



Fachhochschule Vorarlberg GmbH.
Informatik - Software and Information Engineering

Bachelorarbeit im Fachhochschul-Bachelorstudiengang Informatik - Software and
Information Engineering

Einsatz von Computer Vision zur Ermöglichung eines Schachspiels gegen einen Computer

Ausgeführt von
Clara Tschamon
2110247035

Betreut von
Dipl.-Ing. Dr. techn. Ralph Hoch

Dornbirn, am 20. Mai 2024

Kurzreferat

Einsatz von Computer Vision zur Ermöglichung eines Schachspiels gegen einen Computer

Mit dem stetigen Fortschritt der Rechenleistung von Computern und der Verfügbarkeit umfangreicher Datenmengen hat die Objekterkennung mittels maschinellem Sehen eine revolutionäre Bedeutung erlangt, die die Art und Weise, wie Menschen mit Technologie interagieren, grundlegend transformiert.

Diese Arbeit konzentriert sich auf die Entwicklung eines Bilderkennungssystems zur Identifizierung eines Schachbretts und von Schachfiguren, mit dem Ziel, das Schachspiel auf einem realen Schachbrett gegen einen Computer zu ermöglichen.

Es wird mit einer Erläuterung der allgemeine Funktionsweise von Deep Learning begonnen. Anschließend werden die Objekterkennungsalgorithmen YOLO und Faster R-CNN vorgestellt. Außerdem werden die wichtigsten Metriken erläutert, die zur Bewertung der Leistung von Objekterkennungssystemen verwendet werden.

Die technische Umsetzung gliedert sich in mehrere Schritte. Zunächst wird die Schachbretterkennung mit verschiedenen Techniken der Computer Vision durchgeführt. Anschließend erfolgt die Erkennung der Schachfiguren durch das Trainieren von drei verschiedenen Objekterkennungsalgorithmen. Deren Ergebnisse dieser werden miteinander verglichen. Im Anschluss erfolgt die Zuordnung der Schachfiguren auf das Schachbrett. Abschließend erfolgt die Implementierung der Spiellogik mit Hilfe einer Schachengine.

Zum Schluss werden potenzielle Verbesserungen des Systems diskutiert.

Das Ziel dieser Arbeit, ein funktionierendes Objekterkennungssystem zu erhalten, konnte erreicht werden.

Abstract

Use of computer vision to enable a game of chess against a computer

With the continual advancement of computing power and the availability of large amounts of data, object recognition through machine vision has gained revolutionary significance, fundamentally transforming the way humans interact with technology.

This thesis focuses on the development of an image recognition system for the identification of a chessboard and chess pieces, with the aim of enabling chess to be played on a real chessboard against a computer.

Starting with an explanation of how deep learning works in general, the object recognition algorithms YOLO and Faster R-CNN are then presented. Furthermore the most important metrics used to assess the performance of object recognition systems are explained.

The technical implementation is divided into several steps. First, the chessboard is recognised using various computer vision techniques. Then the chess pieces are recognised by training three different state-of-the-art object recognition algorithms. The results are compared. The chess pieces are then assigned to the chessboard. Finally, the game logic is implemented using a chess engine.

Finally, possible improvements to the system are discussed.

The aim of this work, to obtain a working object recognition system, has been achieved.

Inhaltsverzeichnis

Abbildungsverzeichnis	VI
Abkürzungsverzeichnis	VIII
1 Einleitung	1
1.1 Motivation	1
1.2 Problemstellung	1
1.3 Zielsetzung	2
2 Stand der Technik	3
2.1 Deep Learning	3
2.1.1 Künstliche Intelligenz vs. Machine Learning vs. Deep Learning	3
2.1.2 Deep Learning Funktionsweise	4
2.2 Objekterkennungsalgorithmen	6
2.2.1 You Only Look Once (YOLO)	7
2.2.2 Faster Region-based Convolutional Neural Network (R-CNN) .	10
2.3 Metriken und Losses	13
2.3.1 Precision	13
2.3.2 Recall	14
2.3.3 Precision-Recall-Kurve	14
2.3.4 Average Precision und Mean Average Precision	14
2.3.5 Training Loss und Validation Loss	15
2.3.6 Underfitting	15
2.3.7 Overfitting	16
2.3.8 Generalisierung	16
3 Anforderungen und Methodik	17
3.1 Anforderungen	17

3.2	Methodische Umsetzung	18
3.2.1	Ablauf	18
3.2.2	Architektur	18
3.2.3	Schachbretterkennung	21
3.2.4	Schachfigurenerkennung	22
3.2.5	Zuordnen der Schachfiguren auf das Schachfeld	22
3.2.6	Generierung von nächstem Spielzug	22
4	Technische Umsetzung	23
4.1	Schachbetterkennung mit OpenCV	23
4.1.1	Erkennung von Rahmen	23
4.1.2	Erkennung der Felder	24
4.2	Erkennung der Schachfiguren durch Objekterkennung	26
4.2.1	Auswahl und Vorbereitung des Datensatzes	26
4.2.2	Trainieren mit YOLOv5m Algorithmus	28
4.2.3	Anpassen des Datensatzes	30
4.2.4	Erneutes Trainieren mit YOLOv5m Algorithmus	31
4.2.5	Trainieren mit YOLOv8m Algorithmus	33
4.2.6	Trainieren mit Faster R-CNN Algorithmus	33
4.3	Zuordnen der Schachfiguren auf das Schachbrett	35
4.4	Spiellogik mit python-chess und Stockfish Schach-Engine	35
5	Evaluierung	38
5.1	Vergleich YOLOv5m vs. YOLOv8m	38
5.2	Vergleich YOLO vs. Faster R-CNN	39
5.3	Verbesserungsmöglichkeiten	41
6	Zusammenfassung und Ausblick	42
Literaturverzeichnis		45
Eidesstattliche Erklärung		46

Abbildungsverzeichnis

2.1	Künstliche Intelligenz, Machine Learning, Deep Learning	5
2.2	Convolutional Neural Network (CNN) [1]	5
2.3	Vergleich von Bildklassifikation, Objekterkennung und Instanzsegmentierung [2]	6
2.4	YOLO Funktionsweise [3]	8
2.5	Intersection over Union (IoU) [4]	8
2.6	Vergleich von YOLOv5 und YOLOv8 [5]	10
2.7	Faster Region-based Convolutional Neural Networks (R-CNN) Architektur [6]	11
3.1	Spielablauf	19
3.2	Architektur des Systems	20
4.1	Erkannte Eckpunkte des Schachbretts	24
4.2	Erkannte Kanten	24
4.3	Erkannte Schnittpunkte	25
4.4	Erkannte Felder	25
4.5	Klassenverteilung in Datensatz	27
4.6	Ergebnisse YOLOv5m Versuch 1	29
4.7	Precision-Recall Kurve YOLOv5m Versuch 1	30
4.8	Confusion Matrix YOLOv5m Versuch 1	31
4.9	mAP Vergleich YOLOv5m	32
4.10	Precision und Recall Vergleich YOLOv5m	32
4.11	Losskurven Vergleich YOLOv5m	32
4.12	Zuordnung der Schachfiguren auf das Schachfeld	35
4.13	Schachbrett Anfangsposition	36
4.14	Benutzeroberfläche	37
5.1	Ergebnisse YOLOv8m	38

5.2	YOLOv5m vs. YOLOv8m	39
5.3	Metriken Faster R-CNN	40

Abkürzungsverzeichnis

KI Künstliche Intelligenz

ML Machine Learning

DL Deep Learning

CNN Convolutional Neural Network

YOLO You Only Look Once

NMS Non-Maximum Suppression

IoU Intersection over Union

TP True Positive

FP False Positive

FN False Negative

AP Average Precision

mAP Mean Average Precision

R-CNN Region-based Convolutional Neural Networks

RPN Region Proposal Network

ROI Region of Interest

FEN Forsyth-Edwards Notation

1 Einleitung

Diese Bachelorarbeit beschäftigt sich mit der Computer Vision und Objekterkennung. Die automatische Erkennung von Schachfiguren und Schachbrettern ist ein Beispiel, um die Leistungsfähigkeit von Objekterkennungsalgorithmen zu demonstrieren. Diese Arbeit strebt an, die Leistung und Effektivität von drei führenden Algorithmen für die Objekterkennung zu vergleichen. Darüber hinaus wird die Erkennung des Schachbretts mithilfe von Computer Vision-Techniken umgesetzt.

1.1 Motivation

Die Fortschritte in den Bereichen der Computer Vision und maschinelles Lernen ermöglichen es Maschinen und Systemen, ihre Umgebung wahrzunehmen, zu interpretieren und entsprechend zu agieren. Insbesondere die Erkennung von Objekten in Bildern und Videos hat sich zu einem Schlüsselbereich entwickelt, der eine Vielzahl aufregender und praktischer Anwendungen bietet. Von autonomen Fahrzeugen, die Fußgänger und Verkehrsschilder erkennen, bis hin zur Erkennung von Gesichtern für biometrische Sicherheitssysteme gibt es eine Vielzahl an Möglichkeiten.

Die Unterstützung für eine Mensch-Maschine-Interaktion soll hier am Beispiel des Brettspiels Schach umgesetzt werden. Mit Hilfe von Objekterkennung und Computer Vision soll der Zustand eines Schachbretts ermittelt werden.

1.2 Problemstellung

Der Ablauf des Schachspiels ist wie folgt:

Als erstes wird das Schachbrett mit Hilfe von Techniken der Computer Vision erkannt. Anschließend werden die Schachfiguren aufgestellt. Der menschliche Spieler beginnt mit

dem ersten Zug. Nun folgt die Ermittlung des Spielstands mittels Objekterkennung. Außerdem wird die Gültigkeit des Zugs validiert. Anschließend nimmt der Computer den nächsten Zug vor.

Die Hauptherausforderung dieser Arbeit besteht darin, die einzelnen Schachfiguren mithilfe einer einzelnen Kamera effizient und zuverlässig zu erkennen und das Schachbrett mithilfe von Computer Vision Tools zu detektieren.

Die Erkennung von Schachfiguren stellt eine anspruchsvolle Herausforderung dar, insbesondere in Situationen, in denen die Figuren hinter anderen Schachfiguren auf einem voll besetzten Schachbrett teilweise verborgen sind. Die genaue Identifizierung jeder einzelnen Figur ist unerlässlich, ebenso wie die präzise Erfassung des Schachbretts selbst. Nur so kann eine exakte Darstellung des aktuellen Zustands des Schachbretts sichergestellt werden.

Die Wahl des geeigneten Modells zur Objekterkennung und die effiziente Umsetzung der Schachbretterkennung sind von entscheidender Bedeutung, da sie sich unmittelbar auf das Benutzererlebnis auswirken.

1.3 Zielsetzung

Die Zielsetzung dieser Arbeit ist, Methoden zur Objekterkennung für die Erkennung von Schachfiguren und Schachbrettern zu erforschen, zu implementieren und zu bewerten. Der Schwerpunkt liegt auf der Bewertung der Leistung und Eignung von YOLO-basierten Modellen (YOLOv5 und YOLOv8) im Vergleich zu Faster R-CNN. Mittels eines Datensatzes von Roboflow¹ werden die Modelle trainiert und optimiert, um eine zuverlässige und effiziente Objekterkennung zu erzielen. Darüber hinaus zielt die Arbeit darauf ab, einen Mechanismus zur Schachbretterkennung auf der Grundlage von OpenCV zu implementieren.

Das Ergebnis dieser Arbeit ist die Entwicklung eines funktionsfähigen Systems zur Objekterkennung, das es einem Menschen ermöglicht, Schach auf einem realen Schachbrett gegen einen Computer zu spielen. Dieses System sollte robust sein, um ein reibungsloses Spielerlebnis zu ermöglichen.

¹<https://universe.roboflow.com/deneme-iq931/chessv1-sswl9>

2 Stand der Technik

Dieser Abschnitt behandelt den Stand der Technik im Bereich von Deep Learning und Objekterkennungsalgorithmen. Zunächst werden die Grundlagen des Deep Learning behandelt, gefolgt von einer detaillierten Betrachtung der Funktionsweise von Objekterkennungsalgorithmen, insbesondere You Only Look Once (YOLO) und Faster R-CNN. Dabei werden auch die Unterschiede zwischen verschiedenen YOLO-Algorithmusversionen beleuchtet. Darüber hinaus werden Metriken vorgestellt, um die Trainingsergebnisse von Objekterkennungsalgorithmen zu evaluieren und zu bewerten.

2.1 Deep Learning

In diesem Abschnitt werden die grundlegenden Begriffe des Deep Learnings erläutert.

2.1.1 Künstliche Intelligenz vs. Machine Learning vs. Deep Learning

Da die Begriffe Künstliche Intelligenz (KI), Machine Learning (ML), Deep Learning (DL) und neuronale Netzwerke oft miteinander vermischt werden, entsteht häufig Verwirrung über ihre jeweiligen Unterschiede.

