Analyse von Bild- und Videoquellen an Bienenstöcken zur Identifizierung der Anwesenheit von Bienen, Wespen und Hummeln.

Studienarbeit T3100

Studiengang Informatik

Studienrichtung Informationstechnik

Dualen Hochschule Baden-Württemberg Ravensburg

Von

Jonas Alber

Abgabedatum:	17.07.2023
Bearbeitungszeitraum:	09.01.2023 - 17.07.2023
Matrikelnummer:	8862191
Kurs:	TIT-20
Ausbildungsort / -firma:	DHBW Ravensburg
Betreuer der Ausbilungsfirma:	Dr. Jochen Heistermann

Erklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017.

Ich versichere hiermit, dass ich meine Bachelorarbeit (bzw. Projektarbeit oder Studienarbeit bzw. Hausarbeit) mit dem Thema:

Analyse von Bild- und Videoquellen an Bienenstöcken zur Identifizierung der Anwesenheit von Bienen, Wespen und Hummeln.

Selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Markdorf, den 17.07.2023

Jonas Alber

Kurzfassung

Die Studienarbeit T3100 **„Analyse von Bild- und Videoquellen an Bienenstöcken zur Identifizierung der Anwesenheit von Bienen, Wespen und Hummeln.“,** hat zum Ziel, ein neuronales Netzwerk zur Erkennung verschiedener Insektenarten zu entwickeln.

Zu Beginn der Ausarbeitung wird auf grundlegende Informationen des Projekts eingegangen. Hierunter fallen Informationen über den Aufbau und die Funktionsweise von neuronalen Netzen. Darüber wird auf die eingesetzte Bibliothek zum Erstellen und Trainieren von Modellen eingegangen. Zusätzlich wird ein Überblick über die Effekte Overfitting und Underfitting gegeben.

Aufbauend auf diesen Grundlagen wird der Datensatz ermittelt, welcher für die Modelle zum Trainieren genutzt werden soll. Außerdem werden verschiedene gängige Netzwerk Architekturen analysiert, um das optimale Netzwerk zu ermitteln, für diese Projektarbeit.

An die Auswahl anschließend werden die Lokalisierungstechniken Sliding Window und Lokalisierung mittels CNN untersucht. Auf dieser Untersuchung aufbauend ein System zur Objektlokalisierung und Klassifikation entwickelt.

Abstract

The student research project T3100 „Analysis of image and video sources on beehives to identify the presence of bees, wasps and bumblebees.“, aims to develop a neural network for the recognition of different insect species.

At the beginning of the elaboration, basic information of the project is discussed. This includes information about the structure and the functionality of neural networks. Furthermore, the library used to create and train models is discussed. Additionally, an overview of the effects overfitting and underfitting is given.

Based on these basics, the data set to be used for training the models is determined. Furthermore, different common network architectures are analyzed to determine the optimal network for this project work.

Following the selection, the localization techniques Sliding Window and localization using CNN are investigated. Based on this investigation, a system for object localization and classification will be developed.

Inhaltsverzeichnis

1. Einleitung 1
   1.1. Motivation der Thesis 1
   1.2. Zielsetzung der Thesis 2
2. Grundlagen 3
   2.1. Neuronale Netze 3
   2.2. Convolutional Neuronal Network 5
   2.3. Tensorflow 6
   2.4. Architektur der eigenentwickelten CNN 7
   2.5. Effekte beim Modelltraining 8
      2.5.1. Overfitting 8
      2.5.2. Underfitting 8
   2.6. Labelimg 9
3. Datenvorbereitung 10
   3.1. Auswahl der Datenquellen 10
   3.2. Annotation der Datenquellen 14
      3.2.1. Datensatzfilterung 14
      3.2.2. Auswahl des Annotierungssystems 15
4. Modellanalysen 16
   4.1. Faktoren bei der Modellauswahl 16
   4.2. Auswahl der zu untersuchenden CNN-Architekturen 18
   4.3. Analyse der CNN-Architekturen 21
5. Objekt Lokalisierung 23
   5.1. Sliding Window 23
   5.2. Lokalisierung durch ein CNN 27
      5.2.1. Einzelobjekt Lokalisierung 27
      5.2.2. Grundlage der Multiobjekterkennung 29
      5.2.3. Analyse der YOLO V1 Loss-Funktion 34
      5.2.4. Objektüberfluss 37
6. Objekt Erkennung 38
7. Zusammenfassung und Ausblick 40
8. Quellenverzeichnis 41
A. Custom Lokalisierungsmodells V1 44
B.Custom Lokalisierungsmodells V2 45
C.Custom Erkennungsmodell 46

Tabellenverzeichnis

Tabelle 1 Auswertung Datensätze 12
Tabelle 2 Auswertung CNN Punktebewertung 22

Abbildungsverzeichnis

Abbildung 1 Aufbau eines neuronalen Netzes (3) 3
Abbildung 2 Struktur eines Neurons (4) 4
Abbildung 3 Overfitting und Underfitting 8
Abbildung 4 LabelImg Beispielanwendung (25) 9
Abbildung 5 CNN-Modell Übersicht (26) 18
Abbildung 6 Auswertung CNN-Modell Untersuchung 21
Abbildung 7 Sliding Window Vorgehen 23
Abbildung 8 Sliding Window 24
Abbildung 9 Biene auf einer Blume 25
Abbildung 10 Schwebefliege auf einer Blume 25
Abbildung 11 Ausgabe der Objektlokalisierung 28
Abbildung 12 Tabellenstruktur bei 2 Objekten in der Objektlokalisierung 29
Abbildung 13 Beispielbild Multiobjekterkennung 30
Abbildung 14 Berechnung Mean-Square-Error 31
Abbildung 15 Darstellung Berechnung Mean-Square-Error 32
Abbildung 16 Korrekte Berechnung MSE 33
Abbildung 17 Koordinaten Loss Funktion YOLO V1 (20) 35
Abbildung 18 Gitter- struktur bei YOLO V1(20) 36
Abbildung 19 Struktur Objekterkennungssegment 38
Abbildung 20 Objekterkennungsmatrix 39
Abbildung 21 Struktur des Custom Lokalisierungsmodells V1 44
Abbildung 22 Struktur des Custom Lokalisierungsmodells V2 45
Abbildung 23 Custom Erkennungsmodell Teil 1 46
Abbildung 24 Custom Erkennungsmodell Teil 2 47
Abbildung 25 Auswahl CNN-Modelle 47

1. Einleitung

1.1. Motivation der Thesis

Imker arbeiten mit Bienenvölkern. Diese sind eine teildomestizierte Spezies. Das bedeutet, dass sie weiterhin ohne dem Menschenüberleben können. Durch die Zucht der vergangenen Jahrzehnte sind sie jedoch deutlich friedfertiger geworden. Daraus resultierend ist bei den Bienenvölkern jedoch auch die Anfälligkeit für Krankheiten gestiegen. Um die Gesundheit der Völker zu überwachen, muss der Imker regelmäßig in den Bienenstock hinein schauen. Dieser Eingriff bedeutet für die Bienen großen Stress. Durch das Öffnen des Stocks wird das Raumklima gestört. Zusätzlich erzeugt das Bewegungen von Waben Mikroerschütterungen, welche die Bienen in ihrer Orientierungsfähigkeit beeinträchtigt. Daher ist es ein erklärtes Ziel vieler Imker, möglichst wenige Eingriffe in ihre Bienenvölker vorzunehmen.

Um Eingriffe zielführend zu planen, hilft dem Imker die Beobachtung des Fluglochs. Das Flugloch ist der Eingang in ein Bienenvolk. Die Anzahl der Bienen, welche die Beute betreten und verlassen geben dabei Aufschluss über die Stärke und damit verbunden über die Gesundheit des Volkes. Ergänzend kann das Vorhandensein von räubernden Insekten wie Wespen und Hornissen ein Hinweis auf eine schlechte Gesundheit sein. Auf der Grundlage dieser Beobachtung können Eingriffe präziser geplant werden.

Dabei kann der Imker nur zeitweise die Fluglöcher beobachten. Dadurch ist seine Beobachtung ausschließlich eine Momentaufnahme. Eine automatisierte Erkennung des Fluglochs soll dem Imker dabei helfen, den Zustand des Bienenvolks einzuschätzen. Die automatische Analyse von Medienquellen (Bilder und Videos) des Fluglochs sollen Informationen über die anwesenden Arten und deren Häufigkeit geben. Durch die kontinuierliche Überwachung soll der Imker frühzeitige mögliche Krankheiten an seinen Völkern erkennen können.

1.2. Zielsetzung der Thesis

Im Rahmen der Studienarbeit T3100 besteht das Ziel darin, eine automatisierte „Erkennung von Insekten“ mittels eines neuronalen Netzwerks zu entwickeln. Dieses Netzwerk soll verschiedene Arten von Insekten unterscheiden können.

Speziell soll es zwischen Bienen, Hummeln und Wespen als spezifische Insektenarten unterscheiden können. Darüber hinaus soll das Modell in der Lage sein, andere Arten von Insekten als separate Klasse zu erkennen. Neben der Klassifizierung ist es auch wichtig, eine präzise Lokalisierung der Insekten durchzuführen. Zu diesem Zweck sollen verschiedene Ansätze analysiert und miteinander verglichen werden.

Die Projektarbeit wurde im Rahmen des dualen Studiums an der dualen Hochschule Baden-Württemberg Ravensburg (DHBW Ravensburg) durchgeführt. Die Entwicklung fand während der Theoriephase statt.

Die Projektarbeit umfasst die Analyse der Anforderungen. Basierend auf dieser Analyse soll eine geeignete Architektur für das neuronale Netzwerk ausgewählt werden. Parallel dazu soll ein geeigneter Datensatz für das Training der Modelle ermittelt werden. Auf Basis der Architektur und des Datensatzes soll ein System zum Erkennen von Insekten in Medienquellen entwickelt werden. Der Fokus soll hierbei auf einer hohen Berechnungsgeschwindigkeit liegen. Dies resultiert daraus, da Bienenvölker meist in der freien Natur stehen. Daher muss das analysierende Gerät Batteriebetrieben sein, wodurch es stromsparend arbeiten muss. Hierfür werden meist Mikrocontroller eingesetzt, beispielsweise ein Raspberry Pi.

2. Grundlagen

2.1. Neuronale Netze

Die Fähigkeit, auf Basis von Erfahrungen Entscheidungen treffen zu können ist eine Kernfähigkeit aller höheren Lebewesen. Dazu dient das zentrale Nervensystem. Dies ist ein Zusammenschluss aus Milliarden von Nervenzellen (Neuronen). Die Neuronen bilden durch ihre Verbindungen ein neuronales Netzwerk.

Ein Neuron besitzt dabei die Möglichkeit von vielen anderen Nervenzellen Signale zu Empfangen. Auf Basis der Stromstärke, der Dauer des Impulses und der Gewichtung des Eingangs entscheidet das Neuron, ob es ebenfalls ein Signal über sein Axion aussende. Das Axion ist dabei mit anderen Nervenzellen verbunden. Diese Struktur wird in künstlichen neuronalen Netzen (NN) nachgebildet.

Ein großer Unterschied zwischen natürlichen und künstlichen Netzen besteht in deren Struktur. In einem natürlichen Netz können sich Neuron beliebig untereinander verbinden. Die Länge der Verbindungen ist dabei auf das nähere Umfeld des Neurons begrenzt. Verbindungen werden dabei während der gesamten Lebenszeit neu gebildet oder aufgelöst. Über die Bildung und Auflösung von Verbindungen entscheiden unsere Erfahrungen und das, was wir lernen.

In der digitalen Welt ist das anders. Hier besteht ein neuronales Netzwerk aus Schichten. Jede Schicht enthält eine definierte Anzahl an Neuronen. Jedes Neuron einer Schicht ist dabei mit allen Neuronen der Vorgänger sowie allen Neuronen der nachfolgenden Schicht verbunden. Im Gegensatz zur Natur können dabei keine Verbindungen entstehen oder sich auflösen. Dadurch ist die Struktur des Netzwerks vor dem Training bereits fest definiert. Daraus resultiert auch die bekannte Darstellung von neuronalen Netzen, wie sie in Abbildung 1 dargestellt wird.

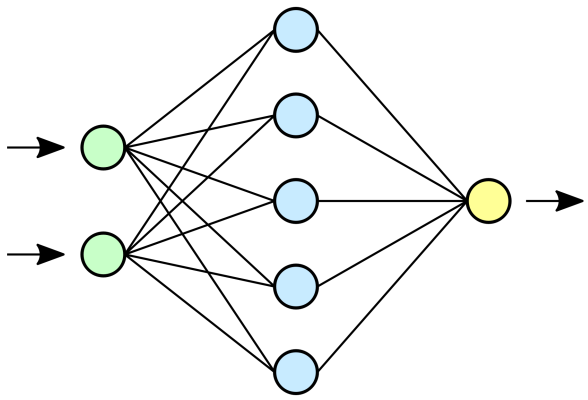

Abbildung 1 Aufbau eines neuronalen Netzes (3)

Bei den Schichten eines NN werden Schichten in drei Kategorien unterschieden. Dies sind Eingabeschichten (eng. Input Layers), versteckte Schichten (eng. Hidden Layers) und Ausgabeschichten (eng. Output Layers).

Die Eingabeschichten sind das Interface in das Netzwerk. Hier werden Informationen in das Netzwerk eingegeben (im Bild die Grünen Punkte). Ausgabeschichten sind das Äquivalent zur Eingabeschicht in umgekehrter Form. Hier gibt das Netzwerk seine berechneten Ergebnisse aus (im Bild die gelben Punkte). Zwischen der Eingabe- und Ausgabeschicht können eine oder mehrere versteckte Schichten existieren. In diesen werden die eingegebene Informationen verarbeiten und verarbeitet. Verstecke Schichten heißen so, da entwickelnde Personen nur aufwändig deren Verhalten nach dem Training verstehen können.

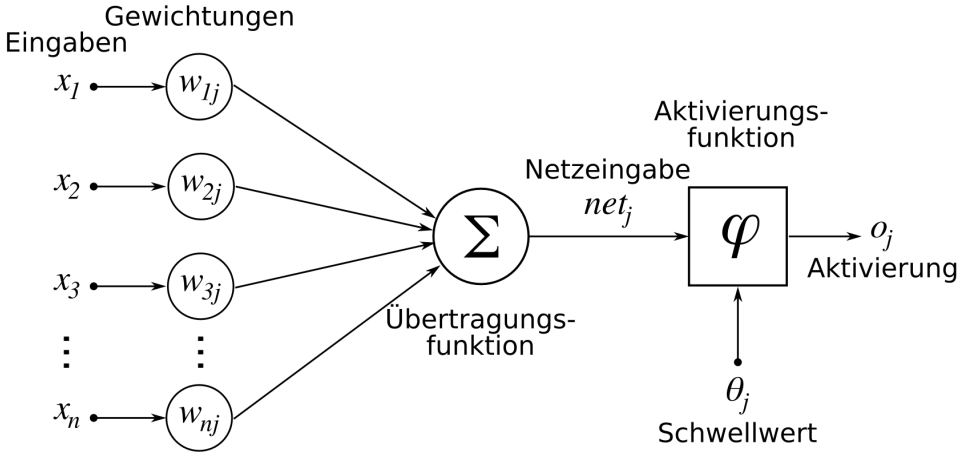

Abbildung 2 Struktur eines Neurons (4)

Jede Verbindung innerhalb des Netzwerks hat ein Gewicht. Diese gibt an, wie stark ein eingehendes Signal vom Neuron beachtet wird. Diese Gewichte werden während des Trainingsprozesses angepasst, um das Netzwerk zu optimieren. Auf Basis der gewichteten Eingaben berechnet das Neuron, welches Ausgabesignal es ausgeben muss.

Im Rahmen des Training werden dem NN Beispiele gezeigt. Das Modell sagt für diese Beispiele Vorhersagen voraus. Die Vorhersagen werden mit manuell definierten korrekten Vorhersagen abgeglichen. Auf Basis der Abweichung zwischen Modellvorhersage und korrekter Vorhersage passt das Modell die Gewichtung seiner Neuronen an. Dadurch kann es zugrunde liegende Muster und Zusammenhänge erlernen. Dieser Lernprozess ermöglicht es dem neuronalen Netz, nach dem Training ihm unbekannte Eingaben zu verarbeiten und korrekte Ausgaben zu generieren. (1) (2) (5)

2.2. Convolutional Neuronal Network

NN liefern bei der Analyse von 1-dimensionale Daten sehr erfolgreich korrekte Vorhersagen. Die Genauigkeit nimmt jedoch bei multidimensionalen Daten wie beispielsweise Bildern deutlich ab. Die Gründe hierfür sind zum einen, dass ein NN nur einen flachen Vektor betrachtet. Dadurch ignoriert es sämtliche räumliche Strukturen. Da Pixel in einem Bild aber eine lokale Abhängigkeit zueinander haben erschwert dies den Trainingsprozess. Des Weiteren besitzen NN nicht die Fähigkeit Skalierungen an Objekten wahrnehmen. Das liegt daran, dass sie auf bestimmte Merkmale und Muster trainiert sind. Da Objekte in Bildern unterschiedliche Abstände zum Aufnahmemedium (beispielsweise Kameras) einnehmen können sind NN hier schlecht geeignet.

