HAW HAMBURG

BACHELOR THESIS
Luise Kempa

Automatisierte Objekterkennung und -zählung von Vögeln in großen Schwärmen mit Hilfe von Maschinellem Lernen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Luise Kempa

Automatisierte Objekterkennung und -zählung von Vögeln in großen Schwärmen mit Hilfe von Maschinellem Lernen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Clemen
Zweitgutachter: Prof. Dr. Peer Stelldinger

Eingereicht am: 13.11.2023

Luise Kempa

Thema der Arbeit

Automatisierte Objekterkennung und -zählung von Vögeln in großen Schwärmen mit Hilfe von Maschinellem Lernen

Stichworte

Vogelschwärme, Objekterkennung, Objektzählung, maschinelles Lernen

Kurzzusammenfassung

Die Arbeit befasst sich der Vogelzählung in Schwärmen anhand von Bildern mit Hilfe von maschinellem Lernen. Es wird eine Anwendung konzipiert und implementiert, welche die Anzahl der Vögel auf einem übergebenen Bild herausfindet. Für die Implementierung werden verschiedene bereits bestehende neuronale Netze verglichen und anschließend erweitert. Anhand verschiedener Experimente wird eine Aussage über die Auswahl eines neuronalen Netzes für den Einsatz in der Anwendung der Vogelzählung getroffen.

Luise Kempa

Title of Thesis

Automated object detection and counting of birds in large flocks using machine learning

Keywords

Bird flocks, object detection, object counting, machine learning

Abstract

The work adresses the implementation of bird counting in flocks based on images using machine learning. An application is designed and implemented that determines the number of birds in an image. For the implementation, different existing neural networks are compared and then extended. On the basis of different experiments a statement is made about the selection of a neural network for the use in the application of bird counting.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

1.1 Motivation

Für die Gesundheit des Menschen und die Erhaltung seines natürlichen Lebensraumes spielt der Vogel eine besondere Rolle, indem er wichtige Lebensgrundlagen bereitstellt. Darunter fallen zum Beispiel die Bestäubung, Schädlingsbekämpfung, Nährstoffumwälzung sowie andere Faktoren, die das Leben des Menschen beeinflussen. In dem stetigen Wandel der heutigen Welt zeigt sich, dass auch Vögel auf die Veränderungen der Natur und den Klimawandel reagieren und dass laut Studien der Artenbestand immer weiter zurückgeht. Um dieser Veränderung entgegenzuwirken, werden verbesserte Methoden zur Aufzeichnung der Artenbestände, sowie der Habitate des Vogels benötigt. Durch einen ausgiebigen Einblick in die Vogelwelt können Maßnahmen zur Rettung von Vogelarten gezielter ergriffen werden [4, 2].

Diese Arbeit widmet sich dem Artenschutz von Vögeln, indem mit Hilfe von Bilderkennung jeder Vogel in einem großen Vogelschwarm erkannt und identifiziert werden kann. Durch die anschließende Zählung der Vögel kann ein besseres Verständnis der Vogelverteilung erzielt werden, indem eine Übersicht zu der Vogelanzahl auf regionaler Ebene gegeben wird. Zudem kann die Vogelzählung die Datensammlung auf Plattformen, wie zum Beispiel eBird [19] unterstützen. Mit der Datenbank von eBird können Informationen zur Vogelbeobachtung weltweit festgehalten und für die Forschung der Schutzmaßnahmen von Vögeln verwendet werden.

1.2 Abgrenzung der Arbeit

Für die Datensammlung der Artenbestände von Vögeln wird eine Anwendung für die Identifizierung, Zählung und die Bestimmung der Art des Vogels benötigt. Die nötige Klassifizierung der Arten, welche nach der Identifizierung des Vogels erfolgt, wäre im

Rahmen dieser Arbeit zu umfangreich. Daher konzentriert sich diese Arbeit auf die Identifikation und Zählung der Vögel mit Hilfe von Bilderkennung.

1.3 Ziel der Arbeit

Die endgültige Umsetzung des Projektes soll die Identifikation von Vögeln und deren Anzahl ermöglichen. Dabei soll aus der Arbeit hervorgehen, welche Methoden zur Objekterkennung und welche Modelle und Datensätze hierfür am geeignetsten sind. Gefragt ist eine Anwendung, die eine Funktion bereitstellt, mit der ein Benutzer ein Bild von einem beliebigen Vogelschwarm hochladen kann. Anschließend soll der Benutzer durch das System über die genaue Anzahl der vorhandenen Vögel auf dem hochgeladenen Bild informiert werden. Dabei kann der Einsatz von maschinellem Lernen die Möglichkeit geben, durch bereits vorhandene Datensätze das Identifizieren von Vögeln und ihrer Art zu erlernen und das Erlernte auf neue, von dem Benutzer hochgeladene Bilder anzuwenden. Diese Methodik ermöglicht dem Anwender, statt durch manuelles Zählen mit einem geringeren Aufwand genauere Ergebnisse über die Vogelanzahl zu erhalten.

Der Fokus dieser Arbeit liegt vor allem auf der Zählung von Vögeln zur Brutzeit an der Küste, weshalb sich die Anwendung hauptsächlich auf die Quantifizierung von Vögeln konzentriert, die sich auf dem Boden befinden.

1.4 Aufbau der Arbeit

Nachdem eine detaillierte Einführung in die Thematik und mathematische Grundlagen (2) erklärt worden sind, wird der aktuelle Stand der Forschung und Entwicklung (3) zur Objekterkennung von Vögeln vorgestellt. Anschließend wird eine umfassende Analyse der Anforderungen (4) an das System vorgenommen, welche in das Konzept der Modellumgebung und -beschreibung (5) eingebunden wird. Dabei sind Systemakteure und -aufbau sowie die Funktionalitäten des Modells beschrieben. Aus dem Konzept wird die tatsächliche Realisierung (6) und die daraus resultierenden Experimente (7) über die Auswahl eines neuronalen Netzes aufgestellt, welches im Ausblick (8) diskutiert wird.

2 Einführung in die Thematik

Dieses Kapitel gibt eine Einführung in die angewendeten mathematischen Grundlagen und Begrifflichkeiten, um ein vollständiges Verständnis für die Problemstellung und ihre Bewältigung in dieser Arbeit zu erlangen.

2.1 Objekterkennung

Die Objekterkennung ist eine computergestützte Technologie, die sich mit der Erkennung von Objekten einer spezifischen Klasse in digitalen Bildern und Videos befasst. Dabei nutzt die Objekterkennung Methoden um beispielsweise Menschen, Autos oder Gebäude automatisch zu identifizieren [11, 1].

2.2 Maschinelles Lernen

Bei maschinellem Lernen handelt es sich um die Fähigkeit eines Systems, eigenständig Regeln und Muster von gegebenen Daten zu erkennen. Die daraus neu erlernten Verknüpfungen können auf neue unbekannte Daten angewendet werden [16, 4].

Der Lernprozess kann dabei in zwei verschiedenen Vorgängen erfolgen: das überwachte Lernen und das unüberwachte Lernen [16, 11]. Für das unüberwachte Lernen ist der Ausgabewert des Eingabewertes nicht bekannt oder wird nicht verwendet. Das Lernverfahren wird dafür genutzt, um aus ähnlichen Daten nutzbare Gruppierungen zu bilden. Überwachtes Lernen beschreibt den Vorgang, in welchem jedem Eingabewert ein erwartetes Ergebnis für seinen Ausgabewert zugewiesen wird. Ein Lernverfahren des überwachten Lernens ist die Klassifikation. Dabei können alle Eingabewerte einer bestimmten Klasse zugeordnet werden[16, 11]. Aus dem Lernvorgang wird ein maschinelles Modell gebildet [16, 99].

2.3 Künstliches neuronales Netz

Das künstliche neuronale Netze gilt vermutlich als das am weitesten verbreitete Verfahren im Bereich des maschinellen Lernens [16, 114]. Ein künstliches neuronales Netz leitet sein Vorgehen nach dem „Prinzip biologischer Nervenzellen (Neuronen)“ [16, 114] ab, bei dem das Netz über Neuronen verfügt, welche Impulse von benachbarten Neuronen erhalten und diese wiederum weitergeben. Aus mathematischer Sicht kann ein neuronales Netz als eine vergleichsweise „komplexe Hypothesenfunktion“ [4, 6] angesehen werden, die in der Thematik dieser Arbeit Bilddaten auf die gewünschten Ausgaben wie Klassenbezeichnungen für die Klassifizierung abbildet. Dies geschieht durch Berechnungen in mehreren Schichten. Das Netz bekommt als Eingabewert eine Bilddatei, welche über die Eingabeschicht angenommen wird. Jeder Pixelwert, sprich Farbwert des Bildes, repräsentiert ein Neuron dieser Eingabeschicht. Informationen werden anschließend von einer Schicht des neuronalen Netzes zur nächsten weitergegeben. Bis schließlich die letzte Schicht erreicht wird, die dem Ausgabewert der definierten Funktion entspricht [4, 6].

Wie in dem Abschnitt 2.2 erläutert wurde, handelt es sich bei der Ausgabe eines maschinellen Lernvorganges um ein maschinelles Modell, weshalb im Kontext eines neuronalen Netzes im weiteren auch von einem Modell gesprochen werden kann.

2.3.1 Neuron

Ein Neuron eines künstlichen neuronalen Netzes erhält mindestens einen Eingang (x_1 bis x_n) und einen Ausgang A (zu sehen in Abbildung 2.1) [16, 114].

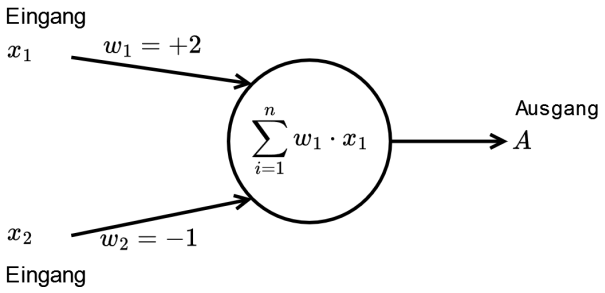


Abbildung 2.1: „Einzelnes künstliches Neuron (Perzeptron) mit Eingängen x_1 und x_2 und einem Ausgang A“ [16, 114]

Für jeden Eingang gibt es eine Gewichtung w, welche mit w1 bis wn gewichtet ist. Die Gewichtung eines Eingangs kann positiv wie oder negativ sein. Im Neuron selbt ergibt sich die Summe aller eingehenden Eingänge mit deren Gewichtungen [16, 114].

2.3.2 Schichten

Für komplexere Aufgaben kann ein Neuron durch mehrere Neuronen zu einem neuronalen Netzwerk vergrößert werden. Zur Veranschaulichung der Schichten ist die Abbildung 2.2 zusehen, welche aus drei Schichten besteht. Zu sehen sind die Ein- und Ausgabeschichten sowie die mittlere Schicht, welche verdeckt und nicht von außen einzusehen ist. Die Neuronen verschiedener Schichten sind über gewichtete Kanten miteinander vernetzt [16, 117-118].

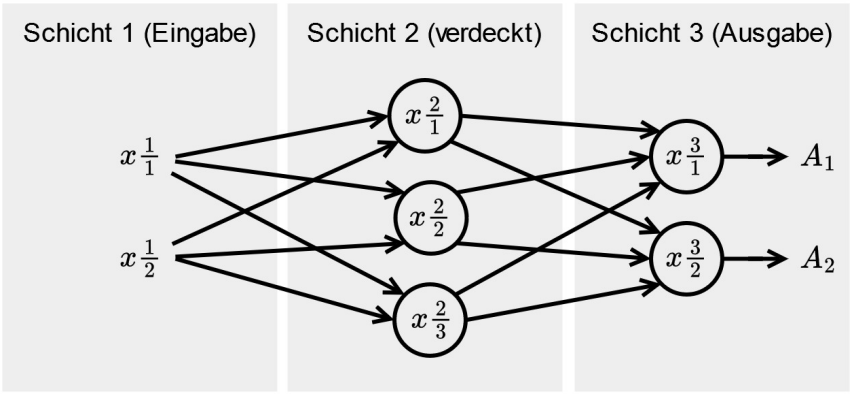


Abbildung 2.2: Netz mit 3 Schichten: Ein- und Ausgabeschicht und eine verdeckte Schicht [16, 117]

Die Tiefe des Modells (Anzahl der Schichten) und die Breite jeder Schicht (Anzahl der Neuronen je Schicht) ist je nach Anwendungsfall und dessen Anforderungen unterschiedlich. Die Breite der Eingabeschicht beschreibt die „Anzahl der Eingangseigenschaften“ [16, 118], die Breite der Ausgangsschicht die Anzahl der Klassen. Für die Klassifikation werden die Neuronen der Ausgabeschicht jeweils einer Klasse zugeordnet. In der Abbildung 2.2 sind die beiden Neuronen x_1^3 und x_2^3 vorhanden und somit zwei Klassen für die Klassifikation definiert [16, 118]. Eine größere Tiefe des Modells ermöglicht die Umsetzung schwieriger Anwendungsfälle, bedeutet jedoch auch eine längere Klassifikations- und Trainingszeit [16, 131].

2.4 Form eines neuronalen Netzes (Architektur)

Bei Abbildung 2.2 handelt es sich um eine einfache Struktur eines neuronalen Netzes, einem Feed-Forward-Netz. Hier sind die Verbindungen des Netzes nur von Eingabewerten in Richtung der Ausgabewerten gerichtet [16, 117]. Es gibt eine große Anzahl an verschiedener Architekturen von neuronalen Netzwerken, welche unterschiedliche Verbindungsformen sowie Schichten für verschiedene Anwendungsfälle bereitstellen [16, 128-129].

2.4.1 Faltungsnetzwerk (*CNN*)

Das Faltungsnetzwerk (im Englischen: Convolutional Neural Network (CNN)) ist eine Art eines neuronalen Netzwerkes, welches zur Verarbeitung von Bildern genutzt werden kann. Die drei Hauptmerkmale des Faltungsnetzwerkes sind die Faltungsschichten, die Pooling-Schichten und die vollständig verbundenen Schichten. Die Neuronen der Eingabeschicht entsprechen den Pixeln in den blauen, roten und grünen Kanälen des auswertenden Bildes. Die Ausgabeneuronen repräsentieren die gewünschten Informationen, wie beispielsweise die Position und Größe der Objekte [4, 6-7]

Faltungsschicht

Die Faltungschichten können aus einer Vielzahl von Schichten bestehen, wobei jede Schicht den Neuronenausgang (entsprechend einem Pixel) der vorherigen Schicht mit unterschiedlichen Faltungsfiltern verarbeitet. Jeder dieser Filter ist eine lineare Kombination der benachbarten Neuronen innerhalb eines festen Fensters und kann in verschiedenen Dimensionen bzw. „Filtergrößen“ [24, 3] vorkommen. Nach dieser linearen Kombination wird auf die Ergebnisse der Faltung eine nichtlineare Aktivierungsfunktion angewendet. In jeder Faltungsschicht werden separate Merkmalskarten erzeugt, die auf Informationen der vorherigen Schicht basieren. Die Merkmalskarte der letzten Schicht wird in der Regel als diejenige betrachtet, die relevante und bedeutungsvolle Informationen in Bezug auf das Eingabeobjekt enthält [4, 6-7].

Max-Pooling-Schichten

Die Max-Pooling-Schichten geben das Maximum der Werte der Eingangsneuronen in rechteckigen Gittern aus, gleich dem stärksten Merkmal, welches nicht überlappende

Bereiche darstellen. Die Informationen werden von der untersten Schicht, die einfache Merkmale repräsentieren, zu den abstrakteren Merkmalen in den oberen Ebenen weitergeleitet [4, 6-7].

Vollständig verbundene Schicht

Die vollständig verbundenen Schichten berechnen die Ausgabe jedes Neurons als nichtlineare Funktion. Diese Funktion wird auf eine lineare Kombination der Ausgaben aller Neuronen in der vorherigen Schicht angewendet. Diese Schicht sammelt Informationen von allen Neuronen der Max-Pooling-Schichten. Die Neuronen der Ausgabeschicht generieren dabei die gewünschten Ausgabewerte [4, 6-7].

2.4.2 *Resnet*-Architektur

Die Resnet-Architektur implementiert das residuale Lernen. Die Architektur basiert auf Residualblöcken, die es ermöglichen Faltungsschichten (2.4.1) zu überspringen, indem die Ausgabe einer Schicht des Modells mit einer späteren Schicht addiert wird. Somit können die Blöcke die Effizienz des Trainings steigern und weitgehend ein eventuelles Degradationsproblem lösen, bei dem das Netz mit zunehmender Tiefe eine schlechtere Leistung erbringt [30, 1304].

2.4.3 *Inception*-Architektur

Die Verwendung einer Inception-Architektur hat gezeigt, dass sehr gute Ergebnisse bei vergleichsweise geringem Rechenaufwand erzielt werden können [25, 1]. Dafür werden die so genannten „Inception Module“ [24, 3] eingesetzt. Die Module bestehen dabei aus verschiedenen Faltungsschichten mit unterschiedlichen Dimensionen (1x1, 3x3, 5x5) sowie weiteren Poolingschichten. Die Ausgabefilter der Schichten werden anschließend als ein Ausgabevektor zusammengefasst und an die nächste Schicht als Eingabewert weitergegeben. Um den daraus resultierenden Anstieg der Zahl der Ausgänge von Schicht zu Schicht zu vermeiden, werden die Schichten der Blöcke, wenn nötig, mit einer 1x1 Faltungsschicht in dem Modul kombiniert, um die Dimension zu reduzieren [24, 3-4].

2.5 Modellbildung

Für die Durchführung von Objekterkennung können Deep-Learning-Modelle verwendet werden. Besitzt ein maschinelles Modell eine umfangreiche Anzahl von verdeckten Schichten, wird es als Deep-Learning-Modell bezeichnet [16, 118]. Die Modelle verwenden Pixelwerte und Begrenzungsrahmen von Objekten als Input und erlernen automatisch die relevanten, verborgenen visuellen Merkmale, die für die spezifische Anwendung von Bedeutung sind [4, 3].

Damit ein Model in der Lage ist, einen Pixelwert des Bildes in eine Ausgabe umzuwandeln, muss es zunächst erstellt und trainiert werden. Für das Training und die Modellbildung werden Trainingsdaten (2.5.1) und Trainingsparameter (2.6) benötigt. Beides wird im Folgenden näher erläutert.

2.5.1 Datensätze

Deep-Learning-Modelle benötigen Trainingsdaten, um zu lernen, wie sie die Pixelwerte eines Bildes in die gewünschte Ausgabe umwandeln können. Während des Trainingsvorgangs werden Bilder mit Objekten und den dazugehörenden sogenannten *Bounding Boxes* als Eingabe eingesetzt. *Bounding Box* bezeichnet eine viereckige Umfassung eines Objektes in einem auszuwertenden Bild. Das Ziel ist, ein tiefes neuronales Netzwerk zu erstellen und die Überschneidung der Ausgaben des Netzes und den entsprechenden *Bounding Boxes* zu maximieren [4, 3-4].

2.6 Trainingsparameter

Für den Trainingsvorgang der Modellbildung müssen Trainingsparameter durch die Bestimmungen bestimmter Funktionen und Werte des Modells festgelegt werden. Wie in dem Abschnitt 2.3.1 erläutert, besitzt ein Modell Parameter, wie die Gewichtungen und Schwellenwerte jedes Neurons. Diese müssen während des Trainings optimiert werden. Für das Training eines künstlichen neuronalen Netzes muss die Festlegung der sechs verschiedenen Hyperparameter erfolgen [16, 130-131]:

- Die Form des künstlichen neuronalen Netzes (2.4)
- Die Tiefe des Netzes

- Die Breite der Schichten
- Die Aktivierungsfunktion (2.6.1)
- Die Lernrate (2.6.4)
- Die Losgröße (2.6.5)

Die Form und die Tiefe des Netzes sowie die Breite der Schicht wurden bereits in dem Unterabschnitt 2.3.2 vorgestellt. Die Aktivierungsfunktion, Lernrate und Losgröße werden in den folgenden Unterabschnitten vorgestellt. Außerdem wird die Verlustfunktion eingeführt, welche Einfluss auf den Trainingvorgang hat.

2.6.1 Aktivierungsfunktion

Die Aktivierungsfunktion optimiert die Gewichtung w sowie den Schwellenwert b. Die Aktivierung eines Neurons erfolgt durch die Anwendung einer Aktivierungsfunktion auf die gewichtete Summe seiner Eingangswerte. Damit ein Neuron aktiviert werden kann, muss ein bestimmter Schwellenwert b überschritten werden, welcher ebenfalls positiv oder negativ ausfallen kann. Durch die Aktivierungsfunktion kann eine stetige Optimierung der Modellparameter erfolgen. Wenn zum Beispiel die Gewichtung w vergrößert wird, kommt ein entsprechend anderer Ausgabewert A heraus. Befindet sich A näher an dem korrekten Wert der Trainingsdaten als vor der Änderung von w, wird w weiter erhöht. Ansonsten ist es sinnvoll, w zu verkleinern. Je nach Aktivierungsfunktion werden die Neuronen ab einem bestimmten Wert aktiviert [16, 122-123].

