Fachhochschul-Bachelorstudiengang
MEDIZIN- UND BIOINFORMATIK
A-4232 Hagenberg, Austria

Erkennung und Klassifikation von Wildtieren mittels Computer Vision

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science in Engineering

Eingereicht von

Lukas N. Nepelius

Begutachtet von DI (FH) Dr. Gerald Zwettler MSc

Hagenberg, Mai 2022

Erklärung

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

28.05.2022 Lukas Kepelino

Datum Unterschrift

Inhaltsverzeichnis

Kurzfassung

Mit dem Aufkommen von Computer-Vision-Algorithmen in Bereichen wie Gesichtserkennung, Videoüberwachung und autonomes Fahren besteht auch die Nachfrage, Software für ökologische Zwecke zu produzieren. Der Nationalpark Hohe Tauern in Österreich profitiert von einem Software-Prototypen, der Tiere auf Wildkamera-Videos erkennt und klassifiziert. Unter diesen Videos gibt es viele unbedeutende Inhalte, die nicht zur Erforschung des Lebensraums der Wildtiere aus dem Hochgebirge beitragen. Bedingt durch Witterungseinflüsse wie Wind, Regen oder Schnee erkennt eine Wildtierkamera Bewegungen und kann dementsprechend ein Video aufnehmen, obwohl gar keine Tiere zu sehen sind. Der Prozess des manuellen Herausfilterns irrelevanter Videodaten ist mühsam und repetitiv. Daher ist es praktisch, einen Softwareprototypen zu entwerfen, der unwesentliche Videodaten heraussuchen und eine empirische Analyse der Wildtiervideos liefern kann. Algorithmen für maschinelles Lernen sind in der Lage aus einem bestimmten Datensatz zu lernen und ein Vorhersagemodell zu erstellen. Dieses Modell kann dann Objekte, wie zum Beispiel Schafe, Hasen, Vögel usw., mit einer gewissen Wahrscheinlichkeit kategorisieren. Letztendlich wird ein Software-Prototyp erstellt, um die Anzahl der einzelnen Tierarten zu zählen, die in einem bestimmten Video zu sehen sind.

Die Leistung der Objekterkennung und Objektverfolgung wird in dieser Arbeit berechnet, indem der Algorithmus mit einer Grundwahrheit verglichen wird. Für die Objekterkennung werden die Werte für Genauigkeit (0,800), Präzision (0,980), Empfindlichkeit (0,577), F1 (0,639) und Jaccard (0,563) mit dem Deep-Learning-Modell YOLOv3 bestimmt. Im besten Fall zählt der Objektverfolgungsalgorithmus 11 von 11 Tieren in einem Video, von denen 10 von 11 auch richtig klassifiziert werden.

Abstract

With the rise of computer vision algorithms in areas like facial recognition, video surveillance and autonomous driving, there is also demand to produce software for ecological purposes. The national park Hohe Tauern in Austria benefits from a software prototype that detects and classifies animals on wildlife camera videos. There is a lot of unsignificant content in these videos, which does not contribute to the research of the wildlife habitat from the high mountains. It is due to weather conditions such as wind, rain or snow, a wildlife camera detects movement and accordingly records a video, even though no animals can be seen. The process of manually filtering out irrelevant video data is tedious and repetitive. Therefore it is practical to design a software prototype that can pick out unsignificant video data and provide empirical analysis of the wildlife videos. Machine learning algorithms are capable of learning from a specific data set and constructing a prediction model. This model can then predict to categorize objects, in this example the animals like sheeps, rabbits, birds, etc. with a certain confidence. Ultimately, a software prototype is implemented to count the number of individual animal species seen in a particular video.

The performance of object recognition and object tracking is calculated by comparing the algorithm with a ground truth. For object recognition, the accurcy (0.800), precision (0.980), sensitivity (0.577), f1 (0.639) and jaccard (0.563) scores are determined using the deep learning model YOLOv3. In a best case scenario, the object tracking algorithm counts 11 out of 11 animals in a video, of which 10 out of 11 are also classified correctly.

Kapitel 1

Einleitung

1.1 Motivation

Computer Vision ist ein interdisziplinärer technologischer Bereich mit immer zunehmender Relevanz, vor allem in Hinsicht auf Themen wie selbstfahrende Autos [1], Überwachungskameras [8] und Gesichtserkennung [11]. Computer Vision heißt computerbasiertes oder maschinelles Sehen und beschreibt die Analyse von Bilddaten [29].

Um Strukturen im Vordergrund von jenen im Hintergrund differenzieren zu können gibt es verschiedenste Verfahren und Methoden [45]. Dabei wird die Struktur anhand ihrer Form, Größe und ihrem Kontrast erkannt und auch klassifiziert. Solche Strukturen nennt man auch Objekte und Verfahren, die sich mit der Erkennung jener beschäftigen, werden unter dem Namen Objekterkennung [9] zusammengefasst.

Grundsätzlich bedarf es in verschiedensten Echtwelt-Szenarien der Computer Vision. Es können Autos auf einem Autobahnabschnitt gezählt oder Sicherheitsaspekte durch Gesichtserkennung verbessert werden. In allen praktischen Anwendungsfällen von Computer Vision wird eine gewisse Arbeit von einem Computer verrichtet, wodurch Prozesse automatisiert werden.

Diese Automatisierung findet auch Verwendung für den Nationalpark Hohe Tauern, da es ein mühsames Verfahren ist, stundenlanges Videomaterial von Wildtierkameras manuell zu analysieren. Ein Algorithmus kann stattdessen diese Analyse der Tierwelt in den österrechischen Hochgebirgsregionen vollziehen.

1.2 Problemstellung

Die Auswertung von Wildtierkameras zur Überwachung der Lebensräume des Nationalpark Hohe Tauern stellt einen immensen Aufwand dar, der mit manueller Sichtung der Tiere verbunden ist. Aus diesem Grund soll die Sichtung computerunterstützt erfolgen, wobei sich der Einsatz von künstlichen neuronalen Netzwerken (KNN) anbietet. Im Zuge dieser Arbeit ist festzustellen, ob Computer Vision das Potential zur genauen und effektiven Erkennung von Tieren unterschiedlicher Größe und unter verschiedensten Wettereinflüssen erfüllen kann, obwohl relevante künstliche Intelligenzmodelle nicht perfekt auf die Tierwelt der österreichischen Alpen abgestimmt sind.

Der/Die Benutzer/in soll auf einen Blick sehen können, in welchen Videos sich Tie-

re befinden und welche sich als „Fehleraufnahmen“ herausstellen. Falsche Aufnahmen könnten zum Beispiel solche sein, in denen keine Tiere sondern nur Menschen oder andere sich bewegende Objekte, wie raschelnde Blätter oder Regen/Schnee, zu sehen sind. Somit muss der/die Anwender*in nicht mehr mühsam alle Videos der Wildtierkameras manuell kontrollieren und nach Relevanz aussortieren.

1.3 Stand der Technik

Um Objekte im gegebenen Bild erkennen zu können gibt es viele unterschiedliche Verfahren. Die verschiedenen Ansätze können unterteilt werden in traditionelle Methoden, bei denen mittels lokalem Kontrast und Segmentation vorgegangen wird und den lernbasierten Methoden, die auf KNNs aufbauen [30]. Das Gebiet der generischen Objekterkennung, das via KNN verwirklicht wird, ist wiederum in Gesichtserkennung und Fußgänger-Erkennung differenziert.

Die frei verfügbare Bibliothek OpenCV [32] beinhaltet eine Vielzahl an Klassen und Funktionen für die Bildverarbeitung. Unter Anderem gibt es auch eine Klasse für die sogenannte Hintergrund-Subtraktion. Dabei wird ein aktuelles Bild eines Videos mit einem statischen Hintergrundmodell verglichen, sodass sich bewegende Objekte lokalisiert werden ([4] S. 191). Obwohl es mit diesem Ansatz gelingt, Bewegung in einem Video festzustellen, gibt es bei dieser Methode einen entscheidenden Nachteil. Wenn die Kamera bewegt wird, dann wird das Ergebnis des Algorithmus verfälscht, weil nun der Hintergrund nicht mehr statisch und das Hintergrundmodell fehlerhaft ist.

Optical Flow beschreibt, wie sich Pixel innerhalb einer Bildsequenz bewegen. Diese Bewegung der Pixel wird meist als Pfeil oder „Nadel“ visualisiert ([4] S. 196). Es wurde Anfang der 1980er Jahre entwickelt/erfunden und wird heute noch unter anderem im Bereich der Bildverarbeitung, Deep Learning und weiteren verwendet [12]. Die Voraussetzung für ein gutes Ergebnis sind relativ gleichbleibende Lichtverhältnisse, denn der Algorithmus würde nämlich auch einen Schatten als sich bewegendes Objekt erkennen.

Ein Überblick über die Methoden der Bild-Registrierung wird in [21] gegeben. Für die Lösung des Problems der Bild-Registrierung gibt es grundsätzlich zwei prominente Ansätze. Ein Ansatz, der sich auf die Korrelation zweier Bilder konzentriert, ist ein für Computer aufwändiger Prozess und bedient sich der originalen Daten des Bildes. Der andere Ansatz beschäftigt sich mit dem Feature Matching. Dieses beschreibt das Finden von Übereinstimmungen zweier Bilder der gleichen Szene/des gleichen Objektes anhand von Features und wurde erstmals um die 2000er Jahre entwickelt. Features sind hierbei markante Stellen im Bild wie Kanten oder Liniensegmente.

Dieser Absatz beschreibt die aktuellen Methoden zum Erkennen von Features und zum Verbinden dieser Features, die von folgenden Artikeln zusammengetragen wurden: [44], [30], [15]. Algorithmen zur Feststellung der Features sind SIFT [18] (Scale Invariant Feature Transform), Harris Corner [7], FAST [26] (Features from Accelerated Segment Test), HOG [5] (Histograms of oriented gradients), Haar-like [16] und weitere. Zum Verbinden der Features beider Bilder wird häufig der Brute-Force [13] oder FLANN [22] (Fast Library for Approximate Nearest Neighbors) Matcher verwendet.

Mit der Entwicklung von Deep-Learning Algorithmen wurde auch der sogenannte YOLO [25] (You Only Look Once) Algorithmus, zum Erkennen und Klassifizieren von Objekten auf Bildern, entwickelt. Ein zentraler Aspekt von Deep-Learning Algorithmen

sind neuronale Netzwerke. Es gibt viele verschiedene Formen solcher Netze wie das feedforward- [27], recurrent- [2] und convolutional [24] neural network [38]. Der YOLO Algorithmus funktioniert nach der feedforward Variante.

Bei der Objekterkennung werden Bilddaten manuell mit dem richtigen semantischen Klassenbezeichner markiert, sodass das Netzwerk zum Schluss korrigiert wird wenn es eine falsche Klassifikation getätigt hat. Es handelt sich also um ein überwachtes maschinelles Lernverfahren [23]. Dieser Vorgang wird oft mit vielen verschiedenen Input-Bildern wiederholt, sodass das Netzwerk anhand von Bilddaten auf das richtige Klassifizieren von Objekten trainiert wird.

OpenCV stellt eine Funktion zum Detektieren von Objekten mittels Haar featurebasierend kaskadierenden Klassifizierer zur Verfügung [41]. Keras ist eine frei verfügbare Deep-Learning-Bibliothek, die unter anderem das sogenannte RetinaNet zur Objekterkennung anbietet [37]. Verwendet werden hierbei Feature Pyramid Networks, die eine robuste und schnelle Alternative zu gängigen Deep-Learning Verfahren darstellt [17]. Das Framework TensorFlow verwirklicht Objekterkennung anhand von zwei Modulen [36]. Das erste Modul behilft sich der Faster RCNN und der InceptionResNet V2 Methode und besitzt eine hohe Genauigkeit. Das andere Modul ist performanter und kleiner, dafür aber ungenauer und verwendet ssd und MobileNet V2. In ImageJ gibt es eine Methode zum Multi Template Matching [40].

1.4 Lösungsansatz

Die automatisierte Erkennung von Wildtieren, wie sie in der Problembeschreibung skizziert wurde, kann durch Anwendung von Verfahren zur Objekterkennung durchgeführt werden. Im Zuge dieser Arbeit wird hierzu ein Verfahren entwickelt, das mittels YOLO Algorithmus arbeitet und relevante Objekte aus Videodaten erkennen und klassifizieren kann. Das Modell des YOLO Algorithmus ist dabei nicht in perfekter Übereinstimmung mit den vom Nationalpark Hohe Tauern bereitgestellten Daten, da das Modell nicht in der Lage ist alle Tierarten zu kategorisieren.

