsetzung wird das DSR-Paradigma angewendet. Bei DSR handelt es sich um einen anerkannten Forschungsansatz in Informationssystemen, der darauf abzielt, durch die Entwicklung und Evaluation innovativer Artefakte, Probleme in der Praxis zu lösen und gleichzeitig einen Beitrag zur wissenschaftlichen Wissensbasis zu leisten.[102] Im Rahmen von DSR werden Artefakte, wie z.B. Systeme, Methoden oder Modelle

[100] Sprachlich geglättet durch ChatGPT-5
[101] Vgl. Bai et al. 2023, S. 2
[102] Vgl. The Australian National University et al. 2013, S.337

entworfen und iterativ in einem *Build-and-Evaluate*-Zyklus weiterentwickelt, um ihre Eignung zur Problemlösung zu überprüfen und zu verbessern.[103] Dieser Prozess umfasst die Identifikation des Problems, die Definition von Anforderungen, die Entwicklung eines Artefakts sowie dessen Demonstration, Evaluation und Kommunikation. [104]

Das in dieser Arbeit entwickelte Artefakt besteht aus einer Pipeline zur Generierung synthetischer Bilddaten, die auf einem digitalen Modell in einer Simulationsumgebung basiert. Das Ziel ist es, die Datengrundlage für KI-Modelle zur Fehlererkennung im Fertigungsbereich zu verbessern, insbesondere in Hinblick auf Fortschritte im Bereich des Remote Monitoring. Bereits in Abschnitt 2.2 genannte Probleme, wie Kosten- und Zeitaufwand sowie der *Domain Gap* sollen durch dieses Artefakt konzeptionell adressiert werden.

Nach der Definition der Problemstellung und Zielsetzung sowie Erläuterung zentraler theoretischer Grundlagen wird das Artefakt entworfen, implementiert und in einem Demonstrator angewendet. Anschließend erfolgt die Evaluation auf synthetischen und realen Bilddaten sowie eine kritische Reflexion der Ergebnisse.

Die Wahl des DSR-Paradigmas ist für den vorliegenden Use Case besonders geeignet, da es sich um ein praxisnahes Problemfeld handelt. Die genannten Herausforderungen werden durch DSR adressiert, indem ein Artefakt in Form einer synthetischen Datenpipeline entwickelt und evaluiert wird. Es wird dadurch einerseits ein praktischer Nutzen erzielt,[105] indem der Umfang und die Qualität der Daten für das Training von KI-Modellen verbessert werden. Andererseits wird die Effektivität synthetischer Daten empirisch untersucht und insbesondere im Hinblick auf den *Domain Gap* reflektiert, wodurch ein wissenschaftlicher Beitrag im Sinne des DSR geleistet wird.[106]

Die vorliegende Arbeit lässt sich im Sinne des *DSR Knowledge Contribution Framework* als *Improvement* einordnen.[107] Zwar sind Probleme wie Datenknappheit, Zeitaufwand und *Domain Gap* bei synthetischen Daten bereits bekannt, jedoch stellt die in dieser Arbeit entwickelte Datenpipeline eine neue und domänenspezifische Lösung für das Training von *KI*-Modellen zur Fehlererkennung dar. Damit trägt die Arbeit durch die Weiterentwicklung bestehender Konzepte zur verbesserten Lösung eines etablierten Problems bei.

3.3 Darlegung des Forschungsdesigns

3.3.1 Artefaktdefinition

Bei dem entwickelten Artefakt handelt es sich um eine Datenpipeline zur Generierung umfangreicher synthetischer Bilddaten und zum Training von KI-Modellen auf diesen zur Fehlererkennung

[103] Vgl. Hevner et al. 2004, S. 78
[104] Vgl. Peffers et al. 2007
[105] Vgl. The Australian National University et al. 2013, S. 341
[106] Vgl. The Australian National University et al. 2013, S. 342
[107] Vgl. The Australian National University et al. 2013, S. 345 f.

in der Fertigungsindustrie. Hierzu soll ein Fertigungsprozess simuliert werden, welcher mit realitätsnah konfigurierten Kameras erfasst werden soll. Um die Generalisierungsfähigkeit des Modells zu erhöhen soll im Rahmen der Datengenerierung auf *Domain Randomization* zurückgegriffen werden.[108] Dabei werden verschiedene Parameter, wie beispielsweise Lichtverhältnisse, das bearbeitete Material oder Kameraperspektiven, zufällig verändert, um das Modell robuster gegen äußere Einflüsse zu gestalten und die Gefahr von *Overfitting* zu reduzieren.[109] Auch Elemente des Produktionsprozesses, wie verschiedene Fehlersituationen oder Störobjekte werden zufällig verändert, um die Übertragbarkeit auf reale Anwendungsfälle zu verbessern.[110] Zentral ist dabei, dass die Pipeline flexibel und erweiterbar gestaltet ist, sodass problemlos weitere Fehlerfälle integriert sowie Qualität und Umfang der generierten Daten angepasst werden können. Bei der Umsetzung und Implementierung wird zudem auf eine systematische und detaillierte Dokumentation geachtet, um die Reproduzierbarkeit der Experimente sicherzustellen.

Die Bilddaten sollen aus dem Simulationsprogramm exportiert werden und für das Training verschiedener KI-Modelle und Architekturen Verwendung finden. Die verschiedenen Modelle werden aufgrund etablierter Metriken verglichen und bezüglich ihrer Leistung und Generalisierungsfähigkeit evaluiert. Essenziell ist dabei, dass die Trainings- und Testvoraussetzungen für alle Modelle und Architekturen möglichst identisch sind, um eine bestmögliche Vergleichbarkeit zu gewährleisten. Wie schon in Abschnitt 3.1 beschrieben steht allerdings die Validierung der generierten Daten im Vordergrund und nicht die Entwicklung eines optimalen Modells für den Anwendungsfall.

3.3.2 Vorgehensweise

Das methodische Vorgehen in dieser Arbeit orientiert sich wie bereits in Abschnitt 3.2 beschrieben am DSR-Paradigma. Zunächst erfolgt die Konzeption des Artefakts in Form einer Datenpipeline. Anschließend wird diese Pipeline für die Generierung synthetischer Bilddaten eingesetzt, wobei mittels *Domain Randomization* verschiedene Parameter zufällig variiert werden. Pixelgenaue Annotationen der synthetischen Bilddaten werden automatisch durch die Simulationsumgebung hinzugefügt und gemeinsam mit den Bilddaten gespeichert. Die Annotationen werden in ein für das Training des Modells geeignetes Format überführt und anschließend zusammen mit den Bilddaten in ein Trainings-, Validierungs- und Testdatensatz unterteilt. Diese Datensätze werden für das Training verschiedener *KI*-Modelle sowie deren anschließende Evaluation verwendet. Während des Trainings kommt zusätzlich eine weitere Variation und Vergrößerung des Datensatzes durch Datenaugmentation (engl. *Data Augmentation*) zum Einsatz. Unter Datenaugmentation wird die gezielte Variation vorhandener Trainingsbilder, beispielsweise durch Anpassung von Belichtung, Sättigung, Farben, Zoom oder Bildkomposition verstanden. Ziel ist es, den Datensatz künstlich zu vergrößern und die Generalisierungsfähigkeit des Modells, ins-

[108] Vgl. Fulir et al. 2023, S. 4427
[109] Vgl. Urgo, Terkaj, Simonetti 2024, S. 263; Fulir et al. 2023, S. 4427
[110] Vgl. Monnet, Petrovic, Herfs 2024, S. 767

besondere bei Limitationen durch kleine Datensätze, zu verbessern.[111] Nach dem Training der *KI*-Modelle erfolgt deren Evaluation auf Basis der in Abschnitt 3.3.4 definierten Metriken. Die erzielten Ergebnisse dienen in erster Linie als Validierung der generierten synthetischen Daten und sollen Aufschluss darüber geben, inwiefern diese für industrielle Anwendungen geeignet sind. Darüber hinaus werden die Erkenntnisse hinsichtlich des *Domain Gaps* reflektiert. Im Sinne des DSR ist es vorgesehen, die gewonnenen Erkenntnisse zu nutzen, um das Artefakt iterativ weiterzuentwickeln.

3.3.3 Datengrundlage

Die Datengrundlage der Experimente bilden überwiegend synthetische Bilddaten, welche mithilfe eines digitalen Modells in einer Simulationsumgebung generiert werden. Dabei werden Zustände und Fehlerfälle des Fertigungsprozesses simuliert und wie bereits in Abschnitt 3.3.1 erläutert, verschiedene Parameter wie unter anderem Lichtverhältnisse, Materialien und die Kameraperspektive durch *Domain Randomization* variiert.

Ergänzend zu den synthetischen Daten wird eine kleinere Menge an realen Bilddaten herangezogen, welche als Testdaten zur Evaluation der trainierten Modelle dient. Dadurch soll überprüft werden, inwieweit Modelle, welche ausschließlich auf synthetischen Daten trainiert wurden, in industriellen Anwendungen leistungsfähig sind und welche Schwierigkeiten bei der Übertragbarkeit bestehen.

Als Basis für die Generierung der Bilddaten dienen Universal Scene Description (USD)-Modelle, welche primär verschiedene Zustände von einem Blech im Schneidprozess abbilden. Die zugrunde liegenden 3D-Modelle wurden durch die TRUMPF SE + Co. KG bereitgestellt und in die Simulationsumgebung übertragen.

3.3.4 Evaluationsdesign

Die Evaluation des entwickelten Artefakts erfolgt anhand etablierter Metriken aus dem Bereich der Objekterkennung. Insbesondere die Kennzahlen Average Precision (AP) und Mean Average Precision (mAP), die auf der Intersection over Union (IoU) basieren, sind für die Objekterkennung und auch für diesen Anwendungsfall zentral.[112] Verwendet werden dabei insbesondere die Varianten mAP@50, bei der die mAP bei einer IoU-Schwelle von 0,5 berechnet wird, sowie mAP@[.5:.95], bei der die AP-Werte über die IoU-Schwellen 0,5 bis 0,95 berechnet und gemittelt werden. Die Metrik mAP@[.5:.95] erfordert daher ein höheres Maß an Präzision, wodurch eine genauere Analyse der Lokalisierungsgenauigkeit ermöglicht wird.[113]

[111] Vgl. Shorten, Khoshgoftaar 2019, S. 2 f. Ultralytics 2025b
[112] Vgl. Khanam et al. 2024, S. 94272
[113] Vgl. Khanam et al. 2024, S. 94272 f.

Vor der Berechnung dieser Kennzahlen werden die Modellvorhersagen einem Non-Maximum Suppression (NMS) unterzogen. NMS ist ein Verfahren, bei dem überlappende Bounding Boxes entfernt werden, um Mehrfacherkennungen desselben Objekts zu vermeiden und die Vergleichbarkeit der Ergebnisse zu gewährleisten.[114]

Die Vergleichbarkeit der Ergebnisse zwischen verschiedenen Architekturen und Modellvarianten wird durch konsistente Rahmenbedingungen sichergestellt. Dazu zählen unter anderem identische Datensätze, einheitliche Trainings-, Validierungs- und Test-Splits sowie identische Hyperparameter bei Modellvarianten der gleichen Architektur. Auf diese Weise kann gewährleistet werden, dass Unterschiede in der Modellleistung auf die Datengrundlage oder gewählte Trainingsstrategie zurückzuführen sind und nicht auf methodische Inkonsistenzen.

3.3.5 Reproduzierbarkeit und Validität

Um die Experimente und deren Ergebnisse bestmöglich reproduzieren zu können, werden sämtliche Parameter der Datengenerierung sowie des Modelltrainings systematisch dokumentiert. Dazu zählen beispielsweise die Konfiguration von Licht- und Materialverhältnisse sowie von Kamerapositionen im Rahmen der *Domain-Randomization*, die verwendeten Software- und Modellversionen sowie die Hyperparameter des Modelltrainings.

Hinsichtlich der Validität kann zwischen interner und externer Validität unterschieden werden.[115] Interne Validität bezeichnet, ob die Auswertungen der Arbeit zuverlässige und korrekte Ergebnisse auf die Forschungsfrage gibt.[116] Sie wird durch eine einheitliche Wahl und korrekte Dokumentierung relevanter Parameter gewährleistet. Externe Validität hingegen bezieht sich auf die Übertragbarkeit der Ergebnisse auf andere Kontexte.[117] Da es sich in dieser Arbeit um ein *Proof-of-Concept* handelt, ist die externe Validität nur eingeschränkt gegeben. Gleichzeitig sind der *Domain Gap* und die beschriebenen Herausforderungen auch auf andere Domänen übertragbar, sodass die Ergebnisse der Arbeit eine potenzielle Relevanz über den spezifischen Anwendungsfall hinaus liefern.

[114] Vgl. Hosang, Benenson, Schiele 2017, S. 4508 f.
[115] Vgl. Andrade 2018, S. 499
[116] Vgl. Andrade 2018, S. 499
[117] Vgl. Andrade 2018, S. 499

4 Training von KI-Modellen auf synthetischen Daten [118]

Die vorliegende Umsetzung wird bei der TRUMPF SE + Co. KG (im Folgenden TRUMPF) durchgeführt. Das familiengeführte Hochtechnologieunternehmen mit Sitz in Ditzingen bietet Fertigungslösungen in den Bereichen Werkzeugmaschinen, Lasertechnik, Elektronik und Elektrowerkzeuge an. Als einer der führenden Anbieter im Laserschneiden treibt TRUMPF die Digitalisierung in der industriellen Fertigung maßgeblich voran.[119]

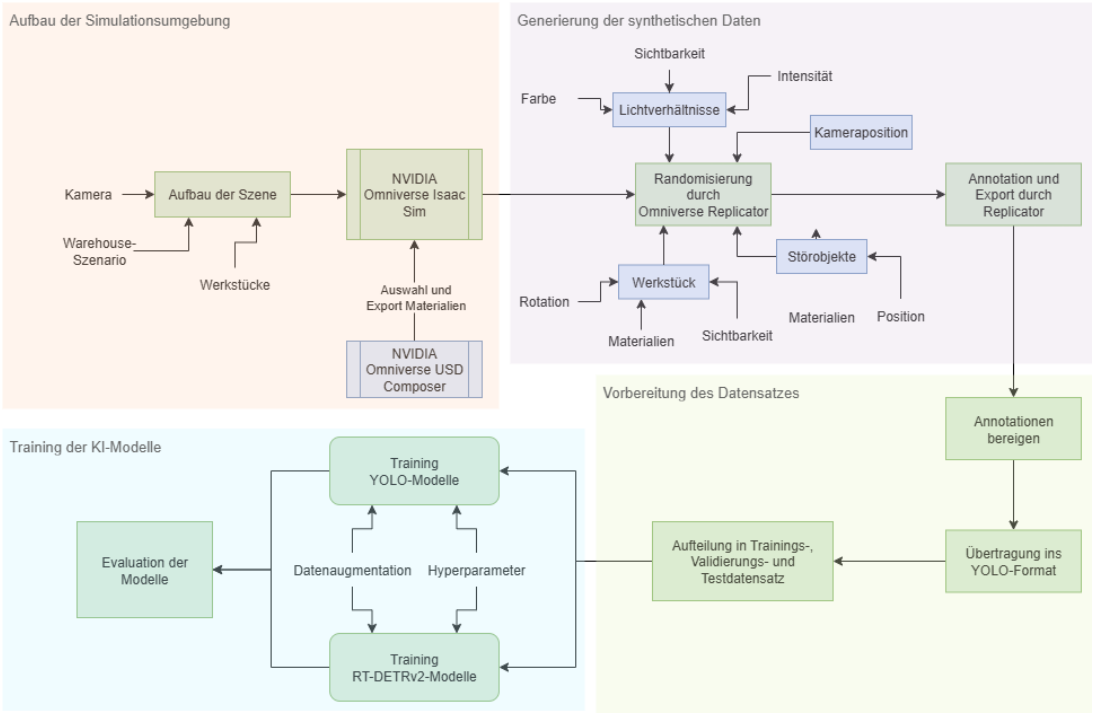


Abb. 3: Grafische Visualisierung der Datengenerierungs- und Trainingspipeline

Abbildung 3 bietet eine grafische Übersicht über die in dieser Arbeit entwickelte Datengenerierungs- und Trainingspipeline. Diese Pipeline wird im folgenden Kapitel detailliert beschrieben.

