Master of Science Thesis in Media Technology
Department of Electrical Engineering, Linköping University, 2020

Edge Machine Learning for Animal Detection, Classification, and Tracking

Amanda Tydén and Sara Olsson

Master of Science Thesis in Media Technology

Edge Machine Learning for Animal Detection, Classification, and Tracking:

Amanda Tydén and Sara Olsson

LiTH-ISY-EX--20/5308--SE

Supervisor: **Magnus Malmström**
ISY, Linköpings universitet

Examiner: **Fredrik Gustafsson**
ISY, Linköpings universitet

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

A research field currently advancing is the use of machine learning on camera trap data, yet few explore deep learning for camera traps to be run in real-time. A camera trap has the purpose to capture images of bypassing animals and is traditionally based only on motion detection. This work integrates machine learning on the edge device to also perform object detection. Related research is brought up and model tests are performed with a focus on the trade-off regarding inference speed and model accuracy. *Transfer learning* is used to utilize pre-trained models and thus reduce training time and the amount of training data. Four models with slightly different architecture are compared to evaluate which model performs best for the use case. The models tested are *SSD MobileNet V2*, *SSD Inception V2*, and *SSDLite MobileNet V2*, *SSD MobileNet V2 quantized*. Since the client-side usage of the model, the SSD MobileNet V2 was finally selected due to a satisfying trade-off between inference speed and accuracy. Even though it is less accurate in its detections, its ability to detect more images per second makes it outperform the more accurate Inception network in object tracking.

A contribution of this work is a light-weight tracking solution using *tubelet proposal*. This work further discusses the *open set recognition* problem, where just a few object classes are of interest while many others are present. The subject of open set recognition influences data collection and evaluation tests, it is however left for further work to research how to integrate support for open set recognition in object detection models. The proposed system handles detection, classification, and tracking of animals in the African savannah, and has potential for real usage as it produces meaningful events.

Acknowledgments

We wish to express our appreciation to our supervisor Magnus Malmström, and examiner Fredrik Gustafsson, for their commitment and helpful advice. Thanks to the Project Ngulia team for support and input during the project. Thanks to Kolmården Zoo for enabling us to both work on-site at the savannah enclosure, and to connect remotely using their network. Thanks to the people at HiQ for lending us an office workplace in Norrköping.

Norrköping, June 2020
Amanda Tydén and Sara Olsson

Contents

Notation

ABBREVIATIONS

Abbreviation	Meaning
AP	Average Precision
API	Application Programming Interface
AR	Average Recall
CNN	Convolutional Neural Network
COCO	Common Objects in Context Dataset
CSV	Comma separated values
FPS	Frames per second
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
IoU	Intersection over Union
IR	Infrared
LIDAR	Light Detection and Ranging
mAP	mean Average Precision
ML	Machine Learning
PASCAL	Pattern Analysis, Statistical modelling and Computational Learning
PTZ	Pan-Tilt-Zoom
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Units
SSD	Single Shot Detector
SVM	Support Vector Machine
VGG	Visual Geometry Group
VOC	Visual Object Classes
XML	Extensible Markup Language
YOLO	You Only Look Once

1 Introduction

Biodiversity is declining at a rapid pace, and conservationists are faced with the tough quest of surveying wildlife populations. A research field currently advancing is the use of machine learning on camera trap images. Some applications aim to achieve overall biodiversity statistics, analysis which does not need to be run in real-time. However, few explore the usage of animal detection for use on-device. In environments where animals are directly threatened, it is naturally desirable with a system that requires minimal manual work and low latency, and which can generate warning alarms. Moreover, if the device is to be placed out in nature and run self-sufficient in terms of energy supply, there is a need for thoughtful design choices of the system.

1.1 Background

Ngulia is a rhino sanctuary in Kenya and home to about 80 individuals of black rhinoceros, vulnerable to poaching. This drives Project Ngulia, a public-private partnership to combat this problem. Camera traps are to be placed around the park, powered by solar cells, and connected to a mobile network. With a detector that automatically sends interesting observations in real-time to a server, it can improve decision making on actions for protection.

1.2 Motivation

Deploying a camera trap system out in the field involves a set of challenges. For example, to handle night vision, close-up shots, or network constraints. In this work, the following challenges are addressed:

- The need to pick events sparsely in time but with high confidence, due to limited power and bandwidth.
- Training a model without access to image data from the deployment site.
- Prioritizing some classes while still being able to detect the others. For instance, to detect a rhinoceros while a herd of buffalo is visible at the same time.

1.3 Aim

The aim is to develop a detector that analyses the video images from camera traps in real-time. Classes to identify are rhinoceros, humans and a set of six common large animals in the African savannah. The scope of this project also includes to extract events of importance.

1.4 Research questions

1. How can a multi-class detector be designed to handle the open set problem, where just a few of the species are of interest? Should other species be included in the training data, and if so, to what extent?
2. Given the continuous output from object detection, what lightweight solution can be proposed to assemble events of interest?
3. How will the trade-off between inference speed and accuracy for a CNN model affect the quality of real-time object tracking?

1.5 Limitations

The accuracy of the prediction model will depend on the amount of training data, which is limited. This is due to manual collection and annotation of images, and sparse resources of images that are available to public use for some species. Further, detection models which requires too long inference time for real-time object tracking will not be considered.

2

Theory

All research questions arise from the topic of deep learning, and especially convolutional neural networks (CNN). In this chapter, the core concepts of deep learning are covered as well as related work within the field of lightweight machine learning and event extraction.

2.1 Deep learning

Deep learning is a sub-field of machine learning and its methods are based on artificial neural networks, algorithms inspired by the structure and function of the human brain. Today's computer power and the extensive amount of data makes deep learning more powerful than ever before. Common applications of computer vision are image classification and object detection. With image classification, the objective is to categorize the input image as a whole, an approach mainly used for single object images. For the case where multiple objects are present, object detection is widely used as it both classifies and localizes each detected object. [1, 2]

2.1.1 Convolutional neural networks

CNN is a well known tool used for computer vision and it has increased in accuracy during the last decades. A CNN is built up of an input layer, an output layer, and intermediate hidden layers. An overview of the architecture of a convolutional neural network can be seen in Figure 2.1. [2]

Convolution is a mathematical operation that is fundamental for CNNs. The result of an image convolution can express how information from an image source matches with another. When this second source is a small matrix, a *kernel*, it can

