Detecting Animals in Challenging Outdoor Environments

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
vorgelegt dem Fachbereich 1 (Physik / Elektrotechnik)
der Universität Bremen

von
Dipl.-Ing. Jens Dede

Gutachterin:	Prof. Dr. Anna Förster	Universität Bremen
Gutachter:	Prof. Dr. Peter Haddawy	Mahidol University
Prüfer:	Prof. Dr. Alberto Garcia-Ortiz	Universität Bremen
Prüfer:	Prof. Dr. Lars Wolf	Technische Universität Braunschweig

Tag des Kolloquiums: 05.03.2024

Bremen, 2. Januar 2024

I assure that this work has been done solely by me without any further help from others except for the official support of the Sustainable Communication Networks group of the University of Bremen. The literature used is entirely listed in the bibliography.

Bremen, Tuesday 2nd January, 2024

(Jens Dede)

Acknowledgments

First, I would like to thank my family, especially my parents, Frank and Martina Dede, for their support and for always offering me a silent place to concentrate on this thesis.

I am very thankful to Anna Förster, who supervised my work, for all the discussions, guidance, suggestions, and especially trust she gave me.

Thank you also to Peter Haddawy for coming to Germany, for the appraisal and for allowing me to work on parts of this work in Thailand.

I would also like to thank my examiners, Lars Wolf and Alberto Garcia-Ortiz.

A special thanks also to all my current and former ComNets colleagues! We all together had a lot of fun, nice events and also interesting discussions.

I have written this thesis the context of the *mAInZaun* project. Thank you to all current and former partners for the workshops, discussions, and time together.

I wrote some parts of this work at the Mahidol University in Thailand. A big thank you to all the people I was working with and especially to Myat Su Yin for all the support.

A big thank you also to Michela Papandrea, Andrea Quattrini, and Davide Nasi from SUPSI, Switzerland, for the time together and all the fruitful and enjoyable discussions.

Thank you also to Wildparks, who allowed us to place our cameras and collect our images: Alternativer Bärenpark Worbis, Wildpark Lüneburger Heide, Wingster Waldzoo.

During this work, I supervised many students, some directly and some indirectly. They helped a lot to form this work.

I received a lot of valuable feedback, had good discussions, and got new ideas for this work from many people, private and professional. A big thank you to all those I have not mentioned here.

This work is part of the *mAInZaun* project supported by the Federal Ministry of Food and Agriculture, Germany under the grand 28DK114B20.

Abstract

Monitoring wildlife is vital for sustainable coexistence between humans, flora, and fauna. Do we have a healthy population? Is an animal near extinction, or did new species arrive in the habitat? Where do the animals stay, hunt, and live? Forest rangers, livestock owners, and other interested people place camera traps to answer those questions. Those camera traps automatically take photos if motion is detected, which easily leads to many images to be analyzed. The quality of those photos often varies and depends on the environment. Animals usually do not stand directly close in front of the lens but are partly covered by the fauna or only barely visible in the distance. Furthermore, precipitation, fog, dirt, and many other conditions additionally decrease the image quality and hinder the detection of animals.

Our interest comes from our *mAInZaun* project. The idea is to place cameras on the poles of pasture fences. Deterrents are automatically activated using artificial intelligence if predators like wolves, golden jackals, stray dogs, etc., are detected. Ideally, this approach should reduce the predator attacks on grazing animals in a non-lethal way. The automatic activation of the deterrents requires a good detection of predators. Both, false alarms and undetected predators, must be prevented and require a good detection model.

Training a model detecting animals requires a certain amount of training data in the form of labeled images. The interest is not only in the clearly visible animals but especially in the hard-to-see ones. Furthermore, the model has to be adapted occasionally to ensure good performance in different environments or if new species have to be detected.

This work addresses those challenges with our *ShadowWolf* framework, which offers an assisted approach for automatically labeling camera trap images. Using state-of-the-art machine-learning algorithms in combination with internet-based crowdsourcing significantly increases the detection of animals and reduces the workload for individual domain experts simultaneously. The outcome is a training dataset that can be used to train arbitrary object detection and classification algorithms. The user can select the appropriate model to run on different devices – from constrained edge devices to a high-end server.

We also collected and analyzed our dataset containing more than 100,000 camera trap images to evaluate our *ShadowWolf* framework. We also discuss deploying a sensor network tailored for our agriculture application.

In the end, this work benefits everybody dealing with real-world wildlife camera trap images: Counting animals, detecting predators, and optimizing automatic machine learning models are simplified and also usable by non-technical people.

Kurzfassung

Für ein nachhaltiges Zusammenleben zwischen Mensch und Tier ist die genaue Beobachtung der Flora und Fauna von fundamentaler Bedeutung. Welche Tiere halten sich in der Umgebung auf? Sind neue Tierarten eingewandert? Wo jagen diese? Förster, Weidetierhalter und andere interessierte Personen sammeln diese Informationen mit Wildkameras. Diese nehmen bei Bewegung automatisch Bilder auf, was schnell zu vielen Fotos führt, die analysiert werden müssen. Die Qualität variiert und hängt stark von der Umgebung ab. Die Tiere stehen häufig nicht direkt vor der Linse, sondern im Hintergrund. Auch die Vegetation kann die Tiere zum Teil verdecken. Zusätzlich variiert die Bildqualität auch noch durch Umgebungseinflüsse wie z.B. Niederschlag, Nebel, Schmutz und viele weitere. Dies erschwert die Erkennung.

Unser Interesse in diesem Bereich stammt aus dem *mAInZaun*-Projekt. Die Idee ist es, Kameras auf Pfosten von Weidezäunen zu platzieren. Vergrämungsstimuli werden automatisch aktiviert, sobald eine künstliche Intelligenz (KI) ein Raubtier wie Wolf, Goldschakal oder einen streunenden Hund erkennt. Im Idealfall reduziert dieser Ansatz die Übergriffe auf Weidetiere. Diese automatische Auslösung setzt eine zuverlässige Erkennung durch die KI voraus. Sowohl Fehlalarme als auch nicht erkannte Angriffe müssen verhindert werden.

Das Training einer solchen KI benötigt einen Trainingsdatensatz bestehend aus annotierten Bildern. Hier liegt der Fokus nicht nur auf die gut zu erkennenden Tiere, sondern besonders auf die schwierig zu erkennenden. Weiterhin muss das Modell der KI von Zeit zu Zeit angepasst werden, damit diese auch unter angepassten Bedingungen gute Leistung zeigt.

All diese Herausforderungen geht diese Arbeit mit unserem *ShadowWolf* an. Es nutzt einen geführten Modus für das automatische Annotieren von Bildern. Hierzu kombiniert es aktuelle Technologien aus dem Bereich der Objekterkennung mit einer internetbasierten Validierung durch Interessierte. Dieser Ansatz erhöht die Erkennung von Tieren und reduziert den Arbeitsaufwand für die Fachleute. Als Ergebnis erhält man einen Trainingsdatensatz, mit dem beliebige KI-Modelle trainiert werden können. Diese können dann an die individuellen Ansprüche und entsprechend der verfügbaren Ressourcen skaliert werden.

Für diese Arbeit haben wir außerdem einen Trainingsdatensatz mit über 100.000 Fotos aus Wildkameras gesammelt und analysiert. Weiterhin diskutieren wir den Einsatz des Systems im Feld mit einem Sensornetz speziell für den Einsatz im landwirtschaftlichen Umfeld.

Am Ende bietet diese Arbeit einen Mehrwert für alle, die mit Bildern aus Fotofallen arbeiten. Sowohl das Zählen von Tieren, die Erkennung von Raubtieren oder das automatische Training von KI-Modellen wird auch für nicht technisch versierte Personen anwendbar.

Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSI	Camera Serial Interface
CSS	Chirp Spread Spectrum
DNN	Deep Neural Network
ETSI	European Telecommunications Standards Institute
EXIF	Exchangeable Image File Format
FLOPS	Floating Point Operations Per Second
FPN	Feature Pyramid Network
GNSS	Global Navigation Satellite System
GPIO	General-Purpose Input/Output
GPU	Graphics Processing Unit
HDR	High Dynamic Range
I^{2}C	Inter-Integrated Circuit
IoT	Internet of Things
IoU	Intersection over Union
IPTC	IPTC Information Interchange Model
ISM	Industrial, Scientific and Medical
LiDAR	Light imaging, detection and ranging
MAC	Medium Access Control
MOT	Multi-Object Tracking
NMS	Non-Maximum Suppression
OS	Operating System
PHY	Physical Layer
PoE	Power over Ethernet
PRR	Packet Reception Ratio

R-CNN	Region-Based Convolutional Neural Network
ReLU	Rectified Linear Unit
ROI	Region of Interest
RPN	Region Proposal Network
SPL	Sound Pressure Level
SVM	Support Vector Machine
ToA	Time on Air
TPU	Tensor Processing Unit
UAV	Unmanned Aerial Vehicles
VPU	Vision Processing Unit
WPAN	Wireless Personal Area Networks
WSAN	Wireless Sensor / Actuator Network
WSN	Wireless Sensor Network

Chapter 1

Introduction

With approximately 84 million residents or 238 people per square kilometer[1], Germany is quite densely populated. In contrast to other countries like, for example, France (118), Italy (199), Poland (134), or Spain (95), the risk of potential conflicts between the interests of humanity and wildlife is comparably high. Especially after the resettlement of wolves around the year 2000, the number of kills of farm animals is continuously increasing.

This leads to two oppositional parties: One welcomes the wolves back to Germany and wants to protect them. The other focuses more on the damages caused by wolves – mainly financial, livestock, and emotional – and prefers regulating the number of animals. This thesis focuses not on discussing the pros and cons of hunting wolves. Instead, we show the current technologies to foster living together with them. The main idea is to develop a system to reliably detect all kinds of predators, adapt dynamically to new species, and help to monitor their living habitats. It further discusses some non-lethal technologies that might help deter predators from farm animals.

The following sections will discuss the current situation regarding predators in Germany. We also explain the idea and the foci of the *mAInZaun* project, in which context this work was written. We further list the contributions of this thesis and outline the following chapters.

1.1 Carnivora in Germany

Carnivores, which can hunt large animals or can even become dangerous to humans, are rare in Germany. Besides a few sightings of brown bears, golden jackals, or Eurasian lynxes, Canidae, like (stray) dogs and wolves, are by far gaining the most public interest. Wolves (Canis lupus), or more precisely, wolves killing farm animals, play a dominant role in the news nowadays.

After excessive hunting in the 15th century, wolves have been extinct in central Europe since the 19th century. Since the 1980s, wolves have been protected in Europe and Germany by several laws and regulations [1, 2, 3], leading to an increasing number of wolves. In 2000, the first wolf pups were born in freedom in Germany. In the monitoring year 2022/2023, 184 packs of wolves were counted

[1] `https://ourworldindata.org/most-densely-populated-countries` accessed: 2023-10-23

in Germany[2]. Wolf sightings are reported from 15 out of 16 states of Germany, with a focus on Brandenburg, Saxony, Mecklenburg Vorpommern, Saxony Anhalt, and Lower Saxony. The DBBW (Dokumentations- und Beratungsstelle des Bundes zum Thema Wolf)[3] gives details about the number of kills on farm animals and the financial aspects. In 2022, the DBBW registered 1136 attacks of wolves on farm animals, leading to 4366 dead animals. On average, wolves killed 3.8 animals per attack. 89% of the animals were sheep or goats. The actual numbers of attacks are expected to be higher as not every farmer reports the attacks for various reasons.

The states offer compensation payment for confirmed kills by wolves. In 2022, 616,000 € were paid mainly by Brandenburg, Lower Saxony, and Saxony. This money compensates only part of the actual damage. Besides the financial damages of losing several animals, the psychological damage to the flock is challenging: The remaining herd is often traumatized, nervous, and hard to handle for several weeks. Additionally, the number of lambs in the following years is often reduced, leading to even higher financial losses.

Therefore, the focus is on protecting the farm animals from the wolves. The states invested 2022 18,428,830 € in protective measures like wolf-proven fences. Ideally, a fixed fence is used. This should have five electrical wires: the lowest 20 cm above the ground, the highest 120 cm above the ground. If an animal touches it, it will get a painful electrical shock. Building such a fence is costly and requires lots of effort.

In some cases, portable fences are used. For example, the shepherds in northern Germany move their flocks along the dike, and the fences must also move. Also, those fences give electrical shocks if touched. Those portable fences are less durable and reliable than the fixed ones.

All fences require proper maintenance to be protective. For example, if the electrical wires touch the grass or bushes, the circuit is short-circuited, and the power of the shocks is reduced, limiting the protection. Therefore, regular mowing under the fences is required as the sheep will not eat the grass next to the electrical wires. The fences must also be checked for large gaps where the wolves can crawl through.

The wolves carefully examine the existing fences for such vulnerabilities and overcome barriers. Once on the pasture, they often kill more animals than required. Ideally, the pain and the expenses for the wolf accessing the pastureland are so high that it prefers hunting wild animals, like roe deer or wild boars, and neglects the farm animals.

The objective of the *mAInZaun* project, as described in the following section, is to make existing fences more resilient against wolves.

1.2 The *mAInZaun* Project

Animals, which are kept outside for farming purposes, must be protected from all possible hazards, including escaping the pasture or unwanted access by predators [4, 5, 6]. Building a wolf-proven fence is complex and expensive. In some cases, like movable fences, even impossible. Those portable fences are essential to

[2]`https://www.bfn.de/daten-und-fakten/wolfsvorkommen-deutschland` accessed: 2023-10-24

[3]`https://www.dbb-wolf.de/,accessed:2023-12-26`

protect the dikes in northern Germany. Here, they are used for shepherding: The sheep are ideal for maintaining the dikes by cutting and fertilizing the grass and compacting the surface without destroying it. Heavy, wolf-proven fixed fences are not allowed on dikes as they are part of a critical flood-protection infrastructure.

The idea of our *mAInZaun* project[4] is to strengthen the existing fences by using state-of-the-art Artificial Intelligence (AI) technology. The project focuses on wolves; the overall system should be adaptable to other predators. The objective is to increase the protection level of the fences in two parts: Detection and Deterrence.

The detection part is sensing the environment: Is a predator accessing the fence, examining it for weaknesses, or trying to overcome the barrier? If the sensors detect such activity near the perimeter, the second part is activated: deterrence. Here, the neophobia of wolves is used: a variety of possible deterrents like ultrasound, flashing lights, odor, etc., can be activated to confuse the wolf's senses. The idea is to make the investigation of the fence perimeter for the wolves as unpleasant as possible. Another behavior of the wolves is also advantageous for the project: They do not spontaneously approach and overcome fences. Before that, the fence is usually examined, increasing the chance of detecting the wolves with our sensors.

This thesis focuses mainly on the detection of wolves and the activation of the deterrents. We also show the options and constraints of running such a system independent of infrastructure in the wild. The effect of the deterrents on predators and farm animals is evaluated by the Professorship of Animal Husbandry, Behaviour, and Welfare at the Justus-Liebig University in Gießen, Germany. Another partner, *RoFlexs*[5], has experience in fencing and supports the project with its knowledge. The complete architecture of the *mAInZaun* project is depicted in Figure 1.1.

1.3 Contributions of the Thesis

Detecting animals in automatically captured real-world images is beneficial for several use cases. In this work, we introduce *ShadowWolf*, a framework to handle those images and help to improve arbitrary object detection and classification models continuously. The main contributions of this work are manifold and discussed in this section.

Working with real-world images is essential for this work. Most existing works in the area of animal detection focus on good images or synthetically created ones. Bad images are usually removed from the datasets. We need the complete dataset from the camera and therefore collected our **wolf datasets with more than 100,000 photos** from animal parks in northern Germany. This also includes bad images, i.e., photos captured during the night or rain.

Labeling and evaluating those images can be a tedious job. To offload this work from individuals to a larger group of interested people, we created *Wolf-or-Not*. This is an internet-based **application to annotate images using crowdsourcing** and helps to label and evaluate the wolves in our image dataset.

[4]`https://intelligenter-herdenschutz.de/`
[5]`https://www.roflexs.shop`

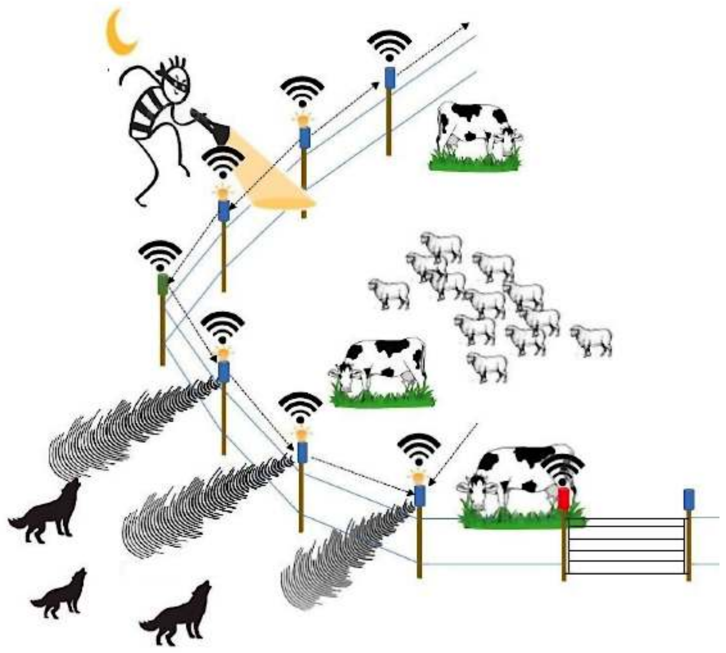


Figure 1.1: In the *mAInZaun* project, a traditional pasture fence is extended. Cameras are located on the fence poles and detect predators like wolves. If an endangerment is detected near the fence, deterrents or alarms are activated to protect the farm animals from the attackers. Ideally, the system should also be able to detect humans like burglars or other persons with evil intentions and inform the owner in time. All this requires reliable object detection.

While *Wolf-or-Not* is quite flexible in annotating the images, the overall process is much more complex and requires pre- and postprocessing steps. Furthermore, the number of images processed by *Wolf-or-Not* should not be too high to get results on time. For that, we developed the main contribution of this work: *ShadowWolf*. This is **a framework to train models automatically and continuously**. It automatically combines three steps: First, we preprocess the images and find areas with activity. Second, we run state-of-the-art object detection to find the objects of interest. Last, we validate the detections with our *Wolf-or-Not* and create a training dataset. Everything works with only minor human interaction and is highly flexible to adapt and extend. Especially the combination of state-of-the-art object detection and classification models with a *Wolf-or-Not*-based validation helps to find hidden objects that were only detected with low probabilities by the model. As an output, we get labeled images that can be used to train a new model or help to screen images. *ShadowWolf* also evaluates the quality of the results. Further, it is a modular system that allows the replacement and adaptation of individual parts.

The object detection used in *Wolf-or-Not* is a crucial component. Therefore, we **evaluated several state-of-the-art frameworks**. For the most promising one, YOLO, we went further. Also, we assessed the required computing resources, the inference speed, and the capabilities of running on different machines starting from a RaspberryPi to a Graphics Processing Unit (GPU) server.

The objective of the overarching *mAInZaun* project is detecting and deterring predators in agricultural environments. For that, we offer a system architecture design for an outdoor environment deployment using a **wireless sensor network tailored for animal detection and deterrence**.

Summarized, the main contributions handled in this work are:

Contribution 1:
Camera trap image dataset with wolves

Contribution 2:
Webservice for crowd-based image annotation (*Wolf-or-Not*)

Contribution 3:
Toolchain to automatically label images and create training datasets (*ShadowWolf*)

Contribution 4:
Comparison and fine-tuning of current machine-learning models in the area of object detection and classification

Contribution 5:
Deployment in the field for the *mAInZaun* project

1.4 Outline of the Thesis

We structure this work as follows: Chapter 2 creates the overall context of this work. It states the problem and discusses the current state-of-the-art in the area of camera trap images, animal detection, wireless technologies, and deterrents.

Chapter 3 focuses on machine learning and discusses the currently used terms and technologies. Here, we also discuss the current available frameworks and their performance as well as the commonly used metrics in this area.

The objective of this work is to deploy all parts outside. We selected LoRa as the wireless technology to connect the system's components. Chapter 4 discusses and evaluates LoRa and its properties in detail.

Suitable images are essential for tasks in image processing. In our case, we require a lot of realistic wolf images for the individual parts of this thesis. Therefore, we describe the challenges in those images, our data collection, and the datasets in detail in Chapter 5.

In our work, we use the YOLO framework for object detection and evaluate its performance and ability to run inference on different hardware platforms in Chapter 6.

Wolf-or-Not is our tool to distribute the effort of labeling and evaluating images to many interested people using crowdsourcing. We discuss it in detail in Chapter 7.

ShadowWolf is the main contribution of this work and is described and evaluated in chapter 8.

This work and the *mAInZaun* project aim to deploy the complete system, i.e., the animal detection and deterrent network outdoors. We describe this in Chapter 9.

Finally, Chapter 10 concludes this work and gives an outlook for future steps in the context of this work.

Chapter 2

Problem Statement and State of the Art

This thesis is written in the context of the *mAInZaun* project. The project aims to detect and deter predators near pasture fences, thus protecting the farm animals. This results in several challenges. Besides the deterrence of the predators, the reliable detection of those is the main challenge and the central contribution of this work. The animals must be reliably detected in the foreground, background, or even partly visible. Here, environmental effects like rain, snow, dust, dirt, changing light conditions, etc., play an important role. Furthermore, distant animals should be detected reliably, leading to further challenges.

This chapter discusses the problems in this field. Section 2.1 describes our main problem and the related tasks. We detail the recent research in object detection for animal detection in outside environments in Section 2.2. To train a new model, we need a training dataset. In Section 2.3, we review the related ones. Section 2.4 discusses the currently used tools used for image labeling. After the successful detection of a wolf, it should be expelled from the fence. We briefly review those deterrents' state of the art in Section 2.5. It is planned to place the deterrents independently from the detection system and activate everything wirelessly. Therefore, we review infrastructure-less communication technologies in Section 2.6. Finally, we discuss our proposed setup in Section 2.7

2.1 Problem Statement

Image processing and semantic analysis (*What can be seen?*) arrived in our everyday lives. We all use it regularly. Sometimes, actively by using a smartphone app analyzing images like Flora Incognita [7], sometimes passively by reading automatically generated image descriptions, for example, on Facebook.

All those applications focus on apparent detections: In most cases, the object of interest is the dominant part in good light conditions. Figure 2.1 shows an example. Here, a wolf is clearly visible and not covered by, for example, the vegetation. The lens is clean, and the photo was taken during the daytime. The situation is clear for humans and AI: A wolf!

Figure 2.2 shows a contrasting image: Two wolves are marked with red

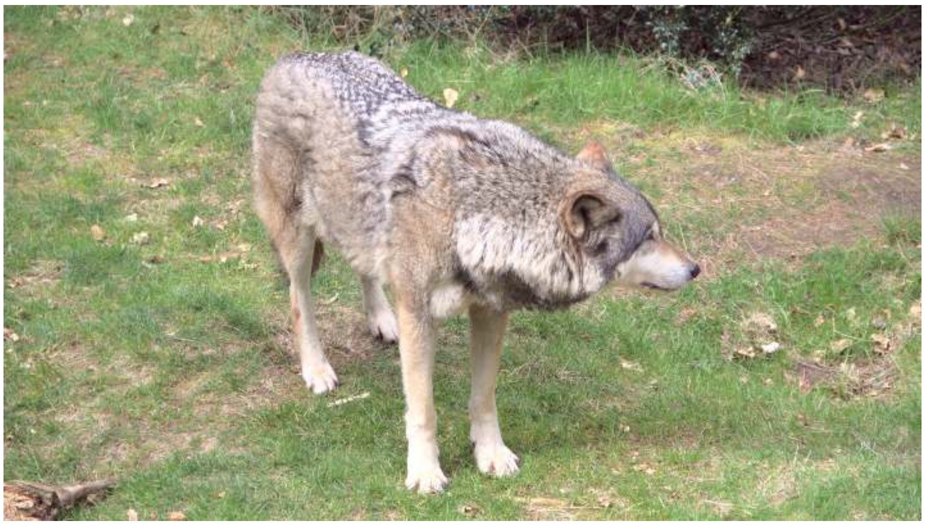


Figure 2.1: A good picture of a wolf: The animal is in the focus in perfect light conditions. In this situation, the detection is easy.

boxes for better visibility. While one can recognize the right, walking wolf – at least with some training – the upper, lying one can barely be seen. The light conditions are not as good as in Figure 2.1. The light is more greyish, and the wolves cover only a small amount of the total photo. The third example shown in Figure 2.3 shows an extreme case: raindrops on the lens and barely visible wolves during the night. Ideally, those should be detected and identified as wolves reliably.

The reliable detection of – generally speaking – any animal or object in this kind of automatically captured outdoor images is beneficial for several domains. One is generating a labeled dataset for training an object detection model. For that, the objects of interest as the predators in our *mAInZaun*-project have to be manually marked and tagged as, for example, wolves. This can be a tedious task.

Another application of reliable detection is the screening of collected camera trap images. With the rising number of wolves in Germany, the interest in the location of the packs is rising. Foresters and hunters place camera traps in the wild, producing many images. Helping them by at least partly automatizing the processing reduces their workload and increases efficiency at the same time and better detect animals.

Object detection and image processing under these circumstances differ significantly from other applications. We use a series of automatically captured camera trap images for training and detection to benefit from the movement of the animals over time.

Our target environment is farmland, i.e., a meadow or a forest. It can be in a hilly environment or the flatlands. Depending on the vegetation, we have a free field of view or a limited sight due to bushes and trees. We must adapt to those environments quickly and offer good detection even in changing environments.

Although this work focuses on the wolf, the system should be able to detect arbitrary animals or humans. We want to be able to adapt to other species

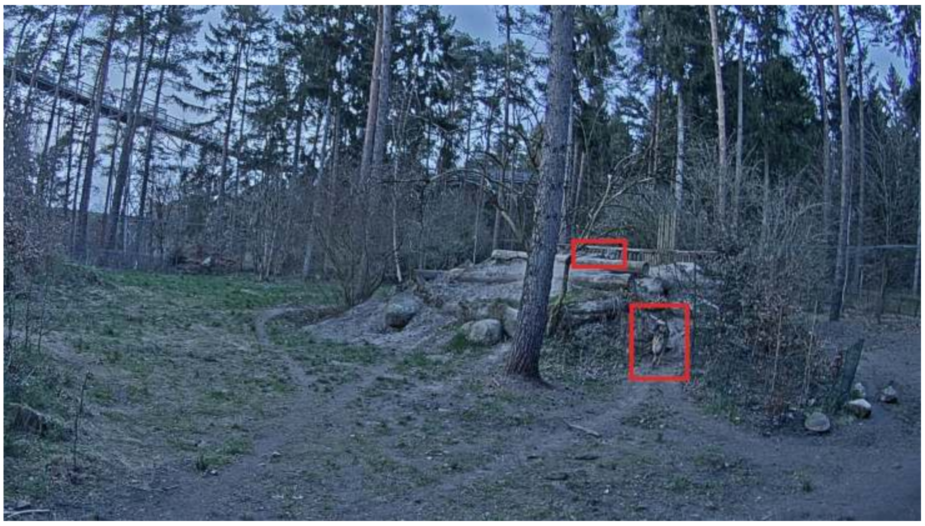


Figure 2.2: Two wolves marked with red boxes: One is walking, the upper one is lying. Both are hard to recognize. One can also see how well their fur is adapted to the environment. In this situation, the detection of both is challenging.

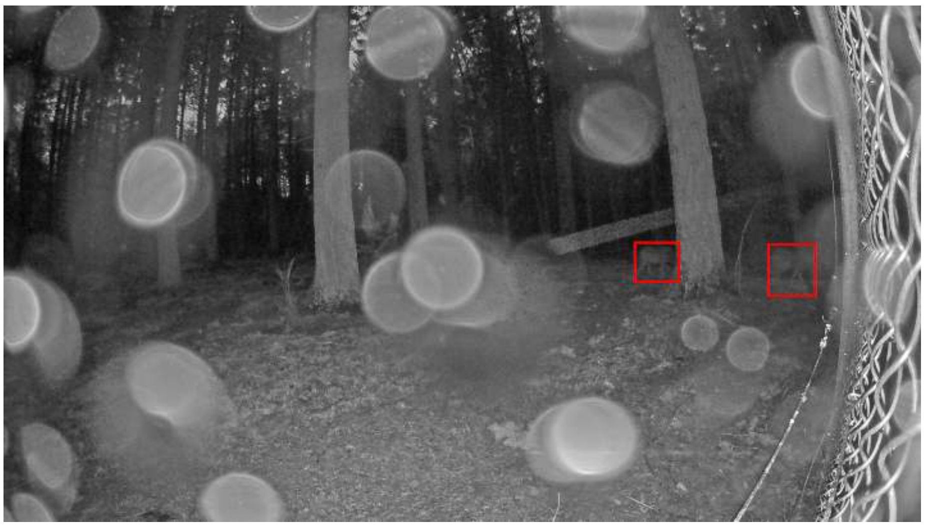


Figure 2.3: Two wolves marked with red boxes at night while it is raining. They are almost impossible to recognize.

easily.

Also, the environmental conditions can vary. The images on a sunny day are mostly straightforward. In our scenario, we have to deal with several challenges: We do not always have a well-lighted and tidy environment. Instead, bushes, trees, dirt, changing light, insects, etc. affect our images. We need a system that operates in those natural outdoor environments day and night in all seasons and results in usable detections.

When automatically capturing images, the animal of interest is not always in focus. We would like to detect the apparent, easy-to-see animals and the partly visible, distant ones. The detection should work for animals at least till 10 m (also during the night), ideally much further.

We do not consider the energy consumption of the detection. We focus on the detection itself and optimize the training and the dataset. Energy optimization can be done later, depending on the specific application requirements.

The *mAInZaun* project focuses not only on the detection but also on the deterrence of wolves. After the reliable detection, the wolf should be expelled to ensure he keeps a safe distance from the perimeter of the fence. Those deterrents should be sustainable and last for a longer time. They should also operate over a certain distance, i.e., at least to cover the area between two neighboring deterrents. The wolf should not be able to pass in the middle easily.

The deterrents should also be able to be operated on battery power. Therefore, we focus on systems that require a maximum of 12 V as a wide range of transportable power sources can provide this.

The detection part and the deterrents have to be connected flexibly: One detection system can trigger several different deterrents, the locations can be different for all parts, and everything should be portable. Connecting all components wirelessly has several advantages: It reduces the effort of setting up everything and prevents potential connection errors caused by the connectors or wrong handling. Also, the potential for errors is reduced as no wires can be cut accidentally, for example, by a lawn mower. Further, the effort to set up the system and place all the components is lower, and also the weight and costs of long cables should not be neglected.

The overall system should be able to operate reliably even in remote environments. Even though cellular coverage increases continuously, a good connection can not be guaranteed. Further, the complete system can have several hundred devices, resulting in high running costs when using cellular connections.

The deterrents should also be activated if no internet connectivity or other required infrastructure is available. For that, the object detection device must activate the neighboring deterrents within less than a second in a range of several hundred meters. This ensures the system can react if a running wolf passes by.

The remainder of this chapter is structured as follows: In Section 2.2, we review the work done in the area of animal detection using camera traps or similar technologies. In Section 2.3, we evaluate the existing image datasets: Which publicly available datasets can be beneficial for us? We collect our wolf datasets and need to label at least parts of them. We review the available tools in Section 2.4. After the successful detection, the objective is to deter the wolves. We review the available deterrents in Section 2.5. Section 2.6 discusses the current state of the art regarding the wireless communication technologies and concepts applicable for this work. We summarize the individual reviews and select the components we will use for this work in Section 2.7.

2.2 State of the Art: Animal Detection

With the increasing number of wolves, the number of projects related to wolf or wild animal detection is increasing.

One example is the German *Herdenschutz Wolf*[1]. Outdoor surveillance cameras send live videos to a server operated by the provider. On this server, an AI model detects wolves and humans. The system warns not only of wolves but also of unauthorized access to the protected area. In the case of detection, the owner of the cameras receives a notification on the mobile phone. Due to the high amount of data and the continuous video analysis, ethernet and a high bandwidth internet connection are required on the premise.

Another German project is the school project from *Jugend Forscht*: *Perimeterschutzsystem*[2] [3]. They used radar to create a virtual fence next to the real fence and detect animals passing it in a distance of up to 60 m. In case of detection, they take photos and videos as proof. They do not run object detection on the images to identify the animal passing the fence.

In [8], the authors took a different approach and used microphones. They analyzed the howls of 170 wolves from three subspecies. This gives a general overview of the number of wolves in proximity but not the exact positions. We need to detect wolves when examining the fence or attacking animals. In this phase, they are usually silent and can not be localized by the howls.

Analyzing the selectivity and sensitivity of different infrared light-barrier activated camera traps are reviewed in [9]. The authors evaluated the detection of animals in three different sizes in front of six camera systems depending on the camera height and the distance to the animal. This is used to trigger automatic camera traps. They do not continuously monitor the environment, identify animals using machine learning, or catch distant objects.

Where's the bear (WTB) is a distributed IoT system for wildlife monitoring [10]. The authors mostly use WiFi cameras to catch wildlife pictures during the daytime. Those are classified using TensorFlow, and the Inception-v3 architecture [11] to detect bears, deer, and coyotes. For the training, the authors used synthetic images: They downloaded images from the required animals from Google and inserted them into empty background images. They also describe and evaluate the complete infrastructure to operate the detection with limited internet access. They only use good-quality daytime images with clearly visible animals. They do not consider distant, hard-to-detect ones.

The authors in [12] use the Serengeti dataset [13] with approximately 7.1 million animal snapshots. They tested the ability of modern computer vision methods to count and identify wild animals from 48 species in those camera trap images. They use a two-stage approach: In the first step, their model detects whether the picture shows an animal or is empty. The second step identifies the species in the image. They use this information to add attributes like *standing*, *eating*, etc., and also to count the animals. They implemented six deep learning architectures using TensorFlow and compared their performance on the dataset. The results are good; they identify more than 90 % of the animals. This work shows promising results regarding

[1]`https://www.herdenschutz-wolf.de/`

[2]`https://www.saechsische.de/neustadt-in-sachsen/junge-neustaedter-forscher-erfolgreich-5424642-plus.html`, accessed: 2023-12-01

[3]`https://jufo-dresden.de/projekt/archiv/2021/arbeitswelt/A4`, accessed: 2023-12-01

animal identification and can be combined with our work as we are model-independent. We can offer the training dataset to adapt their evaluated models.

In [14], the authors suggest an active learning approach for camera trap images. They use a three-step pipeline and a human oracle for active learning. The overall objective is to train a Faster-RCNN (ResNet-50, [15]) model implemented in PyTorch to perform classification. They did not change the detection model but focused on reducing the human interaction time during the training phase. Using their active learning approach, they reduced the workload for the oracle by 99.5% for their datasets. We go further in several points: Our generic toolchain integrates the pre- and postprocessing of the images and the detections. Additionally, we are model-independent and can use arbitrary models for the detections. Also, our crowdsourcing approach reduces the workload for individuals during the detection.

The authors in [16] focus on detecting animals like bears, bison, cows, coyotes, horses, etc. in human-habituated environments, especially for security and road safety. They evaluate eight object detectors regarding their generalization capabilities, i.e., using different data for training and deployment. They conclude that none of the used models operates well on new backgrounds, resulting in an increased number of undetected objects. They further state that creating synthetic images by inserting animals in a new background improves the performance. Two models, RETINA [17] and YOLO, offer good performance while being comparable and lightweight to run on multi-camera deployments. Their conclusion is one of the motivations of our work: To achieve the best performance even with changing backgrounds requires continuous training of the models. Our work offers a toolchain to perform this task automatically.

In [18], the authors describe an active learning strategy for animal detection on Unmanned Aerial Vehicles (UAV)-captured images. They use a ResNet-18-based [15] classifier in an active-learning approach to find the animals in the aerial photos. During the active learning phase, they showed only a small amount of images to the human expert and were able to increase the detection of animals in their dataset to 80 %. The detection in aerial photos is different from ours: In our scenario, we would like to detect the animals in all possible positions, i.e., directly in front of the camera and also in distant ones in different light conditions. In aerial images, all objects are located at a similar distance.

In [19], the authors trained a classifier based on the ResNet-18 [15] architecture using TensorFlow. Using camera trap images from North America, they achieved an accuracy of 97 % in correctly identifying the local species. They offer an R package that allows even inexperienced users in machine learning to use their pre-trained model or re-train the model to detect other animals. They do not provide an automated toolchain that integrates the pre- and postprocessing of the images or allows easy adaptation to images and species.

Real-time wildlife detection for endangered species is the objective of [20]. They propose an adapted YOLOv4 structure for a fully automated animal observation system for field applications. They train and validate their model using a dataset collected from the internet and focus on challenging conditions like low visibility, multiple animals in one image, different aspect ratios, complex backgrounds, and animals similar to the surrounding environment. Their model detects the individual animals with an F_1 score of 98%. They use full-scale, high-quality images to identify and count the animals. We require identification even on bad images with barely visible, distant animals.

MegaDetector [21] focuses on creating boxes of interest and identifies the general classes: animal, human, or vehicle. A classifier or object detection model can later treat those boxes to determine the exact species. The authors are currently extending MegaDetector to a PyTorch Wildlife Framework. They aim to offer a *Collaborative Deep Learning Framework for Conservation*. The focus is on detecting animals in general, not on individual species. MegaDetector can be combined with our work to find the animals in the images. Another model can then identify the animal. We offer the flexibility for this kind of integration. Unfortunately, MegaDetector requires quite a powerful machine, which hinders its deployment on constrained devices.

The authors in [22] use thermal cameras and the YOLO framework to detect humans in cold environments. Due to the differences to the normal (RGB-coloured) images, i.e., lower resolution and false-colors, they required training on an extended thermal image dataset. They did not evaluate how the system works on different classes or in warm environments. We need to detect and identify animals in all environments, i.e., also in the warm summer where the outside temperature is close to the body temperature of mammals.

The authors in [23] developed two object detection models on a set of thermal images. They used three classes (car, bike, person) and got promising results. They did not evaluate the detection at higher distances. The limitations of Thermal imaging are summarized in [24]: The weather conditions – especially the solar radiation, the distance between the camera and the object, the field of view, the fur of the animal, etc. are challenging factors reducing the use of thermal imaging in the area of object detection and identification.

Two alternatives to classical images to detect animals or objects weather-independent are radar and Light imaging, detection and ranging (LiDAR). Those come mostly from the automotive sector and are mainly designed for autonomous driving. The authors in [25] offer a dataset for deep learning based three-dimensional object detection. They provide a frame-synchronized dataset of traffic-relevant classes collected using a radar, a LiDAR, and an optical camera. In [26], the authors used a Convolutional Neural Network (CNN) to classify pedestrians, cyclists, or cars based on radar measurements for road user classification. The road environment significantly differs from our wildlife and is therefore not further considered.

Commercial companies offer hybrid approaches like, for example, the *AdvanceGuard* system[4]. They use radar to detect activity in an area like an airfield. A camera zooms automatically into the area of interest, and an alarm is triggered. This should help prevent collisions between aircraft and wild animals. Such systems are suitable for protecting an area with a free field of view, such as an airfield. They are hard to deploy in farmland. Also, the maintenance in harsh environments and the power supply are challenging.

To the best of our knowledge, there is no framework for wildlife images available that 1) offers a complete toolchain for automated image labeling, including pre- and postprocessing of the images, 2) is model-independent and allows the training of arbitrary detection models, 3) is modular and can be adapted to different input and output requirements and 4) also focusses on the bad, hard to detect objects in challenging environmental environments.

[4]`https://navtechradar.com/solutions/advanceguard/`, accessed: 2023-12-01

2.3 State of the Art: Image Datasets

Detecting animals or objects in images is a common task and is often used as an example in tutorials. Challenges like the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)[5] exist for some datasets: Who can detect more objects correctly from the given images? This section lists the publicly available datasets that might be relevant to our work.

When talking about sources of images, most people suggest internet image databases like flickr[6] or Google Images[7]. They offer a vast amount of images. The problem with the results is the quality and the source: Most images – especially the ones from flickr – are often high-quality, glossy ones. Unfortunately, one can sometimes not distinguish: Is it a photo, a drawing, an AI-generated image? Also, the license has to be considered. Therefore, we do not use images from these internet databases for training.