4.1 Zielsetzung und Forschungsmethodik

Ziel dieses Kapitels ist die praktische Umsetzung des in Kapitel 3.3 beschriebenen Forschungsdesigns. Der Fokus liegt auf der Realisierung einer Datenpipeline, mithilfe welcher automatisch synthetische Bilddaten inklusive deren Annotationen in einer Simulationsumgebung generiert werden. Die Pipeline soll so gestaltet sein, dass zentrale Parameter, die den Umfang und die Qualität der Daten betreffen, flexibel angepasst werden können. Darüber hinaus soll die Pipeline

[118] Sprachlich geglättet durch ChatGPT-5
[119] TRUMPF SE & Co. KG 2025

so konzipiert sein, dass sie leicht auf andere Fehlerfälle anpassbar ist, wodurch ihre Wiederverwendbarkeit und Übertragbarkeit auf andere Anwendungsszenarien sichergestellt wird. Die generierten Daten werden anschließend für das Training von KI-Modellen aufbereitet. Darauf aufbauend erfolgt exemplarisch ein Training verschiedener Modelle und Modellarchitekturen zur Fehlererkennung, um die Qualität der durch die Pipeline erzeugten Daten zu validieren.

Methodisch folgt das Vorgehen dem DSR-Paradigma, nach welchem das Artefakt in einem *Build-and-Evaluate* Prozess entwickelt wird. Konkret wird das Artefakt in iterativen Zyklen entwickelt, getestet und verbessert. Dieses Kapitels konzentriert sich auf die Entwicklung des Artefakts, während eine detaillierte Evaluation der erzielten Ergebnisse in Kapitel 5 folgt.

4.2 Aufbau der Simulationsumgebung

Im Sinne der Abbildung 3 wurden für die Umsetzung der Simulationsumgebung *NVIDIA Omniverse*-Tools eingesetzt. Konkret kamen *NVIDIA Omniverse USD Composer* auf Basis von *Omniverse Kit* (Version 107.3.0) sowie *NVIDIA Isaac Sim* (Version 4.5.0 rc.36) zum Einsatz. Diese Tools sind dafür bekannt, hochqualitative und realistische Bilddaten erzeugen zu können und wurden auch schon in ähnlichen Arbeiten, wie der Arbeit von Monnet, Petrovic, Herfs 2024 zur Generierung von Kratzern auf Metalloberflächen, verwendet.[120]

Der *USD Composer* ist primär auf den Aufbau und die Gestaltung von Computer-Aided Design (CAD)-Szenen konzipiert[121] und wurde in diesem Anwendungsfall hauptsächlich für die Auswahl und Speicherung unterschiedlicher Materialvarianten für das Werkstück und der Störobjekte verwendet. Der Großteil der Umsetzung erfolgte allerdings in *Isaac Sim*[122], da dort die Implementierung des *Omniverse Replicator* eine effiziente Realisierung der *Domain Randomization* sowie den Export der generierten Bilddaten ermöglichte.[123]

Als Grundlage der Simulationsumgebung diente ein von *Isaac Sim* bereitgestelltes Warehouse-Szenario, welches aufgrund der bereits implementierten Beleuchtungselementen ausgewählt wurde. Hierdurch kann eine realitätsnahe Umsetzung, insbesondere in Hinblick auf die Lichtverhältnisse, erreicht werden. In die Szene wurde außerdem eine von TRUMPF bereitgestellte Fertigungsmaschine platziert sowie ergänzend 84 Zustände eines Blechs im Schneideprozess, einschließlich der vollständig ausgeschnittenen Teile. Die CAD-Dateien für Blech und Maschine wurden aus der Software *Siemens NX* im USD-Format exportiert. Grundsätzlich können jedoch jegliche USD-Modelle eingebunden werden, wodurch die Simulationsumgebung flexibel erweiterbar bleibt.

Zur Bildaufnahme wurde eine virtuelle Kamera implementiert, welche ebenfalls durch *Isaac Sim* bereitgestellt wird. Diese wurde mit Parametern konfiguriert, die den realen Produktionskameras

[120] Vgl. Monnet, Petrovic, Herfs 2024, 769 ff.
[121] NVIDIA 2025c
[122] NVIDIA 2025a
[123] NVIDIA 2025b

entsprechen, wodurch eine möglichst enge Annäherung an reale Produktionsszenarien erreicht werden kann.

4.3 Generierung der synthetischen Bilddaten

Aufbauend auf der in Abbildung 3 dargestellten Systematik erfolgte im Anschluss an den Aufbau der Simulationsumgebung die Generierung der Bilddaten mit zufälliger Variation zentraler Parameter im Sinne der *Domain Randomization* mithilfe des *Omniverse Replicator*.[124] Bei dem *Omniverse Replicator* handelt es sich um ein Framework, mithilfe welchem Pipelines zur Generierung umfassender synthetischer Daten gebaut werden können. Durch diesen ist es möglich, bestehende USD-Szenen zu Nutzen oder neue Szenen dynamisch aufzubauen. Verschiedene Elemente und deren Eigenschaften in der Szene können mit diesem Tool zufällig variiert, automatisch annotiert und exportiert werden.[125]

Aufgrund zeitlicher Beschränkungen wurde exemplarisch ein konkreter Anwendungsfall umgesetzt, der wie folgt definiert ist: Ein vollständig ausgeschnittenes Blech wird auf der Maschine platziert und die ausgeschnittenen Teile für jedes Bild zufällig in die vorgesehen Aussparung zurück platziert. Die Teile werden dabei zufällig um einen Winkel rotiert, deren Spannweite durch einen Parameter in der Pipeline festgelegt werden kann. In diesem Anwendungsfall betrug der Rotationsbereich -4° bis +4° relativ zur Ausgangsposition. Weitere potenzielle Szenarien, wie beispielsweise verschobene oder nicht vollständig herausgetrennte Teile, konnten aufgrund zeitlicher Einschränkungen nicht mehr implementiert werden, können jedoch aufgrund der flexiblen Architektur problemlos ergänzt werden.

Neben der Variation der Werkstücke wurden auch Umgebungsparameter randomisiert. Dazu zählen insbesondere die Materialien des Blechs und der ausgeschnittenen Teile sowie die Umgebungsbeleuchtung. Bei letzterer wurde sowohl die Sichtbarkeit einzelner Lichter als auch deren Intensität und Farbe zufällig variiert. Zudem wurde die Position der Kamera randomisiert, wobei sowohl die Entfernung zum Werkstück, als auch die horizontale und vertikale Position innerhalb vorgegebener Grenzen variiert wurde. Auf diese Weise sollte die Robustheit des Modells gegenüber unterschiedlicher Blickwinkel und Bildausschnitte erhöht werden.

Zentrale Parameter der Pipeline können zu Beginn des Skripts nach Bedarf angepasst werden. Hierzu zählen verschiedene Einstellungen zur Bildqualität, wie beispielsweise Antialiasing, Bildauflösung und Parameter zur Simulation des Lichts. Auch die Konfiguration der *Domain Randomization* kann im Skript angepasst werden. Die konkrete Implementierung der Datengenerierungspipeline ist in Anhang 1 aufgeführt.[126]

Abbildung 4 zeigt zwei beispielhafte Bilder, welche mithilfe des Replicators generiert wurden. Insgesamt wurden 1077 Bilder exportiert. Ein größerer Umfang wäre prinzipiell möglich gewesen,

[124] NVIDIA 2025b
[125] Vgl. NVIDIA 2025b
[126] Vgl. Anhang 1

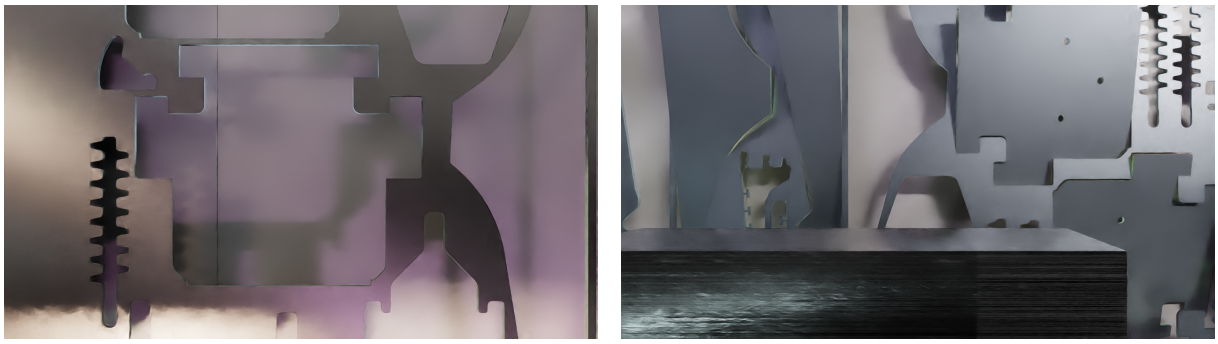


Abb. 4: Beispiele für exportierte Bilder aus dem Replicator

war jedoch aufgrund der verfügbaren Rechenleistung und des damit verbundenen Zeitaufwands im Rahmen dieser Arbeit nicht umsetzbar.

Diese Bilddaten werden dabei durch den Replicator automatisch annotiert. Ein Beispiel hierfür ist in Abbildung 5 dargestellt. Zwar liegen die exportierten Annotationen im Textformat vor, für dieses Beispiel wurden sie jedoch zur Veranschaulichung visualisiert.

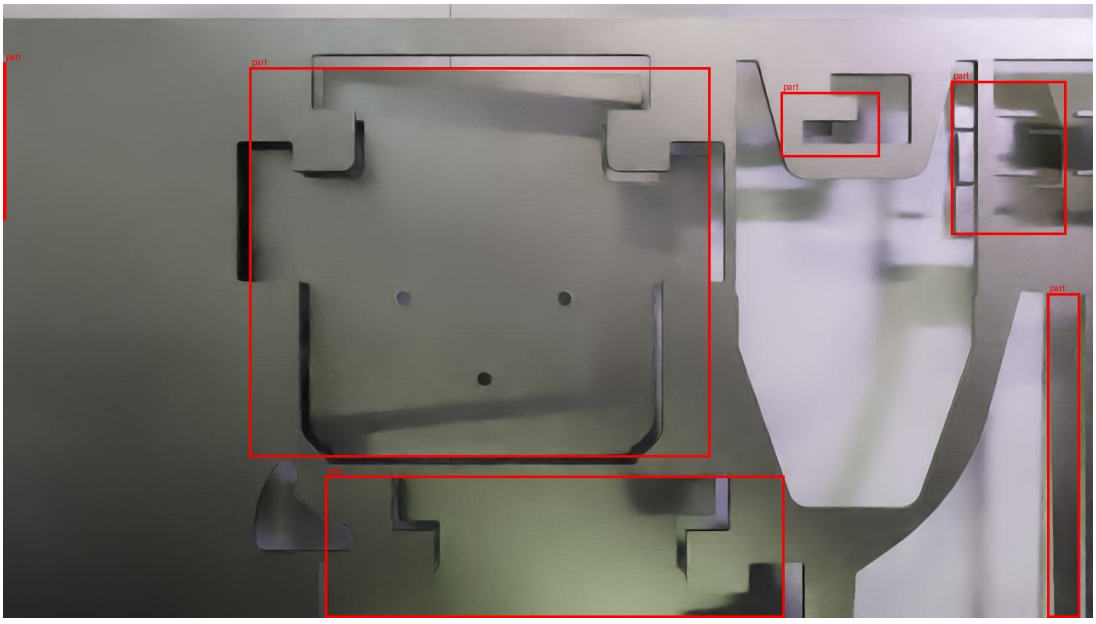


Abb. 5: Beispiel einer durch den Replicator automatisch erzeugten Annotation (grafisch visualisiert)

4.4 Training von KI-Modellen auf synthetischen Bilddaten

4.4.1 Auswahl der KI Architekturen und Modelle

Als erstes Modell wurde die YOLO-Architektur ausgewählt, welche auf CNNs basiert und sich in zahlreichen Benchmarks als leistungsfähiger und effizienter Standard im Bereich der Objekterken-

nung erwiesen hat. [127] Verwendet wurde dabei die Implementierung von *Ultralytics*, die aufgrund ihrer einfachen Handhabung, umfangreicher Dokumentation und integrierter Möglichkeiten zur Datenaugmentation ein besonders praxistaugliches Framework bietet.[128] Aufgrund begrenzter Hardwarekapazitäten konzentrierte sich diese Arbeit auf das Training kleinerer Modellvarianten (Nano, Small und Medium).

Angesichts der kleinen Größe des Datensatzes und der Handhabung von synthetischen Daten wurde ein relativ hoher Grad an Datenaugmentation beim Training eingesetzt, um die Generalisierungsfähigkeit der Modelle zu verbessern.

4.4.2 Vorbereitung des Datensatzes und Training der Modelle

Vor dem tatsächlichen Training der KI-Modelle wurden die durch den *Omniverse Replicator* exportierten Bild- und Annotationsdaten im Sinne der Abbildung 3 zuerst in ein geeignetes Format überführt. Der Export aus dem Replicator erfolgte im sogenannten KITTI-Format, welches sich nicht unmittelbar für die nachfolgenden Trainingspipelines eignete. Für das Training der YOLO-Modelle mussten die Annotationsdaten in das YOLO-Format konvertiert werden. Dazu wurden Bild- und Annotationsdateien in einer automatisierten Pipeline konsolidiert, das Format der Annotationsdaten angepasst und der Datensatz in Trainings-, Validierungs- und Test-Splits unterteilt.[129] Dabei wurde ein Split von 70/20/10 (70% Training, 20% Validierung, 10% Test) gewählt, was eine anerkannte Aufteilung in der Literatur darstellt.[130]

Auf diesem Datensatz wurden drei Modelle der YOLO11-Architektur trainiert (Nano, Small und Medium). Um die effektive Größe und Varianz des Trainingsdatensatzes zu erhöhen, kamen umfangreiche Verfahren der Datenaugmentation zum Einsatz, welche durch das Trainingsframework von *Ultralytics* bereitgestellt werden.[131] Die Auswahl der Hyperparameter bestimmte sich primär aus der offiziellen Dokumentation von *Ultralytics*[132] sowie einem iterativen Trainingsansatz.[133] Die Anzahl der Epochen wurde auf 200 festgelegt, mit dem Ziel, durch eine hohe Augmentierung in Kombination mit einer hohen Anzahl an Epochen den kleinen Datensatz bestmöglich auszunutzen und *Overfitting* zu vermeiden.

Neben den YOLO-Modellen wurde ergänzend ein transformerbasiertes Modell trainiert. Da herkömmliche Transformer-Architekturen bei kleinen Datensätzen ineffizient sind, wurde der RT-DETRv2 gewählt, eine speziell optimierte Variante für ressourcenschonendes Training [134]. Hier wurde sowohl ein Modell mit Datenaugmentation als auch eines ohne Datenaugmentation

[127] Vgl. Monnet, Petrovic, Herfs 2024, S. 771; Bilous et al. 2024
[128] Vgl. Jocher, Qiu 2024
[129] Vgl. Anhang 2
[130] Vgl. Urgo, Terkaj, Simonetti 2024, S. 261; Griem et al. 2025, S. 3; Khirodkar, Yoo, Kitani 2018, S. 5
[131] Vgl. Anhang 3
[132] Vgl. Ultralytics 2025a; Ultralytics 2025b
[133] ChatGPT-5 wurde unterstützend bei der Optimierung der Hyperparameter eingesetzt.
[134] Vgl. Zhao et al. 2023

trainiert. Aus zeitlichen Gründen wurde hierbei ausschließlich eine kleinere Variante (RT-DETR-18) eingesetzt.[135] Auch hier wurde eine hohe Anzahl an Epochen (220) gewählt, um den kleinen Datensatz bestmöglich auszuschöpfen. Allerdings konvergierten beide Modelle bereits nach kurzer Zeit und beendeten das Training bereits nach etwa 40 Epochen, da keine weiteren Verbesserungen erzielt werden konnten.

4.4.3 Probleme während des Trainings

Im Rahmen der Evaluation traten spezifische Probleme im Datenpipeline-Design auf. So zeigte sich, dass die mit dem KITTI-Writer erzeugten Bounding Boxes fehlerhaft erzeugt wurden, wodurch mehr Annotationen als tatsächlich Objekte vorhanden waren. Dies hatte zur Folge, dass Bounding Boxes nicht mit den tatsächlichen Objektpositionen übereinstimmten, was eine Bereinigung der Annotationsdaten erforderlich machte. Zudem wurden teilweise Bild-Label-Paare doppelt in verschiedenen Datensplits abgelegt (*Data Leakage*), was zu einer Überschätzung der Modellleistung in der Validierung führte. Diese Inkonsistenzen wurden behoben und die Trainingsläufe wiederholt.

4.4.4 Evaluation der KI-Modelle

Die trainierten Modelle wurden zunächst auf dem zuvor definierten Test-Split des synthetischen Datensatzes evaluiert.[136] Anhand der in Abschnitt 3.3.4 beschriebenen Metriken konnte so eine erste Einschätzung der Modellleistung erfolgen. Bei unzureichenden Ergebnissen wurden Trainingsparameter angepasst und das Training erneut durchgeführt. Durch diesen iterativen Prozess konnte die Modellleistung auf synthetischen Daten stetig verbessert werden.