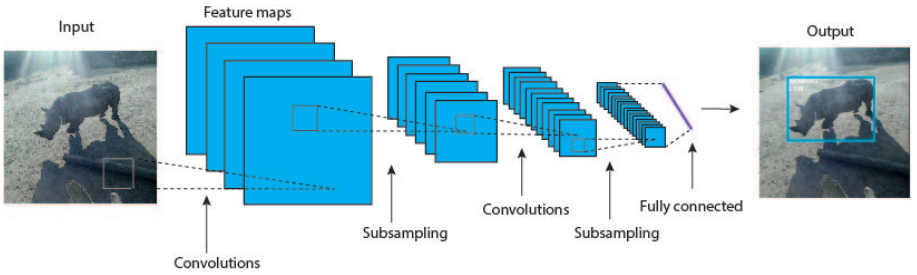


Figure 2.1: *The general architecture of a CNN.*

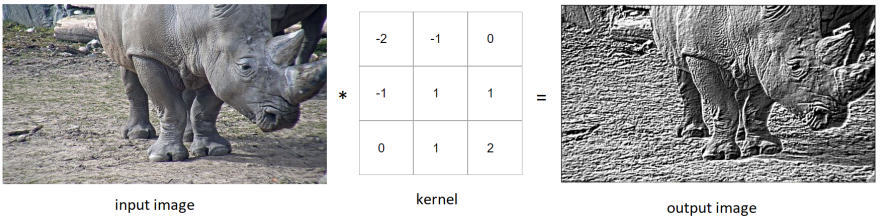


Figure 2.2: *As the kernel slides over the input image, it is filtered by convolution. This example kernel emphasizes pixel differences along a line from the top left to the bottom right.*

slide over the whole image and produce a filtered version of the image. Patches of the image which agree with the filter will contribute to meaningful information in the resulting image, as seen in Figure 2.2.

The input of the CNN is a tensor, a generalization of a matrix in higher dimensions. When the tensor has passed a convolutional layer, the input image is abstracted to a feature map. The output value is calculated by each neuron by applying a non-linear function, called *activation function*, determined by a vector of weights and biases. As the learning phase of a neural network is iterative, the biases and weights are adjusted such that the neurons represent meaningful filters. That is, the weights, biases, and activation function together determine how much the given input should activate a neuron. To tune these parameters, a *loss function* measures the discrepancy between the current output of the network and the desired output. Given the loss values, weights and biases are adjusted so that the loss value is minimized. [3]

2.1.2 Definitions

To configure a neural network, functions of its behavior are to be set. These include activation functions, loss functions and optimizers. Loss functions measure the penalty given a prediction and the true annotation, which are used to

optimize the weights and biases of a model during training. In the case of object detection, a common approach is to use a combination of classification loss and localization loss.

Activation functions

Activation functions are applied to represent a neuron's output as a function of its input. The activation function *Rectified Linear Units* (ReLU), shown in Equation (2.1), has shown to speed up model optimization. Alternative activation functions, like *Sigmoid* and *tanh*, have been found to cause slower training, due to the functions causing the weights to become very large or very small after some epochs of training. [4]

$$f(x) = max(0, x) \tag{2.1}$$

Classification loss functions

Classification loss measures the predicted class probability towards the ground truth. Both *Sigmoid* and *Softmax* are common classification functions and also used for loss computation. The Sigmoid function is a logistic function which limits the output to a range between 0 and 1, which is convenient in the prediction of probabilities and often used in classification problems with two classes. The logistic Sigmoid is defined by Equation (2.2) and the prediction $\hat{y}$ from the Sigmoid is defined by Equation (2.3), where W are the weights, h the hidden features and b the bias. [3]

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{2.2}$$

$$\hat{y} = \sigma(W^T h + b) \tag{2.3}$$

The Softmax function is a multi-class generalisation of the logistic function. Softmax is often used as output to a classifier, to represent the probability distribution over n different classes. Elements in the prediction $\hat{y}_i$ are between 0 and 1 and the entire vector sums up to 1. A linear layer predicts unnormalized log probabilities z as Equation (2.4). Formally, the Softmax function is given by Equation (2.5) where the Softmax can exponentiate and normalize z to obtain desired $\hat{y}$. [3]

$$z = W^T h + b \tag{2.4}$$

$$Softmax(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{2.5}$$

For the loss function, these probabilities are measures towards the true probability distribution with *cross entropy*, which quantifies the difference between two probability distributions. Equation (2.6) presents cross entropy, where $\hat{y}$ is the

predicted value, y is the ground truth value and M is the number of classes.

$$cross\ entropy = -\sum_{i}^{M} y_i \log(\hat{y}_i) \tag{2.6}$$

Localization loss

Localization is the ability of the network to locate an object in an image. The localization loss is the computed error between the ground truth and the predicted bounding boxes. *Smooth L1 localization loss* is defined by Equation (2.7) and is less sensitive to outliers than the $L2$ loss. Both $L1$ and $L2$ are penalty terms to the loss function with the purpose to avoid overfitting. [5]

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{2.7}$$

Regularization loss

Regularization loss is a technique used to penalise large weights. It makes slight modifications to the learning algorithm such that the model can generalize better. This in turn improves the model's performance on unseen data. The *L2 regularization* is also known as *weight decay* as it forces the weights to decay towards zero, but not exactly zero, see definition in Equation (2.8) where θ_i represents the feature weights. [3]

$$R(\theta) = \|\theta\|_2^2 = \sum_{i=1}^{n} \theta_i^2 \tag{2.8}$$

Optimizer

An *optimizer* updates the model given the output of the loss function, a required component to minimize the loss. A popular choice of optimizer is *Adam*, which has empirically been shown to achieve good performance. It combines the best features of other well known optimizers and can handle sparse gradients on noisy problems [6].

2.1.3 Transfer learning

CNNs require a large amount of training data in order to avoid overfitting. That is, the prediction model would fail to predict future observations reliably if it corresponds too close to a particular dataset. The ability to perform well on previously unobserved inputs is called *generalization* and to achieve it, transfer learning is a helpful learning approach. [3]

Transfer learning is defined as the improvement of learning a new task through

the transfer of knowledge from a related task that has already been learned [7]. Deep neural networks trained on image data sets often show the phenomenon of early layers learn to detect simple patterns, such as edges. These patterns are not specific to a dataset or task but are general to many different tasks, and thus a transfer of knowledge can be done by initializing the weights from a pre-trained model. Additionally, it is common to freeze the first layers of the network and replace only the last layers with the fine-tuning, layers which detect patterns more specific to the dataset.

2.2 Deep learning for camera traps

Camera traps are motion-activated cameras that are generally inexpensive and non-intrusive. The extraction of useful information from camera traps can be a cumbersome process since they can produce millions of images. Fortunately, automatic extraction of interesting information can be performed with computer vision, and more particularly, CNNs.

2.2.1 Related work on camera traps

There is a collection of works done concerning deep learning on camera trap data. Recurring challenges are multi-species images and limited-data problems. In the wild, one can count on many different species captured in the same image due to animals moving in flock. However, both examples using object detection and image classification exist.

Schneider et al. [8] compare two deep learning object detection classifiers, Faster Region-based Convolutional Neural Networks (Faster R-CNN), and You Only Look Once (YOLO) v2.0. They also present a generous summary of previous work on the topic of automating the analysis of camera trap images. From 2013 and on-wards, researchers have tried a wide range of model architectures including AlexNet, the Visual Geometry Group (VGG) network, GoogLeNet and ResNet. However, they point out that all approaches they inspected share the limitation of returning only one output per image, which they state to be unrealistic for meaningful camera trap data analysis. Schneider et al. apply object detection and show that Faster R-CNN outperformed YOLO v2.0 in terms of accuracy, however the trade-off regarding inference speed is evident. Moreover, they test and evaluate if transfer learning is applicable for ecological research scenarios, with promising results. The authors advocate for further work to utilize object detection with Faster R-CNN or one of its successors.