The ImageNet dataset[8] [27] is a commonly used dataset for training and evaluation of all kinds of classification and object detection algorithms. It has 1000 object classes, 1,281,167 training images, 50,000 validation, and 100,000 test images. Therefore, it is used for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). It contains four classes relevant to our project, including the grey wolf.

Common Objects in Context (COCO)[9] [28] is a large set of segmented everyday high-quality objects from up to 91 classes. It does not contain wolf images.

The PASCAL (Pattern Analysis, Statistical Modelling and Computational Learning) dataset[10] was mainly developed between 2005 and 2012. It contains several thousands of images with annotated and partly segmented objects from up to 20 classes, including dogs.

Kaggle is an online platform for data science competitions and belongs to Google. It hosts a dataset called *Dogs vs. Wolves*[11] containing 1000 dog and 1000 wolf images. The dataset contains high-quality images cut to the animals, i.e., full-scale animal images without a lot of background.

ObjectNet[12] [29] contains objects from 313 classes. In contrast to other datasets, the objects are shown from different angles. This work aims to identify common objects even if they are not aligned as usual but rotated.

We found no publicly available dataset offering high-resolution camera trap images of European grey wolves in their natural habitat. Instead, the existing datasets show high-quality images with full-scale wolves. For us, especially the hard-to-see wolves in bad light conditions and dirt on the lens are of interest: We need images that are as realistic as possible. Those images are considered unusable, removed, and not published for most datasets. Therefore, we decided to collect our own wolf dataset as described in Chapter 5.

[5]`http://image-net.org/challenges/LSVRC/`, accessed: 2023-12-12
[6]`https://flickr.com/`
[7]`https://images.google.com/`
[8]`https://image-net.org/`
[9]`https://cocodataset.org`
[10]`http://host.robots.ox.ac.uk/pascal/VOC/`
[11]`https://www.kaggle.com/harishvutukuri/dogs-vs-wolves`
[12]`https://objectnet.dev/`

2.4 State of the Art: Image Labeling

Collecting or having images is one task; knowing what is inside is another. Annotating images is essential to train a model for computer vision. In our case, we screen the images, mark the wolves using a box, and assign the class *wolf* to it. This is a tiring and annoying job, especially for animals in the background or barely visible. Therefore, we evaluated tools that can reduce the workload for individual persons. Further, the import of the images and the export of the results should be executable automatically. The labeling task should be as simple as possible so non-experts can contribute.

Several tools like LabelImg[13] or its successor Label Studio[14] as well as several web services like cvat[15] exists to support the users for the labeling. The user draws a box around the object and assigns the corresponding class. This requires certain expertise, effort, and concentration.

Other tools reduce the workload for humans during this labeling. The authors in [30] created thousands of pixel-accurate labels for a cancer cell classifier with only a few manually labeled training data. In [31], the authors automatically train a model for object detection. They present the object in front of a steady background for the training. This kind of training can be done with clearly separated objects that can be distinguished from the background. In our case, we deal with animals that are highly adapted to their environment and not easy to see.

Labeling images for a classifier using a human-in-the-loop system is the objective of [32]. In their approach, a user manually marks the regions and assigns them to a class. However, it focuses on training data for a classifier and still requires a lot of user interaction, like marking the corresponding areas.

Replacing the human during the labeling task with a second detection model is done by the authors in [33]: They use two or more models to automatically label the foreground objects in images. It is not optimized for partly visible objects in the background.

OneLabeler, as presented in [34], is a framework for creating flexible labeling tools. They do not consider the crowdsourcing approach, allowing even non-experts to participate in the labeling.

Revolt, as presented in [35], is the most promising approach for our application. The authors distribute the labeling task to several co-workers to reduce the workload for the individuals. Our approach differs in three main points: Firstly, we optimize our web-user interface to the minimum and select the most probable image based on the statistics of the user votes, whereas Revolt requests more detailed information from the user. Secondly, we use a machine learning model to create the objects to label, further reducing the developer's effort. Thirdly, our iterative approach is developed to label, evaluate, and validate a continuously adapted model.

We found no software that easily offloads the labeling task to many non-experts, offers interfaces for import and export, and can also be used for image evaluation. Therefore, we decided to create our *Wolf-or-Not* web service as introduced in Chapter 7.

[13]`https://github.com/heartexlabs/labelImg`
[14]`https://labelstud.io/`
[15]`https://github.com/opencv/cvat`

2.5 State of the Art: Wolf Deterrents

Wolves are pretty clever: They do not just approach a fence and overcome it. They examine the surroundings and try to find the weak spots. Using deterrents, the time the wolf examines a fence should be made as uncomfortable as possible. Ideally, it gets annoyed, neglects the farm animals, and hunts wild animals like roe deer or wild boars.

Wolves are protected animals. Therefore, the literature about deterrents in this area is sparse and results from dogs as the closest relatives are often used instead. Most works focus on the sensible senses of the wolf: the sight and the hearing. The behavioral analysis of the deterrents is done by our project partners from the *Professorship of Animal Husbandry, Behaviour and Welfare at the Justus-Liebig University* in Gießen, Germany. Here, we give a brief overview and list the deterrents available on the market.

The effectiveness of ultrasound as a deterrent varies and depends on the objective. Some are used to get their attention by evoking the *startle reflex* [36] others for causing discomfort [37], which is influenced by the perceived Sound Pressure Level (SPL). Also, the rising time of the sound is essential. Experiments show that suddenly starting, loud sounds with more than 93 dB are effective deterrents.

For dogs, the authors in [38] observed aversive reactions to ultrasound frequency sweeps from 17 kHz to 55 kHz with an average output of 117 dB to 120 dB, but not to the same frequency sweep at a lower SPL. They also tested the effect of light flashes on the dogs, which highly depended on the surrounding light level.

Also, loud noise like gunshots, shouting, or music can deter dogs and wolves according to the literature [39, 40, 41, 42].

In the project, we also have to consider the effect on the environment and people living nearby. Therefore, we can not create arbitrary loud sounds. We have to consider the limits even for the ultrasound, which humans usually do not hear. According to the parameters above, we searched for devices capable of deterring dogs and wolves. They should be able to operate via battery and be available off-the-shelf. We found three devices. The WAS W130[16] light spot was originally developed for construction machines and is available in flood and spot versions with different light angles. Due to its original purpose, it can be operated on battery and is well protected from harsh outdoor environments. Our hardware can pulse it with 20 Hz, so it can be used to create disturbing flashes. We also found two ultrasound deterrents. The M161 Cannon[17] outputs directed pulses of a fixed frequency with a long range. The M175 Repeller[18] offers a shorter but broader range. Here, the frequency can be changed on the device, allowing the validation of several frequency bands.

Table 2.1 compares those three commercially available devices according to their datasheets. We list the output type, the expected range, and the required power.

All three deterrents can work on wolves. The W130 spot gives a bright light that can confuse and deter the wolf, especially when flashing. Light works best in

[16] `https://www.was.eu/de/arbeitsscheinwerfer/w130-8000/1214I/2276`

[17] `https://www.kemo-electronic.de/en/Car/M161-Ultrasonic-Power-Cannon.php`

[18] `https://www.kemo-electronic.de/en/Car/M175-Animal-Repeller-Ultrasonic-High-Performance.php`

Model	**Type**	**Output**	**Range**	**Power**
WAS W130 Spot	Light	8000 lm, manually pulsed with 20 Hz	5 m to 10 m	80 W
M161 *Cannon*	Ultrasound	directed 22 kHz pulses with 120 dB	300 m	1.8 W
M175 *Repeller*	Ultrasound	8 kHz to 41 kHz siren-like with max. 135 dB	100 m	1.8 W

Table 2.1: Comparison of different of-the-shelf deterrents according to their datasheets. All operate at 12 V. The effective range of the light depends on the direction and the ambient light. Here, we only give an estimate for the night. The M161 is directed and, thus, has a longer range than the M175. The given ranges are from the datasheets. The project partners are evaluating the effective range against wolves.

dark environments, as the eye needs time to adapt to different brightness levels. Therefore, we assume that the effect is reduced during dusk and dawn and not there during daytime. The M161 cannon generates pulses of loud ultrasound. According to the literature, this should work as a deterrent. Unclear is the effect of the directivity. The output of the M175 is not as directed as that of the M161. It outputs a siren-like sound with an adaptable frequency. Our project partners are also evaluating the effect of all three deterrents regarding the range, impact, and sustainability.

2.6 State of the Art: Wireless Technologies

As discussed in the problem statement, we will implement a wireless connection between the deterrents and the detection. This simplifies the setup, reduces the weight, offers more flexibility, and reduces the risk of failures.

Wireless communication in the area of sensing has been a topic for several decades, especially using small, constrained devices. Here, several concepts, architectures, and terms that are still relevant for this kind of communication have been developed. The objectives are low energy, flexible connection, and many devices in one network.

Wireless Sensor Networks (WSNs) are one option to transmit information with such requirements. Their application is vast and closely connected to other applications and terms. For example, Wireless Sensor / Actuator Networks (WSANs) include actuators, i.e., perform some actions based on the sensings. The Internet of Things (IoT) uses in some variations WSN-based technologies. Ad-hoc networks focus on dynamic networks without a fixed infrastructure. In this work, we use the term WSN as the most generic one, even though we are dealing more with a version of an ad-hoc WSAN. A detailed introduction to infrastructure-less networking concepts is given in [43]. [44] gives a general introduction to WSNs.

A WSN usually describes sparsely distributed, usually low-energy devices – called nodes – collecting and forwarding data to a certain point in the network wirelessly. The nodes can be mobile and reorganize their wireless connections automatically if moved. Sensor networks were first used in military applications during the Cold War to detect submarines in the ocean [45]. In 2003, the IEEE 802.15.4 standard defined a Physical Layer (PHY) and Medium Access Control (MAC) for low datarate Wireless Personal Area Networks (WPAN). Those operate usually in the license-free 2.4 GHz ISM-band and can be used worldwide without charge. This increased the research and the applications in this field, resulting in various applications like monitoring goods, buildings, or general environmental monitoring. New technologies and mechanisms like IPv6-based WSNs [46] or the discovery of services [47] simplified the application and opened it to a broader field. Today, those technologies are used as a base to control the smart homes, for example, with Thread[19] and Matter[20].

Some of the terms and concepts from this area are relevant to this work even though they are combined with different underlying technologies. In this section, we focus on the transmission technologies.

IEEE 802.15.4 [48] is a flexible and powerful technology used for WSNs and other related technologies. It is mainly used in the 2.4 GHz Industrial, Scientific and Medical (ISM) band that allows global usage. The communication range is similar to WiFi and limited to approximately 50 m to 150 m. The actual range depends on the overall setup. Mesh technologies are commonly used to extend the communication range. Here, the nodes create dynamic links and forward the messages to other nodes in the network, allowing mesh-wide communication. Usually, a border router coordinates the connections (routes) inside the network and also enables connections to outside, i.e., the internet. Direct communication between the nodes is usually also possible.

Bluetooth is related to IEEE 802.15.4, but the achievable communication range is more restricted and lies in the range of 10 m to 100 m. The typical range is even lower and too low for our application.

WiFi, as known from most modern laptops, mobiles, tablets, etc., is another option. It can form a local network used to activate the deterrents in direct proximity. The disadvantage is a comparable low range with 50 m to 150 m – depending on the environment, WiFi standard, and antenna. In the worst case, this is too little to activate the deterrents. Furthermore, the most recent WiFi technologies focus on high datarates and not on low energy, remote deployments as required for our application.

Cellular networks like 5G require the network operator to place base stations to supply sufficient coverage. Lower frequencies can be used to cover remote areas with fewer base stations. Here, a base station can achieve a higher range of several kilometers. How to keep up the communication if the infrastructure fails is discussed in several works [49, 50, 51]. To the best of our knowledge, no currently available system supports such an infrastructure-less 5G mode.

Another option for communication in remote areas is the satellite-based internet service *Starlink*[21]. The satellites are less affected by earth-based disasters, network outages, or just a lack of coverage. As a drawback, it requires a direct line of sight to the sky and requires with 50 W to 75 W quite a lot of energy.

Another provider for satellite-based connectivity is Inmarsat[22]. It offers worldwide phone and data connections and is commonly used for emergency

[19]https://www.threadgroup.org
[20]https://buildwithmatter.com/
[21]https://www.starlink.com/
[22]https://www.inmarsat.com

messaging. The costs for worldwide communication are very high or – in the case of emergency messages – limited to only a few messages.

SigFox[23] provides an infrastructure for long-distance low data transmissions. It is used to transfer meter information to the electricity or gas provider. The amount of data and the number of messages per day are restricted to few bytes.

LoRa is a low-power, long-range communication standard. It can achieve 15 km with a direct line of sight [52]. In some cases, also significantly longer distances are possible as the current record of 832 km shows[24]. The data transmission between nodes in the communication range can be done without a base station or network coordinator. LoRa operates using different frequency bands, while for Europe, the frequencies around 433 MHz and 868 MHz are permitted. The free field communication range varies from several hundred meters to several kilometers, allowing simple, direct communication without a mesh infrastructure. Duty cycle requirements limit the length of transmittable data and the number of packets per hour. Several manufacturers offer hardware for various applications for usually comparatively low prices. It is the physical layer of LoRaWAN and has a huge community.

LoRaWAN extends LoRa with a network layer and turns it into an infrastructure-based IoT communication standard. It is managed by the LoRa Alliance[25]. Provider like the community-based *The Things Network*[26] (TTN) or *Helium*[27] form a network to transmit small messages to internet servers. As LoRaWAN is based on LoRa, similar properties hold for the communication range and properties.

LoRa is also used for satellite-based communication. For example, TinyGS[28] is a global ground station network for LoRaWAN-based satellites. It focuses on communication with flying objects like drones, satellites, etc. and allows cheap communication with those self-built devices. It is not designed to communicate with ground-based devices like in our case.

Lacuna [29] uses LoRaWAN to communicate with sensors and trackers using satellites. They offer worldwide coverage for low-power sensing- and tracking devices. The satellites receive the sensor messages and relay them back to ground stations after a short time. It can not be used for the time-critical triggering of our deterrents.

Table 2.2 lists those technologies and their main properties. In the first column, we evaluated, if they are commonly used to transmit data to a central internet server and if they usually rely on infrastructure. Here, we distinguish between no, yes (using classical base stations), or if they use satellites or drones as their communication partner. If applicable, we also give the typical maximum communication range. If available, we give the range of the initial costs, i.e., for the end devices, and if a subscription is required, leading to further costs. We also give an idea of if the device can run on battery power.

The number of technologies offering wireless connectivity, even in remote environments, is continuously increasing. This is done either by using special

[23]`https://www.sigfox.com/`

[24]`https://www.thethingsnetwork.org/article/lorawan-world-record-broken-twice-in-single-experiment-1`, accessed: 2023-12-25

[25]`https://lora-alliance.org/`

[26]`https://www.thethingsnetwork.org/`

[27]`https://www.helium.com`

[28]`https://tinygs.com/`

[29]`https://lacuna.space/`

	Internet	Infrastructure	Max. range (typ.)	Initial Costs	Monthly Costs	Battery
5G	✓	✓	600 m	€	✓	✓
Starlink	✓	📡	N/A	€€€	✓	✗
Inmarsat	✓	📡	N/A	€€€	✓	✓
sigfox	✓	✓	40 km	€	✓	✓
LoRa	✗	✗	15 km	€	✗	✓
LoRaWAN: TTN	✓	✓	15 km	€	✗	✓
LoRaWAN: Helium	✓	✓	15 km	€	✓	✓
TinyGS	✓	📡	N/A	€	✗	✓
Lacuna	✓	📡	N/A	unknown	✓	✓
WiFi	✗	✗	150 m	€	✗	✓
IEEE 802.15.4	✗	✗	150 m	€	✗	✓
Bluetooth	✗	✗	100 m	€	✗	✓

Table 2.2: Comparison of selected radio technologies we reviewed in this work in their typical use cases. For the infrastructure, we distinguish if the technology is usually used with infrastructure (✓), no infrastructure is used (✗), or if it is based on satellite communication (📡). The marked line highlights LoRa, which we use in this work.

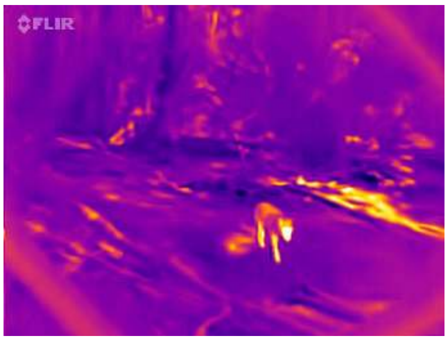
Costs: €: <50 €, €€: 50 - 100 €, €€€: >200 €. We consider a device to be battery-powered if the average power is $\leq$10 W. (Prices and data from Dec. 2023) technologies for a longer communication range, deploying new base stations, or using satellite-assisted technologies. We require direct, reliable, and ideally low-power communication between our devices. Internet access is not required. More critical is activating devices nearby on time in a range of several hundred meters. Cellular networks like 5G are aiming at this kind of device: small, low-power devices communicating only a little. Unfortunately, the coverage can not yet be guaranteed for our remote deployment areas. Therefore, we focus more on the technologies, not relying on local ground stations. Satellite-based technologies offer worldwide coverage but are expensive, require a lot of energy, or have a notable delay, making a timely activation impossible. Sigfox is a promising technology with a long-range but very limited in the number of messages. The standard technologies in wireless communication are WiFi, IEEE 802.15.4, and Bluetooth, which are widely used. Unfortunately, their communication range is quite limited. One can extend it with multihop- or mesh-architectures. However, this increases the complexity of the overall system and required infrastructure. LoRa-based technologies have several advantages: they are low-power, cheap hardware is available, they can be configured flexibly, and they offer a sufficiently long communication range. LoRaWAN is based on a base station and does not allow direct communication between the nodes. Therefore, we decided to use the plain LoRa: It offers the same advantages regarding energy, costs, range, and configuration but allows direct communication between the devices without needing a central coordinator, leading to a higher reliability for our scenario.

2.7 Summary and System Design

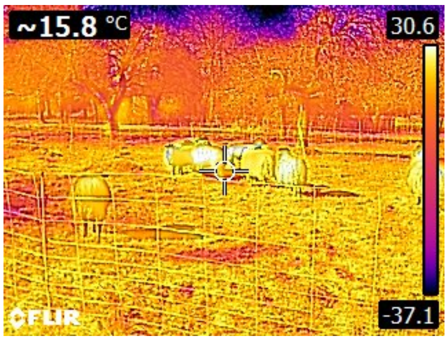
Most animal detection work is based on classical RGB images, which have several advantages. Cameras are widely used in object detection and are available in all possible resolutions and prices. A large number of models exist for different applications and requirements. They have drawbacks in bad lighting and weather conditions. The application of those cameras varies between camera trap analysis, aerial counting, and identification of the animals. Several works mention the challenges of changing the background of different cameras or positions. We also experienced such problems in our first studies and decided to create our *ShadowWolf* to deal with them.

Thermal imaging is an often suggested alternative or extension to the classical images, especially during fog or rain. We briefly evaluated those images by ourselves. Figure 2.4 shows pictures taken with two thermal cameras. In cold environments, one can detect mammals but not identify them due to the typically low resolution of the thermal photos. The animals can barely be distinguished in warm environments compared to a heated area, as shown in Figure 2.4a. Some cameras offer a mixed mode, i.e., they combine a standard photo with a thermal one, as depicted in Figure 2.4b. This combination might help during the daytime but fails during the night. Furthermore, thermal cameras are costly (our FLIR E6XT costs approximately 1500 €) and, therefore, not affordable for larger deployments in farming. Our findings match with the results from the literature [24].

Other sensors like LiDAR, Radar, etc., offer a three-dimensional and weather-independent representation of the environment. The research in this



(a) A thermal image taken during summer. The animal in the front can be seen but can not be identified. Part of the area at the right is heated by the sunlight, making detecting and identifying warm-blooded animals difficult. This picture was taken using a *Blackview BV9800* using a built-in FLIR Lepton thermal camera module with a resolution of 80 × 60 pixels.



(b) Several sheep on a meadow: The temperature of the sheep is quite close to the grass. This picture was taken in mixed mode, i.e., combined with a normal image, showing more details and structures. The camera was a FLIR E6XT at a resolution of 240 × 180 pixels.

Figure 2.4: Comparison of two thermal images taken with different cameras.

area is mainly from the automotive sector and focuses on detecting bikes, pedestrians, and other things relevant to autonomous driving. Some applications exist where this kind of sensor is used in a hybrid scenario joint with camera systems. Also, those sensors are expensive and not affordable for larger deployments.

The majority of the work in the area of animal detection is based on standard RGB images. Therefore, we decided to use classical surveillance outdoor cameras for this work: They are available at different prices (and qualities), offer good low-light capabilities, and sometimes even provide infrared light for night images. Also, the lens type and, thus, the camera's field of view can be selected. They produce standard RGB or greyscale images, which allows us to use traditional object detection or classification algorithms. We describe how we collect our dataset, which hardware we use, and the challenges in Chapter 5.

Existing databases with read-to-use images for machine learning are available online. For many applications, those are an alternative to collecting own datasets or at least a suitable extension for some cases. Section 2.3 discusses the ones relevant to our work. Only a few of them contain images of wolves. One of the most extensive ones is the ImageNet dataset. Unfortunately, it has high-quality images and is not very relevant to us. The Kaggle datasets contain high-quality images of dogs and wolves. Those do not show the animals in their natural environment, but a cutout. Such images are good for the classification of other photos, but not for our application. This strengthens our decision to collect a new wolf-specific dataset in Chapter 5.

Collecting images is only the first step. Next, they have to be screened, evaluated, and labeled. We would like to automate this task and be able to integrate it in other processes. Further, the tool should be able to distribute the work to many people, ideally online. We evaluated the existing tools in this area in Section 2.4, but found nothing suitable for us. As a result, we created our own app *Wolf-or-Not* as introduced in Chapter 7.

Most work focuses on creating new or optimizing models or architectures for their application scenarios. They state the challenges in getting good training data or updating the models to new domains, new camera positions, and, thus, new backgrounds. To the best of our knowledge, no framework offers the option to continuously label and analyze camera trap images and be model-agnostic simultaneously, as provided by our *ShadowWolf* as provided in Chapter 8 of this work.

Besides those software- and model-focussed tasks, the deployment and evaluation are part of this work. One objective is to deter the wolves. Here, we listed the commercially available components that have a chance of working on wolves in Section 2.5. The exact effect of those on the wolves, the effective distance, as well as the sustainability of the deterrents over time are evaluated by our project partners. Here, we activate those three deterrents using our *WolfNet* as described in Chapter 9.

Wireless technologies – especially in the context of the deployment – have several advantages for this work. We compared the common technologies and concepts in Section 2.6 and Table 2.2. We require a system that does not rely on infrastructure, like cellular networks (4G, 5G, etc.). The price should also not be too high, as each deployment requires plenty of devices to communicate. This also includes the running costs. It should be possible to communicate directly, as this reduces the complexity of the overall system. The LoRa-based technologies offer the best performance regarding our metrics and are commonly used for environmental monitoring. Therefore, we describe and evaluate LoRa in detail in Chapter 4.

We searched for WSNs based on LoRa and found a lot of hobby projects and some research. For example, the authors in [53] present a LoRa-based field monitoring system for the Andean area. Similarly, [54] evaluates the efficiency of LoRa for smart farming. In [55], the authors used LoRaWAN for landslide and rockfall monitoring, while [56] focusses on flood monitoring. They all transmit the data to a central entity and do not communicate directly with each other. We found no well-documented work offering a resilient LoRa-based sensor network that does not require a central coordinator. Therefore, we decided to write our *WolfNet* in Chapter 9.

The next chapter will introduce the field of machine vision and clarify the most important terms and concepts.

Chapter 3

Machine Vision

In 1943, the neurophysiologist Warren Sturgis McCulloch and the logician Walter Pitts discussed in their work *A logical calculus of the ideas immanent in nervous activity* [57] how the nervous activity correlates with neural activity using propositional logic. They modeled the relationships using simple logic operations. The results from this work can be seen as the birth of deep learning: The behavior of the brain, or more specifically, the neocortex, is mimicked by the layers of an Artificial Neural Network (ANN). Figure 3.1 shows such an example ANN. The circles represent the neurons: Three green neurons represent the input layer, and the two red neurons represent the output layer. The yellow neurons have neither an input nor an output and, thus, are called hidden layers. The example has two hidden layers consisting of 5 nodes each. For real applications, the number of hidden layers is usually higher. The output of each layer provides the input of the following layers.

Training creates connections between the individual neurons of the layers. A specific input should produce a corresponding output. In our case, the input is an image, and the output is the class *wolf* or another animal. The objective is to connect the neurons so that, ideally, all wolf images are detected correctly, and nothing is detected wrongly as a wolf. The input and the expected output must be known to train such a neural network. One can calculate the difference between the output of the neural network and the desired, known output. This difference, i.e., the error between both, has to be minimized. In the training phase, the connections between the individual nodes are varied till the error becomes minimal. The resulting connections between the individual nodes are called weights. This trained network should now give the same output when it is fed with a similar input.

For image processing, as performed in this work, the input layer is the image in a normalized format like, for example, 224×224 pixels and three colors, called a *tensor*. The output is the class detected in the image: If a neural network can detect 100 different types of objects – called classes – the output layer size is 100. The exact functionality of the hidden layers depends on the architecture. We provide an overview of those in Section 3.2.

To give an example for the numbers: A widely used model is the MobileNetV2 model [58]. Google developed it for mobile phones and embedded applications. For example, the implementation in TensorFlow consists of 156 layers and 3,504,872 trainable parameters. It maps an input image of the size

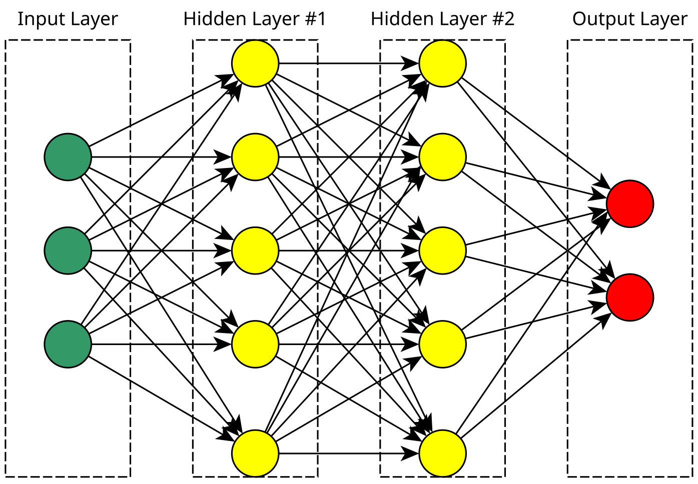


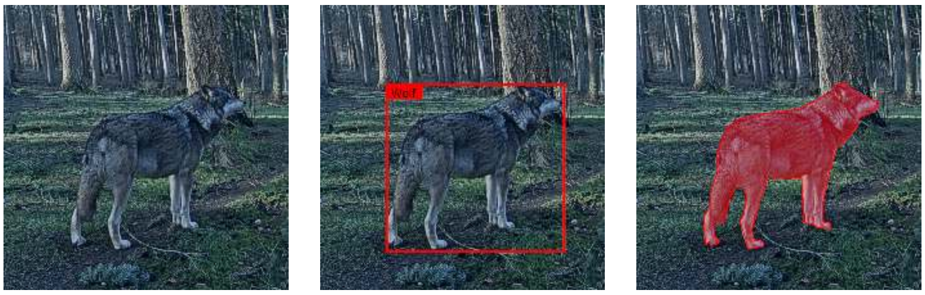
Figure 3.1: A simple example of an artificial neural network. The three green nodes (circles) are the input layer. The yellow nodes represent the two hidden layers with five nodes each. Two red nodes form the output layer. Each node represents one neuron. The arrows show the interconnections.

224×224 pixels and three colors to one out of 1000 classes from the ImageNet dataset. Other models can easily have more than 300 layers and 80,000,000 parameters. Training all those parameters to achieve a good mapping between the input and the output layer requires a lot of vector computations. In contrast to a standard, general-purpose Central Processing Unit (CPU), Graphics Processing Units (GPUs) are optimized for this kind of parallel vector calculations. An NVIDIA RTX 4090 GPU has 16,384 cores, yielding to 82.6×10^{12} Floating Point Operations Per Second (FLOPS). Compared to that, a high-end CPU like the Intel i9-13900 provides 24 cores and 32 threads, yielding approximately 10×10^{9} FLOPS – a significant difference of a factor of 8260. Especially for the training, using a GPU can yield a drastically increased performance compared to a modern multicore CPU.

The following sections will give an introduction to the main terms, concepts, and frameworks in the area of machine vision. Section 3.1 describes and explains the application of the three main tasks: classification, object detection, and segmentation. Section 3.2 explains this work's most important terms and concepts, whereas Section 3.3 focuses on the widely used frameworks. Section 3.4 focuses on the tools, and Section 3.5 introduces the metrics we use in this work.

3.1 Tasks in the Area of Computer Vision

In computer vision, three image detection tasks can be distinguished: classification, object detection, and (semantic) segmentation. Figure 3.2 shows one example for each task. The classification is the most fundamental task. Those models give a list of objects, including the probabilities being in the image. Our example Figure 3.2a might return *wolf*, *grass*, and *tree* – depending on the



(a) **Classification** returns a probability of an object being in the image. In this case, the probabilities for the class *wolf*, *grass*, and *tree* should be non-zero.

(b) In the case of **object detection**, the model returns a class and a corresponding bounding box. In contrast to the classifier, we also get the object's location.

(c) In the case of **segmentation**, an area of interest marked with a mask. It is mainly used to mark the objects in the foreground for further processing.

Figure 3.2: The three main tasks in computer vision are classification, object detection, and segmentation.

training data and classes. Those models often identify images and automatically perform context-sensitive searches on images: *Show me all images with boats.*

Several models with different properties exist for the classification. MobileNet V2 [58], and MobileNet V3 [59] were developed by Google researchers. They focus on running on mobile phones and embedded devices. The Visual Geometry Group (VGG) from the University of Oxford introduced the VGG model [60]. This one focuses on (at the time of development) large-scale images with a size of 224 $\times$ 224 pixels. Residual Networks (ResNet) [15], developed by Microsoft, is another famous convolutional neural network architecture. It is available in different depths, resulting in other performances. Besides the commonly used ResNet-50 with 50 layers, ResNet-18, ResNet-34, ResNet-101, and ResNet-152 with 18, 34, 101, and 152 layers, respectively, are available.

Another task in computer vision is object detection, as displayed in Figure 3.2b. Algorithms in this area return a bounding box and the class of the object inside this box. Most architectures use deep learning and convert the object detection problem to a classification problem. They create many small boxes, run the classification on those, and combine the results using regression. This is, for example, done by the Faster R-CNN [61] architecture. Here, a Region Proposal Network (RPN) creates proposals – so-called Region of Interests (ROIs) – which are boxes representing part of the image. Those are analyzed using a classifier like the VGG, as mentioned above. Performing Non-Maximum Suppression (NMS) on the results from the classifier returns one class for one box. YOLO (You Only Look Once) [62] optimizes the classical Faster R-CNN approaches by simplifying the structure and combining the box creation and classification. RetinaNet [17] is a different architecture focusing on small and dense objects like aerial and satellite images with the drawback of higher memory- and computing requirements. The SSD (Single Shot MultiBox Detector) architecture [63] uses a classifier like the ResNet-34 or VGG as a base. The SSD replaces the final classification layers and calculates the probabilities of objects being in predefined boxes out of the classifier's features. The advantage is fast execution time with reduced precision of the output boxes.

The (semantic) segmentation is the third task in computer vision. Figure 3.2c shows the output of classes for this class: They do not only return a bounding box but a mask framing the object of interest. Segmentation is of interest for several applications. One is gesture recognition. For that, not only a human has to be detected, but also the arms, hands, the head, etc. Segmentation is one step in this direction. Another task is the foreground detection. The segmentation detects the objects in the foreground by understanding the overall image more deeply. Those foreground images are then identified using a classifier. DeepLabV3 [64] by Google adapts a pre-trained ResNet classifier to a segmentation model. The resulting model is able to detect 20 foreground object classes and one background class. FCN (Fully Convolutional Networks for Semantic Segmentation) [65] is another segmentation architecture. They adapt classification networks to a fully convolutional neural network, achieving pixel-wise segmentation.

In this work, we are interested in detecting predators, i.e., the class and the positions. Therefore, we focus on object detection algorithms. A classification could also be used but requires further image preprocessing, i.e., detecting an object, creating a box, and identifying the animal in the box in a second step using a classifier. This is the way how most modern object detection algorithms work internally. Using a classifier on the complete image will not work as expected, as the wolves usually only cover a small amount of the image and are not the dominant class. We can also use segmentation, but it requires significantly more effort to label the images and run the model. Anyhow, our architecture offers the flexibility to use any of the three classes of algorithms.

3.2 Terms and Concepts in Neural Networks and Computer Vision

A vast amount of literature and tutorials exists in the area of neural networks, machine learning, and computer vision. Several books like [66] introduce neural networks and machine learning more abstractly. [67] gives a German overview about AI, the fundamentals and applications. The author in [68, 69] offers an English and German introduction to writing a neural network using Python.

On the internet, Stanford University offers an introduction to CNN[1]. From the Harvard University, an online course focussing on embedded systems is available[2]. Also, the widely-used frameworks PyTorch[3] and TensorFlow[4] offer good tutorials with a lot of examples and references to the corresponding literature.

Many new terms and definitions arise, especially for newcomers in this area. This section gives a brief overview without claiming completeness.

[1] `https://cs231n.github.io/convolutional-networks/` accessed: 2023-11-27
[2] `https://harvard-edge.github.io/cs249r_book/` accessed: 2023-11-27
[3] `https://pytorch.org/tutorials/` accessed: 2023-11-27
[4] `https://www.tensorflow.org/overview` accessed: 2023-11-27

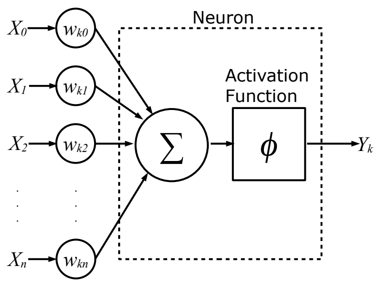


Figure 3.3: Graphical representation of a neuron

3.2.1 Neuron

An artificial neuron, as used in ANNs, is a mathematical function described by Equation (3.1) and depicted in Figure 3.3. We use the parameters as defined in Table 3.1.

Parameter	**Meaning**
x_i	The $n + 1$ inputs
y_k	The output of the layer k
w_{ki}	The weights for the input i of the layer k
ϕ	The transfer function / activation function (ReLU)

Table 3.1: The parameters for an artificial neuron as described by Equation (3.1).

The input x_0 is often often set to +1 making the weight w_{ko} the bias. This bias helps to shift the activation function to the desired threshold. This leads to n independent inputs for this layer. The remaining inputs $x_1...x_n$ are multiplied with the corresponding weights $w_1...w_n$ and summed up. The activation function – in most cases, the Rectified Linear Unit (ReLU)-function as described in Section 3.2.2 creates the output of this layer. The weights are varied during the training phase until the input matches the expected output for, ideally, all training data.

$$y_k = \phi\left(\sum_{i=0}^{n} w_{ki}x_i\right) \tag{3.1}$$

3.2.2 Rectified Linear Unit (ReLU)

The Rectified Linear Unit (ReLU) is a rectifier and can mathematically be described as shown in Equation (3.2). It is widely used in neural networks as an activation function on the neuron (c.f. Section 3.2.1) to activate the output to another layer. It results in sparse activation by suppressing outputs below 0 and only using outputs greater than 0, as shown in Figure 3.4. Furthermore, it can efficiently be computed. The activation function is often denoted by phi (ϕ).

$$f(x) = \phi(x) = max(0, x) \tag{3.2}$$

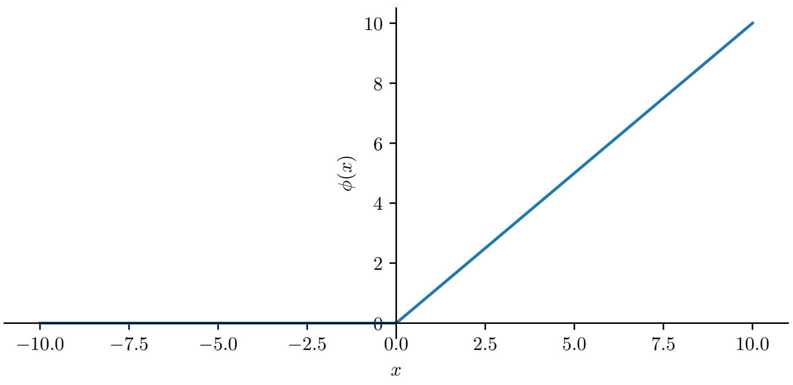


Figure 3.4: The relationship between the input x and the output $\phi(x)$ of the ReLU function.

3.2.3 Logit

A Logit is an unnormalized prediction. Predictions are usually in the range between 0 and 1. A layer might output several predictions using other scales, i.e., numbers outside this range. After normalizing them, one gets the required normalized predictions.

3.2.4 Tensor

A tensor is a multidimensional array of numbers and is often used for data representation in machine learning, especially in neural networks. Equation (3.3) shows the differences. a is a scalar, i.e., a number. Adding one dimension results in a vector like b or, when adding another dimension, a matrix, like c. A tensor like d is the most generic version and can have arbitrary dimensions. An image with three colors (RGB) and a size of 160×160 pixels results in a tensor with the shape $(160, 160, 3)$.

$$
\begin{aligned}
a &= 1 \\
b &= \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \\
c &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \\
d &= \begin{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} & \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} & \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} \end{bmatrix}
\end{aligned}
\tag{3.3}
$$

3.2.5 Input Size

The image fed into a neural network should match the desired size of the network. Normal pictures often have an aspect ratio of 4:3 or 16:9 and a resolution of, for example, 1920×1080 pixels. Neural networks usually require squared images of, for example, 128×128 or 160×160 pixels. Sometimes, the input

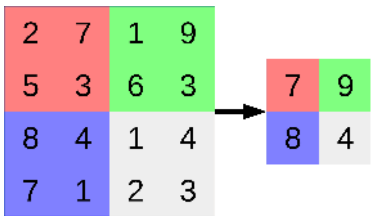


Figure 3.5: In a pooling layer, the input is divided into partitions. For the most common type of pooling layer, max pooling, the output is the maximum value of each partition, as shown in this figure.

is adapted automatically by using padding and rescaling. Not performing this kind of adaptation – manually or automatically – results in significantly poorer performance.

3.2.6 Pooling

Pooling is a type of downsampling of the input and is commonly used in CNN (c.f. Section 3.2.14). The input is divided into several partitions. The downsampling is performed for each of the partitions and reduces the dimension of the output. The most common pooling method is the max pooling as shown in Figure 3.5: From each partition, the maximum value is used for the lower-dimension output. This method reduces the complexity of the input while only focusing on the most dominant parts of the input. This downsampling generalizes the data and prevents overfitting, i.e., creating a model that is not generic enough.

3.2.7 Feature Map

A feature in general computer vision is a property of the image, like an edge or a circle. Therefore, the output of a convolutional layer is also called a feature map. It focuses on the critical shapes and structures while suppressing the less relevant parts of the image. This feature map is then mapped to the individual output classes.

3.2.8 Quantization

Models in machine learning are computationally heavy and require a GPU or a powerful CPU even to run the inference. Most of the operations rely on floating-point numbers or even doubles, which require 32 bit or 64 bit of memory, respectively. Quantization reduces the resolution and, thus, the required memory by approximating the numbers to ones with fewer bits. In an extreme case, that can be an unsigned short integer with just 8 bit. This is also required to run such a model on a specialized Tensor Processing Unit (TPU). Especially edge-TPUs like the Google Coral require models quantized to 8 bit. According to [70], the quantization's effect on the detection accuracy can be kept low, especially if it is considered during the training.

3.2.9 Supervised and Unsupervised Learning

One can distinguish two ways to train algorithms: supervised and unsupervised ones. Unsupervised algorithms do not need the support of humans as they discover (hidden) patterns in data by themself. They are used, for example, to cluster similar objects or reduce the number of dimensions. Most of the algorithms discussed in this work perform supervised learning. Here, a human offers a labeled training dataset so the algorithms can evaluate how well it matches the expected labels. A method in between both is semi-supervised learning. Here, a small amount of labeled data is combined with a large amount of unlabeled data to train a model. This method works well if the labeled dataset is highly relevant for the classes in all required situations. We can not guarantee such kind of a labeled dataset and, therefore, focus on supervised learning.