Wie bereits in Abschnitt 3.3.4 beschrieben, wurden die Modellvorhersagen vor der Berechnung der Metriken einem NMS unterzogen. Hierbei wurde eine IoU-Schwelle von 0.5 verwendet, welches ein üblicher Wert in der Praxis darstellt.[137] Überlappende Bounding Boxes mit einem Überschneidungsgrad von 50% werden dementsprechend durch NMS reduziert.

Um die Generalisierungsfähigkeit der Modelle und ihre Übertragbarkeit auf reale Anwendungsfaälle zu überprüfen, wurde die Modellleistung ergänzend auf realen Bildern evaluiert.[138] Dazu wurde aus *TRUMPF Visual Insights*, einer unternehmensinternen Software, in welcher Videos der Produktionsmaschinen bei Fehlerfällen gespeichert sind, 34 Bilder gesammelt, welche dem simulierten Fehlerfall ähnelten. Diese Bilder wurden manuell annotiert und dienten als unabhängiger Referenzdatensatz zur abschließenden Evaluation der Modelle. Bei unzureichender Leistung wurde auch hier das Training mit angepassten Parametern wiederholt, wobei die Anzahl an Iterationen aufgrund zeitlicher Limitationen eingeschränkt war.

[135] Vgl. Anhang 4
[136] Vgl. Anhang Anhang 5/1 und Anhang Anhang 5/2
[137] Vgl. He et al. 2017, S. 8; Bodla et al. 2017, S. 3
[138] Vgl. Anhang Anhang 5/1 und Anhang Anhang 5/2

5 Evaluation der KI-Modelle zur Fehlererkennung bei Werkzeugmaschinen [139]

5.1 Zielsetzung und Forschungsmethodik

Ziel der Evaluation ist, die Leistungsfähigkeit der auf synthetischen Daten trainierten KI-Modelle zur Erkennung von Fehlerfällen bei Werkzeugmaschinen zu überprüfen. Dabei wird sowohl die Leistung der KI-Modelle auf synthetischen Daten als auch der Übertrag auf die Realität durch Tests auf realen Daten betrachtet. Dies schafft eine Basis, auf welcher die Eignung synthetischer Daten für Anwendungszwecke solcher Art diskutiert werden kann.

Methodisch orientiert sich die Evaluation am im Kapitel 3.3.4 dargestellten Design. Zunächst werden die Modelle auf dem Test-Split des synthetischen Datensatzes evaluiert, um einen ersten Eindruck der Leistungs- und Generalisierungsfähigkeit zu erlangen. Gleichzeitig dient dieser Schritt der Validierung der Pipeline zur Datengenerierung, da hierdurch Konsistenz und Eignung synthetischer Bilddaten für das Training von KI-Modellen untersucht werden kann.

Ergänzend erfolgt eine Evaluation auf einem realen Testdatensatz aus 34 Bildern, welcher wie in Kapitel 4.4.4 beschrieben zusammengestellt und annotiert wurde. Diese Untersuchung dient primär der Analyse der Eignung von auf synthetischen Daten trainierten Modellen in realen Anwendungsumgebungen sowie der Betrachtung des Einflusses des *Domain Gaps*. Auf dieser Grundlage lässt sich auch der Bedarf an zukünftiger Forschung in diesem Themengebiet ableiten.

5.1.1 Evaluation der YOLO-Modelle

Zuerst wurden die drei YOLO-Modelle auf dem synthetischen Datensatz evaluiert. Der Verlauf zentraler Metriken während des Trainings ist exemplarisch für das YOLO-Medium Modell in Abbildung 6 dargestellt. Es ist zu erkennen, dass die Modelle bereits nach wenigen Epochen eine gute Leistung auf dem Validierungsdatensatz erzielen konnten und schlussendlich eine Konvergenz zu erkennen ist. Ein ähnlicher Verlauf zeigte sich auch bei den beiden anderen Modellvarianten.[140]

Wie in Tabelle 1 dargestellt, zeigten alle drei Modelle eine konsistente Leistung auf dem synthetischen Test-Split. Das YOLO-Small Modell erzielte dabei die beste Leistung mit einer mAP@50 von 0.9080 und einer mAP@[.5:.95] von 0.6605. Auch beim Sensitivität Wert erreichte dieses Modell mit 0.8094 den höchsten Wert. Bei allen drei Modellen lag die Präzision über 0.9, was auf eine geringe Rate an FP hinweist. Es ist jedoch zu beachten, dass alle Modelle bei allen Metriken nahe beieinander liegen und keine signifikanten Unterschiede zwischen dem kleinsten und dem größten Modell erkennbar waren.

[139] Sprachlich geglättet durch ChatGPT-5
[140] Vgl. Anhang 6

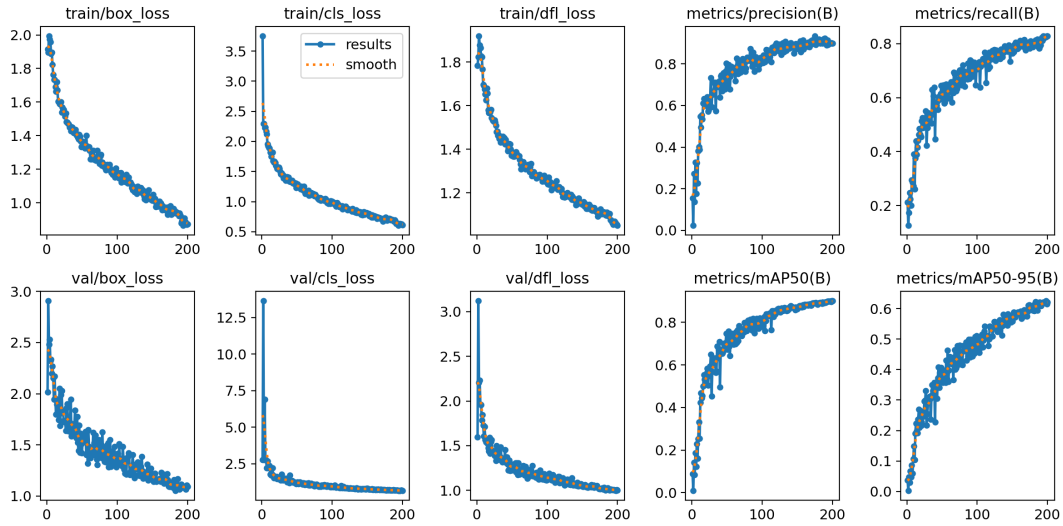
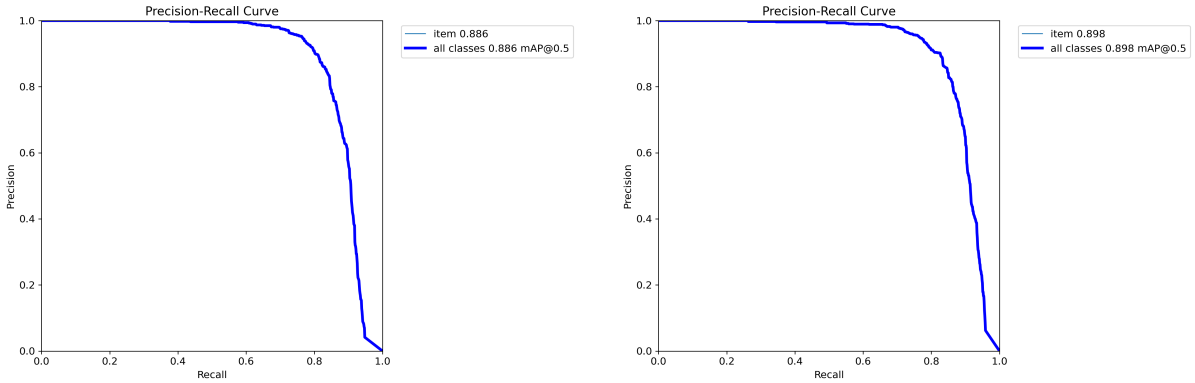


Abb. 6: Trainingsverlauf des YOLO-Modells Medium

Abbildung 7 zeigt ergänzend die Präzisions-Sensitivitäts-Kurven des größten und kleinsten trainierten Modells. Beide Kurven zeigen einen ähnlichen Verlauf, wobei das YOLO-Medium Modell etwas höhere Präzisions-Werte bei höherer Sensitivät erreichte. Insgesamt zeigen alle Modelle jedoch keine signifikanten Unterschiede,[141]sondern vergleichbar gute Ergebnisse auch bei hohen Sensitivitätswerten. Zusammenfassend lässt sich festhalten, dass alle drei Modelle auf dem synthetischen Datensatz eine gute Leistung erbringen konnten, was die Konsistenz und Eignung der mittels der Pipeline generierten synthetischen Daten für das Training von KI-Modellen unterstreicht.



(a) YOLO-Modell Nano

(b) YOLO-Modell Medium

Abb. 7: Präzision-Sensitivität-Kurven der YOLO-Modelle Nano und Medium

Auf dem realen Datensatz hingegen zeigten alle drei YOLO-Modelle eine deutlich reduzierte Leistung. Das YOLO-Medium-Modell erzielte mit einer mAP@50 von 0.0712 die beste Leistung, kann jedoch mit der erbrachten Leistung nicht als zufriedenstellend angesehen werden. Die anderen beiden Modelle wiesen einen noch stärkeren Leistungsabfall auf. Alle Modelle hatten hier

[141]Vgl. Anhang Anhang 6/3

deutlich geringe Präzisions- und Sensitivität-Werte, was auf eine höhere Anzahl an FP und FN schließen lässt. Dies verdeutlicht den Einfluss des *Domain Gaps*, da die Modelle, obwohl sie auf synthetischen Daten gute Ergebnisse lieferten, Schwierigkeiten hatten, ihre Leistung auf reale Anwendungsfälle zu übertragen und zeigten aufgrund der geringen Anzahl an TP keine Konsistenz in der korrekten Erkennung von Fehlerfällen.

Modell	Datensatz	Augmentierung	mAP@50	mAP@[.5:.95]	Präzision	Sensitivität
YOLO-n	synthetisch	ja	0.8872	0.6628	0.9368	0.7510
YOLO-s	synthetisch	ja	0.9176	0.6673	0.9156	0.8250
YOLO-m	synthetisch	ja	0.8957	0.6607	0.9153	0.7996
RT-DETR-v2-r18	synthetisch	ja	0.2790	0.1892	0.1966	0.6335
RT-DETR-v2-r18	synthetisch	nein	0.5160	0.3913	0.1634	0.8070
YOLO-n	real	ja	0.0161	0.0062	0.0425	0.0667
YOLO-s	real	ja	0.0115	0.0028	0.0240	0.1000
YOLO-m	real	ja	0.0712	0.0402	0.1087	0.1000
RT-DETR-v2-r18	real	ja	0.0050	0.0020	0.0066	0.4000
RT-DETR-v2-r18	real	nein	0.0052	0.0016	0.0066	0.4667

Tab. 1: Ergebnisse zentraler Metriken der KI-Modelle mit *Confidence Threshold 0.1*

5.1.2 Evaluation des RT-DETR Modells

Betrachtet man die Ergebnisse der beiden RT-DETR-r18-Modelle, so zeigt sich ein deutlich anderes Bild als bei den YOLO-Modellen. Trotz der schnellen Konvergenz erzielten beide Modelle bereits auf dem synthetischen Test-Split eine signifkant schlechtere Leistung als alle drei CNN-basierten Modelle. Auffällig ist dabei, dass die bessere Leistung das Modell, welches ohne Datenaugmentierung trainiert wurde, erzielen konnte. Dieses erreichte jedoch dennoch nur eine mAP@50 von 0.5160 und eine mAP@[.5:.95] von 0.3913. Besonders auffällig sind die sehr niedrigen Präzisions-Werte bei beiden Trainingsansätzen, was auf eine hohe Rate an FP hinweist, während beim Sensitivität mit 0.8070 beim Training ohne Datenaugmentation ein vergleichsweiser akzeptabler Wert erreicht werden konnten. Dies deutet darauf hin, dass das Modell zwar einige der tatsächlichen Fehlerfälle erkennt, jedoch auch viele FP generiert. Angesichts dieser hohen Zahl an FP ist keine Konsistenz in der korrekten Erkennung ersichtlich und es könnte sich auch um zufällige Treffer handeln.

Bei der Evaluation auf dem realen Datensatz verschlechterte sich die Leistung beider Modelle weiter. Mit mAP@50 und mAP@[.5:.95] Werten nahe 0 zeigten die Modelle eine nahezu unbrauchbare Leistung. Sowohl Präzisions als auch Sensitivitäts-Werte fielen im Vergleich zu dem synthetischen Datensatz nochmals deutlich ab. Daraus ist abzuleiten, dass die Modelle zwar viele Elemente des Bildes als Fehler klassifizieren, jedoch kaum korrekte Vorhersagen treffen und tatsächliche Fehler übersehen. Betrachtet man die in Abbildung 8 dargestellte visualisierte Vorhersage des Modells mit Datenaugmentierung, wird dieser Verdacht bestätigt. Es sind zahlreiche

Bounding Boxes zu erkennen, welche jedoch kaum mit den tatsächlichen Fehlern übereinstimmen. Damit kann zusammengefasst werden, dass die RT-DETR-Modelle sowohl auf synthetischen als auch auf realen Daten eine deutlich schlechtere Leistung erbrachten als die YOLO-Modelle und auf keinem der Datensätze eine zufriedenstellende Leistung aufweisen konnten.

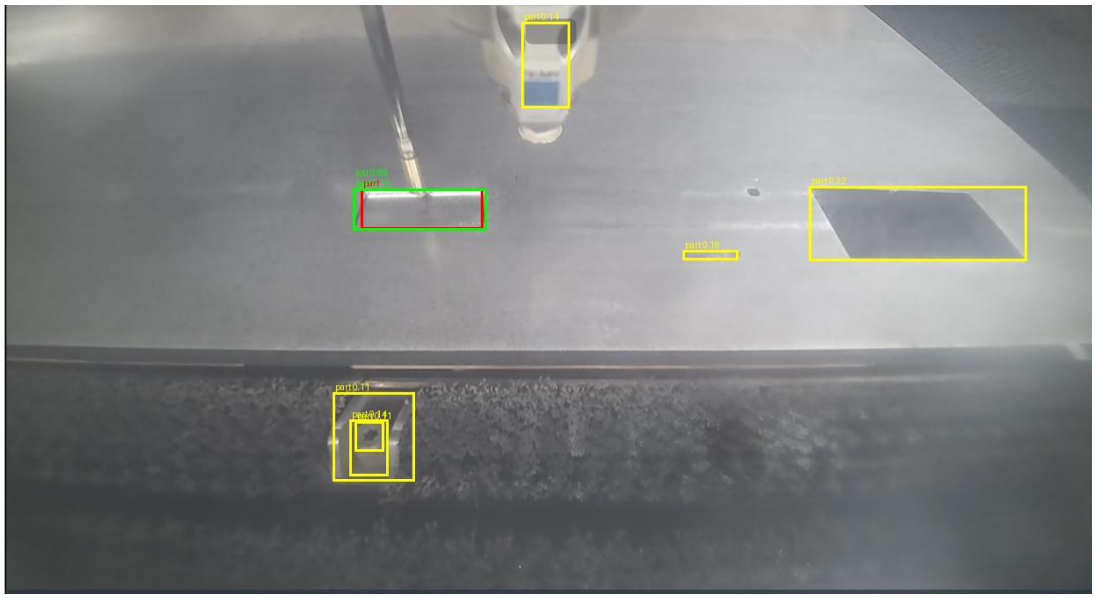


Abb. 8: Visualisierung einer Vorhersage des RT-DETR-Modells mit Datenaugmentierung auf einem realen Bild (*Confidence Threshold 0.1*)

5.1.3 Zusammenfassung der Evaluationsergebnisse

Es lässt sich zusammenfassen, dass die YOLO-Modelle auf dem synthetischen Datensatz eine gute Leistung erbringen konnten, wobei alle drei Modelle in ihrer Leistung keine signifikanten Unterschiede aufwiesen. Das transformerbasierte Modell hingegen zeigte sowohl auf dem synthetischen als auch auf dem realen Datensatz eine deutlich schlechtere Leistung, trotz dass beide Varianten während des Trainings überraschend schnell konvergierten. Alle trainierten KI-Modelle zeigten Schwierigkeiten bei der Übertragung ihrer Leistung auf realen Anwendungsfälle, was den Einfluss des *Domain Gaps* verdeutlicht. Das YOLO-Medium-Modell konnte dabei noch die beste Leistung auf dem realen Datensatz erzielen, erbrachte jedoch dennoch keine zufriedenstellende Leistung.