Ein Lösungsweg für diese Problematiken bietet ein weiterer Blick in die Natur. Das menschliche Auge ist vergleichbar mit einer digitalen Kamera. Anstelle von Pixeln auf einem Bildsensor verfügt das menschliche Auge über Stäbchen und Zäpfchen. Diese erkennen Helligkeiten (Stäbchen) respektive Farbe (Zäpfchen) des einfallenden Lichts. Sie sind wie bei einem Bildsensor auf der Innenseite der Netzhaut angebracht. Bevor das Bild im Gehirn verarbeitet wird, wandert es durch den Sehnerv. Aus der Medizin ist bekannt, dass innerhalb des Sehnervs Filter auf die Bilder angewendet werden. Diese Filter verstärken Kanten, Kontraste und Konturen in den Bildern. Mit den verstärkten Merkmalen erleichtern sie dem Gehirn die Analyse der Bilder.

Eine solche Filterung kann auch in der digitalen Welt angewendet werden. Dabei wird der Filter mittels einer Faltungsoperation (eng. Convolution) auf das Bild angewendet. Der Filter besteht dabei aus einer 2 Dimensionalen Matrix, häufig einer 3x3 oder 5x5 Matrix. Ein neuronales Netzwerk mit vorgeschalteten Convolutions wird als onvolutional neuronal Networks kurz CNN bezeichnet.

Im Training erlernt das CNN, welche Matrizen es auf die eingegebenen Bilddaten anwenden muss. Im Anschluss an die Filterung werden die Merkmale von einem klassischen neuronalen Netzwerk analysiert. Die Gewichte der Neuronen werden ebenfalls während des Trainings angepasst. Durch die Verbindung von Filterung und neuronalen Netzen sind CNNs fähig, komplexe Muster in Bildern zu erkennen und effizient vorhersagen zu tätigen.

CNNs sind dabei fähig, Bild- und Videoerkennungsaufgaben zu analysieren. Dadurch werden sie in Bereichen wie der medizinischen Bildgebung, der Fahrzeugerkennung und in Sicherheitskameras eingesetzt. (6) (7)

2.3. Tensorflow

Damit ein CNN entwickelt werden kann, ist ein Framework hilfreich. Dieses sollte Strukturen und Tools zum Erstellen und Trainieren der Netzwerke bereitstellen.

Tensorflow ist eine solche Bibliothek. Sie ist für maschinelles Lernen, die Entwicklung von NN und CNN entwickelt. Tensorflow ist in C++ entwickelt. Durch die hardwarenahe Sprache ist die Library sehr leistungsstark. Das Open-Source-Software-Toolkit stellt umfangreiche Funktionen und Werkzeuge für das Design, Training und die Implementierung von komplexen Modellen bietet.

Mit TensorFlow können neuronale Netzwerkarchitekturen entworfen und anpassen. Dies ermöglich die bearbeitung spezifische Aufgaben wie Bilderkennung, Spracherkennung, Textanalyse und vieles mehr. Außerdem bietet es eine Vielzahl von vorgefertigten Schichten und Algorithmen. Durch diese wird das Modellieren und Trainieren von neuronalen Netzen erleichtert. Zu den verfügbaren Schichten gehören Convolutional Layers, Pooling Layers, Fully Connected Layers, Recurrent Layers und vieles mehr.

Eine besondere Stärke von Tensorflow ist seiner Fähigkeit, komplexe Modelle auf GPUs auszuführen. GPUs sind fähig, viele kleine Berechnungen effizient und parallel durchzuführen. Dadurch ist eine schnelle Verarbeitung großer Datenmengen möglich.

Darüber hinaus bietet TensorFlow eine breite Palette von Funktionen zur Evaluierung und Optimierung von Modellen. Es stehen verschiedene Metriken und Loss-Funktionen zur Verfügung. Mit diesen kann die Leistung eines Modells bewertet werden.

Tensorflow ist auch für seine umfangreiche und aktive Community bekannt. Diese stellt eine Vielzahl von Ressourcen, Tutorials und Beispielprojekten bereitstellt. Dadurch ist der Einstieg in das maschinelle Lernen leicht und es bietet Unterstützung bei der Lösung von Problemen oder der Erweiterung der Funktionalität. (8)

2.4. Architektur der eigenentwickelten CNN

Das Entwickeln eines effizienten CNN ist eine komplexe Tätigkeit. Viele Forschungen wurden in den vergangenen Jahren getätigt, um die effizient von CNNs zu erhöhen.

Im Rahmen dieser Studienarbeit werden einige dieser Architekturen untersucht. Ergänz werden soll diese Untersuchung durch eigene Modelle. Diese eigenen Modelle haben den Zweck, zu untersuchen, ab welcher Größe CNNs die Merkmale von Insekten erkennen können.

Das in Anhang A. Custom Lokalisierungsmodells V1 dargestellte Modell fungiert als Basismodell für das Training zur Objektklassifikation. Für das Training zur Objektlokalisierung wurde die Ausgabe von 4 Neuronen (je 1 Neuron pro Klasse) auf 3 Ausgaben erweitert. Hierbei gibt das Denselayer „moreObjekts“ an, ob das Modell im Bild mehr als 1 Objekt erkannt hat. Das Denselayer „location0“ liefert die Koordinaten des erkannten Objekts, während das Denselayer „prediction0“ die erkannte Klasse für das lokalisierte Objekt angibt. Die Verwendung der Zahl 0 bei „Location“ und „Prediction“ wurde gewählt, um zu ermöglichen, dass dieses Modell durch das Hinzufügen von N weiteren Location- und Prediction-Schicht bis zu N Objekte vorhersagen kann.

Das in Anhang B.Custom Lokalisierungsmodells V2 dargestellte Modell baut auf dem in Anhang A. Custom Lokalisierungsmodells V1 beschriebenen Modell auf. Hierbei wurde die Vorhersage, ob mehrere Objekte vorhanden sind (dargestellt durch den blauen Kasten im Bild), früher ausgegeben, um den Einfluss auf die Genauigkeit des Modells zu untersuchen. Das Gleiche wurde für die Klassifikation (dargestellt durch den grünen Kasten im Bild) umgesetzt. Der Hintergedanke dieser Untersuchung war, ein tieferes Verständnis bezüglich der erforderlichen Modelltiefe zu erlangen, da für die reine Klassifikation die Größe bis zur Ebene „max_pooling2d_11“ ausreichend ist, während dies für die Lokalisierung nicht der Fall ist.

Das in Anhang C.Custom Erkennungsmodell dargestellte Modell stellt eine Weiterentwicklung des Modells aus Anhang A. Custom Lokalisierungsmodells V1. Durch das Hinzufügen zusätzlicher Schichten wurde die Vorhersagegenauigkeit für die Objekterkennung erhöht. Zusätzlich wurde dieses Modell bereits für den finalen Einsatz im Rahmen von Kapitel 6. Objekt Erkennung angepasst und gibt einen flachen Vektor der Größe 648 aus. Dieser Vektor kann in eine 3-dimensionale Matrix der Größe 9x9x8 umgewandelt werden.

2.5. Effekte beim Modelltraining

2.5.1. Overfitting

Negative Effekte können die Effizienz von CNNs und NNs beeinflussen. Hierbei sind Overfitting und Underfitting besonders problematisch.

Overfitting tritt auf, wenn ein Modell zu stark an die Trainingsdaten angepasst ist. Dadurch wird Rauschen und zufällige Variationen in den Daten erkannt. Dadurch kann das Modell falsche oder irrelevante Muster lernen. Wenn solche in den Trainingsdaten vorhanden sind, aber nicht in unbekannten Daten, kann das Modell sein Wissen nicht auf diese Bilder übertragen. Abbildung 3 verdeutlicht das Overfitting. Dabei zeichnet das Modell eine Linie, welche sämtliche Trainingspunkte einschließt. Diese Linie erfasst zwar alle Datenpunkte, vernachlässigt jedoch Ausreißer und Rauschen. Um Overfitting zu reduzieren, kann die Komplexität des Modells reduziert werden. Zusätzlich kann die Verwendung eines größeren Datensatzes das Risiko von Overfitting verringern (21).

2.5.2. Underfitting

Underfitting tritt auf, wenn ein Modell zu einfach oder zu unflexibel ist, um die zugrunde liegende Struktur der Trainingsdaten angemessen zu erfassen. Das Modell erzielt sowohl auf den Trainings- als auch auf den Testdaten eine schlechte Leistung, da es nicht in der Lage ist, die grundlegenden Muster oder Zusammenhänge in den Daten zu erfassen. In Abbildung 3 wird dieses Verhalten veranschaulicht, indem eine einfache Linie gezogen wird, die die komplexere Verteilung der Datenpunkte vernachlässigt. Underfitting tritt auf, wenn das Modell zu stark vereinfacht ist oder wenn die Daten zu komplex sind, um von dem gewählten Modelltyp erfasst zu werden. Um Underfitting zu beheben, können komplexere Modelle verwendet oder zusätzliche Features hinzugefügt werden, um die Modellkapazität zu erhöhen. (22)

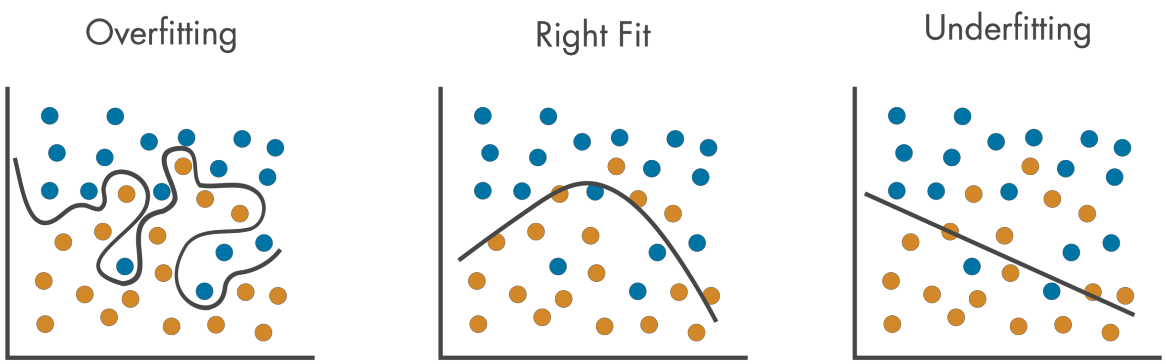

Abbildung 3 Overfitting und Underfitting

2.6. Labelimg

Für das Training von CNNs sind Trainingsdaten unerlässlich. Im Bereich der Bilderkennung besteht ein Trainingssatz aus einem Bild, sowie den zugehörigen Informationen. Diese Informationen sind die Position eines Objektes im Bild und dessen Klasse. Um korrekten Vorhersagen zu erstellen, sind Annotationstools als Unterstützungsfunktion hilfreich.

In dieser Arbeit wird das Tool LabelImg eingesetzt. Dies ist ein Open-Source-Projekt, welches von TensorFlow für die Bildannotation empfohlen wird. Mit LabelImg können Rechtecke um Objekte im Bild gezeichnet werden. Diese Rechtecke werden als Bonding Boxes bezeichnet. Jeder Bonding Box muss zudem eine Klassen zugeordnet werden. Die Annotierung eines Bildes ist in Abbildung 4 dargestellt. Die Informationen über die Bonding Boxes werden in einer getrennten XML-Datei gespeichert. Die XML-Datei besitzt dabei denselben Namen wie das annotierte Bild.

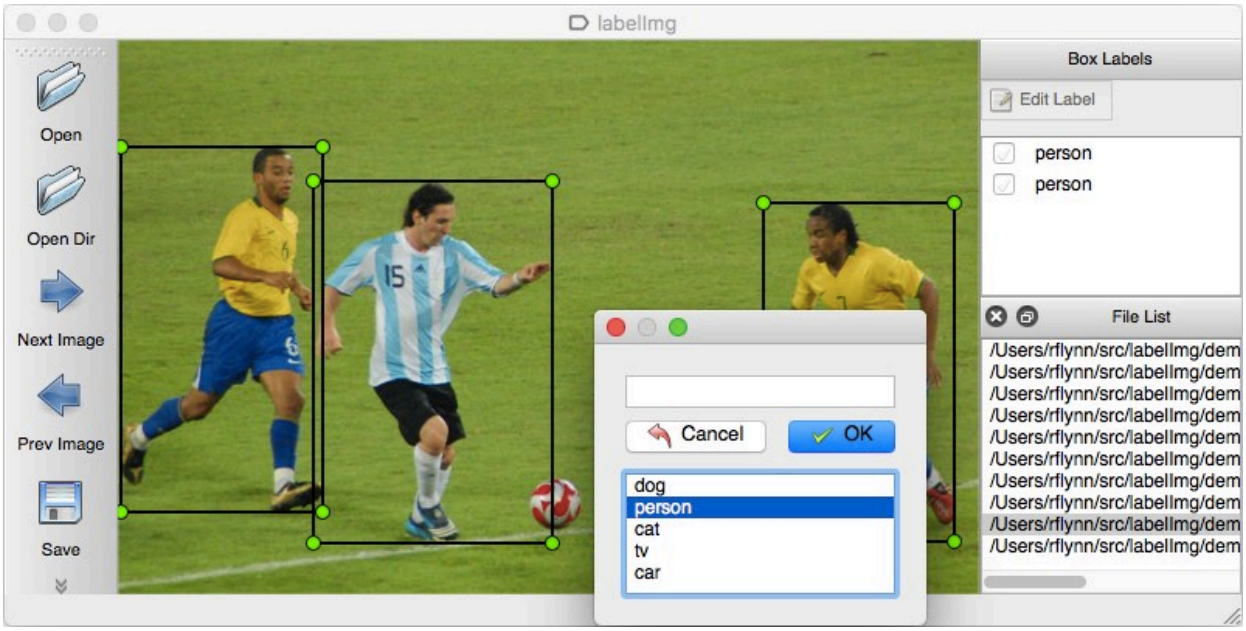

Abbildung 4 LabelImg Beispielanwendung (25)

Durch das Speichern Bildinformationen in einer getrennten XML Datei bleiben die Bilder unverändert. Daher können diese weiterhin unabhängig von den XML Dateien genutzt werden. (25)

3. Datenvorbereitung

3.1. Auswahl der Datenquellen

Damit ein neuronales Netzwerk effektiv trainieren kann, ist die Qualität der Trainingsdaten von entscheidender Bedeutung. Die Trainingsdaten bestehen aus zwei Hauptkomponenten. Erstens beinhalten sie Beispielbilder, die die zu erkennenden Objekte enthalten. Diese Bilder dienen dazu, dem Netzwerk die charakteristischen Merkmale der Objekte beizubringen. Zweitens benötigt das Modell die entsprechenden korrekten Vorhersagen für diese Beispielbilder. Diese Vorhersagen werden mit den eigenen Vorhersagen des Modells verglichen, um Abweichungen festzustellen und das Modell zu verbessern. Daraus resultierend beeinflusst die Qualität der Trainingsdaten maßgeblich die Qualität des trainierten Modells.

Aus dieser Erkenntnis lässt sich die folgende Faustregel ableiten: Bilder sind geeignete Trainingsdaten, wenn Menschen in der Lage sind, das Objekt zu erkennen. Dies liegt daran, dass im Rahmen der Datenvorbereitung manuell Bounding Boxes in die Trainingsbilder gezeichnet werden müssen. Wenn es während dieses Prozesses zu Fehlern oder einer falschen Erkennung der Objekte durch den Menschen kommt, werden die Trainingsdaten verfälscht. Daraus resultierend kann der Trainingserfolg erschwert werden und es kann zu schlechteren Ergebnissen führen.

Darüber hinaus ist es wichtig, dass die Trainingsdaten vielfältig sind. Die Bilder sollten die zu erkennenden Objekte aus verschiedenen Winkeln und mit unterschiedlichen Lichtverhältnissen zeigen.