Kapitel 2

Grundlagen

2.1 OpenCV Grundlagen

Dieses Kapitel wurde aus den beiden Büchern [3] und [10] zusammengetragen. OpenCV ist eine umfangreiche Bibliothek für Bildverarbeitungs- und Computer Vision Methoden. Die Benützung von OpenCV bringt einige Vorteile, die durch die grundsätzliche Architektur des Frameworks ermöglicht werden. Ein Aspekt wäre das Konzept des Speichermanagements, das das Lesen / Schreiben von Bildern und Videos in OpenCV, ohne Speicher manuell allokieren oder wieder freigeben zu müssen, ermöglicht.

Die Bibliothek wurde erstmals Ende der 1990er Jahre von der Firma Intel als Forschungsprojekt in Auftrag gegeben. Die Vision hinter dem Projekt war, auf Computer Vision aufbauende Algorithmen frei verfügbar und optimiert zu implementieren, damit das Neuerfinden von schon etablierten Algorithmen reduziert und die Forschung in diesem Bereich vorangetrieben wird. Entwickler sollen auf den Methoden des Frameworks aufbauen und somit den Programmcode leicht verständlich machen. Die erste Version von OpenCV wurde im Jahr 2000 auf der IEEE Conference on Computer Vision and Pattern Recognition präsentiert und seither kontinuierlich weiterentwickelt.

Die Bibliothek umfasst eine Menge an Methoden mit denen man Bild- und Videodaten in jeglicher Art und Weise manipulieren kann. Dazu zählen Verfahren der Bildbearbeitung und Bildverarbeitung wie Kontrastanhebung, Kantendetektion, Optical Flow oder auch komplexere Methoden der Computer Vision wie Objekterkennung, Klassifizierung mittels Machine Learning. Aufgeteilt ist das Framework in verschiedene Module, die dann beispielsweise Funktionen für die Bildmanipulation enthalten. Das wichtigste Modul in OpenCV heißt *Core* und ist für die grundlegenden Datentypen, Datenstrukturen und die Speicherzuweisung zuständig. Eines dieser wichtigen Datentypen ist *Mat*, das im Prinzip ein Bild in Matrizenform abspeichert. Ein Überblick der wichtigsten Module in OpenCV ist in Tabelle 2.1 zu sehen.

Modul	Beschreibung
Highgui	GUI von OpenCV
Video	Bewegungsanalysen und Objektverfolgung
Features2d	Features herausfiltern, beschreiben und verbinden
Imgcodecs	Bilder lesen und schreiben
videoio	Videos lesen und schreiben
dnn	Objekterkennung und Klassifizierung mittels Deep Learning

Tabelle 2.1: Überblick der wichtigsten Module in OpenCV.

2.2 YOLO Algorithmus

Folgendes Kapitel wurde aus dem wissenschaftlichen Artikel von Redmon et al. [25] über den YOLO Algorithmus zusammengetragen.

Bei dem YOLO (Engl.: You Only Look Once) Algorithmus wird ein einzelnes neuronales Netzwerk verwendet, das die Begrenzungsrahmen der Objekte auf Bildern voraussagt. Der YOLO Algorithmus ist im Vergleich zu anderen Methoden am aktuellen Stand der Technik wie dem RCNN (Engl.: Recurrent Convolutional Neural Network) relativ schnell. Er verarbeitet im Schnitt 45 Bilder in der Sekunde, wobei die YOLOv3-tiny Variante des Algorithmus sogar 155 Bilder pro Sekunde schafft. In Kontrast zu RCNN benötigt YOLO nur einen Prozess, der alle Objekte auf dem gegebenen Bild erkennt und klassifiziert, wovon auch das Akronym „You Only Look Once“ abgeleitet ist. In Abbildung 2.1 werden die wichtigsten Funktionen illustriert.

Der Algorithmus funktioniert so, dass zuerst alle Eingabebilder in eine S mal S große Matrix aufgeteilt werden. Die Bilder werden dabei so skaliert, dass alle die gleichen Dimensionen besitzen. Dabei bestimmt jede Zelle in der Matrix die Begrenzungsrahmen und eine Wahrscheinlichkeit. Diese Wahrscheinlichkeit gibt an, wie sicher sich der Algorithmus ist, dass sich in der aktuellen Zelle ein Objekt befindet. Wenn in einer Zelle ein Objekt gefunden wird, wird das IoU-Verhältnis (Engl.: Intersection Over Union) des vorhergesagten und tatsächlichen Begrenzungsrahmens des Objektes gebildet. Zusätzlich wird für jede Zelle auch noch die Wahrscheinlichkeit für die jeweiligen Klassen gespeichert. Dieser Wert gibt an, wie sicher sich der Algorithmus ist, dass es sich bei einem gefundenen Objekt um eine konkrete Klasse handelt. Beide dieser Wahrscheinlichkeiten werden in weiterer Folge multipliziert, sodass ein Wert entsteht, der darüber Auskunft gibt, wie gut eine Klasse zu einer Rahmen bzw. wie gut ein Rahmen zu einem Objekt passt. In Abbildung 2.2 ist die Architektur des Netzwerkes zu sehen.

Die erste Schicht des CNN von YOLO, die sogenannte Eingabeschicht, ist für die Bestimmung der Eigenschaften des Bildes zuständig. Die vollständig verbundenen Schichten (Engl.: Fully Connected Layers) in den versteckten Schichten (Engl.: Hidden Layers) berechnen die Wahrscheinlichkeiten und Koordinaten der Begrenzungsrahmen der Ob-

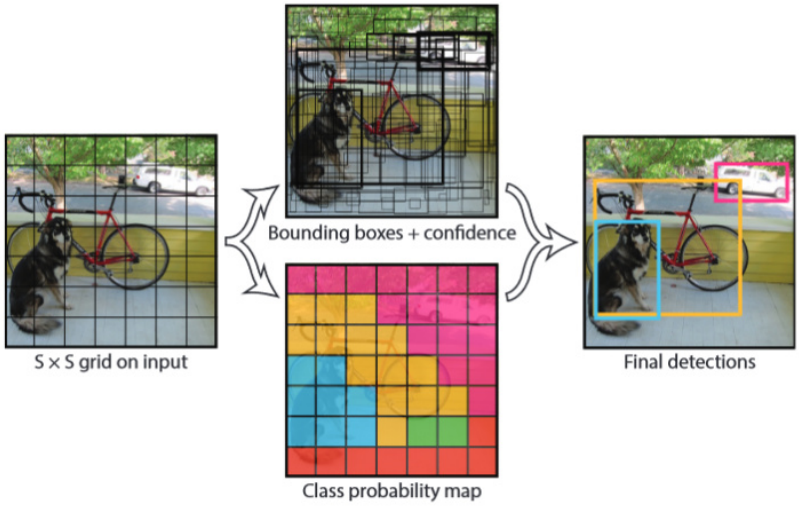


Abbildung 2.1: Überblick der wichtigsten Funktionsweisen des YOLO Algorithmus. Abbildung entnommen aus [25].

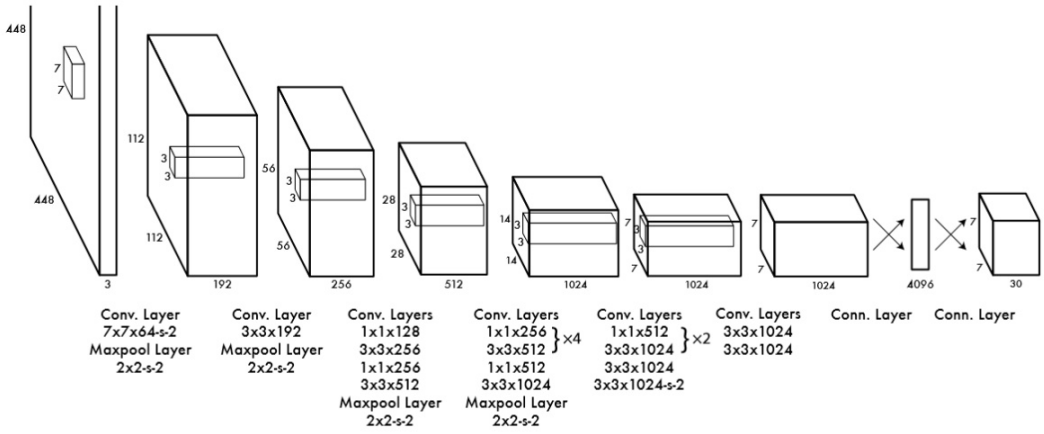


Abbildung 2.2: Der skizzierte Aufbau und die Struktur des CNN. Abbildung entnommen aus [25].

jekte. In der Standardversion von YOLOv3 gibt es im Netzwerk 24 gefaltete (Engl.: Convolved) und zwei vollständig verbundene Schichten. Das Netzwerk der „tiny“ Version besitzt nur 9 statt 24 gefaltete Schichten.

Trainiert wird das CNN mit dem ImageNet [31] Datensatz, das Bilder der Größe 448×448 Pixel annimmt. Die Höhe, Breite, x- und y-Koordinaten der Begrenzungsrahmen werden relativ zur Bildgröße normalisiert, sodass jeweils ein Wert zwischen 0 und 1 verwendet wird.

Die Nicht-Maxima-Unterdrückung (Engl.: Non-Maximum Suppression) kann dem Algorithmus ergänzt werden, um ineinander verschachtelte Rahmen am gleichen Objekt

zu verhindern / reduzieren.

Jede Zelle der Matrix kann nur maximal 2 Boxen und eine Klasse vorhersagen. Der YOLO Algorithmus hat Probleme mit kleinen und gruppierten Objekten, wie zum Beispiel einem Vogelschwarm. Die meisten Fehler der Methode passieren mit falschen Lokalisationen von Begrenzungsrahmen.

Der große Vorteil, den YOLO gegenüber RCNN und ähnlichen Methoden hat, ist, dass er nur ein gefaltetes neuronales Netzwerk benötigt, das die ganze Arbeit der Objekterkennung und Klassifikation übernimmt.

2.3 Nicht-Maxima-Unterdrückung

Beim Erkennen von Objekten tritt oft der Fall ein, dass pro Objekt mehrere Begrenzungsrahmen vorhergesagt werden. Um unnötig viele Rahmen um ein Objekt reduzieren zu können, wird die Non-Maximum Unterdrückung angewandt [39].

Im Prinzip wird dabei eine Eingabe-Liste an Boxen mit den korrespondierenden Wahrscheinlichkeiten und ein Schwellwert der Funktion übergeben. Der erste Schritt des Algorithmus ist das Übertragen des Rahmens mit dem höchsten Wahrscheinlichkeitswert der ursprünglichen Liste in die Ausgabe-Liste. Nun wird die aktuelle Box in der Ausgabe-Liste mit allen anderen Boxen in der Eingabe-Liste verglichen, indem das IoU-Verfahren, also die Überlappung zweier Boxen, angewandt wird. Unter allen verglichenen Begrenzungsrahmen wird dann jener Rahmen der Ausgabe-Liste hinzugefügt, der den anfänglich bestimmten Schwellwert überschreitet. Dieser zuvor beschriebene Vorgang wird solange wiederholt bis sich keine Boxen mehr in der Eingabe-Liste befinden. Schlussendlich wird also anhand eines Schwellwerts eine Untermenge an Boxen bestimmt, die eine möglichst große Schnittmenge mit allen eingegebenen Boxen teilen. Das Resultat dieses Filters wird in Abbildung 2.3 dargestellt.

Abbildung 2.3: Der Begrenzungsrahmen mit der höchsten IoU wird gefiltert. Abbildung entnommen aus [39].

2.4 Objektverfolgung

Eines der Probleme beim Erkennen von Objekten ist, dass der Algorithmus Bilder in einem Video isoliert betrachtet und analysiert. Ein Zählen der Objekte ist daher nicht möglich, was aber durchaus sinnvoll zum Erheben von relevanten Daten der Tiere wäre. Um nun diesen Zusammenhang zwischen den Bildern und den Objekten herstellen zu können, müssen die Objekte sozusagen verfolgt werden. In anderen Worten muss der Algorithmus dasselbe Objekt über mehrere Bilder hinweg bestimmen können. Aus diesem Grund werden nun Algorithmen zur Objektverfolgung besprochen.