6 Kritische Reflexion und Ausblick[142]

6.1 Auftrag der Arbeit

Das zentrale Ziel dieser Arbeit war die Entwicklung einer Pipeline zur automatisierten Generierung synthetischer Bilddaten für das Training von KI-Modellen zur Fehlererkennung bei Werkzeugmaschinen. Dieses Ziel ergab sich aus der in der Forschungsliteratur beschriebenen Problematik, dass insbesondere im industriellen Kontext häufig nur unzureichende oder einseitige reale Daten für das Training von KI-Modellen zur Verfügung stehen.[143]

Um diesen Datenmangel zu adressieren, wurde eine Datenpipeline konzipiert, welche eine einfache und flexible Generierung umfangreicher, ausgeglichener und realistischer synthetischer Datensätze ermöglicht. Darüber hinaus sollten verschiedene KI-Modelle auf diesen generierten Daten trainiert werden, um die Eignung synthetischer Daten zu validieren.

Im praktischen Teil der Arbeit wurde daher eine Simulationsumgebung in *NVIDIA Omniverse* aufgebaut und eine Pipeline zur automatisierten Generierung synthetischer Bilddaten implementiert, bei welcher zentrale Parameter hinsichtlich der Bildqualität, der Anzahl der Bilder und der Simulationsumgebung einfach modifiziert werden konnten. Diese Daten wurden in ein für das Modelltraining geeignetes Format überführt und die Eignung und Qualität der generierten Daten durch ein Training von drei Varianten der YOLO-Architektur sowie ein transformerbasiertes Modell validiert. Als methodische Grundlage diente dabei das DSR-Paradigma, welches es ermöglichte, das entwickelte Artefakt in einem iterativen Prozess zu evaluieren und zu verbessern.

6.2 Kritische Reflexion der Methodik und Ergebnisse

Die Ergebnisse der Evaluation zeigen, dass die entwickelte Pipeline erfolgreich synthetische Bilddaten generieren konnte, welche konsistent waren und sich für das Training von KI-Modellen eigneten. Alle trainierten YOLO-Modelle konnten auf dem synthetischen Testdatensatz eine gute Leistung erbringen und auch ihr Trainingsverlauf sah vielversprechend aus. Zwar zeigte die transformerbasierten Modelle hier deutlich schlechtere Ergebnisse, dies lässt sich jedoch nicht unmittelbar auf die Qualität der generierten Daten zurückführen, da die CNN-basierten Modelle konsistente Resultate erbrachten. Damit kann die Forschungsfrage der Arbeit, wie synthetische Daten für das Training von KI-Modellen zur Fehlererkennung bei Werkzeugmaschinen genutzt werden können und inwieweit die daraus generierten synthetischen Daten für den Einsatz in realen Anwendungsfällen geeignet sind, grundsätzlich beantwortet werden.

[142] Sprachlich geglättet durch ChatGPT-5
[143] Vgl. Kapitel 1.1

Der fehlende Erfolg beim Transfer auf reale Daten kann verschiedene Gründe haben und ist nicht zwangsläufig auf die Qualität oder den mangelnden Realismus der generierten Daten zurückzuführen. Wie schon in Tabelle 1 zu sehen ist, erzielte das größte trainierte YOLO-Modell auf dem realen Datensatz bessere Ergebnisse als kleinere Modelle. Dies legt nahe, dass die Modellgröße ein entscheidender Faktor für die Übertragbarkeit sein kann. Insbesondere bei transformerbasierten Modellen spielt außerdem die Größe und Vielfalt des Datensatzes eine zentrale Rolle, da ihre Leistung in besonderem Maße von umfangreichen Trainingsdaten abhängt.[144] Der Schwerpunkt dieser Arbeit lag außerdem nicht auf der Optimierung der Modelleistung, beispielsweise durch die Ermittlung optimaler Hyperparameter, sondern auf der Konzeption und Umsetzung der Pipeline zur Generierung synthetischer Daten.

Es ergaben sich im Verlauf der Umsetzung und Evaluation des weiteren diverse Herausforderungen und Limitationen, welche den Umfang und die Qualität der generierten Daten sowie die Leistung der KI-Modelle beeinflusst haben:

- **Rechenleistung und Ressourcen**: Das Training stieß insbesondere bei den größeren Modellen auf Hardware- und Zeitbeschränkungen. Die begrenzte Rechenleistung wirkte sich sowohl auf den Realismus als auch auf den Umfang der generierten Daten aus. Auch die Auswahl der Größe der KI-Modelle wurde durch die verfügbare Hardware limitiert, was insbesondere die Übertragbarkeit auf reale Daten beeinträchtigt haben könnte.
- **Umfang und Qualität der Daten**: Der synthetische Datensatz umfasste lediglich eine begrenzte Anzahl von Fehlerfällen, Perspektiven und Variationen. Die Größe des Datensatzes war mit 1.077 Bildern relativ klein, was die Generalisierungsfähigkeit und Leistung der Modelle eingeschränkt haben könnte.
- **Fehlerquellen in der Datengenerierung**: Unerwartete Fehler wie etwa ungewollte Objektrotationen oder fehlerhafte Annotationen durch den *Omniverse Replicator* führten wiederholt zu Verzögerungen.
- **Modellarchitekturen**: Während die YOLO-Modelle auf synthetischen Daten konsistente Ergebnisse lieferten, zeigte das transformerbasierte RT-DETR-Modell eine deutlich geringere Leistung. Ursachen hierfür können im kleinen Datensatz, in einer unzureichenden oder fehlerhaften Datenaugmentierung sowie in fehlender Expertise bei der Konfiguration von Transformermodellen und deren Hyperparametern liegen.

Das DSR-Paradigma erwies sich als geeignete Methodik, um die Entwicklung der Pipeline zu strukturieren und iterative Verbesserungen basierend auf den Evaluationsergebnissen vorzunehmen. Es konnten somit Fehler frühzeitig erkannt und korrigiert sowie die Modelleistungen und Datenqualität stetig verbessert werden. Schlussendlich wäre jedoch eine intensivere Auseinandersetzung mit verschiedenen KI-Architekturen und deren konkreten Anforderungen an Trainingsdaten, Datenaugmentation und Hyperparameter-Konfigurationen für die Evaluation der

[144] Vgl. Jamil, Piran, Kwon 2022, S. 8; Berroukham, Housni, Lahraichi 2023, S. 209

Datenpipeline hilfreich gewesen, da aktuell zahlreiche Parameter Einfluss auf die erzielten Ergebnisse haben könnten. Dies war jedoch aufgrund zeitlicher Limitationen nur eingeschränkt möglich. Auch die Anzahl der iterativen Trainings- und Evaluationsdurchläufe wurde durch die zeitlichen Beschränkungen limitiert. Weitere Iterationen hätten die Qualität der Daten sowie die Leistung der Modelle weiter verbessern können.

6.3 Implikationen für die Forschung

Die Ergebnisse machen deutlich, dass der Einsatz synthetischer Daten ein hohes Potenzial bietet, jedoch weitere Forschung notwendig ist, um die Übertragbarkeit auf reale Anwendungsfälle zu verbessern:

- **Effizientere Simulationssoftware**: Die eingesetzte Software *NVIDIA Omniverse* bietet zwar eine hohe Flexibilität und einen hohen möglichen Realismus, ist jedoch sehr ressourcenintensiv.[145] Alternativ könnte die Nutzung von effizienteren Simulationssoftware wie beispielsweise *Genesis*[146] untersucht werden, die speziell auf Effizienz und geringere Hardwareanforderungen ausgelegt ist.
- **Größere und realistischere Datensätze**: Für eine bessere Generalisierungsfähigkeit sollten mehr Daten mit höherer Variabilität und höhere Qualität generiert werden. Hierzu zählen zusätzliche Fehlerklassen, variierende Perspektiven sowie realitätsnähere Darstellungen, sodass ein größerer und umfangreicherer Datensatz für das Training der KI-Modelle zur Verfügung steht.
- **Leistungsfähigere Modelle**: Die erzielten Ergebnisse deuten darauf hin, dass größere Modelle besser geeignet sein könnten, um den *Domain Gap* zu überbrücken. Dies setzt jedoch eine deutlich höhere Rechenleistung sowie die Möglichkeit für längere Trainingszeiten voraus.
- **Diffusionsmodelle**: Um den Realismus der synthetischen Bilddaten zu erhöhen, ohne die Hardwareanforderungen unverhältnismäßig zu steigern, könnte der Einsatz von Diffusionsmodellen eine vielversprechende Möglichkeit darstellen. Hierbei werden synthetisch generierte Bilddaten genutzt und anschließend Oberflächen und Materialien durch den Einsatz eines Diffusionsmodells realistischer gestaltet.[147]
- **Hybrides Training**: Eine vielversprechende Möglichkeit besteht darin, synthetische und reale Daten kombiniert einzusetzen. Dabei wird das Modell auf synthetischen Daten trainiert und anschließend ein *Fine-Tuning* auf realen Daten durchgeführt. Ein solches hybrides Training könnte die Generalisierungsfähigkeit der Modelle verbessern und den *Domain Gap* verringern.[148]

[145] Vgl. *Isaac Sim Requirements — Isaac Sim Documentation* 2025
[146] Vgl. *Genesis-Embodied-AI/Genesis* 2025
[147] Vgl. Hadadan et al. 2025
[148] Vgl. Zaripov, Kulshin, Sidorov 2025, S. 17

- **Architekturen**: Die Eignung weiterer Architekturen, wie beispielsweise die eines Hybrid Vision Transformer (HVT), die Elemente von CNN- und Transformer-Ansätzen kombiniert, könnte untersucht werden.[149]

Es lässt sich insgesamt feststellen, dass insbesondere in Hinblick auf das Remote Monitoring von Werkzeugmaschinen, der Einsatz von synthetischen Daten eine vielversprechende Möglichkeit und zugleich eine große Chance darstellt. Durch weitere Forschung in diesem Bereich kann die Übertragbarkeit auf reale Anwendungsfälle verbessert werden und die Nutzung synthetischer Daten einen maßgeblichen Beitrag zur automatisierten Fehlererkennung und -klassifizierung leisten, was insbesondere bei der Fernüberwachung von Maschinen von hoher Relevanz ist.

6.4 Einordnung und Ausblick

Trotz der begrenzten Übertragbarkeit auf reale Daten wurde das zentrale Ziel dieser Arbeit erreicht: Die entwickelte Pipeline konnte erfolgreich synthetische Bilddaten inklusive Annotationen generieren und für das Training von KI-Modellen aufbereiten. Die gute Performance der YOLO-Modelle auf dem synthetischen Datensatz unterstreicht die Konsistenz und Eignung der generierten Daten für das Training solcher Modelle. Die in Kapitel 1.1 gestellte Forschungsfrage konnte durch das entwickelte Artefakt, dessen Evaluation sowie kritische Diskussion beantwortet werden.

Die schwache Performance auf realen Daten verdeutlicht jedoch die Auswirkungen des bestehenden *Domain Gaps* und zeigt somit auf konkrete Handlungsfelder für die weitere Forschung auf. Dazu zählen der Einsatz effizienterer Simulationssoftware, eine stärkere Recheninfrastruktur, alternative generative Verfahren (z.B. Diffusionsmodelle) sowie die Untersuchung weiterer Modellarchitekturen und Trainingsansätze, um die Übertragbarkeit von synthetischen Daten auf reale Anwendungsfälle zu verbessern und den *Domain Gap* zu schließen.

Insgesamt bestätigt die Arbeit, dass synthetische Daten ein wertvolles Werkzeug für das Training von KI-Modellen darstellen. Ihre erfolgreiche Integration in der industriellen Praxis erfordert jedoch zusätzliche Investitionen in Datenqualität, Modellgröße und Trainingsstrategien.

[149] Vgl. Khan et al. 2023, S. 4 f.

Anhang

Anhangverzeichnis

Anhang 1: Generierung synthetische Daten mithilfe des *Omniverse Replicators* Quellcode

150

```
from omni.isaac.kit import SimulationApp
import argparse
import sys

OUTPUT_DIR_PATH = ""
USD_FILE_PATH = ""
USED_CAMERA_INDEX = 10  # da mehrere Kameras in der Szene implementiert sind, spezifizieren, welche Kamera verwendet werden soll

# ----- Argumente & Konfiguration -----
parser = argparse.ArgumentParser("Dataset generator")
parser.add_argument("--headless", type=bool, default=False)
parser.add_argument("--height", type=int, default=720)
parser.add_argument("--width", type=int, default=1280)
parser.add_argument("--num_frames", type=int, default=75)
parser.add_argument("--output-dir", type=str, default=OUTPUT_DIR_PATH)
args, _ = parser.parse_known_args()

# ----- Konfiguration der Bildqualität und Anzahl -----
CONFIG = {
    "renderer": "RayTracedLighting",
    "headless": args.headless,
    "width": args.width,
    "height": args.height,
    "num_frames": args.num_frames,
    "samples_per_pixel_per_frame": 256,
    "max_bounces": 8,
    "max_specular_transmission_bounces": 12,
    "max_volume_bounces": 4,
    "subdiv_refinement_level": 2,
    "anti_aliasing": 5
}

# ----- Simulation starten -----
simulation_app = SimulationApp(launch_config=CONFIG)

import omni.replicator.core as rep
import omni.usd

```

[150] Erstellung mit Unterstützung von ChatGPT-5 und Claude Sonnet 4

```
# ----- Replicator-Einstellungen -----
import carb.settings
carb.settings.get_settings().set(
    "/exts/omni.replicator.core/maxAssetLoadingTime",
    600.0
)
# ----- Stage laden -----
usd_path = USD_FILE_PATH
omni.usd.get_context().open_stage(usd_path, None)

# ----- Replizierbare Objekte erstellen -----
k09_machine = rep.create.from_usd(usd_path, semantics=[("class", "k09")])
rep.create.group([k09_machine])

with k09_machine:
    rep.modify.pose(
        position=(0, 0, 0),
        rotation=(0, 0, 0),
        scale=(1.0, 1.0, 1.0)
    )

# ----- Kamera und Materialien finden -----
stage = omni.usd.get_context().get_stage()
camera_paths = [str(prim.GetPath()) for prim in stage.Traverse() if prim.GetTypeName() == "Camera"]

all_material_paths = [
    str(prim.GetPath())
    for prim in stage.Traverse()
    if prim.GetTypeName() == "Material"
       and str(prim.GetPath()).startswith("/World/Looks/")
]

camera_index = USED_CAMERA_INDEX

print(camera_paths[camera_index])

cam = rep.get.prims(path_pattern=camera_paths[camera_index])

render_product = rep.create.render_product(cam, (args.width, args.height))

# ----- Writer -----
writer = rep.WriterRegistry.get("KittiWriter")
```

```
writer.initialize(output_dir=args.output_dir,
                  omit_semantic_type=True)
writer.attach(render_product)

with rep.get.prims(
    path_pattern="/World/Sheets/States_Sheets/JT45__2__unload_84_sheet/
       JT45__2__unload_84_sheet_temp_stl",
    prim_types=["Xform"]
):
    rep.modify.semantics(
        semantics=[("class", "sheet")]
    )

with rep.get.prims(
    path_pattern="/World/Sheets/States_Parts/*/*",
    prim_types=["Xform"]
):
    rep.modify.semantics(
        semantics=[("class", "part")]
    )


import random

def run_orchestrator():
    print("Starting Orchestrator")
    rep.orchestrator.run()

    # Warten bis App gestartet ist
    while not rep.orchestrator.get_is_started():
        simulation_app.update()

    # Frame-Schleife: Für jeden Frame Elemente randomisieren
    for frame in range(args.num_frames):
        random_material_path1 = random.choice(all_material_paths)
        random_material_path2 = random.choice(all_material_paths)
        random_material = rep.get.prims(path_pattern=
           random_material_path1, prim_types=["Material"])
        random_material_distractor = rep.get.prims(path_pattern=
           random_material_path2, prim_types=["Material"])

        # Randomisierung Lichter
        with rep.get.prims(path_pattern="RectLight"):
            rep.modify.attribute("color", rep.distribution.uniform((0.3,
                0.3, 0.3), (1, 1, 1)))
```

```
122             rep.modify.attribute("intensity", rep.distribution.normal
                    (100000.0, 400000.0))
123             rep.modify.visibility(rep.distribution.choice([True, False,
                    False, False, False]))
124
125         # Randomisierung Parts
126         with rep.get.prims(
127             path_pattern="/World/Sheets/States_Parts/*/*",
128             prim_types=["Xform"]
129         ):
130             rep.modify.pose(rotation=(random.uniform(-4, 4), random.
                    uniform(-2, 2), 0))
131             rep.modify.visibility(rep.distribution.choice([True, True,
                    False]))
132             rep.modify.material(random_material)
133
134         # Randomisierung Material des Blechs
135         with rep.get.prims(
136             path_pattern="/World/Sheets/States_Sheets/*/*",  # Sheet
137             prim_types=["Xform"]
138         ):
139             rep.modify.material(random_material)
140
141         # Randomisierung Störobjekte
142         with rep.get.prims(
143             path_pattern="/World/Sheets/Distractors/",
144             prim_types=["Mesh"]
145         ):
146             rep.modify.pose(position=(random.uniform(-150, 50), random.
                    uniform(-680, -570), random.uniform(140, 210)))
147             rep.modify.material(random_material_distractor)
148
149         # Randomisierung Kamera
150         with rep.get.prims(path_pattern=camera_paths[camera_index]):
151             rep.modify.pose(position=(random.uniform(265, 440), random.
                    uniform(130, 245), random.uniform(190, 280)))
152
153         # Frame rendern
154         rep.orchestrator.step(rt_subframes=8, pause_timeline=True,
                delta_time=0.0)
155
156     rep.BackendDispatch.wait_until_done()
157     rep.orchestrator.stop()
158
159 run_orchestrator()
```

```
simulation_app.update()
```

