# Komparative Analyse von YOLO Object Detection Modellen zur Erkennung von Wildtieren auf Drohnenaufnahmen

Florian Rakos


BACHELORARBEIT

eingereicht am

Fachhochschul-Bachelorstudiengang

Medientechnik und -design

in Hagenberg im Mühlkreis

im April 2024

Betreuung:

FH-Prof. Dr. David Christian Schedl BSc MSc

© Copyright 2024 Florian Rakos

Diese Arbeit wird unter den Bedingungen der Creative Commons Lizenz *Attribution-NonCommercial-NoDerivatives 4.0 International* (CC BY-NC-ND 4.0) veröffentlicht – siehe https://creativecommons.org/licenses/by-nc-nd/4.0/.

# Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt. Die vorliegende, gedruckte Arbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Hagenberg im Mühlkreis, am 15. April 2024

Florian Rakos

# Inhaltsverzeichnis

# Kurzfassung

Im letzten Jahrzehnt hat die künstliche Objekterkennung enorme Fortschritte gemacht. Das von der FH Oberösterreich initiierte Forschungsprojekt BAMBI, will diese technologischen Entwicklungen nutzen, um Object Detection Techniken zur Bestandserfassung von Wildtieren einzusetzen. Dafür wurde bereits eine Vielzahl an Drohnen-Thermalbildern von Wildtieren (großteils Rotwild und Rehwild) erfasst und rund 3000 Bilder händisch annotiert.

Aufbauend auf diesen Daten wurden in der vorliegenden Arbeit zwei State of the Art Single-Stage Object Detection Modelle YOLO-V8 und YOLO-NAS trainiert und verglichen. Ziel war es dabei, eine möglichst präzise und schnelle Echtzeitdetektion von Drohnenaufnahmen zu ermöglichen. Das beste YOLO-V8 Modell erzielte eine mean Average Precision (IoU-0.5) von 95.7% auf einem ungesehenen Testdatensatz. Das beste YOLO-NAS Modell erreichte dabei hingegen nur eine mAP von 77.4% und wies einen sehr instabilen Trainingsprozess auf. Auch in der Schnelligkeit übertraf das YOLO-V8 Modell mit einer Inferenzdauer im Bereich von 10ms. Das YOLO-NAS Modell benötigt hingegen deutlich über 100ms, um ein Bild zu verarbeiten.

Die vorliegende Arbeit kommt zu dem Schluss, dass die Nutzung von YOLO-V8 der Verwendung von YOLO-NAS vorzuziehen ist. Dies basiert auf der besseren Leistung, einer einfacheren Implementierung und einer umfassenderen Dokumentation die für dieses Modell zur Verfügung steht.

# Abstract

In the last decade, the area of Object Detection has seen tremendous progress. The research project BAMBI, initiated by the University of Applied Sciences Upper Austria, aims to leverage these technological advancements to employ Object Detection techniques for wildlife population monitoring. With this goal in mind, a large number of drone thermal images of wildlife, mostly red deer and roe deer, have already been captured and approximately 3000 images manually annotated.

Building on this data, two state-of-the-art single-stage object detection models, YOLO-V8 and YOLO-NAS, were trained and compared in this work. The goal was to enable precise and fast real-time detection of drone images. The best YOLO-V8 model achieved a mean Average Precision (IoU-0.5) of 95.7% on an unseen test dataset. In contrast, the best YOLO-NAS model only achieved an mAP of 77.4% and exhibited a very unstable training process. Additionally, in terms of speed, the YOLO-V8 model outperformed, with an inference duration in the range of 10ms, while the YOLO-NAS model required over 100ms to process an image.

Overall, this work recommends the use of YOLO-V8 over YOLO-NAS. The recommendation is based on its superior performance, easier implementation, and more comprehensive documentation available for this model.

# Kapitel 1

# Einleitung

## 1.1 Kontext und Motivation

Durch die Einführung von Deep Neural Networks, die Verfügbarkeit von großen Datenmengen und die steigende Prozessorleistung hat die künstliche Objekterkennung in der jüngsten Vergangenheit sehr große Leistungssprünge gemacht. So erzielen heutige Detektions Modelle in vielen Anwendungsgebieten menschenähnliche oder sogar noch bessere Ergebnisse [12]. Diese Technologie bietet das Potenzial für die Lösung vieler Probleme, wenn sie in neuen Kontexten angewandt wird. Ein solcher Kontext ist das Forschungsprojekt BAMBI (Biodiversity Airborne Monitoring Based on Intelligent UAV sampling) [75]. Dieses Projekt will Object Recognition Techniken anwenden um die Bestandsaufnahme von Wildtieren zu erleichtern. Hierfür wurde bereits eine große Menge an Drohnen-Thermalbildern von Wildtieren, sowie Tieren in Tierparks gesammelt.

Im vierten Semester des MTD Studiums wurden im Zuge des Semesterprojektes rund 3000 Bilder aus den gesammelten Daten mithilfe des Online-Tools CVAT annotiert und klassifiziert. Dieser Datensatz umfasst zum größten Teil Bilder von Reh- und Rotwild, aber auch einige andere Tiere wie Wildschweine, Hasen, Damwild, etc. Auf diesen Daten wurden mehrere unterschiedliche Object Detection (OD) Modelle trainiert, um die automatische Erkennung der Tiere zu ermöglichen.

Als Weiterführung dieses Semesterprojektes entstand die vorliegende Bachelorarbeit, um ein Modell mit möglichst präzisen Echtzeit-Detektionen zu implementieren. In dieser Arbeit werden zwei State-of-the-Art OD-Modelle analysiert.

Als erstes wurde das YOLO-V8-Modell von Ultralytics [68] implementiert, welches bereits im Rahmen des Semesterprojekts die besten Ergebnisse unter den implementierten Modellen erzielt hat.

Das zweite Modell ist das YOLO-NAS von DECI [80], welches erst im Mai 2023 veröffentlicht worden ist und nach Angaben von DECI noch bessere Detektionsleistungen als YOLO-V8 erreichen soll, insbesondere bei der Erkennung kleiner Objekte. Dies erscheint vielversprechend, da die Wildtiere in den Thermalbildern oft nur aus wenigen Pixeln zusammengesetzt sind. Die Annahme dieser Arbeit war deshalb, dass das YOLO-NAS-Modell eine noch bessere Performance als das YOLO-V8-Modell aufweisen sollte.

Aus diesen Ansätzen hat sich folgende **Forschungsfrage** ergeben:

**Welches Echtzeit YOLO Object Detection Modell erzielt die besten Ergebnisse bei der Erkennung von Wildtieren auf Drohnen-Thermalbildern?**

## 1.2 Struktur der Arbeit

Die vorliegende Arbeit ist in mehrere aufeinander aufbauende Kapitel gegliedert.

Im ersten Kapitel, **Supervised Machine Learning** (Kapitel 2), werden die grundlegenden Funktionsmechanismen von Machine Learning und Neural Networks erläutert. Dabei werden die verschiedenen Techniken analysiert, die es einem Neural Network ermöglichen, aus Daten zu lernen. Zudem werden unterschiedliche Ansätze zur Konfiguration dieser Netzwerke vorgestellt und verschiedene Modellarchitekturen sowie deren Anwendungsbereiche beleuchtet.

In weiterer Folge liegt der Fokus auf dem Thema **Object Detection** (Kapitel 3). Dabei werden die Grundlagen des Trainings von Objekterkennungsmodellen sowie die dafür benötigten Daten behandelt. Zudem erfolgt eine historische Einordnung der Entwicklung von Objekterkennungsmodellen. Zuletzt werden die Architekturen und die Besonderheiten der beiden, in dieser Arbeit verglichenen Modelle YOLO-V8 und YOLO-NAS analysiert.

Im Kapitel **Methodik** (Kapitel 4) wird die praktische Vorgehensweise dieser Arbeit beschrieben, angefangen von der Datenerhebung und -annotation, sowie Anwendung von Augmentationstechniken. Anschließend wird die Implementierung, das Training und die Evaluierung der Object Detection Modelle erklärt.

Danach folgt der **Vergleich** (Kapitel 5), in dem die Ergebnisse aus den Trainingsprozessen der beiden Modelle präsentiert, analysiert und verglichen werden. Dabei wird besonders auf die Metriken Präzision und Geschwindigkeit der beiden Modelle eingegangen.

Abschließend erfolgt in der **Diskussion** (Kapitel 6) eine eingehende Betrachtung der Ergebnisse, bei der sowohl die Erkenntnisse herausgearbeitet als auch die Limitationen der durchgeführten Arbeit aufgezeigt werden.

# Kapitel 2

# Supervised Machine Learning

Allgemein befasst sich Machine Learning mit der Entwicklung von Algorithmen und Modellen, die es Computern ermöglichen, aus Daten zu lernen und Vorhersagen oder Entscheidungen zu treffen, ohne explizit programmiert zu werden. Es basiert auf dem Konzept, dass Maschinen durch das Analysieren von Daten Muster und Zusammenhänge erkennen können, um dann auf neue, unbekannte Daten zu reagieren [78].
Hierbei gibt es verschiedene Arten des Lernens, darunter:

- **Supervised Learning**: Dabei werden Modelle auf der Grundlage von annotierten Trainingsdaten entwickelt, um Vorhersagen für neue, unannotierte Daten zu treffen. Typische Anwendungen sind Klassifikation und Regression. Diese Art des Lernens wird auch für die Object Detection Modelle, die in dieser Arbeit verwendet werden, angewandt.
- **Unsupervised Learning**: Hierbei werden Modelle entwickelt, um Muster in unannotierten Daten zu entdecken, wie beispielsweise Clustering und Dimensionalitätsreduktion.
- **Reinforcement Learning**: Hier agiert ein Agent, der in Wechselwirkung mit einer Umgebung steht und ein spezifisches Ziel verfolgt. Sein Verhalten führt entweder zu Bestrafungen oder Belohnungen, wodurch der Agent seine Parameter entsprechend anpasst, um den Belohnungswert zu maximieren und das Ziel zu erreichen [88].

## 2.1 Neural Networks

Die Grundlage auf der Machinelearning basiert sind Neural Networks (NN). Neuronale Netzwerke sind Software Strukturen welche von natürlichen Nervenstrukturen inspiriert sind. Sie bestehen aus Schichten (Layers) von Knoten (Neuronen) die miteinander verbunden sind. In diesem Kapitel wird hierbei auf die klassische Struktur von NNs im Kontext von Supervised Machine Learning eingegangen. Hierbei bestehen NNs aus einer Input Schicht, versteckten Schichten (Hidden Layers), sowie einer Output Schicht (siehe Abb. 2.1).

Bei dem sogenannten **Forward Pass** berechnet das Netzwerk aus gegebenen Input-

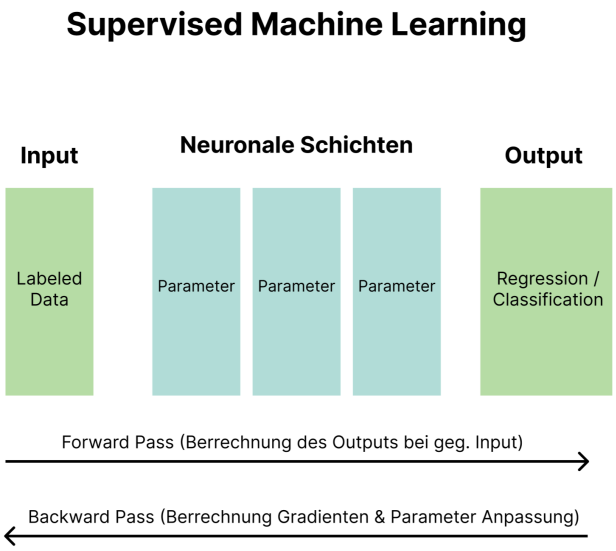


**Abbildung 2.1:** Schematische Darstellung - Supervised Machine Learning

werten (z.B. RGB-Werte eines Bildes) neue Outputwerte (z.B. Klasse und Begrenzungsrahmen eines erkannten Objektes). Ein einzelnes Neuron bekommt dabei in der Regel viele Inputwerte. Diese werden jeweils mit einem spezifischem erlernbaren Weight (Gewicht) multipliziert und anschließend summiert. In weiterer Folge wird ein Neuronenspezifischer und erlernbarer Bias dazu addiert. Zuletzt wird das Ergebnis durch eine Activation function (siehe Kapitel 2.2) aktiviert um die Nonlinearität des Modells zu gewährleisten (Übersicht siehe Abb. 2.2). Der Output des Neurons wird als neuer Input an die nächste Neuronen Schicht weitergegeben. Wenn alle Outputs von einer Layer mit allen Inputs der nächsten Layer verknüpft sind spricht man auch von einer „Fully Connected Layer“ [82].

Um ein Neural Network zu trainieren, bekommt dieses (im Fall von Supervised Learning) annotierte Trainingsdaten als Input und produziert anfangs einen zufälligen Output. Eine Lossfunction ergründet wie weit der produzierte Output von dem gewünschten Output entfernt ist. Anschließend werden im **Backward Pass** Gradienten berechnet, welche aussagen wie die Parameter angepasst werden müssen um den Loss zu minimieren. Durch den Prozess des „Gradient Descents“ werden anschließend, nach Definition des Optimisers (siehe Kapitel 2.2), bei jeder Iteration die Weights und Biases der einzelnen Neuronen angepasst. Dadurch beginnt das NN kontinuierlich zu lernen und bessere Voraussagen zu treffen. Wenn ein neuronales Netzwerk eine große Anzahl solcher Schichten in Sequenz anordnet, spricht man auch von einem Deep Neural Network (Tiefes neuronales Netzwerk).

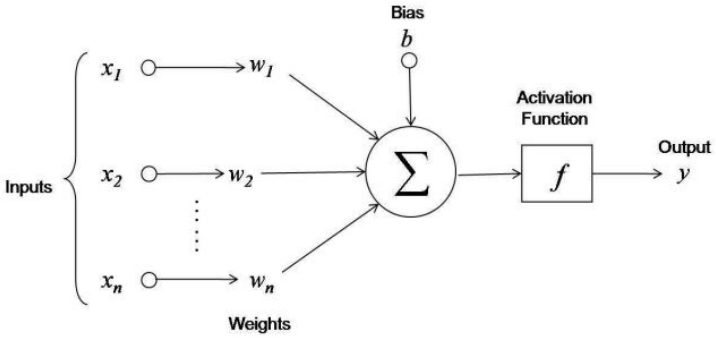


**Abbildung 2.2:** Berechnung des Outputs eines einzelnen Neurons, Quelle: [60]

## 2.2 NN - Konfigurationen und Techniken

### Activation Function

Die Verwendung einer Activation Function ist unerlässlich, da sie eine Nonlinearität in das Netzwerk einführt. Ohne der Verwendung einer Aktivierungsfunktion würde lediglich eine lineare Berechnung stattfinden, und sämtliche Operationen, vom Eingang bis zum Ausgang des neuronalen Netzwerks, könnten auf eine einzelne Matrixmultiplikation reduziert werden. Die Aktivierungsfunktion ermöglicht es dem neuronalen Netzwerk erst, komplexe Muster und nichtlineare Zusammenhänge in den Daten zu erfassen [6]. Hierfür können unterschiedliche Funktionen verwendet werden. Heute wird häufig die ReLU-Funktion (Rectified Linear Unit) [35] angewandt, welche alle negativen Werte auf 0 setzt und positive Werte beibehält.