Sigmoid-Funktion

Da durch die beiden Ausgabewerte 0 und 1 für ein Neuron nicht ganz schlüssig definiert ist, in welche Richtung die Gewichtung und der Schwellenwert geänderte werden müsste, gibt es bereits weitere Funktionen, die dabei helfen diese Problematik zu umgehen. Eine bekannte Aktivierungsfunktion ist die Sigmoid-Funktion. Die Sigmoid-Funktion bewegt sich in einem Wertebereich von 0 bis 1 und ist somit gut für eine binäre Klassifikation von zwei Klassen einzusetzen. Auf Grund des Wertebereiches kann die Funktion die Rückpropagierung (2.6.2) verwenden, was wiederum die Richtung des zu ändernden Wertes der

Modellparameter anzeigt. Die Funktion (2.1) stellt sich wie folgt zusammen [16, 122]:

$$A(z) = sig(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z} \tag{2.1}$$

Der Wert z ergibt sich durch das Ergebnis der Multiplikation der eingehenden Gewichtungen w und Eingänge x subtrahiert dem Schwellenwert b (2.2) [16, 122].

$$z = \sum_i w_i \cdot x_i - b \tag{2.2}$$

Die Sigmoid-Funktion zeigt für negative Werte von z eine Annäherung an den Ausgabewert 0 und für positive Werte von z eine Annäherung an den Ausgabewert 1 an. Innerhalb dieses Bereichs zeigt sie einen charakteristischen S-förmigen Verlauf (siehe Abbildung 2.3) [16, 122].

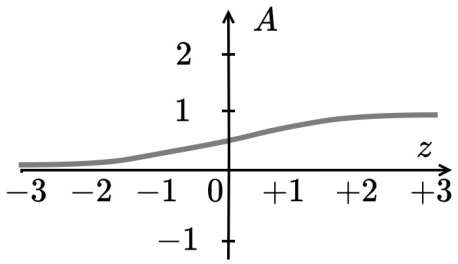


Abbildung 2.3: Sigmoid-Funktion [16, 122]

2.6.2 Rückpropagierung

Damit der Optimierungsvorgang der Modellparameter durch die Aktivierungsfunktion beschleunigt wird, werden die Parameter von dem Algorithmus gleichzeitig optimiert. Dieser Vorgang heißt Rückpropagierung. Dabei wird das Netz von der Ausgabe- bis zur Eingabeschicht durchlaufen. Um die Übereinstimmung der Ausgabewerte mit den tatsächlichen Werten des Trainingsdatensatzes festzustellen, wird eine Funktion zur Berechnung der Abweichung benötigt [16, 124].

Die Funktion ist je nach Anwendungsfall unterschiedlich. Beispielsweise kann die Verlustfunktion (2.6.3) für die Rückpropagierung eingesetzt werden. Die Rückpropagierung geschieht dabei schrittweise. Für die Durchführung müssen zunächst die Lernrate (2.6.4),

die Losgröße (2.6.5) sowie die Verlustfunktion festgelegt werden. Anschließend erfolgt die Durchführung der folgenden Schritte 1-5 [16, 124-126]:

1. Im ersten Durchgang werden die Parameter w und b zufällig gewählt und durch die Eingabe des ersten Loses an Daten wird die Ausgaben A berechnet. Mithilfe der Verlustfunktion erfolgt die Kalkulation der Abweichung E.

2. Im zweiten Durchgang werden die beiden Parameter zu zufälligen Werten δw und δb geändert.

3. Durch die erneute Eingabe für einen neuen Los an Daten wird die Ausgaben A berechnet und es erfolgt mit Hilfe der Verlustfunktion die Kalkulation der Abweichung E.

4. In diesem Schritt werden die Gradienten berechnet. Aufgrund der bereits festgelegten Gewichtung der Verbindung zur nächsten Schicht ist es möglich, die Auswirkung jedes Neurons in der vorherigen Schicht auf die Änderung des Fehlers δE zu ermitteln. Dies geschieht von der Ausgabe bis zu Eingabeschicht und heißt Rückpropagierung.

5. Die neuen Parameter w und b bildet sich in der Formel 2.3 aus dem Gradientenverfahren mit Hilfe der Lernrate (2.6.4) und dem Verhältnis von $\frac{\delta E}{\delta x}$ [16, 126]:

$$x_{neu} = x_{alt} - (\text{Lernrate} \cdot \frac{\delta E}{\delta x_{alt}}) \tag{2.3}$$

   Wobei x für die Parameter w und b steht.

Die Schritte 3 bis 5 werden solange durchlaufen, bis eine bestimmte Anzahl an Durchläufen stattgefunden hat oder keine Veränderung der Verlustfunktion mehr erkennbar ist [16, 125].

2.6.3 Verlustfunktion

Die Verlustfunktion (im Englischen: Loss) stellt die Abweichung zwischen den vorhergesagten und den tatsächlichen Werten eines Modells dar. Sie wird während des Trainings verwendet, um die Parameter des Modells und die damit einhergehende Qualität des fertig trainierten Modells zu optimieren. Ziel dabei ist es, die Funktion anhand der Parameter zu minimieren [4, 3].

Binäre Kreuzentropie-Funktion

Die Binäre Kreuzentropie-Funktion ist eine Verlustfunktion für die binäre Klassifikationsproblematik. Durch die Anwendung dieser Verlustfunktion ist es möglich, die Abweichung für jede Kategorie zu ermitteln. Die Verlustfunktion L (2.4) stellt sich wie folgt zusammen [23, 112]:

$$L(o,t) = -\frac{1}{N} \cdot \sum_{i=1}^{N} [(t[i] \cdot \log(o[i]) + (1 - t[i]) \cdot \log(1 - o[i]))] \tag{2.4}$$

t = target, das Label für Klasse 0 oder Klasse 1
o = output, Vorhersage für Labelwahrscheinlichkeit t für den Punkt i der Punkte N

2.6.4 Lernrate

Die Lernrate ist ein wichtiger Bestandteil zur Parameteroptimierung, da sie den Grad beeinflusst, mit dem der Fehler-Parameter-Gradient (5. Schritt von 2.6.2) multipliziert wird, um die Optimierungsgeschwindigkeit zu steuern [16, 206]. Somit wird verhindert, dass es zu extremen Änderungen der Parameter kommt und der optimale Wert möglicherweise übergangen wird. Es folgt eine langsamere Änderung der Parameter in die Richtung des optimalen Wertes. Je niedriger die Lernrate ist, desto langsamer wird die Optimierung der Modellparameter durchgeführt. Ist die Lernrate jedoch zu hoch angesetzt, kann es dazu führen, dass der Algorithmus nicht konvergiert und damit das minimale Ergebnis der Verlustfunktion (2.6.3) nicht erzielt wird [16, 132].

2.6.5 Losgröße

Die Losgröße (im Englischen: Batch Size) beschreibt die „Anzahl der Datenpunkte, die für eine Aktualisierung der Modellparameter w und b verwendet werden“ [16, 133]. Eine kleinere Losgröße kann eine Überanpassung des Modells verhindern, indem mehr unterschiedliche Datenlose als Trainingsdaten verfügbar sind und somit die Daten in ihrer Klassifikation nicht verallgemeinert werden. Bei einer Überanpassung lernt der Algorithmus die Daten auswendig, anstatt die Muster hinter den Daten zu erkennen und daraus zu erlernen. Eine größere Losgröße hat den Vorteil, dass die Bestimmung der Fehlerfunktion eine größere Aussagekraft hat, da hier ein gewisser Mittelwert gefunden werden kann [16, 133].

2.7 Transferlernen

Besonders große Netze müssen mit vielen verschiedenen Bildern trainiert werden, da es ansonsten zu einer Überanpassung (2.6.5) kommen könnte. Um mehr Trainingsdaten zu erhalten und den kostspieligen Trainingsvorgang eines neuen Modells zu vermeiden, kann auf ein bereits vortrainiertes Model zurückgegriffen und mit Hilfe dieses Modells mit dem eigentlichen Datensatz weitertrainiert werden. Diese Methode wird als Transferlernen [4, 9] bezeichnet. Eine Methode des Transferlernens ist die Feinabstimmung, in der die oberen Schichten eines Modells aufgetaut und verändert werden. Anschließend erfolgt die Ergänzung einer Klassifizierungschicht der neuen Klassen sowie das erneute Training dieser Schichten. Dieser Ansatz ermöglicht die Merkmale des Grundmodells für den spezifischen Anwendungsfall zu verfeinern [2].

2.8 Evaluierung

Für die Bewertung des Trainingsdurchlaufes muss das durch das Training gebildete Modell validiert werden. Dabei können auch mehrere Trainingsdurchläufe mit unterschiedlichen Hyperparametern einbezogen und miteinander verglichen werden [16, 139]. Für die Evaluierung stehen unterschiedliche Metriken für die Überprüfung der Klassifikation zur Verfügung.

2.8.1 Positiver Vorhersagewert

Für die Analyse der Leistung der Klassifikation eines Modells ist die Definition von Richtwerten nötig. Die Berechnungen der richtig-negativen Erkennung werden hier nicht weiter betrachtet, da nur positive Erkennung auf den Bildern angezeigt werden. Der postive Vorhersagewert (PVW) (im Englischen: Precision) gibt Auskunft über die richtig-positiven Erkennungen eines Bildes. Dabei wird die Anzahl richtig-positiver Objekte durch die Gesamtzahl der gefundenen Objekte ermittelt. Dementsprechend werden damit die falsch-positiven Fälle ausgeschlossen. Der optimale PVW (2.5) eines Modells beträgt 1 [16, 149].

$$\text{PVW} = \frac{\text{RP}}{\text{RP} + \text{FP}} \qquad \text{RP} = \text{richtig positiv, FP} = \text{falsch positiv} \tag{2.5}$$

2.8.2 Sensitivität

Weitere Aussagen über die richtig-positiven Erkennungen eines Bildes können über die Sensitivität eines Modells gegeben werden. Die Sensitivität (im Englischen: Recall) zeigt, wie viele richtig-positive Objekte von der Gesamtzahl der richtig-positiven Objekte gefunden wurden (2.6). Damit beschreibt der PVW den Ausschluss der falsch-negativ Fälle. Der optimale Wert der Sensitivität eines Modells wäre 1 [16, 148-149].

$$\text{Sensitivität} = \frac{\text{RP}}{\text{RP} + \text{FN}} \qquad \text{RP} = \text{richtig positiv, FN} = \text{falsch negativ} \tag{2.6}$$

2.8.3 *F1-Score*

Der „harmonische Mittelwert“ [16, 150] der Sensitivität und des PVWs wird durch den *F1-Score* berechnet. Die Formel (2.7) setzt sich wie folgt zusammen [16, 150]:

$$\textit{F1-Score} = 2 \cdot \frac{\text{PVW} \cdot \text{Sensitivität}}{\text{PVW} + \text{Sensitivität}} \tag{2.7}$$

2.8.4 *Intersection of Union (IoU)*

Um richtig-positiv oder falsch-positiv zu bestimmen, muss die *Intersection of Union (IoU)* ermittelt werden. Der *IoU* beschreibt den Anteil der überschneidenden Fläche der vorhersagten *Bounding Boxes*, im Verhältnis zu den wahren *Bounding Boxes* (*Ground Truth*), welche dem Model durch den Datensatz definiert sind (siehe 2.4) [21, 8].

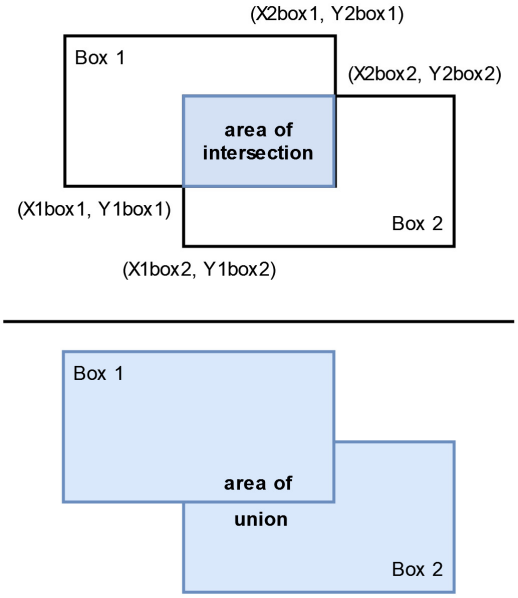


Abbildung 2.4: Veranschaulichung zur Berechnung des IoU

Im Allgemeinen wird bei einem IoU-Wert von 0.5 von einer richtig-positiven Erkennung gesprochen. Liegt der IoU bei 1, stimmen die Koordinaten der Vorhersagung durch das Modell und der *Ground Truth* komplett überein. Der Wert setzt sich aus der folgenden Formel (2.8) zusammen [21, 8]:

$$IoU = \frac{area\ of\ intersection}{area\ of\ union} \tag{2.8}$$

$$\begin{aligned} area\ of\ intersection &= (|min(X_{2box1}, X_{2box2}) - max(X_{1box1}, X_{1box2})|) \\ &\quad \cdot (|min(Y_{2box1}, Y_{2box2}) - max(Y_{1box1}, Y_{1box2})|) \\ area\ of\ union &= ((X_{2box1} - X_{1box1}) \cdot (Y_{2box1} - Y_{1box1})) \\ &\quad + ((X_{2box2} - X_{1box2}) \cdot (Y_{2box2} - Y_{1box2})) \\ &\quad - area\ of\ intersection \end{aligned} \quad \begin{cases} \\ X_1 < X_2 \\ Y_1 < Y_2 \\ \\ \\ \end{cases}$$

3 Aktueller Stand der Forschung

Dieses Kapitel gibt einen Einblick in den aktuellen Forschungstand zu der Objekterkennung und -zählung von Vögeln. Dabei wird vorgestellt, welche Ansätze bereits existieren, um daraus in den folgenden Kapiteln zu schlussfolgern, welche Methoden erweitert werden könnten.

Die Anzahl der gefundenen Artikel in Bezug auf Objekterkennung und -zählung von Vögeln durch maschinelles Lernen ist sehr begrenzt. Es wurden vor allem Techniken der Vogelerkennung gefunden, welche sich mit der Erkennung einzelner Vögel und deren Klassifizierung beschäftigen. Der Artikel von Ce Li [14] beschäftigt sich mit der Vogelerkennung auf der Basis von tiefen Faltungsnetzwerken (2.4.1) und untersucht dabei, wie die Vogelerkennung bei mangelnder Bildqualität verbessert werden kann. Andere Artikel, wie von Shazzadul Islam [10] beschreiben die Umsetzung einer Klassifizierung von Vögeln auf Grund ihrer Vogelart.

Es wurde eine Methode entwickelt und getestet, welche sich mit der Vogelzählung anhand von Bilderkennung beschäftigt. Der Artikel „Automated Bird Counting with Deep Learning for Regional Bird Distribution Mapping“ [4] beschreibt, wie mit einer Methodik die Zählung von Vögeln ermöglicht wird, um daraus anschließend eine Übersicht über die Vogelverteilung zu erhalten [4, 2]. Die verschiedenen Methoden zeigen dabei mögliche Ansätze für die in dieser Arbeit durchgeführten Experimente. Dafür werden in den folgenden Abschnitten die Methoden, welche relevant für die Thesis sein könnten, näher erläutert.

3.1 Datensatz

Die Anzahl der Datensätze von Vögeln in Vogelschwärmen ist begrenzt, da sich der Forschungstand vor allem auf Vogelartenunterscheidung und die Erkennung einzelner Vögel bezieht. Für die Evaluation in dem Artikel [14] wird zum Teil ein öffentlicher Datensatz

CUB-200 [29] mit Vögeln unterschiedlicher Arten verwendet, in welchem Bilder mit nur jeweils einer *Bounding Box*, sprich einem Vogel, vorhanden sind [14, 804-805].

Die in dem Artikel [10] vorgestellte Arbeit verwendet einen eigenen nicht öffentlich zugänglichen Datensatz mit Vögeln in verschiedenen Bewegungen und mit unterschiedlichen Hintergründen sowie Umgebungen. Als Beispielbilder werden nur Bilder verwendet, die einen Vogel zeigen [10, 39-40].

Für das Trainieren und Validieren des Modells in dem Artikel [4] wurde ein Datensatz verwendet, welcher auf jedem Bild Vogelschwärme zeigt. Dort sind zum größeren Teil Vögel in der Luft, aber auch auf dem Boden aus unterschiedlicher Entfernung zu sehen. Für den Datensatz wurden 3436 Bilder mit Vogelszenen fotografiert und daraus ein großer Datensatz [3] erstellt [4, 9]. Die Bilder aus dem Datensatz wurden verwendet, um verschiedene Methodiken des maschinellen Lernens zur Erkennung von Vögeln in Schwärmen zu testen.

MS COCO und Pascal VOC

In den vergangenen Jahren wurden einige umfangreiche Datensätze zur Klassifikation von Objekten in Bildern veröffentlicht, darunter *MS COCO* [6] und *Pascal VOC*. Beide Datensätze beinhalten verschiedene Klassen von alltäglichen Objekten. Im Vergleich zu *Pascal VOC* verfügt *COCO* über eine größere Anzahl von Objektkategorien und eine beträchtliche Menge von Objektinstanzen. *MS COCO* umfasst komplexere Szenarien mit besonders häufig auftretenden Objekten. Zusätzlich sind die Objekte in *MS COCO* vollständig segmentiert und die Instanzen sind sorgfältig beschriftet, was eine umfassendere Bewertung der Detektorleistung ermöglicht[5, 4].

3.2 Architektur des Modells

Für die Modellbildung muss zunächst eine Architektur festgelegt werden. Die verschiedenen Ansätze zur Vogelerkennung haben teils Unterschiede, aber auch Gemeinsamkeiten in dem Aufbau ihrer Modelarchitektur. Die Modelle von [4] und [14] basieren beide auf der *Faster R-CNN*-Architektur (3.2.1). Grund dafür ist, dass der Algorithmus vor allem für seine Leistung in Bezug auf die schnelle Erkennung von Objekten (5 Bilder pro Sekunde für alle Verarbeitungsschritte auf einer *GPU*) sowie für seine hohe Genauigkeit

bekannt ist, was sich durch den ersten Platz bei den *ILSVRC*- und *COCO*-Wettbewerben im Jahr 2015 bestätigte [14, 803].

In [10] wird von der Methodik des Transferlernens Gebrauch gemacht, um ein Faltungsnetzwerk (2.4.1) zu optimieren. Es handelt sich dabei um das *VGG-16*, welches jedoch nicht mehr als State-of-the-Art-Modell gilt [10, 39].

Der Artikel von Hüseyin Gökhan Akçay [4] stellt verschiedene Methoden zur Objekterkennung vor. Dabei wurde festgestellt, dass im Vergleich zu anderen, in dem Artikel vorgestellten Detektoren der *Faster R-CNN* die geringste falsch-positive und falsch-negative Rate aufweist [4, 11]. Dabei wird ebenfalls die Methodik des Transferlernens (2.7) verwendet. Das *Inception-v2*, ein Netz mit Merkmalsextraktion, besteht aus mehr als 13 Faltungsschichten, welche mit dem *Oxford-IIT Pet Dataset* trainiert wurden [4, 10]. Der verwendete Datensatz von Oxford verfügt über zwei verschiedene Klassen mit Hunden und Katzen [7]. Aus der Trainingskonfigurationsdatei des Projektes [1] geht hervor, dass das Feinabstimmung mit der Funktion *tf.nn.softmax_ cross_ entropy_ with_ logits* von *Tensorflow* durchgeführt wurde. Dies ist eine Funktion, die sich auf die Multi-Klassen-Klassifikation konzentriert [2].

3.2.1 *Faster R-CNN*-Architektur

Mehrere Artikel greifen auf den Gebrauch einer *Faster R-CNN*-Architektur zurück. Die *Faster R-CNN*-Architektur besteht aus mehreren Komponenten. Zunächst gibt es ein Netz (CNN 2.4.1) zur Merkmalsextraktion, das aus verschiedenen Faltungsschichten besteht. Die Schichten extrahieren dabei unterschiedliche Merkmale des Bildes. Die unteren Schichten erfassen vor allem Kanten und Kleckse. Die oberen Schichten erkennen spezifischere Merkmale eines Objektes. Anschließend folgen zwei Unternetze: das *Regional Proposal Network* (RPN) und der *Fast R-CNN*-Detektor [4, 7-8]. Die *Faster R-CNN*-Architektur ist ein Zweistufen-Detektor mit dem RPN und den Interessensregionen (RoI) [5, 1]. Das RPN nutzt als Eingabe die letzte Merkmalskarte aus dem CNN, um Regionen von Interesse (RoI) zu generieren. Diese ROIs stellen Bereiche im Bild dar, in denen Objekte mit hoher Wahrscheinlichkeit vorhanden sind. Die ROIs werden anschließend dem *Fast R-CNN*-Detektor übergeben, der wiederum jedem Objekt die entsprechende Klasse zuweist. Die abschließende Ausgabe des Detektors umfasst das endgültige Klassenlabel sowie die berechneten Koordinaten gefundener *Bounding Boxes* (2.5.1) [4, 7-8].

3.2.2 *Mask R-CNN*-Architektur

In dem Artikel „Object Detection and Localization in Natural Scenes Through Single-Step and Two-Step Models“ [5] werden unterschiedliche Architekturen und ihre Leistung verglichen, indem sie auf zwei bekannte Datensätze, *MS COCO* und *Pascal VOC*, (3.1) angewandt werden. Der Artikel stellt eine erweiterte Version des *Faster R-CNN* vor, bekannt als *Mask R-CNN*. Dieser zeichnet sich durch die gleichzeitige Durchführung von Segmentierung und Erkennung in einem Arbeitsschritt aus. Unter allen anderen Objekterkennungsalgorithmen wird *Mask R-CNN* aufgrund seines flexiblen Ansatzes für die Kombination von Objekterkennung und Segmentierung als vergleichsweise am präzisesten angesehen [5, 2]. Es zeigt sich, dass *Mask R-CNN* im Allgemeinen genauer im Vergleich zu den vorgestellten Architekturen arbeitet [5, 6].