Anhang 2: Erstellung des Datensatzes für die Modelltrainings Quellcode

```
# Klassen zu Id's
map_classes = {
    "part": 0,
}


# Einstellungen und Variablen festlegen
import os

OUTPUT_FOLDER = ""
CAMERA = ""
DATASET_NAME = ""
OUTPUT_DIR = ""
height, width = 720, 1280
SPLIT_RATIO = [0.7, 0.2, 0.1] # train, val, test

all_folders = os.listdir(OUTPUT_FOLDER)
annotation_paths = [f"annot_path/{folder}/Replicator/{camera}/
    object_detection" for folder in all_folders]
yolo_rgb_paths = [f"annot_path/{folder}/Replicator/{camera}/rgb" for
    folder in all_folders]



# .txt Dateien aufräumen und in das YOLO-Format übertragen
# .txt Dateien müssen aufgeräumt werden, da es Fehler bei den Annotationen durch den
    Replicator gibt
from collections import Counter
import shutil

os.makedirs(OUTPUT_DIR, exist_ok=True)
if os.path.exists("yolo_annotated"):
    shutil.rmtree("yolo_annotated")
os.makedirs("yolo_annotated", exist_ok=True)

for idx, annot_path in enumerate(annotation_paths):
    for file in os.listdir(annot_path):
        if not file.endswith(".txt"):
```

```
36              continue
37
38          file_path = os.path.join(annotation_paths[idx], file)
39          with open(file_path, "r") as f:
40              raw_lines = [ln.strip() for ln in f if ln.strip()]
41
42          counts = Counter(raw_lines)  # Zählen, wie oft jede Zeile in der Datei
                vorkommt
43
44          output_lines = set()
45          for line in raw_lines:
46              if counts[line] <= 1:
47                  continue    # Zeilen, welche nur einmal vorkommen, werden
                        gefiltert
48
49              items = line.split(" ")
50              items[0] = items[0].replace("k09,", "").replace("k09", "").
                    replace("sheet", "")
51
52              if items[0] in list(map_classes.keys()):
53                  [class_id, x_min, y_min, x_max, y_max] = [map_classes[
                        items[0]], int(items[4]), int(items[5]), int(items
                        [6]), int(items[7])]
54
55                  x_center = ((x_min) + (x_max)) / 2 / width
56                  y_center = ((y_min) + (y_max)) / 2 / height
57                  item_width = ((x_max) - (x_min)) / width
58                  item_height = ((y_max) - (y_min)) / height
59
60                  output_data = " ".join([str(class_id), str(x_center),
                        str(y_center), str(item_width), str(item_height)])
61
62                  output_lines.add(output_data + "\n")
63
64          file_path = os.path.join(OUTPUT_DIR, "data_" + str(idx))
65          os.makedirs(file_path, exist_ok=True)
66          with open(os.path.join(file_path, file), "w") as out_file:
67              out_file.writelines(list(output_lines))
68
69
70 # Aufteilen in ein Trainings-, Validierungs- und Testdatensatz
71 import random
72
73 for i, yolo_rgb_path in enumerate(yolo_rgb_paths):
74     yolo_annoted_path = os.path.join(OUTPUT_DIR, f"data_{i}")
```

```
75
76      all_images = [f for f in os.listdir(yolo_rgb_path) if f.endswith(".
            png")]
77      all_labels = [f for f in os.listdir(yolo_annoted_path) if f.endswith
            (".txt")]
78
79      if len(all_images) != len(all_labels):
80          print(f"Warning: Mismatch in number of images and labels in run
                {i}. Images: {len(all_images)}, Labels: {len(all_labels)}")
81
82      # shuffle the images with their corresponding annotations
83      pairs = [(img, label) for (img, label) in zip(all_images, all_labels
            )]
84      random.shuffle(pairs)
85
86      print(pairs)
87
88      n = len(all_images)
89
90      train, val = int(n * SPLIT_RATIO[0]), int(n * (SPLIT_RATIO[0] +
            SPLIT_RATIO[1]))
91
92      splits = {
93          "train": pairs[:train],
94          "val": pairs[train:val],
95          "test": pairs[val:]
96      }
97
98      for split, files in splits.items():
99          os.makedirs(f"{DATASET_NAME}/images/{split}", exist_ok=True)
100         os.makedirs(f"{DATASET_NAME}/labels/{split}", exist_ok=True)
101
102         for img_file, label_file in files:
103             shutil.copy(f"{yolo_rgb_path}/{img_file}", f"{DATASET_NAME}/
                    images/{split}/{i}_{img_file}")
104             shutil.copy(f"{yolo_annoted_path}/{label_file}", f"{
                    DATASET_NAME}/labels/{split}/{i}_{label_file}")
105
106
107 # .yaml Datei erstellen
108 import yaml
109
110 class_names = list(map_classes.keys())
111
112 data = {
```

```
    'path': f'./{DATASET_NAME}',
    'train': 'images/train',
    'val': 'images/val',
    'test': 'images/test',
    'nc': len(class_names),
    'names': class_names
}

with open(f'{DATASET_NAME}/data.yaml', 'w') as f:
    yaml.dump(data, f, default_flow_style=False)
```

Anhang 3: Training der YOLO Modelle Quellcode

```
from ultralytics import YOLO

yolo_model_name = 'yolo11n.pt' # yolo11n.pt, yolo11s.pt, yolo11m.pt
model = YOLO(yolo_model_name)
dataset_name = "dataset_random_1077"
output_dir_name = "1 - exp_200_1077_imgs_n"


model.train(
    data=f"{dataset_name}/data.yaml",
    epochs=200,
    patience=20,
    imgsz=640,
    batch=16,
    project="runs/train",
    name=output_dir_name,
    workers=4,
    verbose=True,
    device=0,

    degrees=40,      # rotation random between -180 and 180 degrees
    hsv_h=0.5,       # hue
    hsv_s=0.8,       # saturation
    hsv_v=0.6,       # brightness
    translate=0.2,   # translates images
    scale=0.4,       # scale between 0.6 and 1.4
    shear=6,         # shear along x and y axis
    lr0=0.01,        # learning rate at the beginning (default 0.01)
    perspective=0.0005, # simulates different perspective
    flipud=0.5,      # 50% chance to flip the image vertically
    fliplr=0.5,      # 50% chance to flip the image horizontally
    bgr=0.2,         # 20% chance to flip color channels from rgb to bgr
```

```
    erasing=0.3,     # 30% chance that a certain part of the image is erased
    weight_decay=0.003, # regularization to avoid overfitting
    dropout=0.1,    # 10% of activations are randomly set to zero during training

    single_cls=True,
    classes=[0],
    mosaic = 0.1,
    mixup = 0.0,
    cutmix = 0.2,
    copy_paste = 0.0
)
```

Anhang 4: Training of RT-DETR Model Source Code

```
# Einstellungen des Trainings
from pathlib import Path

checkpoint = "PekingU/rtdetr_v2_r18vd"
image_size = 640
DATASET_PATH = ""
OUTPUT_DIR = "rtdetr-v2-r18-finetune-1"

# Prozessor laden
from transformers import AutoImageProcessor

image_processor = AutoImageProcessor.from_pretrained(
    checkpoint,
    do_resize=True,
    size={"width": image_size, "height": image_size},
    use_fast=True,
)


# Data Augmentation für das Training
import albumentations as A

train_augmentation_and_transform = A.Compose(
    [
        A.Perspective(p=0.1),
        A.HorizontalFlip(p=0.5),
        A.RandomBrightnessContrast(p=0.5),
        A.HueSaturationValue(p=0.1),
        A.Affine(scale=(0.9, 1.1), translate_percent=(0.05, 0.05),
            rotate=(-5, 5), shear=(-5, 5), p=0.7),
        A.OneOf([A.MotionBlur(blur_limit=3, p=1.0), A.GaussianBlur(
```

```
            blur_limit=3, p=1.0), A.GaussNoise(var_limit=(5.0, 30.0), p
            =1.0)], p=0.2),
31          A.RandomFog(p=0.1),
32          A.ColorJitter(p=0.1),
33          A.ToGray(p=0.1),
34          A.RandomGamma(gamma_limit=(80,120), p=0.1)
35      ],
36      bbox_params=A.BboxParams(format="coco", label_fields=["category"],
            clip=True, min_area=25, min_width=1, min_height=1),
37  )
38
39  # keine Augmentation für die Validierung
40  validation_transform = A.Compose(
41      [A.NoOp()],
42      bbox_params=A.BboxParams(format="coco", label_fields=["category"],
            clip=True, min_area=1, min_width=1, min_height=1),
43  )
44
45
46  # Dataset-Klasse für das Laden der Bilder und Labels für Trainings-, Validierungs- und
        Testdatensatz
47  from torch.utils.data import Dataset
48
49  class ImageDataset(Dataset):
50      def __init__(self, img_dir, lbl_dir, image_processor, transform=None
            ):
51          self.img_dir = Path(img_dir)
52          self.lbl_dir = Path(lbl_dir)
53          self.image_processor = image_processor
54          self.transform = transform
55
56          self.image_files = [f for f in sorted(img_dir.iterdir()) if f.
                suffix.lower() in {".jpg", ".jpeg", ".png", ".bmp", ".webp"}]
57          self.label_files = [f for f in sorted(lbl_dir.iterdir()) if f.
                suffix.lower() in {".txt"}]
58
59          self.images = [Image.open(self.image_files[element_idx]).convert
                ("RGB") for element_idx in range(len(self.image_files))]
60
61          print(f"Dataset: {len(self.image_files)} Bilder gefunden")
62
63      def __len__(self):
64          return len(self.image_files)
65
66      def __getitem__(self, idx):
```

```
67          image = self.images[idx]
68          W, H = image.size
69
70          label_path = self.label_files[idx]
71
72          # Labels laden und konvertieren
73          boxes, categories = read_yolo_to_coco(label_path, W, H)
74
75          # Zu numpy für Augmentationen
76          image_array = np.array(image)
77
78          # Augmentationen anwenden
79          if self.transform and len(boxes) > 0:
80              try:
81                  transformed = self.transform(
82                      image=image_array,
83                      bboxes=boxes,
84                      category=categories
85                  )
86                  image_array = transformed["image"]
87                  boxes = transformed["bboxes"]
88                  categories = transformed["category"]
89              except Exception as e:
90                  print(f"Augmentation failed for {label_path.name}: {e}")
91                  # Fallback: Original verwenden
92
93          # Format für image processor
94          formatted_annotations = {
95              "image_id": idx,
96              "annotations": [
97                  {
98                      "image_id": idx,
99                      "category_id": cat,
100                     "bbox": list(box),
101                     "iscrowd": 0,
102                     "area": box[2] * box[3],
103                 } for cat, box in zip(categories, boxes)
104             ]
105         }
106
107         # Image processor anwenden
108         result = self.image_processor(
109             images=image_array,
110             annotations=formatted_annotations,
111             return_tensors="pt"
```

```
        )

        # Batch-Dimension entfernen
        return {
            "pixel_values": result["pixel_values"].squeeze(0),
            "labels": result["labels"][0],
        }

# Ersetze die kaputte ImagesDataset:
train_dataset = ImageDataset(
    img_dir=DATASET_PATH / "images" / "train",
    lbl_dir=DATASET_PATH / "labels" / "train",
    image_processor=image_processor,
    transform=train_augmentation_and_transform
)

validation_dataset = ImageDataset(
    img_dir=DATASET_PATH / "images" / "val",
    lbl_dir=DATASET_PATH / "labels" / "val",
    image_processor=image_processor,
    transform=validation_transform
)

test_dataset = ImageDataset(
    img_dir=DATASET_PATH / "images" / "test",
    lbl_dir=DATASET_PATH / "labels" / "test",
    image_processor=image_processor,
    transform=validation_transform
)


# DataLoader für das Training
import torch

def collate_fn(batch):
    data = {}
    data["pixel_values"] = torch.stack([x["pixel_values"] for x in batch
        ])
    data["labels"] = [x["labels"] for x in batch]
    return data


# Modell laden
from transformers import AutoModelForObjectDetection

```

```
156 model = AutoModelForObjectDetection.from_pretrained(
157     checkpoint,
158     id2label=id2label,
159     label2id=label2id,
160     ignore_mismatched_sizes=True,
161 )
162
163
164 # Hyperparameter für das Training definieren
165 from transformers import TrainingArguments, Trainer,
        EarlyStoppingCallback
166 import torch
167
168 use_cuda = torch.cuda.is_available()
169
170 args = TrainingArguments(
171     output_dir=OUTPUT_DIR,
172     per_device_train_batch_size=8,
173     fp16=use_cuda,     # nur auf GPU
174     learning_rate=1e-5,
175     weight_decay=0.05,
176     num_train_epochs=220,
177     warmup_ratio=0.10,
178     lr_scheduler_type="cosine",
179     eval_strategy="steps",# gültige Werte: "no ßteps ëpoch"
180     save_strategy="steps",
181     save_steps=100,
182     eval_steps=100,
183     save_total_limit=500,
184     metric_for_best_model="eval_loss",
185     greater_is_better=False,
186     logging_strategy="steps",
187     logging_first_step=True,
188     logging_steps=50,
189     disable_tqdm=False, # zeige Progressbar in Notebooks
190     load_best_model_at_end=True,
191     remove_unused_columns=False, # wichtig für Object Detection
192     dataloader_num_workers=0,
193     dataloader_pin_memory=use_cuda,
194     report_to="none",
195     seed=42
196 )
197
198 # Trainier initialisieren
199 trainer = Trainer(
```

```
200     model=model,
201     args=args,
202     train_dataset=train_dataset,
203     eval_dataset=validation_dataset,
204     data_collator=collate_fn,
205     processing_class=image_processor,
206     callbacks=[EarlyStoppingCallback(early_stopping_patience=6)] # wenn
            sich Leistung nicht mehr verbessert
207 )
208
209 # Training starten
210 trainer.train()
```

Anhang 5: Evaluation der trainierten YOLO-Modelle und des RT-DETR-Modells Quellcode

Anhang 5/1: Evaluation der YOLO-Modelle auf synthetischen und realen Bilddaten Quellcode

151

```
1 from ultralytics import YOLO
2 from pathlib import Path
3
4 # Vergleich verschiedener Modelle auf realen und synthetischen Datensätzen
5 MODEL_N_PATH = Path("")
6 MODEL_S_PATH = Path("")
7 MODEL_M_PATH = Path("")
8 DATASET_REAL_PATH = Path("")
9 DATASET_SYNTHETIC_PATH = Path("")
10
11 test_model_paths = [MODEL_N_PATH, MODEL_S_PATH, MODEL_M_PATH]
12 test_model_names = ["Model N", "Model S", "Model M"]
13
14 for model_path, model_name in zip(test_model_paths, test_model_names):
15     for dataset_path in [DATASET_REAL_PATH, DATASET_SYNTHETIC_PATH]:
16         model = YOLO(model_path / "weights" / "best.pt")
17
18         metrics = model.val(
19             data=dataset_path / "data.yaml",
20             split="test",
21             imgsz=640,
22             conf=0.1,
```

[151] Erstellung mit Unterstützung von ChatGPT-5 und Claude Sonnet 4

```
            save_json=True,
            save_txt=True,
            plots=True
        )
        print(f"----------- {model_name} ----------")
        print("mAP50-95:", metrics.box.map)        # mean AP (IoU 0.5:0.95)
        print("mAP50:", metrics.box.map50)         # mean AP (IoU 0.5)
        print("Precision:", metrics.box.mp)        # mean Precision
        print("Recall:", metrics.box.mr)           # mean Recall
```

Anhang 5/2: Evaluation des RT-DETR-Modells auf synthetischen und realen Bilddaten Quellcode