- Künstliche Intelligenz (KI)

KI ist der umfassendste Begriff und bezieht sich auf Maschinen, die menschliche Intelligenz und kognitive Funktionen nachahmen, um komplexe Aufgaben zu lösen. KI nutzt Vorhersagen und Automatisierung, um Aufgaben wie Gesichts- und Spracherkennung, Entscheidungsfindung und Übersetzung zu optimieren. KI umfasst verschiedene Kategorien, darunter Artificial Narrow Intelligence (ANI), Artificial General Intelligence (AGI) und Artificial Super Intelligence

(ASI). ANI bezieht sich auf „schwache“ KI, die spezifische Aufgaben ausführen kann, während AGI und ASI als „starke“ KI eingestuft werden, die menschliche Verhaltensweisen besser nachahmen können. [7]

- Machine Learning (ML)

Machine Learning (ML) ist eine Teilmenge der KI, die darauf abzielt, Vorhersagen zu optimieren und Fehler zu minimieren. Im klassischen oder „nicht-tiefen“ ML erfordert der Prozess menschliches Eingreifen, um Muster zu identifizieren und spezifische Aufgaben auszuführen. Es gibt verschiedene Arten von ML, darunter *überwachtes Lernen (Supervised Learning)*, bei dem gelabelte Daten verwendet werden, und *unüberwachtes Lernen (Unsupervised Learning)*, bei dem unstrukturierte Daten genutzt werden. Darüber hinaus gibt es verstärkendes Lernen (Reinforcement Learning) und Online-Lernen, bei denen Modelle durch Interaktion mit der Umgebung und Feedback kontinuierlich verbessert werden. [7]

Beispiele für ML-Anwendungen sind Email-Spamfilter und E-Commerce Empfehlungssysteme. [8]

- Deep Learning (DL)

Deep Learning ist eine Unterdisziplin des Machine Learning, die sich durch tiefe neuronale Netzwerke auszeichnet. Diese Netzwerke automatisieren den Feature-Extraktionsprozess und benötigen weniger manuelle Eingriffe. [7] Deep-Learning-Algorithmen eignen sich besonders gut für Aufgaben, die eine präzise Mustererkennung erfordern, wie beispielsweise die Klassifizierung von Bildern und die Erkennung von Objekten. [9]

Wie Abbildung 2.1 zeigt, kann zusammengefasst werden, dass Künstliche Intelligenz den umfassendsten Begriff darstellt, Machine Learning eine Teilmenge davon ist und Deep Learning eine spezifische Methode innerhalb von Machine Learning darstellt, die tiefe neuronale Netzwerke verwendet. [7]

2.1.2 Deep Learning Funktionsweise

Oft werden die Begriffe Deep Learning und neuronale Netzwerke synonym verwendet, was zu Verwirrung führen kann. Der Begriff „deep“ in Deep Learning bezieht sich

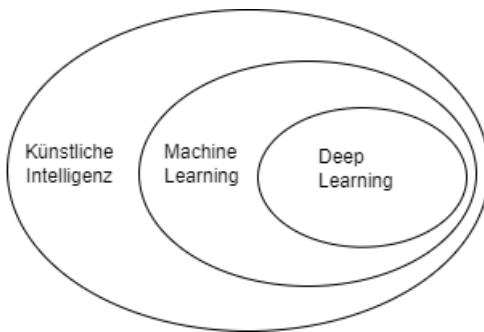


Abbildung 2.1: Künstliche Intelligenz, Machine Learning, Deep Learning

lediglich auf die Tiefe der Schichten in einem neuronalen Netzwerk.

Ein Deep-Learning-Algorithmus bezeichnet ein neuronales Netzwerk, das über mehr als drei Schichten von miteinander verbundenen Knoten verfügt. Die Ein- und Ausgangsschicht mit eingeschlossen. [1]

Abbildung 2.2 zeigt ein Convolutional Neural Network (CNN) mit 5 Schichten.

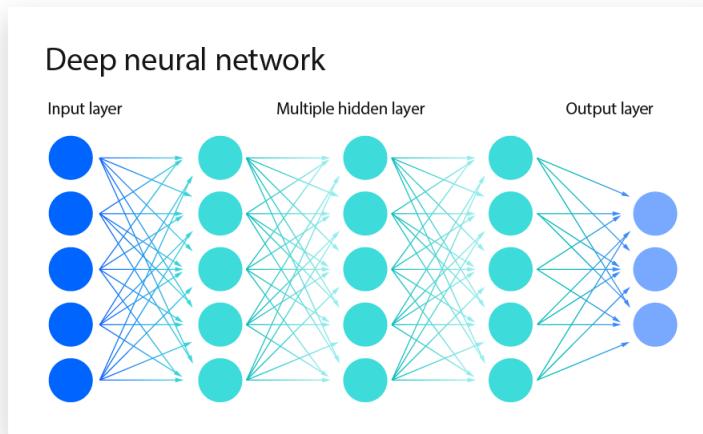


Abbildung 2.2: Convolutional Neural Network (CNN) [1]

Jeder Knoten wird trainiert, eine spezifische Eigenschaft der Daten zu erlernen. In einem Beispiel zur Bilderkennung könnte die erste Schicht lernen, Kanten zu identifizieren, die zweite Schicht Formen und die dritte Schicht, komplexere Merkmale Objekten zuzuordnen. Während das Netzwerk lernt, werden die Gewichtungen der Verbindungen zwischen den Knoten angepasst, um die Daten besser klassifizieren zu können. Dieser Vorgang nennt sich Training. [9]

In Abbildung 2.3 ist ein Vergleich von verschiedenen Aufgaben der Bilderkennung abgebildet. [2]

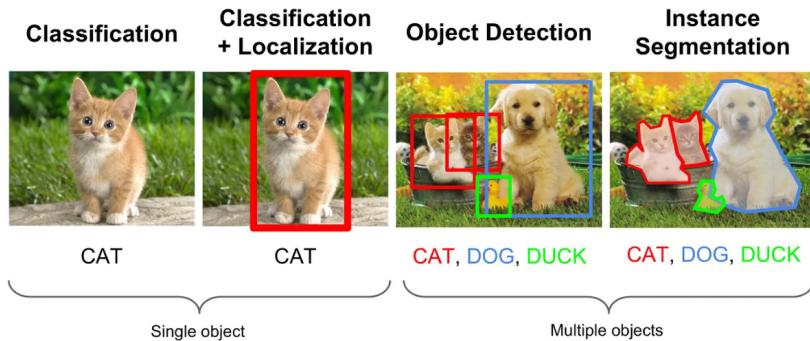


Abbildung 2.3: Vergleich von Bildklassifikation, Objekterkennung und Instanzsegmentierung [2]

2.2 Objekterkennungsalgorithmen

Unter Objekterkennung wird das Identifizieren von Objekten und deren Positionen in einem Bild verstanden. Ein Objekterkennungsmodell gibt die Koordinaten der erkannten Objekte in einem Bild als sogenannte „Bounding Boxes“ zurück. Außerdem gibt das System für jede Bounding Box einen „Confidence Score“ aus, welcher zeigt, wie sicher sich das System ist, dass die Vorhersage korrekt ist. [10]

Objekterkennungsalgorithmen erlernen die Fähigkeit, Objekte in Bildern zu identifizieren, indem sie mit beschrifteten Bildern arbeiten. Diese Beschriftungen zeigen, wo sich die zu identifizierenden Objekte im Bild befinden. Das Ergebnis dieses Trainingsprozesses ist ein Objekterkennungsmodell. [10]

Für das Training existieren verschiedene Algorithmen, die zur Erstellung von Objekterkennungsmodellen verwendet werden können. [10]

In den letzten Jahren haben zwei Architekturen die Objekterkennung maßgeblich maßgeblich beeinflusst: CNNs und das You Only Look Once (YOLO)-Modell. [10]

Objekterkennungsalgorithmen können basierend darauf, wie oft das Eingangsbild durch das CNN läuft, in die folgenden zwei Hauptkategorien unterteilt werden.

Ein-Stufen-Erkennung (One-stage Detection)

Bei der Ein-Stufen-Erkennung erfolgt die Objekterkennung in einem einzigen Durchlauf des Eingangsbildes. Das bedeutet, dass der Algorithmus das gesamte Bild auf einmal verarbeitet, um Vorhersagen über das Vorhandensein und die Position der Objekte im Bild zu treffen. Diese Methode ist in der Regel rechnerisch effizient und zeiteffizient, da nur ein Durchlauf des Bildes erforderlich ist. Sie eignet sich daher gut für Echtzeitanwendungen.

Ein Nachteil der Ein-Stufen-Erkennung ist, dass sie oft weniger genau ist und Schwierigkeiten bei der Erkennung kleiner Objekte haben kann. [11]

Zwei-Stufen-Erkennung (Two-stage Detection)

Bei der Zwei-Stufen-Erkennung erfolgt die Objekterkennung in zwei Durchläufen durch das Neuronale Netz. Der erste Durchlauf generiert eine Reihe von Vorschlägen oder potenziellen Objektpositionen und der zweite Durchlauf verfeinert diese Vorschläge und trifft endgültige Vorhersagen.

Dieser Ansatz ist in der Regel genauer als die Ein-Stufen-Erkennung, erfordert jedoch auch mehr Rechenleistung. [11]

2.2.1 You Only Look Once (YOLO)

YOLO wurde 2016 von Joseph Redmon, Santosh Divvala, Ross Girshick, und Ali Farhadi veröffentlicht. [3] YOLO war einer der ersten One-Stage Object Detection Algorithmen. Durch die kontinuierliche Weiterentwicklung von YOLO gibt es mittlerweile über 10 Versionen. [12] YOLOv5 und YOLOv8 sind zwei der neuesten und genauesten und wurden beide von der Firma Ultralytics¹ entwickelt.

Bei der Objekterkennung mit YOLO wird, wie Abbildung 2.4 zeigt, zuerst das Bild in einen NxN-Raster unterteilt. Jede Zelle im Raster ist dafür verantwortlich Objekte in ihrem Bereich zu lokalisieren und deren Klasse sowie die Wahrscheinlichkeit vorherzusagen. Dies erfolgt durch sogenannte "Residual Blocks". [13]

Im nächsten Schritt werden *Bounding Boxes* erzeugt, die alle Objekte im Bild umrahmen. Für jedes Objekt können mehrere Bounding Boxes erstellt werden. YOLO verwendet eine einzelne Regressionskomponente um die Attribute dieser Bounding Boxes zu bestimmen. Diese Attribute umfassen die Wahrscheinlichkeit, dass sich ein

¹<https://www.ultralytics.com/>

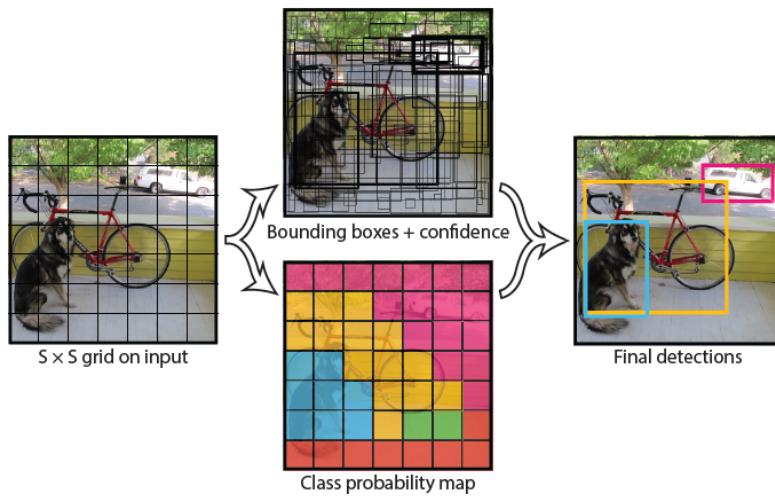


Abbildung 2.4: YOLO Funktionsweise [3]

Objekt in der Zelle befindet (pc), die x- und y-Koordinaten der Bounding Box im Verhältnis zur Zelle (bx, by), die Höhe und Breite der Bounding Box im Verhältnis zur Zelle (bh, bw) sowie die Klasse des erkannten Objekts. [13]

Ein Problem, das bei der Vorhersage von Objekten auftreten kann, ist, dass der Algorithmus mehrere Bounding Boxes für eine Klasse vorhersagt. Um dieses Problem zu lösen, wird ein Non-Maximum Suppression (NMS)-Algorithmus verwendet. Zunächst wird die Bounding Box mit der höchsten Wahrscheinlichkeit ausgewählt und mit allen anderen Bounding Boxes dieser bestimmten Klasse unter Verwendung der Intersection over Union (IoU) verglichen. Wie Abbildung 2.5 zeigt, ist die IoU das Verhältnis der Fläche des Schnitts zwischen zwei Bounding Boxes zur Fläche der Vereinigung der beiden Bounding Boxes.

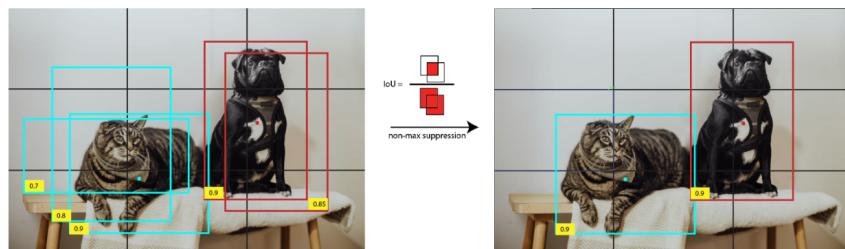


Abbildung 2.5: Intersection over Union (IoU) [4]

Wenn die IoU höher als der vordefinierte Schwellenwert ist, wird angenommen, dass

sie dasselbe Objekt anzeigen und in Folge wird die Bounding Box mit der geringeren Wahrscheinlichkeit ausgeschlossen. [4]

Vergleich von YOLOv5 und YOLOv8

YOLOv5 wurde 2020 von der Firma Ultralytics als Open Source Projekt veröffentlicht und ist in PyTorch implementiert. [14]

YOLOv8 ist eine Weiterentwicklung von YOLOv5 und wurde im Jänner 2023 ebenfalls von der Firma Ultralytics als Open Source Projekt veröffentlicht.

Ein wesentlicher Unterschied von YOLOv8 zu YOLOv5 ist die Anchor-Free Detection, bei der das Modell direkt das Zentrum eines Objekts vorhersagt, anstatt den Offset von einer bekannten Anchor-Box.

Anchor-Free Detection reduziert die Anzahl der Box-Vorhersagen, was den Non-Maximum Suppression (NMS) Schritt beschleunigt, welcher eine aufwendige Nachbearbeitung ist, um Kandidaten-Erkennungen nach der Inferenz zu filtern.

Beide Ansätze verwenden Mosaik Augmentierung. Die Mosaikdatenaugmentierung ist eine Technik, bei der vier Bilder zu einem einzigen Mosaikbild kombiniert werden. Dies zwingt das Modell, Objekte an neuen Positionen, in teilweiser Verdeckung und gegen verschiedene umgebende Pixel zu erkennen. Es wurde jedoch empirisch gezeigt, dass diese Augmentierung die Leistung beeinträchtigt wenn sie während des gesamten Trainingsvorgangs durchgeführt wird. Es ist vorteilhaft, sie für die letzten zehn Trainingsepochen auszuschalten, wie in YOLOv8 umgesetzt.

[15]

Sowohl YOLOv5 als auch YOLOv8 umfassen fünf verschiedene Modellgrößen: nano (n), small (s), medium (m), large (l) und extra large (x).

Wie Abbildung 2.6 zeigt, bietet YOLOv8 33% mehr Mean Average Precision (mAP) für nano-modelle im Vergleich zu YOLOv5 und insgesamt für alle Modelle eine höhere mAP.

Größere Modelle benötigen mehr Zeit für die Inferenz, bieten jedoch eine höhere mAP.

Größere Modelle sind besser, wenn weniger Daten zum Training verfügbar sind und kleinere Modelle sind die bessere Wahl, wenn wenig Speicherplatz vorhanden ist.

[16]

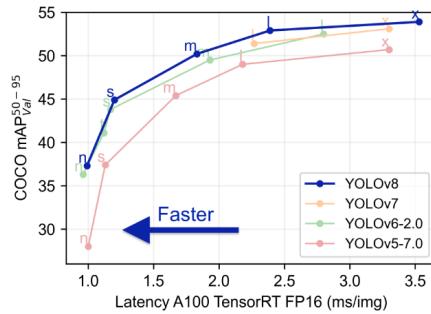


Abbildung 2.6: Vergleich von YOLOv5 und YOLOv8 [5]

2.2.2 Faster Region-based Convolutional Neural Network (R-CNN)

Faster Region-based Convolutional Neural Networks (R-CNN), ist eine Architektur zur Objekterkennung aus der R-CNN-Familie, die 2015 von Shaoqing Ren, Kaiming He, Ross B. Girshick und Jian Sun vorgestellt wurde. [17]

Faster R-CNN ist ein Two-stage detector und besteht aus zwei Komponenten.

1. Region Proposal Network (RPN)
2. Fast R-CNN detector

Wie Abbildung 2.7 zeigt, ist der Vorteil der Architektur die gemeinsame Nutzung der CNN-Schichten sowohl durch das Region Proposal Network (RPN) als auch durch den anschließenden Fast R-CNN Detektor. Dieses Teilen reduziert signifikant die Rechenzeit und den Speicherverbrauch, da die Berechnungen für die Generierung von Regionenvorschlägen und die Objekterkennung wiederverwendet werden kann. [6]

Der geteilte CNN Backbone verarbeitet Eingangsbilder und extrahiert hierarchische Merkmale. Diese Merkmale reichen von niedrigen Details wie Kanten und Texturen in den frühen Schichten, bis zu hochrangigen semantischen Informationen wie Objektteilen und Formen in den tieferen Schichten. Sie werden Feature Maps genannt. [6]

Stage 1: Region Proposal Network (RPN)

Das RPN ist eine entscheidende Komponente, die dafür verantwortlich ist, potenzielle regions of interest (Regionenvorschläge) in Bildern vorzuschlagen. Es verwendet einen Sliding-Window-Ansatz auf den Feature Maps, die aus dem geteilten CNN-Backbone

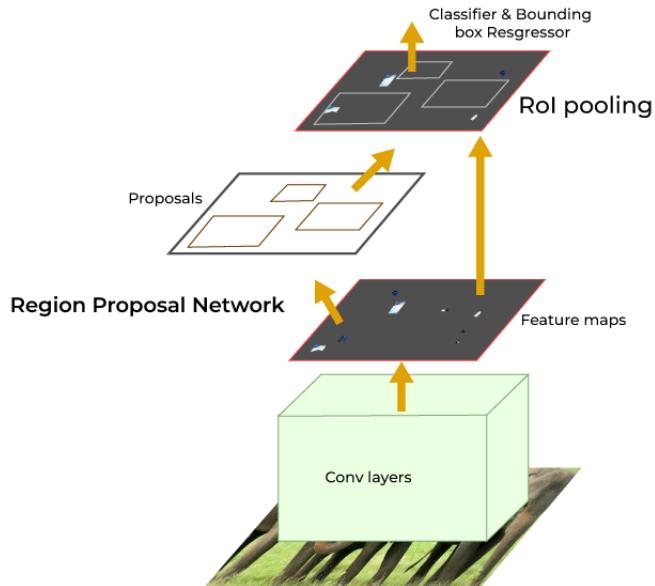


Abbildung 2.7: Faster R-CNN Architektur [6]

abgeleitet sind. [6]

Die Hauptkomponenten sind:

- Anchor Boxes

Dies sind vordefinierte Boxen mit verschiedenen Skalen und Seitenverhältnissen. Platziert an verschiedenen Positionen auf den Feature Maps dienen Anchor Boxes als potenzielle Objektstandorte.

- Sliding Window-Mechanismus

Das RPN fungiert als Sliding-Window-Mechanismus über den Feature Maps. Ein kleines CNN verarbeitet Merkmale innerhalb des Rezeptionsbereichs des Sliding Windows und generiert Scores. Diese Scores zeigen die Wahrscheinlichkeit des Vorhandenseins eines Objekts an und dienen zur Anpassung der Anchor Boxes.

- Objectness Score

Der Objekt-Wahrscheinlichkeitsscore des RPN sagt für jeden Anchor die Wahrscheinlichkeit voraus, dass das Anchor ein Objekt enthält. Dieser Score ist

entscheidend für die Klassifizierung von Anchors als positiv (Objekt) oder negativ (Hintergrund) während des Trainings.

- Intersection over Union (IoU)

Die IoU ist eine Metrik zur Messung der Überlappung zwischen zwei Bounding Boxes und führt die Auswahl genauer Regionenvorschläge.

- Non-Maximum Suppression (NMS)

NMS entfernt Redundanz, indem es die genauesten Vorschläge aufgrund der Objektivitätsscores auswählt und sicherstellt, dass die endgültigen Vorschläge nicht überlappen und somit pro Objekt nur eine Bounding Box erkannt wird. [6]

Stage 2: Fast-R-CNN Detector

Der Fast R-CNN Detektor verarbeitet die Regionenvorschläge des RPN. Seine Funktionsweise kann in mehrere Schritte unterteilt werden:

- Region of Interest (ROI) Pooling

Der ROI Pooling-Schritt transformiert variabel große Regionenvorschläge in Feature Maps, die in nachfolgende Schichten eingespeist werden können. Dies beinhaltet die Aufteilung jedes Regionenvorschlags in ein Raster und die Anwendung von Max-Pooling innerhalb jeder Zelle.

- Merkmalsextraktion

Die durch ROI-Pooling generierten Feature Maps werden durch den geteilten CNN-Backbone geschleust, um aussagekräftige Merkmale zu extrahieren, die spezifisch für das erkannte Objekt sind.

- Fully Connected Layers

Die ROI-gepoolten und featureextrahierten Regionen durchlaufen vollständig die fully connected layers, die für die Objektklassifizierung und die Regression der Bounding boxes verantwortlich sind.

- Objektklassifikation

Das Netzwerk sagt die Klassenwahrscheinlichkeiten für jeden Regionenvorschlag vorher.

- Bounding Box Regression

Zusätzlich zu den Klassenwahrscheinlichkeiten prognostiziert das Netzwerk Anpassungen und Verfeinerungen der Bounding Box für jeden Regionenvorschlag, um sie genauer und besser auf die tatsächlichen Objektgrenzen auszurichten.

- Multitasking-Verlustfunktion

Eine Multitasking-Verlustfunktion kombiniert Klassifikations- und Regressionsverluste und leitet den training Prozess.

- Nachbearbeitung

Die endgültigen Erkennungsergebnisse durchlaufen die Nachbearbeitung, einschließlich Non-Maximum Suppression (NMS), um die Vorschläge zu verfeinern und die zuverlässigsten und nicht überlappenden Erkennungen auszuwählen.

[6]

2.3 Metriken und Losses

In diesem Abschnitt werden grundlegende Begriffe erläutert, die zur Evaluierung und zum Vergleich von Objekterkennungsmodellen verwendet werden.

2.3.1 Precision

Die Precision misst den Anteil der korrekten positiven Vorhersagen (True Positive (TP)) im Verhältnis zu allen positiven Vorhersagen des Modells. Sie wird als Verhältnis der wahren positiven Ergebnisse (TP) zur Summe aus wahren positiven (TP) und falschen positiven (False Positive (FP)) Ergebnissen berechnet. Eine hohe Precision deutet darauf hin, dass das Modell, wenn es ein positives Ergebnis vorhersagt, wahrscheinlich korrekt ist. [18]

$$Precision = \frac{TP}{(TP + FP)}$$

2.3.2 Recall

Der Recall misst den Anteil der wahren positiven Vorhersagen (TP) im Verhältnis zu allen tatsächlich positiven Instanzen im Datensatz. Er wird als Verhältnis der wahren positiven Ergebnisse zur Summe aus wahren positiven (TP) und falschen negativen (False Negative (FN)) Ergebnissen berechnet. Ein hoher Recall zeigt an, dass das Modell in der Lage ist, die meisten tatsächlichen positiven Instanzen im Datensatz zu identifizieren. [18]

$$Recall = \frac{TP}{(TP + FN)}$$

2.3.3 Precision-Recall-Kurve

Die Beziehung zwischen Precision und Recall wird oft mit einer Precision-Recall-Kurve visualisiert, die den Kompromiss zwischen den beiden Metriken bei verschiedenen Konfidenzschwellen zeigt. Anhand der Precision-Recall-Kurve wird der Schwellwert für die Erkennungen gewählt. [18]

Der F1 Score ist eine Metrik, welche die Genauigkeit eines Modells misst. Er kombiniert die Precision und Recall Werte des Modells und wird wie folgt berechnet. [19]

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

2.3.4 Average Precision und Mean Average Precision

Die Mean Average Precision (mAP) ist ein einzelner Skalarwert, der die Precision-Recall-Kurve zusammenfasst. Sie wird als Fläche unter der Precision-Recall-Kurve berechnet und repräsentiert die Fähigkeit des Modells, genaue und robuste Vorhersagen bei verschiedenen Konfidenzlevels zu treffen.

Die AP misst die Präzision für eine bestimmte Klasse. Die mAP ist der Durchschnittswert der Average Precision (AP) über mehrere Klassen oder Kategorien in einem Datensatz. Sie wird häufig verwendet, um die Gesamtleistung von Objekterkennungsmodellen über verschiedene Objektklassen hinweg zu bewerten. Um die mAP zu erhalten, wird der Durchschnitt der AP-Werte herangezogen.

Varianten von mAP:

- mAP_0.5: Der IoU-Schwellenwert wird auf 0.5 gesetzt.
- mAP_0.5:0.95 Diese Metrik berechnet die Durchschnittsgenauigkeit mit verschiedenen Schwellenwerten für die Intersection over Union (IoU) im Bereich von 0.5 bis 0.95 und bildet den Durchschnitt der berechneten Ergebnisse.

[18]

2.3.5 Training Loss und Validation Loss

Der Begriff „Verlust“, hier *Loss* genannt, gibt an, in welchem Ausmaß das Modell von der korrekten Vorhersage abweicht. Wenn der Loss hoch ist, bedeutet das, dass das Modell wahrscheinlich viele Fehler in seinen Vorhersagen hat. Ein niedriger Loss zeigt dagegen an, dass das Modell weniger Fehler macht. [20]

- Training Loss

Der Training Loss ist eine Metrik, die dazu dient, zu beurteilen, wie viele Fehler das Deep Learning Modell auf den Trainingsdatensatz macht. [20]

- Validation Loss

Anhand der Metrik Validation Loss kann bewertet werden, wie hoch die Fehlerquote auf das Validierungsset ist. [20] Das Validierungsset ist eine separate Datensammlung neben dem Trainingsset, die dazu dient, die Leistung des Modells während des Trainings zu überprüfen. [21] Das Modell lernt nicht von diesen Daten, weswegen die Bewertung des Modells objektiv und unvoreingenommen ist und eine zuverlässige Auswertung ermöglicht. [22]

2.3.6 Underfitting

Underfitting tritt auf, wenn das Modell nicht in der Lage ist, die Trainingsdaten genau abzubilden und daher viele Fehler macht.

Wenn sowohl der Validation Loss, als auch der Training Loss hoch sind, weist dies darauf hin, dass das Modell underfitted ist.

Mögliche Gründe dafür könnten unter anderem eine zu geringe Anzahl von Trainingsiterationen oder eine unzureichende Menge an input features oder Daten sein. [23]

2.3.7 Overfitting

Overfitting tritt auf, wenn das maschinelle Lernmodell das Muster in den Trainingsdaten so detailliert erlernt, dass es Schwierigkeiten hat, neue Daten korrekt zu klassifizieren. [21]

Ein Anzeichen dafür ist ein niedriger Training Loss und ein hoher Validation Loss. [20]

Um dies zu vermeiden, kann beispielsweise der Trainingsprozess frühzeitig gestoppt werden - Early Stopping. Dadurch wird verhindert, dass das Modell während des Trainings zu stark auf die Traingsdaten spezialisiert wird und dadurch Muster erkennt, die spezifisch für die Trainingsdaten, aber nicht für neue, bisher ungesehene Daten sind. [24]

2.3.8 Generalisierung

Generalisierung beschreibt Fähigkeit eines Modells, korrekte Vorhersagen für neue, zuvor nicht gesehene Daten zu treffen. Ein Modell, das gut generalisiert, steht im Gegensatz zu einem Modell, das overfitted ist. [25]

3 Anforderungen und Methodik

In den folgenden Abschnitten Anforderungen und Methodische Umsetzung fokussiert sich die Arbeit nun auf die Planung und Umsetzung des zu entwicklenden Systems zur Erkennung des Spielzustands eines Schachbretts und der Generierung von Spielzügen.

3.1 Anforderungen

- Die Koordinaten der Eckpunkte der Schachfelder müssen ermittelt werden. Dies kann einfacheitshalber zu Beginn des Spiels erfolgen, noch bevor die Schachfiguren aufgestellt werden.
Es kann davon ausgegangen werden, dass die Kamera immer auf der rechten Seite des weißen Spielers aufgestellt wird und, dass das Schachbrett auf einem weißen Untergrund liegt.
- Die Schachfiguren müssen mit hoher Genauigkeit ermittelt werden.
- Jeder Schachfigur muss ein Schachfeld zugeordnet werden.
- Zum Zweck einer Zweitkontrolle, ob der Spielzustand richtig erkannt wurde, soll das erkannte Schachbrett in einem User Interface dargestellt werden.
- Aus dem Ergebnis der Schachfiguren- und Schachbretterkennung muss ein Forsyth-Edwards Notation (FEN)-String erstellt werden. Mit Hilfe diesem muss die Überprüfung des Spielzugs und die Feststellung von Schach und Schachmatt erfolgen.
- Ereignisse wie Schach, Schachmatt, ungültige Aufstellung und ungültiger Spielzug sollen dem/der Benutzer*in dargestellt werden.
- Es muss eine Generierung des Spielzugs des Computers vorgenommen werden.

- Der Einfachheit halber sollte die Farbe Schwarz immer vom Computer gespielt werden.
- Die Wahl des Schwierigkeitslevels ist in einem User Interface zu implementieren.
- Die Kommunikation des Frontends mit dem Backend soll über ein REST-Interface erfolgen.

3.2 Methodische Umsetzung

Dieser Abschnitt gibt einen Überblick über die Architektur des Systems und die Vorgehensweise der Umsetzung.

3.2.1 Ablauf

Abbildung 3.1 zeigt den Spielablauf.

3.2.2 Architektur

In Abbildung 3.2 ist die Architektur des Systems dargestellt. Das aufgenommene Bild wird über eine REST-Schnittstelle an das System gesendet, welches die Bildverarbeitung und Objekterkennung durchführt und die Generierung des nächsten Schachzugs des Computers vornimmt.

Die Implementierung der verschiedenen Anforderungen wird auf vier logische Blöcke aufgeteilt:

Chessboard Detector

- **Eingabe:** Ein Bild von dem Schachbrett ohne Schachfiguren
- **Rückgabe:** Die Koordinaten der Eckpunkte jedes Schachfeldes

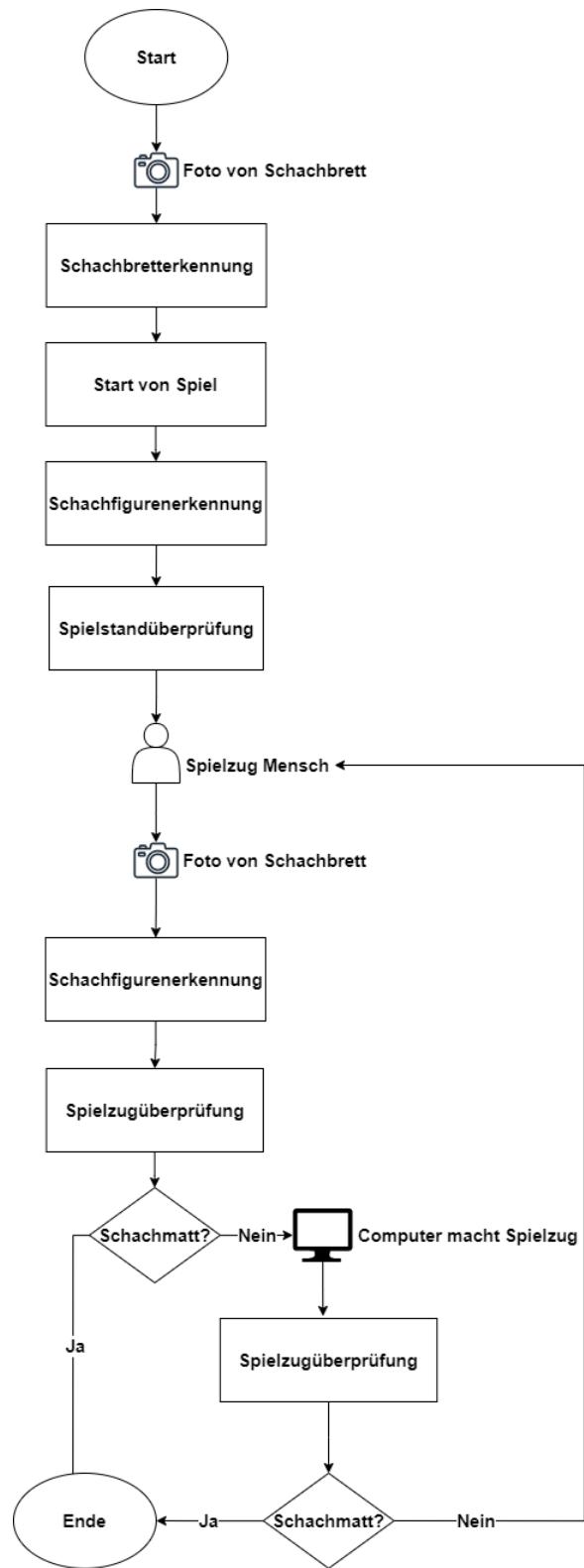


Abbildung 3.1: Spielablauf

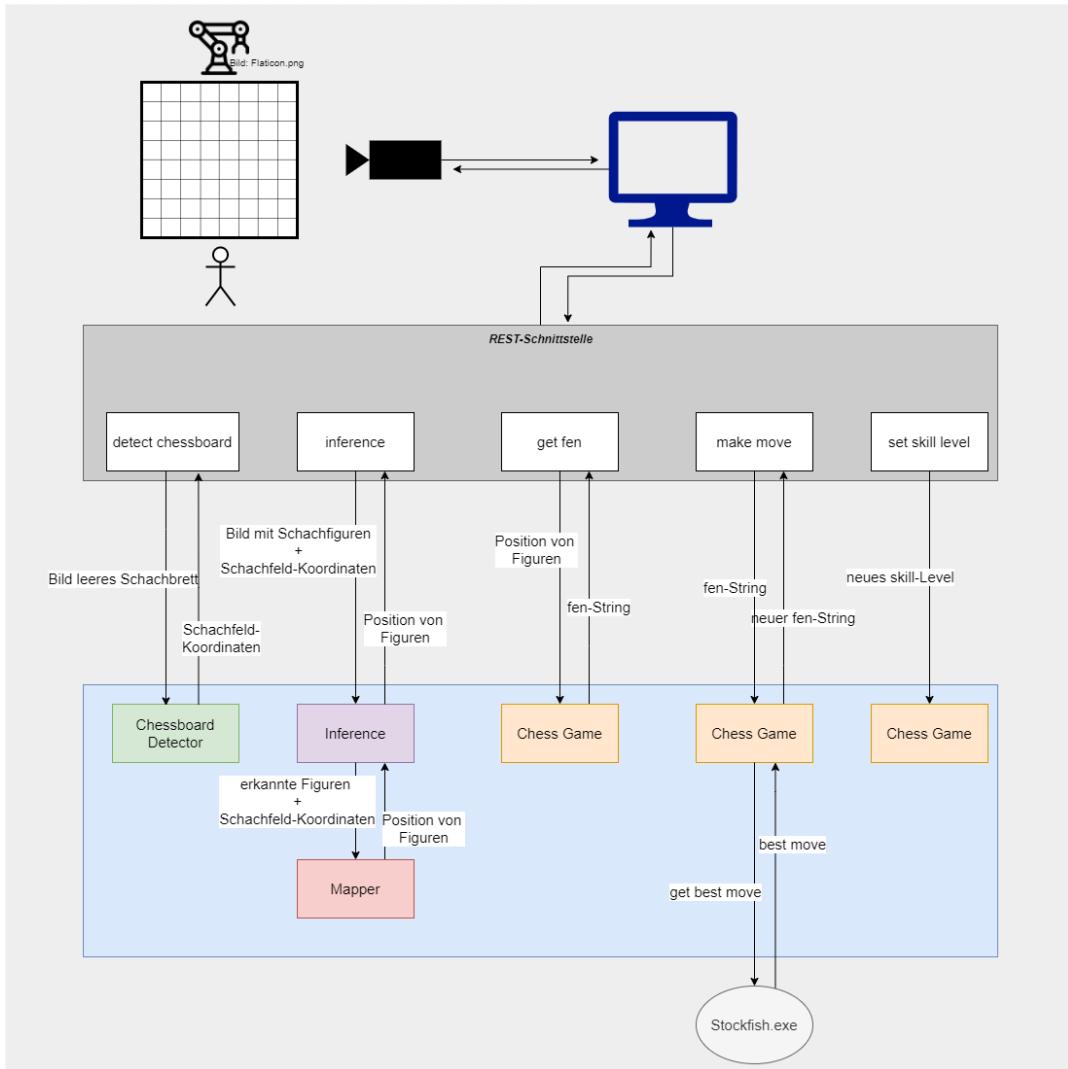


Abbildung 3.2: Architektur des Systems

Inference

- **Eingabe:** Ein Bild des Schachbretts mit den platzierten Schachfiguren. Die Schachfiguren werden identifiziert, was als Inference bezeichnet wird. Anschließend werden die Bounding Box Koordinaten der erkannten Objekte an den Mapper übergeben.
- **Rückgabe:** Die Rückgabe des Mappers.

Mapper

- **Eingabe:** Die Bounding Box Koordinaten der bei der Inference erkannten Objekte
- **Rückgabe:** Die Information, welche Schachfigur auf welchem Feld steht.

Chess Game

Das Chess Game hat verschiedene Funktionen. Beispielsweise kann durch Eingabe eines Forsyth-Edwards Notation (FEN)-Strings überprüft werden, ob die Aufstellung der Figuren auf dem Schachbrett gültig ist, ob der Spielzug gültig war, oder, ob ein/e Spieler*in Schach oder Schachmatt ist. Außerdem kann über das Chess Game der Schwierigkeitsgrad eingestellt werden.

Die FEN Notation ist eine standardisierte Schachnotation. Sie dient dazu, Schachpositionen in einer einzigen Textzeile zu beschreiben. [26]

3.2.3 Schachbretterkennung

Die Schachbretterkennung wird durch die Anwendung verschiedener Bildverarbeitungsschritte mithilfe von Techniken der Computer Vision ermöglicht.

Die eingesetzte Kernmethode ist die Hough Line Transform für die Erkennung von Linien in einem Bild. Diese können dann in horizontale und vertikale Linien eingeteilt werden, um ihre Schnittpunkte zu bestimmen. So können die Koordinaten der einzelnen Schachfelder ermittelt werden.

3.2.4 Schachfigurenerkennung

Für die Auswahl des effektivsten Objekterkennungsalgorithmus werden drei der aktuell führenden Objekterkennungsalgorithmen auf den selben Datensatz trainiert und ihre Ergebnismetriken verglichen. Das Modell, das die besten Metriken aufweist, wird im System eingesetzt.

3.2.5 Zuordnen der Schachfiguren auf das Schachfeld

Für die Zuordnung der erkannten Schachfiguren auf das Schachfeld werden die Koordinaten der Bounding Boxes verwendet und überprüft, auf welchem Feld sich die Bounding Box befindet.

3.2.6 Generierung von nächstem Spielzug

Die Schachspiellogik in diesem Projekt wird durch die Einbindung von Bibliotheken und einer externen Chess Engine realisiert.

4 Technische Umsetzung

In diesem Abschnitt wird die Implementierung der festgelegten Anforderungen unter Einsatz der ausgewählten Technologien beschrieben. Zunächst wird die Schachbrettkennung erläutert, gefolgt von einer Beschreibung des verwendeten Datensatzes für das Training der Objekterkennungsalgorithmen. Im Anschluss wird der Trainingsprozess für alle drei Objekterkennungsalgorithmen detailliert dargelegt. Danach wird die Umsetzung der Zuordnung der Schachfiguren auf das Schachbrett erklärt. Abschließend erfolgt eine Beschreibung der Integration der Schachspiellogik.

4.1 Schachbetterkennung mit OpenCV

Die Erkennung der Koordinaten der Eckpunkte der Schachfelder kann in zwei Schritte aufgeteilt werden. Der erste Schritt ist die Erkennung des Rahmens des Schachbretts und der zweite Schritt ist die Erkennung der einzelnen Felder.

Im Folgenden wird auf die einige der vielen eingesetzten Bildverarbeitungsschritte eingegangen.

4.1.1 Erkennung von Rahmen

Zuerst wird die größte rechteckige Kontur gesucht. Diese Kontur ist das Schachbrett. Sie wird eingezeichnet und dann wird die Hough Lines Funktion von OpenCV aufgerufen, um Linienerkennung auf die Farbe des vorhin eingezeichneten Rechteckes durchzuführen.

Diese erkannten Linien werden in horizontale und vertikale Linien eingeteilt und einer Funktion übergeben, welche die Schnittpunkte der horizontalen Linien mit den vertikalen Linien berechnet. Da oft Linien mehrfach eingezeichnet werden, werden

Schnittpunkte, welche nahe beieinander liegen, zu einem Schnittpunkt zusammengefasst.

Somit sind alle 4 Eckpunkte ermittelt wie in Abbildung 4.1 ersichtlich.

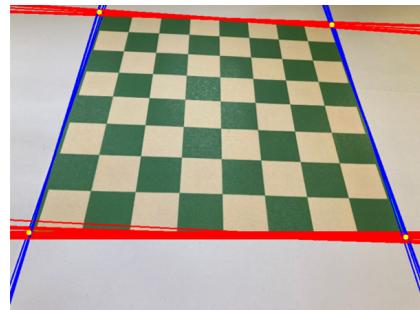


Abbildung 4.1: Erkannte Eckpunkte des Schachbretts

4.1.2 Erkennung der Felder

Für die Erkennung der Felder wird als erstes eine *Canny Edge Detection* gemacht. Das Ergebnis ist ein binäres Bild, das die starken Kanten im ursprünglichen Bild hervorhebt, wie in Abbildung 4.2 dargestellt.

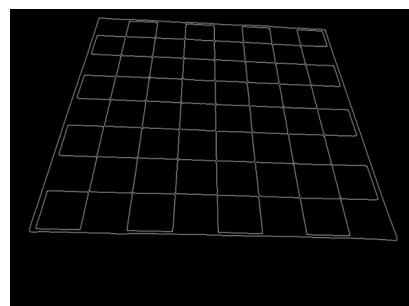


Abbildung 4.2: Erkannte Kanten

Anschließend wird mit den vorhin ermittelten Eckpunkten des Schachbretts eine Region of Interest (ROI) festgelegt, um den Bereich im Bild, auf dem das Schachbrett abgebildet ist, zu isolieren.

Anschließend wird wieder eine Hough Line Transformation durchgeführt.

Die erkannten Linien werden erneut in horizontale und vertikale Linien aufgeteilt. Dabei wurden bei der Erkennung des Rahmens maximale und minimale Grade festgelegt, welche eine Linie haben darf, um als horizontal bzw. vertikal zu gelten.

Als nächstes werden wieder die Schnittpunkte der horizontalen Linien mit den vertikalen Linien bestimmt und nahe zusammenliegende Punkte zu einem Punkt zusammengefasst. Für eine korrekte Erkennung des Schachbretts müssen genau 81 Punkte vorhanden sein. Das Ergebnis ist in Abbildung 4.3 dargestellt.

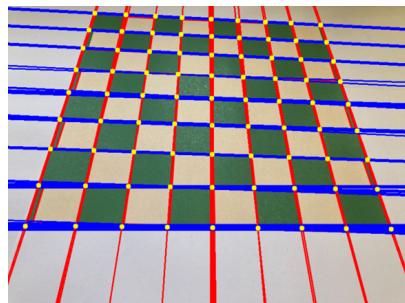


Abbildung 4.3: Erkannte Schnittpunkte

Mit diesen 81 Punkten können nun die Eckpunkte der einzelnen Felder festgelegt werden. Links oben auf dem Bild ist immer das Feld A1, da das Bild stets von der rechten Seite des weißen Spielers aufgenommen wird.

In Abbildung 4.4 sind die Felder des Schachbretts eingezeichnet.

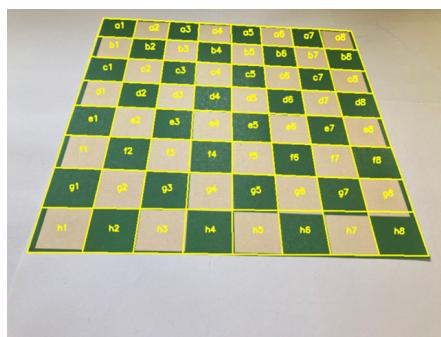


Abbildung 4.4: Erkannte Felder

4.2 Erkennung der Schachfiguren durch Objekterkennung

In diesem Abschnitt wird die Methodik zur Schachfigurenerkennung durch Objekterkennung beschrieben und die dafür notwendigen Schritte erläutert. Eine umfangreiche Evaluierung der Ergebnisse erfolgt in Abschnitt 5.

Folgende Objekterkennungsalgorithmen wurden ausgewählt:

- YOLOv5m
- YOLOv8m
- Faster R-CNN

Die Implementierung und Evaluierung der beiden YOLO-Modelle erfolgt unter Verwendung von PyTorch, während das Faster R-CNN-Modell mit Detectron2¹, einer von Facebook AI Research entwickelten Bibliothek, trainiert und evaluiert wird.

4.2.1 Auswahl und Vorbereitung des Datensatzes

Auf der Webseite von Roboflow, ist ein sehr guter, bereits gelabelter Datensatz mit 7252 Bildern erhältlich. [27]

Die Bilder sind von verschiedenen Schachbrettern, aus diversen Winkeln aufgenommen und mit unterschiedlich vielen Figuren auf dem Feld. In diesem Datensatz waren einige Bilder verdreht und somit die Bounding Boxes am falschen Platz. Diese Bilder mussten aussortiert werden.

Die Aufteilung in Trainingsset, Validierungsset und Testset wurde wie folgt vorgenommen:

- 5662 Bilder im Trainingsset (86%)
- 910 Bilder im Validierungsset (14%)
- 16 Bilder im Testset (0%)

¹<https://github.com/facebookresearch/detectron2/>

Abbildung 4.5 zeigt, dass einige Klassen im Datensatz mehr repräsentiert sind, als andere.

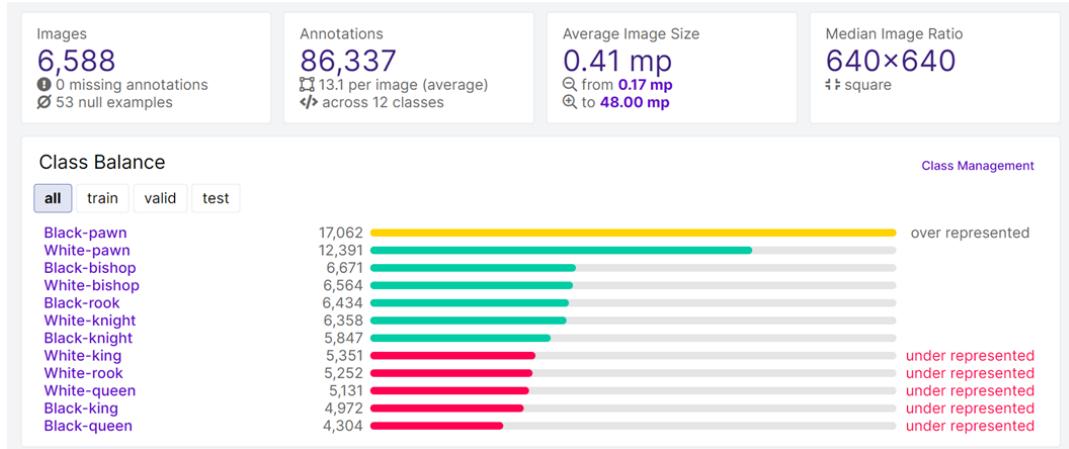


Abbildung 4.5: Klassenverteilung in Datensatz

Als Preprocessing der Daten wurde die Option 'Resize: Fit (white edges) in 640x640' gewählt. Dabei werden die Bilder auf die Größe 640x640 skaliert, wobei das Seitenverhältnis beibehalten wird. Es werden weiße Ränder hinzugefügt, um das Bild quadratisch zu machen. [28]

Preprocessing wird auf alle Bilder angewendet. Augmentierungen, künstliche Veränderungen oder Transformationen auf Bildern, werden nur auf die Bilder des Trainingsatzes gemacht. [28]

Augmentierungen können beispielsweise Rotationen, Spiegelungen, Unschärfe oder Farbveränderungen sein. Der Zweck ist, den Datensatz zu diversifizieren, um dadurch die Generalisierungsfähigkeiten des Modells zu verbessern. [29]

Laut der offiziellen Dokumentation wird bei der Verwendung von YOLOv5 für das Training jedoch keine separate Bildaugmentierung empfohlen, da YOLOv5 bereits während des Trainings Online-Augmentierung durchführt. [30]

4.2.2 Trainieren mit YOLOv5m Algorithmus

Auswahl der Trainingsparameter

Ultralytics empfiehlt in der offiziellen Dokumentation [31], zunächst mit den Standard-Einstellungen zu trainieren, da diese in der Regel gute Ergebnisse liefern.

Für kleine bis mittelgroße Datensätze wird empfohlen, das Training auf Basis von vortrainierten Gewichten zu beginnen. Hier sind einige spezifische Parametervorschläge:

- Epochen: Es soll mit 300 Epochen gestartet werden. Wenn Overfitting stattfindet soll das Modell mit einer geringeren Anzahl von Epochen erneut trainiert werden.
- Bildgröße: Es werden die besten Ergebnisse erzielt, wenn das Modell mit derselben Bildgröße trainiert und getestet wird.
- Batch Größe: Es soll die größtmögliche Batch Größe verwendet werden, die die Hardware zulässt. [31]

Das Training wird mittels folgendem Befehl gestartet:

```
1 !python train.py \
2   --data 'data.yaml'\
3   --weights yolov5m.pt\
4   --img 640\
5   --epochs 300\
6   --batch-size 64\
7   --cfg 'models/yolov5m.yaml'\
8   --cache disk\
9   --device 0
```

Nach dem ersten Durchlauf hat sich gezeigt, dass 300 Epochen zu viel waren. Der Validation Object Loss stieg nach 70 Epochen wieder an. Das ist ein Zeichen für Overfitting. Daher wurde die Anzahl der Epochen auf 70 reduziert.

Evaluation von Ergebnis

Die Abbildung 4.6 zeigt die Ergebnisse des Trainings mit 70 Epochen.

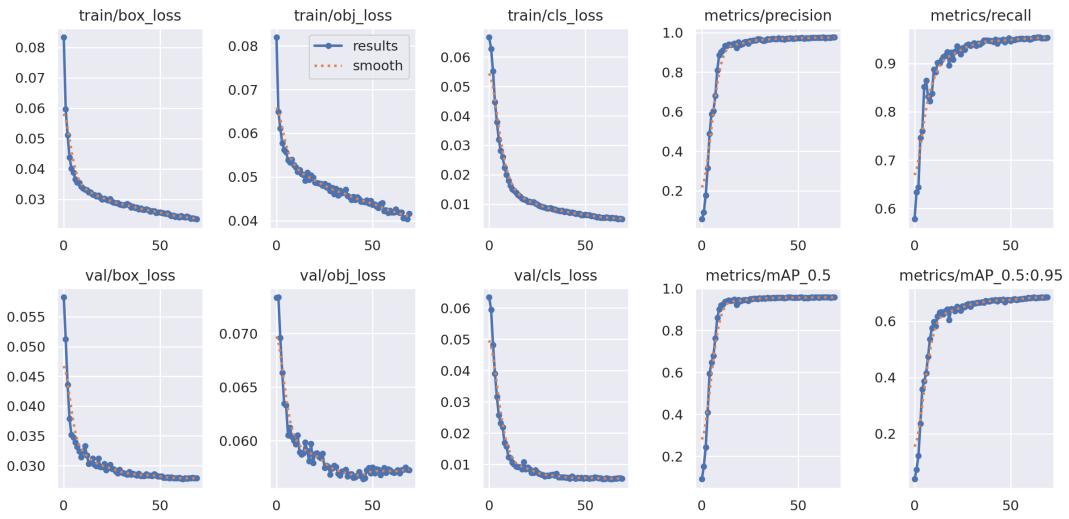


Abbildung 4.6: Ergebnisse YOLOv5m Versuch 1

Die Loss-Kurven bieten wichtige Einblicke in das Trainingsergebnis.

- Der Box Loss gibt die Fehlerquote an bei der Lokalisierung des Objekts innerhalb der Rasterzelle.
- Der Object Loss gibt die Fehlerquote an bei der Feststellung, ob ein Objekt in einer bestimmten Rasterzelle vorhanden ist oder nicht.
- Der Class Loss gibt die Fehlerquote für die Klassifikationsaufgabe an.

[32]

Die Loss Werte sind alle relativ niedrig, wohingegen die mAPs relativ hoch sind.

Die Precision-Recall Kurve in Abbildung 4.7 weist auf eine gute Leistung der Modells bei der Klassifikation hin.

Die Erkennung auf neuen, eigenen Bildern zeigt eine unzureichende Leistung, was auf eine mangelhafte Generalisierung hindeutet.

Wenn es ein Ungleichgewicht zwischen den Klassen gibt, besteht die Gefahr, dass das Modell ein Bias gegenüber der häufigeren Klasse entwickelt. Dieser Bias kann sich in der Confusionmatrix zeigen, wo das Modell möglicherweise überwiegend die

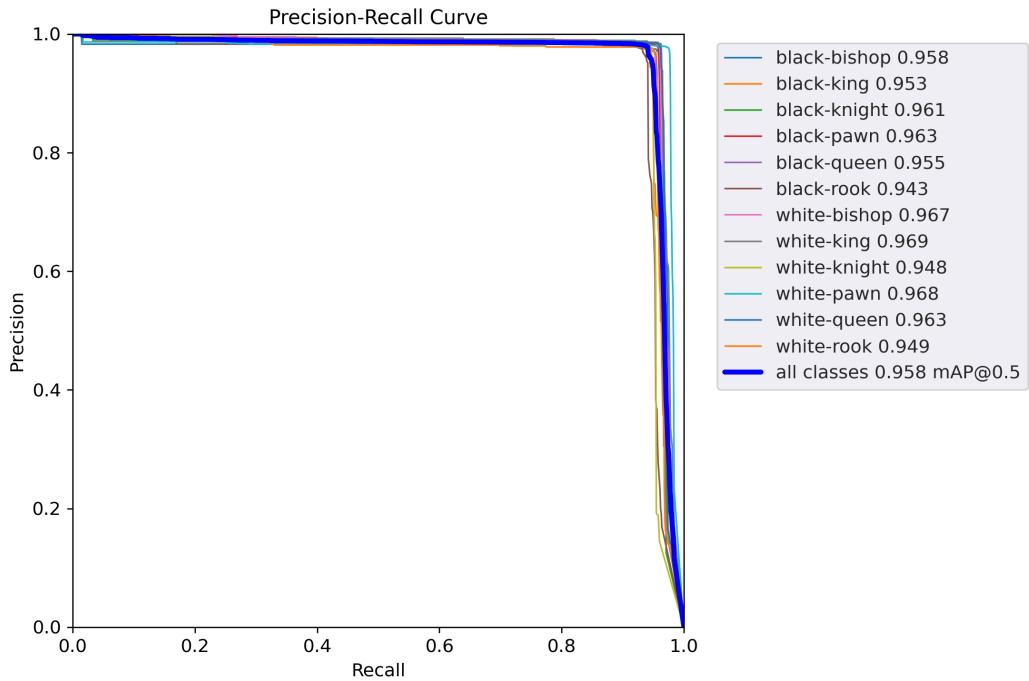


Abbildung 4.7: Precision-Recall Kurve YOLOv5m Versuch 1

Mehrheitsklasse vorhersagt. [33] Die Confusion Matrix in Abbildung 4.8 zeigt jedoch, dass es kaum Verwechslungen gibt.

Als Schlussfolgerung lässt sich feststellen, dass der Datensatz zu klein ist. Eine einfache Lösung besteht darin, Augmentationen durchzuführen, um den Datensatz zu vergrößern.

4.2.3 Anpassen des Datensatzes

Folgende Augmentationen wurden angewandt:

- Horizontales spiegeln
- Sättigung: Zwischen -25% and +25%
- Unschärfe: Bis zu 1,25 Pixel. Das bedeutet, dass ein Gausscher Weichzeichner angewendet wird, wobei der Unschärfeeefekt durch eine Pixelanzahl gesteuert wird. 25 Pixel stellen das Maximum dar.

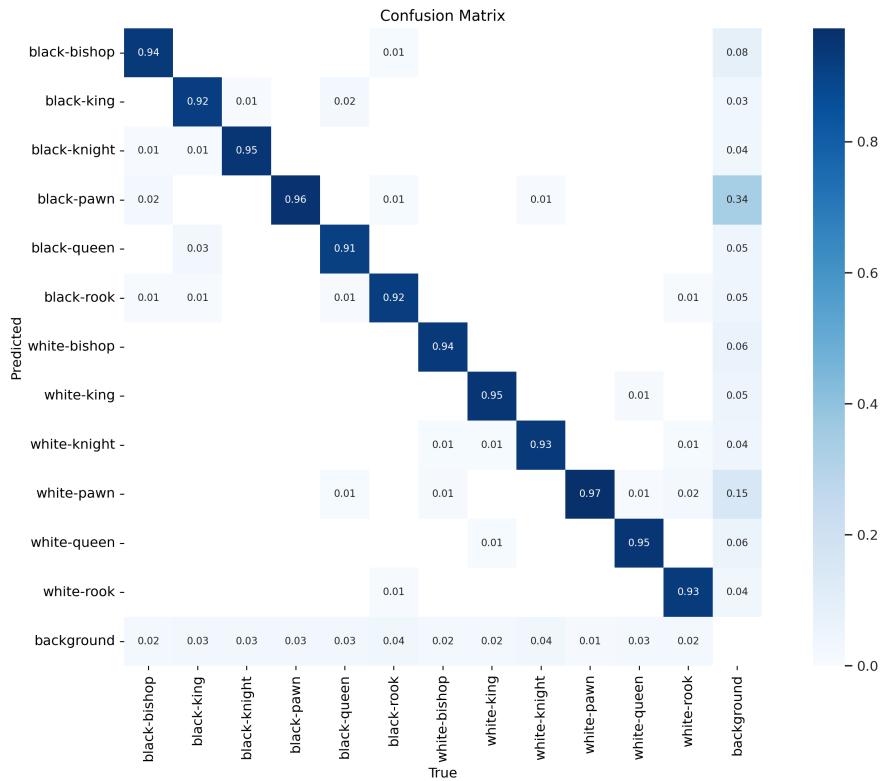


Abbildung 4.8: Confusion Matrix YOLOv5m Versuch 1

- Rauschen: Bis zu 5% der Pixel
- Cutout: 4 Boxen, jeweils mit 15% Größe.

Als Ergebnis hat sich die Anzahl der Bilder im Trainingsset nun verdreifacht.

- 16986 Bilder im Trainingsset
- 910 Bilder im Validierungsset
- 16 Bilder im Testset

4.2.4 Erneutes Trainieren mit YOLOv5m Algorithmus

Es kann aus den Abbildungen 4.9, 4.10 und 4.11 abgelesen werden, dass nur ein kleiner Unterschied zu den Ergebnissen vom ersten Versuch besteht. Die graue Kurve zeigt den Versuch mit dem kleineren Datensatz und die gelbe Kurve ist der Versuch mit

dem größeren Datensatz. Das Training mit dem größeren Datensatz liefert nur leicht bessere Ergebnisse.

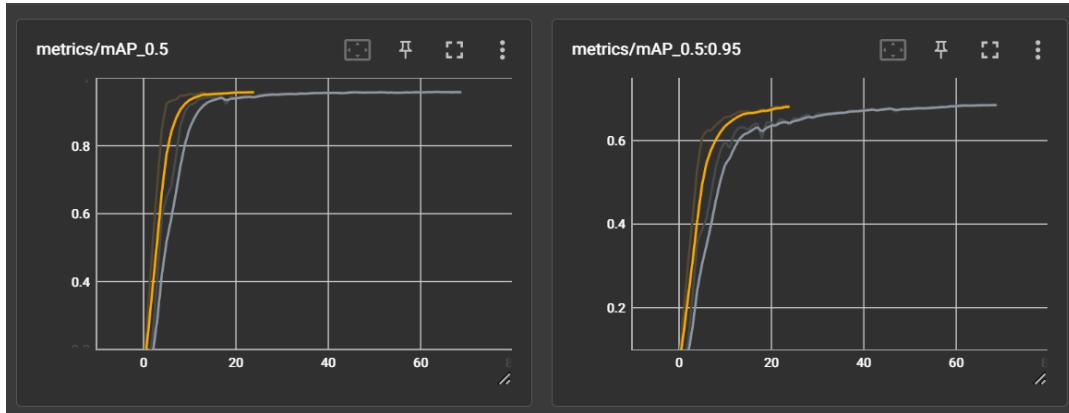


Abbildung 4.9: mAP Vergleich YOLOv5m

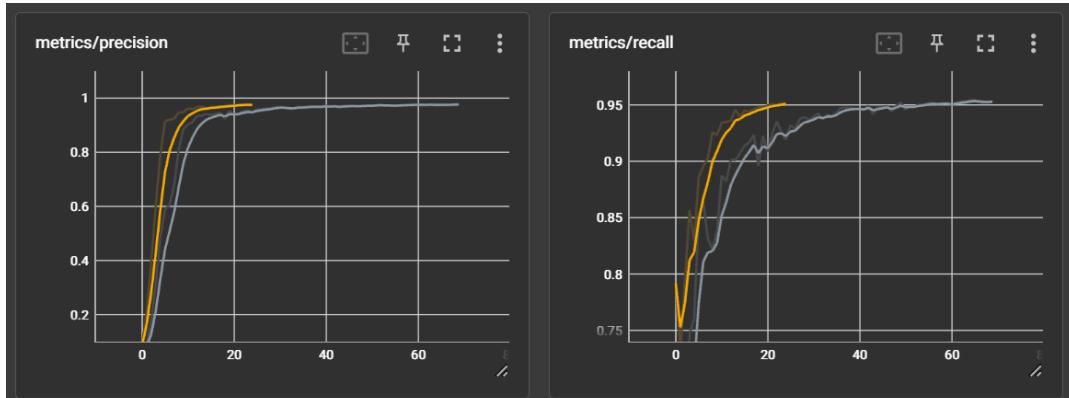


Abbildung 4.10: Precision und Recall Vergleich YOLOv5m

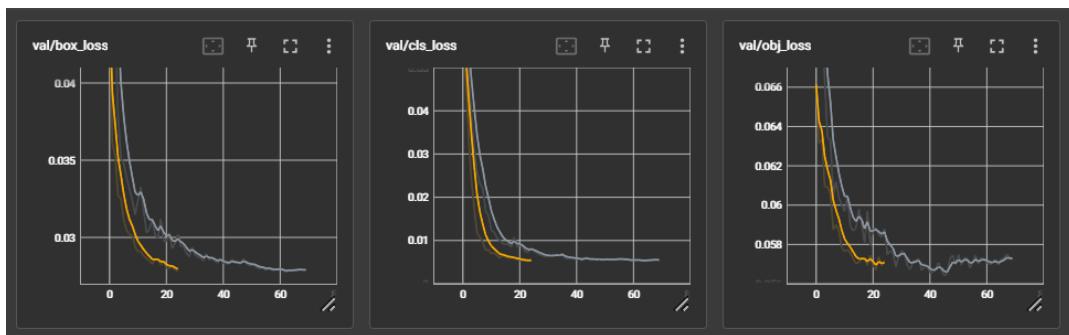


Abbildung 4.11: Losskurven Vergleich YOLOv5m

Jedoch generalisiert das Modell, welches mit dem größeren Datensatz trainiert wurde auf die eigenen, nicht im Datensatz enthaltenen Bilder, deutlich besser.

4.2.5 Trainieren mit YOLOv8m Algorithmus

Die Auswahl der Trainingsepochen erfolgte durch Trial and Error. Wenn zu viele Trainingsepochen durchgeführt wurden, neigte das Modell zum Overfitting, was sich dadurch zeigte, dass die Validation Loss Kurven an einem bestimmten Punkt wieder anstiegen. Die Batchsize 32 war die größtmögliche mit der verwendeten GPU.

Die Hyperparametereinstellungen wurden beibehalten, um eine direkte Vergleichbarkeit zwischen YOLOv5 und YOLOv8 zu gewährleisten, da beide Modelle standardmäßig dieselben Hyperparameter verwenden.

Hyperparameter sind Einstellungen für den Algorithmus, die vor der Trainingsphase festgelegt werden und währenddessen konstant bleiben. Beispiele dafür sind die Learning Rate und die Batch Size. [34]

Es ist zu erwähnen, dass beide Modelle Hyperparameter haben, welche exklusiv für jeden Algorithmus konfiguriert sind und nicht in beiden verwendet werden.

Das Training wird mittels folgendem Befehl gestartet:

```
1 model = YOLO('yolov8m.pt')
2 results = model.train(data='/data1/home/tscl/Chess-Pieces-3-2/data.yaml',
3                         imgsz=640, epochs=25, batch=32, cache=True, device=0)
```

Die Ergebnisse und der Vergleich mit den YOLOv5m Ergebnissen werden detailliert in Abschnitt 5 besprochen.

4.2.6 Trainieren mit Faster R-CNN Algorithmus

Die besten Ergebnisse mit dem Faster R-CNN Algorithmus konnten durch bestmögliche Anpassung der Hyperparameter an die YOLO-Hyperparameter erzielt werden. Mit folgendem Code wurden die Hyperparameter festgelegt und der Trainingsprozess gestartet.

```

1  from detectron2.config import get_cfg
2  import os
3
4  cfg = get_cfg()
5  cfg.merge_from_file(model_zoo.get_config_file(
6      "COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"))
7  cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(
8      "COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml")
9  cfg.MODEL.DEVICE = 'cuda:0'
10
11 cfg.DATASETS.TRAIN = ("my_dataset_train",)
12 cfg.DATASETS.TEST = ("my_dataset_test",)
13 cfg.DATASETS.VALID = ("my_dataset_val",)
14
15 cfg.DATALOADER.NUM_WORKERS = 8 # Number of data loading threads
16 cfg.SOLVER.IMS_PER_BATCH = 8
17 cfg.SOLVER.BASE_LR = 0.01 # Learning Rate
18 cfg.SOLVER.WARMUP_ITERS = 1000
19 cfg.SOLVER.MAX_ITER = 8000 # Iterations
20 cfg.SOLVER.STEPS = (4000, 6000, 7000)
21 cfg.SOLVER.GAMMA = 0.1 # Reduce LR by a factor of 0.1 at each step
22 cfg.MODEL.ROI_HEADS.NUM_CLASSES = 13 # number of classes + 1 (superclass)
23 cfg.TEST.EVAL_PERIOD = 200 # nr. iterations after which the val set is evaluated.
24
25 f = open('config.yml', 'w')
26 f.write(cfg.dump()) # Save the configuration to a YAML file
27 f.close()
28 os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
29 trainer = CocoTrainer(cfg)
30 trainer.resume_or_load(resume=False)
31 trainer.train()

```

In diesem Fall hat das Modell konvergiert. Das bedeutet, dass die Verlustfunktion sich einem stabilen Minimum angenähert hat. Daher waren keine weiteren Trainingsepochen erforderlich.

Die Ergebnisse und der Vergleich mit anderen Algorithmen werden in Abschnitt 5 besprochen.

4.3 Zuordnen der Schachfiguren auf das Schachbrett

Nach der Erkennung des Schachbretts und der Objekterkennung der Schachfiguren wird für jede Bounding Box ein Punkt im unteren Viertel in der Mitte berechnet, welcher dann die Bounding Box repräsentiert. Das ist auf Abbildung 4.12 ersichtlich.



Abbildung 4.12: Zuordnung der Schachfiguren auf das Schachfeld

Anhand von diesem Punkt kann die Schachfigur dem Feld, auf dem sie steht zugeordnet werden. Aus den Informationen, welche Figur auf welchem Feld steht, wird ein Dictionary erstellt, welches im nächsten Schritt weiterverarbeitet wird.

4.4 Spiellogik mit python-chess und Stockfish Schach-Engine

In diesem Abschnitt wird die Implementierung der Spiellogik mit Hilfe der Bibliothek python-chess und der Schach-Engine Stockfish erläutert.

FEN-String Generierung

Um die FEN-Repräsentation des aktuellen Schachbrett-Zustands zu erstellen, wurde eine Methode geschrieben, die eine Dictionary von erkannten Schachfiguren als Eingabe akzeptiert, das die Schlüssel “square” und “class_name” enthält. Dieses Dictionary wurde im vorherigen Schritt 4.3 erstellt. “square” gibt die Position der Figur auf dem Schachbrett an und “class_name” gibt die Klasse der Figur an. Die Methode erstellt eine leere 8x8 Schachbrett-Matrix und platziert die erkannten Figuren entsprechend. Anschließend wird die Matrix in einen FEN-String umgewandelt. Dabei werden leere

Felder durch ihre Anzahl ersetzt. Der resultierende FEN-String repräsentiert den aktuellen Zustand des Schachbretts.

Hier folgt ein Beispiel für einen FEN-String. Dieser stellt die Anfangsaufstellung dar.

`rnbqbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR`

In der Abbildung 4.13 ist die grafische Repräsentation dieses FEN-Strings einsehbar. Diese kann erstellt werden, indem mit Hilfe der python-chess Bibliothek ein Board Objekt erstellt wird. Der FEN-String ist der Eingabeparamter.

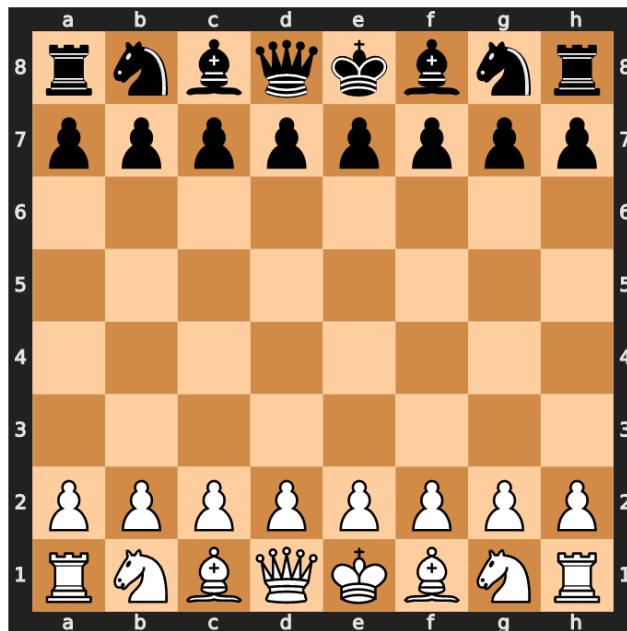


Abbildung 4.13: Schachbrett Anfangsposition

Spielerzug, Gültigkeitsüberprüfung und Zuggenerierung

Die Methode `make_move` ermöglicht es dem/der menschlichen Spieler*in, einen Zug durchzuführen. Sie prüft die Gültigkeit des Zugs und aktualisiert das Schachbrett entsprechend. Dafür wird die python-chess Bibliothek genutzt. Wenn der/die Spieler*in einen gültigen Zug gemacht hat, wird der FEN-String aktualisiert. Des weiteren wird mit Hilfe der python-chess Bibliothek überprüft, ob der Gegner im Schach oder Schachmatt ist. Im Falle eines gültigen Zugs wird daraufhin die Stockfish-Engine genutzt, um den besten Zug für den Computergegner zu ermitteln.

Darstellung in einer Benutzeroberfläche

Um den Benutzern eine visuelle Kontrolle der Objekterkennung zu ermöglichen, wurde eine Benutzeroberfläche mit Python Tkinter implementiert, wie Abbildung 4.14 zeigt.



Abbildung 4.14: Benutzeroberfläche

Über die Benutzeroberfläche erhalten Benutzer*innen Rückmeldungen. Es wird rückgemeldet, ob der Zug gültig war und ob es Schach oder Schachmatt steht. Außerdem besteht die Möglichkeit, die Objekterkennung beliebig oft laufen zu lassen, falls eine Schachfigur falsch erkannt wurde.

Die Kommunikation mit dem Backend erfolgt über ein Python Flask REST-Interface.

5 Evaluierung

Im Rahmen dieses Abschnitts werden die Ergebnisse des Trainings eines Modells unter Verwendung aller drei Objekterkennungsalgorithmen evaluiert. Zudem werden einige Vorschläge zur Verbesserung der Objekterkennung vorgestellt.

5.1 Vergleich YOLOv5m vs. YOLOv8m

Abbildung 5.1 zeigt die Ergebnisse des Trainings mit YOLOv8m. Die Ergebnisse zeigen, dass fast genau dieselbe mAP erreicht wurde wie mit YOLOv5. Siehe Abbildung 4.9. Die mAP_0.5:0.95 der YOLOv5 Ergebnisse liegt bei 95% und die mAP_0.5 liegt bei 68%. Die mAP50-95 von YOLOv8 beträgt 70% und die mAP50 beträgt 96%.

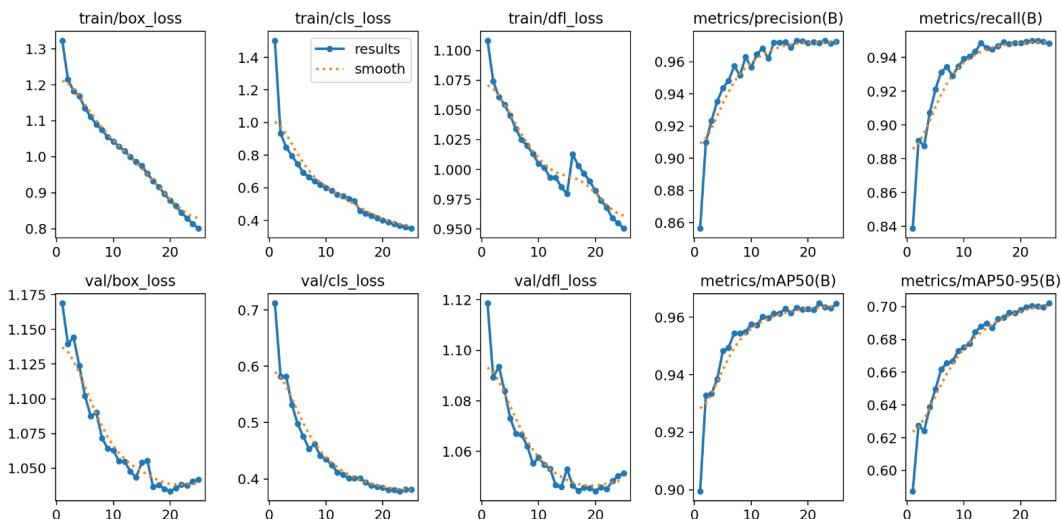


Abbildung 5.1: Ergebnisse YOLOv8m

Dies entspricht dem in Abschnitt 2.2.1 geschilderten Fakt, dass mit YOLOv8 eine höhere mAP erreicht werden kann.

Die Precision-Recall-Kurven und die F1-Scores zeigen keine signifikanten Unterschiede. Auch die Confusion Matrix zeigt in beiden Ergebnissen keine erwähnenswerte Anzahl an Verwechslungen.

