

**SYSTEMANALYSE UND
-OPTIMIERUNG**

Carl von Ossietzky Universität Oldenburg

Studiengang: Master Wirtschaftsinformatik

**Automatische Identifizierung von statischen Hafenobjekten auf der Basis
von Kamerabildern eines hochautonomen Baggerschiffes**

Masterarbeit

Fakultät II
Department für Informatik
Abteilung Systemanalyse und -optimierung

Themensteller: Prof. Dr. Axel Hahn
Betreuer: M.Sc. Matthias Steidel

vorgelegt von: Timo Siering
Alte Färberei 3b
4139987
26129 Oldenburg
015152297606
E-Mail: timo.siering@uni-oldenburg.de

Abgabetermin: 25. Mai 2023

Inhaltsverzeichnis

INHALTSVERZEICHNIS	I
ABBILDUNGSVERZEICHNIS	III
TABELLENVERZEICHNIS	IV
1 EINLEITUNG	1
1.1 PROBLEMSTELLUNG.....	1
1.2 ZIELE DER ARBEIT	4
2 GRUNDLAGEN	6
2.1 STATISCHE HAFENOBJEKTE	6
2.2 KÜNSTLICHE INTELLIGENZ.....	8
2.2.1 <i>Machine Learning</i>	8
2.2.2 <i>Deep Learning</i>	9
2.2.3 <i>Arten von künstlichen neuronalen Netzwerken</i>	10
2.2.4 <i>Training und Validierung von ML-Modellen</i>	13
2.3 COMPUTER VISION	28
2.3.1 <i>Einstufige Detektoren</i>	30
2.3.2 <i>Zweistufige Detektoren</i>	32
3 STAND DER WISSENSCHAFT UND TECHNIK.....	33
3.1 AUTOMOBILINDUSTRIE	33
3.2 MARITIME DOMÄNE.....	34
4 ANFORDERUNGSERHEBUNG.....	39
4.1 METHODIK.....	39
4.2 ANFORDERUNGEN	39
5 KONZEPT	42
5.1 TEILZIELE UND ARBEITSSCHRITTE	42
5.1.1 <i>Konzeptionierung des Datensatzes</i>	43
5.1.2 <i>Aufbereitung des Rohmaterials</i>	44
5.1.3 <i>Annotation der Bilder</i>	45
5.1.4 <i>Aufbereitung des Datensatzes</i>	46
5.1.5 <i>Auswahl eines geeigneten KI-Modells</i>	49
5.1.6 <i>Training und Parametrisierung</i>	54
5.1.7 <i>Performance-Evaluation</i>	57

6	IMPLEMENTIERUNG	58
6.1	DATENVORVERARBEITUNG	58
6.2	ANNOTATION DER BILDER	59
6.3	AUFBEREITUNG DES DATENSATZES	59
6.4	VISUALISIERUNG DES DATENSATZES	60
6.5	TRAINING DES MODELLS	61
6.6	ERHEBUNG DER MODELLPERFORMANCE	61
7	EVALUATION	62
7.1	DATENBESTAND	62
7.2	DATENSATZ	63
7.2.1	<i>Grundlegender Aufbau</i>	63
7.2.2	<i>Aufbereiteter Datensatz</i>	64
7.3	HYPERPARAMETEROPTIMIERUNG	65
7.3.1	<i>Experiment 1: Optimierung der Bildauflösung</i>	66
7.3.2	<i>Experiment 2: Optimierung der Batchgröße</i>	69
7.3.3	<i>Experiment 3: Auswahl des Optimierungsalgorithmus</i>	71
7.3.4	<i>Experiment 4: Optimierung der sekundären Hyperparameter</i>	76
7.4	FINALES MODELL	82
8	FAZIT UND AUSBLICK	87
9	LITERATURVERZEICHNIS	V
10	ANHANG	XIV

Abbildungsverzeichnis

Abbildung 1: Hamburger Hafen [vgl. Möll21].....	6
Abbildung 2: Stahlspundwand [vgl. Bars00].....	7
Abbildung 3: Kaimauer aus Stein [vgl. Kaim00].....	7
Abbildung 4: Dalben aus Holz [vgl. Bage22].....	7
Abbildung 5: Dalben aus Stahl [vgl. Dalb15].....	7
Abbildung 6: Klassische Programmierung vs. ML nach Chollet [vgl. Chol18, S.5].....	8
Abbildung 7: Aufbau eines künstlichen Neurons nach Glassner [vgl. Glas21, S.320].....	10
Abbildung 8: Aufbau eines CNNs nach Zschech et al. [vgl. ZSSP21, S.325].....	11
Abbildung 9: Transformer-Architektur nach Vaswani et al. [vgl. VSPU17, S.3].....	12
Abbildung 10: Trainingsprozess nach Glassner [vgl. Glas21, S.182].....	14
Abbildung 11: Verlustrückführung nach Chollet [vgl. Chol18, S.11].....	16
Abbildung 12: Gradient Descent nach Glassner [vgl. Glas21, S.355].....	16
Abbildung 13: Funktionsweise des Nesterov-Momentums nach Glassner [vgl. Glas21, S.414].....	18
Abbildung 14: Grid Search nach Liashchynskiy [vgl. LiLi19, S.6].....	21
Abbildung 15: Random Search nach Liashchynskiy [vgl. LiLi19, S.6].....	22
Abbildung 16: Zyklus eines genetischen Algorithmus nach Kramer [vgl. Kram17b, S.5].....	23
Abbildung 17: Intersect Over Union (IoU) nach Padilla et al. [vgl. PaLA20, S.2].....	25
Abbildung 18: Precision-Recall-Kurve.....	27
Abbildung 19: Anwendungen der Computer Vision nach Sharma et al. [vgl. SSLB22, S.2].....	29
Abbildung 20: Funktionsweise von YOLO nach Redmon et al. [vgl. RDGF15, S.780].....	31
Abbildung 21: YOLO-Architektur nach Redmon et al. [vgl. RDGF15, S. 781].....	32
Abbildung 22: Ablauf der durchzuführenden Arbeitsschritte.....	43
Abbildung 23: Aufbereitung des Rohmaterials.....	44
Abbildung 24: Datenaugmentierung.....	48
Abbildung 25: Parametrisierung des Modells.....	55
Abbildung 26: Aufnahme aus dem Emden Hafen.....	62
Abbildung 27: Anzahl an (annotierten) Bildern und Null-Beispielen.....	63
Abbildung 28: Kaimauer-Heatmap.....	64
Abbildung 29: Dalben-Heatmap.....	64
Abbildung 30: Aufbau des verarbeiteten Datensatzes.....	65
Abbildung 31: Experiment 3: Box-Verlust im Trainingsset.....	72
Abbildung 32: Experiment 3: Box-Verlust im Validierungsset.....	72
Abbildung 33: Experiment 3: Mean Average Precision (0.5).....	73
Abbildung 34: Experiment 3: Mean Average Precision (0.5:0.95).....	74
Abbildung 35: Experiment 3: Recall-Kurve: Adam-Optimierer.....	75
Abbildung 36: Fitnessfunktion nach Ultralytics [vgl. Ultr23b].....	77
Abbildung 37: Ergebnisse der Hyperparameteroptimierung.....	78
Abbildung 38: Experiment 4: Box-Verlust im Trainingsset.....	79
Abbildung 39: Experiment 4: Box-Verlust im Validierungsset.....	79
Abbildung 40: Experiment 4: Klassenverlust im Validierungsset.....	80
Abbildung 41: Experiment 4: Mean Average Precision (0.5).....	80
Abbildung 42: Vorhersagen des finalen Modells.....	84
Abbildung 43: Beispiele für fehlerhafte Klassifikationen [vgl. Welc00] und [vgl. Stre00].....	85
Abbildung 44: Beispiel für eine fehlerhafte Lokalisation [vgl. Sani14].....	86
Abbildung 45: Modellvorhersagen [vgl. Roev21] und [vgl. 50bi00].....	86
Abbildung 46: Experiment 3: Recall.....	XIV
Abbildung 47: Experiment 3: Klassenverlust im Trainingsset.....	XIV
Abbildung 48: Experiment 4: Klassenverlust im Trainingsset.....	XV
Abbildung 49: Experiment 4: Precision.....	XV
Abbildung 50: Experiment 4: Recall.....	XVI
Abbildung 51: Experiment 4: Mean Average Precision (0.5:0.95).....	XVI

Tabellenverzeichnis

Tabelle 1: YOLOv5-Modellvarianten nach Jocher [vgl. Joch20].....	53
Tabelle 2: Einstellungen der Datenaugmentierung.....	60
Tabelle 3: Experiment 1: Mean Average Precision (0.5).....	66
Tabelle 4: Experiment 1: Mean Average Precision (0.5:0.95).....	67
Tabelle 5: Experiment 1: Trainingsressourcen.....	68
Tabelle 6: Experiment 2: Vorhersagegenauigkeit.....	70
Tabelle 7: Experiment 2: Trainingsressourcen.....	70
Tabelle 8: Experiment 3: Vorhersagegenauigkeit.....	74
Tabelle 9: Evolution der Hyperparameter.....	81
Tabelle 10: Performance-Kennzahlen des finalen Modells.....	82

1 Einleitung

Dieses Kapitel soll dazu dienen, einen ersten Einblick in den Themenkomplex der automatisierten Objekterkennung auf Kamerabildern in der maritimen Domäne zu erhalten. Dafür wird zunächst auf die hier behandelte Problemstellung eingegangen, um eine Motivation für das Thema zu schaffen. Anschließend werden die Ziele der Arbeit dargelegt, die erfüllt werden müssen, um die beschriebene Problemstellung zu adressieren, und die daraus resultierende Forschungsfrage zu beantworten.

1.1 Problemstellung

Die Nutzung von Kameras zur Überwachung von Hafenanlagen, Küstenstreifen, Flüssen und Gebieten auf hoher See ist bereits seit Jahrzehnten eine etablierte Praktik. Durch die Kamerasysteme sollen gesperrte Bereiche im zivilen und militärischen Umfeld überwacht und so illegale Aktivitäten vorzeitig erkannt werden [vgl. SnFP09] [vgl. HACS08] [vgl. BuEs10]. Seit dem Aufkommen der *Künstlichen Intelligenz* (KI) haben sich in diesem Bereich eine ganze Reihe von neuen Anwendungsmöglichkeiten ergeben. So wird aktuell daran geforscht, die Navigation von Schiffen mithilfe von Kamerasystemen und KI-Algorithmen zu automatisieren. Ähnliche Anstrengungen sind in der Automobilbranche erkennbar [vgl. KOVA21] [vgl. WoFM19, S.4].

Hochautomatisierte Schiffe sollen in der Zukunft in die Lage versetzt werden, ohne menschliches Eingreifen auf dem Wasser zu navigieren und in eigenständiger Weise eine Vielzahl an unterschiedlichen Aufgaben auszuführen [vgl. SpSS20, S.1]. Derartige Systeme haben damit das Potenzial, Schaden von Personen, Wasserfahrzeugen und der maritimen Infrastruktur abzuwenden. Dadurch, dass bei der Navigation alle Aktionen von einem Algorithmus übernommen werden, lässt sich die Anzahl an Unfällen, die auf menschliche Fehlentscheidungen zurückzuführen sind, reduzieren. Mit der Einführung von hochautomatisierten Schiffen würde sich damit auch in der herkömmlichen Schifffahrt die Sicherheit der Besatzungen an Board verbessern.

Laut der Unfallstatistik der *European Maritime Safety Organization* (EMSA) kam es im Jahr 2020 zu mehr als 2.800 Schiffsunfällen in Europa [vgl. Euro21, S.3]. Bei diesen Unfällen waren mehr als 3.000 Schiffe involviert – 675 Personen wurden verletzt und 38 Menschen kamen dabei ums Leben [vgl. Euro21, S.3]. Weltweit gesehen sind neben Schäden an Maschinen vor allem Kollisionen (3.134) und Kontakte (2.029) mit anderen Schiffen und feststehenden Hindernissen als Unfallgrund zu identifizieren [vgl. Alli22, S.17]. In Europa war in den Jahren 2014 bis 2020 mit 53 Prozent ein Großteil der Schiffsunfälle auf menschliches Versagen zurückzuführen; weitere 35 Prozent entstanden

durch Fehler in den technischen Systemen oder durch den Ausfall von Equipment [vgl. Euro21, S.26].

Damit hochautomatisierte Schiffe unfallfrei navigieren können, müssen sie in die Lage versetzt werden, Hindernisse auf ihrer Route zu erkennen. Eine weitverbreitete Möglichkeit zur automatisierten Hinderniserkennung besteht darin, Algorithmen zur Bilderkennung zu verwenden. Für die Bilderkennung werden seit einigen Jahren zunehmend KI-basierte Systeme auf der Basis von neuronalen Netzen eingesetzt [vgl. Chol18, S.17]. Diese Algorithmen nutzen Bilddaten aus den Kamerasystemen an Bord, um potenzielle Hindernisse vorzeitig zu erkennen. Diese Informationen können der Besatzung an Bord entscheidungsunterstützend zur Verfügung gestellt werden.

Im Bereich der KI-basierten Hinderniserkennung gibt es Produkte, die bereits heute kommerziell vertrieben werden und auf Schiffen zum Einsatz kommen. Als eine von mehreren Lösungen ist hier das Produkt „OrcaAI“ zu nennen. OrcaAI ist ein System zur Kollisionsfrüherkennung, die es den Reedereien ermöglichen soll, Bojen und unterschiedliche Schiffsarten wie Containerschiffe oder Schlepper auf Kamerabildern mithilfe eines KI-Systems automatisiert zu erkennen [vgl. Orca00]. Die Anwendung verbindet GPS-, AIS- und Bildinformationen, um die Gefahr einer Kollision zu errechnen [vgl. Orca00]. Das Unternehmen konnte damit die Anzahl an gefährlichen Nahbegegnungen für seine Kunden nach eigenen Angaben zufolge um bis zu 29 Prozent reduzieren [vgl. Orca00]. Ein weiterer Anbieter ist das Unternehmen „Sea Machines“ mit seinem Produkt „AI-ris“. Sea Machines nutzt Kamerasensoren und einen KI-Algorithmus, um motorisierte Schiffe, Segelboote, Meereslebewesen wie Wale und andere Objekte wie Bojen auf der Meeresoberfläche automatisiert zu klassifizieren, zu tracken und zu lokalisieren [vgl. Ai-r22]. Das Frühwarnsystem soll das Situationsbewusstsein der Besatzung an Bord verbessern und diese frühzeitig über mögliche Gefahrensituationen informieren [vgl. Ai-r22].

Derartige Angebote richten sich in erster Linie an große Reedereien und finden primär in der Frachtschiffahrt auf hoher See Anwendung. Für die Hinderniserkennung in Hafengebieten sind diese Systeme weniger gut geeignet, da sie nicht auf die Erkennung der dort vorkommenden Objekte bzw. Hindernisse optimiert wurden. Während auf hoher See die Erkennung von Schiffen im Vordergrund steht, stellen in Häfen und der unmittelbaren Küstenregion auch feststehende Hindernisse – sogenannte *statische Hafengebiete* – wie Kaimauern, Dalben und Bojen eine Gefahr dar. Für Schiffe, die sich ausschließlich im Hafenbecken oder sehr nah an der Küste aufhalten, reichen die Möglichkeiten zur Kollisionsfrüherkennung bei den bestehenden Lösungen somit nicht aus.

Nicht nur in der freien Wirtschaft, sondern auch in der Forschung, wird die Erkennung von statischen Hafengebäuden nur selten behandelt. In den letzten Jahren war das Interesse an der Erforschung der Objekterkennung in maritimen Umgebungen gering [vgl. MKJT19, S.1]. Dies hat zur Folge, dass heute nur wenige geeignete Datensätze für die Erkennung von maritimen Objekten im Allgemeinen und von statischen Hafengebäuden im Besonderen existieren, die als Ausgangspunkt für weitere Forschungsaktivitäten und das Training von eigenen KI-Modellen genutzt werden könnten. In der maritimen Domäne existieren zwei Arten von öffentlich verfügbaren Datensätzen. Zum einen gibt es sogenannte *universelle Datensätze*. Diese Datensätze bestehen aus einer großen Anzahl an vorannotierten Bildern und können bis zu 1000 unterschiedliche Objekte erkennen. Damit lassen sie sich für die unterschiedlichsten Anwendungsfälle einsetzen. Zum anderen gibt es auch sogenannte *domänenspezifische Datensätze*. Diese repräsentieren in der Regel deutlich weniger Bilder bzw. Klassen und sind auf die Erkennung von Objekten in ausgewählten Domänen spezialisiert.

Bei der Objekterkennung in der maritimen Domäne bestehen eine Reihe von domänenspezifischen Herausforderungen. Vor allem bei Anwendungsfällen, in denen Objekte auf dem Wasser erkannt werden sollen, schwanken die zu erkennenden Objekte stark in ihrer Größe, da sie aus unterschiedlichen Entfernungen aufgenommen werden [vgl. SpSS20, S.2] [vgl. ISZL21, S.2]. Dazu kommt, dass die zu detektierenden Objekte aufgrund von Überschneidungen mit anderen Objekten oder durch das Abschneiden am Bildrand nicht immer vollständig von der Kamera erfasst werden können [vgl. SpSS20, S.2] [vgl. ISZL21, S.2 und 6]. Weiterhin werden KI-Algorithmen in der maritimen Domäne in dynamischen Umgebungen eingesetzt. Aus diesem Grund müssen die verwendeten Trainingsdatensätze verschiedene Tageszeiten und Lichtverhältnisse, Wetterbedingungen wie Sonnenschein, Bewölkung, Niederschlag oder Nebel sowie atmosphärische Effekte abbilden [vgl. SpSS20, S.2] [vgl. ISZL21, S.2]. Nicht zuletzt kann auch die Qualität der aufgenommenen Bilder durch die Wellenbewegungen und der daraus resultierenden Bewegungsunschärfe beeinträchtigt werden [vgl. SpSS20, S.2]. All diese domänenspezifischen Umgebungsbedingungen werden von den universellen Datensätzen entweder gar nicht oder nur teilweise adressiert.

Im Bereich der domänenspezifischen Datensätze besteht im maritimen Bereich nur eine sehr geringe Auswahl an geeigneten Kandidaten [vgl. ISZL21, S.4] [vgl. SpSS20, S.2]. Zum aktuellen Zeitpunkt existieren lediglich zwei Datensätze, die sich für das Training von KI-Modellen auf der Basis von neuronalen Netzen eignen – das „Singapur Maritime Dataset“ (SMD) und der sogenannte „SeaShips7000“-Datensatz [vgl. SpSS20, S.4]. Alle

anderen Datensätze sind entweder zu klein, nicht auf Objekterkennungs-Anwendungsfälle optimiert oder verwenden Satellitenbilder [vgl. SpSS20, S.4]. Sowohl das SMD als auch der SeaShips7000-Datensatz konzentrieren sich allerdings auf die Erkennung von unterschiedlichen Schiffstypen [vgl. SpSS20, S.4]. Das SMD enthält insgesamt zehn Klassen; davon sechs Schiffs-Klassen [vgl. ISZL21, S.4]. Als statisches Hafengebiet wird hier nur die Klasse „Boje“ repräsentiert [vgl. MKJT19, S.3]. Der Datensatz SeaShips7000 bildet ausschließlich Schiffstypen ab und ist auf die Objekterkennung auf Flüssen spezialisiert [vgl. ISZL21, S.4] [vgl. SWWD18, S.2596] [vgl. SpSS20, S.4].

Dadurch, dass statische Hafengebiete in den existierenden Datensätzen nur sehr unzureichend repräsentiert sind, besteht in der Forschung auch nur begrenztes Wissen darüber, inwiefern sich KI-basierte Algorithmen für die Erkennung derartiger Objekte eignen. Um diese Forschungslücken zu schließen und die in diesem Abschnitt beschriebene Problemstellung zu adressieren, soll in dieser Arbeit die folgende Forschungsfrage beantwortet werden:

Wie lassen sich statische Hafengebiete auf Kamerabildern automatisiert identifizieren?

1.2 Ziele der Arbeit

Der Fokus dieser Arbeit liegt auf der Verwendung von KI-Algorithmen zur automatisierten Bilderkennung bei hochautomatisierten Schiffen. Es soll untersucht werden, inwiefern sich derartige Algorithmen einsetzen lassen, um eine automatisierte Hinderniserkennung im Hafengebiet zu ermöglichen. Dabei soll sich ausschließlich auf die Erkennung von statischen Hafengebieten konzentriert werden. Die Identifizierung von dynamischen Hafengebieten wie Schiffen wird explizit nicht betrachtet, da in diesem Bereich bereits ein erheblicher Forschungsaufwand betrieben wurde.

Um die Forschungsfrage zu beantworten, sollen im Rahmen dieser Arbeit die folgenden zwei Teilziele erfüllt werden:

Teilziel 1: Aufbau eines domänenspezifischen Datensatzes für die Erkennung von statischen Hafengebieten auf der Basis von Kamerabildern aus dem Emdener Hafen.

Teilziel 2: Auswahl, Parametrisierung und Evaluation eines geeigneten KI-Modells für die automatisierte Erkennung von statischen Hafengebieten auf der Basis des zuvor erstellten Datensatzes.

Wie bereits in Abschnitt 1.1 erläutert, existiert aktuell kein Datensatz, der sich für die Erkennung von statischen Hafengebieten eignet. Um die Eignung von KI-Modellen untersuchen zu können, muss deshalb zunächst ein geeigneter domänenspezifischer Datensatz aufgebaut werden, der diese statischen Hafengebiete in geeigneter Weise repräsentiert (siehe Teilziel 1).

Im Rahmen von Teilziel 2 wird das Ziel verfolgt, ein geeignetes KI-Modell für die automatisierte Erkennung von statischen Hafengebieten auszuwählen, zu parametrisieren und dessen Leistungsfähigkeit mithilfe des im Rahmen von Teilziel 1 erstellten Datensatzes zu evaluieren. Um eine begründete Entscheidung bei der Auswahl des Modells treffen zu können, müssen dafür zunächst geeignete Auswahlkriterien definiert werden. Anschließend müssen die Modellparameter eingestellt werden, damit eine optimale Anpassung an den vorliegenden Anwendungsfall erreicht werden kann. Um eine geeignete Konfiguration zu finden, soll der Einfluss unterschiedlicher Parameter auf die Gesamtleistung des Modells untersucht werden. Danach lässt sich die Vorhersagegenauigkeit und -geschwindigkeit des trainierten Modells anhand von geeigneten Performance-Metriken bewerten und mit den in Kapitel 4 erhobenen Modellanforderungen abgleichen. Durch dieses Vorgehen lassen sich die Stärken und Schwächen des finalen Modells quantifizieren und untersuchen, inwiefern es sich für die automatisierte Erkennung von statischen Hafengebieten eignet. Im Rahmen dieses Teilziels soll außerdem untersucht werden, ob die hier erlangten Untersuchungsergebnisse auch auf andere Anwendungsfälle übertragbar sind. Dafür soll analysiert werden, inwieweit das Modell in der Lage ist, statische Hafengebiete auf bisher ungesesehenen Bildern aus anderen Hafengebieten korrekt zu erkennen.

Das konkrete Konzept zur Umsetzung der oben beschriebenen Teilziele und die damit verknüpften Arbeitsschritte sind in Kapitel 5 zu finden. Das dort beschriebene Konzept stellt sicher, dass begründete Entscheidungen beim Aufbau des Datensatzes und der Auswahl bzw. Implementierung des KI-Modells getroffen werden und garantiert, dass die in dieser Arbeit erlangten Untersuchungsergebnisse belastbar sind und reproduzierbar bleiben.

2 Grundlagen

In diesem Kapitel wird zunächst geklärt, was unter dem Begriff der statischen Hafenobjekte zu verstehen ist, welche Objekte in diese Kategorie fallen und welche Aufgaben durch sie im Hafengebiet wahrgenommen werden. Im zweiten Teil werden die wichtigsten Konzepte und Theorien aus dem Bereich der Künstlichen Intelligenz erläutert. Zu guter Letzt wird der Themenbereich der Computer Vision zum Zwecke der KI-basierten Bilderkennung genauer betrachtet.

2.1 Statische Hafenobjekte

Innerhalb von Hafenanlagen kommen neben beweglichen Hindernissen wie Schiffen auch statische bzw. feststehende Hafenobjekte vor. Eine Kollision mit diesen Objekten kann zu erheblichen Schäden am Schiff und an der maritimen Infrastruktur führen.



Abbildung 1: Hamburger Hafen [vgl. Möll21]

Abbildung 1 zeigt den Hamburger Hafen aus der Vogelperspektive. Entlang des eigentlichen Hafenbeckens befinden sich diverse Anlegestellen für Frachtschiffe an den Containerterminals, an denen die Schiffe abgefertigt werden können. Dieses senkrecht zum Ufer verlaufende Bauwerk wird auch als *Kai* bezeichnet [vgl. Quay00].

Bei einem Kai wird in der Regel eine senkrechte Mauer – auch *Kaimauer* genannt – angelegt, um die Anlegestelle vom Hafenbecken abzugrenzen [vgl. Begr17a]. Kaimauern sind massive Bauwerke, die oftmals mithilfe von Stahlspundwänden (siehe Abbildung 2) oder Stahlbetonplatten errichtet werden [vgl. Begr17a]. Häufig werden Kaimauern auch zusätzlich mit Steinplatten oder Holz verkleidet (siehe Abbildung 3).



Abbildung 2: Stahlspundwand [vgl. Bars00]



Abbildung 3: Kaimauer aus Stein [vgl. Kaim00]

Ein weiteres wichtiges Bauwerk in Hafengebieten ist der sogenannte *Pier*. Ein Pier verläuft im Gegensatz zu einem Kai nicht parallel zum Ufer, sondern ragt in das Hafenbecken hinein und wird für das Anlegen von Schiffen genutzt [vgl. Iala00]. Kleinere Piere, Schwimmstege und Schwimmanleger sind in Häfen oft von sogenannten *Dalben* umgeben, um die Schiffe festzumachen und diese zu schützen [vgl. Dolp00]. Außerdem können Dalben als Markierungen in der Fahrrinne verwendet werden [vgl. Begr17b]. Neben den klassischen Holz-Dalben (siehe Abbildung 4) werden diese heute zunehmend aus Stahl gefertigt (siehe Abbildung 5).



Abbildung 4: Dalben aus Holz [vgl. Bage22]



Abbildung 5: Dalben aus Stahl [vgl. Dalb15]

2.2 Künstliche Intelligenz

In diesem Kapitel soll ein Überblick über die aktuell vorherrschenden KI-Technologien gegeben werden. Dafür werden zunächst die Konzepte des Machine- und des Deep Learnings näher erläutert. In Kapitel 2.2.3 werden anschließend die wichtigsten Arten von neuronalen Netzwerken für die Objekterkennung auf Bildern herausgestellt. Kapitel 2.2.4 untersucht die Möglichkeiten zum Training der Validierung von KI-Modellen. Zu guter Letzt erfolgt die Beschreibung der relevanten Metriken für die Performance-messung bei KI-Modellen aus dem Bereich der Bilderkennung.

2.2.1 Machine Learning

Das *Machine Learning* (ML) stellt einen zentralen Themenkomplex in der Künstlichen Intelligenz dar und ist aktuell die populärste Subdomäne in diesem Forschungsfeld [vgl. Chol18, S.6]. ML-Algorithmen sind in der Lage, aus vergangenen Erkenntnissen und Erfahrungen zu lernen [vgl. Witt19, S.24]. Damit unterscheiden sie sich stark von den traditionellen Programmieransätzen aus den 1980er Jahren. Abbildung 6 zeigt diesen Unterschied auf. Bei der klassischen Programmierung erfolgte die Regeldefinition für die spätere Entscheidungsfindung mithilfe von statischem Code [vgl. Chol18, S.6].

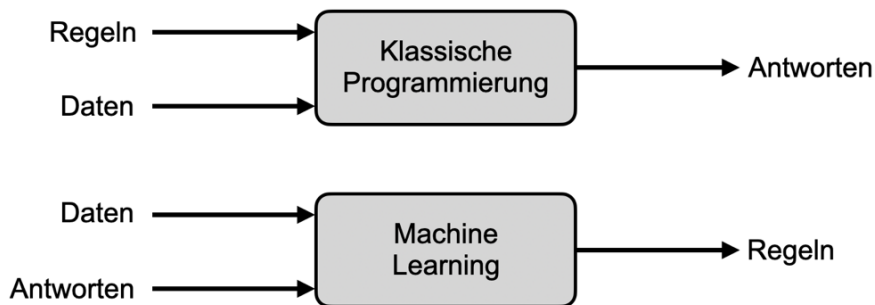


Abbildung 6: Klassische Programmierung vs. ML nach Chollet [vgl. Chol18, S.5]

Beim Machine Learning leitet ein Algorithmus diese Entscheidungsregeln dagegen selbstständig ab, indem er eine Verbindung zwischen den bekannten Eingabedaten und den gewünschten Ausgabedaten knüpft [vgl. Chol18, S.5]. Aus den erlernten Mustern erstellt der Algorithmus ein sogenanntes *Modell*, das diese Informationen enthält [vgl.

HDPR17, S.8]. ML-Algorithmen sind nicht nur in der Lage, Muster in den Daten eigenständig abzuleiten, sie können ihr vorhandenes Wissen in Form von bestehenden Entscheidungsregeln bei Bedarf auch dynamisch und iterativ anpassen [vgl. Witt19, S.25].

2.2.2 Deep Learning

Das *Deep Learning* (DL) kann als eine Subdomäne des Machine Learnings verstanden werden [vgl. Glas21, S.3]. Im Gegensatz zu den ML-Ansätzen bestehen Deep-Learning-Algorithmen aus sogenannten *künstlichen Neuronen*, die in einem Verbund – einem sogenannten *künstlichen neuronalen Netzwerk* (KNN) – miteinander verknüpft sind [vgl. Glas21, S.11] [vgl. Chol18, S.8]. Durch das Training eines künstlichen neuronalen Netzwerks können die Zusammenhänge in den Daten automatisiert abgeleitet werden; jedes künstliche Neuron ist dabei für sich in der Lage, Merkmale (auch *Features* genannt) aus den Eingangsdaten zu extrahieren [vgl. Glas21, S.6 und 12] [vgl. Chol18, S.18] [vgl. XCZW21, S.2].

Deep-Learning-Algorithmen haben durch die Einführung von immer leistungsstärkerer Hardware wie Grafikkarten und speziellen Deep-Learning-Chips immer weiter an Bedeutung gewonnen [vgl. Chol18, S.20 f.] [vgl. ISZL21, S.3]. Ein weiterer Treiber war das Aufkommen und die wachsende Verfügbarkeit von großen Datensätzen (z.B. „PASCAL VOC“, „ImageNet“ oder „MS COCO“) für das Training und die Evaluation der Modelle [vgl. Chol18, S.6 und 21] [vgl. ISZL21, S.3] [vgl. ZCSG23, S.4 f.]. Durch diese Einflussfaktoren konnten die Vorhersagegenauigkeit und die Performance von DL-Modellen insbesondere in der Bildverarbeitung und Objekterkennung in den letzten Jahren stark verbessert werden [vgl. ISZL21, S.3] [vgl. HDPR17, S.6] [vgl. SpSS20, S.1 f.].

Abbildung 7 visualisiert den grundlegenden Aufbau eines künstlichen Neurons. Jedes künstliche Neuron nimmt Eingangswerte aus einem anderen vorgeschalteten Neuron entgegen und führt auf diesen Werten bestimmte Berechnungen durch. In der Regel werden die Eingangswerte mit den dazugehörigen *Gewichten* multipliziert und anschließend aufsummiert. Außerdem besitzt jedes Neuron einen sogenannten *Bias*, der in diese Summe mit eingerechnet wird. Im Laufe des Trainings werden die Gewichte so lange verändert, bis das gewünschte Ergebnis bei einer gegebenen Eingabe erreicht wird. [vgl. Glas21, S.12 und 318]

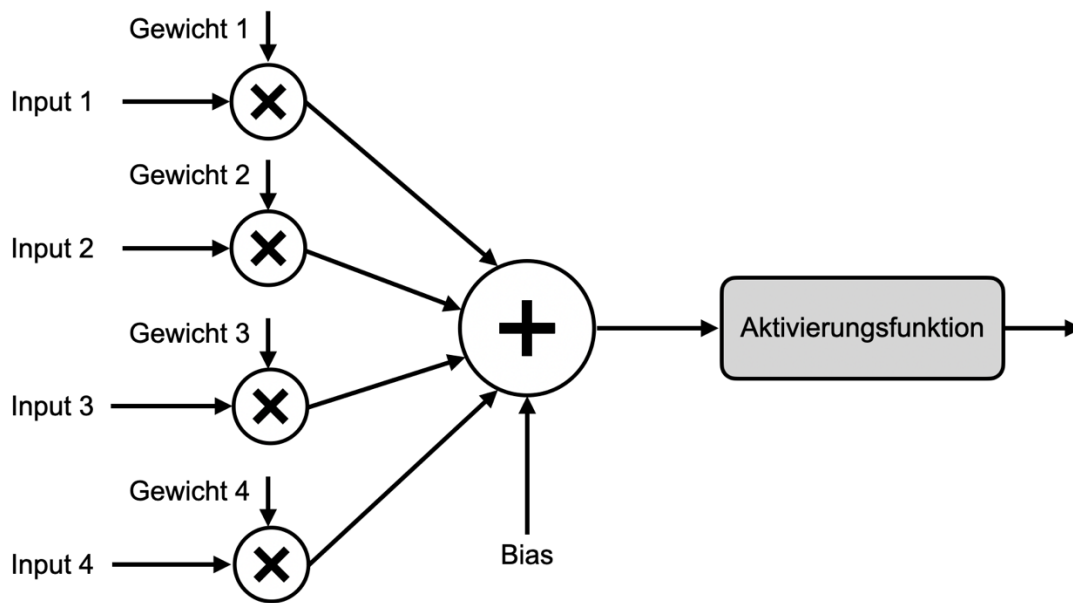


Abbildung 7: Aufbau eines künstlichen Neurons nach Glassner [vgl. Glas21, S.320]

Die Kommunikation zwischen den künstlichen Neuronen wird durch sogenannte *Aktivierungsfunktionen* gesteuert. Eine derartige Funktion gibt vor, ab wann ein Signal stark genug ist, um an das nächste Neuron weitergeleitet zu werden [vgl. Wenn20, S.19] [vgl. HDPR17, S.8 f.]. Insbesondere die Kategorie der *glatten Aktivierungsfunktionen* spielt im Deep Learning eine wichtige Rolle. Im Gegensatz zu allen anderen Arten von Aktivierungsfunktionen (z.B. *lineare* oder *stückweise lineare Funktionen*) sind die glatten Aktivierungsfunktionen an jeder Stelle ableitbar [vgl. Glas21, S.339]. Die Differenzierbarkeit einer Funktion ist notwendig, damit ein neuronales Netzwerk lernen kann [vgl. Glas21, S.339]. Nur durch die Verwendung einer mehrschichtigen Architektur und die Möglichkeit der Verwendung von nicht-linearen Aktivierungsfunktionen sind Deep-Learning-Algorithmen den reinen ML-Ansätzen heute deutlich überlegen, da sie dadurch komplexere Muster erlernen können [vgl. Xin22, S.125] [vgl. HDPR17, S.9].

2.2.3 Arten von künstlichen neuronalen Netzwerken

In diesem Kapitel werden *konvolutionale neuronale Netzwerke* (CNNs) und *Transformer-Netzwerke* als die beiden wichtigsten Arten von neuronalen Netzwerken erläutert, die für die Bilderkennung eingesetzt werden können.