Soll das System vor einem variablen Hintergrund genutzt werden, ist es darüber hinaus von Bedeutung, dass Bilder mit unterschiedlichen Hintergründen einbezogen werden. Durch diese Variationen wird dazu beigetragen dass das trainierte Modell effektiv auf unbekannte Trainingsdaten reagiert.

Für die Auswahl der Trainingsdaten gibt es zwei Ansätze. Zum einen können die benötigten Bilder selbstständig aufgenommen werden. Alternativ kann auch ein oder mehrere Datensätze aus dem Internet für das Training verwendet werden.

Die Verwendung eigener Bilder hat den Vorteil, dass das Urheberrecht gesichert ist. Zudem können die Bilder gezielt am zukünftigen Einsatzort aufgenommen werden. Dadurch werden mögliche Probleme mit abweichenden Hintergründen vermieden. Ein großer Nachteil der eigenen Bilder ist, dass ausreichend unterschiedliche Bilddaten aufgenommen werden. Wie zuvor erwähnt, ist es wichtig, verschiedene Kamerawinkel und Lichtverhältnisse zu berücksichtigen. Daraus resultierend sind Aufnahmen eigener Daten mit einem erheblichen Zeitaufwand verbunden.

Bei der Verwendung von Fremdaufnahmen besteht die Möglichkeit, mehrere Datensätze parallel zu nutzen, um eine größere Varianz der Aufnahmeorte zu erzielen. Es ist jedoch wichtig, das Urheberrecht der jeweiligen Bildersteller zu beachte. Darüber hinaus können in den Bildsätzen fehlerhafte Daten enthalten sein. Deshalb müssen diese vor ihrer Verwendung manuell überprüft werden.

Das zu entwickelnde Modell soll möglichst flexibel in der Natur eingesetzt werden. Daraus resultierend ist eine hohe Anzahl von unterschiedlichen Bilddaten relevant. Angesichts des begrenzten Zeitrahmens für die Arbeit stellt die Verwendung von Fremddatensätzen in diesem Fall die bessere Alternative dar.

Da das Erkennen Insektenarten ein spezieller Anwendungsfall der Objekterkennung ist, ist die Anzahl der verfügbaren Datensätze begrenzt. Im Rahmen der Recherchen wurden einige Modelle identifiziert, welche geeignet sind. Diese sind die Modelle „bee_dataset“ von Tensorflow (28), „Bee and non-bee dataset“ von der University of South Florida (27) und „Bee or wasp?“ Von George Rey (24).

Für den vergleich der Datensätze sollen in den folgenden Kategorien genutzt werden:

1. Datensatzgröße: Die Größe des Datensatzes gibt an, wie viele Bilder insgesamt in dem jeweiligen Datensatz enthalten sind. Eine hohe Datenmenge hilft bei der Verbesserung der Modellleistung und kann eine größere Vielfalt von Merkmalen und Variationen abdecken. Ein umfangreicher Datensatz kann dazu beitragen, Overfitting zu reduzieren und die Robustheit des Modells gegenüber neuen Daten zu verbessern.

2. Varianz der Bilder: Die Varianz der Bilder bezieht sich auf die Vielfalt der visuellen Eigenschaften innerhalb des Datensatzes. Dies umfasst Variationen in Bezug auf Blickwinkel, Beleuchtung, Bildausschnitt, Hintergrund und andere visuelle Merkmale. Eine größere Varianz der Bilder kann dazu beitragen, dass das Modell robust gegenüber Veränderungen und Unterschieden in den Eingabedaten ist und somit besser auf neue und unbekannte Bilder reagieren kann.

3. Klassenanzahl: Die Klassenanzahl gibt an, wie viele verschiedene Kategorien an Insekten in dem Datensatz enthalten sind. Es ist wichtig, dass ausreichend Bilder der vier zu erkennenden Klassen enthalten sind.

4. Verhältnis der Klassen: Das Verhältnis der Klassen gibt Auskunft über die Verteilung der Klassen innerhalb des Datensatzes. Es zeigt, ob die Bilder einer Klasse überrepräsentiert oder unterrepräsentiert sind. Ein ausgeglichenes Verhältnis der Klassen ist das Ziel, um sicherzustellen, dass das Modell gleichermaßen auf alle Klassen trainiert wird.
5. Hintergrundtyp: Diese Kategorie gibt an, ob die Bilder in den Datensätzen vor einem definierten Hintergrund (z. B. Greenscreen) aufgenommen wurden oder ob verschiedene Hintergründe in den Bildern enthalten sind.
6. Nutzbarkeit: Die Nutzbarkeit gibt an, ob der Datensatz direkt verwendet werden kann. Falls Änderungen erforderlich sind ist dies ein Zeitfaktor, welcher bei der Auswahl beachtet werden muss.
7. Lizenz: Die Lizenz gibt an, unter welchen Bedingungen der Datensatz verwendet werden kann. Dies gilt insbesondere in Bezug auf die Nutzung, Weiterverteilung und Modifikation der Daten.

Durch die Kategorien sollen die Datensätze vergleichbar gemacht werden. Die Ergebnisse der Auswertung werden in Tabelle 1 dargestellt.

Kategorie	bee_dataset	Bee and non-bee	Bee or wasp?
Datensatzgröße	7.500	6332	10579
Varianz der Bilder	Niedrig	Hoch	Hoch
Klassenanzahl	2	53	3
Verhältnis der Klassen	Unausgeglichen	Ausgeglichen	Ausgeglichen
Hintergrundtyp	Greenscreen	Unterschiedlich	Unterschiedlich
Nutzbarkeit	Direkt	Mit Änderungen	Mit Änderungen
Lizenz	CC BY 4.0	CC BY 4.0	CC0

Tabelle 1 Auswertung Datensätze

Im Rahmen der Auswertung der Tabelleninformationen ergab sich, dass der Datensatz „bee_dataset“ nicht den Anforderungen genügt. Obwohl er eine hohe Datensatzgröße von 7.500 Bildern aufweist, enthält er nur zwei Klassen. Das Verhältnis der Klassen zudem unausgeglichen. Innerhalb der Bilder sind deutlich mehr Bienen als Wespen enthalten sind. Zudem wurden sämtliche Bilder vor einem Greenscreen aufgenommen, wodurch das trainierte Modell nicht für den Einsatz in der Natur geeignet ist.

Der Datensatz „Bee and non-bee“ weist eine hohe Varianz der Bilder auf. Die Bilder sind in 53 Klassen unterteilt aufgeteilt. Das bedeutet, dass für jede Klasse nur zwischen 200 und 500 Bilder vorhanden sind.

Aufgrund dieser Auswertung wurde die Entscheidung getroffen, das Modell „Bee or Wasp?“ zu verwenden. Dieses verfügt über 10.579 Bilddaten. Die grundlegende Verteilung der Klassen ist wie folgt: 3.183 Bilder zeigen Bienen, 4.943 Bilder zeigen Wespen und 2.453 Bilder zeigen sonstige Insekten. Dadurch ergibt sich ein akzeptables Verhältnis der Bildverteilung. Bei den Insekten und den Bienenbildern konnte eine ausrechende Menge von ca 1000 Bildern ausgemacht, welche der Klasse Hummeln zugeordnet werden kann. Dadurch sind genügend Trainingsdaten für die vier vorgesehenen Klassen vorhanden.

Zusätzlich ist der Datensatz unter der CC0: Public Domain-Lizenz veröffentlicht, was flexible Veränderungen am Datensatz ermöglicht.

3.2. Annotation der Datenquellen

3.2.1. Datensatzfilterung

Um die ausgewählten Datenquellen effektiv nutzen zu können, ist es notwendig, die Bilder zu annotieren. Die Annotation bezieht sich auf den Prozess der Zuordnung von Informationen zu Bildern. Soll eine Objekterkennung trainiert werden ist es erforderlich Objekte innerhalb der Bilder einzeln zu annotieren. Dazu gehören Koordinaten zur Lokalisierung des Objekts sowie die Zuweisung zu einer spezifischen Klasse. Der Annotierungsprozess besteht aus zwei Schritten: der Filterung der Bilder und der eigentlichen Annotation.

Im Rahmen der Filterung werden die Bilder im Datensatz überprüft. Es muss sichergestellt werden, dass die Bilder Insekten zeigen und der richtigen Klasse zugeordnet sind. Beispielsweise werden Bilder, welche kein Insekt, sondern beispielsweise eine Papierbiene zeigen entfernt.

Die Neusortierung der Bilder muss manuell vorgenommen werden, um die Korrektheit der Daten zu gewährleisten. Angesichts der großen Datenmenge von mehr als zehntausend Bildern ist dies zeitaufwendig. Daher wird im ersten Schritt nur eine kleinere Menge von Bildern zu sortieren. Mit diesen können erste Trainingsversuche gestartet werden. Für jede der vier Klassen (Biene, Hummel, Wespe und sonstige Insekten) werden dabei 200 Bilder annotiert. Dabei ist es wichtig, eine möglichst hohe Varianz in Bezug auf die Darstellung der einzelnen Klassen sicherzustellen, um trotz der begrenzten Anzahl an Bildern möglichst genaue Ergebnisse bei Untersuchungen und Modellvergleichen zu erzielen.

Sollte im Verlauf der Entwicklung der Bedarf an weitere Bilder steigen, können diese nachannotiert werden. Dadurch kann die Datenmenge schrittweise erweitert werden, um eine optimale Leistung des Modells zu erreichen.

3.2.2. Auswahl des Annotierungssystems

Bevor der ausgewählte Datensatz zum Trainieren eines Modells verwendet werden kann, müssen die Bilder annotiert werden. Die Annotation bezeichnet den Vorgang des Zuweisens von Informationen zu den Bildern. Im Falle der Objekterkennung handelt es sich dabei um Informationen zur Position und Klasse der Objekte auf den Bildern. Für die Annotierung gibt es zwei gängige Vorgehensweisen.

Die erste Variante besteht darin, die sortierten Bilder je Klasse innerhalb eines Dateiordners abzulegen. Mithilfe von TensorFlow können diese Dateiordner geladen werden. Dabei wird jeder Dateiordner als eine Klasse im Rahmen des Trainings betrachtet. Diese Art der Annotierung ist innerhalb kürzester Zeit umsetzbar. Er ermöglicht jedoch nur eine Klassifizierung der Objekte, da bisher keine Informationen über den genauen Standort und die Anzahl der Objekte innerhalb des Bildes hinterlegt sind. Demzufolge müssten sämtliche Bilder, die mehr als ein Objekt enthalten, von diesem Trainingsverfahren ausgeschlossen werden.

Die Alternative dazu ist das vollständige Annotieren der Bilder mittels eines Annotierungstools. Auf dem Markt existieren verschiedene Tools für das Annotieren von Bildquellen. Diese unterscheiden sich in ihrem Aussehen, ihren Funktionen und der Art, wie die Annotierungen abgespeichert werde. Der Vorteil von Annotierungstools besteht darin, dass die annotierten Bilder sowohl für das Training zur Objektklassifizierungsnetzen als auch für die Objektlokalisierung verwendet werden können. Zudem können Bilder genutzt werden, die mehr als ein Objekt pro Bild enthalten. Der Nachteil dieser Alternative besteht darin, dass das Annotieren der einzelnen Objekte deutlich zeitaufwändiger ist, da für jedes Objekt innerhalb des Bildes eine möglichst genaue Bounding Box gezeichnet werden muss.

Basierend auf diesem Vergleich wird eine begrenzte Anzahl an Bildern mittels des Annotierungstools LabelImg annotiert. Der Vorteil des universellen Einsatzes überwiegt bei der begrenzten Anzahl an Bildern den Arbeitsaufwand. Zudem kann bei Bedarf während der Entwicklung jederzeit auf die erste Variante zurückgegriffen werden, falls es erforderlich ist, die Anzahl der Bildquellen schnell und drastisch zu erhöhen.

4. Modellanalysen

4.1. Faktoren bei der Modellauswahl

Neben dem Datensatz ist die Struktur des NN der wichtigste Faktor für eine effektive Vorhersage. Die Struktur sollte dabei entsprechend des Einsatzortes und Einsatzzwecks ausgewählt werden.

Bei der Auswahl des Einsatzorts ist es wichtig zu beachten, wo das Netzwerk nach dem Training eingesetzt wird. GPUs besitzen eine große Anzahl an Rechenkernen. Außerdem haben sie speziellen Kernen für die Matrizenberechnung. Daraus resultierend sind GPUs bei der Berechnung von CNNs deutlich schneller als CPUs. Im Durchschnitt sind sie etwa zehnmal so schnell wie vergleichbare CPUs. Diese Aussage gilt, für Geräte derselben Preiskategorie und aus demselben Entwicklungsjahr.

Ein weiterer wichtiger Faktor beim Einsatzort ist die Performance der verwendeten Recheneinheit. Auf einer High-End-CPU kann ein CNN deutlich schneller berechnet werden als auf einem kleinen Mikrocontroller. Dies liegt daran, dass Mikrocontroller in der Regel eine niedrigere Taktrate und weniger Rechenkerne haben als High-End-CPUs. Mikrocontroller werden zudem häufig in batteriebetriebenen oder IoT-Geräten eingesetzt, weshalb sie energieeffizient arbeiten müssen.

Der Einsatzzweck bestimmt die Anforderungen an die Genauigkeit des neuronalen Netzes. Bei Anwendungen wie der Schildererkennung in selbstfahrenden Autos ist eine hohe Genauigkeit von entscheidender Bedeutung. Falsche Vorhersagen können hier zu schwerwiegende Konsequenzen führen. Dem gegenüber kann ein Algorithmus zur Schrifterkennung von Hausaufgaben eine geringere Genauigkeit verfügen, da Fehler in diesem Kontext weniger schwerwiegend sind.

Die Genauigkeit eines neuronalen Netzes hängt stark von der Größe des Modells ab. Größere Modelle haben in der Regel mehr Verbindungen zwischen den Neuronen, was zu einer komplexeren Entscheidungsfindung führt. Dies kann zu einer höheren Genauigkeit führen, wenn ausreichend Daten und Ressourcen vorhanden sind. Es ist jedoch wichtig zu beachten, dass die Größe allein kein Garant für die Genauigkeit ist. Zu große Modelle sind anfällig für Overfitting (siehe 2.5.1. Overfitting).

In den letzten Jahren wurden auch neue Ansätze und Methoden entwickelt, um mit kleineren neuronalen Netzen performantere und genauere Vorhersagen zu erzielen. Forschungsgruppen haben verschiedene Techniken wie Transfer Learning, Model Compression und Regularisierung eingesetzt, um die Leistung kleinerer Modelle zu verbessern. Diese Ansätze ermöglichen es, kompaktere Modelle mit weniger Parametern zu trainieren, ohne dabei signifikante Genauigkeitseinbußen hinnehmen zu müssen.

Auf Basis dieser beiden Faktoren ergibt sich für diese Arbeit, dass ein Fokus auf die Geschwindigkeit gelegt werden muss. Dies ergibt sich aus der Zielsetzung, dass das Modell auf batteriebetriebenen Geräten wie einem Handy oder einem Mikrocontroller Einsatz finden soll. Da solche Geräte im Vergleich zu Netzbasierten Computern begrenzte Ressourcen besitzen, ist es wichtig, ein Modell zu wählen, das schnell und effizient arbeitet. Sollen neben Bildern auch Videos analysiert werden können ist die Geschwindigkeit noch viel wichtiger. Denn bei der Echtzeitverarbeitung von Videostreams ist eine schnelle Verarbeitungsgeschwindigkeit von großer Bedeutung, um eine flüssige und reaktionsschnelle Anwendung zu gewährleisten.

Da es sich bei der Bienenerkennung nicht um eine sicherheitskritische Anwendung handelt, kann die Genauigkeit des Modells zugunsten der Geschwindigkeit reduziert werden. Ein Modell, das etwas weniger genau ist, aber schnellere Vorhersagen trifft, kann in diesem Fall die bessere Wahl sein.

4.2. Auswahl der zu untersuchenden CNN-Architekturen

Auf Basis der im vorausgegangenen Kapitel definierten Auswahlkriterien für CNN soll im Folgenden das zu verwendende Modell ausgewählt werden. Die Auswahl erfolgt in zwei Schritten. Im ersten Schritt wird aus der Vielzahl verfügbarer Modelle eine repräsentative Auswahl getroffen. Im zweiten Schritt werden diese ausgewählten Modelle analysiert und die Ergebnisse miteinander verglichen. Basierend auf dieser Analyse wird anschließend eine fundierte Entscheidung für die weitere Entwicklung getroffen.