Die frei verfügbare Bibliothek OpenCV stellt mehrere Verfahren am aktuellen Stand der Technik zur Verfügung unter anderem die CSRT Methode, die nach dem Schema des diskriminativen Korrelationsfilters mit Kanal- und Lokalisationszuverlässigkeit (Engl.: Discriminative Correlation Filter with Channel and Spatial Reliability; Abk.: CSR-DCF) Verfahrens implementiert wird. In weiterer Folge wurde dieses Verfahren verwendet, auch aufgrund der hohen Präzision der Methode. Die Videos vom Nationalpark Hohe Tauern haben eine relativ geringe Auflösung und Bildwiederholrate, weshalb dieser Aspekt sehr wichtig und eine längere Laufzeit der Methode tolerierbar ist. Im folgenden Absatz wird der CSR-DCF [19] beschrieben.

Der CSRT Tracker von OpenCV funktioniert mittels diskriminativen Korrelationsfilters (DCF) in Kombination mit Kanal- und Lokalisationszuverlässigkeit. Bei DCF Methoden wird das Bild in den Frequenzraum mit der Fast Fourier Transformation (FFT) überführt. Der Nachteil bei dieser Transformation ist, dass dabei künstliche kreisförmige Strukturen entstehen, die den Suchbereich des Bildes verzerren. Das zweite Problem bei DCF Methoden ist, dass der Algorithmus annimmt, dass die Box um das Objekt nach der x- bzw. y-Achse ausgerichtet wird. Dadurch können unregelmäßig geformte Objekte oder auch hohle Objekte nicht erkannt werden.

Diese zwei Probleme werden mit dem CSR-DCF gelöst, indem eine sogenannte räumliche Zuverlässigkeitskarte (Engl.: Spatial Reliability Map) erzeugt wird. Der Vorgang des Verfahrens wird in folgender Aufzählungsliste textuell und in Abbildung 2.4 auch visuell erläutert.

- Trainingsbereich: Zuerst wird ein Trainingsbereich mit einer Box festgelegt.
- Räumliche A-Priori Wahrscheinlichkeit: Die räumliche A-Priori Wahrscheinlichkeit wird als unärer Term in der Markov-Zufallsfelder-Optimierung verwendet.
- Rückprojektion: Die logarithmische Objektwahrscheinlichkeit nach Vorder- und Hintergrund Farbmodellen.
- A-Posteriori Wahrscheinlichkeit: Die A-Posteriori Objektwahrscheinlichkeit nach der Markov-Zufallsfelder-Regulierung.
- Überlagerter Bereich: Der Trainingsbereich wird mit der finalen binären Zuverlässigkeitskarte maskiert.

Im Prinzip geht der Algorithmus in zwei Schritten vor: Zuerst wird in einem Lokalisationsschritt die neue Zielposition des Objektes berechnet und anhand dieser Position wird dann die Größe geschätzt. Der zweite Schritt ist ein Aktualisierungsvorgang, bei dem jeweils für Vorder- und Hintergrund ein Histogramm erstellt, die Zuverlässigkeitskarte berechnet, ein neuer Filter geschätzt und die Kanalzuverlässigkeit erzeugt wird, wie Abbildung 2.5 zeigt.

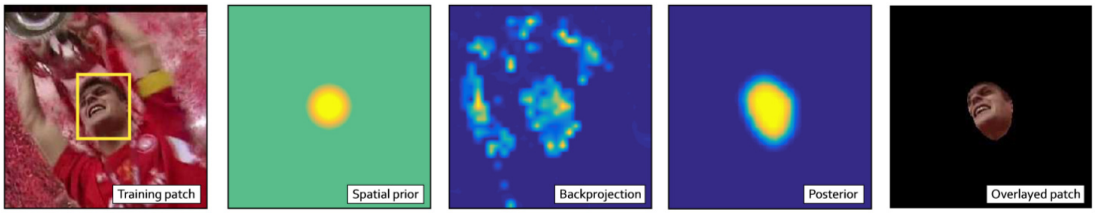


Abbildung 2.4: Einzelne Schritte beim Erzeugen einer räumlichen Zuverlässigkeitskarte. Abbildung entnommen aus [6].

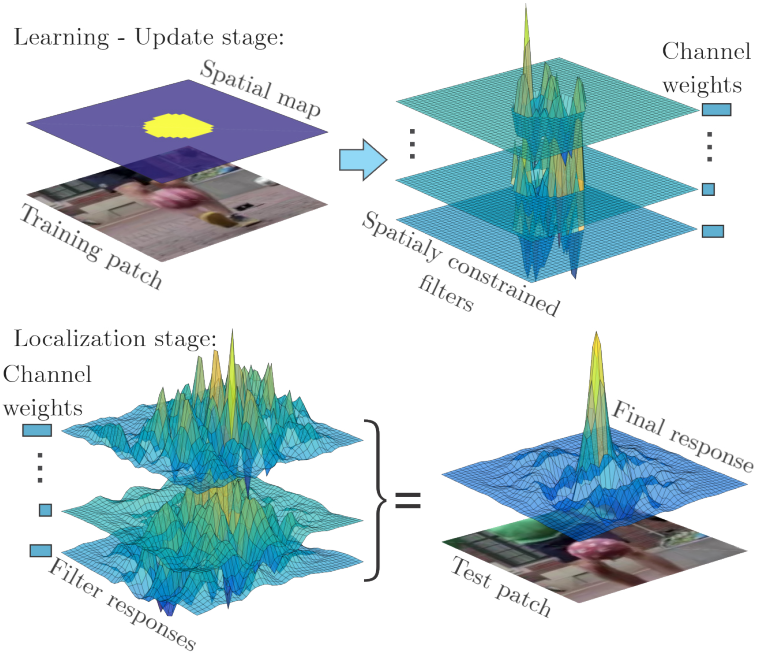


Abbildung 2.5: Der Lokalisationsschritt wird initial durchgeführt um die Position des Objektes festzustellen. Pro Bild wird dann die räumliche Zuverlässigkeitskarte berechnet und der Filter optimiert. Abbildung entnommen aus [20].

Der eben beschriebene Algorithmus arbeitet mit HOG und Colornames [28] Features, was ein präzises Verfolgungs-Ergebnis ermöglicht. Vergleicht man die CSR-DCF Methode mit anderen Ansätzen nach dem aktuellen Stand der Technik wie dem SRDCF [6] oder dem CFLB [14] vor allem in Hinsicht auf Präzision, dann schneidet die CSR-DCF Methode vergleichsweise besser ab. Objektverfolgungsverfahren wie der KCF erzielen zwar eine kürzere Laufzeit, andererseits leidet aber auch die Qualität der Verfolgung darunter.

Zusammenfassend führt der CSR-DCF Ansatz mit dem Erstellen einer Zuverlässigkeitskarte und dem Adaptieren des Filters, zu einem genauerem Verfolgen des Objekts. Man umgeht dabei auch die zwei Probleme des DCF Verfahrens, einerseits die kreisförmigen Verzerrungen durch die Fourier Transformation und dem Rechteck-Problem.

Kapitel 3

Material

Der Prototyp und auch die grafische Benutzeroberfläche wird gemäß dem vom Nationalpark Hohe Tauern bereitgestellten Datensatz entworfen. Der Datensatz besteht aus 65 Videos im WMV-Format. Neben den Videos gibt es auch 78 Dateien im JPEG- und 162 Dateien im TLV-Format, welche aber innerhalb dieser Arbeit zu vernachlässigen sind. In Abbildung 3.1 ist eine Gams und in Abbildung 3.2 sind Vögel zu sehen.

Abbildung 3.1: Ausschnitt aus einem Video, auf dem eine Gams zu sehen ist.

Die Wildtierkameras werden an unterschiedlichen geographischen Standorten innerhalb des Nationalparks aufgestellt, welche ausgelöst werden, sobald die Kamera eine gewisse Bewegung wahrnimmt. Um Speicherplatz und Batterie der Kamera zu sparen, nimmt diese mit einer schlechten Auflösung und Bildwiederholrate auf, was eine akkurate Objekterkennung und Klassifikation erschwert.

Inhaltlich sieht man auf den Videos den Lebensraum der Wildtiere der Zentralalpen in Österreich, welche von Tieren wie Hasen, Schafe, Gämse, Steinböcke, Füchse, Murmeltiere und vielen Weiteren bewohnt wird [43]. Eine andere Schwierigkeit bei der Entwicklung des Prototypen ist die Erfassung der Bandbreite an spezifischen Tierarten, welches vor allem im Methodik Kapitel noch genauer besprochen wird.

Abbildung 3.2: Ausschnitt aus einem Video, auf dem Krähen zu sehen sind.

Kapitel 4

Methodik

4.1 Vorbereitung und Installation

Bevor nun der Code des Prototyps erklärt wird, müssen einige Vorbereitungen getroffen werden. Die Bibliothek OpenCV muss in weiterer Folge installiert werden, um die Methoden und Datentypen dieses Pakets zugänglich zu machen. Zur Installation aller notwendiger Module für den Prototypen empfiehlt sich den Inhalt der *requirements.txt* Datei mit folgendem Konsolenkommando zu laden.

```
1    pip install -r requirements.txt
```

Es empfiehlt sich auch eine virtuelle Umgebung für den Python-Interpreter zu erstellen, um Konflikte mit anderen Paketen zu vermeiden.

In den ersten Zeilen Code des Programms werden die eben installierten Bibliotheken importiert.

```
1    import csv
2    import os
3    import cv2 as cv
4    import numpy as np
5    import PySimpleGUI as ps
```

Im weiteren Schritt kann nun ein Video gelesen und angezeigt werden. OpenCV stellt hierfür, wie schon im Grundlagen Kapitel angesprochen, eine Methode zur Verfügung.

```
1    capture = cv.VideoCapture("myVideo.mp4")
```

Mit dem *capture* Objekt kann nun jedes einzelne Bild in dem gegebenen Video nacheinander verarbeitet werden. Die Methode *capture.read()* ließt ein aus dem Video extrahiertes Bild und gibt dabei das Bild an sich und eine boolesche Variable zurück, welche angibt, ob das Bild richtig gelesen wird. Folgendes Code-Beispiel demonstriert das Abspielen eines Videos mittels OpenCV.

```
1 import cv2 as cv
2
3 if __name__ == '__main__':
4     capture = cv.VideoCapture("vtest.avi")
5
6     while True:
7         success, img = capture.read()
8
9         if success:
10            cv.imshow("test", img)
11            cv.waitKey(10)
12        else:
13            break
```

Die Funktion *cv.imshow()* in Zeile 10 zeigt dabei ein Bild aus dem Video in einem eigenen Fenster, in diesem Fall mit dem Namen „test“, an. In Zeile 11 wird eine Funktion definiert, die den Prozess des Programms für 10 Millisekunden anhält. Die Funktionen für das Einlesen und Anzeigen der Bilder nehmen auch eine gewisse Rechenzeit in Anspruch und mit dieser Verzögerung lässt man der Methode ein wenig Zeit für diesen Prozess. Stellt man keine Verzögerung ein, so ist das Programm überfordert und das Video „friert“ ein. In Abbildung 4.1 ist das Fenster zu sehen, das das Video anzeigt.

Abbildung 4.1: Video wird in einem Fenster abgespielt.

Um das Abspielen des Videos auch anwendungsfreundlich zu gestalten, ist eine Pausieren / Fortsetzen Funktion, sowie eine Funktion, mit der man Bild für Bild das Video ansehen kann, sinnvoll. OpenCV ermöglicht auch das Zeichnen von einfachen geometrischen Figuren, wie zum Beispiel einem Rechteck auf das Video. Um dem/die Anwender/in zu signalisieren, dass das Video pausiert oder abgespielt wird, kann ein Pausiert-Symbol auf das aktuelle Bild im Video projiziert werden, wie in Abbildung 4.2 zu sehen. Der/Die Anwender/in sollte zusätzlich noch in der Lage sein, das Abspielen des Videos jederzeit abbrechen zu können. Für das Pausieren / Fortsetzen des Videos wurde die Taste „r“, für das schrittweise Abspielen des Videos „s“ und zum Abbrechen „esc“ verwendet.