2.2.3.1 Konvolutionale neuronale Netzwerke

Konvolutionale neuronale Netzwerke waren die neuronalen Netze, die in der jüngsten Vergangenheit zu den größten Performancesteigerungen in der Bildverarbeitung im Allgemeinen und der Objekterkennung im Besonderen beigetragen haben [vgl. ISZL21, S.1] [vgl. LeRO20, S.416] [vgl. Chol18, S.17]. Abbildung 8 zeigt den generellen Aufbau eines konvolutionalen neuronalen Netzwerkes auf. Die CNN-Architektur lässt sich in zwei wesentliche Bereiche unterteilen: den *Feature-Extraktor* (auch *Backbone* genannt) und den *Klassifikator* [vgl. ZSSP21, S.324 f.].

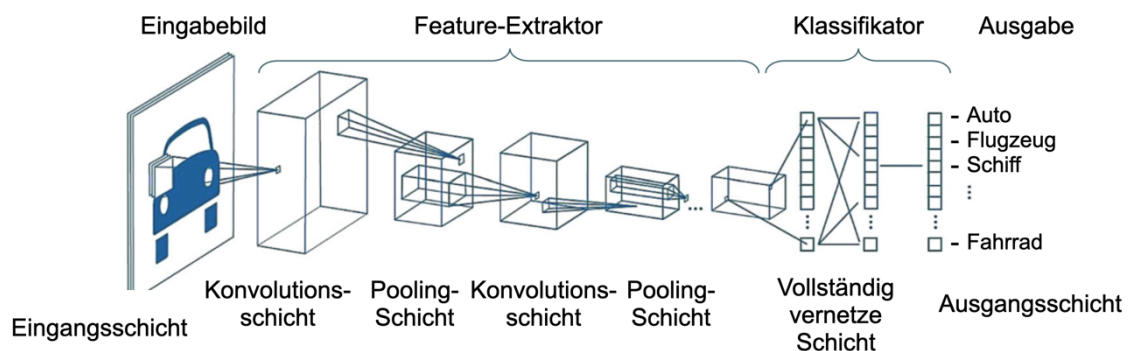


Abbildung 8: Aufbau eines CNNs nach Zschech et al. [vgl. ZSSP21, S.325]

Im Rahmen der Feature-Extraktion kommen sogenannte *Faltungsfunktionen* zum Einsatz, die in *Konvolutionsschichten* zusammengefasst werden [vgl. ZSSP21, S.324]. Diese Funktionen haben den Zweck, zunehmend komplexe lokale Merkmale wie Kanten, Texturen oder Formen aus einem Bild zu extrahieren [vgl. ZSSP21, S.324]. Um dies zu erreichen, unterteilen die Faltungsfunktionen die Bilder in kleinere Bereiche, scannen diese und speichern die wichtigsten Informationen in einer sogenannten *Feature-Map* ab [vgl. ZSSP21, S.324] [vgl. ISZL21, S.2] [vgl. Glas21, S.453]. Anschließend werden sogenannte *Pooling-Schichten* verwendet, um die Komplexität der Feature-Maps zu reduzieren, damit weniger Speicherplatz verbraucht wird und Bilder schneller vom Netz verarbeitet werden können [vgl. ZSSP21, S.324] [vgl. Glas21, S.453]. Die Pooling-Schicht achtet darauf, dass die in den Feature-Maps enthaltenen dominanten Bildmerkmale bei der Dimensionsreduktion nicht verloren gehen [vgl. Glas21, S.453].

Nach der Feature-Extraktion wird versucht, das Bild anhand der extrahierten Bildinformationen zu klassifizieren [vgl. ZSSP21, S.325]. Dabei werden sogenannte *vollständig vernetzte Schichten* eingesetzt, um die extrahierten Merkmale aus den vorherigen

Schichten zu kombinieren und die auf den Bildern abgebildeten Objekte unter der Verwendung einer sogenannten *Softmax-Schicht* auf der Basis von Zugehörigkeitswahrscheinlichkeiten einer bestimmten Klasse zuzuordnen [vgl. ISZL21, S.2] [vgl. ZSSP21, S.325].

2.2.3.2 Transformer-Netzwerke

Ein Transformer-Netzwerk nutzt einen sogenannten *Aufmerksamkeitsmechanismus*, um sequenzielle Daten zu verarbeiten. Im Gegensatz zu CNNs verzichten Transformer vollständig auf Konvolutionsschichten, um den Zusammenhang zwischen den Ein- und Ausgaben in einer Sequenz zu speichern. Transformer wurden ursprünglich für das *Natural Language Processing* (NLP) eingeführt, können heute allerdings auch für die Bildererkennung verwendet werden. [vgl. VSPU17, S.1 ff.] [vgl. Glas21, S.582]

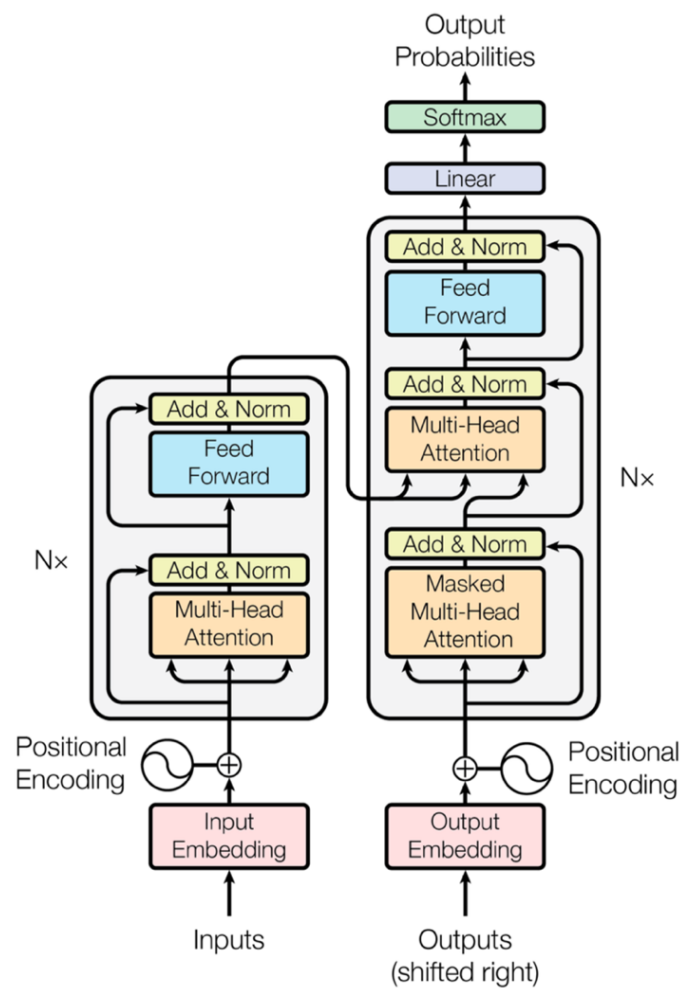


Abbildung 9: Transformer-Architektur nach Vaswani et al. [vgl. VSPU17, S.3]

Abbildung 9 zeigt den Aufbau eines klassischen Transformer-Netzwerkes auf. Es besteht aus einem *Encoder* und einem *Decoder*, die jeweils aus sechs übereinandergestapelten Schichten bestehen. Der Encoder nimmt eine Input-Sequenz entgegen und übersetzt diese in einen n-dimensionalen Vektor, der selbst wieder vom Decoder in eine Output-Sequenz übersetzt wird. In jeder Schicht des Decoders kommen diverse Subschichten in Form von Aufmerksamkeits-Modulen und vollständig vernetzten neuronalen Netzen zum Einsatz, die anschließend durch eine weitere Schicht normalisiert werden. [vgl. VSPU17, S.2 ff.]

Nachdem die Input- und Output-Sequenzen in n-dimensionalen Vektoren übersetzt wurden, werden die relativen Positionen der Elemente in der Sequenz festgehalten und dem Vektor hinzugefügt. Als Ergebnis gibt der Transformer Wahrscheinlichkeiten aus, mit der bestimmte Elemente in der Output-Sequenz vorkommen werden. Beim Aufmerksamkeitsmechanismus werden die Anfrage Q (n-dimensionaler Vektor eines der Elemente in der Sequenz) und Schlüssel-Wert-Paare K und V (n-dimensionale Vektoren aller Elemente in der Sequenz) auf einen Output (gewichtete Summe der Werte) abgebildet. Die Gewichtungen der Werte erlernt der Transformer im Laufe des Trainings. [vgl. VSPU17, S.2 ff.] [vgl. Glas21, S.586]

Transformer-Netzwerke konnten in den letzten Jahren große Erfolge im Deep Learning und insbesondere der Objekterkennung verzeichnen, indem sie die Schwächen der klassischen CNNs, wie deren eingeschränkte Kontextualisierung von Bildmerkmalen, adressieren. Während sich die klassischen CNNs auf die lokale Feature-Extraktion fokussieren, gelingt es Transformern, die Beziehung (z.B. Positionsinformationen) eines jeden Pixels in der Eingabe zu jedem anderen Pixel im Bild zu überblicken. Durch diese Eigenschaft bleibt der globale Kontext aller Features im Bild erhalten. [vgl. ZCSG23, S.4] [vgl. ZMXF22, S.2]

2.2.4 Training und Validierung von ML-Modellen

In diesem Abschnitt das grundsätzliche Prinzip für das Training von ML-Modellen erläutert. Dabei wird auf die *Fehlerrückführung* als zentrales Konzept im Machine Learning eingegangen. Anschließend werden verschiedene Optimierungsalgorithmen zur Optimierung des Trainingsprozess vorgestellt. Im letzten Abschnitt wird auf das Konzept der *Hyperparameteroptimierung* eingegangen und dessen Bedeutung für die Optimierung von ML-Modellen untersucht.

2.2.4.1 Trainingsprozess

In der Praxis werden Datensätze in der Regel in drei unterschiedliche Teildatensätze aufgeteilt: ein Trainingsset, ein Validierungsset und ein Testset. Das Trainingsset dient dazu, die Muster bzw. Zusammenhänge zwischen den Merkmalen im Datensatz zu erlernen. Das Validierungsset wird dazu verwendet, nach jedem Trainingsdurchlauf (auch *Epoche* genannt) eine Performanceschätzung des Modells durchzuführen. Es ist auch nützlich, um die automatisierte Auswahl von *Hyperparametern* zu unterstützen (mehr dazu in Kapitel 2.2.4.3). Das Testset dient schließlich dazu, die eigentliche Performance eines KI-Modells anhand von ungesehenen Bildern zu evaluieren. [vgl. Glas21, S.187 f.]

Eine der wichtigsten Anwendungsgebiete im ML-Algorithmen ist die Klassifizierung [vgl. IMAR22, S.156]. Bei diesem Verfahren extrahiert der Algorithmus die Regeln zur Klassenbildung aus vorhandenen Daten-Wert-Paaren [vgl. Chol18, S.5]. Da es sich bei der Klassifikation um eine *überwachte Lernmethode* handelt, müssen die Klassenlabel vor dem Training manuell annotiert werden.

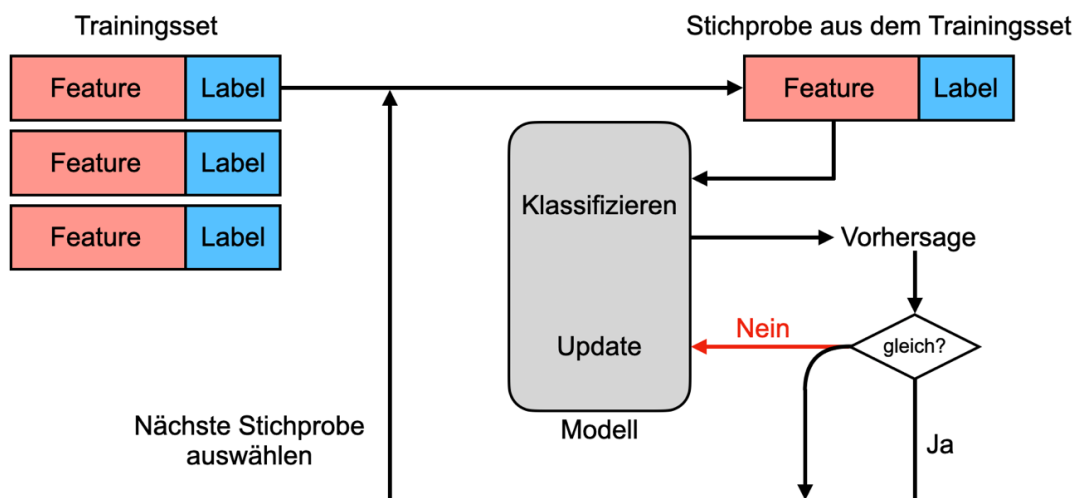


Abbildung 10: Trainingsprozess nach Glassner [vgl. Glas21, S.182]

Abbildung 10 zeigt, wie der Trainingsprozess bei einer Klassifizierung abläuft. In der Regel werden die Modellparameter vor dem Training mit zufälligen Werten initialisiert. Im Verlauf des Trainings werden die Feature-Label-Paare dann nacheinander aus dem Trainingsset extrahiert. In diesem Fall entspricht ein Feature zum Beispiel den numerischen Informationen eines Bildausschnittes, die einer bestimmten Klasse zugeordnet werden sollen. Anschließend versucht das Modell dem vorliegenden Feature eine

Klasse zuzuordnen und gleicht seine eigene Vorhersage mit dem vorannotierten Klassenlabel (auch *Grundwahrheit* genannt) ab. Stimmen die beiden Klassifizierungen überein, wird die nächste Stichprobe aus dem Trainingsset ausgewählt. Ist dies nicht der Fall, werden die internen Modellparameter angepasst; anschließend wird die nächste Stichprobe für die Klassifizierung herangezogen. Dieser Vorgang wird so lange wiederholt, bis alle Feature-Wert-Paare aus dem Trainingset verarbeitet wurden. Es ist üblich, ein Modell für mehrere Epochen zu trainieren, damit es die Objekte auf den Bildern zuverlässig erkennen kann. Mit einer steigenden Anzahl an vielfältigen Trainingsbildern wird das Modell robuster und kann auch auf bisher ungesesehenen Bildern korrekte Vorhersagen treffen. [vgl. Glas21, S.182 f.]

2.2.4.2 Fehlerrückführung und Optimierungsalgorithmen

Wie bereits erläutert, erlernt ein Deep-Learning-Modell beim überwachten Lernen Zusammenhänge zwischen den gegebenen Eingangsdaten und den gewünschten Ausgabedaten, indem die Gewichte an den künstlichen Neuronen iterativ angepasst werden. Um zu entscheiden, wie diese Gewichte verändert werden müssen, wird eine sogenannte *Verlustfunktion* verwendet [vgl. Chol18, S.10]. Diese Zielfunktion berechnet einen sogenannten *Verlust-Score*, der die Differenz zwischen den tatsächlichen und den vorhergesagten Ausgaben eines neuronalen Netzes quantifiziert [vgl. Glas21, S.352] [vgl. Chol18, S.10]. Bei einer Bildklassifizierung würde dieser Wert angeben, wie gut die vom neuronalen Netzwerk gemachten Vorhersagen mit der Grundwahrheit übereinstimmen; umso niedriger der Verlust-Score ist, desto höher ist die Vorhersagegenauigkeit des Modells [vgl. Glas21, S.353]. Ziel eines jeden Modells ist es demnach, die Verlustfunktion zu minimieren.

Abbildung 11 zeigt auf, wie die berechneten Verlust-Scores dazu verwendet werden, die Gewichte der Verbindungen in einem neuronalen Netz iterativ anzupassen. Nach jedem Trainingsdurchlauf wird der errechnete Verlust-Score als Feedback-Signal von hinten nach vorne durch das Netz zurückgereicht [vgl. Chol18, S.11]. Dieser Prozess wird auch als Fehlerrückführung oder *Backpropagation* bezeichnet [vgl. Chol18, S.11]. Die eigentliche Anpassung der Gewichte erfolgt durch einen sogenannten *Optimierungsalgorithmus* – auch *Optimierer* genannt –, der die Fehlerrückführung implementiert [vgl. Chol18, S.11]. Die Art und Weise der Netzanpassung hängt von der verwendeten Verlustfunktion ab (z.B. Regression oder Klassifizierung) [vgl. Chol18, S.60]. Das Ziel eines Optimierers besteht darin, die Genauigkeit eines Deep-Learning-Modells zu verbessern und den Lernprozess zu beschleunigen [vgl. Glas21, S.387].

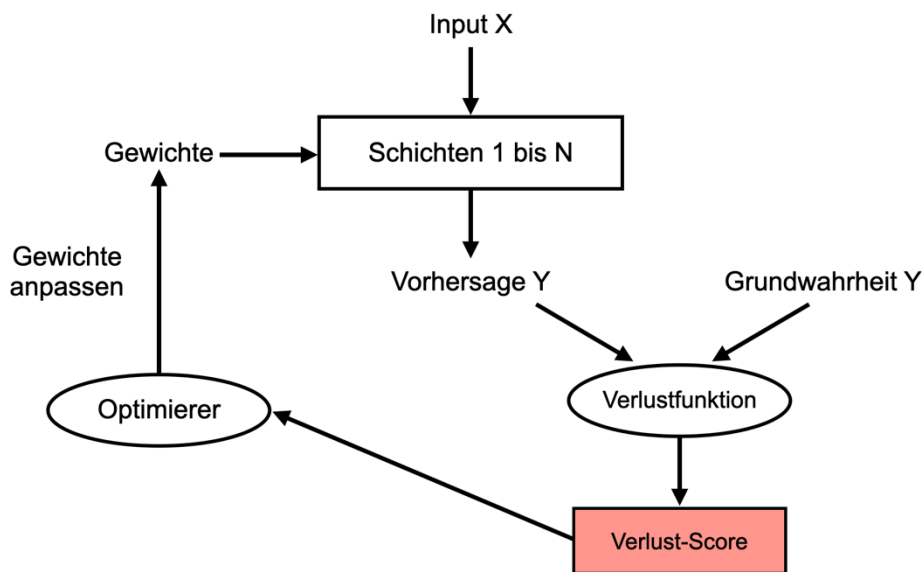


Abbildung 11: Verlustrückführung nach Chollet [vgl. Chol18, S.11]

Der meistverwendete Algorithmus zur Anpassung der Gewichte in einem neuronalen Netzwerk heißt *Gradient Descent*. Der Gradient-Descent-Algorithmus hat zum Ziel, ein lokales Minimum einer ableitbaren Funktion – in diesem Fall der Verlustfunktion – zu finden, um den Verlust-Score des gesamten Netzwerkes zu minimieren. Abbildung 12 visualisiert das Vorgehen beim Gradient Descent. Die Abbildung zeigt den Verlust des gesamten neuronalen Netzes in Abhängigkeit von der Ausprägung eines einzelnen Gewichts im Netz auf. Beim Gradient Descent wird die Fehlerkurve an der Stelle der aktuellen Gewichtsausprägung abgeleitet. Durch diese Ableitung kann ein sogenannter *Gradient* errechnet werden, der die Steigung der Kurve in diesem Punkt wiedergibt. [vgl. Glas21, S.355 f.]

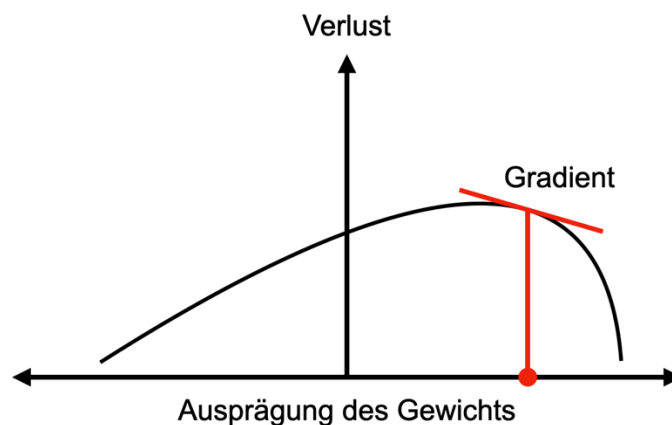


Abbildung 12: Gradient Descent nach Glassner [vgl. Glas21, S.355]

Der Gradient gibt Aufschluss darüber, ob das vorliegende Gewicht erhöht oder verringert werden sollte. Um den Verlust-Score des gesamten Netzes zu minimieren, muss das Gewicht verringert werden, wenn der Gradient positiv ist. Ist der Gradient negativ, muss das Gewicht erhöht werden. Beim Gradient Descent wird dieser Vorgang für jedes Gewicht im Netz einzeln und parallel durchgeführt. [vgl. Glas21, S.355 f.]

$$x(t) = x(t - 1) - \alpha * \nabla f[x(t - 1)]$$

Formel 1: Gradient-Descent nach Graetz [vgl. Grae20]

Formel 1 zeigt auf, wie der Gradient verwendet wird, um ein Gewicht anzupassen. Zunächst wird der negative Gradient (zeigt in die Richtung des stärksten Gefälles) im aktuellen Punkt auf der Verlustfunktion mit einer sogenannten *Lernrate* α multipliziert [vgl. Grae20]. Die Lernrate – auch *Schrittgröße* genannt – bestimmt damit, wie stark ein gegebenes Gewicht in jedem Schritt des Gradient-Descent-Algorithmus angepasst werden soll [vgl. Glas21, S.377] [vgl. IRYM22, S.52819]. Der so errechnete Wert wird anschließend herangezogen, um das neue Gewicht zu ermitteln, dass die Verlustfunktion weiter minimiert [vgl. Grae20].

Ein Optimierungsalgorithmus hat zum Ziel, die Effizienz des Gradient-Descent-Algorithmus zu erhöhen und gleichzeitig zu verhindern, dass der Algorithmus in einem lokalen Minimum stecken bleibt [vgl. Glas21, S.387]. Um dies zu erreichen, implementiert der Optimierer verschiedene Techniken, um die Anpassung der Lernrate des Netzwerks dynamisch zu steuern [vgl. Glas21, S.387]. Eine Technik besteht darin, mit einer hohen Lernrate zu beginnen und diese mit der Zeit immer weiter zu reduzieren [vgl. Glas21, S.396]. Im Vergleich zur Verwendung einer konstanten Lernrate hat dieses Vorgehen den Vorteil, dass der Gradient-Descent-Algorithmus in die Lage versetzt wird, zu Beginn schnelle Ergebnisse zu liefern und am Ende im Bestfall in einem globalen Minimum zu konvergieren [vgl. Glas21, S.396]. Die Verwendung einer gleichbleibenden Lernrate führt dazu, dass der Algorithmus nicht konvergieren kann, da er um das lokale Minimum hin- und herspringt [vgl. Glas21, S.394 f.]. Eine zu hohe Lernrate kann dazu führen, dass schmale Täler in der Verlustfunktion übersehen werden [vgl. Glas21, S.389]. Eine zu geringe Lernrate erhöht die Zeit, die der Algorithmus für das Finden eines Minimums benötigt; außerdem steigt dadurch die Wahrscheinlichkeit, dass er in einem lokalen Optimum stecken bleibt [vgl. Glas21, S.389].

Eine weitere Optimierungstechnik für die Beschleunigung des Trainings ist das sogenannte *Momentum*. Durch ein Momentum werden nicht nur die aktuelle Steigung des Gradienten, sondern auch die Bewegungen aus den vorherigen Schritten miteinbezogen [vgl. Chol18, S.51]. Durch die Implementierung dieses Trägheitsprinzips kann erreicht werden, dass der Algorithmus seltener steckenbleibt, wenn die Steigung in der Verlustkurve einen Wert von Null erreicht (flache Ebene) [vgl. Glas21, S.409].

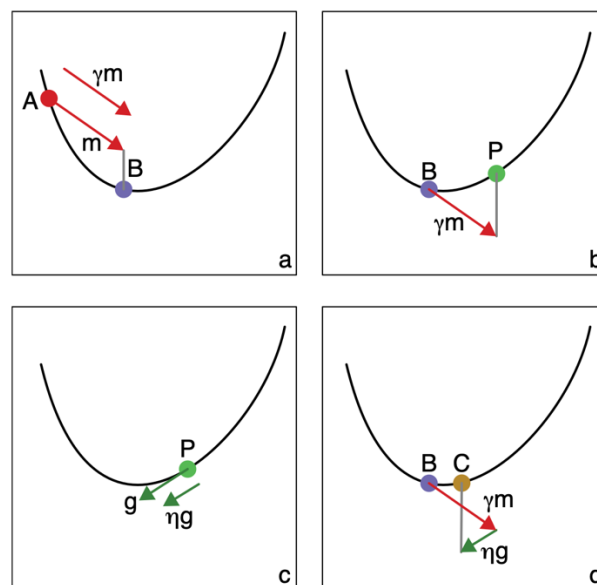


Abbildung 13: Funktionsweise des Nesterov-Momentums nach Glassner [vgl. Glas21, S.414]

Eine verbesserte Variante des Momentums ist das sogenannte Nesterov-Momentum. Bei diesem Algorithmus fließt neben dem Gradienten im aktuellen Punkt auch der Gradient in die Berechnungen ein, der im nächsten Schritt schätzungsweise vorliegen wird [vgl. Glas21, S.414]. Dieses Vorgehen soll dabei helfen, die Konvergenzgeschwindigkeit des Gradient-Descent-Algorithmus zu beschleunigen [vgl. Glas21, S.416].

Abbildung 13 visualisiert dieses Prinzip. Nach der Anwendung des Gradienten γm wird der Punkt B auf der Verlustfunktion erreicht. Das γ steht für die Lernrate, die mit dem Gradienten verrechnet wird. Anstatt direkt den nächsten Schritt auf Punkt P zu machen (siehe Schritt zwei), wird nun errechnet, wie der Gradient im übernächsten Schritt aussehen würde (siehe Schritt drei). In diesem Fall erkennt der Algorithmus, dass sich das Vorzeichen des Gradienten ändern wird. Um sich nicht zu weit vom lokalen Minimum zu entfernen, werden die Gradienten aus Schritt zwei und drei miteinander verrechnet und

direkt ein kleinerer Gradient verwendet, als dies ohne die Anwendung des Nesterov-Momentums der Fall gewesen wäre (siehe Schritt 4). [vgl. Glas21, S.416]

Es existieren verschiedene Gradient-Descent-Varianten. Beim *Batch-Gradient-Descent* erfolgt die Aktualisierung der Gewichte nach jeder Epoche. Der Algorithmus speichert alle Verlust-Scores ab und führt auf der Basis dieser Informationen am Ende eines jeden Trainingsdurchlaufs eine Aktualisierung der Gewichte im Netz durch. Dieses Vorgehen hat den Nachteil, dass alle Proben aus dem Trainingsset im Speicher vorgehalten werden müssen. Reicht der Speicher nicht aus, kann dies die Trainingszeit stark erhöhen. Außerdem benötigt der Algorithmus verhältnismäßig viele Epochen, um zu konvergieren. Vorteilhaft ist, dass der Algorithmus eine ebenmäßige Verlust-Kurve erzeugt. [vgl. Glas21, S.401 ff.]

Ein zweiter Algorithmus-Typ ist der sogenannte *Stochastic Gradient Descent*. Dieser Algorithmus verarbeitet die Label-Wert-Paare im Trainingsset in einer zufälligen Reihenfolge; die Gewichte des Netzes werden hier nicht nach jeder Epoche, sondern nach jedem Bild-Label-Paar aktualisiert. Dieses Verfahren hat den Vorteil, dass der Algorithmus schneller Ergebnisse liefert und eine detaillierte Verlust-Kurve produziert. Auf der anderen Seite ist die Verlust-Kurve nicht so ebenmäßig wie beim Batch-Gradient-Descent. Positiv ist auch, dass die Proben nicht im Speicher vorgehalten werden müssen. Da die Proben in zufälliger Reihenfolge verarbeitet werden, kann der Verlust-Score zwischen den einzelnen Proben und Epochen stark schwanken. Dieses Verhalten hat den Vorteil, dass der Algorithmus nicht so häufig in lokalen Optima der Verlustkurve hängenbleibt. Nachteil ist jedoch, dass es dadurch schwierig wird zu beurteilen, ab wann ein Modell zu stark auf die vorliegenden Daten angepasst wird. [vgl. Glas21, S.403 ff.]

In der Praxis werden unterschiedliche Optimierungsalgorithmen verwendet, um den Stochastic-Gradient-Descent-Algorithmus weiter zu verbessern. Ein häufig verwendeter Algorithmus ist *Adaptive Momentum Estimation* (Adam). Adam errechnet eine eigene Lernrate für jedes Gewicht im Netz. Dabei werden für jedes Gewicht zwei Variablen vorgehalten, die den gleitenden Durchschnitt der Gradienten aus den letzten Aktualisierungsschritten beinhalten. In der ersten Variable werden die Gradienten jeweils quadriert und aufsummiert. Neuere Gradienten werden dabei stärker gewichtet als ältere. In der zweiten Variable werden die gleichen Informationen vorgehalten. Der einzige Unterschied ist, dass die vergangenen Gradienten hier nicht quadriert werden. Damit gehen die Informationen über die Steigungsrichtungen der Gradienten aus den vorangegangenen Schritten nicht verloren. Die beiden Variablen werden dafür verwendet, die Lernrate dynamisch zu steuern. Sind die vergangenen Gradienten konstant (z.B. beim Abstieg oder Aufstieg in der Verlustfunktion), wird die Lernrate erhöht und es können größere

Schritte gewagt werden; bei sich häufig ändernden Gradienten wird die Lernrate verringert, damit das neuronale Netz effizient konvergiert. Im Vergleich zu anderen SGD-Optimierungsalgorithmen benötigt Adam aufgrund der dynamischen Lernrate signifikant weniger Epochen, um die Verlustfunktion zu minimieren. [vgl. Glas21, S.420 f.] [vgl. Grae20]

Es wurde bereits eine optimierte Version des Adam-Optimierers entwickelt: Adam mit *Weight Decay (AdamW)*. Der Weight Decay ist eine Regularisierungstechnik, die dazu beiträgt, die Komplexität eines neuronalen Netzes zu reduzieren [vgl. Chol18, S.107]. Neuronale Netzwerke mit kleineren Gewichten neigen weniger stark zu einer Überanpassung an die unterliegenden Daten und sind in der Lage, besser zu generalisieren [vgl. Grae20] [vgl. Chol18, S.107]. Um das Netzwerk zur Verwendung von kleineren Gewichten zu animieren, wird die Verlustfunktion um einen Kostenfaktor erweitert [vgl. Chol18, S.107]. Die Höhe des Kostenfaktors richtet sich nach der Größe der Gewichte; die Verwendung von großen Gewichten wird härter bestraft als die Verwendung von kleinen Gewichten [vgl. Chol18, S.107]. Zwar implementiert auch der ursprüngliche Adam-Optimierer ein Weight Decay, dieser ist dort aber weniger effektiv [vgl. Grae20]. Das verbesserte Weight-Decay-Konzept vom AdamW-Optimierer trägt dazu bei, dass Modelle eine ähnlich hohe Generalisierbarkeit erreichen können, wie bei der Verwendung vom SGD-Algorithmus [vgl. Grae20]. Außerdem soll AdamW dabei helfen, die Trainingsdauer von Modellen zu beschleunigen [vgl. Grae20].

2.2.4.3 Hyperparameteroptimierung

Im Machine- und Deep Learning wird zwischen Parametern und Hyperparametern unterschieden. Parameter sind zum Beispiel die Gewichte, die während des Trainingsprozesses automatisch vom Modell angepasst werden, wenn es die Muster aus den ihm zur Verfügung gestellten Daten erlernt [vgl. WeMV20, S.1]. Hyperparameter werden nicht vom Modell selbst angepasst, sondern müssen im Rahmen einer Hyperparameteroptimierung manuell festgelegt werden, um den Lernprozess des Modells zu kontrollieren [vgl. WeMV20, S.1]. Die Hyperparameteroptimierung hat einen großen Einfluss auf die endgültige Performance der Modelle und die Geschwindigkeit des Trainings [vgl. WeMV20, S.1] [vgl. IRYM22, S.52819 f.]. Beispiele für Hyperparameter sind die eingestellte Anzahl an Epochen, die Auflösung der Eingabebilder oder die Lernrate. Wird keine Hyperparameteroptimierung vorgenommen, wird eine Unter- bzw. Überanpassung (auch *Under-* bzw. *Overfitting* genannt) des Modells begünstigt und die Trainingszeit nimmt zu [vgl. IRYM22, S.52820]. Beim Overfitting verliert ein Modell die Fähigkeit, sein

erlerntes Wissen auf bisher ungesehene Daten zu übertragen, da es zu stark auf die Bilder im zum Training verwendeten Datensatz optimiert wurde [vgl. Glas21, S.196 f.].

2.2.4.3.1 Grid Search

Die klassische Hyperparameteroptimierung verläuft in einer Schleife: Zuerst wird ein neues Modell mit zufällig eingestellten Parametern initialisiert [vgl. Glas21, S.188]. Danach wird eine sogenannte *Such-Heuristik* verwendet, die eine Hyperparameter-Kombination auswählt [vgl. WeMV20, S.1]. Anschließend erfolgen das Training und die Validierung des Modells. Nach jeder Evaluation wird das Ergebnis zwischengespeichert; dieser Vorgang wird so lange wiederholt, bis alle Hyperparameter-Kandidaten ausprobiert wurden oder eine andere Abbruchbedingung eintritt [vgl. Glas21, S.188]. Danach lässt sich die Performance der verschiedenen Hyperparameter-Kombination vergleichen und der beste Kandidat auswählen [vgl. Glas21, S.188].

Bei der *Grid Search* wird jeder mögliche Wert für jeden Hyperparameter in einem vordefinierten Suchraum überprüft, um die beste Hyperparameter-Kombination zu ermitteln [vgl. BeBe12, S.282] [vgl. LiLi19, S.6]. Abbildung 14 visualisiert die Funktionsweise der Grid Search. An der x- und y-Achse sind zwei Hyperparameter aufgetragen; die Matrix in der Mitte zeigt die Kombinationsmöglichkeiten der beiden Hyperparameter im Suchraum. Wie zu erkennen ist, werden bei der Grid Search alle Hyperparameter-Kombinationen in einem vordefinierten Bereich untersucht. In diesem Beispiel sind die Ausprägungsmöglichkeiten der beiden Hyperparameter jeweils 0, 0,2, 0,4, 0,6, 0,8 und 1.