[152]

```
import shutil
import csv
from pathlib import Path
import torch
import json
from PIL import Image, ImageDraw
from transformers import AutoImageProcessor, AutoModelForObjectDetection

# -------------------------
# Konfiguration
# -------------------------
REAL_DIR = Path("")
SYNTHETIC_DIR = Path("")
CHECKPOINT_ROOT = Path("") # Modell-Checkpoint Ordner
if not CHECKPOINT_ROOT.exists():
    raise ValueError(f"Checkpoint-Ordner nicht gefunden: {
        CHECKPOINT_ROOT}")

IOU_THR = 0.50  # Schwelle für als korrekt klassifierte Bounding Box
CONF_THR = 0.1   # Anzeige/Export-Schwelle für Predictions
NMS_IOU_THR = 0.5  # IoU-Threshold für Non-Maximum Suppression


img_dir_real   = REAL_DIR / "images"
label_dir_real = REAL_DIR / "labels"
img_dir_synthetic   = SYNTHETIC_DIR / "images" / "test"
label_dir_synthetic = SYNTHETIC_DIR / "labels" / "test"

```

[152] Erstellung mit Unterstützung von ChatGPT-5 und Claude Sonnet 4

```
28 # -------------------------
29 # Hilfsfunktionen
30 # -------------------------
31 # aktuellsten Checkpoint finden
32 def find_best_checkpoint(root: Path, metric="eval_loss"):
33     if not root.exists():
34         return None
35 
36     candidates = [d for d in root.iterdir() if d.is_dir() and d.name.
       startswith("checkpoint-")]
37     if not candidates:
38         return None
39 
40     best_checkpoint = None
41     best_metric_value = float('inf')
42 
43     for checkpoint_dir in candidates:
44         trainer_state_file = checkpoint_dir / "trainer_state.json"
45 
46         if not trainer_state_file.exists():
47             print(f"  {checkpoint_dir.name}: Keine trainer_state.json
                gefunden")
48             continue
49 
50         with open(trainer_state_file, 'r') as f:
51             trainer_state = json.load(f)
52 
53         # Suche die Metrik in log_history
54         log_history = trainer_state.get('log_history', [])
55         metric_values = []
56 
57         for entry in log_history:
58             if metric in entry:
59                 metric_values.append(entry[metric])
60 
61         if not metric_values:
62             print(f"  {checkpoint_dir.name}: Metrik '{metric}' nicht
                gefunden")
63             continue
64 
65         # aktuellsten Wert dieser Metrik nehmen
66         current_value = metric_values[-1]
67 
68         # Prüfe ob dieser Checkpoint besser ist
69         if (current_value < best_metric_value):
```

```
70              best_metric_value = current_value
71              best_checkpoint = checkpoint_dir
72
73      return str(best_checkpoint)
74
75  # yolo format zu x1y1x2y2
76  def read_yolo_to_xyxy(label_path: Path, img_w: int, img_h: int):
77      boxes_xyxy, labels = [], []
78      if label_path.exists():
79          with open(label_path, "r") as f:
80              for line in f:
81                  parts = line.strip().split()
82                  if len(parts) != 5:
83                      continue
84                  c, cx, cy, bw, bh = map(float, parts)
85                  # normierte YOLO (cx,cy,w,h) -> Pixel-xyxy
86                  x = (cx - bw / 2.0) * img_w
87                  y = (cy - bh / 2.0) * img_h
88                  w = bw * img_w
89                  h = bh * img_h
90                  x1, y1 = max(0.0, x), max(0.0, y)
91                  x2, y2 = min(img_w - 1, x + w), min(img_h - 1, y + h)
92                  if x2 - x1 <= 0 or y2 - y1 <= 0:
93                      continue
94                  boxes_xyxy.append([x1, y1, x2, y2])
95                  labels.append(int(c))
96      return boxes_xyxy, labels
97
98  # x1y1x2y2 zu cx,cy,w,h (normiert)
99  def xyxy_to_yolo_norm(x1, y1, x2, y2, img_w, img_h):
100     w = max(0.0, x2 - x1)
101     h = max(0.0, y2 - y1)
102     cx = x1 + w / 2.0
103     cy = y1 + h / 2.0
104     # normalisieren
105     return cx / img_w, cy / img_h, w / img_w, h / img_h
106
107 # Boxen zeichnen
108 def draw_boxes(img: Image.Image, boxes_xyxy, labels=None, scores=None,
        id2label=None, color=(0, 255, 0)):
109     draw = ImageDraw.Draw(img)
110     for i, b in enumerate(boxes_xyxy):
111         x1, y1, x2, y2 = [int(v) for v in b]
112         draw.rectangle([x1, y1, x2, y2], outline=color, width=3)
113         txt = ""
```

```
114         if labels is not None:
115             lab_id = int(labels[i])
116             lab = id2label.get(lab_id, str(lab_id)) if id2label else str
                    (lab_id)
117             txt = lab
118         if scores is not None:
119             sc = float(scores[i])
120             txt = f"{txt} {sc:.2f}" if txt else f"{sc:.2f}"
121         if txt:
122             draw.text((x1 + 3, max(0, y1 - 12)), txt, fill=color)
123
124 # IoU zweier Boxen (x1y1x2y2) berechnen
125 def box_iou_xyxy(a, b):
126     ax1, ay1, ax2, ay2 = a
127     bx1, by1, bx2, by2 = b
128     ix1, iy1 = max(ax1, bx1), max(ay1, by1)
129     ix2, iy2 = min(ax2, bx2), min(ay2, by2)
130     iw, ih = max(0.0, ix2 - ix1), max(0.0, iy2 - iy1)
131     inter = iw * ih
132     if inter <= 0:
133         return 0.0
134     aw, ah = max(0.0, ax2 - ax1), max(0.0, ay2 - ay1)
135     bw, bh = max(0.0, bx2 - bx1), max(0.0, by2 - by1)
136     union = aw * ah + bw * bh - inter
137     return inter / union if union > 0 else 0.0
138
139 def non_max_suppression(boxes, scores, labels, iou_threshold=0.5):
140     # Non-Maximum Suppression: Entfernt überlappende Boxes mit niedrigeren
            Confidence-Scores
141
142     # Konvertiere zu Listen falls nötig
143     boxes = [list(box) for box in boxes]
144     scores = list(scores)
145     labels = list(labels)
146
147     # Indizes nach Score sortieren (höchste zuerst)
148     indices = sorted(range(len(scores)), key=lambda i: scores[i],
            reverse=True)
149     keep = []
150     suppressed = set()
151
152     for i in indices:
153         if i in suppressed:
154             continue
155
```

```
156         keep.append(i)
157
158         # Vergleiche mit allen verbleibenden Boxes
159         for j in indices:
160             if j == i or j in suppressed:
161                 continue
162
163             # Prüfe nur Boxes derselben Klasse
164             # aktuell nur eine Klasse implementiert, für zukünftige Erweiterungen
                    jedoch relevant
165             if labels[i] == labels[j]:
166                 iou = box_iou_xyxy(boxes[i], boxes[j])
167                 if iou > iou_threshold:
168                     suppressed.add(j)
169
170     # Behalte nur nicht-unterdrückte Detections
171     filtered_boxes = [boxes[i] for i in keep]
172     filtered_scores = [scores[i] for i in keep]
173     filtered_labels = [labels[i] for i in keep]
174
175     return filtered_boxes, filtered_scores, filtered_labels
176
177 # Boxen abgleichen (IoU >= Schwelle)
178 def match_detections(pred_boxes, gt_boxes, iou_thr=0.5):
179     matches = []
180     used_pred = set()
181     used_gt = set()
182     pairs = []
183     for pi, pb in enumerate(pred_boxes):
184         for gi, gb in enumerate(gt_boxes):
185             iou = box_iou_xyxy(pb, gb)
186             if iou >= iou_thr:
187                 pairs.append((iou, pi, gi))
188     pairs.sort(reverse=True, key=lambda x: x[0])
189     for iou, pi, gi in pairs:
190         if pi in used_pred or gi in used_gt:
191             continue
192         used_pred.add(pi)
193         used_gt.add(gi)
194         matches.append((pi, gi, iou))
195     tp_idx = {pi for (pi, _, _) in matches}
196     fp_idx = set(range(len(pred_boxes))) - tp_idx
197     fn_cnt = len(gt_boxes) - len(tp_idx)
198     return matches, tp_idx, fp_idx, fn_cnt
199
```

```
200 # Vergleichsbild zeichnen (GT rot, Preds grün/gelb)
201 def draw_compare(img: Image.Image, gt_boxes, pred_boxes, pred_scores,
        matches, id2label=None):
202     # GT rot
203     draw_boxes(img, gt_boxes, labels=[0]*len(gt_boxes), scores=None,
            id2label=id2label, color=(255, 0, 0))
204     # Preds: grün (TP) / gelb (FP)
205     tp_pred_idx = {pi for (pi, _, _) in matches}
206     for i, b in enumerate(pred_boxes):
207         col = (0, 255, 0) if i in tp_pred_idx else (255, 255, 0)
208         draw_boxes(img, [b], labels=[0], scores=[pred_scores[i]],
                id2label=id2label, color=col)
209     # IoU-Text an TPs
210     dr = ImageDraw.Draw(img)
211     for (pi, _, iou) in matches:
212         x1, y1, _, _ = pred_boxes[pi]
213         dr.text((int(x1)+3, max(0, int(y1)-24)), f"IoU {iou:.2f}", fill
                =(0, 255, 0))
214
215 # ------------------------
216 # Modell + Prozessor laden
217 # ------------------------
218 load_from = find_best_checkpoint(CHECKPOINT_ROOT)
219 processor = AutoImageProcessor.from_pretrained(load_from)
220 model = AutoModelForObjectDetection.from_pretrained(load_from)
221 id2label = getattr(model.config, "id2label", None) or {0: "object"}
222
223 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
224 model.to(device).eval()
225 print(f"Lade Modell aus: {load_from} | Device: {device}")
226
227 for data_type in ["real", "synthetic"]:
228     # Ordnerstruktur definieren
229     out_base = Path("runs/test_daten") / load_from.split("\\")[0] /
            data_type
230     pred_images_dir = out_base / "pred" / "images"
231     pred_labels_dir = out_base / "pred" / "labels"
232     gt_images_dir   = out_base / "gt" / "images"
233     gt_labels_dir   = out_base / "gt" / "labels"
234     copy_images_dir = out_base / "images"
235     pred_csv_path   = out_base / "predictions.csv"
236     compare_images_dir = out_base / "compare_images"
237     metrics_txt_path = out_base / "metrics.txt"
238
239     # wenn Ordner schon existiert, löschen
```

```
240     if out_base.exists():
241         print(f"Entferne alten Test-Ordner: {out_base}")
242         shutil.rmtree(out_base)
243 
244     # je nach Datentyp die Quellordner setzen
245     img_dir, label_dir = img_dir_synthetic, label_dir_synthetic
246     if data_type == "real":
247         img_dir = img_dir_real
248         label_dir = label_dir_real
249 
250     # -------------------------
251     # Ordner vorbereiten
252     # -------------------------
253     for data_type in ["real", "synthetic"]:
254         for d in [pred_images_dir, pred_labels_dir, gt_images_dir,
                gt_labels_dir, copy_images_dir, compare_images_dir]:
255             d.mkdir(parents=True, exist_ok=True)
256 
257     # Testbilder kopieren (wenn Ordner leer)
258     if not any(copy_images_dir.iterdir()):
259         for fn in sorted(img_dir.iterdir()):
260             if fn.suffix.lower() in {".jpg", ".jpeg", ".png", ".bmp", ".
                    webp"}:
261                 shutil.copy2(fn, copy_images_dir / fn.name)
262 
263     # -------------------------
264     # Durchlauf: GT zeichnen + Predictions exportieren
265     # -------------------------
266     # Metriken und Variablen initialisieren
267     rows = []
268     n_images = 0
269     n_gt_total = 0
270     n_pred_total = 0
271     TP = 0
272     FP = 0
273     FN = 0
274     sum_iou_tp = 0.0
275     n_tp_for_miou = 0
276     ap_records = []
277     all_pred_boxes = []
278     all_pred_scores = []
279     all_gt_boxes = []
280 
281     img_files = [f for f in sorted(img_dir.iterdir()) if f.suffix.lower
            () in {".jpg", ".jpeg", ".png", ".bmp", ".webp"}]
```

```
282     for img_path in img_files:
283         image = Image.open(img_path).convert("RGB")
284         w, h = image.size
285         stem = img_path.stem
286         n_images += 1
287 
288         # -- Ground Truth laden und Bild mit GT speichern --
289         gt_txt = label_dir / f"{stem}.txt"
290         gt_boxes, gt_labels = read_yolo_to_xyxy(gt_txt, w, h)
291         n_gt_total += len(gt_boxes)
292 
293         gt_img = image.copy()
294         if gt_boxes:
295             draw_boxes(gt_img, gt_boxes, labels=gt_labels, scores=None,
                    id2label=id2label, color=(255, 0, 0))
296         gt_img.save(gt_images_dir / f"{stem}.jpg")
297 
298         if gt_txt.exists():
299             shutil.copy2(gt_txt, gt_labels_dir / gt_txt.name)
300 
301         # -- Prediction --
302         inputs = processor(images=image, return_tensors="pt")
303         inputs = {k: v.to(device) for k, v in inputs.items()}
304         with torch.no_grad():
305             outputs = model(**inputs)
306 
307         processed = processor.post_process_object_detection(
308             outputs,
309             threshold=CONF_THR,
310             target_sizes=torch.tensor([[h, w]], device=device)
311         )[0]
312 
313         preds_img = image.copy()
314         pred_boxes = processed["boxes"].detach().cpu().tolist()
315         pred_scores = processed["scores"].detach().cpu().tolist()
316         pred_labels = processed["labels"].detach().cpu().tolist()
317 
318         # Non-Maximum Suppression anwenden
319         pred_boxes_nms, pred_scores_nms, pred_labels_nms =
                non_max_suppression(
320             pred_boxes, pred_scores, pred_labels, iou_threshold=
                    NMS_IOU_THR
321         )
322 
323         # Verwende NMS-gefilterte Predictions weiter
```

```
324         pred_boxes = pred_boxes_nms
325         pred_scores = pred_scores_nms
326         pred_labels = pred_labels_nms
327 
328         n_pred_total += len(pred_boxes)
329 
330         all_pred_boxes.append(pred_boxes)
331         all_pred_scores.append(pred_scores)
332         all_gt_boxes.append(gt_boxes)
333 
334         # annotierte Predictions speichern
335         draw_boxes(preds_img, pred_boxes, labels=pred_labels, scores=
                pred_scores, id2label=id2label, color=(0, 255, 0))
336         preds_img.save(pred_images_dir / f"{stem}.jpg")
337 
338         # YOLO-Pred-Labels speichern:
339         out_txt = pred_labels_dir / f"{stem}.txt"
340         with open(out_txt, "w", newline="\n") as f:
341             for c, score, (x1, y1, x2, y2) in zip(pred_labels,
                        pred_scores, pred_boxes):
342                 cx, cy, bw, bh = xyxy_to_yolo_norm(x1, y1, x2, y2, w, h)
343                 f.write(f"{int(c)} {cx:.6f} {cy:.6f} {bw:.6f} {bh:.6f} {
                        float(score):.6f}\n")
344                 rows.append([
345                     img_path.name, int(c), id2label.get(int(c), str(int(
                            c))),
346                     float(score),
347                     float(x1), float(y1), float(x2), float(y2),
348                     float(cx), float(cy), float(bw), float(bh)
349                 ])
350 
351         # -- Vergleiche + Metriken --
352         matches, tp_idx, fp_idx, fn_cnt = match_detections(pred_boxes,
                gt_boxes, iou_thr=IOU_THR)
353         TP += len(tp_idx)
354         FP += len(fp_idx)
355         FN += fn_cnt
356         for (_, _, iou) in matches:
357             sum_iou_tp += iou
358             n_tp_for_miou += 1
359         for i, s in enumerate(pred_scores):
360             ap_records.append((float(s), 1 if i in tp_idx else 0))
361 
362         # Vergleichsbild speichern: GT rot, TP grün, FP gelb + IoU
363         cmp_img = image.copy()
```

```
364         draw_compare(cmp_img, gt_boxes, pred_boxes, pred_scores, matches
                , id2label=id2label)
365         compare_images_dir.mkdir(parents=True, exist_ok=True)
366         (compare_images_dir / f"{stem}.jpg").parent.mkdir(parents=True,
                exist_ok=True)
367         cmp_img.save(compare_images_dir / f"{stem}.jpg")
368
369     # CSV speichern
370     with open(pred_csv_path, "w", newline="", encoding="utf-8") as f:
371         writer = csv.writer(f)
372         writer.writerow(["image", "class_id", "class_name", "conf", "x1"
                , "y1", "x2", "y2", "cx", "cy", "w", "h"])
373         writer.writerows(rows)
374
375     # Metriken berechnen und metrics.txt schreiben
376     def compute_pr_ap(records, n_gt):
377         if n_gt == 0 or not records:
378             return 0.0, [], []
379         recs = sorted(records, key=lambda x: x[0], reverse=True)
380         tp_cum = 0
381         fp_cum = 0
382         precisions = []
383         recalls = []
384         for _, is_tp in recs:
385             if is_tp:
386                 tp_cum += 1
387             else:
388                 fp_cum += 1
389             precisions.append(tp_cum / max(1, (tp_cum + fp_cum)))
390             recalls.append(tp_cum / max(1, n_gt))
391
392         mrec = [0.0] + recalls + [1.0]
393         mpre = [0.0] + precisions + [0.0]
394         for i in range(len(mpre) - 2, -1, -1):
395             mpre[i] = max(mpre[i], mpre[i+1])
396         ap = 0.0
397         for i in range(1, len(mrec)):
398             ap += (mrec[i] - mrec[i-1]) * mpre[i]
399         return ap, precisions, recalls
400
401     # Funktion, um die AP für mehrere IoU zu berechnen (für AP50-95)
402     def compute_ap_multiple_iou(pred_boxes, pred_scores, gt_boxes_all,
            iou_thresholds):
403         aps = []
404
```

```
405         for iou_thr in iou_thresholds:
406             ap_records = []
407             for _, (pred_b, pred_s, gt_b) in enumerate(zip(pred_boxes,
                    pred_scores, gt_boxes_all)):
408                 if not pred_b:  # Keine Predictions
409                     continue
410 
411                 _, tp_idx, _, _ = match_detections(pred_b, gt_b, iou_thr
                        =iou_thr)
412 
413                 for i, s in enumerate(pred_s):
414                     ap_records.append((float(s), 1 if i in tp_idx else
                            0))
415 
416             total_gt = sum(len(gt_b) for gt_b in gt_boxes_all)
417             ap, _, _ = compute_pr_ap(ap_records, total_gt)
418             aps.append(ap)
419 
420         return aps
421 
422     precision = TP / max(1, (TP + FP))
423     recall    = TP / max(1, n_gt_total)
424     f1        = 2 * precision * recall / max(1e-12, (precision + recall)
            )
425     miou      = (sum_iou_tp / n_tp_for_miou) if n_tp_for_miou > 0 else
            0.0
426     ap50, _, _ = compute_pr_ap(ap_records, n_gt_total)
427 
428     iou_thresholds = [0.5 + 0.05 * i for i in range(10)]  # 0.5, 0.55,
            0.6, ..., 0.95
429     aps_multiple = compute_ap_multiple_iou(all_pred_boxes,
            all_pred_scores, all_gt_boxes, iou_thresholds)
430     ap50_95 = sum(aps_multiple) / len(aps_multiple) if aps_multiple else
             0.0
431 
432     # Metriken in Datei schreiben
433     with open(metrics_txt_path, "w", encoding="utf-8") as f:
434         f.write(f"Images: {n_images}\n")
435         f.write(f"GT boxes: {n_gt_total}\n")
436         f.write(f"Pred boxes: {n_pred_total}\n")
437         f.write(f"IoU threshold: {IOU_THR}\n")
438         f.write(f"NMS IoU threshold: {NMS_IOU_THR}\n")
439         f.write(f"Confidence threshold: {CONF_THR}\n")
440         f.write(f"TP: {TP}\nFP: {FP}\nFN: {FN}\n")
441         f.write(f"Precision: {precision:.4f}\n")
```

```
442             f.write(f"Recall:    {recall:.4f}\n")
443             f.write(f"F1:        {f1:.4f}\n")
444             f.write(f"mIoU(TP):  {miou:.4f}\n")
445             f.write(f"AP@{IOU_THR}: {ap50:.4f}\n")
446             f.write(f"AP@0.50-0.95: {ap50_95:.4f}\n")
447
448 print(f"Fertig. Export unter: {out_base}")
449 print(f"- Predictions: {pred_images_dir} (Bilder), {pred_labels_dir} (
        YOLO-TXT mit conf)")
450 print(f"- Ground Truth: {gt_images_dir} (Bilder), {gt_labels_dir} (GT-
        TXT Kopie)")
451 print(f"- Compare: {compare_images_dir} (GT=rot, TP=grün, FP=gelb, IoU-
        Text an TPs)")
452 print(f"- CSV: {pred_csv_path}")
453 print(f"- Metrics: {metrics_txt_path}")
```