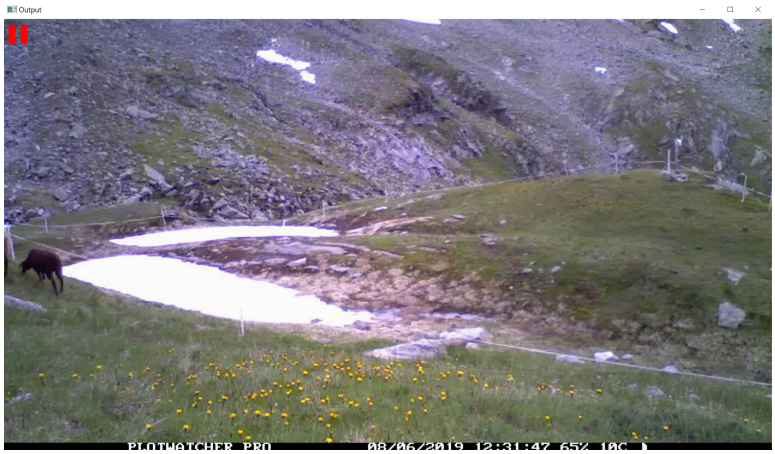


Abbildung 4.2: Bei einem pausierten Video werden zwei rote Rechtecke angezeigt.

Das Abbrechen des Videos erfolgt hierbei mit der OpenCV Methode *cv.destroyAllWindows()*, wodurch auch alle derzeit geöffneten Fenster geschlossen werden. Damit das Video angehalten bzw. wieder fortgeführt oder Bild für Bild iteriert werden kann, wird die Methode *cv.waitkey()* verwendet. Die Zahl, die bei dieser Funktion als Parameter übergeben werden kann, gibt die Zeit der Verzögerung pro Bild in Sekunden an. Der zuvor erstellte Quellcode wird im die folgenden Zeilen Code innerhalb der *if success* Bedingung erweitert.

```
1  # stepping video
2  if not stepping:
3      key = cv.waitKey(10)
4  else:
5      key = cv.waitKey()
6      stepping = False
7
8  if key == 27:  # press 'esc' to exit
9      cv.destroyAllWindows()
10     break
11
12 # stop / resume video
13 if key == ord('r'):
14     # red rectangle
15     cv.rectangle(img, (10, 10), (20, 40), (0, 0, 255), -1)
16     cv.rectangle(img, (30, 10), (40, 40), (0, 0, 255), -1)
17
18     cv.imshow("Output", img)
19     cv.waitKey()
20
21 if key == ord('s'):
22     stepping = True
```

4.2 Einsetzen des YOLO Algorithmus

Der nächste Schritt ist nun die Objekterkennung und Klassifikation mit dem YOLOv3 Modell in den Quellcode zu integrieren.

Für das folgende Programm wird ein vortrainiertes Modell verwendet, dass schon auf einem großen Datensatz von Bildern angewandt wurde. Dabei gibt es 80 Klassen, die das Modell erkennen kann, von denen nur die Klassen Vogel, Katze, Hund, Pferd, Schaf und Kuh (Engl.: Bird, cat, dog, horse, sheep, cow) für dieses Programm benötigt werden. Zuerst werden mit der Datei *coco.names* die entsprechenden Klassennamen geladen.

```
1 # Get classnames
2 classNames = []
3 classFile = 'coco.names'
4 with open(classFile, 'rt') as f:
5     classNames = f.read().rstrip('\n').split('\n')
```

Anschließend wird pro Klasse eine zufällige Farbe für den Rahmen berechnet, welches folgende Zeilen Code realisieren.

```
1 # random color for each class
2 np.random.seed(0)
3 colors = np.random.randint(0, 255, size=(len(classNames), 3)).tolist()
```

Das Deep Learning Modell wird mit der in Zeile 5 definierten Funktion des folgenden Codeausschnittes geladen.

```
1 # YOLO file paths
2 configPath = 'yolov3.cfg'
3 weights = 'yolov3.weights'
4
5 net = cv.dnn.readNet(weights, configPath)
```

Der *weights* Paramter enthält den Pfad für die *.weights* Datei, die im wesentlichen alle Knoten des trainierten Modells enthält. Das *configPath* Argument entspricht der generellen Struktur des Modells, also wie viele Schichten welcher Art in welcher Reihenfolge im Netzwerk auftreten.

Nun muss die Ausgabeschicht des eben geladenen Netzwerkes bestimmt werden, was durch jene zwei Zeilen Code realisiert wird.

```
1 layer_names = net.getLayerNames()
2 outputlayers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
```

Anschließend muss pro Bild ein sogenannter *blob* erstellt werden, der ein skaliertes, normalisiertes Bild nach Mittelwerts-Subtraktion und tauschen der Rot / Blau Kanäle des originalen Bildes darstellt. Dieser *blob* kann dann in die Eingabeschicht des Netzwerkes übergeben werden. Nach vielen verschiedenen Arten von Schichten gelangt das

Signal schlussendlich in die Ausgabeschicht, die das Klassifikationsergebnis beinhaltet. Folgender Programmausschnitt zeigt den entsprechenden Quellcode.

```
1 blob = cv.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
2 net.setInput(blob)
3 outs = net.forward(outputlayers)
```

Die Variable *outs* beinhaltet alle Knoten des Netzwerks in der Ausgabeschicht und innerhalb jedes Knotens hat man wiederum Zugriff auf wichtige Informationen für den weiteren Programmverlauf. Wichtige Informationen sind die Klassenidentitäten und Wahrscheinlichkeiten für eine spezifische Klasse. Die Datei *coco.names* speichert eine fixe Liste an Klassennamen, die von dem neuronalen Netz zum identifizieren der Klassen verwendet wird. Bei der zweiten Variable handelt es sich um einen Wahrscheinlichkeitswert, der angibt, mit welcher Sicherheit der Algorithmus ein Objekt mit einer Klasse in Verbindung setzt. Dieser Wert kann in weiterer Folge dazu verwendet werden, um schlechte bzw. unsichere Klassifizierungen aussortieren zu können. Folgender Programmausschnitt illustriert das in diesem Absatz erklärte Vorgehen.

```
1 for out in outs:
2     for detection in out:
3         scores = detection[5:]
4         class_id = np.argmax(scores)
5         confidence = scores[class_id]
6         if confidence > 0.5:
7             # do something
```

In der Variable *detection* werden die einzelnen Knoten in der Ausgabeschicht angesprochen und die Variable *confidence* speichert die Wahrscheinlichkeiten pro Objekt. Die If-Bedingung schließt aus, dass Objekte, bei denen die Klassensicherheit unter 50% fällt, ausgeschlossen werden.

Pro Ausgangsknoten können sowohl Höhe und Breite, als auch konkrete Punkte des Objektrahmens berechnet werden. Die linke obere Ecke des Rahmens kann man anhand des Mittelpunktes minus der halben Länge bzw. der Breite des Rahmens ausrechnen. Die Skizze in Abbildung 4.3 zeigt die Berechnungen.

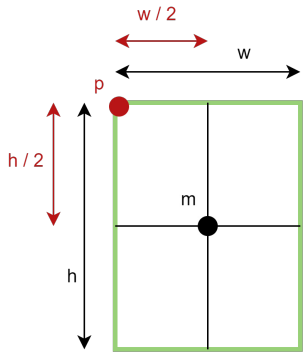


Abbildung 4.3: Die schwarzen Elemente im Bild sind gegeben und die Roten werden ausgerechnet, um den grünen Rahmen zu zeichnen.

Nachfolgend ist der Quellcode zum Extrahieren des linken oberen Punktes, Länge und Breite des Rahmens ersichtlich.

```
1 # center point
2 center_x = int(detection[0] * width)
3 center_y = int(detection[1] * height)
4
5 # width, height
6 w = int(detection[2] * width)
7 h = int(detection[3] * height)
8
9 # upper left point
10 x = int(center_x - w / 2.0)
11 y = int(center_y - h / 2.0)
```

Zuletzt wird nur noch die Funktion benötigt, mit der man den Rahmen zeichnet. Über jede gezeichnete Box wird auch noch die Wahrscheinlichkeit für das Objekt angezeigt. Folgender Codeausschnitt realisiert diese Funktionalität.

```
1 cv.rectangle(img, (x, y), (x + w, y + h), colors[class_ids[i]], 2)
2 cv.putText(img, label + ": " + str(int(round(confidences[i], 2) * 100)) +
3            "\%", (x, y - 10), cv.FONT_HERSHEY_SIMPLEX, 0.5,
4            colors[class_ids[i]], 2)
```

Fügt man diese Codeteile zusammen, dann hat man schon einen funktionierenden Prototypen, der verschiedene Klassen erkennen kann. In Abbildung 4.4 ist der Prototyp im Einsatz zu sehen. Ein Problem, das schnell auffällt ist, dass pro Objekt mehrere Rahmen gezeichnet werden. Dieses Problem wird im nächsten Abschnitt behandelt.

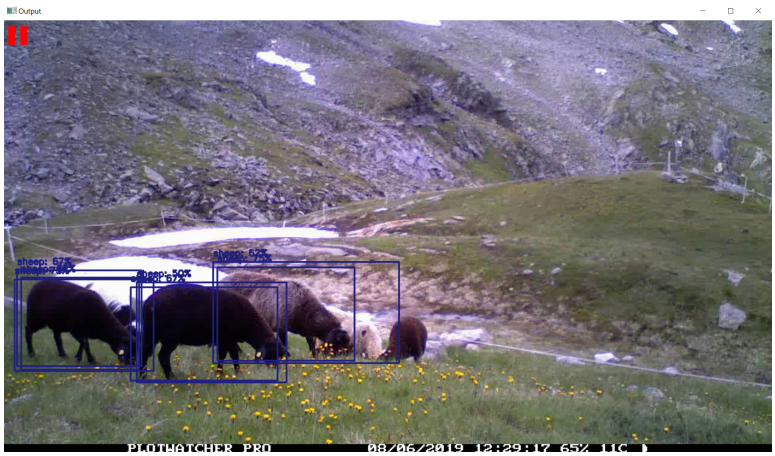


Abbildung 4.4: Ein Ausschnitt aus dem Video mit Klassenerkennung.

4.3 Ergebnisqualität der Objekterkennung verbessern

Wie schon im Kapitel Grundlagen beschrieben, können unnötige Begrenzungsrahmen mit Nicht-Maxima-Unterdrückung reduziert werden. Um den NMS Algorithmus anwenden zu können, müssen pro Bild drei verschiedene Listen gespeichert werden. Es müssen alle Boxen mit den Wahrscheinlichkeiten und den Klassenidentitäten gespeichert werden, wie im folgenden Programmabschnitt definiert.

```
1 boxes = []
2 class_ids = []
3 confidences = []
```

Nachdem in weiterer Folge diese Listen für ein aktuelles Bild befüllt worden sind, wird die NMS Funktion von OpenCV angewandt, wie im folgenden Ausschnitt in Zeile 2 dargestellt. Dabei gibt *0.5* den Schwellwert für die Wahrscheinlichkeiten und *0.4* den im Kapitel Grundlagen besprochenen NMS Schwellwert an. In den Zeilen 4 bis 12 werden nur jene Boxen schlussendlich auf das Bild gezeichnet, die von der NMS Methode gefiltert wurden.

```
1  # Non-Maximum supression to reduce number of boxes
2  indices = cv.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
3  # loop through all boxes on current image and draw them
4  for i in range(len(boxes)):
5      if i in indices:
6          x, y, w, h = boxes[i]
7          label = str(classNames[class_ids[i]])
8
9          cv.rectangle(img, (x, y), (x + w, y + h), colors[class_ids[i]], 2)
10         cv.putText(img, label + ": " + str(int(round(confidences[i], 2) * 100)) +
11                    "%", (x, y - 10), cv.FONT_HERSHEY_SIMPLEX, 0.5,
12                    colors[class_ids[i]], 2)
```

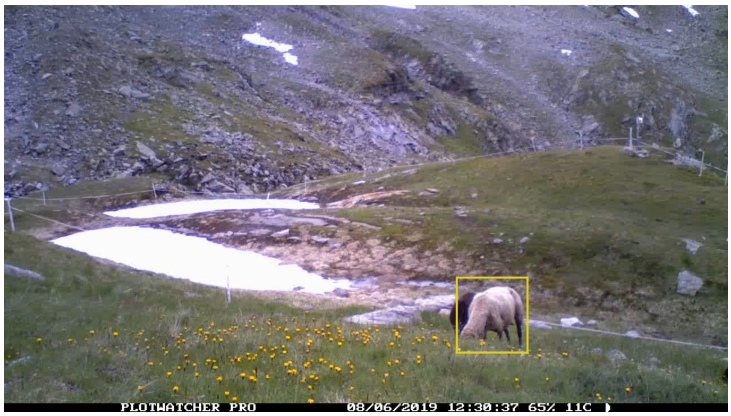
Der zuvor erstellte Programm-Prototyp kann erweitert werden, sodass nun die Funktionalität der Nicht-Maxima-Unterdrückung mit inbegriffen ist. Abbildung 4.5 zeigt einen Schnappschuss des neuen Prototypen in Aktion.