More recently, research within this field tend to explore object detection rather than classification. Cheema et al. embrace object detection in their pipeline for recognition of individuals of patterned species [9]. Further, they point out the benefits of Faster R-CNN to provide robustness to blur, partial occlusions, illumination, and pose variations. To handle the challenge of limited data, they apply

data augmentation and thus thrice the number of training images. More specifically, augmentation techniques used were horizontally flipping, contrast enhancement and random filtering.

Norouzzadeh et al. [10] present a deep *active* learning system which main contribution is a system that simplifies the process of bounding box annotation of data collected with camera traps. For clarity, active learning is a special case of machine learning in which a learning algorithm can interactively ask a user for corrections and continue its learning [11]. Moreover, the work of Norouzzadeh et al. strengthens the assumption that detectors can generalize better than classification models. Their paper hypothesized that an object detection model is less sensitive to image backgrounds. They apply transfer learning based on a pretrained Faster R-CNN model, and inference does not seem to be performed with any computational restrictions [10].

As stated earlier, few explore deep learning for camera traps to be run in real-time, however, some examples were found. Wilber et al. [12] aim to offer lightweight detection algorithms that run on inexpensive mobile hardware. They present their work on building vision tools for field biologists to study threatened species in the Mojave Desert. The two species are Mohave Ground Squirrel and Desert Tortoise and, which is interesting as they enter the subject of detection of small objects. They refer to the open set problem and how it complicates the training, as it is not feasible to create a training set for all possible negative examples that may pass by.

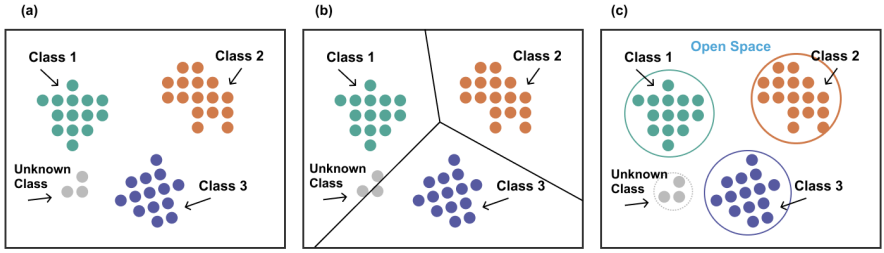
Wilber et al. [12] proposed a lightweight solution for recognition, where they adopt *Support-vector machine* (SVM) as a non-probabilistic linear classifier. Their full systems performs the following steps; images are periodically captured from the video stream, animals within the frame are detected and classified, and finally the results are tabulated and presented to the biologists. The classification has a second stage model to classify subspecies if the first model signaled for an animal of interest. Wilber et al. also confirm that the majority of previous research requires a massive amounts of computing resources, or make inappropriate assumptions that limit the systems for very constrained problems. Further, they point out that while biologist could use off-the-shelf commercial surveillance software, however, those are in general designed for completely different scenarios and thus ill-suited for fauna surveys. An interesting discussion they bring up is that in animal population studies, it is not crucial to achieve perfect accuracy on a per frame basis, as long as its species can be correctly identified across a sequence of captured frames.

In contrast to research with focus mainly on the model, Badrinath et al. aim to propose a more complete architecture for a real time surveillance system [13]. The purpose of their application is to distinguish harmful animals from innocuous ones. For the classification task, their work also utilizes SVM, and histogram of oriented gradients (HOG) used for feature extraction. Unfortunately, no details

on the tracking component of the system are given, although they emphasize its importance as existing surveillance systems in general limit to notify only when an animal enters the scene.

2.2.2 Open set recognition

When handling data from a camera trap, it can be assumed that many species will be captured. Training data should typically include the classes to label, whereas it is an ongoing research topic on how to handle all other objects that may appear. This is referred to as open set recognition, which includes several aspects in the pipeline; from model representations to datasets and evaluation criteria. The main goal is, as shown in Figure 2.3, to cluster the known classes such that outliers will be rejected as unknown classes, rather than label them as the most promising class. [14]



Visualization adapted from Geng et al. [14]

Figure 2.3: *For object detection in an open set domain, the detections in (a) should either belong to one of the known classes or be considered as an unknown class, as seen in (c). Figure (b) shows traditional multi-class classification, where each detection always fall into one of the distinct classes.*

Previous work [15, 16] share a similar approach to achieve rejection for unknown classes in the task of image classification. These approaches are based on the idea to perform projection for all training samples to a single point, a mean activation vector, and thus be able to calculate an *outlier score* based on distance. Hassen et al. [15] propose a loss function to achieve the following two properties; that instances of the same class are closer together, and that instances of different classes are further apart. Their customized loss function enables the use of the same distance function both during training and also for computation of the outlier score as inference is run. Bodesheim et al. [16] implicate that standard multi-class classification is related to novelty detection but does not treat the regions between classes such that they can be distinctly separated. That is, outlier detection is difficult especially when classes overlap in feature space, which is also inferred by Hassen et al. regarding their two beneficial properties.

Miller et al. [17] discuss how previous work on open set recognition mainly focuses on image classification rather than object detection. Especially, it is no longer obvious how to generate a mean activation vector when detections are generated from different image regions of different sizes and resolutions.

2.2.3 Detection of small objects

An important aspect is how much variation in object size the model should handle to work with. Camera traps placed on open plains will often capture objects on a far distance, and careful design decisions are necessary for the model to detected those. As earlier described, a CNN generally down-sample the input at each layer of the feature extraction. Naturally, an already small image does not contain enough information to be down-sampled as many times as a large image. The CNN model Single Shot Multiple Detector (SSD), further described in Section 2.3.3, presents an approach to perform the prediction from the layer depth that makes most sense according to the image resolution [18]. When a pre-trained model is used for transfer learning, one should keep in mind what dataset it is trained on. As for example, objects in the *Common Objects in Context Dataset* (COCO) dataset has a width and height in the range between 40 and 140 pixels, and thus weights for the feature extraction are trained accordingly [19]. That is, the decision has to be made whether it is reasonable to freeze layers or re-train them, depending on the object sizes present in the target domain.

2.3 Edge machine learning

Machine learning is typically run on the server, but moves more and more to edge devices [20]. As for example, sensors can be integrated for usage with self-driving cars [21] and voice-activated device control [22]. Common challenges with deployment on these devices are the reduced computer power, limited restricted memory and eventual battery constraints [20].

2.3.1 TensorFlow Lite

In May 2017, Google announced TensorFlow Lite, a deep learning framework for mobile and embedded devices [23]. Already in early 2019, over 2 billion devices were deployed with TensorFlow Lite. Training of the original TensorFlow model can still take place on a high powered machine. After training, a model file is saved which defines all nodes; its operations, inputs and outputs. Additionally, training produces a checkpoint file which stores a capture of all weights and biases values. These files together create a frozen graph with all nodes frozen except for the input and output. It is then no longer accessible for training, however it is well fitted for inference.