Anhang 6: Metriken zur Evaluation der YOLO-Modelle

Anhang 6/1: Trainingsverlauf des Nano Modells

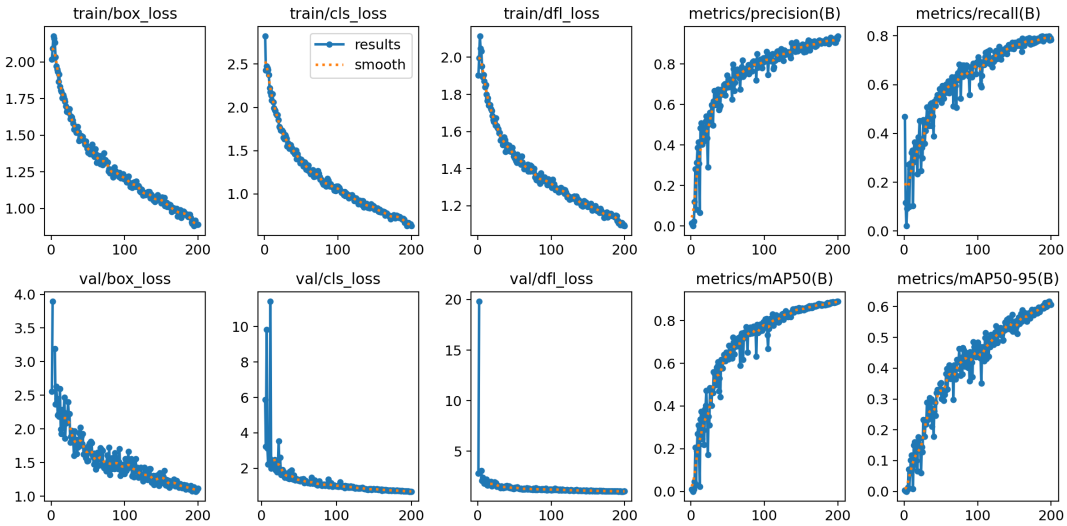


Abb. 9: Verlauf von Metriken während des Trainings des YOLO Modells Nano

Anhang 6/2: Trainingsverlauf des Small Modells

Anhang 6/3: Precision-Recall-Kurve für das Small Modell

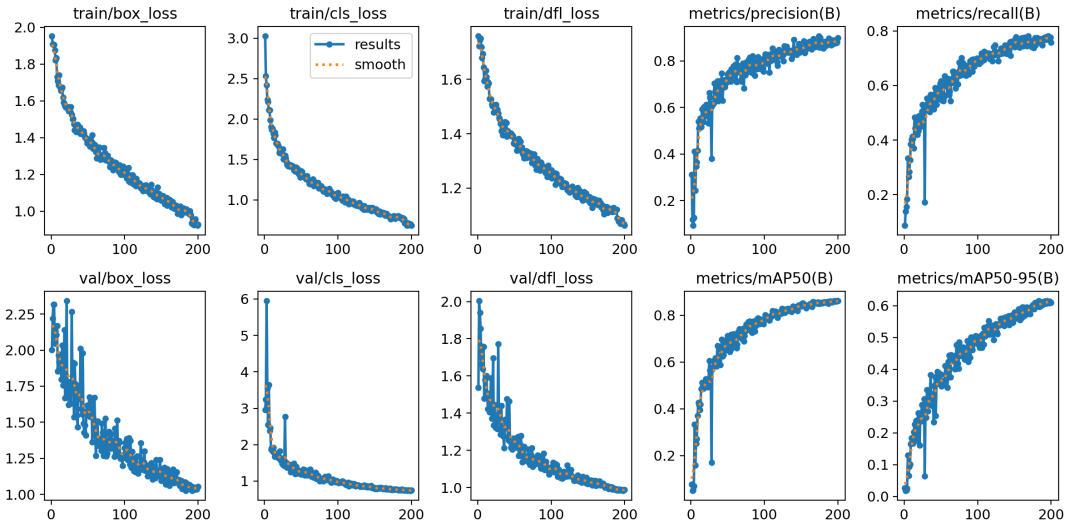


Abb. 10: Verlauf von Metriken während des Trainings des YOLO Modells Small

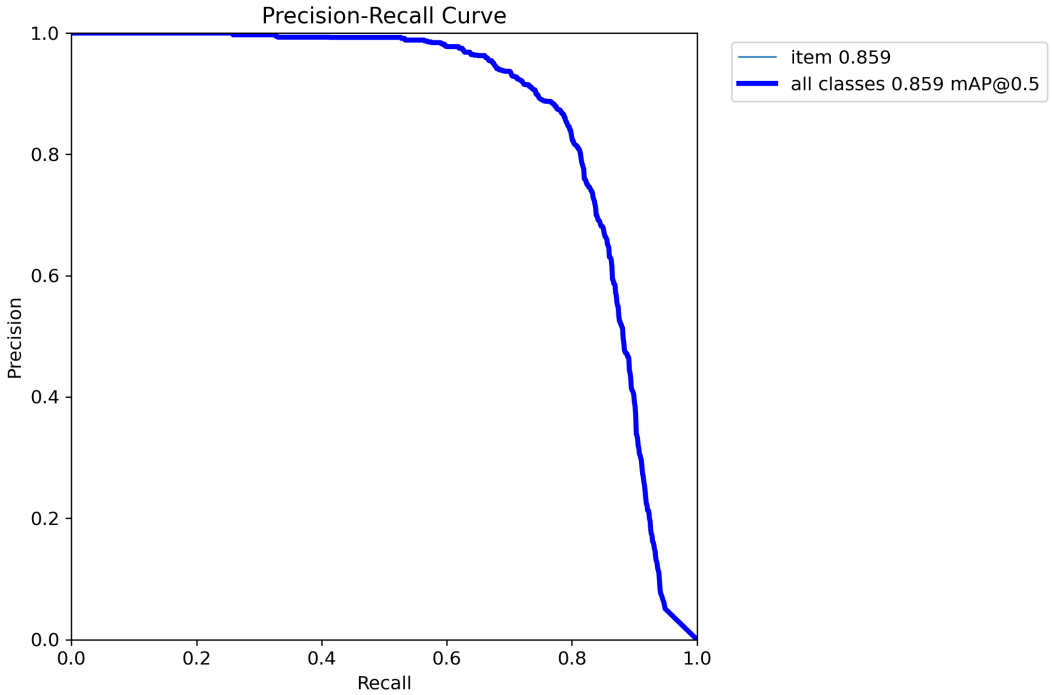


Abb. 11: Precision-Recall-Kurve des YOLO Small Modells

Literaturverzeichnis

Alfaro-Viquez, David; Zamora-Hernandez, Mauricio; Fernandez-Vega, Michael; Garcia-Rodriguez, Jose; Azorin-Lopez, Jorge (2025): A Comprehensive Review of AI-Based Digital Twin Applications in Manufacturing: Integration Across Operator, Product, and Process Dimensions. en. In: *Electronics* 14.4. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, S. 646. ISSN: 2079-9292. DOI: 10.3390/electronics14040646. URL: https://www.mdpi.com/2079-9292/14/4/646 (Abruf: 28. 06. 2025).

Ali, Kazim; Bhatti, Muhammad Shahid; Saeed, Atif; Athar, Atifa; Al Ghamdi, Mohammed A.; Almotiri, Sultan H.; Akram, Samina (2024): Adversarial Robustness of Vision Transformers Versus Convolutional Neural Networks. In: *IEEE Access* 12, S. 105281–105293. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3435347. URL: https://ieeexplore.ieee.org/document/10614176/ (Abruf: 13. 08. 2025).

Andrade, Chittaranjan (2018): Internal, External, and Ecological Validity in Research Design, Conduct, and Evaluation. In: *Indian Journal of Psychological Medicine* 40.5, S. 498–499. ISSN: 0253-7176, 0975-1564. DOI: 10.4103/IJPSYM.IJPSYM_334_18. URL: https://journals.sagepub.com/doi/10.4103/IJPSYM.IJPSYM_334_18 (Abruf: 18. 08. 2025).

Arslanoglu, Muhammed Cihad; Albayrak, Abdulkadir; Acar, Huseyin (2025): Vision Transformers Versus Convolutional Neural Networks: Comparing Robustness by Exploiting Varying Local Features. In: *IEEE Access* 13, S. 65232–65245. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2025.3559794. URL: https://ieeexplore.ieee.org/document/10962160/ (Abruf: 13. 08. 2025).

Bai, Xiangyu; Luo, Yedi; Jiang, Le; Gupta, Aniket; Kaveti, Pushyami; Singh, Hanumant; Ostadabbas, Sarah (2023): Bridging the Domain Gap between Synthetic and Real-World Data for Autonomous Driving. Version Number: 1. DOI: 10.48550/ARXIV.2306.02631. URL: https://arxiv.org/abs/2306.02631 (Abruf: 15. 08. 2025).

Berroukham, Abdelhafid; Housni, Khalid; Lahraichi, Mohammed (2023): Vision Transformers: A Review of Architecture, Applications, and Future Directions. In: *2023 7th IEEE Congress on Information Science and Technology (CiSt)*. 2023 7th IEEE Congress on Information Science and Technology (CiSt). Agadir - Essaouira, Morocco: IEEE, S. 205–210. DOI: 10.1109/cist56084.2023.10410015. URL: https://ieeexplore.ieee.org/document/10410015/ (Abruf: 25. 07. 2025).

Bhatt, Dulari; Patel, Chirag; Talsania, Hardik; Patel, Jigar; Vaghela, Rasmika; Pandya, Sharnil; Modi, Kirit; Ghayvat, Hemant (2021): CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope. en. In: *Electronics* 10.20. Publisher: MDPI AG, S. 2470. ISSN: 2079-9292. DOI: 10.3390/electronics10202470. URL: https://www.mdpi.com/2079-9292/10/20/2470 (Abruf: 15. 07. 2025).

Bilous, Nataliya; Malko, Vladyslav; Frohme, Marcus; Nechyporenko, Alina (2024): Comparison of CNN-Based Architectures for Detection of Different Object Classes. In: *AI* 5.4, S. 2300–2320. ISSN: 2673-2688. DOI: 10.3390/ai5040113. URL: https://www.mdpi.com/2673-2688/5/4/113 (Abruf: 21. 08. 2025).

Bodla, Navaneeth; Singh, Bharat; Chellappa, Rama; Davis, Larry S. (2017): Soft-NMS – Improving Object Detection With One Line of Code. Version Number: 2. DOI: 10.48550/ARXIV.1704.04503. URL: https://arxiv.org/abs/1704.04503 (Abruf: 26. 08. 2025).

Boikov, Aleksei; Payor, Vladimir; Savelev, Roman; Kolesnikov, Alexandr (2021): Synthetic Data Generation for Steel Defect Detection and Classification Using Deep Learning. en. In: *Symmetry* 13.7. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, S. 1176. ISSN: 2073-8994. DOI: 10.3390/sym13071176. URL: https://www.mdpi.com/2073-8994/13/7/1176 (Abruf: 27. 06. 2025).

Chai, Junyi; Zeng, Hao; Li, Anming; Ngai, Eric W.T. (2021): Deep learning in computer vision: A critical review of emerging techniques and application scenarios. en. In: *Machine Learning with Applications* 6. Publisher: Elsevier BV, S. 100134. ISSN: 2666-8270. DOI: 10.1016/j.mlwa.2021.100134. URL: https://linkinghub.elsevier.com/retrieve/pii/S2666827021000670 (Abruf: 15. 07. 2025).

„Computer Vision Overview“ (2023). en. In: *3-D Computer Vision*. Singapore: Springer Nature Singapore, S. 1–35. ISBN: 978-981-19-7579-0 978-981-19-7580-6. DOI: 10.1007/978-981-19-7580-6_1. URL: https://link.springer.com/10.1007/978-981-19-7580-6_1 (Abruf: 15. 07. 2025).

Dosovitskiy, Alexey; Beyer, Lucas; Kolesnikov, Alexander; Weissenborn, Dirk; Zhai, Xiaohua; Unterthiner, Thomas; Dehghani, Mostafa; Minderer, Matthias; Heigold, Georg; Gelly, Sylvain; Uszkoreit, Jakob; Houlsby, Neil (2020): An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Version Number: 2. DOI: 10.48550/ARXIV.2010.11929. URL: https://arxiv.org/abs/2010.11929 (Abruf: 15. 08. 2025).

Fulir, Juraj; Bosnar, Lovro; Hagen, Hans; Gospodnetić, Petra (2023): Synthetic Data for Defect Segmentation on Complex Metal Surfaces. In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Vancouver, BC, Canada: IEEE, S. 4424–4434. ISBN: 979-8-3503-0249-3. DOI: 10.1109/CVPRW59228.2023.00465. URL: https://ieeexplore.ieee.org/document/10208376/ (Abruf: 27. 06. 2025).