Abbildung 4.5: Ein Ausschnitt aus dem Video mit verbesserter NMS Klassifizierung.

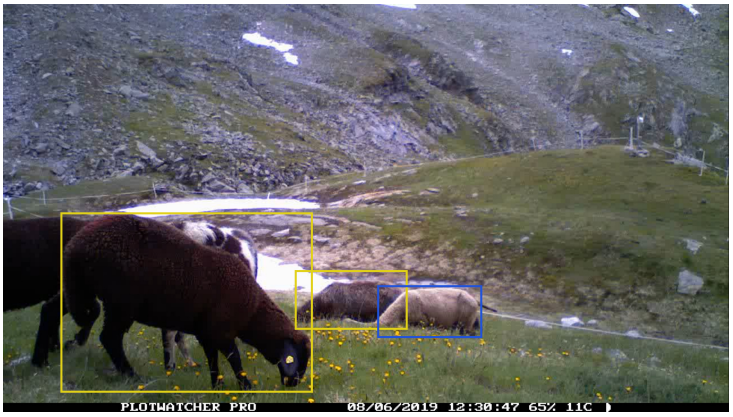
4.4 Einsetzen des CSRT Algorithmus

Wie im Kapitel Grundlagen bereits angedeutet, soll das Programm auch Tiere in den Videos möglichst akkurat zählen können, um statistische Auswertungen über den Lebensraum der Wildtiere des Nationalparks Hohe Tauern erheben zu können. Hierfür wird eine Methode benötigt, die über mehrere Bilder des Videos hinweg bestimmen kann, ob es sich um dasselbe Tier handelt. Tracker (dt.: Verfolger) sind Algorithmen, die ein konkretes Objekt über mehrere Bilder „verfolgen“ können.

Für die Funktionalität des Zählens ist aber nun nicht nur ein einziger Tracker notwendig, sondern es muss für jedes zu verfolgende Objekt eine weitere Instanz eines Trackers initialisiert und aktualisiert werden. Ein zusätzlich wichtiger Aspekt ist, dass bei einem aktuellen Bild schon eine Menge an Objekten vom Tracker verfolgt werden können und somit für jene Objekte kein neuer Tracker mehr initialisiert werden soll. Um diese Situation im Programm zu beachten, werden alle bereits verfolgten Begrenzungsrahmen in eine Liste gespeichert und mittels dieser Liste überprüft, ob es sich bei einem Objekt um ein schon verfolgtes Objekt handelt. Wenn das Objekt in Frage sich nicht in unmittelbarer Nähe eines schon verfolgten Begrenzungsrahmens befindet, wird ein neuer Tracker initialisiert und der Liste aller Tracker hinzugefügt. Bei jedem neuen Tracker wird eine Zählervariable erhöht, die dann die Anzahl der Tiere pro Klasse angibt. Abbildung 4.6 illustriert den eben beschriebenen Vorgang.



(a)



(b)

Abbildung 4.6: (a) Das erste Bild, bei dem für das weiße Schaf ein Tracker initialisiert wird. (b) Das darauf folgende Bild, bei dem zwei neue Tracker (gelb) initialisiert werden und das weiße Schaf vom vorherigem Bild schon verfolgt wird (blau). Der Zähler für die Klasse Schaf wird also nur bei den gelben Boxen erhöht.

Zuerst muss eine Liste, die alle Tracker speichert, definiert und eine Liste aller schon verfolgter Boxen erstellt werden.

```
1 trackers = []
2 staticValues.tracking_boxes = []
```

Als nächstes wird eine Funktion implementiert, die die Ähnlichkeit zweier Boxen in Bezug auf deren Distanz zueinander vergleicht. Es wird immer die aktuelle Box mit allen vorzeitig verfolgten Objekten in der Liste *tracking_boxes* verglichen. Wenn die Boxen eine gewisse Ähnlichkeit aufweisen, kann der Algorithmus bestimmen, dass es sich bei der aktuellen Box um ein schon verfolgtes Objekt handelt. Die Funktion sieht

dabei wie folgt aus.

```
def IsValidBox(box):
    buffer = 75
    for tr in staticValues.tracking_boxes:
        if tr[0] - buffer < box[0] < tr[0] + buffer and
        tr[1] - buffer < box[1] < tr[1] + buffer and
        tr[2] - buffer < box[2] < tr[2] + buffer and
        tr[3] - buffer < box[3] < tr[3] + buffer:
            return False
    return True
```

Je höher die *buffer* Variable ist, desto toleranter wird der Algorithmus beim Vergleichen der Begrenzungsrahmen.

Für die Zählervariablen der einzelnen Klassen wird eine Klasse *staticValues* erstellt, die unter Anderem auch die Liste aller verfolgten Boxen *tracking_boxes* beinhaltet.

```
class staticValues:
    animals = []
    frames = []
    videos = []
    tracking_boxes = []
    fastSelected = True
    slowSelected = False
    AllAnimals = 0
    Birds = 0
    Cats = 0
    Dogs = 0
    Horses = 0
    Sheeps = 0
    Cows = 0
```

Einige Variablen in dieser Klasse werden für das nachfolgende Kapitel relevant.

Das Erhöhen der einzelnen Zählervariablen erfolgt mit der eben beschriebenen Variable *IsValidBox*. Mit dem Erhöhen einer Zählervariable wird natürlich gleichzeitig auch ein neuer Tracker initialisiert.

```
if IsValidBox(box):
# Object tracking

    if classNames[class_id] == "bird":
        staticValues.Birds += 1
        staticValues.AllAnimals += 1
        print(classNames[class_id] + ": " + str(staticValues.Birds))
    elif classNames[class_id] == "cat":
        staticValues.Cats += 1
        staticValues.AllAnimals += 1
        print(classNames[class_id] + ": " + str(staticValues.Cats))
    elif classNames[class_id] == "dog":
        staticValues.Dogs += 1
        staticValues.AllAnimals += 1
        print(classNames[class_id] + ": " + str(staticValues.Dogs))
    elif classNames[class_id] == "horse":
        staticValues.Horses += 1
        staticValues.AllAnimals += 1
        print(classNames[class_id] + ": " + str(staticValues.Horses))
    elif classNames[class_id] == "sheep":
        staticValues.Sheeps += 1
        staticValues.AllAnimals += 1
    print(classNames[class_id] + ": " + str(staticValues.Sheeps))
        elif classNames[class_id] == "cow":
        staticValues.Cows += 1
        staticValues.AllAnimals += 1
        print(classNames[class_id] + ": " + str(staticValues.Cows))

    tracker = cv.TrackerCSRT_create()
    tracker.init(img, box)
    trackers.append(tracker)
```

Diese Tracker, gespeichert in der Liste *trackers*, müssen für jedes Bild aktualisiert werden. Falls ein Tracker nicht erfolgreich aktualisiert werden konnte, wird dieser von der Liste der Tracker entfernt.

```
# update trackers every frame
for t in trackers:
    su, bbox = t.update(img)
    if su:
        staticValues.tracking_boxes.append(bbox)
    else:
        trackers.remove(t)
```

Bei Ausführung des Programms sieht die Ausgabe in der Konsole so wie auf Abbildung 4.7 aus.

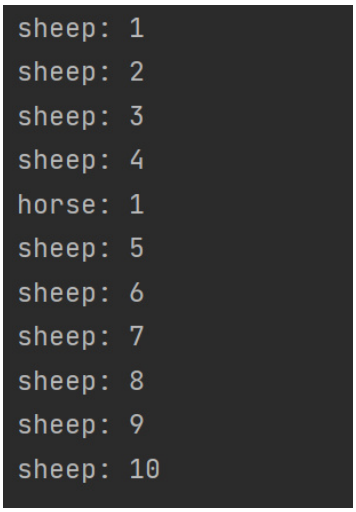


Abbildung 4.7: Das Programm zählt 10 Schafe und 1 Pferd im Video.

4.5 Grafische Benutzeroberfläche (GUI)

Der im vorherigen Kapitel implementierte Prototyp wird nun so erweitert, dass das Anwenden der Funktionalitäten intuitiver für den/die Benutzer/in wird. Das Angeben des Dateipfads ist zum Beispiel jetzt nur durch manuelles Umschreiben des Quellcodes möglich und für den/die durchschnittliche/n Benutzer/in sehr unpraktisch.

Um den entwickelten Prototypen praktikabler zu gestalten, kommt das Framework PySimpleGUI [33] zum Einsatz, mit dem man grafische Benutzeroberflächen in Python erstellen kann. PySimpleGUI ist ein noch relativ neues Paket, deren erste Version im Jahr 2018 veröffentlicht wurde. Vergleicht man dieses Modul mit anderen prominenten GUI Plattformen wie tkinter [34], spricht vor allem die leichte Handhabung für Entwickler für das Modul. Als ersten Schritt wird mittels PySimpleGUI ein einfaches Fenster erzeugt.

```
1 import PySimpleGUI as ps
2
3 ps.theme('DarkAmber')
4
5 layout = [
6     [ps.Text('Hello World')]
7 ]
8
9 window = ps.Window("Animal detector", layout)
10
11 # GUI main loop
12
13 while True:
14     event, values = window.read()
15     if event == ps.WIN_CLOSED or event == 'Cancel':   # if user closes window or clicks
       cancel
16         break
17
18 window.close()
19 cv.destroyAllWindows()
```

In Zeile 9 wird das Hauptfenster definiert und die Liste *layout* gibt dabei den strukturellen Aufbau der GUI an. Die While-Schleife in Zeile 13 ist für die ständige Aktualisierung des Fensters verantwortlich und fängt auch gewisse Ereignisse, wie in Zeile 15, ab, welche auch von dem/der Entwickler/in definiert werden können. Die Konsolenausgabe des Programmes ist in Abbildung 2.8 zu sehen. Anschließend soll ein Knopf und eine Text-

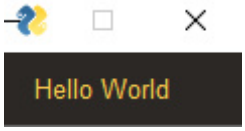


Abbildung 4.8: Ein einfaches „Hello World“ Fenster.

zeile eingerichtet werden, mit denen man einen Ordner auf dem Computer auswählen und in das Programm laden kann. Dazu werden zusätzlich zur *layout* Liste noch zwei andere Listen initialisiert und somit das Fenster der GUI in zwei Bereiche aufgeteilt. Auf der linken Seite des Fensters sieht man dann die Bedienelemente zum Auswählen des Ordners und darunter werden noch alle relevanten Dateien innerhalb des Ordners angezeigt. Zuerst wird das vorher erstellte „Hello World“ Programm um die besprochenen grafischen Elemente erweitert.

Zuerst wird die linke Seite der GUI mit der nachfolgenden Liste definiert. In Zeile 6 ist ein Textfeld definiert mit dem Argument *enable_events=True*, das die Ereignisabfrage dieses GUI Elements ermöglicht. Mit dem *key=„-FOLDER-“* definiert man den Namen des Ereignisses, das in der Hauptschleife abgefragt werden kann. Etwas weiter unten in Zeile 10 ist die Ansicht definiert, die alle Videos innerhalb eines gewählten Ordners auflistet. Der Parameter *select_mode=„extended“* aktiviert die Möglichkeit zur Auswahl von mehreren Videos in der Liste.

```
1 file_frame = [
2     [
3         ps.Text("Load Video or folder"),
4         ps.In(enable_events=True, key="-FOLDER-"),
5         ps.FolderBrowse()
6     ],
7     [
8         ps.Listbox(values=[], size=(70,20), enable_events=True, key="-FILE LIST-",
      select_mode="extended")
9     ]
10 ]
```

Die rechte Seite der GUI wird so aufgebaut, dass in Zeile 3 des folgenden Ausschnittes, die Anzeige für das Zählen der Tiere definiert wird. In Zeile 9 und 10 werden die zwei Knöpfe zum Auswählen des jeweiligen Modells angegeben. Der Rest des Codes erstellt einen Knopf für das Exportieren der Ergebnisse des Algorithmus in eine CSV- oder TXT-Datei.

```
1 video_viewer = [
2     [
3         ps.Text("All animals count: \nBird count: \nCat count: \nDog count: \nHorse
      count: \n" +
4                 "Sheep count: \nCow count: \nElephant count: \nBear count: \n" +
5                 "Zebra count: \nGiraffe count: \n", key="-STATS-"),
6     ],
7     [
8         ps.Text("Press 'r' to pause/resume the video\nPress 's' to step\nPress 'esc'
       to exit")
9     ],
10    [
11        ps.Radio("Slow", key="-SLOW-", enable_events=True, group_id=0),
12        ps.Radio("Fast", key="-FAST-", enable_events=True, group_id=0, default=True)
13    ],
14    [
15        ps.Button("Play", key="-PLAY-", enable_events=True),
16        ps.InputText(visible=False, enable_events=True, key='fig_path'),
17        ps.FileSaveAs
18        (
19            button_text="Export data as...",
20            key="-FILE SAVE-",
21            file_types=(('CSV', '.csv'), ('TXT', '.txt')),
22            enable_events=True
23        )
24    ]
25 ]
```

Der letzte Schritt besteht aus dem Zusammenfügen der zwei Hälften der GUI zu einem ganzen Fenster.

```
layout = [
    [
        ps.Column(file_frame),
        ps.VSeparator(),
        ps.Column(video_viewer)
    ]
]
```

Wird das Programm nun gestartet, sind GUI Elemente wie Knöpfe und Textfelder zu sehen, diese Bedienelemente reagieren aber noch auf keine Benutzereingabe, da ihnen noch keine Funktionalität zugewiesen worden ist.