Der erste Schritt der Modellauswahl basiert auf einem Benchmark der Cornell University. In diesem Benchmark wurden gängige Architekturen für Neuronale Netze hinsichtlich ihrer Größe, Genauigkeit und Geschwindigkeit untersucht. Die Ergebnisse dieser Untersuchung sind in Abbildung 5 dargestellt.

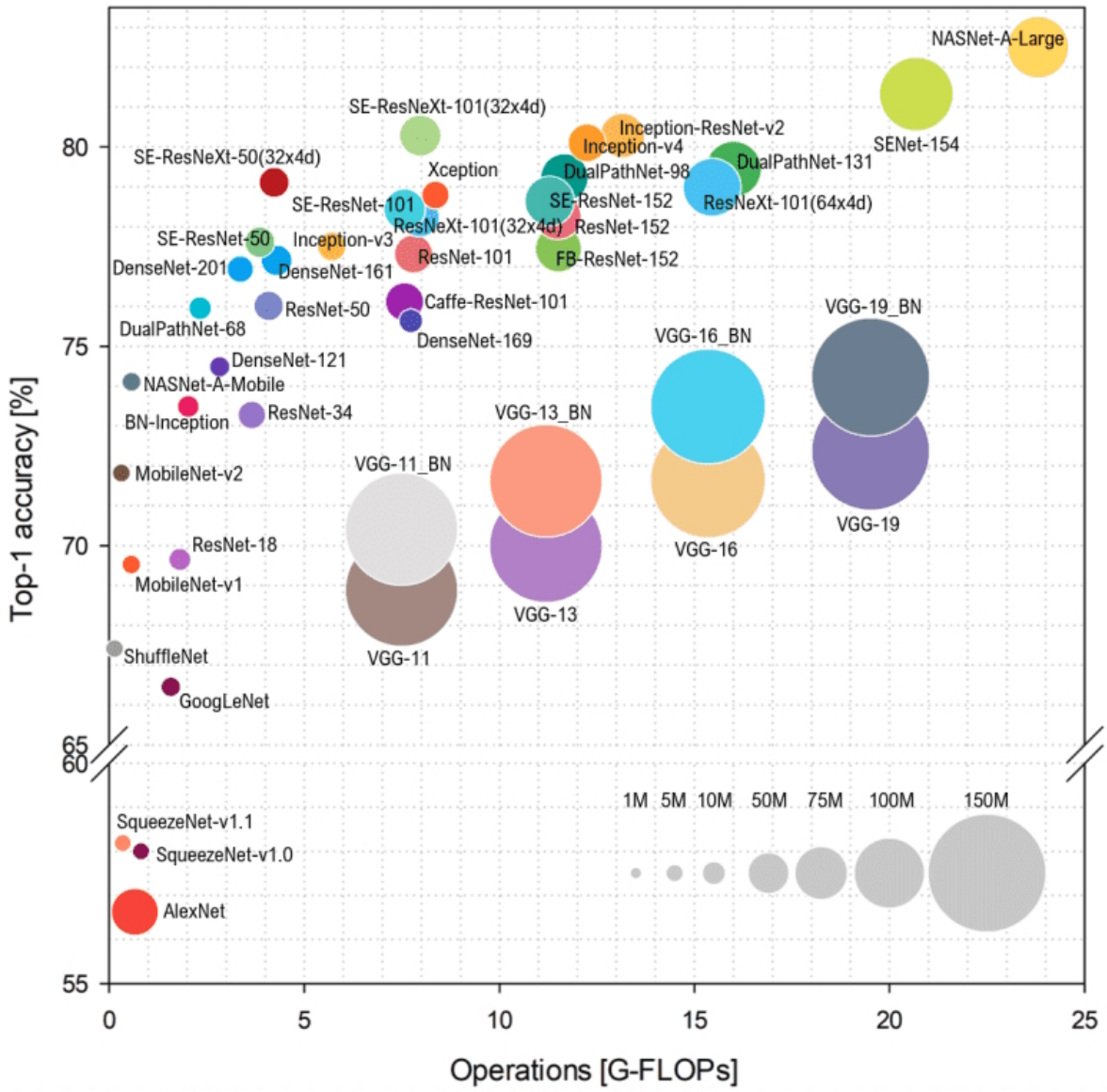

Abbildung 5 CNN-Modell Übersicht (26)

Die Grafik bietet einen Überblick über das Verhältnis von Genauigkeit zu Geschwindigkeit der Modelle. Die Genauigkeit wird in Prozent angegeben und bezieht sich auf die Top-1-Genauigkeit. Diese gibt an, wie oft das Modell die korrekte Klasse als die wahrscheinlichste vorhersagt. Die Geschwindigkeit der Modelle wird in G-FLOPs gemessen. FLOPs steht für Floating Point Operations und beschreibt die Anzahl der Fließkommazahl-Berechnungen, die die Recheneinheit zur Durchführung einer Vorhersage im Modell ausführt. Das G steht in diesem Fall für Giga, was eine Milliarde bedeutet. Somit entspricht 1 G-FLOPs einer Milliarde Fließkommazahl-Berechnungen.

Die Grafik gibt einen Überblick über das Verhältnis von Genauigkeit zu Geschwindigkeit. Die Genauigkeit wird in Prozent angegeben. Die Top-1-Genauigkeit bezieht sich dabei darauf, wie oft das Modell die korrekte Klasse als die am besten geeignete vorhersagt. Es ist die Genauigkeit bei der Betrachtung nur des wahrscheinlichsten Ergebnisses. (26)

Daneben gibt die Grafik mittels der Größe der Punkte an, wie viele Parameter das Modell besitzt. Parameter sind die Gewichte und Biases, die während des Trainingsprozesses des Modells optimiert werden, um die gewünschte Funktion zu erlernen. Diese werden in der Grafik in Millionen Parametern angegeben.

Basierend auf den vorgegebenen Kriterien wurden Modelle aus verschiedenen Kategorien ausgewählt. Hier ist eine Zusammenfassung der ausgewählten Modelle:

1. Kategorie: Modelle mit über 75% Top-1 Genauigkeit und weniger als 5 G-FLOPs:

- DenseNet201: Dieses Modell hat die geringsten Rechenoperationen und erfüllt den Genauigkeitsanforderungen. (11) (12)
- SE-ResNeXt-50: Dieses Modell bietet einen guten Kompromiss zwischen Genauigkeit und Rechenoperationen. (13) (14)

2. Kategorie: Modelle unterhalb von 75% Top-1 Genauigkeit und weniger als 5 G-FLOPs:

- NasNet-A-Mobile: Dieses Modell hat die höchste Genauigkeit unter den Modellen mit weniger als einem Giga-FLOP. (17) (18)
- ResNet-34: Ein älteres Modell mit guter Geschwindigkeit und akzeptabler Genauigkeit. (9) (10)

3. Kategorie: Modelle mit über 50 Millionen Parametern:

- Inception-ResNet-v2: Das Modell mit der besten Performance in dieser Kategorie. Es hat eine hohe Anzahl von Parametern. (15) (16)
- ResNet-101: Ein weniger genaues Modell als Inception-ResNet-v2, benötigt aber weniger Rechenoperationen. (9) (10)

4. Kategorie: Eigens entwickelte Modelle mit sehr geringere Größe (siehe 2.4. Architektur der eigenentwickelten CNN)

- Custom CNN V1
- Custom CNN V2

Diese Auswahl von Modellen ermöglicht es, verschiedene Aspekte wie Genauigkeit, Rechenoperationen und Modellgröße zu berücksichtigen. Die Modelle wurden basierend auf den gegebenen Kriterien ausgewählt, um die Anforderungen an kleine und schnelle Modelle zu erfüllen und gleichzeitig eine ausgewogene Leistung zu bieten. Sie sollten helfen, den Untersuchungsprozess weiterzuführen und Erkenntnisse darüber zu gewinnen, ob die kleinen Modelle für die Erkennung der geplanten Klassen ausreichend sind oder ob größere Modelle für höhere Genauigkeit erforderlich sind.

Ein ausschlaggebender Grund ist, dass die Modelle nur 4 Klassen erkennen müssen. Die meisten der zuvor genannten Modelle werden auf Datensätzen mit mehreren Millionen Bildern und hunderten Klassen trainiert. Die hohe Anzahl an Klassen bei der Entwicklung könnte dazu führen, dass diese Modelle unnötig groß sind und kleinere Modelle trotz ähnlicher Genauigkeit eine bessere Leistung erzielen. Um diese Vermutung zu überprüfen, werden die Modelle mit den wissenschaftlichen Modellen verglichen.

4.3. Analyse der CNN-Architekturen

Die zuvor ausgewählten Modelle sollen nun genauer untersucht werden. Hierfür werden sie auf einer standardisierten Hardware, einer NVIDIA L4 Grafikkarte vom Typ G2-Standard-4, mithilfe von Google Colab trainiert. Der Trainingsdatensatz besteht aus den zuvor annotierten 800 Bildern (siehe 3.2.2. Auswahl des Annotierungssystems).

Die Untersuchungskriterien umfassen die Trainingszeit, die Genauigkeit auf den Validierungsdaten und die Genauigkeit auf den Trainingsdaten. Besonders wichtig ist dabei die Genauigkeit auf den Validierungsdaten, da diese Daten dem Modell unbekannt sind und Aufschluss darüber geben, wie gut das Modell auf neue, unbekannte Daten reagiert. Die Genauigkeit beim Training dient als unterstützende Funktion. Falls diese Genauigkeit nicht nahe an 100% liegt, deutet das auf Probleme hin, da das Modell die erforderlichen Muster nicht erlernen kann, um die Objekte erfolgreich zu erkennen.

Die Trainingszeit gibt einen groben Anhaltspunkt dafür, wie schnell ein Modell Vorhersagen treffen kann. Die Modelle werden über 50 Trainingsepochen trainiert, wobei eine Epoche den Durchlauf aller Trainingsdaten und die Anpassung der Gewichte beschreibt. Die Trainingszeit ermöglicht einen groben Vergleich zwischen den Modellen, um festzustellen, wie schnell sie im Verhältnis zueinander Vorhersagen treffen können.

Typ	Train. Accuracy	Val. Accuracy	Training Time (50 Epoch
ResNet-34	99,64%	84,15%	00:23:26
NASNet-A-Mobile	93,75%	70,42%	00:35:34
DenseNet201	99,13%	90,00%	00:31:03
SE-ResNeXt-50	99,89%	85,51%	00:49:15
ResNet-101	99,70%	84,45%	00:48:11
Interception-Resnet	99,62%	92,23%	00:39:28
Custom CNN-V1	99,06%	89,85%	00:29:51
Custom CNN-V2	86,25%	87,32%	00:58:23

Abbildung 6 Auswertung CNN-Modell Untersuchung

Die Abbildung 6 Auswertung CNN-Modell Untersuchung auf Seite 21 zeigt die Ergebnisse der durchgeführten Untersuchung des CNN-Modells. In Bezug auf die Trainingsgenauigkeit (Train. Accuracy) zeigt sich, dass fast alle Modelle eine Genauigkeit von über 99% aufweisen und somit für die Erkennung der Objekte in dieser Studienarbeit geeignet sind. Der einzige Ausreißer ist das Custom CNN-V2, das keine Genauigkeit von über 90% erreicht.

Um das am besten geeignete Modell für die geplante Anwendung sicher zu bestimmen, soll eine Gesamtpunktzahl ermittelt werden. Diese Gesamtpunktzahl hilft bei der Auswahl der besten Modelle aus den getesteten Modellen. Dazu werden den Faktoren Validierungsgenauigkeit (Validation Accuracy) und Trainingszeit (Training Time) jeweils Dezimalzahlen zwischen 1 und 8 zugeordnet. Das Modell mit der höchsten Genauigkeit bzw. der schnellsten Trainingszeit erhält den Wert 8, und die weiteren Modelle erhalten die entsprechenden Zahlen in absteigender Reihenfolge.

Die Gesamtpunktzahl wird aus den Validierungspunkten und den Trainingspunkten berechnet. Dabei wird ein Gewichtungsfaktor von 1,5 auf die Trainingspunkte angewendet. Dadurch erhalten Modelle mit einer niedrigeren Genauigkeit, aber gleichzeitig höherer Geschwindigkeit, in der Gesamtbewertung einen Vorteil. Die Formel hierzu ist in Formel 1 dargestellt.

$$Gesamtpunktzahl = Validierungspunkte + (Trainingspunkte * 1,5)$$

Formel 1 Berechnung Gesamtpunktzahl

Die folgende Tabelle 2, zeigt die Gesamtpunktzahlen, die aus der Formel berechnet wurden. Die Modelle Custom CNN-V1 und DenseNet201 zeichnen sich mit 16,5 bzw. 16 Punkten im Vergleich zu den anderen Modellen besonders aus. Beide Modelle wurden in etwa 30 Minuten mit einer Genauigkeit von ca. 90% trainiert, was eine sehr gute Leistung in beiden Kategorien darstellt. Daher sollten beide Modelle parallel für die Entwicklung der Objekterkennung verwendet werden. Darüber hinaus hat die Auswertung ergeben, dass das ResNet-34 Modell eine akzeptable Genauigkeit bei hoher Geschwindigkeit aufweist. Daher sollte dieses Modell als Backup behalten werden, falls die beiden verwendeten Modelle für die Echtzeitbilderkennung nicht schnell genug sind.

Typ	Val. Accuracy	Val. Points	Training Time	Train Points	Gesamtpunkte
ResNet-34	**84,15%**	**2**	**00:23:26**	**8**	**14,0**
NASNet-A-Mobile	70,42%	1	00:35:34	5	8,5
DenseNet201	**90,00%**	**7**	**00:31:03**	**6**	**16,0**
SE-ResNeXt-50	85,51%	4	00:49:15	2	7,0
ResNet-101	84,45%	3	00:48:11	3	7,5
Interception-Resnet	92,23%	8	00:39:28	4	14,0
Custom CNN-V1	**89,85%**	**6**	**00:29:51**	**7**	**16,5**
Custom CNN-V2	87,32%	5	00:58:23	1	6,5

Tabelle 2 Auswertung CNN Punktebewertung

5. Objekt Lokalisierung

In der Praxis ist es selten der Fall, dass Objekte in einem Bild perfekt zur Klassifikation positioniert sind. Außerdem können auf einem einzelnen Bild häufig mehrere Objekte unterschiedlicher Klassen sichtbar sein. Wenn das Ziel eines neuronalen Netzwerks beispielsweise das Zählen oder Erfassen bestimmter Objekte ist, zum Beispiel für Ausweichmanöver in einem selbstfahrenden Auto oder Luftabwehr im militärischen Bereich, ist es erforderlich, Objekte zu lokalisieren. Im Folgenden werden zwei Ansätze zur Lokalisierung parallel betrachtet, und basierend auf den Fähigkeiten und Herausforderungen beider Ansätze wird eine Auswahl für den endgültigen Einsatz getroffen.

5.1. Sliding Window

Eine seit langem bekannte Technik zur Objektlokalisierung ist die Sliding Window Technik. Die Basis für diese Technik ist ein Modell zur Objektklassifizierung. Dieses soll nun genutzt werden, um Objekte auf einem Bild zu lokalisieren. Um dies zu erreichen, wird das Bild in Teilbilder aufgeteilt. Jedes Teilbild wird separat an das Klassifizierungsnetzwerk übergeben und die Vorhersagen des Modells werden gespeichert. Anschließend werden die Vorhersagewerte analysiert, um die Position der Objekte auf dem Bild zu ermitteln.

Bei der Analyse der Abbildung 7 wird dies ersichtlich. In Feld 2 erkennt das Modell die Klasse „Biene“ mit einer hohen Wahrscheinlichkeit, während die Vorhersagewerte für die restlichen Felder sehr niedrig ausfallen. Aufgrund dieser Unterschiede kann mathematisch eingegrenzt werden, in welchem Teilbild sich das Objekt befindet.