Fuller, Aidan; Fan, Zhong; Day, Charles; Barlow, Chris (2020): Digital Twin: Enabling Technologies, Challenges and Open Research. In: *IEEE Access* 8. Publisher: Institute of Electrical and Electronics Engineers (IEEE), S. 108952–108971. ISSN: 2169-3536. DOI: 10.1109/access.2020.2998358. URL: https://ieeexplore.ieee.org/document/9103025/ (Abruf: 07. 07. 2025).

Genesis-Embodied-AI/Genesis (2025). original-date: 2023-10-31T03:33:11Z. URL: https://github.com/Genesis-Embodied-AI/Genesis (Abruf: 24. 08. 2025).

Griem, Lars; Koeppe, Arnd; Greß, Alexander; Feser, Thomas; Nestler, Britta (2025): Synthetic training data for CT image segmentation of microstructures. In: *Acta Materialia* 296, S. 121220. ISSN: 13596454. DOI: 10.1016/j.actamat.2025.121220. URL: https://linkinghub.elsevier.com/retrieve/pii/S1359645425005075 (Abruf: 21. 08. 2025).

Hadadan, Saeed; Bitterli, Benedikt; Zeltner, Tizian; Novák, Jan; Rousselle, Fabrice; Munkberg, Jacob; Hasselgren, Jon; Wronski, Bartlomiej; Zwicker, Matthias (2025): Generative Detail Enhancement for Physically Based Materials. In: Publisher: arXiv

Version Number: 2. DOI: 10.48550/ARXIV.2502.13994. URL: https://arxiv.org/abs/2502.13994 (Abruf: 11. 07. 2025).

He, Kaiming; Gkioxari, Georgia; Dollár, Piotr; Girshick, Ross (2017): Mask R-CNN. Version Number: 3. DOI: 10.48550/ARXIV.1703.06870. URL: https://arxiv.org/abs/1703.06870 (Abruf: 26. 08. 2025).

Hevner; March; Park; Ram (2004): Design Science in Information Systems Research. In: *MIS Quarterly* 28.1, S. 75. ISSN: 02767783. DOI: 10.2307/25148625. URL: https://www.jstor.org/stable/10.2307/25148625 (Abruf: 15. 08. 2025).

Holtbrügge, Dirk; Holzmüller, Hartmut H.; Von Wangenheim, Florian (2007): Remote Services: neue Formen der Internationalisierung von Dienstleistungen. 1. Aufl. OCLC: 401509773. Wiesbaden: Deutscher Universitäts-Verlag. ISBN: 978-3-8350-9515-1.

Hosang, Jan; Benenson, Rodrigo; Schiele, Bernt (2017): Learning non-maximum suppression. Version Number: 2. DOI: 10.48550/ARXIV.1705.02950. URL: https://arxiv.org/abs/1705.02950 (Abruf: 26. 08. 2025).

Hütten, Nils; Meyes, Richard; Meisen, Tobias (2022): Vision Transformer in Industrial Visual Inspection. In: *Applied Sciences* 12.23. Publisher: MDPI AG, S. 11981. ISSN: 2076-3417. DOI: 10.3390/app122311981. URL: https://www.mdpi.com/2076-3417/12/23/11981 (Abruf: 25. 07. 2025).

Isaac Sim Requirements — Isaac Sim Documentation (2025). URL: https://docs.isaacsim.omniverse.nvidia.com/latest/installation/requirements.html (Abruf: 24. 08. 2025).

Jain, Saksham; Seth, Gautam; Paruthi, Arpit; Soni, Umang; Kumar, Girish (2022): Synthetic data augmentation for surface defect detection and classification using deep learning. en. In: *Journal of Intelligent Manufacturing* 33.4, S. 1007–1020. ISSN: 1572-8145. DOI: 10.1007/s10845-020-01710-x. URL: https://doi.org/10.1007/s10845-020-01710-x (Abruf: 27. 06. 2025).

Jamil, Sonain; Piran, Md. Jalil; Kwon, Oh-Jin (2022): A Comprehensive Survey of Transformers for Computer Vision. Version Number: 1. DOI: 10.48550/ARXIV.2211.06004. URL: https://arxiv.org/abs/2211.06004 (Abruf: 25. 07. 2025).

Jing, Luyang; Zhao, Ming; Li, Pin; Xu, Xiaoqiang (2017): A convolutional neural network based feature learning and fault diagnosis method for the condition monitoring of gearbox. In: *Measurement* 111. Publisher: Elsevier BV, S. 1–10. ISSN: 0263-2241. DOI: 10.1016/j.measurement.2017.07.017. URL: https://linkinghub.elsevier.com/retrieve/pii/S0263224117304517 (Abruf: 25. 07. 2025).

Jocher, Glenn; Qiu, Jing (2024): *Ultralytics YOLO11*. Version 11.0.0. URL: https://github.com/ultralytics/ultralytics.

Khan, Asifullah; Rauf, Zunaira; Sohail, Anabia; Khan, Abdul Rehman; Asif, Hifsa; Asif, Aqsa; Farooq, Umair (2023): A survey of the vision transformers and their CNN-transformer based variants. In: *Artificial Intelligence Review* 56 (S3), S. 2917–2970. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-023-10595-0. URL: https://link.springer.com/10.1007/s10462-023-10595-0 (Abruf: 31. 07. 2025).

Khanam, Rahima; Hussain, Muhammad; Hill, Richard; Allen, Paul (2024): A Comprehensive Review of Convolutional Neural Networks for Defect Detection in Industrial Applications. In: *IEEE Access* 12, S. 94250–94295. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3425166. URL: https://ieeexplore.ieee.org/abstract/document/10589380 (Abruf: 28. 06. 2025).

Khirodkar, Rawal; Yoo, Donghyun; Kitani, Kris M. (2018): Domain Randomization for Scene-Specific Car Detection and Pose Estimation. Version Number: 1. DOI: 10.48550/ARXIV.1811.05939. URL: https://arxiv.org/abs/1811.05939 (Abruf: 21. 08. 2025).

Kritzinger, Werner; Karner, Matthias; Traar, Georg; Henjes, Jan; Sihn, Wilfried (2018): Digital Twin in manufacturing: A categorical literature review and classification. In: *IFAC-PapersOnLine* 51.11. Publisher: Elsevier BV, S. 1016–1022. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.08.474. URL: https://linkinghub.elsevier.com/retrieve/pii/S2405896318316021 (Abruf: 07. 07. 2025).

Leite, Denis; Andrade, Emmanuel; Rativa, Diego; Maciel, Alexandre M. A. (2024): Fault Detection and Diagnosis in Industry 4.0: A Review on Challenges and Opportunities. In: *Sensors* 25.1, S. 60. ISSN: 1424-8220. DOI: 10.3390/s25010060. URL: https://www.mdpi.com/1424-8220/25/1/60 (Abruf: 26. 08. 2025).

Liu, Mengnan; Fang, Shuiliang; Dong, Huiyue; Xu, Cunzhi (2021): Review of digital twin about concepts, technologies, and industrial applications. In: *Journal of Manufacturing Systems* 58, S. 346–361. ISSN: 02786125. DOI: 10.1016/j.jmsy.2020.06.017. URL: https://linkinghub.elsevier.com/retrieve/pii/S0278612520301072 (Abruf: 31. 07. 2025).

Manettas, Christos; Nikolakis, Nikolaos; Alexopoulos, Kosmas (2021): Synthetic datasets for Deep Learning in computer-vision assisted tasks in manufacturing. Bd. 103. Journal Abbreviation: Procedia CIRP Pages: 242 Publication Title: Procedia CIRP. DOI: 10.1016/j.procir.2021.10.038.

Mercorelli, Paolo (2024): Recent Advances in Intelligent Algorithms for Fault Detection and Diagnosis. In: *Sensors* 24.8, S. 2656. ISSN: 1424-8220. DOI: 10.3390/s24082656. URL: https://www.mdpi.com/1424-8220/24/8/2656 (Abruf: 08. 08. 2025).

Mikołajewska, Emilia; Mikołajewski, Dariusz; Mikołajczyk, Tadeusz; Paczkowski, Tomasz (2025): Generative AI in AI-Based Digital Twins for Fault Diagnosis for Predictive Maintenance in Industry 4.0/5.0. In: *Applied Sciences* 15.6, S. 3166. ISSN: 2076-3417. DOI: 10.3390/app15063166. URL: https://www.mdpi.com/2076-3417/15/6/3166 (Abruf: 31. 07. 2025).

Monnet, Josefine; Petrovic, Oliver; Herfs, Werner (2024): Investigating the generation of synthetic data for surface defect detection: A comparative analysis. en. In: *Procedia CIRP* 130, S. 767–773. ISSN: 22128271. DOI: 10.1016/j.procir.2024.10.162. URL: https://linkinghub.elsevier.com/retrieve/pii/S2212827124013192 (Abruf: 27. 06. 2025).

NVIDIA (2025a): Isaac Sim Documentation. URL: https://docs.isaacsim.omniverse.nvidia.com/latest/index.html (Abruf: 20. 08. 2025).

NVIDIA (2025b): Omniverse Replicator. URL: https://docs.omniverse.nvidia.com/extensions/latest/ext_replicator.html (Abruf: 20. 08. 2025).

NVIDIA (2025c): Omniverse USD Composer. URL: https://docs.omniverse.nvidia.com/composer/latest/index.html (Abruf: 20. 08. 2025).

Peffers, Ken; Tuunanen, Tuure; Rothenberger, Marcus A.; Chatterjee, Samir (2007): A Design Science Research Methodology for Information Systems Research. In: *Journal of Management Information Systems* 24.3, S. 45–77. ISSN: 0742-1222, 1557-928X. DOI: 10.2753/MIS0742-1222240302. URL: https://www.tandfonline.com/doi/full/10.2753/MIS0742-1222240302 (Abruf: 15. 08. 2025).

Schmedemann, Ole; Baaß, Melvin; Schoepflin, Daniel; Schüppstuhl, Thorsten (2022): Procedural synthetic training data generation for AI-based defect detection in industrial surface inspection. en. In: *Procedia CIRP* 107, S. 1101–1106. ISSN: 22128271. DOI: 10.1016/j.procir.2022.05.115. URL: https://linkinghub.elsevier.com/retrieve/pii/S2212827122003997 (Abruf: 27. 06. 2025).

Seid Ahmed, Yassmin; Abubakar, Abba A.; Arif, Abul Fazal M.; Al-Badour, Fadi A. (2025): Advances in fault detection techniques for automated manufacturing systems in industry 4.0. In: *Frontiers in Mechanical Engineering* 11, S. 1564846. ISSN: 2297-3079. DOI: 10.3389/fmech.2025.1564846. URL: https://www.frontiersin.org/articles/10.3389/fmech.2025.1564846/full (Abruf: 08. 08. 2025).

Shorten, Connor; Khoshgoftaar, Taghi M. (2019): A survey on Image Data Augmentation for Deep Learning. en. In: *Journal of Big Data* 6.1. Publisher: Springer Science and Business Media LLC. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0 (Abruf: 07. 07. 2025).

Singh, Navdeep; Sabrol, Hiteshwari (2021): Convolutional Neural Networks-An Extensive arena of Deep Learning. A Comprehensive Study. In: *Archives of Computational Methods in Engineering* 28.7. Publisher: Springer Science and Business Media LLC, S. 4755–4780. ISSN: 1134-3060, 1886-1784. DOI: 10.1007/s11831-021-09551-4. URL: https://link.springer.com/10.1007/s11831-021-09551-4 (Abruf: 25. 07. 2025).

The Australian National University; Gregor, Shirley; Hevner, Alan R.; University of South Florida (2013): Positioning and Presenting Design Science Research for Maximum Impact. In: *MIS Quarterly* 37.2, S. 337–355. ISSN: 02767783, 21629730. DOI: 10.25300/MISQ/2013/37.2.01. URL: https://misq.org/positioning-and-presenting-design-science-research-for-maximum-impact.html (Abruf: 15. 08. 2025).

TRUMPF SE & Co. KG (2025): TRUMPF SE & Co. KG. URL: https://www.trumpf.com/de_DE/ (Abruf: 20. 08. 2025).

Trunzer, Emanuel; Pirehgalin, Mina Fahimi; Vogel-Heuser, Birgit; Odenweller, Matthias (2024): „Remote Operations: Fernüberwachung von Produktionsanlagen“. In: *Handbuch Industrie 4.0*. Hrsg. von Birgit Vogel-Heuser; Michael Ten Hompel; Thomas Bauernhansl. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 591–598. ISBN: 978-3-662-58527-6 978-3-662-58528-3. DOI: 10.1007/978-3-662-58528-3_173. URL: https://link.springer.com/10.1007/978-3-662-58528-3_173 (Abruf: 12. 08. 2025).

Ultralytics (2025a): Hyperparameter-Optimierung. URL: https://docs.ultralytics.com/de/guides/hyperparameter-tuning (Abruf: 22. 08. 2025).

Ultralytics (2025b): YOLO-Datenerweiterung. URL: https://docs.ultralytics.com/de/guides/yolo-data-augmentation (Abruf: 22. 08. 2025).

Urgo, Marcello; Terkaj, Walter; Simonetti, Gabriele (2024): Monitoring manufacturing systems using AI: A method based on a digital factory twin to train CNNs on synthetic data. In: *CIRP Journal of Manufacturing Science and Technology* 50, S. 249–268. ISSN: 1755-5817. DOI: 10.1016/j.cirpj.2024.03.005. URL: https://www.sciencedirect.com/science/article/pii/S1755581724000361 (Abruf: 27. 06. 2025).

Wen, Long; Li, Xinyu; Gao, Liang; Zhang, Yuyan (2018): A New Convolutional Neural Network-Based Data-Driven Fault Diagnosis Method. In: *IEEE Transactions on Industrial Electronics* 65.7, S. 5990–5998. ISSN: 0278-0046, 1557-9948. DOI: 10.1109/TIE.2017.2774777. URL: http://ieeexplore.ieee.org/document/8114247/ (Abruf: 08. 08. 2025).

Wu, Haiyue; Triebe, Matthew J.; Sutherland, John W. (2023): A transformer-based approach for novel fault detection and fault classification/diagnosis in manufacturing: A rotary system application. In: *Journal of Manufacturing Systems* 67, S. 439–452. ISSN: 02786125. DOI: 10.1016/j.jmsy.2023.02.018. URL: https://linkinghub.elsevier.com/retrieve/pii/S0278612523000419 (Abruf: 08. 08. 2025).

Zaripov, Alexey; Kulshin, Roman; Sidorov, Anatoly (2025): The Creation of Artificial Data for Training a Neural Network Using the Example of a Conveyor Production Line for Flooring. en. In: *Journal of Imaging* 11.5. Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, S. 168. ISSN: 2313-433X. DOI: 10.3390/jimaging11050168. URL: https://www.mdpi.com/2313-433X/11/5/168 (Abruf: 27. 06. 2025).

Zhao, Yian; Lv, Wenyu; Xu, Shangliang; Wei, Jinman; Wang, Guanzhong; Dang, Qingqing; Liu, Yi; Chen, Jie (2023): DETRs Beat YOLOs on Real-time Object Detection. Version Number: 3. DOI: 10.48550/ARXIV.2304.08069. URL: https://arxiv.org/abs/2304.08069 (Abruf: 21. 08. 2025).

Erklärung zur Verwendung von KI-Systemen

Ich erkläre, dass ich

- ☑ mich aktiv über die Leistungsfähigkeit und Beschränkungen der in meiner Arbeit eingesetzten KI-Systeme informiert habe;

- ☑ alle Inhalte aus wissenschaftlich anerkannten Quellen entnommen und entsprechend gekennzeichnet habe; alle Inhalte unter Anwendung wissenschaftlicher Methoden im Rahmen der vorliegenden Arbeit von mir selbst entwickelt wurden;

- ☑ mir bewusst bin, dass ich als Autorin dieser Arbeit die Verantwortung für die in ihr gemachten Angaben und Aussagen trage.

Bei der Erstellung der Arbeit habe ich die folgenden auf KI basierenden Systeme in der im Folgenden dargestellten Weise benutzt:

Arbeitsschritt	**Eingesetzte KI-Systeme**	**Beschreibung der Verwendungsweise**
Literatursuche	Consensus	Teilweise unterstützender Charakter für die Auffindung relevanter Literatur
Codeerstellung und -unterstützung	ChatGPT-5, Claude Sonnet 4	Unterstützung bei Code-Vervollständigungen und Fehlersuche
Formulierung des Textes der Arbeit	ChatGPT-5	Sprachliche Unterstützung in Form von Formulierungsvorschläge zur sprachlichen Präzisierung und Variation
Redigieren des Textes	ChatGPT-5	Unterstützung bei der sprachlichen Glättung und Verbesserung der Lesbarkeit

Ort, Datum, Unterschrift

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: *Entwicklung und Evaluation von KI-Modellen auf Basis synthetischer Daten aus digitalen Modellen zur Fehlererkennung bei Werkzeugmaschinen* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

(Ort, Datum) (Unterschrift)