3.2.10 Active Learning

Active learning is a feedback method that is, in some cases, performed during the training. The model can ask an oracle (teacher), which can be a user or another model, to label new data. This can be used for training or for the evaluation of the current model. Active learning is commonly used if the input data is hard to label or a large amount of data is available. We use a kind of active learning in our work, as we combine the detection of the model with the evaluation of humans from our crowdsourcing app *Wolf-or-Not*.

3.2.11 Hyperparameters in Machine Learning

In contrast to the weights, which are changed during the training, the Hyperparameters control the training or the learning process. Those are, for example, the following:

- The **Number of Epochs** determines how often the training data is shown to the network during the training phase.
- The **Batch Size** defines how many samples are fed into the network between two parameter updates and validation steps.
- The **Learning Rate** determines how fast and how exactly the network might converge.
- When using transfer-learning, i.e., partly retraining an existing model, the **Number of Trainable Layers** is also a hyperparameter.
- The **Activtion Function** can also be selected to be different to the commonly used ReLU function.

3.2.12 Backpropagation

Training a neural network is performed to match the output to the ground truth of a given training dataset. For that, the internal weights are varied until the (squared) error between the output of the neural network and the expected output is minimal. The number of weights can easily be several thousand to millions. It is obvious that randomly varying those weights can take very long.

For the backpropagation, the derivative of the error is calculated, fed back, and multiplied by -1 to the corresponding weight. This is done from the last layer and then repeated for each layer till the first trained layer: The derivative of the error propagates back through the network and, thus, reduces the error.

3.2.13 Deep Neural Network (DNN)

A Deep Neural Network (DNN), also known as *deep learning*, refers to machine learning algorithms having multiple layers of artificial neural networks. The input is abstracted for each layer, resulting in more features and a deeper understanding but also increasing complexity and, thus, required resources for the execution. The number of (hidden) layers determines if a neural network is deep. The exact value is not fixed, but networks with two or more hidden layers are usually called *deep*.

3.2.14 Convolutional Neural Network (CNN)

A convolution is a mathematical operation describing how the shape of two functions affects each other. It can be interpreted as a filtering. A Convolutional Neural Network (CNN) is a neural network with at least one convolutional layer, usually combined with a following pooling layer. The convolution with a small, usually 3×3 matrix – the so-called filter, feature detector, or kernel – helps to detect features like shapes in an image. For that, the filter is successively applied to a region of the input till it covers the complete area: it sweeps over the entire input. The output is the feature map or activation map. Afterward, a ReLU function is applied, and a pooling layer reduces the complexity by downsampling using averaging or taking the maximum (c.f. Section 3.2.6). This method reduces the complexity of the input by detecting shapes. Usually, several convolutional blocks, each containing a convolutional and a pooling layer, are combined. After the last block, a fully connected layer maps the features extracted by the previous convolutional blocks to the expected output classes. It creates an output probability usually between 0 and 1.

3.2.15 Transformer

Transformer networks [71] are comparatively new and used to transfer information. One widely used application is the automatic translation. For that, the transformer network is trained with texts in different languages. It finds the correlations and is able to transform the information from one language into another. Transformer networks are also used to summarize text.

3.2.16 Region-Based Convolutional Neural Network

Region-Based Convolutional Neural Networks (R-CNNs) are used in object detection. The input image is converted into several boxes. A classification is performed on those, resulting in a probability of the object in the box belonging to a specific class. The challenge is the smart and efficient creation of the boxes, and varies between the different algorithms.

3.2.17 Support Vector Machine (SVM)

Support Vector Machines (SVMs) is a mathematical method for pattern recognition. The objective is to split a set of objects by dividing them using a hyperplane. This hyperplane is placed such that a maximum area between the objects of two classes is kept free.

3.2.18 Feature Pyramid Network (FPN)

Feature pyramids contain feature maps in different resolutions [72]. Such a pyramid helps to detect objects in various scales and performs better for small objects at higher computational power and memory costs.

3.2.19 Data Augmentation

A good training dataset with sufficient input images is essential to train a good-performing model. Data augmentation can increase the number of training images by flipping and tilting the images, adapting the brightness, cutting subimages, etc., leading to an extended number of images for the training.

3.2.20 Transfer Learning

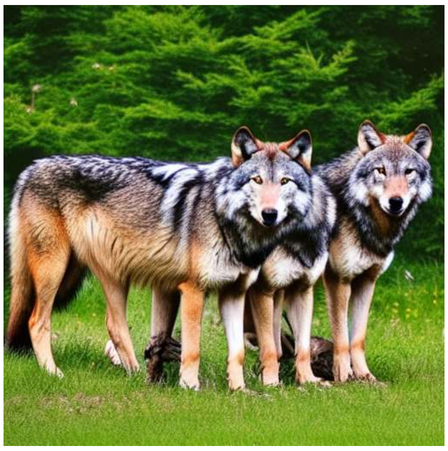
Thoroughly training a model can – depending on the structure and complexity – take a lot of resources regarding time, computing power, and energy. For several years, it has been common practice to use transfer learning [73]. One uses an existing model which was completely trained on a set of classes. If those classes are later adapted or used in a different domain, it is sufficient to only re-train part of the model. The lower layers detect the shapes and fundamental properties. They stay unchanged and are used as a so-called *feature extractor*. The head of the model maps those features to the classes and is retrained. This process of partly retraining the model is called transfer learning and significantly speeds up development.

3.2.21 Domain Adaptation

Training an object detection algorithm can lead to overfitting on the static background, especially when using a stationary camera position in combination with only a few classes [74]. Such a model tends to detect parts of the changing background as the majority class. Applying this model to another domain, i.e., images from another camera or environment, significantly reduces the performance. Domain adaptation is a type of transfer learning that deals with the transferability of a predictive model from a source domain to a different yet related target domain while still trying to solve the same task.

3.2.22 Object Tracking

Multi-Object Tracking (MOT) is an additional task in the field of object detection. The objective is to detect a particular object, i.e., a wolf, and track the movement [75]. The combination of tracking and detection can increase trust in the detection by analyzing whether the movement makes sense over time. We


(a) Even though this picture looks highly realistic on the first view, it has some inconsistencies: The number and positions of the legs and the heads do not match reality.

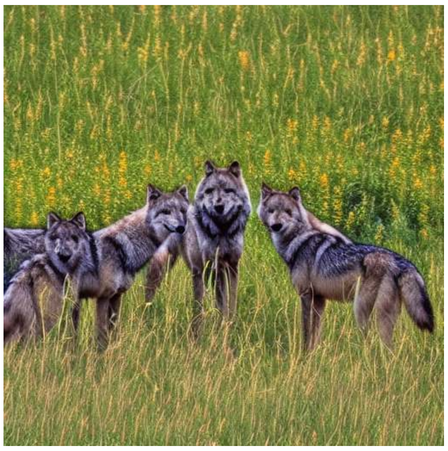

(b) Also, in this image, the faces and the legs show unnatural positions.

Figure 3.6: Two packs of wolves created using Stable Diffusion with the sentence "*A photograph of some wolves. The wolves stand in a field. Some are partly hidden by bushes. The image should be highly realistic and detailed.*"

currently evaluate this object tracking and time series analysis on camera trap images as a master project and do not further discuss it in this work.

3.2.23 Artificial Image Creation

Several technologies like *Stable Diffusion* or *Deepfake* exist to create images using neural networks. They can create photorealistic images and videos as shown, for example, in Figure 3.6. Here, we used Stable Diffusion to generate several photos of wolves. The results look promising at first view but show inconsistencies when looking more carefully at the photo. Problems with AI-generated images mainly occur in the details like the number of fingers, legs, or the details of the face. Those frameworks aim to create good, photorealistic images, contradicting our work with low-quality camera trap images.

3.3 Computer Vision Frameworks

Several frameworks exist in the area of computer vision. They offer ready-to-use models, code blocks to create new architectures, support to train, run, and evaluate the models, and usually a lot of documentation and tutorials. As the focus of this work is not to create a new architecture but to use the existing ones, we give a brief overview of the currently used frameworks. We focus on the frameworks used for classification and object detection and – if applicable – with a focus on wildlife images.

3.3.1 TensorFlow

One of the most well-known frameworks is TensorFlow [76]. It is developed by Google and available for many devices, including embedded ones or even as a Javascript implementation. With the *Model Garden*, it offers various models adaptable to multiple machine learning tasks. On the webpage[5], a lot of documentation and tutorials are available. TensorFlow's primary focus is developing and deploying new models on different devices.

We tested the MobileNetV2 architecture[58] on our wolf image dataset. The MobileNetV2 model was initially trained on the ImageNet dataset, a popular training dataset containing 1000 classes. It includes, for example, the class with the id 269: *timber wolf, grey wolf, Canis lupus.* In theory, this classifier should be capable of detecting wolves.

We first evaluated this model on a wolf dataset we downloaded from Flickr. It contains 289 high-quality images tagged with the keyword *wolf.* The resulting detections with the highest probabilities were *timber wolf* (203), *coyote* (54), and *red wolf* (17). Afterward, we ran the same model on our dataset with 315 camera trap images. Here, the detections with the highest probabilities were *coyote* (31), *African hunting dog* (25), and *tabby* (24). Only three images were correctly classified as *wolf.* One option to avoid this problem is to perform transfer learning. We trained the head of the pre-trained MobileNetV2 on our first wolf dataset containing 1883 images of the classes wolves, bears, and squirrels. The transfer learning resulted in poor results: The accuracy during training and evaluation was too low to be used. The reason for that was our highly imbalanced dataset, which can lead to such results [77]. Several options, like data augmentation, resampling, or creating a balanced dataset, can be used to cope with this problem. This is one of the motivations for this work, as the pre- and post-steps are obviously of high importance.

3.3.2 PyTorch

PyTorch [78] is another well-known machine-learning framework developed by META AI (Facebook). It is widely used in research and focuses more on a clean structure and well-understandable architecture, training, and evaluation of the models. Similar to TensorFlow, it offers many models and pre-trained weights.

We performed the same evaluation of PyTorch as we did for TensorFlow and evaluated the MobileNetv2 pre-trained on the ImageNet dataset. Running the unchanged model on the Flickr dataset results in the following detections: *timber wolf* (208) *coyote* (62), *kit fox* (6). Running the model on our wolf dataset returns *grey whale* (64), *velvet* (24), and *snow leopard* (19) as the main classes. Only ten were detected correctly as wolves. For PyTorch, we also performed transfer learning. In contrast to TensorFlow, the results on our dataset were good: The model detected all wolves correctly. For the Flickr dataset, only 19 wolves were detected correctly. Here, overfitting occurred: The model was trained on a dataset that was too small and similar. This is common for camera trap images. Also, the static camera position of our camera trap can lead to this kind of problem. One solution is here again: a good training dataset as will be offered with our *ShadowWolf*. The different results in the transfer learning compared to the TensorFlow case were caused by slightly different

[5]`https://www.tensorflow.org/`

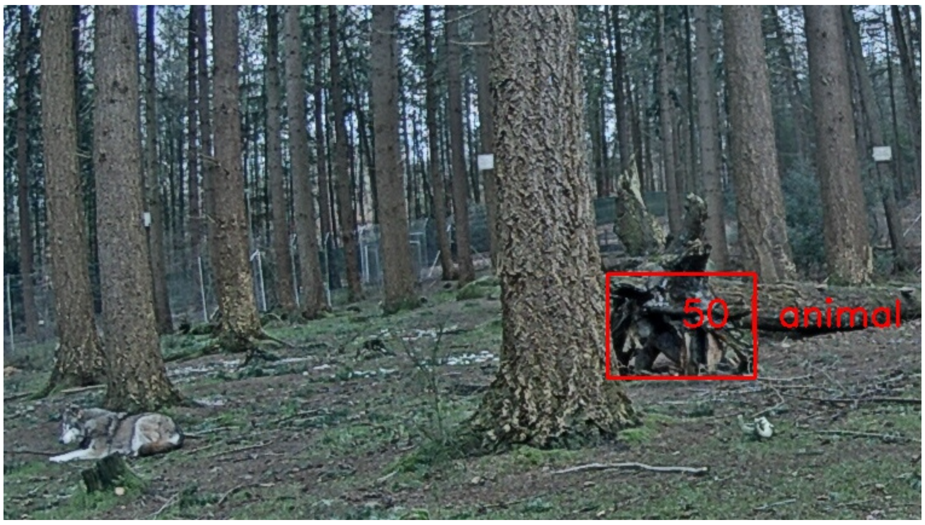


Figure 3.7: An example from MegaDetector: The tree was detected as an animal with 50% certainty. The wolf in the lower left corner was not detected.

transfer learning approaches: The number of fixed and retrained layers differ between TensorFlow and PyTorch.

3.3.3 Keras

Keras [79] was initially developed as an interface for TensorFlow. It is currently extended to support other backends like PyTorch. The objective is to unify the interfaces between different frameworks and simplify the daily work. Autokeras [80] is based on Keras and tries to further simplify the generating and training of new models in machine learning. At the time of writing, object detection is not supported by AutoKeras but is mentioned as currently being developed. Therefore, we did not further evaluate this framework.

3.3.4 MegaDetector

MegaDetector [21] focuses on detecting animals, people, and vehicles on camera trap images. In contrast to the frameworks mentioned above, it only detects if there is an animal but does not identify it on a species level. MegaDetector is based on YOLO (earlier versions on TensorFlow). We ran MegaDetector on our dataset and evaluated the detections manually: How well does it detect wolves? An example image is shown in Figure 3.7. Here, the tree trunk was detected with 50% certainty as a wolf. MegaDetector did not catch the wolf in the lower left corner. The detections are generally good but show weaknesses for sitting, lying, or distant wolves. Furthermore, the models used are pretty big (267M) and require, even on our server, 3.3 seconds per image for the inference.

MegaDetector can be used in *ShadowWolf* to create the bounding boxes in the segmentation phase. Due to the comparatively long inference run-time per image and large model size, we used classic approaches for the background subtraction, which run significantly faster. They also result in more false detections, but the subsequent steps of *ShadowWolf* will filter those out.

3.3.5 YOLO

YOLO (You Only Look Once) [62] is an algorithm with a corresponding toolchain simplifying object detection and based on PyTorch. It produces good results with minimum effort for training and evaluation. Furthermore, it can easily scale to different hardware with varying computing capabilities. Whereas TensorFlow and PyTorch can be used to develop completely new architectures and adapt existing ones, YOLO offers several pre-trained models using their architecture. They also include image pre-processing and data augmentation – steps that should be done manually for the other frameworks and require additional knowledge.

YOLO offers various models with different image resolutions and sizes, resulting in different precisions. Those resolutions are denoted by the letter after the YOLO version number. For example, YOLOv5n is the nano model, YOLOv5s is the small one, and YOLOv5l is the large one. The size affects the training time, the model size, and the execution time. The default models are using images with 640 px. Models with a higher resolution of 1280 px are also available and marked with a six at the end, i.e., YOLOv5n6, YOLOv5s6, and YOLOv5l6. Details about the pre-trained models can be found online[6].

We evaluated YOLO on our dataset and achieved good results. The models being assessed resulted in significantly higher true positives with fewer false positives than the ones from PyTorch and TensorFlow. In contrast to those, YOLO has preprocessing, like scaling the images, adapting the model's input parameters, augmentation, etc. build-in. This reduces the time to optimize the model. Furthermore, the different model sizes help to deploy the object detection to different machine types. This benefits our project context, as one idea is to deploy our trained object detection on embedded hardware. YOLO only has some minor drawbacks in detecting small or distant objects. We counter those with additional steps in our *ShadowWolf* as discussed in Chapter 8.

3.3.6 Summary

In this section, we evaluated several frameworks and architectures commonly used in the field of object detection. TensorFlow is the most prominent one. The tested MobileNetV2 model performs well but requires good training data for the transfer learning. Similar holds for PyTorch: The general performance is good, but a good training dataset is key to train a well-performing model. This is one of our motivations for writing our *ShadowWolf*.

Keras unifies the interfaces for different frameworks and aims to make the start with neural networks as easy as possible. It uses TensorFlow as one of the possible backends.

MegaDetector focuses on camera trap images. It detects animals but does not identify the species. It can be used for the preprocessing in our work but not to detect wolves.

YOLO is a complete ecosystem for machine learning. It performs well and offers different architectures to scale to various applications and devices.

[6] `https://github.com/ultralytics/yolov5#pretrained-checkpoints`, accessed: 2023-10-11

Therefore, we select YOLO and further discuss the evaluation and implementation in Chapter 6.

3.4 Computer Vision Tools

Besides the frameworks mentioned before performing the object detection and classification, several tools are required or recommended to simplify the process of preparing the training data or exporting the results for a specific platform. Tools like cvat [81], LabelImg [82] or Timeworx[7] help to annotate images. They support humans in labeling or to delegate the work to others. Some tools can also use models to detect and annotate objects, reducing the workload.

Depending on the final device and architecture the model should run on, some platform-specific adaptation might be required. The OpenVINO toolkit[8] by Intel takes a model from a variety of frameworks like PyTorch or Tensorflow, converts and optimizes the model, and allows the deployment on different devices. Those can be optimized embedded systems such as the TPU used by the Google Coral.

Handling the images or videos in the pre- or postprocessing of the detection is also an essential part. For that, several toolkits are used to cut parts of the image, import and export the results, draw boxes to visualize the outcomes, etc. *OpenCV*[9] is a general computer vision library mainly offering real-time processing and some machine learning functionality. We use OpenCV to cut and scale the images, draw boxes around our detections, and add text for debugging. Furthermore, it also offers a Saliency API, which we use for background subtraction. *Pillow*[10] is another Python library to process images. In contrast to OpenCV, it has extended support for reading image metadata. Therefore, we use it in our work to extract the Exchangeable Image File Format (EXIF) and IPTC Information Interchange Model (IPTC) information from the images.

The main task of the detection and classification is performed using the state-of-the-art framework discussed and evaluated in Chapter 6.

3.5 Common Metrics in Machine Vision

This work aims to detect wolves in various environments. The objective is to detect as many wolves as possible. Furthermore, all detections should be correct, i.e., every detected wolf should be a wolf. For that, we compare the detection from the model with manual labels, called ground truth. Both are boxes with varying sizes and a class identifying what is assumed to be in this box. Ideally, those boxes should match by 100 %: All wolves were detected and marked correctly, and the model did not miss one. This chapter introduces the metrics we use in this work.

[7]`https://timeworx.io/`
[8]`https://openvino.ai`
[9]`https://opencv.org/`
[10]`https://python-pillow.org/`

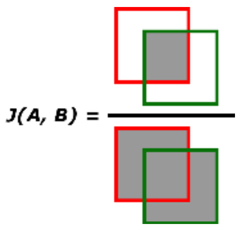


Figure 3.8: The IoU ($J(A, B)$) is defined as the size of the intersections divided by the size of the union of two sets or areas.

3.5.1 Intersection over Union

Comparing the boxes of the ground truth and the detection is done using the Intersection over Union (IoU), also known as the Jaccard index as described by Equation (3.4) and visualized in Figure 3.8. The output lies between 0 (no overlap) and 1 (total overlap). A decision value of $\alpha = 0.5$ is commonly used. If $J(A, B) \geq \alpha$, the detection is considered correct, also called a true positive. In our case, we compare the ground truth from the manual created labels with the detections.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.4}$$

3.5.2 Possible Detection Outcomes

Using the IoU from the previous section, one can decide if a box from the detection matches a box from the ground truth. Here, in general, we can distinguish four cases. In the case of a **True Positive** (abbreviated as TP), an element was detected as the corresponding class and verified, i.e., the detection was correct. This can be expressed as $J(A, B) \geq \alpha$. The second case is the **False Positive** (FP). Here, the algorithm detected an instance that is not confirmed by the ground truth, i.e., $J(A, B) < \alpha$. A **False Negative** (FN) occurs if an instance was not detected, i.e., it is in the ground truth but not in the detection. A **True Negative** (TN) is a special case that is not applicable in an object detection task as not every part of the image belongs to a class. Therefore, we do not have true negative detections.

3.5.3 Precision and Recall

Two commonly used metrics in detection tasks are precision and recall. Both measure the relevance of the detections.

The Precision reflects how accurate the prediction is, i.e., the percentage of correct detections. It is calculated as shown in Equation (3.5).

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.5}$$

The Recall (sometimes referred to as sensitivity) measures if the algorithm found all the instances of a class. Equation (3.6) depicts the corresponding calculation.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.6}$$

Both the, precision and the recall, can have values between 0 and 1, where 1 is the best. We require for our application a high precision, i.e., detect all wolves correctly. At the same time, we also need a high recall as we do not want to miss a wolf.

3.5.4 F_1-score

The harmonic mean of Precision and Recall is the **$\mathbf{F_1}$-Score**. A perfect model has an F_1-Score of one, i.e., all instances are detected, and we do not have any false positives. The F_1-Score is calculated as shown in Equation (3.7). The optimization objective is to get an F_1 as close as possible to 1.

$$\mathrm{F}_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3.7}$$

3.5.5 Other Metrics

The literature commonly uses average precision (AP) or mean average precision (mAP). Here, the precision over multiple classes is averaged; for multi-class object detection, the precision over all classes must be considered. As most of our dataset contains only one class (wolf), we focus on the precision and recall for this class and calculate the F_1-score.

Chapter 4

LoRa Technology

The communication between the components of this work, i.e., the camera system and the deterrents, should be wireless. As we discussed in Chapter 2, LoRa is the technology we use in this work. Figure 4.1 shows the proposed communication architecture. The horizontal dashed line represents the fence. In the upper half, the farm animals – in this case sheep – are located. A wolf is strolling around and examining the fence in the lower half. In the center, the camera system observes the environment. Additional cameras are indicated at the boundaries. The fence is protected with four sound and two flash deterrents. If the camera system detects the wolf, a message is sent out to activate the deterrents. The packet might be received or not depending on the distance and wireless technology. This is indicated by the Packet Reception Ratio (PRR): The deterrents in the white area receive all the packets. The further away the deterrents are, the lower the PRR and the higher the probability that the packet is lost. LoRa can be parametrized in a way that the white area, i.e., where the PRR is close to one, fits our requirements, and the deterrents are activated with a high probability. Furthermore, we can send the same packet several times to increase the chance of successful reception. As a third option, we can also use acknowledgments to confirm the successful reception and resend the packet in case of an error.

This chapter introduces the main functions and parameters in Section 4.1 and also evaluates the performance in Section 4.2. Finally, in Section 4.3, we summarize this chapter.

4.1 Overview of the LoRa Technology

LoRa (short for *long range*) is a proprietary communication technology by Semtech[1]. It was developed for battery-powered, low-energy devices and uses a patented Chirp Spread Spectrum (CSS) modulation. The LoRa alliance[2] focuses on developing LoRaWAN, an extension of LoRa with a networking layer. Some of the specifications and requirements of the LoRa alliance are also relevant to the base LoRa. An evaluation of the technology and the communication ranges are given in [83] and also evaluated by us in Section 4.2.

[1]`https://www.semtech.com/lora/what-is-lora`, accessed: 2023-12-10

[2]`https://lora-alliance.org`

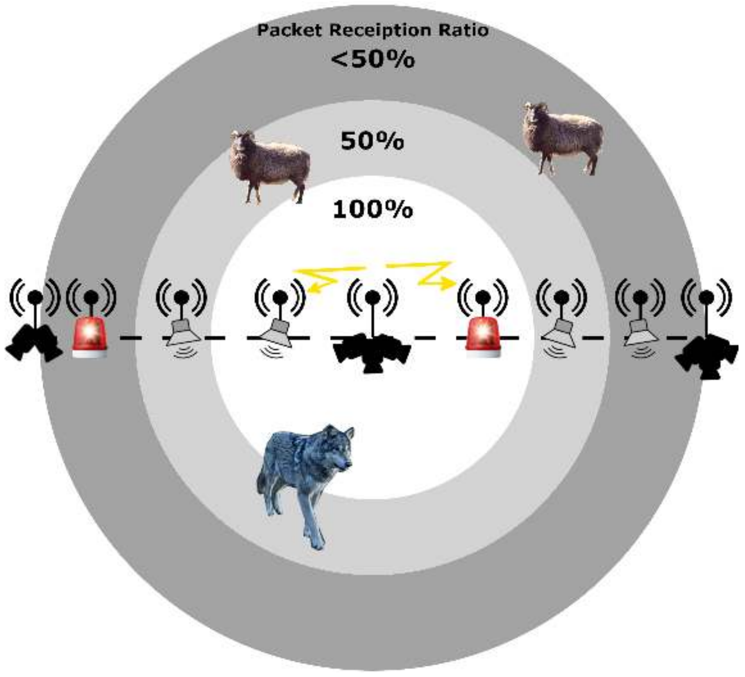


Figure 4.1: Schematic representation of the proposed communication architecture in dependency of the PRR. The top half contains the farm animals. The dashed, horizontal line marks the fence. One camera is located in the center, deterrents, and additional cameras are located along the fence.
The camera activates the deterrents wirelessly. The white area reflects the PRR of 100 %: all sent activation messages are received. This changes over distance and also depends on the environment. The darker the area is, the lower the chance of a successful activation. The required communication distance depends on the application and can be varied by adapting the LoRa parameters.

LoRa can operate in several frequency bands, whereas the most common ones are 868 MHz and 433 MHz for Europe, 915 MHz for North America and 433 MHz for Asia [84, 85]. The communication range varies and highly depends on the environment, devices, and modulation parameters. For outside communication, it varies between 1 km to 5 km in urban regions and 5 km to 15 km in rural areas. In the case of a direct line of sight, the range can be even higher.

Such a long communication range affects all nodes in the range, and the communication should be limited to a minimum. For that, the local authorities, like European Telecommunications Standards Institute (ETSI) for Europe, require duty-cycling. For the 868 MHz as mainly used in Europe, this is $\leq 1\%$ [86], i.e., a node is allowed to transmit at maximum 36 s per hour. Consequently, if transmitting a packet takes 100 ms, the node should not transmit anything the following 10 s. The time it takes to transmit a packet over the air is called Time on Air (ToA) and should be as short as possible. Several LoRa parameters influence the ToA and can be set up on the transceiver.

The **packet size** is one option to affect the ToA. The larger the packet is, the longer it takes to transmit it. For LoRa, the maximum size of a packet is 256 B [87] but depends on the other modulation parameters. In the worst case, only 11 B can be transmitted.

The modulation itself is characterized by three parameters: The spreading factor, the code rate, and the bandwidth. The **spreading factor** (SF) can be set in the range of 7 to 12 and defines the number of symbols used to transmit the data. Practically speaking, a low spreading factor of 7 results in a shorter ToA and a reduced communication range, whereas a longer spreading factor increases the range but also the ToA. The LoRa spreading factors are orthogonal, i.e., transmissions on the same channel with different spreading factors do not interfere.

The **code rate** (CR) influences the amount of error correction added to the data and can be $\frac{4}{5}$, $\frac{4}{6}$, $\frac{4}{7}$ or $\frac{4}{8}$. Usually, only the denominator is given as, for example, 5 in the case of $\frac{4}{5}$, meaning that 4 bits of data are transmitted using 5 bits. As for the spreading factor, the code rate highly influences the ToA.

The third parameter is the **bandwidth** (BW) and can be 125 kHz, 250 kHz, and 500 kHz. The higher the bandwidth, the shorter the ToA, but also leads to a lower receiver sensitivity and, thus, a reduced communication range.

Besides the three modulation parameters, two options affecting the payload size itself are available. One is the header with a payload length and a Cyclic Redundancy Check (CRC) checksum. The second is an CRC over the complete data. Both are optional.

Table 4.1 shows the effect of the different LoRa modulation parameters. We calculated the time it takes to transmit a packet with a given length of 48 B, the amount of data we later use in this work. Considering the duty cycle of $\leq 1\%$, we also calculated how long we must wait to transmit the next packet. We deactivate the CRC and use the explicit header as, according to our experience, this offers the best performance and interoperability. One can see the effect of the parameters: high bandwidth, low code rate, and low spreading factor lead to a fast transmission and allow sending more packets at the drawback of a lower range. In contrast, low bandwidth, high code rate, and a high spreading factor reduce the number of packets that can be sent but increase the range.

SF	BW	CR	TX Duration	Next packet after
7	125 kHz	4/5	87.30 ms	9 s
7	125 kHz	4/8	127.23 ms	13 s
7	250 kHz	4/5	43.65 ms	4 s
7	250 kHz	4/8	63.62 ms	6 s
12	125 kHz	4/5	2138.11 ms	214 s
12	125 kHz	4/8	3022.85 ms	302 s
12	250 kHz	4/5	1069.06 ms	107 s
12	250 kHz	4/8	1511.42 ms	151 s

Table 4.1: The effect of the different LoRa parameters for a payload of 48 B, explicit header and no CRC for a spreading factor of 7 and 12, a bandwidth of 125 kHz and 250 kHz and the code rates of $\frac{4}{5}$ and $\frac{4}{8}$. The grey line marks the parameters we use in this work.
SF = Spreading Factor, BW = Bandwidth, CR = Code Rate

The datasheet for the SX1276/77/78/79 LoRa transceivers from Semtech[3] describes the calculations used in Table 4.1. An online calculation tool is also available[4].

In the next section, we will evaluate LoRa's real-world communication range.

4.2 Evaluation of LoRa

In our department, we use LoRa for a variety of different projects, both indoor and outdoor. A colleague used LoRa for vehicular communication and evaluated the effect of the speed in [88]. Students use LoRa to build an in-house communication and location system. Other colleagues made a LoRa-based messaging system in [52] and achieved a communication distance of several kilometers. In [89], we used LoRa in the rainforest to transmit mosquito vector counts just to give some examples.

We usually start with the standard LoRa parameters and set the bandwidth to 125 kHz, the spreading factor to 7, and the code rate to $\frac{4}{5}$. Only if the range is not sufficient, we adapt those. With this, we ensure we do not occupy the channel more than necessary.

The landscapes where wolves were sighted vary. Sometimes, the system has to work in the flat north of Germany, sometimes in the hilly south, sometimes covered by a dense forest. We did not evaluate LoRa for all those environments but took a comparable example from the Suan Phueng, Thailand Rainforest. Due to the denser vegetation compared to Germany, this test is a worst-case evaluation. Here, we used two nodes: A fixed receiver and a moving transmitter. Every time a packet was sent, the current position was stored. For the hardware, we used lopy4 by pycom. Those devices consist of an ESP32 microcontroller combined with a Semtech SX127x transceiver. We used the default LoRa parameters as mentioned above an marked in Table 4.1. The software

[3]`https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiAbqcpqh9qSjE`, accessed: 2023-12-10
[4]`https://loratools.nl/#/airtime`, accessed: 2023-12-10

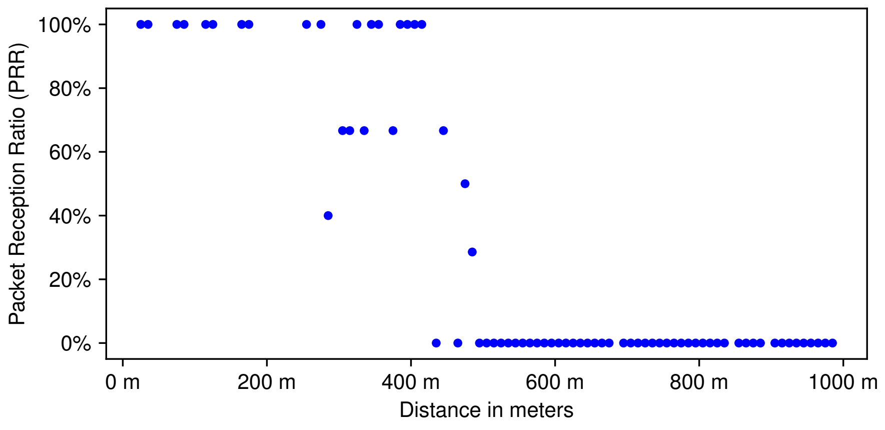


Figure 4.2: The PRR as measured in Thailand. All packets are received up to 300 m, and packets were lost after 500 m. In total, 74 packets were evaluated.

and the measurement results can be found in our GitHub repository[5].

From this information, we can plot the PRR over the distance as shown in Figure 4.2. In total, we evaluated 74 packets. Up to a distance of 300 m, all packets were received, leading to a PRR of 1. Increasing the distance to more than 500 m results in a complete loss of all packets. Those results can be related to the circles in Figure 4.1: The white circle corresponds to the distance up to 300 m: In this range, the deterrents will most likely be activated. The actual range will be higher in a free field with a direct line of sight. Also, the antennas used and their orientation can influence the communication range: Well-aligned external antennas usually perform better than internal ones. As a drawback, they break easily if not handled carefully.

The expected range of LoRa is significantly higher than the expected ones for the object detection and the deterrents used in this work. Therefore, we find LoRa a valid choice for the communication.

4.3 Summary

In this chapter, we introduced our wireless architecture based on LoRa. We discussed the most important parameters and their effect on the transmission time. We also discussed the maximum number of packets we can send in a given time. We evaluated the communication range regarding the packet losses: Up to 300 m, no losses occur. This range is higher than the ones of the other parts of the system. Therefore, we find LoRa a valid choice. We will further evaluate LoRa for different environments during the test phase of the complete system at the end of the scheduled project time of *mAInZaun*.

[5]`https://github.com/ComNets-Bremen/LoPy-Distance`

Chapter 5

Data Collection in the Wild

A good dataset is essential to train a machine learning algorithm. Good, in our case, means also realistic. Several datasets exist showing glossy, high-quality images of various animals. We focus on real images that are automatically captured in the wild, including the ones that are usually removed because of their low quality. We did not find a suiable dataset meeting our requirements and collected our own, as discussed in Chapter 2.

This chapter introduces our hardware setup in Section 5.1. The challenges we faced while collecting our datasets are discussed in Section 5.2. Section 5.3 describes the datasets we use in this work. Section 5.4 summarizes this chapter.

5.1 Our Camera Setup

This section describes the setup we used to collect our wolf image dataset. It is related to the hardware in Chapter 9 where we discuss the considerations for deploying the entire system. The objective of the setup in this section is to collect as many training images as quickly as possible. The detailed list of parts we used, as well as some pictures, can be found in Appendix A. Table 5.1 lists an excerpt with the main expenses.

As a camera, we used a model from Axis: The M2025-LE. It offers good low-light capabilities, has a built-in infrared light to record images during the night, and is certified with IP66, i.e., it comes with good weather protection. The sensor has a full-HD resolution (1920×1080 pixels), and the lens has a horizontal field of view of 115°. Furthermore, it supports capturing images in High Dynamic Range (HDR)-mode, which is highly interesting to us: If the sun points directly to the camera (contre-jour), the dark areas in the shadow become invisible using a standard camera, as they usually do not support such high differences in brightness. Cameras with HDR-support perform tone-mapping to adapt to this increased dynamic range: One can see more details in this light condition as it will be shown and discussed in Figure 5.1 on page 54.

Also, cheaper cameras are available. The different prices depend mainly on the low light capabilities, the quality of the lenses, and the sensor's resolution. Systems from several tenths of euros up to several thousands of euros are available on the market, offering many options for the user. One has to find a trade-off: For example, a high-resolution system might work better, but it also requires more storage and computing power to evaluate the images and run the inference. This is, especially for in-field computing, a critical point.

In our case, the Axis M2025-LE met all the requirements regarding price, field of view, resolution, weather protection, low-light capabilities (including infrared-spots), and further options like HDR-mode. The price is approximately 425 € as of December 2023.

The camera has built-in motion detection. We used that to capture our images: Two images per second are recorded if motion is detected. The recording starts one second before the motion is detected and stops five seconds after the movement ends. Furthermore, we take one picture every full hour to have a regular image as a kind of heartbeat. For motion detection, we ignored areas with treetops using the camera's built-in masking capabilities and focused only on the spots where animals could pass by (a wolf usually does not fly). Still, bushes and other moving things were in the area, resulting in images with no animals.

The collected images were stored on a RaspberryPi version 4 with 4 GB of RAM and an SD card with 16 GB. Every night, the images from the previous day were moved to an external, 1 TB SSD, preventing the SD card from running full. This setup is used to collect images for later training. Keeping the images is not required for a real deployment while running the detection. Everything is packed in weather-protected cases. The detailed list of parts is available in Appendix A.

Every hour, 10 images were uploaded via a cellular connection to our server to ensure the system was running and remotely monitor the image quality. We did not optimize it regarding energy as we always had the main power available.

The following section will discuss the considerations and challenges we faced during the image collection.

5.2 Challenges in Outdoor Image Collection

Images automatically captured in outdoor environments significantly differ from those collected in indoor or lab environments. This section describes the various challenges and pitfalls we encountered during our collections. Those were also published in our previous work [90, 91].

5.2.1 Casing and Protection

Deploying a high-tech system in the wild yields several challenges. The obvious ones are in the hardware itself: The devices have to survive in harsh outdoor environments, i.e., dust, rain, snow, hail, etc. Also, people might drop it, stand on it, turn it over, etc. For that, a proper casing is required. The protection standard regarding the ingress in a casing is defined using the so-called IP Codes, which consist of two numbers: The first stands for the protection against solid particles, the second for the water protection [92, 93]. For our system, no dust should enter, i.e., the first digit is 6. Regarding water for a final product, we aim to achieve 7 or 8 so the devices can be submerged for a short period of time. This is quite challenging, and for our currently used setup, we reach 5 or 6 (waterjets from all directions). The resulting protection is called IP 65 or

Component	**Price**	**Comment**
Axis M2025-LE	425 €	Camera
Raspberry Pi 4	120 €	Bundle with case, SD-card etc.
SSD 1 TB	70 €	For image storage
Huawei E8372 LTE	75 €	WiFi Router (LTE)
Outdoor Case	45 €	For the complete system
PoE-injector	35 €	Power supply for the camera
USB power supply	35 €	For router and RaspberryPi
Incidentals	50 €	Cables, breakthrough, connectors, plugs, glue, etc.
Total:	**855 €**	

Table 5.1: The costs of the main components of our camera system used for the data collection. We took care that the system can operate outdoors.

IP 66. It is essential to ensure high IP levels for all involved parts of the system, especially plugs, switches, cable glands, lights, etc.

Another commonly neglected challenge is the humidity and temperature differences, which continuously change in outdoor environments. Hot air contains more water than cold air. Due to the changes between day and night, the humidity from the air condenses inside the case, resulting in a notable amount of water, even in entirely tight cases. Pressurization valves (breathable valves) can act as a countermeasure. Those valves let the air, but not the humidity, pass. This technology is also commonly used for clothing and shoes.

The complete system must be protected against shock damage during handling. Here, several boxes are available on the market, and selecting one is not a real challenge. According to our experience, equipment from the automotive sector is a good starting point for the casing: They are protected against harsh conditions on the road, are not very expensive, and are available. The case should also be rated for outside usage, i.e., in the direct sun. Ultraviolet light causes standard plastic to age faster, and the case breaks earlier. Depending on the environment, theft protection and protection against vandalism should also be considered.

5.2.2 Costs of the System

Regarding the costs, *mAInZaun* has larger deployments in mind. Therefore, we focus on off-the-shelf commodity components to keep the costs comparatively low. Table 5.1 gives an overview of the costs of the main components of our camera system. The most expensive part is the camera itself. We selected this model mainly because of its good low light and night vision capabilities and IP66 protection against heavy rain. The remaining components are consumer parts. The total cost of one box sums up to 855 €.

Those costs can be reduced for a real product, and the possible options are discussed in Chapter 9.

5.2.3 The Power Supply

Operating a camera system, especially with object detection, requires energy. For the data collection as discussed in this chapter, we always had mains power available. The options for a completely autonomous, self-contained system in a remote environment without permanent access to the power grid are briefly discussed in this section.

We neglect the energy required by the deterrents for now and only focus on the system's detection parts. An overview of selected parts and their power consumption is given in Table 5.2. We focus on two types of systems: One type uses an off-the-shelf surveillance camera from Axis connected via Ethernet to a computing device, like the RaspberryPi, NVIDIA Jetson Nano, or Google Coral. This solution has the advantage that we can use all the flexibility of the hardware and easily extend the system. Alternative solutions are, for example, the Luxonis cameras, which can run AI models directly on the camera itself. Those are very powerful but restricted regarding additional functionality like additional image processing. For reference, we also added a WiFi-LTE router from Huawei for Internet access.

Component	**Functionality**	**Power (W)**
RaspberryPi 4	Processing	12.5 W to 15 W
NVIDIA Jetson Nano	Processing	10 W to 20 W
CORAL Dev Board	Processing	10 W to 15 W
AXIS M2026 LE	Camera	5.1 W to 7.9 W
Luxonis OAK-D Pro	Camera, Processing	10 W
Huawei E8372	Internet Router	5 W

Table 5.2: Comparison of different components and their energy requirements according to their datasheets.