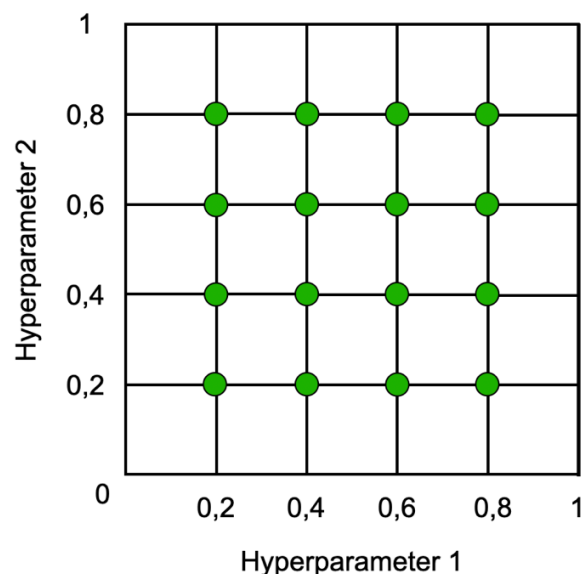


Abbildung 14: Grid Search nach Liashchynskyi [vgl. LiLi19, S.6]

Dieses Vorgehen bringt den Nachteil mit sich, dass die Menge an zu untersuchenden Hyperparameter-Kombinationen mit einer steigenden Anzahl an Hyperparametern und zulässigen Ausprägungsmöglichkeiten exponentiell zunimmt. Damit steigt auch der benötigte Zeit- und Rechenaufwand stark an. Eine weitere Schwachstelle der Grid Search ist, dass eine suboptimale Auswahl des Suchraums dazu führen kann, dass viele unnötige Hyperparameter-Kombinationen ausprobiert werden und der Algorithmus das globale Optimum und das damit verbundene Potenzial für die Verbesserung der Genauigkeit des Modells nicht realisieren kann. Vorteilhaft ist, dass die Grid Search einfach zu implementieren, parallelisieren und zu überwachen ist (systematische Methode). Außerdem ist die Performance – insbesondere in niedrig dimensionalen Suchräumen – besser als bei der manuellen Suche nach Hyperparametern. [vgl. BeBe12, S.282 ff.] [vgl. LiLi19, S.9]

2.2.4.3.2 Random Search

Bei der sogenannten *Random Search* wird die Auswahl der Hyperparameter-Kombinationen im vordefinierten Suchraum zufällig getroffen [vgl. BeBe12, S.283]. Wie in Abbildung 15 zu erkennen ist, erfolgt die Suche nach einem globalen Optimum durch die Kombination von Hyperparametern und deren Ausprägungen nicht in Form eines starren Rasters. Hier werden lediglich eine Ober- und eine Untergrenze (0 bzw. 1) für die Ausprägungsmöglichkeiten der beiden Hyperparameter und die Anzahl an Versuchen (12) definiert [vgl. LiLi19, S.6].

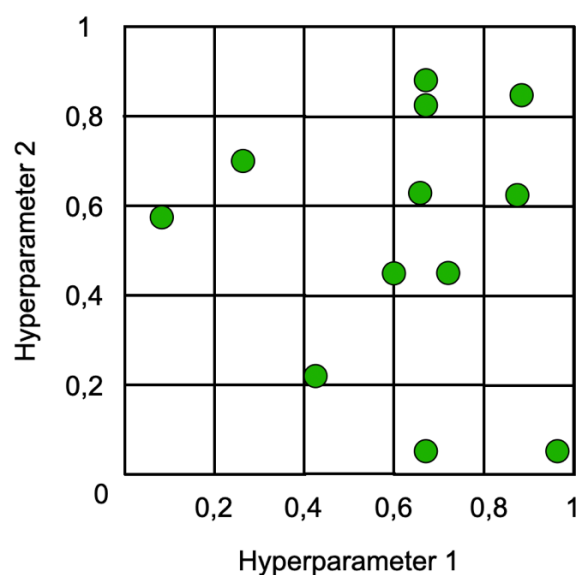


Abbildung 15: Random Search nach Liashchynskyi [vgl. LiLi19, S.6]

Damit ist der Random-Search-Algorithmus besonders für die Optimierung in hochdimensionalen Suchräumen effizienter anwendbar, da hier nicht alle Kombinationsmöglichkeiten ausprobiert werden müssen. Auch die Anzahl an überflüssigen Versuchen ist hier deutlich geringer als bei der Grid Search. Der Nachteil des Verfahrens ist allerdings, dass aufgrund der zufallsbedingten Auswahl der Hyperparameter-Kombinationen nicht garantiert werden kann, dass eine optimale Lösung gefunden wird (unvorhersehbare Ergebnisse). [vgl. BeBe12, S.283 f.] [vgl. LiLi19, S.9]

2.2.4.3.3 Genetische Algorithmen

Ein weiterer Ansatz für die Hyperparameteroptimierung ist die Verwendung sogenannter *genetischer Algorithmen*. Neben der Grid Search und der Random Search gehören die genetischen Algorithmen zu den drei am häufigsten verwendeten Methoden für die Optimierung von Hyperparametern [vgl. LiLi19, S.1]. Genetische Algorithmen gehören zur Klasse der *evolutionären Algorithmen*; sie machen sich die Mechanismen der natürlichen biologischen Evolution zu Nutze [vgl. LiLi19, S.3] [vgl. Kram17, S.4]. In der Natur besitzt jedes Individuum Chromosomen, die das Erbgut enthalten. Bei genetischen Algorithmen wird das „Erbgut“ der Individuen in Form eines Strings repräsentiert [vgl. LiLi19, S.3]. Dieser String besteht aus mehreren Buchstaben, Integern und/oder Float-Werten und beschreibt eine spezifische Hyperparameter-Kombination [vgl. LiLi19, S.3].

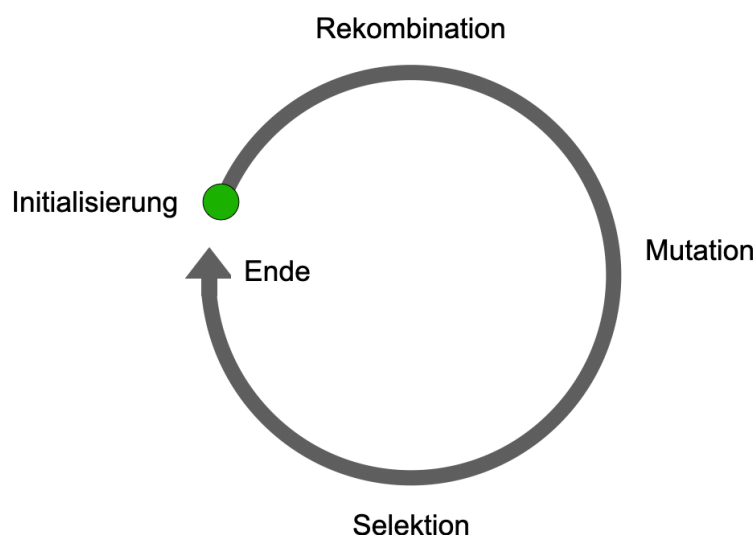


Abbildung 16: Zyklus eines genetischen Algorithmus nach Kramer [vgl. Kram17b, S.5]

Abbildung 16 zeigt die Funktionsweise eines genetischen Algorithmus auf. In einem ersten Schritt wird eine sogenannte *Population* initialisiert, die aus zufällig parametrisierten Individuen besteht [vgl. LiLi19, S.3] [vgl. Kram17, S.5]. Jedes Individuum ist ein Lösungskandidat, der einen speziellen Punkt im mehrdimensionalen Lösungsraum repräsentiert [vgl. LiLi19, S.3]. Anschließend werden zwei oder mehr Individuen ausgewählt, deren genetisches Material kombiniert wird, um einen neuen Lösungskandidaten zu erzeugen [vgl. Kram17, S.5]. Dieser Vorgang wird auch als *Crossover* bezeichnet [vgl. Kram17, S.5]. Danach wird das genetische Material (die Hyperparameter-Kombination) des neu-entstandenen Individuums zufällig mutiert [vgl. Kram17, S.5]. Die so entstehenden Nachkommen werden nach der Mutation auf ihre Fitness hin untersucht. Die Evaluation der Fitness erfolgt anhand einer sogenannten *Fitnessfunktion*, welche die Qualität der Lösungskandidaten misst [vgl. Kram17, S.15]. In einem letzten Schritt werden die Lösungskandidaten mit der besten Fitness ausgewählt und in die nächste Generation übernommen [vgl. Kram17, S.5]. Durch dieses Verfahren wird das Prinzip der *natürlichen Selektion* simuliert; es überleben nur die Individuen, die am besten an ihre Umgebung angepasst sind [vgl. LiLi19, S.3]. Dieser Zyklus wird so lange wiederholt, bis ein bestimmtes Abbruchkriterium eintritt [vgl. LiLi19, S.3] [vgl. Kram17, S.5].

Die Verwendung eines genetischen Algorithmus für die Hyperparameteroptimierung ist vor allem dann sinnvoll, wenn die Anzahl an vorliegenden Hyperparametern besonders hoch ist. In diesem Fall wäre die Grid Search nicht geeignet, da hier zu viele Hyperparameter-Kombinationen geprüft werden müssten. Die Random Search ist zwar schneller und weniger rechenaufwändig als die Grid Search, dafür ist hier aber nicht garantiert, dass eine optimale Lösung gefunden wird. Die Verwendung von genetischen Algorithmen ist im Vergleich zur Grid Search in hochdimensionalen Suchräumen deutlich weniger rechen- und zeitintensiv. Außerdem können bessere Lösungen gefunden werden als bei der Random Search, da der Suchraum nicht im Vorhinein eingeschränkt wird. [vgl. LiLi19, S.9]

2.2.4.4 Performance-Metriken

In diesem Kapitel werden die wichtigsten Performance-Metriken zur Messung der Vorhersagegenauigkeit und Vorhersagegeschwindigkeit für KI-Modelle aus dem Bereich der Bilderkennung näher erläutert.

2.2.4.4.1 Intersection Over Union (IoU), Precision und Recall

In der Objekterkennung erfolgt die Markierung von Objekten über sogenannte *Bounding Boxen*. Bounding Boxen sind Rechtecke, die verwendet werden, um Zielobjekte auf Bildern zu markieren [vgl. Joka22, S.4]. Um die Vorhersagequalität eines Modells zu bestimmen, wird untersucht, wie gut die vom Modell vorgeschlagenen Bounding Boxen mit den Bounding Boxen der Grundwahrheiten übereinstimmen. Die dafür verwendete Formel wird auch als *Intersect Over Union (IoU)* bezeichnet. Die IoU-Formel (siehe Abbildung 17) berechnet den Anteil der Fläche der Überschneidung (Überlappung) an der Fläche der Vereinigung (Gesamtfläche inklusive Überlappung) der vom Modell vorgeschlagenen Box und der Bounding Box der Grundwahrheit [vgl. PaLA20, S.2].

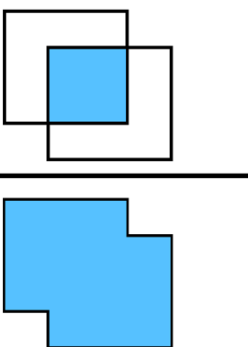
$$\text{IoU} = \frac{\text{Fläche der Überschneidung}}{\text{Fläche der Vereinigung}} = \frac{\text{Diagramm 1}}{\text{Diagramm 2}}$$


Abbildung 17: Intersect Over Union (IoU) nach Padilla et al. [vgl. PaLA20, S.2]

Der IoU-Wert wird anschließend dafür genutzt, um die vom Modell vorgeschlagene Bounding Box als korrekt (*False Positive*) oder inkorrekt (*False Negative*) zu klassifizieren. Dafür wird ein Schwellenwert t definiert. Nur, wenn der IoU-Wert mindestens so groß ist wie t , gilt die vorgeschlagene Bounding Box als korrekt; andernfalls als inkorrekt. [vgl. PaLA20, S.2]

In der Objekterkennung gibt es zwei wesentliche Metriken, die für die Güte eines Modells verwendet werden können; die *Precision* und den *Recall*. Die Precision (zu deutsch: Genauigkeit) errechnet den Anteil der korrekt erkannten Bounding Boxen ($\text{IoU} \geq t$) an der Menge aller Vorhersagen des Modells (siehe Formel 2). False Positives sind Objekte, die vom Modell nicht korrekt erkannt wurden ($\text{IoU} < t$). [vgl. PaLA20, S.2]

$$Precision = \frac{TP}{TP + FP} = \frac{\text{Alle vom Modelle korrekt erkannten Bounding Boxen}}{\text{Alle vom Modelle getätigten Vorhersagen}}$$

Formel 2: Precision nach Padilla et al. [vgl. PaLA20, S.2]

Der Recall (zu deutsch: Sensitivität) gibt an, wie viele der Bounding Boxen der Grundwahrheit das Modell korrekt erkannt hat (siehe Formel 3) [vgl. PaLA20, S. 2]. False Negatives sind Objekte, die eigentlich existieren (es gibt Grundwahrheiten), die aber vom Modell fälschlicherweise nicht als diese Objekte erkannt wurden [vgl. OIDe08, S.138].

$$Recall = \frac{TP}{TP + FN} = \frac{\text{Alle vom Modelle korrekt erkannten Bounding Boxen}}{\text{Alle Bounding Boxen der Grundwahrheit}}$$

Formel 3: Recall nach Padilla et al. [vgl. PaLA20, S. 2]

2.2.4.4.2 Mean Average Precision (mAP)

Sowohl bei der Precision als auch beim Recall wird somit ein möglichst hoher Wert angestrebt, um die Anzahl an False Positives und False Negatives zu minimieren. In der Praxis gibt es jedoch kein Modell mit einem Precision- und Recall-Wert von 100 Prozent. Die beiden Metriken stehen in einem Zielkonflikt zueinander; mit einem steigenden Recall sinkt die Precision und umgekehrt. Dieser Kompromiss lässt sich in einer sogenannten *Precision-Recall-Kurve* visualisieren (siehe Abbildung 18). [vgl. Wang22, S.2]

Die Precision-Recall-Kurve kann herangezogen werden, um die sogenannte *Average Precision* (AP) zu berechnen [vgl. Wang22, S.2]. Die AP für eine Klasse und einen bestimmten IoU-Schwellenwert entspricht der Fläche unter der Precision-Recall-Kurve [vgl. Wang22, S.2]. Mit der Precision-Recall-Kurve lässt sich feststellen, wie gut ein Modell darin ist, Objekte korrekt zu lokalisieren [vgl. IRYM22, S.52823]. Ein Modell ist als robust zu bewerten, wenn auch bei einem steigenden Recall eine hohe Precision aufrecht erhalten werden kann [vgl. IRYM22, S.52823].

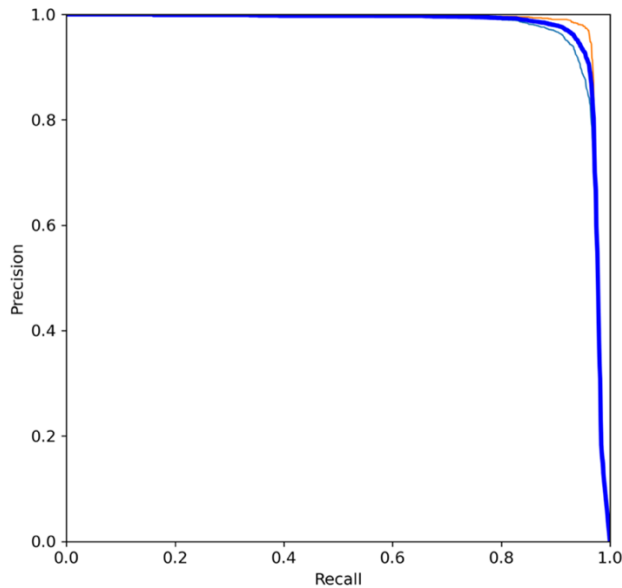


Abbildung 18: Precision-Recall-Kurve

Die AP-Werte lassen sich schließlich dazu verwenden, die sogenannte *Mean Average Precision* (mAP) zu berechnen. Die Mean Average Precision ist die am häufigsten verwendete Kennzahl für die Vorhersagegenauigkeit bei Bilderkennungsmodellen und berechnet den Durchschnitt der *Average-Precision-Werte* über alle Klassen für einen oder mehrere IoU-Schwellenwerte [vgl. PaLA20, S.3] [vgl. ZCSG23, S.5] (siehe Formel 4). Die am weitesten verbreitete mAP-Variante ist $mAP_{0.5}$, welche den Durchschnitt aller Klassen-AP-Werte für $t = 0,5$ ermittelt [vgl. ZCSG23, S.6]. Es besteht auch die Möglichkeit, die mAP für ein Intervall anzugeben. Die Metrik $mAP_{0.5:0.95}$ mittelt die Average-Precision-Werte über mehrere IoU-Schwellenwerte von $t = 0,5$ bis $0,95$; dabei wird der IoU-Schwellenwert schrittweise um fünf Prozentpunkte erhöht ($t = 0,5; 0,55; 0,60$ etc.) [vgl. ZCSG23, S.6].

$$mAP@t = \frac{1}{k} \sum_0^{k-1} AP@t$$

Formel 4: Mean Average Precision nach Wang [vgl. Wang22, S.2]

2.2.4.4.3 Vorhersagegeschwindigkeit

Neben der Vorhersagegenauigkeit spielt auch die Vorhersagegeschwindigkeit eine wichtige Rolle für die Performancemessung. Die *Inferenzzeit* gibt an, wie viele Millisekunden das Modell benötigt, um Vorhersagen zur Erkennung der Objekte auf einem bisher un-gesehenen Bild zu tätigen. Es ist auch möglich die Anzahl an verarbeiteten Bildern pro Sekunde anzugeben. Die Vorhersagegeschwindigkeit hängt stark vom verwendeten Feature-Extraktor, der Größe der verwendeten Bilder, der eingesetzten Hardware (CPU und GPU) und der verwendeten Modellarchitektur ab. Generell lässt sich sagen, dass die Vorhersagegenauigkeit mit steigender Bildauflösung und Komplexität der aus-gewählten Feature-Extraktor-Architektur zwar zunimmt. In diesem Fall steigt allerdings auch parallel der Rechen- und damit Zeitbedarf für die Verarbeitung von un-gesehenen Bildern (*Speed/Accuracy Trade-Off*). [vgl. HRSZ16]

$$FPS = \frac{\text{Anzahl an Bildern}}{\text{Gesamte Verarbeitungszeit in Sekunden}}$$

Formel 5: Frames pro Sekunde Second nach Sazanita Isa et al. [vgl. IRYM22, S.52821]

Eine andere Möglichkeit die Vorhersagegeschwindigkeit eines Modells zu bewerten ist die Metrik *Frames pro Sekunde* (FPS) (siehe Formel 5). Sie gibt an, wie viele Bilder das Modell pro Sekunde verarbeiten kann [vgl. IRYM22, S.52821].

2.3 Computer Vision

Computer Vision (CV) ist ein interdisziplinäres Feld, dass als Teildisziplin der Künstlichen Intelligenz verstanden werden kann [vgl. Xin22, S.125] [vgl. Gao22, S.80]. Im For-schungsfeld Computer Vision wird das Ziel verfolgt, nutzbare Informationen aus visuel-len Daten wie Bildern oder Videos zu extrahieren [vgl. Xin22, S.125]. Die dabei ange-wendeten Techniken imitieren das menschliche Auge und sollen Maschinen (Computer) in die Lage versetzen, ihre Umgebung wahrzunehmen, Objekte zu identifizieren und da-raufhin Entscheidungen zu treffen [vgl. Xin22, S.125] [vgl. IMAR22, S.33]. Das For-schungsfeld Computer Vision macht sich viele unterschiedliche Techniken und Metho-den zunutze. Auch der Stand der Technik der dort verwendeten Methoden hat sich über die vergangenen Jahre immer wieder verändert.

Abbildung 19 zeigt die vier großen Anwendungsbereiche in der Computer Vision (von links oben nach rechts unten): Die *Classification*, die *Semantic Segmentation*, die *Object Detection* und die *Instance Segmentation*. Bei der *Classification* wird das gesamte Bild untersucht und einer vordefinierten Klasse zugeordnet. Hier ist keine individuelle Erkennung von Objekten möglich. Bei der *Semantic Segmentation* wird jeder Pixel auf einem Bild einer Klasse zugeordnet. Hier ist es ebenfalls nicht möglich, einzelne Instanzen einer Klasse zu detektieren. [vgl. SSLB22, S.2]



Abbildung 19: Anwendungen der Computer Vision nach Sharma et al. [vgl. SSLB22, S.2]

Die *Object Detection* hat zum Ziel, sowohl unterschiedliche Objekte zu klassifizieren, als auch deren Position auf dem Bild zu lokalisieren [vgl. SSLB22, S.2] [vgl. ZCSG23, S.1]. Die Erkennung erfolgt hier nicht auf der Pixel-Ebene, sondern mithilfe von Bounding Boxes. Zu guter Letzt gibt es den Anwendungsbereich der *Instance Segmentation*. Die *Instance Segmentation* ähnelt der *Object Detection* sehr stark, da hier ebenfalls einzelne Objektinstanzen einer Klasse zugeordnet und deren Position auf dem Bild vermerkt werden. [vgl. SSLB22, S.2]. Der einzige Unterschied ist, dass die Objekte pixelgenau annotiert werden; anstatt von Rechtecken, die in der Regel einen Teil des Hintergrundes mit annotieren, werden hier Polygone als Label verwendet [vgl. SSLB22, S.2].

2.3.1 Einstufige Detektoren

Seit dem Aufkommen der Deep-Learning-basierten Object Detection werden heute in den modernen Modellarchitekturen vor allem CNNs verwendet [vgl. ISZL21, S.2] [vgl. Xin22, S.125] [vgl. ZCSG23, S.2 f.]. Dabei lassen sich zwei Arten von Ansätzen unterscheiden: *Einstufige* und *zweistufige Detektoren* [vgl. ISZL21, S.2] [vgl. LLLY20, S.1]. Einstufige Detektoren arbeiten die Bilder in einem einzigen Schritt ab, um die Objekte auf einem Bild zu erkennen [vgl. ISZL21, S.2] [vgl. ZCSG23, S.3]. Damit sind sie in der Lage, Objekte deutlich schneller zu klassifizieren (Echtzeitfähigkeit) [vgl. LLLY20, S.2] [vgl. ZCSG23, S.4]. Nachteilig ist, dass es einstufigen Detektoren schwerer fällt, kleinere Objekte zu identifizieren und, dass diese, im Vergleich zu den zweistufigen Detektoren, in der Regel eine generell niedrigere Vorhersagegenauigkeit aufweisen [vgl. ZCSG23, S.4].

2.3.1.1 You Only Lock Once (YOLO)

Das sogenannte *YOLO-Modell* (You Only Look Once) ist ein einstufiges Object-Detection-Modell auf der Basis eines CNNs [vgl. RDGF15, S.779]. Das Modell ist in der Lage, das gesamte Bild und alle globalen Bildinformationen zu verarbeiten und in einem Schritt alle Objekte auf einem Bild zu lokalisieren bzw. zu klassifizieren [vgl. RDGF15, S.779] [vgl. IRYM22, S.52819]. Durch diese Eigenschaft weist YOLO im Gegensatz zu Modellen mit *Region Proposal Network* (RPN) oder *Sliding Windows* eine deutlich geringere Komplexität auf, ist schneller und kann einfacher optimiert werden [vgl. RDGF15, S.779 f.]. Außerdem wird dadurch die Fähigkeit zur Generalisierung des Modells verbessert [vgl. RDGF15, S.780].

Abbildung 20 zeigt die allgemeine Funktionsweise des YOLO-Modells auf. In einem ersten Schritt wird das Eingabebild in ein Raster mit der Abmessung von S mal S Pixeln unterteilt. Jede Gitterzelle ist dafür verantwortlich, B -viele Bounding Boxen (auch *Anchor-Boxen* genannt) vorzuschlagen und diesen Kandidaten jeweils eine sogenannte *Konfidenz* zuzuordnen. Diese gibt an, wie sicher das Modell ist, dass die betroffene Bounding Box ein Zielobjekt enthält und wie gut der vorgeschlagene Bounding-Box-Kandidat mit der Bounding Box der Grundwahrheit übereinstimmt (IoU). Aufgrund der begrenzten Anzahl an Bounding-Box-Kandidaten pro Gitterzelle hat YOLO im Allgemeinen Probleme damit, insbesondere kleine und nahaneinander liegende Objekte zuverlässig zu lokalisieren. Die vorgeschlagenen Bounding-Box-Koordinaten setzen sich aus vier Werten zusammen: x , y , w , und h . Die Koordinaten (x, y) geben das Zentrum der Bounding Box in Relation zu den Rändern der Gitterzelle an. Die Koordinaten w und h

entsprechen der Breite und Höhe der vorgeschlagenen Bounding Boxen in Relation zum gesamten Bild. [vgl. RDGF15, S.780] [vgl. ZCSG23, S.4]

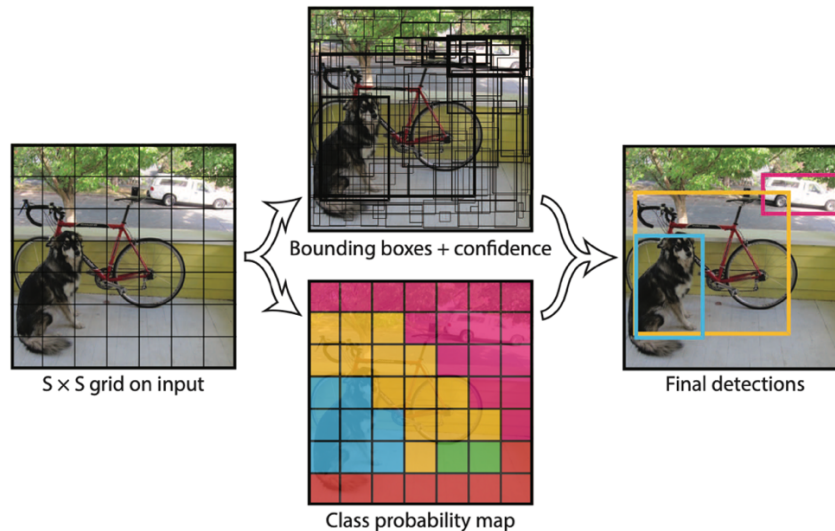


Abbildung 20: Funktionsweise von YOLO nach Redmon et al. [vgl. RDGF15, S.780]

Neben den Bounding-Boxen-Koordinaten und Konfidenzen werden für jede Gitterzelle, die ein Objekt enthält, C-viele Klassenzugehörigkeitswahrscheinlichkeiten ausgegeben – für jede Zielklasse eine. Die Anzahl an Klassenzugehörigkeitswahrscheinlichkeiten ist unabhängig davon, wie viele Bounding Boxen für eine Gitterzelle vorgeschlagen wurden. Die Koordinaten der Anchor-Boxen, die Konfidenz und Klassenzugehörigkeitswahrscheinlichkeiten werden parallel für alle Gitterboxen generiert ($S \times S \times (4 \text{ Koordinaten} + 1 \text{ Konfidenz} + C)$). Um die Anzahl an vorhergesagten Bounding Boxen zu reduzieren, werden alle Anchor-Box-Kandidaten entfernt, bei denen die Konfidenz unterhalb eines vordefinierten Schwellenwertes liegt. Außerdem sind die Gitterzellen nur dann für die Vorhersage von Objekten verantwortlich, wenn der Mittelpunkt eines Zielobjektes in der entsprechenden Gitterzelle liegt. [vgl. RDGF15, S.780]

Für seine Vorhersagen nutzt das YOLO-Modell ein durchgehendes CNN (siehe Abbildung 21). Das Netz besteht aus insgesamt 24 Konvolutionsschichten und zwei vollständig vernetzte Schichten, welche für die Vorhersage der Bounding-Box-Koordinaten und der Klassenzugehörigkeitswahrscheinlichkeiten zuständig sind, sowie zusätzlichen Pooling-Schichten [vgl. RDGF15, S.780 f.]. Als Aktivierungsfunktionen werden *Sigmoid* und die stückweise lineare Funktion *Leaky Rectified Linear Unit* (ReLU) verwendet [vgl. Rajp20].

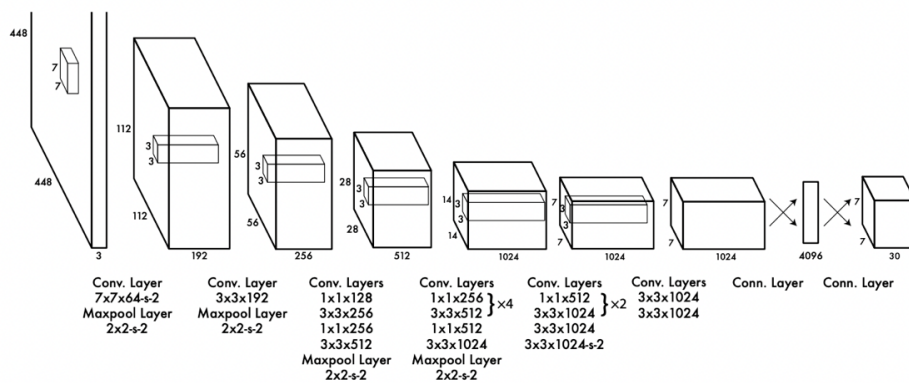


Abbildung 21: YOLO-Architektur nach Redmon et al. [vgl. RDGF15, S. 781]

2.3.2 Zweistufige Detektoren

Bei zweistufigen Detektoren läuft die Objekterkennung in zwei separaten Schritten ab. In einem ersten Schritt werden zunächst Regionen aus dem Bild herausgefiltert, die mit einer größeren Wahrscheinlichkeit ein gesuchtes Objekt abbilden [vgl. LLLY20, S.1 f.]. Die Generierung dieser sogenannten *Proposals* erfolgt durch ein Region Proposal Network [vgl. LLLY20, S.2]. Anschließend werden die auf den Proposals erkennbaren Objekte von einem separaten CNN – in der Regel einem sogenannten *Region Convolutional Neuronal Network* (R-CNN) – klassifiziert [vgl. LLLY20, S.2]. Da zweistufige Modelle nur wenige ausgewählte Proposals verarbeiten, können sie detailliertere Merkmale aus Bildern extrahieren als einstufige Detektoren [vgl. LLLY20, S.2]. Durch die beiden Stufen ist die Klassifizierung von Objekten bei diesen Modellen langsamer und komplexer, dafür in der Regel aber auch präziser [vgl. LLLY20, S.2] [vgl. ZCSG23, S.3 f.].

3 Stand der Wissenschaft und Technik

In diesem Kapitel wird eine Auswahl an verwandten Arbeiten aus dem Themenbereich der KI-basierten Bilderkennung vorgestellt. Durch die Untersuchung dieser Arbeiten soll nachfolgend die bestehende Forschungslücke im Bereich der Erkennung von statischen Hafengebieten herausgearbeitet werden.

3.1 Automobilindustrie

Neben der maritimen Domäne finden Algorithmen zur automatisierten Bilderkennung auch in der Automobilbranche Anwendung. In der Arbeit von [KOVA21] untersuchen die Autoren die Eignung von unterschiedlichen Sensoren wie RGB-Kameras, Radar- und Lidar-Sensoren sowie verschiedene Deep-Learning-Modellen wie R-CNNs, dem „Single Shot Detector“ (SSD) und „YOLO“, um eine automatisierte Objekterkennung im Rahmen des Autonomen Fahrens zu realisieren. Die Arbeit kommt zum Schluss, dass eine optimale Objekterkennung nur durch die Kombination der unterschiedlichen Sensoren, eine diverse und domänenspezifische Datenbasis für das Training der KI-Modelle und die Nutzung von echtzeitfähigen Deep-Learning-Modellen wie YOLO gewährleistet werden kann. Die Autoren sind der Meinung, dass nur durch die Ergänzung der RGB-Kameras durch aktive Sensoren (Radar und Lidar) die Entfernungen von Objekten im dreidimensionalen Raum zuverlässig erfasst werden können. [vgl. KOVA21]

Die Autoren in [KPPJ21] haben sich ebenfalls mit der automatisierten Objekterkennung im Rahmen des Autonomen Fahrens beschäftigt. Dabei liegt der Fokus auf der Evaluation von verschiedenen ML- und DL-Modellen aus dem Bereich der Computer Vision. Die Evaluation erfolgte mithilfe einer Metanalyse von relevanten wissenschaftlichen Artikeln. Im Paper wird sowohl die Eignung von zweistufigen Detektoren wie „Fast R-CNN“ und „Faster R-CNN“ als auch einstufigen Detektoren wie „YOLO“ und „SSD“ beleuchtet. Neben einfachen CNNs werden darüber hinaus auch R-CNNs in die Analyse mit einbezogen. Außerdem werden aktive Sensoren (einfache Kameras) und passive Sensoren (Radar und Lidar) gegenübergestellt. Die Arbeit kommt zu dem Schluss, dass sich ML- und DL-Modelle durchaus für die Realisierung des Autonomen Fahrens eignen würden. Es sei dafür aber notwendig, zunächst die Präzision der Modelle weiter zu verbessern, um keine Menschenleben zu gefährden. Ein großes Problem sei aktuell, dass es aktuell noch zu wenige qualitativ hochwertige und ausreichend große Trainingsdatensätze für das Training derartiger Modelle gäbe. Mithilfe moderner Technologien werde es jedoch immer einfacher werden, diese Daten zu sammeln und damit die Vorhersagegenauigkeit der Modelle zu verbessern. [vgl. KPPJ21]

[SiAr19] untersucht die Anwendungsmöglichkeiten von DL-Algorithmen in der Automobilbranche. Der mit Abstand wichtigste Anwendungsfall zur Anwendung von Deep-Learning-Modellen sei die Unterstützung des Autonomen Fahrens. Das DL wird dafür eingesetzt, um in Kombination von Kamera-, Lidar- und Radardaten Objekte wie Ampeln oder Straßenschilder und Spurmarkierungen automatisiert zu erkennen. Es kann auch dafür genutzt werden, das Aufmerksamkeitslevel des Fahrers zu überwachen. Aufgrund ihrer hohen Effizienz werden in diesen Bereichen vor allem CNNs verwendet. Ein weiterer Anwendungsfall von DL ist die Messung der Fahrzeugdynamik (z.B. Grip-Level zwischen den Rädern und der Straße oder Schwimmwinkel). Außerdem lassen sich DL-Algorithmen für das Gesundheitsmonitoring bei Fahrzeugen einsetzen. Die Modelle können genutzt werden, um Verschleiß am Fahrzeug vorherzusagen und den Fahrer rechtzeitig zu warnen. Außerdem lassen sich Kameras einsetzen, um bestehende Schäden an Fahrzeugen automatisiert zu erkennen und zu bewerten. Als ein vierter Deep-Learning-Anwendungsfall konnten die Autoren die Unterstützung einer datengetriebenen Produktentwicklung identifizieren. Diese könne den Herstellern dabei helfen, die Anzahl an Rückrufen zu reduzieren. [vgl. SiAr19]

3.2 Maritime Domäne

In [StHM08] wird die Eignung von mathematischen Modellen und Soft-Computing-Ansätzen wie neuronale Netze, evolutionäre Algorithmen und Fuzzy-Logik untersucht, um die Navigation und Kollisionsvermeidung mittels Bilderkennung bei autonomen Schiffen zu implementieren. Laut der Autoren seien mathematische Modelle dann besonders hilfreich, wenn die Umgebungseinflüsse konstant bleiben. Mit zunehmend dynamischeren Umgebungen steige – auch im Hinblick auf die Echtzeitfähigkeit – der Nutzen von KI-Ansätzen. Durch die Kombination von neuronalen Netzen, Fuzzy-Logik und mathematischen Algorithmen könne sowohl in statischen als auch dynamischen Umgebungen eine gute Performance bei der Kollisionsvermeidung erreicht werden, auch wenn derartige Systeme schwer zu konzipieren und implementieren seien. Die Autoren sind der Meinung, dass autonome Technologien erst dann in der Schifffahrt Anwendung finden werden, wenn die von den autonomen Technologien verbundenen Unsicherheiten kleiner sind als jene, die von menschlichen Akteuren ausgehen. [vgl. StHM08]

Die Autoren in [NCLP22] stellen einen quelloffenen Object-Detection-Datensatz namens „KOLOMVERSE“ für die Erkennung von unterschiedlichen maritimen Objekten vor. Der Datensatz besteht insgesamt aus 2.151.470 4K-Bildern, die von 17 Kameraschiffen vor der südkoreanischen Küste aufgenommen wurden. Neben seinem großen Umfang und der hohen Bildauflösung setzt sich KOLOMVERSE auch durch seine hohe Diversität ab.