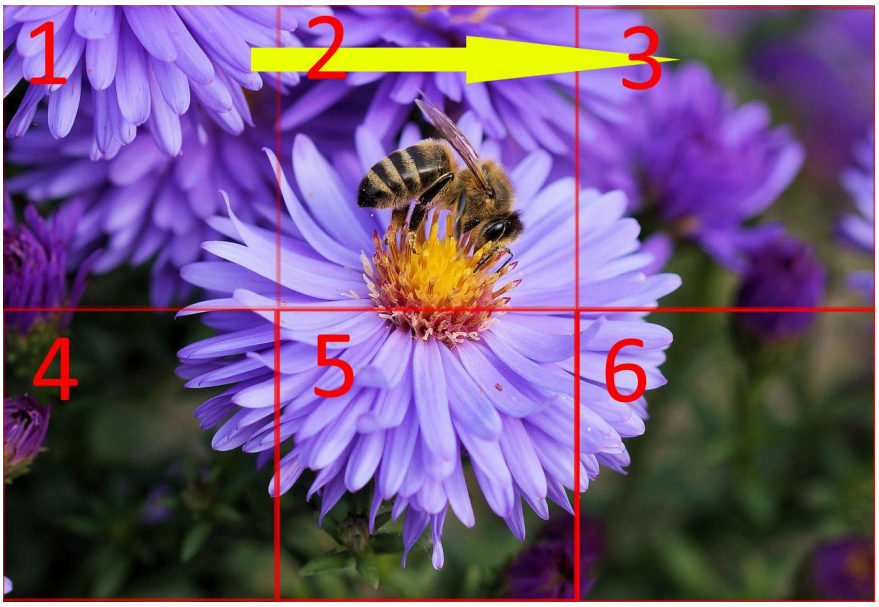

Abbildung 7 Sliding Window Vorgehen

Im gezeigten Beispielbild ist das Fenster um die Biene noch ziemlich groß. Dies kann dazu führen, dass mehrere Objekte innerhalb eines Fensters enthalten sind. Außerdem kann es vorkommen, dass die Biene in mehreren Fenstern liegt. Beide Effekte führen zu einer ungenauen Lokalisierung. Um die Genauigkeit der Lokalisierung zu verbessern, kann das Beobachtungsfenster anstatt einer ganzen Fensterbreite ausschließlich um einige Pixel verschoben werden. Dabei wird bei jeder Verschiebung eine Analyse des Teilbildes durchgeführt, wodurch sich die Vorhersagen stark überlappen. Ein Beispiel für die Verschiebung des Fensters ist in Abbildung 8 gegeben.

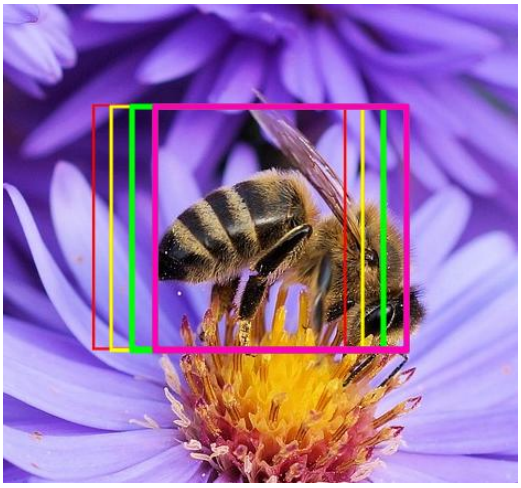

Abbildung 8 Sliding Window

Dieses Vorgehen erhöht die Genauigkeit des Modells, da die Schrittgröße beim Verschieben des Fensters verkleinert wird. Dadurch steigt die Wahrscheinlichkeit, dass Objekte in einem der Teilbilder vollständig erfasst werden. Im Beispielbild ist dies bei dem lila Teilfenster der Fall.

Da Objekte im Bild unterschiedliche Größen haben können, kann es vorkommen, dass das Sliding-Window-Verfahren entweder das gesamte Objekt zu keinem Zeitpunkt erfasst oder zu jedem Zeitpunkt mehrere Objekte erfasst. Um die Vorhersagegenauigkeit weiter zu verbessern, kann der Prozess mit verschiedenen Fenstergrößen durchgeführt werden. Die Größenunterschiede zwischen den einzelnen Schritten des Fensters hängen von den analysierten Bildern und der Größe der Objekte auf diesen Bildern ab.

Bei der Entwicklung des Sliding-Window-Verfahrens für die Insektenerkennung hat sich gezeigt, dass drei Durchläufe sinnvoll sind, wobei die Fenstergröße jede Durchlaufgröße 20% bis 35% kleiner ist als die vorherige. Durch diese Verkleinerung der Fenstergröße wird sichergestellt, dass sowohl kleine als auch große Objekte auf dem Bild erkannt werden können, und die Genauigkeit steht in einem guten Verhältnis zur Geschwindigkeit des Modells. Die Startgröße des Sliding Windows im ersten Durchlauf beträgt 10% der Bildgröße, sowohl in der Höhe als auch in der Breite des zu analysierenden Bildes.

In der Praxis traten bei der Umsetzung des Sliding-Window-Ansatzes deutliche Probleme auf. So kam es häufig vor, dass die eingesetzten Klassifizierungsmodelle bei Teilausschnitten ohne Objekt trotzdem eine Vorhersage für verschiedene Klassen lieferten. Dieses Problem trat sowohl bei Trainingsversuchen auf, bei denen sowohl Insekten als auch viel Hintergrund auf den Trainingsdaten zu sehen waren, als auch bei Trainingsläufen, bei denen die Trainingsdaten ausschließlich Bilder des Objekts enthielten.

Ein möglicher Ansatz zur Lösung dieses Problems besteht darin, Bildaufnahmen zu ergänzen, welche kein Insekt erhalten. Auf diese Weise könnte das Modell darauf trainiert werden, bei solchen Bildern alle Klassen nahezu null vorherzusagen. Allerdings erfordert dies eine vollständige Änderung des Ladealgorithmus, da dieser derzeit die Klassen basierend auf dem Ordner lädt, in dem die Bilder einer Klasse gespeichert sind.

Die Umsetzung des Sliding-Window-Verfahrens wurde zusätzlich durch die Ähnlichkeit verschiedener Insektenarten erschwert. Ein Beispiel dafür sind Schwebefliegen. Diese besitzen zur Tarnung ein Muster, welches sie aussehen läst wie Bienen. Dies dient zur Tarnung vor Fressfeinden, erschwert jedoch auch die Erkennung durch das Netzwerk. Die Abbildung 10 und Abbildung 9 zeigen beide ein Insekt auf einer pinken Blume. Im linken Bild handelt es sich um eine Schwebefliege (Klasse: sonstige Insekten), im rechten Bild um eine Honigbiene (Klasse: Biene). Die Unterscheidung ist der beiden Tiere nur anhand des Gesamtbildes auf Basis des Körperbaus möglich.

Wenn dem Klassifizierungsmodell nun im Verfahren ein Ausschnitt der Abbildung 10 gegeben wird, welcher nur den Kopf einer der Fliege zeigt, kann es vorkommen, dass das Modell diesen Ausschnitt als Biene klassifiziert. Durch das Verschieben des Fensters werden die Merkmale der Fliege sichtbar, und das Modell erkennt das weitere Fenster als Fliege.

Abbildung 10 Schwebefliege auf einer Blume

Abbildung 9 Biene auf einer Blume

Die Problematik hierbei ist, dass das Modell bei der Erkennung des Kopfausschnittes teilweise eine höhere Sicherheit für die Erkennung als Biene angibt, als dies bei der Erkennung als Fliege der Fall ist. Würden nun die beiden Fenster vereinigt und die Klasse mit der höheren Sicherheit ausgewählt resultiert dies in einer falschen Erkennung der Schwebefliege als Biene.

Darüber hinaus besteht die Möglichkeit, dass auf einem Bild sowohl eine Fliege als auch eine Wespe nebeneinander vorhanden sind, da Wespen Fliegen als Beutetiere jagen.

Daraus resultierend kann eine Vereinigung von zwei unterschiedlichen Vorhersagen nebeneinander nicht automatisch vereinigt werden. Dies führt dazu, dass ein solches Objekt als zwei unterschiedliche Objekte final erkannt wird.

Zusätzlich zu den bisher beschriebenen Problematiken zeigte sich in der Entwicklung, dass das Sliding Window verfahren nicht für den Einsatz in der Bilderkennung verwendet werden kann. Für die Analyse eines gesamten Bildes kann die benötigte Gesamtzeit (t_{ges}) mit der Formel 2 berechnet werden. Bedingung für die Anwendung dieser vereinfachten Formel ist, dass sowohl das Bild als auch das Sliding Window eine quadratische Größe besitzen.

$$t_{ges} = t_{pred} \times \sum_{i=0}^{n} \left(\frac{S_{img}}{S_{win} + (S_{win} \times F_{slid} \times n)} \times A_{step} \right)^2$$

Formel 2 Gesamtzeit Sliding Window

Um die benötigte Zeit zu berechnen, muss die Berechnungszeit pro Bildausschnitt (t_{pred}) mit der Summe der Bildausschnitte multipliziert werden. Die Anzahl der Bildausschnitte ergibt sich dabei aus der Schrittweite des Sliding Windows (A_{step}) multipliziert mit dem Quotienten aus der Bildgröße (S_{img}) und der Größe des Sliding Windows.

Die Größe des Sliding Windows ist dabei abhängig von der Ursprünglichen Sliding Window Größe (S_{win}) sowie dem Änderungsfaktor des Sliding Windows (F_{slid}) und abhängig von der Anzahl der Durchlaufschritte (n) berechnet.

Auf Basis dieser Formel ergibt sich bei einem Bild mit der Auflösung 600x600 Pixeln ($S_{img} = 600$) sowie einer Startwindowsize von 30x30 Pixeln ($S_{win} = 30$) bei drei Durchläufen des Sliding Windows ($n = 3$) und einer Vergrößerung des Sliding Windows je durchlauf um ($F_{slid} = 0{,}3$) von 30 Prozent sowie keinem Zwischenschritt ($A_{step} = 1$)eine Gesamtzeit des Algorithmus (t_{ges}) von 14,92 Sekunden.

Durch die hohe Berechnungszeit in Kombination mit der ungenauen Vorhersage des Modells kann das Sliding Window verfahren nur schlecht zur Erkennung von Insekten in raschen Bildabfolgen genutzt werden. Daher wird in der weiteren Betrachtung der Fokus auf die Lokalisierung mittels CNN gelegt.

5.2. Lokalisierung durch ein CNN

5.2.1. Einzelobjekt Lokalisierung

Im Rahmen der CNN-Architekturanalyse (siehe Kapitel 4.3.) wurden Neuronale Netze zur Klassifizierung von Bildern trainiert. Hierbei kann das CNN durch die Anwendung von Filtern innerhalb der Convolutional-Layer Kanten und Merkmale verstärken machen. Basierend auf diesen Merkmalen trifft das CNN eine Entscheidung über die Klassenzuordnung des Bildes. Das neuronale Netzwerk weiß nicht, dass es sich bei den vorhergesagten Werten um Klassenzuweisungen handelt. Es hat ausschließlich auf Basis der Trainingsdaten gelernt, wie es die Merkmale von Objekten den gewünschten Ausgabewerten zuordnet.

Aufgrund dieses Wissens kann vermutet werden, dass ein Modell nicht nur auf die Klassifikation von Daten trainiert werden kann. Es besteht die Möglichkeit, durch Änderung der korrekten Vorhersagen innerhalb der Trainingsdaten von Klasseninformationen auf Koordinaten umzusteigen und dem Modell beizubringen, die Position von Objekten auf Bildern zu erkennen. Diese Theorie soll im Folgenden genauer untersucht werden.

Die erste Herausforderung besteht darin, eine Methode zu entwickeln, um die Koordinaten der Bounding Boxes mit dem Modell zu teilen. Das neuronale Netzwerk gibt seine Vorhersagen an den Ausgabeneuronen in Form von Fließkommawerten zwischen 0 und 1 aus. Da ein Punkt auf einem digitalen Bild durch Koordinaten bestehend aus X und Y angegeben werden, müssen diese Ganzzahlen zuerst umgewandelt werden. Hierzu wird die Formel 3 angewendet. Diese konvertiert die X- und Y-Koordinaten getrennt voneinander in prozentuale Positionswerte. er Wert qMax hängt vom verwendeten Koordinatenteil ab. Wenn es sich um den X-Teil handelt, entspricht qMax der Breite des Gesamtbildes in Pixeln. Wenn es sich um den Y-Teil handelt, entspricht qMax der Höhe des Bildes in Pixeln.

$$Floatkoordinate = \frac{q}{qMax}$$

Formel 3 Floatkoordinatenberechnung

Da der resultierende Wert nach der Umwandlung liegt zwischen 0 und 1 und repräsentativ für die Position des Pixels steht, ist dieser unabhängig von der Auflösung des Bildes. Das bedeutet, dass das neuronale Netzwerk die Position der Objekte unabhängig von der Größe des Bildes vorhersagen kann.

Allerdings reicht ein einzelner Koordinatenpunkt nicht aus, um ein Objekt zu lokalisieren. Dieser liegt daran, da dieser keine Informationen über die Größe des Objekts auf dem Bild enthält. Um eine Bounding Box zeichnen zu können, werden daher mindestens zwei Koordinatenpunkte benötigt. Diese müssen aus zwei entgegenliegenden Ecken des Rechtecks bestehen, beispielsweise der oberen linken und der unteren rechten Ecke. Auf Basis dieser beiden Koordinaten können die fehlenden beiden Ecken des Rechtecks (oben rechts und unten links) berechnet werden. Daraus resultierend benötigt das CNN vier Ausgabeneuronen, um die vier Koordinatenwerte für die Bounding Box auszugeben. Eine mögliche Struktur der Ausgabeneuronen ist in Abbildung 11 dargestellt.

xMin	yMin	xMax	yMax

Abbildung 11 Ausgabe der Objektlokalisierung

Die Koordinaten xMin und yMin stehen hierbei für die obere Linke Ecke, während die untere rechte Ecke mittels xMax und yMax ausgegeben werden. Auf Basis der Formel 3 in Pixelwerte zurückgerechnet werden.

5.2.2. Grundlage der Multiobjekterkennung

Das bisher entwickelte Modell ist derzeit nur in der Lage, ein einzelnes Objekt pro Bild zu erkennen. Wenn dem Modell jedoch ein Bild mit mehreren Objekten präsentiert wird, tritt ein unvorhersehbares Verhalten auf. In solchen Fällen wird die Bounding Box willkürlich im Bild platziert. Dieses Verhalten lässt sich erklären, da das Modell bisher ausschließlich auf Bilder trainiert wurde, die jeweils ein einzelnes Objekt enthalten. Daher ist das Modell nicht in der Lage zu wissen, wie es mit mehreren Objekten umgehen soll. Infolgedessen kann es passieren, dass ein zufälliges Objekt erkannt wird oder dass eine Bounding Box gezogen wird, die alle Objekte einschließt.

Um das Modell zu erweitern, müssen die Ausgabeneuronen entsprechend erhöht werden. Das bisherige Modell verwendet vier Ausgabeneuronen, um die Koordinaten der oberen linken und unteren rechten Ecke der Bounding Box zu beschreiben. Wenn nun x Objekte erkannt werden sollen, werden für jede Bounding Box 4 Ausgabeneuronen benötigt. Daraus resultierend kann die Anzahl der Ausgabeneuronen mittels der Formel 4 berechnet werden.

$$y = 4 * x$$

Formel 4 Berechnung Ausgabeneuronen bei Multiobjekterkennung

Auf Basis der Formel folgt, dass ein Modell zur Erkennung von zwei Objekten über 8 Ausgabeneuronen verfügen muss. Für eine bessere Übersicht werden die Koordinaten für das Zeichnen der Bounding Boxes im Folgenden als „Objekte“ bezeichnet. Zur Anordnung der Informationen wirddie in der Einzelobjekterkennung verwendete Matrix erweitert. Dabei werden die Objekte hintereinander aufgelistet, wie in Abbildung 13 dargestellt ist.

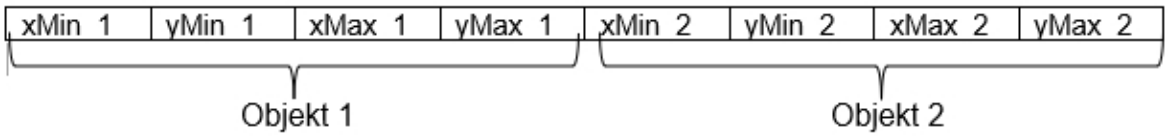

Abbildung 12 Tabellenstruktur bei 2 Objekten in der Objektlokalisierung

Diese Ausgabestruktur kann durch beliebig viele Objekte ergänzt werden, indem diese an das Ende der Tabelle angehängt werden. Die Tabelle ist dabei eine Darstellung, wie die Informationen für die weitere Bearbeitung in einem Array strukturiert sind.

Mit der angepassten Struktur der Ausgabeneuronen ist es erforderlich, die Trainingsdaten zu erweitern. Bisher wurde das neuronale Netz nur auf Bilder trainiert, die jeweils ein Objekt einer Klasse enthalten haben. Um dem CNN beizubringen, die maximale Anzahl von x erkennbaren Objekten zu erkennen, müssen die Trainingsdaten Bilder enthalten, die eine variable Anzahl von Objekten aufweisen können.

Bei dieser Erweiterung ist es wichtig, sowohl Bilder hinzuzufügen, welche über die maximal mögliche Anzahl Objekte verfügen. Zusätzlich sollten Bilder enthalten sein, welche eine Objektanzahl zwischen 1 und Maximum (x) besitzen. Durch diese Mischung wird sichergestellt, dass das Modell robuster auf verschiedene Objektmengen in einem Bild reagiert. Hierbei ist es erforderlich, dem Modell eine Möglichkeit zu geben, wie viele Objekte tatsächlich enthalten sind. Das Netzwerk muss bei jeder Berechnung sämtliche Felder der Ausgabetabelle berechnen, weshalb durch den Entwickler eine Definition für leere Objekte spezifiziert werden muss.

Eine Lösung für dieses Problem besteht darin, die ungenutzten Zellen der Ausgabetabelle auf null oder eins zu setzen. Es besteht jedoch die Gefahr, dass das Modell aufgrund von Unsicherheiten die nicht genutzten Objektfelder nicht exakt auf null setzt, sondern auf Werte nahe null, was zu Artefakten führen könnte. Um zu untersuchen, ob dieses Problem tatsächlich auftritt, muss das Modell in der Praxis trainiert werden und die Ergebnisse analysiert werden.

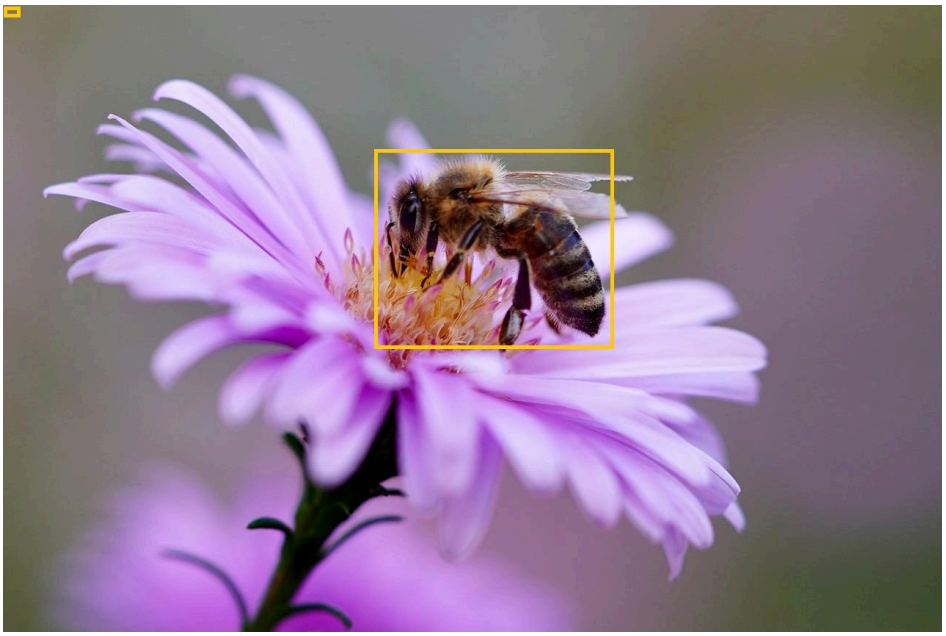

Abbildung 13 Beispielbild Multiobjekterkennung

Die Analyse des trainierten Modells zeigt dabei, dass Artefakte auftreten und nicht vorhandene Objekte mit werten größer als null gefüllt werden.

Die Koordinatenpunkte der nicht genutzten Objekte liegen nahe am Nullpunkt und sind eng beieinander, wodurch sie herausgerechnet werden können. Dabei werden Bounding Boxes, die weniger als 1% der Bildbreite einnehmen, nicht angezeigt. Dieser prozentuale Schwellenwert wurde durch verschiedene Testläufe ermittelt. Die Abbildung 13 zeigt ein Beispiel für die Artefaktbildung in der oberen linken Ecke.

Eine weitere Erkenntnis aus dem Training mit zwei Objekten ist, dass die Genauigkeit des Modells im Vergleich zur Einzelobjekterkennung deutlich abnimmt. Es konnten zwei mögliche Gründe für diese Verschlechterung ausgemacht werden. Zum einen besteht die Möglichkeit, dass die Anzahl der Neuronen und damit die Größe des Modells zu gering sein, um mehrere Objekte zuverlässig erkennen zu können. Dieser Theorie wiederspricht jedoch, dass sowohl das kleine Custom_CNN_V1 als auch das deutlich größere DenseNet201 ähnlich schlecht performen. Da das DenseNet201 für deutlich mehr Objekte und Klassen in der Erkennung entwickelt wurde.

Eine Analyse des Trainingsprotokolls zeigt, dass das Modell im Vergleich zur Einzelobjekterkennung deutlich höheren Fehlerwerte aufweist. Diese Fehler werden im Englischen als Loss bezeichnet, und die Funktion zur Berechnung des Loss wird als Loss-Funktion bezeichnet. Der Loss-Wert gibt an, wie stark die Vorhersagen des Modells von den korrekten Vorhersagen abweichen. Es ist daher wahrscheinlich, dass die Loss-Funktion in Kombination mit der Modellstruktur zu einer Verschlechterung der Genauigkeit führt.

Für die Umsetzung der Loss-Funktion gibt es verschiedene mathematische Ansätze, um die Abweichungen zwischen den Vorhersagen des Modells und den korrekten Vorhersagen zu berechnen. Eine häufig verwendete Formel für die Objekterkennung ist der Mean-Square-Error (MSE). Diese Formel wurde für die bisherigen Trainingsdurchläufe verwendet. Die genaue Formel zur Berechnung des MSE ist in Abbildung 14 dargestellt.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_{\text{true}_i} - y_{\text{pred}_i})^2$$

Abbildung 14 Berechnung Mean-Square-Error

Beim MSE wird jeweils der vorhergesagte Wert vom zugehörigen korrekten Vorhersagewert abgezogen. Dies wird für alle Zellen der Ausgabetabelle durchgeführt. Die Summe dieser Differenzen wird anschließend durch die Anzahl der Zellen geteilt, um einen Mittelwert zu erhalten. Dieser Mittelwert gibt an, wie stark die Vorhersagen des Modells im Durchschnitt von den korrekten Werten abweichen.

Das Problem bei der Verwendung des MSE liegt darin, dass das Modell im Wesentlichen darauf trainiert wird, die Positionierung der Objekte innerhalb der Tabelle vorherzusagen, anstatt die genauen Werte zur Lokalisierung der Objekte zu erlernen.

Der MSE berechnet die Differenz zwischen den korrekten und vorhergesagten Werten für jede Zelle des neuronalen Netzwerks separat und gibt dann einen Mittelwert für diese Differenzen an. Dies führt dazu, dass der MSE-Wert gegen 0 strebt, wenn die Positionen der Objekte korrekt bestimmt und die Objekte in der richtigen Reihenfolge in der Ausgabetabelle angeordnet sind. Im Gegensatz dazu steigt der MSE stark an, wenn die Objekte in der falschen Reihenfolge angeordnet sind, selbst dann wenn die Positionen korrekt vorhergesagt wurden. Ein Beispiel für diese Problematik wird in Abbildung 15 dargestellt. Hier sind die Objekte eins und zwei in der Vorhersagen vertauscht zu den Trainingsdaten.

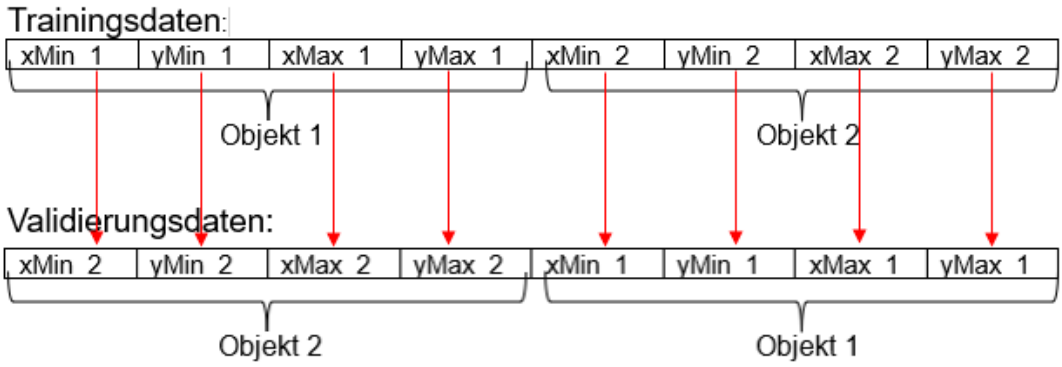

Abbildung 15 Darstellung Berechnung Mean-Square-Error

Diese Theorie wurde durch ein Training mit einer maximalen Anzahl von 10 zu erkennenden Objekten bestätigt. Dabei konnte das neuronale Netzwerk nicht mehr richtig verstehen, worauf es trainiert werden sollte, wodurch die Genauigkeit beim Training nicht über eine Validierungsgenauigkeit von 20 Prozent hinauskam. Gleichzeitig stieg die Loss-Funktion während des Trainings stark an.

Um das Problem der Loss-Berechnung zu lösen, ist es erforderlich, dass die Vorhersagen des Modells mit den entsprechenden korrekten Vorhersagen abgeglichen werden. Das Hauptziel besteht darin, eine korrekte Sortierung der Koordinaten bei der Berechnung des Mean Square Error (MSE) zu ermöglichen. Dies wird in Abbildung 16 dargestellt.

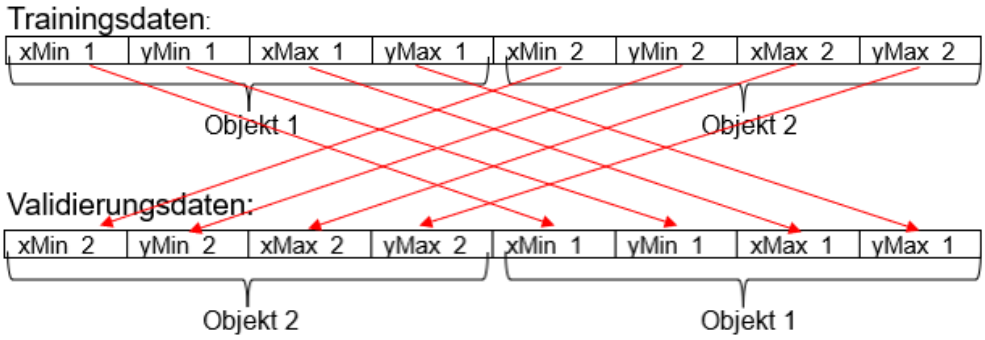

Abbildung 16 Korrekte Berechnung MSE

5.2.3. Analyse der YOLO V1 Loss-Funktion

Die Entwicklung einer eigenen Loss-Funktion stellt eine Herausforderung dar. Dies liegt in der Struktur von Tensorflow begründet. Python, als eine hardwareabstrakte Programmiersprache, legt den Fokus auf eine einfache und lesbarere Syntax, während die Performance im Vergleich zu hardwarenahen Sprachen wie C oder C++ geringer ist. Zusätzlich verwendet Python einen Just-in-Time Compiler, der den Code zur Laufzeit in Maschinensprache umwandelt. Diese Eigenschaften führen dazu, dass nativer Python-Code langsamer ist als in hardwarenahen Sprachen.

Um die Performance in rechenintensiven Anwendungen zu verbessern, werden Frameworks wie Tensorflow in C++ entwickelt und vor der Verwendung kompiliert. Tensorflow stellt eine API für die darunterliegende C++-Bibliothek bereit, um eine höhere Performance zu erreichen. Aufgrund dieser Struktur kann auf Tensorflow Daten nur mittels der API zugegriffen werden.

In Tensorflow werden Variablen und Informationen nicht in herkömmlichen Variablen gespeichert, sondern in Tensoren. Tensoren sind Datenstrukturen, die die Informationen der einzelnen Vorhersagevariablen enthalten. Eine Besonderheit ist jedoch, dass nicht garantiert ist, dass alle Variablen zu jedem Zeitpunkt der Laufzeit innerhalb eines Tensors enthalten sind. Dies liegt daran, dass Tensorflow die Parallelverarbeitung auf der GPU unterstützt und die Verfügbarkeit der Variablen dies ermöglicht.

Aufgrund der Tensorstruktur und der Verwendung der Tensorflow-API ist es notwendig, entweder die mathematischen Funktionen von Tensorflow für die Berechnung der Loss-Funktion zu verwenden oder den Tensor in ein Numpy Array umzuwandeln. Das Umwandeln des Tensors in ein Numpy Array ist technisch möglich, jedoch führt dies zu einer erheblichen Verlangsamung der Trainingszeit, um mehr als das 20-fache.

Aus diesem Grund wird empfohlen, die mathematischen Funktionen von Tensorflow für die Berechnung der Loss-Funktion zu nutzen. Allerdings besteht bei diesen Funktionen ein wesentliches Problem. Die von Tensorflow angebotenen Funktionen ermöglichen es nicht direkt, die Strukturierung innerhalb der Tabelle basierend auf Vergleichen zu sortieren. Daher ist es erforderlich, die gesamte Struktur des Modells, einschließlich des Datenladens und der Datenverarbeitung, zu überarbeiten, um eine auf das Projekt angepasste Loss-Funktion zu implementieren. Diese Restrukturierung der Entwicklungsumgebung ist mit einem hohen zeitlichen Aufwand verbunden. Daher soll im Zuge der Neuentwicklung analysiert werden, wie CNN zur Objektlokalisierung welche bereits verfügbar sind die Problematik der Loss-Funktion lösen.

Das YOLO-Netzwerk (You Only Look Once) ist ein bekanntes Modell für die Objekterkennung. Der Name „You Only Look Once“ bedeutet, dass das Modell das gesamte Bild nur einmal betrachtet und in einer einzigen Vorhersage sowohl die Position als auch die Klasse der erkannten Objekte liefert.

Das erste YOLO-Modell, YOLO_V1, wurde 2016 in einem wissenschaftlichen Paper veröffentlicht. Seitdem wurde das Modell kontinuierlich weiterentwickelt und die aktuelle Version, YOLO_V7 im Jahr 2023, zählt zu den effizientesten Modellen für die Objekterkennung. Die Funktionsweise der Loss-Funktion wurden im Laufe der Jahre nur geringfügig angepasst. Das YOLO_V1-Paper ist öffentlich zugänglich und kann als nützliche Referenz für das Problem der Objektlokalisierung dienen. Es bietet wertvolle Informationen zur Implementierung und Optimierung von YOLO-Netzwerken.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \nVdash_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \tag{1}$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \nVdash_{ij}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \tag{2}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \nVdash_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \nVdash_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \tag{3}$$

$$+ \sum_{i=0}^{S^2} \nVdash_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \tag{4}$$

Abbildung 17 Koordinaten Loss Funktion YOLO V1 (20)

Die in Abbildung 17 dargestellte Formel zeigt die Loss Funktion des YOLO_V1-Algorithmus. Für die Objektlokalisierung sind hauptsächlich die Formelteile (1) und (2) relevant. Die Teile (3) und (4) beziehen sich auf die Klassifikation und die Sicherheitsvorhersage (Confidence) und sind für die Lokalisierung nicht von Bedeutung.

Der erste Teil der Formel berechnet den Loss-Wert für die Vorhersage der Startpunkte der Bounding Boxes. YOLO legt ein Gitter über das Bild und sagt für jedes Gitterelement separate Bounding Boxes voraus. Die Operation wird über S^2 durchgeführt, wobei S die Größe des Gitters ist. YOLO kann pro Gitterbox mehrere Bounding Boxes vorhersagen, wobei im YOLO V1 die Anzahl auf 2 begrenzt ist (siehe X.X. You Look Only

Once für mehr Informationen). Die Berechnung dieses Teils der Formel entspricht dem MSE aus Abbildung 16.

Der zweite Teil der Formel ermittelt den Loss-Wert für die Breite und Höhe der Bounding Box. Auch hier wird der MSE verwendet, um die Abweichung der vorhergesagten Werte von den korrekten Werten zu berechnen. Basierend auf dieser Formel wird der MSE sowohl für die Startposition als auch für die Größe der Bounding Box getrennt berechnet, und diese beiden Loss-Werte werden am Ende addiert.

Auf Basis dieser Analyse zeigt sich, dass die Loss-Funktion innerhalb von YOLO nicht stark von der grundlegenden MSE Formel abweicht. Zudem zeigt sich, dass für die Sortierung der Objekte nicht die Loss-Funktion zuständig ist, sondern die Gitterstruktur bei der Bildanalyse.