As an example, we consider a USB power bank with an approximate capacity of 10.000 mA h at 5 V, i.e., 50 W h, and ignore all the losses and side effects, additional converters, etc. We can estimate the resulting lifetime: Assuming all devices operate at 5 V, a 10 W device draws a current of 2 A, and the power bank will be empty after approximately 5 h. If we consider the surveillance camera with a RaspberryPi and an internet router, we draw 30 W or 6 A. The power bank will be empty after roughly 100 min. For this setup, the voltages have to be converted, and losses occur, yielding a significantly lower real lifetime.

Also, an alternative battery like a lithium or a car battery can be used. Those have a higher capacity, for example (55 A h) at a higher voltage (12.8 V, resulting in 704 W h) and higher weight (10 kg) for the car battery. Assuming a perfect voltage regulator again, we could get 140.000 mA hat 5 V leading to an estimated lifetime of 24 h. With the 10 W load, the battery will last 70 h.

Using a solar panel might be an option to extend the lifetime. As the system also has to work during nighttime, cloudy days, and wintertime with only a little sunlight, the solar panel should offer at least 4-6 times the power that is required for the system. When it is placed in the field, it has to be aligned properly, cleaned regularly, should not be shadowed by trees and bushes, etc. Also, the size of the required panels and the required stand are notable challenges. Right now, they might only be considered for quasi-stationary use cases but not for mobile ones.

The power supply can be split into two main challenges: First, carefully selecting and fine-tuning components regarding energy consumption is vital for a long lifetime in remote environments. Solutions like wakeup mechanisms or using different sensors can significantly increase the system's lifetime. Second: The power supply design is crucial if real-time data processing or even collection is planned.

For the final product, custom-made hardware might be an option. The options for that are discussed in Chapter 9.

5.2.4 Transport: Weight and Size

The third challenge is the deployment itself. Farmers, who are usually non-technical users, should be able to deploy the systems into sometimes very remote areas. Therefore, everything should be as lightweight as possible and can be set up quickly. Complex alignments of the devices or software setup procedures should be avoided. Ideally, the overall system should be able to perform some fundamental self-tests and notify the user in case of any issues.

In some cases, like for the levees in Germany, no heavy machines can be used. Therefore, the weight (and thus the battery size) is limited as the device might be carried by hand. Also, regular maintenance, like changing batteries, refilling materials, or just cleaning lenses or solar panels, should be minimal.

5.2.5 Notifications and Communication

The status of the system should be communicated regularly: Is a wolf detected? Is a battery running low? Did a device fail? This kind of communication requires a cellular or, more generally, internet connection. The communication in the field, i.e., to activate the deterrents, can be different and is discussed in Section 9.4.

The typical technologies in this area depend on the environment, such as WiFi, cellular networks (4G, 5G), satellite (Starlink, Iridium, etc.), SigFox, LoRaWAN, etc.

Here, the bandwidth and the energy requirements highly vary between less than a Watt (LoRaWAN) up to 50 W to 75 W (150 W in high-performance mode) for Starlink. The latter power requirements can not be met over a longer time using batteries. We assume that a simple cellular connection, optionally with an external antenna, might usually work to transmit status messages.

5.2.6 Environmental Challenges

Besides the technical challenges, as discussed in the previous sections, several environmental occurred during our data collection.

5.2.6.1 Light Conditions and Shadows

Light is essential to capture images. This can be the normal sunlight or artificial light as offered, for example, by the built-in infrared spots of our camera. Not only the light but also the direction is important. Figure 5.1 shows an image

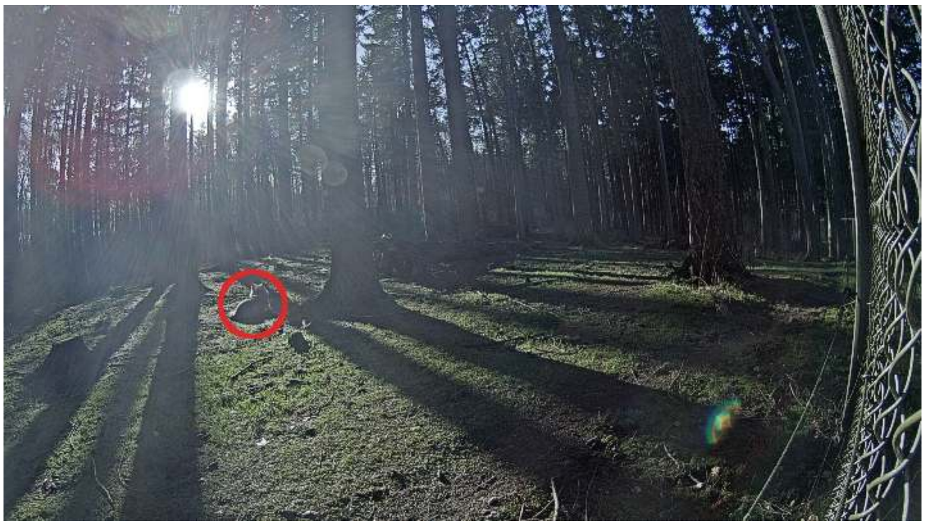


Figure 5.1: An image captured in contre-jour: The sun pointing directly to the camera hinders detection. Furthermore, the trees result in significant drop shadows.

in contre-jour: The sun shines directly to the camera's lens. Due to the HDR mode, one can still see some details. For cheaper cameras, this picture would show fewer details. In this light situation, the resting wolf marked by the red circle is hard to detect and can barely be distinguished from bushes or tree trunks.

The vegetation, clouds, and sunny days also create shadows, as can be seen in Figure 5.1. During windy days, single clouds let the tree shadows seem to move quickly. The motion detection will act accordingly and record many images and, thus, possibly triggering false positive detections.

Shadows can also occur during the night: Figure 5.2 shows such an example: The wolf shows a large shadow caused by the moon.

Capturing images requires a certain amount of light, either daylight or artificial light as the infrared light provided by our camera. The minimum amount of light necessary to capture images is often given in the unit lux (lx). For our camera (Axis M2025-LE), the minimum light in color mode is 0.2 lx, and for greyscale 0.04 lx.

Another parameter for the light is the exposure time of the camera. In our case, this can be between $\frac{1}{66500}$ and 2 s. If only little light is available, the exposure time has to be high to get enough light for the CMOS sensor. This results in blurry and unsharp images for moving objects, as shown in Figure 5.3. Here, a wolf runs through the field of view of the camera. The three pictures were taken within one second. Low exposure times in an environment with insufficient light result in too dark images.

This kind of image is deleted in many datasets because of the low quality. For our project, we especially want them to evaluate the performance of the models. This is one of the reasons why we created a new dataset.

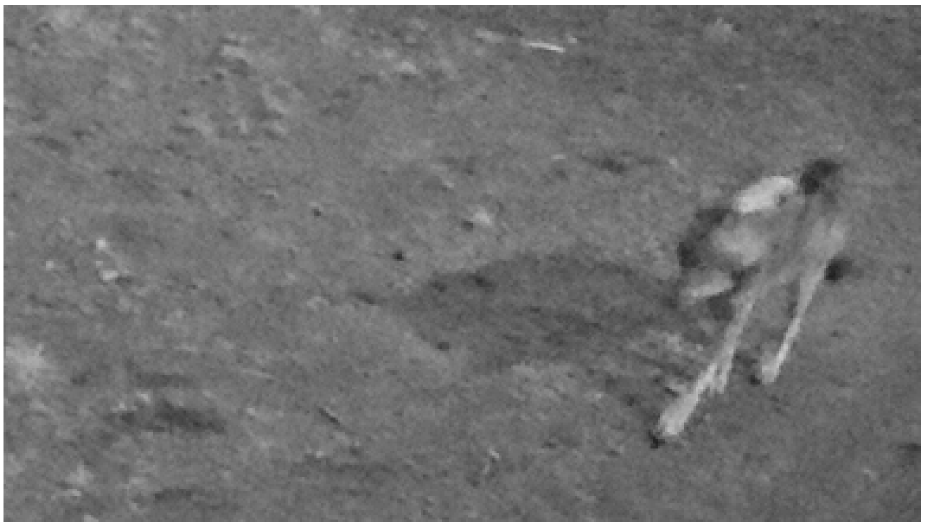


Figure 5.2: A picture captured in night vision mode in the early morning. The shadow of the wolf is larger than the wolf itself.

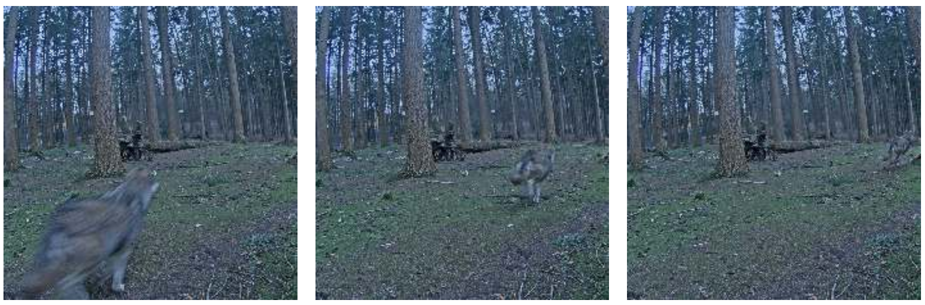


Figure 5.3: Three pictures taken within one second: A wolf runs through the camera's field of view.

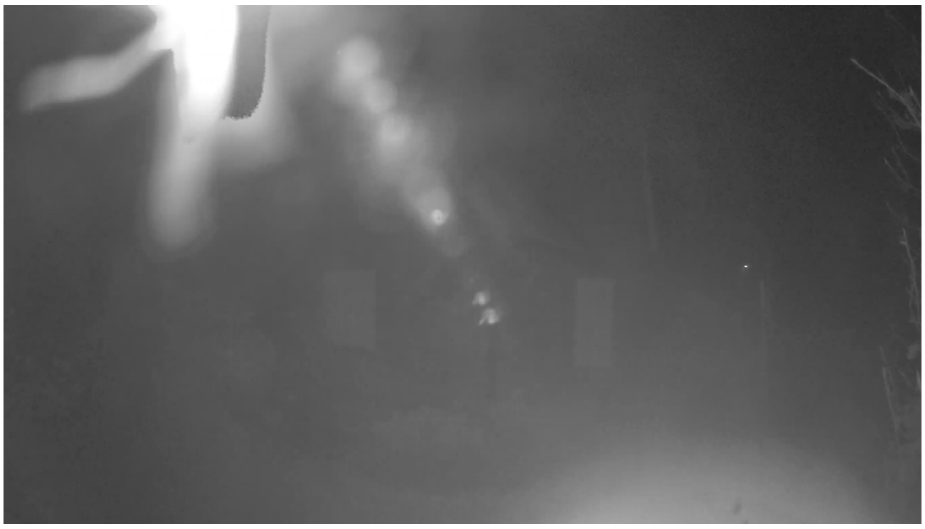


Figure 5.4: A spider in front of the lens. The infrared spot enlightens the insect, blinding the sensor. This kind of image can barely be used.

5.2.6.2 Dirt and Pollution on the Lenses

Depending on the environment, the weather, and the camera's orientation, dirt can play a significant role. Fortunately, this issue did not significantly affect our wolf dataset. We know from our other camera images captured outside that not cleaning the lenses regularly can result in a large amount of at least partly unusable images.

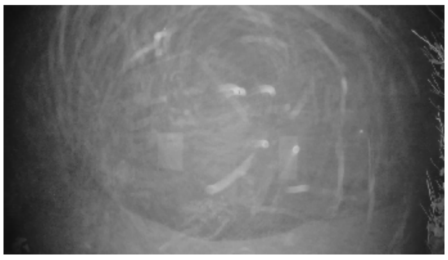
5.2.6.3 Insects

Insects can also significantly affect image capture. For example, Figure 5.4 shows a spider building its web directly in front of the lens during nighttime. The infrared light, which is located just a few millimeters from the lens, enlightened the spider, resulting in unusable images. We tried several methods like odorants or herbs to deter the spider, but nothing worked effectively or lasted longer.

Flying insects like mosquitos, moths, and bugs also reflect infrared light, resulting in bright spots passing through the image. Those are usually short-timed and do not significantly affect the overall detection.

5.2.6.4 Precipitation

Precipitations like rain, snow, fog, etc. can reduce the range of the optical camera system significantly. Figure 5.5 shows the effect of rain. Heavy rain during the night can make the images unusable. Figure 5.5a shows this effect. The infrared light next to the lens supports the negative impact of the rain by blinding the camera. During the daytime, this effect does not occur. Figure 5.5b shows a usable image taken during rain in the daytime.


(a) Heavy rain during the night. Due to the infrared light, almost nothing can be seen.

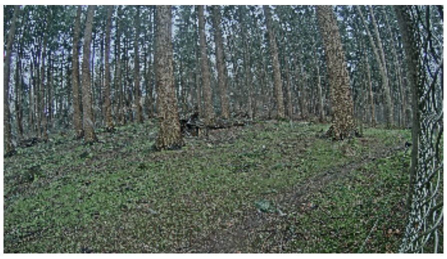

(b) Heavy rain during the day. It can be seen more than at night, but the sight is still reduced.

Figure 5.5: The effect of heavy rain during the daytime and the nighttime.

5.2.7 Lenses and Optical Challenges

Besides the environmental challenges discussed in the previous section, physics also sets limitations regarding optical properties. The camera's field of view is critical for the detection. Our Axis M2025-LE has a horizontal resolution of 1920 pixel for a field of view of 115°. The vertical resolution is 1080 pixel for a field of view of 64°. With three perfectly aligned cameras, one can almost achieve a 360° view ($3 \cdot 115° = 345°$). Such a wide-angle lens has two drawbacks: First, the resolution, especially for far away objects, is comparatively low, which hinders good detections. Second, such a wide-angle lens results in distortion, i.e., straight lines might look bent. This can be problematic for objects not located in the center of the images: The objects look different depending on the area they are located at, as shown in Figure 5.6. One can now use a different lens with a smaller field of view. This helps to get high-resolution images from distant objects but also increases the required number of cameras to cover a particular area.

Especially for deployment in the field, one also has to keep in mind that an increased number of images and a high resolution lead to higher processing power requirements. This directly increases the costs and also the required energy.

5.2.8 Camera Position

Images captured using camera traps are prone to overspecialisation in a specific domain. This happens due to the relatively static background of camera trap images, which mainly differ in the weather and lighting conditions. The low background variance might lead the model to learn by leveraging the specific backgrounds of specific camera traps. In the classification case, if the camera traps have an unbalanced class distribution, the model might often predict the majority class for a given background, meaning that it is not learning to classify objects but backgrounds. That means that the model learns to predict "*wolf*" for the images from a specific camera position because most training images for this camera are of class "*wolf*". The model detects that a particular camera position took the images. Similarly, in the detection case, this might lead the model to detect non-background objects as the majority class.

One has to be aware of this issue when working with static camera trap

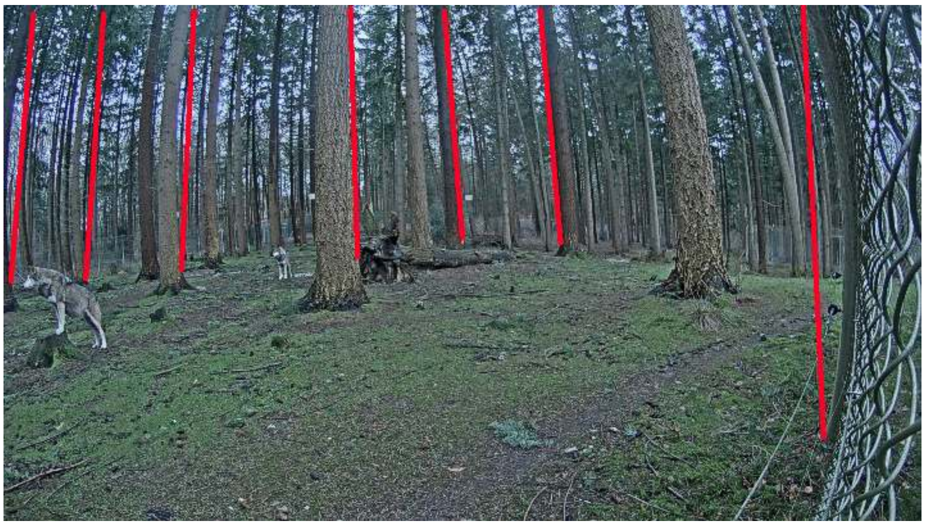


Figure 5.6: The effect of the distortion using a wide-angle camera: The red lines connect two points in the image, which should be straight like the trees or the metal pole at the right side of the photo. Due to the distortion, those look more bent the more they are located at the side of the image.

		Images	
Park	**Timespan**	**Number**	**Size**
Alternativer Bärenpark Worbis	2021-09-13 – 2021-09-14	9536	13.5GB
Wildpark Lüneburger Heide	2022-03-28 – 2022-03-30	27543	47.5GB
Wingster Waldzoo	2023-01-11 – 2023-02-02	63249	91.5GB

Table 5.3: Overview of the parks and the amount of collected images. The complete information about our datasets is given in Appendix B.

images to train a new model. One can use different camera angles by changing positions, cutting the collected images, or using post-processing technologies like domain adaptation [74]. In our work, we address this issue by offering the option to cut the training images to the area containing the object of interest. Further, we can easily adapt to new camera positions using our automated approach. Both help to mitigate the effect of static camera positions.

5.3 Our Datasets

In Section 5.1, we described our camera system's hardware setup and configuration. We use this system to collect our dataset. The details, like the number of images, the time and location where we collected the images, and examples are provided in Appendix B and summarized in Table 5.3.

We collected more than 100.000 images in three parks in northern Germany. The first images were collected in September 2021 in Worbis. In the *Alterna-*

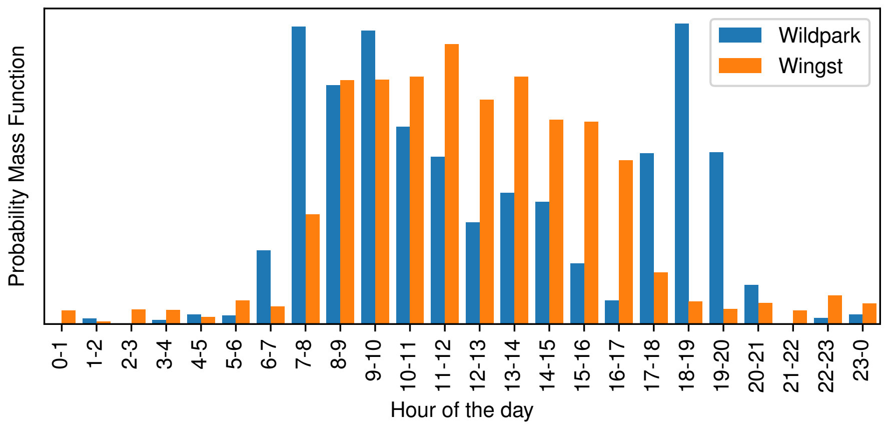


Figure 5.7: The distribution of our image dataset over a day: The time most images were captured varies from park to park. During the night, images are only recorded if motion in the range of the built-in infrared spot is detected. For the Wildpark, we collected 27538 images in two days. For the Waltzoo at the Wingst, we collected 63212 images in 22 days.

tiver Bärenpark Worbis, three American timber wolves live together with several brown bears. This was our first set of images. Approximately half a year later, we collected the first images showing European Grey Wolves in the *Wildpark Lüneburger Heide*. Here, we had the chance to monitor two wolves at the end of March 2022. Our largest dataset originates from the *Wingster Waldzoo*. We took more than 60.000 images of five Grey Wolves in January 2023.

The wolf breed mainly living in the wild in Germany is the European Grey Wolf, as kept in the last two parks, namely Wingst and Lüneburger Heide. Therefore, we focus on these datasets and neglect the one from the park in Worbis showing the Timber Wolves. Anyhow, even the timber wolves might be of interest to the community. Therefore, we plan to publish this data as well.

A few images were corrupted and removed. We collected 90750 images from Grey Wolves, where 79638 (88%) were captured during the daytime and 11112 (12%) at night in infrared mode, as shown in Figure 5.2.

The number of taken images depends on two things: 1) the activity of the wolves and 2) the range of the camera. The latter is highly affected by the environmental conditions. During the night, the camera only detects motion if it occurs in the area covered by the infrared spot. Furthermore, fog, rain, and snow also affect the range. Figure 5.7 shows how many pictures were taken in which timespan of the day for the individual parks.

The wildpark shows two peaks: one in the morning and one in the afternoon/evening. Our largest dataset from Wingst shows activity during the opening and working hours from 07:00 - 18:00.

For this work, we focus on our own collected dataset as we know the properties of the images and the used hardware best. We use it to create our reference dataset described in the next section.

5.3.1 The Annotated Reference Dataset

In Section 5.3, we described the source of our images. We focus on the ones containing the European Grey Wolves, as one can find in the wild in Germany. From those images, we create our reference dataset. It should contain images from different angles and cover various light and weather conditions. Furthermore, the time series should be kept, i.e., we focus on a series of images.

For the dataset from the Wildpark, we took one time series per hour, approximately 30 minutes after the full hour (07:30, 08:30 etc.) between 06:30 and 20:30. We ensured the complete series of images were used. This results in 390 images, all collected on the 29th of March 2022. We have data over several days for the dataset from Wingst. To reflect also the changing weather conditions, we created a representative day of images: Every time series is taken from another day: The images for 06:30 were taken on 2023-01-14, the ones from 07:30 on 2023-01-23, from 08:30 on 2023-01-15 etc. In total, we got 750 images between 06:30 and 21:30 from the Wingst.

This results in 1140 images captured in different environmental conditions. 952 of them contain at least one wolf. Those images were manually labeled with bounding boxes for individual wolves with the tool LabelImg and used as our reference dataset.

Additionally, we use the IPTC-IIM standard to store additional metadata (i.e., keywords) in the image file. Many standard image processing tools and graphics editors can read and write this data. In our case, we used the open source software digiKam[1] to set weather and light relevant keywords: *rain*, *snow*, *fog*, *sunny*, *twilight*, *night*, *overcast* and *contre-jour*. Each image can have several (and arbitrary) keywords. As we will analyze our detections regarding the light conditions later, we use this predefined list of terms.

Furthermore, we use the number of colors of the images to determine if they were captured during day or night: For our camera, the day images contain all three color channels (RGB), the night images only one (i.e., the values for the red, green and blue channel are identical). Table 5.4 lists the number of images for each property and the corresponding source.

This reference dataset is our primary labeled one and was created to reflect as many realistic scenarios as possible. It contains only the European Grey Wolves in different situations. It also reflects various weather and light conditions.

5.3.2 Base Dataset

While collecting and analyzing the first images, we created several smaller datasets. One of these is the *base dataset*. It contains random images from the first two parks (Worbis, Wildpark Lüneburger Heide) where we placed our camera and, thus, contains not only the European Wolves but also brown bears and American Timber Wolves. This dataset contains in total 1607 images and is split into a train (65 %), test (15 %) and validation (20 %) dataset. We manually created this dataset's labels and used them to train our first wolf detection model. We use this model as the base model for the performance evaluation of *Wolf-or-Not* and to show how the model adapts to the different wolves.

[1]https://www.digikam.org/

Property	Data Source	Count
Day	Image	806
Night	Image	334
Overcast	IPTC	389
Sunny	IPTC	357
Night	IPTC	334
Twilight	IPTC	18
Rain	IPTC	50
Contre-jour	IPTC	0
Snow	IPTC	0
Fog	IPTC	0
Total		1140

Table 5.4: The number of images in our reference dataset with a given property and data source. One image can have multiple IPTC keywords. Therefore, those do not sum up to the total number of images.

5.3.3 Test Dataset

Another dataset is our *test dataset*. We created it to evaluate the performance of *Wolf-or-Not*. It contains 1526 randomly selected, manually labeled images from the five wolves living in the Wingster Waldzoo. We use this dataset to evaluate the change of the detections while using *Wolf-or-Not* to create new labels and models.

5.3.4 Automatically Labelled Datasets

This work aims to create automatically labeled datasets using voluntary users, models, or a combination of both. We use iterative processes, creating a new dataset with each run. Those automatically generated datasets will be discussed and evaluated in the corresponding chapters.

5.4 Summary

A good, realistic dataset is important for this work. This chapter described our setup: How did we collect our images? Further, we detailed all our challenges, from financial to environmental to technical. Finally, we described the datasets we used in this work.

Chapter 6

Evaluation of YOLO

In Chapter 3, we introduced machine vision and the relevant frameworks. In Section 3.3. YOLO (You Only Look Once) showed several advantages compared to the other frameworks, including good detections, different models that allow adaptation to the available hardware, and an active community. In this chapter, we evaluate YOLO and give further details.

6.1 Evaluation Setup

We used several physical machines to evaluate the performance of YOLO and compare it regarding the runtime. On all of them, we installed the same YOLO version from the GitHub repository[1]. Due to the different Operating System (OS) versions, the Python versions and some libraries differ. During the tests, we were the only user on the machine to reduce the effect of other processes on the results. We did not repeat the experiments due to the comparably high runtime. We trained three high-resolution YOLO models with different complexities, namely YOLOv5l6, YOLOv5s6, and YOLOv5n6. We used our reference dataset from Section 5.3.1 with 1140 images for the inference.

Our **GPU-server** is the main machine. We used it for the training and inference. The CPU is one 6-core *Intel Xeon Bronze 3204*. GPU processing is performed using an *NVIDIA RTX A5000*. The complete system has 128 GB of RAM and an SSD for the storage. The OS is a *Ubuntu 22.04.3 LTS* with *Python 3.9.12* and *torch-1.13.0*. For the GPU processing, we used version 520.61.05 of the NVIDIA driver together with *CUDA 11.8*.

A normal **CPU-server** was used to evaluate the inference on a powerful machine. It uses two *Intel Xeon E5-2699 v3 with* 2.3 GHz resulting in total in 36 cores. The system has an SSD for data storage and 256 GB of RAM. The OS is *Debian 10.13*. We used *Python 3.11.4* and *torch-2.1.0*.

We also tested the model on a standard **PC** with an *Intel Core i7-5600U* with 4 cores. It has 12 GB of RAM and an SSD. Die OS was a *Debian Trixie* with *Python 3.11.6* and *pytorch-2.1.0*.

The objective of the *mAInZaun* project is to run the models in the field. Exemplarily, we selected a **RaspberryPi** to run the inference on such a device.

[1]`https://github.com/ultralytics/yolov5/`, YOLOv5 V7.0, master branch. SHA1: `454dae1301abb3fbf4fd1f54d5dc706cc69f8e7e` from the 7th December 2022

Model	Epochs	Layers	Params.	Size	Total	per Epoch
v5l6	285	346	76,134,048	147 MB	8.2 h	1.7 min
v5s6	400	206	12,315,904	25 MB	10.1 h	1.5 min
v5n6	265	206	3,091,120	6.6 MB	6.6 h	1.5 min

Table 6.1: Training using YOLO models on our GPU server. We configured the training with *early stopping*: In this case, the training is stopped if no significant increase in the model can be achieved. This results in a different number of epochs for the training. The maximum was set to 400 epochs. We also give the normalized training time to one epoch as a reference.
The grey line corresponds to the model we later use in this work.

We used the *RaspberryPi Model 4 B* with 4 GB of RAM and a 32 GB SD-card from Kingston. The OS was *Debian 12.2* and *Python 3.9.17* with *torch-1.10.2* were installed.

Using these four devices, we evaluated YOLO and will discuss the results in the following section.

6.2 Perfomance Evaluation of YOLO

We evaluate three selected YOLO models with different complexities for high-resolution images, namely the large (l) one YOLOv5l6, the small (s) YOLOv5s6, and the nano (n) YOLOv5n6. We trained the models on our GPU server with automatic batch size optimization and early stopping with a maximum of 400 epochs. The latter stops the training if no further improvement of the model occurs. The results are listed in Table 6.1. The effect of the model complexity, i.e., the number of layers and parameters of the model, influences the training times. Due to the varying number of epochs, we normalized the training time to one epoch. Depending on the type of the model, the training takes 6.6 h to 10.1 h. The model size is also relevant for the later deployment: Not all model sizes can run on all devices due to limited resources.

After the training, we tested the models on our Reference dataset described in Section 5.3.1. Table 6.2 lists the results, and one can see the effect of the different devices. Obviously, the more powerful devices can run the inference faster than the constrained ones. The smallest model, the YOLOv5n6, can also run quite performant on a RaspberryPi and requires just 388 ms per image for the inference using the smallest model. The RaspberryPi shows a special behavior when running the large model: When the temperature of the CPU reaches 85 °C, throttling is activated and reduces the performance, i.e., the inference speed of the model. The RaspberryPi required approximately 3.4 s per image for the first images, i.e., when the CPU was cold. After a while, the temperature reached the critical temperature of 85 °C and the inference time increased to more than 5 s and then varied between 3.4 s to 5 s, resulting in the average of 5 s as shown in Table 6.2. This throttling kept the temperature around 85 °C and reduced the risk of damage due to overheating. As a countermeasure, one could actively cool the CPU with a fan and increase the number of images that can be processed until the throttling is activated.

This relationship between the inference time and the computing power shows

Model	RaspberryPi	PC	Server	GPU
YOLOv5l6	4961 ms	863 ms	163 ms	42 ms
YOLOv5s6	797 ms	156 ms	71 ms	24 ms
YOLOv5n6	388 ms	58 ms	53 ms	22 ms

Table 6.2: Inference times using YOLO models on different computers. The results show one advantage of YOLO: One can easily adapt to the available computing resources for training and inference by selecting the best model for the given device and application.
The grey line corresponds to the model we later use in this work.

Model	FP	TP	FN	Prec.	Recall	F_1
YOLO5l6	97	867	622	0.899	0.582	0.707
YOLO5s6	179	873	616	0.830	0.586	0.687
YOLO5n6	130	826	663	0.864	0.555	0.676

Table 6.3: Evaluation of the YOLO models on the Reference Dataset. It shows the effect of the model size: The more complex the model, the better the performance.
The grey line corresponds to the model we later use in this work.
FP = False Positive, TP = True Positive, FN = False Negative

the importance of the *ShadowWolf* as described in this work: We aim to be model-independent to train a model fitting best to the hardware constraints. One can quickly adapt to the available resources: Should the system run on an embedded computer, the nano versions (i.e. YOLOv5n6) might be the choice. More complex models can be used if a more powerful machine is available.

Not only is the inference time of interest, but also the quality of the detections. After the training, we evaluated the performance regarding the F_1-score for three different YOLOv5 models using the metrics, which were introduced in Section 3.5. Table 6.3 shows the results: Similar to Table 6.1, the model complexity has a notable effect: The largest, most complex model performed better than the nano model version.

6.3 Summary of the Evaluation of YOLO

Our evaluation shows the three main advantages offered by the YOLO framework: Firstly, it provides good object detection results with little effort for training. Secondly, it can be easily scaled to run the inference on devices with different computing capabilities. Lastly, the developers of YOLO are pretty active and release new versions regularly. Therefore, we decided to use YOLO, more precisely, the YOLOv5l6 model, for our *ShadowWolf*.

Labeling a large amount of images requires a lot of time. The following section introduces *Wolf-or-Not*, our web service to offload this work to many people using crowdsourcing.

Chapter 7

Wolf-or-Not: Online Labeling

Chapter 2 discussed in Section 2.4 the challenges in labeling images. We want to go a step further from standard labeling approaches for this work and create a tool that can be used to evaluate images automatically. The question "*What can be seen in this part of the image?*" is not answered by a low number of domain experts but by an extended number of interested online users. This crowdsourcing approach, combined with the automatization of the input and output data handling, are the key features of *Wolf-or-Not*, as discussed in this chapter. It offers flexibility to adapt easily to different usage scenarios.

In this chapter, we use *Wolf-or-Not* as a toolchain to remove false positive detections from a detection model and improve its performance.

This approach is massively extended and generalized for *ShadowWolf* as we discuss it in Chapter 8. Here, *Wolf-or-Not* is used as a general tool to collect additional user feedback on the decision of a particular model. The combination of state-of-the-art machine learning models with flexible user feedback paired with several pre- and postprocessing steps make the main difference between the plain *Wolf-or-Not* and the *ShadowWolf*.

In this chapter, we evaluate *Wolf-or-Not*'s capabilities to reduce the number of false positives as published in [94] and take the first step towards *ShadowWolf*.

7.1 Idea of *Wolf-or-Not*

As discussed in Chapter 5, we deal with realistic images of wolves. To use those to train a new detection model requires good labels. Acquiring those can be a tedious task for individual persons. Here, the idea of *Wolf-or-Not* kicks in: It splits the images into certain regions and displays those to interested users in a simple web interface. Here, the user clicks on the class he believes to see. This information is stored, and the next image is loaded. The results can be used to retrain a model and improve its performance.

The main idea of our approach is to use crowdsourcing for this labeling task: We start with a small dataset of labeled images to create a first model. We use this model to detect wolves in an unlabeled dataset. Especially in a new environment, we sometimes get many false detections, i.e., false positives. We use those detections to create sub-images that only contain the detection. Those are uploaded to our web application we call *Wolf-or-Not*. Here, users click a button denoting what is visible in the subimage. After a subimage has received enough votes, the results can be downloaded and used to create labels for the original image.

Since the app is optimized for mobile devices, every user can click through the images and easily create and confirm hundreds of labels within a few minutes. Especially with a continuously increasing dataset with different backgrounds and camera angles, we are sure that this approach will reduce the effort for model developers by offloading the labelling task to many other users.

7.2 Methodology and System Architecture

Our system is designed to be model-agnostic. We use YOLO as our model as discussed in Chapter 6 together with Python and PyTorch[1] for the implementation. We use two datasets for this work. The *base dataset* from Section 5.3.2 is used as the initial one and used to create our *base model*. The second dataset is our *test dataset* from Section 5.3.3 and is used to evaluate the performance. Both datasets are entirely different from each other, contain day and night images, and are fully manually labeled.

7.2.1 *Wolf-or-Not* Architecture

The architecture of our system is depicted in Figure 7.1. We consider 5 classes: wolf, dog, person, unsure, and nothing from the listed, whereas most images only contain wolves or nothing from the listed.

We start with our base dataset and generate our *base model* using the yolov5l6 weights. With this model, we can run the inference on a new dataset of unlabelled images and get soft decisions with a bounding box, i.e., a probability and a given class. For each detection, the corresponding part of the original image is cut and stored as a separate subimage. One image can result in multiple subimages if it contains multipe detections. Those subimages are uploaded to our *Wolf-or-Not* web app. A set of those subimages is shown in Figure 7.2. Some objects can easily be identified as wolves, while others are hard to identify.

In the web app, as shown in Figure 7.3, the user clicks on the button corresponding to the class shown in the image. After clicking one of the buttons, the next randomly selected image is shown to the user. Depending on the user's experience, voting for an image only takes a few seconds, leading to many votes. In our case, even inexperienced users created 500 votes in less than 25 minutes. Section Section 7.2.2 gives more details about this app.

We do not require a registration or authentication to allow as many users as possible to participate. Due to this, a user can click on the wrong button accidentally or on purpose, which will result in wrong labels. To overcome this problem, we take two countermeasures: First, we require several votes to accept the image. Depending on the expected users (friendly users or anybody), we assume that 2-4 votes per image are sufficient to get good results. In the evaluation phase, most users were from our department, students, or other friendly persons. Therefore, we accept images with at least two votes. The second countermeasure is the number of images in the app. During the evaluation, we

[1]`https://pytorch.org/`

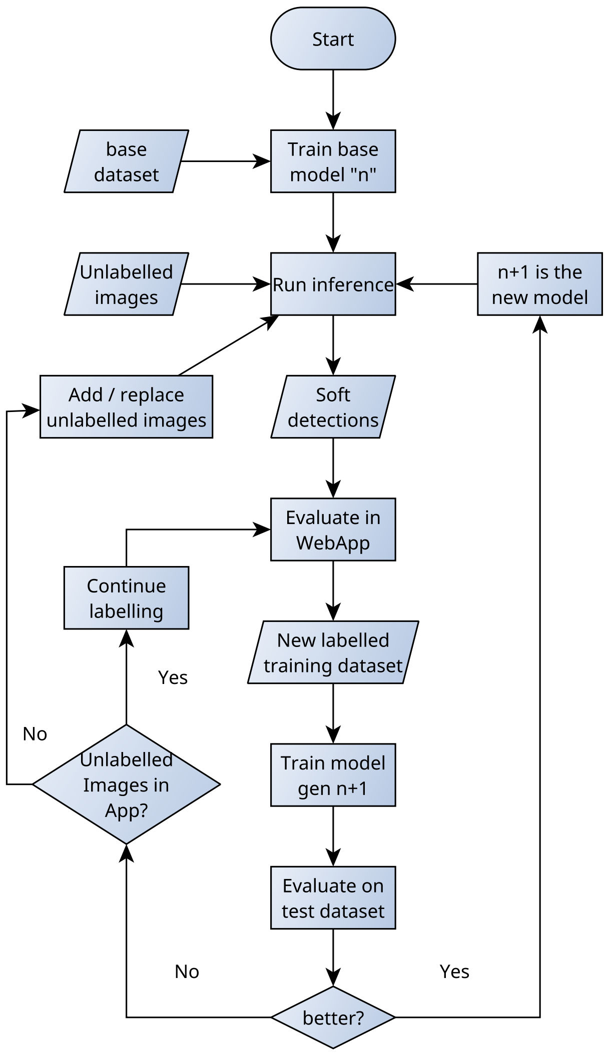


Figure 7.1: In the proposed architecture for *Wolf-or-Not*, we start with an initial model and refine it using user feedback from our Web App. This iterative approach helps to improve an existing model by adding more images.

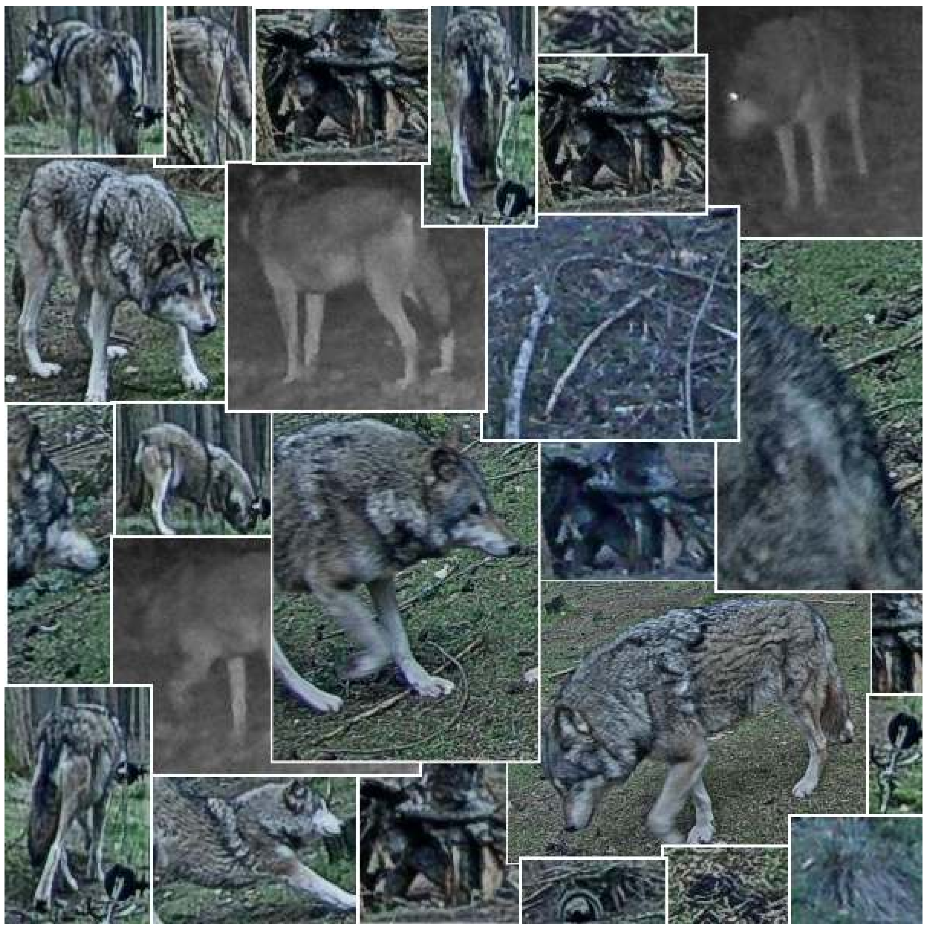


Figure 7.2: A variety of subimages detected and cut after the inference process using the base model. All have a similar probability of 0.7 to 0.9 being a wolf.

always had at least 1000 photos in the system. This reduces the risk of clicking purposely on the wrong class twice for the same image.