Im Gegensatz zu einigen anderen Datensätzen in der maritimen Domäne werden hier nicht nur ausschließlich Schiffe als Klassen annotiert, sondern auch andere maritime Objekte wie Off-Shore-Windräder, Leuchttürme und (Fischer)-Bojen. Außerdem bilden die Bilder unterschiedliche Tageszeiten, Wellenhöhen, Windgeschwindigkeiten, Sichtverhältnisse und Wetterbedingungen wie Regen oder Schnee ab. Nach der Erstellung des Datensatzes testeten die Autoren dessen Eignung, indem sie unterschiedliche Object-Detection-Modelle wie „YOLOv3“, „YOLOv4“, „SSD“, „FRCNN“ und „CenterNet“ auf diesem trainierten. Dabei kamen sie zu dem Ergebnis, dass das CenterNet-Modell die höchste Vorhersagegenauigkeit ($mAP_{0.5} = 61,86$ Prozent) erreicht und sich der Datensatz als Benchmark und Trainingsdatensatz für die Anwendung in der maritimen Domäne eignet. Um die Diversität des Datensatzes weiter zu verbessern, sollten jedoch noch weitere Objektklassen in KOLOMVERSE integriert werden, um der aktuell bestehenden Klassen-Imbalance (82,9 Prozent aller Klasseninstanzen sind Schiffe) entgegenzuwirken. [vgl. NCLP22]

Das Paper [FeGL10] evaluiert eine Möglichkeit, Schiffe mit Hilfe einer Kamera-Boje zu erkennen und deren Bewegungen zu verfolgen (Tracking). Dabei stellte sich heraus, dass insbesondere die häufigen, durch den Wellengang bedingten, Perspektivwechsel die Vorhersagegenauigkeit der Modelle minderten. Um dieses Problem zu lösen, wurde ein Algorithmus zur Horzonterkennung entwickelt, um die Perspektive der Kamerabilder zu vereinheitlichen. Durch dieses Vorgehen konnten die verwendeten KI-Algorithmen für die Erkennung und das Tracking der Schiffe deutlich verlässlichere Ergebnisse erzielen. [vgl. FeGL10]

[ZGLZ22] beschäftigt sich mit der Optimierung eines Object-Detection-Modells auf die Erkennung von Schiffen und Menschen bei nebeligem Wetter. Für die Optimierung greifen die Autoren auf das bereits bestehende „YOLOv4_tiny“-Modell zurück, da es eine hohe Genauigkeit aufweise und Vorhersagen in Echtzeit treffen könne. Da zum Zeitpunkt dieser Arbeit noch kein geeigneter Datensatz für die Erkennung von maritimen Objekten in nebeligen Umgebungen existierte, wurde ein eigener Datensatz erstellt. Dieser besteht aus insgesamt 3.838 Bildern und bildet sowohl Schiffe als auch Menschen im Wasser ab. Zur Simulation des Nebels wurde ein entsprechender Algorithmus eingesetzt. Das YOLOv4-Modell wurde anschließend an drei Stellen optimiert. Zuerst wurde ein sogenannter *Dehazing*-Algorithmus namens „Single Scale Retinex“ eingesetzt, der den Nebel vor der Feature-Extraktion aus den Bildern herausrechnet. So soll die Fehleranfälligkeit (falsche und fehlende Vorhersagen) des Modells verringert werden. Anschließend wurde ein sogenanntes „Receptive Field Block Module“ verwendet. Dieses Modul vergrößert das sogenannte *rezeptive Feld* des Modells, wodurch detailliertere Feature-

Informationen extrahiert und kleiner Objekte wie Menschen besser erkannt werden können. Als letzte Modifikation wandten die Autoren einen Aufmerksamkeitsmechanismus namens „Convolutional Block Attention Module“ (CBAM) an. Das CBAM verbessert die Fähigkeit zur Feature-Extraktion des Modells, indem ein Fokus auf die wichtigsten Features im Bild gelegt wird. Das so entwickelte „SRC-YOLO“-Modell konnte die Mean Average Precision im Vergleich zum ursprünglichen YOLOv4-tiny-Modell von 79,56 auf 86,15 Prozent steigern. Um die Robustheit weiter zu erhöhen, sei es notwendig, das Modell auch auf andere Wetterbedingungen wie Wind, Regen, starken Wellengang sowie Sonnenschein und die daraus resultierenden Reflexionen und Spiegelungen zu optimieren. Außerdem müsse weitere Forschung betrieben werden, um die Erkennung von sehr kleinen Objekten zu verbessern. [vgl. ZGLZ22]

In [KaYH16] beschäftigen sich die Autoren damit, Schiffe auf Kamerabildern erkennen und tracken zu lassen. Für die Erkennung und Lokalisation der Schiffe wird eine Kombination aus *Laplace-Filter* und *Support Vector Machine* (SVM) eingesetzt. Die Support Vector Machine dient dazu, die zuvor vom Laplace-Filter erzeugten Vorschläge für Schiffe zu untersuchen und fehlerhafte Kandidaten wie Wellen und Reflexionen auszusortieren. Das Tracking der Schiffe wird durch die Nutzung eines Partikelfilters umgesetzt. Auch wenn der hier entwickelte Ansatz bereits gute Ergebnisse erziele, könne vor allem das Tracking der Schiffe durch weitere Informationen wie der aktuellen Geschwindigkeit und dem Kurs des Schiffes, durch 3D-Kameras, Radar- und AIS-Daten noch weiter verbessert werden. [vgl. KaYH16]

Die Arbeit von [MKJT19] beschäftigt sich damit, einen Benchmark-Datensatz für die maritime Domäne zu etablieren. Als Basis wird dafür der bereits existierende domänenspezifische Datensatz Singapore Maritime Dataset herangezogen. Die Autoren schlagen einen Train-Valid-Test-Split für das SMD vor, um eine Reproduzierbarkeit und Vergleichbarkeit sicherzustellen. Zudem wurde hier ein Algorithmus konzipiert, um das SMD um Segmentations-Label zu erweitern. Auf dem so neukonzipierten SMD wurde anschließend die Performance der beiden Modelle „Faster R-CNN“ und „Mask R-CNN“ evaluiert. Es konnte festgestellt werden, dass insbesondere Mask R-CNN eine hohe Vorhersagegenauigkeit bei der Objekterkennung in der maritimen Domäne aufweist. Beide Modelle hatten Probleme damit, kleine und sich gegenseitig überlappende Objekte zu erkennen. Auch durch schlechte Wetterbedingungen wie Nebel entstanden Detektionsfehler. [vgl. MKJT19]

In [BMPK18] wird ein Algorithmus namens „ISSM“ zur Erkennung von Hindernissen im Wasser und des Horizonts (Übergang zwischen Meer und Küste oder Meer und Himmel) in MATLAB entwickelt. Für die Evaluation des Algorithmus wurde ein eigener, umfassender Datensatz namens „Maritime Obstacle Detection Dataset 2“ (MODD2) aufgebaut. Die darin enthaltenen Bilder wurden von einem kleinen *Unmanned Surface Vehicle* (USV) über mehrere Monate mithilfe eines Stereo-Kamerasystems aufgenommen. Der Datensatz deckt verschiedene Wetterbedingungen wie Nebel und Sonnenreflexionen ab. Für die Entwicklung des Algorithmus wurden auch die Daten der *Inertial Measurement Unit* (IMU) integriert, um die Detektion des Horizonts zu verbessern. Im Rahmen des hier entwickelten Ansatzes werden alle Pixel auf den Bildern den drei Bereichen Himmel, Küste und Meer zugewiesen. Neben dem Horizont bzw. dem Meeresrand kann die Methode außerdem kleine und große Objekte im Wasser erkennen. Die Evaluation ergab, dass ISSM die Performance des damaligen State-of-the-Art Modells namens „SSM“ bei der Horizont- und Objekterkennung im Hinblick auf Vorhersagegenauigkeit übertrifft. Außerdem ermöglichte die MATLAB-Implementierung, dass der Algorithmus in Echtzeit arbeiten könne. [vgl. BMPK18]

[SpSS20] untersucht die Eignung von unterschiedlichen State-of-the-Art Object-Detection-Modellen, eine Schiffserkennung durchzuführen. Hier analysieren die Autoren frei verfügbare Datensätze dahingehend, wie gut sie sich für die Anwendung als Trainingsdatensatz und Benchmark in der maritimen Domäne eignen. Neben den großen und bekannten Datensätzen „MS COCO“ und „PASCAL VOC“ wurden auch domänenspezifische Datensätze wie das „Singapur Maritime Dataset“ (SGD) oder „SeaShips7000“ mitberücksichtigt. Die Autoren bemängeln, dass Machine-Learning-Modelle stark auf die großen und bekannten Datensätze optimiert werden. Dieses Vorgehen führe dazu, dass die vortrainierten Modelle und deren Architekturen einen starken Bias aufweisen und die Fähigkeit verlieren, auch auf bisher unbekanntem Bildern gut zu performen. Für die Evaluation wurden die Modelle „Faster R-CNN“, „Cascade Faster R-CNN“, „Libra R-CNN“, „RetinaNet“ und „FCOS“ getestet. Es stellte sich heraus, dass das FCOS-Modell in diesem Anwendungsfall am besten geeignet war. Das trainierte FCOS-Modell wurde anschließend dazu verwendet, die oben genannten Datensätze auf deren Fähigkeit zur Generalisierung zu untersuchen. Dazu wurde das FCOS-Modell nacheinander ausschließlich auf einem der vier Datensätze trainiert und anschließend auf allen Datensätzen getestet. Dabei konnte festgestellt werden, dass das Modell nach jedem Training Probleme damit hatte, die Objekte in den anderen Datensätzen zu erkennen. Insbesondere beim SeaShips7000-Datensatz konnte eine starke Überanpassung des Modells beobachtet werden. Die besten Ergebnisse wurden erzielt, als das FCOS-Modell auf einer

Kombination aller Trainingsdatensätze trainiert und das Training durch weitere Webcrawler-Inhalte ergänzt wurde. [vgl. SpSS20]

Das Paper [FSNP21] untersucht die Anwendungsgebiete von DL-Techniken in der Computer Vision zur Anwendung in der maritimen Domäne. Dafür reviewten die Autoren mehrere wissenschaftliche Artikel und untersuchten jeweils das Ziel der Arbeit, den verwendeten Datensatz und das eingesetzte Deep-Learning-Modell. Der häufigste Anwendungsfall von DL-Algorithmen war die Erkennung und Lokalisierung von Schiffen (Object Detection oder Instance Segmentation). Die dafür verwendeten Datensätze enthielten entweder Bilder von der Wasseroberfläche oder Satellitenbilder. Weitere Ziele waren die Erkennung von Unterseeequipment wie Rohre oder Ventile, die Lokalisierung und Klassifizierung von unterschiedlichen Korallenarten und anderen Meereslebewesen, das Auffinden von Unterwasserminen und die Detektion von Ölteppichen. Je nach Anwendungsfall bestanden die Datensätze entweder aus Satellitenbildern oder Unterwasser-aufnahmen. Die mit Abstand am häufigsten eingesetzte KNN-Architektur war das CNN. Für die Schiffserkennung wurden die Modelle „YOLO“, „Fast R-CNN“ und „SSD“ evaluiert, wobei das Fast R-CNN die beste Performance aufwies. Auch bei der Korallenerkennung wurden CNN-basierte Modelle wie „ResNet-50“ und „VGG-16“ eingesetzt. [vgl. FSNP21]

Wie aus der Beschreibung der verwandten Arbeiten hervorgeht, konzentriert sich die aktuelle Forschung im Bereich der automatisierten Objekterkennung in der maritimen Domäne nahezu ausschließlich auf die Detektion von Schiffen (siehe [NCLP22], [FeGL10], [ZGLZ22], [KaYH16], [MKJT19], [BMPK18] und [SpSS20]), während statische Hafenobjekte entweder gar nicht oder nur unzureichend betrachtet werden. Kaimauern und Dalben sind in keinem der untersuchten Arbeiten zu finden. Bojen wurden nur in [vgl. NCLP22] als Zielklasse mitberücksichtigt. Diese bestehende Forschungslücke soll im Rahmen dieser Arbeit adressiert werden. Dazu soll die Eignung von KI-Modellen im Hinblick auf die automatisierte Erkennung von statischen Hafenobjekten untersucht werden.

4 Anforderungserhebung

In diesem Kapitel werden Anforderungen an das zu implementierende KI-Modell zur automatisierten Erkennung von statischen Hafenobjekten gestellt. Die Erhebung dieser Anforderungen stellt sicher, dass die Bearbeitung der im Rahmen dieser Arbeit definierten Teilziele zur Beantwortung der Forschungsfrage unter der Berücksichtigung geeigneter Vorgaben erfolgt. Das konkrete Konzept zur Umsetzung der Anforderungen und Erreichung der definierten Teilziele wird in Kapitel 5 erläutert. Die Auswertung der Anforderungen erfolgt in Kapitel 7.

4.1 Methodik

Für die Anforderungserhebung werden die sogenannten *MASTeR-Schablonen* verwendet. Diese Schablonen dienen als Vorlagen für die strukturierte Formulierung von funktionalen und nicht-funktionalen Anforderungen im Rahmen des Requirement Engineering [vgl. Soph16, S.7]. Das Ziel dieser Arbeit liegt nicht darin, ein vollumfängliches Softwaresystem mit Schnittstellen zu anderen Systemen und Benutzern zu implementieren. Stattdessen soll ein konkretes ML-fähiges KI-Modell aus dem Bereich der automatisierten Objekterkennung ausgewählt und implementiert werden. Aus diesem Grund beschränkt sich diese Anforderungserhebung auf die Erhebung von funktionalen Anforderungen an das zu implementierende Modell unter Zuhilfenahme des sogenannten *FunktionsMASTeRs* [vgl. Soph16, S.11]. Muss-Anforderungen sind zwingend zu erfüllen; Soll-Anforderungen sind optional umsetzbar.

4.2 Anforderungen

Anforderung 1

Das Modell muss im maritimen Umfeld einsetzbar sein.

Anforderung 2

Das Modell muss statische Hafenobjekte auf Bildern erkennen können.

Anforderung 3

Das Modell muss statische Hafenobjekte bei Tageslicht erkennen können.

Anforderung 4

Das Modell soll statische Hafenobjekte bei Dunkelheit bzw. geringem Lichteinfall erkennen können.

Anforderung 5

Das Modell muss statische Hafenobjekte variierender Größe erkennen können.

Diese Anforderung stellt sicher, dass das Modell in der Lage ist, die statischen Hafenobjekte sowohl aus nächster Nähe als auch aus einiger Entfernung zuverlässig zu erkennen. Die frühzeitige Erkennung der Hindernisse ist wichtig, damit das Modell frühzeitig über mögliche Gefahren informieren kann.

Anforderung 6

Das Modell muss statische Hafenobjekte auch dann zuverlässig detektieren können, wenn diese aufgrund von Überlappungen mit anderen Objekten nicht vollständig sichtbar sind.

Anforderung 7

Das Modell muss vom Bildrand abgeschnittene Objekte erkennen können.

Anforderung 8

Das Modell muss statische Hafenobjekte bei Wellenbewegungen erkennen können.

Bei anhaltendem Wellengang schwankt die Kamera auf dem Boot mit. Dies führt dazu, dass auch die Perspektive auf die erkennenden Hafenobjekte variiert (leichte Neigung). Diese Anforderung stellt sicher, dass das zu implementierende Modell in diesem Fall robuste Ergebnisse liefert.

Anforderung 9

Das Modell sollte statische Hafenobjekte bei unterschiedlichen Wetterbedingungen wie Sonnenschein, Niederschlag und Nebel erkennen können.

Anforderung 10

Das Modell muss eine Vorhersagegenauigkeit (mAP_0.5) von mindestens 95 Prozent erreichen.

Anforderung 11

Das Modell sollte eine Vorhersagegeschwindigkeit von 30 Bildern pro Sekunde erreichen.

Diese Anforderung stellt sicher, dass das Modell bei der Verarbeitung von bisher ungesesehenen Bildern schnelle Vorhersagen trifft. Dies soll ermöglichen, dass das Modell bei Bedarf in ein echtzeitfähiges Frühwarn- und Navigationssystem eingebunden werden kann.

Anforderung 12

Das Modell sollte auch bisher ungesehene Kaimauern und Dalben auf Bildern aus anderen Hafengebieten zuverlässig erkennen können.

Dalben und Kaimauern unterscheiden sich je nach Hafengebiet stark in ihrem Aussehen (z.B. Größe, Farbe und Zusammensetzung). Mit dieser Anforderung soll erreicht werden, dass sich das implementierte Modell für die automatisierte Erkennung von statischen Hafengebieten in anderen Hafengebieten verwenden lässt.

5 Konzept

In diesem Kapitel wird das Konzept für die Erreichung der in dieser Arbeit definierten Teilziele zur Beantwortung der Forschungsfrage beschrieben. Durch das Konzept soll sichergestellt werden, dass alle Maßnahmen zum Aufbau des Datensatzes und zur Implementierung des KI-Modells auf der Grundlage von begründeten Entscheidungen getroffen werden.

5.1 Teilziele und Arbeitsschritte

Ziel dieser Arbeit ist es, die in Kapitel 1.1 beschriebene Problemstellung zu adressieren und die formulierte Forschungsfrage zu beantworten. Um dies zu erreichen, wurden zwei konkrete Teilziele definiert, die im Rahmen dieser Arbeit erfüllt werden sollen (siehe Kapitel 1.2). Außerdem wurden Anforderungen erhoben, die bei der Umsetzung der Teilziele zu berücksichtigen sind (siehe Kapitel 4). In diesem Kapitel wird das konkrete Konzept zur Umsetzung der Teilziele erläutert. Das in Abbildung 22 dargestellte Flussdiagramm visualisiert den Ablauf der durchzuführenden Arbeitsschritte.

Das erste Teilziel der Arbeit besteht darin, einen domänenspezifischen Datensatz für die Erkennung von statischen Hafengebäuden auf der Basis von Kamerabildern aufzubauen. Als Grundlage für den Datensatz werden Bilder aus dem Emdener Hafen verwendet. Bevor diese annotiert werden können, müssen die Frames zunächst aus den Videos des vorliegenden Rohmaterials extrahiert und aufbereitet werden. Danach sollen die statischen Hafengebäude mithilfe eines geeigneten Labeling-Tools und auf der Basis einer vordefinierten Labeling-Strategie gelabelt werden. Ist dieser Vorgang abgeschlossen, sollen die Annotationen einer Qualitätskontrolle unterzogen werden, um mögliche Fehler zu korrigieren.

Anschließend kann damit begonnen werden, eine passende KI-Modellarchitektur auf der Basis der in Kapitel 1.2 erhobenen Anforderungen auszuwählen. Danach wird das ausgewählte Modell parametrisiert, um eine optimale Anpassung an den vorliegenden Anwendungsfall – die automatische Erkennung von statischen Hafengebäuden auf der Basis von Kamerabildern aus dem Emdener Hafen – zu erreichen. Ist die Parametrisierung abgeschlossen, soll die Leistungsfähigkeit des finalen Modells anhand von geeigneten Performance-Metriken evaluiert werden. Durch dieses Vorgehen lässt sich feststellen, inwiefern das implementierte Modell für eine automatisierte Erkennung von statischen Hafengebäuden geeignet ist. Außerdem lassen sich dadurch die Stärken und Schwächen des Modells quantifizieren und bewerten.

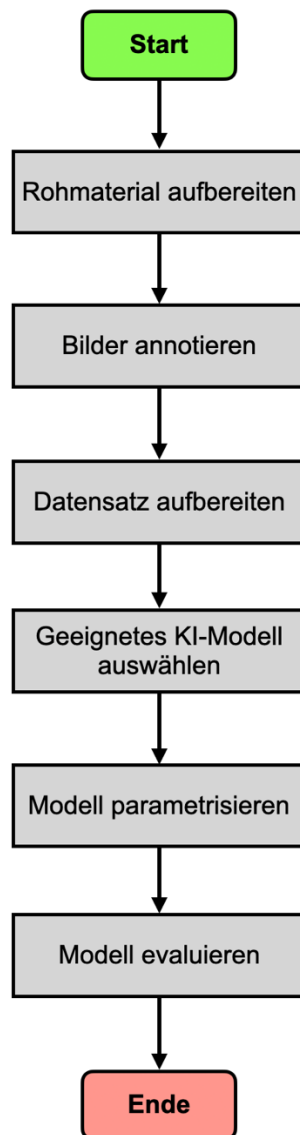


Abbildung 22: Ablauf der durchzuführenden Arbeitsschritte

Als Teil der Performanceevaluation soll in einem letzten Schritt untersucht werden, inwiefern das Modell in der Lage ist, Bilder von statischen Hafengebieten zuverlässig zu erkennen.

5.1.1 Konzeptionierung des Datensatzes

Es existiert aktuell kein Datensatz, der ausschließlich statische Hafengebiete in ausreichender Anzahl und Variation repräsentiert. Aus diesem Grund soll ein eigener Datensatz zur Repräsentation der statischen Hafengebiete auf der Basis von Kamerabildern aus dem Emdener Hafen aufgebaut werden. Zuerst muss dabei entschieden werden, welche konkreten Hafengebiete im Datensatz repräsentiert werden sollen. In dieser Arbeit

wurde sich dafür entschieden, ausschließlich Kaimauern und Dalben als Zielklassen abzubilden. Grund dafür ist, dass diese beiden Objekte in keiner der untersuchten wissenschaftlichen Arbeiten oder marktreifen Lösungen berücksichtigt wurden. Außerdem kommen sowohl Kaimauern als auch Dalben häufig im Rohmaterial aus dem Emdener Hafen vor. Damit ist sichergestellt, dass das Modell ausreichend viele Beispiele zur Verfügung gestellt bekommt, um das Aussehen dieser Objekte zu erlernen.

Eine wichtige Eigenschaft eines Datensatz ist die Anzahl an Bildern und Annotationen pro Zielklasse. Nur wenn dem Modell eine ausreichend große Anzahl an Grundwahrheits-Bounding-Boxen präsentiert wird, ist es in der Lage, die Kaimauern und Dalben in verschiedenen Situationen zuverlässig zu erkennen. Im Allgemeinen wird empfohlen, dass jede Klasse im Datensatz jeweils mindestens 10.000 Annotationen besitzt und jeweils in mindestens 1.500 unterschiedlichen Bildern vorkommt [vgl. Ultr23a]. Daher soll diese Mindestanforderung auch beim Aufbau dieses Datensatz erfüllt werden.

5.1.2 Aufbereitung des Rohmaterials

Durch die Aufbereitung des Rohmaterials werden die Bilder für das Labeling vorbereitet. Abbildung 23 visualisiert die vier Arbeitsschritte, die im Rahmen der Datenaufbereitung durchgeführt werden müssen.

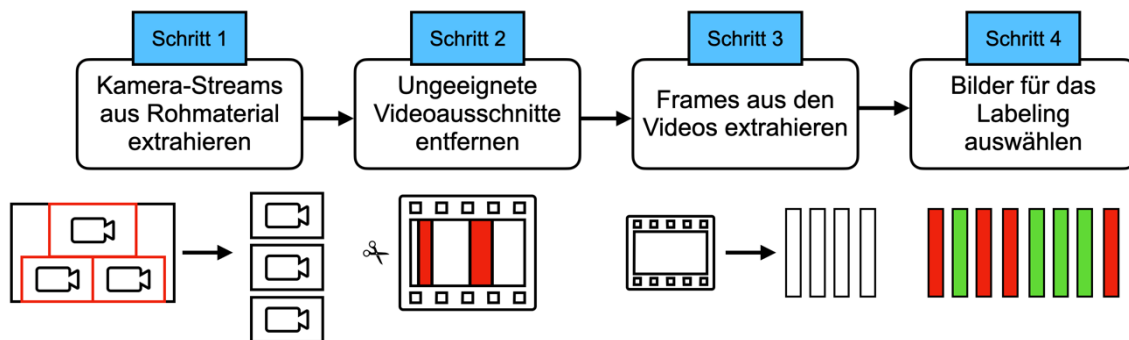


Abbildung 23: Aufbereitung des Rohmaterials

In einem ersten Schritt muss sichergestellt werden, dass die drei Kamera-Streams aus den Rohvideos extrahiert werden. Ein ML-Algorithmus lernt, in welchen Bildbereichen Zielobjekte häufiger vorkommen. Ein Training auf dem unverarbeiteten Rohmaterial hätte demnach zur Folge, dass das Modell nicht korrekt auf den Einsatz im Emdener Hafen vorbereitet werden würde. Der ML-Algorithmus würde sich im Verlaufe des Trainings darauf einstellen, dass Objekte ausschließlich an den Stellen vorkommen, an denen sich

im Rohmaterial die drei Kamera-Streams befinden. In der Realität kommen allerdings keine Bilder vor, die drei Kamerabilder in einem Frame zusammenfassen. Die großen schwarzen Bereiche zwischen den Kamera-Streams sind dort nicht vorhanden. Im Produktivbetrieb arbeitet das Modell stattdessen auf Frames, die das gesamte Bild ausfüllen. Das Training auf den unverarbeiteten Rohdaten hätte somit außerdem den Nachteil, dass das Modell davon ausgehen würde, dass die Zielobjekte deutlich kleiner sind, als dies eigentlich in der Realität der Fall ist. All dies würde die Genauigkeit des Modells im Produktivbetrieb schmälern.

In einem zweiten Schritt müssen ungeeignete Videoausschnitte aus den zuvor extrahierten Video-Streams entfernt werden. Dies betrifft vor allem Passagen, in denen sich das Kameraboot nicht bewegt. Diese Maßnahme soll dafür sorgen, dass nicht zu viele nahezu identische Bilder in den Datensatz aufgenommen werden. Die Integration derartiger Bilder in den Datensatz hätte zur Folge, dass sich das Modell zu stark auf die Erkennung der immergleichen Kaimauern und Dalben anpassen würde. Dies hätte zur Folge, dass das Modell seine Fähigkeit zur Generalisierung verlieren würde.

Anschließend müssen die Videos in ihre einzelnen Frames unterteilt werden. Hier muss ebenfalls sichergestellt werden, dass nicht zu viele nahezu identische Bilder entstehen. Deshalb soll nicht jeder Frame, sondern nur ein Bild pro Sekunde, aus den Videos extrahiert werden. Alle anderen Frames werden verworfen. Dieses Vorgehen wird auch von [MKJT19] empfohlen, um eine Überanpassung des Modells zu verhindern.

In einem letzten Schritt werden die Bilder ausgewählt, die Teil des Datensatzes werden sollen. Die Auswahl der Bilder aus dem Pool der extrahierten Frames erfolgt zufällig. Diese Maßnahme stellt sicher, dass alle Videopassagen im Datensatz repräsentiert sind. Damit nimmt die Vielfalt des Datensatzes zu, was die Robustheit des Modells verbessert.

5.1.3 Annotation der Bilder

Nach der Aufbereitung des Rohmaterials müssen die einzelnen Frames bzw. Bilder annotiert werden. Dafür muss zunächst bestimmt werden, welche grundlegende Methodik für die Annotation der Bilder verwendet werden soll. Diese Entscheidung hat einen großen Einfluss darauf, welche KI-Modelltypen für das Training auf dem so entstehenden Datensatz in Frage kommen. Ziel der Arbeit ist es, eine automatisierte Erkennung von Objekten auf der Basis von Kamerabildern umzusetzen. Dafür soll ein ML-fähiges KI-Modell auf der Basis eines neuronalen Netzes verwendet werden. Für derartige Anwendungsfälle bieten sich KI-basierte Algorithmen aus dem Bereich der Computer Vision an. In der Computer Vision gibt es vier unterschiedliche Ansätze, um Objekte auf Bildern zu

erkennen – die Classification, die Semantic Segmentation, die Object Detection und die Instance Segmentation. Im vorliegenden Anwendungsfall kommt es häufiger vor, dass mehrere Objektinstanzen aus unterschiedlichen Klassen in einem einzelnen Bild erkennbar sind (z.B. eine Kaimauer und mehrere Dalben). Von den vier großen Anwendungsgebieten der Computer Vision lassen sich demnach nur die Object Detection und Instance Segmentation sinnvoll einsetzen. Ein Classification-Modell klassifiziert das gesamte Bild und ist nicht in der Lage, einzelne Objekte im Bild zu identifizieren. Bei der Image Segmentation können keine individuellen Objektinstanzen (Dalbe 1, Kaimauer 4) erkannt werden. Im Vergleich zur herkömmlichen Object Detection nimmt das Labeling von Objekten bei der Instance Segmentation in der Regel ein Vielfaches an Zeit in Anspruch. Das liegt daran, dass dort Polygone verwendet werden, um die Objekte zu markieren. Aufgrund der zeitlichen Limitation wurde sich daher dafür entschieden, die Dalben und Kaimauern mithilfe der klassischen, achsenparallelen Object-Detection-Bounding-Boxen zu annotieren.

Nach der Beendigung des Labelings, soll eine abschließende Qualitätskontrolle des Datensatzes durchgeführt werden. Im Rahmen dieser Kontrolle soll die Qualität der Label überprüft werden, indem unsaubere bzw. fehlerhafte Bounding Boxen korrigiert werden. Dies betrifft Annotationen, bei denen ein falsches Klassenlabel gesetzt oder die Bounding Box nicht korrekt angeordnet wurde. Auch sollen Objekte manuell nachannotiert werden, falls diese im ersten Durchgang übersehen wurden.

5.1.4 Aufbereitung des Datensatzes

Es kann passieren, dass der gelabelte Datensatz ein Ungleichgewicht bei der Anzahl an Klassenannotationen aufweist. Dieses Ungleichgewicht kann dazu führen, dass ein Modell die dominante Klasse bevorzugt und die seltenere Klasse im Trainingsprozess vernachlässigt [vgl. Brem20]. Dies hat zur Folge, dass das Modell verzerrte Vorhersagen trifft und die Fähigkeit verliert, die unterrepräsentierte Klasse auf bisher ungesehenen Bildern zuverlässig zu erkennen, wenn dort dieses Ungleichgewicht nicht besteht [vgl. Brem20]. Es gibt keine einheitliche Regel dafür, ab wann eine Klassen-Imbalance vorliegt. Im Allgemeinen wird angenommen, dass dies der Fall ist, wenn eine Klasse ungefähr zehnmal so viele Annotationen ausweist wie die nächsthäufigere Klasse im Datensatz [vgl. Volp19]. In diesem Fall müssten Maßnahmen ergriffen werden, um das Ungleichgewicht zu beheben. Eine klassische Maßnahme besteht darin, entweder die Anzahl an Bilder und Annotationen der Minderheitsklasse künstlich zu erhöhen oder die Anzahl an Bilder und Annotationen der Mehrheitsklasse künstlich zu verringern [vgl. Brem20]. Die künstliche Anpassung der Anzahl an Annotationen in einzelnen Klassen

kann allerdings dazu führen, dass eine Überanpassung des Modells begünstigt wird oder ein großer Teil der Bilder aus dem Datensatz entfernt werden muss [vgl. Brem20]. Außerdem gibt es die Möglichkeit, die Klassen unterschiedlich stark zu gewichten [vgl. Brem20]. Für den Fall, dass ein starkes Klassenungleichgewicht (Verhältnis 1:10 oder kleiner) im gelabelten Datensatz besteht, soll eine Anpassung der Klassengewichte vorgenommen werden. Andernfalls haben keine weiteren Maßnahmen zu erfolgen.

Im Produktivbetrieb kann es häufiger vorkommen, dass sich zu einem gewissen Zeitpunkt kein Zielobjekt im Blickfeld der Kameras befindet. In diesem Fall muss das Modell lernen, dass nicht immer eine Vorhersage getätigt werden kann [vgl. Dwyer20]. Dies kann erreicht werden, indem ein gewisser Anteil dieser Bilder ohne Annotationen (auch *Null-Beispiele* genannt) im Datensatz belassen werden. Es sollte verhindert werden, dass ein Modell auf zu vielen Null-Beispielen trainiert wird, damit es nicht dazu neigt, weniger Vorhersagen zu machen, als eigentlich notwendig wären [vgl. Dwyer20]. Im Allgemeinen wird empfohlen, bis zu zehn Prozent der Null-Beispiele im Datensatz zu behalten, um das Auftreten von False Positives zu vermeiden [vgl. Ultr23a]. Im Rahmen der Evaluation des fertiggestellten Datensatzes soll darauf geachtet werden, dass diese Obergrenze eingehalten wird. Überschüssige Null-Beispiele sollen erst dann entfernt werden, wenn der Anteil an Null-Beispielen im Datensatz mindestens elf Prozent beträgt.

Es ist wichtig, dass der Datensatz die beiden Zielklassen auf eine möglichst vielfältige Art und Weise repräsentiert, sodass das darauf trainierte Modell auf den Einsatz in der echten Welt vorbereitet wird. Um dies zu erreichen, sollten die Objekte aus unterschiedlichen Perspektiven und Blickwinkeln, Entfernungen, unter variierenden Wetterbedingungen (Sonnenschein, Bewölkung, Niederschlag oder Nebel) und Lichtverhältnissen sowie zu unterschiedlichen Tageszeiten abgebildet werden. Auch Überlappungen der Objekte untereinander, Überschneidungen mit anderen Objekten in der Umgebung und dem Bildrand sollten vertreten sein, um die Robustheit des KI-Modells zu erhöhen. Das Problem ist, dass die vorliegenden Bilder aus dem Emden Hafen nur einen Teil dieser Diversitätsanforderungen erfüllen. Vorteilhaft ist, dass die Aufnahmen von drei unterschiedlichen Kameras aufgezeichnet wurden. Daher sind auch die Kaimauern und Dalben aus jeweils drei unterschiedlichen Perspektiven bzw. Blickwinkeln abgebildet. Auch die Neigung der Kameras unterscheidet sich leicht, sodass der Neigungswinkel der aufgezeichneten Kaimauern und Dalben ebenfalls variiert. Auch Überlappungen der Dalben und Kaimauern untereinander, Überschneidungen mit anderen Objekten in der Umgebung und dem Bildrand sind repräsentiert. Nachteilig ist, dass die Aufnahmen ausschließlich am helllichten Tag und bei Sonnenschein bzw. bewölktem Himmel gemacht wurden.

Um die Vielfältigkeit des Datensatzes zu verbessern und das Modell auf den Produktivbetrieb vorzubereiten, soll eine sogenannte *Datenaugmentierung* verwendet werden. Die Datenaugmentierung ermöglicht es, den Umfang des Datensatzes durch das Hinzufügen von modifizierten Kopien der ursprünglichen Bilder künstlich zu erweitern [vgl. Tens00]. Durch die Anwendung einer Datenaugmentierung wird die Vielfältigkeit des Datensatz erhöht, was die Gefahr für eine Überanpassung des Modells verringert [vgl. Tens00]. Ein weiterer Vorteil ist, dass sich durch diese Technik verschiedene Wetterbedingungen, Lichtverhältnisse und Perspektivwechsel imitieren lassen.