2.3.2 Related work on edge machine learning

Warden and Situnayake released the book TinyML [24] at the end of 2019, both involved in the TensorFlow team at the time of writing. The book guides through a series of example projects, and discusses the common challenges with deep learning on tiny devices. The majority of their implementations are based on ultra-low-power microcontrollers and while Raspberry Pi is rather a micro-computer, valuable insights can be retrieved from this source. Details about the TensorFlow Lite conversion are covered, and optimization options as model *quantization*. Another reduction of computational time mentioned is the possibility to cut off the image pre- and post-processing part of the network, as it may not be needed in the client application.

Object detection and tracking is on the verge of being affiliated with TinyML, due to its computational complexity [20]. Previous implementations which are light-weight exist, which prove that and multi-class detectors can run on mobile devices. For instance, Liu et. al [25] presents a mobile solution where real-time object detection assists an application for augmented reality. Their solution achieves an end-to-end latency of 16.7 milliseconds for each processed frame. Likewise, Königshof et. al [21] developed a real-time vision system that recognizes and tracks lanes, road boundaries, and multiple vehicles in videos, which is able to run in real-time with simple, low-cost hardware.

2.3.3 Efficient detection models

Depending on the use case for the network, its structure is chosen accordingly. A networks size depends on the number of calculations needed, and some models are designed with the purpose of being efficient over accurate. A detection model consists of a base network and a classifier head as seen in Figure 2.4. Efficient base networks are for example MobileNet and Inception, and a typical classifier head to combine with them is SSD [2]. Other fast detection models exist, such as the YOLO network. It is equally fast as the SSD but not as accurate [26]. Moreover, the YOLO network does not work well in real-time applications on non-GPU computers [27].

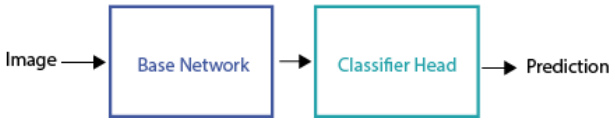


Figure 2.4: *A detection model consists of a base network and a classifier head.*

Another way to increase efficiency of a model is to use quantization. It can reduce latency by simplifying the calculations that occur during inference, potentially at the expense of some accuracy. Quantization works by reducing the precision of

the numbers used to represent a model's parameters. This results in a smaller model size and faster computation. [28]

SSD - Single Shot multiple detector

The SSD is one of the first attempts at using CNN's pyramid feature hierarchy for efficient detection of objects of various sizes. It uses the VGG-16 model for extracting useful image features [29], and on top of that adds additional convolution layers of decreasing sizes. These convolution layers can be seen as a pyramid representation of images of various scales, see Figure 2.5 [30]. Large fine-grained feature maps at earlier levels capture small objects and small coarse-grained feature maps detect large objects. The detection happens in every pyramidal layer, targeting objects of various sizes.

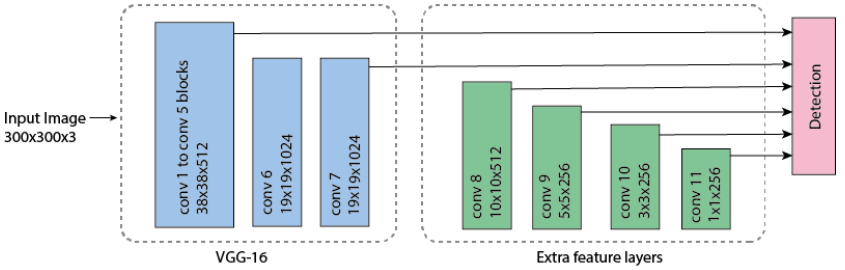


Figure 2.5: *The model architecture of SSD.*

Some object detection systems follow the approach of hypothesizing bounding boxes, re-sample pixels or features for each box and applying a high-quality classifier. While being accurate, this approach is computationally heavy and not fit for embedded devices and is too slow for real-time applications. In contrast, the SSD decreases the output space into a set of default bounding boxes over different aspect ratios and scales per feature map. At prediction, the network generates scores for the presence of each object category in each default box to better match the object shape. [18]

MobileNet V2

MobileNet V2 is a light-weight architecture which utilizes shortcut connections between layers, as well as depth-wise convolutions, both methods to keep the number of mathematical operations low. The model has three convolution layers in a block, see Figure 2.6. The first layer of the block is a 1x1 convolution and its purpose is to expand the number of channels in the data before it goes into the depth-wise convolution. The depth-wise convolution filters the inputs, which is then followed by a 1x1 point-wise convolution layer. The point-wise convolution, called the projection layer or bottleneck layer, makes the number of channels smaller and thus reduces the amount of data that flows through the network. [31]

The input and output of the block are low-dimensional tensors, while the filtering inside a block is done on a high-dimensional tensor. The residual connection exists to help with the flow of gradients through the network and is only used when the number of channels going into the block is the same as the channels going out of it. [31]

Each layer, except from the projection layer, has batch normalization and an activation function named ReLU-6. ReLU-6 is a ReLU with an upper bound at 6 and according to Krizhevsky who first used it, it encouraged their model to learn sparse features earlier [32]. The full MobileNet V2 architecture consists of 17 of the building blocks seen in Figure 2.6, in a row. This is followed by a regular 1x1 convolution, a global average pooling layer, and a classification layer, the full architecture can be seen in Figure 2.7.

SSD is designed to be independent of the base network, and can thus run on top of any base network, including MobileNet. A variant called SSDLite uses

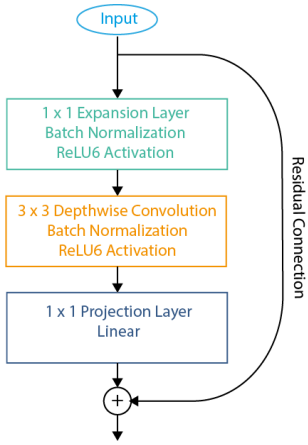


Figure 2.6: *Layers in each residual block of MobileNet V2.*

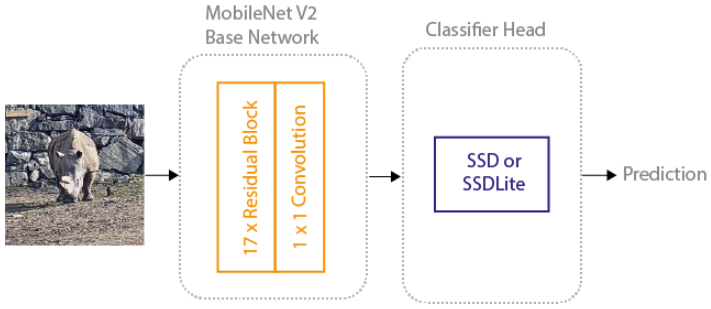


Figure 2.7: *Architecture of the MobileNetV2 with classifier head.*

depthwise separable layers instead of regular convolutions. With SSDLite on top of MobileNet, true real-time results can be achieved (30 FPS or more).