After a sufficient number of images got enough votes – we assume a median of 2 votes per image as discussed before – the results can be downloaded as a JSON file. From that, we create the new labeled dataset in the following steps:

1. We only consider subimages where the votes are at least 60% certain for a particular class. This filters images where the users were not confident or clicked on the wrong class.
2. We neglect subimages labeled as *unsure* or *nothing from the listed*. Those are usually not relevant to our wolf class.
3. We group the subimages according to the original image. The idea is to handle only images, where all subimages are available. We drop incomplete images.
4. Lastly, we create the labels for the original – formerly unlabelled – images according to the votes from the subimages.

The outcome is a set of labelled images that we add to our *base dataset*. The result is a new training dataset n+1. This, we use to generate the model n+1 as shown in Figure 7.1. We evaluate this model against the *test dataset* regarding the performance, namely the F_1-score. Details about the metrics are given in Section 3.5. If the performance has increased, i.e., the F_1-score is closer to 1, the loop is continued by replacing the *base model* by the new model n+1.

In case of a reduced performance, we have two options:

1. If unlabeled images are available in the app, we continue on the current images and wait for more votes. We get a new dataset from the app and continue from there.
2. If no more images are available in the app, we add more unlabeled images, run the inference with the previous model version, and continue from this step.

This procedure improves the detection model.

7.2.2 Our Web App

With our *Wolf-or-Not*, we want to be accessible to as many users as possible. Therefore, the user interface is kept simple and intuitive. Figure 7.3 shows the mobile view. The user sees an image and clicks on the corresponding class. In the example image, those classes are wolf, dog, or person. The user can also select if he is unsure or if nothing from the listed classes is shown. The interface also allows to add additional properties: Are multiple of the shown classes visible, is the object hard to see, or are various instances of one class shown? After the user clicks one of the buttons, the result is stored as a vote, and the next image is loaded. This simple concept of click and load next allows quick labeling on the go on the mobile device. In the current version, no authentication is required to keep the burden of participation as low as possible.

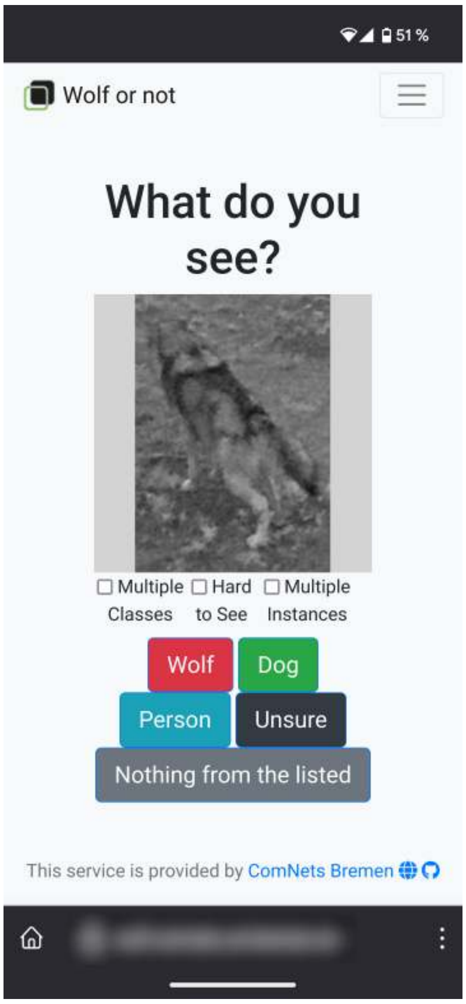


Figure 7.3: *Wolf-or-Not* shows an image of an object. The user decides what he sees: wolf, dog, person, etc. He can also add attributes like *hard to see.*

To access the current status of the labeling process, all users have access to the status page. Here, some simple statistics are displayed. We show the number of votes, the number of images currently assessed, and the active datasets. We also provide statistics like the mean, median, min, and max values of votes per image. A significant value to decide if it makes sense to download the current data is the percentage of subimages with a certain number of polls.

After collecting sufficient votes, the user can download the results as a JSON file. For this evaluation, we only consider results with at least two and, optimally, three votes to reduce the risk of wrongly clicked images. The results contain the name of the image, the number of votes, the individual vote results (how often did one click on which class), and the overall class results. The latter tells the probability of the given image being to a specific class. This data is later used to create or evaluate the new labels.

7.3 Evaluation

This section will evaluate the performance of *Wolf-or-Not*. We use the metrics from the earlier Section 3.5. The use case for the evaluation is described in Section 7.3.1. The results are shown in Section 7.3.2 and discussed in Section 7.3.3.

7.3.1 Use Case

For this evaluation, we consider two rounds using our *Wolf-or-Not*. We start with 2069 unlabelled images and our *base model*, which creates 4000 subimages, i.e., on average, two subimages per image. We uploaded those and waited for the users' votes. We downloaded the results after we got 7395 votes (median: 2.0 votes per subimage). The *base model* created many false positives, leading to several images with multiple subimages as depicted in Figure 7.4. Those wrong labels were removed by the users using our app. As the output, we generated a relatively small number of 96 completely labeled images, which were added to the *base dataset*. This dataset now contains 1703 images and is used to generate the model *n+1*. We evaluate the performance regarding the F_1-score as described in Section 3.5. The performance has improved, and we continue with a second round.

We used another set of 2000 unlabelled images for the second round and split it into 1544 subimages using the previously generated model *n+1*. Compared to the first run with our base model, the number of false positives is reduced, leading to a lower number of subimages. Therefore, the number of complete images increased as more images generated only one subimage in contrast to several for the base model. We add those to the *n+1* dataset and get our *n+2* training dataset with 2259 images used to generate the model *n+2*. We give the exact results in Table 7.1.

7.3.2 Results

After we have generated our three models, the *base model* and the two new ones *n+1* and *n+2*, we run them on our test dataset from Section 5.3.3 and calculate the metrics as described in Section 3.5. Here, we focus on the F_1-score but also give the precision, recall, and absolute number of false positives, true

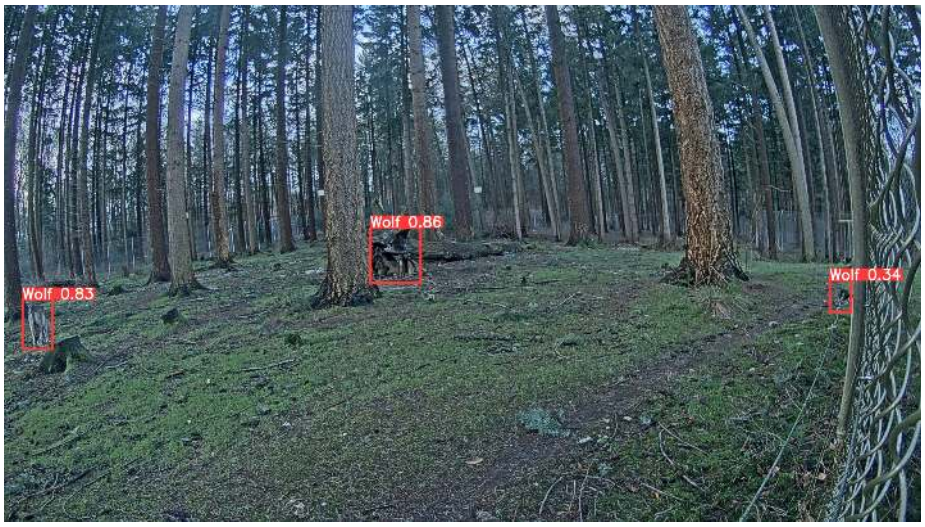


Figure 7.4: Wrong detections of wolves with the base model: The first model detected a tree and an isolator as a wolf.

	n+1	**n+2**
Input images	2069	2000
Subimages in App	4000	1544
Votes in App	7395	2638
Median Votes per subimage	2.0	2.0
Mean Votes per subimage	1.85	1.71
Subimages with 2+ Votes	2022	781
Subimages with 3+ Votes	1120	385
Resulting complete labels	96	556
Total number of images for training	1703	2259
Image collection	23-01-23	22-03-29
Weather	cloudy	sunny

Table 7.1: Input images used for the training of the two new model generations

	base model	**n+1**	**n+2**
Precision	0.779	0.935	0.95
Recall	0.843	0.853	0.845
$\mathbf{F_1}$	**0.810**	**0.892**	**0.896**
False positives (FP)	481	120	84
True positives (TP)	1700	1720	1704
False negatives (FN)	317	297	313

Table 7.2: Comparison of the given metrics for different models evaluated on our test dataset consisting. The test dataset consists of 2017 instances of wolves in 1526 images. The main metric is the F_1-score which improved for all runs.

positives, and false negatives in Table 7.2. The results show that false positives are significantly reduced, especially when comparing the *base model* to the *n+1* model. We evaluated this result by manually checking the detections from the base model: A notable number of false positives can be seen. Especially a tree and an isolator were often incorrectly detected as a wolf, as shown in Figure 7.4. The improvement of the F_1-score between *n+1* and *n+2* is comparably low.

We are further interested in the statistics: Where did we adapt the dataset? For that, Figure 7.5 and Figure 7.6 show the normalized frequencies of the scores for the creation of model n+1 and n+2, respectively. Here, we compare the detections of the previous model (the one used to create the subimages) with the resulting user votes. Correct votes are marked green, and incorrect ones are marked red. Figure 7.5 shows that the *base model* created a lot of false positives with high probability: The red peak between 80% and 90%. In the next iteration step, as shown in Figure 7.6, this peak disappeared: On the test dataset, all detections with a score higher than 80% are correct.

7.3.3 Discussion

The first iteration step showed that using our approach significantly increased the performance. The F_1-score increased from 0.810 to 0.892 and to 0.896 for the second iteration. The absolute number of false positives is reduced from 481 (*base model*) to 120 (*n+1*) to 84 (*n+2*). The true positives and false negatives stay almost constant. Figure 7.5 and 7.6 support this statement: Mainly the strong false positives are removed using our approach.

Handling the images is mainly done using scripts, and the evaluation is distributed over several volunteers. Therefore, using this crowdsourcing approach, *Wolf-or-Not* offers a good option to evaluate continuously arriving camera images.

7.4 Summary

Wolf-or-Not is a flexible web service for various applications. In the chapter, we evaluated it to reduce the number of false positive detections in a continuously growing training dataset. Finding animals not detected by an object detection model is more difficult, and the objective of *ShadowWolf*, as described in Chapter 8. There, we use *Wolf-or-Not* in a different way: It supports object detection to find hidden wolves. This is possible due to the flexibility of *Wolf-or-Not*. We

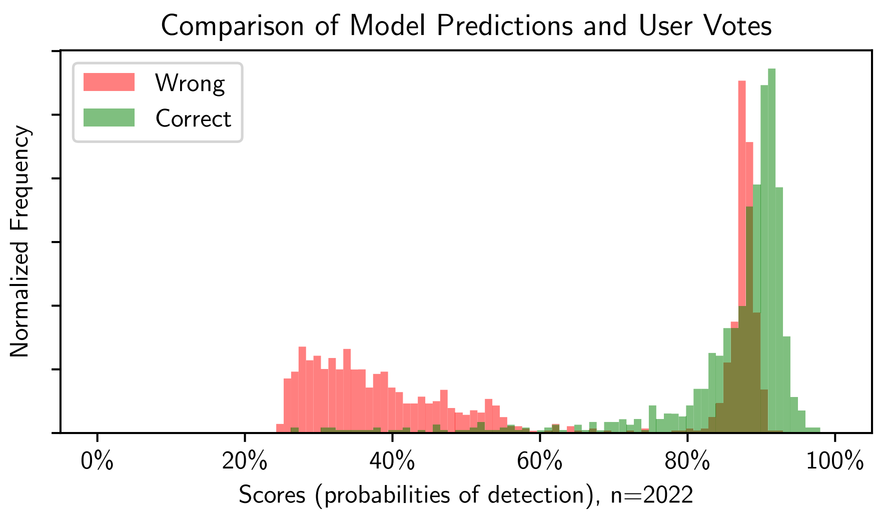


Figure 7.5: Normalized frequencies of the probabilities / scores for the model n+1. Correct means that the detected class is the same as the user's. If a model detectes a class different from the one the user voted for, the detection is considered wrong.

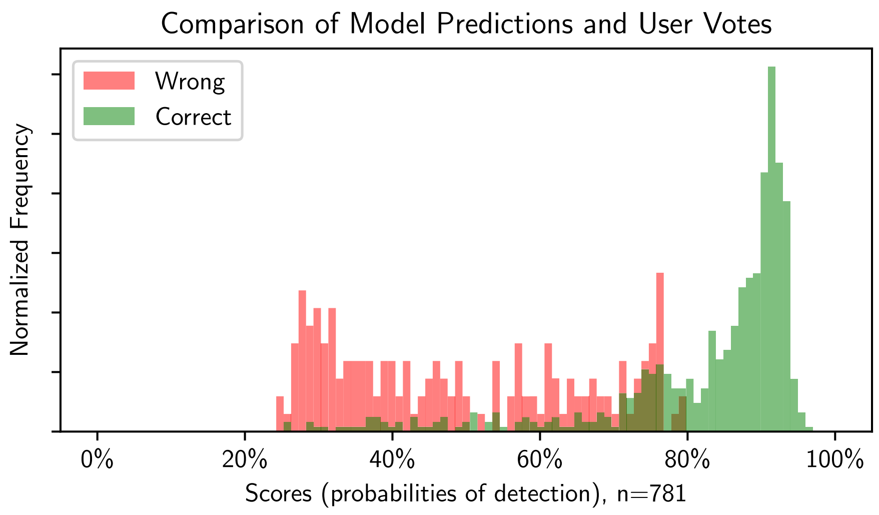


Figure 7.6: Normalized frequencies of the probabilities / scores for the model n+2. In comparison to Figure 7.5, the number of wrongly detected images with high probabilities is significantly reduced: All detections with a probability higher than 80% are correct.

continuously improve *Wolf-or-Not* and have made it publicly available under the *GNU General Public License v3.0* in our repository on GitHub[2]. Furthermore, a test server is available online[3].

[2]`https://github.com/ComNets-Bremen/Wolf-or-Not/`
[3]`https://wolf.comnets.uni-bremen.de/`

Chapter 8

ShadowWolf

With *Wolf-or-Not*, as presented in Chapter 7, we offered a flexible tool usable to label and evaluate images. In the use case, it showed good performance in reducing the number of false positives. For *mAInZaun*, we are also interested in the hidden and only partly visible wolves. Those are hard to detect by most object detection algorithms and are often neglected in the research. Our *ShadowWolf* focuses on those false negatives and combines object detection with human validation. The usage of the output is manifold and can be used, for example, to train arbitrary models or support people screening wildlife images. We published this work in [91].

8.1 Motivation for *ShadowWolf*

Detecting animals in camera trap images is not an easy task. As discussed in Chapter 2, most work focuses on the obvious animals in the direct foreground or clearly visible in the background. While this might be acceptable for general animal monitoring, the requirements for our *mAInZaun* project are higher. We aim to detect also the barely visible predators. The challenges for this are manifold and were discussed in Chapter 5. At the same time, we would like to run the inference directly in the wild. Here, the available hardware for the detection might vary, and we need the option to adapt to various platforms with different computing capabilities. Also, the cameras' field of view can change and requires additional training data. To quickly adapt to this, a continuous flow of new labels and images needs to be served to optimize the performance of the system.

Therefore, we decided not to focus on developing a new model but create a complete framework for our project: *ShadowWolf*. It allows to analyse a stream of continuous images or even videos from camera traps, detects and classifies objects with a combination of AI and human support, and creates the training data for arbitrary models. The complete workflow can run automatically with only minor human interaction. We also introduce a flow to evaluate and improve new models automatically.

For the latter, we use an iterative approach as depicted in Figure 8.1. We assume that we regularly receive new images from different camera traps. We label them using our *ShadowWolf*, create a new detection model, and evaluate

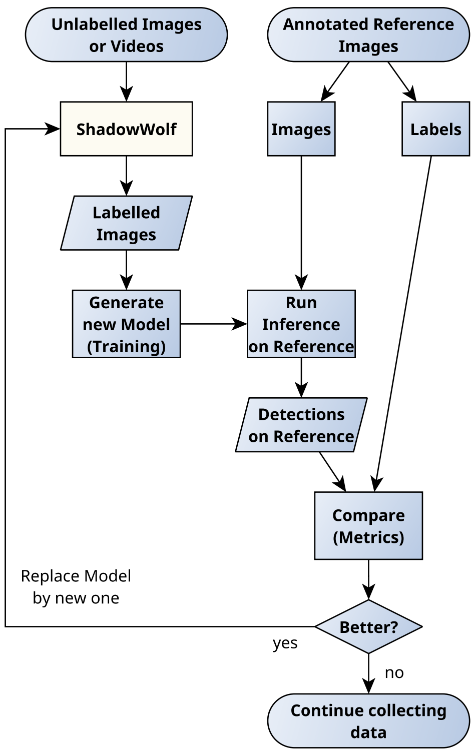


Figure 8.1: Continuous learning and automatic adaptation to new environments and animals are beneficial, especially for camera trap images. The depicted iterative approach labels the images using our *ShadowWolf* as introduced in this work. We use the generated labels to train a new model generation. Afterward, we evaluate the performance of the model on our reference dataset. If the performance is improved, we have an improved model. Otherwise, we continue with the data collection.

its perfromance on our reference dataset as introduced in Section 5.3.1. If the performance regarding our metrics (c.f. Section 3.5) is improved, we have a new, improved model we can use for the *ShadowWolf* and other applications. All steps are automated (besides the crowd-sourcing evaluation of images).

8.2 Methodology and System Architecture

We discussed in the state-of-the-art in Chapter 2 that most projects focus on optimizing the models on the camera images. We propose a different approach: Our toolchain named *ShadowWolf* focuses on the training and the evaluation of arbitrary models. The idea is to combine state-of-the-art technologies for image (pre)processing and data handling into one flow combined with crowd-assisted validation and enhancement. *ShadowWolf* also focuses on the repeatability and reproducibility of the runs by packing the results, including the configuration and the excessive log files, into one output directory per run. The script only takes some minor parameters, forcing the user to make all changes in the configuration file, leading to better documentation.

In the end, this also helps to use *ShadowWolf* in a continuous training environment: The training dataset can be continuously improved by labeling camera trap images with minor administrative user interaction as depicted in Figure 8.1.

The idea of *ShadowWolf* is to handle three types of input data that come from automatic capturing systems like camera traps:

1. A single image
2. A series of images over a fixed period (time-correlated)
3. A (short) video sequence (which can be converted into a series of images)

The output is a labeled training dataset usable for training a new model. Due to the variety of models in image recognition and the speedy development, we try to be as model-independent as possible.

ShadowWolf consists of several steps that can be connected in different ways. Figure 8.2 on page 82 depicts the main structure and possible options. *ShadowWolf* is the part inside the dotted box. Outside, the performance evaluation steps used in Section 8.3 are shown.

In the preprocessing block, we analyze the image, create batches of time-correlated images, and optimize the quality optionally. Afterward, we try to find changing parts by removing the background and creating image segments with potentially moving objects. The object detection follows those steps. Here, an object detection model is used to detect animals, as discussed in Chapter 3. In this work, we use YOLO as evaluated in Chapter 6. This is the central part of *ShadowWolf* as an optimized version of a model can replace the existing one. Afterward, we perform further steps to remove duplicate detections, evaluate the results, and map everything back to the original images. Finally, we calculate the detections and generate the output depending on the requirements. In our case, we create a training dataset.

The main idea is to allow a flexible connection and usage of the separate steps: Each step gets all the information from the previous ones and benefits from the knowledge gained. The drawback is the increased complexity of handling missing parameters. The steps are configured in a central configuration file. This allows us to re-run the steps and document the results quickly. After each step, the current status, i.e., the variable files and created images, are stored. In case of an error, *ShadowWolf* can be re-started from the given point. This is beneficial, especially when using long-running operations like complex image processing or analysis on large datasets.

We created this work with two main applications in mind. The first is the (re)training of models. Camera traps often have static positions and, thus, a fixed field of view. This can lead to domain-specific models: In doubt, the model detects everything that is not background as an object. Easily adding new camera trap images to the training dataset can reduce this problem. Additionally, new species might become interesting. Also here, an adapted training dataset is required. This application is directly related to our project. Second, camera trap images have to be reviewed: What species triggered the camera? Our *ShadowWolf* processes those images, and only the critical, i.e., ambiguous photos, are shown to the human experts. Both applications take advantage of

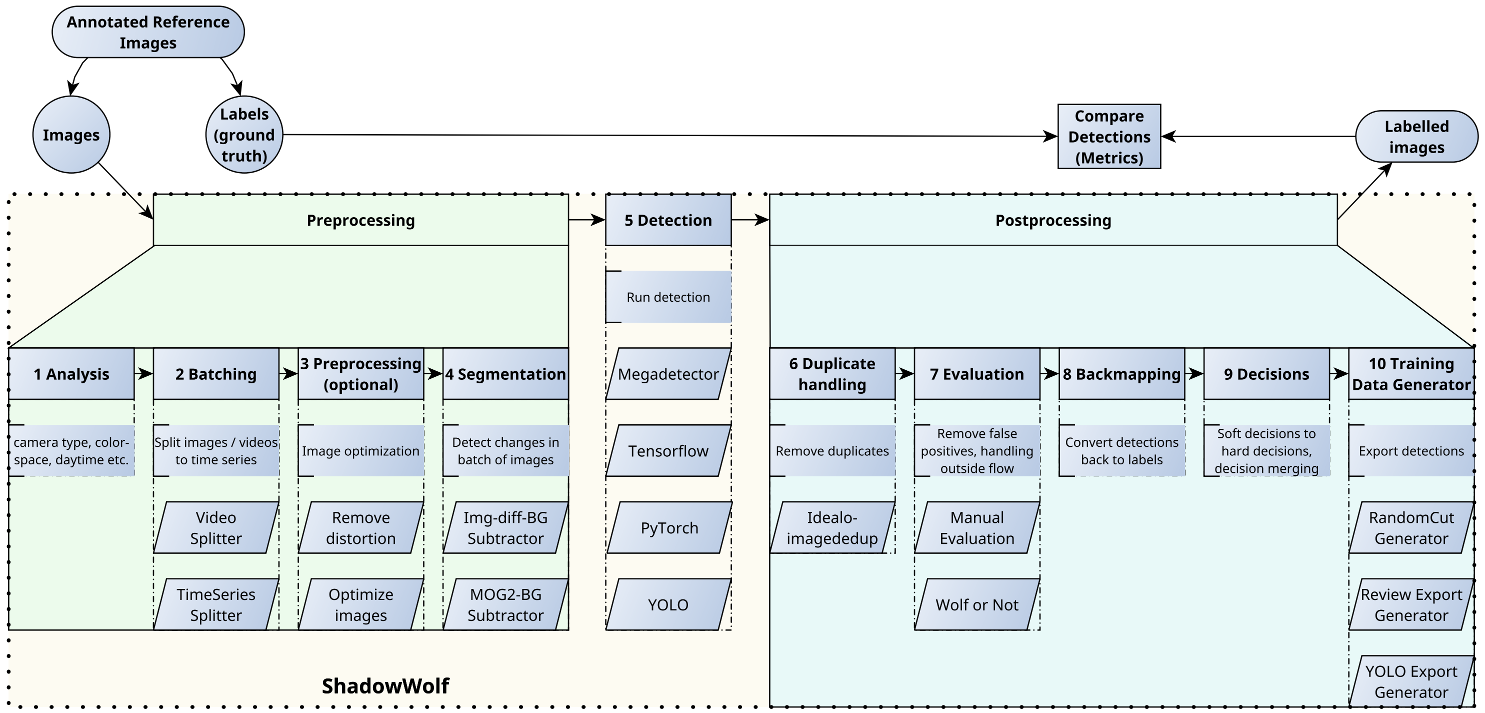


Figure 8.2: The *ShadowWolf* framework and the evaluation steps used in this chapter: The objective is to convert an input video, image, or series of images into good-labeled images. *ShadowWolf* itself is the part in the dotted box. It has three main parts: Preprocessing, detection, and postprocessing. For each part, several options for different submodules and configurations exist. The detection (step 5) contains a trained animal detection model. In the iterative process, as depicted in Figure 8.1, it will be replaced by the improved version. We evaluate the performance using our reference dataset and the metrics. The boxes outside the dotted area indicate this evaluation.

such an automated framework. The collected images are uploaded to a server where *ShadowWolf* performs the processing. In the end, the results can be collected with only minor human interaction. The following subsection explains the individual steps in detail.

8.2.1 Analysis

First, we extract meta information from the original images: the EXIF, IPTC data, and image size. Those can contain information about the camera, time, copyright, keywords, image descriptions, etc., and can get lost during image processing and storage. We also store if the images are greyscale or RGB. We do not save the file system attributes (created, modified, accessed) as those are easily accidentally changed and unreliable. The most important attributes are:

- The **colorspace** of the image. Especially if the source images contain day and night images, this information can be important if color-dependent functions or algorithms are used in one of the following steps.
- We also store the **camera type**, **lens type**, and **serial number** to be able to remove camera-specific distortions or handle other camera-specific operations.
- The EXIF **DateTime-Fields** are used to group the input images by the time, i.e., create a series of images. Our experience shows that those times are more reliable than the file system attributes.
- For the IPTC tags, we store the keywords to filter the images during the analysis phase.
- The original image **resolution**.

8.2.2 Batching

Camera Traps often give a series of images: A motion detector activates the camera, and it takes a certain number of images. Those are usually collected in a fixed interval, in our case, two images per second. Knowing about those time-correlated images can be beneficial for the next steps. For example, the light, weather, and general environmental conditions will be similar within such a batch of images. We use this information for the segmentation in one of the following steps.

We have tested two different types of splitters:

- **Time Series Splitter** analyzes the EXIF time between two images for the given series of input images. We assume that a camera stores several images when motion is detected. In most of our cases, 10 to 70 images show an object moving through the recorded area. The splitter detects those series and converts them to separate batches of images.
- **Video Splitter** converts a short video sequence into a batch of images. This batch can be handled the same way as the output of the *Time Series Splitter*.

Our collected data data consists of individual images, so we focus on the *Time Series Splitter*. For the batching, we use the EXIF-tag *DateTime*, which is, in the case of our camera, identical to *DateTimeOriginal* and *DateTimeDigitized*. The input images are sorted by date and time. Is the time between two consecutive images longer than a certain period defined in the config file, *ShadowWolf* splits the series to a new batch. For our experiments, a value of 5 seconds was a reasonable tradeoff.

8.2.3 Preprocessing

In the preprocessing, *ShadowWolf* optimizes the images for the following steps. The objective is to normalize the images from different sources in a way the next steps get – based on the meta-data – as similar images as possible and will focus on the content, i.e., objects. For example, preprocessing can be used to remove some parts from the camera trap images: Some manufacturers add a logo and the date and time to the pictures. Those additional images can be problematic in the training phase of a new model, especially when using images from heterogeneous sources: The model might focus on the camera brand logo instead of the real content.

Another option for this step is to remove the spherical distortions as discussed in Section 5.2.7.

In the current version, we have implemented a handler that removes the distortion using OpenCV[1] based on the camera serial number. Using this approach, we can use different parameters based on the camera metainformation collected in step 8.2.1.

If required, one can easily add more optimizations like normalizing the colorspace or adapting the brightness for some images.

8.2.4 Segmentation

One challenge in camera trap images is that the object of interest is not always the main object in the image: Wolves might hide behind bushes and trees, and a deer might be running in the background.

Information losses occur due to the downscaling, especially when using high-resolution images combined with a lower-resolution model. This step aims to find changing regions in the image and cut them into subimages. The model in the next step can focus on this particular area. We are aware that some models automatically perform this step using, for example, a Region Proposal Network (RPN). To be as model-independent as possible, we implemented this step separately.

Another advantage of this step is that we might catch an object of interest that the model does not detect, i.e., a false negative. This will be evaluated in the evaluation step described in subsection 8.2.7. Our App *Wolf-or-Not* shows this subimage to the user. The user might recognize an object not detected by the other algorithms. Therefore, this will help to detect hard-to-see objects and improve the overall performance.

[1] `https://learnopencv.com/understanding-lens-distortion/` accessed: 2023-08-19

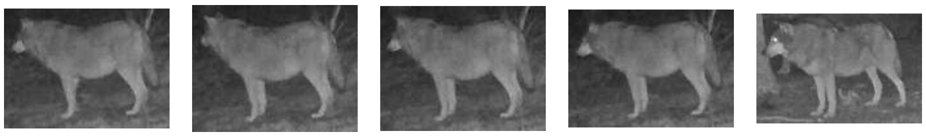


Figure 8.3: Five images of wolves that were detected as similar by the image deduplication. They are reduced to one image, and the following steps' workload is reduced. Later, the detections are used for all similar images to prevent information loss.

We currently offer two options for the segmentation. One is the **MOG2-BG subtractor** (Gaussian Mixture-based Background/Foreground Segmentation Algorithm [95, 96]). This well-known algorithm creates boxes around moving objects in a batch of images by removing the background. It benefits from the batching described in subsection 8.2.2.

Another option is implemented in the **Img-diff-BG Subtractor**. This module creates an average image over the batch of images, compares this one to each image, and marks the differences as boxes. This simple approach can detect movements quite well and is an adapted and extended version of the method described on pyimagesearch.com[2].

Megadetector, as discussed in Section 3.3.4, is also an alternative that can be used here. It requires quite a lot of computing resources. Therefore, we did not implement it.

8.2.5 Detection

The detection is the heart of the framework. In Section 3.3 in Chaper 3, we evaluated several frameworks in the area of machine vision and decided to use YOLO in this step. Chapter 6 gives further details and evaluates YOLO.

The output of this block is a soft decision, i.e., a bounding box with detected classes and their probabilities. We focus on the class with the highest probability for the following steps and suppress the others, i.e., NMS.

The model used here can come from a previous run of *ShadowWolf*: The output, i.e., the labeled images, is used to train a new model that can be used here. This iterative approach is discussed in Figure 8.1 in Section 8.1.

8.2.6 Duplicate Handling

During our extensive evaluation, we sometimes got a lot of similar images. i.e., in most cases, the same or similar objects were detected. Figure 8.3 shows five example images. In another case, 62 similar-looking wolves were detected. To reduce the workload for the next steps and only focus on images resulting in a gain during the training, we grouped those images and handled them as one. In the *Backmapping*, as described in subsection 8.2.8, the detections are expanded to similar images to prevent information loss. The **imagededup-Handler** is based on [97] by idealo and is quite powerful in detecting similar images. This

[2]`https://pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/`, accessed: 2023-08-19

module groups similar images and treats them as one image for the following steps.

8.2.7 Evaluation

This step is one of the most important ones in our work, as we can use user interaction to strengthen or weaken the detections. At this point, we allow a review of the detections by humans and increase the quality of the output training data. We have two options:

- ***Wolf-or-Not*** is our web service introduced in Chapter 7. Here, we use it to show the given detections to interested users. They review the detections and select what they see. This information is used to remove false positives or strengthen the outcome of the previous detections.
- **Manual Evaluation** is another option: One can manually validate the detections and remove the wrong ones. This option is very time-consuming and is only considered for debugging reasons.

8.2.8 Backmapping

In Subsection 8.2.4, we (optionally) split the image into subimages. During the detection, as described in Subsection 8.2.5, we further narrowed the images to smaller ones. This has to be undone: We want to have the highest resolution of our findings from the previous steps in the original image. We call this step *backmapping*. The output of this module contains information about all detections for each image converted back to the original image. Those include the origin of the detection (user information, detector, etc.), the probability for a particular class, and the bounding box for the original image. Figure 8.4 shows an image created for debugging reasons. For the left wolf, two boxes exist: The outer one is from the segmentation. In the evaluation step, the users of *Wolf-or-Not* agreed with 100%, that this image contains an object of class wolf. The inner box comes from the detection. In this case, YOLO was 90.8% sure to detect an object of class wolf. Additionally, the users in *Wolf-or-Not* agreed to see a wolf by 100%. YOLO did not detect the animals on the right side. Users in *Wolf-or-Not* agreed with 67% that they see a wolf. This is a good example of the power of the combined approach.

8.2.9 Decisions

After increasing the data for each image in the previous steps, we are now condensing the information. We combine the information and decisions from the previous steps to get one main decision. The information comes mainly from the module *Detection* (step 8.2.5) and *Evaluation* (step 8.2.7). Information from the batches, i.e., time information, can also be used.

First, we have to combine the boxes from the previous step as they might overlap, as seen in Figure 8.4. We get all boxes from an image, and check if they overlap. This is the case if the IoU is over a certain threshold. Section 3.5 gives more details about the metrics. In this case, we combine the boxes and also the detections. As a result, we get an array containing each box and all corresponding detections.

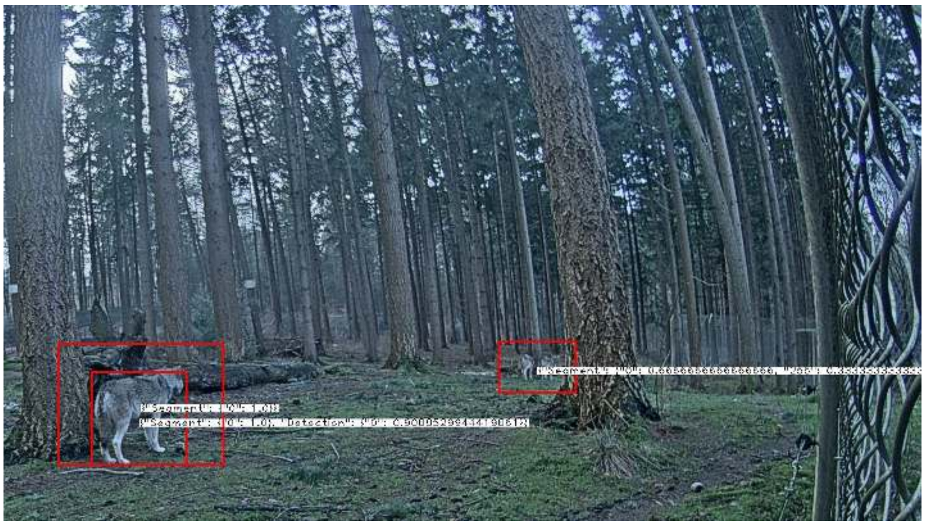


Figure 8.4: Example output from the backmapping module. For the left wolf, there are two segments: One comes from the segmentation and one from the detection. Both have probabilities from *Wolf-or-Not*. The detector did not recognize the right wolf. Here, only the user-generated probabilities from *Wolf-or-Not* are available.

Symbol	Meaning
c	The specific class
i	The detector
$d_{c,i}$	The detection from the given detector for a specific class
w_i	The weight of the given detector
$\bar{d}_c$	The result of the object being in a specific class

Table 8.1: The notations for the weighted probabilities.

We use the weighted arithmetic mean to calculate the probabilities of an object being in one class. The used notations are listed in Table 8.1. Classes are, for example, *Wolf*, *Human* etc. A detector is one step returning a probability for an object being in a specific class. As the reliability of the different detectors might vary, the individual detectors can be weighted: One might trust the output from *Wolf-or-Not* more than the one from the YOLO detector. We do this as follows:

First, we assume that all detectors return the detections in the closed interval $0 - 1$ (8.1).

$$d_{c,i} \in \mathbb{R} \mid 0 \leq d_{c,i} \leq 1, \forall c, \forall i \tag{8.1}$$

The weights for all detectors have to sum up to one to form a valid probability function (8.2)

$$\sum_{\forall i} w_i = 1 \tag{8.2}$$

Under the assumptions (8.1) and (8.2), the weighted arithmetic mean can be described as in (8.3).

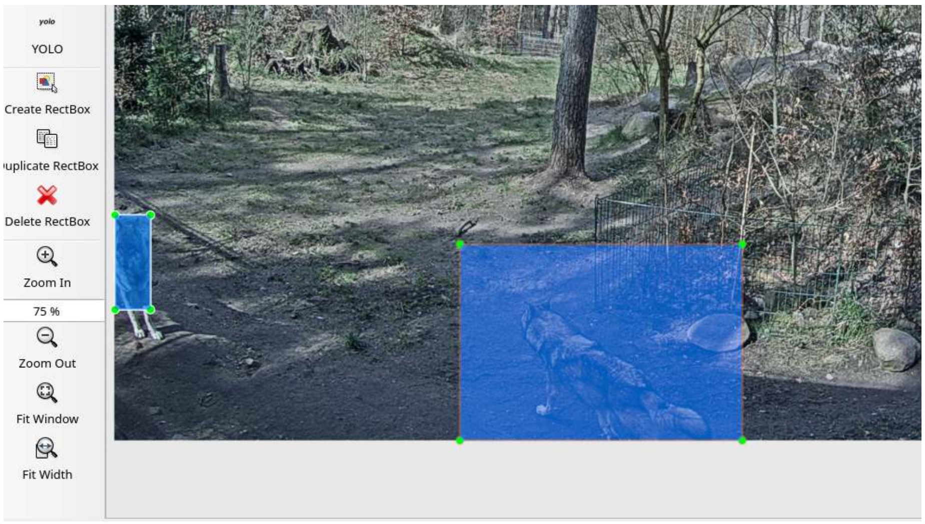


Figure 8.5: The automatically generated training dataset is checked using, for example, LabelImg. The boxes can be fine-tuned to increase the quality of the generated model. The left box could be slightly larger, whereas the right one is too big.

$$\bar{d}_c = \sum_{\forall i} w_i \cdot d_{c,i} \tag{8.3}$$

It returns the combined probability that the result belongs to a specific class. If all detectors' weights are equal, this results in the arithmetic mean. If several detections from one origin are available, they are used twice and normalized again. This happens in the case of combined boxes from one source.

We use NMS, i.e., select the class with the highest combined probability and store the class and the probability for the processing in the next step. The training data generator will convert those combined probabilities into a usable output format for further processing.

8.2.10 Training Data Generator

In this last step, we stop the processing and offer several options to export and validate the results. The detections from the splitting might not result in well-aligned boxes: They tend to be a little bit larger than necessary, leading to a decreased quality of the training data and, thus, for the generated model. The bounding boxes can be reviewed and adapted with a tool like LabelImg (c.f. Section 3.4) as depicted in Figure 8.5.

We got full-scale, full-resolution images from the previous step with the corresponding labels and decisions. These images are now converted depending on the requirements. Right now, we offer three possible options:

1. We export the full images and labels with the hard decisions in the YOLO data format. This can directly be used to train a new YOLO model.

2. We export the full images and labels with the soft decisions. This can be used to evaluate the results.
3. We create no output and use the previous step's decisions directly. This is mainly done for debugging.

Other outputs can be implemented easily. For example, if the output of *ShadowWolf* is used to train a classifier as we did in Section 3.3, one can cut the detections to squared images and export them in the required directory structure. This can be used to train, for example, a MobileNet model in TensorFlow or PyTorch.

The following subsection describes additional functionalities tha can be implemented using *ShadowWolf*. Furthermore, we explain some more technical details of our implementation.

8.2.11 Additional Functionality

As mentioned before, the objective of *ShadowWolf* is to 1) run with only little user interaction, ideally automatically (for example, as a regular batch job) and 2) focus on repeatability and reproducibility. Those two objectives make it ideal for an iterative process for continuous labeling and detecting objects in camera trap images.

For the use case, we assume that new images arrive regularly. Those are processed by *ShadowWolf*, resulting in a new set of annotated training images. This is used to generate a new model, which can also be used by *ShadowWolf*.

Figure 8.1 shows the idea: We use the output of *ShadowWolf* to train a new model. With this model, we run the inference on images from a separate, annotated reference dataset. We compare the detections to the labels from the reference dataset regarding the metrics. If the results get better, we have an improved model. Otherwise, we will continue collecting data. The model quality should ideally improve with every successful iteration, as we demonstrate in Section 8.3.

8.2.12 Implementation Details

This section briefly describes the technical background of *ShadowWolf*. For more detailed documentation and the source code, we refer to our git repository[3].

ShadowWolf is written in Python. At the time of writing, we used Python 3.11 with several libraries and modules. The `requirements.txt` in the git repository lists the exact versions.