Im bisherigen Training wurden alle Modelle mit einer eindimensionalen Tabelle trainiert, bei der keine spezifische Strukturierung der Objekte innerhalb der Tabelle berücksichtigt wurde. Versuche, die Objekte zufällig anzuordnen, führten nicht zu einer Verbesserung der Trainingsergebnisse. Durch die Verwendung eines Gitters kann eine logische Regel für die Anordnung der Objekte innerhalb der Tabelle eingeführt werden, die vom Modell erlernt und verstanden werden kann. Im bisherigen Training wurden alle Modelle mit einer eindimensionalen Tabelle trainiert, bei der keine spezifische Strukturierung der Objekte innerhalb der Tabelle berücksichtigt wurde. Versuche, die Objekte zufällig anzuordnen, führten nicht zu einer Verbesserung der Trainingsergebnisse.

Durch die Verwendung eines Gitters bei der Analyse der Bilder kann eine Regel zur Position der Objekte geschaffen werden, welche das Modell erlernen kann. Bei dieser Regel wird zuerst der Mittelpunkt eines Objektes berechnet. Im Anschluss daran wird bestimmt, in welcher Zelle der Gitterstruktur dieser Mittelpunkt liegt. In die ermittelte Zelle werden die Informationen zu dem erkannten Objekt abgelegt. Ein Beispiel hierfür wurde anhand des Hundes in der Abbildung 18 gegeben. Hierbei ergab die Mittelpunktberechnung, dass die Informationen in die Zelle (2;5) gespeichert werden.

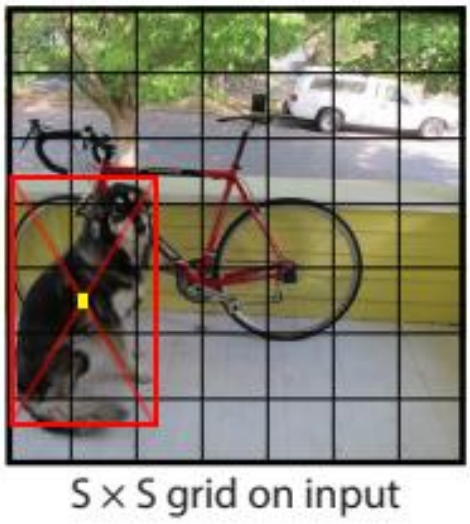

Abbildung 18 Gitter- struktur bei YOLO V1(20)

Die Untersuchung hat ergeben, dass die Entwicklung einer eigenen Loss-Funktion nicht erforderlich ist. Daher liegt der Fokus nun auf der Implementierung einer Gitterstruktur für die Analyse der Bilder. Die Verwendung eines Gitters ermöglicht eine gezielte Anordnung und Strukturierung der Objekte innerhalb der Tabelle, was dem Modell helfen kann, räumliche Beziehungen zwischen den Objekten zu verstehen und zu lernen.

(19) (20)

5.2.4. Objektüberfluss

Eine weitere Herausforderung bei der Multiobjekterkennung besteht darin, dass die Anzahl der maximal zu erkennenden Objekte durch den Entwickler beim Training des Modells bestimmt und begrenzt wird. Im Gegensatz zum Sliding Window-Ansatz, bei dem beliebig viele Objekte erkannt werden können, ist die Anzahl der erkannten Objekte bei diesem Ansatz begrenzt. Um jedoch das Ziel zu erreichen, mehrere hundert Objekte pro Bild erkennen zu können, müssen Lösungen für diese Limitierung gefunden werden.

Eine Möglichkeit besteht darin, die Anzahl der maximal möglichen auszugebenden Objekte direkt zu erhöhen, indem die Größe der Ausgabetabelle entsprechend angepasst wird. Dies kann relativ einfach umgesetzt werden. Allerdings gibt es physikalische Grenzen, ab der eine zu große Ausgabetabelle die Performance des Modells beeinträchtigt. Zudem löst dies das Problem der begrenzten Ausgabemenge nicht vollständig, sondern verschiebt es nur. Selbst bei einer sehr großen Ausgabemenge bleibt diese endlich, während theoretisch auf einem unendlich großen Bild unendlich viele Objekte vorhanden sein könnten.

Ein anderer Lösungsansatz für dieses Problem basiert auf der Technik des Sliding Windows. Hierbei wird dem Modell die Möglichkeit gegeben, zusätzlich zu den vier Koordinatendaten pro Objekt einen binären Wert auszugeben welcher anzeigt, ob in der analysierten Zelle mehr als ein Objekt enthalten war. Dadurch kann dieser Bereich gezielt ausgewählt und erneut durch das Modell analysiert werden. Da dieser dabei auf die ursprüngliche Bildgröße vergrößert wird, kann das Modell wieder mehrere Vorhersagen treffen. Dieser Prozess kann, falls erforderlich mehrfach pro Bildsegment wiederholt werden. Da die Information über das Vorhandensein von mehr Objekten je Bildsegment getrennt ausgegeben wird müssen auch nur diese Segmente genauer betrachtet werden, wodurch der zusätzliche Analyseaufwand geringer ist, als bei der Sliding Window Technik.

Durch diese Erweiterung des Modells können mehr Objekte pro Bild erkannt werden, wodurch die Limitierung der maximalen Anzahl von Objekten überwunden wird.

6. Objekt Erkennung

Um eine Objekterkennung zu entwickeln, gibt es zwei Möglichkeiten, die Objektlokalisierung und die Objektklassifizierung zu kombinieren. Die einfachere Umsetzung besteht darin, die Objekte mithilfe der Objektlokalisierung zu lokalisieren und anschließend die Ausschnitte der lokalisierten Objekte dem Klassifizierungsalgorithmus zur Klassifizierung zu übergeben. Bei dieser Struktur wird jedes einzelne Objekt separat durch den Klassifizierungsalgorithmus geschickt, was zu einer erhöhten Berechnungszeit führt, insbesondere wenn das Bild mehrere Objekte enthält.

Die alternative Umsetzung besteht darin, ein gemeinsames Modell für die Objektlokalisierung und Klassifizierung zu trainieren. Bei diesem Ansatz kann die Sliding Window-Technik nicht angewendet werden und wird daher in dieser Betrachtung nicht weiter berücksichtigt. Ein Vorteil dieser Verbindung besteht darin, dass das Modell durch die Objektklassifizierung auch Informationen darüber liefern kann, wie sicher es bei der Lokalisierung der Objekte ist. Basierend auf den Entscheidungen aus dem vorherigen Kapitel (siehe 5.2.2. Grundlage der Multiobjekterkennung) wird eine dreidimensionale Matrix mit der Form X*X*Y verwendet.

Die Objekterkennungsmatrix besteht aus X^2 Segmenten, wobei jedes Segment 9 Zellen enthält. Abbildung 19 zeigt die Struktur eines Objekterkennungssegments.

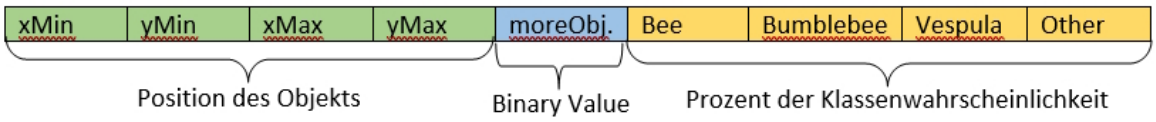

Abbildung 19 Struktur Objekterkennungssegment

Die ersten 4 Zellen jedes Segments werden zur Lokalisierung des Objekts auf dem Bild verwendet. Zusätzlich gibt es eine Zelle, die angibt, ob in diesem Fenster mehr als ein Objekt vorhanden ist (moreObj.).

Die restlichen Zellen am Ende jedes Segments dienen der Zuordnung des Objekts zu den verschiedenen Klassen. Die Anzahl dieser Zellen (N) hängt von der Anzahl der möglichen zu erkennenden Klassen ab. Für diese Studienarbeit sind dies Zellen. Soll zu einem späteren Zeitpunkt weitere Klassen erkannt werden muss nur die Anzahl (N) erhöht werden auf die gewünschte Klassenzahl.

Aufgrund dieser Struktur kann die Matrix X^2 Objekte erkennen. Für den ersten Testlauf wurde X auf 9 festgelegt, was bedeutet, dass maximal 81 Objekte pro Bild erkannt werden können. Die Objekterkennungsmatrix ist in Abbildung 20 dargestellt.

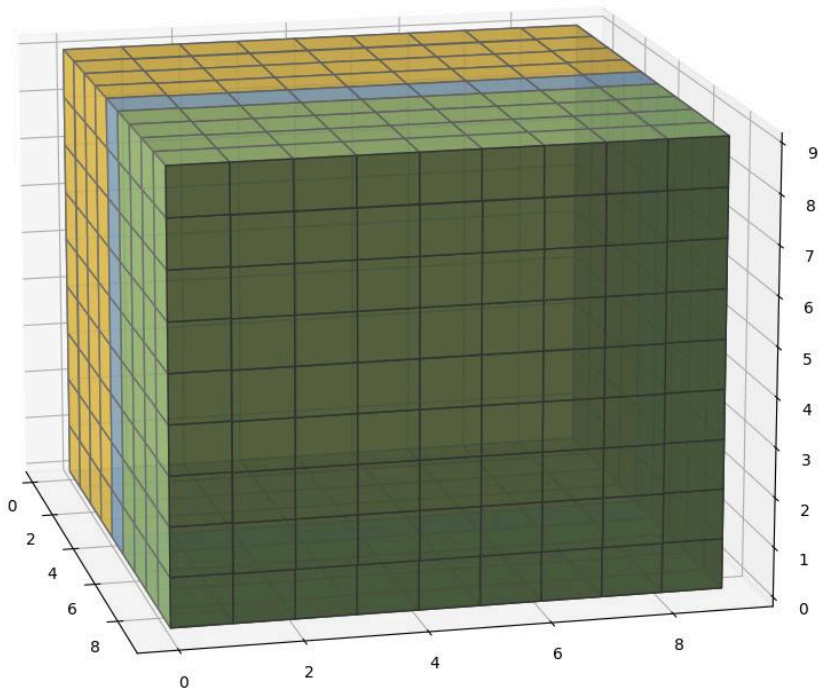

Abbildung 20 Objekterkennungsmatrix

7. Zusammenfassung und Ausblick

Im Verlauf der Studienarbeit wurden verschiedene Ansätze und Netzwerke zur Erkennung von Objekten in Bildern entwickelt. Dabei wurden Konzepte basierend auf den definierten Zielen entwickelt.

Zunächst wurden geeignete Datenquellen ausgewählt, um Trainingsdaten für die Modelle zu erhalten. Anschließend wurden die Datenquellen annotiert, wobei die Filterung des Datensatzes zur Entfernung irrelevanter oder fehlerhafter Daten sowie die Auswahl eines geeigneten Annotierungssystems berücksichtigt wurden. Diese bilden die Grundlage für das erfolgreiche Trainieren der CNN.

Des Weiteren wurde eine Modellanalyse durchgeführt. Dabei wurde auf Basis einer wissenschaftlichen Untersuchung Modelle ausgewählt und für den Einsatzzweck dieser Arbeit analysiert. Dabei wurden auch bestimmte CNN-Architekturen ausgewählt und detailliert untersucht.

Auf Basis des ausgewählten Modelles wurden zwei Ansätze zur Lokalsierung von Objekten untersucht. Das Sliding Window, bei dem ein festes Fenster über das Bild geschoben wird, um potenzielle Objekte zu finden, sowie die Lokalisierung mittels eines neuronalen Netzwerks. Bei der Entwicklung der Lokalisierung mittels eines Neuronalen Netzwerks wurden Konzepte zum Training auf Koordinatendaten entwickelt. Im Rahmen der Untersuchung zeigte sich, dass die Lokalsierung mittels neuronalem Netzwerk die bessere Wahl für diese Arbeit ist.

Abschließend wurden die Erkenntnisse aus der Objektlokalisierung genutzt,um ein Modell zur Objekterkennng zu trainieren. Dieses vereinigt die Fähigkeiten der Klassifikation mit denen der Objektlokalisierung und findet dabei Lösungen für die Erkennung von großen Objektmengen innerhalb eines Bildes. Dadurch konnte ein leistungsfähiges Netzwerk zur Erkennung von Objekten entwickelt werden.

Das Ziel der Studienarbeit wurde erreicht. Durch die Konvertierung in ein Tensorflow Lite Objekt kann es umgehend innerhalb von Mikrocontrollern und mobilen Endgeräten eingesetzt werden. Durch die Generalisierung der Objekterkennung kann das Modell durch Austauschen des verwendeten Datensatzes auch auf die Erkennung anderer Objekte trainiert werden. Ein weiterer Vorteil besteht darin, dass die Größe der Erkennungsmatrix schnell erweitert werden kann, um verschiedenen Anforderungen gerecht zu werden.

8. Quellenverzeichnis

(1) IBM (): Was ist ein neuronales Netz?, https://www.ibm.com/de-de/topics/neural-networks [abgerufen am 12.07.2023 08:28]

(2) AWS (): Was ist ein neuronales Netzwerk?, https://aws.amazon.com/de/what-is/neural-network/ [abgerufen am 12.07.2023 08:29]

(3) Dake, Mysid (): Vereinfachte Darstellung eines künstlichen neuronalen Netzes, https://upload.wikimedia.org/wikipedia/commons/thumb/3/3d/Neural_network.svg/1920px-Neural_network.svg.png [abgerufen am 12.07.2023 08:31]

(4) Chrislb (): Schema eines künstlichen Neurons: Künstliches Neuron mit Index j, https://upload.wikimedia.org/wikipedia/commons/7/7f/ArtificialNeuronModel_deutsch.png [abgerufen am 12.07.2023 08:33]

(5) Ralf Hollstein (2023): Neuronale Netze in Optimierungsmethoden, Springer Fachmedien Wiesbaden 1. Auflage Springer Fachmedien Wiesbaden [abgerufen am 12.07.2023 08:36]

(6) Amin Zollanvari (): Convolutional Neural Networks in Machine Learning with Python, Springer Nature Switzerland AG 1. Auflage Springer Nature Switzerland AG [abgerufen am 12.07.2023 08:42]

(7) IBM (): Convolutional Neural Networks, https://www.ibm.com/topics/convolutional-neural-networks [abgerufen am 12.07.2023 08:44]

(8) Tensorflow ():

Why TensorFlow, https://www.tensorflow.org/about [abgerufen am 12.07.2023 08:45]

(9) Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (10.12.2015): Deep Residual Learning for Image Recognition, https://arxiv.org/pdf/1512.03385.pdf Microsoft Research [abgerufen am 12.07.2023 08:47]

(10) Data Base Camp (18.03.2023): ResNet: Residual Neural Networks – einfach erklärt!, https://databasecamp.de/ki/resnet#:~:text=Residual%20Neural%20Networks%20(kurz%3A%20ResNet,noch%20geringe%20Fehlerraten%20hervorrufen%20kann. [abgerufen am 12.07.2023 08:48]

(11) Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger (28.01.2018): Densely Connected Convolutional Networks, https://arxiv.org/pdf/1608.06993.pdf [abgerufen am 12.07.2023 08:50]

(12) Sik-Ho Tsang (25.11.2018): Review: DenseNet — Dense Convolutional Network (Image Classification), https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803 [abgerufen am 12.07.2023 08:51]

(13) NVIDIA (): SE-RESNEXT101, https://pytorch.org/hub/nvidia_deeplearninge-xamples_se-resnext/ [abgerufen am 12.07.2023 08:53]

(14) Jie Hu, Li Shen, Samuel Albanie, Gang Sun, Enhua Wu (16.05.2019): Squeeze-and-Excitation Networks, https://arxiv.org/pdf/1709.01507v4.pdf [abgerufen am 12.07.2023 08:53]

(15) Keras, Tensorflow (): InceptionResNetV2, https://keras.io/api/applications/inceptionresnetv2/ [abgerufen am 12.07.2023 08:55]

(16) Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi (23.06.2016): Inception-v4, Inception-ResNet and

the Impact of Residual Connections on Learning, https://arxiv.org/pdf/1602.07261.pdf [abgerufen am 12.07.2023 08:56]

(17) Sik-Ho Tsang (18.05.2019): Review: NASNet — Neural Architecture Search Network (Image Classification), https://sh-tsang.medium.com/review-nasnet-neural-architecture-search-network-image-classification-23139ea0425d [abgerufen am 12.07.2023 08:57]