Der nächste Schritt ist nun die Logik der Bedienelemente zu realisieren, wozu vor allem die Hauptschleife der GUI um entsprechende Ereignisse erweitert wird. Zwischen Zeile 1 und 8 werden die Anzahlen an verschiedenen Tieren, zu sehen auf der rechten Seite der grafischen Benutzeroberfläche, aktualisiert.

```
window["-STATS-"].update("All animals: " +
str(staticValues.AllAnimals) +
"\nBirds: " + str(staticValues.Birds) +
"\nCats: " + str(staticValues.Cats) +
"\nDogs: " + str(staticValues.Dogs) +
"\nHorses: " + str(staticValues.Horses) +
"\nSheeps: " + str(staticValues.Sheeps +
"\nCows: " + str(staticValues.Cows) + "\n")
```

In den nächsten Zeilen wird überprüft, ob der/die Anwender/in das YOLOv3-608 („-SLOW-“) oder das YOLOv3-tiny Modell ausgewählt hat.

```
if event == "-SLOW-":
    staticValues.slowSelected = True
    staticValues.fastSelected = False
elif event == "-FAST-":
    staticValues.fastSelected = True
    staticValues.slowSelected = False
```

Einer der wichtigsten Abfragen befindet sich in dem folgendem Abschnitt, denn beim Klicken des „Play“ Knopfes wird das Video mit Objekterkennung und Verfolgung abgespielt, wie in diesem Kapitel schon im Detail erläutert.

```
if event == "-PLAY-":
    if inVids:
        for vid in inVids:
            window.disable()
            play_video(vid)
            window.enable()
    else:
        ps.popup_error("Please choose a file")
```

Zuletzt wird in Zeile 3 eine Funktion aufgerufen, die die Ergebnisse der Objekterkennung

und Objektverfolgung in eine CSV Datei exportiert. Die restlichen Zeilen Code sind für das Extrahieren der Videos aus einem gegebenen Ordner und die Auflistung dieser Videos auf der Benutzeroberfläche zuständig.

```
1 if event == 'fig_path' and values['fig_path'] != '':
2     print(values['-FILE SAVE-'])
3     ExportToCSV(values['-FILE SAVE-'])
4 
5 if event == "-FOLDER-":
6     folder = values["-FOLDER-"]
7     try:
8         file_list = os.listdir(folder)
9     except:
10        file_list = []
11 
12    names = [
13        f
14        for f in file_list
15        if os.path.isfile(os.path.join(folder, f))
16        and f.lower().endswith((".wmv", ".avi", ".mp4"))
17    ]
18 
19    window["-FILE LIST-"].update(names)
20 
21 if event == "-FILE LIST-":
22    inVids = []
23    try:
24        for i in values["-FILE LIST-"]:
25            filename = os.path.join(values["-FOLDER-"],i)
26            inVids.append(filename)
27    except:
28        pass
```

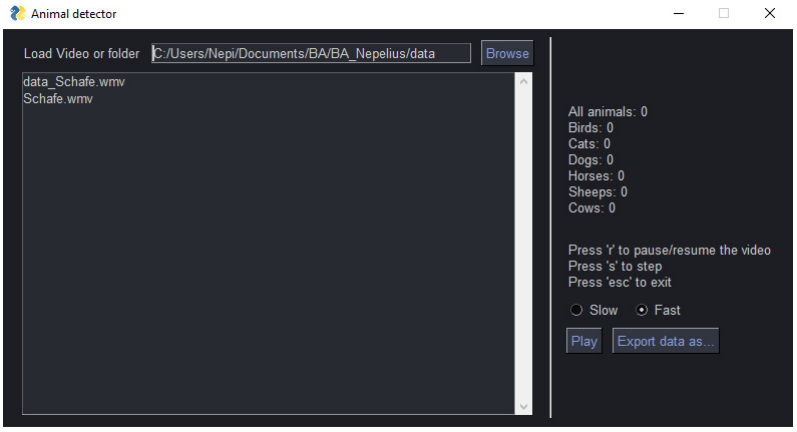


Abbildung 4.9: Ausschnitt aus der finalen grafischen Benutzeroberfläche.

Kapitel 5

Tests und Analysen

5.1 Begriffserklärung Metriken

In diesem Abschnitt werden die Metriken Genauigkeit (Engl.: accuracy), Präzision (Engl.: precision), Sensitivität (Engl.: sensitivity), der F1 und der Jaccard Wert geklärt [35] [42], um die folgenden Ergebnisse der Objekterkennung bzw. Objektdetektion diskutieren zu können.

Die Genauigkeit ist ein Wert, der den Anteil an korrekt klassifizierten Objekten unter allen Objekten angibt. Das heißt also je näher der Wert bei 1 liegt, desto mehr Objekte wurden korrekterweise klassifiziert. Die Genauigkeit lässt sich mit folgender Formel ausrechnen, wobei die Abkürzungen so zu verstehen sind: RP (Richtig Positiv), RN (Richtig Negativ), FP (Falsch Positiv) und FN (Falsch Negativ):

$$Accuracy = \frac{RP + RN}{RP + RN + FP + FN}\,. \tag{5.1}$$

Die Präzision gibt an, wie viele unter den korrekten Daten auch positiv sind. Im spezifischen Fall der Tierdetektion gibt der Wert also an, wie viele der korrekt identifizierten Objekte, auch wirklich Tiere sind. Berechnet wird die Präzision folgendermaßen:

$$Precision = \frac{RP}{RP + FP}\,. \tag{5.2}$$

Die Sensitivität geht von allen positiven Daten aus und gibt den Anteil an korrekten Daten unter dieser Menge an. In unserem Fall klassifiziert der Algorithmus eine Menge an Objekten / Tieren, welche die positiven Daten repräsentieren. Unter diesen positiven Daten können aber Tiere auch falsch vom Algorithmus eingestuft worden sein und dieses Verhältnis gibt die Sensitivität an:

$$Sensitivity = \frac{RP}{RP + FN}\,. \tag{5.3}$$

Der harmonische Mittelwert zwischen Präzision und Sensitivität gibt der F1 Wert an. Diese Metrik wird zum Beispiel in einem Fall verwendet, wo falsch Positive und falsch Negative gleich schlecht für ein gegebenes Problem sind:

$$F1 = \frac{precision * sensitivity}{precision + sensitivity}\,. \tag{5.4}$$

Zuletzt noch der Jaccard Wert, der die Überschneidung zweier Mengen angibt. Ähnlich dem IoU-Verfahren wird hier die Schnittmenge der Grundwahrheit mit dem Ergebnis des Algorithmus bestimmt und ein entsprechender Ähnlichkeitswert berechnet:

$$Jaccard = \frac{RP}{RP + FN + FP}\,. \tag{5.5}$$

5.2 Performanz der Objektdetektion

In diesem Abschnitt werden die Ergebnisse der Objekterkennung getestet und analysiert. Als Metriken werden die eben erklärten Verfahren, wie Genauigkeit, Präzision, Sensibilität, F1 und Jaccard Wert eingesetzt. Die Grundwahrheit wird für jedes einzelne Bild in einem Wildtierkameravideo erhoben und für die Analyse der neuronalen Netze, wird immer der Mittelwert der Ergebnisse aus drei Videos ausgerechnet. Getestet wird einerseits das YOLOv3-608 und andererseits die komprimierte Variante des YOLOv3 Modells (YOLOv3-tiny). Für beide Modelle werden jeweils die Metriken für die Performanz der Klassifikation und der allgemeinen Objekterkennung kalkuliert. Letzteres bedeutet hierbei, dass nur zwischen Vorder- und Hintergrund des Bildes unterschieden wird, ohne dass Klassen bei der Detektion berücksichtigt werden. Metriken dieser Art geben darüber Auskunft, wie gut das Modell beim generellen Erkennen von Objekten abschneidet.

In Tabelle 5.1 werden die pro Objekt erkannten Klassen beachtet und in die einzelnen Metriken miteinbezogen. Die Werte wurden beim Vergleich von Grundwahrheit und dem YOLOv3-tiny Modell berechnet. Anhand der Tabelle 5.1 kann man ablesen, dass der Algorithmus beim Beachten der Klassen schlechter abscheidet als bei der reinen Objekterkennung.

Beim Analysieren der Metriken für das YOLOv3-608 Modells ist das Ergebnis in Tabelle 5.2 zu sehen.

	Klassen werden nicht beachtet	Klassen werden beachtet
accuracy	0.745	0.580
precision	1.000	0.495
sensitivity	0.512	0.373
F1	0.600	0.397
Jaccard	0.512	0.285

Tabelle 5.1: Die Tabelle zeigt die Performanz des Algorithmus bei Beachtung bzw. nicht Beachtung der Klassifizierungen.

	Klassen werden nicht beachtet	Klassen werden beachtet
accuracy	0.800	0.445
precision	0.980	0.470
sensitivity	0.577	0.324
F1	0.639	0.276
Jaccard	0.563	0.167

Tabelle 5.2: Vergleich wie bei Tabelle 5.1, diesmal mit dem YOLOv3-608 Modell.

Performanz der Objektverfolgung

In dieser Passage wird die Performanz der Objektverfolgung erhoben, indem die gezählten Objekte gegenüber gestellt werden. Es werden immer Grundwahrheit mit dem YOLOv3-tiny oder dem YOLOv3-608 Modell verglichen.

Im ersten Video „Schafe“ ist eine kleinere Horde an Schafen zu sehen, die knapp vor der Wildtierkamera vorbeizieht. Als Grundwahrheit wurden 11 Schafe erhoben und vom Algorithmus 13, wie Tabelle 5.3 veranschaulicht.

	Gezählte Tiere	
	Grundwahrheit	Algorithmus
Alle Tiere	11	13
Schafe	11	13

Tabelle 5.3: Vergleich Grundwahrheit und Algorithmus mit dem YOLOv3-tiny Modell beim Zählen der Tiere im Video „Schafe“.

Bei Verwendung des YOLOv3-608 Modells fällt die Zählung noch besser aus, denn nun werden 10 Schafe, 1 Pferd und 11 Tiere insgesamt vom Algorithmus angegeben, wie in Tabelle 5.4 zu sehen.

Bei den nächsten zwei Tabellen 5.5 und 5.6 wird wieder die Grundwahrheit und Algorithmus miteinander verglichen, einmal mit dem YOLOv3-tiny und dem YOLOv3-

	Gezählte Tiere	
	Grundwahrheit	Algorithmus
Alle Tiere	11	11
Schafe	11	10
Pferde	0	1

Tabelle 5.4: Vergleich Grundwahrheit und Algorithmus mit dem YOLOv3-608 Modell beim Zählen der Tiere im Video „Schafe“.

608 Modell. Im nun analysierten Video „Schafhorde“ befindet sich eine große Schafhorde mit 24 Tieren, die größtenteils eine relativ lange Distanz zur Kamera halten.

	Gezählte Tiere	
	Grundwahrheit	Algorithmus
Alle Tiere	24	19
Schafe	24	11
Kühe	0	7
Vögel	0	1

Tabelle 5.5: Vergleich Grundwahrheit und Algorithmus mit dem YOLOv3-tiny Modell beim Zählen der Tiere im Video „Schafhorde“.