The main configuration is done using a JSON file. It consists of two main parts: The general configuration with the input directory and file type. The second is an ordered list of modules and the corresponding module-level configurations. Each module corresponds to a step in *ShadowWolf*. Appendix C presents the configurations used in this work. The intermediate results and connections between them are stored in an SQLite database. Each module uses a separate directory to store binary data, like preprocessed images. The documentation and source code in our git repository give further details.

[3] `https://github.com/ComNets-Bremen/ShadowWolf`

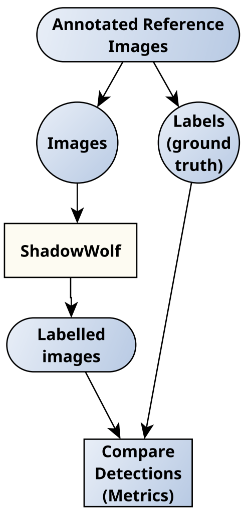


Figure 8.6: The summarised evaluation flow from Figure 8.2. We run *ShadowWolf* and compare the results with our ground truth labels.

The following section discusses the performance of our toolchain: How do we evaluate *ShadowWolf* and compare the results?

8.3 Performance Evaluation

We use our reference dataset from our existing data for the performance evaluation. The creation of the dataset and its properties were described in detail in Chapter 5. It contains 1140 images from two different wolf parks. 952 images have at least one wolf. 806 images were taken during daytime, 334 during nighttime.

For the evaluation, we compare the generated labels from *ShadowWolf* with the ground truth labels from the annotated reference images as shown in Figure 8.2 and in the summarising Figure 8.6. Ideally, all boxes should match perfectly. In real life, we try to get as close as possible to this. The following sections describe the details of our evaluation.

We use the metrics introduced in Section 3.5 for the evaluation. In the Section 8.3.1, we describe our use case. The results are presented in Section 8.3.2, 8.3.3 and 8.3.4. Section 8.3.5 discusses the results, whereas Section 8.3.6 lists future steps and challenges regarding *ShadowWolf*.

8.3.1 Use Case

We apply the toolchain to our reference dataset introduced in Section 5.3.1. The manually created labels are only used for the validation and are not used for the toolchain itself. The dataset includes day- and nighttime images. The main class is "wolf" which has to be detected. For the individual modules, we evaluate the following ones (c.f. Figure 8.2 and Section 8.2):

Analysis: We extract all available information, including the IPTC tags and the daytime.

Batching: We have time-correlated images, so we use the *Time Series Splitter*.

Preprocessing: The model used later on was trained using the unchanged images. Therefore, we do not perform preprocessing on the input images.

Segmentation: To evaluate the effect of the image segmentation, we ran one experiment with and another without the *MOG2-BG-Subtractor*.

Detection: We use the second generation (n+2) YOLO-based model from Chapter 7.

Duplicate Handling: To reduce the workload in the next step (evaluation), we use the deduplication for the complete *ShadowWolf* run.

Evaluation: For some experiments, we use our *Wolf-or-Not* tool to further evaluate the results from the detection.

Backmapping: We map the detections back to the original images.

Decision: We merge the decisions from all possible inputs (*Detection*, *Evaluation*) by applying the weighted arithmetic mean. We give a slightly higher weight to the Evaluation part than to the detection (0.6 and 0.4) as we trust the human evaluators more.

Training Data Generator: We use the YOLO data generator to create a training dataset. Here, we need hard decisions and set the threshold to 0.5. Objects rated with higher probabilities are mapped to the corresponding class. Furthermore, we also create soft decision output in images, offering a deeper insight.

In total, we run three experiments

- Evaluation of YOLO: Baseline experiment. We configured *ShadowWolf* to only run the YOLO detection.
- Evaluation of Segmentation: We create subimages and let YOLO also run on those.
- Evaluation of the complete *ShadowWolf*, including the *Wolf-or-Not* webservice.

All experiments result in the same output format, which simplifies the evaluation. The next section compares the results regarding the metrics discussed in Section 3.5.

8.3.2 General Results

First, we evaluate the complete dataset regarding the three experiments. The results are shown in Table 8.2. Here, we analyze the complete set of 1140 images, i.e., the day and night images. One can see that the performance of pure YOLO is quite similar to YOLO with the segmentation. The resulting

	FP	**TP**	**FN**	**Prec.**	**Recall**	**F1**
YOLO	81	846	643	0.904	0.568	0.700
YOLO & Segm.	90	846	643	0.904	0.568	0.698
Full flow, $\alpha = 0.5$	204	808	681	0.798	0.543	0.646
Full flow, $\alpha = 0.25$	47	965	524	0.953	0.648	0.772
Full flow, $\alpha = 0.1$	26	986	503	0.974	0.662	0.788

Table 8.2: The results from the three runs over the complete dataset containing 1140 images. α is the decision value for the IoU. If this value is larger than the given α, the detection is correct.

F_1-score is the same. Only minor differences occur for the false positives, true positives, and false negatives. This was expected as YOLO internally performs a similar technique, which seems to perform quite well.

For the complete *ShadowWolf*, we already know from the review in Section 8.2.10 and Figure 8.5 that the generated boxes might not fit perfectly to the objects, resulting in too many false positives. To take this into account, we evaluate the results with different values for α when calculating the IoU: Besides the default value of $\alpha = 0.5$, we also use $\alpha = 0.25$ and $\alpha = 0.1$. Table 8.2 shows the expected results: Lowering α significantly reduces the number of false positives.

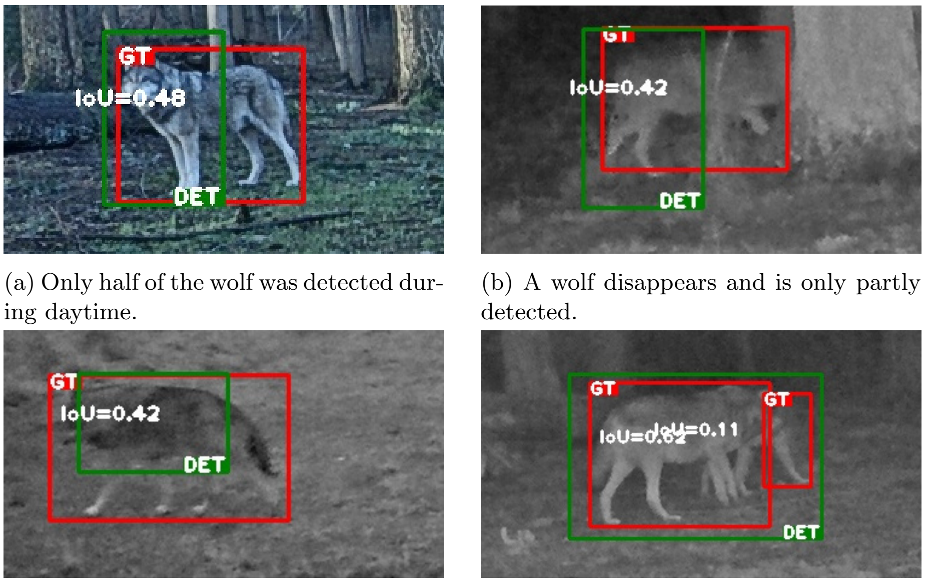
We further visualized the effect of α by plotting the detections with the ground truth and the IoU in one figure, as displayed in Figure 8.7. The Figures 8.7a and 8.7c show only partly detected wolves. This happens if the wolf is barely moving and not detected by the YOLO model. In this case, the motion detector gets only part of the animal, resulting in boxes that are too small and leave out essential details. A similar happened in Figure 8.7b: The person creating the bounding box can correctly mark the wolf disappearing behind the tree. For our algorithm, only the clearly visible part is detected correctly. The third case is shown in Figure 8.7d: Our image dataset shows that the wolves often move pretty close to each other, resulting in this kind of image. *Shadow-Wolf* detected one wolf. In the ground truth, the second wolf was marked as another instance. In these cases, we mainly deal with boxes created using the background subtractor, i.e., not the detection part. Those boxes are generally not as exact as the ones from the machine learning model, but better than no detection. Therefore, we find it valid to decrease α.

Table 8.2 shows that the false positives decrease and the true positives and the F_1-score increase according to our experience and expectations.

8.3.3 Effect of the Daytime

In Chapter 5, we discussed that the lighting conditions are essential to get high-quality detections. We used the meta data stored in the analysis step to filter the detections accordingly. Table 8.3 shows the same metrics from Table 8.2 for our 806 daytime images, whereas Table 8.4 shows them for the 334 night images.

Comparing the results from the daytime images to the complete results shows a general decrease in all metrics. This outcome is surprising as we assumed that good light conditions result in good detections. The good visibility can explain



(a) Only half of the wolf was detected during daytime.

(b) A wolf disappears and is only partly detected.

(c) Only a part of the wolf is detected during the night.

(d) Two wolves were detected as one

Figure 8.7: Several detections resulting in an IoU smaller than 0.5. The green boxes are the detections, and the red ones are the ground truth. The IoU is given for each set of two boxes. C.f. also Figure 3.8 and Equation (3.4) in Section 3.5.

	FP	**TP**	**FN**	**Prec.**	**Recall**	**F1**
YOLO	72	560	538	0.886	0.510	0.647
YOLO & Segm.	76	562	536	0.881	0.512	0.647
Full flow, $\alpha = 0.5$	138	571	527	0.805	0.520	0.632
Full flow, $\alpha = 0.25$	23	686	412	0.968	0.625	0.760
Full flow, $\alpha = 0.1$	8	701	397	0.989	0.638	0.776

Table 8.3: The results from the three runs over the 806 daytime images. The reduced number of total images has to be considered for the false positives, true positives, and false negatives.

	FP	**TP**	**FN**	**Prec.**	**Recall**	**F1**
YOLO	9	286	105	0.969	0.731	0.834
YOLO & Segm.	14	284	107	0.953	0.725	0.824
Full flow, $\alpha = 0.5$	66	237	154	0.782	0.606	0.683
Full flow, $\alpha = 0.25$	24	279	112	0.921	0.714	0.804
Full flow, $\alpha = 0.1$	18	285	106	0.941	0.729	0.821

Table 8.4: The results from the three runs over the 334 nighttime images. Again, the reduced number of total images has to be considered.

the decrease in precision, recall, and F_1-score during daytime: Humans can easily see distant wolves as such, especially when annotating a time series. For the detector, this is quite hard and results in decreased results.

On the other hand, our YOLO model seems to have problems with wolves directly coming or leaving the camera, i.e., one can only see the head or tail. Here, the human interaction in *Wolf-or-Not* helps to detect those wolves and increases the overall detection.

For the 334 nighttime images as shown in Table 8.4, the F_1-score, in general, is increased for all experiments. The limited detection range can explain this: The built-in infrared spot of the camera can only illuminate a certain area, according to the datasheet, a maximum of 15 m. The advantage of a human annotating distant wolves vanishes, and the general F_1-scores increase. The benefits of the motion detection for distant wolves in *ShadowWolf* are also reduced during nighttime, so the pure YOLO approaches perform similarly. For this dataset, we also see the improvement of the metrics by reducing the α for the IoU.

8.3.4 Required Computing Capabilities

We also evaluated the computing resources required by *ShadowWolf*. We ran the experiments on a standard computer with an Intel-i7 with four cores and 12 GB of RAM. The runtimes of the individual experiments are listed in Table 8.5. The *ShadowWolf* run is split into two parts: part 1 prepares everything for the export to the *Wolf-or-Not* web service. Part 2 processes the data from the web service. The time required for voting on *Wolf-or-Not* depends on the number of users and their motivation and activity. In our case, it took 1-4 days to get sufficient votes for the 2274 subimages. The runtime for "YOLO & segmentation" and "part 1 of *ShadowWolf*" should technically be similar. For *ShadowWolf*, we use the advantage of the deduplication: There are fewer images to be preprocessed by the YOLO framework, resulting in approximately 20 min less runtime. Running the model on the images requires most of the time and can be significantly reduced using a GPU or TPU.

We require a certain amount of disk space to create and store many intermediate images for better understanding and debugging. The size of the reference dataset with 1140 images is 1.7 GB. The amount of data exported to *Wolf-or-Not* is just 71 MB. The labeled images again have a size of 1.7 GB. The debugging images created in between are about 7.3 GB. Those can be disabled to reduce the disk usage.

The next section discusses those results in detail.

Experiment	**Runtime**
YOLO only	30.48 min
YOLO & Segm.	88.03 min
Full *ShadowWolf* part 1	57.35 min
Full *ShadowWolf* part 2	9.22 min
Full *ShadowWolf* complete	66.58 min

Table 8.5: Comparison of the processing times on our computer. *ShadowWolf* is divided into two steps. In between, the evaluation using the web service is done. Here, we can not give a time as it depends on the number of users.

8.3.5 Discussion

The idea of *ShadowWolf* is to improve the detection of objects in camera trap images, i.e., low-quality images. The performance evaluation shows that it can significantly increase the detection compared to a pure model-based approach. Compared to the pure YOLO, the F_1-score increased from 0.70 to 0.79 for the overall reference dataset. The disadvantage is that the automatically created boxes (by the motion detection) do not perfectly fit the objects of interest, in our case, the wolves. Those can be checked and adapted manually with a tool like LabelImg (c.f. Figure 8.5).

Problematic cases are those with hard-to-see and barely moving objects. Wolves distant from the camera are often not detected reliably. Our second step, the background subtracter or motion detector, requires certain animal activity. If this is missing, the object is not or just partly detected (c.f. Subfigure 8.7a). Here, one can use alternative technologies: If the camera is not moving, static reference images like our hourly taken ones can detect slow-moving animals reliably. MegaDetector can also be involved here to create boxes around possible objects of interest. Due to the modularity, one can add this to *ShadowWolf* easily.

Another question is the objective of the run of *ShadowWolf*: Is it to label a dataset or to reduce the effort for a visual image inspection? If an extensive dataset has to be checked for wolves, *ShadowWolf* can lessen the workload drastically by only showing boxes within a specific probability range. Detections with a high or very low probability do not have to be checked. Only the critical ones, i.e., *Maybe a wolf?*, must be checked manually by domain specialists. The increasing number of wolf sightings will increase the workload of those people. For that, we designed everything to be modular and extendable without requiring user interaction. *ShadowWolf* can be started with a given dataset, for example, uploaded to the web service from camera traps. The detection, segmentation, etc. can run fully automatic. Also, the integration in our *Wolf-or-Not* is easy, as an API for image uploading and result downloading already exists. As a result, the user gets an annotated dataset and can decide how to continue the work.

This can be used for the iterative approach: The training dataset can be used to train a new model generation. This one can be automatically evaluated and compared to another model using our toolset. If the performance increases, the new model can be used within *ShadowWolf*, resulting in an overall increase in the performance of the toolchain.

Regarding the running costs, *ShadowWolf* does not require a lot of computing resources and can even run on standard consumer hardware in an acceptable time. However, the run time of the object detection can reduced drastically by using a GPU or TPU, as the execution of the YOLO model takes most of the time. This is key for cost-sensitive users as they are often in the area of wildlife monitoring.

8.3.6 Further Ideas and Challenges

We are continuously extending and adapting the parts of *ShadowWolf*. Currently, we are evaluating the time series evaluation. We plan to add a module that benefits from the batching as described in Subsection 8.2.2. The focus is here on the series of images, or more precisely, the location of the detections joined with the probabilities. Those can be combined to increase or decrease the trust in the detections. In simple words, does the movement of the detections over time make sense? This can increase confidence in a specific detection over time.

Another topic is the deployment of *ShadowWolf*: How can the images from camera traps automatically be transferred to the system? Cellular Internet is not always available, or the speed and the amount of data might be limited. The workflows tailored to the potential users have to be discussed.

In the next section, we will summarise our work on *ShadowWolf*.

8.4 Summary

In this chapter, we introduced *ShadowWolf*– a toolchain designed to label camera trap images and train new models automatically. The objective is also to detect animals that are only partly visible or in challenging light conditions. For that, we use in *ShadowWolf* a combined approach: The detection from a state-of-the-art object detection algorithm is improved with user evaluation in combination with a background subtractor. We create possible detections: Where might be an animal of interest? Those are evaluated using our *Wolf-or-Not* web service. Both results are combined to a probability. This helps to remove false positives and add false negatives not detected by the object detection. The outcome is a labeled dataset that can be used to train a new model or manually analyze the occurrence of wild animals.

We showed that our approach significantly improved the detections in the performance evaluation. The F_1-score on our dataset increased for our primary class of interest *wolf* from 0.7 to 0.8 over all images and from 0.6 to 0.8 for daytime images using our framework compared to the plain object detector. We further evaluated the effect of the light conditions and the required computing capabilities for our approach and the used object detection model.

The source code is available under the *GNU General Public License v3.0* on GitHub[4].

[4]`https://github.com/ComNets-Bremen/ShadowWolf`

Chapter 9

Deployment

The objective of the *mAInZaun* project is to detect and deter the wolves or, more general, predators directly at the fence perimeter. Although the project's objective is not a ready-to-sell product, we discuss several options for a possible deployment. Ideally, all parts should operate autonomously without needing maintenance or other human interaction over a certain time, at least several days. The challenges are the same as discussed in Chapter 5: Mainly, the limited energy, but also, weather and the weight of the devices are just some points that are also relevant to this chapter.

We split this chapter into six parts. Section 9.1 describes our architecture and the challenges. Section 9.2 focuses on the individual parts of the detection hardware and selects the ones we use. Section 9.3 discusses the deterrents and their activation. The wireless setup connecting the individual parts is described in Section 9.4. Our implementation used for the deployment is called *WolfNet* and topic of Section 9.5. The evaluation is discussed in Section 9.6.

9.1 Architecture

In this section, we introduce the deployment architecture, including the challenges. Figure 9.1 shows an exemplary deployment. The center line is the fence with the *mAInZaun* components. We have cameras in three positions (center, left, and right). Between those, we have several deterrents: two times flash and four times sound. The fence divides the two sheep in the upper half from the rest of the world (including one wolf) in the lower part. If the cameras detect a wolf, the nearby deterrents are activated.

The challenges are manifold and discussed in Chapter 5. Exemplarily, we added some to Figure 9.1. The vegetation and the weather are two challenges affecting the detection and deterrents: The vegetation limits the camera's view and the effective range of the deterrents. Similar holds for the weather: fog, rain, snow, etc. have a notable effect on camera, light, and sound. Other animals, like dogs, or humans, are more challenging for object detection. The system should reliably distinguish between wolves and others. If the system is unsure, it can consider other classes of objects in the vicinity: If a human is nearby, it is more likely to detect a dog and not a wolf. This requires powerful detection hardware and also communication between the individual components of the

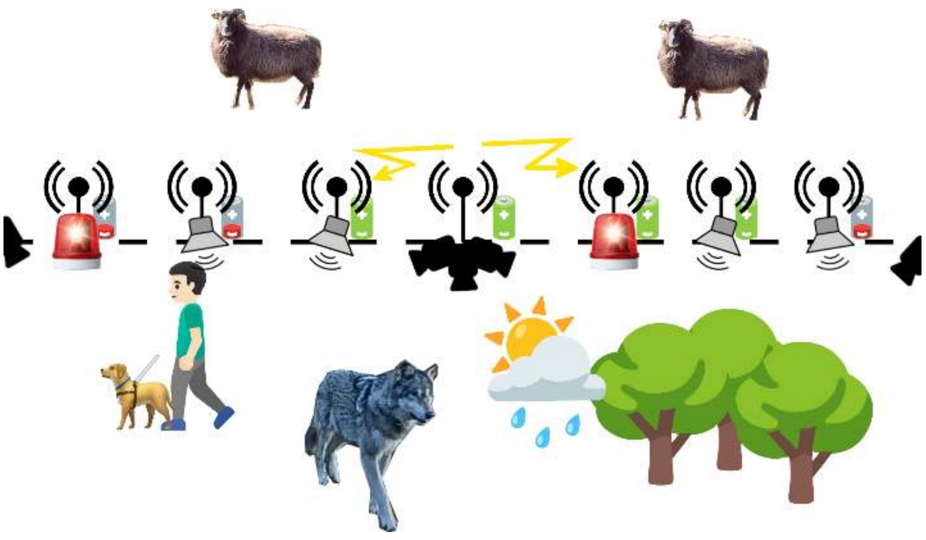


Figure 9.1: The proposed architecture and some challenges. For simplicity, the fence dividing the farm animals on the top and the rest of the world at the bottom is indicated as the dashed, horizontal line in the center. At this perimeter, we have cameras – shown in the center and partly left and right – and several deterrents like sound or light-based ones. If one of the cameras detects a predator, the nearby deterrents are activated.
The deterrents and the cameras can be affected by several phenomena like other animals (dogs, deer, humans, etc.), weather (reducing sight and range), vegetation, etc. Also, the energy is a limiting factor. The objective is to detect and deter the predators.

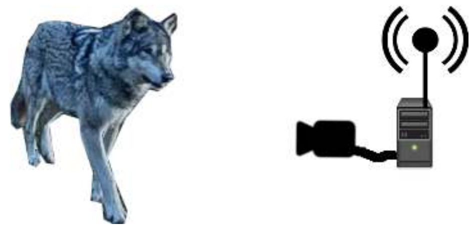


Figure 9.2: The wolf detection works as follows: The detection system uses a camera and a processing unit. Both can be combined in one device. If the model running on the processing unit detects a wolf, it sends a message to the deterrents wirelessly.

overall system. Also, the power supply is challenging: Most of the depicted devices show a green battery. But three deterrents run low on power. This has to be prevented, and the owner should be informed on time.

In the next section, we will discuss the design of our proposed system and its possible deployment.

9.2 Animal Detection Hardware

The first part is the detection hardware. It comprises the camera and the processing unit, as depicted in Figure 9.2. The processing requires also an interface to activate the deterrents wirelessly in case of a positive detection.

9.2.1 The Camera

The market offers various cameras with varying functionality, technical specifications, and prices for all possible budgets. For this project, we focus on cameras from well-known manufacturers, expecting them to provide reasonable support and reliable spare part supply. We distinguish between standard RGB cameras and thermal ones. We do not give the light sensitivity for the RGB cameras as those values from the datasheets depend on the manufacturer and are not always realistic. The camera's resolution gives, especially for the thermal systems, an idea of usability: It is hard to distinguish a wolf from a teapot with just a few pixels. We also discuss the connections: Is the camera connected using Ethernet and Power over Ethernet (PoE), via the General-Purpose Input/Outputs (GPIOs) and Inter-Integrated Circuit (I^{2}C), or via the special camera connector Camera Serial Interface (CSI)? Cameras using WiFi require additional power in the form of wires or batteries and, therefore, are not considered. We checked if the system could be placed outside without the need for additional protection. Figure 9.1 shows the results.

The first line of Table 9.1 shows the camera we also selected for this work. The *Axis M2025-LE* is a surveillance camera built for outdoor usage. It can store images and videos in full HD resolution and also has infrared spots to operate at night. It is connected to the processing unit and powered via one Ethernet cable, reducing wiring effort.

Embedded devices often have a CSI interface to connect a camera to the board directly. Here, various cameras with different resolutions, lenses, qualities,

Model	**Type**	**Resoultion**	**Conn.**	**Out?**	**Price**
Axis M2025-LE	RGB	1920×1080	PoE	✓	425€
RaspberryPi Cam	RGB	1920×1080	CSI	✗	29€
Logitech C920	RGB	1920×1080	USB	✗	100€
Grid-EYE IR-Array	Thermal	8×8	I^{2}C	✗	50€
Axis Q1941-E	Thermal	384×288	PoE	✓	4500€
Flir Elara	Thermal	320×240	PoE	✓	3600€
Flir E6XT	Thermal	240×180	–	✗	2000€
Flir C5	Thermal	160×120	–	✗	690€

Table 9.1: Selected Camera Systems regarding their type, resolution, connectivity, outside protection, and price. We used the *Axis M2025-LE* from the first line for our work. We concentrated on well-known manufacturers to ensure long-term supply. The last two lines contain our two hand-held cameras. Those do not offer a live image stream. (Prices from June 2022)

and prices exist. We took the *RaspberryPi Cam* as an example. It is directly mounted on the RaspberryPi and also offers full HD resolution. It is a plain PCB without a casing and designed for an experimental setup rather than a deployment. For an outside deployment, we have to build a custom case that protects the camera from the weather and does not influence the camera's lenses or field of view. Everything should last outside for a longer period, which requires some engineering effort. Therefore, we preferred cameras that were already designed for outdoor usage.

Similar hold for the *Logitech C920*: This is a standard computer webcam connected via USB. It can be connected to all computers via USB and offers HD videos. However, it is designed for office usage and not for outdoor applications.

As discussed in the Problem statement in Section 2.7, thermal cameras might also be an option. In some situations, they might perform better as they detect the heat radiated by mammals. As a drawback, they are expensive and offer only a low resolution. The cheapest option is the *Grid-EYE IR array*, which is a small 8×8 array of thermal detectors. No real object detection is possible with such a low resolution. The *Axis Q1941-E* and the *Flir Elara* are two thermal cameras meant for outside usage and surveillance applications. Their resolution is sufficient to detect objects but not to identify them in distance. Furthermore, they are costly and unsuitable for mass deployment on a fence.

The last two Flir models, *E6XT* and *C5*, are given as a reference. They do not offer the output of a live image but were used by us to evaluate the limits of thermal imaging.

Altogether, we decided to use a camera that offers normal RGB images and is designed for outdoor use. This is achieved by outdoor surveillance cameras like the *Axis M2025-LE* we use in this thesis. Some processing, like automatic image capturing, can be performed on the camera, but the main object detection task must be performed on separate hardware. The next section evaluates the available hardware platforms for this task.

9.2.2 The Processing Unit

After we have discussed the available cameras in the previous section and selected the Axis M2025-LE as the main model, we require a device to run the inference on. Several devices and options exist to scale the performance according to costs, energy, inference speed, etc. We also evaluate devices that combine cameras and object detection in one device.

The board needs some communication capabilities to signal the detections to the activators. This could be a standard GPIO-pin, a USB connection, or a built-in wireless technology. Most devices offer an option to connect an arbitrary wireless technology such as LoRa, as discussed in Chapter 4.

We evaluated the available systems that are used to perform object detection or other kind of image processing offline, i.e., without the need for an internet connection. We focus on systems with a comparable low price, i.e., less than 500 €, having a more extensive deployment in mind. We checked the computing capabilities regarding the CPU used, the number of cores, and the clock frequency. We also considered if an additional acceleration is built-in to speed-up the inference task. Also, the available RAM and the type of storage are evaluated. Many devices use a standard micro-SD card as a file system. Here, the standard sets the limit: The SDXC-standard limits the maximum size to 2 TB. Newer standards like SDUC increase this boundary to 128 TB. Build-in communication technologies like WiFi and Bluetooth can simplify the setup phase and debugging. Therefore, we stated which are available. Operating everything in remote environments sets limits regarding the power. Therefore, we list the maximum power requirements according to the datasheets. In the column camera, we list how one can connect an external camera to the system or, if available, the properties of the built-in camera. Table 9.2 lists the hardware we had a closer look at. We only consider the bare hardware shipped without external components like case, SD-card, power supply, etc.

The most well-known system is the *RaspberryPi* ecosystem[1]. It offers a variety of different platforms and computing modules. The quad-core CPU of version 5 is announced to be up to three times faster than the predecessor. In Table 9.2, we focus on the standard versions. Besides those, several compute modules, embedded systems, or microcontrollers are offered by the RaspberryPi Foundation but lack computing resources or are meant to design a completely new system. The OS *Raspberry Pi OS* is an adapted Debian version and receives regular updates. From the hardware perspective, several GPIO pins are available, and many add-on boards extending the RaspberryPi with various functionality, like communication capabilities, are available. It does not have an optimized GPU or TPU to speed up the inference task, but such can be added, for example, using the Google Coral Accelerator. Besides the standard connections like USB and Ethernet, a RaspberryPi has a CSI connector for cameras available.

Another commonly used embedded hardware platform is the *Jetson Nano Developer Kit*[2] (commonly called Jetson Nano). NVIDIA, the developer, also offers several compute modules to embed this system into new hardware. In contrast to the RaspberryPi, the Jetson has a built-in GPU speeding up the

[1] `https://www.raspberrypi.com/`

[2] `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`, accessed: 2023-12-10

Model	**CPU**	**Acceleration**	**RAM**	**Storage**	**Wireless**	**Power**	**Camera**	**Price**
RaspberryPi 4B	Cortex-A72 4 × 1.5 GHz to 1.8 GHz	No	1 GB to 8 GB	Micro-SD	WiFi, Bluetooth	3 A @ 5 V	External: Ethernet, CSI, USB	40-84 €
RaspberryPi 5B	Cortex-A76 4 × 2.4 GHz	No	4 GB to 8 GB	Micro-SD	WiFi, Bluetooth	5 A @ 5 V	External: Ethernet, CSI, USB	69-93 €
NVidia JETSON NANO	Cortex-A57 4 × 1.43 GHz	GPU	4 GB	Micro-SD	No	2 A to 4 A @ 5 V	External: Ethernet, CSI, USB	205 €
Google Coral DevBoard	Cortex-A53 4 × 4 GHz	TPU	1 GB to 4 GB	Micro-SD / 8 GB internal	WiFi, Bluetooth	3 A @ 5 V	External: Ethernet, CSI, USB	167-230 €
Luxonis OAK-1 AF	Intel Myriad X 700 MHz	VPU	0.5 GB to 2 GB	16 MB internal	No	1.5 A @ 5 V	Internal: 12 MP	191 €
Luxonis OAK-D Pro	Intel Myriad X 700 MHz	VPU	0.5 GB to 2 GB	16 MB internal	No	1.5 A @ 5 V	Internal: 12 MP, nightvision	383 €
ESP-S3-EYE	ESP32-S3 2 × 40 MHz to 240 MHz	No	up to 32 MB	8 MB	WiFi, Bluetooth	USB	Internal: 2 MP	30 €
Arduino Nano 33 BLE Sense	Cortex-M4F 64 MHz	No	256 kB	1 MB	Bluetooth	USB	External: GPIOs	49 €
Google Coral Accelerator	N/A	TPU	N/A	N/A	N/A	USB	N/A	91 €

Table 9.2: Comparison of selected hardware platforms for the inference in the field. We split it into three main groups: systems that run a complete operating system, complete embedded systems (Luxonis), and microcontroller-based ones (ESP32, Arduino). The Coral Accelerator takes a special role: It extends other systems like the RaspberryPi with an external TPU and speeds up the inference.
The range in the prices for the Coral and RaspberryPis results from versions with different RAM. Models with SD-storage can usually handle up to 2 TB. We use the RaspberryPi 4B (marked grey). (Prices from Dec 2023)

inference using the standard AI frameworks. The operating system is an adapted Debian with some extensions for the GPU. Like for the RaspberryPi, Ethernet, CSI, and GPIOs are available. It does not have Bluetooth or WiFi integrated.

The *Google Coral DevBoard*[3] is the third platform often used for AI deployments. Similar to the previously mentioned ones, Coral offers several OEM-options to integrate the functionality into new hardware developments. It has a TPU accelerating the tasks in machine learning. In contrast to a GPU, the models must be quantized before they can be executed on such a TPU as it only operates with 8 bit integers. This adaptation slightly affects the detection performance but significantly increases the energy and computing efficiency. The Coral offers similar interfaces compared to the RaspberryPi: WiFi, Bluetooth, Ethernet, GPIOs, and CSI are available on board. The operating system on the Coral boards is Mendel Linux, a lightweight adaptation of Debian.

The three systems before do not have a camera system integrated. Arbitrary cameras can be connected, for example, using Ethernet or USB, gaining flexibility. Other options are integrated systems as, for example, *Luxonis*[4] offers. Their embedded hardware platform includes an acceleration unit they call Vision Processing Unit (VPU) and at least one camera. They offer a variety of different modules with different camera types. Table 9.2 exemplary lists two. The *OAK-1* is a cheaper system with one 12 MP color camera. The more expensive *OAK-D Pro* additionally offers depth information and night vision capabilities with built-in infrared LEDs. They also provide models with PoE rated for outdoor usage or the individual parts to embed everything to own developments. They do not give direct access to a GPIO, which requires either additional hardware for the notification or a customized hardware design to activate the deterrents. Existing object detection or classification models can be adapted using their toolchain, i.e., quantized and uploaded to their modules. The inference runs directly on the module in real-time.

In the last couple of years, microcontrollers have become more and more powerful and offer today comparable high computing power at a low price with low energy requirements. As an example, we take the *ESP-EYE* by espressif[5]. It combines a camera with a microcontroller and allows it to run simple tasks like face detection. Due to the constraints in memory and computing power, only small models with low-resolution images can be run on such microcontroller-based systems. The authors in [98] used the MAX78000 microcontroller by analog devices to run a minimal, ultra-lightweight YOLO model (*TinyissimoYOLO*) for a limited set of classes and RGB input images with 88×88 pixels. Similar holds for the *Arduino Nano 33 BLE Sense*[6]. An adapted version of TensorFlow allows running minimalistic person detection on the device[7]. A camera can be connected using the device GPIOs. Those microcontroller-based systems are primarily programmed with the manufacturer's toolchain using C or C++ or using the Arduino ecosystem.

The last row of Table 9.2 shows a special case: The *Google Coral Accelerator* also comes from the Coal ecosystem. It is a TPU which can be attached to

[3]`https://coral.ai/`

[4]`https://www.luxonis.com/`

[5]`https://www.espressif.com/en/products/devkits/esp-eye/`, accessed: 2023-12-10

[6]`https://docs.arduino.cc/hardware/nano-33-ble-sense`, accessed: 2023-12-10

[7]`https://github.com/tensorflow/tflite-micro-arduino-examples`, accessed: 2023-12-10

other devices via USB. It adds the advantage of TPU acceleration, for example, to a RaspberryPi.

It should be mentioned that we focus in Table 9.2 on the most widely used devices. This holds especially for the RaspberryPi: It is the only prominent device from a complete class of small computers. Similar devices with slightly different hardware and application backgrounds exist but are not further discussed.

9.2.3 Summary

We discussed the two main parts of the detection system in the previous sections. Table 9.1 lists suitable cameras, and the overview in Table 9.2 shows an excerpt of the available hardware options for the inference. Regarding the camera, we decided to go for outdoor surveillance cameras as they offer all the required properties. An additional device is required to run the object detection. The most promising options are the devices from the RaspberryPi ecosystem. As shown in Table 6.2 on Page 65, the inference times are still acceptable and improve with every new version of the RaspberryPi or by adding the Coral Accelerator. The prices are comparably low, and we can easily add additional communication capabilities using the GPIOs. It also has a large community offering support, libraries, and hardware.

The devices with on-board acceleration, i.e., GPU or TPU, like the Coral or Jetson boards, show better performance but are more expensive and have a smaller community. Their OEM boards might be an option for a customized hardware design.

The Luxonis cameras show promising performance and are ready to use. Unfortunately, there are no GPIOs available that can trigger external events. Here, we require additional hardware like a RaspberryPi to activate the deterrents. Also, here, the OEM parts might be a choice for the later development of custom hardware.

From an energy and cost point of view, the embedded systems would be perfect. Unfortunately, the maximum size of the models and camera images is restricted: Only low-resolution images and models with few classes can be executed. The example code focuses mostly on simple person detection close to the camera. With the rapid developments in this area, this will change soon.

Therefore, we focus this work on the RaspberryPi in combination with the Axis camera to detect animals in the field. The following section will discuss the possible deterrents.

9.3 Deterrents

After discussing wolves' detection in the previous section, this section focuses on the deterrents. Those are activated wirelessly and should reliably and sustainably deter the wolves. Ideally, the animals will keep their distance from the pasture for a longer time.

Chapter 2 discussed commercially available hardware to deter dogs and wolves. This section briefly discusses the deterrents currently used in the *mAIn-Zaun* project from the technical side. It also relates those to our *WolfNet* sensor

Figure 9.3: The deterrents are activated wirelessly by the system from Figure 9.2 and expel the detected wolf. Exemplary, this figure shows light and sound as two possible deterrents.

network, as it will be discussed in Section 9.5. Our project partners evaluate the effect of the deterrents on predators and farm animals.

The currently selected deterrents aim at the senses of the predators, especially the eyes and the hearing. Figure 9.4 shows the currently used devices. All operate at 12 V.

Figure 9.4a is a powerful floodlight developed for heavy-duty construction vehicles[8]. It can provide a maximum luminous flux of 8000 lm, operates at 12 V with a maximum power of 70 W. As wolves react sensitively to strobe light, we use *WolfNet* to power cycle this light with 20 Hz. This device is also the most powerful one from the energy point of view. It sets the limit for the current offered by the *WolfNet* devices.

The second option, as shown in Figure 9.4b, is the ready-to-use ultrasonic generator M175 by Kemo[9]. It requires 12 V to 14 V and draws a maximum of 150 mA. The siren-like sound can be changed on the device in a range of 8 kHz to 41 kHz. The deterrent range is given with a maximum of 100 m and sound pressure of maximum 135 dB.

The ultrasound cannon M161, also from Kemo[10], is depicted in Figure 9.4c. Here, we use it as an arrangement of three cannons, each placed in one, grey, 30 cm long tube as recommended by the manufacturer. It it powered with 12 V to 14.4 V and draws a maximum of 150 mA. It generates ultrasonic pulses with a fixed frequency of 22 kHz and maximum 120 dB. According to the manufacturer, this results in an acoustic range of up to 300 m. The real range of sound, especially ultrasound depends on a variety of different parameters, including humidity, temperature, wind, general environment, surface etc. The required tables, equations, and considerations are available in various publications [99, 100, 101].

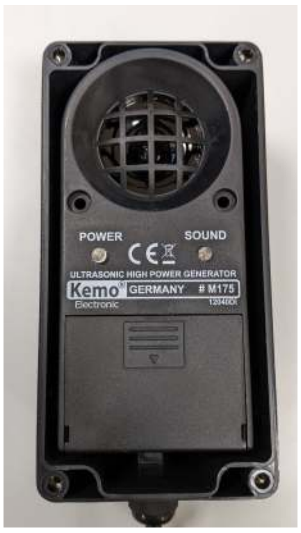
Our project partners from the *Professorship of Animal Husbandry, Be-*

[8]`https://www.was.eu/de/arbeitsscheinwerfer/w130-8000/1214I/2276`, accessed: 2023-12-10

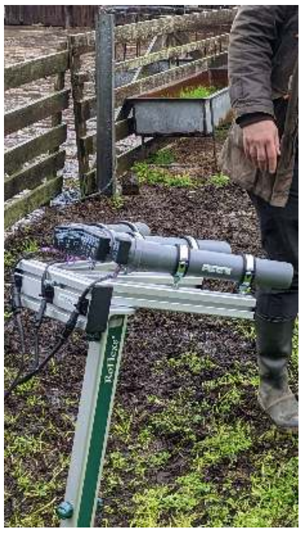
[9]`https://www.kemo-electronic.de/de/Auto/Module/M175-Tiervertreiber-Ultraschall-Leistungsstark.php`, accessed: 2023-12-10

[10]`https://www.kemo-electronic.de/de/Auto/Module/M161-Ultraschall-Power-Kanone.php`, accessed: 2023-12-10

(a) A powerful light creating flashes with up to 8000 lm.



(b) The Kemo M175 creates siren-like ultrasound in different frequencies.



(c) The Kemo M161 creates directed ultrasound pulses using a fixed frequency. The tubes help to direct the sound.

Figure 9.4: The three deterrents currently used with *WolfNet* in the *mAInZaun* project.

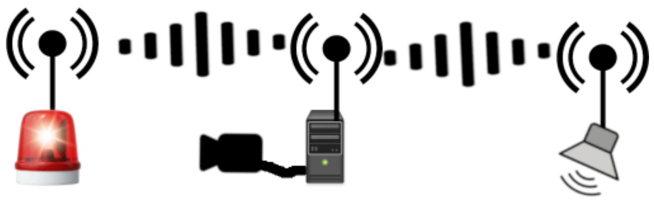


Figure 9.5: After designing the detection and the deterrents, we use LoRa as the communication technology to activate the deterrents and send status messages.

haviour and Welfare at the Justus-Liebig University in Gießen currently evaluate the effect of the deterrents on farm animals and possible predators. The objective is to find, align, and parametrize the deterrents so that the farm animals are only marginally affected and show a sustainable effect on the predators.

Our task in this thesis regarding the deterrents is to ensure the general functionality from the technical point of view. The animal tests, especially on wolves, require preliminary studies and a lot of paperwork. Due to the wolf's high protection level in Europe, applying for a license for animal testing is complex and takes a long time. We are optimistic that those required permissions will be available till the end of the *mAInZaun* project.