Inception V2

Inception is another light-weight architecture which can be used as the base network in a detection model. A motivation for the first work of Szegedy et al. [33], Inception V1, was to improve the performance of deep CNNs without increasing the network size and computational cost. Especially, their focus lies on extracting appropriate features regardless on the size of the object, which they achieve by the design choice to have filters of multiple sizes operate on the same network level. Figure 2.8 shows the type of module which an Inception model is mainly built up with.

They also included extra 1x1 convolutions to limit the number of input channels and thus lower the computational needs. With Inception V2 [34], the problem on "representational bottlenecks" was tackled, that is, when the convolutions alter the input dimensions to the extent that causes too large information loss. Their solution was to expand the filters to make them wider rather than deeper, as going deeper in a network implicates reduction in dimensions. As for example, the 5x5 convolution seen in Figure 2.8 was replaced with two 3x3 convolutions.

2.4 Evaluation of a detector

In this section, evaluation metrics used within object detection is covered. This includes the terms *precision* and *recall*, which are general within the field of machine learning, as well as metrics specific to object detection.

2.4.1 Precision and Recall

To calculate the average precision of a model, precision and recall are used, defined in Equation (2.9) and (2.10). Precision measures how accurate the predictions are, measured in percentage of correct predictions. Recall measures how

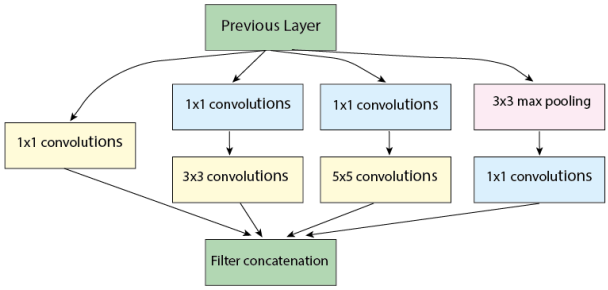


Figure 2.8: *Architecture of the Inception module.*

well the model finds all the positives. An overview of the terms *true positive*, *false positive*, *false negative* and *true negative* can be seen in Table 2.1. As a hands on description of the terms with an example of the rhinoceros class: true positive is when a rhinoceros is detected as a rhinoceros, false positive is when something else is detected as a rhinoceros, false negative is when the detector fails to recognize a rhinoceros, true negative is when the detector does not detect something else as a rhinoceros.

Table 2.1: *Terms to describe the relationship between the predicted detection and the actual.*

		Actual	
		Positive	**Negative**
Predicted	**Positive**	true positive	false positive
	Negative	false negative	true negative

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{2.9}$$

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{2.10}$$

When the number of true positives, false negatives and false positives is determined from the test set, the precision and recall can be calculated. This can be plotted in a precision-recall curve. The *average precision* (AP) is defined as the area under the precision-recall curve and is given by Equation (2.11), where p is precision and r is recall. [35]

$$\text{AP} = \int_0^1 p(r)\,\mathrm{d}r \tag{2.11}$$

To evaluate a detector the metric *mean average precision* (mAP) can be used. mAP is the average value of the AP for N classes, it is defined by Equation (2.12).

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}_i \tag{2.12}$$

2.4.2 Intersection over union

To get an account of how well the detector locates the objects the *intersection over union* (IoU) is calculated, defined in Equation (2.13). IoU is the measure of the overlap of two bounding boxes. It measures how much the predicted bounding box overlaps with the ground truth, see Figure 2.9. A threshold value is set to label the detection as true positive or false positive. If the IoU is greater than the threshold it is considered as a true positive, and else a false positive.

$$\text{IoU} = \frac{area\ of\ overlap}{area\ of\ union} \tag{2.13}$$

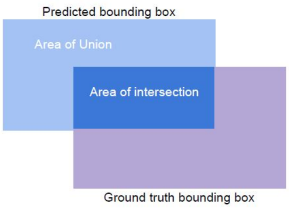


Figure 2.9: *Area of the union and area of the intersection.*

2.4.3 COCO evaluation

The public dataset COCO is widely used for training image-based models. It has 80 object categories and is often used to compare state-of-the-art models [36]. COCO's detection evaluation presents 12 metrics given under four categories [37]. COCO uses an interpolated AP, where the AP is an average of multiple IoU:s, that is, where AP is mentioned it is actually referred to mAP since it is averaged over all classes. The evaluation is given in the following categories:

- Average Precision (AP) - given over different IoU thresholds
- Average Recall (AR) - given with 1, 10 or 100 detections per image.
- Average Precision (AP) across scales - small, medium and large objects defined by pixel size.
- Average Recall (AR) across scales - small, medium and large objects defined by pixel size.

2.4.4 PASCAL VOC evaluation

PASCAL VOC is measured with mAP, where a value close to 1 reflects good performance [38]. In order to use mAP in object detection, the predicted boxes and classes are sorted in decreasing order of probability and matched with the ground truth boxes and classes. The IoU is calculated and to be considered a correct detection the IoU needs to exceed 50%. The match is predicted as true positive if and only if it has not previously been seen, to mitigate duplicate detections of objects.

2.5 Event extraction

To prevent the server from overflooding with data, it is crucial to ensure that only relevant data is uploaded. A single-image detector has no temporal aspect, however its results can be evaluated over time. For this purpose, different approaches to extract high-confident events are discussed in this section.

2.5.1 Precision for single image versus for image sequence

Previous research presents approaches to achieve high precision on a single image, like seen in [19, 39], their models got better precision by cropping the image around interesting objects, or include more of the background around bounding boxes. Both are techniques which imply resizing the image patch to cover more or less pixels, so that features can be extracted at the most appropriate scale. However, as Wilber et al. [12] stated, it might be more important to achieve high precision over a sequence of images when it comes to animal surveillance. Likewise, Wu et al. [40] describe the challenge on object detection in video as to utilize the temporal continuity of videos to improve the accuracy. The task to treat detections over time is often referred to as *object tracking*. Exactly how object tracking is applied, and whether additional processing steps are included differs quite between implementations.

2.5.2 Object tracking

In addition to the task of detecting objects in an image, it is often valuable to track them in a sequence of consecutive images [41]. With tracking, objects can be assigned with unique ID numbers assigned to them and extra information can be retrieved, such as how long a particular individual has been present. A set of challenges are associated with object tracking, as object occlusion and measurement noise due to occasionally missed detections. Complete tracking solutions generally consist of two main components, a detector and a tracker. The task of the detector is to obtain information about detected objects, like bounding box coordinates, which is fed as input to the tracker. The method for the detections itself varies. For simple objects, processing as edge and contour detection can be used for separation. In more complex scenarios, as with several classes and various backgrounds, detection with pure image processing is difficult to achieve [41]. As described in Section 2.1.1, machine learning methods are widely used for the purpose of object detection nowadays.

Recently, deep learning based approaches have arisen also for the tracking component. Emami et al. [42] presents an overview of machine learning methods for multi-target tracking. These methods are based on CNN with several consecutive images as input, that is, resourceful training data. For this work, solutions that are not associated with a set of fixed classes are aimed for, and among those are standard Kalman filtering and tubelet proposal.

2.5.3 Kalman filter for box tracking

The Kalman filter is a recursive algorithm to provide an estimate of the current state, given previous measurements and predictions [43]. Thus, when a new measurement is not available, the state can still be estimated. Bewley et al. [41] presents how Kalman filters can be used for box tracking, which is convenient as bounding boxes are generally the output from object detection. In their work, the state to be estimated with a Kalman filter represents the bounding box location for an object, its position and velocity over time.

2.5.4 Tubelet Proposal

The counterpart to bounding box proposals in static object detection is called tubelets for video tracking, that is, sequences of bounding box proposals as seen in Figure 2.10. Within object detection from video, a major topic is how to propose and filter these tubelets.

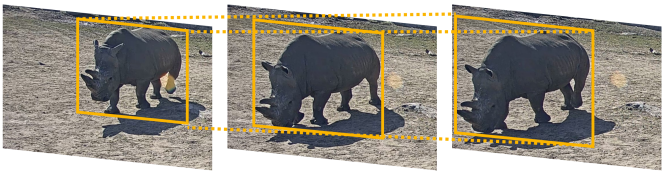


Figure 2.10: *Visualization of the tubelet definition. Consecutive detections on the same object are handled as a tubelet. The aim of a tubelet proposal algorithm is to propose which detections should be associated*

ImageNet has a challenge on object detection from video, referred to as the VID task [44]. Kang et al. [45] introduce a complete framework for the VID task based on still-image object detection and object tracking. In VID, each video frame contains an unknown number of object instances and classes. This is closer to real-world scenarios than many previous problems worked on, according to their study of related work. The proposed solution of Kang et al. consists of two main modules, one that proposes tubelets, and one which classifies the tubelets and performs re-scoring. The results are promising and show consistent performance improvement over still-image detections. Unfortunately, the module for re-scoring requires one temporal convolutional network per class, trained on annotated video data. As this kind of dataset requires substantial resources, it is impractical for a solution that wishes to be adapted to arbitrary classes. However, parts of their solution are of interest, as the implementation steps they refer to as "High-confidence Tracking" and "Tubelet Perturbation and Max Pooling". That is, they describe how detection boxes can be treated and filtered to match them in consecutive frames. As for example, an interesting insight is how they allow a relatively low threshold of 0.1 to keep a track. Moreover, if a detection overlaps more than 30% with an existing track, it will not be chosen as a new tubelet anchor.

3
Method

As is often the case with machine learning, development is an iterative process. After a round of data collection and pre-processing, training of the very first model could be run. Model requirements were identified as it got clear what scenarios were critical, i.e. what images the detector could not handle, and design choices were made thereafter. Moreover, a light-weight client application was developed, which includes an algorithm for event extraction as well as the server communication.

3.1 Tools and frameworks

For training of the neural network, the deep learning library TensorFlow was used along with the TensorFlow Object Detection API, which both simplifies the process of training models to detect objects by using pre-trained models. Google Colaboratory was used to edit and run Colab notebooks as it is hosted online and offers free use of GPU, where a notebook is a document composed of cells, each of which can contain code, text, figures and more [46].

LabelImg [47] was used to annotate the images for training, which is a graphical image annotation tool written in Python and available as open source. Further, all components for the tracking algorithm are written i Python.

3.2 System overview

Figure 3.1 illustrates the workflow of the full system. The different development areas are:

- **Detection.** Training a neural network, or practically, a mathematical model that takes an image as input and generates bounding boxes with associated scores and object class. Section 3.3 describes the process of training the detection model.
- **Tracking.** Designing an algorithm to keep track on detections over time with the purpose to extract events. The method to establish object tracking is described in Section 3.4.
- **Edge application.** Building a complete application that manages both the video stream, object detection, tracking, and server communication. Section 3.5 provides a short description of the edge application.

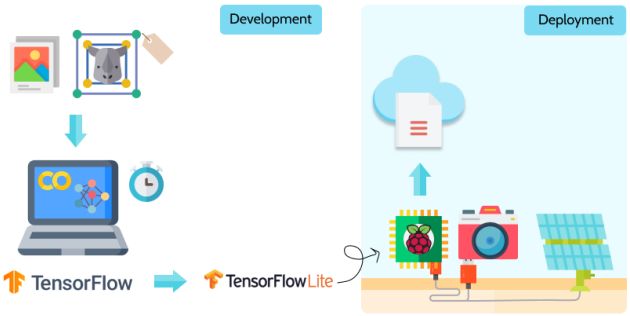


Figure 3.1: Workflow of the full system. The development stage illustrates the process to train and prepare a CNN model for deployment. Scripts for tracking was integrated directly with the edge application on a Raspberry Pi.

3.3 Object detector

A model was built for detection of savannah animals and humans, based on a pre-trained model using transfer learning. The first steps of building a detector was to collect and pre-process data. Four models were selected and compared against each other based on some objectives and the best performing model was selected for the application.

3.3.1 Identified model objectives

The following model objectives were identified to aim for:

- Handle multiple spatial scales and aspect ratios.
- Model should have low inference speed (< 100 ms) and small model size (< 100 MB).
- Achieve an AP of at least 70% for each class.

3.3.2 Data collection

Classes included in the training data were rhinoceros, lion, leopard, zebra, buffalo, giraffe, elephant and human. The data was collected from the sources seen in Table 3.1 and visualized in Figure 3.2. Source 1 and 7-12 are camera trap images, which represents the target setting. In particular, source 1 is available for direct access, as it is a partner of the project. However, as Cheem et. al. [9] stated and is commonly known, camera trap images typically suffer from blur, poor lighting conditions and occlusion of the animals. Thus, the complementary sources 2-6 were included to ensure that the training data contains good images of the main features of the animals. Images from Google and YouTube were collected by searches on *buffalo*, *humans on the savannah*, *rhinoceros* and *giraffe*.

Table 3.1: *Images collected from different data sources.*

Nr	Source	Images	Species Collected
1	Kolmården Zoo	200	Rhinoceros, Zebra
2	Animals with Attributes [48, 49]	1677	Elephant, Giraffe, Leopard, Lion, Rhinoceros, Zebra
3	iNaturalist [50]	33	Rhinoceros
4	Google	152	Buffalo, Human
5	YouTube	441	Buffalo, Giraffe, Human Rhinoceros
6	Unsplash [51]	320	Buffalo, Elephant, Giraffe, Leopard,Lion, Rhinoceros
7	Wildlife Image and localization Dataset [52]	400	Giraffe, Zebra
8	Snapshot Mountain Zebra [53]	108	Buffalo
9	Snapshot Enonkishu [54]	57	Buffalo, Elephant
10	Snapshot Kruger [55]	369	Buffalo, Elephant, Leopard, Lion, Zebra
11	Snapshot Karoo [56]	145	Zebra
12	Snapshot Serengeti [57, 58]	943	Buffalo, Leopard, Lion, Rhinoceros

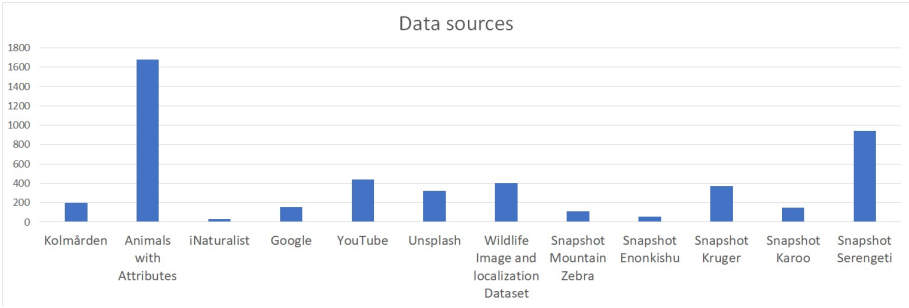


Figure 3.2: *Overview of collected images from data sources.*

In total, 4845 images were collected and annotated. However, a script was written to sort out images which had bounding boxes covering more than 90% of the area to remove images which were taken to close up to the animal. Moreover, a set of images was picked for validation purposes. In the end, the training dataset contained 4008 images. The distribution over the classes can be seen in Figure 3.3.

Camera trap images have been prioritized when selecting the data sources, as training data similar to the target domain is beneficial for the model performance. Figure 3.4 shows example images from the eight different camera trap locations used in the data collection.

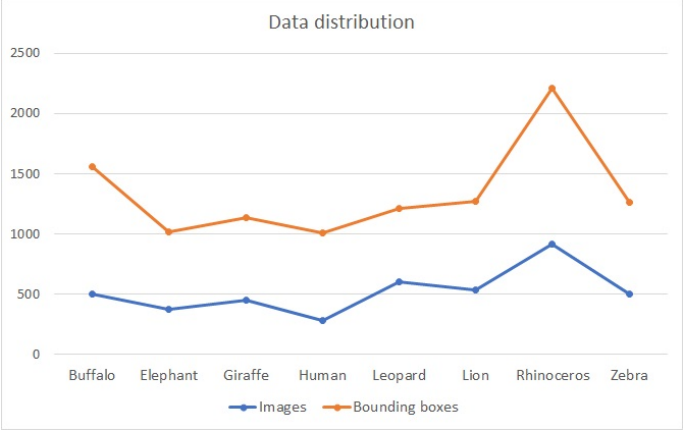


Figure 3.3: *Data distribution over the classes. The vertical axis represents the number of images for each object class*

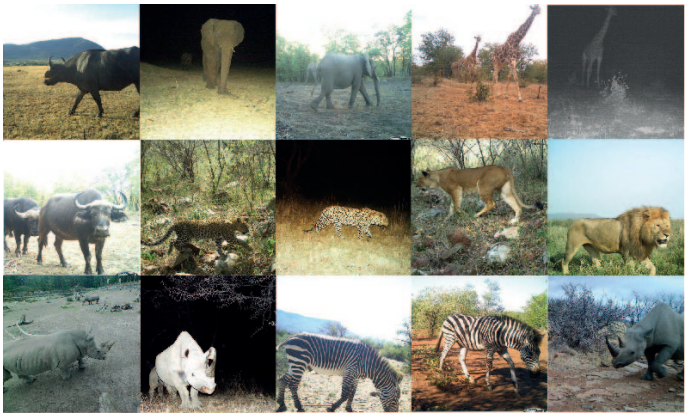


Figure 3.4: *Example of images collected from camera traps and used for training.*

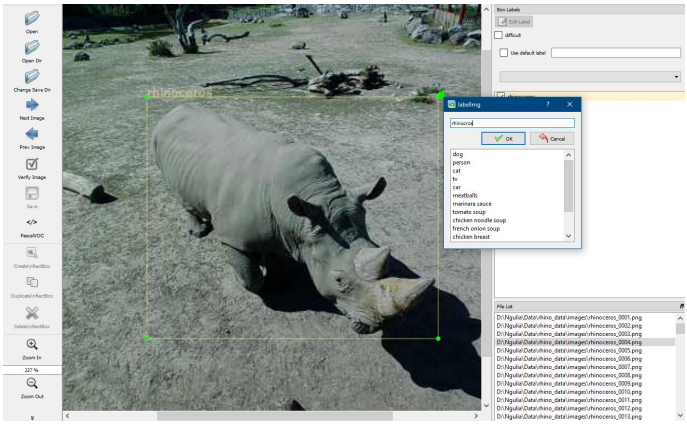


Figure 3.5: *Annotation of images in LabelImg.*

3.3.3 Data preprocessing

All images were resized and cropped to 400x400 pixels. The images were then annotated in LabelImg, see Figure 3.5, where bounding boxes are drawn tight around objects of classes to detect. The program generates an XML-file in the format of PASCAL VOC with object annotations. Each file contains the image name and the width and height, and one annotation object per animal in the image. Each annotation holds bounding box coordinates, describing the position of the animal in the image. The bounding box is specified with the values: *xmin*, *xmax*, *ymin* and *ymax*. These values make up the points acting as corners of the bounding box.

filename	width	height	class	xmin	ymin	xmax	ymax
rhinoceros_0013.jpg	400	400	rhinoceros	186	130	389	400
rhinoceros_0014.jpg	400	400	rhinoceros	23	162	399	400
rhinoceros_0015.jpg	400	400	rhinoceros	118	112	315	400
rhinoceros_0016.jpg	400	400	rhinoceros	106	101	213	231
rhinoceros_0016.jpg	400	400	rhinoceros	0	255	151	400
rhinoceros_0018.jpg	400	400	rhinoceros	222	53	260	76
rhinoceros_0018.jpg	400	400	rhinoceros	0	74	31	100
rhinoceros_0018.jpg	400	400	zebra	145	168	400	351

Figure 3.6: *Table with annotation of the training and testing images.*

Table 3.2: *Comparison between pre-trained models.*

Model name	COCO mAP	Speed (ms)
ssd_mobilenet_v2_quantized_coco	0.22	29
ssd_mobilenet_v2_coco	0.22	31
ssd_inception_v2_coco	0.24	42
ssdlite_mobilenet_v2_coco	0.22	27

The XML files were converted into a CSV file holding all the annotations from the dataset. Each image may have one or multiple annotations, that is, one or several rows in the CSV file. An extract of the CSV table can be seen in Figure 3.6. The data was separated with a ratio of 9:1 into train and test. Importantly, when dealing with a multi-class detector, an even distribution over the classes in the train and test data is important. Thus, the splitting was achieved through *stratification*, a method which take class attribute into account. Annotations were then converted into *TFRecord* format, a format created by TensorFlow for the purpose to read linearly efficiently in the training phase [59].