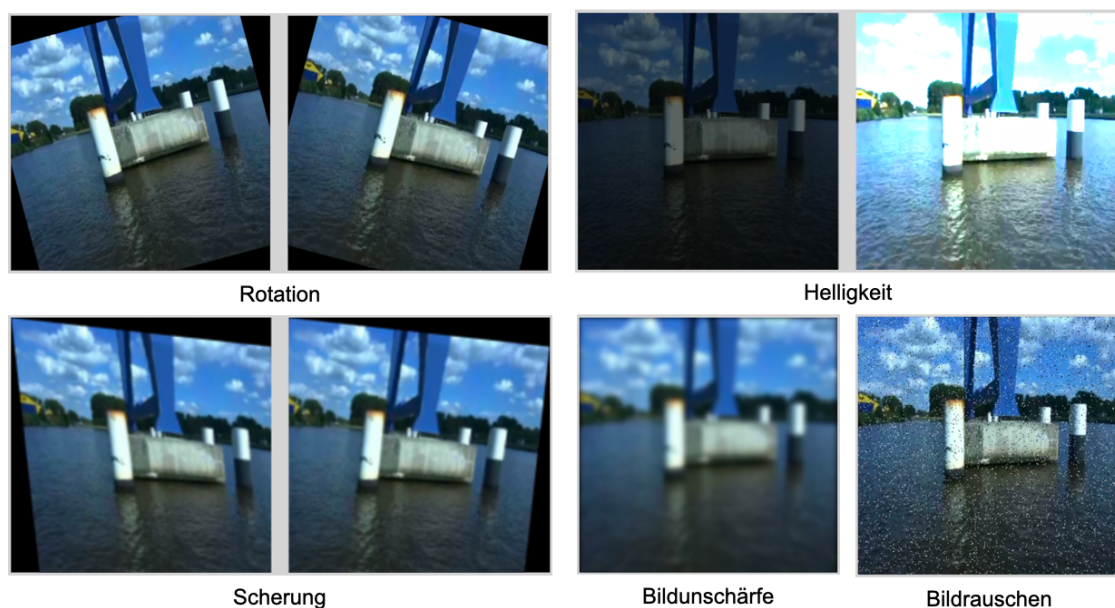


Abbildung 24: Datenaugmentierung

Abbildung 24 zeigt die Transformationsschritte auf, die im Rahmen der Augmentierung angewendet werden sollen. Zum einen soll eine Rotation bzw. Scherung der Bilder vorgenommen werden. Diese beiden Augmentierungen sollen die durch starken Wind und/oder Wellengang entstehenden Schwankungen simulieren (siehe Anforderung 8). Diese Schwankungen führen dazu, dass sich auch der Blickwinkel bzw. die Perspektive der Kameras auf die Hafengebäude verändert (leichte Neigung). Die Helligkeits-Augmentierung wird dafür verwendet, um unterschiedliche Lichtverhältnisse zu imitieren. Die künstliche Abdunkelung der Bilder soll es ermöglichen, dass das Modell die Objekte im späteren Produktivbetrieb auch in dämmerigem Licht erkennen kann (siehe Anforderung 4). Die Implementierung des Bildrauschens kann dabei helfen, Regen, Hagel, Schnee oder Nebel zu simulieren (siehe Anforderung 9) und das Modell auf technische Störungen vorzubereiten [vgl. Imag23]. Die künstliche Bildunschärfe soll verwendet werden,

um Wasserstopfen auf der Kameralinse zu simulieren. Außerdem lässt sich damit simulieren, dass eine oder mehrere Kameras ihren Fokus verlieren.

5.1.5 Auswahl eines geeigneten KI-Modells

Wie bereits erläutert, soll in dieser Arbeit ein domänenspezifischer Datensatz aufgebaut werden, der Kaimauern und Dalben als statische Hafenobjekte repräsentiert. Die Annotation der Objekte erfolgt auf der Basis von Object-Detection-Bounding-Boxen. Das in dieser Arbeit zu implementierende KI-Modell muss daher ebenfalls in der Lage sein, derartige Label zu verarbeiten. Aus diesem Grund kommen für die Auswahl einer geeigneten Modellarchitektur ausschließlich Modelle aus dem Bereich der Object Detection in Frage. Dies stellt sicher, dass das Modell mit dem aufgebauten Datensatz kompatibel ist und dieser damit als Benchmark für das Training und die Evaluation verwendet werden kann.

Das Kernziel dieser Arbeit besteht darin, die Eignung einer konkreten Modellarchitektur für die automatisierte Erkennung von statischen Hafenobjekten zu untersuchen. Dabei soll das Modell auf den in dieser Arbeit behandelten Anwendungsfall – die Erkennung von Kaimauern und Dalben auf der Basis von Kamerabildern eines hochautomatisierten Baggerschiffes im Emdener Hafen – optimiert werden. Wie bereits in den Kapiteln 2.3.1 und 2.3.2 erläutert, gibt es zwei grundlegende Typen von Object-Detection-Modellen – einstufige und zweistufige Detektoren. Beide Ansätze bringen unterschiedliche Vor- und Nachteile mit sich. Seit einigen Jahren werden zweistufige Detektoren mit Region Proposal Network immer seltener verwendet [vgl. ZCSG23, S.6]. Grund dafür ist, dass einstufige Detektoren mittlerweile in der Lage sind, vergleichbar hohe Mean-Average-Precision-Werte zu erreichen und gleichzeitig eine deutlich niedrigere Inferenzzeit als zweistufige Detektoren aufweisen [vgl. ZCSG23, S.4]. Diese Entwicklung wird auch dadurch widerspiegelt, dass seit 2017 kein neuer Meilenstein-Detektor bei der Entwicklung von zweistufigen CNN-Architekturen mehr entwickelt wurde [vgl. ZCSG23, S.2]. Im Bereich der einstufigen Detektoren wurden in den vergangenen Jahren immer wieder neue Modellarchitekturen hervorgebracht [vgl. ZCSG23, S.2].

Bei dem in dieser Arbeit behandelten Anwendungsfall spielt neben der Präzision auch die Vorhersagegeschwindigkeit des zu implementierenden Modells eine wichtige Rolle. Echtzeitfähige KI-Modelle sollten immer dann eingesetzt werden, wenn schnell auf mögliche Hindernisse reagiert werden muss [vgl. ISZL21, S.14]. Das hier zu implementierende Modell muss ebenfalls in der Lage sein, die statischen Hafenobjekte möglichst schnell zu erkennen und diese Informationen entweder automatisiert in seine Routenplanung einfließen zu lassen und/oder dafür zu nutzen, menschliche Akteure frühzeitig

über mögliche Hindernisse zu informieren. Die Verwendung eines echtzeitfähigen Modells könnte damit dazu beitragen, die Gefahr für mögliche Kollisionen zu verringern. Aus den genannten Gründen wurde sich dafür entschieden, ein einstufiges Modell aus dem Bereich der Object Detection zu implementieren.

Neben klassischen CNN-basierten Modellen lassen sich heute mittlerweile auch Transformer-Netzwerke als einstufige Object-Detection-Detektoren verwenden. Transformer versprechen eine ähnlich hohe oder bessere Performance zu erreichen wie die herkömmlichen CNN-Architekturen [vgl. ZCSG23, S.4] [vgl. ZMXF22, S.2]. In der Praxis ist erkennbar, dass aktuelle Transformer wie „Swin-B“ und „Dual-Swin-T“ und „Deformable DETR“ zwar mit der Vorhersagegenauigkeit der leistungsstärksten einstufigen CNN-Detektoren wie „YOLO“ mithalten können; sie benötigen allerdings etwa zehnmal so viel Zeit, um Objekte auf einem zuvor ungesehenen Bild zu erkennen (Inferenzzeit 100 vs. 10 Millisekunden) [vgl. WaBL22, S.11]. Dazu kommt, dass Transformer-Netzwerke in der maritimen KI-Forschung bisher kaum betrachtet wurden. Dies hat zur Folge, dass nur wenige Informationen über die Eignung derartiger Architekturen zur Objekterkennung im maritimen Umfeld vorliegen. Aus all diesen Gründen wird in dieser Arbeit auf die Implementierung eines Transformer-Netzwerks verzichtet und stattdessen auf einen klassischen, einstufigen CNN-Detektor zurückgegriffen.

Um ein geeignetes Modell zu finden, wird die verwandte Arbeit nach geeigneten Architekturen durchsucht werden. Dieses Vorgehen stellt sicher, dass das Modell grundsätzlich für den Einsatz im maritimen Umfeld geeignet ist (siehe Anforderung 1). Auch wenn sich in der maritimen Domäne bisher nahezu ausschließlich auf die Erforschung der Schiffserkennung fokussiert wurde, herrschen dort dieselben Umgebungsbedingungen wie bei der Erkennung von statischen Hafenobjekten. In beiden Fällen müssen die Modelle in der Lage sein, in der Größe variierende und/oder sich überlappende Objekte zu erkennen und dabei robust auf sich schnell ändernde Wetterbedingungen und Lichtverhältnisse zu reagieren. Darüber hinaus kann starker Wellengang und Wind dazu führen, dass die Objekte aus unterschiedlichen Neigungswinkeln aufgezeichnet werden. Durch das Zurückgreifen auf ein etabliertes Modell zur Anwendung in der maritimen Domäne kann zusätzlich auf bereits vorhandene Forschungsergebnisse im Hinblick auf die Parametrisierung des Modells zurückgegriffen werden.

Auf der Basis der oben beschriebenen Auswahlkriterien wurde sich dafür entschieden, die „YOLO“-Modellarchitektur für die automatisierte Erkennung der Kaimauern und Dalben zu verwenden. Das YOLO-Modell ist der am häufigsten verwendete CNN-Detektor im Bereich der Object Detection; darüber hinaus ist das Modell in der Lage, echtzeitfähige Anwendungsfälle zu unterstützen [vgl. WaBL22, S.7] [vgl. IRYM22, S.52819].

YOLO erreicht eine Vorhersagegenauigkeit, die dem aktuellen Stand der Technik entspricht [vgl. WaBL22, S.11]. Nicht zuletzt stellt YOLO eine sehr etablierte und weitverbreitete Detektor-Familie für die Objekterkennung in der maritimen Domäne dar. Außerdem wurde das Modell bereits für die Erkennung von Schiffen und anderen Hindernissen auf der Wasseroberfläche sowie von Objekten am Meeresgrund eingesetzt (siehe [SpSS20], [ISZL21], [SWWD18], [FSNP21], [ZGLZ22], [NCLP22] und [IRYM22]).

Das ursprüngliche YOLO-Modell existiert seit 2017. Seitdem werden immer wieder neue YOLO-Modellversionen veröffentlicht. Die drei neusten YOLO-Versionen sind YOLOv8¹, YOLOv7² und YOLOv5³. Um sich für eine dieser drei Modellarchitekturen zu entscheiden, wurde als erstes darauf geachtet, dass das Modell die Möglichkeit bietet, Bilder in höherer Auflösung zu verarbeiten. Grund dafür ist, dass im vorliegenden Datensatz viele kleine Objekte abgebildet werden, da die Hafengebäude auch aus größerer Entfernung aufgenommen wurden. Eine der effektivsten Maßnahmen, um die Erkennung von kleinen Objekten zu verbessern, ist es, die Auflösung der Eingabebilder zu erhöhen [vgl. Sola20]. Im Bestfall verarbeitet das Modell die Bilder in der nativen Auflösung des Datensatzes [vgl. Ultr23a]. YOLOv8 unterstützt aktuell eine maximale Bildauflösung von lediglich 640 mal 640 Pixeln. YOLOv7 und YOLOv5 bieten Modellvarianten an, die Eingabebilder von einer Maximalauflösung von 1280 mal 1280 Bildern verarbeiten können. Außerdem unterstützt YOLOv5 die Möglichkeit, nicht nur rechteckige, sondern auch quadratische Bilder zu verarbeiten. Dies ist von Vorteil, da die Bilder im erstellten Datensatz ein Seitenverhältnis von 19:9 und eine Auflösung von 1280 mal 720 Pixeln aufweisen. Das YOLOv5-Modell ist damit in der Lage, die Bilder in ihrer nativen Auflösung zu verarbeiten, sodass weniger Bildinformationen – insbesondere bei kleinen Objekten – verloren gehen. Bei der Verwendung von YOLOv7 müssten die Bilder entweder auf 1280 mal 1280 Pixel gestreckt oder mit schwarzen Balken am oberen und unteren Bildrand aufgefüllt werden. Das Strecken des Bildes hätte den Nachteil, dass auch die Bilder im Produktivbetrieb entsprechend gestreckt werden müssten, damit das Modell die Objekte wiedererkennt [vgl. Nels20a]. Durch die Einführung der schwarzen Balken würde das ursprüngliche Seitenverhältnis zwar erhalten bleiben, in diesem Fall hätte das Modell allerdings weniger Bildinformationen zur Verfügung.

¹ <https://github.com/ultralytics/ultralytics>, zugegriffen am 04.02.2023

² <https://github.com/WongKinYiu/yolov7>, zugegriffen am 04.02.2023

³ <https://github.com/ultralytics/yolov5>, zugegriffen am 04.02.2023

Neben dieser Eigenschaft bietet YOLOv5 unter den drei Modellen die umfangreichsten Möglichkeiten zur Hyperparameteroptimierung an. Dazu kommt, dass YOLOv5 als einziges Modell sogenannte *selbstlernende Anchor-Boxen* verwendet. Selbstlernende Anchor-Boxen ermöglichen es, dass sich das Modell bereits vor dem Training automatisiert auf die Form und Größe der zu detektierenden Objekte einstellen kann. Damit wird sichergestellt, dass auch sehr kleine, sehr große oder irregulär geformte Objekte nicht vom Modell übersehen werden [vgl. Chri22]. Die Verwendung von selbstlernenden Anchor-Boxen stellt eine weitere Maßnahme dar, um die Erkennung von kleinen Objekten zu verbessern [vgl. Sola20]. Da sich die Form und Größe der Kaimauern je nach Blickwinkel und Nähe zur Kamera stark unterscheiden kann, ist davon auszugehen, dass die Verwendung der selbstlernenden Anchor-Boxen auch bei der Erkennung dieser Objekte zu einer höheren Genauigkeit führen wird.

Auf der Basis der untersuchten Eigenschaften wurde sich dafür entschieden, das Modell „YOLOv5“ zu implementieren. Auch wenn dieses im Vergleich zu den neueren Architekturen eine insgesamt leicht niedrigere Vorhersagegenauigkeit aufweist, bietet YOLOv5 einzigartige Eigenschaften, die es für die Anwendung im vorliegenden Anwendungsfall prädestinieren. Zum einen ermöglicht YOLOv5, die Bilder im Datensatz in ihrer nativen Auflösung zu verarbeiten. Durch die Eigenschaften ermöglicht es YOLOv5, dass im Produktivbetrieb keine weiteren Vorverarbeitungsschritte auf die Bilder angewendet werden müssen. Damit lässt sich die Komplexität des gesamten Navigations- und Frühwarnsystems reduzieren und eine echtzeitfähige Erkennung der Dalben und Kaimauern unterstützen. In Kombination mit den selbstlernenden Anchor-Boxen wird dadurch außerdem die Erkennung von kleineren Objekten verbessert. Dazu kommt, dass YOLOv5 die umfangreichsten Möglichkeiten zur Optimierung der Hyperparameter anbietet. Auch wenn YOLOv5 das älteste der drei Modelle ist, werden bis heute immer wieder neue Updates veröffentlicht. Das YOLOv5-Repository wird bis heute kontinuierlich gepflegt.

Die aktuelle Version von YOLOv5 bietet fünf verschiedene Modelluntersvarianten an (siehe Tabelle 1). Die Kürzel „n“ bis „x“ geben die Komplexität der entsprechenden Modelluntersvariante an. Die konkrete Komplexität der Modelle wird durch die Anzahl ihrer Modellparameter in Millionen Stück angegeben. Mit steigender Komplexität nimmt auch der Rechenaufwand (benötigte Gleitkomma-Operationen pro Sekunde (FLOPs)) für das Training und die Inferenzzeit bei der Erkennung von Objekten auf ungesesehenen Bildern zu. Auf der anderen Seite haben die komplexeren Modelle den Vorteil, dass sie detaillierter Bildinformationen verarbeiten können und dadurch eine höhere Vorhersagegenauigkeit erreichen. Alle in der Tabelle aufgelisteten mAP-Werte wurden auf dem Datensatz „COCO val2017“ gemessen [vgl. Joch20]. Für die Erhebung der Inferenzzeit wurde

eine „Nvidia-V100-GPU“ verwendet [vgl. Joch20]. Wie zu erkennen ist, gibt es zwei Modellfamilien, die sich in ihrer maximalen Bildgröße unterscheiden. Die Modelle mit der Endung „6“ sind in der Lage, Bilder von einer Auflösung von bis zu 1280 mal 1280 Pixeln zu verarbeiten.

Tabelle 1: YOLOv5-Modellvarianten nach Jocher [vgl. Joch20]

Modell	Max. Bildgröße	mAP_0.5 (%)	Inferenzzeit (ms/Bild)	Parameter (Mio.)	FLOPs
YOLOv5n	640	45,7	0,6	1,9	4,5
YOLOv5s	640	56,8	0,9	7,2	16,5
YOLOv5m	640	64,1	1,7	21,2	49,0
YOLOv5l	640	67,3	2,7	46,5	109,1
YOLOv5x	640	68,9	4,8	86,7	205,7
YOLOv5n6	1280	54,4	2,1	3,2	4,6
YOLOv5s6	1280	63,7	3,6	12,6	16,8
YOLOv5m6	1280	69,3	6,8	35,7	50,0
YOLOv5l6	1280	71,3	10,5	76,8	111,4
YOLOv5x6	1280	72,7	19,4	140,7	209,8

Die Basisvariante von YOLOv5 unterstützt im Gegensatz dazu eine maximale Bildauflösung von lediglich [640x640]. Um die native Verarbeitung der Bilder im aufgebauten Datensatz zu ermöglichen, kommen in dieser Arbeit nur die Modelluntervarianten in Frage, die eine maximale Auflösung von 1280 mal 1280 Pixeln unterstützen. Da für die Parametrisierung und das Training des Modells nur begrenzte Rechenkapazitäten zur Verfügung stehen, soll eine weniger rechenintensive Modelluntervariante gewählt werden. Diese Entscheidung hat außerdem den Vorteil, dass das generierte Videomaterial im späteren Produktivbetrieb bei Bedarf direkt an Bord verarbeitet werden kann, ohne eine Verbindung zu einem externen leistungsstarken Server herstellen zu müssen. Auf der

Basis dieser Auswahlkriterien wurde sich dafür entschieden, die Modelluntervariante „YOLOv5s6“ zu implementieren.

5.1.6 Training und Parametrisierung

Im Rahmen dieser Arbeit soll auf ein auf dem COCO-Datensatz vortrainiertes YOLO-Modell verwendet werden. Dieses Vorgehen wird auch als *Transfer Learning* bezeichnet und ist besonders für kleine und mittelgroße Datensätze sinnvoll einsetzbar [vgl. LeRO20, S.432]. Da die Gewichte im neuronalen Netzwerk bereits angepasst wurden, muss das Modell nicht jedes Mal von Grund auf neu trainiert werden; stattdessen wird lediglich eine Feinanpassung der Gewichte vorgenommen [vgl. LeRO20, S.422] [vgl. SiAr19, S.5] [vgl. ISZL21, S.3]. Dieses Vorgehen minimiert die Gefahr für eine Überanpassung des Modells und beschleunigt den Trainingsprozess [vgl. LeRO20, S.421 f.].

Das YOLO-Modell besitzt eine Vielzahl von einstellbaren Hyperparametern. Dabei besteht die Möglichkeit, einen Teil dieser Parameter direkt bei der Ausführung des Trainingskripts manuell zu übergeben. Beispiele für diese Parameter sind die Auflösung der Eingabebilder, die Anzahl an trainierten Epochen, die sogenannte *Batchgröße* und der verwendete Optimierungsalgorithmus. Die Batchgröße gibt an, wie viele Trainingsbeispiele durch das Modell verarbeitet werden, bevor die internen Modellparameter durch den Optimierer aktualisiert werden [vgl. Chol18, S.34] [vgl. Glas21, S.405]. Diese vier Modellparameter haben eine große Auswirkung auf die spätere Modellperformance und die Zeit und Rechenleistung, die für das Training benötigt werden.

Neben diesen „primären“ Hyperparametern existiert noch eine Vielzahl an weiteren Hyperparametern, die beim YOLOv5-Modell optimiert werden können. Im Gegensatz zu den primären Hyperparametern können diese „sekundären“ Parameter nicht manuell als Parameter für das Trainingskript übergeben werden, sondern müssen in einer separaten Hyperparameter-Konfigurationsdatei übergeben werden. Beispiele für sekundäre Hyperparameter sind die Lernrate, das Momentum oder der für das Training verwendete IoU-Schwellenwert. Aufgrund der großen Anzahl an sekundären Hyperparametern ist es nicht praktikabel, dort alle Hyperparameter-Kombinationen manuell auszuprobieren. Stattdessen soll dafür ein Hyperparameteroptimierungs-Algorithmus verwendet werden. Wie bereits in Kapitel 2.2.4.3 erläutert, stellen die Grid Search, die Random Search und genetische Algorithmen die drei am häufigsten verwendeten Methoden für die Hyperparameteroptimierung in der Praxis dar. Da die Anzahl an sekundären Hyperparametern in der Regel mehr als ein Dutzend beträgt, ist die Grid Search in diesem Fall nicht geeignet. Im Vergleich zu den genetischen Algorithmen hat die Random Search den Nachteil, dass nur Hyperparameter-Kombinationen in einem Bereich zwischen einer Ober- und einer

Untergrenze ausprobiert werden [vgl. BeBe12, S.283 f.] [vgl. LiLi19, S.9]. Damit besteht hier – genau wie bei der Grid Search – die Gefahr, dass optimale oder nahezu optimale Lösungen nicht im Suchraum liegen und daher übersehen werden. Auch wenn diese Methode im Vergleich zur Random Search zeitaufwendiger ist, kann sich ein genetischer Algorithmus frei im Lösungsraum bewegen; damit steigt die Wahrscheinlichkeit, dass eine optimale Hyperparameter-Kombination gefunden wird [vgl. LiLi19, S.9]. Aus den genannten Gründen soll die Optimierung der sekundären Hyperparameter durch die Verwendung eines genetischen Algorithmus erfolgen.

Abbildung 25 zeigt den Ablauf der Hyperparameteroptimierung auf. Insgesamt sollen dabei vier Experimente durchgeführt werden. Durch die Aufteilung in vier separate Experimente soll erreicht werden, dass die Ursache-Wirkungs-Beziehung zwischen den einzelnen Hyperparametereinstellungen und der Modellperformance klar herausgestellt werden kann.

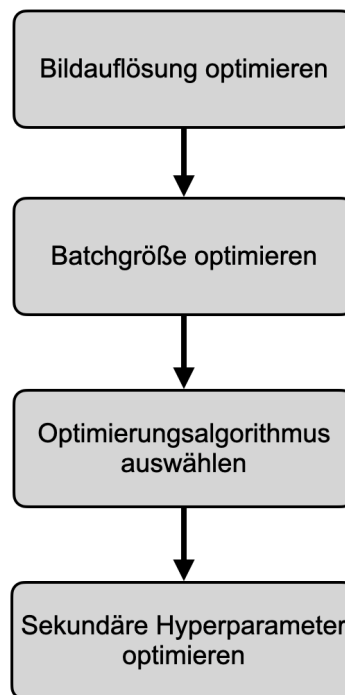


Abbildung 25: Parametrisierung des Modells

Die ersten drei Experimente fokussieren sich auf die Einstellung der drei primären Hyperparameter der Bildgröße, Batchgröße und des verwendeten Optimierungsalgorithmus. Im vierten Schritt erfolgt die Optimierung der sekundären Hyperparameter. Um eine

Vergleichsgrundlage zu schaffen, werden die im Rahmen der vier Experimente parametrisierten Modell-Instanzen für jeweils 100 Epochen trainiert.

Im ersten Experiment soll eine geeignete Auflösung für die Eingabebilder gefunden werden. Dafür sollen die drei Auflösungen [640x640], [1280x1280] und [1280x720] gegenübergestellt werden. [640x640] entspricht der Standardbildauflösung von YOLOv5. Wie zuvor erwähnt, bietet YOLOv5 neben den Standardmodelluntervarianten eine zusätzliche Modellfamilie an, die eine maximale Bildauflösung von [1280x1280] unterstützt. Auch die Verarbeitung von rechteckigen Bildern mit einem Seitenverhältnis von 19:9 wird von YOLOv5 unterstützt. Die Auflösung von [1280x720] entspricht der nativen Auflösung der Bilder im zuvor erstellten Datensatz. Durch die Gegenüberstellung der beiden Auflösungen [640x640] und [1280x1280] soll evaluiert werden, ob eine Erhöhung der Bildauflösung tatsächlich zu einer Performancesteigerung – insbesondere bei kleineren Objekten – führt. Durch die Untersuchung der Auflösung [1280x720] soll evaluiert werden, inwiefern sich die Verwendung der nativen Bildauflösung auf die Leistungsfähigkeit des Modells und den Trainingsprozess auswirkt.

Im zweiten Experiment soll der Einfluss unterschiedlicher Batchgrößen auf die benötigten Trainingsressourcen und die Modellperformance untersucht werden. Im Allgemeinen wird empfohlen die Batchgröße so hoch wie möglich zu anzusetzen [vgl. Ultr23a]. Im Rahmen dieses Experimentes soll begonnen werden, ein Modell mit der Batchgröße 16 zu trainieren. Anschließend soll der Wert – so wie in der Praxis üblich – fortlaufend verdoppelt werden (32, 64, 128 etc.). Dieser Vorgang soll so lange wiederholt werden, wie dies von der Hardware unterstützt wird.

Im dritten Experiment erfolgt die Auswahl eines passenden Optimierungsalgorithmus. Dabei sollen alle Optimierer untersucht werden, die vom YOLOv5-Modell standardmäßig als Auswahlmöglichkeiten zur Verfügung gestellt werden. YOLOv5 bietet die Möglichkeit, zwischen den drei Optimierungsalgorithmen „SGD mit Nesterov-Momentum“, „Adam“ und „AdamW“ zu wählen.

Im letzten Experiment sollen die sekundären Hyperparameter unter Zuhilfenahme des in YOLOv5 integrierten genetischen Algorithmus optimiert werden. Anschließend soll untersucht werden, inwiefern dieses Vorgehen dazu beitragen kann, die Modellperformance zu verbessern.

Für das Training des finalen Modells werden die Parameter verwendet, die im Rahmen der Hyperparameteroptimierung als optimal ermittelt werden konnten. In allen vorangegangenen Experimenten wurden die Modelle für jeweils 100 Epochen trainiert. Dieses Limit soll hier aufgehoben werden, um die optimale Epochenanzahl für das Training des

finalen Modells zu verwenden. Eine zu geringe Epochenanzahl kann dazu führen, dass das Modell nicht ausreichend an die Trainingsdaten angepasst wird und damit Performancepotenzial verloren geht; auf der anderen Seite führt ein zu langes Training zu einer Überanpassung des Modells [vgl. IRYM22, S.52820]. Für die Ermittlung der optimalen Epochenanzahl soll ein sogenanntes *Early Stopping* implementiert werden. Beim Early Stopping wird das Training vorzeitig abgebrochen, sobald für eine gewisse Anzahl an Epochen kein Performancezuwachs mehr gemessen werden kann [vgl. Chol18, S.249]. Um ein Overfitting auszuschließen, ist parallel darauf zu achten, dass der Validierungsverlust konstant niedrig bleibt und in den letzten Epochen nicht wieder ansteigt [vgl. Glas21, S.197 f.].

5.1.7 Performance-Evaluation

Im Rahmen der durchzuführenden Experimente zur Hyperparameteroptimierung soll der Einfluss von unterschiedlichen Konfigurationen auf die Vorhersagegenauigkeit und Geschwindigkeit des zu implementierenden Modells untersucht werden. Für die Evaluation der Vorhersagegenauigkeit werden die beiden Performance-Metriken mAP_0.5 und mAP_0.5:0.95 genutzt. Die Geschwindigkeit der Modelle wird über die beiden Metriken Inferenzzeit und FPS evaluiert. Neben diesen primären Performance-Metriken sollen bei Bedarf auch weitere Kennzahlen wie PR- und Verlustkurven herangezogen werden, um die Ergebnisse zu interpretieren. Zusätzlich zur Modellperformance soll auch der Einfluss von unterschiedlichen Hyperparametern auf die benötigten Trainingsressourcen untersucht werden. Als Performance-Kennzahlen sollen dafür sowohl die Trainingsdauer als auch der benötigte Speicherbedarf (System-RAM und Grafikkartenspeicher) erhoben und ausgewertet werden.

Im Rahmen der Performance-Evaluation soll außerdem untersucht werden, ob eine Übertragbarkeit der beobachteten Leistungskennzahlen des finalen Modells auf andere Anwendungsfälle gegeben ist. Da kein geeigneter Datensatz zur Abbildung von Kaimauern und Dalben aus anderen Hafengebieten existiert, muss an dieser Stelle eine eigene Sammlung aus passenden Internetbildern zusammengestellt werden. Diese sollen anschließend vom trainierten Modell verarbeitet werden, um eigene Vorhersagen zu tätigen. Danach lässt sich beurteilen, inwiefern das Modell in der Lage ist, die fremden Hafengebiete zu erkennen.

6 Implementierung

In diesem Kapitel werden die Tools, Programme und Hardware beschrieben, die für die Aufbereitung der Rohdaten, das Labeling, die Aufbereitung des Datensatzes und das Training sowie die Auswertung der Modellperformance verwendet wurden.

6.1 Datenvorverarbeitung

Im Rahmen der Datenvorverarbeitung wurden die drei separaten Kamera-Streams in einem ersten Schritt zunächst mithilfe von Bildschirmaufnahmen aus dem Rohmaterial extrahiert. Anschließend wurden unbrauchbare Video-Passagen aus den extrahierten Videos entfernt. Nach diesem Filterungsschritt reduzierte sich die Menge des nutzbaren Videomaterials von ca. zwei auf eineinhalb Stunden. Für beide Arbeitsschritte wurde das Programm *QuickTime Player* verwendet.

In einem weiteren Schritt wurden die vorliegenden Videos in unterschiedlich hohen Auflösungen abgespeichert, um herauszufinden, welche Auflösung sich für das Labeling der Bilder am besten eignet. Zu geringe Auflösungen führten dazu, dass die Objekte auf den Bildern nicht mehr ausreichend gut zu erkennen waren, was die Qualität der Label verringert hätte. Auf der anderen Seite führte eine Erhöhung der Bildauflösung ab einem gewissen Zeitpunkt nicht mehr dazu, dass die Objekte auf den Bildern besser zu erkennen waren. Auf der Grundlage dieser Überlegungen wurde sich schließlich dafür entschieden, die Videos in einer Auflösung von [1280x720] abzuspeichern. Für die Konvertierung der Videos wurde das Programm *VideoProc Converter* verwendet.

Im nächsten Verarbeitungsschritt wurden die Videos in ihre einzelnen Frames aufgeteilt, um diese für das Labeling des Datensatzes nutzen zu können. Die Extraktion der Frames wurde unter Zuhilfenahme des Python-Moduls *math* und der Computer-Vision-Bibliothek *OpenCV* implementiert. Die Aufteilung erfolgte so, dass pro Sekunde im Video ein Frame extrahiert wurde. Durch dieses Vorgehen konnten aus den insgesamt ca. 4,5 Stunden Videomaterial (1,5 Stunden pro Video-Stream) insgesamt 16.228 Frames gewonnen werden.

6.2 Annotation der Bilder

Für die Annotation der vorliegenden Bilder wurde eine lokale Instanz des Object-Detection-fähigen Labeling-Tools CVAT⁴ verwendet. Das Hosting von CVAT erfolgte mithilfe der Containersoftware *Docker*. Grund für die Nutzung einer lokalen CVAT-Instanz war, dass hier im Gegensatz zur Online-Version alle Funktionen unbegrenzt kostenlos nutzbar sind. Außerdem blieb durch die Nutzung der lokalen Instanz die Kontrolle über die Daten erhalten und es entstand keine Abhängigkeit von einem externen Anbieter.

Im nächsten Schritt wurden die vorliegenden Videoframes unter der Verwendung von OpenCV in Pakete von jeweils 250 Bildern unterteilt. Die Auswahl der Frames in jedem Paket erfolgte zufällig, um zu verhindern, dass zu viele nahezu identische Bilder in den Datensatz gelangen. Für die Implementierung wurde das Python-Modul *random* verwendet. Anschließend wurden die Bilder aus den Paketen als einzelne Tasks in CVAT importiert. Insgesamt wurden so 30 Tasks mit jeweils 250 Bildern angelegt.

Die Annotation der Dalben und Kaimauern erfolgte unter der Berücksichtigung der gängigen Best-Practice-Vorgaben für das Labeling von Object-Detection-Datensätzen [vgl. Nels20b]. Es wurde darauf geachtet, die Objekte korrekt und konsistent zu annotieren. Um die Konsistenz zu wahren und False Positives zu vermeiden, wurden außerdem alle sichtbaren Kaimauern und Dalben auf den Bildern mit Bounding Boxen versehen. Ziel war es, die Bounding Boxen weder zu eng noch zu weit anzulegen, um die zu labelenden Objekte nicht abzuschneiden und gleichzeitig möglichst wenig Hintergrund-Pixel zu annotieren. Auch wenn die Kaimauern und Dalben von anderen Objekten überdeckt waren, wurden diese so gelabelt, als wären sie vollständig sichtbar. Insbesondere bei den Kaimauern wurde weiterhin darauf geachtet, dass die Bounding Boxen nicht über den Bildrand hinausragten.

6.3 Aufbereitung des Datensatzes

Die Aufbereitung des Datensatzes wurde mithilfe des Online-Tools *roboflow*⁵ implementiert. Roboflow wurde dafür verwendet, die über den Bildrand hinausragenden Bounding Boxen automatisiert zu stutzen und den Train-Valid-Test-Split einzustellen. Außerdem wurde das Tool dafür genutzt, um die Bildauflösung für die im Rahmen der Parametrisierung durchgeführten Experimente anzupassen und den Datensatz im Format „YOLO

⁴ <https://www.cvat.ai/>, zugegriffen am 17.12.2022

⁵ <https://roboflow.com/>, zugegriffen am 09.01.2023

v5 PyTorch“ zu exportieren. Die Formatierung des Datensatzes war notwendig, da das YOLOv5-Modell nur dieses Dateiformat unterstützt.

Tabelle 2: Einstellungen der Datenaugmentierung

Augmentierung	Maximale Augmentierungs-Stärke
Rotation	Horizontal: -5 %, Vertikal: +5 %
Scherung	Horizontal: -5 %, Vertikal +5 %
Helligkeit	60 %
Bildrauschen	10 %
Bildunschärfe	2 Pixel

Roboflow bietet die Möglichkeit, eine Datenaugmentierung zu nutzen, um den Umfang des Datensatzes künstlich zu erweitern. Diese künstliche Erweiterung wird dadurch realisiert, dass für die Bilder im Trainingsset künstliche Kopien erstellt und diese anschließend dem Trainingsset hinzugefügt werden. Es werden dabei immer alle ausgewählten Transformationen simultan auf jedes augmentierte Bild angewendet. Die so entstehenden Kopien unterscheiden sich lediglich darin, wie stark die Transformationen jeweils ausgeprägt sind. Tabelle 2 zeigt, welche Transformationen implementiert und welche maximalen Augmentierungs-Stärken dabei jeweils ausgewählt wurden.

6.4 Visualisierung des Datensatzes

Für die Visualisierung der Zusammensetzung des Datensatzes wurde Python verwendet. Für die Implementierung der erstellten Balkendiagramme wurde die *matplotlib*-Bibliothek verwendet. Um die Anzahl an Bildern, Annotationen und Null-Beispielen zu ermitteln, wurden die JSON-Dateien des Trainings-, Validierungs- und Testsets unter Zuhilfenahme des Python-Paketes *json* ausgelesen.