9.4 Communication in the Field

After we selected the hardware for the detection in Section 9.2 and also discussed the deterrents in Section 9.3, we introduce the communication between both. In Chapter 2, wireless communication based on LoRa was selected and evaluated in Chapter 4. In contrast to other technologies, it allows direct communication between two nodes without needing an infrastructure, as depicted in Figure 9.5.

Now, LoRa has to be connected to the other parts of the system. Fortunately, many LoRa transceivers, adaptation boards, hardware platforms, and drivers exist. One can basically select the hardware according to the requirements. For the RaspberryPi, we chose the *Raspberry Pi LoRa/GPS HAT* from seedstudio[11]. It has a standard Semtech SX127x LoRa transceiver for 868 MHz and additionally, a Global Navigation Satellite System (GNSS) module. The latter is not required, but having the clock time available is advantageous. Also, automatically knowing the exact position of some nodes can be beneficial during the system setup phase. This module can be used with standard Python libraries to send and receive packets.

For the deterrents, we do not require a lot of computing power. Therefore, we decided to use an ESP32-based microcontroller. Here, we use a *Heltec*[12] board. It uses the same Semtech SX127x LoRa transceiver we selected for the RaspberryPi. Additionally, it has a small display, which helps to show some status information like the number of packets received, the battery status, and the device address while working in the field. Like the RaspberryPi, the Heltec can be programmed in Python, allowing us to reuse our code on both devices.

The transceivers are from the same family as the ones used in the Evaluation in Section 4.2. Therefore, we are confident that our setup will lead to similar results regarding the communication range in the final deployment.

The following section will describe our software implementation, combining the individual components from the previous sections into a complete ecosystem.

9.5 *WolfNet*: A LoRa-based Sensor Network

Bringing the hardware to life requires suitable software. We created *WolfNet* as one implementation that connects the individual parts from the previous sections to a simple WSAN to activate the deterrents. We publish it as an open-source project and optimize it for LoRa. The source is available on GitHub[13]. For the design, we defined the following requirements:

- **Reliable**:
  The system should be able to operate even if some nodes fail. It should not rely on a single point of failure.
- **Secure**:
  The data transmission should be encrypted.

[11] `https://www.seeedstudio.com/Raspberry-Pi-LoRa-GPS-HAT-support-868M-frequency.html`, accessed: 2023-10-12
[12] `https://heltec.org/project/wifi-lora-32/`, accessed: 2023-12-10
[13] `https://github.com/ComNets-Bremen/WolfNet/`

- **LoRa Optimized**:
  The system should be optimized to LoRa and meet the duty-cycle constraints for the 868 MHz frequency band.
- **Adressing**:
  The default LoRa packets do not have an addressing scheme. We require broadcast and node addresses to allow selective activation of the deterrents.
- **Status Update**:
  It should be possible to collect the system's status, for example, by using a maintenance device.
- **Standard Hardware**:
  Everything should run using commodity hardware.
- **Actuator Driver**:
  We need an option to connect different, sometimes high-power, deterrent actuators.

We base *WolfNet* on LoRa and implement a packet format fitting into normal LoRa frames. As the range of LoRa – especially in the free field – is usually at least several hundred meters (c.f. Chapter 4), we do not require routing or packet forwarding technologies. Therefore, we implement only a simple addressing scheme supporting direct node addressing (called unicast) and also broadcast, i.e., sending a message to all nodes of the network in the communication range. For that, we use 32 bit pseudo-random addresses. We also support acknowledgments to confirm the successful reception, if required. The packets are validated using a CRC and encrypted using a simple Advanced Encryption Standard (AES). AES operates with a fixed block size, leading to the resulting packet size being a multiple of 16 B. Smaller packets have to be padded, i.e., extended to match this size. As a result, all currently used packets (besides the acknowledgments) have a fixed size of 48 B.

For the transmission, we use LoRa with the default parameters according to our experience from other projects: a spreading factor of 7, a code rate of $\frac{4}{5}$, and a bandwidth of 125 kHz at a frequency of 868 MHz. Together with the packet size of 48 B, this corresponds to the first line of Table 4.1 on page 46.

We mainly use two packet types: The normal packets are sent in case of the detection event of a predator and activate the deterrents. If this packet is sent as a broadcast, all deterrents of the network in radio range are activated. As an alternative, this activation message can also be sent as a unicast message if only a specific deterrent should be activated. The required message type depends on the application, deterrents, and configuration and can vary.

The second message type is the beacon message, transmitted in the configurable interval of 120 s. This message type fulfills three duties: Firstly, it helps to detect the nodes in the communication range and can also help to detect failures. Secondly, it is used to estimate the distances and the reliability of the network by the received signal strength. Thirdly, it transmits the battery level as voltage and percentage, helping monitor the devices and prevent low batteries.

We implemented *WolfNet* on two hardware systems. First is a RaspberryPi with a LoRa shield. This one is used to send deterrent activation messages in

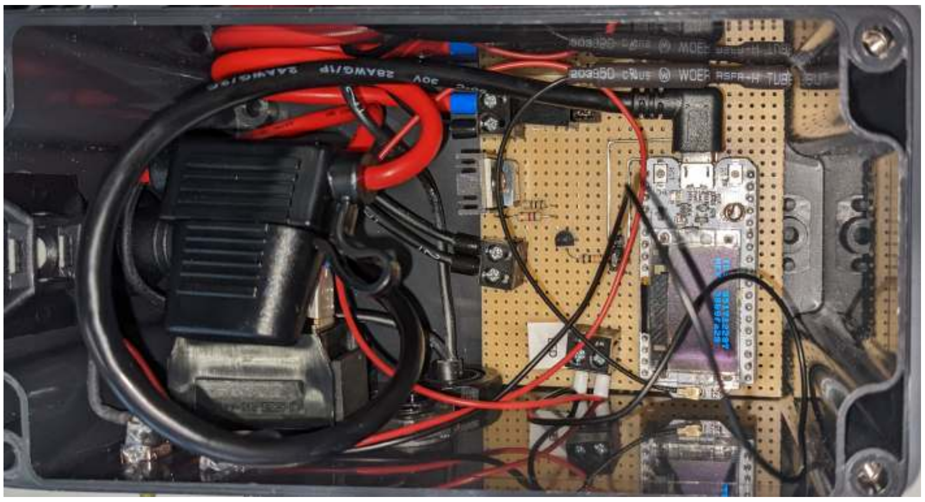


Figure 9.6: The first version of our *WolfNet* actuator in a weather-protected box. The 10 A fuse in a black protection, and the power switch can be seen on the left. On the right side, the circuit according to the schematic from Appendix D, including the Heltec, is visible. The box is powered using a 12 V battery.

case of a positive detection. The second is used on the deterrents and based on *Heltec* microcontrollers.

A third hardware option exists for debugging and evaluation reasons. It is also based on the *Heltec* and used to manually activate the deterrents using a button or motion sensor.

The deterrents used in this project require 12 V and draw a current of up to 7 A. This requires a powerful battery, in our case, a 12 V car battery. The ESP32 on the *Heltec* is not able to handle such voltages and currents. Therefore, we created a small circuit to supply the microcontroller and operate the deterrents. The schematic and the used components are listed in Appendix D. Everything is packed in a weather-protected box and shown in Figure 9.6. The box is powered using the 12 V battery and can activate deterrents with 12 V and up to 10 A.

The next section will discuss the overall evaluation of the complete system.

9.6 Performance Evaluation of *WolfNet*

We validated the overall performance of *WolfNet* as follows: To ensure that the circuit operates as expected, we ran it with the maximum load, i.e., the spot, and measured the voltage drop, current, and temperature of the most important parts. We tested it with flashing and continuously switched on light. In case of design problems, the output voltage should significantly differ from the input voltage. If parts are not correctly dimensioned or wrong assumptions are made, those parts usually become hot. We confirmed with our thermal camera that the critical parts operate within the specified temperature ranges. We found no such issues on the circuit.

We have a built-in battery monitoring system to estimate the remaining lifetime. We evaluated the voltage and the state of charge of the battery measured by the system and found no significant differences compared to the externally measured values.

Regarding the software and possible bugs, we ran the devices in our lab for 24 days, regularly triggered the detections, and monitored the debug output. Till the end of this test, the deterrents were activated reliably without any errors or system reboots. We regularly received the broadcasts with the status information and the unicast messages to start and also stop the deterrents.

Due to our experience with LoRa and the discussion and results in Chapter 4, we are confident that wireless communication is not a real challenge in the expected applications. We can also easily adapt some of the LoRa parameters to adapt in case of minor communication issues. We are more interested in the overall system's performance: How everything is placed, how non-experts work with the hardware, what has to be considered regarding the power supply and possible false alarms, etc. Testing all those cases separately requires a lot of effort, time, and resources.

On the other hand, experiments with the complete system are scheduled at the end of the *mAInZaun* project. Here, everything will be tested from all relevant perspectives, i.e., technically and from the animal behavioral point of view. Especially for the latter, the application to run experiments with animals, especially wolves, is complex and very time-consuming. Therefore, we had to reschedule planned field trials in the past.

It makes sense to combine those experiments together and evaluate the overall system. We are confident that those final experiments in the *mAInZaun* project can be performed till the end of the project. We will evaluate and publish those results later.

Chapter 10

Summary, Conclusions, and Outlook

This thesis presented several novelties in the area of object detection for outdoor images. In this chapter, we summarize our work, provide conclusions, and discuss the areas for future work.

10.1 Summary

Taking care of wildlife and striving for a sustainable life between wild animals and humans becomes more and more important. This coexistence is not always easy, as can be seen in the case of predators or Carnivora. This thesis takes the wolf in Germany as an example. The number of killed farm animals increases continuously, and therefore, the number of humans demanding to limit the number of wolves also rises. The *mAInZaun* project, in which context this thesis has been written, offers an alternative solution. The objective is to use the recent developments in the area of artificial intelligence, combined with automatically activated, non-lethal deterrents, to keep the wolf away from farm animals.

For that, a camera continuously monitors the farmland environment and analyzes the pictures using an object detection model. If this model detects a predator like the wolf, deterrents are activated and expel the attacker. Ideally, this will protect the farm animals from attacks.

Such a system should run automatically with only minor maintenance. Also, environmental challenges like vegetation and weather have to be considered.

In the context of the overall tasks, we provide several contributions. The **first contribution** is the **image dataset**. Most existing datasets focus on high-quality images. For the application in this project, realistic images are required. We have discussed the challenges in outdoor images and what can happen unexpectedly. In the end, we collected more than 100.000 images of different animals, primarily wolves. In contrast to the existing dataset, we explicitly include bad images that are often sorted out. Our dataset further consists of a time series that allows further applications.

Working with those images in the area of computer vision requires labels: What shows this image? Creating those labels is a tedious task. Here, our **second contribution** provides a **webservice for crowd-based image labeling** called *Wolf-or-Not*. In contrast to existing services and applications, we reduce the user interface to a bare minimum and open it to the general public. With a kind of gamification, even non-experts can easily help label or evaluate many images quickly.

In our **third contribution**, we focus on the **automatic creation of training datasets**. We combine our *Wolf-or-Not* web service with state-of-the-art object detection and several pre- and postprocessing steps. The resulting *ShadowWolf* can run automatically and continuously create training datasets without human interaction. It performs well for hidden objects, especially distant wolves, as in our case. We use those training datasets to train arbitrary, scalable object detection algorithms. *ShadowWolf* is modular and extensible and can also be used for other tasks like evaluating camera trap images. This tool is beneficial for everyone working with this kind of picture.

Object detection and classification are essential parts of this thesis. Our **fourth contribution** is the **evaluation of current models** in this area. We have compared the existing frameworks and models and selected one, namely YOLO, that we have evaluated in more detail, especially regarding the model runtime on different hardware. The outcome is interesting for everybody deploying such models on various hardware and offers guidance for the model selection.

The objective of the project is to deploy everything in the field. This **deployment** is our **fifth contribution**. We have evaluated the suitable cameras and computing devices for outdoor deployments. We have further discussed the available deterrents and suitable communication technologies for such an application. Finally, we offer a new wireless sensor network implementation, called *WolfNet*, tailored for wireless wolf detection and deterrents.

The following section will conclude this work.

10.2 Conclusions

Working with our real-world, low-quality camera trap images differs from most other tasks in the field of object detection and image processing. For this work, we have a strong focus on wolves and have collected more than 100,000 images of them in various environmental conditions. Our system shows good performance and is beneficial for many applications and users. In contrast to pure machine-learning technologies, we show significantly higher performance, especially for hard-to-see animals. We designed it to be flexible and adaptable to other animals, models, and applications.

This flexibility regarding other animals has to be evaluated ideally by adding dogs. Unfortunately, dogs' visual appearances vary widely: Bernese Mountain Dogs, Bulldogs, German Shepherds, Staffordshire Bull Terriers, etc., look completely different. Furthermore, uncountable mixed-breed dogs exist. The objective is that our system can also detect those dogs and deter them if no human accompanies them. Collecting a suitable dataset with our camera in the outside environment is quite challenging and takes a lot of time. We are confident that we will benefit from our iterative approach in this task and adapt continuously to new classes.

Regarding deploying those models, we evaluated several architectures and focussed on YOLO for our work. Here, we tested the performance of the different models on various machines. Depending on the available resources, one can select a suitable model. The nano YOLO model with just 6.6 MB needs on a RaspberryPi just 388 ms to analyze one full-HD image. Those findings are essential for later optimizing and deploying the detection system.

We also discussed the overall system deployment. Here, we focus on how to activate deterrents automatically and wirelessly. With *WolfNet*, we offer an implementation of a new LoRa-based outdoor sensor network for remote environments. We evaluated the main functionality in our lab and with communication range tests. In the next step, together with our project partners, we will evaluate the complete system, including the deterrents and their effect on the wolves. Due to our experience with wireless sensor networks, we do not expect significant technical challenges in this part.

This work handled several aspects in the area of wildlife detection and deterrence. Due to the complexity and variety of the tasks and challenges, we plan to continue this work. The following section gives an outlook on future work.

10.3 Outlook

The project and the variety of tasks are quite extensive. We have several ideas and visions for extending this work further.

During the work, we collected a lot of images of wolves in their habitat. Others might be interested in those datasets with the rising interest in the wolf. Therefore, we plan to publish our datasets soon.

This work has a strong focus on the wolf. We would like to evaluate everything also on the closest relative, the dog. Especially stray dogs can also lead to panic and might also kill sheep. We did not collect sufficient and realistic dog images to compare the sensitivity of our models between dogs and wolves in a natural environment. The same holds also for other predators like jackals or lynxes. Also, outside the context of predators, we plan to use our system as a general wildlife monitoring support system for all kinds of animals. It can easily be adapted for (assisted) animal detection for arbitrary wildlife images.

Another idea currently being handled as a student project is evaluating time series and object tracking for *ShadowWolf*. Here, we test if *ShadowWolf* can benefit from the movement of an object through the image over time to increase (or decrease) the trust in a particular detection. The idea is to analyze a series of images and track the position of an individual over it. We assume this approach can increase the detection in images where the wolf is only partly visible and covered, for example, by the vegetation.

Finally, we plan to deploy the complete system, including the sensor network, deterrents, and camera system, at the end of the project. Here, aspects like reliability, communication, usability by non-experts, etc, will be evaluated and published as separate works.

Appendix

Appendix A

Camera System

This appendix gives the details of the camera system we used to collect our data. We give the part list as well as some insights into the assembly.

We accept no liability or responsibility for the system, the structure, or the parts used. This documentation and the parts list are intended solely as a source of ideas and inspiration.

A.1 List of Materials

Component	Type	Descripion
Outer box	Thalassa NSYTBS292412T	Outer casing and weather protection
Cable connection box	misc.	For the high voltage connections
Power Cable	misc.	For outdoor usage
Ethernet cable	30 cm	For the connection from the RaspberryPi to the PoE injector
Ethernet cable	30 cm	From the PoE injector to the ethernet through
Ethernet cable	20 m	From the box to the camera
Ethernet inline coupler	Conec IK10024	Connection for the ethernet cable
PoE injector	AXIS T8120	Power supply for the camera
5 V power supply	LOGILINK PA0122	60 W, with external cable. For RaspberryPi and WiFi stick.
RaspberryPi (RPi)	RaspberryPi 4, 4 GB	Processing and storage of data
Case for RPi	misc.	Additional protection for the RaspberryPi
SD-Card for RPi	16 GB	Operating system and storage
Power cable for RPi	USB-C cable	Connection between 5 V power supply and RaspberryPi
Cable Gland	M12	For the power cable. Including nut
Nylon breathable valve	M12	Reduce the risk of condensation inside the case. Including nut
Spring clamps	Wago 221	For high voltage connections
Camera	Axis M2025-LE	Weather-protected camera
SSD	1 TB USB SSD	Data storage
WiFi stick	HUAWEI E8372-W	LTE internet connectivity

A.2 Assembly of Components

This section gives some impressions on the assembly of the box.

Figure A.1: A short video of the assembly is available on YouTube:
`https://youtu.be/fM83phMp5wA`

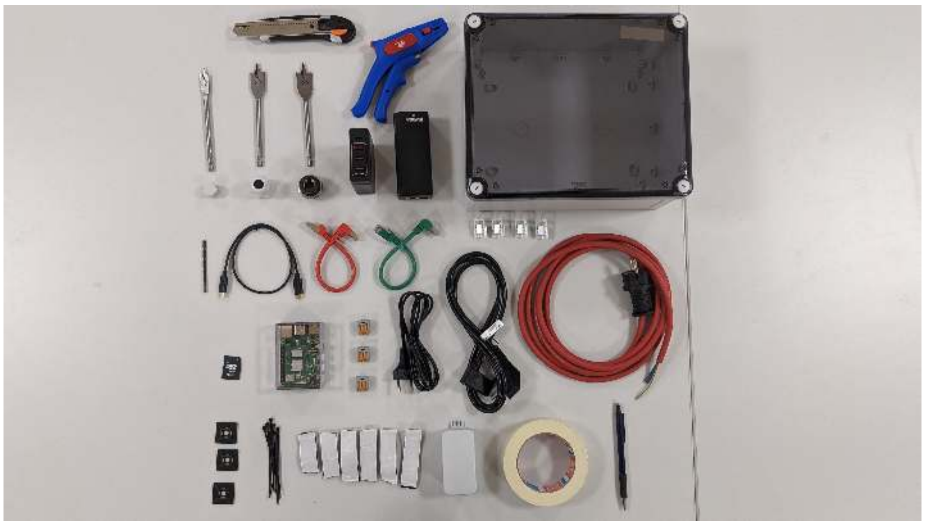


Figure A.2: Overview of the main parts of the camera system according to the Table in Section A.1. Also, some required tools are shown. Not shown: camera, WiFi stick, and ethernet cable for the camera

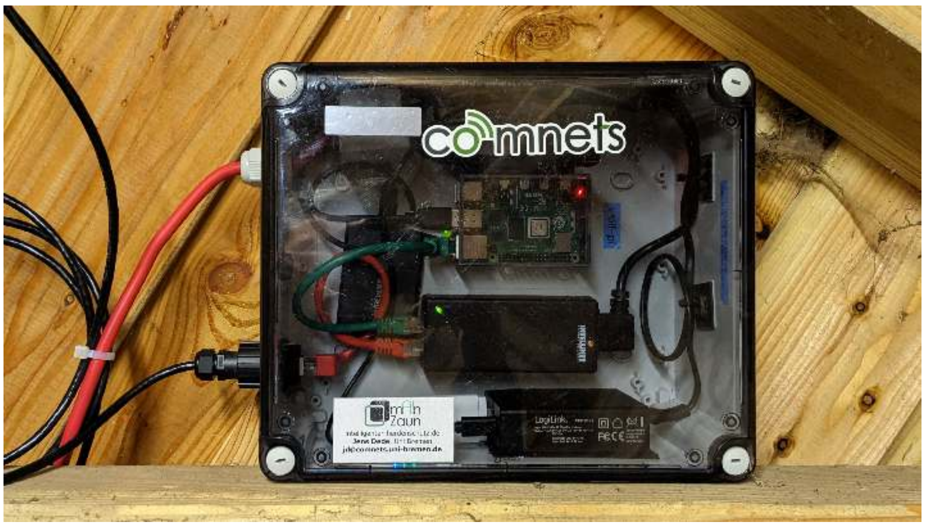


Figure A.3: A picture showing the camera system during the field test.

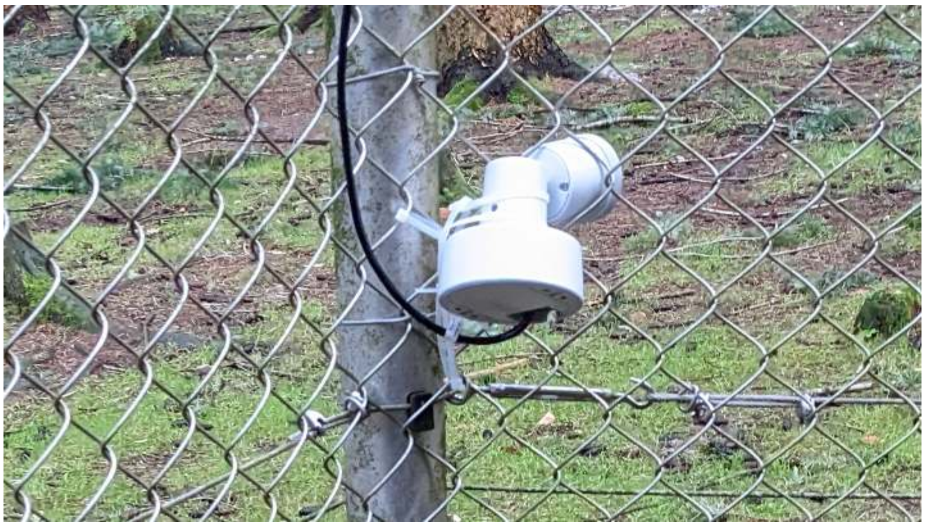


Figure A.4: During the tests, the camera was mostly connected to the fence built around the wolves.

Appendix B

Our Datasets

Chapter 5 described the collection of our images and our main datasets. This appendix lists the datasets and also gives some example images. All photos were taken with an Axis M2025-LE in Full HD resolution (1920×1080 pixels).

B.1 Alternativer Bärenpark Worbis

In this park, we collected our first dataset. The wolves there were hybrid American Timber Wolves. This breed does not live in the wild in Germany. Thus, we used this dataset only for the first evaluation and collected further images of European Grey Wolves living in Germany.

Location	Alternativer Bärenpark Worbis, Duderstädter Allee 49, 37339 Leinefelde-Worbis
Animals	Two American Timber Wolves (black and white), several Brown Bears, some Squirrels
Data collection	2021-09-13 - 2021-09-14
Number of images	9536
Size	15.5 GB
Night images	203
Day images	9333
Camera positions	1

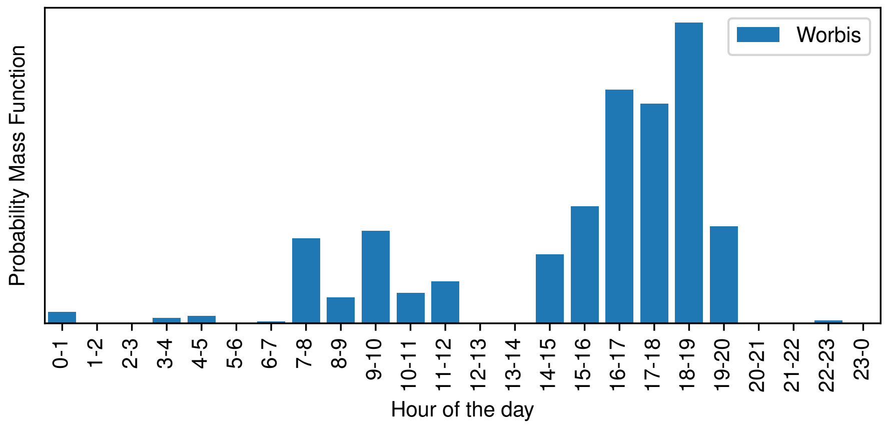


Figure B.1: The distribution of the images collected in Worbis over the hour of the day.

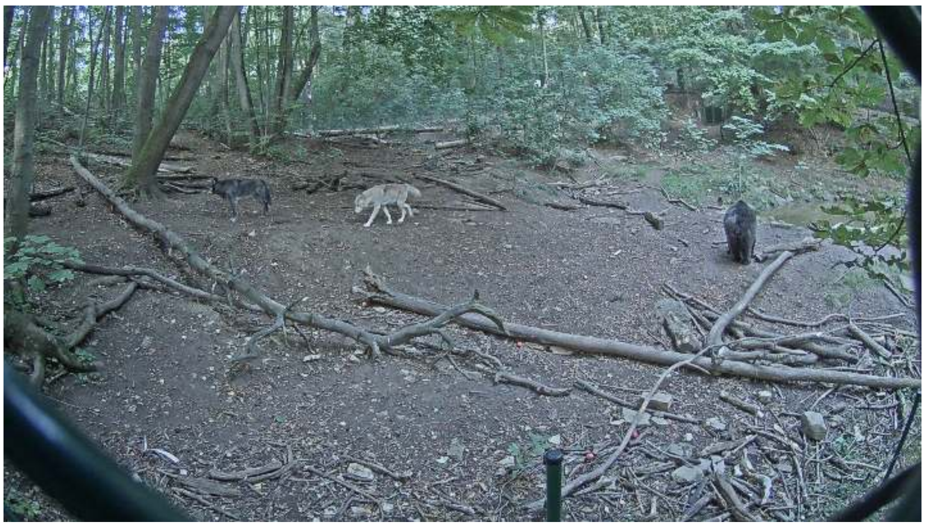


Figure B.2: Two wolves and one bear were captured in Worbis in the daytime.

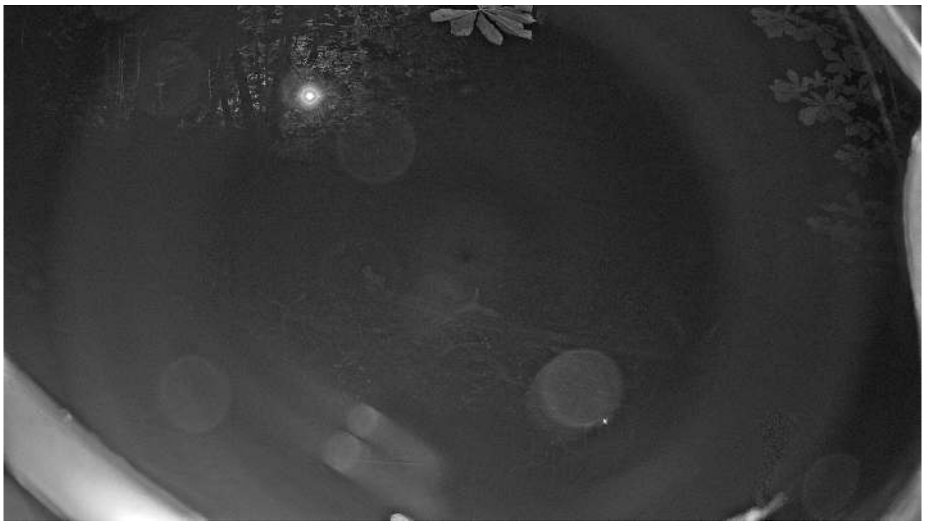


Figure B.3: During the night, another camera with infrared light pointed at our camera, resulting in unusable images.

B.2 Wildpark Lüneburger Heide

In this park, we collected our second dataset and the first one containing only European Grey Wolves.

Location	Wildpark Lüneburger Heide, Wildpark 1, 21271 Nindorf-Hanstedt
Animals	Two European Grey Wolves
Data collection	2022-03-28 - 2022-03-30
Number of images	27538
Size	47.5 GB
Night images	991
Day images	26547
Camera positions	2

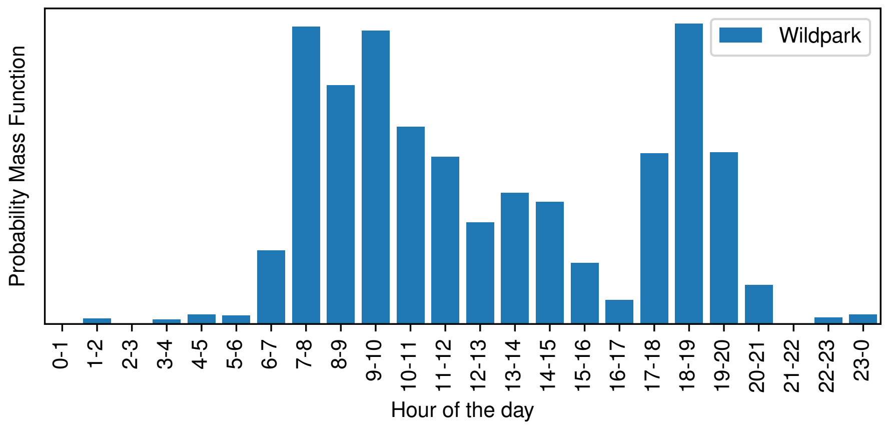


Figure B.4: The distribution of the images collected in Wildpark over the hour of the day.

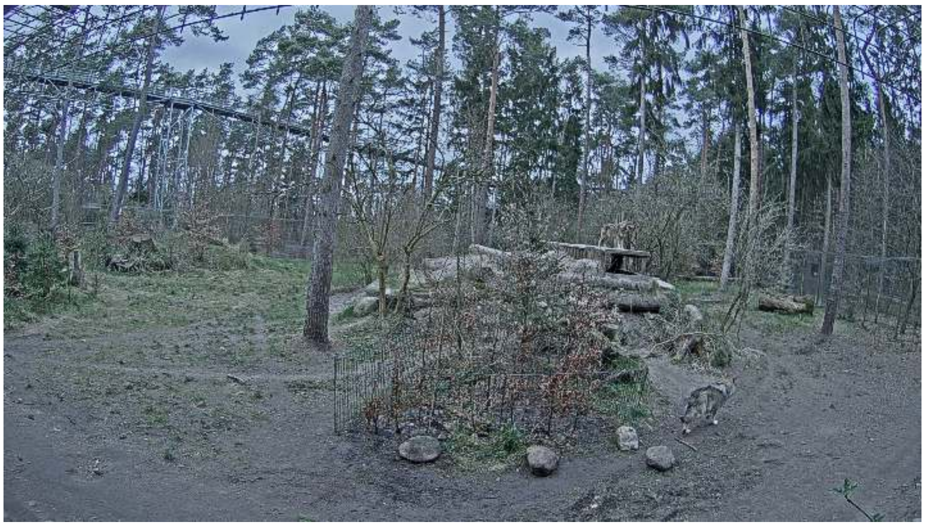


Figure B.5: Wildpark, camera position 1 during the day with two wolves.

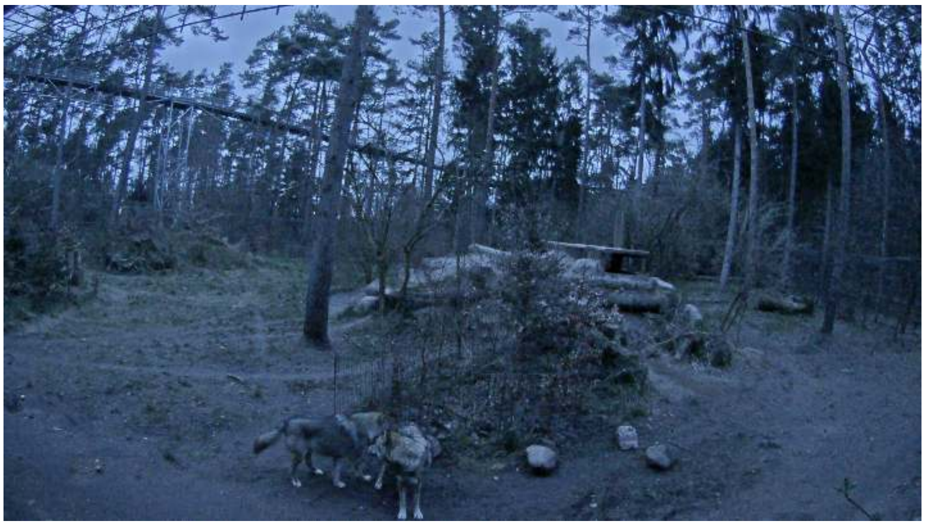


Figure B.6: Wildpark, camera position 1 during the dusk with both wolves.

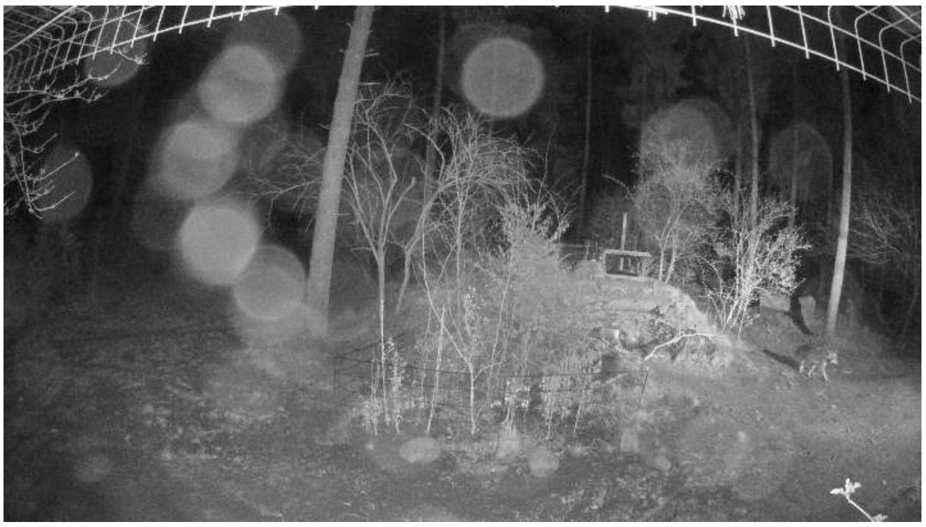


Figure B.7: Wilpark, camera position 1 during the night with raindrops on the lens and one wolf.

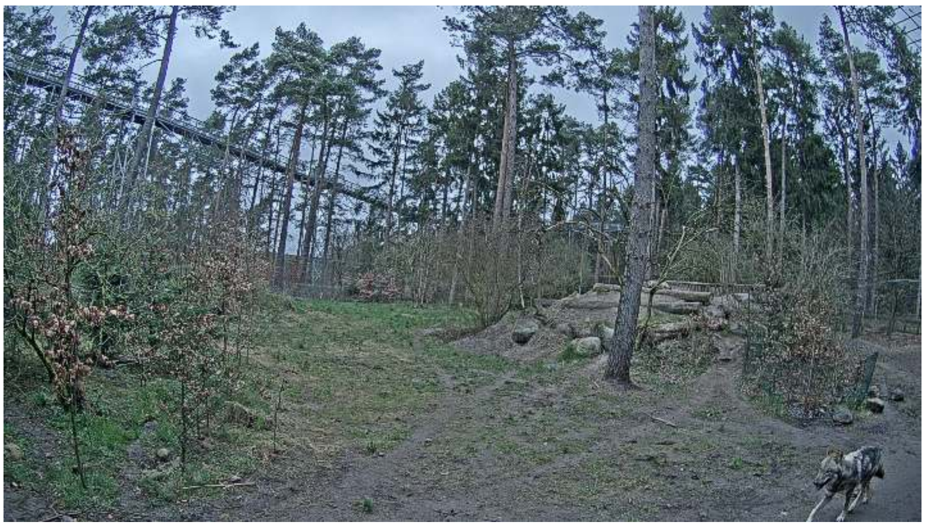


Figure B.8: Wildpark, camera position 2 during the day with one clearly visible and one hidden wolf.

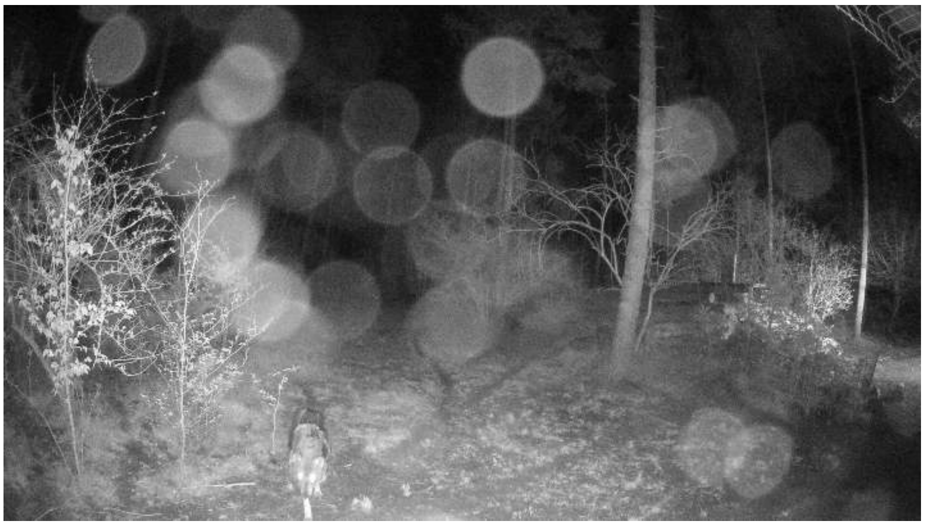


Figure B.9: Wildpark, camera position 2 during the night with one wolf approaching the camera with some raindrops on the lens.

B.3 Wingster Waldzoo

At the Wingster Waldzoo, we collected most of our images over several weeks.

Location	Wingster Waldzoo, Am Olymp 1, 21789 Wingst
Animals	Five European Grey Wolves
Data collection	2023-01-11 - 2023-02-02
Number of images	63212
Size	91.5 GB
Night images	10121
Day images	53091
Camera positions	1

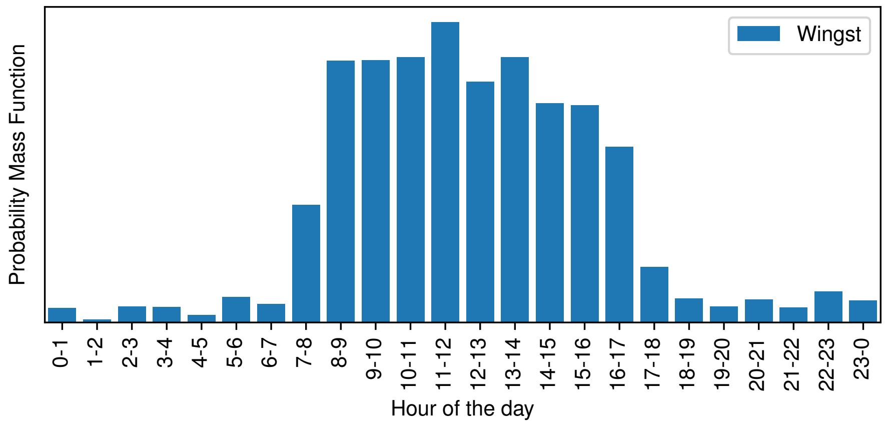


Figure B.10: The distribution of the images collected in Wingst over the hour of the day.

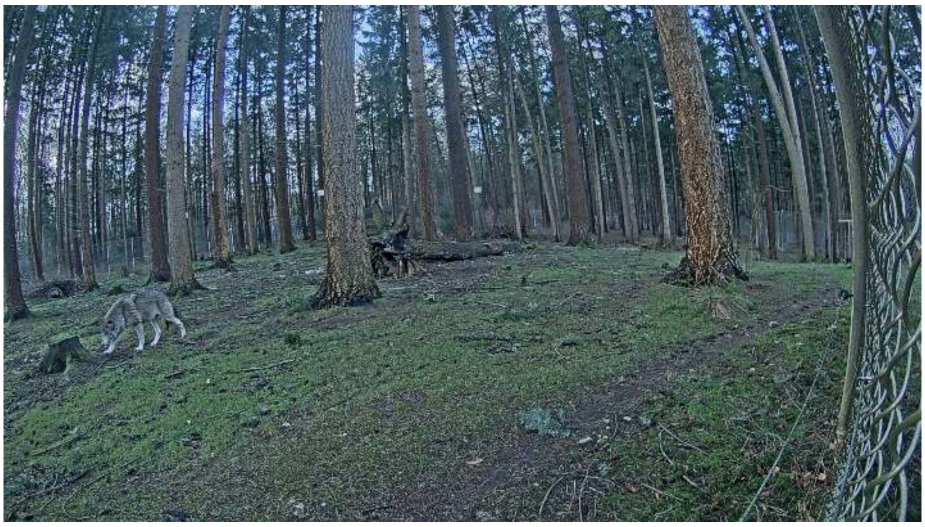


Figure B.11: A picture from Wingst during the day showing one wolf.