	Gezählte Tiere	
	Grundwahrheit	Algorithmus
Alle Tiere	24	40
Schafe	24	7
Pferde	0	12
Kühe	0	21

Tabelle 5.6: Vergleich Grundwahrheit und Algorithmus mit dem YOLOv3-608 Modell beim Zählen der Tiere im Video „Schafhorde“.

Abschließend noch das Video „Gams“, bei dem ein einziges Tier zu sehen ist, das sich langsam der Wildtierkamera nähert. Das Vorgehen bei der Analyse ist wieder dasselbe wie bei den vorherigen Tabellen.

	Gezählte Tiere	
	Grundwahrheit	Algorithmus
Alle Tiere	1	3
Schafe	0	1
Pferde	1	2

Tabelle 5.7: Vergleich Grundwahrheit und Algorithmus mit dem YOLOv3-tiny Modell beim Zählen der Tiere im Video „Gams“.

	Gezählte Tiere	
	Grundwahrheit	Algorithmus
Alle Tiere	1	2
Pferde	1	2

Tabelle 5.8: Vergleich Grundwahrheit und Algorithmus mit dem YOLOv3-608 Modell beim Zählen der Tiere im Video „Gams“.

5.3 GUI Tests

In diesem Abschnitt wird die implementierte GUI getestet, sodass sichergestellt ist, dass es beim Ausführen des Programms zu keinen unvorhergesehenen Komplikationen kommt. Dazu werden eine Reihe an Testfällen ausgeführt, die verschiedene Situationen prüfen.

5.3.1 Testfall - Keine Datei ausgewählt

Wird der „Play“ Button, ohne davor eine Datei ausgewählt zu haben, gedrückt, dann wird die Fehlernachricht in einem Pop-Up Fenster angezeigt, wie Abbildung 5.1 zeigt.

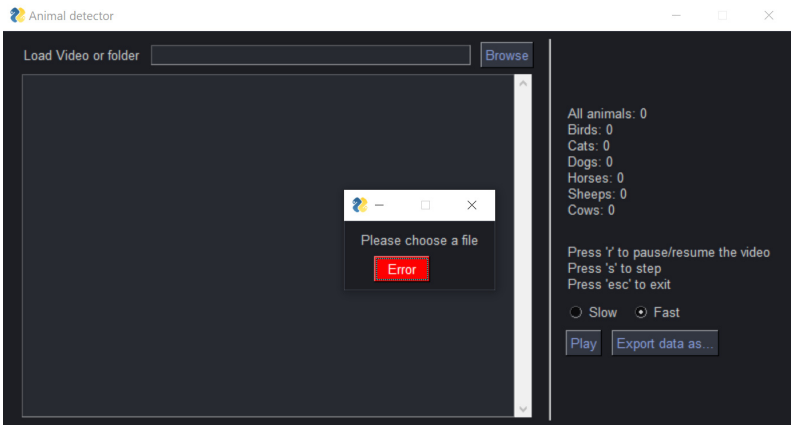


Abbildung 5.1: Situation, wenn der Benutzer keine Datei ausgewählt hat und auf den „Play“ Button drückt.

5.3.2 Testfall - Leere CSV Datei

Gegeben die Situation, dass der/die Benutzer/in noch kein Video abgespielt hat und sich trotzdem dazu entscheidet den „Export data as...“ Button zu klicken, wird schlichtweg eine leere CSV an gegebenen Speicherort erstellt.

5.3.3 Testfall - Standardfall

Wird zuerst die Datei geladen und dann auf den Button „Play“ gedrückt, dann wird das Video gestartet und zeitgleich vom Algorithmus analysiert. In Abbildung 5.2 ist die GUI nach dem Abspielen des Videos zu sehen.

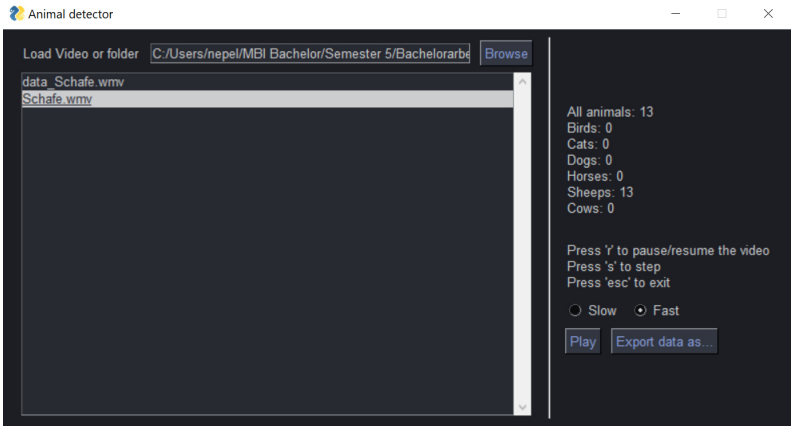


Abbildung 5.2: Der Status der GUI nachdem ein Video abgespielt worden ist. Auf der rechten Seite werden die gezählten Tiere aufgelistet.

5.3.4 Testfall - Mehrere Videos hintereinander abspielen

Auch die Möglichkeit, mehrere Videos mit nur einem Klick auf „Play“ abzuspielen, besteht. Mit L-Shift + Linksklick oder STRG + Linksklick können zusätzliche Videos ausgewählt werden. Diese Funktion soll das Anwenden bei großer Anzahl an Videodaten vereinfachen, wie in Abbildung 5.3 zu sehen. Beim Exportieren der Ergebnisse kommt eine CSV Datei zustande wie in Abbildung 5.4 zu sehen.

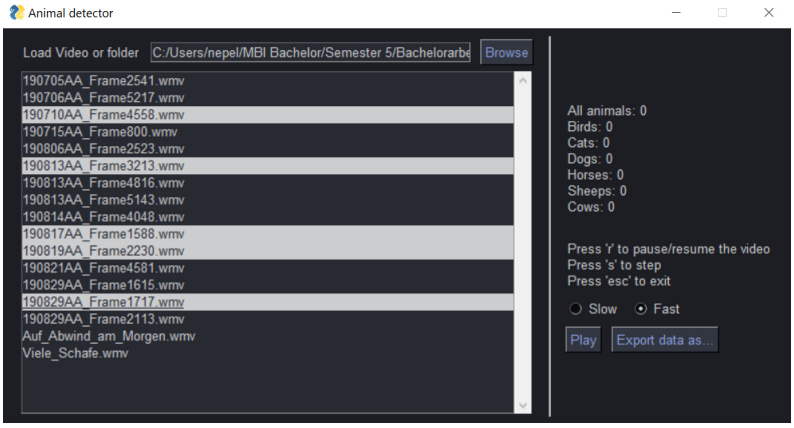


Abbildung 5.3: In der Abbildung wurden fünf Videos ausgewählt, die nach Drücken auf „Play“ abgespielt werden.

	A	B	C	D	E	F	G	H	I	J
1			All Animals	Birds	Cats	Dogs	Horses	Sheeps	Cows	
2	C:/Users/nep	Frame: 1	0	0	0	0	0	0	0	
3	C:/Users/nep	Frame: 2	0	0	0	0	0	0	0	
4	C:/Users/nep	Frame: 3	0	0	0	0	0	0	0	
140	C:/Users/nep	Frame: 39	1	1	0	0	0	0	0	
141	C:/Users/nep	Frame: 40	1	1	0	0	0	0	0	
142	C:/Users/nep	Frame: 41	1	1	0	0	0	0	0	
546	C:/Users/nep	Frame: 45	2	2	0	0	0	0	0	
547	C:/Users/nep	Frame: 46	2	2	0	0	0	0	0	
548	C:/Users/nep	Frame: 47	2	2	0	0	0	0	0	
699	C:/Users/nep	Frame: 198	3	3	0	0	0	0	0	
700	C:/Users/nep	Frame: 199	3	3	0	0	0	0	0	
701	C:/Users/nep	Frame: 200	3	3	0	0	0	0	0	
702										

Abbildung 5.4: Nach Drücken auf „Export data as...“ werden die Ergebnisse aller Videos in einer CSV Datei gespeichert. Die zwei linken Spalten geben den Speicherort des Videos und pro Video das jeweilige Bild im Video an. Es wird also pro Bild im Video angegeben, wie viele Tiere welcher Art gefunden worden sind.

Kapitel 6

Diskussion

Die durchgeführten Analysen aus Kapitel 5 werden in diesem Kapitel diskutiert und interpretiert. Zuerst werden die Resultate der Objektdetektion und anschließend der Objektverfolgung untersucht.

6.1 Objektdetektion

Wenn man die zwei Spalten „Klassen werden nicht beachtet“ und „Klassen werden beachtet“ in Tabelle 5.1 vergleicht, dann fällt die Tatsache auf, dass wenn nur Objekte ohne Klassifizierung erkannt werden sollen, die Objektdetektion deutlich besser abschneidet als wenn Klassen bei der Detektion miteinbezogen werden. Dieses Phänomen ist kein Zufall, denn natürlich schneidet der Algorithmus bei der Unterscheidung von Vorder- und Hintergrund besser ab als in einem Klassifikationsszenario. Um gewisse Klassen voneinander unterscheiden zu können, benötigt es nämlich ein auf Daten trainiertes neuronales Netz, das gewisse Eigenschaften von Objektklassen bereits gelernt hat. Anders ist die Situation bei einer einfachen Objekterkennung, welche zum Beispiel auch mit einer Hintergrundsubtraktion unabhängig von Machine Learning gelöst werden kann.

Zwischen den Tabellen 5.1 und 5.2 kann man auch erkennen, dass in der Spalte „Klassen werden nicht beachtet“, die Werte der Metriken bei dem YOLOv3-608 Modell besser als bei dem YOLOv3-tiny Modell ausfallen. Es wäre nachvollziehbar zu behaupten, dass das selbe auch für die zweite Spalte der beiden Tabellen gelten sollte, nun ist aber das YOLOv3-tiny Modell besser als das YOLOv3-608. Erklären könnte man diesen Zusammenhang damit, dass die Modelle eventuell unabhängig voneinander mit verschiedenen Datensätzen trainiert wurden und somit das YOLOv3-tiny bei den Wildtiervideos besser abschneidet.

6.2 Objektverfolgung

Bei der Objektverfolgung wurde pro Video gezählt, wie viele Tiere sich in der Szene aufhalten und wieder eine erhobene Grundwahrheit mit dem Resultat des Algorithmus verglichen.

Vergleicht man YOLOv3-608 und YOLOv3-tiny im ersten Video „Schafe“, dann fallen die Ergebnisse zwischen Grundwahrheit und Algorithmus ziemlich gleich aus,

wie in Tabelle 5.3 und 5.4 zu sehen ist. Hier ist das YOLOv3-608 Modell die bessere Lösung, denn es wurden hier vom Algorithmus 10 statt 11 Schafe erkannt, dafür aber fälschlicherweise ein Schaf als Pferd klassifiziert. In dem analysierten Video ziehen die Schafe relativ knapp an der Wildtierkamera vorbei und sind damit gut sichtbar, was auch die Arbeit des Algorithmus erleichtert.

Im zweiten Video „Schafhorde“ fällt es sowohl dem YOLOv3-608 als auch dem YOLOv3-tiny Modell schwerer, ein akkurates Ergebnis zu erzielen. Vom Algorithmus werden neben den Schafen auch Kühe, Vögel und Pferde erkannt, obwohl nur Schafe zu sehen sind. Die verschlechterte Performanz hat die Begründung, dass die Distanz zwischen Tieren und Kamera in diesem Fall größer ist, als im vorherigen Video. Die kleinen Objekte in Kombination mit einer niedrigen Auflösung und Bildwiederholrate verunsichern den Algorithmus.

Auf dem dritten Video „Gams“ ist nur ein einziges Tier zu sehen. Der vorgeschlagene Algorithmus kann keine spezifischen Tierklassen wie Gams, Murmeltier, Steinbock, usw. erkennen, darum wird fairerweise bei der Grundwahrheit angenommen, dass es sich bei der Gams um ein Pferd handelt. YOLOv3-tiny erkennt hierbei ein Schaf und 2 Pferde, wohingegen YOLOv3-608 etwas besser mit zwei Pferden abschneidet. Auf dem Video nähert sich das Tier langsam der Kamera und ab einer gewissen Distanz ist der Algorithmus dann in der Lage das Tier zu erkennen.