(18) Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le (11.04.2018): Learning Transferable Architectures for Scalable Image Recognition, https://arxiv.org/pdf/1707.07012.pdf [abgerufen am 12.07.2023 08:58]

(19) Ching (Chingis) (06.07.2021): Object Detection Explained: YOLO v1., https://medium.com/mlearning-ai/object-detection-explained-yolo-v1-fb4bcd3d87a1 [abgerufen am 12.07.2023 08:59]

(20) Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi (09.05.2016): You Only Look Once:

Unified, Real-Time Object Detection, https://arxiv.org/pdf/1506.02640.pdf [abgerufen am 12.07.2023 09:00]

(21) Liangqu Long, Xiangming Zeng (27.01.2022): Overfitting in Beginning Deep Learning with TensorFlow

Work with Keras, MNIST Data Sets, and Advanced Neural Networks, Apress Berkeley, CA Auflage. 1 Apress Berkeley, CA [abgerufen am 12.07.2023 09:02]

(22) Pádraig Cunningham, Sarah Jane Delany (13.04.2021): Underestimation Bias and Underfitting in Machine Learning in Trustworthy AI - Integrating Learning, Optimiza-

tion and Reasoning, Springer International Publishing 1. Auflage Springer International Publishing [abgerufen am 12.07.2023 09:05]

(23) Chi-Feng Wang (08.01.2019): The Vanishing Gradient Problem, https://towards-datascience.com/the-vanishing-gradient-problem-69bf08b15484 [abgerufen am 12.07.2023 09:07]

(24) George Rey, Callum Robertson (2018): Datensatz: Bee or wasp?, https://www.kaggle.com/datasets/jerzydziewierz/bee-vs-wasp [abgerufen am 12.07.2023 09:20]

(25) labelImg (3.12.2018): , https://raw.githubusercontent.com/tzutalin/labelImg/master/demo/demo3.jpg [abgerufen am 12.07.2023 09:23]

(26) Simone Bianco, Remi Cadene, Luigi Celona, Paolo Napoletano (19.10.2018): Benchmark Analysis of Representative

Deep Neural Network Architectures, https://arxiv.org/pdf/1810.00736.pdf [abgerufen am 12.07.2023 09:25]

(27) Tanvir Bhuiyan, Ryan Carney, Sriram Chellappan (25.08.2022): Bee and non-bee dataset, https://data.mendeley.com/datasets/jykt6862s4/3 [abgerufen am 13.07.2023 09:06]

(28) Fabian Hickert (25.05.2022): bee_dataset, https://www.tensorflow.org/datasets/catalog/bee_dataset#bee_datasetbee_dataset_300_default_config [abgerufen am 13.07.2023 09:06]

5.

A. Custom Lokalisierungsmodells V1

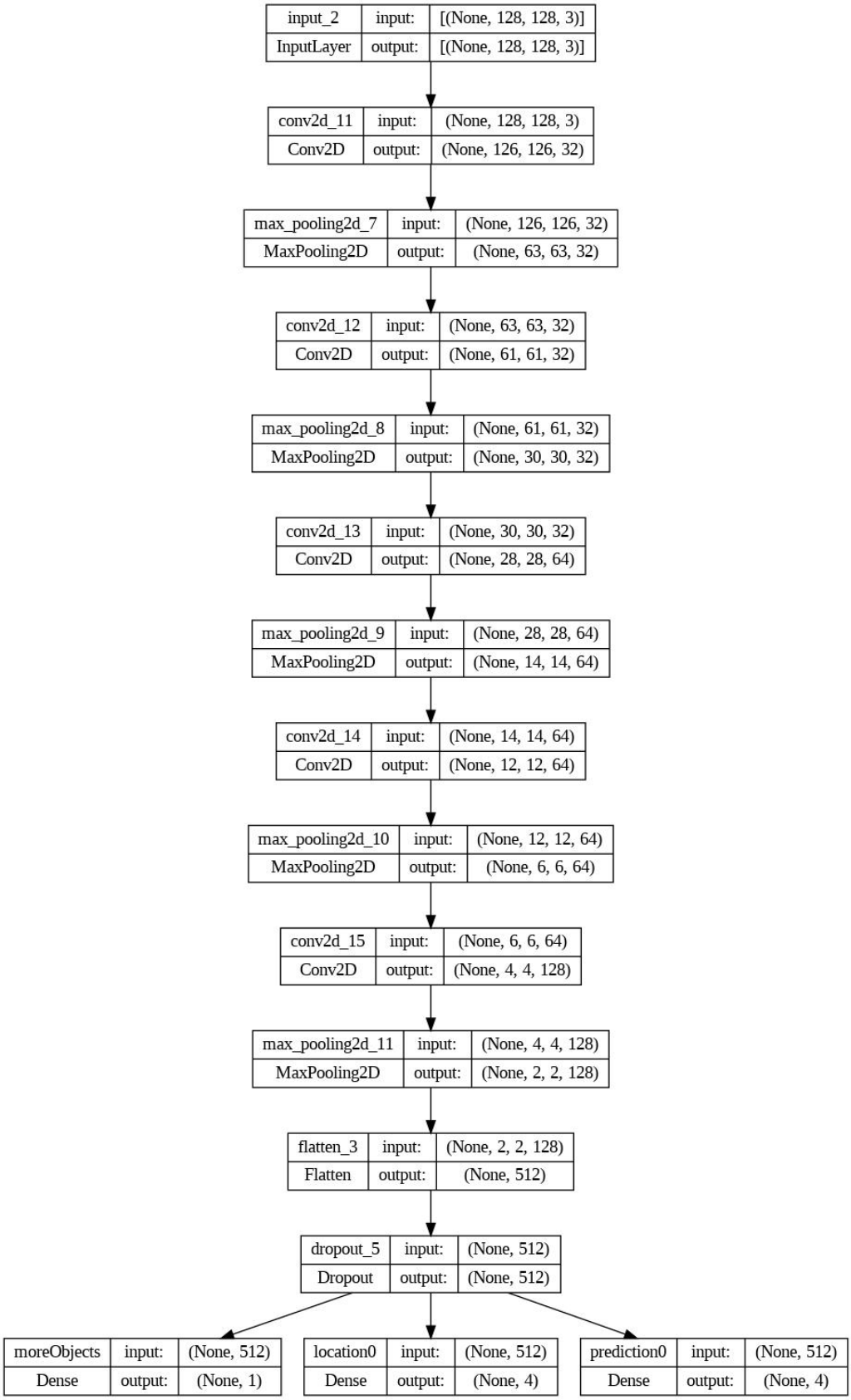

Abbildung 21 Struktur des Custom Lokalisierungsmodells V1

B.Custom Lokalisierungsmodells V2

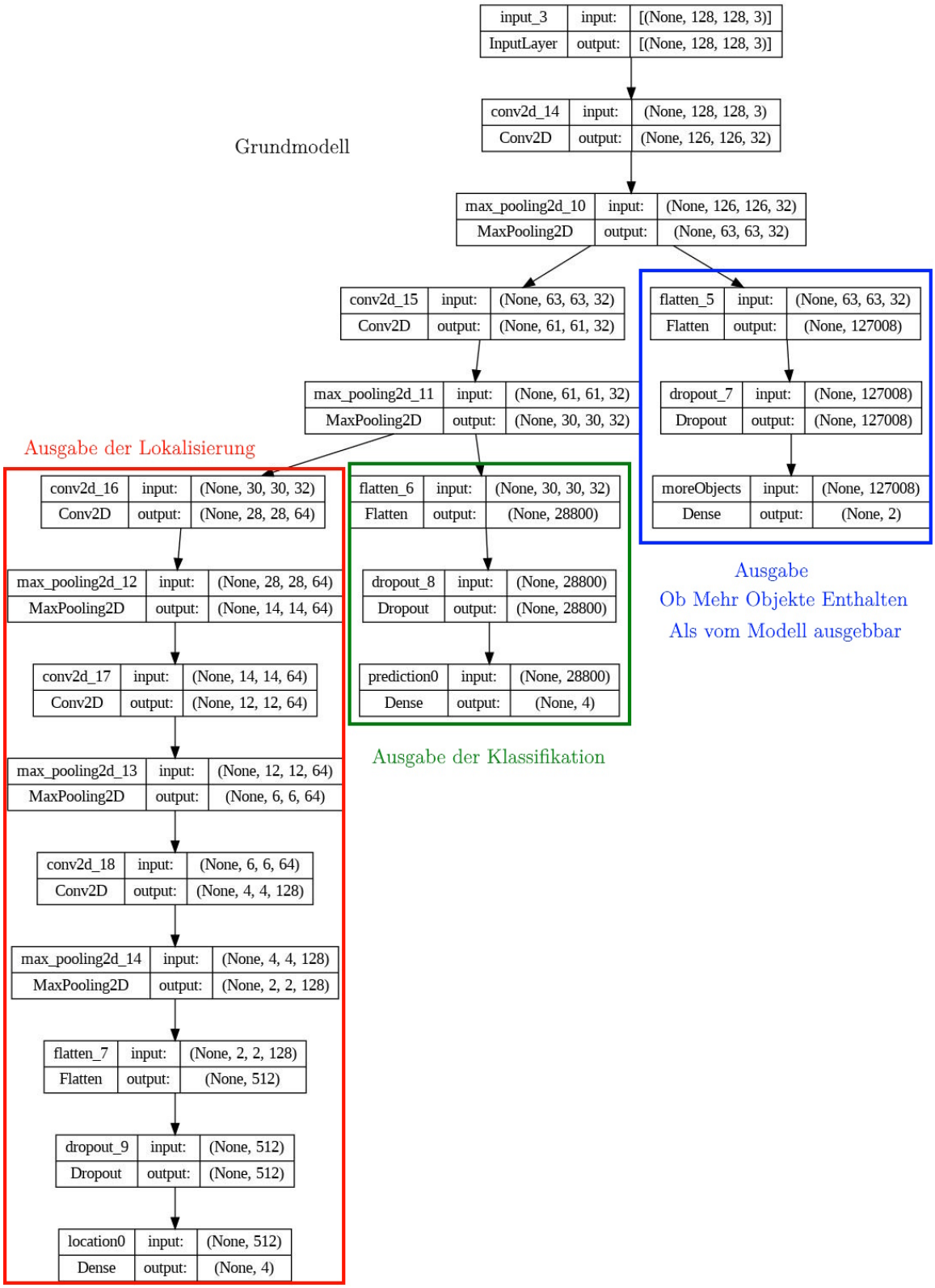

Abbildung 22 Struktur des Custom Lokalisierungsmodells V2

C.Custom Erkennungsmodell

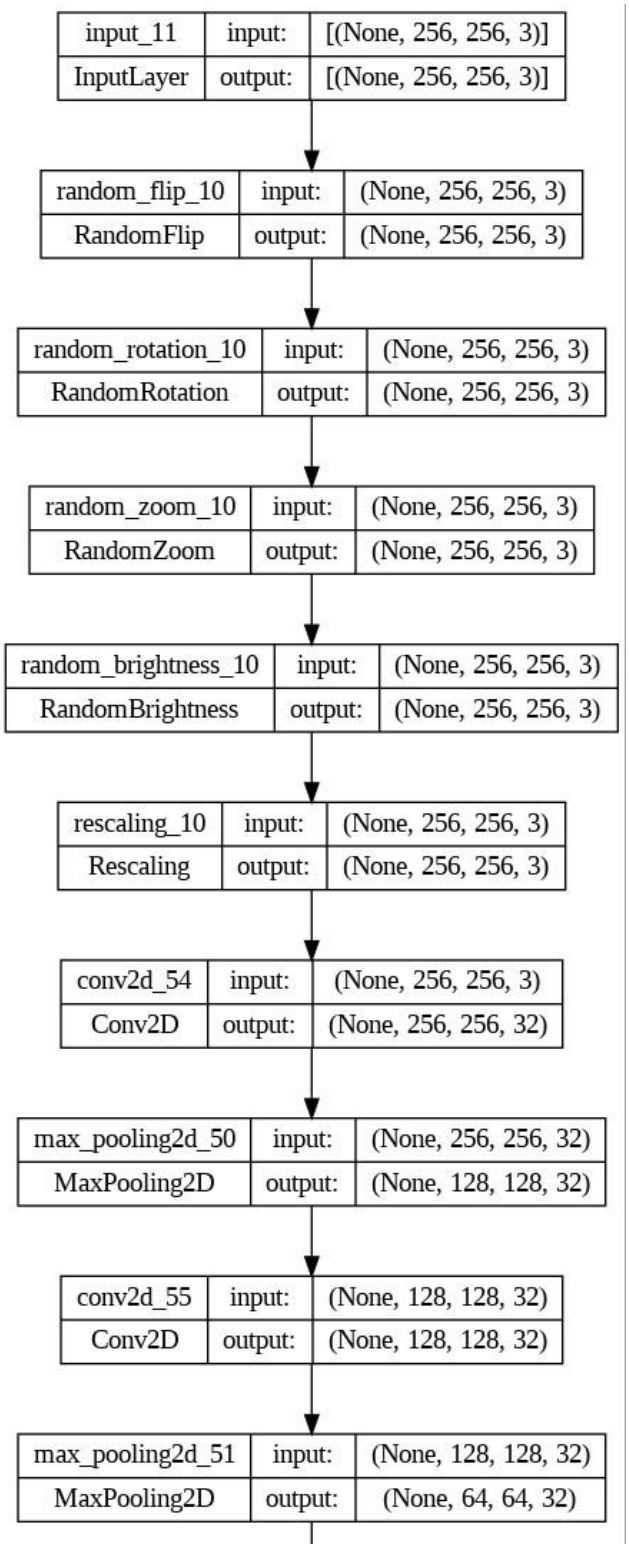

Abbildung 23 Custom Erkennungsmodell Teil 1

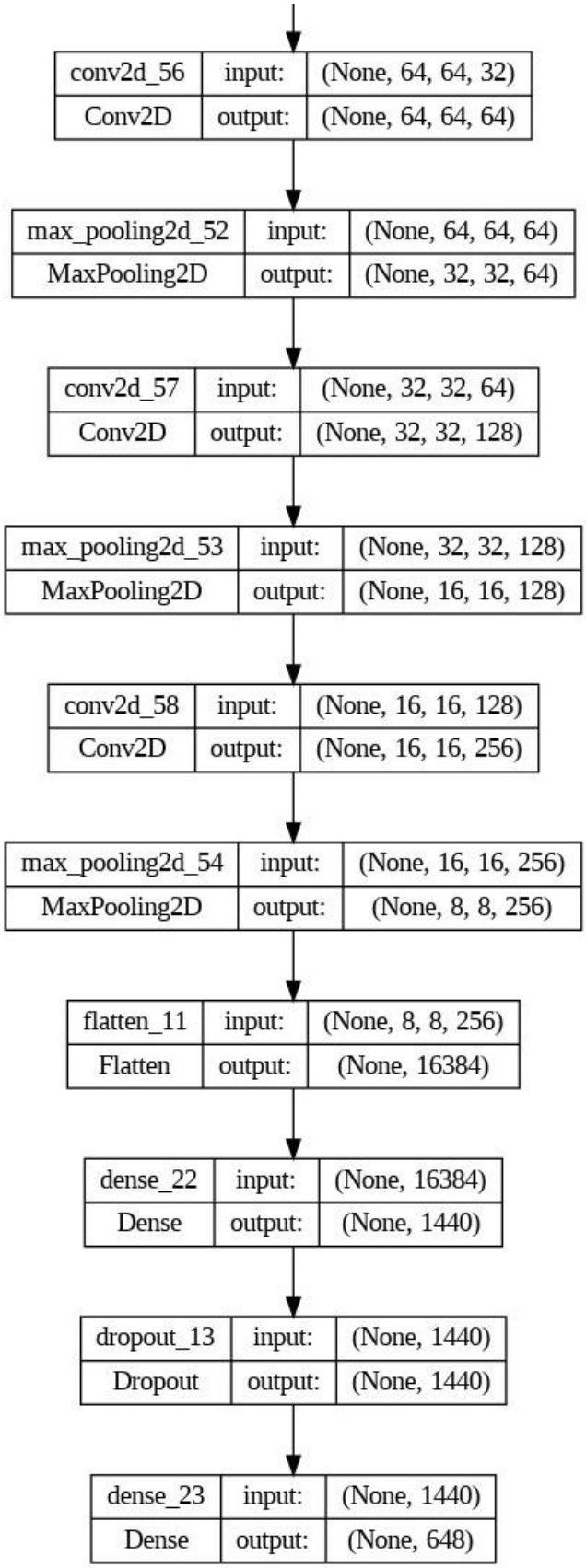

Abbildung 24 Custom Erkennungsmodell
Teil 2