6.5 Training des Modells

Für das Training und die Parametrisierung des Modells wurde *Google Colab*⁶ verwendet. Google Colab ist eine Cloud-Anwendung, die es ermöglicht, Python-Code in Jupyter-Notebooks im Webbrowser auszuführen. Durch den Service lassen sich außerdem spezielle Grafikkarten nutzen, die für das Training von ML-Modellen optimiert sind.

Für das Training des Modells wurde eine Grafikkarte des Typs „NVIDIA A100-SXM4-40GB“ mit 40 GB VRAM in Verbindung mit der CUDA-Version 12.0 verwendet. Das gesamte System verfügte über 83,5 GB System-RAM und einen Laufwerkspeicher von insgesamt 166,8 GB. Bei der Konfiguration des Laufzeittyps des Jupyter-Notebooks wurde die GPU-Hardwarebeschleunigung aktiviert. Weiterhin wurden die Einstellungen „GPU-Klasse: Premium“ und „Laufzeitkonfiguration: Erweiterter RAM“ gewählt, um den Trainingsprozess zu beschleunigen.

6.6 Erhebung der Modellperformance

Die Erhebung der Performance-Kennzahlen erfolgte mithilfe des in *TensorFlow* integrierten Visualisierungstools *TensorBoard* und der Webanwendung *Weights & Biases*⁷. Für das Training der Modelle wurde die API von *Weights & Biases* verwendet. Mit dieser Software ließen sich die Ergebnisse der Experimente und des finalen Modells kontinuierlich aufzeichnen und miteinander vergleichen.

⁶ <https://colab.research.google.com/>, zugegriffen am 15.03.2023

⁷ <https://wandb.ai/site>, zugegriffen am 06.04.2023

7 Evaluation

In diesem Kapitel wird zunächst der Aufbau der Rohdaten beschrieben. Anschließend erfolgt die Beschreibung des (aufbereiteten) Datensatzes. Danach werden die Untersuchungsergebnisse der vier Experimente zum Zwecke der Hyperparameteroptimierung beschrieben und diskutiert. In einem letzten Schritt wird die Performance des finalen Modells untersucht. Dabei wird auch die Erfüllung der an das Modell erhobenen Anforderungen und die Übertragbarkeit der Ergebnisse überprüft.

7.1 Datenbestand

Als Grundlage für den Datensatz wurden Kamerabilder aus dem Emdener Hafen verwendet. Die Videos wurden von Mitarbeitern des *Deutschen Zentrums für Luft- und Raumfahrt* (DLR) im Hafenbecken aufgenommen und stammen von Juni 2022. Insgesamt liegen etwa zwei Stunden Videomaterial vor. In Abbildung 26 ist zu erkennen, wie das Rohmaterial der Aufnahmen aufgebaut ist. Jede Aufnahme setzt sich aus drei individuellen Video-Streams zusammen. Die drei Kameras vom Typ „Blackmagic Micro Studio 4K“ wurden am Dach des Bootes befestigt. Kamera eins (oben) filmt in Fahrtrichtung und Kamera zwei (unten links) bzw. drei (unten rechts) nach Backbord.



Abbildung 26: Aufnahme aus dem Emdener Hafen

7.2 Datensatz

In diesem Abschnitt wird der Aufbau des zugrundeliegenden Datensatzes erläutert und visualisiert. Dabei wird sowohl auf die Struktur des unverarbeiteten als auch die des verarbeiteten Datensatzes nach der Anwendung des Train-Valid-Test-Splits und der Anwendung der Datenaugmentierung näher beschrieben.

7.2.1 Grundlegender Aufbau

Der gelabelte Datensatz besteht aus 6.811 Bildern mit insgesamt 20.110 Annotationen. Mit durchschnittlich drei Bounding Boxen pro Bild weist er eine hohe Merkmalsdichte auf. Insgesamt enthalten 6.081 Bilder mindestens ein Label (siehe Abbildung 27). Im Datensatz sind 730 Bilder enthalten, die keine Annotationen aufweisen (Null-Beispiele).

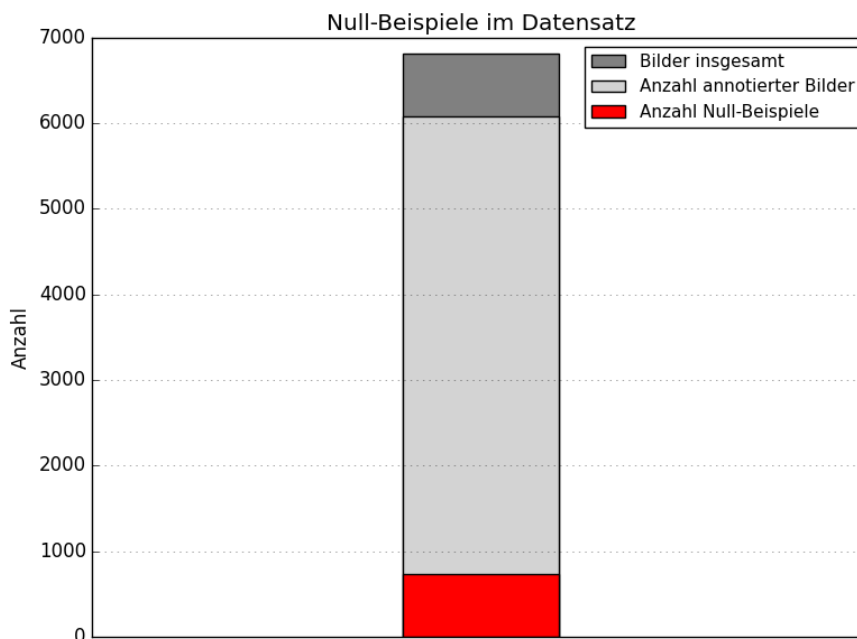


Abbildung 27: Anzahl an (annotierten) Bildern und Null-Beispielen

Der Datensatz enthält die beiden Zielklassen „Dalbe“ und „Kaimauer“. Insgesamt wurden 6.270 Kaimauern und 13.840 Dalben annotiert. Jedes gelabelte Bild bildet im Durchschnitt 1,02 Kaimauern und 2,28 Dalben ab.

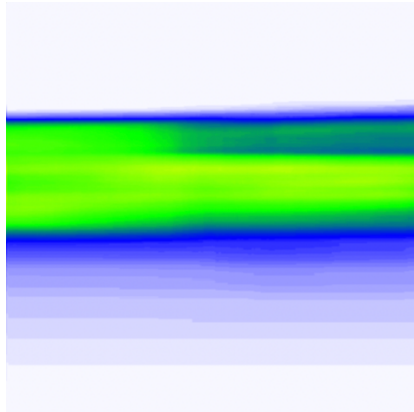


Abbildung 28: Kaimauer-Heatmap

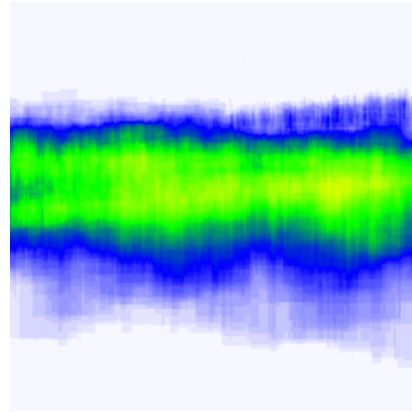


Abbildung 29: Dalben-Heatmap

Auf den Heatmaps in Abbildung 28 und Abbildung 29 ist die Verteilung der beiden Klassen auf den Bildern visualisiert. Die grünen Bereiche geben die Stellen an, an denen die Kaimauern und Dalben jeweils sehr häufig vorkommen. Es ist zu erkennen, dass sich die Proportionen der beiden Objektklassen signifikant unterscheiden. Kaimauer-Objekte bedecken im Durchschnitt einen größeren Teil des Bildes und sind in der Regel so breit, dass sie an den Seiten vom Bildrand abgeschnitten werden. Dalben besitzen deutlich kleinere Abmessungen und sind im Gegensatz zu den Kaimauern meist höher, als sie breit sind.

7.2.2 Aufbereiteter Datensatz

Abbildung 30 visualisiert den Aufbau des Datensatzes nach der Anwendung der im Konzept beschriebenen Maßnahmen zur Vorverarbeitung und Augmentierung. Für die Aufteilung der ursprünglichen 6.811 Bilder wurde ein Train-Valid-Test-Split von 60:20:20 verwendet, so wie er in der Praxis empfohlen wird [vgl. Glas21, S.187]. Nach der Anwendung des Splits befinden sich 4.087 Bilder im Trainingsset und 1.357 im Validierungsset. Das Testset besteht aus 1.362 Bildern.

Nach der Anwendung der Datenaugmentierung besteht der gesamte Datensatz aus insgesamt 10.240 Bildern und enthält 30.315 Annotationen. Das Trainingsset besteht aus 7.514 Bildern mit insgesamt 22.238 Annotationen. Durch die Anwendung der Datenaugmentierung (siehe Kapitel 6.3) wurde das Trainingsset damit um 3.427 Bilder erweitert. Das Validierungsset besteht weiterhin aus 1.357 Bildern und 3.880 Annotationen. Das Testset wird durch die Datenaugmentierung ebenfalls nicht erweitert, sodass es weiterhin aus 1.362 Bildern und 4.136 Annotationen besteht.

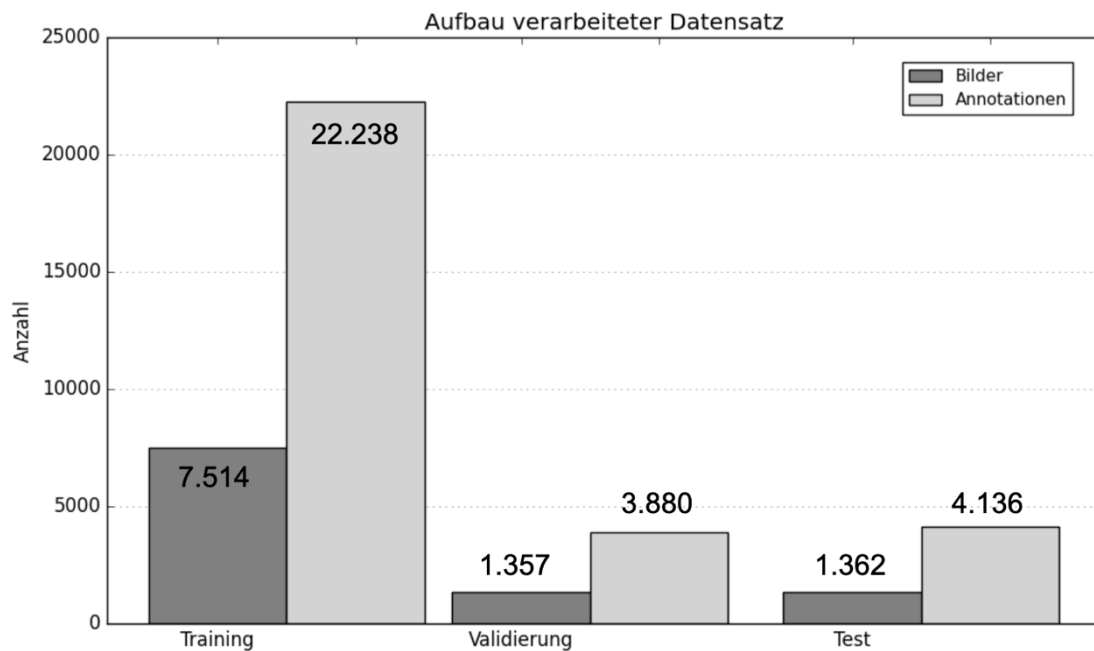


Abbildung 30: Aufbau des verarbeiteten Datensatzes

Der verarbeitete Datensatz enthält insgesamt 20.877 Dalben- und 9.438 Kaimauer-Annotationen. Damit besteht ein als leicht zu bewertendes Klassenungleichgewicht. Wie im Konzept beschrieben, sollen erst Maßnahmen ergriffen werden, wenn das Klassenungleichgewicht stark ausfällt (Verhältnis 1:10), da die Ausgleichstechniken eine Überanpassung des Modells begünstigen können [vgl. Volp19]. Im Rahmen der Vorverarbeitung wurde ein Teil der Null-Beispiele im Datensatz belassen, um die Robustheit des Modells zu erhöhen. Dadurch, dass durch die Augmentierung von den ursprünglich 730 Null-Beispielen ebenfalls künstliche Kopien erzeugt wurden, befinden sich nun 1.095 Null-Beispiele im Datensatz. Damit wird die im Konzept definierte Grenze nicht überschritten ($1.095 \text{ Null-Beispiele} / 10.240 \text{ Bilder} = 10,69 \%$) und es sind keine weiteren Maßnahmen zur Entfernung überflüssiger Null-Beispiele zu ergreifen.

7.3 Hyperparameteroptimierung

Es wurden insgesamt vier Experimente durchgeführt, um die Auswirkung von unterschiedlichen Hyperparametern auf die Modellperformance und die Trainingsressourcen zu untersuchen. Diese Untersuchungsergebnisse können anschließend dafür genutzt werden, um die optimale Hyperparameter-Kombination für das YOLOv5-Modell zu ermitteln. Im Rahmen dieses Kapitels werden die Evaluationsergebnisse der vier Experimente jeweils zunächst beschrieben und anschließend diskutiert.

7.3.1 Experiment 1: Optimierung der Bildauflösung

Das Ziel des ersten Experimentes ist es, die Auswirkung der drei Bildauflösungen [640x640], [1280x1280] und [1280x720] auf die Modellperformance und die benötigten Trainingsressourcen zu untersuchen. Auf der Basis der Untersuchungsergebnisse soll anschließend eine begründete Entscheidung bei der Auswahl einer geeigneten Bildauflösung getroffen werden.

Es ist zu berücksichtigen, dass bei den beiden Auflösungen [640x640] und [1280x1280] eine Vorverarbeitung der Bilder vorgenommen wurde. Anstatt die Bilder zu strecken, wurde das ursprüngliche Seitenverhältnis von 16:9 beibehalten und das Bild mit schwarzen Balken aufgefüllt. Dieses Vorgehen wird empfohlen, damit das Modell die Objekte im späteren Produktivbetrieb besser wiedererkennen kann [vgl. Nels20a].

Alle drei der im Rahmen dieses Experimentes evaluierten Modelle wurden für 100 Epochen trainiert. Als Optimierer kam der SGD-Algorithmus mit Nesterov-Momentum zum Einsatz und die Batchgröße betrug 32. Für die Einstellung der sekundären Hyperparameter wurde die voreingestellte Standardkonfiguration verwendet.

7.3.1.1 Ergebnisse

Tabelle 3 zeigt die Vorhersagegenauigkeit der drei trainierten Modelle auf. Es ist zu erkennen, dass die mAP_0.5-Metrik ansteigt, wenn die Auflösung der Bilder von [640x640] auf [1280x1280] erhöht wird. Die mAP_0.5 steigt in diesem Fall um 2,2 Prozentpunkte an. Es fällt auf, dass das Modell mit der Auflösung von [640x640] Probleme damit hat, die kleineren Dalben-Objekte zuverlässig zu lokalisieren. Hier liegt die mAP_0.5 um etwa vier Prozentpunkte niedriger als bei dem Modell mit der Bildauflösung von [1280x1280]. Bei der Kaimauer-Klasse liegt dieser Performanceunterschied lediglich bei 0,4 Prozent.

Tabelle 3: Experiment 1: Mean Average Precision (0.5)

Auflösung	mAP_0.5 (%)	mAP_0.5 (Dalbe) (%)	mAP_0.5 (Kaimauer) (%)
[640x640]	95,9	93,7	98,2
[1280x1280]	98,1	97,7	98,6
[1280x720]	97,6	97,1	98,1

Wird die native Auflösung der Bilder im Datensatz verwendet, fällt die Genauigkeit der Vorhersagen leicht ab. Bei einer Bildauflösung von [1280x720] liegt die mAP_0.5 um 0,5 Prozentpunkte niedriger als beim Modell mit der Bildauflösung von [1280x1280]. Bei der Erkennung der Dalben und Kaimauern ist ebenfalls ein leichter Performanceunterschied von 0,6 bzw. 0,5 Prozentpunkten feststellbar.

Tabelle 4 zeigt die erhobenen Mean-Average-Precision-Werte für den IoU-Intervall von 0,5 bis 0,95 auf. Ähnlich wie bei der mAP_0.5 lässt sich durch die Erhöhung der Bildauflösung eine Steigerung der Vorhersagegenauigkeit feststellen. Mit der Verdopplung der Bildauflösung steigt die mAP_0.5:0.95 von 64,8 auf 69,9 Prozent; das entspricht einer Zunahme von 5,1 Prozentpunkten. Genau wie bei der mAP_0.5 ist zu erkennen, dass durch die Erhöhung der Bildauflösung vor allem die Fähigkeit zur Lokalisierung der Dalben verbessert werden kann (+ 7,9 Prozentpunkte). Während bei der mAP_0.5 lediglich für die Dalben-Klasse eine signifikanter Performancesteigerung erkennbar war, trifft dies hier auch für die Kaimauer-Klasse zu (+ 2,0 Prozentpunkte).

Tabelle 4: Experiment 1: Mean Average Precision (0.5:0.95)

Auflösung	mAP_0.5:0.95 (%)	mAP_0.5:0.95 (Dalbe) (%)	mAP_0.5:0.95 (Kaimauer) (%)
[640x640]	64,8	57,2	72,4
[1280x1280]	69,9	65,1	74,4
[1280x720]	66,8	64,0	69,9

Auch der Performanceabfall des Modells mit der Bildauflösung von [1280x720] tritt hier wieder auf. Die mAP_0.5:0.95 fällt um 3,1 Prozentpunkte ab, wenn die Bildauflösung von [1280x1280] auf [1280x720] verringert wird. Auffällig ist, dass es dem Modell mit der nativen Bildauflösung schwerfällt, präzise Vorhersagen bei der Erkennung der Kaimauern zu treffen. Die Differenz zum Modell mit der Bildauflösung von [1280x1280] liegt bei 4,5 Prozentpunkten. Damit erreicht das Modell mit der nativen Bildauflösung einen um 2,5 Prozentpunkte niedrigeren Klassen-mAP-Wert als das Modell mit der niedrigsten Auflösung.

Tabelle 5 zeigt den Unterschied in der Trainingsperformance der drei Modelle auf. Es ist zu erkennen, dass das Training des Modells mit der Bildauflösung [1280x1280] mit 2,73 Stunden fast dreimal so lange dauerte, wie beim Modell mit der Auflösung [640x640]. Der Bedarf an System-RAM ist hier etwa dreimal so hoch und es wird fast viermal so viel Grafikkartenspeicher benötigt. Außerdem wird viermal so viel Speicher angefragt, um die Bilder aus dem Trainings- und Validierungsset zwischenspeichern.

Tabelle 5: Experiment 1: Trainingsressourcen

Auflösung	Dauer (Std.)	GPU-RAM (GB)	System-RAM (GB)	Cache (GB)
[640x640]	1,087	8,4	20,1	10,2
[1280x1280]	2,735	31,8	64,8	40,6
[1280x720]	1,825	19,1	38,3	22,8

Durch die Verwendung einer Bildauflösung von [1280x720] lässt sich der Bedarf an GPU- und System-RAM um ca. 40 Prozent reduzieren. Außerdem wird dadurch nur die Hälfte an Kapazität für das Zwischenspeichern der Bilder benötigt, als dies bei der Auflösung von [1280x1280] der Fall ist. Durch die geringere Bildauflösung kann die Trainingsdauer von 2,7 auf 1,8 Stunden verringert werden.

7.3.1.2 Diskussion

Die schlechte Vorhersagegenauigkeit des Modells mit der niedrigsten Bildauflösung lässt sich dadurch begründen, dass dieses Modell im Vergleich zu den anderen beiden Modellen die wenigsten Bildinformationen zur Verfügung gestellt bekommt. Die niedrigere Bildauflösung hat auch zur Folge, dass die abgebildeten Hafengebäude kleiner sind. Wie bereits erläutert, hat die YOLO-Modellfamilie Probleme damit, insbesondere kleine Objekte zuverlässig zu lokalisieren [vgl. ZCSG23, S.4] [vgl. RDGF15, S.784]. Dieser Umstand lässt sich auch in den erhobenen Performance-Metriken erkennen. Alle drei Modelle erkennen die Kaimauern besser als die Dalben. Beim Modell mit der niedrigsten Auflösung ist dieses Problem durch die Verkleinerung der ohnehin kleineren Dalben-Objekte deutlich stärker ausgeprägt.

Es konnte auch festgestellt werden, dass das Modell mit der nativen Bildauflösung niedrigere mAP-Werte erreicht als jenes Modell, das eine Bildauflösung von [1280x1280] verwendet. Dieser Performanceunterschied kann nicht darin begründet liegen, dass das Modell weniger detaillierte Bildinformationen zur Verfügung gestellt bekommt. Der einzige Unterschied der beiden Modelle besteht lediglich darin, dass die Bilder mit der Auflösung von [1280x1280] am oberen und unteren Bildrand mit schwarzen Balken aufgefüllt wurden; die Anzahl an nutzbaren Pixeln ist in beiden Fällen gleich ($1280 \times 720 = 921.600$). Da sich die Trainingseinstellungen und Hyperparameter der drei Modelle ansonsten nicht unterscheiden, kann die Performancedifferenz lediglich auf die interne Umsetzung des Trainingsargumentes „rect“ zur Verarbeitung von rechteckigen Bildern zurückzuführen sein.

Der Unterschied in den benötigten Trainingsressourcen liegt in der Anzahl der zu verarbeitenden Pixel begründet. Bei einer Bildauflösung von [1280x1280] müssen entsprechend auch viermal so viele Pixel im Cache vorgehalten werden, als dies bei einer Auflösung von [640x640] der Fall ist. Das eigentliche Training der Modelle findet auf der Grafikkarte statt. Da die GPU in allen drei Fällen die gleiche Anzahl an Rechenoperationen (FLOPs) durchführen kann, nimmt die Trainingsdauer zu, wenn mehr Pixel verarbeitet werden müssen.

Auf der Basis dieser Untersuchungsergebnisse wurde sich dafür entschieden, dass in den nachfolgenden Experimenten fortan eine Bildauflösung von [1280x720] verwendet werden soll. Durch diese Parametereinstellung lässt sich ein günstiges Kosten-Nutzen-Verhältnis erreichen. Auf der einen Seite ist das Modell im Vergleich zu dem Modell mit der Bildauflösung von [640x640] in der Lage, eine signifikant höhere Vorhersagegenauigkeit – insbesondere bei der Erkennung der Dalben-Objekte – zu erreichen. Auf der anderen Seite können die für das Training benötigten Ressourcen im Vergleich zum Modell mit der Bildauflösung [1280x720] um bis zu 50 Prozent reduziert werden. Ein weiteres Vorteil ist, dass die im Produktivbetrieb anfallenden Kamerabilder in diesem Fall keinem zusätzlichen Vorverarbeitungsschritt unterzogen werden müssen, wie dies bei den anderen beiden Modellen der Fall wäre. Dadurch lässt sich die Reaktionsfähigkeit des Modells verbessern.

7.3.2 Experiment 2: Optimierung der Batchgröße

Im Rahmen dieses Experiment soll die Auswirkung von unterschiedlichen Batchgrößen auf die Modell- und Trainingsperformance untersucht werden. Die maximal durch die vorliegende Hardware unterstützte Batchgröße betrug 64, sodass hier die drei Parameterausprägungen 16, 32 und 64 evaluiert werden sollen.

7.3.2.1 Ergebnisse

Tabelle 6 bildet die Mean-Average-Precision-Werte für die drei trainierten Modelle ab. Es ist zu erkennen, dass alle Performance-Metriken mit einer steigenden Batchgröße ebenfalls zunehmen. Insbesondere bei der mAP_0.5:0.95 ist ein Performanceunterschied erkennbar. Das Modell mit der höchsten Batchgröße erreichte eine um etwa 1,2 Prozentpunkte höhere Vorhersagegenauigkeit als das Modell mit der niedrigsten Batchgröße. Ein Blick auf die beiden Klassen-mAP-Werte zeigt, dass die Vorhersagegenauigkeit bei den Kaimauern um 1,6 Prozentpunkte höher liegt, wenn die Batchgröße von 16 auf 64 erhöht wird. Die Fähigkeit zur Erkennung der Dalben wird durch diese Erhöhung lediglich um 0,8 Prozentpunkte verbessert. Die Differenz in der mAP_0.5 liegt bei ca. 0,4 Prozent und kann damit als nicht signifikant bewertet werden.

Tabelle 6: Experiment 2: Vorhersagegenauigkeit

Batchgröße	mAP_0.5 (%)	mAP_0.5:0.95 (%)	mAP_0.5:0.95 (Dalbe) (%)	mAP_0.5:0.95 (Kaimauer) (%)
16	97,2	66,2	63,3	69,1
32	97,6	66,8	64,0	69,9
64	97,6	67,4	64,1	70,7

Bei der Auswertung der Trainingsperformance lässt sich ein klarer Zusammenhang zwischen der Batchgröße und den benötigten Trainingsressourcen erkennen (siehe Tabelle 7).

Tabelle 7: Experiment 2: Trainingsressourcen

Batchgröße	Dauer (Std.)	GPU-RAM (GB)	System-RAM (GB)
16	2,063	9,8	33,6
32	1,825	19,1	38,3
64	1,725	36,2	47,3

Mit einer steigenden Batchgröße steigt auch der benötigte System- und GPU-RAM an. Dieser Unterschied wird vor allem beim verwendeten Grafikkartenspeicher sichtbar: Das Modell mit der höchsten Batchgröße benötigt mehr als dreieinhalb mal so viel GPU-RAM wie das Modell mit der niedrigsten Batchgröße. Es ist auch zu erkennen, dass die Trainingsdauer mit einer größeren Batchgröße signifikant abnimmt. Im Vergleich zur Batchgröße 16 kann die Trainingsdauer durch die Verwendung einer Batchgröße von 64 um ca. 20 Minuten verringert werden.

7.3.2.2 Diskussion

Auf der Basis dieser Untersuchungsergebnisse wurde sich dafür entschieden, in den nachfolgenden Experimenten fortan eine Batchgröße von 64 für das Training der Modelle zu verwenden. Diese Parametereinstellung hat zwar den Nachteil, dass mehr System- und GPU-RAM benötigt wird, dafür kann jedoch auch die Vorhersagegenauigkeit bei der Erkennung der Kaimauern und Dalben im IoU-Intervall von 0,5 bis 0,95 verbessert werden. Außerdem lässt sich durch die Verwendung der maximal unterstützten Batchgröße die Trainingsdauer reduzieren.

7.3.3 Experiment 3: Auswahl des Optimierungsalgorithmus

Das YOLOv5-Modell bietet die Möglichkeit zwischen den drei Optimierungsalgorithmen „Stochastic Gradient Descent mit Nesterov-Momentum“, „Adam“ und „AdamW“ zu wählen. Im Rahmen dieses Experimentes soll die Auswirkung dieser drei Optimierungsalgorithmen auf die Modellperformance untersucht werden. Auf der Basis der Untersuchungsergebnisse soll anschließend eine begründete Entscheidung bei der Auswahl eines geeigneten Optimierers getroffen werden.

7.3.3.1 Ergebnisse

Die beiden Abbildung 31 und Abbildung 32 visualisieren die Fehlerkurven für den Box-Verlust der drei Modelle. Der Box-Verlust gibt an, wie gut das Modell in der Lage ist, Objekte zu lokalisieren. Für die Lokalisation der Bounding Boxen nutzt YOLOv5 eine Verlustfunktion namens *Complete Intersect over Union* (CIoU). Die CIoU-Verlustfunktion berücksichtigt drei Aspekte: die Überschneidung der vom Modell vorgeschlagenen Bounding Boxen (B) und der Grundwahrheits-Bounding-Boxen (B_{gt}) S , die Distanz der Mittelpunkte dieser Boxen D und die Konsistenz der Seitenverhältnisse zwischen B und B_{gt} V [vgl. Khan21] [vgl. GLZJ21, S.317 f.]. Diese drei Komponenten werden aufaddiert, um den Box-Verlust zu berechnen (siehe Formel 6).

$$L = S(B, B_{gt}) + D(B, B_{gt}) + V(B, B_{gt})$$

Formel 6: CloU-Verlustfunktion nach Khandelwal [vgl. Khan21]

Abbildung 31 gibt Auskunft darüber, wie gut das Modell in der Lage ist, sich im Laufe der 100 Epochen an die vorliegenden Trainingsbilder anzupassen; Abbildung 32 zeigt die Anpassungsfähigkeit der Modelle an bisher ungesehene Bilder [vgl. Glas21, S.196 f.].

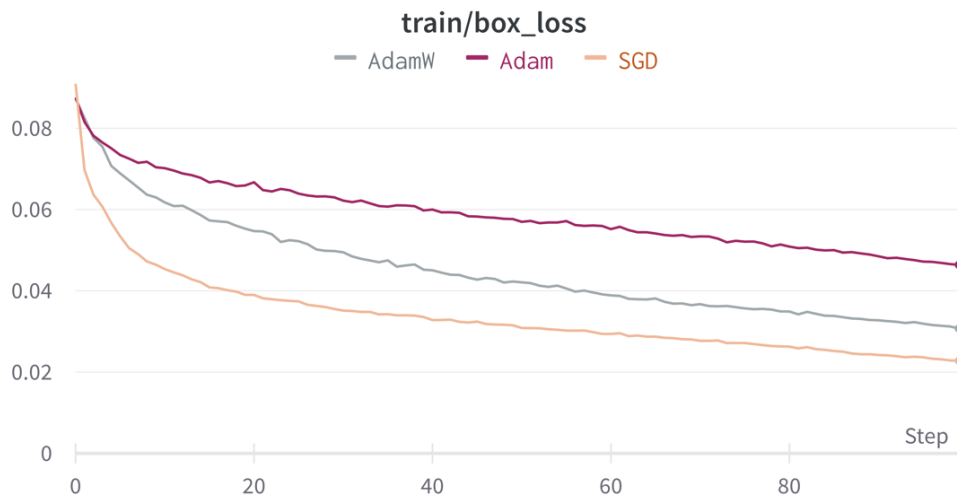


Abbildung 31: Experiment 3: Box-Verlust im Trainingsset

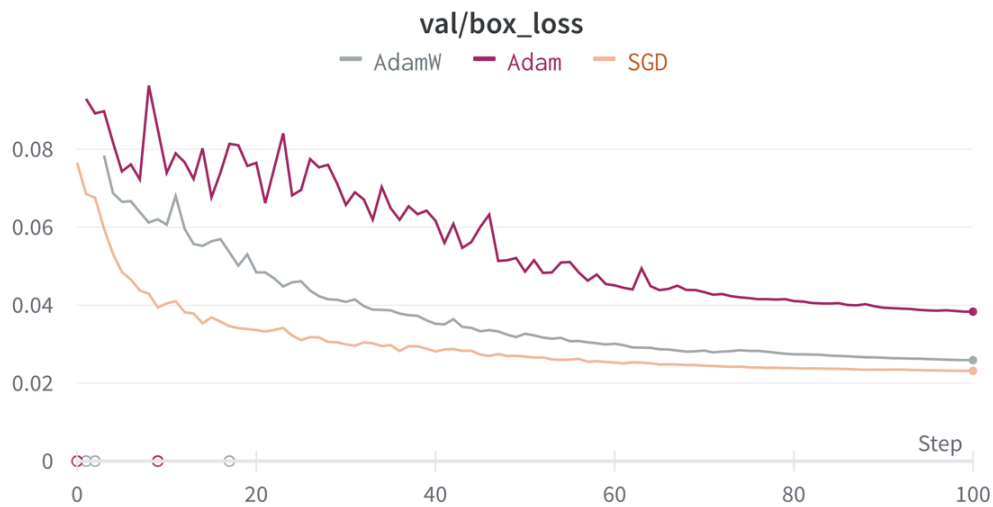


Abbildung 32: Experiment 3: Box-Verlust im Validierungsset

Es ist erkennbar, dass der Verlust in beiden Fällen am effektivsten minimiert werden kann, wenn das Modell den SGD-Optimierer verwendet. Nach den 100 Epochen erreichen der AdamW- und SGD-Optimierer einen Box-Verlust-Wert von 0,0259 bzw. 0,0232 im Validierungsset. Beim Adam-Optimierer fällt der Verlust mit 0,0383 (+ 48 bzw. 65 Prozent) deutlich höher aus. Im Trainingsset liegt der Box-Verlust vom AdamW-Optimierer ca. 35 Prozent über dem des SGD-Optimierers; im Validierungsset beträgt diese Differenz lediglich 12 Prozent.

Die jeweilige Fähigkeit zur Reduzierung der Verlustfunktionen spiegelt sich auch in der Vorhersagegenauigkeit der drei Modelle wider. Abbildung 33 zeigt die Entwicklung der mAP_0.5-Metrik über die 100 Epochen auf. Es ist zu erkennen, dass die mAP_0.5 beim SGD-Optimierer in den ersten 40 Epochen exponentiell ansteigt. Beim AdamW-Optimierer ist dagegen lediglich eine lineare Entwicklung erkennbar. Beim Adam-Optimierer liegt die finale mAP_0.5 um etwa zwei Prozentpunkte niedriger als beim SGD-Modell (97,6 vs. 95,8 Prozent). Durch die Verwendung des Adam-Optimierers wird lediglich ein mAP_0.5-Wert von 67,8 Prozent erreicht.

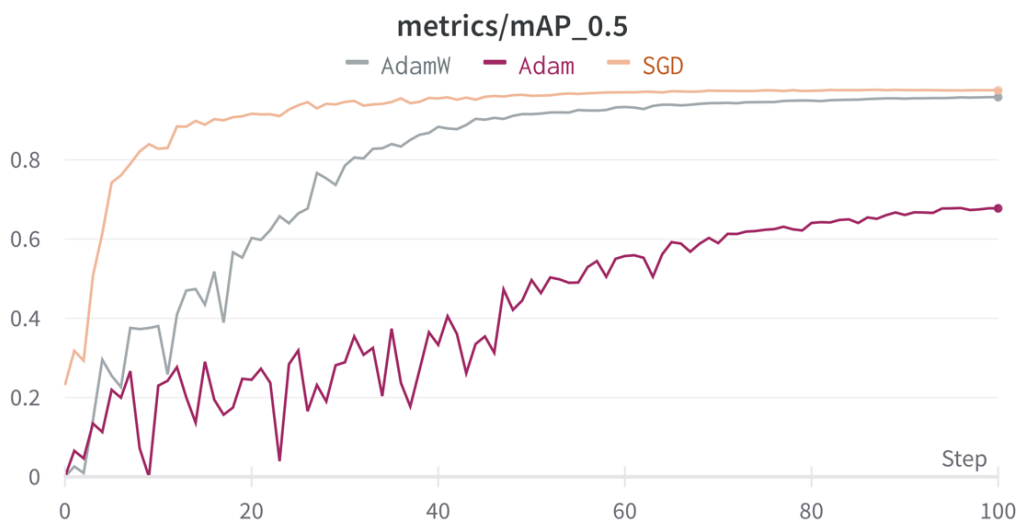


Abbildung 33: Experiment 3: Mean Average Precision (0.5)

Auch bei der mAP_0.5:0.95 ist eine ähnliche Staffelung der drei Modelle erkennbar (siehe Abbildung 34). Während der Performanceunterschied bei der mAP_0.5-Metrik zwischen dem SGD- und dem AdamW-Optimierer etwa 1,9 Prozentpunkte betrug, liegt dieser hier bei mehr als vier Prozent. Sowohl bei der mAP_0.5 als auch bei der mAP_0.5:95 trennen den Adam- und AdamW-Optimierer ca. 28 Prozentpunkte.