Kapitel 7

Zusammenfassung und Ausblick

In dieser Arbeit wird mit einem Forschungsprototypen gezeigt, dass ein relativ neues Konzept im Bereich der künstlichen Intelligenz wie Machine Learning, auch Anwendung in der Erforschung eines Nationalparks findet. Mittels Tests und Analysen kann auch die Performanz der Implementierung gemessen werden, welche vielversprechende Ergebnisse liefern. Als beste Messwerte für die Objekterkennung wird die Genauigkeit (0.800), Präzision (0.980), Sensibilität (0.577), der F1 (0.639) und der Jaccard (0.563) Wert errechnet. Bei der Objektverfolgung werden in einem optimalen Fall 11 von 11 Tieren korrekt gezählt, von denen 10 von 11 auch korrekt klassifiziert werden.

Trotz der schnell erreichten guten Resultate ist der Prototyp noch weit davon entfernt ein optimales Ergebnis zu liefern und kann in verschiedenen Aspekten verbessert werden. Ein zentraler Punkt wäre das Trainieren eines besseren Modells hinsichtlich der Daten des Nationalpark Hohe Tauern, einerseits um das Erkennen von spezifischen Tierarten wie Gams, Murmeltier, Steinbock, usw. und andererseits um die generelle Präzision der Objekterkennung und Klassifikation zu steigern. Im Kapitel Tests und Analysen wurde auch die Erkenntnis gewonnen, dass Tiere mit einer größeren Distanz zur Kamera vom Algorithmus kaum erkannt werden. Die Autoren der YOLOv3 Architektur erwähnen auch in ihren Publikationen, dass der Algorithmus Schwierigkeiten mit der Erkennung von Gruppierungen kleiner Objekte hat. Aus diesem Grund könnte man auch die Architektur des Deep Learning Modells zum Beispiel mit einem RCNN austauschen, das zwar die Laufzeit verlängert, dafür aber etwas genauer bei der Objekterkennung und Klassifikation abschneidet.

Die Lösung der Aufgabenstellung des Nationalpark Hohe Tauern kann auch ohne neuronale Netze, stattdessen mit traditionellen Computer Vision Algorithmen umgesetzt werden. Der Vorteil eines solchen Ansatzes ist eine wesentlich präzisere und auch schnellere Erkennung von sich bewegenden Objekten in Videos. Der große Nachteil hierbei ist jedoch, dass die Information der spezifischen Objektklassen verloren geht, welche durchaus im Interesse der Erforschung der Lebensräume der Tiere liegt. Aus diesem Grund wurde für diese Arbeit ein Ansatz mit neuronalen Netz gewählt.

Quellenverzeichnis

Literatur

[1] Keshav Bimbraw. „Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology“. In: *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*. Bd. 01. 2015, S. 191–198 (siehe S. 1).

[2] Mikael Boden. „A guide to recurrent neural networks and backpropagation“. *the Dallas project* (2002) (siehe S. 3).

[3] Samarth Brahmbhatt. „Introduction to Computer Vision and OpenCV“. In: *Practical OpenCV*. Berkeley, CA: Apress, 2013, S. 3–5. URL: https://doi.org/10.1007/978-1-4302-6080-6_1 (siehe S. 4).

[4] Bryan Chung. *Pro Processing for Images and Computer Vision with OpenCV*. Apress, 2017. URL: https://doi.org/10.1007%2F978-1-4842-2775-6 (siehe S. 2).

[5] Navneet Dalal und Bill Triggs. „Histograms of oriented gradients for human detection“. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Bd. 1. IEEE. 2005, S. 886–893 (siehe S. 2).

[6] Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan und Michael Felsberg. „Learning spatially regularized correlation filters for visual tracking“. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, S. 4310–4318 (siehe S. 9).

[7] Konstantinos Derpanis. „The harris corner detector“. *York University* 2 (2004) (siehe S. 2).

[8] Vandit Gajjar, Yash Khandhediya und Ayesha Gurnani. „Human Detection and Tracking for Video Surveillance: A Cognitive Science Approach“. In: *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*. 2017, S. 2805–2809 (siehe S. 1).

[9] Reagan Galvez, Argel Bandala, Elmer Dadios, Ryan Rhay Vicerra und Jose Martin Maningo. „Object detection using convolutional neural networks“. In: *TENCON 2018-2018 IEEE Region 10 Conference*. IEEE. 2018, S. 2023–2027 (siehe S. 1).

[10] Sunila Gollapudi. „OpenCV with Python“. In: *Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs*. Berkeley, CA: Apress, 2019, S. 31–50. URL: https://doi.org/10.1007/978-1-4842-4261-2_2 (siehe S. 4).

[11] G Hemalatha und CP Sumathi. „A study of techniques for facial detection and expression classification“. *International Journal of Computer Science and Engineering Survey* 5.2 (2014), S. 27 (siehe S. 1).

[12] Berthold Horn und Brian Schunck. „Determining optical flow“. *Artificial intelligence* 17.1-3 (1981), S. 185–203 (siehe S. 2).

[13] Amila Jakubović und Jasmin Velagić. „Image feature matching and object detection using brute-force matchers“. In: *2018 International Symposium ELMAR*. IEEE. 2018, S. 83–86 (siehe S. 2).

[14] Hamed Kiani Galoogahi, Terence Sim und Simon Lucey. „Correlation filters with limited boundaries“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, S. 4630–4638 (siehe S. 9).

[15] Shimiao Li. „A review of feature detection and match algorithms for localization and mapping“. In: *IOP Conference Series: Materials Science and Engineering*. Bd. 231. 1. IOP Publishing. 2017, S. 012003 (siehe S. 2).

[16] Rainer Lienhart und Jochen Maydt. „An extended set of haar-like features for rapid object detection“. In: *Proceedings. International Conference on Image Processing*. Bd. 1. IEEE. 2002, S. I–I (siehe S. 2).

[17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan und Serge Belongie. „Feature Pyramid Networks for Object Detection“. *CoRR* abs/1612.03144 (2016). arXiv: 1612.03144. URL: http://arxiv.org/abs/1612.03144 (siehe S. 3).

[18] David Lowe. „Distinctive image features from scale-invariant keypoints“. *International Journal of Computer Vision* 60.2 (2004), S. 91–110 (siehe S. 2).

[19] Alan Lukezic, Tomas Vojir, Luka ˇCehovin Zajc, Jiri Matas und Matej Kristan. „Discriminative correlation filter with channel and spatial reliability“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, S. 6309–6318 (siehe S. 8).

[20] Alan LuNežič, Tomáš Vojıř, Luka Čehovin Zajc, Jiřı Matas und Matej Kristan. „Discriminative correlation filter TracNer with channel and spatial reliability“. *International Journal of Computer Vision* 126.7 (2018), S. 671–688 (siehe S. 9).

[21] David Mount, Nathan Netanyahu und Jacqueline Le Moigne. „Efficient algorithms for robust feature matching“. *Pattern Recognition* 32.1 (1999), S. 17–38 (siehe S. 2).

[22] Marius Muja und David Lowe. „Fast approximate nearest neighbors with automatic algorithm configuration.“ *VISAPP (1)* 2.331-340 (2009), S. 2 (siehe S. 2).

[23] Vladimir Nasteski. „An overview of the supervised machine learning methods“. *HORIZONS.B* 4 (Dez. 2017), S. 51–62 (siehe S. 3).

[24] Keiron O'Shea und Ryan Nash. „An Introduction to Convolutional Neural Networks“. *ArXiv e-prints* (Nov. 2015) (siehe S. 3).

[25] Joseph Redmon, Santosh Divvala, Ross Girshick und Ali Farhadi. „You Only Look Once: Unified, Real-Time Object Detection“. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, S. 779–788 (siehe S. 2, 5, 6).

[26] Edward Rosten und Tom Drummond. „Machine learning for high-speed corner detection“. In: *European Conference on Computer Vision*. Springer. 2006, S. 430–443 (siehe S. 2).

[27] Daniel Svozil, Vladimir Kvasnicka und Jiri Pospichal. „Introduction to multi-layer feed-forward neural networks“. *Chemometrics and intelligent laboratory systems* 39.1 (1997), S. 43–62 (siehe S. 3).

[28] Joost Van De Weijer, Cordelia Schmid, Jakob Verbeek und Diane Larlus. „Learning color names for real-world applications“. *IEEE Transactions on Image Processing* 18.7 (2009), S. 1512–1523 (siehe S. 9).

[29] Wikipedia. *Computer Vision — Wikipedia, die freie Enzyklopädie*. [Online; Stand 17. November 2021]. 2021. URL: https://de.wikipedia.org/w/index.php?title=Computer_Vision&oldid=216733361 (siehe S. 1).

[30] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu und Xindong Wu. „Object detection with deep learning: A review“. *IEEE Transactions on Neural Networks and Learning Systems* 30.11 (2019), S. 3212–3232 (siehe S. 2).

Software

[31] *ImageNet*. URL: https://www.image-net.org/update-mar-11-2021.php (siehe S. 6).

[32] *OpenCV*. URL: https://opencv.org/ (siehe S. 2).

[33] *PySimpleGUI*. URL: https://pysimplegui.readthedocs.io/en/latest/ (siehe S. 23).

[34] *tkinter*. URL: https://docs.python.org/3/library/tkinter.html (siehe S. 23).

Online-Quellen

[35] Sumeet Kumar Agrawal. *Metrics to Evaluate your Classification Model to take the right decisions*. 20. Juli 2021. URL: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/ (besucht am 26. 04. 2022) (siehe S. 28).

[36] The TensorFlow Hub Authors. *Object Detection*. 29. Juli 2021. URL: https://www.tensorflow.org/hub/tutorials/object_detection (besucht am 08. 12. 2021) (siehe S. 3).

[37] Srihari Humbarwadi. *Object Detection with RetinaNet*. 14. Juli 2020. URL: https://keras.io/examples/vision/retinanet/ (besucht am 08. 12. 2021) (siehe S. 3).

[38] IBM. *Convolutional Neural Networks*. 20. Aug. 2020. URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks (besucht am 08. 11. 2021) (siehe S. 3).

[39] Sambasivarao K. *Non-maximum Suppression (NMS)*. 1. Okt. 2019. URL: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c (besucht am 29. 03. 2022) (siehe S. 7).

[40] Thomas Laurent. *Template Matching, How does it work*. 26. Apr. 2021. URL: https://github.com/multi-template-matching/MultiTemplateMatching-Fiji/wiki/Template-Matching%2C-How-does-it-work (besucht am 09. 12. 2021) (siehe S. 3).

[41] OpenCV. *Cascade Classifiers*. 8. Dez. 2021. URL: https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html (besucht am 08. 12. 2021) (siehe S. 3).

[42] Roel Peters. *Performance Metrics: Jaccard Index*. 28. Dez. 2020. URL: https://www.roelpeters.be/glossary/machine-learning-jaccard-index/ (besucht am 26. 04. 2022) (siehe S. 28).

[43] Nationalpark Hohe Tauern. *Tierwelt*. 1. Okt. 2019. URL: https://hohetauern.at/de/natur/tierwelt.html (besucht am 25. 04. 2022) (siehe S. 10).

[44] Deepanshu Tyagi. *Introduction To Feature Detection And Matching*. 3. Jan. 2019. URL: https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d (besucht am 16. 09. 2021) (siehe S. 2).

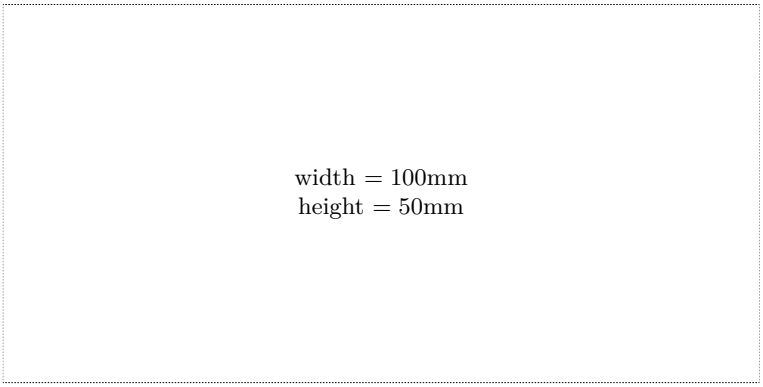
[45] Nathan Zhao. *Understanding Objectness in Object Detection Models*. 14. Aug. 2020. URL: https://medium.com/@zhao.nathan/understanding-objectness-in-object-detection-models-5d8c9d032488 (besucht am 17. 11. 2021) (siehe S. 1).

Abbildungsverzeichnis

Tabellenverzeichnis

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —