Universiteit Leiden
The Netherlands

Opleiding Informatica

Monitoring wildlife of the Oostvaardersplassen through trap-camera and computer vision

Kenneth Dirker

Supervisors:
Mitra Baratchi & Nuno César de Sa

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

30/12/2021

Abstract

The Oostvaardersplassen is a nature reserve in the Netherlands and is one of the 4 big examples of rewilding conservation projects in the world. Cattle, horses and deers were introduced to naturally manage the grassland biomass, but their populations have grown due to the lack of predators. The Staatsbosbeheer manages the population levels using a nature-driven approach, which means culling according to death rates in natural populations, and allowing starvation to occur. In recent years, there has been an increased reaction against this approach and more precise monitoring of the animal population numbers and well-being is required.

Trap-cameras can be used to monitor the animals in the park, minimizing physical monitoring of the target populations, which is one of the main objectives of the rewilding project. Computer Vision is often used together with trap-cameras to automate observation and population counting, removing the need for human work, which often consumes a lot of time and resources. From a photo dataset, captured by 6 trap-cameras located in the Oostvaardersplassen, a sample was taken and labelled, resulting in 2817 images with 21102 labels. These labels were used to train a Faster-R-CNN model to detect horses, cattle, deer and geese: The 4 main grazers of the Oostvaardersplassen. Models were validated using stratified k-fold cross validation with 5 folds. Accuracy was calculated according to the definition provided by the COCO detection challenge.

With a threshold of 0.50, the resulting detector hit a precision and recall around 0.80 for each class. Taking all thresholds into account, a mAP of 0.471 is achieved. We discovered that the detector is good at detecting spread-out animals in the image. The model struggles with dense herds, resulting in wrong and missing localization of the animals. In the future work, better hyper-parameter optimization might result in better performing models. Furthermore, a model trained on a big dataset for animal detection may be used with transfer learning to function as starting point.

Contents

1 Introduction

The Oostvaardersplassen (OVP) [1] is a nature reserve located in the Netherlands. The reserve was created in 1968 when the South Flevoland polder was reclaimed from the Ijssel lake. It is comprised of approximately 60 square kilometers of open water, marshland, wet and dry open grasslands, and areas with trees and shrubs and is often considered one of the 4 big examples of Rewilding conservation [2]. Large herbivores were introduced in the OVP as a nature-oriented solution to manage the vegetation of the park OVP [3]. The lack of predators has created overpopulation problems within the park, which are leading to increasing conflicts between the Staatbosbeheer and the general population [4]. Therefore, there is a need for accurate monitoring of the animal population to assist in decision making and to allow for better and more effective management actions.

By default, Rewilding or nature-oriented conservation approaches aim to minimize the need for physical monitoring of the target population and this implies a need for the development of alternative monitoring systems. Wildlife monitoring is becoming more and more viable as sensor software and hardware become more advanced [5].

Especially trap-cameras have been shown to be useful to monitor animal activity in the wild [6]. Many studies have shown the effectiveness of trap-cameras as a solution to animal detection [7] and animal behaviour observation [8].

Machine Learning (ML) [9] has proven to be an excellent method to extract useful data from datasets and make assumptions based on large amounts of data. ML can be used to detect and segment objects in images and videos, which is applied to the field of Computer Vision (CV) [10]. Especially Deep Learning (DL) [11], an approach that became very popular in the field of CV in the last decade and sprouted many different DL architectures [12, 13, 14, 15], has proven to be a reliable method for object detection. Where other ML methods often need hand-crafted features to detect objects, DL models extract these features by themselves during training, removing the need for extensive DL knowledge to create features and train a DL model on a new dataset. Easy to use ML frameworks like PyTorch [16] and scikit-learn [17] have made object detection available to researchers outside of the field of DL, resulting in an explosion of DL applications.

The field of ecology is such a field in which DL is being used abundantly [18]. DL has been used to automate many tasks, ranging from counting animals in an area to observing whole ecosystems, allowing research from a bigger perspective than previously possible.

Using already manually labelled capture data from trap-cameras in the OVP, a DL model can be trained to detect the grazers of the OVP, removing the need for further human labelling and creating new data that can be applied to other OVP related research. Datasets contain a bias towards certain patterns in the set [19]. The bias of a dataset is always transmitted to models that are trained on the set [20], making it important to reduce any bias as much as possible by filtering the dataset.

Creating a good object detector and efficiently training said detector require good hyper-parameters [21]. A general good hyper-parameter setting does not exist, every learning problem requires a different set of hyper-parameters for an optimal learning process. Thus, in order to train a good detector, good hyper-parameters need to be found first.

Research Goals

The goal of this project is to train a model that can automatically label the grazers of the OVP in trap-camera images, using computer vision methods. The first part of this research will be focused on acquiring, analysing and adjusting the labelled image dataset. The second part of the research will consist of finding a good set of hyper-parameters to train the model with. The third part of the research will be training a model with the found hyper-parameters. To complete this research, the following research questions will be answered:

- What is the composition of our labelled dataset?
- How can we adjust our labelled dataset to improve the training process?
- What hyper-parameters settings optimize the learning process?

2 Related work

Trap-cameras are used to a great extend in ecological research. Bengsen et al. [22] used trap-cameras to estimate feral cat population sizes, successfully showing a decline in population size. Rovero et al. [23] used trap-cameras in rain forests to estimate the number of different mammal species in those regions. McCarthy et al. [8] were able to construct social networks of wild chimpanzees with trap-cameras that were highly similar to social networks created by humans, proving that trap-cameras can also be used for more complex tasks than counting population sizes and different animal species.

Deep learning is often used in unison with trap-camera data to train detectors that achieve great accuracy. Tabak et al. [24] classified 26 different animal species using the ResNet-18 architecture [25]. The classifier was trained on 3,367,383 images taken by trap-cameras in 5 different regions across the United States, achieving an accuracy of 98% on trap-camera images from the United States and achieving accuracies above 82% on out-of-sample trap-camera data from Canada and Tanzania. Although the accuracy of the model is high, the classifier predicts a class for the image as a whole and is not capable of recognizing and labelling multiple classes in a single image, making this approach unfit for our problem, where animals also need to be counted.

Miao et al. [26] used DL to identify 20 different species from a fully annotated dataset of 111,467 images from Gorongosa National Park, Mozambique. The images were gathered by trap-cameras that are spread around the park. Miao et al. went with the 20 most abundant species, as other species had too few pictures and skewed the dataset to be unbalanced, which results in worse detection accuracy. Our dataset is very unbalanced and has about 3000 images, which makes detection of our classes more difficult.

Norouzzadeh et al. [27] identified, counted and described the behaviour of wild animals, using a two-stage pipeline. In the first stage, images are checked for any animal presence. The second stage identifies these animals, counts the number of animals of each class, and tries to specify some behavioural attributes of the animals. Norouzzadeh et al. reported an accuracy of over 93.8%. We will not be using a separate stage for detecting animal presence, since a little less than 25% of our dataset contains empty images, as opposed to the workload in [27]. Furthermore, our research will not be focusing on behaviour detection due to a lack of data on behaviour in our dataset. Lastly,

we will be counting by counting detections, instead of counting with a network specifically trained for that task.

3 Methodology

In this section we will describe the study area, which is the Oostvaardersplassen, and how and where the cameras were set up in the OVP. After that, the methods behind the 4 phases of this research are explained, from which an overview is shown in figure 3.1. In the first phase, we gather the dataset, which is already made for us. Phase two is about analysing the trap-camera data that is available and adjusting the data to be better suited for training object detection models. In the third phase, the hyper-parameters of the neural network optimizer are tuned and in the fourth phase, those hyper-parameters are used to fully train a model.

Figure 3.1: Overview of the 4 phases of this research.

3.1 Study area

The Oostvaardersplassen is an artificial wetland in the Southern Flevoland polder in the Netherlands, located to the east of Amsterdam as shown in figure 3.2. It is comprised of approximately 60 square kilometres of open water, marshland, wet and dry open grasslands and areas with trees and shrubs and is often considered one of the big 4 examples of Rewilding conservation [2]. The reserve was created when the polder was endiked in 1968, creating a marshy landscape. In 1982 additional land was added to the reserve in the form of a dry border, which would later become the home to big herbivores. The different species of herbivores were specifically selected and transferred to graze on certain types of vegetation that needed regulation [28], removing the need for human interaction to keep the ecosystems stable. The natural predators of the herbivores do not roam the OVP, resulting in overpopulation problems [4], which is accompanied by food shortages.
6 trap-cameras were used to capture animal pictures, from which the placement can be seen in figure 3.2. Section 3.2 describes the cameras and their placement in more detail. Cameras E228 and E217 are both located in wetlands, which become submerged in water and feature mint plants when dried up in the summer. Camera D86 is located in grassland and is aimed at a small lake, which is a hotspot for birds, and more interesting, geese. Big grazers also gather here to drink water. Cameras D278 and E287 are also located in grassland, although closer to the border of the OVP. The horses were gathered close to these locations for transport out of the OVP, which caused the horses to graze in these parts, resulting in a landscape with short grass. . Lastly, camera E302 is located in grassland where the big grazers have not grazed upon, resulting in tall grass and thistles.

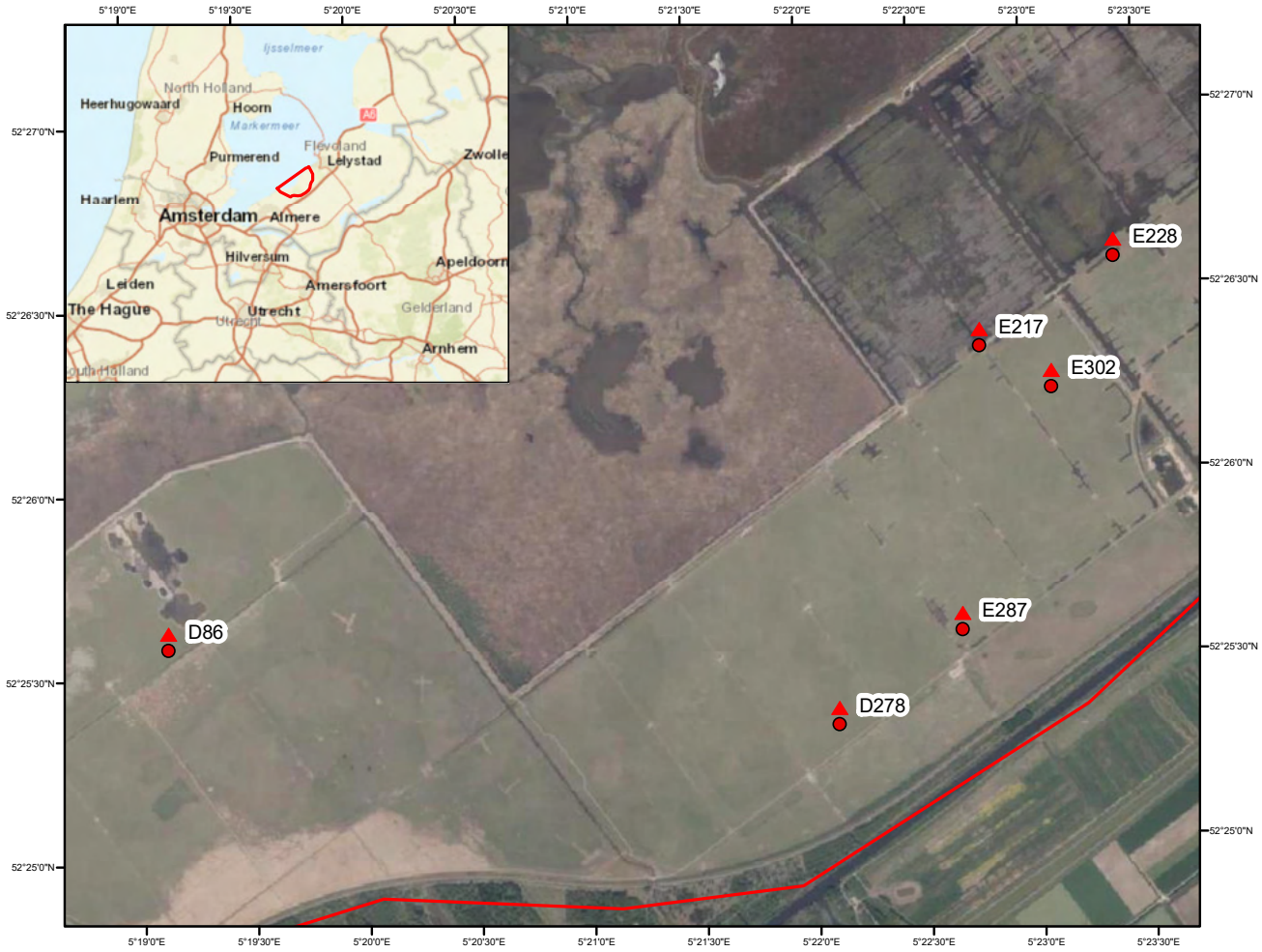


Figure 3.2: Camera locations in the OVP, provided by Nuno Cesar de Sa. All cameras were oriented to the magnetic north using a field compass.

3.2 Camera set-up

For this research, 6 cameras were used to monitor the animal activity in 6 different locations. All cameras were of the product line Bushnell Natureview, 5 of the cameras were 12MP versions and the last one was the newer 16MP version (see https://www.bushnell.com). The cameras were positioned in the two main different areas of interest, the first being dry and wet grasslands. The second areas of interest include locations that are known for animal activity, according to private communications with the park management teams. Figure 3.2 shows where the cameras were set up and their northbound orientation.

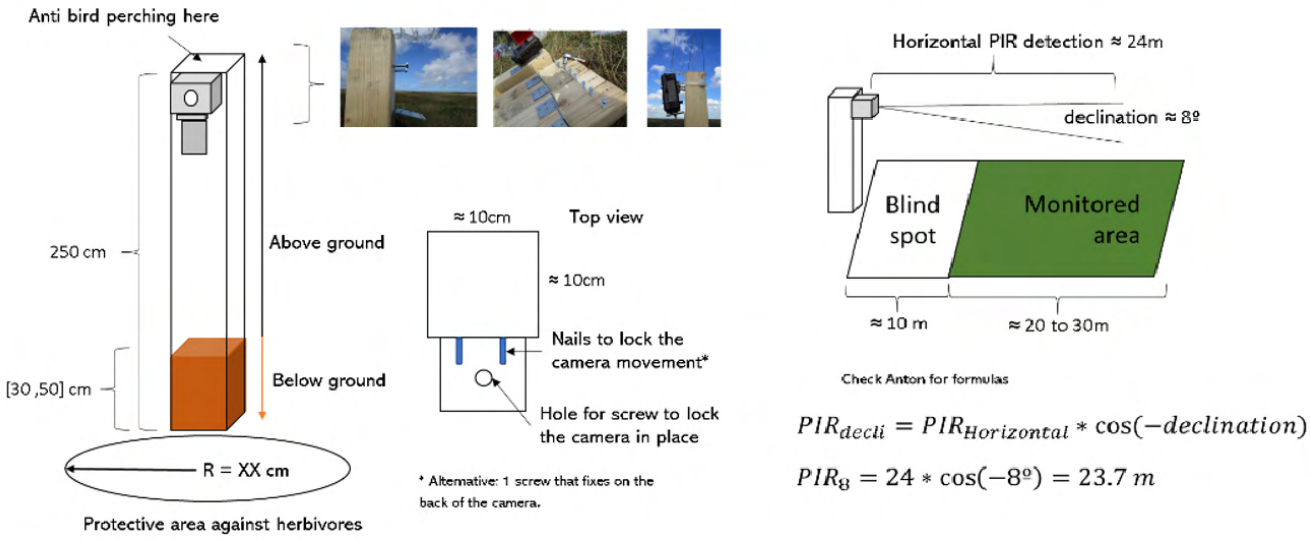


Figure 3.3: Overview of the camera configurations, provided by Nuno Cesar de Sa.

All cameras were set up at an approximate height of 2m using wooden poles and were oriented towards the magnetic north. To minimize the effect of camera movements as much as possible, the poles were set 30cm deep into the ground and a 1m high metal fence was set up around them to avoid animal interactions. The camera itself was locked in position on the pole by using 2 nails as shown in figure 3.3. To reduce the blind spot under the camera, the camera was slightly angled towards the ground with a declination angle of approximately 8°, which allowed for the PIR sensor to detect movement within 5 meters of the camera. To capture more than only animal activity in the proximity of the camera, a 15-minute interval was used for automatic acquisition of imagery, taking 3 pictures in quick succession. The cameras were initially planned to be set up for an entire year, starting from June 2019 to June 2020, but all poles on which the cameras were located, were damaged beyond repair by deer by September 2019. This occurred due to the lack of trees and other vertical vegetation that deer might use during rutting season. The cameras were removed from the field between September and October 2019 to avoid any risk to the health of the animals.

3.3 Dataset

In this section, we will talk about how we acquired the image data used in this research. First, we will talk about where the pictures originate from and what the data looks like. After that, we will talk about how these pictures were labelled and about the different distributions in the labelled data.

3.3.1 Composition

The pictures that were taken by the 6 cameras in the OVP are all available to us, an example of a day and a night photo can be seen in figure 3.4. From this collection of photos a sample of 3785 pictures was taken, which was put into Labelbox [29], see section 3.3.2. The newer camera described in section 3.2 produced images with a resolution of 3840x2880, whereas the other 5 cameras produced images with a resolution of 4000x3000.

The picture metadata contains information on the image resolution and time of acquisition, which can be used to filter according to time of day. In some cases, the metadata of an image was wrongly stored, but this did not cause any major disruption to the overall work.

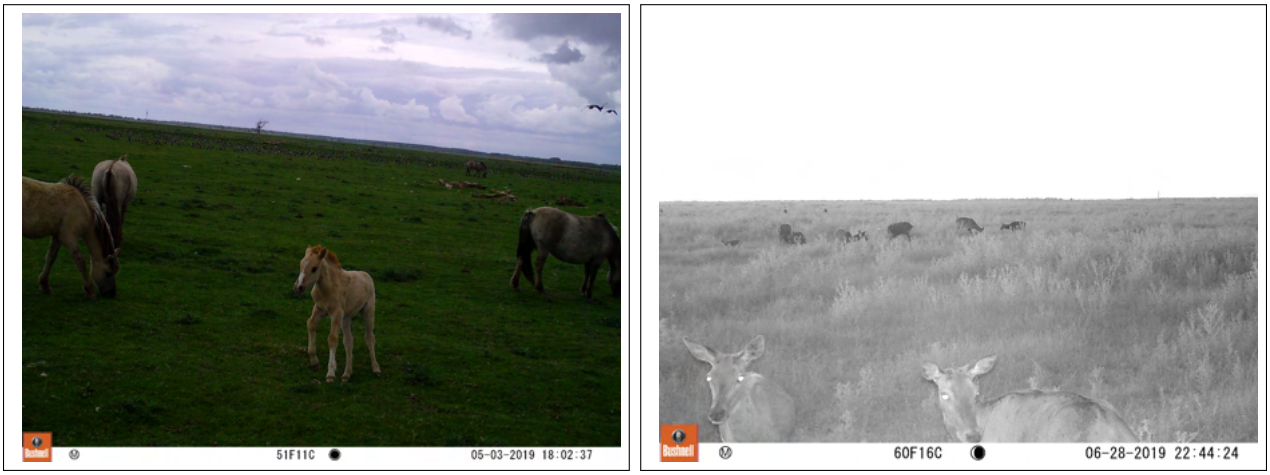


Figure 3.4: A day and a night photo taken at the Oostvaardersplassen by the trap-cameras.

3.3.2 Annotations

Deep learning models cannot train on plain pictures, annotations are required. A random sample of 3785 images has been taken from the photo set and has partly been labelled by a group of researchers and students. Labelbox [29] was used as labelling platform. The group was given a guideline on how to annotate the images, as shown in Appendix C. An example of a labelled photo can be seen in figure 3.5.

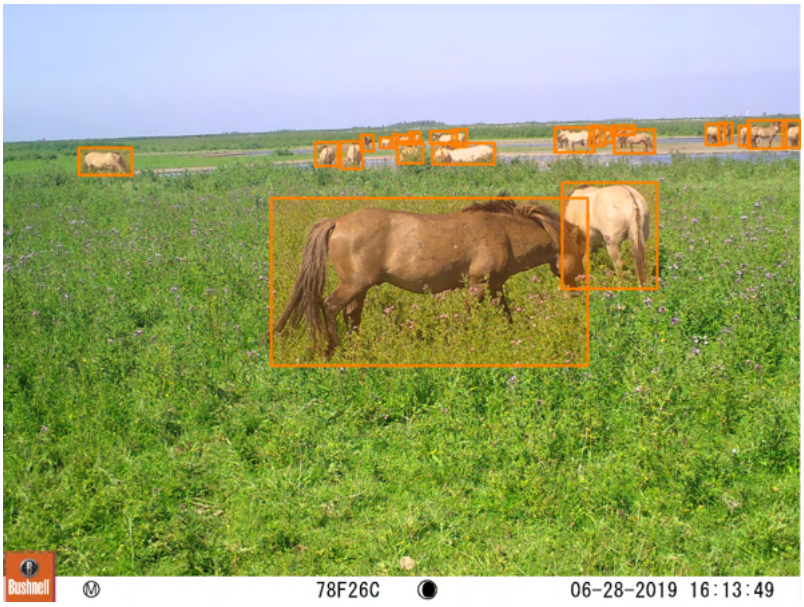


Figure 3.5: An image labelled with Labelbox.

The guideline makes a distinction between 6 animal classes, being deer, cattle, fox, goose, horse and bird, where geese do not belong to the bird class. Bounding boxes should envelop the animal entirely, but should not be bigger than necessary, and may overlap with other bounding boxes. The latter can occur when multiple animals are stacked, notice the overlapping boxes in figure 3.5. Animals that can only be seen partly should also be labelled, but only the part that is actually shown in the image, see figure 3.6. Animals that cannot be distinguished clearly should not be labelled, this includes animals that are too far away. Further, images can have some classifications, which can be used to indicate whether herds/flocks are present in the image, the image is taken in infrared mode (night picture), there are no animals in the image, or when some abnormalities obstruct the photo. The abnormalities classification includes a human/vehicle in the picture, poor visibility, a tilted/fallen camera or an obstructed view. Images that are too hard to label can be skipped by giving the image the "skip" tag, but this is discouraged, as these labels are essentially unlabelled.

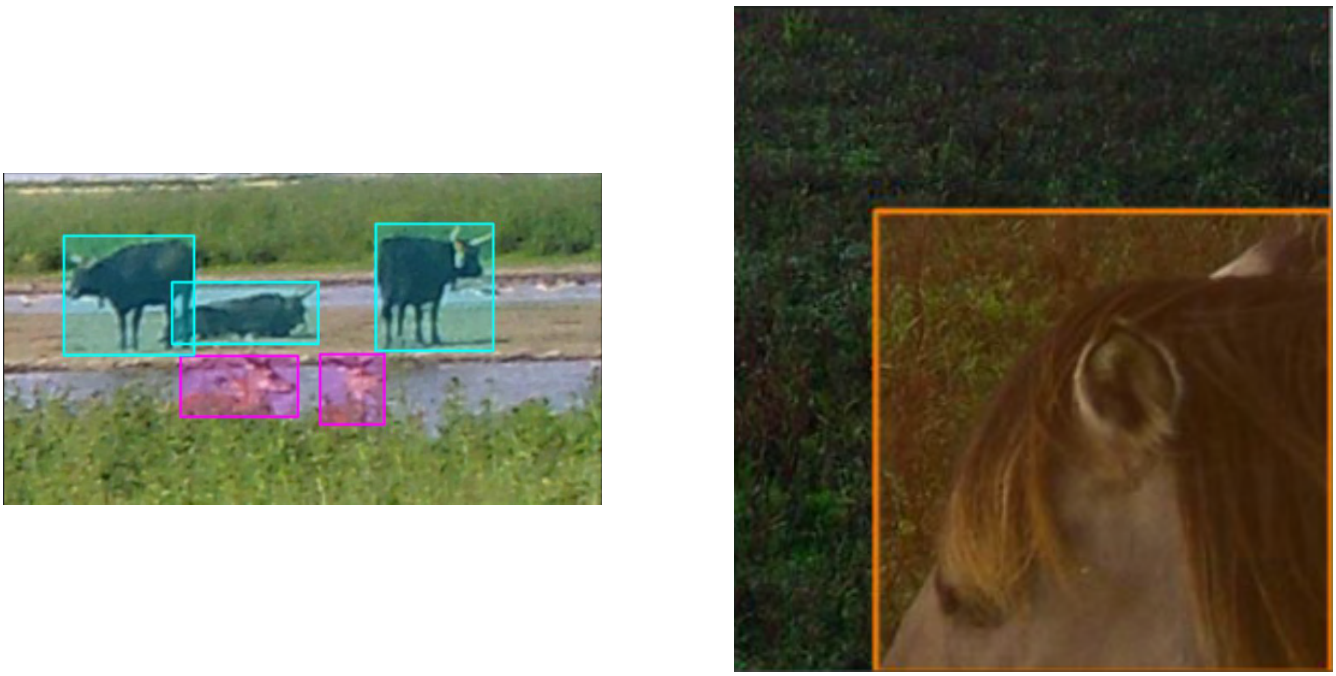


Figure 3.6: Labelled animals that are only partly visible.

The labelling effort yielded 3018 labelled images, including 211 skipped images, resulting in 2817 usable images. This collection contains images from the original photo set that were labelled more than once. The duplicate images do not have the exact same bounding boxes as their original counterpart, providing new bounding boxes to the set. However, we have chosen to not use duplicate images in the final labelled photo set, since the duplicates caused the labelled photo set to be skewed even more, which we will describe in section 3.3.3. To remove confusion about the different photo sets, we from now on refer to "labelled photos" set as "dataset".

3.3.3 Dataset distributions

In an ideal dataset, every possible label or state occurs an equal amount of times, as this benefits the training of the models [10]. In our dataset that would mean a similar distribution between each

Table 3.1: Label Distributions of the original label set, duplicates included.

Class	Tot. Amount	Tot. %	Day	Tot. %	Night	Tot. %
Deer	7750	36,73%	6869	32,55%	881	4,17%
Horse	7140	33,84%	6738	31.93%	402	1,91%
Cattle	4020	19,05%	3928	18.61%	92	0,44%
Goose	1809	8,57%	1809	8,57%	0	0%
Bird	379	1,80%	379	1,80%	0	0%
Fox	4	0,02%	3	0,01%	1	0,00%
Total	21102	100%	19726	93,48%	1376	6,52%

Table 3.2: Label Distributions of the sample of night images.

Class	Tot. Amount	Tot. %	Day	Tot. %	Night	Tot. %
Deer	1087	57.70%	553	29.35%	534	28.24%
Horse	457	24.25%	197	10.46%	260	13.80%
Cattle	171	9.08%	44	2.24%	127	6.74%
Goose	123	6.53%	0	0.0%	123	6.53%
Bird	9	0.48%	9	0.48%	0	0.0%
Fox	37	19.64%	2	0.11%	35	1.86%
Total	1884	100%	805	42.73%	1079	57.27%

class, both in day and night pictures. Deviating from this ideal dataset causes bias in the dataset [20], which also introduces bias in the models trained with this data.
The distribution of the different classes in the dataset can be seen in figure 3.7 and graph 3.1. The proportions of the classes differ strongly, deer and horses together make up around 70% of all labels, where geese, together with birds and foxes, only account for roughly 10% of all labels.
Furthermore, there are differences between the class distributions of day and night images, as can be seen in figure 3.8, which is likely the result of some animal species being more active at dusk/night while others are more active during the day.
The distribution of day and infrared images is also imbalanced. This is expected, as there are more daylight hours in a day than nighttime hours. The animals being more active during the day also worsens the dataset bias. To improve the ratio between day and night images, we added 150 images, which we have labelled manually, from the original photo set to the database, see graph 3.2.
Some limitations that have to be mentioned. The labelbox dataset has shortcomings that can be improved upon by filtering the set.
Empty images also need to be looked at. Not every deep learning model supports empty examples during training, Faster-R-CNN being such an example. This means that empty images are useless

Table 3.3: Label Distributions of the duplicate labels in the original set.

Class	Tot. Amount	Tot. %	Day	Tot. %	Night	Tot. %
Deer	2282	36.78%	2033	32.76%	249	4.01%
Horse	2674	43.09%	2606	42.00%	68	1.10%
Cattle	605	9.75%	605	9.75%	0	0.0%
Goose	576	9.28%	576	9.28%	0	0.0%
Bird	68	1.10%	68	1.10%	0	0.0%
Fox	0	0.0%	0	0.0%	0	0.0%
Total	6205	100%	11976	94.89%	317	5.11%

Table 3.4: Label Distributions of the original label set plus the added night labels, without duplicates.

Class	Tot. Amount	Tot. %	Day	Tot. %	Night	Tot. %
Deer	6635	39.42%	5469	32.49%	1166	6.93%
Horse	4923	29.25%	4329	25.72%	594	3.53%
Cattle	3556	21.13%	3337	19.83%	219	1.30%
Goose	1356	8.06%	1233	7.33%	123	0.73%
Bird	320	1.90%	320	1.90%	0	0%
Fox	41	0.24%	5	0.03%	36	0.21%
Total	16831	100%	14693	100,00%	2138	100,00%

during the training of such models. Figure 3.9 shows the ratio between empty images and images with objects. Almost a quarter of all images do not contain objects that the models should detect. Then, duplicate images should be talked about. Labelbox [29] can give the same image to multiple people to be labelled as a means of validation. The validation method itself was not used, resulting in some images being labelled up to 5 different times. This introduces near-duplicate labels, which increase the effect of those images during training [11]. Table 3.3 shows the number of duplicates per class in the original label dataset, including the 150 added night images. As shown in figure 3.10, some classes consist out of more than 25% of duplicates. We have decided to remove all duplicate labels from the dataset. The class distributions of this final dataset are shown in figure 3.11 and table 3.5.

Lastly, one of the main causes of the imbalance between classes is the lack of labelled photos of foxes and birds, as can be seen in graph 3.4. Birds and foxes together make up less than 2% of the database: More labelled photos are needed to create a good predictor for the Bird and Fox classes. We did not have the resources to label new images containing birds or foxes. Since this research mainly focuses on the grazers of the OVP, and foxes and birds (excluding geese) do not graze, we have decided to exclude the bird and fox classes from training.

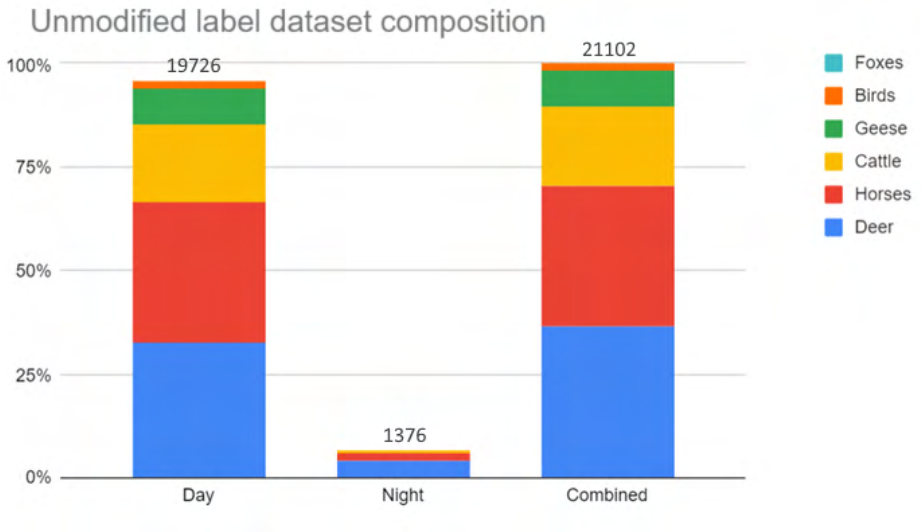


Figure 3.7: Proportion of day/night/combined classes in unmodified dataset (21102 labels total). Duplicate labels are included in the percentages.

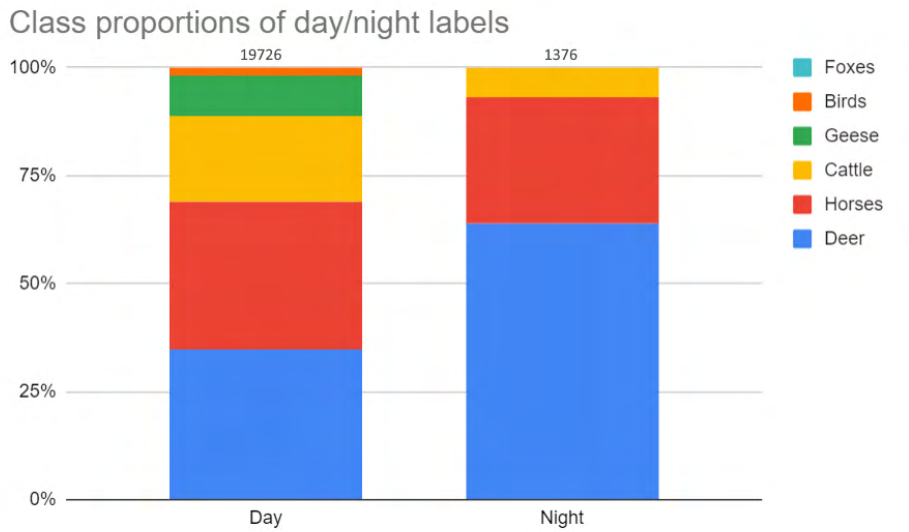


Figure 3.8: Comparison between the class composition of day labels (20531 labels) versus the class composition of the night labels (2455 labels). Duplicate labels are included in these counts.

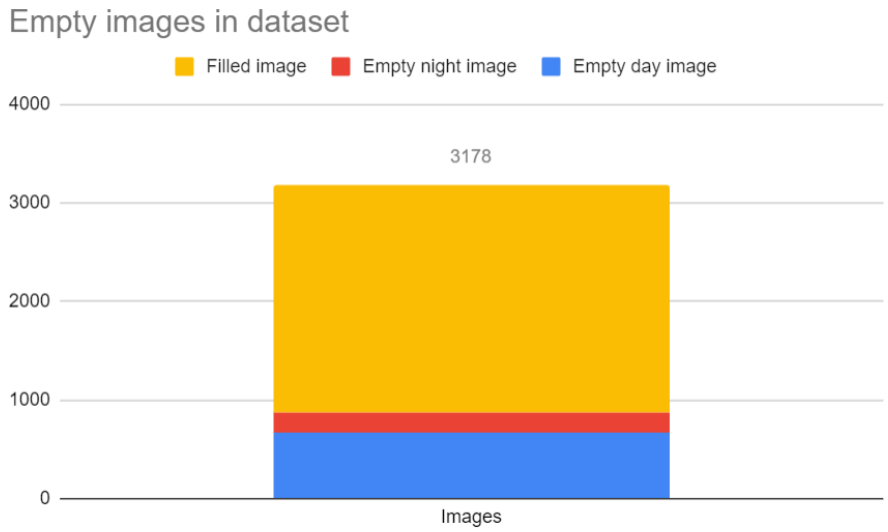


Figure 3.9: The amount of empty images in the labelled photo set. Duplicate images are included in the count.

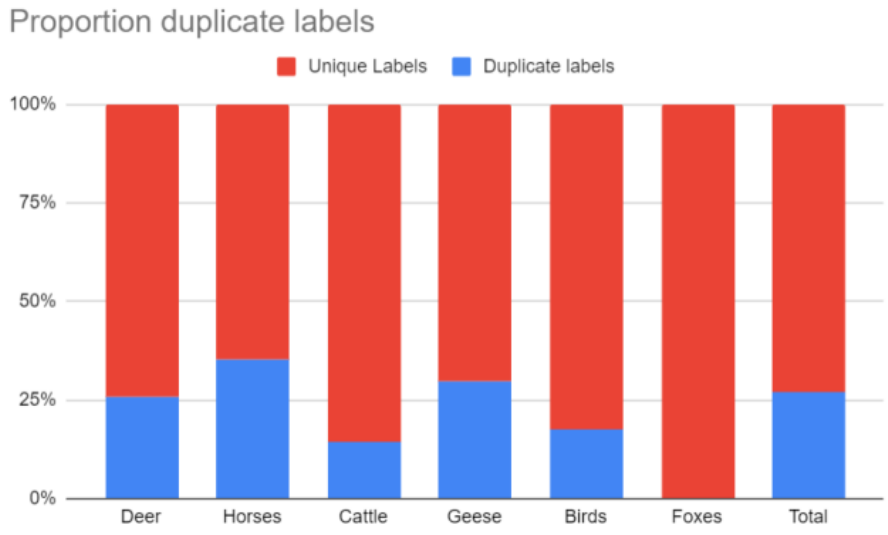


Figure 3.10: Amount of duplicate labels per class.

Table 3.5: Composition of the modified dataset which is used to train the DL models. 150 night images were added and all duplicates were removed from the original dataset to create this final dataset.

Class	Tot. Amount	Tot. %	Day	Tot. %	Night	Tot%
Deer	6635	39.42%	5469	32.49%	1166	6.93%
Horse	4923	29.25%	4329	25.72%	594	3.53%
Cattle	3556	21.13%	3337	19.83%	219	1.30%
Goose	1356	8.06%	1233	7.33%	123	0.73%
Total	16470	100%	14693	85.37%	2138	14.63%

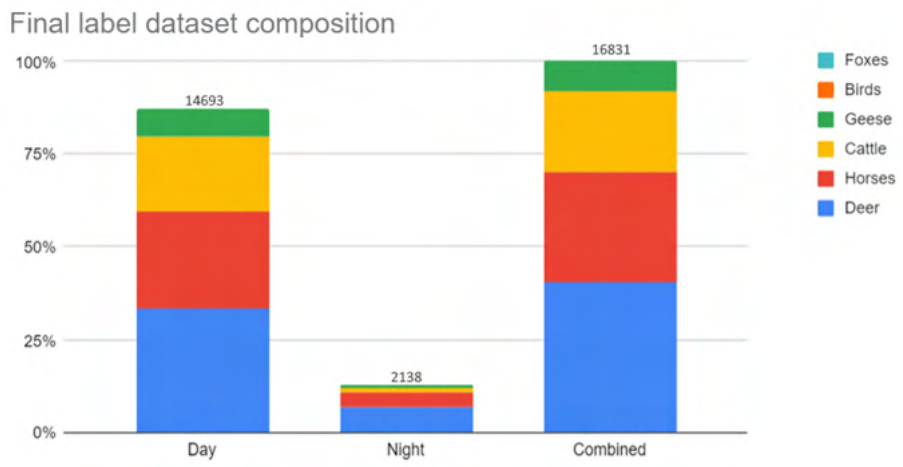


Figure 3.11: Composition of the modified dataset which is used to train the DL models. 150 night images were added and all duplicates were removed from the original dataset to create this final dataset.

3.4 Experimental set-up

In this section, the theory behind the implementation of the model optimizer and the experimentation will be described. Firstly, data augmentation will be discussed, which is used on images from the dataset before being fed to the model during training. Secondly, the hyper-parameters of the optimizer will be explained. Lastly, performance evaluation and validation will be talked about.

3.4.1 Data augmentation

The size of the training set influences the performance of a model. Models trained on large datasets tend to perform better than models trained on smaller datasets [30]. Datasets can be made larger artificially by applying transformations to an image before being passed on to the model during training. The transformations also help models to become more resistant to small deviations from

the training set [12]. Two simple transformations that are used in the training algorithm are a horizontal image flip and changes to the brightness and contrast of the image.

3.4.2 Hyper-Parameters

A DL model is trained by a learning algorithm, which is called an optimizer, that adjusts the weights of the model [11]. Optimizers often have hyper-parameters that influence the learning behaviour of the model, which enables fine-tuning of the learning process. Different DL architectures and problems are optimized by different hyper-parameters values, meaning that a general hyper-parameter setting that works well for all problems does not exist [21]. Grid search is a traditional method to find good hyper-parameter values [31], but has the downside of being very slow: Each parameter adds an additional dimension to the search space of the optimization. We do not have enough time to run a full grid search, which is why we decided to optimise each parameter somewhat serially, consult section 3.6 for a precise description of our search method.
The hyper-parameters all have different influences on the training process. Next follows a description and the impact of the optimizer hyper-parameters [21].

- **Initial learning rate**. The learning rate dictates the effect of prediction errors on the changes to the weights and affects both training times and the ability to effectively search the problem space. Higher learning rates encourage exploration of the search space, while smaller learning rates encourage exploitation of the search space. The initial learning rate is an important hyper-parameter and should be optimized if possible.

- **Learning rate reduction**, also called gamma. As the exploration of the search space takes place, the optimizer might encounter gradients that could lead to local minima that cannot be explored with a learning rate that is too high. Lowering the learning rate makes it possible to explore these potential local minima. Lowering the learning rate too slow results in a slow descent down the gradient, increasing training time, while reducing the learning rate too fast can cause the optimizer to get stuck in a local minimum, potentially decreasing the performance of the model.

- **Learning rate schedule**. Learning rate schedulers control the changes that happen to the learning rate. A learning rate scheduler enables the use of a dynamic learning rate schedule, optimizing when the learning rate should be changed.

- **Patience**. Some learning rate schedulers make use of a hyper-parameter called patience. The patience indicates how many epochs should be between 2 learning rate changes. Low patience values might change the learning rate too fast, reducing performance, and high values increase the training time.

- **Batch size**. The batch size is the number of images that will be processed at the same time. This hyper-parameter is used mostly to increase training speed by exploiting the parallelism of GPU's. The batch size is dependant on the amount of available memory in the GPU and the size of the training data, where more memory can hold more images, increasing the batch size.

- **Epochs**. The amount of epochs determines how often the model is fitted on the entire training set. If the amount of epochs is too small, the model may be affected by underfitting, while too many epochs cause overfitting [32], both withholding the model from reaching full potential.
- **Seed**. Not setting a seed removes the ability to reproduce training results with the same hyper-parameters. A fixed seed enables valid comparison between training runs with different hyper-parameter settings, which is important for finding optimal hyper-parameter values.

3.4.3 Validation method

For validating our models we used Stratified k-fold cross-validation (SCV) [33], which can be used as a validation method when a dataset is unbalanced, illustrated in figure 3.12. SCV works by evenly dividing the whole dataset into k subsets called folds, each fold having the same class distributions as the whole dataset, guaranteeing that each fold equally represents the whole dataset. Instead of training one model on a training set, k models are trained, each using $k-1$ of the folds as training set and using the leftover fold as validation set. After training the k models, the best performing one can be selected for use.
We have adjusted how folds are structured to improve training times. Normally folds are created on the level of labels: Every label from the same class has the same chance to be included in the fold, meaning that labels that originate from the same image can end up in different folds. Images have to be loaded into memory every time a label from that image needs to be processed by the model. Processing all the labels from one image after one another guarantees that each image is loaded only once, speeding up training significantly. We have achieved SCV by assigning weights to images based on their labels and taking samples based on those weights, instead of assigning weights to the labels themselves and sampling them. This way the principles behind stratified samples are satisfied and memory loads are kept to a minimum. For our experiments we use 5 folds; 4 for training and 1 for validating.

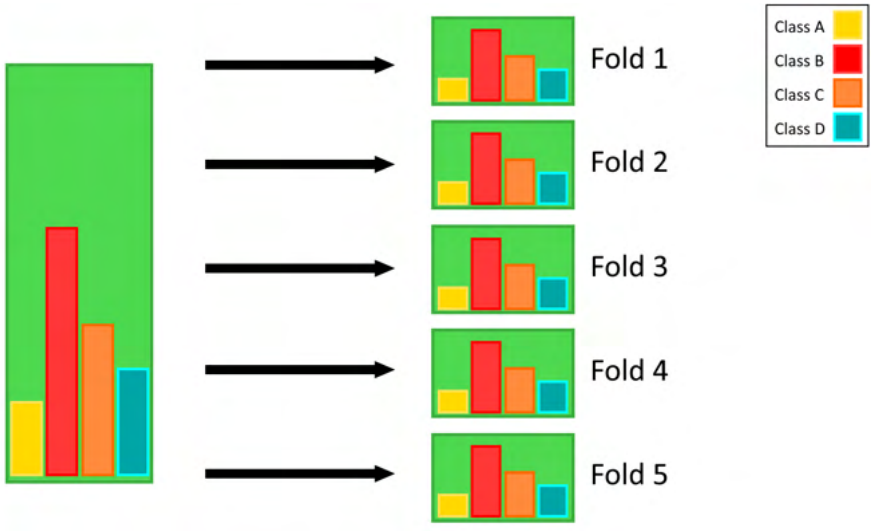


Figure 3.12: Division of the labelled data into equally sized folds. Each fold has the same class composition as the whole dataset.

Intersection over Union (IoU) is commonly used to measure accuracy in object detection tasks [34], see figure 3.13. Confusion matrix data calculated with IoU can be used to calculate the Average Precision (AP) of a model. Different CV competitions [35] [36] use different definitions for AP. We use the AP definition from the COCO object detection challenges [36], using 10 IoU thresholds from 0.5 to 0.95 with an interval of 0.05.

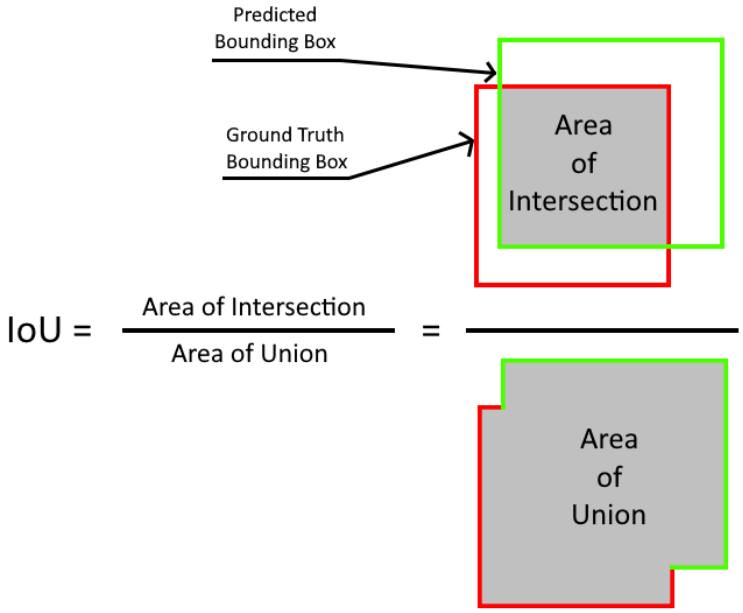


Figure 3.13: Here is shown how to calculate the Intersection over Union, which is used to calculate which predicted bounding box matches the ground truth the most.

3.5 Implementation

3.5.1 Framework

The training program is written in python 3.8, using the PyTorch library [16] as framework. Out-of-the-box models can be loaded from the internet or can be included from local storage. PyTorch is widely used in the ML community and many CV architectures often either have a PyTorch implementation or are included in the PyTorch model zoo, which can be included from a python module. The wide usage, active help forum and documentation have convinced us to use PyTorch as training framework.

3.5.2 Architecture

The chosen architecture to train is Faster-R-CNN [13]. Faster-R-CNN is a two-stage DL architecture designed for Computer Vision and extends upon its predecessor Fast-R-CNN [37] by adding a Region Proposal Network (RPN) to identify potential objects with. The RPN searches for regions in the image that might contain objects, which are then sent through the Fast-R-CNN architecture, significantly speeding up the inference process. Faster-R-CNN was state-of-the-art when it was first

developed and still holds on as a fast and competitive architecture. The Faster-R-CNN architecture is the foundation of many other DL architectures, making it an important addition in the field of CV. A PyTorch implementation of Faster-R-CNN can be downloaded from the PyTorch model zoo, which makes it easy to set up and use. The implementation also comes with the option to load a pretrained network, which is trained on the COCO17 [38] dataset. Pretrained models have already adjusted their network to extract features from the dataset that they have been trained on. This method principle is called Transfer Learning [39], and can speed-up training times and improve model performance [40]. When pretrained on a new dataset, pretrained models can make use of the already present feature maps, since datasets might have similar features, like lines or corners. This helps the models converge faster, speeding up training times.
Faster-R-CNN is not as fast as architectures that were built for real-time detection like SSD [15] and YOLOv5 [41], but as real-time detection speed is not necessary for this project, the higher accuracy of Faster-R-CNN has convinced us to use it as our architecture of choice. We have used the pretrained network from PyTorch to give our model a head start and to decrease the overall training time.

3.5.3 Dataset

Dataset exports from Labelbox [29] are in JSON format, which allows for simple conversion to an easy to use label format. We have used the following format:

image_ID , left , right , top , bottom , class

After converting the JSON file to a text file and sorting the file on the image_ID column, all Bird and Fox labels were removed from the file. To create stratified folds, firstly, weights of each class (W_{class}) is calculated using Eq. 3.1 with the total occurrences of a class N_{class} in the set and the total amount of labels N_{labels} in the set. Then, the weight W of each image is calculated using Eq. 3.2, where C is the set of the 4 animal classes, N_{class} equals the amount of occurrences of a class in the image and N_{img} equals the number of labels in the image. These weights are then used to form 5 folds, using a weighted data sampler.

$$W_{class} = 1 - \frac{N_{class}}{N_{labels}} \tag{3.1}$$

$$W = \frac{\sum_{class \in C} W_{class} * N_{class}}{N_{img}} \tag{3.2}$$

Loading batches of full-sized images into GPU memory caused a memory shortage, which is why images are downsized to a resolution of 1000 X 1000 pixels. Then, the image has a 50% chance to be flipped horizontally and the brightness and contrast of the image are shifted by a random amount. After that, the images are loaded into GPU memory.

3.5.4 Training Loop

As stochastic optimizer, we have chosen the ADAM algorithm [42], which is widely used to adjust the weights of neural networks [43, 44]. PyTorch contains an ADAM implementation that is very easy to use. To update the learning rate, 2 different methods were implemented. In the first method,

the learning rate was decreased by a constant factor after a constant amount of epochs. A constant learning rate schedule does not optimize the learning path and can result in suboptimal results. To counteract this, a dynamic learning rate schedule was implemented, which reduces the learning rate when the validation process does not detect performance improvements in the model for N amount of epochs, where N is the patience value of the learning rate scheduler. The calculated mAP described in section 3.5.5 is used as a metric for the performance of a model.

3.5.5 Evaluation Loop

The performance of the model is calculated with the IoU [34] of the ground truths of the labels and the predicted labels. When matching the predicted labels to the ground truths, the ground truth with the highest IoU value is omitted from being matched to other predictions, which ensures that ground truth is matched to at most one prediction. The next step is to calculate a confusion matrix from the predicted labels and their IoU value. The IoU value of each prediction is measured against the threshold (explained in section 3.4.3) that must be exceeded for the prediction to be flagged as a true positive (TP). If the threshold is exceeded, but the predicted class is false, or if the threshold is not exceeded at all, the prediction is flagged as a false positive (FP). Ground truths that are not matched with any predicted label are flagged as false negatives (FN).
Next, the recall (R) and precision (P) are calculated [45] at every threshold, using the confusion matrices. Recall is defined as the proportion of labels that were predicted correctly and is calculated by dividing the number of true positives by the addition of true positives and false negatives (Eq. 3.3). Precision is defined as the proportion of predicted labels that match a ground truth and is calculated by dividing the number of true positives by the addition of true positives and false positives (Eq. 3.4). Instead of using the precision to plot a precision-recall (PR) curve, the interpolated precision is used [35]. The interpolated precision is calculated for every recall value r by taking the highest precision measurement P at that recall value $\tilde{r}$ (Eq. 3.5). Precision and recall are 2 values that often correlate negatively, making it hard to optimize both values. To solve this problem, the F1-score can be used, which combines precision and recall in a single value, as calculated with Eq. 3.6. The F1-score is a value between 0 and 1, 1 being the best score.

$$R = \frac{TP}{TP + FN} \tag{3.3}$$

$$P = \frac{TP}{TP + FP} \tag{3.4}$$

$$P_{interpolated}(r) = \max_{\tilde{r}:\tilde{r} \geq r} P(\tilde{r}) \tag{3.5}$$

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{3.6}$$

The average precision (AP) for each class is then calculated by taking the area under the PR-curve. The mean average precision (mAP) then equals the mean value of all AP values.
If the mAP of the current model state is an improvement over the previous best state the current state is saved, which prevents overfitting and removes the need for an early stopping algorithm, which stops the learning process after no improvements are made after several epochs.

3.6 Experiments

The experiments are divided into 2 phases. The experiments in phase one are small and focus on finding promising hyper-parameter settings that can be used for the experiments in phase 2, which train models until the performance of the models converge.
In the first phase, hyper-parameters are tuned to optimize the learning process of the Faster-R-CNN architecture [13], which is done in short running experiments of 60-100 epochs. These experiments run long enough to show the influence of the hyper-parameter value on the performance curve and do not take too much time to finish. The first few experiments will be run with a learning rate reduction value, also called gamma, of 0.1, a batch size of 2 and a seed of 1 for both the PyTorch and Random python modules. The batch sizes larger than 2 resulted in crashing due to memory shortages.
In the first experiments, different recommended values [21] for the initial learning rate are tried out. Once acceptable values for the initial learning rate have been found, these values will be used in a small grid search along with different patience values. We have not found recommended values for patience, forcing us to simply try out different values in a random range. We experimented with patience values between 5 and 15.
Lastly, gamma will be experimented with to see if the learning process takes advantage of a learning rate schedule that decreases more gradually. The value that is used in the previous experiments is the recommended value of 0.1 [21], which will be replaced by 0.05, using promising hyper-parameter settings from the previous experiments.
Once good hyper-parameter settings are found, phase 2 starts, in which promising settings are used to train models for 200 epochs, which is enough for the models to converge.

4 Results

4.1 Hardware

The experiments were conducted on the Duranium server of LIACS, which is specially made available for students. This is a server dedicated to training ML models. The server has six NVIDIA GeForce GTX 980 Ti cards with 6 GB RAM each and two NVIDIA GeForce GTX TITAN X cards with 12 GB RAM each.

4.2 Experimental Results

All experiments in this section were conducted with a batch size of 2, a random seed value of 1. The learning rate schedule that was used reduced the learning rate after the model did not show an increase in performance after a fixed amount of epochs (the patience hyper-parameter). The hyper-parameter settings and the peak mAP performance of each experiment are shown in table 4.1.

Table 4.1: Hyper-parameters and peak mAP of the experiments. Peak mAP is calculated as the mean of the peak mAP values of the 5 validation folds.

Epochs	lr	gamma	patience	mAP
60	0.001	0.1	5	0.136
60	0.0001	0.1	5	0.395
60	0.001	0.1	7	0.18
60	0.0001	0.1	7	0.421
60	0.0001	0.1	9	0.425
60	0.0001	0.1	11	0.411
100	0.0001	0.05	15	0.450
200	0.0001	0.1	11	0.468
200	0.0001	0.1	15	0.471

4.2.1 Hyper-Parameter Tuning

Figure 4.1 shows the mAP curves of 4 experiments with hyper-parameter combinations of learning rate values 0.001 and 0.0001 and patience values 5 and 7. The experiments were run for 60 epochs with a learning rate reduction value of 0.1 and each curve is calculated as the mean of the performance of the 5 different validation folds that each experiment uses. Both experiments with an initial learning rate of 0.0001 converge at a higher mAP than the experiments with an initial learning rate of 0.001, which may even crash to 0, as is shown in figure 4.2.

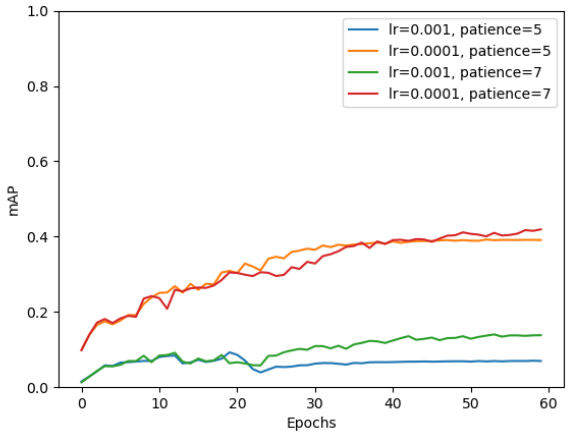


Figure 4.1: mAP curves of experiments with varying intitial learning rates and patience values.

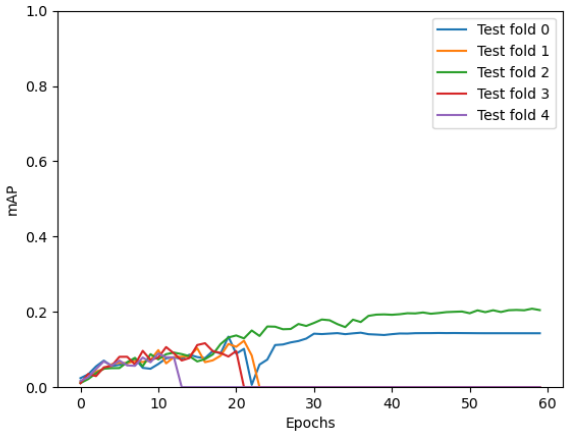


Figure 4.2: mAP curves of each validation fold for a training run with an initial learning rate of 0.001, a patience of 5, a gamma of 0.1 and a runtime of 60 epochs.

Figure 4.3 shows the mAP curves of 5 experiments where different patience values were tried out. The experiments were run for 60 epochs with a learning rate reduction value of 0.1 and each curve is calculated as the mean of the performance of the 5 different validation folds that each experiment uses. After 60 epochs, all models achieve an mAP of around 0.42. Models trained with lower patience values start converging faster than models with higher patience values.
Figure 4.4 shows the mAP curves of 2 experiments where different gamma values were used. The experiments were run for 100 epochs, with an initial learning rate of 0.001 and each curve is calculated as the mean of the performance of the 5 different validation folds that each experiment uses. Both curves follow the same mAP increase over time, however, the curve with a gamma value of 0.1 seems to fluctuate more than the curve with a gamma value of 0.05. Both curves have started converging around the same mAP value.

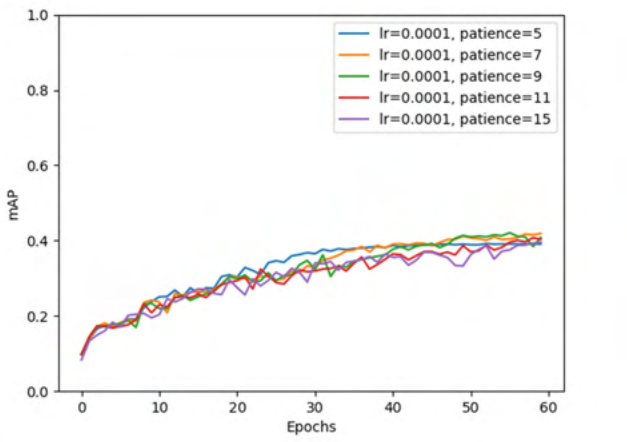


Figure 4.3: mAP curves of experiments with varying patience values and a runtime of 60 epochs.

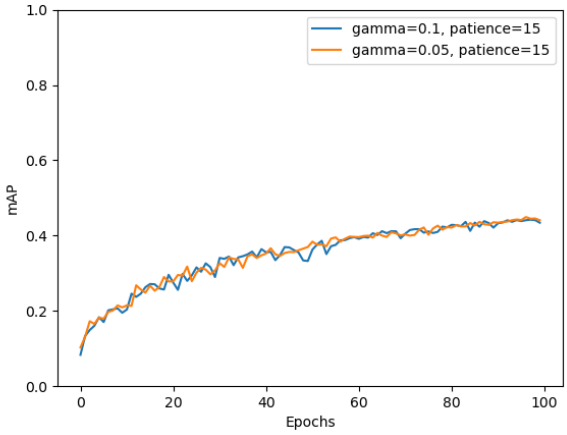


Figure 4.4: mAP curves of experiments with varying gamma values and a runtime of 100 epochs.

4.2.2 Model Training

Figure 4.6 shows the mAP curves of 2 experiments that used the hyper-parameters values gathered from the tuning experiments. Both models were trained for 200 epochs, with an initial learning rate of 0.001 and a gamma value of 0.1 and each curve is calculated as the mean of the performance of the 5 different validation folds that each experiment uses. The difference between both models is the patience value, one used a value of 10 epochs, while the other used a value of 15 epochs. The individual runs can be seen in Appendix A. The curve of both models converge around the same mAP value, as shown in table 4.1. Precision, recall and F1 values of the best performing model (last row of 4.1) are shown in table 4.2 at different thresholds, figure 4.5 shows the precision-recall curve for each animal class and figure 4.3 shows the AP values of the different classes.

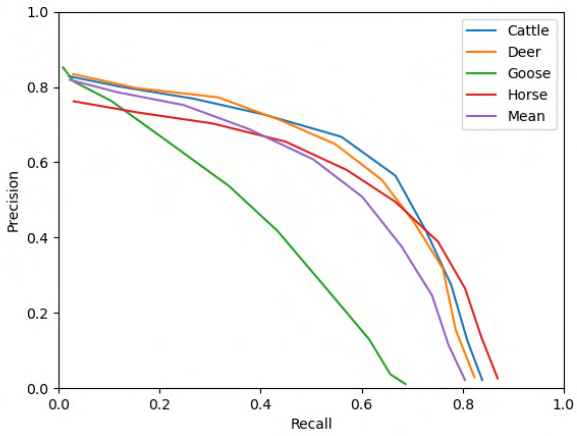


Figure 4.5: Precision-Recall curves of the 4 animal classes and their mean from a 200 epoch run with an initial learning rate of 0.0001, a gamma value of 0.1 and a patience value of 15.

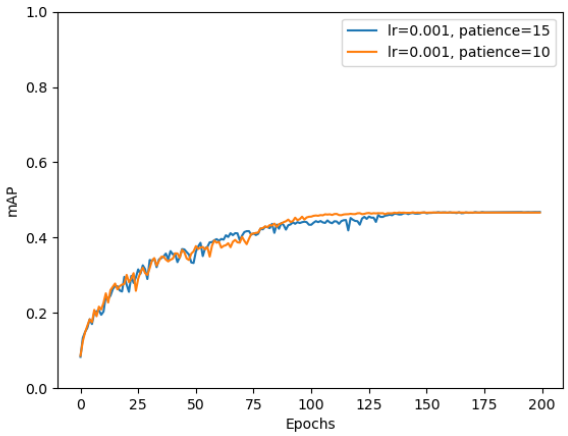


Figure 4.6: mAP curves of two 200-epoch runs, using hyper-parameters found in previous experiments. An initial learning rate of 0.0001 was used, as well as a gamma value of 0.1. Both curves are calculated by taking the mAP mean of the 5 validation folds.

Table 4.2: Precision, recall and F1-scores from a 200 epoch run with an initial learning rate of 0.0001, a gamma value of 0.1 and a patience value of 15, for every threshold. Legend: 1=Cattle, 2=Deer, 3=Goose, 4=Horse.

Threshold	**Precision**				**Recall**				**F1**			
Class	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
0.50	0.83	0.83	0.85	0,76	0.84	0.82	0.69	0.87	0.83	0.83	0.76	0.81
0.55	0.80	0.79	0.81	0.73	0.81	0.79	0.66	0.83	0.80	0.79	0.73	0.78
0.60	0.77	0.77	0.76	0.70	0.78	0.76	0.61	0.80	0.77	0.77	0.68	0.75
0.65	0.73	0.71	0.66	0.65	0.73	0.70	0.53	0.75	0.73	0.71	0.59	0.70
0.70	0.67	0.65	0.54	0.58	0.67	0.64	0.43	0.67	0.67	0.64	0.48	0.62
0.75	0.56	0.55	0.42	0.50	0.56	0.54	0.34	0.57	0.56	0.55	0.37	0.53
0.80	0.42	0.44	0.26	0.39	0.41	0.44	0.22	0.45	0.41	0.44	0.23	0.41
0.85	0.27	0.32	0.13	0.26	0.27	0.32	0.10	0.31	0.27	0.32	0.12	0.28
0.90	0.13	0.15	0.04	0.14	0.12	0.15	0.02	0.16	0.13	0.15	0.03	0.15
0.95	0.02	0.03	0.01	0.03	0.02	0.03	0.01	0.03	0.02	0.03	0.01	0.03

Table 4.3: Model AP values for every class from a 200 epoch run with an initial learning rate of 0.001, a gamma value of 0.1 and a patience value of 15.

	Cattle	**Deer**	**Goose**	**Horse**
AP	0.52	0.53	0.45	0.47

5 Discussion

5.1 Hyper-parameter Tuning

The effects of the different hyper-parameters vary in size. The initial learning rate is an example of a hyper-parameter with a sizable impact on performance, as shown in figure 4.1, which was also stated by Bengio et al. [21]. An initial learning rate of 0.0001 results in a good training process, even under different patience and gamma values. The mAP drops in 4.2 can be explained by gradient explosions, causing a high loss value to drastically alter neural network weights, leaving the local minima [46]. Appendix A shows a side to side comparison between the mAP curves of figure 4.2 and their loss curves. Gradient explosions hurt pretrained networks especially, as they have already converged to detect and classify objects, degrading the network to a worse state than it started in. Figure 4.3 shows that patience values have a small effect on the performance of the model. The low patience value increases the speed at which the learning rate declines, resulting in less exploration and more exploitation. The low learning rate inhibits the ability to escape local minima and stops the learning process prematurely. Although higher patience values increase training times, they do not affect the overall performance of the model if enough training time is given, as shown in figure 4.6, which can be explained by the fact that the same set of learning rates is used as in the experiments with lower patience values. An optimal patience value appears to be in the range of [11,15], more experimentation is needed to decide a specific value.
Figure 4.4 shows 2 training runs with different gamma values, one with a gamma value of 0.1 and one with a value of 0.05. The difference in gamma does not result in higher mAP peaks, the models have both converged around the same mAP value.
The hyper-parameter setting that we found best performing has an initial learning rate of 0.001, a gamma of 0.1, and a patience value of 15. 200 training epochs proved to be enough to let the model fully converge. These values are the same or come close to the values mentioned in the work of Bengio et al. [21].

5.2 Model Training

The best performing model resulted from an optimization process with an initial learning rate of 0.001, a gamma value of 0.1, and a patience value of 15. Table 4.2 shows the precision, recall and F1-score of each class at each threshold. At a threshold of 0.5, both precision and recall reach values around 0.8 for all classes, which is summarized in the F1-score. The detector finds and correctly classifies a large part of the animals, although many false detections are still made. Geese have the highest precision, which can be explained by the distinct form of the bird, which is entirely different compared to other classes. The model sometimes misclassifies the mammals, as they all assume the

same poses. Geese also have the lowest recall, which may be caused by insufficient training data containing geese, or by the small size of geese.
Above a threshold of 0.75, the model produces more false predictions than true predictions, caused by false classification, missed objects and detections of animals that do not exist in the picture. The detector has a lower performance than the detectors from Norouzzadeh et al. [27] and Miao et al. [26] by a large margin. Both works had access to large datasets with more than a million labelled images, where we had access to 3000 labelled images. Both works also tried out numerous different architectures and deeply optimized hyper-parameters, which we did not have the time for. The AP values of the best performing model are represented in table 4.3, which shows that cattle and deer have an AP slightly above 0.5, while the goose and horse classes are slightly below 0.5. The AP values reflect the number of available labels for each class during training, which was shown in figure 3.11. Remarkable is the difference between the performance of the model on detecting geese, as there were relatively few geese in the training set, but the model still reached an AP of 0.45, a comparable performance to the other classes. The Precision-Recall curve in figure 4.5 shows the relation between precision and recall at different threshold levels. The worse performance of the model on predicting geese becomes more clear here: Where all other classes show a curve, the goose curve almost resembles a linear line, indicating low performance. The other classes are closer together, resulting in a mean curve, and thus mAP, that is close to the deer, cattle and horse classes.
In order to look more closely at the workings of our model, inference has been done on a sample of 500 unlabelled images from the image set with a threshold of 0.5. A large portion of the sample does not contain any animals and the model does not produce false positives in those images. Some interesting image parts are shown in the next part of this section. The full images can be found in Appendix B. Something to note is that detections often sport a very high confidence score, all the while table 4.2 shows that the confidence scores should be lower. Figures 5.1 and 5.2 are good examples of this phenomenon, as they show that most detected objects have a confidence score close to or equal to 1.0. We cannot explain the discrepancy between what we would expect in theory and practice.

Figure 5.1: Picture of a herd of horses taken by a trap-camera in the Oostvaardersplassen, labels generated by our Faster-R-CNN model.

Figure 5.2: Picture of a flock of geese taken by a trap-camera in the Oostvaardersplassen, labels generated by our Faster-R-CNN model.

Figure 5.3 shows 2 images in which the model did not detect any animals, while many geese can be seen in the background. The guideline given to the labelling group (Appendix C) states that animals too far away should not be labelled, omitting labels from animals in the background of most pictures, causing the model to not recognize distant animals. Figure 5.4 shows another example of this, where deer closer to the camera are detected and identified, but deer in the distance are not.

Figure 5.3: Pictures taken by trap-cameras in the Oostvaardersplassen. Animals can be seen in the distance, but our Faster-R-CNN model was not able to detect them.

Figure 5.4: Picture of a herd of deer taken by trap-cameras in the Oostvaardersplassen. Deer closer to the camera get detected by our Faster-R-CNN model, while deer towards the back are less likely to be detected.

The model detects animals multiple times, giving multiple labels to one object, an example is shown in figure 5.5. This seems to be especially true for calves, which resemble deer. The reverse may also happen, where the model cannot differentiate between different animals and sees them as one, as shown in figure 5.6. Combining animals is especially prevalent in herds and happens more often as the herd is further away from the camera, as can be seen in figure 5.7. The combined labels generated by the model result from images in the training set that were labelled manually like this: It is sometimes hard to differentiate animals when they are stacked on top of one other, especially distant animals. As a model can only get as good as its training data, combined labels were expected.

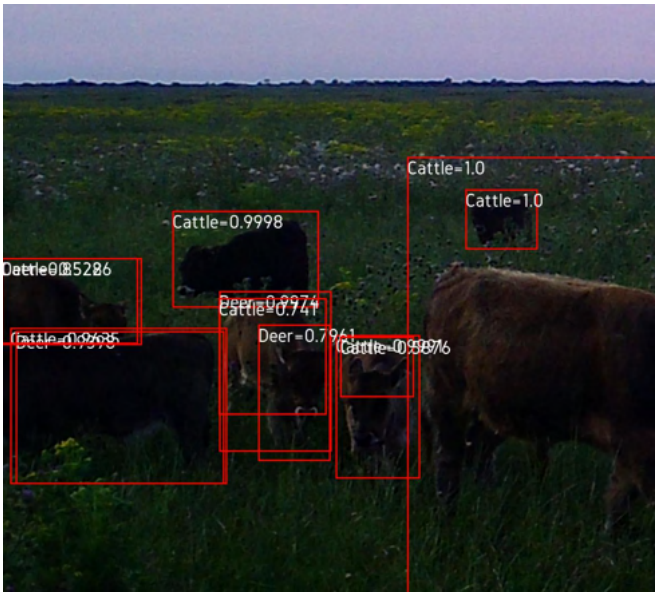


Figure 5.5: Picture of cattle and their young, taken by trap-cameras in the Oostvaardersplassen with labels generated by our Faster-R-CNN model. The model sometimes generates 2 different labels for the same object.

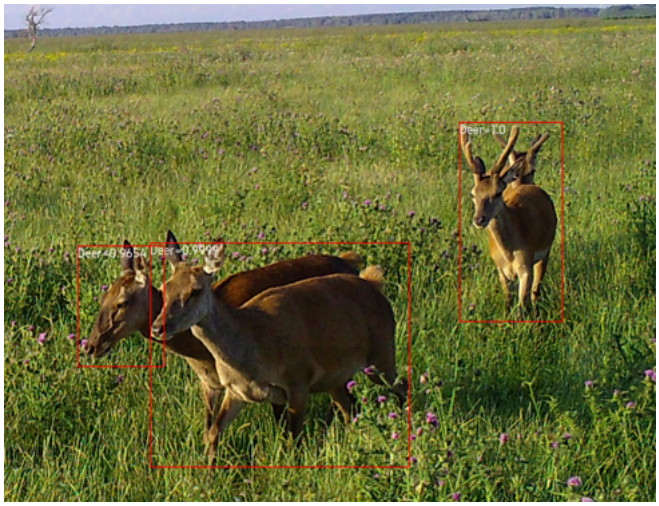


Figure 5.6: Picture of deer, taken by trap-cameras in the Oostvaardersplassen with labels generated by our Faster-R-CNN model. The model is not always able to correctly differentiate between objects, resulting in incorrect labels.

Figure 5.7: Picture of a herd of horses, taken by trap-cameras in the Oostvaardersplassen with labels generated by our Faster-R-CNN model. The model has a hard time generating correct labels for distant herds.

6 Conclusion

In this thesis, we trained a computer vision model to detect and classify animals in trap-camera pictures taken at the Oostvaardersplassen (OVP). First, the labelled dataset from the trap-cameras

in the OVP was analysed to identify how the dataset could be improved for training. Then we adjusted the dataset by adding and removing images. After balancing the dataset, different hyper-parameters were optimized for Faster-R-CNN by trying out configurations recommended in other works. Lastly, a Faster-R-CNN model was trained using these hyper-parameters. We showed that it is possible to train a model that can detect and classify the different grazing animals when they are not clustered together.

7 Future Work

In the future, better datasets of the OVP or similar areas may be available, which can be used to build better training sets. Transfer Learning [40] could also be used to adjust neural networks trained on larger datasets from reserves in other countries with similar animals. Our small time window did not allow for thorough hyper-parameter optimization. Better hyper-parameters may be found with methods such as a full grid search or random search [31]. Other learning rate schedulers, such as a cyclic scheduler which also momentarily increases the learning rate, can improve the training process. Aside from tuning hyper-parameters ourselves, Automated Machine Learning [47] may circumvent the optimization problem altogether. Lastly, other CV architectures aside from Faster-R-CNN can be looked at. More modern architectures are available and different architectures are better at detecting different types of objects. Multiple different architectures could be combined into an assemblage to better cover classes with different properties [48]. SSD [15], for instance, is good at detecting smaller objects and worse at detecting larger objects, whereas R-FCN [49] is worse at detecting smaller objects and better at detecting larger objects: Using both architectures in parallel covers both smaller and larger objects.

References

[1] Frans WM Vera. Large-scale nature development–the oostvaardersplassen. *British Wildlife*, 20(5):28, 2009.

[2] Jamie Lorimer, Christopher Sandom, Paul Jepson, Chris Doughty, Maan Barua, and Keith Kirby. Rewilding: Science, practice, and politics. *Annual Review of Environment and Resources*, 40:150902153650003, 11 2015.

[3] JT Vulink, MR Van Eerden, M Platteeuw, and M De Roos. De oostvaardersplassen, deel 1. *Waterpeil en begrazing sturen het systeem. Landschap*, 26:109–120, 2009.

[4] ICMO2. Natural processes, animal welfare, moral aspects and management of the oostvaardersplassen, 2010.

[5] Mitra Baratchi, Nirvana Meratnia, Paul JM Havinga, Andrew K Skidmore, and Bert AG Toxopeus. Sensing solutions for collecting spatio-temporal data for wildlife monitoring applications: a review. *Sensors*, 13(5):6054–6088, 2013.

[6] Allan F O'Connell, James D Nichols, and K Ullas Karanth. *Camera traps in animal ecology: methods and analyses*. Springer Science & Business Media, 2010.

[7] J Marcus Rowcliffe, Roland Kays, Bart Kranstauber, Chris Carbone, and Patrick A Jansen. Quantifying levels of animal activity using camera trap data. *Methods in Ecology and Evolution*, 5(11):1170–1179, 2014.

[8] Maureen S McCarthy, Marie-Lyne Després-Einspenner, Damien R Farine, Liran Samuni, Samuel Angedakin, Mimi Arandjelovic, Christophe Boesch, Paula Dieguez, Kristin Havercamp, Alex Knight, et al. Camera traps provide a robust alternative to direct observations for constructing social networks of wild chimpanzees. *Animal Behaviour*, 157:227–238, 2019.

[9] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[10] David A Forsyth and Jean Ponce. *Computer vision: a modern approach*. Pearson,, 2012.

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[17] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[18] Sylvain Christin, Éric Hervet, and Nicolas Lecomte. Applications for deep learning in ecology. *Methods in Ecology and Evolution*, 10(10):1632–1644, 2019.

[19] Tatiana Tommasi, Novi Patricia, Barbara Caputo, and Tinne Tuytelaars. *A Deeper Look at Dataset Bias*, pages 37–55. Springer International Publishing, Cham, 2017.

[20] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018.

[21] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[22] Andrew Bengsen, John Butler, and Pip Masters. Estimating and indexing feral cat population abundances using camera traps. *Wildlife Research*, 38(8):732–739, 2011.

[23] Francesco Rovero, Emanuel Martin, Melissa Rosa, Jorge A Ahumada, and Daniel Spitale. Estimating species richness and modelling habitat preferences of tropical forest mammals from camera trap data. *PloS one*, 9(7):e103300, 2014.

[24] Michael A Tabak, Mohammad S Norouzzadeh, David W Wolfson, Steven J Sweeney, Kurt C VerCauteren, Nathan P Snow, Joseph M Halseth, Paul A Di Salvo, Jesse S Lewis, Michael D White, et al. Machine learning to classify animal species in camera trap images: Applications in ecology. *Methods in Ecology and Evolution*, 10(4):585–590, 2019.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[26] Zhongqi Miao, Kaitlyn M Gaynor, Jiayun Wang, Ziwei Liu, Oliver Muellerklein, Mohammad Sadegh Norouzzadeh, Alex McInturff, Rauri CK Bowie, Ran Nathan, X Yu Stella, et al. Insights and approaches using deep learning to classify wildlife. *Scientific reports*, 9(1):1–9, 2019.

[27] Mohammad Sadegh Norouzzadeh, Anh Nguyen, Margaret Kosmala, Alexandra Swanson, Meredith S Palmer, Craig Packer, and Jeff Clune. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences*, 115(25):E5716–E5725, 2018.

[28] Perry Cornelissen, Marca C Gresnigt, Roeland A Vermeulen, Jan Bokdam, and Ruben Smit. Transition of a sambucus nigra l. dominated woody vegetation into grassland by a multi-species herbivore assemblage. *Journal for Nature Conservation*, 22(1):84–92, 2014.

[29] Labelbox.

[30] Michael Bernico, Yuntao Li, and Dingchao Zhang. Investigating the impact of data volume and domain similarity on transfer learning applications. In *Proceedings of the Future Technologies Conference*, pages 53–62. Springer, 2018.

[31] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv preprint arXiv:1912.06059*, 2019.

[32] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8093–8104. PMLR, 13–18 Jul 2020.

[33] Daniel Berrar. Cross-validation., 2019.

[34] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019.

[35] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[36] Common objects in context.

[37] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[38] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[39] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[40] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[41] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, yxNONG, Adam Hogan, lorenzomammana, AlexWang1900, Ayush Chaurasia, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Durgesh, Francisco Ingham, Frederik, Guilhen, Adrien Colmagro, Hu Ye, Jacobsolawetz, Jake Poznanski, Jiacong Fang, Junghoon Kim, Khiem Doan, and Lijun Yu . ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration, January 2021.

[42] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[43] Xiongwei Wu, Doyen Sahoo, and Steven CH Hoi. Recent advances in deep learning for object detection. *Neurocomputing*, 396:39–64, 2020.

[44] Míriam Bellver Bueno, Xavier Giró-i Nieto, Ferran Marqués, and Jordi Torres. Hierarchical object detection with deep reinforcement learning. *Deep Learning for Image Processing Applications*, 31(164):3, 2017.

[45] Peter A Flach and Meelis Kull. Precision-recall-gain curves: Pr analysis done right. In *NIPS*, volume 15, 2015.

[46] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[47] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[48] MA Ganaie, Minghui Hu, et al. Ensemble deep learning: A review. *arXiv preprint arXiv:2104.02395*, 2021.

[49] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. *arXiv preprint arXiv:1605.06409*, 2016.

A Results

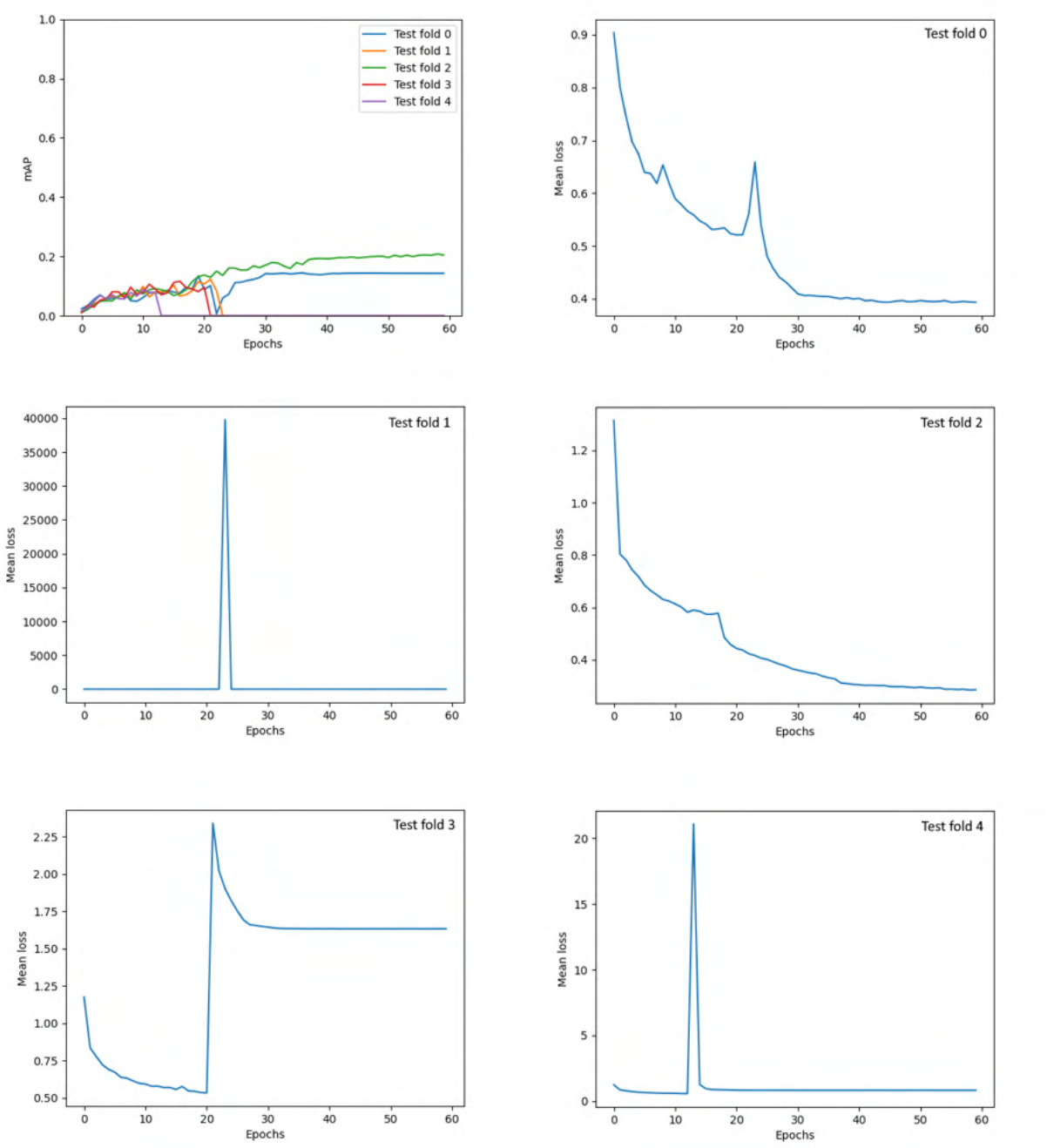


Figure A.1: mAP and loss curves of each validation fold for a training run with an initial learning rate of 0.001, a patience of 5, a gamma of 0.1 and a runtime of 60 epochs. The loss graphs show where gradient explosions took place.

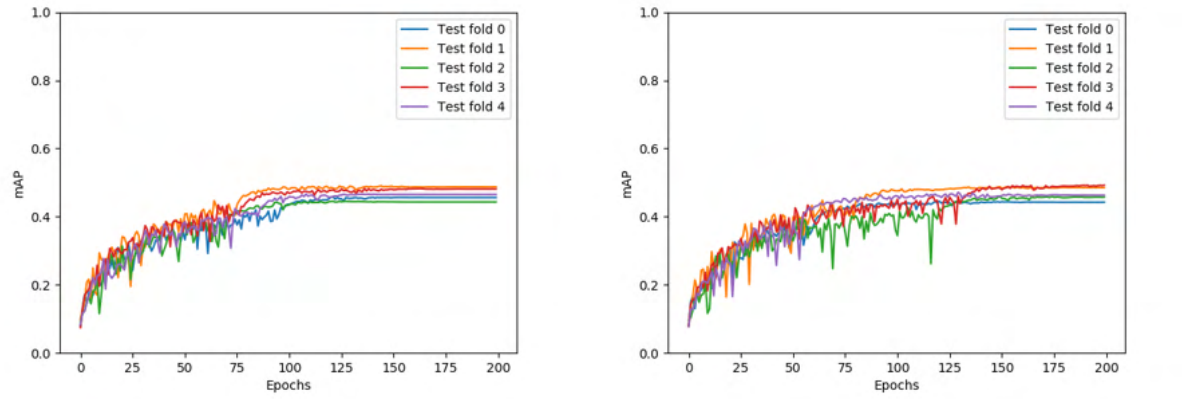


Figure A.2: mAP and curves of each validation fold for a the best 2 runs with an initial learning rate of 0.0001, a gamma of 0.1, a runtime of 200 epochs and a patience of 11 (left) and 15 (right).

B Inference Examples

Figure B.1: Pictures of deer taken by trap-cameras in the Oostvaardersplassen. The detector has a hard time detecting animals in the far distance.

Figure B.2: Pictures taken by trap-cameras in the Oostvaardersplassen. When the animals are not packed together, the detector is able to both localize and classify the animals almost perfectly.

Figure B.3: Pictures taken by trap-cameras in the Oostvaardersplassen. The detector cannot always distinguish different animals, seeing them as one.

Figure B.4: Pictures taken by trap-cameras in the Oostvaardersplassen. The detector sometimes generates 2 labels for the same animal (left). If animals are far away and in herds, the detector is not able to correctly localize individual animals (right).

Figure B.5: Pictures taken by trap-cameras in the Oostvaardersplassen. The detector is sometimes unable to detect and classify cattle when they get photographed in a pose that the detector is not familiar with.

Figure B.6: Pictures taken by trap-cameras in the Oostvaardersplassen. The detector is able to detect animals in images taken in night-mode.

C Label Guideline

Labeling guide

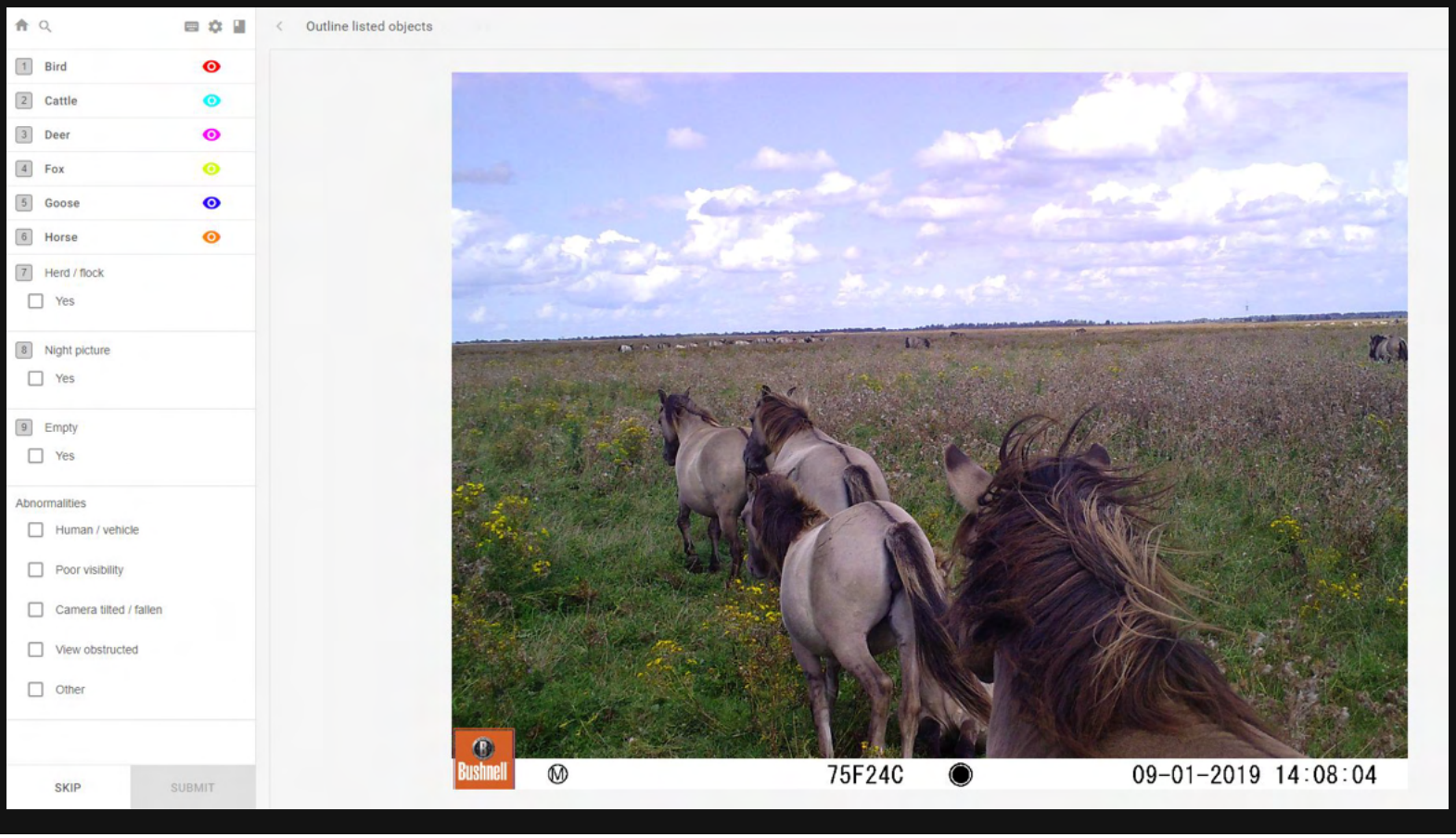
Why?

- Using the data to train a CV model
- Identify which animal is in the picture
- Give an estimate of the number of animals

Good to know

- Images are preselected using a basic model
- +/- 10 percent of the images will be empty
- 10% of data will be labeled multiple times

- We will be using labelbox



Objects

- Bird*
- Cattle
- Deer
- Fox
- Goose
- Horse

*Bird does not include geese

What to label

- Make sure the box is not too small or to big
- If necessary, boxes may overlap

What to label

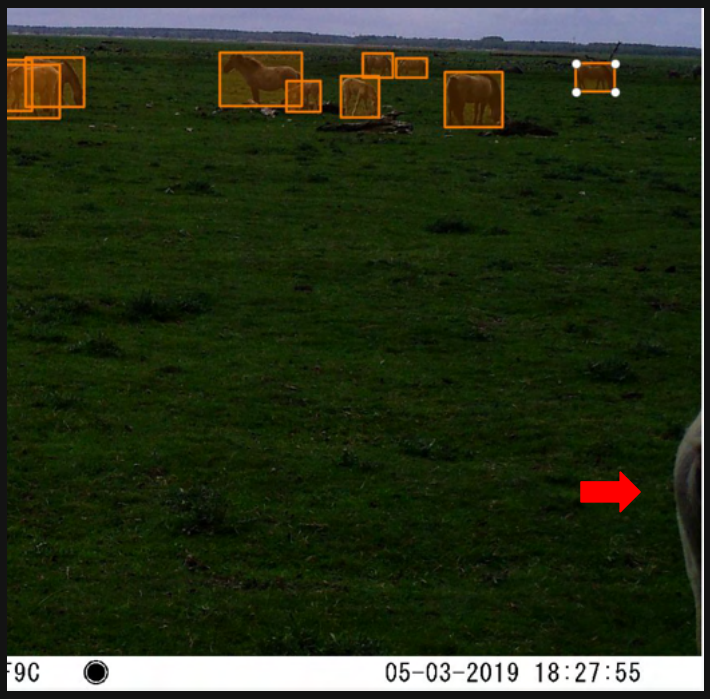
- Don't label animals which are too far away and therefore not clearly distinguishable
- Birds / geese in the background do not have to be labeled

What to label

- “Parts” of animals also need to be labeled
- e.g. Heads sticking out of the grass

What to label

- “Parts” of animals also need to be labeled
- Unless the animal is not clearly distinguishable



What to label

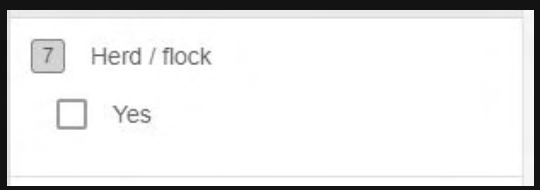
- “Parts” of animals also need to be labeled
- If necessary, boxes may overlap
- Only label label the parts of the animal you can see!

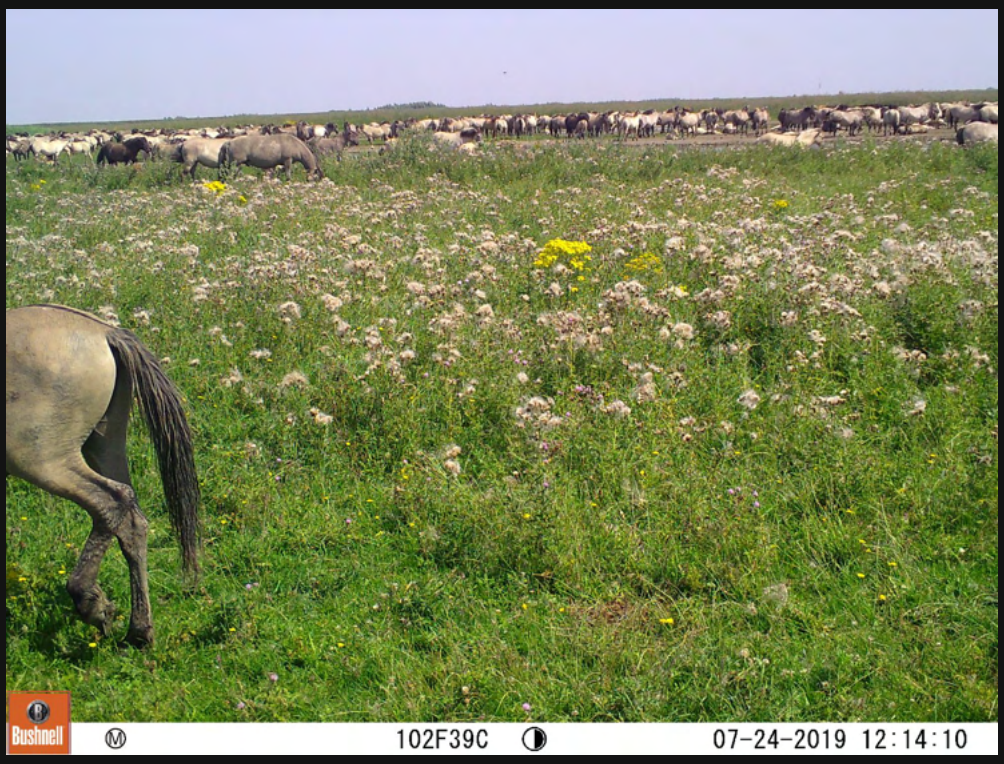
Classifications

- Herds / flocks
- Night / infrared pictures
- “Empty” pictures
- Abnormalities

What to label

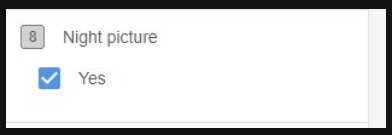
- In case of big herds, label animals which are clearly visible.
- If the herd is very compact, try to label the first row.
- Make sure to check the “herd” box.

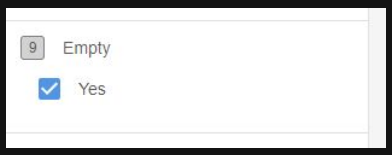




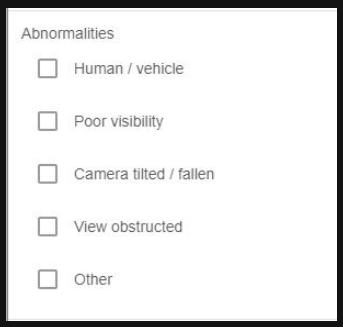
What to label

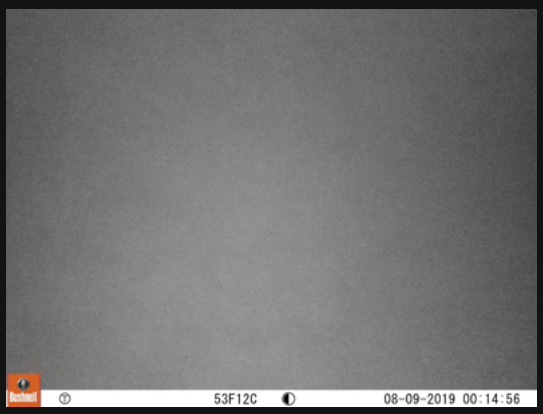
- Pictures with no animals can be labeled “empty”.
- **Infrared** / night pictures should be labeled “Night”





What to label





Notes

- Please don’t skip any pictures