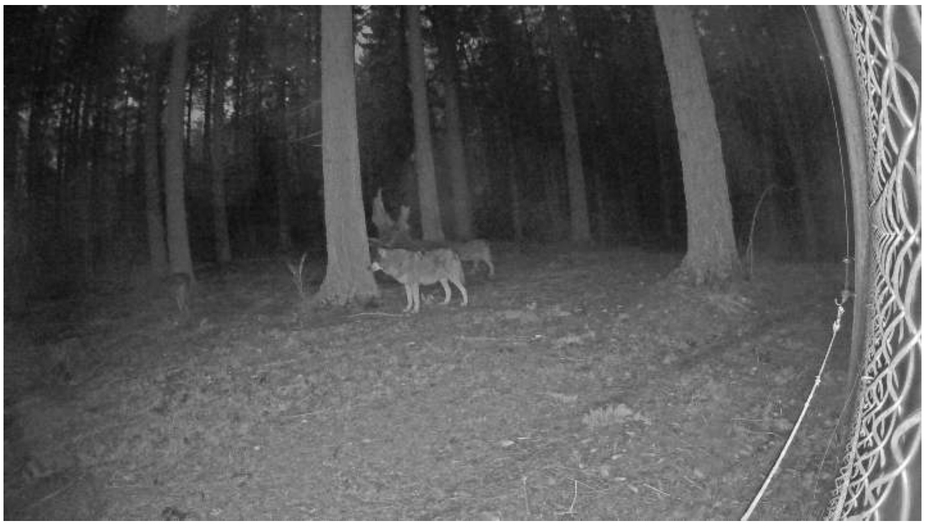


Figure B.12: A picture from Wingst during the night showing three wolves.

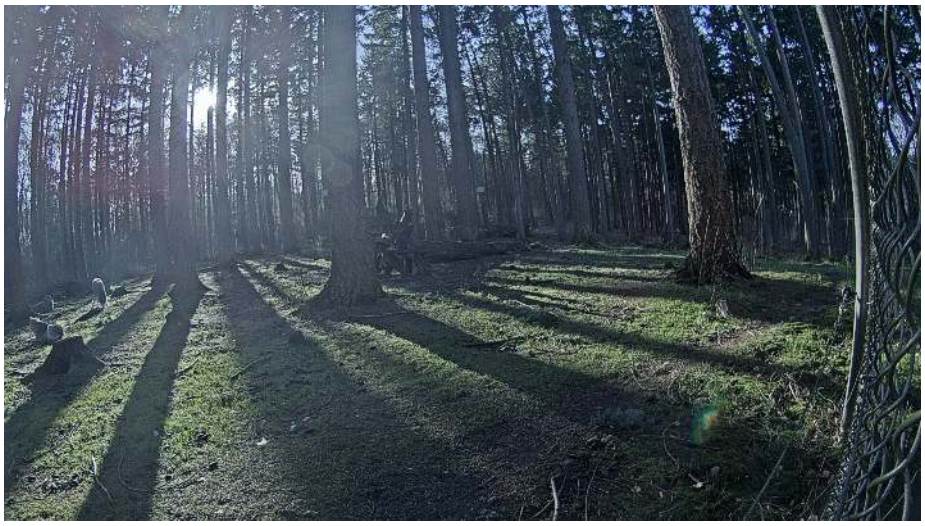


Figure B.13: A picture from Wingst during the night showing three wolves in contre jour.

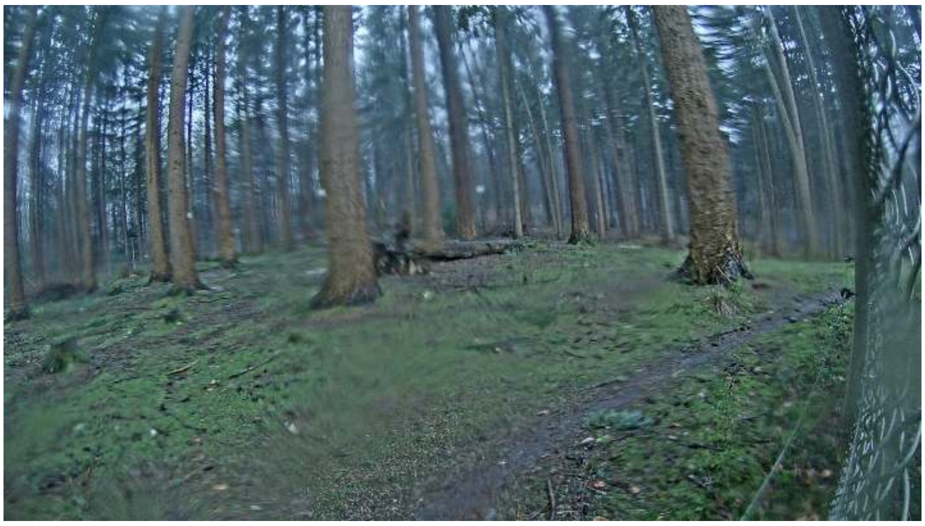


Figure B.14: A picture from Wingst during the day with a wet lens.

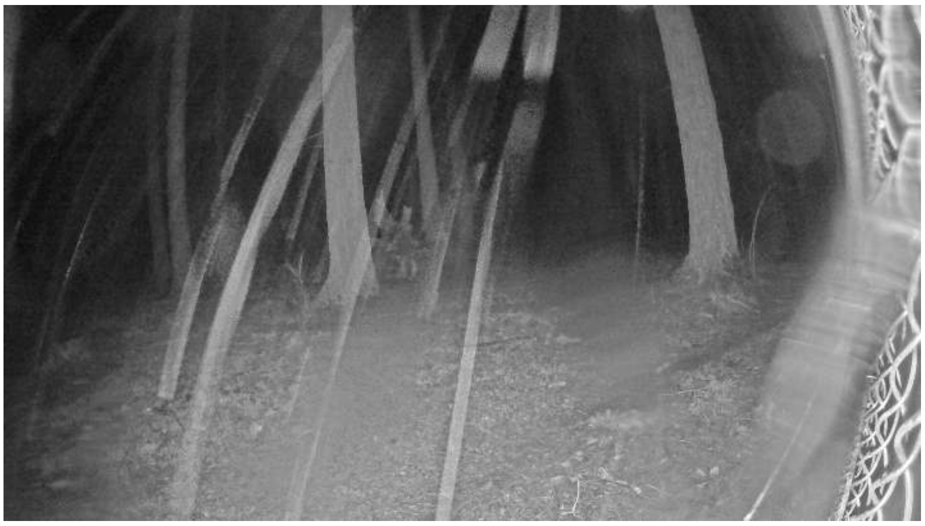


Figure B.15: A picture from Wingst during the night while it rains.

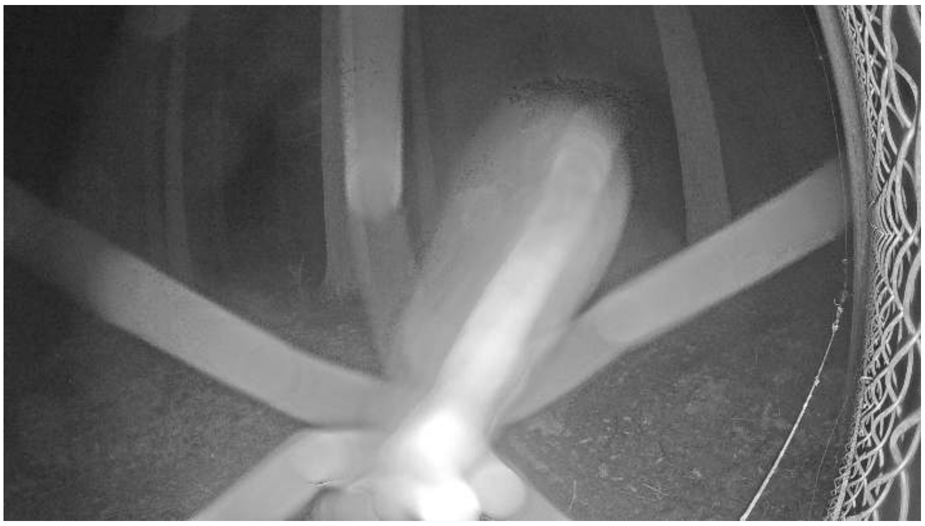


Figure B.16: A picture from Wingst during the night with an insect in front of the lens.

Appendix C

ShadowWolf Configurations

This Appendix lists the *ShadowWolf* configurations used in this work.

C.1 Complete *ShadowWolf* Configuration

Listing C.1 shows the `config.json` with all currently available options of *ShadowWolf* enabled.

```
{
  "main_config": {
    "image_dir"   : "/home/jd/src/comnets-github/mAInZaun/ShadowWolf/
        ground_truth/reference",
    "image_filetype": "jpg"
  },
  "modules": [
    {
      "name": "Analysis.BasicAnalysis.BasicAnalysisClass"
    },
    {
      "name": "Batching.TimeBatching.TimeBatchingClass",
      "exif_time_source": "DateTime",
      "max_timediff_s": 5
    },
    {
      "name": "Preprocessing.NullProcessing.NullPreprocessingClass"
    },
    {
      "name": "Segmentation.MOG2Segmentation.MOG2Class",
      "segments_dir": "segments",
      "extra_dir": "extra_images",
      "segmentation_min_area": 0.0001,
      "segmentation_grey_limit": 10,
      "segmentation_extend_boxes": 90,
      "segmentation_detector_history": 0,
      "segmentation_detector_varThreshold": 60,
      "segmentation_detector_detectShadows": false,
      "segmentation_detector_min_wh": 50,
      "average_image_percentage": 20,
      "average_image_min_images": 5,
      "inputs": [
        {
          "dataclass": "Storage.DataStorage.BatchingDataStorage",
          "getter": "get_batches"
        }
      ]
    },
    {
      "name": "Detection.YoloDetection.YoloDetectionClass",
      "detect_model": "best.pt",
      "detect_repository": "ultralytics/yolov5",
      "detect_force_reload": false,
      "detect_batchsize": 4,
      "inputs": [
        {
          "dataclass": "Storage.DataStorage.SegmentDataStorage",
          "getter": "get_segments"
        },{
          "dataclass": "Storage.DataStorage.BasicAnalysisDataStorage",
          "getter": "get_all_images"
        }
      ]
    },
    {
      "name": "Deduplication.Imagededup.ImagededupClass",
      "inputs": [
        {
          "dataclass": "Storage.DataStorage.SegmentDataStorage",
          "getter": "get_segments"
        },
        {
          "dataclass": "Storage.DetectionStorage.DetectionStorage",
          "getter": "get_cut_images"
        }
      ]
    },
    {
      "name": "Evaluation.SimpleLabelEvaluation.SimpleLabelEvaluationClass",
      "inputs": [
        {
          "dataclass": "Storage.DetectionStorage.DetectionStorage",
          "getter": "get_cut_images"
        },
        {
          "dataclass": "Storage.DataStorage.SegmentDataStorage",
          "getter": "get_segments"
        }
      ],
      "duplicates": {
        "dataclass": "Storage.DuplicateStorage.DuplicateImageStorage",
        "getter": "get_main_similar"
      }
    },
    {
      "name": "Backmapping.SimpleBackmapping.BackmappingClass",
      "inputs": [
        {
          "dataclass": "Storage.SimpleLabelStorage.SimpleLabelStorage",
          "getter": "get_images"
        }
      ],
      "duplicates": {
        "dataclass": "Storage.DuplicateStorage.DuplicateImageStorage",
        "getter": "get_similar"
      }
    },{
      "name":"Decision.WeightedDecision.WeightedDecisionClass",
      "iou_threshold": 0.2,
      "weights" : {"Segment": 0.6, "Detection": 0.4},
      "ignore_classes_higher" : 200,
      "box_combine_method" : "bbox_smaller_box"
    },{
      "name": "Generators.YoloExportGenerator.YoloExportClass",
      "detection_threshold": 0.5
    },{
      "name": "Generators.ReviewExportGenerator.ReviewExportClass"
    }
  ]
}
```

Listing C.1: Complete *ShadowWolf* configuration with all options enabled.

C.2 *ShadowWolf* with YOLO

Listing C.2 shows the `config.json` running only the YOLO-part of *ShadowWolf*. No segmentation or evaluation using *Wolf-or-Not* is performed.

```
{
  "main_config": {
    "image_dir"    : "/home/jd/src/comnets-github/mAInZaun/ShadowWolf/
         ground_truth/reference",
    "image_filetype": "jpg"
  },
  "modules": [
    {
      "name": "Analysis.BasicAnalysis.BasicAnalysisClass"
    },
    {
      "name": "Detection.YoloDetection.YoloDetectionClass",
      "detect_model": "best.pt",
      "detect_repository": "ultralytics/yolov5",
      "detect_force_reload": false,
      "detect_batchsize": 4,
      "inputs": [
        {
          "dataclass": "Storage.DataStorage.BasicAnalysisDataStorage",
          "getter": "get_all_images"
        }
      ]
    },
    {
      "name": "Backmapping.SimpleBackmapping.BackmappingClass",
      "inputs": [
        {
          "dataclass": "Storage.DetectionStorage.DetectionStorage",
          "getter": "get_images"
        }
      ]
    },{
      "name":"Decision.WeightedDecision.WeightedDecisionClass",
      "iou_threshold": 0.2,
      "weights" : {"Segment": 0.6, "Detection": 0.4},
      "ignore_classes_higher" : 200,
      "box_combine_method" : "bbox_smaller_box"
    },{
      "name": "Generators.YoloExportGenerator.YoloExportClass",
      "detection_threshold": 0.5
    },{
      "name": "Generators.ReviewExportGenerator.ReviewExportClass"
    }
  ]
}
```

Listing C.2: *ShadowWolf* configuration with only the YOLO detection enabled.

C.3 *ShadowWolf* with YOLO and Segmentation

Listing C.3 shows the `config.json` with the YOLO-part and the segmentation of *ShadowWolf* enabled. No evaluation using *Wolf-or-Not* is performed.

```
{
  "main_config": {
    "image_dir"    : "/home/jd/src/comnets-github/mAInZaun/ShadowWolf/
        ground_truth/reference",
    "image_filetype": "jpg"
  },
  "modules": [
    {
      "name": "Analysis.BasicAnalysis.BasicAnalysisClass"
    },
    {
      "name": "Batching.TimeBatching.TimeBatchingClass",
      "exif_time_source": "DateTime",
      "max_timediff_s": 5
    },
    {
      "name": "Preprocessing.NullProcessing.NullPreprocessingClass"
    },
    {
      "name": "Segmentation.MOG2Segmentation.MOG2Class",
      "segments_dir": "segments",
      "extra_dir": "extra_images",
      "segmentation_min_area": 0.0001,
      "segmentation_grey_limit": 10,
      "segmentation_extend_boxes": 90,
      "segmentation_detector_history": 0,
      "segmentation_detector_varThreshold": 60,
      "segmentation_detector_detectShadows": false,
      "segmentation_detector_min_wh": 50,
      "average_image_percentage": 20,
      "average_image_min_images": 5,
      "inputs": [
        {
          "dataclass": "Storage.DataStorage.BatchingDataStorage",
          "getter": "get_batches"
        }
      ]
    },
    {
      "name": "Detection.YoloDetection.YoloDetectionClass",
      "detect_model": "best.pt",
      "detect_repository": "ultralytics/yolov5",
      "detect_force_reload": false,
      "detect_batchsize": 4,
      "inputs": [
        {
          "dataclass": "Storage.DataStorage.SegmentDataStorage",
          "getter": "get_segments"
        },
        {
          "dataclass": "Storage.DataStorage.BasicAnalysisDataStorage",
          "getter": "get_all_images"
        }
      ]
    },
    {
      "name": "Backmapping.SimpleBackmapping.BackmappingClass",
      "inputs": [
        {
          "dataclass": "Storage.DetectionStorage.DetectionStorage",
          "getter": "get_images"
        }
      ]
    },{
      "name":"Decision.WeightedDecision.WeightedDecisionClass",
      "iou_threshold": 0.2,
      "weights" : {"Segment": 0.6, "Detection": 0.4},
      "ignore_classes_higher" : 200,
      "box_combine_method" : "bbox_smaller_box"
    },{
      "name": "Generators.YoloExportGenerator.YoloExportClass",
      "detection_threshold": 0.5
    },{
      "name": "Generators.ReviewExportGenerator.ReviewExportClass"
    }
  ]
}
```

Listing C.3: *ShadowWolf* configuration with the YOLO detection and the segmentation enabled.

Appendix D

Hardware for WolfNet

The deterrents require a higher voltage and draw a higher current as it can be delivered by a standard microcontroller. For that, we designed a circuit with three main tasks: 1) activate deterrents requiring 12 V and higher currents, 2) use a 12 V (car) battery and supply the microcontroller with the required 5 V, and 3), measure the voltage and thus the remaining capacity of the battery. The assembled box is depicted in Figure 9.6 on page 109.

This appendix lists the main technical details of the design. We also give the list of the used components and their function as well as the schematic. The most recent version is available in the *WolfNet* GitHub repository:
`https://github.com/ComNets-Bremen/WolfNet/`

D.1 Technical Parameters

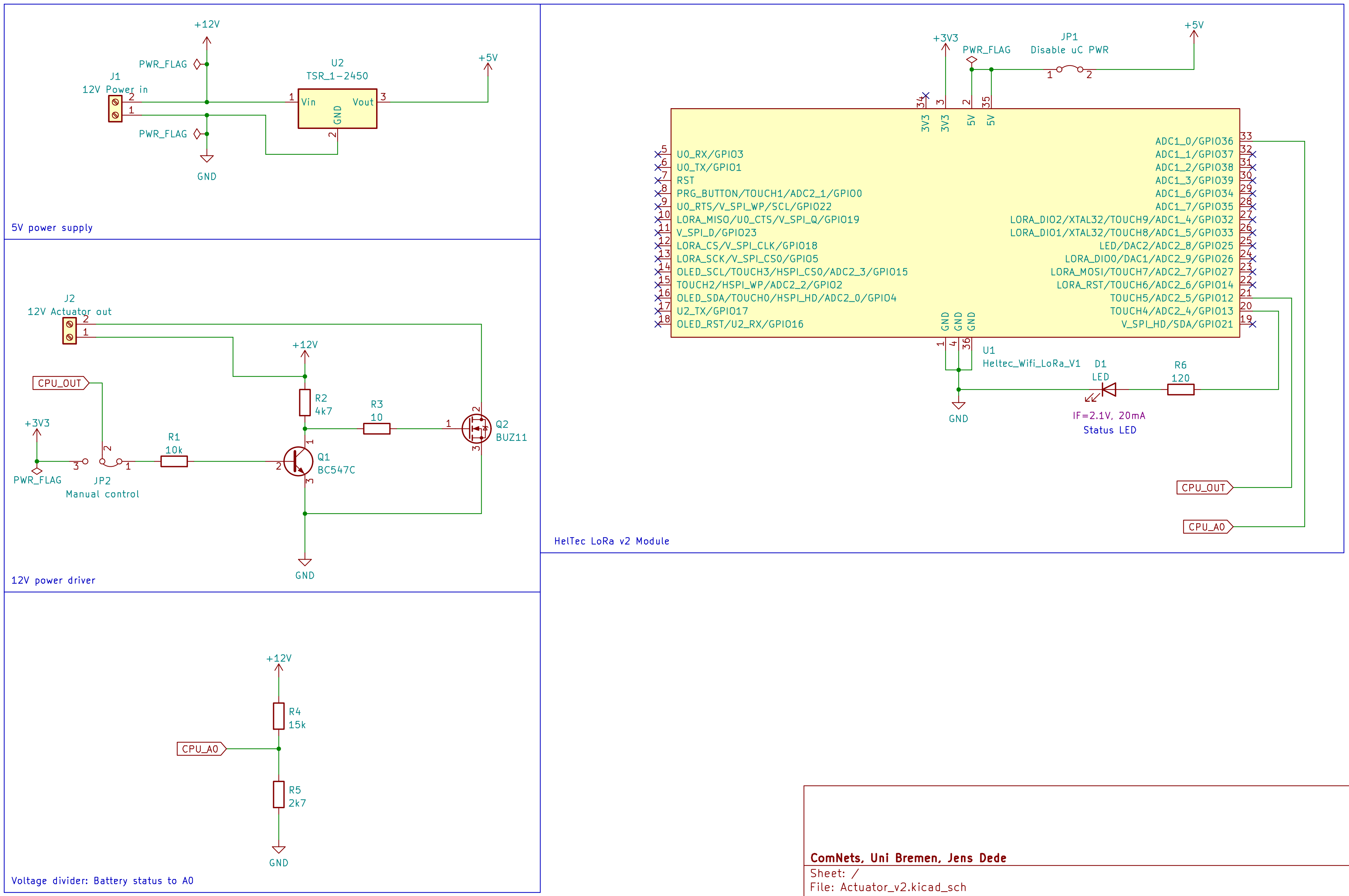
Input voltage	Typ. 12 V (11 V to 14.4 V)
Input current	Max. 11 A
Actuator output voltage	Equals input voltage, typ. 12 V
Actuator output current	Max. 10 A

D.2 Bill of Materials

Ref	**Value**	**Comment**
D1	LED	Status LED
J1	Input 12 V	Power supply
J2	Output, 12 V, 12 A	Connection for actuators
JP1	Power microcontroller	5 V power for microcontroller
JP2	Manual control	Disable actuator permanently
Q1	BC547C	Preamp for actuator MOSFET Q2
Q2	BUZ11	MOSFET power for actuator
R1	10 kΩ	Power control
R2	4.7 kΩ	Power control
R3	10 Ω	Power control
R4	15 kΩ	Voltage divider for battery status
R5	2.7 kΩ	Voltage divider for battery status
R6	120 Ω	Series resistance for LED
U1	Heltec WiFi LoRa 32	Microcontroller and LoRa
U2	TSR 1-2450	DC-DC converter, 6.5 V to 36 V in, 5 V out

D.3 Schematics of the Actuator Box

The following page shows the schematic of the actuator box.



ComNets, Uni Bremen, Jens Dede

Sheet: /

File: Actuator_v2.kicad_sch

Title: Actuator WolfNet

Size: A4	Date: 2022-08-04	Rev: 2
KiCad E.D.A. kicad 7.0.9+dfsg-1		Id: 1/1

Bibliography

[1] N. Schoof, A. Reif, R. Luick, E. Jedicke, G. Kämmer, and J. Metzner, "Der Wolf in Deutschland – Herausforderungen für weidebasierte Tierhaltungen und den praktischen Naturschutz," *Naturschutz und Landschaftsplanung: Zeitschrift für angewandte Ökologie*, 2021.

[2] Council of European Union, "Council regulation (EU) no 92/43/eec," 1992, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:31992L0043.

[3] Council of Europe, "European treaty series - no. 104: Convention on the conservation of european wildlife and natural habitats," 1979, https://rm.coe.int/1680078aff.

[4] Council of European Union, "Council regulation (EU) no 58/1998: concerning the protection of animals kept for farming purposes," 1998, http://data.europa.eu/eli/dir/1998/58/2019-12-14, Retrieved: 2023-05-09.

[5] Bundesministerium der Justiz und für Verbraucherschutz, "Verordnung zum Schutz landwirtschaftlicher Nutztiere und anderer zur Erzeugung tierischer Produkte gehaltener Tiere bei ihrer Haltung (Tierschutz-Nutztierhaltungsverordnung - TierSchNutztV),
§ 3 Allgemeine Anforderungen an Haltungseinrichtungen," available online: https://www.gesetze-im-internet.de/tierschnutztv/__3.html, Retrieved: 2023-05-09.

[6] Landwirtschaftskammer Nordrhein-Westfalen, "Checkliste Cross Compliance 2022 für landwirtschaftliche Unternehmen in Nordrhein-Westfalen," 2022, accessed: 2023-05-05. [Online]. Available: https://www.landwirtschaftskammer.de/landwirtschaft/beratung/pdf/cross-compliance-checkliste-nrw.pdf

[7] P. Mäder, D. Boho, M. Rzanny, M. Seeland, H. C. Wittich, A. Deggelmann, and J. Wäldchen, "The flora incognita app–interactive plant species identification," *Methods in Ecology and Evolution*, vol. 12, no. 7, pp. 1335–1342, 2021.

[8] H. L. Larsen, C. Pertoldi, N. Madsen, E. Randi, A. V. Stronen, H. Root-Gutteridge, and S. Pagh, "Bioacoustic detection of wolves: identifying subspecies and individuals by howls," *Animals*, vol. 12, no. 5, p. 631, 2022.

[9] D. E. Swann, C. C. Hass, D. C. Dalton, and S. A. Wolf, “Infrared-triggered cameras for detecting wildlife: an evaluation and review,” *Wildlife Society Bulletin*, vol. 32, no. 2, pp. 357–365, 2004.

[10] A. R. Elias, N. Golubovic, C. Krintz, and R. Wolski, “Where’s the bear? automating wildlife image processing using iot and edge cloud systems,” in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, ser. IoTDI ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 247–258. [Online]. Available: https://doi.org/10.1145/3054977.3054986

[11] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1512.00567

[12] M. S. Norouzzadeh, A. Nguyen, M. Kosmala, A. Swanson, M. S. Palmer, C. Packer, and J. Clune, “Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning,” *Proceedings of the National Academy of Sciences*, vol. 115, no. 25, pp. E5716–E5725, 2018.

[13] A. Swanson, M. Kosmala, C. Lintott, R. Simpson, A. Smith, and C. Packer, “Data from: Snapshot serengeti, high-frequency annotated camera trap images of 40 mammalian species in an african savanna,” 2015. [Online]. Available: https://doi.org/10.5061/dryad.5pt92

[14] M. S. Norouzzadeh, D. Morris, S. Beery, N. Joshi, N. Jojic, and J. Clune, “A deep active learning system for species identification and counting in camera trap images,” *Methods in ecology and evolution*, vol. 12, no. 1, pp. 150–161, 2021.

[15] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1512.03385

[16] A. Singh, M. Pietrasik, G. Natha, N. Ghouaiel, K. Brizel, and N. Ray, “Animal detection in man-made environments,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 1438–1449.

[17] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1708.02002

[18] B. Kellenberger, D. Marcos, S. Lobry, and D. Tuia, “Half a percent of labels is enough: Efficient animal detection in uav imagery using deep cnns and active learning,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 12, pp. 9524–9533, 2019.

[19] M. A. Tabak, M. S. Norouzzadeh, D. W. Wolfson, S. J. Sweeney, K. C. VerCauteren, N. P. Snow, J. M. Halseth, P. A. Di Salvo, J. S. Lewis, M. D. White *et al.*, “Machine learning to classify animal species in camera trap images: Applications in ecology,” *Methods in Ecology and Evolution*, vol. 10, no. 4, pp. 585–590, 2019.

[20] A. M. Roy, J. Bhaduri, T. Kumar, and K. Raj, “Wildect-yolo: An efficient and robust computer vision-based accurate object localization model for automated endangered wildlife detection,” *Ecological Informatics*, vol. 75, p. 101919, 2023.

[21] S. Beery, D. Morris, and S. Yang, “Efficient pipeline for camera trap image review,” *arXiv preprint arXiv:1907.06772*, 2019.

[22] M. Ivašić-Kos, M. Krišto, and M. Pobar, “Human detection in thermal imaging using yolo,” in *Proceedings of the 2019 5th International Conference on Computer and Technology Applications*, 2019, pp. 20–24.

[23] R. Ippalapally, S. H. Mudumba, M. Adkay, and N. V. HR, “Object detection using thermal imaging,” in *2020 IEEE 17th India Council International Conference (INDICON).* IEEE, 2020, pp. 1–6.

[24] J. Cilulko, P. Janiszewski, M. Bogdaszewski, and E. Szczygielska, “Infrared thermal imaging in studies of wild animals,” *European Journal of Wildlife Research*, vol. 59, pp. 17–23, 2013.

[25] M. Meyer and G. Kuschk, “Automotive radar dataset for deep learning based 3d object detection,” in *2019 16th European Radar Conference (EuRAD)*, 2019, pp. 129–132.

[26] R. Pérez, F. Schubert, R. Rasshofer, and E. Biebl, “Single-frame vulnerable road users classification with a 77 ghz fmcw radar sensor and a convolutional neural network,” in *2018 19th International Radar Symposium (IRS)*, 2018, pp. 1–10.

[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[28] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll’a r, and C. L. Zitnick, “Microsoft COCO: common objects in context,” 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[29] A. Barbu, D. Mayo, J. Alverio, W. Luo, C. Wang, D. Gutfreund, J. Tenenbaum, and B. Katz, “Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models,” *Advances in neural information processing systems*, vol. 32, 2019.

[30] A. Su, H. Lee, X. Tan, C. J. Suarez, N. Andor, Q. Nguyen, and H. P. Ji, “A deep learning model for molecular label transfer that enables cancer cell identification from histopathology images,” *NPJ precision oncology*, vol. 6, no. 1, p. 14, 2022.

[31] M. Geiß, R. Wagner, M. Baresch, J. Steiner, and M. Zwick, “Automatic bounding box annotation with small training datasets for industrial manufacturing,” *Micromachines*, vol. 14, no. 2, p. 442, 2023.

[32] M. Längkvist, M. Alirezaie, A. Kiselev, and A. Loutfi, "Interactive learning with convolutional neural networks for image labeling," in *International Joint Conference on Artificial Intelligence (IJCAI), New York, USA, 9-15th July, 2016*, 2016.

[33] M. Xu, Z. Zhang, H. Hu, J. Wang, L. Wang, F. Wei, X. Bai, and Z. Liu, "End-to-end semi-supervised object detection with soft teacher," *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[34] Y. Zhang, Y. Wang, H. Zhang, B. Zhu, S. Chen, and D. Zhang, "Onelabeler: A flexible system for building data labeling tools," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–22.

[35] J. C. Chang, S. Amershi, and E. Kamar, "Revolt: Collaborative crowdsourcing for labeling machine learning datasets," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, pp. 2334–2346.

[36] T. Götz and V. M. Janik, "Repeated elicitation of the acoustic startle reflex leads to sensitisation in subsequent avoidance behaviour and induces fear conditioning," *BMC neuroscience*, vol. 12, no. 1, pp. 1–13, 2011.

[37] Z. A. Schakner and D. T. Blumstein, "Behavioral biology of marine mammal deterrents: a review and prospectus," *Biological Conservation*, vol. 167, pp. 380–389, 2013.

[38] J. Blackshaw, G. Cook, P. Harding, C. Day, W. Bates, J. Rose, and D. Bramham, "Aversive responses of dogs to ultrasonic, sonic and flashing light units," *Applied Animal Behaviour Science*, vol. 25, no. 1-2, pp. 1–8, 1990.

[39] S. W. Breck, R. Williamson, C. Niemeyer, and J. A. Shivik, "Non-lethal radio activated guard for deterring wolf depredation in idaho: summary and call for research," in *Proceedings of the Vertebrate Pest Conference*, vol. 20, no. 20, 2002.

[40] P. A. Darrow and J. A. Shivik, "Bold, shy, and persistent: Variable coyote response to light and sound stimuli," *Applied Animal Behaviour Science*, vol. 116, no. 1, pp. 82–87, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168159108001901

[41] J. A. Shivik, A. Treves, and P. Callahan, "Nonlethal techniques for managing predation: primary and secondary repellents," *Conservation Biology*, vol. 17, no. 6, pp. 1531–1537, 2003.

[42] S. A. Stone, S. W. Breck, J. Timberlake, P. M. Haswell, F. Najera, B. S. Bean, and D. J. Thornhill, "Adaptive use of nonlethal strategies for minimizing wolf–sheep conflict in idaho," *Journal of Mammalogy*, vol. 98, no. 1, pp. 33–44, 2017.

[43] A. Förster, J. Dede, A. Könsgen, K. Kuladinithi, V. Kuppusamy, A. Timm-Giel, A. Udugama, and A. Willig, “A beginner’s guide to infrastructure-less networking concepts,” *IET Networks*, 2023. [Online]. Available: https://doi.org/10.1049/ntw2.12094

[44] A. Forster, *Introduction to wireless sensor networks.* John Wiley & Sons, 2016.

[45] C.-Y. Chong and S. P. Kumar, “Sensor networks: evolution, opportunities, and challenges,” *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247–1256, 2003.

[46] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, “Transmission of IPv6 packets over IEEE 802.15. 4 networks,” 2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4944

[47] M. Becker, *Services in wireless sensor networks: modelling and optimisation for the efficient discovery of services.* Springer Science & Business Media, 2014.

[48] IEEE Computer Society, “Ieee standard for low-rate wireless networks,” *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, 2016.

[49] P. Rawat, M. Haddad, and E. Altman, “Towards efficient disaster management: 5g and device to device communication,” in *2015 2nd International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*, 2015, pp. 79–87.

[50] A. F. M. S. Shah, “Architecture of emergency communication systems in disasters through uavs in 5g and beyond,” *Drones*, vol. 7, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2504-446X/7/1/25

[51] S. Ahmed, M. Rashid, F. Alam, and B. Fakhruddin, “A disaster response framework based on iot and d2d communication under 5g network technology,” in *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)*, 2019, pp. 1–6.

[52] A. M. Cardenas, M. K. Nakamura Pinto, E. Pietrosemoli, M. Zennaro, M. Rainone, and P. Manzoni, “A low-cost and low-power messaging system based on the lora wireless technology,” *Mobile networks and applications*, vol. 25, pp. 961–968, 2020.

[53] E. F. Rivera Guzmán, E. D. Mañay Chochos, M. D. Chiliquinga Malliquinga, P. F. Baldeón Egas, and R. M. Toasa Guachi, “Lora network-based system for monitoring the agricultural sector in andean areas: Case study ecuador,” *Sensors*, vol. 22, no. 18, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/18/6743

[54] V. Križanović, K. Grgić, J. Spišić, and D. Žagar, “An advanced energy-efficient environmental monitoring in precision agriculture using lora-based wireless sensor networks,” *Sensors*, vol. 23, no. 14, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/14/6332

[55] M. Ragnoli, A. Leoni, G. Barile, G. Ferri, and V. Stornelli, "Lora-based wireless sensors network for rockfall and landslide monitoring: A case study in pantelleria island with portable lorawan access," *Journal of Low Power Electronics and Applications*, vol. 12, no. 3, 2022. [Online]. Available: https://www.mdpi.com/2079-9268/12/3/47

[56] M. Ragnoli, G. Barile, A. Leoni, G. Ferri, A. Pelliccione, V. Stornelli, and D. Del Tosto, "Lora-based wireless sensor network system for aquatic elements and flood early warning monitoring," in *Sensors and Microsystems*, G. Di Francia and C. Di Natale, Eds. Cham: Springer Nature Switzerland, 2023, pp. 124–128.

[57] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[58] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[59] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," 2019.

[60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1409.1556

[61] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1506.01497

[62] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[63] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[64] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation," 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.05587

[65] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440. [Online]. Available: https://doi.org/10.48550/arXiv.1411.4038

[66] R. Rojas, *Neural Networks – A systematic introduction*. Springer, 1996. [Online]. Available: https://doi.org/10.1007/978-3-642-61068-4

[67] G. Paaß and D. Hecker, *Künstliche Intelligenz: was steckt hinter der Technologie der Zukunft?* Springer, 2020.

[68] T. Rashid, *Make your own neural network.* CreateSpace Independent Publishing Platform North Charleston, SC, USA, 2016, vol. 29.

[69] ——, *Neuronale Netze selbst programmieren: ein verständlicher Einstieg mit Python.* O'Reilly, 2017.

[70] R. Li, Y. Wang, F. Liang, H. Qin, J. Yan, and R. Fan, "Fully quantized network for object detection," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2805–2814.

[71] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[72] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2017.

[73] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[74] T. Das, R.-J. Bruintjes, A. Lengyel, J. van Gemert, and S. Beery, "Domain adaptation for rare classes augmented with synthetic samples," *arXiv preprint arXiv:2110.12216*, 2021.

[75] Y. Wang, K. Kitani, and X. Weng, "Joint object detection and multi-object tracking with graph neural networks," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 13 708–13 715.

[76] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[77] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training deep neural networks on imbalanced data sets," in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 4368–4374.

[78] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[79] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.

[80] H. Jin, F. Chollet, Q. Song, and X. Hu, “Autokeras: An automl library for deep learning,” *Journal of Machine Learning Research*, vol. 24, no. 6, pp. 1–6, 2023. [Online]. Available: http://jmlr.org/papers/v24/20-1355.html

[81] CVAT.ai Corporation, “Computer Vision Annotation Tool (CVAT),” Sep. 2022. [Online]. Available: https://github.com/opencv/cvat

[82] Tzutalin, “Labelimg,” Free Software: MIT License, 2015. [Online]. Available: https://github.com/tzutalin/labelImg

[83] A. Zourmand, A. L. Kun Hing, C. Wai Hung, and M. AbdulRehman, “Internet of things (iot) using lora technology,” in *2019 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, 2019, pp. 324–330.

[84] LoRa Alliance, “A technical overview of LoRa and LoRaWAN,” *White Paper*, November 2015. [Online]. Available: https://www.tuv.com/content-media-files/master-content/services/products/1555-tuv-rheinland-lora-alliance-certification/tuv-rheinland-lora-alliance-certification-overview-lora-and-lorawan-en.pdf

[85] Semtech Corporation, “LoRa and LoRaWAN: A technical overview,” *White Paper*, December 2019. [Online]. Available: https://lora-developers.semtech.com/uploads/documents/files/LoRa_and_LoRaWAN-A_Tech_Overview-Downloadable.pdf

[86] E. ETSI, “Etsi en 300 220-2 v3. 2.1-short range devices (srd) operating in the frequency range 25 mhz to 1 000 mhz; part 2: Harmonised standard for access to radio spectrum for non specific radio equipment.”

[87] B. S. Chaudhari and M. Zennaro, *LPWAN Technologies for IoT and M2M Applications*. Academic Press, 2020.

[88] T. Karunathilake and A. Förster, “Using lora communication for urban vanets: Feasibility and challenges,” 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2311.18070

[89] D. Vasconcelos, M. S. Yin, F. Wetjen, A. Herbst, T. Ziemer, A. Förster, T. Barkowsky, N. Nunes, and P. Haddawy, “Counting mosquitoes in the wild: An internet of things approach,” in *Proceedings of the Conference on Information Technology for Social Good*, ser. GoodIT ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 43–48. [Online]. Available: https://doi.org/10.1145/3462203.3475914

[90] J. Dede, D. Wewetzer, and A. Förster, “AI in the Wild: Challenges of Remote Deployments,” in *INFORMATIK 2023 - Designing Futures: Zukünfte gestalten*. Bonn: Gesellschaft für Informatik e.V., 2023, pp. 1583–1589. [Online]. Available: https://doi.org/10.18420/inf2023_161

[91] J. Dede and A. Förster, “ShadowWolf – Automatic Labelling, Evaluation and Model Training Optimised for Camera Trap Wildlife Images,” *Under Review*, 2024.

[92] International Organization for Standardization, "Iso 20653:2013-02: Road vehicles – degrees of protection (ip code) – protection of electrical equipment against foreign objects, water and access," 02 2013.

[93] European Committee for Electrotechnical Standardization, "Degrees of protection provided by enclosures (ip code)," 10 2013.

[94] J. Dede and A. Förster, "Animals in the Wild: Using Crowdsourcing to Enhance the Labelling of Camera Trap Images," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, 2023, pp. 748–754.

[95] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.

[96] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 2. IEEE, 2004, pp. 28–31.

[97] T. Jain, C. Lennan, Z. John, and D. Tran, "Imagededup," https://github.com/idealo/imagededup, 2019.

[98] J. Moosmann, M. Giordano, C. Vogt, and M. Magno, "Tinyissimoyolo: A quantized, low-memory footprint, tinyml object detection network for low power microcontrollers," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, Jun. 2023. [Online]. Available: http://dx.doi.org/10.1109/AICAS57966.2023.10168657

[99] E. J. Rickley, G. G. Fleming, and C. J. Roof, "Simplified procedure for computing the absorption of sound by the atmosphere," *Noise control engineering journal*, vol. 55, no. 6, pp. 482–494, 2007.

[100] Organización Internacional de Normalización, *ISO 9613-1: 1993, Acoustics: Attenuation of sound during propagation outdoors. Part 1: Calculation of the absorption of sound by the atmosphere.* International Organization for Standardization, 1993.

[101] ——, *ISO 9613-2: 1996, Acoustics: Attenuation of sound during propagation outdoors. Part 2: General method of calculation.* International Organization for Standardization, 1993.