Eine weitere Activation Function die in dieser Arbeit Anwendung findet ist die SiLU-Funktion (Sigmoid-Weighted Linear Unit) [7] welche sich ähnlich wie ReLU verhällt, aber weichere Übergänge aufweist, und ein kleines Spektrum ein negativen Werten zulässt (siehe Abb. 2.3).

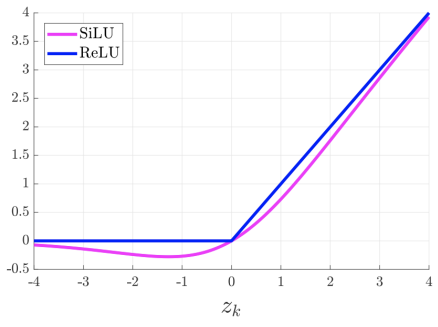


**Abbildung 2.3:** Vergleich der ReLU und SiLU Activation Functions, Quelle: [59]

### Learning Rate

Die Learning Rate (LR) ist vermutlich der wichtigsten Hyperparameter für das Training von NNs [16] und übt einen sehr großen Einfluss auf den Trainingsprozess aus. Die LR bewegt sich im Bereich zwischen 1 und 0 und steuert, wie stark die Gewichte bei jedem Trainingsschritt angepasst werden. Die Wahl einer optimalen LR hängt dabei auch von dem verwendeten Optimiser ab.

Bei einer hohen LR erfolgt eine stärker Anpassung der Gewichte. Allerdings kann eine zu hohe LR dazu führen, dass das gesuchte Minimum der Verlustfunktion im Verlauf des Gradientdescents übersprungen wird. Eine zu niedrige LR hingegen kann bewirken, dass der Trainingsprozess nur sehr langsam fortschreitet, wie in Abb. 2.4 ersichtlich [53, 77].

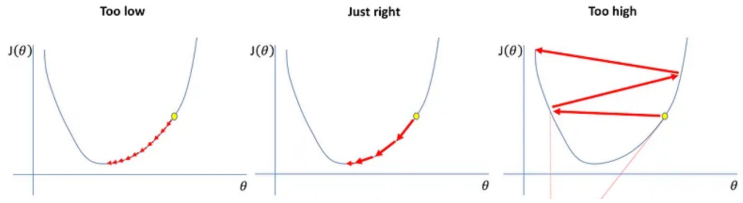


**Abbildung 2.4:** Learning Rate - Gradient Descent. Quelle: [61]

Die Learning Rate muss jedoch nicht über den gesamten Trainingsprozess konstant bleiben. Oft wird sie während des Trainings dynamisch angepasst, um sowohl die Trainingsdauer als auch die Leistung des Modells zu optimieren. Ein klassischer Ansatz besteht darin, die LR kontinuierlich zu reduzieren. Dabei werden in der Regel eine Start-LR sowie eine End-LR oder eine Reduzierungsmenge und ein Intervall festgelegt. Durch diese Vorgehensweise können Modelle zu Beginn des Trainings schnell grobe Merkmale in den Trainingsdaten erlernen und im weiteren Verlauf immer feinere Details erfassen. Dabei kann die Reduzierung der LR linear oder nach einer zyklischen Funktion (z.B. Cosinus) erfolgen [53].

### Optimizer

Der Optimizer ist der Algorithmus der während des Trainingsprozesses verwendet wird, um die Modellparameter so anzupassen, dass der Verlust (Loss) minimiert wird. Das Ziel des Optimizers besteht darin, die besten Werte für die Weights und Biases des Modells zu finden [44][86].

**Stochastic Gradient Descent (SGD)** ist ein klassischer Optimiser, der seit langem in Machine Learning verwendet wird und nach wie vor Anwendung findet. SDG funktioniert, indem er iterativ die Modellparameter anpasst, um den Verlust zu minimieren, indem er sich entlang des Gradienten der Verlustfunktion bewegt. Auch wenn SGD einfacher ist und weniger Ressourcen erfordert als einige der fortgeschritteneren Optimierungsalgorithmen, kann er manchmal langsamer konvergieren und anfälliger für lokale Minima sein.

Ein später entwickelter Optimiser ist **Adam (Adaptive Moment Estimation)** [25], der heutzutage einer der am häufigsten angewandten Optimierungs-Algorithmen

ist. Adam kombiniert die Vorteile von Momentum (siehe nächstes Kapitel) und RMSprop (ein nicht veröffentlichter Algorithmus von Geoff Hinton), um die Learning Rate für jeden Parameter dynamisch anzupassen und eine schnelle Konvergenz zu ermöglichen. Dieser Ansatz macht Adam besonders effektiv in der Praxis und ermöglicht eine effiziente Optimierung von neuronalen Netzwerken. [44]

### Momentum

Anstatt die Modellparameter ausschließlich aufgrund des aktuellen Gradienten zu aktualisieren, führt Momentum einen Geschwindigkeitsterm ein, der die akkumulierten Gradienten aus vorherigen Schritten berücksichtigt. Dies ermöglicht dem Optimiser, die Learning Rate für spezifische Weights zu erhöhen und den Lernprozess voranzutreiben, wenn Gradienten über mehrere Iterationen hinweg in die selbe Richtung zeigen. Das führt zu einer schnelleren Konvergenz und überwindet Probleme wie Oszillationen oder langsamen Konvergenzprozess, die bei herkömmlichem SGD auftreten können. Zusammengefasst verbessert Momentum im maschinellen Lernen die Effizienz und Geschwindigkeit der Konvergenz während des Modelltrainings, indem Informationen über die historischen Gradienten der Verlustfunktion berücksichtigt werden [46].

### Weight decay

Weight decay ist eine Technik die dazu dient, Overfitting zu reduzieren, indem die Größe der Weights während des Trainings eingeschränkt wird. Es ist auch als L2-Regularisierung oder Gewichtsnormalisierung bekannt.

Bei Weight Decay wird der Verlustfunktionsausdruck um einen zusätzlichen Term erweitert, der proportional zum Quadrat der Gewichte ist.Dieser Term wirkt als Strafe für große Gewichtswerte und zieht die Parameter in Richtung kleinerer Werte. Dies trägt dazu bei, Overfitting zu vermindern [46].

### Residual Connections

Umso tiefer Neurale Netzwerke werden, umso schwerer wird es, diese zu trainieren und beim Training eine Konvergenz zu erreichen. Dies liegt oft an dem Vanishing oder Exploding Gradient Problem, bei welchem die berechneten Gradienten in der Backpropagation gegen 0 gehen bzw. extrem groß werden. Durch sogenannte Residual Connections kann dieses Problem minimiert werden. Dabei werden die Outputs einer Schicht als Eingang einer späteren Schicht weitergeleitet, wobei ein oder mehrere Layers dabei übersprungen werden. Dies ermöglicht es, auch bei sehr tiefen Netzwerken relevante Informationen effektiver zu übertragen und zu erhalten. [18]

### Batch Normalization

Batch Normalization wurde 2015 von S. Ioffe et al. [23] eingeführt und ist in den heutigen Object Detectors omnipräsent vorhanden. Sie funktioniert durch die Normalisierung der Schicht Outputs über die Mini-Batches während des Trainings. Dadurch wird die Eingabe für jede Schicht in einer ähnlichen Größenverteilung gehalten, was dazu beiträgt, das Problem des „Internal Covariate Shift“ zu reduzieren. Diese Methode stabilisiert und beschleunigt das Training und ermöglicht die Verwendung höherer Learning Rates.

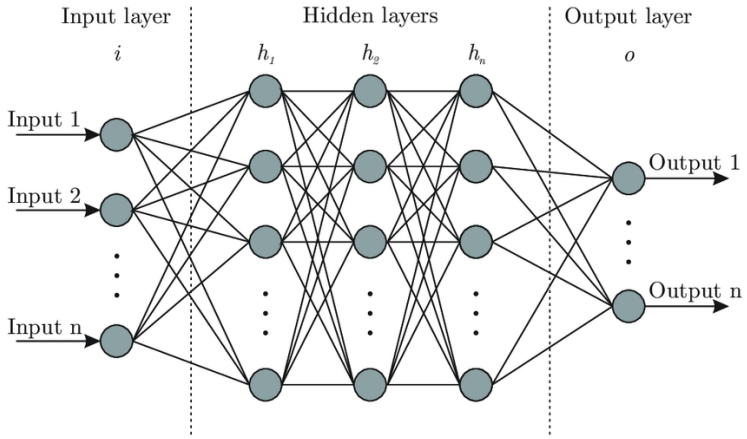


**Abbildung 2.5:** Der Aufbau eines Multi Layer Perceptron, Quelle: [58]

## 2.3 Modellarchitektur

In den letzten Jahren wurden zahlreiche NN-Architekturen entwickelt und optimiert, um unterschiedlichste Aufgaben in verschiedenen Anwendungsgebieten zu lösen. Die Wahl der richtigen Architektur hängt oft von der spezifischen Aufgabe und den vorhandenen Daten ab und spielt eine entscheidende Rolle bei der Leistungsfähigkeit von ML-Modellen. Der Fokus in diesem Kapitel liegt auf Architekturen die für den Themenbereich Object Detection (siehe Kapitel 3) relevant sind.

### 2.3.1 Multi Layer Perceptron

Ein Multilayer Perceptron (MLP) ist eine Weiterentwicklung des ursprünglichen Perceptron-Modells, das von Rosenblatt in den 1950er Jahren vorgeschlagen wurde [43].

Der MLP bezeichnet ein vorwärts gerichtetes künstliches neuronales Netzwerk, bestehend aus mindestens 3 vollständig verbundenen Neuronalen Schichten. Der MLP lässt sich in eine Input Layer, einer oder mehreren Hiden Layers und einer Output Layer gliedern [37].

Im Object Detection Kontext ist der MLP eine relativ primitive Architektur. Die Modellarchitektur versucht keine räumliche Merkmale in Bildern zu erkennen. Die Pixelwerte-Inputwerte werden einfach in einer flatten-Layer zu einem 1-Dimensionalen Vektor verarbeitet. Das Prinzip eines MLP wird aber auch noch heute oft beim Klassifizierungsteil (auch Head genannt) von Objectdetection Modellen angewandt.

### 2.3.2 Convolutional Neural Network (CNN)

CNNs haben Image Classification, sowie Object Detection revolutioniert.

Die ersten CNN Architekturen wurden bereits 1998 von Yan Lecun entwickelt [27]. Damals fehlte aber noch die Rechenleistung moderner GPUs um Objectdetection Modelle effektiv trainieren und einsetzen zu können.

Ein CNN besitz einige Unterbestandteile:

- **Convolutional layer**: In einer Convolutional layer werden die Eingangsdaten

mit einem parametrisierten Kernel multipliziert. Die Parameter des Kernel werden beim Trainingsprozess erlernt. Ein CNN kann eine sehr hohe Anzahl an Convolutional layers haben. Somit sind unterschiedliche Kernels verantwortlich um unterschiedliche Eigenschaften eines Bildes zu extrahieren. Durch die Convolution werden auch die Dimensionen des Bildes, je nach Kernelsize und Stride reduziert.

- **Pooling Layer**: Pooling-Layers kommen zum Einsatz, um eine Dimensionsreduzierung der Eingangsdaten zu erzielen und damit den Rechenaufwand zu minimieren, während gleichzeitig die essenziellen Informationen des Bildes bewahrt werden sollen. Zudem stellen Pooling-Layers eine effektive Maßnahme zur Reduzierung von Overfitting dar. Es existieren verschiedene Arten von Pooling-Layers, wobei Max Pooling eine besonders häufig verwendete Methode ist. Bei dieser durchläuft ein Kernel die Feature Map aus der vorangegangenen Convolution, wobei ausschließlich die maximalen Pixelwerte beibehalten werden. Ein Beispiel hierfür wäre die Anwendung eines 2x2 Kernels mit einer Stride von 2, was zu einer Halbierung der Bildlänge führt [13].

Bei modernen CNNs werden meist sehr viele Convolutional Layers aneinandergereiht. Anschließend kann ein einfacher Multi Layer Perceptron folgen, welcher die Outputs des CNNs erhält und in weiterer Folge die Klassifizierung durchführt.

Im Vergleich zu einfachen feed-forward Neural Networks (z.B. MLP) haben CNNs wesentlich weniger Neuronale Verbindungen und Parameter wodurch sie auch einfacher zu trainieren sind [26].

### 2.3.3 Transformer

Ein Transformer ist eine Machine-Learning-Architektur, die erstmals in der im Jahr 2017 veröffentlichten Arbeit „Attention Is All You Need“ [49] vorgeschlagen wurde.

Die Transformer-Architektur hat besonders im Bereich des Natural Language Processing (NLP) maßgebliche Fortschritte verzeichnet. Transformer-Modelle zeichnen sich durch herausragende Leistungen in verschiedensten sprachbezogenen Aufgaben aus, darunter Textklassifikation, maschinelles Übersetzen und Fragebeantwortung. Unter den prominentesten Modellen dieser Kategorie finden sich das Bidirectional Encoder Representations from Transformers (BERT)[5] sowie das Generative Pre-trained Transformer (GPT)[2].

Transformer Modelle wurden entwickelt um Sequence-to-Sequence Aufgaben zu lösen. Diese Modelle werden dazu verwendet, eine Sequenz von Daten in eine neue Sequenz zu transformieren, beispielsweise bei Übersetzungsaufgaben. Ein entscheidender Vorteil der Transformer-Architektur liegt in ihrer Fähigkeit zur Parallelisierung und die daraus folgende Skalierbarkeit. Durch den dabei angewandten Self-Attention Mechanismus können Transformer im Gegensatz zu RNNs, die Beziehungen zwischen Elementen über längere Sequenzen hinweg erfassen und somit auch weiter reichende Abhängigkeiten erlernen. Durch den großen Erfolg den Transformer in Large Language Models (LLMs) erreicht haben wurde die Architektur auch bald angepasst um für Object Detection Aufgaben verwendet werden zu können. Somit entstanden die sogenannten Vision Transformer [24] welche häufig eine ausgezeichnete Präzision aufweisen aber tendenziell langsamer sind als CNN basierte Single Stage Detektoren.

# Kapitel 3

# Object Detection

Object Detection (OD) ist ein Teilbereich des Feldes Computer Vision. Ziel von OD-Techniken ist es Instanzen von bestimmten Objekten in Bildern und Videos zu lokalisieren und zu klassifizieren. Bei der Erkennung eines Objektes erhält dieses eine Klasse (z.B. Auto), Koordinaten einer Boundingbox (Begrenzungsrahmen), sowie einen Confidence Score zwischen 1 und 0, welcher beschreibt wie sicher sich das Modell in der Detektion ist. Dabei wird heute üblicherweise Machinelearning angewandt um ein Detektions-Modell zu trainieren um Objekte in unterschiedlichsten Perspektiven, Lichtverhältnissen und Skalierungen erkennen zu können.[85].

Beim Training von Object Detection Modellen gibt es zwei unterschiedliche Ansätze:

- **Normales Training**: Dabei werden die Weights und Biases eines Models mit einem mehr oder weniger zufälligen Wert initialisiert. Hierfür braucht man eine große Menge an Trainingsdaten und eine lange Trainingsdauer, damit das Modell gut Ergebnisse erzielt.
- **Transfer Learning**: Hier werden die Weights und Biases von einem Modell genommen, welches bereits an einem großen allgemeinen Datensatz (z.B. MS COCO - siehe Kapitel 3.1.1) trainiert wurde. Durch einen fine-tuning Prozess wird dieses Modell dann auf die eigenen Trainingsdaten angepasst. Dieser Ansatz hat den Vorteil, dass man bereits mit weniger Trainingsdaten und Trainingsdauer gute Ergebnisse erzielen kann [85].

## 3.1 Trainingsdaten

Die Grundlage eines effektiven Object Detection Modells liegt maßgeblich in der Qualität und Quantität der zugrunde liegenden Daten. Ein effektiver und präziser Algorithmus kann nur dann sein volles Potenzial entfalten, wenn er mit umfassenden, aussagekräftigen und repräsentativen Datensätzen trainiert wird [17].

### 3.1.1 Etablierte Datensätze

In der Object Detection gibt es einige große Datensätze die wiederholt verwendet werden. Diese sind nützlich dafür die Präzision verschiedener Modelle vergleichen zu kön-

nen. Beim Transferlearning werden ebenfalls oft die Weights eines Modelles verwendet, welches bereits auf einem dieser großen Datensätze trainiert wurde.

- **ImageNet** Der ImageNet-Datensatz [4] enthält etwa 14 Millionen annotierte Bilder wobei davon rund 1 Million mit Boundingboxen versehen ist. Diese Bilder beinhalten 200 unterschiedliche Objektklassen welche wiederum in Unterkategorien aufgeteilt sind. Dabei sind die Klassen Substantive die nach der WordNet-Hierarchie [9] ausgewählt wurden. Zu jeder Klasse gibt es im Schnitt mehr als 500 Bilder [90].

  Von 2010 bis 2017 wurde der Datensatz jährlich in der ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [45] verwendet, ein Wettbewerb für die Klassifizierung und Objekterkennung von Bildern. Das ImageNet-Projekt besitzt nicht das Urheberrecht an den Bildern. Daher werden nur Miniaturansichten und URLs der Bilder bereitgestellt [83].
- **MS COCO** Das MS COCO (Microsoft Common Objects in Context) ist ein sehr umfangreicher Datensatz für die Erkennung, die Segmentierung und die Schlüsselpunkterkennung von Objekten. Der Datensatz besteht aus 328.000 Bildern mit 80 verschiedenen Objektkategorien [31]. MS COCO hat weniger Objektkategorien als ImageNet, dafür aber mehr Objektinstanzen. Im Vergleich zur ILSVRC besteht der bedeutendste Fortschritt von MS-COCO darin, dass jedes Objekt neben den Boundingboxen zusätzlich mit einer instanzbasierten Segmentierung versehen ist, um eine präzise Lokalisierung zu unterstützen. Darüber hinaus enthält MS-COCO mehr kleine Objekte (deren Fläche kleiner als 1% des Bildes ist) und dichter platzierte Objekte. Ähnlich wie ImageNet in seiner Zeit ist MS-COCO zum De-facto-Standard im Objectdetection Bereich geworden [56].

  Die YOLO-V8 und YOLO-NAS Modelle die in dieser Arbeit verwendet werden, wurden ebenfalls ursprünglich mit dem COCO Datensatz trainiert.

### 3.1.2 Datensatz Splitting

Beim Training eines OD-Modelles ist es äußerst wichtig die Leistung dieses gut beurteilen zu können. Wenn alle vorhandenen annotierten Daten zum Training des Modelles verwenden werden, ist es nicht möglich die Performance und Generalisierbarkeit des Modelles zu beurteilen. So kann ein Modell bei dem häufigen Problem der Überanpassung (Overfitting) zwar äußerst gute Resultate auf den Trainingsdaten erzielen, zeigt dabei jedoch oft eine schlechte Leistung bei neuen, bisher ungesehenen Daten. Um effektive Modelle zu trainieren, ist es deshalb gängige Praxis, die vorhandenen Daten in drei Teile aufzuteilen: den Trainingsdatensatz , den Validierungsdatensatz und den Testdatensatz [87].

- **Trainingsdatensatz**: Dieser Datensatz wird verwendet, um das Modell zu trainieren und dessen Parameter anzupassen. Das Modell lernt aus diesen Daten, Muster zu erkennen und Beziehungen zwischen den Eingangsvariablen und den Zielvariablen zu verstehen.
- **Validierungsdatensatz**: Der Validierungsdatensatz wird verwendet, um die Performance des Modells während des Trainingsprozesses und bei der Verwendung unterschiedlicher Konfigurationen zu überprüfen. Dieser Schritt hilft dabei, Over-

fitting zu vermeiden, indem die Modelle nicht nur auf den Trainingsdaten, sondern auch auf unabhängigen Daten bewertet werden.

- **Testdatensatz**: Der Testdatensatz dient dazu, die endgültige Leistung des ausgewählten Modells zu evaluieren. Dieser Datensatz sollte dem Modell während des Trainings und der Validierung unbekannt sein. Das Ziel ist es sicherzustellen, dass das Modell in der Lage ist, neue, nicht zuvor gesehene Daten gut vorherzusagen, und nicht nur auf die Daten, auf denen es trainiert und validiert wurde, optimiert ist [38].

### 3.1.3 Datenaugmentation

Wenn es darum geht, OD-Modelle für spezifische Anwendungsfälle zu trainieren, steht oft nur eine begrenzte Menge an Daten zur Verfügung. Diese Datenmenge ist häufig unzureichend, um ein präzises und gut generalisierbares Modell zu trainieren. Eine Strategie zur Bewältigung dieses Problems besteht darin, Datenaugmentation einzusetzen, wodurch bereits mit einem begrenzten Datensatz bessere Ergebnisse erzielt werden können. Datenaugmentationen sind Transformationen durch die die Trainingsbilder verändert und vervielfacht werden [81]. Dabei gibt es verschiedene Arten von Augmentierungen.

- **Farbanpassungen**: Dabei werden die RGB Werte der Trainingsdaten angepasst. So können Änderungen an der Sättigung, Helligkeit, Belichtung und Farbton angewandt werden. Ebenso können Trainingsbilder zu Schwarz-Weiß Bildern geändert werden.
- **Geometrische Transformationen**: Hierbei werden geometrische Transformationen auf die Trainingsdaten angewandt. Dazu gehören Rotationen, Skalierungen und Perspektivische Verzerrungen.
- **Boundingbox Augmentationen**: Bei Boundingbox Augmentationen werden die bereits erwähnten Augmentationen wie Farbanpassungen und Geometrische Transformationen angewandt. Dabei werden diese aber nur auf die Objektinstanz innerhalb der Boundingbox angewandt, der Rest des Bildes bleibt unverändert
- **Andere Augmentationen**: Andere Augmentationen die verwendet werden sind Filter (z.B. Schärfe oder Blur), Einführen von Noise, Mosaic-Augmentation (mehrere Bilder werden zusammengekachelt) und das Ausschneiden von Bildinhalten (Cutout).

Umso kleiner die Menge der ursprünglichen Trainingsdaten ist, umso mehr kann die Präzision durch das Anwenden von Augmentationen gesteigert werden. Ebenso können durch Augmentationen insbesondere bessere Ergebnisse erzielt werden, wenn die Objektinstanzen schwer zu erkennen und klein sind [55].

Die Augmentierungen können dabei entweder „online“, also während des Trainingsprozesses angewandt werden, oder bereits vor dem Trainingsprozess auf den gesamten Trainingsdatensatz appliziert werden.

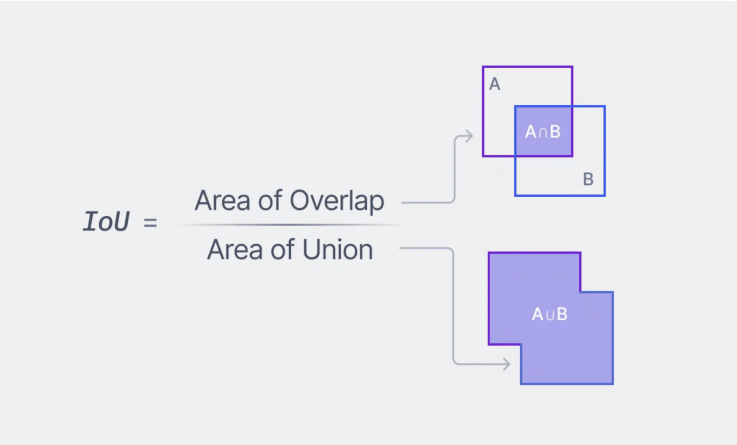


**Abbildung 3.1:** Intersection over Union, Quelle: [62]

## 3.2 Auswertung

### 3.2.1 Metriken

Um die Qualität der Detektionen eines OD-Modelles zu beurteilen wurden einige aussagekräftige Metriken entwickelt. Um diese Metriken zu berechnen müssen allerdings zuerst einige Grundsätze definiert werden.

Generell gibt es vier verschiedene Ergebnisse zu denen eine einzelne Detektion bei der Erkennung eines Objektes resultieren kann:

- **True Positive (TP):** Korrekte Identifikation der richtigen Klasse.
- **False Positive (FP):** Fälschliche Identifikation der Klasse. (Es wird etwas erkannt, obwohl nichts vorhanden ist.)
- **True Negative (TN):** Es wird korrekterweise keine Klasse erkannt.
- **False Negative (FN):** Fälschlicherweise wird keine Klasse erkannt, obwohl ein Objekt vorhanden wäre.

Zusätzlich muss noch definiert werden wie sehr die vorhergesagte Boundingbox mit der Ground-Truth Boundingbox überlappen muss, damit diese auch als korrekte Detektion gezählt wird. Dies wird mit der Intersection over Union Grenze (IoU-Threshold) definiert. Die IoU wird berechnet, indem man den Bereich der Überlappung (Intersection) zweier Bounding-Boxen durch den Bereich der Vereinigung (Union) dieser Boxen teilt (siehe Abb. 3.1). Ein gängiges IoU-Threshold ist 0.5. Es kann aber auch ein Mittelwert von mehreren IoU-Thresholds verwendet werden (z.B. 0.5-0.95 mit allen Zwischenwerten für jeden 0.05 Schritt), um dabei einen größeren Bereich an Präzision zu erfassen [11].

Mit Hilfe dieser Definitionen können die folgenden Metriken berechnet werden um die Qualität der produzierten Detektionen zu bewerten.

- **Recall:** Der Recall sagt aus wie viele der vorhandenen Objekte erkannt wurden. Dabei wird keine Rücksicht darauf genommen ob es falsche Erkennungen gibt.
  $Recall = \frac{TP}{TP+FN}$
- **Precision:** Die Precision misst wie viele der erkannten Objekte korrekte Erkennungen sind, berücksichtigt im Gegensatz aber nicht ob alle vorhandenen Objekte erkannt wurden.
  $Precision = \frac{TP}{TP+FP}$

Wenn die Qualität des Modells in einer Kennzahl festhalten werden soll, muss dabei sowohl Precision als auch Recall berücksichtigt werden. Dafür wird eine Precision-Recall-Kurve erstellt, die für sämtliche Detektionsbeispiele sowohl die Precision als auch den Recall abbildet. Die Fläche unter dieser Kurve, die Average Precision (AP), dient als Maß für die Modellgenauigkeit. Wenn dieser Vorgang für alle Klassen wiederholt und der Durchschnitt gebildet wird, ergibt sich die Mean Average Precision (mAP). Diese Metrik eignet sich sehr gut, um die Gesamtqualität des Modells zu charakterisieren und ermöglicht einen fundierten Vergleich mit anderen Modellen [20].

### 3.2.2 Non Maximum Supression

Derzeitige Objekt Detektoren sind nicht in der Lage, genau eine Erkennung pro Objekt zu produzieren. Jedes Objekt löst somit mehrere Erkennungen mit unterschiedlicher Confidence Scores aus, abhängig davon, wie gut die vorhergesagte Boundingbox das Objekt abdeckt.

Non Maximum Supression (NMS) ist ein Post-Processing-Algorithmus der dafür verantwortlich ist, alle Erkennungen zusammenzuführen, die demselben Objekt gehören. Der vorherrschende Algorithmus der hierfür verwendet wird ist GreedyNMS. Dieser akzeptiert die Vorhersage mit der höchsten Confidence und lehnt anschließend alle Erkennungen ab, welche sich mit dieser über einen bestimmten Schwellwert $\theta$ hinaus überlappen. Dieses Verfahren wird mit den verbleibenden Voraussagen wiederholt. NMS ist somit entscheidend, um überlappende Bounding Boxes zu reduzieren und die Anzahl der relevanten Objekte präzise zu identifizieren. [21]

## 3.3 Historische Entwicklung - Objectdetection

Der Fortschritt in der Objekterkennung wird meist in zwei historischen Perioden beschrieben: Die traditionelle Objekterkennung (vor 2014) und die auf Deep Learning basierende Objekterkennung (nach 2014) [56].

### 3.3.1 Traditionelle Detektoren

In der Zeit, in der traditionelle Detektoren entstanden, war die Leistung von Computer Prozessoren noch sehr begrenzt. Deshalb wurden die meisten frühen Objekterkennungsalgorithmen auf der Grundlage von manuell erstellten Merkmalen entwickelt. Aufgrund des Mangels an effektiver Bildrepräsentation mussten damals komplexe Merkmalsrepräsentationen und verschiedene Beschleunigungstechniken entworfen werden [56].

#### Viola Jones Algorithmus

Im Jahr 2001 erreichten P. Viola und M. Jones mit ihrem **Viola-Jones-Algorithmus** [50] zum ersten Mal eine Echtzeit-Erkennung von menschlichen Gesichtern ohne Einschränkungen (z.B. Hautfarbenerkennung). Der Viola-Jones-Detektor nutzt das sogenannte „sliding windows"-Verfahren, bei dem alle möglichen Positionen und Skalierungen in einem Bild durchlaufen werden, um zu überprüfen, ob ein Fenster ein menschliches Gesicht enthält. Dabei sucht der Algorithmus im Wesentlichen nach 'haarähnlichen' Merkmalen (benannt nach Alfred Haar, der das Konzept der Haar-Wavelets entwickelte) [76][56].

#### HOG Detector

Der HOG (Histogram of Oriented Gradients) Detector[3] wurde 2005 von N. Dalal and B. Triggs, zur Erkennung von Passanten entwickelt und erzielte damals die besten Detektionen. HOG basiert darauf die horizontalen und vertikalen Gradienten zwischen Pixelwerten zu berechnen. Anschließend wird das verarbeitete Bild in kleine Zellen aufgeteilt und für jede Zelle wird einen Histogramm der Gradientenorientierungen berechnet. Diese Histogramme werden zu Blöcken zusammengefasst und normalisiert. HOG-Merkmale sind besonders effektiv bei der Darstellung der Kanten und Strukturen in einem Bild. Dabei sind Detektionen relativ invariant gegenüber Helligkeitsänderungen, da sie auf den Gradienteninformationen basieren und nicht direkt von den absoluten Pixelintensitäten abhängen [3][84].

#### DPM

Der Deformable Part-based Model (DPM) gewann die Pascal Visual Object Classes (VOC) Challenge[8] in den Jahren 2007-2009. Ursprünglich wurde DPM von P. Felzenszwalb [10] im Jahr 2008 als Erweiterung des HOG-Detektors vorgeschlagen. Es folgt der Erkennungsphilosophie des „divide and conquer", bei der das Training als das Erlernen einer geeigneten Art der Zerlegung eines Objekts betrachtet werden kann, und die Inferenz als ein Ensemble von Erkennungen verschiedener Objektteile betrachtet werden kann. Zum Beispiel kann das Problem der Erkennung eines „Autos" auf die Erkennung seiner Fenster, Karosserie und Räder zerlegt werden [56].

### 3.3.2 Deep Learning basierte Detektoren

Das Jahr 2014 markierte den Eintritt in eine neue Ära für Detektoren, in der vermehrt auf immer komplexere Machine-Learning-Architekturen zurückgegriffen wurde. Handgefertigte Merkmalsbeschreibungen verlieren zunehmend an Bedeutung, da Detektoren vermehrt auf automatisch erlernte Merkmale aus Convolutional Neural Networks (CNNs) basieren. Obwohl CNNs bereits 1998 von Yann Lecun entwickelt wurden [27], entfalten sie erst jetzt ihr volles Potenzial, da Prozessoren erheblich an Leistung gewonnen haben. Insbesondere zeichnen sich Graphics Processing Units (GPUs) durch ihre Fähigkeit aus, große Mengen einfacher Matrixmultiplikationen effizient und parallelisiert durchzuführen – eine Aufgabe, die bei Machine-Learning-Anwendungen meist den Großteil des Rechenaufwands ausmacht.

#### CNN based Two-stage Detectors

Im Jahr 2012 erlebten Convolutional Neural Networks ein Comeback [26]. Die Fähigkeit eines tiefen faltenden Neural Networks, robuste Merkmalsrepräsentationen eines Bildes zu erlernen, übertrifft bei weitem diejenigen, die manuell entworfen werden können.

**RCNN**: Region-based Convolutional Neural Network (R-CNN) [15] wurde erstmals 2014 von R. Girshick et al. vorgestellt. Im Gegensatz zu vorherigen Ansätzen, bei denen das gesamte Bild mit einem Sliding Window Ansatz auf Objekte überprüft wurde, führt R-CNN mit der Selective Search Technik [48] eine Region Proposal Methode durch, bei der vielversprechende Regionen die ein Objekt beinhalten könnten identifiziert werden. Diese Regionen werden dann unabhängig voneinander auf Objekte hin klassifiziert. Jeder Vorschlag wird dann auf eine Bildgröße skaliert und in ein CNN-Modell überführt, das auf ImageNet vortrainiert ist, um Merkmale zu extrahieren. Anschließend werden lineare SVM-Classifiers verwendet, um die Anwesenheit eines Objekts in jedem Bereich vorherzusagen. RCNN führt zu einer signifikanten Leistungssteigerung auf VOC07, mit einer deutlichen Verbesserung der mAP von 33,7% (DPM-v5 [10]) auf 58,5%.

Weitere wichtige CNN based Two-stage Detectors, die die Umsetzung des RCNNs kontinuierlich verbessert haben, aber auf die in dieser Arbeit nicht näher eingegangen wird sind **SPP-net** [19], **Fast RCNN**[14], **Faster RCNN** [42] und **Featrue Pyamid Networks (FPN)** [30].

#### CNN-based one stage detectors

Der erste one-stage Detektor **„You only look once"(YOLO)** wurde 2015 von Redmon Joseph et al. entwickelt und 2016 auf der IEEE Conference on Computer Vision and Pattern Recognition (CVPR) veröffentlicht [41]. YOLO verfolgt ein völlig anderes Paradigma als Zwei-Stufen-Detektoren. Dabei wird ein einziges neuronales Netzwerk auf das gesamte Bild angewandt. Dieses Netzwerk unterteilt das Bild in Regionen und sagt gleichzeitig Boundingbox und Wahrscheinlichkeiten einer Objektklasse für jede Region vorher. Trotz der erheblichen Verbesserung der Detektionsgeschwindigkeit leidet YOLO unter einem Rückgang der Lokalisierungsgenauigkeit im Vergleich zu Zwei-Stufen-Detektoren, insbesondere bei kleinen Objekten. Diese Probleme wurden jedoch von nachfolgenden YOLO Modellen adressiert, auf welche im kommenden Kapitel im Detail eingegangen wird.

Ein weiterer one-stage Detektor der kurz nach YOLO-V1 veröffentlicht wurde ist der Single Shot MultiBox Detector (SSD), von Wei Liu et al. [32] welcher Anfangs noch höhere Präzision also YOLO erzielte.

## 3.4 YOLO - You only look once

### 3.4.1 YOLO-V1

YOLO-V1 [41] startete den Umbruch von Two Stage Detektoren zu One Stage Detektoren und war das erste Objectdetection Modell das echte Realtime Detektionen erzielen konnte.

YOLO-V1 vereinfachte die Objekterkennung, indem es in einem Schritt gleichzeitig alle Boundingboxes erkennt und nicht auf die Zusätzliche Berechnung von Region Pro-

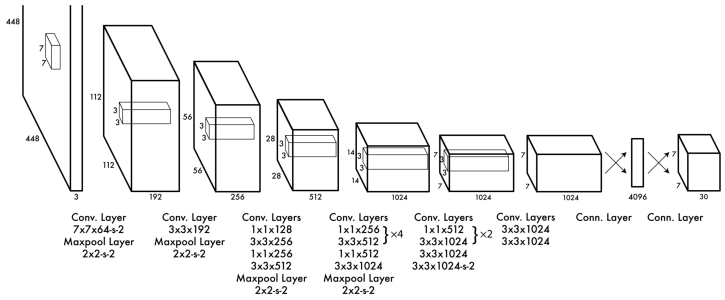


**Abbildung 3.2:** Die Architektur von YOLO-V1, Quelle: [41]

posals wie in RCNN angewiesen ist. Hierfür teilt YOLO das Eingangsbild in einen $S \times S$ Raster auf und sagt $B$ Boundingboxes pro Klasse voraus, zusammen mit einer Confidence $C$ für die unterschiedlichen Klassen pro Rasterelement. Jede Vorhersage besteht aus fünf Werten: $P_c$, $b_x$, $b_y$, $b_h$, $b_w$. Dabei ist $P_c$ der Zuversichtswert (Confidencescore), der widerspiegelt, wie sicher das Modell ist dass die Rasterzelle ein bestimmte Klasse enthält. Die $b_x$- und $b_y$-Koordinaten geben das Zentrum der Boundingbox und $b_h$ und $b_w$ die Höhe und Breite dieser an. Die Ausgabe von YOLO ist somit ein Tensor von $S \times S \times (B \times 5 + C)$. Dieser Output kann durch Non Maximum Supression (siehe Kapitel 3.2.2) modifiziert werden, um die mehrfacher Erkennungen des selben Objektes zu bekämpfen.

In der ursprünglichen YOLO Arbeit verwendeten die Autoren den PASCAL-VOC-Datensatz [8], welcher 20 Klassen enthält (C = 20); ein Raster von 7 × 7 (S = 7) und höchstens 2 Klassen pro Rasterelement (B = 2), was in einer 7 × 7 × 30-Ausgabematrix resultiert. Dabei erreichte YOLO-V1 eine mAP von 63,4 auf dem PASCAL-VOC2007-Datensatz.

Die Architektur von YOLO-V1 besteht aus 24 Convolutional Layers, gefolgt von zwei vollständig verbundenen Schichten, welche die Boungingboxes, sowie den Confidencescore vorhersagen. Dabei werden in allen, außer der letzten Schicht, leaky ReLU [34] Aktivierungsfunktionen angewandt. In Abb. 3.2 wird die Architektur von YOLO-V1 genauer dargestellt.

### 3.4.2 Weiterentwicklung

YOLO-V1 hatte zwar für die damaligen Verhältnisse ungeschlagene Detektionsgeschwindigkeiten, hinkte aber noch nach in der Präzision der Detektionen im Vergleich zu Fast-RCNN. Speziell die Erkennung kleiner Objekte verursachte Probleme für YOLO-V1.

Im Jahr 2017 veröffentlicht J. Redmon mit YOLO-V2 [39] eine verbesserte und weiterentwickelte Version von YOLO. Das Modell wurde dabei mit dem, von Redmon entwickelten C-Framework Darknet [72] implementiert. Neuartig am neuen Modell ist die dabei angewandte Batch Normalization (siehe Kapitel 2.2). Ebenso wurde das YOLO-V2 für die letzten zehn Epochen mit einer doppelt so hohen Bildauflösung von 448x448 Pixel trainiert. Eine weitere Innovation besteht darin, dass erstmals eine vollständige Convolutional-Architektur verwendet wird, ohne lineare Schichten.

Weitere Verbesserungen konnten durch das im Jahr 2018 ebenfalls von J. Redmon

veröffentlichte YOLO-V3 [40] erzielt werden.

Anstatt die Softmax Funktion für die Klassifizierung zu verwenden, wird in YOLO-V3 binary Crossentropy angewandt, um unabhängige logistische Klassifizierer zu trainieren. Diese Änderung ermöglicht es, der selben Detektion mehrere Labels zuzuweisen. Dies kann Vorteilhaft sein wenn einem Objekt eine Überkategorie (z.B. Vogel) und eine Unterkategorie (z.B. Taube) zugewiesen werden soll. YOLO-V3 verwendet dabei als erstes YOLO Modell Residual Connection zwischen den 53 Convolutional Layers, was zu einer weiteren Verbesserung der Leistung beitrug.

In weiterer Folge enstanden sehr viele unterschiedliche Single-Stage YOLO Modelle, welche allerdings nicht mehr von J. Redmon entwickelt wurden. Dazu zählen YOLO-V4 [1], YOLO-V5 [89], YOLO-V6 [28], YOLO-V7 [51], sowie YOLOR [52], PP-YOLO [33] und noch einige andere.

Dabei werden stets neue Techniken angewandt um die Effizienz der Detektoren zu steigern und die Parameter der Modelle möglichst effektiv auszunutzen. Die Beschreibung dieser Modelle würde den Rahmen dieser Arbeit sprengen deshalb wird hier vor allem auf die beiden verwendeten Modelle YOLO-V8 und YOLO-NAS eingegangen.

## 3.5 YOLO-V8

YOLO-V8 [68] wurde so wie auch schon YOLO-V5 von Glen Jocher et al. entwickelt und von der Firma Ultralytics im Jahr 2023 veröffentlicht. YOLO-V8 bietet fünf skalierte Versionen: YOLO-V8-n (nano), YOLO-V8-s (klein), YOLO-V8-m (mittel), YOLOV8l (groß) und YOLOV8x (extra groß). YOLO-V8 unterstützt mehrere Aufgaben im Bereich der Bildverarbeitung wie Objekterkennung, Objektsegmentierung, Pose-Estimation (Erkennung von Körperteil Positionen), Tracking (Objekt Verfolgung über mehrere Frames) und Klassifizierung. Das Modell wurde dabei mit dem Machine Learning Framework Pytorch [71] und der Programmiersprache Python entwickelt.

YOLOv8 verwendet eine ähnliche Backbone-Architektur wie YOLOv5, welches wiederum auf CSPDarknet53[1] von A. Bochovskiy et al. aufbaut. Die genaue Architektur ist in Abb. 3.3 abgebildet. Das YOLO-V8-L Modell besteht aus rund 50 Convolutional Layers. Dabei haben die Layers abnehmende Featuremap-Auflösung und eine zunehmende Anzahl an Filtern. Semantisch lässt sich das Modell auf einige verschiedene Blöcke aufteilen, welche mehrere Operationen durchführen.

Den größten Teil des Modells machen die Conv-Blöcke aus. Diese bestehen aus einer Convolutional Layer, welche stets von Batch-Normalisation (siehe Kapitel 2.2 sowie einer SiLU[7] Activation Function gefolgt werden. Die Conv-Blöcke im Backbone verwenden stets einen 3x3 Kernel mit einer Stride von 2 wordurch sich die Resolution nach jeder Layer halbiert.

An den Stellen im Backbone an denen die Featuremap-Resolution 80x80 sowie 40x40 beträgt sind Cross-Stage Verbindungen vorhanden, welche den Informationsfluss zwischen den Schichten fördern und speziell für die Erkennung kleinerer Objekte durch ihre höhere Resolution positiv beinflussen. An der letzten Schicht des Backbones befindet sich eine Spatial Pyramid Pooling Layer[19] welche die Verwendung unterschiedlicher Bildgrößen im Input ermöglicht.

Ein weiterer häufig verwendeter Block des Modells ist der C2f Block. Dieser wendet mehrere Konvolutionen an. Dabei werden Residual Connections zwischen Anfang und

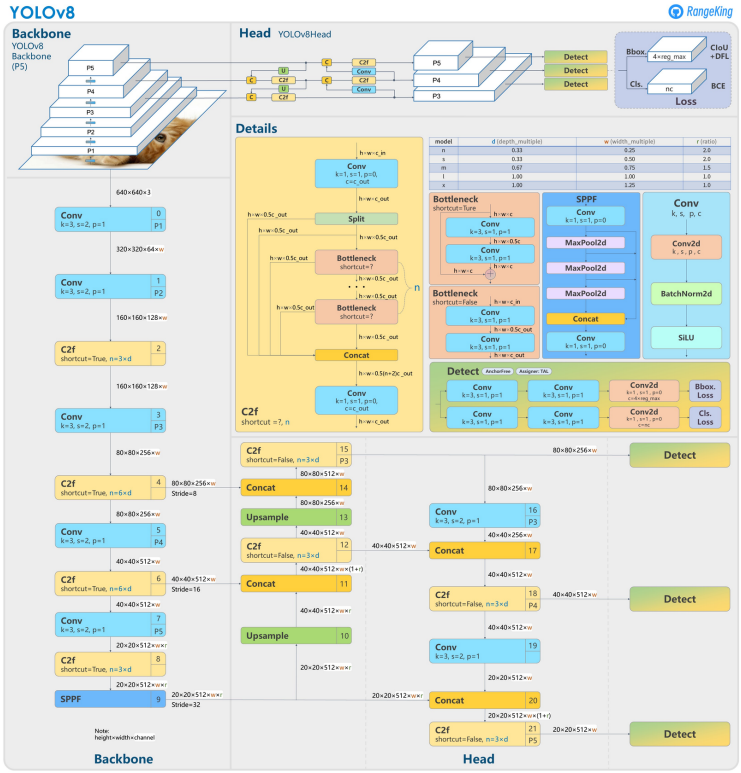


**Abbildung 3.3:** Die Architektur von YOLO-V8, Quelle: [64]

Ende des Blocks verwendet (siehe Kapitel 2.2 welche sicherstellen, dass durch große Anzahl an Schichten weniger Informationen verloren gehen und somit das Vanishing Gradient Problem [36] adressieren.

Im Head von YOLO-V8 werden anschließend die Boundingboxen sowie Klassenwahrscheinlichkeiten berechnet. Hierbei gibt es erneut Querverbindungen aus der Unterschiedlichen Outputs des Backbones, welche durch Upsample Layers and die jeweiligen Dimensionen angepasst werden.

In der Ausgabeschicht von YOLOv8 wird die Sigmoidfunktion als Aktivierungsfunktion für den Objectness Score verwendet, welcher die Wahrscheinlichkeit repräsentiert, dass die Boundingbox ein Objekt enthält. Es wird die Softmax-Funktion um die Klassenwahrscheinlichkeiten, die die Wahrscheinlichkeiten der Objekte darstellen, die zu jeder möglichen Klasse gehören.

YOLOv8 verwendet die CIoU [54] und DFL [29] Verlustfunktionen um den Boundingbox-Loss zu berechnen, sowie binäre Kreuzentropie für den Classification-Loss.

YOLO-V8 bietet einige besondere Features im Vergleich zu üblichen OD-Modell Implementierungen. So werden bei YOLO-V8 standardmäßig und automatisch einige Daten Augmentierungen, wie Farbanpassungen, Translationen, Skalierungen und andere, angewendet. Ebenso unterstützt das Modell das Einspielen von Video Dateien zur Objekterkennung, ohne diese vorerst in einzelnen Frames zerlegen zu müssen.

## 3.6 YOLO-NAS

YOLO-NAS[65] wurde von der Firma DECI-AI [80] entwickelt und 2023 veröffentlicht. Das Modell kann mit dem firmeneigenen Pytorch basierten Framework Supergradients implementiert werden.

Über die YOLO-NAS Architektur sind online nicht sehr viele Informationen zu finden. Die Modell Architektur wurde dabei mit Hilfe des Neural Architecture Searchengines von DECI-AI namens AutoNAC entwickelt. Ziel dabei war es eine möglichst gute Balance zwischen Präzision, sowie Latenz zu erreichen.

Bei einem Blick auf die offizielle Architektur Graphik (Abb. 3.4) fallen einige Ähnlichkeiten zu YOLO-V8 auf. So besitzt das Modell ebenfalls mehrere Cross-Stage Verbindungen. Auch der Head des Modells hat drei seperate Detect Blöcke, welche unterschiedlichen Auflösungen als Input nehmen um sowohl große als auch kleinere Objekte gut erkennen zu können. Nach jeder Konvolution wird ebenfalls Batchnormalisation angewandt. Im Gegensatz zu YOLO-V8 werden anschließend aber ReLU Aktivierungen verwendet. Der Backbone enthält ebenfalls eine Spatial Pyramid Pooling Layer um unterschiedliche Input-Auflösungen zu unterstützen.

Eine Besonderheit von YOLO-NAS ist die Verwendung eines Quantisation Aware RepVGG Blocks. Dieser ermöglicht, dass der Präzisionsverlust bei Verwendung einer niedrigeren Quantisierung z.B. 8Bit reduziert wird. So soll der Präzisionsverlust hierbei nur rund 0.5% betragen während dieser bei herkömmlichen Modellen die mAP um rund 1-2% reduziert [79]. Quantisation Aware Training (QAT) basiert darauf, dass Quantisierungsfehler im Forward- und Backwardpass des Trainings emuliert werden. Diese Quantisierungs Techniken werden durch das TensorRT SDK [70] von NVIDIA ermöglicht.

Um die Geschwindgkeitsvorteile der niedrigeren Quantisierung auszunutzen muss allerdings nach dem normalen Training noch eine seperate Quantisation Aware Training (QAT) Phase mit einer neuen Konfiguration gestartet werden.

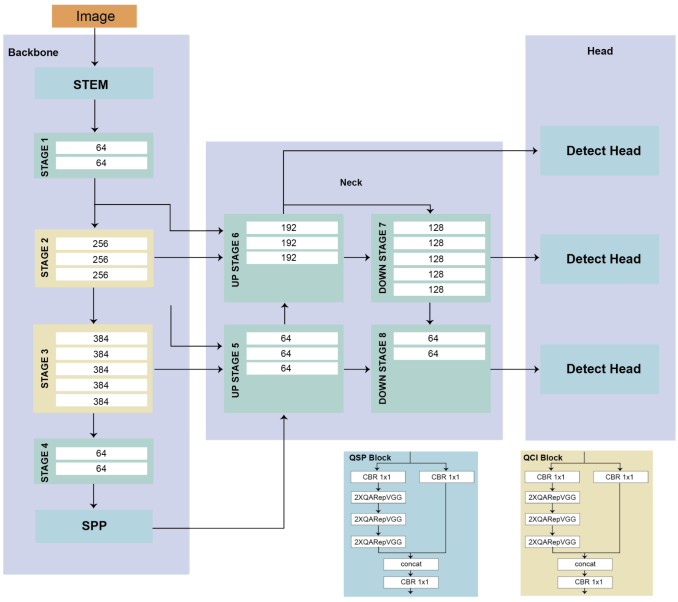


**Abbildung 3.4:** Die Architektur von YOLO-NAS-L, Quelle: [47]

# Kapitel 4

# Methodik

In diesem Kapitel wird beschrieben, wie der praktische Teil der Arbeit umgesetzt wurde. Es werden die Schritte und Entscheidungen erläutert, die während des Forschungsprozesses zur Entwicklung und Bewertung der OD-Modelle getroffen wurden. Die Methodik bildet die Grundlage für den anschließenden Vergleich der Modelle und in weiterer Folge der Diskussion der Ergebnisse.

## 4.1 Datenerhebung und -annotation

Im Rahmen des Forschungsprojektes BAMBI (Biodiversity Airborne Monitoring Based on Intelligent UAV sampling) wurde bereits eine große Menge an Drohnen-Thermalkamera Daten von Wildtieren, sowie Tieren in Tierparks erhoben. Zur Aufnahme der Bilder wurden die Drohnen DJI M30T und DJI M3T verwendet. Dabei werden Lichtfeld Techniken angewandt, um den Fokus der Kamera auf den Waldboden zu setzen und so die Tiere besser durch das Blätterdach erkennen zu können.

Von diesen Aufnahmen wurden im Semesterprojekt des 4. Semesters von dem Projektteam (Rakos Florian, Wassermeyer Gianluca, Eckkrammer Jakob und Plasser Michael) ein Subset von rund 3000 Bildern mit 10 verschiedenen Klassen (Rotwild, Rehwild, Wildschwein, Hase, Sikawild, Damwild, Gamswild, Schaf, Pferd, Büffel) mit Hilfe des Online Tools CVAT annotiert. Dabei ist die Verteilung der Instanzen der unterschiedlichen Klassen unausgewogen (siehe Abb. 4.1). Den überwiegenden Teil der Daten machen Bilder von Rotwild und Rehwild aus. Die verwendeten Daten sind in rund 30 unterschiedlichen Drohnenflügen entstanden. Die Bilder sind dabei in recht kurzen Zeitabständen geschossen worden, was zur Folge hat, dass viele der Bilder recht ähnlich sind, was ein Problem für die Generalisierbarkeit des trainierten Modells darstellen könnte.

## 4.2 Modellauswahl und -vorbereitung

Im Verlauf des Semesterprojekts wurden bereits mehrere Modelle auf dem BAMBI-Datensatz trainiert. Am ausführlichsten wurden dabei die Modelle YOLO-V8 und DETR-Resnet-50 für die Erkennung von Wildtieren getestet. YOLO-V8 erzielte dabei die besten Ergebnisse, insbesondere hinsichtlich der Inferenzdauer. Dies stellt einen erheblichen

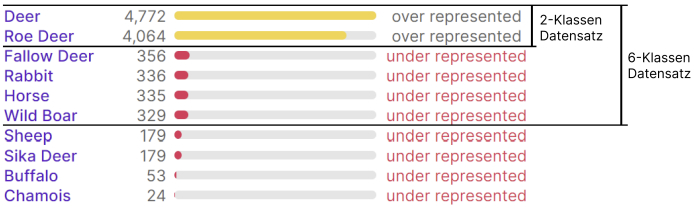


**Abbildung 4.1:** Instanzverteilung der annotierten Daten aus dem Semesterprojekt. Aus den Daten wurde ein 2-Klassen und ein 6-Klassen Datensatz erstellt.

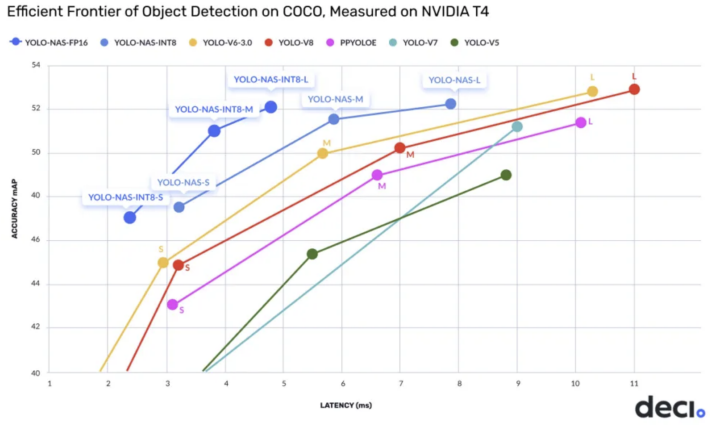


**Abbildung 4.2:** Performance der unterschiedlichen YOLO Modelle, Quelle: [57]

Vorteil dar, da die Modelle im Rahmen des BAMBI-Projekts für Echtzeitdetektionen eingesetzt werden sollen.

Im Mai 2023 wurde das neue YOLO-NAS-Modell veröffentlicht, das eine sehr gute Leistung bei gleichzeitig niedrigerer Latenz, im Vergleich zu anderen YOLO Modellen verspricht (Abb. 4.2).

Deshalb wurde entschieden in dieser Arbeit die Modelle YOLO-NAS und YOLO-V8 zu vergleichen, um zu ermitteln, welches besser für die Echtzeit-Erkennung von Tieren in Thermalbildern von Drohnen geeignet ist. Auf Grund der offiziellen Vergleichswerte (Abb. 4.2) wurde angenommen, dass YOLO-NAS auch in dieser Arbeit bessere Ergebnisse als YOLO-V8 erzielen sollte.

## 4.3 Datensatz Differenzierung

Die Gesamtmenge der annotierten Daten wurde in zwei Datensätze aufgeteilt. Im ersten Datensatz wurden die 6 Tierklassen - Rotwild, Rehwild, Wildschwein, Sikawild, Hase, Pferd beibehalten. Wie in Abb. 4.1 ersichtlich gibt es für die 4 Klassen - Gams, Sikawild, Büffel und Schaf - nur sehr wenige Daten, deshalb wurden diese Bilder aus dem Training völlig ausgeschlossen.

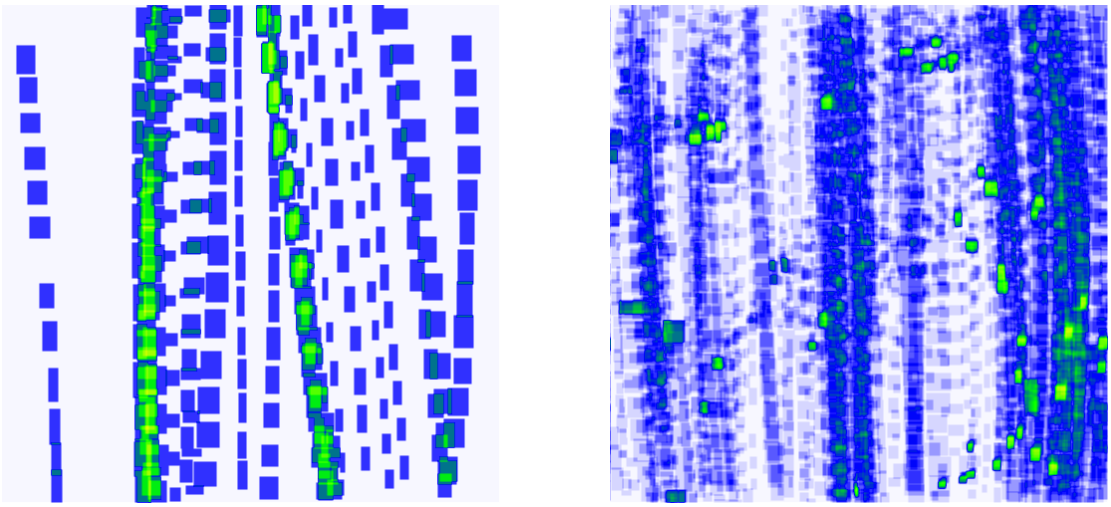

**Abbildung 4.3:** Annotation Heatmaps von Wildschwein (links) und Rotwild (rechts)

Im zweiten Datensatz wurden nur die Bilder von Rotwild und Rehwild beibehalten. Die Klassen Damwild, Hase, Pferd und Wildschwein wurden hierbei ebenfalls ausgeschlossen, da diese mit nur Rund 300 Instanzen pro Klasse ebenfalls sehr schwach vertreten sind, im Vergleich zu den Klassen Rotwild und Rehwild. Auf der Annotation Heatmap, welche die Boundingboxen aller annotierten Instanzen visualisiert (hier am Beispiel der Klasse Wildschwein in Abb. 4.3), ist schnell ersichtlich, dass die Daten der genannten Klassen sehr limitiert sind um einen gut generalisierbares Training zu ermöglichen. Die Annotation Heatmap bei Rotwild ist dabei hingegen schon wesentlich ausgewogener und weist viel größere Varianz zwischen Größen und Positionierungen der Boundingboxen auf.

## 4.4 Datenaugmentation

Um die Varianz der Trainingsdaten zu erhöhen und die Generalisierungsfähigkeit des Modells zu verbessern, kommen Datenaugmentationen zum Einsatz. Diese wurden mithilfe des Online-Tools Roboflow [67] durchgeführt. Eine Augmentierung während des Trainingsprozesses wurde bewusst vermieden, da die Augmentierungen für die beiden unterschiedlichen Modelle identisch sein sollen. Dies kann in diesem Fall jedoch nicht gewährleistet werden, da die verwendeten Frameworks (Supergradients und Ultralytics) unterschiedliche Fähigkeiten zur Augmentierung aufweisen.

Die folgenden Augmentierungen wurden mit Roboflow angewandt:

- **90° Drehungen:** Im Uhrzeigersinn, gegen den Uhrzeigersinn, um 180° gedreht
- **Zuschneiden:** 0% Minimale Zoomstufe, 20% Maximale Zoomstufe
- **Rotation:** Zwischen -15° und +15°
- **Perspektivische Verzerrung:** ±10° Horizontal, ±10° Vertikal
- **Helligkeit:** Zwischen -15% und +15%
- **Noise:** Bis zu 0.1% der Pixel

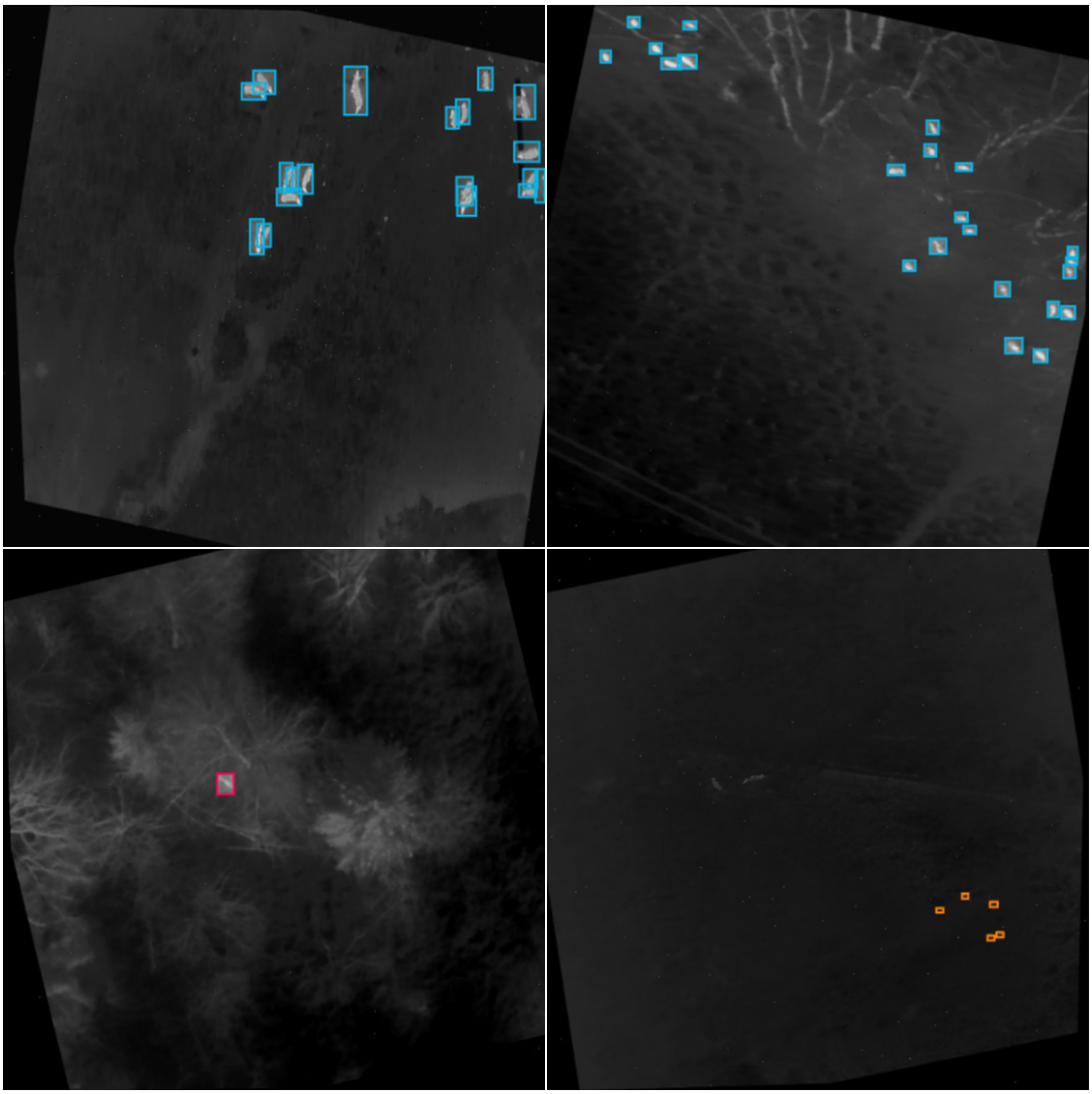

**Abbildung 4.4:** Beispiele der augmentierten Daten

Die spezifischen Augmentierungsparameter wurden anhand der Standardwerte in Roboflow ausgewählt. Dabei wurde die Menge der Trainingsdaten verdreifacht. Bei dem Reh- und Rotwild Datensatz wurden die Trainings Bilder von 1848 auf 5544 und bei dem 6-Klassen Datensatz von 2085 auf 6255 vermehrt. Die Erstellung von noch mehr augmentierten Daten ist in der kostenlosen Mitgliedschaft bei Roboflow nicht inkludiert. Die Validation und Test Datensätze wurden nicht augmentiert.

## 4.5 Modell Implementierung und Training

Es gibt unterschiedliche Implementierungen des YOLO-V8 Modells. In dieser Arbeit wird hierfür das Ultralytics Framework verwendet. Dieses zeichnet sich durch ein sehr

hohes Level an Abstraktion aus. Es muss somit kaum Code geschrieben werden und es werden vor allem die YAML-Konfigurationsdateien bearbeitet um den Trainingsprozess anzupassen.

Das YOLO-NAS Modell ist erst seit kurzem öffentlich verfügbar. Dabei wurde ursprünglich angestrebt, dieses auch über das Ultralytics framework zu implementieren. Allerdings wird hierbei das training auf einem eigenen Datensatz derzeit noch nicht unterstützt. Deshalb wurde das SuperGradients [65] framework dafür verwendet.

Die Modelle (YOLO-NAS und YOLO-V8) werden mit dem Trainingsdatensatz trainiert. Der Trainingsprozess erfolgt in mehreren Epochen, wobei die Gewichtungen des Modells iterativ angepasst werden. Dabei werden Validierungsdaten verwendet, um die Modellleistung während des Trainings zu überwachen und Overfitting zu vermeiden.

Das Training am BAMBI Datensatz erfolgte dabei auf dem DGX-Server der FH Oberösterreich. Dieser enthält 8 NVIDIA DGX-A100 Grafikkarten von welchen 3 für diese Arbeit zur Verfügung standen. Der DGX Server wurde über eine SSH Verbindung und mit Hilfe der VSCode extension „Remote - SSH“ genutzt.

Um die Modelle zu trainieren wurden für jedes Modell ein Docker Container [69] erstellt, welchem jeweils eine A100 GPU zugewiesen wurde. Für YOLO-V8 wurde dabei der „ultralytics/ultralytics:latest“ [73] container vom DockerHub verwendet. Für YOLO-NAS wurde der „deciai/super-gradients:3.0.7“ [66] verwendet.

## 4.6 Evaluierung und Vergleich der Modelle

Nach dem Training werden die entwickelten Modelle auf einem noch nicht gesehenen Testdatensatz getestet, um ihre Leistung in Bezug auf Präzision, Recall und Effizienz zu bewerten. Dabei werden verschiedene Metriken wie mAP (mean Average Precision) und Inferenzgeschwindigkeit gemessen. Die Ergebnisse der Modelle, werden miteinander verglichen, um festzustellen, welches Modell am besten für die Echtzeit-Erkennung von Wildtieren auf Thermalbildern geeignet ist, bzw. was die Vor- und Nachteile der verschiedenen Modelle sind.

# Kapitel 5

# Vergleich

In diesem Kapitel werden die Ergebnisse aus dem Training der beiden Modelle YOLO-V8 und YOLO-NAS vorgestellt. Zuerst wird auf die Konfiguration sowie die Hyperparameter eingegangen, die von den Modellen verwendet wurden. Anschließend folgen die konkreten Detektionsergebnisse auf den 6-Klassen- und 2-Klassen-Datensätzen, wobei besonders der erstgenannte eingehend betrachtet wird. Zuletzt wird die Performance der Modelle untersucht.

Für die Auswertung der beiden Modelle wird die Metrik mAP-0.5 gegenüber dem mAP-0.5-0.95 bevorzugt verwendet, da die Genauigkeit der inferierten Boundingboxen für das BAMBI Projekt nicht essentiell ist. Wenn nur von mAP die Rede ist hierbei die mAP-0.5 gemeint.

## 5.1 Konfiguration

Für das Training von YOLO-V8 wurden die Standard Parameter aus der Ultralytics-default.yaml Datei als Startpunkt genommen. Dabei wurden jegliche Augmentierungseinstellungen entfernt, da die Augmentierung bereits im Vorhinein geschehen ist. Abgesehen von der Anzahl an Workers und der Batchsize wurden die Hyperparameter nicht mehr angepasst da mit den Standard Einstellungen bereits sehr gute Ergebnisse erzielt werden konnten.

Für das Training des YOLO-NAS Modells wurden ebenfalls die Standard Parameter von DECI [74] als Startpunkt hergenommen. Nach den ersten Trainings wurde aber schnell ersichtlich, dass der Trainingsprozess sehr instabil ist. Dabei kam es zwischen einzelnen Learning Steps zu mAP Unterschieden von bis zu 50%. Deshalb wurde die Learningrate deutlich reduziert. Die LR erreichte damit einen ungewöhnlich niedrigen Wert (siehe Tabelle 5.1). Mit dieser wurden aber etwas bessere Ergebnisse erzielte.

## 5.2 6-Klassen Datensatz

Am ausführlichsten wurden die Modelle auf dem 6-Klassen Datensatz getestet.

YOLO-V8 konnten dabei nach den ersten Trainings augenblicklich sehr gute mAP Werte erzielt werden. Die unterschiedlichen Versionen des Modells hatten dabei keinen

| Parameter | YOLO-V8 | YOLO-NAS |
|---|---|---|
| Initial LR | 0.01 | 0.00005 |
| Final LR | 0.0001 | 0.0000025 |
| Optimizer | ADAM | ADAM |
| Weight Decay | 0.0005 | 0.0001 |
| Momentum | 0.937 | (nicht bekannt) |
| Batchsize | 64 | 32 |
| Workers | 16 | 6-12 (nach Modell Größe) |
| LR Mode | Linear | Cosin |
| LR Warmup | 3 Epochen mit LR 0.001 | 3 Epochen mit LR 0.0000005 |

**Tabelle 5.1:** YOLO-V8 und YOLO-NAS Hyperparameter

sehr großen Einfluss auf die Performanz. Der Abstand zwischen dem schlechtesten und bestem Modell beträgt nur rund 2%. Bei der mAP-0.5-0.95 hingegen war der Abstand mit rund 5% höher. Die größeren Modellarchitekturen konnten also präzisere Boundingboxes zeichnen. Die besten Detektionen erzielte das YOLO-V8-M mit einer mAP von 96.5% auf dem Validierungs Datensatz und 95.7% auf dem Test Datensatz. Diese Ergebnisse sind noch wesentlich besser als die Resultate des Trainings während des Semesterprojektes bei dem das YOLO-V8-M eine mAP von rund 83% erzielte. Diese Verbesserungen sind vermutlich auf die Datenaugmentierungen, sowie die Beschränkung auf Klassen mit genügend Instanzen zurückzuführen. Ebenso wurden während der Arbeit noch manche falsch gelabelten Daten korrigiert.

| **Model** | **Size(MB)** | **val-mAP-0.5** | **val-mAP-0.5-0.95** | **test-mAP-0.5** | **Inf.(ms)** |
|---|---|---|---|---|---|
| **V8-n** | 5.9 | 0.944 | 0.650 | 0.91 | 7.7 |
| **V8-s** | 21.4 | 0.957 | 0.680 | 0.94 | 7.9 |
| **V8-m** | 49.7 | **0.965** | 0.704 | **0.957** | 9.7 |
| **V8-l** | 83.7 | 0.964 | **0.706** | 0.933 | 11.2 |
| **NAS-s** | 244 | 0.750 | 0.493 | 0.744 | 125.7 |
| **NAS-m** | 649 | 0.762 | 0.521 | 0.774 | 193.4 |
| **NAS-l** | 851 | 0.711 | 0.482 | 0.691 | 280.9 |

**Tabelle 5.2:** Ergebnisse und Performanz der beiden Modelle auf dem 6-Klassen Datensatz.

Im Gegensatz zu YOLO-V8 wies YOLO-NAS einen sehr volatiles und instabiles Training auf. Die höchste mAP konnte von dem YOLO-NAS-M mit 76.2% auf dem Validierungsdatensatz und 77.4% auf dem Testdatensatz erzielt werden. Ein Vergleich mit dem Trainingsprozess von YOLO-V8 ist in Abb. 5.2 zu sehen. Zwischen den unterschiedlichen Modellvarianten war eine Spannweite von rund 5% bei der mAP-0.5 und rund 4% bei der mAP-0.5-0.95. Der Trend der bei YOLO-V8 vorhanden ist, dass die Modelle bei einer strengeren Bewertung über die Genauigkeit der inferierten Boundingbox, in ihrer Präzision weiter auseinander gehen, ist hier somit nicht vorhanden.

Bei den Ergebnissen überrascht auch, dass das YOLO-NAS-L die schlechtesten Er-

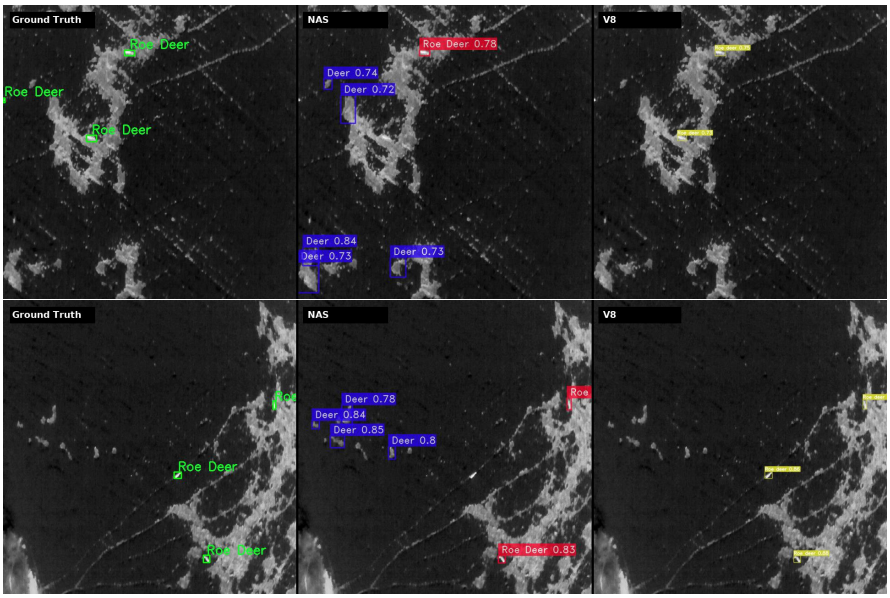


**Abbildung 5.1:** Inferenz der Testdaten von YOLO-NAS-m und YOLO-V8-m, sowie Ground Truth Label. Dabei erkennt das NAS Modell in mehreren Fällen fälschlicherweise eine Rotwild Herde.

gebnisse mit eine Validation mAP von 71.1% erreicht. Bei der Verwendung der Modelle zur Inferenz des Testdatensatzes lässt sich erkennen wodurch die Leistungen gedämpft werden. So ist in Abb. 5.1 zu sehen dass das NAS Modell eine Herde von Rotwild erkennt wo keine ist. Dies könnte das Resultat von Overfitting sein, da das Modell sich gemerkt hat, dass bei einer ähnlichen Position oft Rotwildherden vorkommen. Dies könnte auch erklären, warum das YOLO-NAS-L Modell die schlechteste Perforance hat, da es eine größere Komplexität aufweist, was das Ausmaß von Overfitting verstärken kann [22].

Für die unterschiedlichen Tierklassen weist YOLO-V8 dabei unterschiedliche Präzisionen in der Detektion auf, was in der Confusion Matrix (Abb. 5.3) beobachtet werden kann. Dabei fällt auf, dass das Modell vor allem bei der Klasse Hase schlechte Leistungen vorweist und diese nur bei rund 70% der Instanzen erkennt. Dies ist nicht weiter verwunderlich, da die Hasen auf den Thermalbildern oft nur durch sehr wenige hellere Pixel dargestellt werden und auch von Menschen nicht erkannt werden würden, wenn nicht die Information vorliegt, dass sich in dem jeweiligen Bild ein Hase befindet. Für YOLO-NAS konnte leider keine Confusion Matrix erstellt werden, da das Supergradients Framework keine solche Funktionalität aufweist.

## 5.3 2-Klassen Datensatz

Separat wurden die Modelle auch auf dem 2-Klassen Datensatz (Reh- und Rotwild) trainiert. Die YOLO-V8 Modelle erzielten hierbei alle eine sehr ähnliche mAP, die zwischen 96% und 97% lag. Die Detektionen sind somit ein wenig präziser als die Ergebnisse des größeren Datensatzes. Die mAP-0.5-0.95 wies, wie auch beim 6-Klassen Datensatz, eine größere Spannweite von rund 4,5% auf. Dabei konnten erneut die größeren Modelle

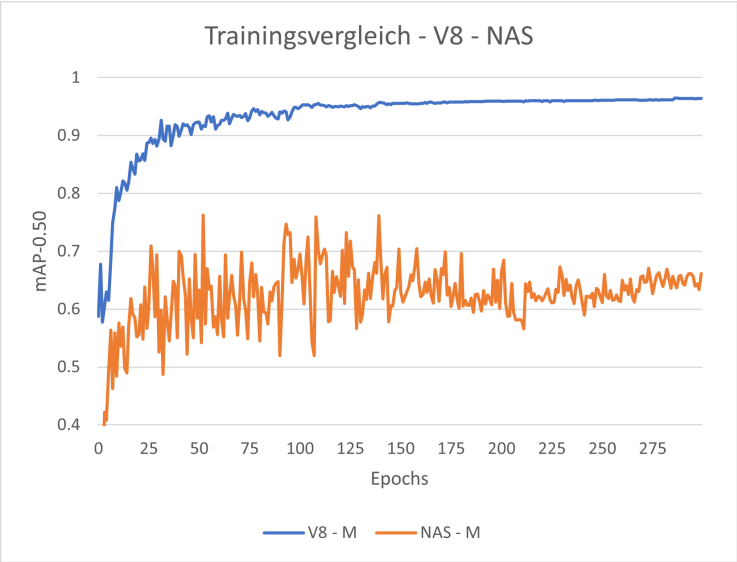


**Abbildung 5.2:** Entwicklung der Validierungs mAP über das Training zwischen YOLO-V8-m und YOLO-NAS-m auf dem 6-Klassen Datensatz

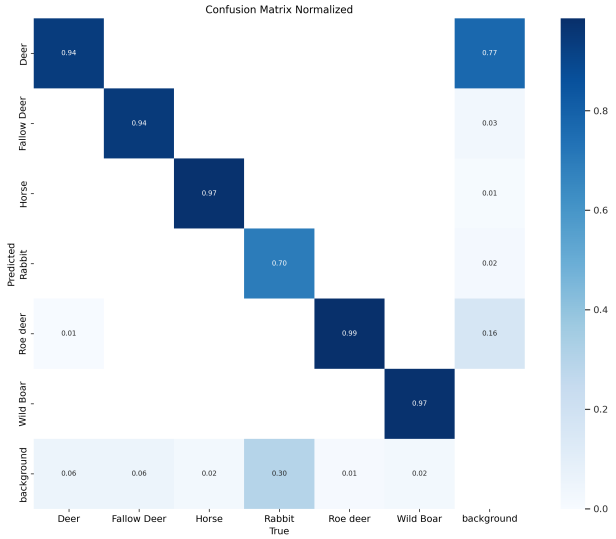


**Abbildung 5.3:** Confusion Matrix - YOLO-V8-M

präzisere Boundingboxen zeichnen.

Anders als bei YOLO-V8 gab es bei YOLO-NAS auf dem 2-Klassen Datensatz einen sehr starken Unterschied der Präzision im Vergleich zum größeren Datensatz (siehe Abb. 5.4). Die Modelle, die auf den Reh- und Rotwild-Datensatz trainiert wurden, erreichten bis zu 15% höhere maximale mAP-Werte. Die besten Ergebnisse auf dem Testdatensatz erzielte dabei das YOLO-NAS-s Modell mit einer mAP von rund 89%. Die Erkennung mehrerer Klassen ist für YOLO-NAS offensichtlich wesentlich schwieriger

| Model | val-mAP-0.5 | val-mAP-0.5-0.95 | test-mAP-0.5 |
|---|---|---|---|
| **V8-n** | 0.961 | 0.666 | 0.944 |
| **V8-s** | 0.966 | 0.696 | 0.950 |
| **V8-m** | 0.966 | 0.707 | 0.947 |
| **V8-l** | **0.969** | **0.716** | **0.968** |
| **NAS-s** | 0.892 | 0.587 | 0.889 |
| **NAS-m** | 0.903 | 0.597 | 0.877 |
| **NAS-l** | 0.872 | 0.554 | 0.886 |

**Tabelle 5.3:** Ergebnisse der beiden Modelle auf dem 2-Klassen Datensatz.

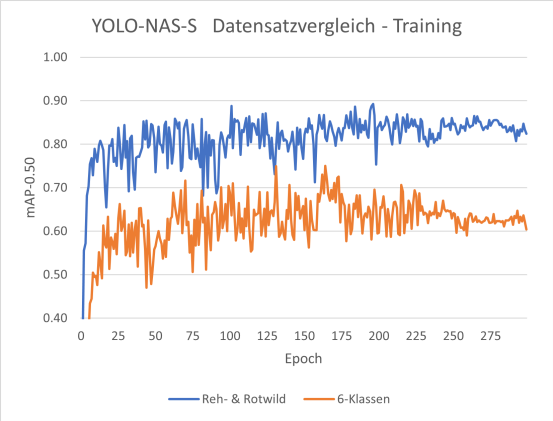


**Abbildung 5.4:** Validierungs mAP von YOLO-NAS-s auf den zwei unterschiedlichen Datensätzen

als für YOLO-V8. Beim Vergleich zwischen mAP und mAP0.5-0.95 gibt es bei YOLO-NAS wiederholt keinen Zusammenhang zwischen den Ergebnissen und der Größe des Modells.

### 5.3.1 Performanz

Auch bei der Inferenzgeschwindigkeit überzeugen die YOLO-V8 Modelle. So benötigten diese nur zwischen 7.7ms (Nano) und 11.2ms (Large) für eine Inferenz inklusive Post- und Preprocessing auf dem DGX-Server der FH Hagenberg.

Bei YOLO-NAS wird die Inferenzgeschwindigkeit offiziell mit zwischen 3.21ms (Small) und 7.87ms (Large) angegeben. Diese Geschwindigkeiten konnten bei den Tests in dieser Arbeit bei weitem nicht erreicht werden. Bei einer Manuellen Inferenz des Testsets auf dem DGX-Server wurden die wesentlich höheren Werte 126ms (Small), 193ms (Medium), 281ms (Large) gemessen. Dabei ist anzumerken, dass bei Aufruf der „model.predict()“ methode unter Umständen noch andere Setupprozesse die Inferenzdauer verlängern. Nichts desto trotz sind die gemessenen Werte sehr weit von den offiziellen Angaben entfernt, was bei YOLO-V8 nicht der Fall ist.

# Kapitel 6

# Diskussion

In der nun folgenden Diskussion wird die Methodik kritisch betrachtet und die Aussagekraft der dabei erzielten Leistungen hinterfragt. Weiterführend werden die Ergebnisse interpretiert und die Schlüsse, die aus dieser Arbeit gezogen werden können, präsentiert.

Eine Limitation der Validierungs und Testergebnisse ist, dass die dabei verwendeten Datensätze dem Trainingsdatensatz teilweise sehr ähneln. Da die Thermalbilder in nur einigen wenigen Drohnenflügen gesammelt wurden, waren nicht genug Daten zur Verfügung um Validierungs- und Testdatensätze aus seperaten Drohnenflügen zusammenzusetzen. Die Daten wurden deshalb randomisiert gesplittet. Es wäre deshalb interessant die Modelle auf einem völlig unbekannten Datensatz zu testen, um beurteilen zu können wie gut die Modelle wirklich performen. Eine echte mAP von beispielsweise 85% wird für wahrscheinlicher gehalten als die von YOLO-V8-M maximal erreichten 95.5%. Somit bleibt fraglich, ob die Modelle bei sämtlichen Drohnen-Thermalbildern ähnlich gute Ergebnisse erzielen kann. Insbesondere bei den 4 Klassen Damwild, Pferd, Hase und Wildschwein, welche jeweils nur rund 300 Trainingsbilder umfassen, ist zweifelhaft, ob diese auch auf Daten aus neuen Drohnenflügen effektiv erkannt werden können. Doch auch bei den Klassen Reh und Rotwild ist es möglich, dass die Präzision der Modelle unter neuartigen Rahmenbedingungen wie Jahreszeit, Wetter und Flughöhe der Drohne stark reduziert werden könnte.

Das YOLO-NAS-Modell konnte den Erwartungen nicht gerecht werden und hat YOLO-V8 weder in Präzision noch in Inferenzgeschwindigkeit übertroffen. Dabei hat sich als schwierig herausgestellt im Trainingsprozess eine effektive Konvergenz zu erreichen. Möglicherweise hätte das Modell mit einer umfangreichen Anpassung der Hyperparameter und Trainingseinstellungen bessere Ergebnisse erzielt. Leider gibt es nur sehr wenige Informationen zum Training der YOLO-NAS-Modelle und alle gefundenen Quellen verwendeten dabei die selben Standard-Trainingsparameter. Generell sind online nur sehr wenige Informationen zu YOLO-NAS verfügbar und es gibt keine aktive Community, die an der Weiterentwicklung des Modells interessiert ist, wie im Fall von YOLO-V8. Dies spiegelt sich auch in der Interessens-Analyse mit Hilfe von Google Trends wider (siehe Abb. 6.1). Darüber hinaus benötigen die YOLO-NAS-Modelle wesentlich mehr Speicherplatz und müssen für eine länger Zeitdauer trainiert werden.

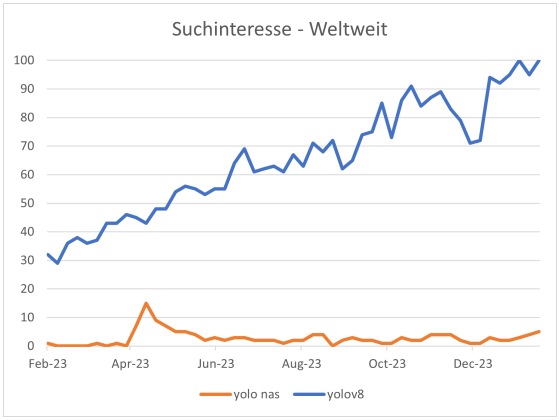


**Abbildung 6.1:** Google Trends - Suchinteresse von YOLO-NAS und YOLO-V8, Quelle: [63]

Zusammenfassend lässt sich sagen, dass YOLO-V8 mit einer maximalen Test-mAP von 95,7 % und einer Inferenzdauer von nur 9,7 ms eine deutlich präzisere, leistungsstärkere und einfachere Wahl für die Erkennung von Tieren auf Drohnen-Thermalbildern darstellt. Die offiziellen Testergebnisse von YOLO-NAS sollten mit einer gewissen Skepsis betrachtet werden, da die erreichte maximale Test-mAP von 77,4 % bei einer Inferenzdauer von 193,5 ms die Leistungen von YOLO-V8 deutlich unterbieten und das Modell umständlicher zu implementieren ist, sowie eine geringfügigere Dokumentation aufweist.

# Quellenverzeichnis

## Literatur


[1] Alexey Bochkovskiy, Chien-Yao Wang und Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. DOI: 10.48550/arXiv.2004.10934. arXiv: 2004.10934 [cs.CV] (siehe S. 18).

[2] Tom B. Brown u. a. *Language Models are Few-Shot Learners*. 2020. DOI: 10.48550/arXiv.2005.14165. arXiv: 2005.14165 [cs.CL] (siehe S. 9).

[3] N. Dalal und B. Triggs. „Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Bd. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177 (siehe S. 15).

[4] Jia Deng u. a. „Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, S. 248–255. DOI: 10.1109/CVPR.2009.5206848 (siehe S. 11).

[5] Jacob Devlin u. a. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805 [cs.CL] (siehe S. 9).

[6] Shiv Ram Dubey, Satish Kumar Singh und Bidyut Baran Chaudhuri. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. 2022. DOI: 10.48550/arXiv.2109.14545. arXiv: 2109.14545 [cs.LG] (siehe S. 5).

[7] Stefan Elfwing, Eiji Uchibe und Kenji Doya. *Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning*. 2017. DOI: 10.48550/arXiv.1702.03118. arXiv: 1702.03118 [cs.LG] (siehe S. 5, 18).

[8] Mark Everingham u. a. „The Pascal Visual Object Classes (VOC) Challenge." *Int. J. Comput. Vis.* 88.2 (2010), S. 303–338. DOI: 10.1007/s11263-009-0275-4 (siehe S. 15, 17).

[9] Christiane Fellbaum, Hrsg. *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication. Cambridge, MA: MIT Press, 1998. DOI: 10.7551/mitpress/7287.001.0001 (siehe S. 11).

[10] Pedro Felzenszwalb, David McAllester und Deva Ramanan. „A discriminatively trained, multiscale, deformable part model". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, S. 1–8. DOI: 10.1109/CVPR.2008.4587597 (siehe S. 15, 16).

[11] Yan Gao u. a. *Decoupled IoU Regression for Object Detection*. 2022. DOI: 10.48550/arXiv.2202.00866. arXiv: 2202.00866 [cs.CV] (siehe S. 13).

[12] Robert Geirhos u. a. *Comparing deep neural networks against humans: object recognition when the signal gets weaker*. 2018. DOI: 10.48550/arXiv.1706.06969. arXiv: 1706.06969 [cs.CV] (siehe S. 1).

[13] Hossein Gholamalinezhad und Hossein Khosravi. *Pooling Methods in Deep Neural Networks, a Review*. 2020. DOI: 10.48550/arXiv.2009.07485. arXiv: 2009.07485 [cs.CV] (siehe S. 9).

[14] Ross Girshick. *Fast R-CNN*. 2015. DOI: 10.48550/arXiv.1504.08083. arXiv: 1504.08083 [cs.CV] (siehe S. 16).

[15] Ross Girshick u. a. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. DOI: 10.48550/arXiv.1311.2524. arXiv: 1311.2524 [cs.CV] (siehe S. 16).

[16] I. Goodfellow, Y. Bengio und A. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. URL: https://books.google.co.in/books?id=Np9SDQAAQBAJ (siehe S. 6).

[17] Alon Halevy, Peter Norvig und Fernando Pereira. „The Unreasonable Effectiveness of Data“. *IEEE Intelligent Systems* 24.2 (2009), S. 8–12. DOI: 10.1109/MIS.2009.36 (siehe S. 10).

[18] Kaiming He u. a. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/arXiv.1512.03385. arXiv: 1512.03385 [cs.CV] (siehe S. 7).

[19] Kaiming He u. a. „Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition“. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2014, S. 346–361. DOI: 10.1007/978-3-319-10578-9_23 (siehe S. 16, 18).

[20] Paul Henderson und Vittorio Ferrari. *End-to-end training of object class detectors for mean average precision*. 2017. DOI: 10.48550/arXiv.1607.03476. arXiv: 1607.03476 [cs.CV] (siehe S. 14).

[21] Jan Hosang, Rodrigo Benenson und Bernt Schiele. *Learning non-maximum suppression*. 2017. DOI: 10.48550/arXiv.1705.02950. arXiv: 1705.02950 [cs.CV] (siehe S. 14).

[22] Xia Hu u. a. *Measuring Model Complexity of Neural Networks with Curve Activation Functions*. 2020. DOI: 10.48550/arXiv.2006.08962. arXiv: 2006.08962 [cs.LG] (siehe S. 29).

[23] Sergey Ioffe und Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. DOI: 10.48550/arXiv.1502.03167. arXiv: 1502.03167 [cs.LG] (siehe S. 7).

[24] Salman Khan u. a. „Transformers in Vision: A Survey“. 54.10s (Sep. 2022). DOI: 10.1145/3505244 (siehe S. 9).

[25] Diederik P. Kingma und Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980 [cs.LG] (siehe S. 6).

[26] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *Advances in Neural Information Processing Systems*. Hrsg. von F. Pereira u. a. Bd. 25. Curran Associates, Inc., 2012. DOI: 10.1145/3065386 (siehe S. 9, 16).

[27] Y. Lecun u. a. „Gradient-based learning applied to document recognition“. *Proceedings of the IEEE* 86.11 (1998), S. 2278–2324. DOI: 10.1109/5.726791 (siehe S. 8, 15).

[28] Chuyi Li u. a. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. 2022. DOI: 10.48550/arXiv.2209.02976. arXiv: 2209.02976 [cs.CV] (siehe S. 18).

[29] Xiang Li u. a. *Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection*. 2020. DOI: 10.48550/arXiv.2006.04388. arXiv: 2006.04388 [cs.CV] (siehe S. 19).

[30] Tsung-Yi Lin u. a. *Feature Pyramid Networks for Object Detection*. 2017. DOI: 10.48550/arXiv.1612.03144. arXiv: 1612.03144 [cs.CV] (siehe S. 16).

[31] Tsung-Yi Lin u. a. „Microsoft COCO: Common Objects in Context“. *CoRR* abs/1405.0312 (2014). DOI: 10.48550/arXiv.1405.0312. arXiv: 1405.0312 (siehe S. 11).

[32] Wei Liu u. a. „SSD: Single Shot MultiBox Detector“. *CoRR* abs/1512.02325 (2015). DOI: 10.48550/arXiv.1512.02325. arXiv: 1512.02325 (siehe S. 16).

[33] Xiang Long u. a. *PP-YOLO: An Effective and Efficient Implementation of Object Detector*. 2020. DOI: 10.48550/arXiv.2007.12099. arXiv: 2007.12099 [cs.CV] (siehe S. 18).

[34] Andrew L. Maas. „Rectifier Nonlinearities Improve Neural Network Acoustic Models“. In: 2013. URL: https://api.semanticscholar.org/CorpusID:16489696 (siehe S. 17).

[35] Vinod Nair und Geoffrey E Hinton. „Rectified linear units improve restricted boltzmann machines“. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, S. 807–814 (siehe S. 5).

[36] Razvan Pascanu, Tomas Mikolov und Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. 2013. DOI: 10.48550/arXiv.1211.5063. arXiv: 1211.5063 [cs.LG] (siehe S. 19).

[37] Hassan Ramchoun u. a. „Multilayer Perceptron: Architecture Optimization and Training“. *Int. J. Interact. Multim. Artif. Intell.* 4 (2016), S. 26–30. DOI: 10.9781/ijimai.2016.415 (siehe S. 8).

[38] Vikas C. Raykar und Amrita Saha. „Data Split Strategiesfor Evolving Predictive Models“. In: *Machine Learning and Knowledge Discovery in Databases*. Hrsg. von Annalisa Appice u. a. Cham: Springer International Publishing, 2015, S. 3–19. DOI: 10.1007/978-3-319-23528-8_1 (siehe S. 12).

[39] Joseph Redmon und Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. DOI: 10.48550/arXiv.1612.08242. arXiv: 1612.08242 [cs.CV] (siehe S. 17).

[40] Joseph Redmon und Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. DOI: 10.48550/arXiv.1804.02767. arXiv: 1804.02767 [cs.CV] (siehe S. 18).

[41] Joseph Redmon u. a. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. DOI: 10.48550/arXiv.1506.02640. arXiv: 1506.02640 [cs.CV] (siehe S. 16, 17).

[42] Shaoqing Ren u. a. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. DOI: 10.48550/arXiv.1506.01497. arXiv: 1506.01497 [cs.CV] (siehe S. 16).

[43] F. Rosenblatt. *The Perceptron: A Theory of Statistical Separability in Cognitive Systems (Project Para)*. Cornell Aeronautical Laboratory, Inc. Cornell Aeronautical Laboratory, 1958. DOI: 10.1037/h0042519 (siehe S. 8).

[44] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. DOI: 10.48550/arXiv.1609.04747. arXiv: 1609.04747 [cs.LG] (siehe S. 6, 7).

[45] Olga Russakovsky u. a. „ImageNet Large Scale Visual Recognition Challenge“. *International Journal of Computer Vision (IJCV)* 115.3 (2015), S. 211–252. DOI: 10.1007/s11263-015-0816-y (siehe S. 11).

[46] Ilya Sutskever u. a. „On the importance of initialization and momentum in deep learning“. In: *Proceedings of the 30th International Conference on Machine Learning*. Hrsg. von Sanjoy Dasgupta und David McAllester. Bd. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, S. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html (siehe S. 7).

[47] Juan Terven, Diana-Margarita Córdova-Esparza und Julio-Alejandro Romero-González. „A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS“. *Machine Learning and Knowledge Extraction* 5.4 (Nov. 2023), S. 1680–1716. DOI: 10.3390/make5040083 (siehe S. 21).

[48] Jasper Uijlings u. a. „Selective Search for Object Recognition“. *International Journal of Computer Vision* 104 (Sep. 2013), S. 154–171. DOI: 10.1007/s11263-013-0620-5 (siehe S. 16).

[49] Ashish Vaswani u. a. „Attention is All you Need“. In: *Advances in Neural Information Processing Systems*. Hrsg. von I. Guyon u. a. Bd. 30. Curran Associates, Inc., 2017. DOI: 10.48550/arXiv.1706.03762 (siehe S. 9).

[50] P. Viola und M. Jones. „Rapid object detection using a boosted cascade of simple features“. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Bd. 1. 2001, S. I–I. DOI: 10.1109/CVPR.2001.990517 (siehe S. 15).

[51] Chien-Yao Wang, Alexey Bochkovskiy und Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. DOI: 10.48550/arXiv.2207.02696. arXiv: 2207.02696 [cs.CV] (siehe S. 18).

[52] Chien-Yao Wang, I-Hau Yeh und Hong-Yuan Mark Liao. *You Only Learn One Representation: Unified Network for Multiple Tasks*. 2021. DOI: 10.48550/arXiv.2105.04206. arXiv: 2105.04206 [cs.CV] (siehe S. 18).

[53] Yanzhao Wu u. a. *Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks*. 2019. DOI: 10.48550/arXiv.1908.06477. arXiv: 1908.06477 [cs.LG] (siehe S. 6).

[54] Zhaohui Zheng u. a. *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression*. 2019. DOI: 10.48550/arXiv.1911.08287. arXiv: 1911.08287 [cs.CV] (siehe S. 19).

[55] Barret Zoph u. a. „Learning Data Augmentation Strategies for Object Detection". *CoRR* abs/1906.11172 (2019). DOI: 10.48550/arXiv.1906.11172. arXiv: 1906.11172 (siehe S. 12).

[56] Zhengxia Zou u. a. *Object Detection in 20 Years: A Survey*. 2023. DOI: 10.48550/arXiv.1905.05055. arXiv: 1905.05055 [cs.CV] (siehe S. 11, 14, 15).

## Medien

[57] Deci AI. *YOLO-NAS*. https://github.com/Deci-AI/super-gradients/blob/master/YOLONAS.md. [Accessed 15-04-2024]. 2023 (siehe S. 23).

[58] Facundo Bre. *Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks*. https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051. [Accessed 15-04-2024]. 2017 (siehe S. 8).

[59] Stefan Elfwing, Eiji Uchibe und Kenji Doya. *Papers with Code - SiLU Explained — paperswithcode.com*. https://paperswithcode.com/method/silu. [Accessed 15-04-2024]. 2017 (siehe S. 5).

[60] Nawin Raj Kumar. *Building a neural network from scratch — kaggle.com*. https://www.kaggle.com/code/nawinrajkumars/building-a-neural-network-from-scratch. [Accessed 15-04-2024]. 2023 (siehe S. 5).

[61] Musstafa. *Optimizers in Deep Learning — medium.com*. https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0. [Accessed 15-04-2024]. 2021 (siehe S. 6).

[62] Deval Shah. *Intersection over Union (IoU): Definition, Calculation, Code*. https://www.v7labs.com/blog/intersection-over-union-guide. [Accessed 15-04-2024]. 2023 (siehe S. 13).

[63] trends.google.com. *Google Trends*. https://trends.google.com/trends/explore?q=yolo%20nas,yolov8&hl=de. [Accessed 15-04-2024]. 2012 (siehe S. 33).

[64] RangeKing (Github User). *Brief summary of YOLOv8 model structure*. https://github.com/ultralytics/ultralytics/issues/189 . [Accessed 15-04-2024]. 2023 (siehe S. 19).

## Software

[65] Shay Aharon u. a. *Super-Gradients*. 2021. DOI: 10.5281/ZENODO.7789328 (siehe S. 20, 26).

[66] DECI. *Docker Container - Supergradients*. [Accessed 28-02-2024]. URL: https://hub.docker.com/r/deciai/super-gradients (siehe S. 26).

[67] B. Dwyer u. a. *Roboflow (Version 1.0)*. Computer Vision. 2022. URL: https://roboflow.com (siehe S. 24).

[68] Glenn Jocher, Ayush Chaurasia und Jing Qiu. *Ultralytics YOLOv8*. Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics (siehe S. 1, 18).

[69] Dirk Merkel. *Docker: lightweight linux containers for consistent development and deployment*. 2014 (siehe S. 26).

[70] NVIDIA. *NVIDIA TensorRT — developer.nvidia.com*. https://developer.nvidia.com/tensorrt. [Accessed 01-04-2024] (siehe S. 20).

[71] Adam Paszke u. a. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. DOI: 10.48550/arXiv.1912.01703. arXiv: 1912.01703 [cs.LG] (siehe S. 18).

[72] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016 (siehe S. 17).

[73] Ultralytics. *Docker Container - Ultralytics*. [Accessed 28-02-2024]. URL: https://hub.docker.com/r/ultralytics/ultralytics (siehe S. 26).

## Online-Quellen

[74] Shay Aharon. *How to Train YOLO-NAS on a Custom Dataset with SuperGradients | Deci — deci.ai*. [Accessed 19-02-2024]. 2023. URL: https://deci.ai/blog/how-to-train-yolo-nas-with-supergradients-a-step-by-step-guide/ (siehe S. 27).

[75] *BAMBI - Biodiversity Airborne Monitoring Based on Intelligent UAV sampling*. [Accessed 28-02-2024]. 2023. URL: https://www.bambi.eco/ (siehe S. 1).

[76] Chinmoy Borah. *Evolution of Object Detection — medium.com*. [Accessed 11-02-2024]. Nov. 2020. URL: https://medium.com/analytics-vidhya/evolution-of-object-detection-582259d2aa9b (siehe S. 15).

[77] Jason Brownlee. *Understand the Impact of Learning Rate on Neural Network Performance - MachineLearningMastery.com — machinelearningmastery.com*. [Accessed 15-02-2024]. 2020. URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (siehe S. 6).

[78] datasolut.com. *Machine Learning: Definition, Algorithmen, Methoden und Beispiele*. 2023. URL: https://datasolut.com/was-ist-machine-learning/ (besucht am 27. 09. 2023) (siehe S. 3).

[79] DECI. *YOLO-NAS - Deci AI Documentation Hub — docs.deci.ai*. [Accessed 01-04-2024] (siehe S. 20).

[80] *Deep Learning Development Platform | Deci — deci.ai*. [Accessed 26-02-2024] (siehe S. 1, 20).

[81] James Gallagher. *What is Data Augmentation? The Ultimate Guide.* Roboflow Blog. Aug. 2023. URL: https://blog.roboflow.com/what-is-data-augmentation-the-ultimate-guide/ (siehe S. 12).

[82] ibm.com. *What is a neural network?* 2023. URL: https://www.ibm.com/topics/neural-networks (besucht am 29. 09. 2023) (siehe S. 4).

[83] *ImageNet.* URL: https://paperswithcode.com/dataset/imagenet (besucht am 22. 01. 2024) (siehe S. 11).

[84] Satya Mallick. *Histogram of Oriented Gradients explained using OpenCV — learnopencv.com.* [Accessed 14-02-2024]. 2016. URL: https://learnopencv.com/histogram-of-oriented-gradients/ (siehe S. 15).

[85] mathworks.com. *What is Object Detection?* 2023. URL: https://www.mathworks.com/discovery/object-detection.html (besucht am 01. 10. 2023) (siehe S. 10).

[86] Musstafa. *Optimizers in Deep Learning — medium.com.* [Accessed 15-02-2024]. 2021. URL: https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0 (siehe S. 6).

[87] Jacob Solawetz. *Train, Validation, Test Split and Why You Need It — blog.roboflow.com.* [Accessed 07-02-2024]. Sep. 2020. URL: https://blog.roboflow.com/train-test-split/ (siehe S. 11).

[88] tecislava.com. *Supervised, Unsupervised und Reinforcement Learning – Was bedeutet was?* 2020. URL: https://www.tecislava.com/blog/supervised-unsupervised-reinforcement (besucht am 27. 09. 2023) (siehe S. 3).

[89] Ultralytics. *YOLOv5: A state-of-the-art real-time object detection system.* Accessed: 23.03.2024. 2021 (siehe S. 18).

[90] Wikipedia-Autoren. *ImageNet.* de. Okt. 2015. URL: https://de.wikipedia.org/wiki/ImageNet (besucht am 22. 01. 2024) (siehe S. 11).