In Abbildung 5.2 sind Validation Loss Kurven zu sehen. Die grüne Kurve stellt das mit YOLOv5 trainierte Modell dar und die blaue Kurve stellt das mit YOLOv8 trainierte Modell dar. Nach Betrachtung dieser Kurven kann nun geschlussfolgert werden, dass YOLOv5m deutlich bessere Ergebnisse liefert, was die Validation Loss Kurven betrifft. Außerdem generalisiert das mit YOLOv8 trainierte Modell schlechter als das mit YOLOv5 trainierte Modell.

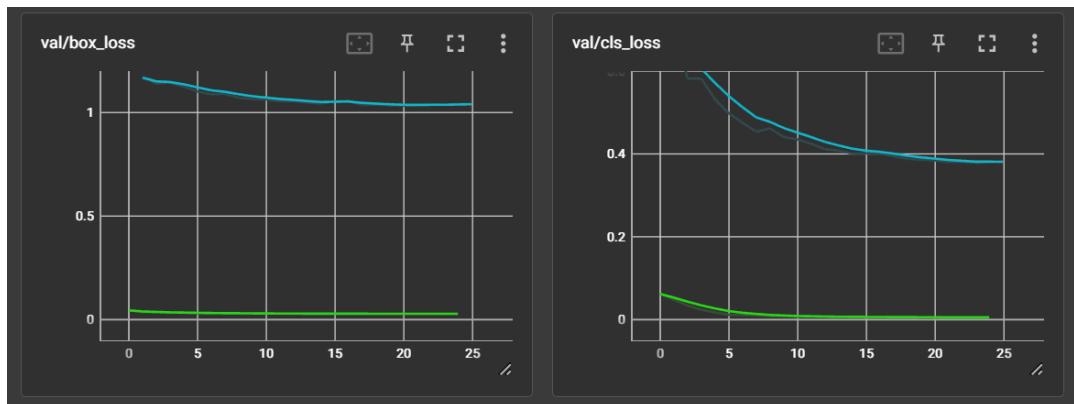


Abbildung 5.2: YOLOv5m vs. YOLOv8m

Um die Ergebnisse von YOLOv8 zu verbessern, müsste ein sogenanntes Hyperparameter Tuning gemacht werden, um die optimalen Hyperparameter für das Training mit dem spezifischen Datensatz zu bestimmen. [34]

Dies ist jedoch sehr ressourcen- und zeitaufwendig und sprengt den Rahmen dieser Bachelorarbeit.

5.2 Vergleich YOLO vs. Faster R-CNN

Bei Faster-RCNN wurden andere Metriken erhalten, ersichtlich in Abbildung 5.3. Beispielsweise kann die AP nicht mit den Ergebnissen des Trainings mit YOLO verglichen werden, da mit YOLO nur die mAP_0.5 und mAP_0.5:0.95 vorliegen. Außerdem sind die Metriken AP und mAP nicht dasselbe wie in Abschnitt 2.3 erläutert.

```

Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.713
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.929
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.914
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.716
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.743
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.496
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.775
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.775
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.782
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.783
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
[12/11 15:52:33 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
| AP | AP50 | AP75 | APs | APm | API |
|-----|-----|-----|-----|-----|-----|
| 71.256 | 92.875 | 91.417 | 71.616 | 74.262 | nan |

[12/11 15:52:33 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[12/11 15:52:33 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category | AP | category | AP | category | AP |
|-----|-----|-----|-----|-----|-----|
| Chess-Pieces-v3uu | nan | Black-bishop | 58.305 | Black-king | 76.707 |
| Black-knight | 77.124 | Black-pawn | 73.093 | Black-queen | 74.195 |
| Black-rook | 72.776 | White-bishop | 67.625 | White-king | 71.289 |
| White-knight | 72.425 | White-pawn | 61.243 | White-queen | 77.455 |
| White-rook | 72.831 | | | | |

[12/11 15:52:33 d2.engine.defaults]: Evaluation results for my_dataset_test in csv format:
[12/11 15:52:33 d2.evaluation.testing]: copypaste: Task: bbox
[12/11 15:52:33 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,API
[12/11 15:52:33 d2.evaluation.testing]: copypaste: 71.2558,92.8749,91.4172,71.6156,74.2615,nan

```

Abbildung 5.3: Metriken Faster R-CNN

Weiters ist das subjektive Empfinden, dass das Training mit Faster R-CNN deutlich komplizierter ist, als mit YOLOv5 und YOLOv8, da das Framework Detectron2 sehr mangelhaft dokumentiert ist. Es gibt beispielsweise keine Empfehlungen, wie die Hyperparameter zu Beginn gewählt werden sollten.

Darüber hinaus konnte das mit Detectron2 trainierte Modell nicht im Programm getestet werden, da dafür Detectron2 lokal installiert werden müsste. Dafür wäre die Installation des CUDA Toolkits erforderlich. Dieses kann nur bei vorhandener NVIDIA Grafikkarte installiert werden.

Somit konnten die Ergebnisse nur durch Hochladen von eigenen Bildern auf den GPU Server getestet werden. Weil das Modell gut generalisiert sind die Ergebnisse als zufriedenstellend zu bewerten.

5.3 Verbesserungsmöglichkeiten

Um die Genauigkeit der Objekterkennung zu erhöhen wäre das Hyperparameter Tuning eine Möglichkeit.

Außerdem könnte der Datensatz mit Bildern der eigenen Schachfiguren ergänzt werden, was zu einer besseren Generalisierung führen könnte.

Eine Möglichkeit, den Datensatz relativ schnell zu vergrößern, besteht darin, synthetische Bilder zu erzeugen. Eine Möglichkeit, synthetische Bilder zu erzeugen ist, die eigenen Figuren mit Blender zu modellieren, ein Skript für das Labeling zu schreiben und dann Bilder mit verschiedenen Feldaufstellungen und Kamerawinkeln aufzunehmen.

Um die Nutzerfreundlichkeit zu steigern, könnte die Schachbretterkennung optimiert werden, sodass sie unabhängig von der Kameraposition funktioniert und das Schachbrett sogar mit Schachfiguren auf dem Brett erkannt wird. Dies würde es ermöglichen, die Kamera während eines Spiels frei zu bewegen. Eine Realisierung dieser Verbesserung könnte durch den Einsatz von Objekterkennungstechnologien erfolgen.

6 Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, ein funktionierendes System zur Erkennung des Spielzustands eines Schachspiels unter Verwendung von Deep Learning und Computer Vision zu entwickeln. Dabei wurden die Algorithmen YOLOv5m, YOLOv8m und Faster R-CNN für die Schachfigurenerkennung evaluiert. Die Ergebnisse der Evaluation zeigten, dass der YOLOv5m-Algorithmus die besten Resultate auf dem verwendeten Datensatz erzielte.

Die Schachbretterkennung wurde mithilfe der OpenCV-Bibliothek implementiert, wobei spezielle Techniken wie die Hough Line Transform und Canny Edge Detection zum Einsatz kamen.

Die Integration der Spiellogik erfolgte mithilfe der python-chess Bibliothek, während für die Generierung der Spielzügen des Computers die Stockfish Schach-Engine verwendet wurde. Die Implementierung eines Benutzerinterfaces mit Python Tkinter ermöglicht eine visuelle Kontrolle, ob die Schachfiguren richtig erkannt wurden.

Die Ergebnisse der technischen Umsetzung¹ ² erfüllen die Anforderungen.

Die Integration einer funktionierenden Schachbretterkennung, die auch mit darauf befindlichen Figuren funktioniert, würde zu einem verbesserten Benutzererlebnis beitragen. Eine Möglichkeit, dies umzusetzen wäre ebenfalls mit Hilfe von Objekterkennung.

Um die Robustheit der Objekterkennung weiter zu steigern, stehen verschiedene vielversprechende Ansätze zur Verfügung. Dazu zählen beispielsweise die Erweiterung des Datensatzes durch synthetisch generierte Daten sowie das Feintuning der Hyperparameter.

¹https://github.com/ClaraTschanon/BachelorThesis_ObjectDetection

²<https://www.youtube.com/watch?v=aD9vcngXvzM&feature=youtu.be>

Literaturverzeichnis

- [1] What are neural networks? | IBM. [Online]. Available: <https://www.ibm.com/topics/neural-networks>
- [2] A. Ouaknine. Review of deep learning algorithms for object detection. [Online]. Available: <https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852>
- [3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection.” [Online]. Available: <http://arxiv.org/abs/1506.02640>
- [4] E. Zvornicanin. What is YOLO algorithm? | baeldung on computer science. [Online]. Available: <https://www.baeldung.com/cs/yolo-algorithm>
- [5] G. Jocher, “YOLOv5 by ultralytics.” [Online]. Available: <https://github.com/ultralytics/yolov5>
- [6] Faster r-CNN | ML. Section: Machine Learning. [Online]. Available: <https://www.geeksforgeeks.org/faster-r-cnn-ml/>
- [7] I. D. a. A. Team. AI vs. machine learning vs. deep learning vs. neural networks: What’s the difference? [Online]. Available: <https://www.ibm.com/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks/>
- [8] Difference between artificial intelligence vs machine learning vs deep learning. Section: Difference Between. [Online]. Available: <https://www.geeksforgeeks.org/difference-between-artificial-intelligence-vs-machine-learning-vs-deep-learning/>
- [9] What is deep learning? applications & examples. [Online]. Available: <https://cloud.google.com/discover/what-is-deep-learning>
- [10] J. G. AUG 22 and . . M. Read. What is object detection? the ultimate guide. [Online]. Available: <https://blog.roboflow.com/object-detection/>

- [11] YOLO algorithm for object detection explained [+examples]. [Online]. Available: <https://www.v7labs.com/blog/yolo-object-detection>,<https://www.v7labs.com/blog/yolo-object-detection>
- [12] J. N. JUN 7 and . . M. Read. The YOLO algorithm: A guide to YOLO models. [Online]. Available: <https://blog.roboflow.com/guide-to-yolo-models/>
- [13] YOLO object detection explained: A beginner’s guide. [Online]. Available: <https://www.datacamp.com/blog/yolo-object-detection-explained>
- [14] J. S. JUN 29 and . . M. Read. What is YOLOv5? a guide for beginners. [Online]. Available: <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>
- [15] J. Solawetz, F. JAN 11, and . . M. Read. What is YOLOv8? the ultimate guide. [Online]. Available: <https://blog.roboflow.com/whats-new-in-yolov8/>
- [16] Weights & biases. [Online]. Available: <https://wandb.ai/mukilan/wildlife-yolov8/reports/A-Gentle-Introduction-to-YOLOv8--Vmldzo0MDU5NDA2>
- [17] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-CNN: Towards real-time object detection with region proposal networks.” [Online]. Available: <http://arxiv.org/abs/1506.01497>
- [18] Object detection leaderboard. [Online]. Available: <https://huggingface.co/blog/object-detection-leaderboard>
- [19] F1 score in machine learning: Intro & calculation. [Online]. Available: <https://www.v7labs.com/blog/f1-score-guide>,<https://www.v7labs.com/blog/f1-score-guide>
- [20] baeldung. Training and validation loss in deep learning | baeldung on computer science. [Online]. Available: <https://www.baeldung.com/cs/training-validation-loss-deep-learning>
- [21] Train test validation split: How to & best practices [2023]. [Online]. Available: <https://www.v7labs.com/blog/train-validation-test-set>,<https://www.v7labs.com/blog/train-validation-test-set>
- [22] Training vs testing vs validation sets. Section: Python. [Online]. Available: <https://www.geeksforgeeks.org/training-vs-testing-vs-validation-sets/>
- [23] What is underfitting? | IBM. [Online]. Available: <https://www.ibm.com/topics/underfitting>

- [24] What is overfitting? | IBM. [Online]. Available: <https://www.ibm.com/topics/overfitting>
- [25] Machine learning glossary. [Online]. Available: <https://developers.google.com/machine-learning/glossary>
- [26] FEN (forsyth-edwards notation) - chess terms. [Online]. Available: <https://www.chess.com/terms/fen-chess>
- [27] chess dataset. [Online]. Available: <https://universe.roboflow.com/deneme-iq93l/chessv1-sswl9>
- [28] Preprocess images. [Online]. Available: <https://docs.roboflow.com/datasets/image-preprocessing>
- [29] Create augmented images. [Online]. Available: <https://docs.roboflow.com/datasets/image-augmentation>
- [30] Ultralytics. Train custom data. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/train_custom_data
- [31] ——. Tips for best training results. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results
- [32] ——. Architecture summary. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/architecture_description
- [33] ——. YOLO common issues. [Online]. Available: <https://docs.ultralytics.com/guides/yolo-common-issues>
- [34] ——. Hyperparameter tuning. [Online]. Available: <https://docs.ultralytics.com/guides/hyperparameter-tuning>

Eidesstattliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, am 20. Mai 2024

Clara Tschamom