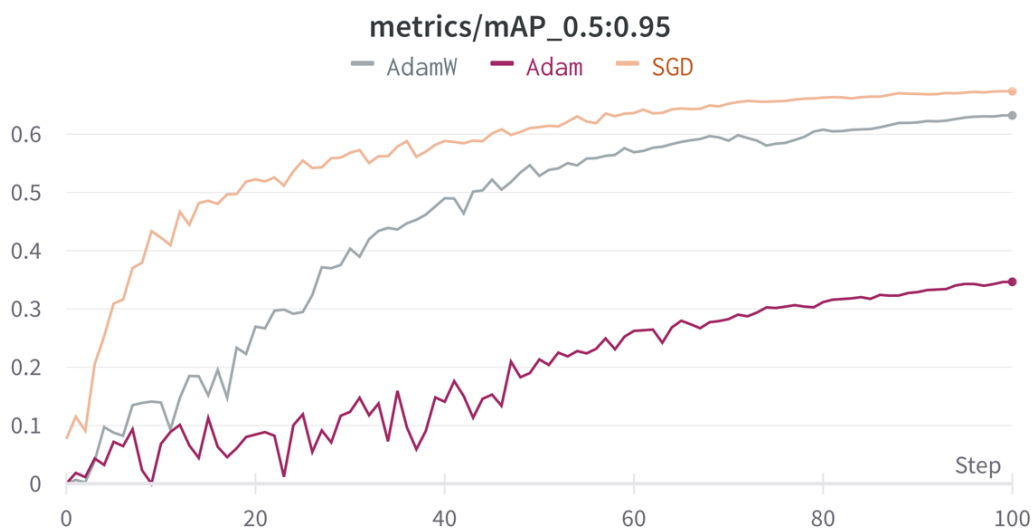


Abbildung 34: Experiment 3: Mean Average Precision (0.5:0.95)

Ein Blick auf Tabelle 8 lässt erkennen, dass der Adam-Optimierer Schwierigkeiten damit hat, sowohl Kaimauern als auch Dalben korrekt vorherzusagen. Insbesondere bei der Erkennung der Kaimauern ist eine große Performancedifferenz feststellbar: hier erreicht das Modell lediglich eine Mean Average Precision von 20,4 Prozent.

Tabelle 8: Experiment 3: Vorhersagegenauigkeit

Optimierer	mAP_0.5:0.95 (Dalbe)	mAP_0.5:0.95 (Kaimauer)
SGD	64,1	70,7
AdamW	60,2	66,3
Adam	48,9	20,4

Die Verwendung des Adam-Optimierers führt nicht nur dazu, dass das Modell die Hafengebäude unzuverlässig lokalisiert; auch die Klassifikationsgenauigkeit des Modells nimmt ab. Zur Ermittlung der Klassifikationsgenauigkeit verwendet YOLOv5 eine eigene Verlustfunktion – den sogenannten *Binary Cross-Entropy-Loss*. Diese Verlustfunktion misst die Differenz zwischen den vom Modell ausgegebenen Klassenwahrscheinlichkeiten und der Grundwahrheits-Klasse für jede Bounding Box [vgl. Koec22]. Abbildung 47 zeigt, dass der Klassenverlust im Trainingsset im Gegensatz zu den anderen beiden

Modellen beim Adam-Optimierer am wenigsten effizient minimiert wird. Zwar erreicht das Modell mit dem Adam-Optimierer auch hier den höchsten finalen Verlust-Wert, die Abstände zwischen den drei Modellen sind aber deutlich kleiner als beim Box-Verlust.

7.3.3.2 Diskussion

Zusammenfassend lässt sich festhalten, dass die Verwendung des Adam-Optimierers im vorliegenden Anwendungsfall dazu führt, dass das Modell ungenaue und fehlerhafte Vorhersagen – insbesondere bei hohen IoU-Schwellenwerten – trifft. Das Modell ist darüber hinaus nicht in der Lage, den Box- und Klassenverlust so schnell und nachhaltig zu minimieren, wie dies bei der Verwendung des AdamW- oder SGD-Optimierers der Fall ist. Eine Betrachtung der Performance-Metriken lässt erkennen, dass der Adam-Optimierer vor allem Probleme damit hat, einen hohen Recall zu erreichen (siehe Abbildung 46). Nach 100 Epochen erreicht das Modell einen finalen Recall-Wert von lediglich 59,8; beim SGD-Optimierer liegt der Recall bei 94,1 Prozent.

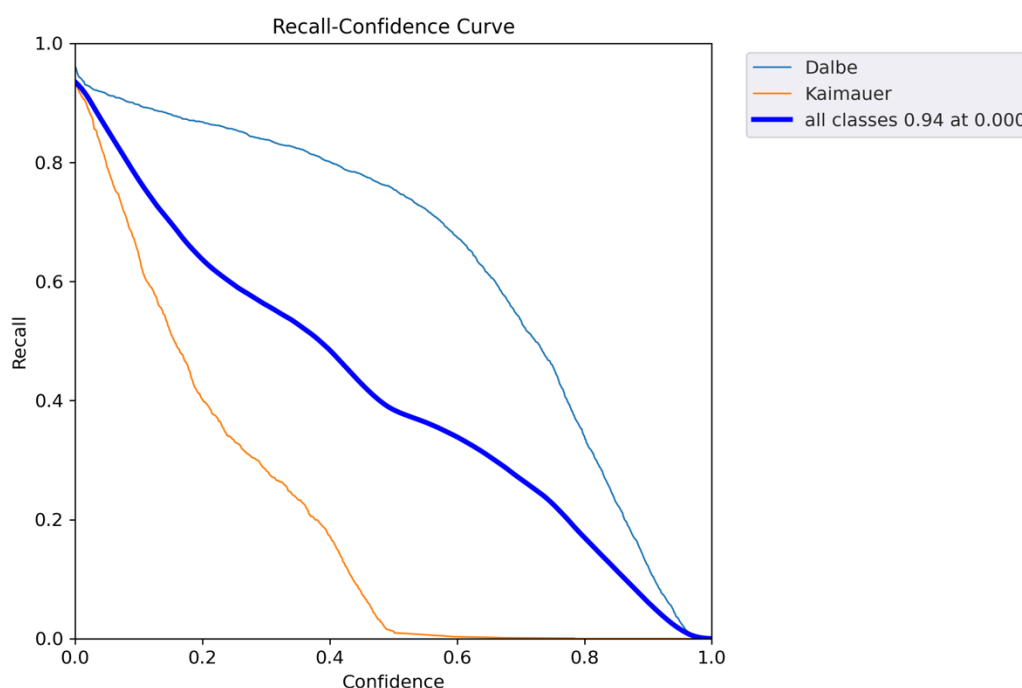


Abbildung 35: Experiment 3: Recall-Kurve: Adam-Optimierer

Wie in Abbildung 35 zu erkennen ist, stagniert der Recall im Konfidenzbereich von 50 bis 100 Prozent. Diese Beobachtung ist darauf zurückzuführen, dass das Modell dort viele Kaimauer-Objekte nicht zuverlässig lokalisiert. Durch die Generierung einer hohen

Anzahl an False Negatives kann der Recall in diesem Konfidenzbereich nicht weiter ansteigen ($\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$). Das Modell übersieht also viele Kaimauer-Objekte, für die Grundwahrheits-Bounding-Boxen existieren.

Im Vergleich zu den anderen Optimierungsalgorithmen ist der SGD-Optimierer im vorliegenden Anwendungsfall am besten in der Lage, die Effizienz des Gradient-Descent-Algorithmus zu verbessern und für eine schnelle und nachhaltige Minimierung der Verlustfunktionen zu sorgen. Insbesondere bei der Minimierung des Box-Verlustes ist der SGD-Optimierer den anderen beiden Algorithmen überlegen. Als Konsequenz erreicht das SGD-Modell auch die höchsten Mean-Average-Precision-Werte. Da die Auswahl des Optimierungsalgorithmus ansonsten keinen weiteren Einfluss auf die benötigten Trainingsressourcen und die Vorhersagegenauigkeit der Modelle hat, wurde sich dafür entschieden, für die weiteren Experimente fortan den SGD-Optimierer mit Nesterov-Momentum zu verwenden.

7.3.4 Experiment 4: Optimierung der sekundären Hyperparameter

Bis hierhin wurde ausschließlich die von YOLOv5 vorgegebene Standardkonfiguration für die Einstellung der sekundären Hyperparameter verwendet. Im Rahmen dieses Experimentes soll nun der in das YOLOv5-Repository integrierte, genetische Algorithmus verwendet werden, um eine optimale Hyperparameter-Kombination für den vorliegenden Anwendungsfall automatisiert zu ermitteln.

Das vorliegende YOLOv5-Modell hat insgesamt 29 einstellbare, sekundäre Hyperparameter. Das Modell verwendet eine initiale und eine finale Lernrate (l_0 und l_f), um die Stärke der Gewichtsadjustierungen über den Zeitverlauf zu steuern. Weitere sekundäre Hyperparameter sind die Länge der *Aufwärmphase*, die Stärke des verwendeten Momentums und *Aufwärm-Momentums*, die *Aufwärm-Lernrate*, der für das Training angewendete IoU-Schwellenwert, der Weight Decay und die Gewichtungsfaktoren für die Verlustfunktionen. Die Aufwärmphase gibt an, für wie viele Epochen die *Aufwärm-Lernrate* und das festgelegte *Aufwärm-Momentum* verwendet werden, bevor die initiale Lernrate und das eigentliche Momentum angewendet werden.

Der evolutionäre Algorithmus wurde so konfiguriert, dass er das Modell für jeweils zehn Epochen trainiert. Nach jeder dieser zehn Epochen wird die Vorhersagegenauigkeit des Modells evaluiert. Der genetische Algorithmus läuft für insgesamt 100 Epochen. Die Hyperparameteroptimierung nimmt damit 1000 Epochen (100 Generationen x 10 Epochen pro Generation) in Anspruch. Die Erzeugung eines neuen Nachkommens läuft so ab,

dass eine konkrete Lösung aus den besten n Lösungskandidaten aller bisherigen Generationen ausgewählt wird. In der ersten Generation existiert lediglich ein Lösungskandidat – die standardmäßig verwendete Hyperparameter-Konfiguration. Mit der Zeit werden immer mehr Hyperparameter-Kombinationen erzeugt. Der Algorithmus sieht vor, dass bei der Elternselektion maximal die fünf besten Lösungen in die nähere Auswahl kommen können. Jedem dieser maximal fünf Lösungskandidaten wird ein Gewicht zugeordnet. Lösungen mit einem größeren Fitnesswert werden stärker gewichtet als Lösungen mit einem kleineren Fitnesswert. Diese Gewichte werden anschließend verwendet, um aus den besten n Lösungen einen konkreten Nachkommen für die aktuelle Generation auszuwählen. Die Gewichtung der Kandidaten stellt sicher, dass Lösungen mit einer hohen Fitness bevorzugt ausgewählt werden. [vgl. Ultr23b]

Die Hyperparameter des so bestimmten Nachkommens werden anschließend komponentenweise mutiert. Dabei beträgt die Wahrscheinlichkeit 80 Prozent, dass ein Hyperparameter verändert wird. Für die Mutation wird der *Gaußsche Mutationsoperator* verwendet. Bei diesem Mutationsoperator wird zunächst ein zufälliger Wert aus der Normalverteilung (Varianz = 0, Mittelwert = 1) gezogen [vgl. Kram17, S.14]. Dieser Wert wird anschließend mit der sogenannten *Mutationsrate* multipliziert, welche die Stärke der Mutation steuert [vgl. Kram17, S.14]. In diesem Fall beträgt die Mutationsrate 0,2. Der dadurch entstehende Wert stellt den eigentlichen Mutationsfaktor dar, der mit dem Hyperparameter verrechnet wird. [vgl. Ultr23b]

```
def fitness(x):  
    # Model fitness as a weighted combination of metrics  
    w = [0.0, 0.0, 0.1, 0.9] # weights for [P, R, mAP@0.5, mAP@0.5:0.95]  
    return (x[:, :4] * w).sum(1)
```

Abbildung 36: Fitnessfunktion nach Ultralytics [vgl. Ultr23b]

Danach wird die Performance der entstandenen Lösung durch eine Fitnessfunktion evaluiert. Die Fitnessfunktion errechnet die gewichtete Summe aus den beiden Performance-Metriken mAP_0.5 (zehn Prozent) und mAP_0.5:0.95 (90 Prozent) (siehe Abbildung 36). [vgl. Ultr23b]

7.3.4.1 Ergebnisse

Abbildung 37 visualisiert die Ergebnisse der evolutionären Hyperparameteroptimierung. Die Boxen veranschaulichen, wie sich die Ausprägung der jeweiligen Parameter über die 100 Generationen entwickelt haben.

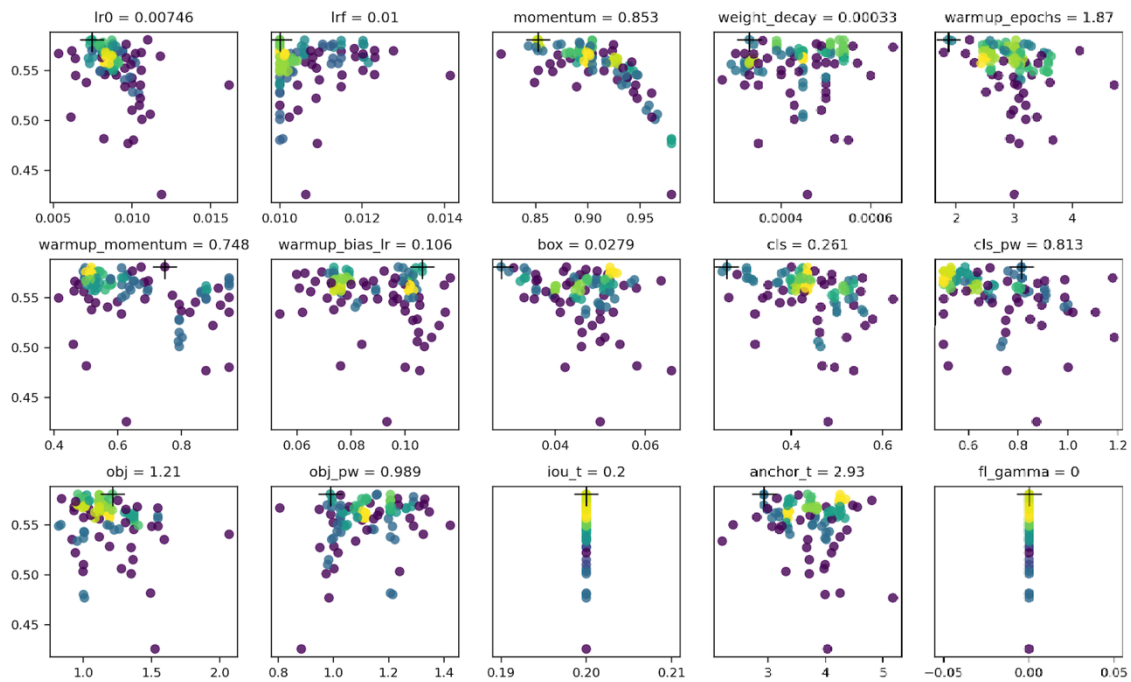


Abbildung 37: Ergebnisse der Hyperparameteroptimierung

Jeder farblich markierte Punkt steht für einen der einhundert Lösungskandidaten. Für jedes Individuum wird die konkrete Ausprägung des jeweiligen Hyperparameters auf der x-Achse abgetragen; auf der y-Achse ist der errechnete Fitnesswert der Lösungen zu erkennen. Die Farben der Punkte markieren Cluster, in denen ähnliche Wertausprägungen zu finden sind. Die Kreuze kennzeichnen jeweils die Hyperparameterausprägung des Individuums mit dem höchsten Fitnesswert. Im betrachteten Fall wurde die leistungsfähigste Hyperparameterkonfiguration nach 78 Generationen gefunden. Dieser Kandidat erreichte mit 93,3 Prozent in den ersten zehn Epochen den höchsten mAP_{0.5}-Wert.

Die beiden Abbildung 38 und Abbildung 39 visualisieren den Box-Verlust im Trainings- bzw. Validierungsset. Auf den Graphen ist die Entwicklung der Verlustkurven des Modells vor und nach der Anwendung der Hyperparameteroptimierung abgetragen. Es ist zu erkennen, dass die Hyperparameteroptimierung dazu geführt hat, dass das Modell besser in der Lage ist, den Box-Verlust zu minimieren. Dies bedeutet, dass das Modell

schneller bessere Ergebnisse bei der Anpassung an den Trainingsdatensatz und bisher unbekannte Eingabebilder erzielt.

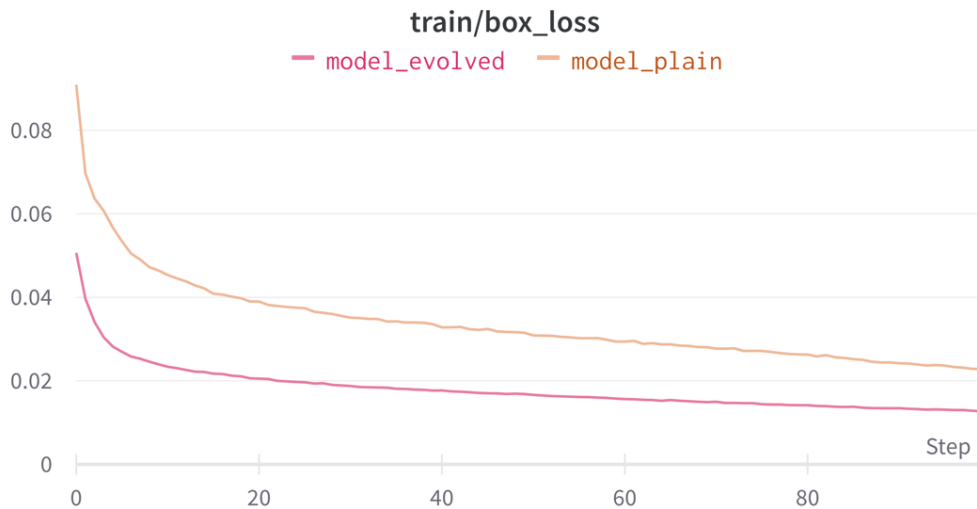


Abbildung 38: Experiment 4: Box-Verlust im Trainingsset

Auch der nach 100 Epochen erreichte Box-Verlust ist niedriger. Für das Trainingsset betrug dieser nach der Optimierung 0,0127, während er davor mit einem Wert von 0,0228 fast doppelt so hoch lag. Im Validierungsset konnte der Box-Verlust nach 100 Epochen durch die Hyperparameteroptimierung um fast 45 Prozent reduziert werden.

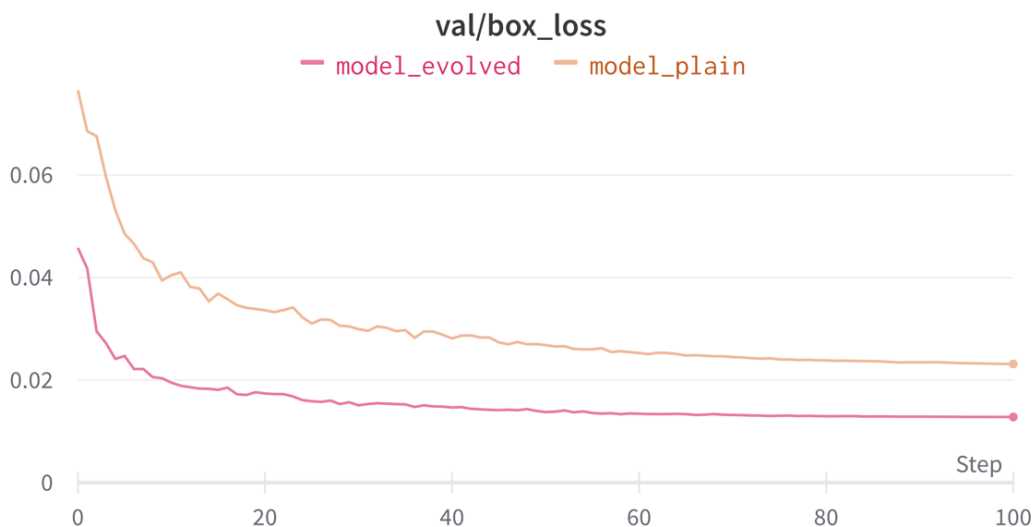


Abbildung 39: Experiment 4: Box-Verlust im Validierungsset

Ähnliche Entwicklungen sind beim Klassenverlust im Validierungsset erkennbar. Der finale Klassenverlust konnte durch die angepassten Hyperparameter von 0,0001837 auf 0,00005904 (-68 Prozent) reduziert werden (siehe Abbildung 40). Die Anpassungsfähigkeit an die Trainingsbilder konnte dagegen kaum verbessert werden (siehe Abbildung 48).

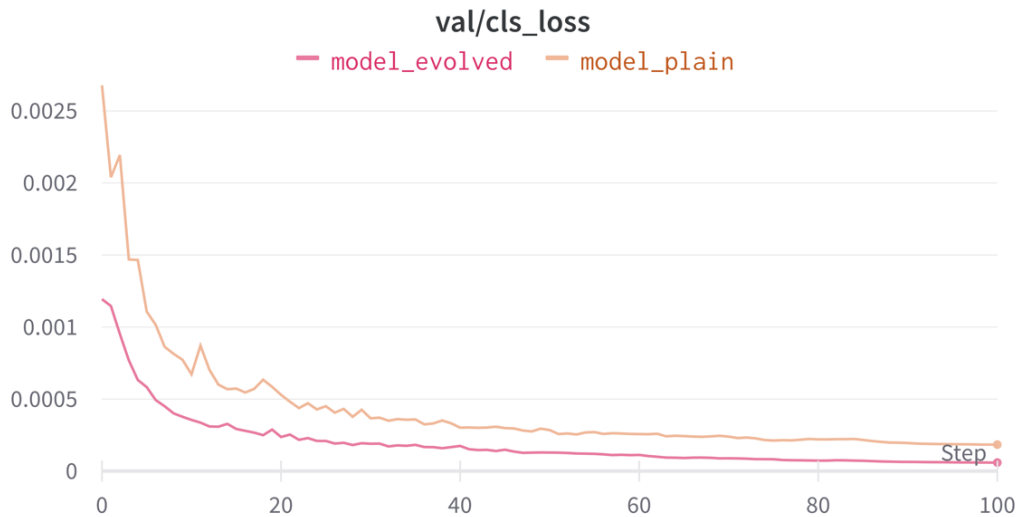


Abbildung 40: Experiment 4: Klassenverlust im Validierungsset

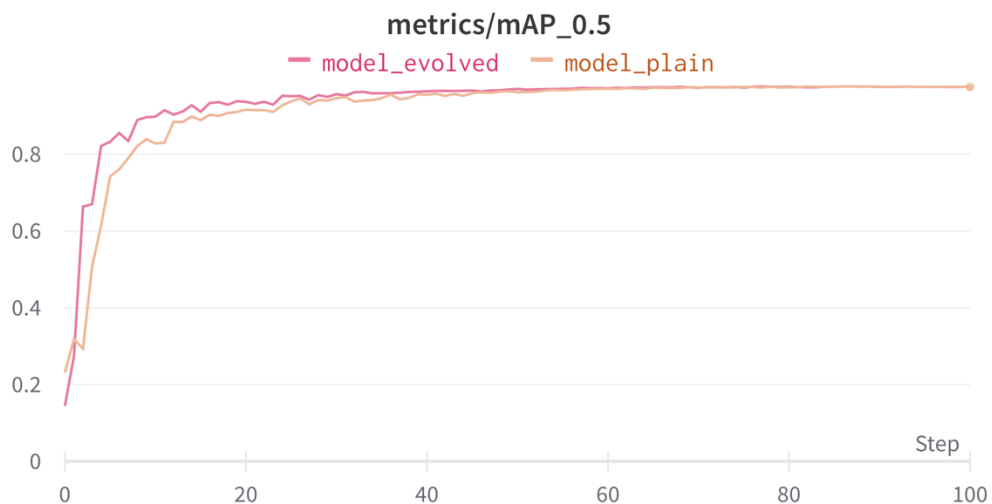


Abbildung 41: Experiment 4: Mean Average Precision (0.5)

Es fällt auf, dass die Precision, der Recall und die mAP-Metriken durch die Anpassung der Hyperparameter in kürzerer Zeit gesteigert werden können (siehe Abbildung 41 und Abbildung 49 bis Abbildung 51). Das ursprüngliche Modell mit der Standard-Hyperparameterkonfiguration benötigt zu Beginn mehr Epochen, um ähnlich hohe Performancewerte zu erreichen. Es ist auch zu erkennen, dass sich die Performancewerte der beiden Modelle immer weiter aneinander annähern bis nach 100 Epochen keine signifikante Differenz in der Mean Average Precision mehr messbar ist.

7.3.4.2 Diskussion

Die Effizienzsteigerung bei der Minimierung der Verlustfunktionen ist auf die Anpassung der Hyperparameter im Rahmen der evolutionären Optimierung zurückzuführen (siehe Tabelle 9).

Tabelle 9: Evolution der Hyperparameter

Hyperparameter	Vorher	Nachher
Initiale Lernrate (l0)	0,01	0,00746
Momentum	0,937	0,853
Aufwärmphase (Epochen)	3,0	1,87
Aufwärm-Momentum	0,8	0,748

Zum einen wurde die Aufwärmphase von drei auf 1,87 Epochen verringert. Durch die Reduktion der initialen Lernrate von 0,01 auf 0,00746, dem Momentum von 0,937 auf 0,853 und dem Aufwärm-Momentum nimmt die Wahrscheinlichkeit ab, dass das Modell direkt zu Beginn des Lernprozesses Täler in der Verlustfunktion überspringt. Es scheint so, als würde das Modell ohne die angepassten Hyperparameter ab einem gewissen Zeitpunkt in einem weniger optimalen Minimum stecken geblieben, als dies beim optimierten Modell der Fall ist. Dies ist der Grund, warum sich die finalen Verlust-Scores der beiden Modelle – insbesondere beim Box-Verlust – nach 100 Epochen unterscheiden.

Zusammenfassend lässt sich festhalten, dass die Optimierung der sekundären Hyperparameter dabei helfen konnte, dass YOLO-Modell besser auf die Trainingsbilder und bisher ungesehene Daten anzupassen. Die Hyperparameteroptimierung führte zwar zu

keiner Verbesserung der finalen Modellperformance, dennoch konnte durch sie eine Effizienzsteigerung im Trainingsprozess erreicht werden. Das Modell ist nun in der Lage, die Verlustfunktionen nachhaltig stärker zu minimieren und die Precision, den Recall und die Mean Average Precision in den ersten Epochen schneller zu steigern. Als Grund für die Annäherung der finalen Mean-Average-Precision-Werte kann die unterschiedlich starke Gewichtung der Verlustfunktionen genannt werden (Box-Verlust: 0,05 (vorher) vs. 0,0279 (nachher); Klassenverlust: 0,5 (vorher) vs. 0,261 (nachher)).

7.4 Finales Modell

Das finale Modell verwendet die optimalen Einstellungen für die Hyperparameter, die im Rahmen der vier Experimente ermittelt wurden (Bildauflösung: [1280x720], Batchgröße: 64, Optimierungsalgorithmus: SGD mit Nesterov-Momentum und die optimierten Hyperparameter). Wie bereits erläutert, soll ein Early Stopping angewendet werden, um das finale Modell zu trainieren. Um dies zu erreichen, wurde dem Trainingskript das Argument „patience = 50“ übergeben. Dadurch wird das Training abgebrochen, wenn für 50 Epochen keine Verbesserung der mAP_0.5-Metrik mehr verzeichnet werden kann. Parallel wurde darauf geachtet, dass der Validierungs-Verlust nicht wieder ansteigt, um ein Overfitting zu vermeiden. Durch die Implementierung des Early Stoppings wurde das Training nach 306 Epochen vorzeitig abgebrochen. Im Folgenden wird die Performance des finalen Modells näher erläutert.

Tabelle 10: Performance-Kennzahlen des finalen Modells

Performance-Metrik	Wert
mAP_0.5 (%)	97,7
mAP_0.5 (Dalbe) (%)	97,2
mAP_0.5 (Kaimauer) (%)	98,2
mAP_0.5:0.95 (%)	68,2
mAP_0.5:0.95 (Dalbe) (%)	64,9
mAP_0.5:0.95 (Kaimauer) (%)	71,5
Inferenzzeit (ms pro Bild)	7,7

Tabelle 10 zeigt die Mean-Average-Precision-Werte und Inferenzzeit des finalen Modells. Es ist zu erkennen, dass die Vorhersagegenauigkeit des Modells bei der Erkennung der Kaimauern sowohl bei der mAP_0.5- als auch der mAP_0.5:0.95-Metrik höher ausfällt als bei der Erkennung der Dalben.

Während die Performancedifferenz in der klassenbezogenen mAP_0.5 lediglich einen Prozentpunkt beträgt, liegt diese bei der mAP_0.5:0.95 bei 6,6 Prozentpunkten. Diese Beobachtung konnte bereits in den vorangegangenen Experimenten gemacht werden. Als Grund für diese Performancedifferenz kann die Größe der Objekte festgemacht werden. Die Dalben-Objekte sind in der Regel deutlich kleiner als die Kaimauer-Objekte. Aufgrund der Funktionsweise der YOLO-Architektur kommt es häufiger vor, dass derartige Objekte nicht immer ausreichend präzise lokalisiert werden können [vgl. ZCSG23, S.4] [vgl. RDGF15, S.784]. Dieses Problem wird besonders dann deutlich, wenn höhere IoU-Schwellenwerte angesetzt werden.

Wie in Kapitel 4.2 beschrieben, muss das finale Modell eine Vorhersagegenauigkeit (mAP_0.5) von mindestens 95 Prozent erreichen. Diese Muss-Anforderung kann durch das finale Modell um 2,7 Prozentpunkte übertroffen werden. Weiterhin sollte das Modell eine Vorhersagegeschwindigkeit von 30 Bildern pro Sekunde (FPS) erreichen, um zu ermöglichen, dass das Modell bei Bedarf in ein echtzeitfähiges Frühwarn- und Navigationssystem eingebunden werden kann. Diese Anforderung wird vom finalen Modell ebenfalls erfüllt. Mit einer Inferenzzeit von 7,7 Millisekunden pro Bild erreicht das finale Modell eine Vorhersagegeschwindigkeit von ca. 130 FPS.

Zusammenfassend lässt sich festhalten, dass das hier implementierte Modell in der maritimen Domäne einsetzbar ist (Anforderung 1) und statische Hafenobjekte auf Bildern bei Tageslicht erkennen kann (Anforderung 2 und 3). Eine Überprüfung des Modelloutputs auf dem Trainingsset lässt erkennen, dass das Modell ebenfalls in der Lage ist, die Kaimauern und Dalben aus unterschiedlich großen Entfernungen zu erkennen (Anforderung 5) (siehe Abbildung 42). Weiterhin werden auch Hafenobjekte korrekt erkannt, wenn diese durch Überschneidungen mit anderen Objekten nicht vollständig sichtbar sind (Anforderung 6) (siehe Abbildung 42 unten) oder durch den Bildrand abgeschnitten werden (Anforderung 7) (siehe Abbildung 42 oben und Mitte).



Abbildung 42: Vorhersagen des finalen Modells

Auch bei Wellenbewegungen und starkem Wellengang erkennt das Modell die Hafenobjekte zuverlässig. Da das Modell achsenparallele Bounding Boxen verwendet, werden die leicht geneigten Objekte in diesem Fall allerdings nicht immer optimal von den vom Modell vorhergesagten Boxen umschlossen.

Wie im Konzept beschrieben, soll auch untersucht werden, ob sich das im Rahmen dieser Arbeit implementierte Modell für die automatisierte Erkennung von Kaimauern und Dalben in anderen Hafengebieten nutzen lässt. Dazu wurden insgesamt 30 Bilder aus dem Internet gesammelt, die Kaimauern und Dalben in anderen Hafengebieten abbilden. Anschließend wurde das finale YOLO-Modell verwendet, um Vorhersagen auf diesen bisher ungesehenen Bildern zu tätigen. Die Evaluation der Ergebnisse ergab, dass die Performance des Modells in nur einem sehr begrenzten Maß auf andere Anwendungsfälle übertragbar ist. Auf 14 der 30 Bilder wurden die Dalben und Kaimauern aufgrund ihres abweichenden Erscheinungsbildes (z.B. Farbe oder Form) überhaupt nicht erkannt. Auf vier Bildern wurden Objekte erkannt, allerdings waren die Vorhersagen falsch, da andere Objekte (z.B. Schiffe oder Schilder) fälschlicherweise als Hafenobjekte klassifiziert wurden (Abbildung 43)



Abbildung 43: Beispiele für fehlerhafte Klassifikationen [vgl. Welc00] und [vgl. Stre00]

Auf weiteren sechs Bildern konnten Hafenobjekte korrekt klassifiziert werden. Das Problem war hier, dass die Objekte nur unzureichend präzise lokalisiert werden konnten. Die Bounding Boxen waren entweder zu weit oder zu eng gesetzt (siehe Abbildung 44).



Abbildung 44: Beispiel für eine fehlerhafte Lokalisation [vgl. Sani14]

Auf lediglich sechs der 30 Bilder konnte das Modell – insbesondere für die Erkennung der Dalben – brauchbare Vorhersagen treffen. Es fiel auf, dass das Modell immer nur dann korrekte Vorhersagen treffen konnte, wenn das Aussehen der Objekte auf den Bildern den Kaimauern und Dalben aus dem Trainingsdatensatz ähnelte. Ein Blick auf Abbildung 45 lässt erkennen, dass auch die korrekten Vorhersagen eine geringe Konfidenz aufweisen und viele ähnlich aussehende Zielobjekte im Hinter- und Vordergrund des Bildes übersehen werden.



Abbildung 45: Modellvorhersagen [vgl. Roev21] und [vgl. 50bi00]

8 Fazit und Ausblick

Im Rahmen dieser Arbeit konnte gezeigt werden, dass die automatisierte Erkennung von statischen Hafengebieten im Hafengebiet bisher nicht hinreichend untersucht wurde. Die aktuelle Forschung und marktreifen Lösungen fokussieren sich nahezu ausschließlich auf die Schiffserkennung. Um diese Forschungslücke zu schließen, sollte in dieser Arbeit die Forschungsfrage: Wie lassen sich statische Hafengebiete auf Kamerabildern automatisch identifizieren? beantwortet werden.

Da aktuell kein geeigneter Datensatz für die Abbildung von statischen Hafengebieten existiert, bestand das erste Teilziel darin, einen eigenen, domänenspezifischen Datensatz aufzubauen. Als Datengrundlage wurden Kamerabilder aus dem Emdener Hafen verwendet. Für die Repräsentation der statischen Hafengebiete wurden sowohl Kaimauern als auch Dalben annotiert.

In einem zweiten Schritt wurde eine passende Methode für die automatisierte Erkennung der statischen Hafengebiete ausgewählt. Auf der Basis der zuvor festgelegten Auswahlkriterien wurde sich entschieden, dafür das Object-Detection-Modell YOLOv5 zu implementieren. Dieses Modell wurde anschließend parametrisiert, um es optimal auf den vorliegenden Anwendungsfall anzupassen. Als Datengrundlage für das Training des Modells wurden die Bilder aus dem Emdener Hafen verwendet. Um eine geeignete Modellkonfiguration zu finden, wurden insgesamt vier Experimente durchgeführt, in denen der Effekt von unterschiedlichen Hyperparameter-Konfigurationen auf die Modellperformance und die benötigten Trainingsressourcen untersucht wird. Die Untersuchungsergebnisse wurden anschließend dafür genutzt, um eine begründete Entscheidung bei der Konfiguration des finalen Modells zu treffen.

Im Rahmen der Hyperparameteroptimierung konnte beobachtet werden, dass insbesondere die Zuverlässigkeit bei der Erkennung von kleineren Objekten wie Dalben verbessert werden kann, wenn die Auflösung der Eingabebilder erhöht wird. Die Verwendung einer nativen Bildauflösung von [1280x720] hat darüber hinaus den Vorteil, dass die Bilder im Produktivbetrieb direkt verarbeitet werden können, ohne, dass zuvor ein weiterer Vorverarbeitungsschritt durchgeführt werden muss. Dadurch lässt sich die Vorhersagegeschwindigkeit des Modells verbessern. Im Rahmen des zweiten Experimentes ließ sich feststellen, dass eine höhere Batchgröße mit einer Verbesserung der Modellgenauigkeit und Reduzierung der Trainingsdauer einhergeht. In diesem Fall wird allerdings auch mehr Speicherplatz benötigt. Das dritte Experiment ergab, dass sich der Algorithmus „Stochastic Gradient Descent mit Momentum“ am besten für die Erkennung der statischen Hafengebiete im vorliegenden Anwendungsfall eignet. In einem letzten Schritt

wurden die übrigen, sekundären Hyperparameter des Modells mithilfe eines genetischen Algorithmus optimiert. Durch die Hyperparameteroptimierung konnte die Verlustfunktion – insbesondere der Box-Verlust – nachhaltiger minimiert und das Modell somit besser an die unterliegenden Bilder angepasst werden. Es ist zu berücksichtigen, dass die im Rahmen der Experimente ermittelte, optimale Hyperparameterkonfiguration nicht vollständig auf andere Anwendungsfälle übertragbar ist. Nur weil sich die Hyperparameterkombination für den hier betrachteten Anwendungsfall eignet, heißt dies nicht, dass diese auch für alle anderen Anwendungsfälle optimale Ergebnisse liefert (*No-Free-Lunch-Theorem*) [vgl. Glas21, S.422]. Dies gilt insbesondere für den verwendeten Optimierungsalgorithmus und die ermittelten, sekundären Hyperparameter.

Die Auswertung des finalen Modellperformance ergab, dass alle erhobenen, funktionalen Muss-Anforderungen erfüllt werden konnten. Die beiden performancebezogenen Anforderungen an die zu erreichende Vorhersagegenauigkeit ($mAP_{0.5} \geq 95\%$) und -geschwindigkeit (≥ 30 FPS) des Modells konnten übertroffen werden. Es konnte auch gezeigt werden, dass die Performance des hier implementierten Modells nicht ohne Weiteres auf andere Hafengebiete übertragbar ist. Grund dafür ist, dass sich das Aussehen der Hafengebiete von Hafengebiet zu Hafengebiet zu stark von den Kaimauern und Dalben aus dem Emden Hafen unterscheidet. Das führt dazu, dass das Modell diese Objekte auf den ungesesehenen Bildern nicht wiedererkennt. Die getätigten Modellvorhersagen sind in den meisten Fällen sehr ungenau und fehlerhaft.

Auch wenn sich das in dieser Arbeit implementierte Modell nicht für die Erkennung von statischen Hafengebieten in anderen Hafengebieten eignet, kann das hier erstellte Konzept dennoch als Orientierungshilfe dienen, um geeignete Maßnahmen beim Aufbau eines eigenen Datensatzes und der Implementierung einer individuellen YOLOv5s6-Instanz zu treffen. Die beschriebenen Schritte für die Aufbereitung der Rohdaten, das Labeling des Datensatzes und die Datenaugmentierung lassen sich bei Bedarf ohne weitere Zwischenschritte auf die Erkennung von beliebigen, statischen Hafengebieten in anderen Hafengebieten übertragen. Auch die Handlungsempfehlungen bei der Einstellung der Bildauflösung und Batchgröße lassen sich universell nutzen. In diesem Fall müssen lediglich die unterliegenden Trainingsbilder ausgetauscht werden. Der im Rahmen dieser Arbeit erstellte Datensatz bietet darüber hinaus eine Datengrundlage, um die Forschung im Bereich der Erkennung von statischen Hafengebieten weiter voranzutreiben.

Zusammenfassend lässt sich festhalten, dass die formulierte Forschungsfrage erfolgreich beantwortet werden konnte. Durch die Evaluation der Modellperformance konnte gezeigt werden, dass das implementierte YOLOv5-Modell in der Lage ist, zuverlässige und schnelle Vorhersagen zu treffen und robust auf die spezifischen Herausforderungen in der maritimen Domäne zu reagieren.

In einem nächsten Schritt kann daran gearbeitet werden, das Modell in bestehende Navigationssysteme zu integrieren, um die automatisierte Hinderniserkennung von Kaimauern und Dalben im Emden Hafen zu unterstützen. Durch die Echtzeitfähigkeit des Modells ist sichergestellt, dass der Besatzung an Bord die Modellausgaben zeitnah entscheidungsunterstützend zur Verfügung gestellt werden können. Damit kann dazu beigetragen werden, dass die Wahrscheinlichkeit für Kollisionen mit Kaimauern und Dalben reduziert wird und somit Schäden von der Besatzung, dem Schiff und der maritimen Infrastruktur abgewendet werden können. Die Verwendung des YOLOv5s6-Modells bietet außerdem den Vorteil, dass Bilder im Produktivbetrieb direkt an Bord verarbeitet werden können. Das KI-basierte Navigationssystem würde sich in erster Linie für den Einsatz auf Schiffen eignen, die sich häufig oder ausschließlich im Emden Hafen aufhalten. Als Beispiele sind hier Arbeitsboote wie Baggerschiffe oder Schlepper zu nennen.

Es ist davon auszugehen, dass in Zukunft die Möglichkeit besteht, Kamerasysteme mit KI-basierten Objekterkennungsalgorithmen zu koppeln, um eine vollständig autonome Navigation von Schiffen im Hafengebiet zu realisieren. Ähnlich wie bei dem autonomen Fahren in der Automobilbranche ist die alleinige Verwendung von Kameras zur Hinderniserkennung heute noch mit großen Risiken verbunden. Nebel, Regen, Schnee oder Dunkelheit können dazu führen, dass die Kameras die statischen Hafenobjekte nicht mehr erkennen können; in diesem Fall würde das Modell keine zuverlässigen Ergebnisse mehr liefern und viele Hindernisse übersehen. Für die Realisierung einer vollautonomen Navigation muss noch mehr Forschung betrieben werden, um die Zuverlässigkeit der Modellvorhersagen zu verbessern. Außerdem können verschiedene Sensortypen wie Kameras, Radare oder Lidar-Systeme kombiniert werden, um die Fehleranfälligkeit dieser Navigationssysteme zu minimieren.

9 Literaturverzeichnis

- [50bi00] 50 Bilder aus Holstenniendorf - Schiffbilder.de. URL <https://www.schiffbilder.de/name/karte/place/holstenniendorf/lon/9.3/lat/54.04.html#9.3,54.04,17>. - abgerufen am 2023-05-10
- [Ai-r22] *AI-ris - Sea Machines*. URL <https://sea-machines.com/ai-ris/>. - abgerufen am 2023-01-13
- [Alli22] ALLIANZ GLOBAL CORPORATE & SPECIALTY (AGCS): *Safety and Shipping Review 2022: Allianz Global Corporate & Specialty, 2022*
- [Bage22] *Bagenkop*. URL <https://en.wikipedia.org/w/index.php?title=Bagenkop&oldid=110077162>. - abgerufen am 2023-01-14. — *Wikipedia*
- [Bars00] *Barschangeln an Spundwänden*. URL <https://www.simfisch.de/barschangeln-an-spundwaenden/>. - abgerufen am 2023-01-14. — *Simfisch.de – Angeln und Outdoor!*
- [BeBe12] BERGSTRA, JAMES; BENGIO, Y.: Random Search for Hyper-Parameter Optimization. In: *The Journal of Machine Learning Research* Bd. 13 (2012), S. 281–305
- [Begr17a] *Begriff: Kai* | *GROSSE-SEEFAHRT.de*. URL <https://www.grosse-seefahrt.de/lexikon/Kai/>. - abgerufen am 2023-01-14
- [Begr17b] *Begriff: Dalbe* | *GROSSE-SEEFAHRT.de*. URL <https://www.grosse-seefahrt.de/lexikon/Dalbe/>. - abgerufen am 2023-01-14
- [BMPK18] BOVCON, BORJA; MANDELJC, ROK; PERŠ, JANEZ; KRISTAN, MATEJ: Stereo obstacle detection for unmanned surface vehicles by IMU-assisted semantic segmentation, arXiv (2018)
- [Brem20] BREMS, MATT: *5 Strategies for Handling Unbalanced Classes*. URL <https://blog.roboflow.com/handling-unbalanced-classes/>. - abgerufen am 2023-03-23. — *Roboflow Blog*
- [BuEs10] BURKLE, AXEL; ESSENDORFER, BARBARA: Maritime surveillance with integrated systems. In: *2010 International WaterSide Security Conference*. Carrara, Italy: IEEE, 2010 — ISBN 978-1-4244-8894-0, S. 1–8

- [Chol18] CHOLLET, FRANÇOIS: *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018 — ISBN 978-1-61729-443-3
- [Chri22] CHRISTIANSEN, ANDERS: *Anchor Boxes — The key to quality object detection*. URL <https://towardsdatascience.com/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9>. - abgerufen am 2023-04-11. — Medium
- [Dalb15] *Dalben im Nord-Ostsee-Kanal sind wichtig für die Schifffahrt*. URL <https://www.brueckenbote.de/dalben-im-nord-ostsee-kanal/>. - abgerufen am 2023-01-14. — Brückenbote
- [Dolp00] *Dolphin - International Dictionary of Marine Aids to Navigation*. URL <https://www.iala-aism.org/wiki/dictionary/index.php/Dolphin>. - abgerufen am 2023-01-14
- [Dwye20] DWYER, BRAD: *Ontology Management for Computer Vision*. URL <https://blog.roboflow.com/label-management-for-computer-vision/>. - abgerufen am 2023-03-22. — Roboflow Blog
- [Euro21] EUROPEAN MARITIME SAFETY AGENCY (EMSA): *Annual Overview of Marine Casualties and Incidents 2021*, 2021
- [FeGL10] FEFILATYEV, SERGIY; GOLDFOG, DMITRY; LEMBKE, CHAD: Tracking Ships from Fast Moving Camera through Image Registration. In: *2010 20th International Conference on Pattern Recognition*. Istanbul, Turkey: IEEE, 2010 — ISBN 978-1-4244-7542-1, S. 3500–3503
- [FSNP21] FERNANDES RAMOS, RAMIRO; STRAUHS, MIKAEL; NETO, SEVERINO VIRGÍNIO; PAULINO DE LIRA, ANTONIO RAFAEL; CEPEDA, MARICRUZ AURELIA FUN SANG; GUAYCURU DE CARVALHO, LUIZ FELIPE; MARQUES DE OLIVEIRA MOITA, JOÃO VITOR; CAPRACE, JEAN-DAVID: A Review of Deep Learning Application for Computational Vision within the Maritime Industry. In: *Anais do 12º Seminário Internacional de Transporte e Desenvolvimento Hidroviário Interior*. Galoa, 2021
- [Gao22] GAO, YI: Research on the Application of Artificial Intelligence Technology in the Development of Computer Vision. In: *Highlights in Science, Engineering and Technology* Bd. 9 (2022), S. 80–84
- [Glas21] GLASSNER, ANDREW S.: *Deep learning: a visual approach*. San Francisco, CA: No Starch Press, Inc, 2021 — ISBN 978-1-71850-073-0

- [GLZJ21] GAI, WENDONG; LIU, YAKUN; ZHANG, JING; JING, GANG: An improved Tiny YOLOv3 for real-time object detection. In: *Systems Science & Control Engineering* Bd. 9 (2021), Nr. 1, S. 314–321
- [Grae20] GRAETZ, FABIO M.: *Why AdamW matters*. URL <https://towardsdatascience.com/why-adamw-matters-736223f31b5d>. - abgerufen am 2023-05-12. — Medium
- [HACS08] HUCK, ROBERT C.; AL AKKOUMI, MUHAMMAD K.; CHENG, SAMUEL; SLUSS, JR., JAMES J.; LANDERS, THOMAS L.: Aerial surveillance vehicles augment security at shipping ports. In: SPIE Europe Security and Defence. (Hrsg.): CARAPEZZA, E. M.. Cardiff, Wales, United Kingdom, 2008, S. 71120J
- [HDPR17] HECKER, DR. DIRK; DÖBEL, INGA; PETERSEN, ULRIKE; RAUSCHERT, ANDRÉ; SCHMITZ, VERLINA; VOSS, DR. ANGELIKA: *Zukunftsmarkt künstliche Intelligenz - Potenziale und Anwendungen*, 2017
- [HRSZ16] HUANG, JONATHAN; RATHOD, VIVEK; SUN, CHEN; ZHU, MENGLONG; KORATTIKARA, ANOOP; FATHI, ALIREZA; FISCHER, IAN; WOJNA, ZBIGNIEW; U. A.: Speed/accuracy trade-offs for modern convolutional object detectors (2016)
- [Iala00] IALA: *Pier - International Dictionary of Marine Aids to Navigation*. URL <https://www.iala-aism.org/wiki/dictionary/index.php/Pier>. - abgerufen am 2023-01-14
- [Imag23] *Image Augmentation*. URL <https://docs.roboflow.com/image-transformations/image-augmentation>. - abgerufen am 2023-04-06. — Roboflow
- [IMAR22] INDHURANI, A; MANIMEGALAI, A; ARUNPANDIYAN, I; RAMACHANDRAN, M; SATHIYARAJ, CHINNASAMY: Exploring Recent Trends in Computer Vision. In: *Electrical and Automation Engineering* Bd. 1 (2022), Nr. 1, S. 33–39
- [IRYM22] ISA, IZA SAZANITA; ROSLI, MOHAMED SYAZWAN ASYRAF; YUSOF, UMI KALSOM; MARUZUKI, MOHD IKMAL FITRI; SULAIMAN, SITI NORAINI: Optimizing the Hyperparameter Tuning of YOLOv5 for Underwater Detection. In: *IEEE Access* Bd. 10 (2022), S. 52818–52831
- [ISZL21] IANCU, BOGDAN; SOLOVIEV, VALENTIN; ZELIOLI, LUCA; LILIUS, JOHAN: ABOships—An Inshore and Offshore Maritime Vessel Detection Dataset with Precise Annotations. In: *Remote Sensing* Bd. 13 (2021), Nr. 5, S. 988

- [Joch20] YOLOv5 by Ultralytics. URL <https://github.com/ultralytics/yolov5>. - abgerufen am 2023-04-19. — Ultralytics
- [Joka22] JOKANOVIĆ, VUKOMAN: Computer Vision and Internet of Things. In: *Computer Vision and Internet of Things, 2022*
- [Kaim00] *Kaimauer kaputt - Reparatur kostet 310000 Euro*. URL https://www.dewezet.de/region/hameln_artikel,-kaimauer-kaputt-reparatur-kostet-310000-euro-_arid,577168.html. - abgerufen am 2023-01-14
- [KaYH16] KAIDO, NAOYA; YAMAMOTO, SHIGEHIRO; HASHIMOTO, TAKESHI: Examination of automatic detection and tracking of ships on camera image in marine environment. In: *2016 Techno-Ocean (Techno-Ocean)*. Kobe, Japan: IEEE, 2016 — ISBN 978-1-5090-2445-2, S. 58–63
- [Khan21] KHANDELWAL, RENU: *Different IoU Losses for Faster and Accurate Object Detection*. URL <https://medium.com/analytics-vidhya/different-iou-losses-for-faster-and-accurate-object-detection-3345781e0bf>. - abgerufen am 2023-05-08. — Analytics Vidhya
- [Koec22] KOECH, KIPRONO ELIJAH: *Cross-Entropy Loss Function*. URL <https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e>. - abgerufen am 2023-05-08. — Medium
- [KOVA21] KHATAB, ESRAA; ONSY, AHMED; VARLEY, MARTIN; ABOUELFARAG, AHMED: Vulnerable objects detection for autonomous driving: A review. In: *Integration* Bd. 78 (2021), S. 36–48
- [KPPJ21] KANCHANA, BIMSARA; PEIRIS, ROJITH; PERERA, DAMITHA; JAYASINGHE, DULANI; KASTHURIRATHNA, DHARSHANA: Computer Vision for Autonomous Driving. In: *2021 3rd International Conference on Advancements in Computing (ICAC)*. Colombo, Sri Lanka: IEEE, 2021 — ISBN 978-1-66540-862-2, S. 175–180
- [Kram17] KRAMER, OLIVER: *Genetic Algorithm Essentials, Studies in Computational Intelligence*. 1st ed. 2017. Cham: Springer International Publishing: Imprint: Springer, 2017 — ISBN 978-3-319-52156-5
- [LeRO20] LEE, SUNG-JUN; ROH, MYUNG-IL; OH, MIN-JAE: Image-based ship detection using deep learning. In: *Ocean Systems Engineering* Bd. 10 (2020), Nr. 4, S. 415–434

- [LiLi19] LIASHCHYNSKYI, PETRO; LIASHCHYNSKYI, PAVLO: Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS, arXiv (2019)
- [LLLY20] LU, XIN; LI, QUANQUAN; LI, BUYU; YAN, JUNJIE: MimicDet: Bridging the Gap Between One-Stage and Two-Stage Object Detection, arXiv (2020)
- [MKJT19] MOOSBAUER, SEBASTIAN; KONIG, DANIEL; JAKEL, JENS; TEUTSCH, MICHAEL: A Benchmark for Deep Learning Based Object Detection in Maritime Environments. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Long Beach, CA, USA: IEEE, 2019 — ISBN 978-1-72812-506-0, S. 916–925
- [Möll21] MÖLLER, BENNET: *Infrastrukturausbau am Hafen soll voranschreiten*. URL <https://fink.hamburg/2021/01/infrastrukturausbau-am-hafen-soll-voranschreiten/>. - abgerufen am 2023-01-14. — FINK.HAMBURG
- [NCLP22] NANDA, ABHILASHA; CHO, SUNG WON; LEE, HYEOPWOO; PARK, JIN HYOUNG: KOLOMVERSE: KRISO open large-scale image dataset for object detection in the maritime universe, arXiv (2022)
- [Nels20a] NELSON, JOSEPH: *You might be resizing your images incorrectly*. URL <https://blog.roboflow.com/you-might-be-resizing-your-images-incorrectly/>. - abgerufen am 2023-03-25. — Roboflow Blog
- [Nels20b] NELSON, JOSEPH: *How to Label Images for Computer Vision Models*. URL <https://blog.roboflow.com/tips-for-how-to-label-images/>. - abgerufen am 2023-03-07. — Roboflow Blog
- [OIDe08] OLSON, DAVID L.; DELEN, DURSUN: *Advanced data mining techniques*. Berlin Heidelberg: Springer, 2008 — ISBN 978-3-540-76917-0
- [Orca00] *Orca-AI - Avoid Collisions, Save Lives*. URL <https://www.orca-ai.io/>. - abgerufen am 2023-01-13
- [PaLA20] PADILLA, RAFAEL; LIMA NETTO, SERGIO; A. B. DA SILVA, EDUARDO: A Survey on Performance Metrics for Object-Detection Algorithms. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. Niteroi, Brazil: IEEE, 2020 — ISBN 978-1-72817-539-3
- [Quay00] *Quay - International Dictionary of Marine Aids to Navigation*. URL <https://www.iala-aism.org/wiki/dictionary/index.php/Quay>. - abgerufen am 2023-01-14

- [Rajp20] RAJPUT, MIHIR: *YOLO V5 — Explained and Demystified*. URL <https://pub.towardsai.net/yolo-v5-explained-and-demystified-4e4719891d69>. - abgerufen am 2023-05-08. — Medium
- [RDGF15] REDMON, JOSEPH; DIVVALA, SANTOSH; GIRSHICK, ROSS; FARHADI, ALI: *You Only Look Once: Unified, Real-Time Object Detection*, arXiv (2015)
- [Roev21] ROEVER, SÖNKE: *weiche-dalben-nord-ostsee-kanal*. URL <https://www.blauwasser.de/reviere/nord-ostsee-kanal/attachment/weiche-dalben-nord-ostsee-kanal/>. - abgerufen am 2023-05-10. — Blauwasser.de
- [Sani14] *Sanierung der Hafenummauer beginnt im Oktober*. URL <https://www.wuerzburg24.com/wuerzburg/aktuelle-nachrichten-aus-wuerzburg/sanierung-der-hafenmauer-beginnt-im-oktober/>. - abgerufen am 2023-05-10
- [SiAr19] SINGH, KANWAR BHARAT; ARAT, MUSTAFA ALI: *Deep Learning in the Automotive Industry: Recent Advances and Application Examples*, arXiv (2019)
- [SnFP09] SNIDARO, LAURO; FORESTI, GIAN LUCA; PICIARELLI, CLAUDIO: *Automatic Video Surveillance of Harbour Structures*. In: SHAHBAZIAN, E.; ROGOVA, G.; DEWEERT, M. J. (Hrsg.): *Harbour Protection Through Data Fusion Technologies, NATO Science for Peace and Security Series C: Environmental Security*. Dordrecht: Springer Netherlands, 2009 — ISBN 978-1-4020-8882-7, S. 223–231
- [Sola20] SOLAWETZ, JACOB: *Small Object Detection Guide*. URL <https://blog.roboflow.com/detect-small-objects/>. - abgerufen am 2023-04-07. — Roboflow Blog
- [Soph16] SOPHIST GMBH: *MASTeR - Schablonen für alle Fälle*. URL https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/MASTeR_Broschuere_3-Auflage_interaktiv.pdf - abgerufen am 2023-02-17
- [SpSS20] SPRUAL, RAPHAEL; SOMMER, LARS; SCHUMANN, ARNE: *A comprehensive analysis of modern object detection methods for maritime vessel detection*. In: *Artificial Intelligence and Machine Learning in Defense Applications II*. Bd. 11543, 2020, S. 1–12

- [SSLB22] SHARMA, RABI; SAQIB, MUHAMMAD; LIN, C. T.; BLUMENSTEIN, MICHAEL: A Survey on Object Instance Segmentation. In: *SN Computer Science* Bd. 3 (2022), Nr. 6, S. 499
- [StHM08] STATHEROS, THOMAS; HOWELLS, GARETH; MAIER, KLAUS MCDONALD: Autonomous Ship Collision Avoidance Navigation Concepts, Technologies and Techniques. In: *Journal of Navigation* Bd. 61 (2008), Nr. 1, S. 129–142
- [Stre00] *Streichdalben aus Holz schützen eine Kaimauer Rosskanal im Hamburger Hafen.* URL <https://bildarchiv-hamburg.com/photo/streichdalben-holz-schuetzen-eine-kaimauer-rosskanal-hamburger-hafen-R6FDmCLpeT>. - abgerufen am 2023-05-10
- [SWWD18] SHAO, ZHENFENG; WU, WENJING; WANG, ZHONGYUAN; DU, WAN; LI, CHENGYUAN: SeaShips: A Large-Scale Precisely Annotated Dataset for Ship Detection. In: *IEEE Transactions on Multimedia* Bd. 20 (2018), Nr. 10, S. 2593–2604
- [Tens00] TENSORFLOW: *Data augmentation | TensorFlow Core.* URL https://www.tensorflow.org/tutorials/images/data_augmentation. - abgerufen am 2023-03-07
- [Ultr23a] ULTRALYTICS: *Tips for Best Training Results - YOLOv5 Docs.* URL https://docs.ultralytics.com/yolov5/tips_for_best_training_results/. - abgerufen am 2023-03-28
- [Ultr23b] ULTRALYTICS: *Hyperparameter evolution - Ultralytics YOLOv8 Docs.* URL https://docs.ultralytics.com/yolov5/tutorials/hyperparameter_evolution/. - abgerufen am 2023-04-28
- [Volp19] VOLPI, GONZALO FERREIRO: *Class Imbalance: a classification headache.* URL <https://towardsdatascience.com/class-imbalance-a-classification-headache-1939297ff4a4>. - abgerufen am 2023-03-23. — Medium
- [VSPU17] VASWANI, ASHISH; SHAZEER, NOAM; PARMAR, NIKI; USZKOREIT, JAKOB; JONES, LLION; GOMEZ, AIDAN N.; KAISER, LUKASZ; POLOSUKHIN, ILLIA: Attention Is All You Need, arXiv (2017)
- [WaBL22] WANG, CHIEN-YAO; BOCHKOVSKIY, ALEXEY; LIAO, HONG-YUAN MARK: YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, arXiv (2022)

- [Wang22] WANG, BEINAN: A Parallel Implementation of Computing Mean Average Precision, arXiv (2022)
- [Welc00] *Welcome to Cowes Harbour Commission.* URL https://www.cowesharbourcommission.co.uk/pile_moorings. - abgerufen am 2023-05-10
- [WeMV20] WEERTS, HILDE J. P.; MUELLER, ANDREAS C.; VANSCHOREN, JOAQUIN: Importance of Tuning Hyperparameters of Machine Learning Algorithms, arXiv (2020)
- [Wenn20] WENNKER, PHIL: Künstliche Intelligenz – Eine kurze Geschichte. In: WENNKER, P.: *Künstliche Intelligenz in der Praxis*. Wiesbaden: Springer Fachmedien Wiesbaden, 2020 — ISBN 978-3-658-30479-9, S. 1–8
- [Witt19] WITTPAHL, V. (Hrsg.): *Künstliche Intelligenz: Technologie | Anwendung | Gesellschaft*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019 — ISBN 978-3-662-58041-7
- [WoFM19] WOO, AMI; FIDAN, BARIS; MELEK, WILLIAM W.: Localization for Autonomous Driving. In: ZEKAVAT, S. A. R.; BUEHRER, R. M. (Hrsg.): *Handbook of Position Location*: Wiley, 2019 — ISBN 978-1-119-43461-0, S. 1051–1087
- [XCZW21] XUE, HAOLIN; CHEN, XIANG; ZHANG, RUO; WU, PENG; LI, XUDONG; LIU, YUANCHANG: Deep Learning-Based Maritime Environment Segmentation for Unmanned Surface Vehicles Using Superpixel Algorithms. In: *Journal of Marine Science and Engineering* Bd. 9 (2021), Nr. 12, S. 1329
- [Xin22] XIN, SUN: Application of Deep learning in computer vision. In: *Highlights in Science, Engineering and Technology* Bd. 16 (2022), S. 125–130
- [ZCSG23] ZOU, ZHENGXIA; CHEN, KEYAN; SHI, ZHENWEI; GUO, YUHONG; YE, JIEPING: Object Detection in 20 Years: A Survey. In: *Proceedings of the IEEE* (2023), S. 1–20
- [ZGLZ22] ZHANG, YIHONG; GE, HANG; LIN, QIN; ZHANG, MING; SUN, QIANTAO: Research of Maritime Object Detection Method in Foggy Environment Based on Improved Model SRC-YOLO. In: *Sensors* Bd. 22 (2022), Nr. 20, S. 7786

- [ZMXF22] ZHANG, HAOFEI; MAO, FENG; XUE, MENGQI; FANG, GONGFANG; FENG, ZUNLEI; SONG, JIE; SONG, MINGLI: Knowledge Amalgamation for Object Detection with Transformers (2022)
- [ZSSP21] ZSCHECH, PATRICK; SAGER, CHRISTOPH; SIEBERS, PHILIPP; PERTERMANN, MAIK: Mit Computer Vision zur automatisierten Qualitätssicherung in der industriellen Fertigung: Eine Fallstudie zur Klassifizierung von Fehlern in Solarzellen mittels Elektrolumineszenz-Bildern. In: *HMD-Praxis der Wirtschaftsinformatik* Bd. 58 (2021), Nr. 2, S. 321–342

10 Anhang

In den nachfolgenden Abschnitten A bis C sind die Performance-Metriken abgebildet, die im Rahmen der durchgeführten Experimente erhoben und auf die im Fließtext Bezug genommen wurde.

A Experiment 3: Optimierungsalgorithmus

a Recall

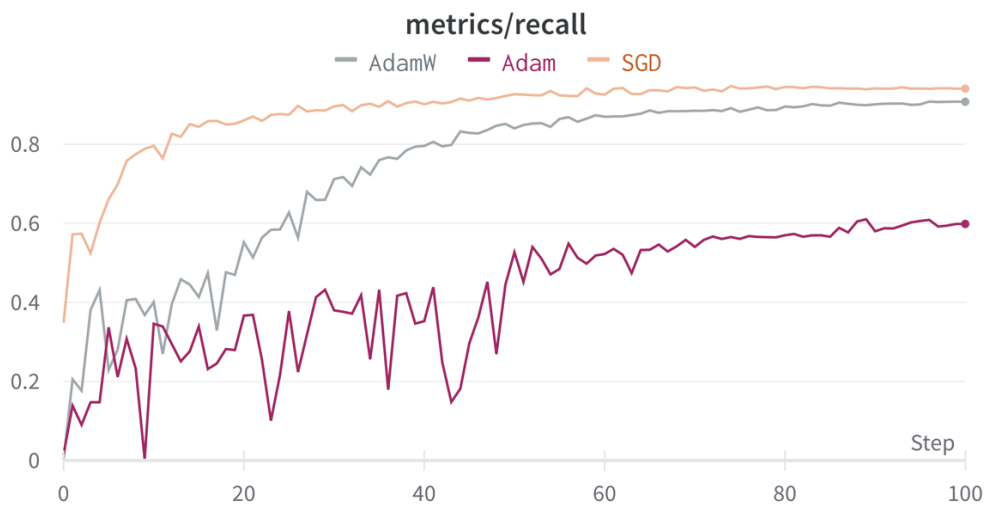


Abbildung 46: Experiment 3: Recall

b Klassenverlust



Abbildung 47: Experiment 3: Klassenverlust im Trainingsset

B Experiment 4: Optimierung der sekundären Hyperparameter

c Klassenverlust



Abbildung 48: Experiment 4: Klassenverlust im Trainingsset

d Precision und Recall

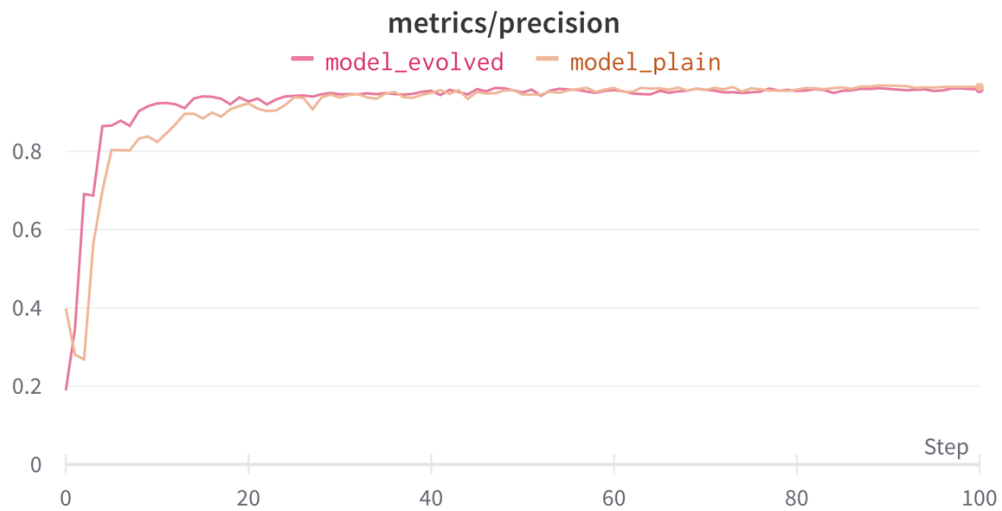


Abbildung 49: Experiment 4: Precision

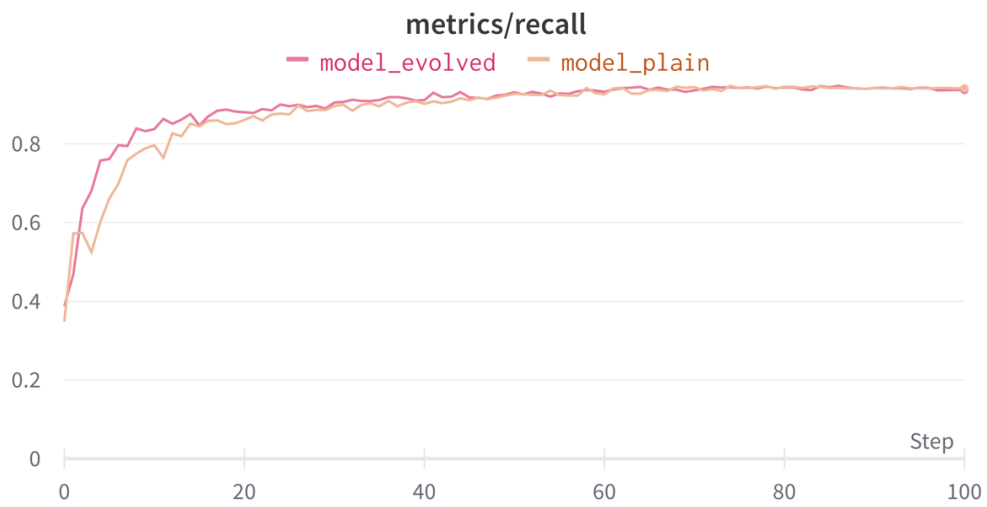


Abbildung 50: Experiment 4: Recall

e Mean Average Precision (0.5:0.95)

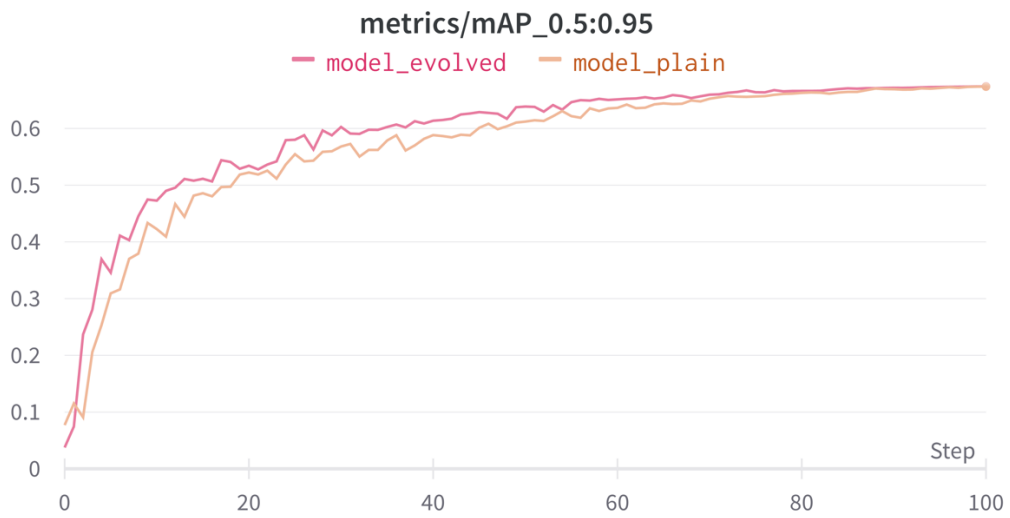


Abbildung 51: Experiment 4: Mean Average Precision (0.5:0.95)

Abschließende Erklärung

Hiermit versichere ich an Eides statt, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Oldenburg, den 23.05.2023

T. Siering