3.3 Framework: *Tensorflow*

Für die Erstellung eines Modells muss ein Framework gewählt werden. *Tensorflow* ist ein vergleichsweise neues Framework von *Google* und kann für die Modellierung eines neuronalen Netzwerks verwendet werden [20, 1611]. Die Nutzung der *Tensorflow API* ist führend bei der Identifizierung von Objekten in Echtzeit, unabhängig davon, ob diese Objekte statisch oder in Bewegung sind. In Kombination mit KI-Anwendungen bietet diese Erweiterung die höchstmögliche Präzision und Genauigkeit [20, 1618]. Die Präferenz für *Tensorflow* gegenüber anderen Frameworks hat verschiedene Gründe. Die einfache Installation, die durch den Paketmanager `pip` für Python [8] vermittelt wird, erleichtert die Verwendung. Im Vergleich zu früheren Modellen bietet dieses Framework eine verbesserte Unterstützung für *GPUs*. Es stellt *APIs* für die Modellerstellung zur Verfügung, wodurch die Entwicklung erleichtert wird [20, 1615].

Tensorflow verfügt über zwei Versionen: *Tensorflow* 1 und *Tensorflow* 2. Bei *Tensorflow* 2 handelt es sich um die neuere Version, welche zahlreiche Vorteile in Bezug auf Benutzerfreundlichkeit und den Entwicklungsprozess bietet. Zum Beispiel werden die *APIs* einheitlicher gestaltet und redundante *APIs* entfernt, wodurch die Benutzerfreundlichkeit erhöht wird. Zudem priorisiert *Tensorflow* 2 Funktionen anstelle von Sitzungen und ist somit besser in die *Python*-Laufzeitumgebung eingebunden [2].

Zusammenfassung

Es zeigt sich, dass die Forschung für die Zählung von Vögeln in Schwärmen kaum Entwicklungen aufweist. Zu sehen ist jedoch, dass bereits viele Methoden zur Objekterkennung entwickelt wurden, auf die zu diesem Zweck zurückgegriffen werden kann. Mehrere Ansätze bauen auf einer *Faster R-CNN*-Architektur auf, um die Erkennung von Vögeln durchzuführen. Es wurde bereits eine Erweiterung der Architektur entwickelt, die *Mask R-CNN*-Architektur, um eine präzisere Segmentierung zu ermöglichen. Im Allgemeinen verwenden die oben genannten Ansätze alle ein Faltungsnetzwerk für die Vogelerkennung (2.4.1). Alle Architekturen sind mit großen Datensätzen trainiert worden und als vortrainierte Modelle über das Framework *Tensorflow* verfügbar.

Grundsätzlich steht die Modellform noch offen. Für den spezifischen Ansatz der Vogelzählung in Schwärmen ist dies in den folgenden Kapiteln zu ergründen.

4 Anforderungsanalyse

Für die Konzeption und Implementierung der Anwendung müssen zunächst die Anforderungen an das System festgelegt werden. Dafür wird in diesem Kapitel eine umfangreiche Analyse der Anforderungen an das System vorgenommen.

4.1 Systemkontext

Zunächst wird ein grober Überblick über das System im Rahmen eines Systemkontextdiagramms gegeben. Alle Aufgaben, die in einem System durch Fremdeinwirkung durchgeführt werden und nicht mehr zum eigentlichen System gehören, werden durch das Diagramm verdeutlicht. Anschließend ist es möglich, aus dem Diagramm genauere Anforderungen und Anwendungsfälle zu kreieren. Bei den Anwendungsfällen handelt es sich um Situationen, die bei der Verwendung des Systems eintreten können [18, 244-247].

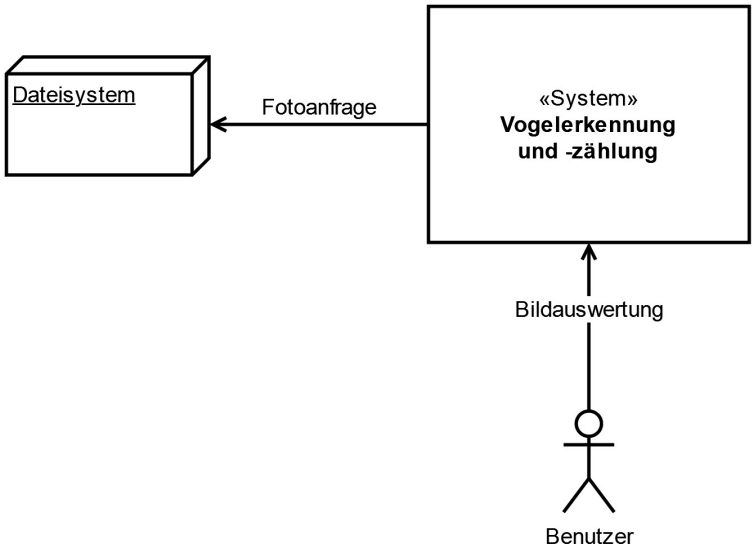


Abbildung 4.1: Systemkontextdiagramm

Die Abbildung 4.1 zeigt das Systemkontextdiagramm der Anwendung dieser Arbeit. Das System der Vogelzählung benötigt zwei Dialogschnittstellen. Die eine Schnittstelle wird durch den Benutzer betätigt, um die Vogelzählung zu starten. Die andere Schnittstelle greift über das System auf das Dateisystem des Benutzers zu, damit dieser sein gewünschtes Bild auswählen kann.

4.2 Anforderungen

Für die Anforderungsanalyse werden sowohl funktionale Anforderungen für die Funktionalitäten eines Systems als auch nichtfunktionale Anforderungen für das Systemverhalten definiert. Dabei werden die Dienste des Systems und dessen Randbedingungen beschrieben, welche für das System als erfüllende Eigenschaften gelten. Für eine Anforderungsanalyse müssen die Akteure des Systems klar definiert sein, ebenso deren damit verbundenen Interessen. Daraus können Schwachstellen der Anwendung entstehen, welche in den Anforderungen berücksichtigt werden müssen. Die Anforderungen sind aus der Sicht der Stakeholder formuliert, bei welchen es sich in dieser Analyse um die Benutzer handelt. Dabei werden die Beziehungen des Systems aus dem Systemkontextdiagramm 4.1 in die Anforderung mit einbezogen.

4.2.1 Funktionale Anforderungen

Die funktionalen Anforderungen sind in der Tabelle nach Priorität absteigend aufgelistet.

Tabelle 4.1: Funktionale Anforderungen

Bezeichnung	Beschreibung	Priorität
Anforderung 1	Das System soll einen Vogel auf einem Bild als diesen erkennen.	Hoch
Anforderung 2	Das System soll viele Vögel bzw. teils verdeckte Vögel auf einem Bild identifizieren.	Hoch
Anforderung 3	Das System soll alle identifizierten Vögel zählen.	Hoch
Anforderung 4	Das System soll keine Vögel erkennen, wo keine sind.	Hoch
Anforderung 5	Das System soll dem Benutzer das Auswählen eines Bildes aus dem Dateisystem ermöglichen.	Hoch
Anforderung 6	Das System soll dem Benutzer das Ergebnis zurückgeben.	Hoch
Anforderung 7	Das System soll dem Benutzer alle Ergebnisse der Vogelzählungen zurückgeben.	mittel
Anforderung 8	Das System soll die Vogelerkennung bei den Bildern eines ganzen Ordners durchführen.	mittel
Anforderung 9	Das System soll jede Art von Bilddateien zulassen.	Niedrig
Anforderung 10	Das System soll auf den Wunsch des Benutzers den Speicherort der Ergebnisse ändern	Niedrig

4.2.2 Anwendungsfälle der funktionalen Anforderungen

Für die funktionalen Anforderungen werden in diesem Unterabschnitt typische Anwendungsfälle definiert, welche bei der Verwendung des Systems stattfinden können. Daraus können Interessen der Akteure hervorgehen und den möglichen Aufbau eines Systems eindeutiger werden lassen. Die Anwendungsfälle beschreiben einen Ablauf der Anwendung aus der Sicht des Benutzers. Aus den Anforderungen der Tabelle 4.1 gehen drei Anwendungsfälle hervor: 1. Identifikation und Zählung von Vögeln auf einem Bild, 2. die Ergebnishistorie anzeigen, 3. einen neuen Speicherort festlegen. Für diese drei Anwendungsfälle werden zunächst Aktivitätsdiagramme erstellt, welche die Abläufe modellieren [12, 10]. Die Aktivitätsdiagramme können eine Grundlage für spätere Testfälle darstellen [12, 76]. Im Weiteren werden die Anwendungsfälle tabellarisch aufgeführt, welche aus den Aktivitätsdiagrammen hervorgehen.

Die Abbildung 4.2 zeigt den Aktivitätenfluss der Vogelzählung durch die Anwendung.

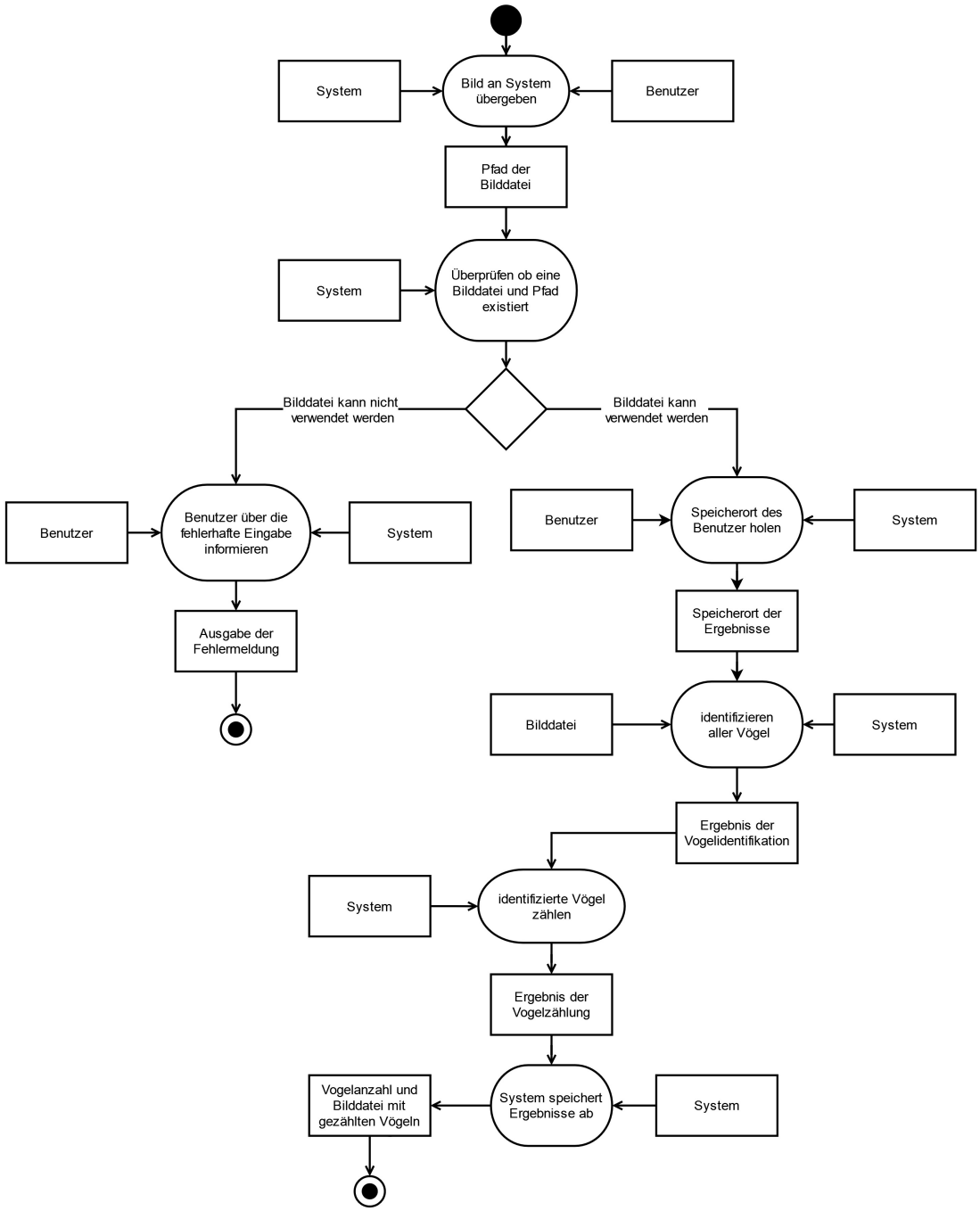


Abbildung 4.2: Aktivitätsdiagramm der Vogelzählung

Anhand des Diagramms 4.2 kann ein Anwendungsfall 4.2 gebildet werden.

Tabelle 4.2: Anwendungsfall 1

Name	US-1: Identifikation und Zählung von Vögeln auf einem Bild
Kurzbeschreibung	Der Benutzer übergibt eine Bilddatei von einem Vogelschwarm an eine Anwendung. Als Rückgabe erhält er die Anzahl der Vögel auf dem Bild.
Akteure	Benutzer
Auslöser, Motivation	Der Benutzer möchte die Verteilung von Vögeln, durch die Anzahl von Vögeln eines Schwarms herausfinden.
Ergebnis	Die Anzahl der Vögel und das Bild inklusive *Bounding Boxes*.
Eingehende Information	eine Bilddatei
Vorbedingung	Der Benutzer hat ein Bild von einem Vogelschwarm.
Nachbedingung	Der Benutzer hat die Anzahl der Vögel auf dem Bild.
Essenzielle Schritte	1. Der Benutzer übergibt die Bilddatei an das Programm. 2. Das System überprüft die Bilddatei. 3. Das System holt sich Speicherpfad des Benutzers. 4. Das System identifiziert alle Vögel. 5. Das System zählt die identifizierten Vögel. 6. Das System gibt dem Benutzer die Vogelanzahl als Text aus und speichert das hochgeladene Bild mit *Bounding Boxes*.

Das Diagramm in der Abbildung 4.3 veranschaulicht den Ablauf der Aktivitäten für die Ausgabe der Ergebnishistorie der Vogelzählungen.

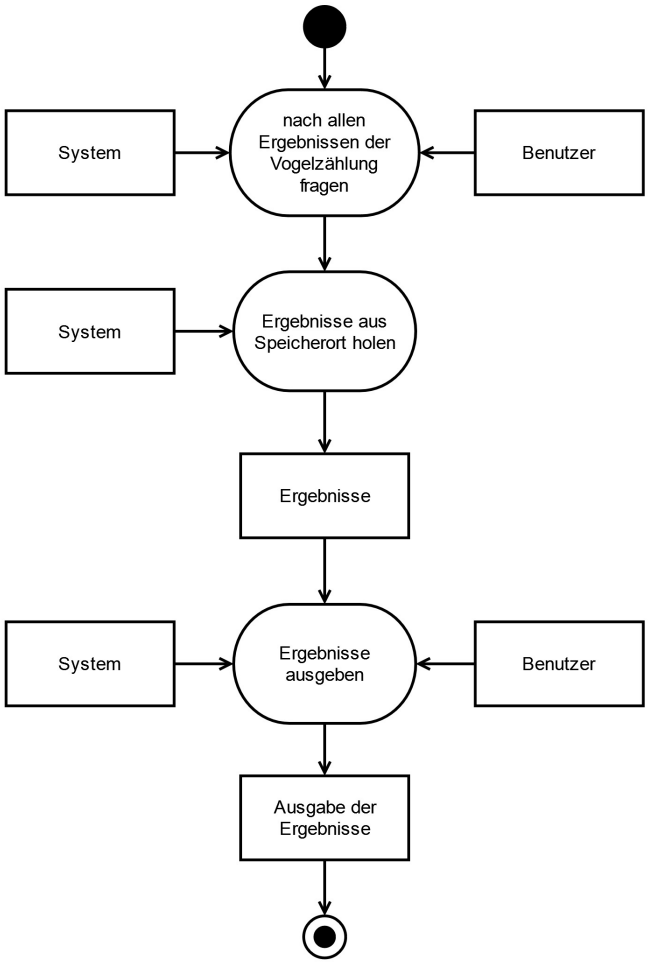


Abbildung 4.3: Aktivitätsdiagramm der Historienausgabe aller Ergebnisse

Das Diagramm 4.3 dient als Grundlage für die Erstellung eines Anwendungsfalls 4.3, der die Ausgabe aller Ergebnisse beschreibt.

Tabelle 4.3: Anwendungsfall 2

Name	US-2: Ergebnishistorie anzeigen
Kurzbeschreibung	Der Benutzer lässt sich alle bereits durchgeführten Vogelzählungen wiedergeben.
Akteur	Benutzer
Auslöser, Motivation	Der Benutzer möchte ein Ergebnis anschauen.
Ergebnis	Eine Ausgabe der Pfade der Bilddateien und die dazugehörige Anzahl von gezählten Vögeln.
Eingehende Information	keine
Vorbedingung	Die Anwendung hat bereits Vögel auf Bildern gezählt.
Nachbedingung	Der Benutzer hat eine Ausgabe mit den Ergebnissen der gezählten Vögel.
Essenzielle Schritte	1. Der Benutzer gibt an, dass er alle Ergebnisse einsehen möchte. 2. Das System holt sich alle Ergebnisse aus dem Speicherort. 3. Das System gibt dem Benutzer alle Ergebnisse aus.

Die Abbildung 4.4 zeigt das Aktivitätsdiagramm der Änderung des Pfades für die Speicherung der Ergebnisse der Vogelzählung.

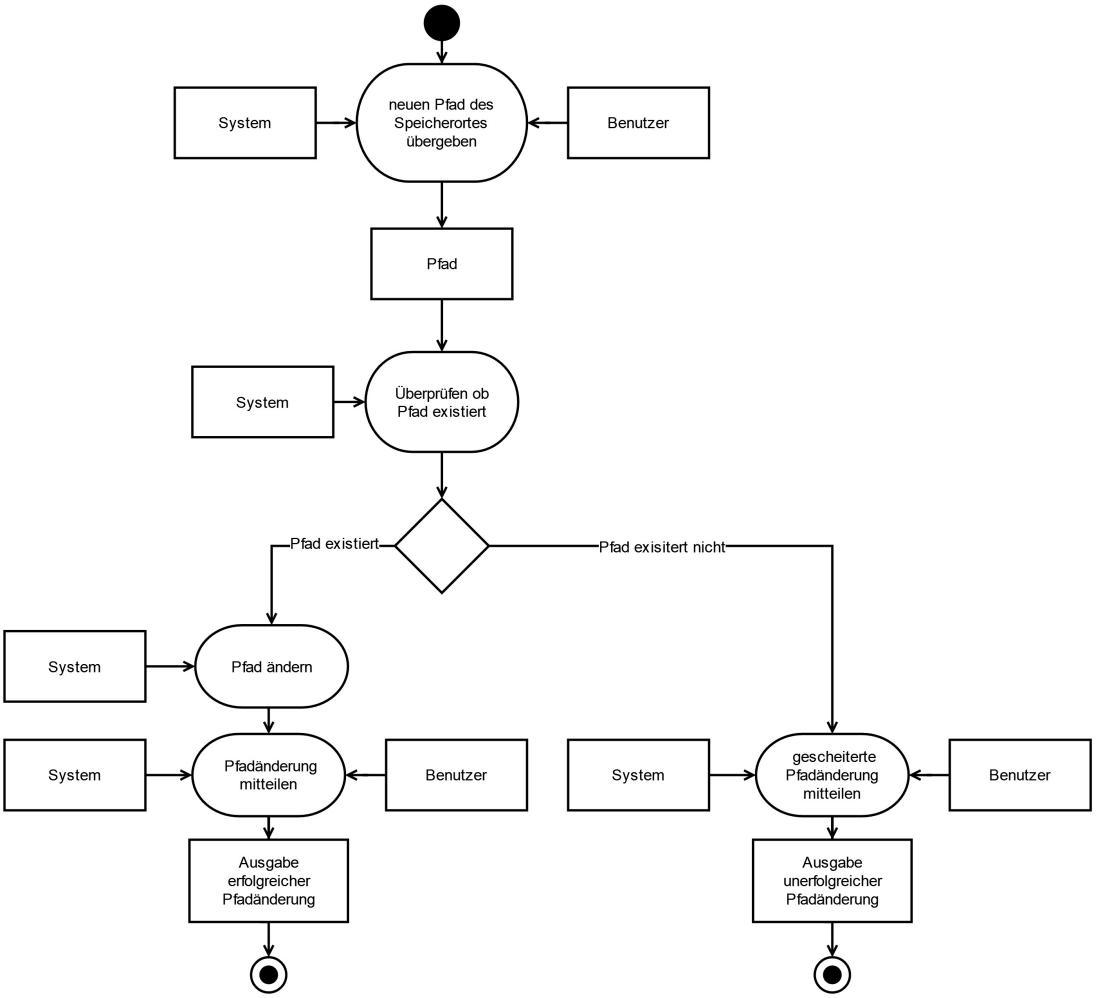


Abbildung 4.4: Aktivitätsdiagramm der Änderung des Speicherortes

Mithilfe des Diagramms 4.4 kann ein Anwendungsfall 4.4 erstellt werden.

Tabelle 4.4: Anwendungsfall 3

Name	US-3: Einen neuen Speicherort festlegen
Kurzbeschreibung	Der Benutzer möchte einen neuen Speicherort für die Ergebnisse der Vogelzählungen festlegen.
Akteur	Benutzer
Auslöser, Motivation	Der Benutzer möchte einen neuen Speicherort festlegen.
Ergebnis	Die Bestätigung des Systems, dass ein neuer Speicherort festgelegt wurde.
Eingehende Information	Einen Pfad für einen neuen Speicherort.
Vorbedingung	Der Benutzer hat einen neuen Pfad bestimmt.
Nachbedingung	Der Speicherort wurde geändert.
Essenzielle Schritte	1. Der Benutzer übergibt den neuen Pfad des Speicherortes an das Programm. 2. Das System überprüft den Pfad. 3. Das System ändert den Pfad des Speicherortes der Ergebnisse. 4. Das System gibt aus, dass der Pfad des Speicherortes geändert wurde.

4.2.3 Nichtfunktionale Anforderungen

Neben den funktionalen Anforderungen müssen auch die nichtfunktionalen Anforderungen an ein System beschrieben werden. Dabei handelt es sich um das Verhalten eines Systems, welches nicht direkt mit den Funktionalitäten eines Systems einhergeht und durch Randbedingungen und Qualitätseigenschaften definiert wird. Die nichtfunktionalen Anforderungen des in dieser Arbeit behandelten Systems werden standardmäßig nach der Norm ISO 9126 wie folgt unterteilt [18, 265].

Zuverlässigkeit

Durch das Testen mittels Testdaten kann das System ausgiebig getestet werden. Das Lernverhalten kann über den Einsatz von Trainingsmetriken (2.6) überwacht werden. Das System soll so viele Vögel auf den Bildern erkennen, dass der Benutzer eine Aussage über den Vogelbestand treffen kann. Dafür wurde als Messwert ein *F1-Score* von mindestens 0,95 über einen großen Satz von Testdaten festgelegt. Um die Fehlertoleranz der Anwendung zu gewähren, soll bei falscher Verwendung der Schnittstellen die Anwendung nicht abbrechen und der Benutzer über eine richtige Verwendung des Systems informiert werden.

Benutzbarkeit

Der Benutzer soll eine einfache Bedienung der Anwendung erhalten, damit die Verwendung des Systems dem tatsächlichen Zählen von Vögeln vorgezogen wird. Da das System die Datenauswertung der Vogelverteilung bedient, soll die Anwendungs innerhalb der Dateisysteme schnell ausführbar sein. Dazu zählt auch die direkte Speicherung der Ergebnisse, welche die anschließende Weiterverarbeitung der Daten ermöglichen kann.

Effizienz

Der Aufwand des selbstständigen Suchens durch einen Menschen soll dabei kleiner sein als das Hochladen eines Bildes. Dabei darf das Zählen der Vögel nicht länger als eine Minute dauern.

Änderbarkeit und Wartbarkeit

Sofern die Leistung des Systems nicht den Wünschen des Benutzers entspricht, kann das Modell des Systems ausgetauscht werden und somit eine eventuelle Leistungsoptimierung stattfinden. Die Software soll mit wenig Aufwand bei Änderungen der Module zu testen sein.

Übertragbarkeit

Das System hat einen modularen Aufbau. Dies kann für eine Weiterentwicklung der Anwendung für die Artenklassifikation (1.2) der Vögel oder den Einbau anderer Benutzeroberflächen hilfreich sein.

4.3 Fachliches Datenmodell

Aus den Anwendungsfällen kann der Aufbau des Systems erstellt werden. Dafür kann ein fachliches Datenmodell modelliert werden, welches die Entitäten eines Systems sowie deren Beziehungen zueinander darstellt. Das Modell gibt einen Einblick in die mögliche Datenstruktur und kann für spätere Entwurfsphasen verwendet werden.

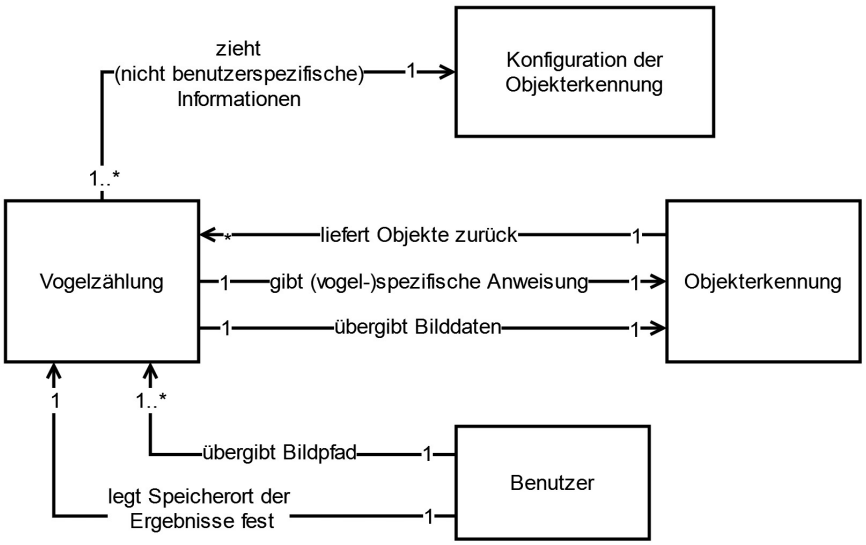


Abbildung 4.5: Fachliches Datenmodell

Aus den Anwendungsfällen geht hervor, dass der Benutzer sowohl Bilder übergeben, wie auch Speicherpfade der Vogelzählung bestimmen kann. Da bereits in dieser Entwicklungsphase klar ist, dass auf eine externe Objekterkennung (3.3) zugegriffen wird, trennt das Datenmodell für die Zählung der Vögel zwischen Vogelzählung und Objekterkennung. Die separate Objekterkennung erhält Bilddaten um daraus Objekte zu extrahieren. Für die Objekterkennung wird eine spezifische Anweisung benötigt, welche über eine Konfigurationsklasse festgelegt sind.

5 Konzept

Um die funktionalen und nicht-funktionalen Anforderungen aus dem vorherigen Kapitel 4 umzusetzen, muss ein Konzept entwickelt werden. In dem Entwurf wird die Architektur des Systems sowie der genaue Aufbau und die Interaktionen mit und innerhalb des Systems deutlich. Es wird nacheinander das Konzept für die Umsetzung der einzelnen Anwendungsfälle aus Kapitel 4 vorgestellt, sowie daraus resultierenden Systeminteraktion (5.5.1) beschrieben. Anschließend werden grundlegende technische Aspekte (5.4) festgelegt und am Ende ein Überblick über das entworfene Klassendiagramm 5.5 gegeben.

5.1 Identifikation und Zählung von Vögeln

Für den Anwendungsfall 4.2 soll die Vogelzählung auf einem Bild möglich sein. Dafür wird ein Sequenzdiagramm in Abbildung 5.1 aus den Abläufen des Aktivitätsdiagramm 4.2 und den Entitäten des fachlichen Datenmodells 4.5 gebildet und durch Steuerung der Benutzerinteraktion ergänzt. Sequenzdiagramme veranschaulichen die dynamische Interaktion von Objekten, die in einem Klassendiagramm statisch miteinander verknüpft sind. Sie erlauben die Visualisierung von Methodenaufrufen und deren Rückgabewerten in chronologischer Abfolge [12, 385]. Dabei wird die Gelegenheit zur Überprüfung gegeben, ob die Abläufe, die in den Aktivitätsdiagrammen beschrieben werden, mit den im Klassendiagramm 5.5 festgelegten Funktionalitäten umgesetzt werden können [12, 93]. Die Klasse *BirdDetection* ruft die Methode *count_ bird(path)* der Klasse *BirdCountController* auf. Anschließend wird der Pfad des Speicherordners für die Ergebnisse von dem *User* über die Methode *get_ saving_ path()* abgefragt. Mit dem Pfad des Bildes und dem Speicherpfad wird die Methode *count_ bird_ on_ file(image_ path, saving_ path)* aufgerufen. Die von der Klasse *BirdCount* zurückgegebene Anzahl der Vögel und der Pfad des Bildes werden durch die Methode *save_ bird_ count(path, count)* über die Klasse *User* gespeichert.

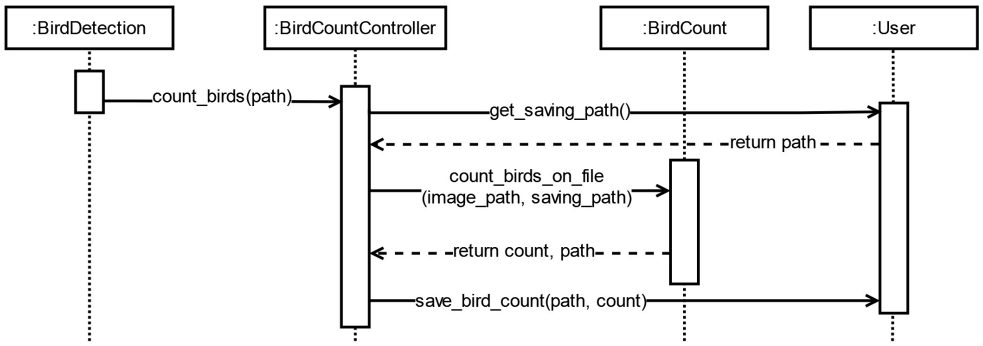


Abbildung 5.1: Sequenzdiagramm des Anwendungsfalls 4.2

5.1.1 Modellwahl

In dieser Arbeit wird die Objekterkennung genutzt, um Objekte auf Bildern zu erkennen. Sie wird mit Hilfe von maschinellem Lernen (2.2) durchgeführt, um Objekte auf Bildern zu identifizieren und diese in die Kategorien „Vogel“ oder „kein Vogel“ einzuordnen. Für die Klassifikation von Vögeln soll auf das in dem Abschnitt 2.2 vorgestellte überwachte Lernen zurückgegriffen werden.

In dem Kapitel über den aktuellen Forschungsstand (3) sind verschiedene Techniken zur Vogelerkennung und -zählung vorgestellt worden. Für den Einstieg soll mit Hilfe von Transferlernen (2.7) ein vortrainiertes Modell ausgewählt werden.

Aus dem Abschnitt 3.2 geht hervor, dass viele Techniken auf der *Faster R-CNN*-Architektur (3.2.1) basieren, weshalb diese Architektur ausgewählt wird. Die darauf aufbauende Erweiterung *Mask R-CNN*-Architektur wird ebenfalls gewählt, da die vorhandene Segmentierungsfunktion nützlich sein könnte für die Extraktion einzelner Merkmale der Vögel, welche durch die gegenseitige Überdeckung der einzelnen Vögel zu sehen sind. Beide Architekturen sollen für die engere Auswahl des Modells betrachtet werden.

Die zu verwendenden vortrainierten Modelle wurden bereits mit Datensätzen trainiert. Der Datensatz des Modells dieser Anwendung sollte bereits die Klasse „Vogel“ kennen, um daraus abzuleiten, welche der Architekturen den Vogel am besten erkennen kann. Für den richtigen Vergleich sollten alle Modelle mit dem gleichen Datensatz trainiert worden sein.

In dem Unterabschnitt 3.1 sind zwei Datensätze vorgestellt worden, die beide die Klasse „Vogel“ beinhalten. Auf Grund der Menge von Objektinstanzen und der Erfassung komplexerer Szenarien soll der Datensatz *MS COCO* [6] verwendet werden.

Die Wahl des vortrainierten Modells wird durch Experimente bestimmt und in dem Kapitel 7 beschrieben.

5.1.2 Datensatz

Für das Training und die Evaluation der Modelle wird wiederum ein Datensatz benötigt, der Vogelschwärme enthält. Der in dem Abschnitt 3.1 beschriebene Datensatz [3] soll für die beiden Vorgänge verwendet werden. Er beinhaltet Bilddateien mit dazugehörigen *xml*-Dateien, in welchen die jeweiligen *Bounding Boxes* mit ihrer Größe, Position und Klasse deklariert sind.

5.1.3 Trainingsparameter

Für das Training müssen Hyperparameter (2.6) eingesetzt werden. Die Form, Tiefe und Breite des künstlichen neuronalen Netzes sowie die Lernrate (2.6.4) werden in dem Kapitel 7 festgelegt. Die Losgröße (2.6.5) muss während des Trainingsvorgangs herausgefunden werden und ist abhängig von der Leistung der Trainingsumgebung (5.4.3).

Für die Anwendung in dieser Arbeit wird als Aktivierungfunktion die Sigmoid-Funktion (2.6.1) eingesetzt, da eine binäre Klassifikation von „Vogel“ (1) und „kein Vogel“ (0) gefragt ist. Für die Aktivierungsfunktion muss eine Verlustfunktion (2.6.3) festgelegt werden. Für die binäre Klassifikation wird die binäre Kreuzentropie-Funktion (2.6.3) verwendet.

5.2 Ergebnishistorie anzeigen

Für die Konzeption (Anwendungsfall 4.3) erfolgt die Ausgabe der Ergebnishistorie aller Vogelzählungen. Dafür wird wie in Abschnitt 5.1 ein Sequenzdiagramm 5.2 entworfen.

Über die Klasse *BirdDetection* wird die *BirdCountController* Methode *print_ all_ count_ results()* aufgerufen. Die Klasse holt sich alle Ergebnisse der Vogelzählungen über die Methode *get_ all_ bird_ counts()* der *User* Klasse.

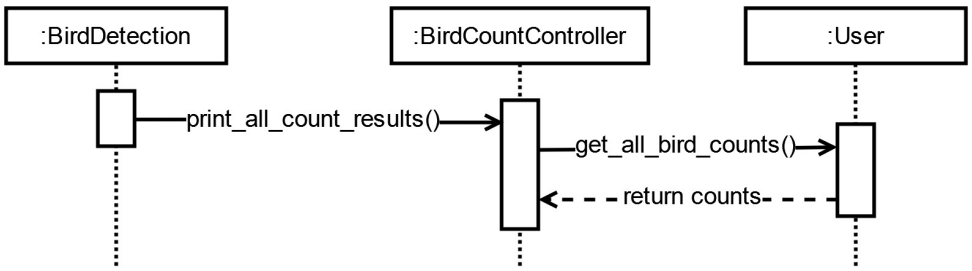


Abbildung 5.2: Sequenzdiagramm des Anwendungsfalls 4.3

5.3 Neuen Speicherort festlegen

Bei einem Speicherort handelt es sich um einen Ordner im Dateisystem. Um einen neuen Speicherordner festzulegen (Anwendungsfall 4.4), erfolgt wie in Abschnitt 5.1, eine Darstellung des Sequenzdiagramms 5.3.
Für diesen Anwendungsfall hat sich herausgestellt, dass es sinnvoll sein kann, dem Benutzer ebenfalls die Einsicht auf den aktuellen Pfad des Speicherordners zugeben. Für das Speichern des neuen Pfades wird von der Klasse *BirdDetection* die Methode des *BirdCountController set_saving_path(path)* aufgerufen. Über den *User* selbst wird der Pfad mit der Methode *set_saving_path(path)* gesetzt.

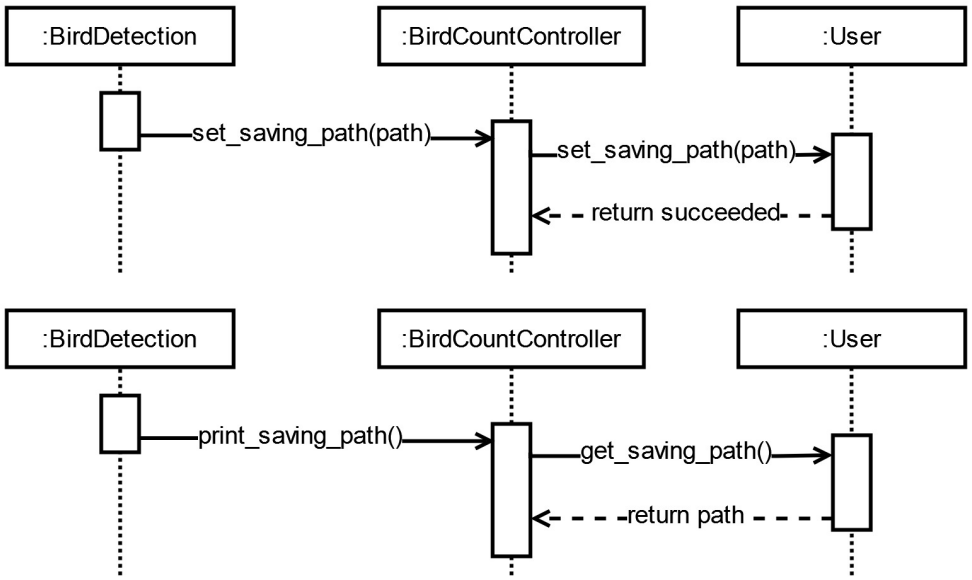


Abbildung 5.3: Sequenzdiagramm des Anwendungsfalls 4.4

Um den aktuellen Pfad des Speicherordners für die Ergebnisse einzusehen, kann dieser über die Klasse *BirdDetection* mit der Methode *print_saving_path()* des *BirdCount-Controller* angefordert werden. Der Pfad wird in *User* gespeichert und mit der Methode *get_saving_path()* abgerufen.

5.4 Allgemeine technische Bausteine

Der Abschnitt beschreibt die technischen Bausteine des Konzeptes, welche aus den Anforderungen (4) und dem aktuellen Forschungsstand (3) ermittelt werden.

5.4.1 Framework: *Tensorflow*

Für die Erstellung und das Trainieren eines Modells muss zunächst ein Framework festgelegt werden. Das Framework *Tensorflow* und seine verschiedenen Versionen wurden bereits im Abschnitt 3.3 vorgestellt. Es stellt für beide Versionen über die *Tensoflow API* [2] das Package *object_detection* zur Verfügung. Die *API* ist über *Github* öffentlich verfügbar [27]. Für die Umsetzung der Objekterkennung in dieser Arbeit wird die 2. Version des Frameworks *Tensorflow* verwendet. Die Vorteile dieser Version können genutzt werden, wenn das Framework vortrainierte Modelle bereitstellt, die den Kriterien aus dem Unterabschnitt 5.1.1 entsprechen. Die von *Tensorflow* bereitgestellten trainierten Modelle sind für die Version 2 in dem *Model Zoo* von *Tensorflow* [28] zu finden. Der *Model Zoo* von *Tensorflow* verfügt über verschiedene vortrainierte Modelle, welche auf den ausgewählten Architekturen (5.1.1) *Faster R-CNN* Architektur oder *Mask R-CNN* Architektur aufbauen und mit dem Datensatz von *MS COCO* (3.1) traininert worden sind.

5.4.2 Wahl der Programmiersprache

Das Framework *Tensorflow* basiert auf der Programmiersprache *Python*. Um eine einheitliche Implementierung zu erlangen und einen problemlosen Übergang zwischen Schnittstellen der Anwendung und *Tensorflow* zu gewährleisten, wird die gesamte Implementierung in *Python* geschrieben.

5.4.3 *Google Colab*

Die Aufsetzung von *Tensorflow* kann durch die Bereitstellung bereits installierter Packages von *Google* sehr effizient durchgeführt werden. Hinzukommt, dass *Google Colab* dem Verwender Rechenleistung zur Verfügung stellt. Die Leistung kann für die Umsetzung des Trainings von Modellen und den Durchlauf der Objekterkennung hilfreich sein, da diese viel Rechenleistung in Anspruch nehmen.

5.5 Modelbeschreibung

Für das Konzept wird ein Klassendiagramm entworfen. Das Diagramm beinhaltet alle Klassen der Anwendung sowie ihre Schnittstellen zu den umliegenden Klassen.
Das Diagramm setzt sich zusammen aus den in diesem Kapitel beschriebenen Sequenzdiagrammen (5.3, 5.2, 5.1) und den Fachklassen des fachlichen Datenmodells 4.5. Außerdem wird es durch Frameworks und Hilfsklassen ergänzt. Die Interaktion der Klassen in der Abbildung 5.4 wird in dem Unterabschnitt 5.5.1 beschrieben.

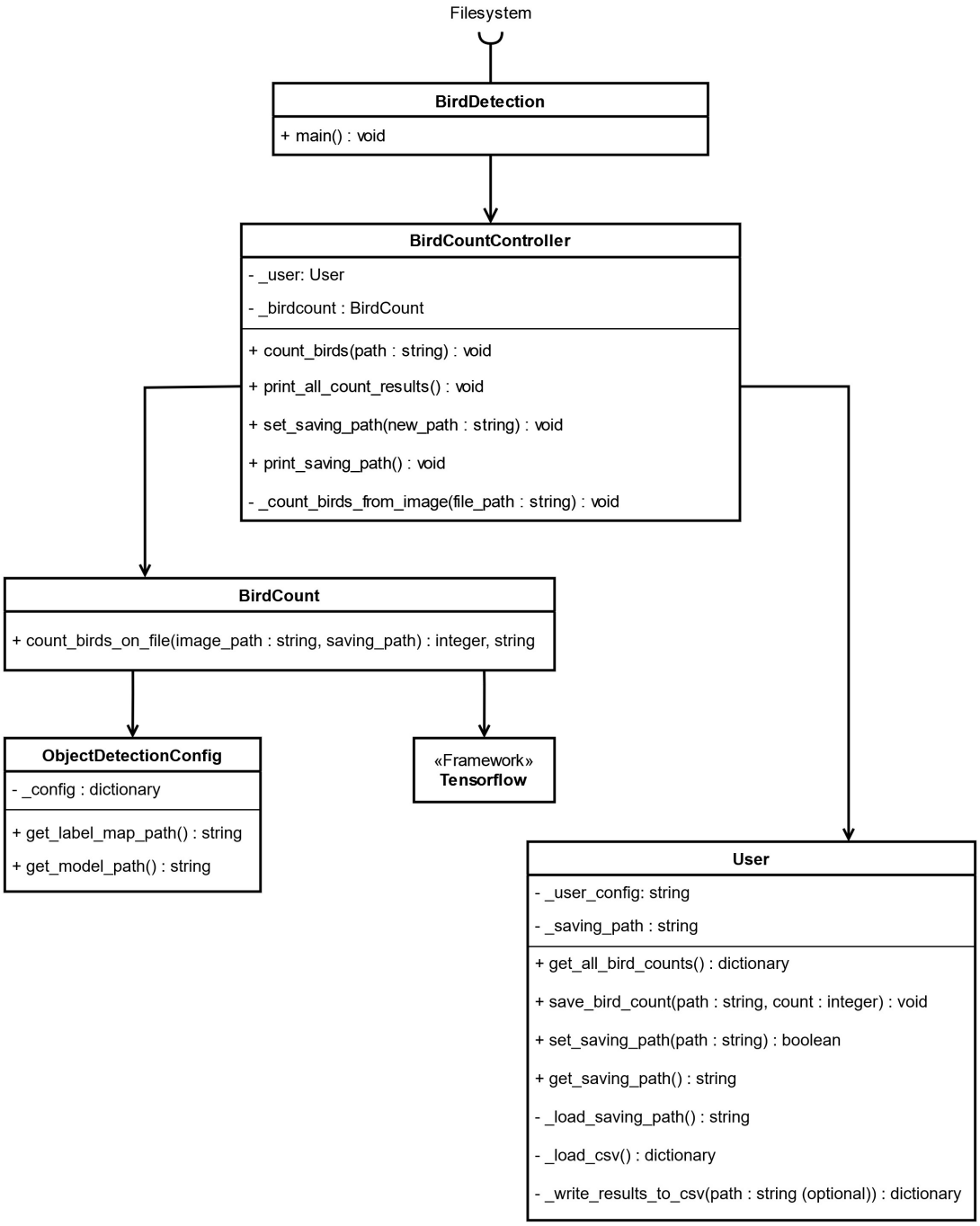


Abbildung 5.4: Klassendiagramm der Anwendung

5.5.1 Systeminteraktion

Der Unterabschnitt beschreibt das Konzept der Systeminteraktion mit dem Benutzter und Interaktionen innerhalb des Systems.

System und Benutzer

Der Benutzer kann über das Starten des Programms durch *BirdDetection* die Anwendungsfälle über die Methoden des *BirdCountController* ausführen. Bei dem *BirdCountController* handelt es sich um eine Steuerungsklasse. Hinsichtlich der Benutzbarkeit (4.2.3) kann der Benutzer die Vogelzählung des Systems nutzen, indem er über die Shell einen Bildpfad oder ganze Ordnerpfade angibt. Anschließend wird dem Benutzer das Ergebnis angezeigt und parallel gespeichert. Die Verwendung der Shell soll die schnelle Verarbeitung der Daten ermöglichen, da nur ein Kommando in der Shell eingegeben werden muss und anschließend in dem Datenverzeichnis weitergearbeitet werden kann.

Interaktionen innerhalb des Systems

Alle drei Anwendungsfälle sind in der Klasse *BirdCountController* repräsentiert, welche über die *Main*-Methode der Klasse *BirdDetection* aufgerufen werden. Während die Klasse *BirdDetection* die eingehenden Daten nur aufbereitet, hat der *BirdCountController* Zugriff auf verschiedene Fachklassen und gibt dem *User* die Ergebnisse aus den Fachklassen heraus. Für die Vogelzählung in der Klasse *BirdCount* wird eine weitere Hilfsklasse *ObjectDetectionConfig* benötigt. Die in der Konfigurationsklasse hinterlegten Pfade zu dem verwendeten neuronalen Netz sind bei Bedarf veränderbar und somit kann die Änderbarkeit (4.2.3) des Systems gewährleistet werden. Diese soll den schnelleren Austausch des neuronalen Netzes erleichtern, um die Ergebnisse der Experimente (7) in die Anwendung mit einzubeziehen. Bei den Pfaden handelt es sich um das trainierte neuronale Netz sowie die Labelwerte der Klassen, welche essenziell für die Objekterkennung sind. Die Objekterkennung wird mit Hilfe des Frameworks *Tensorflow* in der Klasse *BirdCount* durchgeführt. Die Ergebnisse aller erkannten Vögel durch die Klasse *Birdcount* werden in der Fachklasse *User* gespeichert. Die Ergebnishistorie der letzten Vogelzählungen sind in der Klasse *User* gespeicherten Liste über die Steuerungsklasse abzufragen.

6 Realisierung

In diesem Kapitel wird die Realisierung des im letzten Kapitel vorgestellten Konzeptes (5) beschrieben. Dabei werden die nichtfunktionalen Anforderungen (4.2.3) berücksichtigt. Die Implementierungen der Klassen aus dem Klassendiagramm 5.4 beschreiben die kommenden Abschnitte sowie die Tests (6.6) der Implementierung.

6.1 *BirdDetection*: Benutzerinteraktion

Das Starten der Anwendung erfolgt über die Klasse *BirdDetection* mit dem Befehl `BirdDetection.py` und stellt somit die Schnittstelle zwischen Benutzer und Anwendung dar. Dabei decken die zusätzlichen Parameter bei dem Aufruf des Programms die verschiedenen Anwendungsfälle ab. Durch die Speicherung des Bildes inklusive *Bounding Boxes* (2.5.1) und Anzahl der Vögel kann der Benutzer sich vergewissern, dass alle Vögel zuverlässig (4.2.3) gezählt worden sind und ermöglicht anschließend mit den Daten weiterzuarbeiten. Für die Umsetzung des Argumentenparsing wurde durch die von der *Python*-Bibliothek zur Verfügung gestellte *argparse* verwendet. Das Modul gewährt ein benutzerfreundliches Interface (4.2.3) für die Anwendung, indem automatische Hilfsangaben und Verwendungshinweise sowie Fehlermeldung bei falscher Eingabe der Argumenten gegeben werden [8]. Dies ist wichtig in Bezug auf die Fehlertoleranz (4.2.3) der Anwendung.

Je nachdem welches Argument übergeben wird, führt die Klasse *BirdDetection* die verschiedenen Anwendungsfälle über die Klasse *BirdCountController* aus. Dabei werden die von dem Benutzer überreichten Argumente der Methode mitgegeben. Die Befehle der Anwendung sind in der Tabelle 6.1 aufgelistet.

Tabelle 6.1: Bedienung des Programms

Befehl	Beschreibung
`-h, -help`	zeigt die Hilfsausgabe mit der Bedienung der einzelnen Befehle an.
`-count <Pfad>`	gibt die Anzahl der Vögel des Bildpfades wieder. Bei dem Pfad handelt es sich um eine existierende Bilddatei oder um einen existierenden Ordner, welcher Bilddateien beinhaltet.
`-setpath <Pfad>`	legt den angegebenen Pfad als Speicherordner für die Ergebnisse der Vogelzählung fest.
`-getpath`	gibt den Pfad des Speicherordners der Ergebnisse wieder.
`-printresults`	gibt alle Ergebnisse der Vogelzählung wieder, die im Speicherordner hinterlegt sind.

6.2 Identifikation und Zählung von Vögeln

In diesem Abschnitt wird die Implementierung der Vogelzählung des beschriebenen Konzeptes (5.1) beschrieben.

6.2.1 *BirdCountController*: Vogelzählung

Die Klasse lädt im Konstruktor Referenzen auf die Instanzen *BirdCount* und *User*, mit welchen in den Methoden weitergearbeitet wird.

Die Vogelzählung wird über die Methode *count_ birds(path)* des *BirdCountController* ausgeführt. Für die Fehlertoleranz (4.2.3) wird vorerst der übergebene Pfad auf seine Existenz überprüft. Im Zuge dessen wird ermittelt, ob es sich um eine Bilddatei (*.png, .jpg, .jpeg*) handelt oder ob der Ordner Bilddateien enthält. Fehler hinsichtlich der Dateipfade werden über die Standardausgabe von *Python* ausgegeben (4.2.3).

Auf Grund der möglichen Iteration durch die Übergabe eines Ordnerpfades wird bei richtiger Eingabe des Pfades einer Bilddatei eine private Hilfsmethode *_count_birds_from_image(file_path)* aufgerufen, um redundanten Code zu vermeiden. Die Hilfsmethode ruft über die Methode *count_birds_on_file(image_path, saving_path)* der Klasse *BirdCount* die Anzahl der gezählten Vögel sowie den Pfad ab, wo das Bild inklusive der *Bounding Boxes* gespeichert wurde. Die Ergebnisse werden über die Methode *User._save_bird_count(saving_path, count)* abgespeichert.

6.2.2 *BirdCount*

Die Implementierung der Identifikation und Zählung der Vögel erfolgt hauptsächlich über die Methode *count_birds_on_file(image_path, saving_path)* der Klasse *BirdCount*. Dafür importiert die Klasse das Framework *Tensorflow* (3.3), *Numpy* [9], die *Matplotlib* [26], *PIL.Image* [22], von *object_detection.utils visualization_utils* als *vis_utils* und *label_map_util* der *Tensorflow Object Detection API* [27] und die *ObjectDetectionConfig* der eigenen Anwendung.

Für die Speicherung der Ergebnisse des Bildes wird zunächst der Pfad des Speicherordners und der Bildname zusammengefasst und anschließend Pfade des Modells und der *Labelmap* aus der *ObjectDetectionConfig* geladen.

```
1  model_function = tf.saved_model.load(saved_model)
2
3  loaded_labelmap = label_map_util.create_category_index_from_labelmap
4                    (label_map,use_display_name=True)
5
6  image_numpy = np.array(Image.open(image_path))
7  tensor = tf.convert_to_tensor(image_numpy)
8  tensor = tensor[tf.newaxis, ...]
9
10 detections = model_function(tensor)
```

Listing 6.1: *count_birds_on_file*: Objekterkennung starten

Mit der Methode *tf.saved_model.load(export_dir)* (Zeile 1) wird das Modell geladen, das sich an dem vorgegebenen Pfad befindet. Dafür wird der Methode ein Pfad eines Ordners übergeben, in dem nach der Datei *saved_model.pb* gesucht wird und wenn nicht vorhanden eine Fehlermeldung über die Standardausgabe gegeben. Anschließend wird mit *create_category_index_from_labelmap(label_map)* des Modules *label_map_util* (Zeile

3) die *Labelmap*-Datei des Pfades in ein *dictionary* mit Id der Klasse und deren Namen geladen.

In der Zeile 6 wird der übergebene Bildpfad mit der Methode *open* als Bild geöffnet und in ein *Numpy.array* umgewandelt. Dies ist nötig für die darauffolgende Umwandlung in einen Tensor mit *tf.convert_to_tensor* (Zeile 7). Der Tensor ist ein mathematisches Objekt, welches wie ein *Array* aufgebaut ist und für die Weiterarbeitung mit *Tensorflow* benötigt wird [2].

Die am Anfang geladene Funktion des Modells benötigt einen vierdimensionalen Tensor mit Stapelgröße, Höhe des Bildes, Breite des Bildes und die Eigenschaften, welche in dem Fall der Bildverarbeitung Farbkanäle der Pixel sind. Dafür wird mit der Konvention *tf.newaxis* eine Dimension zu dem Tensor hinzugefügt (Zeile 8). Der Tensor kann anschließend der Funktion übergeben werden. Bei dem Ergebnis handelt es sich um die Erkennung der Objekte in dem Bild (Zeile 10).

Als Rückgabewert der Methode von *BirdCount* werden die Anzahl und der neue Pfad des Bildes inklusive der *Bounding Boxes* der gefundenen Vögel benötigt. Dafür muss das Ergebnis der Objekterkennung des Bildes gefiltert werden.

```
1  for _, val in enumerate(loaded_labelmap.items()):
2    if val[1]["name"] == "bird":
3      bird_index = val[1]["id"]
4
5  detections['detection_boxes'] = detections['detection_boxes'][0].numpy()
6  detections['detection_scores'] = detections['detection_scores'][0].numpy()
7  detections['detection_classes'] =
8    detections['detection_classes'][0].numpy().astype(np.int64)
9  detection_boxes=[]
10 detection_classes=[]
11 detection_scores=[]
12
13 for index, val in enumerate(detections['detection_classes']):
14   if detections['detection_scores'][index] > 0.5 and val == bird_index:
15     detection_boxes.append(detections['detection_boxes'][index])
16     detection_classes.append(detections['detection_classes'][index])
17     detection_scores.append(detections['detection_scores'][index])
```

Listing 6.2: *count_birds_on_file*: Vogelerkennungen filtern

Damit nur Objekte der Klasse Vogel gezeigt werden, muss die Id des Vogels gefunden werden (Zeile 1).

Für die Speicherung der Ergebnisse werden die Koordinaten der *Bounding Boxes* und die dazugehörigen *Scores* und Klassen benötigt. Dafür wird die erste Dimension der Schlüssel *detection_ boxes*, *detection_ scores* und *detection_ classes* von *detections* entnommen. Die Verwendung der Daten des ersten Index sorgt dafür, dass die Stapeldimension (siehe 6.1) wieder entfernt wird. Für eine einfachere Verarbeitung werden die *detections* zurück in ein *numpy.array* umgewandelt und die Klassenwerte mit *astype(np.int64)* in *Integer* umgewandelt (Zeilen 5-8). Alle Ergebnisse der drei *Arrays* sind nur weiter zuverarbeiten, wenn der Index von *detection_ scores* größer als 0.5 ist und *val == bird_ index* (Zeile 14). Dabei werden nur Ergebnisse der Erkennung verwendet bei denen die Wahrscheinlichkeit der Klassifizierung größer als 50% (0.5) beträgt und es sich dabei um die Klasse „Vogel“ handelt.

Die gefilterten Ergebnisse werden an *visualize_ boxes_ and_ labels_ on_ image_ array* des Modules *vis_ utils* übergeben, sowie das Bild als *numpy.array*, die geladene *Labelmap*, und der Parameter *use_ normalized_ coordinates=True*, um die Reihenfolge der Koordinaten nach der Norm *[ymin, xmin, ymax, xmax]* zu bestätigen. Die *Bounding Boxes* werden in dem *image_ numpy* gespeichert.

```
1  vis_utils.visualize_boxes_and_labels_on_image_array(
2              image_numpy,
3              np.array(detection_boxes),
4              np.array(detection_classes),
5              np.array(detection_scores),
6              loaded_labelmap,
7              use_normalized_coordinates=True)
8
9          plt.figure(figsize=(12, 8), dpi=200)
10         plt.axis("off")
11         plt.imshow(image_numpy)
12         plt.savefig(saving_path)
```

Listing 6.3: *count_ birds_ on_ file*: Bildergebnis erstellen

Matplotlib.pyplot als *plt* kann in *Python* Grafiken und Visualisierungen erstellt werden. *Plt.figure* erstellt zunächst eine neue Abbildung mit der Größe des Bildes in Zoll, der Auflösung (*dpi*) und mit *plt.axis(„off“)*, wodurch optionale Achsen in der Abbildung nicht abgebildet werden. Mit der Methode *plt.imshow* wird das *image_ numpy* als Bild in die Abbildung geladen. Die Abbildung wird mit *plt.savefig* an dem übergebenen Speicherort mit dem Bildnamen abgespeichert. Die Methode *count_ birds_ on_ file* gibt die Anzahl der gefundenen Vögel zurück, sowie den gesamten Pfad des neu gespeicherten Bildes.

6.2.3 Konfiguration des Modells

Für die Objekterkennung benötigt die Klasse *BirdCount* sowohl ein Modell als auch die dazu gehörigen *Label* einer *Labelmap*. Die beiden Dateien können sich je nach Wahl des Modells der Anwendung ändern. Die Anforderung einer einfachen Konfiguration dieser Variablen wird in dem Unterabschnitt 4.2.3 festgelegt. Für die Austauschbarkeit in der Anwendung wird eine Konfigurationsklasse *ObjectDetectionConfig* bereitgestellt. Die beiden Pfade zu den Dateien sind als *String* in Variablen in einem *dictionary* hinterlegt und können über *get*-Methoden der Klasse abgerufen werden. Falls der Pfad nicht existiert, wird das System beendet und es wird eine Fehlermeldung ausgegeben. Die Pfade können nicht von dem Benutzer angepasst werden, das heißt die Konfiguration der Pfade liegt aufseiten des Entwicklers.

6.3 Speicherungsvorgang der Ergebnisse

Die Ergebnisse werden über die Methode *_save_bird_count(saving_path, count)* der Klasse *User* abgespeichert. Die *User*-Klasse lädt im Konstruktor der Klasse zunächst den Pfad zu der *_user_config* (6.5.2) und den dort abgespeicherten Pfad des aktuellen Speicherordners (6.5.3). Für die Speicherung der Werte wird ein *DataFrame* der Bibliothek *Pandas* verwendet. *Pandas* ist eine Bibliothek für *Python*-Anwendung, welche die Erstellung von einfach zu verwendenden Datenstrukturen ermöglicht und Analysewerkzeuge bereitstellt [17]. Die Bibliothek wird für die Umwandlung zwischen einer *csv*-Datei und einem *DataFrame* in weiteren Teile der Arbeit genutzt.

Dafür wird zunächst die Hilfsmethode *_load_csv* (6.3) aufgerufen, welche die Ergebnisse aus der gespeicherten *birdCountResults.csv* (6.3) lädt.

Sofern der Pfad des neu abzuspeichernden Bildes nicht in den Pfaden des zurückgegebenen *DataFrame* (*DataFrame[„Image Path“].values*) vorhanden ist oder das *DataFrame* leer ist, wird der Pfad dem *DataFrame* hinzugefügt. Für das Hinzufügen wird ein neues *DataFrame* mit dem neuen Pfad und der Vogelanzahl erstellt und anschließend mit der *Pandas* Methode *concat* mit den anderen Ergebnissen vereint.

Sollte der Pfad bereits in der *csv*-Datei existieren, werden die Ergebnisse der Datei sowie die Bildergebnisse überschrieben. Mit Hilfe der Methode *loc* von *Pandas* kann an der Stel-

le, wo der Pfad identisch ist, die Anzahl ausgetauscht werden und mit der Hilfsmethode *_write_results_to_csv* als *birdCountResults.csv* (6.3) gespeichert werden.

birdCountResults.csv

Die Ergebnisse der Vogelzählung werden in der Datei *birdCountResults.csv* gespeichert, welche sich in dem Speicherordner befindet und bei erster Vogelzählung erstellt und erweitert wird. Bei einer *csv*-Datei („*Comma separated values*“) handelt es sich um eine Textdatei für geordnete Daten. Die Datei verfügt über die Spalten *Counted Birds*, der Anzahl der Vögel des Bildes und *Image Path*, mit dem Pfad des Bildes inklusive der *Bounding Boxes*.

_load_csv

Die *birdCountResults.csv* (6.3) wird über die private Methode *_load_csv* der *User*-Klasse geladen. Der Pfad der Datei wird auf seine Existenz überprüft. Anschließend wird die Datei mit der *Pandas* gestellten Methode *read_csv* in ein *DataFrame* umgewandelt. Alle Bildpfade des geladenen *DataFrame* werden nur behalten, wenn diese existieren. Dies soll verhindern, dass *birdCountResults.csv* Ergebnisse beinhaltet, wo das dazugehörige Bild nicht vorhanden ist. Das *DataFrame* wird am Methodenende wiedergegeben. Falls keine *birdCountResults.csv* in dem Speicherordner vorhanden ist, wird ein leeres *DataFrame* zurückgegeben.

_write_results_to_csv

Durch die Methode *_write_results_to_csv* (6.3) wird die existierende Datei von *birdCountResults.csv* (6.3) mit dem neuen *DataFrame* durch die Methode *to_csv* überschrieben. Die Zeilen sollen keine Nummerierung haben, weshalb die Methode *to_csv* den Parameter *index=false* übergeben bekommt.

6.4 Alle Ergebnisse einsehen

In diesem Abschnitt wird die Implementierung der Ausgabe der Ergebnishistorie beschrieben, welche in dem Abschnitt 5.2 vorgestellt wurde.

6.4.1 *BirdCountController*

Die Ergebnisausgabe wird über die Methode *print_all_count_results()* des *BirdCount-Controller* ausgeführt. Dort wird über die Instanzvariable *_user* die Methode *get_all_-bird_counts()* der Klasse *User* aufgerufen. Die Methode gibt alle Ergebnisse der gespeicherten Vogelzählung des aktuellen Speicherordners aus der Klasse *User* wieder. Bei der Ausgabe der Methode handelt es sich um ein *DataFrame* von der Bibliothek *Pandas*. Das *DataFrame* wird auf Inhalt geprüft und über die Standardausgabe ausgegeben. Ansonsten erhält der Benutzer die Nachricht, dass keine Ergebnisse hinterlegt sind.

6.4.2 *User*

Die Methode *get_all_bird_counts()* von *User* gibt alle Ergebnisse der Vogelzählungen wieder. Dafür wird ebenfalls die Hilfsmethode *_load_csv()* (6.3) verwendet. Der zurückgegebene *DataFrame* der Hilfsmethode wird von der *User*-Methode zurückgegeben.

6.5 Neuen Speicherordner für die Ergebnisse festlegen

In diesem Abschnitt wird die Implementierung zur Festlegung eines neuen Speicherordners beschrieben, welche in dem Abschnitt 5.3 konzipiert wurde.

6.5.1 *BirdCountController*

Die Methode *set_saving_path* des *BirdCountController* setzt den neuen Pfad des Speicherordners. Dafür wird die Methode *set_saving_path* der Klasse *User* aufgerufen. Der Erfolg der Durchführung wird über die Standardausgabe zurückgegeben.

Für die Einsicht des aktuellen Pfades des Speicherordners kann mit dem *BirdCount-Controller* die Methode *print_saving_path* aufgerufen werden. Der Pfad wird über die Methode *get_saving_path* der *User*-Klasse abgefragt und über die Standardausgabe wiedergegeben.

6.5.2 *user_ config.csv*

Der Pfad des Speicherordners der Vogelzählung wird in einer *csv*-Datei festgehalten. Dies sorgt dafür, dass der Benutzer des Systems nach erneutem Starten des Programms den zuletzt verwendeten Speicherort weiterverwenden kann. Der Pfad wird in der Spalte *savingpath* hinterlegt.

6.5.3 Pfad des Speicherordners laden

Die Hilfsmethode der *User*-Klasse *_ load_ saving_ path* öffnet die *user_ config.csv* (6.5.2) und liest sich den Pfad der Spalte *savingpath* heraus und gibt diesen wieder.

6.5.4 *User*

Für die Fehlertoleranz (4.2.3) der Methode *set_ saving_ path* der Klasse *User*, wird zunächst überprüft, ob der Pfad des neuen Ordners existiert. Bei einem nicht existierenden Pfad gibt die Methode False wieder. Anschließend wird die Datei überschrieben mit dem Spaltennamen und dem neuen Pfad und gibt True wieder.

Für die Rückgabe des aktuellen Speicherordners verwendet die Klasse *User* die Methode *get_ saving_ path*. Die Methode gibt die Instanzvariable mit dem aktuellen Pfad wieder.

6.6 Tests

Um die Erfüllung der Anforderungen an das System aus der Sicht des Benutzers zu testen, werden Testfälle konstruiert. Die Testfälle werden aus den in der Anforderungsanalyse beschriebenen Aktivitätsdiagramme (4.2.2) abgeleitet. Die Testfälle werden in der Klasse *TestBirdDetection* durch verschiedene Testmethoden abgedeckt. Jede Testmethode wird mit dem Framework `pytest` [13] ausgeführt und eine Setup-Methode für die Pfadbestimmung der Tests, um die Importprobleme von *Python* zu umgehen, vorangestellt. Eine Teardown-Methode, um gespeicherte Ergebnisse der Vogelzählungen wieder zu löschen, wird anschließend ausführt. Da die Ergebnisse über die Standardausgabe erfolgen, wird auf die Ausgaben mit Hilfe der Übergabe des Parameters *capsys* zugegriffen. Die Methode *capsys.readouterr().out* liest und vergleicht die Ausgaben mit den zu erwartenden Werten. Die Tabelle 6.2 zeigt die verschiedenen Testfälle der Anwendung.

Tabelle 6.2: Testfälle

Methode	Beschreibung
test_setpath	prüft, ob der neue Speicherordner gesetzt werden kann.
test_setpath_notexisting	prüft, ob kein falscher Speicherordner angegeben werden kann.
test_getpath	prüft, ob der Speicherordner abgerufen werden kann.
test_printresults	prüft die korrekte Ausgabe aller Ergebnisse der Vogelzählungen.
test_count_pathnotexisting	prüft die Übergabe eines nicht existierenden Pfades für die Vogelzählung.
test_count_textfile	prüft die Übergabe eines falschen Dateityps für die Vogelzählung.
test_count_folder	prüft den Vorgang der Vogelzählung anhand eines Bildes.
test_count_image	prüft den Vorgang der Vogelzählung anhand eines Ordners

Für die Testfälle der Vogelzählung wurde ein beliebiges neuronales Netz von *Tensorflow* verwendet. Die Wahl der Modelle wird in dem nächsten Kapitel 7.1 näher erläutert und eine endgültige Auswahl des Modells durch Experimente festlegt.

7 Experimente

Für die Anwendung der Vogelerkennung wird ein neuronales Netz bzw. Modell benötigt. Da die Qualität des Modells die Ergebnisse der Vogelerkennung bestimmt, beschreibt dieses Kapitel verschiedene Experimente, welche die Bestimmung und Optimierung des Modells ermöglichen sollen. Dabei sollen die Experimente die nichtfunktionalen Anforderungen (4.2.3) in Bezug auf Zuverlässigkeit und Effizienz berücksichtigen. Die Experimente werden in ihrem Aufbau ihrer Durchführung beschrieben und für weiterführende Experimente im Kapitel ausgewertet.

7.1 Modelle

Die Experimente benötigen den Einsatz verschiedener Modelle, welche in diesem Abschnitt vorgestellt werden. Dabei beruhen die Architekturen der Modelle auf den Kriterien, welche in dem Unterabschnitt 5.1.1 des Konzeptes (5) festgelegt worden sind. Bei den fünf ausgewählten Modellen handelt es sich um vortrainierte neuronale Netze aus dem *Tensorflow 2 Detection Model Zoo* [28]:

- *Faster R-CNN ResNet50 V1 1024x1024*
- *Faster R-CNN ResNet101 V1 1024x1024*
- *Faster R-CNN ResNet152 V1 1024x1024*
- *Faster R-CNN Inception ResNet V2 1024x1024*
- *Mask R-CNN Inception ResNet V2 1024x1024*

Während alle fünf Modelle die Eigenschaften besitzen, auf eine Bildgröße bis zu 1024x1024 Pixel ausgelegt und mit Hilfe des Datensatzes von *MS COCO 2017* [6] vortrainiert zu sein, unterscheiden sie sich vor allem in ihrer Architektur. Alle Modelle baut auf einer

Faster RCNN-Architektur auf, kombinieren diese jedoch mit unterschiedlichen anderen Architekturen.

Die Modelle *Faster R-CNN ResNet50 V1*, *Faster R-CNN ResNet101 V1* und *Faster R-CNN ResNet152 V1* erweitern die *Faster R-CNN*-Architektur mit der ersten Version der *Resnet*-Architektur (2.4.2). Die drei Modelle unterscheiden sich lediglich in ihrer Tiefe des *Resnet*-Modells, indem sie unterschiedlich viele Schichten vorweisen. Die Modelle sind nach ihrer Anzahl der *Resnet*-Schichten benannt: *Faster R-CNN ResNet50 V1* mit 50 Schichten, *Faster R-CNN ResNet101 V1* mit 101 Schichten. *Faster R-CNN ResNet152 V1* ist das größte Netz mit 152 Schichten.

Das vierte Modell *Faster R-CNN Inception ResNet V2* verwendet die zweite Version der Inception-Architektur (2.4.3), welche durch Residualblöcke (2.4.2) erweitert wurde.

Das fünfte Modell *Mask R-CNN Inception ResNet V2*, verwendet die gleiche Architektur wie das vierte Modell. Hier wird allerdings die Erweiterung der *Faster RCNN*-Architektur eingesetzt: *Mask R-CNN* (3.2.2).

7.2 Experiment 1: Vergleich der Modelle durch manuelles Testen

Dieses Experiment vergleicht die 5 verschiedenen Modelle aus dem Abschnitt 7.1. Der Vergleich erfolgt durch ausgewählte Bilder. Dabei soll sich zeigen, welches der Modelle am geeignetsten für die Erkennung von Vögeln in Vogelschwärmen ist.

7.2.1 Aufbau

Neben den Modellen wird eine Auswahl an Bildern von Vogelschwärmen getroffen. Sie zeigen Vogelschwärme mit gleichen und/oder unterschiedlichen Arten von Vögeln und/oder anderen Objekten auf einem Bild. Da sich diese Arbeit auf die Zählung von Vögeln an der Küste konzentriert, ist die Erkennung eines im Wasser gespiegelten Vogels interessant, weil sich die Frage stellt, ob das Modell die Spiegelung des Vogels möglicherweise mitzählt. Zu den Bildern werden die exakten Koordinaten der tatsächlichen *Bounding Boxes* (2.5.1) jedes Vogels manuell per Hand mit Hilfe eines Labeling Programms *LabelImg* [15] bestimmt. Für die Vogelzählung wird das in dieser Arbeit entwickelte System verwendet und über *Google Colab* (5.4.3) ausgeführt. Dabei wird der Laufzeittyp *GPU*

Tesla V100-SXM2-16GB verwendet, der in den nachfolgenden Experimenten ebenfalls verwendet wird.

7.2.2 Ablauf

Jedem der fünf Modelle werden die gleichen sieben Bilder übergeben. Nach dem Durchlauf folgt ein Vergleich aller Bilder inklusive der vorhergesagten *Bounding Boxes*. Die Bestimmung der falsch-positiven und richtig-positiven Werte ergibt sich aus der Berechnung des *IoU* (2.8.4) jeder *Bounding Box*. Für den IoU werden die vorhergesagten Koordinaten der *Bounding Boxes* des Modells und die Koordinaten des *Ground Truth* verwendet. Die fünf Modelle werten je sieben verschiedene Bilder aus, von denen je Modell drei ausgewertete Bilder in dieser Arbeit gezeigt werden.

7.2.3 Ergebnisse

Das erste Bild A.1.1 zeigt achtzehn erkennbare Vögel, die sich teilweise gegenseitig verdecken und somit nur teilweise ihre Körper erkennen lassen. Hier soll überprüft werden, ob das Modell nicht nur Vögel in ihrer Gänze erkennt, sondern diese auch anhand einzelner Teile ihrer Körper identifizieren kann. Auf dem Bild ist zu sehen, dass das *Faster R-CNN Inception ResNet V2* (a) und *Mask R-CNN Inception ResNet V2* (e) beide mit 16 von 18 gezählten Vögeln am meisten gezählt haben, von denen wiederum 15 richtig-positiv waren.

Das zweite Bild A.1.2 zeigt voneinander frei stehende Vögel, die sich in der Wasseroberfläche spiegeln. Alle Modelle außer dem *Faster R-CNN ResNet152 V1* (A.2d) und dem *Mask R-CNN Inception ResNet V2*-Modell (A.2e) haben diese Vögel richtig erkannt. Das Bild von dem *Mask R-CNN*-Modell (A.2e) zeigt durch die eindeutige *Bounding Box* um die Spiegelung des einzelnen Vogels, dass das Modell eine Spiegelung eines Vogels ebenfalls als Vogel erkennt.

Da nicht nur Vögel auf den Bildern vorhanden sein können, ist sicherzustellen, dass das Modell nicht alle gefundenen Objekte als Vögel kategorisiert. Das dritte Bild A.1.3 zeigt das übereinstimmende Ergebnis der Vogelzählungen aller Modelle, in dem Menschen am Strand ohne *Bounding Boxes* zusehen sind. Somit ist auszuschließen, dass die Modelle die Menschen auf dem Bild als Vögel kategorisieren.

Die Tabelle 7.1 zeigt die gesamte Auswertung der sieben Bilder für die fünf verschiedenen Modelle. Die Tabelle zeigt den Mittelwert aller *IoU*-Werte (2.8.4) der Modelle. Es ist zu sehen, dass das *Faster R-CNN Inception ResNet V2* im Durchschnitt die größte Überschneidung (0,84) mit seinen vorgesagten *Bounding Boxes* und der *Ground Truth* hat. Daraus lässt sich schließen, dass das Modell die Vögel und die Pixel, welche sie darstellen, am genauesten gefunden hat.

Aus den falsch-positiv und richtig-positiv Werten der Bilder wurde die Sensitivität (2.8.2) und der PVW (2.8.1) der Vorhersagen der Modelle berechnet. Die Sensitivität zeigt hier, dass das *Faster R-CNN Inception ResNet V2* mit 0,85 am meisten richtig-positive Objekte gefunden hat. Die wenigsten falsch-positiven Fälle hat das *Faster R-CNN ResNet101 V1* mit 0,98. Sowohl die Sensitivität als auch der PVW spielen eine Rolle bezüglich der Auswahl des Modells für die Vogelzählung. Es sollen möglichst viele Vögel mit einer möglichst geringen Fehlerquote gefunden werden. Dafür wird der *F1-Score* (2.8.3) bestimmt. Hierbei liegt der Wert bei dem *Faster R-CNN Inception ResNet V2* mit 0,9 am höchsten.

Tabelle 7.1: Auswertung manueller Durchläufe

	IoU (Mittelwert)	Sensitivität	PVW	*F1-Score*	Zeit je Bild
Faster R-CNN ResNet50 V1	0,8100	0,7288	0,9348	0,8190	24s
Faster R-CNN ResNet101 V1	0,8327	0,7627	**0,9783**	0,8571	31s
Faster R-CNN ResNet152 V1	0,8327	0,6780	0,9524	0,7921	38s
Faster R-CNN Inception ResNet V2	**0,8426**	**0,8475**	0,9615	**0,9009**	49s
Mask R-CNN Inception ResNet V2	0,8397	0,8136	0,9231	0,8649	55s

Generell ist zu sehen, dass das *Faster R-CNN ResNet V1* mit 101 Schichten im Vergleich zu 50 und 152 Schichten die besten Werte erzielt. Das Modell sowie das *Faster R-CNN Inception ResNet V2* haben beide die führenden Werte in unterschiedlichen Metriken. Die

Werte des *Mask R-CNN Inception ResNet V2* befinden sich im mittleren bis schlechteren Bereich. Außerdem ist sichtbar, dass das Modell sehr schlecht mit Spiegelung von Vögeln im Wasser umgehen kann.

Alle fünf Modelle erfüllen die nichtfunktionale Anforderung der Effizienz (4.2.3), da sie für die Auswertung eines Bildes nicht länger als eine Minute brauchen.

7.3 Experiment 2: Vergleich der Modelle durch einen Evaluierungsdatensatz

Das zweite Experiment vergleicht die beiden Modelle aus dem Experiment 7.2, die am besten abgeschnitten haben. Dafür werden die Modelle *Faster R-CNN Inception ResNet V2* und das *Faster R-CNN ResNet101 V1* miteinander verglichen.

Für die Validierung der Ergebnisse aus dem ersten Experiment soll das Verhalten der Modelle auf eine größere Anzahl von Vogelbildern getestet werden, um daraus schließen zu können, welches Modell am besten für die Vogelzählung geeignet ist.

7.3.1 Aufbau

Für das Experiment werden die Testdaten des Datensatzes aus dem Abschnitt 5.1.2 verwendet. Die Evaluierung soll mit Hilfe der *Tensorflow API* (3.3) mit dem *Python*-Skript `models/research/object_detection/model_main_tf2.py` durchgeführt werden. Das Skript erhält den aktuellen Stand des Modells, die Konfigurationsdatei mit den Pfaden der Testdaten, den Ergebnisspeicherpfad und den Parameter `-run_once=True`, damit nur die Evaluierung durchgeführt wird.

7.3.2 Ablauf

Die Modelle machen mit den Testdaten jeweils einen Validierungsdurchlauf von *Tensorflow*, um zu zeigen, wie gut das Modell auf die Daten anzuwenden ist. Dabei vergleicht das Skript die vorhergesagten *Bounding Boxes* mit der *Ground Truth* des Datensatzes und wertet diese aus. Anschließend werden die Ergebnisse der Evaluationsdurchläufe aufgezeigt und mit den Ergebnissen aus dem Experiment 7.2 verglichen.

7.3.3 Ergebnisse

Die Tabelle 7.2 zeigt die Ergebnisse der Evaluationsdurchläufe der beiden Modelle. Wie zu sehen ist, liegt sowohl die Sensitivität wie auch der PVW bei beiden Modellen deutlich niedriger als in dem ersten Experiment (7.1). Dies zeigt, dass die Modelle in Bezug auf die Vogelzählung noch optimierbar sind. Trotzdem liegt die Sensitivität des *Faster R-CNN Inception ResNet V2* weiterhin mit 0,4984 am höchsten. Interessanter ist der PVW, welcher im Experiment 7.2 bei dem *Faster R-CNN ResNet101 V1* am höchsten lag. Im Evaluationsdurchlauf wird dieser ebenfalls von dem *Faster R-CNN Inception ResNet V2* übertroffen, mit einem Wert von 0,5692. Da der Verlust möglichst minimiert werden soll (2.6.3), zeigt auch hier das *Faster R-CNN Inception ResNet V2* den besseren Wert.

Tabelle 7.2: Auswertung der Evaluationsdurchläufe

	Sensitivität	PVW	Verlust
Faster R-CNN ResNet101 V1	0,4821	0,5242	**0,9597**
Faster R-CNN Inception ResNet V2	**0,4984**	**0,5692**	0,9613

Durch *Tensorflow* werden Bildausschnitte des Datensatzes 5.1.2 des Evaluationsdurchlaufes gezeigt. Dabei ist zu sehen, dass die Modelle viele Objekte finden, sie jedoch nicht immer als „Vogel“ kategorisieren. Da die Modelle mit dem Datensatz von *MS COCO* [6] vortrainiert wurden sind, kennen die Modelle auch andere Klassifizierungen. Zum Beispiel erkennt das *Faster R-CNN ResNet101 V1* einige Vögel als „Kite“, die von dem anderen Modell richtig als Vogel eingeordnet werden.

Mit den Ergebnissen aus dem ersten Experiment 7.2 und den Ergebnissen dieses Experimentes wird in den kommenden Experimenten nur noch mit dem *Faster R-CNN Inception ResNet V2* gearbeitet, da dieses Modell die besten Werte für die Vogelzählung erzielt.

7.4 Experiment 3: Training des Modells

In diesem Experiment werden mit den Ergebnissen aus Experiment 7.2 und Experiment 7.3 weitergearbeitet. Die Experimente haben ergeben, dass das *Faster R-CNN Inception ResNet V2* am besten geeignet ist für die Zählung von Vögeln in Vogelschwärmen. Die Ergebnisse aus den Experimenten 7.2 und 7.3 zeigen, dass hinsichtlich der Zuverlässigkeit (4.2.3) das Modell nicht den vordefinierten Wert des *F1-Scores* (2.8.3) von 0.95 erzielen konnte. Um die Vogelerkennung weiter zu verbessern, soll das Modell durch eine Feinabstimmung (2.7) optimiert werden, in dem es mit einem Datensatz mit Vogelschwärmen weitertrainiert wird.

7.4.1 Aufbau

Für das Training des Modells wird der Datensatz von [3] verwendet und mit Hilfe des Skriptes `models/research/object_detection/model_main_tf2.py` der *Object Detection API* von *Tensorflow* durchgeführt. Die geeigneten Trainingsparameter wurden bereits in dem Abschnitt 5.1.3 bestimmt und werden über die Trainingsdatei *pipeline.config* gesetzt. Dafür wird die *pipeline.config* des vortrainierten Modells verwendet und angepasst.

Die Funktion *tf.nn.sigmoid_cross_entropy_with_logits()* von *Tensorflow* vereinbart die Sigmoid-Funktion (2.6.1) als Aktivierungsfunktion und die binäre Kreuzentropie-Funktion als Verlustfunktion (2.6.3). Sie kann über die *pipeline.config* angegeben werden (A.2.1).

Die Losgröße wurde auf 2 gesetzt, da die *GPU* nicht genügend Rechenleistung zur Verfügung stellt. Für das Training wurden 25.000 Trainingsschritte festgelegt, in denen die Lernrate *Cosine Decay* von *Tensorflow* verwendet wird. Dabei handelt es sich um eine Lernrate mit linearer Aufwärmphase. In der Aufwärmphase startet die Lernrate bei 0 und wird bist 0.008 erhöht. Die Phase sorgt dafür, dass die Gewichtungen sich zunächst in die richtige Richtung anpassen. Ab 5.000 Schritten kann die Lernrate, für eine feinere Anpassung der Gewichtung gesenkt werden (zusehen in A.2.1).

7.4.2 Ablauf

Das Training wird so lange durchgeführt, bis der Verlustwert kleiner als 0.1 ist. Das resultierende Modell wird anschließend mit den gleichen Bildern und Kriterien aus Experiment 7.2 und Experiment 7.3 geprüft.

7.4.3 Ergebnisse

Die Tabelle 7.3 zeigt, wie die Lernrate und der Verlust bei Trainingsende aussieht.

Tabelle 7.3: Trainingsdaten: *Faster R-CNN Inception ResNet V2* trainiert mit *AU Bird*

Lernrate	0,0019
Verlust	0,099
Trainingszeit	2,5 Stunden

Der Evaluationsdurchlauf, wie in Experiment 7.3 durchgeführt zeigt, dass die Sensitivität und der PVW (in Tabelle 7.4) besser geworden ist als vor der Feinabstimmung. Anders sieht es bei dem Verlustwert aus, welcher sich verschlechtert hat.

Tabelle 7.4: Auswertung des Evaluationsdurchlaufes

Sensitivität	0,541285667
PVW	0,677175167
Verlust	1

Nach den manuellen Tests des Modells (wie in Experiment 7.2) ist vor allem zu sehen, dass durch das Training mit dem Datensatz (5.1.2) nicht nur Vögel als solche erkannt werden, sondern auch andere Objekte wie Menschen und Formen im Hintergrund.

Tabelle 7.5: Auswertung der manuellen Tests

IoU (Mittelwert)	0,7391
Sensitivität	0,7458
PVW	0,6286
F1-Score	0,6822
Auswertungszeit je Bild	45s

Um zu testen, ob die vielen falsch-positiven Erkennungen möglicherweise mit der Wahl des Datensatzes zusammenhängen, wurden die selben Bilder aus dem Experiment 7.2 mit der Anwendung [1] von [4] ausgewertet. Als Ergebnis zeigt sich, dass ebenfalls Menschen und andere Formen im Hintergrund als „Vogel“ klassifiziert werden. In dem Artikel wurden die meisten falschen Vogelerkennungen auf Objekte beschrieben, die eine ähnliche Erscheinung wie der Vogel haben [4, 18]. In diesem Experiment zeigt sich jedoch, dass auch Objekte wie der Mensch als Vogel erkannt werden, der kaum Ähnlichkeiten mit dem Vogel aufweist. Für die binäre Klassifikation werden die Objekte in Klasse oder nicht Klasse bzw. als Hintergrund eingeordnet. Der verwendete Datensatz (5.1.2) beinhaltet viele Bilder, auf denen Vögel am Himmel fliegen, weniger Vögel die sich vor einem Bodenhintergrund befinden, aber kein Bild zeigt andere Objekte als Vögel. Dadurch kann die Merkmalsunterscheidung der Objekte, für „Vogel“ oder „kein Vogel“ schlechter durchgeführt werden. Dies könnte den Anstieg des Verlustes (7.4) erklären, sowie die verschlechterten Evaluationsmetriken der Tabelle 7.5 im Vergleich zu den Werten aus den vorherigen Experimenten (7.1 und 7.2) zeigt.

8 Diskussion und Ausblick

In diesem Kapitel werden die Experimente (7) hinsichtlich der Aussage der Thesis diskutiert und ausgewertet (8.1). Anschließend wird eine Zusammenfassung der Arbeit (8.2) und ein möglicher Ausblick zu der Thematik (8.3) gegeben.

8.1 Diskussion

Die Thesis dieser Arbeit fordert eine automatisierte Objekterkennung und -zählung von Vögeln in Schwärmen mittels maschinellen Lernens. Die Möglichkeiten der Umsetzung einer Automatisierung dieser Aufgabe wurden durch die Auswahl von Methoden des aktuellen technischen Entwicklungsstandes erarbeitet und ihre Grenzen durch die Experimente aufgezeigt. Die in dem Kapitel 7 beschriebenen Experimente ergeben, dass die automatisierte Vogelzählung mit den zur Zeit verfügbaren Mitteln begrenzt ist. Anhand der Ausarbeitung aktueller Techniken zur Erkennung von Vögeln zeigten sich verschiedene Ansätze zur Auswahl an Modellen. Zur Optimierung der Modelle wurden Funktionen für die anwendungsfallspezifische binäre Klassifizierung bestimmt.
Das vortrainierte *Faster R-CNN Inception Resnet v2*-Modell behauptete sich gegenüber anderen Modellen in den ersten beiden Experimenten (zusehen in 7.2, 7.3) auf Grund seiner besseren Werte in den Evaluierungsmetriken und den Veranschaulichungen dieser Ergebnisse durch Bildbeispiele. Anhand der in dieser Arbeit ermittelten Werte wurde festgestellt, dass auch dieses Modell nicht den Anforderungen des Benutzers (4.2.3) standhalten kann.
Bilder eines Vogelschwarms zeigen vor allem Vögel, welche sich teilweise überdecken. Es können deshalb nur einzelne Merkmale der Vögel wie Schnäbel oder Flügel für die Zählung ausgewertet werden. Das Modell erkennt zwar teilweise hintereinanderstehende Vögel anhand ihrer Merkmale, arbeitet jedoch zuverlässiger wenn alle vordefinierten Merkmale eines Vogels dargestellt sind. Dies zeigte sich ebenfalls in einer Evaluierungsmetrik, die den angestrebten Anforderungswert nicht erreichen konnte.

Für das Erkennen eines Vogels anhand unvollständiger Merkmale wurde das Modell im dritten Experiment (7.4) weitertrainiert. Resultat: auch der verwendete Datensatz [3] in Kombination mit den binären Funktionen für das Training des Modells reichte nicht aus, um eine binäre Klassifizierung von „Vogel“ oder „kein Vogel“ zu erlernen.

Das trainierte Modell weist zwar eine verbesserte Sensitivität und PVW auf als vor dem Training. Dies zeigt, dass die Vögel des Testdatensatzes in dem neu trainierten Modell besser erkannt worden sind. Durch das Training mit Hilfe des Trainingdatensatzes ist das Modell besser auf den dazugehörigen Testdatensatz abgestimmt. Der erhöhte Verlustwert deutet jedoch auf eine allgemeine Verschlechterung des Modells hin. Die Ursache dafür wurde im Ungleichgewicht der Klassen des Datensatzes erkannt, da der Datensatz nur Bilder von Vögeln enthält und keine anderen Objekte die in dem Lebensraum des Vogels vorkommen können. Das Modell kennt deshalb nicht ausreichend viele Merkmale zur Unterscheidung, um Objekte den Klassen richtig zuzuordnen. Dadurch konnte es passieren, dass Mensch als Vogel klassifiziert wurde. Hinzu kam, dass der Trainingsdatensatz [3] dieser Experimente besonders viele Vögel in der Luft zeigt, vor einem vergleichsweise kontrastarmen und strukturlosen Hintergrund. Der Vergleich der drei Experimente ergibt, dass das untrainierte *Faster R-CNN Inception Resnet v2* zuvor eine zuverlässigere Vogelzählung erbrachte als nach dem Training.

Zusammenfassend ist zusagen, dass eine effiziente Vogelzählung nur auf einem geeigneten Modell für die Vogelerkennung basieren kann und die tatsächliche Umsetzung der Zählung der erkannten Vögel nicht die Grenze dieser Arbeit ist.

8.2 Zusammenfassung

Für die Anwendung in dieser Arbeit wurden der aktuelle Stand der Forschung (3) der Vogelzählung und grundlegende Techniken (2) betrachtet, um ein Konzept (5) zu entwickeln, welches den benutzerspezifischen Anforderungen (4) gerecht werden soll. Für die Realisierung (6) der Anwendung wurde eine konzeptnahe Umsetzung implementiert, welche ein qualifiziertes Modell voraussetzt. Dafür wurde eine Auswahl der auf *Faster R-CNN*-Architektur basierenden Modelle experimentell (7) miteinander verglichen, um eine Anpassung des ausgewählten Modells an die Anforderung der Vogelzählung vorzunehmen. Die Adaptierung zeigte, dass durch den verwendeten Datensatz des Trainings nicht die geforderten Ergebnisse erzielt werden konnten.

8.3 Ausblick

Für die weitere Entwicklung einer automatisierten Zählung von Individuen eines Schwarmes können die Ergebnisse der vorgestellten Experimente als Grundlage und Anregung für ein weiteres Training des *Faster R-CNN Inception Resnet v2* des *Tensorflow Model Zoos* dienen. Das Modell müsste dafür mit den in der Arbeit beschriebenen Trainingsparametern und einem veränderten Datensatz erneut trainiert werden.

Für eine potentielle Minimierung der falsch-positiven Erkennungen, könnte der Datensatz (5.1.2) durch Negativbeispiele ergänzt werden, welche andere Objekte als die gesuchten Vögel zeigen und diese als „kein Vogel“ deklarieren. Somit könnte der Unterschied zwischen einem Vogel und anderen Objekten anhand ihrer unterschiedlichen Merkmale erlernt werden. Vor allem für die Zählung von Vögeln am Boden wäre ein Training mit dort möglicherweise vorkommenden Objekten nötig.

Da der Trainingsdatensatz [3] besonders viele Vögel in der Luft darstellt, der einen eintönigen Hintergrund vorweist, wäre eine Ergänzung durch weitere Vogelbilder mit vielfältigen und unterschiedlich kontrastreichen Hintergründen erforderlich. Um zu vermeiden, dass Ausschnitte des Hintergrundes als Vogel erkannt und fälschlicherweise als solcher gezählt werden, könnten dem bereits bestehenden Trainingsdatensatz weitere Daten von Vogelschwärmen am Boden mit unterschiedlichen Hintergründen hinzugefügt werden. Dafür wurden bereits ca. 130 Bilddateien einschließlich der *xml*-Dateien mit *Bounding Boxes* erstellt, um für weitere Optimierungen des Modells verwendet werden zu können.

Der modulare Aufbau der umgesetzen Anwendung erlaubt es, ein verbessertes Modell, das die Anforderungen erfüllt, in die bereits vorhandene Implementierung dieser Arbeit einzubinden und somit die Vogelzählung in Schwärmen zu optimieren

Literaturverzeichnis

[1] 2020 MELIHOZ, Copyright (c). – URL https://github.com/melihoz/AUBIRDSTEST/. – Zugriffsdatum 03.11.2023

[2] ABADI, Mart et a.. – URL https://www.tensorflow.org/. – Zugriffsdatum 20.10.2023

[3] AKÇAY, Hüseyin G. ; KABASAKAL, Bekir ; AKSU, Duygugül ; DEMIR, Nusret ; ÖZ, Melih ; ERDOĞAN, Ali: *AU bird scene photo collection*. 2020. – URL https://www.kaggle.com/dsv/1193435

[4] AKÇAY, Hüseyin Gökhan et a.: Automated Bird Counting with Deep Learning for Regional Bird Distribution Mapping. In: *Animals from MPDI* (2020)

[5] ASLAM, Aneela et a.: Object Detection and Localization in Natural Scenes Through Single-Step and Two-Step Models. In: *International Conference on Emerging Trends in Smart Technologies (ICETST)* (2020)

[6] COCO, Common Object in C.. – URL https://cocodataset.org. – Zugriffsdatum 18.07.2023

[7] ENGINEERING SCIENCE, University of O. Department of. – URL https://www.robots.ox.ac.uk/~vgg/data/pets/. – Zugriffsdatum 16.05.2023

[8] FOUNDATION, Python S.. – URL https://www.python.org/. – Zugriffsdatum 06.11.2023

[9] HARRIS, C.R et. a.. – URL https://numpy.org/. – Zugriffsdatum 06.11.2023

[10] ISLAM, Shazzadul et a.: Bird Species Classification from an Image Using VGG-16 Network. In: *ICCCM '19: Proceedings of the 7th International Conference on Computer and Communications Management* (2019)

[11] JIAO, Licheng et a.: A Survey of Deep Learning-based Object Detection. In: *IEEE Access, Volume 7* (2019)

[12] Kleuker, Stephan: *Grundkurs Software-Engineering mit UML*. Springer Fachmedien Wiesbaden GmbH, 2013. – URL https://www.google.de/books/edition/Grundkurs_Software_Engineering_mit_UML/cLolBAAAQBAJ?hl=de&gbpv=1&dq=978-3-658-00641-9&pg=PR4&printsec=frontcover. – ISBN 978-3-658-00641-9

[13] Krekel, Holger et a.. – URL https://docs.pytest.org/. – Zugriffsdatum 03.11.2023

[14] Li, Ce et a.: Evaluation of super-resolution on bird detection performance based on deep convolutional networks. In: *IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (2017)

[15] Lin, TzuTa. – URL https://pypi.org/project/labelImg/. – Copyright (c) <2015-Present> Tzutalin, Copyright (C) 2013 MIT, Computer Science and Artificial Intelligence Laboratory. Bryan Russell, Antonio Torralba, William T. Freeman, Zugriffsdatum 05.11.2023

[16] Matzka, Stephan: *Künstliche Intelligenz in den Ingenieurwissenschaften*. Springer Vieweg, 2021. – URL https://doi.org/10.1007/978-3-658-34641-6. – ISBN 978-3-658-34641-6

[17] McKinney, Wes. – URL https://pandas.pydata.org/docs/index.html. – Zugriffsdatum 24.10.2023

[18] Oestereich, Bernd: *Analyse und Design mit UML 2.3: Objektorientierte Softwareentwicklung*. Oldenbourg, 2009. – URL https://www.google.de/books/edition/Analyse_und_Design_mit_UML_2_3/I2_LwAEACAAJ?hl=de. – ISBN 978-3-486-58855-2

[19] Ornithology, Cornell L. of. – URL https://ebird.org/home. – Zugriffsdatum 02.11.2023

[20] Pachipala, Yellamma et a.: Object Detection using TensorFlow. In: *International Conference on Electronics and Renewable Systems (ICEARS)* (2022)

[21] Padilla, Rafael et a.: A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. In: *Electronics 2021, 10, 279* (2021)

[22] Pillow. – URL https://pillow.readthedocs.io/en/stable/reference/Image.html. – Zugriffsdatum 06.11.2023

[23] Song, Zichen et a.: Learn to Classify and Count: A Unified Framework for Object Classification and Counting. In: *ICIGP '18: Proceedings of the 2018 International Conference on Image and Graphics Processing* (2018)

[24] Szegedy, Christian et a.: Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015)

[25] Szegedy, Christian et a.: *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016

[26] team, The M. development. – URL https://matplotlib.org/. – Zugriffsdatum 06.11.2023

[27] TensorFlow. – URL https://github.com/tensorflow/models/tree/master/research/object_detection. – Zugriffsdatum 18.10.2023

[28] Tensorflow. – URL https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. – Zugriffsdatum 25.09.2023

[29] Welinder, Peter et a.: *Caltech-UCSD Birds 200*. 2010

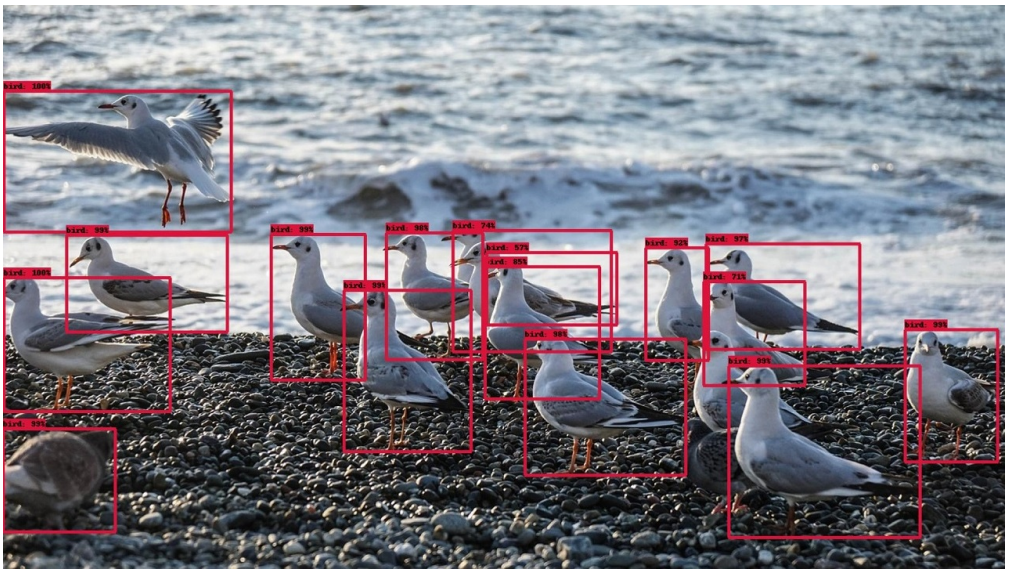
[30] Zhang, Ke et a.: Residual Networks of Residual Networks: Multilevel Residual Networks. In: *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 28, NO. 6* (2018)

A Anhang

A.1 Experiment 1: Bilder

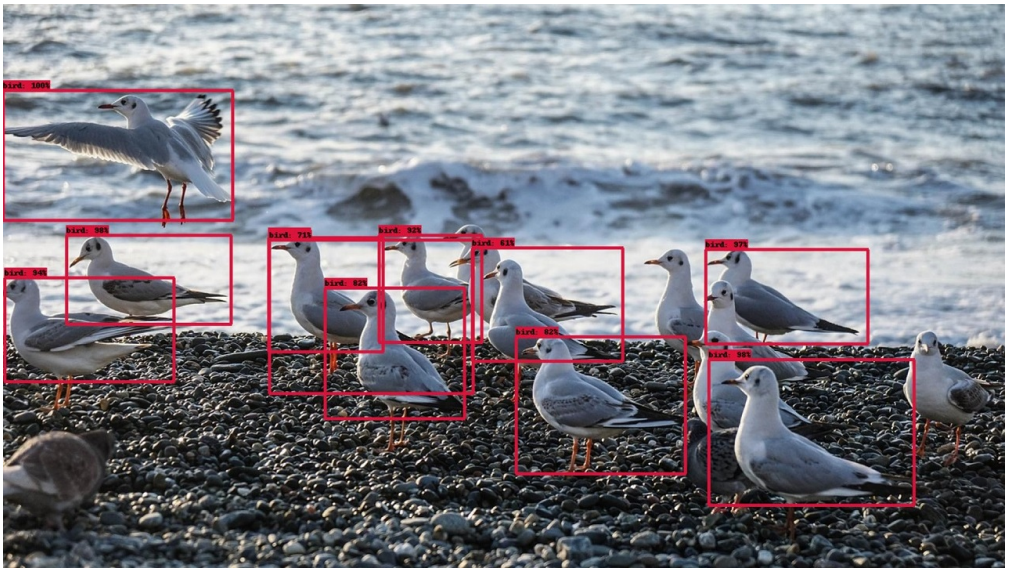
A.1.1 1. Bild: zeigt 18 Vögel

(a) Vogelzählung mit *Faster R-CNN Inception Resnet v2*: 15 richtig-positiv und 1 falsch-positiv Erkennungen



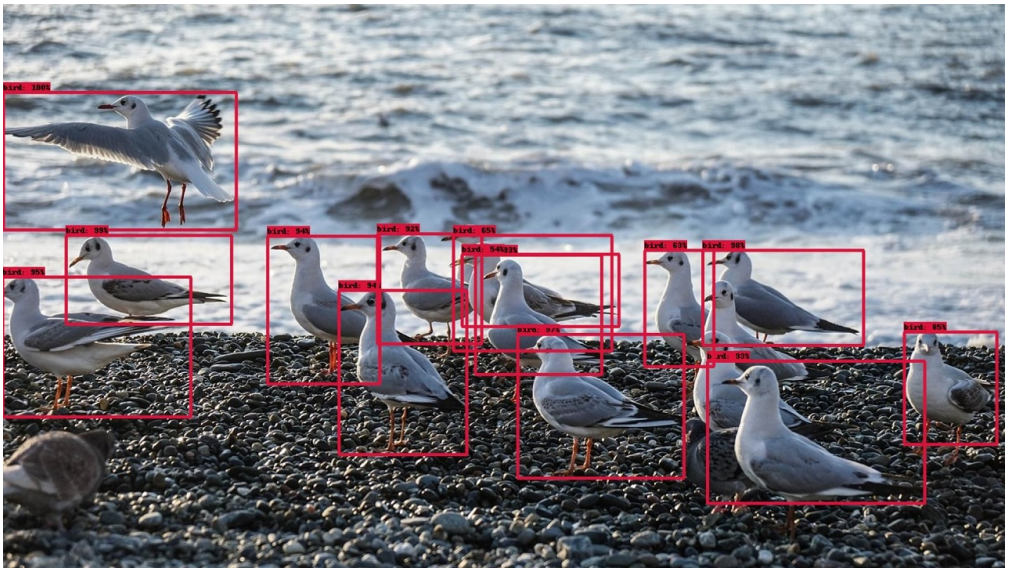
von Maria Saveleva (*SAVA86*) auf *Pixabay* (modifiziert)

(b) Vogelzählung mit *Faster R-CNN Resnet50 V1*: 10 richtig-positiv und 1 falsch-positiv Erkennungen



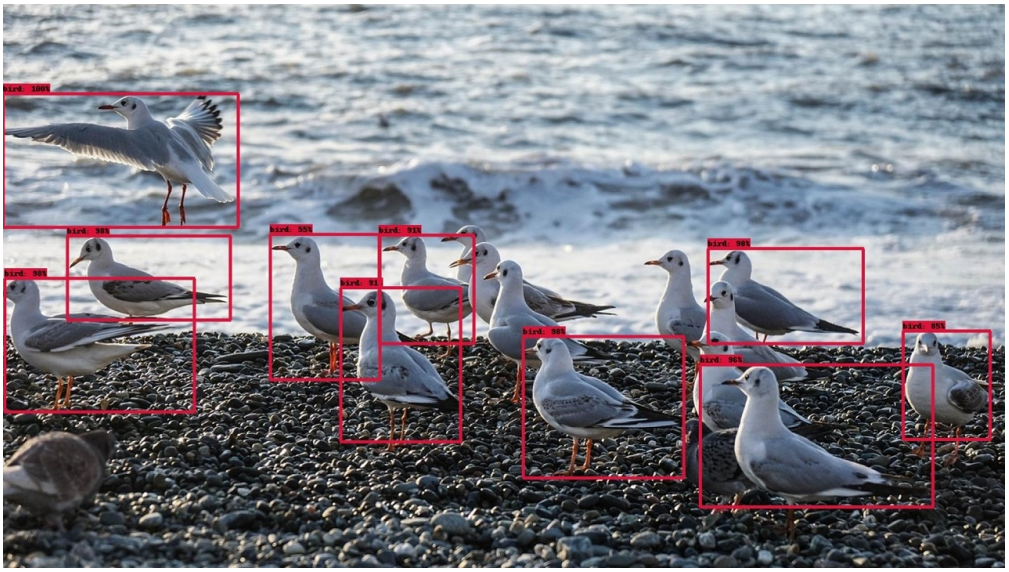
von Maria Saveleva (*SAVA86*) auf *Pixabay* (modifiziert)

(c) Vogelzählung mit *Faster R-CNN Resnet101 V1*: 13 richtig-positiv und 1 falsch-positiv Erkennungen



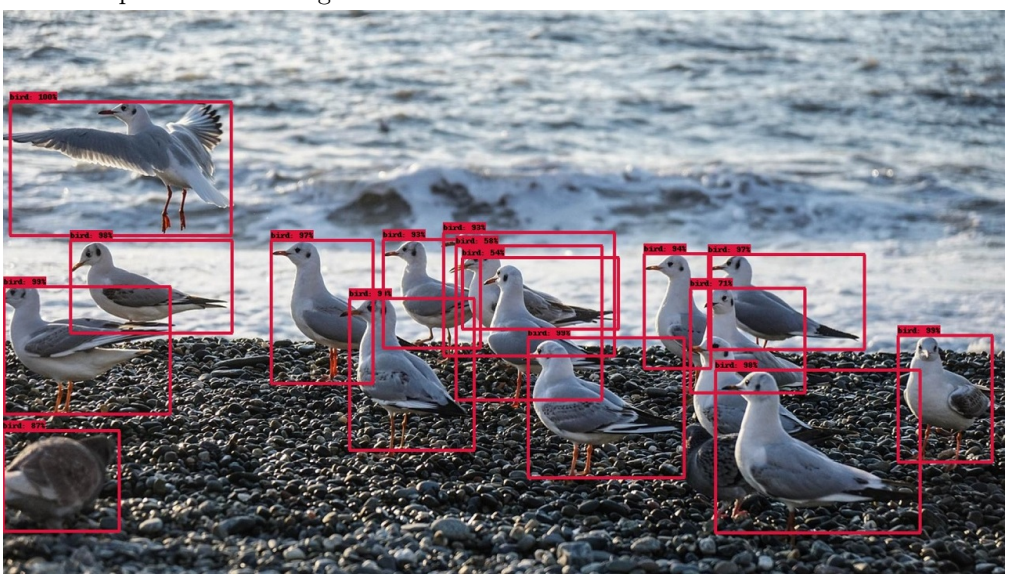
von Maria Saveleva (*SAVA86*) auf *Pixabay* (modifiziert)

(d) Vogelzählung mit *Faster R-CNN Resnet152 V1*: 10 richtig-positiv und 0 falsch-positiv Erkennungen



von Maria Saveleva (*SAVA86*) auf *Pixabay* (modifiziert)

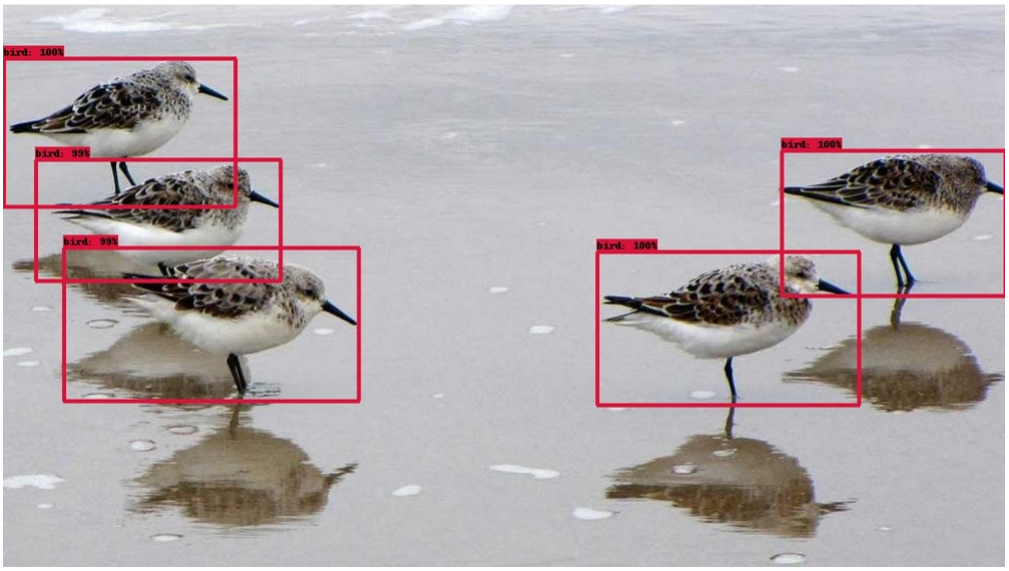
(e) Vogelzählung mit *Mask R-CNN Inception ResNet V2*: 15 richtig-positiv und 1 falsch-positiv Erkennungen



von Maria Saveleva (*SAVA86*) auf *Pixabay* (modifiziert)

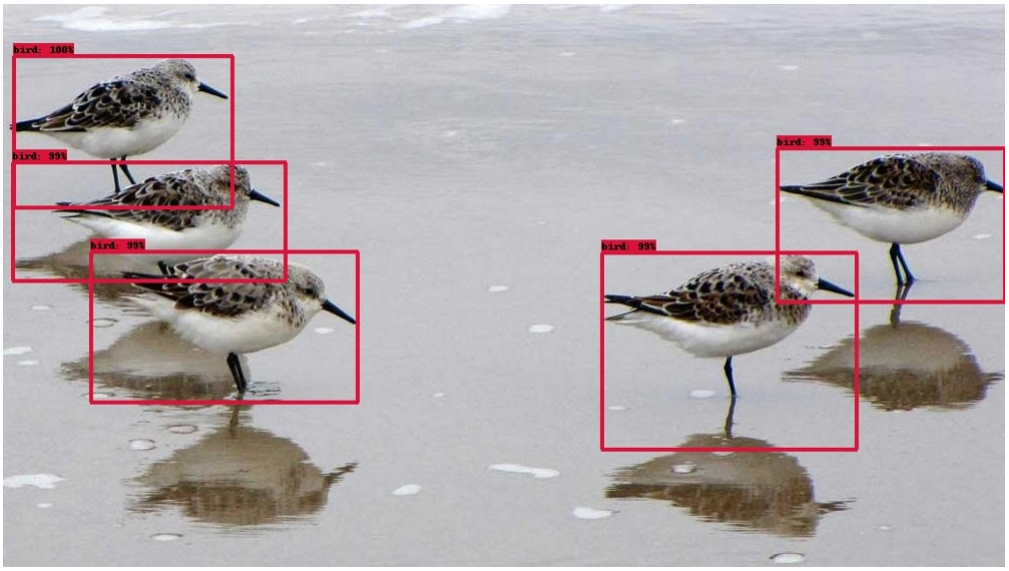
A.1.2 2. Bild: zeigt 5 Vögel und deren Spiegelbilder

(a) Vogelzählung mit *Faster R-CNN Inception Resnet v2*: 5 richtig-positiv und 0 falsch-positiv Erkennungen



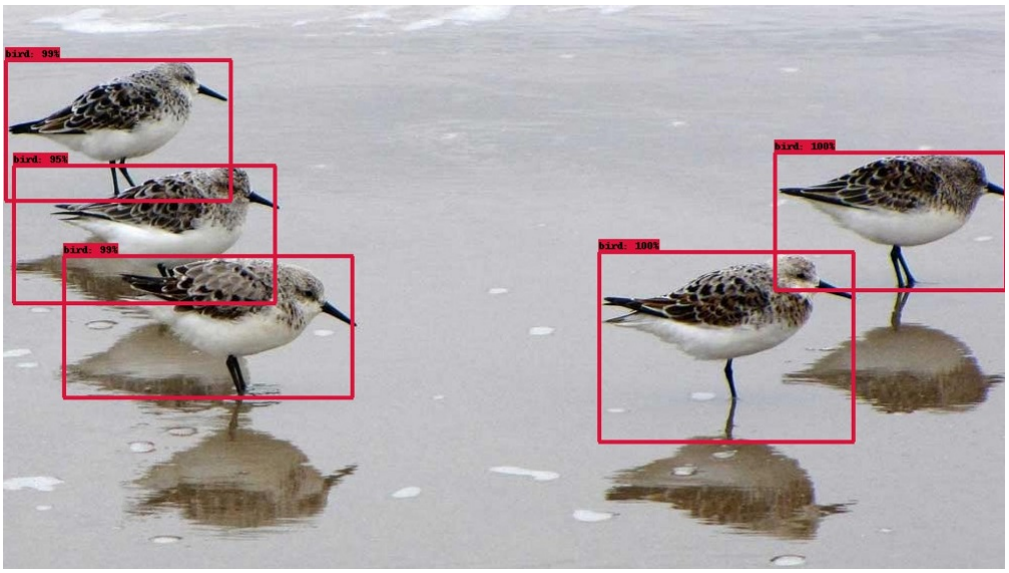
von *emmess* über *Pixabay* (modifiziert)

(b) Vogelzählung mit *Faster R-CNN Resnet50 V1*: 5 richtig-positiv und 0 falsch-positiv Erkennungen



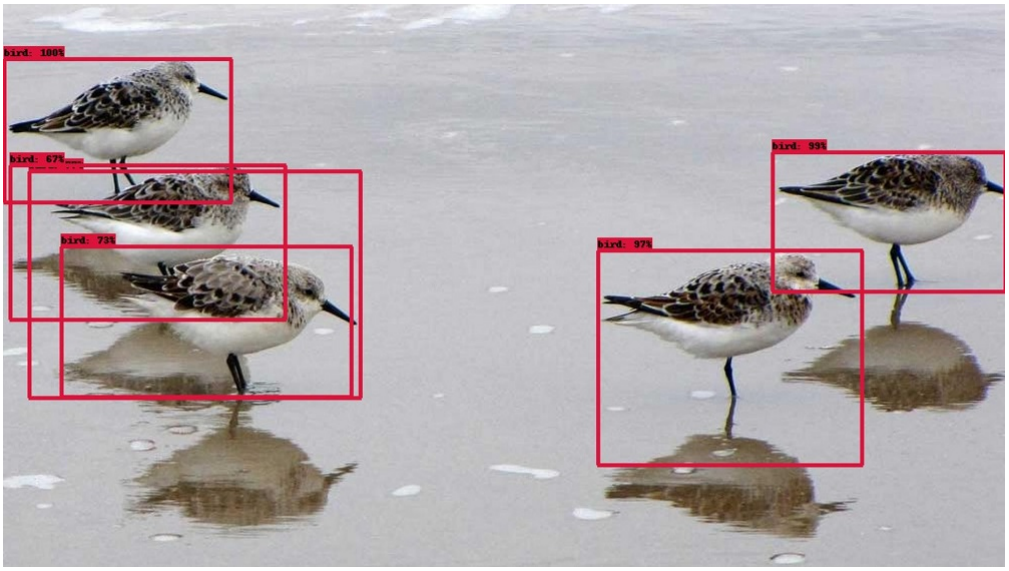
von *emmess* über *Pixabay* (modifiziert)

(c) Vogelzählung mit *Faster R-CNN Resnet101 V1*: 5 richtig-positiv und 0 falsch-positiv Erkennungen



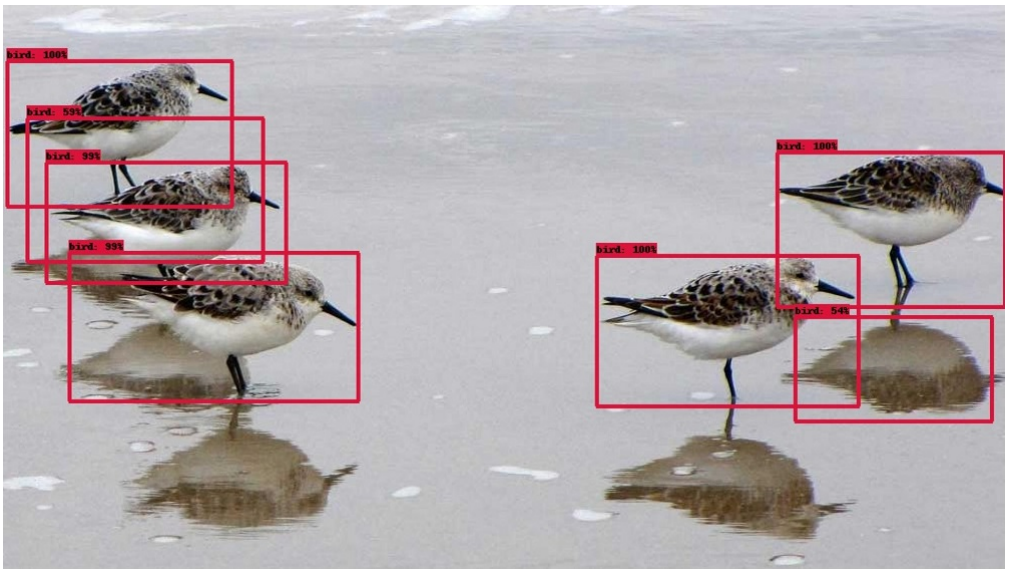
von *emmess* über *Pixabay* (modifiziert)

(d) Vogelzählung mit *Faster R-CNN Resnet152 V1*: 4 richtig-positiv und 2 falsch-positiv Erkennungen



von *emmess* über *Pixabay* (modifiziert)

(e) Vogelzählung mit *Mask R-CNN Inception ResNet V2*: 5 richtig-positiv und 2 falsch-positiv Erkennungen



von *emmess* über *Pixabay* (modifiziert)

A.1.3 3. Bild: zeigt keine Vögel

(a) Ergebnis der Vogelzählung der fünf Modelle: keine falsch-positiv Erkennung

von Sujin jetkasettakorn über *Vecteezy.com*

A.2 Experiment 3: Training des *Faster R-CNN Inception Resnet v2* mit *AU birds*

A.2.1 *Pipeline.config*

Dieser Unterabschnitt zeigt essentielle Ausschnitte aus der *Pipeline.config*

Sigmoid Cross Entropy

```
model {
  faster_rcnn {
    second_stage_post_processing {
      score_converter: SIGMOID
    }
```

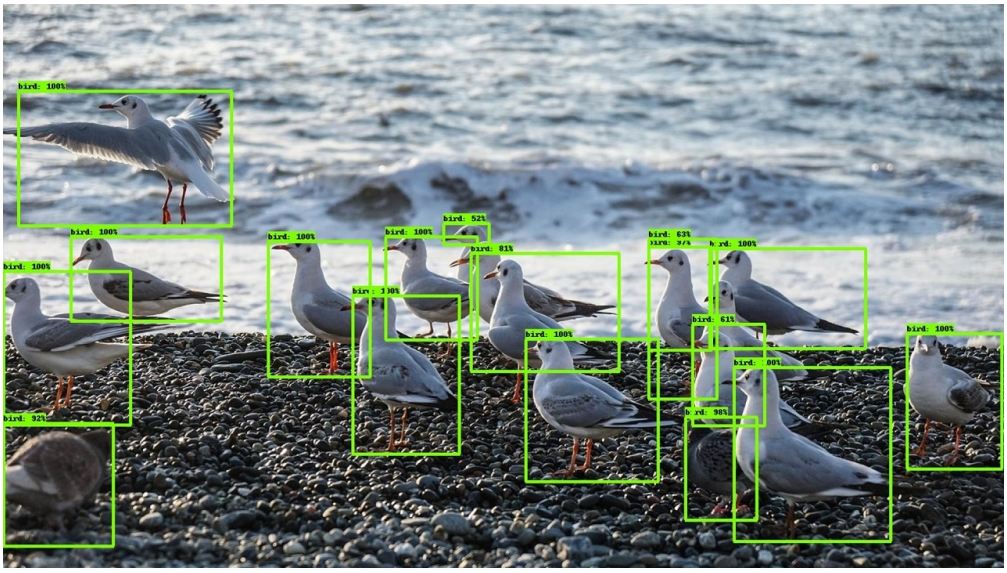
```
    second_stage_classification_loss {
      weighted_sigmoid {
      }
    }
  }
}
```

Trainingsparameter

```
train_config: {
  batch_size: 2
  num_steps: 25000
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: 0.008
          total_steps: 20000
          warmup_learning_rate: 0.0
          warmup_steps: 5000
        }
      }
    }
  }
  fine_tune_checkpoint_version: V2
  fine_tune_checkpoint:
  "/faster_rcnn_inception_resnet_v2_1024x1024_coco17_tpu-8
   /checkpoint/ckpt-0"
  fine_tune_checkpoint_type: "detection"
  }
```

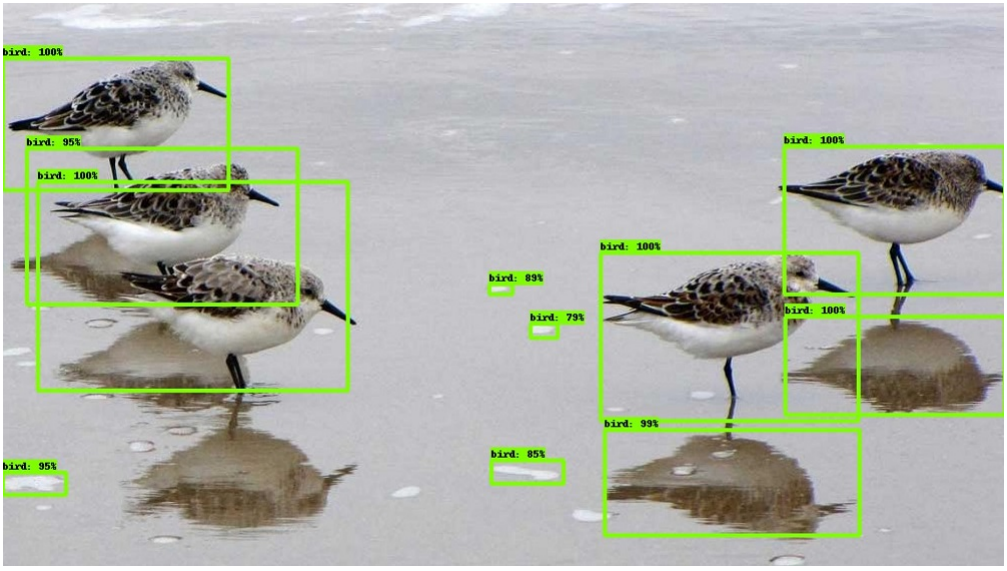
A.2.2 Ergebnisse der manuellen Tests

(a) Das erste Bild zeigt 18 Vögel mit 14 richtig-positiv und 3 falsch-positiv Erkennungen



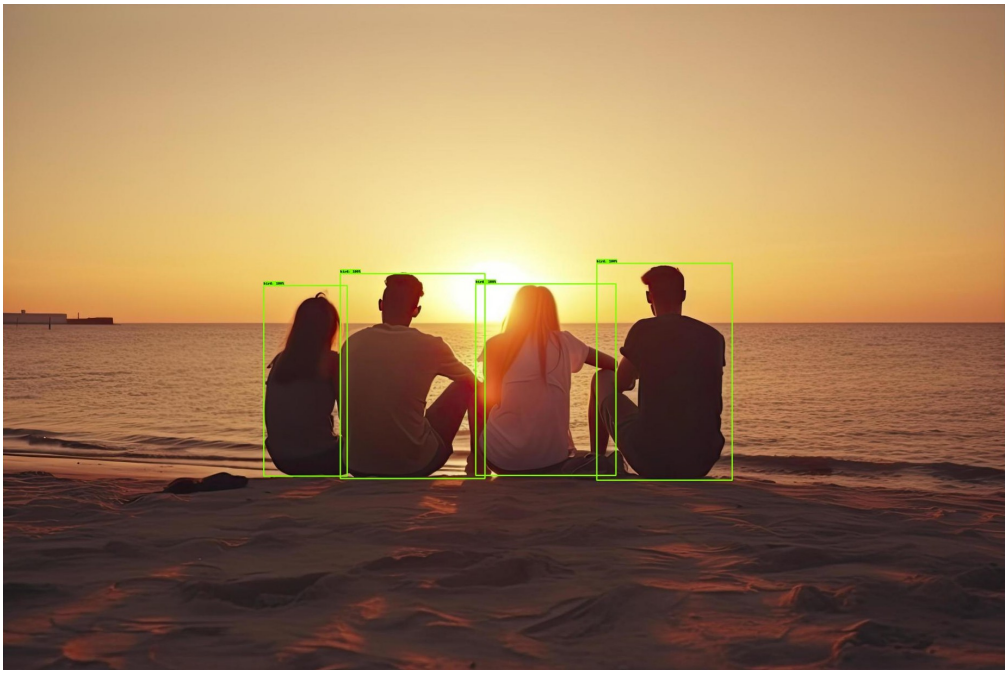
von Maria Saveleva (*SAVA86*) auf *Pixabay* (modifiziert)

(b) Das zweite Bild zeigt 5 Vögel mit 4 richtig-positiv und 7 falsch-positiv Erkennungen



von *emmess* über *Pixabay* (modifiziert)

(a) Das dritte Bild zeigt keine Vögel mit 4 falschen-positiv Erkennungen



von Sujin jetkasettakorn über *Vecteezy.com* (modifiziert)

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original