3.3.4 Training CNN models

Four models were trained with slightly different architectures. MobileNet V2 was set as base network for three models, and Inception V2 for the fourth. Further, one of the MobileNet models was quantized and one used the Lite version of its SSD classifier head, which are both approaches for efficiency. Table 3.2 displays the speed and COCO mAP of the pre-trained models downloaded from TensorFlow's official GitHub page [60].

As earlier mentioned, TensorFlow's Object Detection API was used for training. To apply transfer learning, a model pre-trained on the COCO dataset was loaded. Such model is defined by a set of files:

- A graph protocol, defining the structure of the model.
- A checkpoint file, containing values of the weights.
- A configuration file, with all configurations and hyperparameters that were used to train the model.

The graph protocol and the checkpoint file were used for further training of the model, together with the configuration file. However, as the configuration was previously set for a large dataset, it was partly overridden. Fundamentally, the number of classes for the new detection problem was set in the configuration, as well as the corresponding label map. When training starts, new images are fed as input to the graph, and weights get updated by backpropagation. New checkpoint files are saved regularly and when the model converged, a new frozen graph was exported.

The following hyperparameters were set in the configuration file:

- **Optimizer**: Adam with initial learning rate of 0.0002 [3, 6].
- **Regularization loss**: $L2$
- **Localization loss**: Smooth $L1$ localization loss
- **Classification loss**: Sigmoid cross entropy classification loss
- **Batch size**: 64
- **Image input**: 300x300 pixels
- **Data augmentation**:
  - random horizontal flip
  - random rgb to gray

The numeric output of the last linear layer in the network is converted to probabilities with the Softmax function.

Model design tests

To prove the hypothesis regarding the benefit of transfer learning, a model with all the same data and parameters was trained, except that no pre-trained weights were initialized. Instead, the weights are initialized with a *truncated normal initializer* which generates a random normal distribution, given mean value and standard deviation as hyperparameters. The mean was set to zero and the standard deviation to 0.03, which are the default values in TensorFlow's Object Detection API.

For tests concerning the open set problem, a MobileNet V2 model was trained without the elephant and buffalo classes. A small test set with unseen data was used to compare the detections generated by this model and the normal MobileNet V2 model. Likewise, a model was trained without applying data augmentation.

3.3.5 Model evaluation

As training was running, its evaluation progress could be monitored in TensorBoard which provides graphs over the training progress. To be considered as a well performing model, the AP for each class was desired to be as high as possible and the loss function to be minimized. Initially when training the models, COCO evaluation was used to tune the hyperparameters in the configuration file since it gives an estimate of how well a model performs on small versus large objects. Later, when more classes were added, the evaluation was performed with Pascal VOC evaluation, as PASCAL VOC displays the AP for each class and more information about the loss functions.

Loss values and AP for the classes were taken into consideration when selecting what model to use. As the application is dependent on an efficient model, the models were also evaluated in the object tracking task, with the purpose to investigate which model had the shortest inference time and thus can process the streaming video with a higher sampling rate. This evaluation is further discussed in section 3.4.2.

3.4 Object tracking

Two methods were tested for the purpose of object tracking, Kalman filter for box tracking and tubelet proposal. It soon became clear that while Kalman filter has the advantage to predict the state at detection failures, this exact object tracking may not be necessary. After all, the ambition was mainly to detect whether an object was present or not, rather than following its exact movement. As Kang et al. [45] describe tubelets, it seemed to match well with the needs for this project and was chosen for the final implementation. However, as a light-weight and more general solution was aimed for, no additional networks were used but simply the information given by an arbitrary object detection model: bounding boxes, classes, and scores.

To repeat the definition of a tubelet, it is a set of detections over consecutive frames which are assumed to belong to the same object. Exactly how this data association is achieved differs between implementations, as well as how the tracked tubelets are managed. Below follows a description of how object tracking with tubelets was implemented.

3.4.1 Tubelet proposal implementation

Three classes were implemented to build up the data structure for the problem:

- **Tubelet** - an object which holds all detections that has been estimated to match each other, based on bounding box attributes and detection score. As an active tubelet, the detections may signal for different classes, however, the true class should appear in majority.

- **Event** - a summarized version of a tubelet which is not longer active, as, with memory efficiency in mind, keeps only the information needed to filter and upload events. Importantly, it has a fixed object class, a confidence value, a representative image, and time interval information.
- **TubeletManager** - keeps track of all active tubelets. It can register and deregister tubelets. Importantly it holds the update function which matches detections to active tubelets and finishes tubelets which have not matched for a certain number of frames.

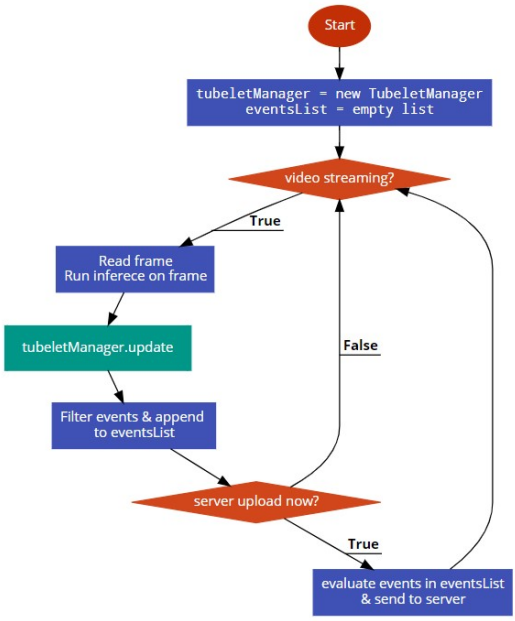


Figure 3.7: *Flow-chart of how the tracking solution was integrated with the CNN model inference during video streaming.*

Figure 3.7 shows a flow-chart of how the tracking solution was integrated with the CNN model inference during video streaming. Below are some implementation details which are commonly discussed for a object tracking solution:

- **High-confidence anchors** - detections with high score are chosen as anchors, i.e the starting detection, for a new tubelet. New tubelets are not initiated if their anchor box has an IoU percentage larger than 0.3 towards the latest detection of another tubelet.
- **Matching of tracking** - every new bounding box with score above a threshold of 0.1 will match existing tubelets based on Euclidean distance. If two detections have the same tubelet as its closest, priority will go to the closest detection.

- **Re-scoring and final class decision** - as a tubelet may sample several object class proposals during its active time, the most occurring class is chosen. This happens since the detection of an animal alter in accuracy over time, as seen in Figure 3.8, and sometimes produces false negatives. Moreover, the average score for the chosen class is calculated to represent the *confidence score*.

- **Release of tracking** - tubelets without a match within the last 2 seconds are deregistered.

The hypothesis behind this implementation is that if an animal is sporadically visible, its detections will be included in some tubelet and after the tubelet summary, the filtration will reveal its character. Filtering that is applied to the tubelets to make events are that the average accuracy over the detections needs to be over 0.5 and that there are at least 5 frames in the tubelet.

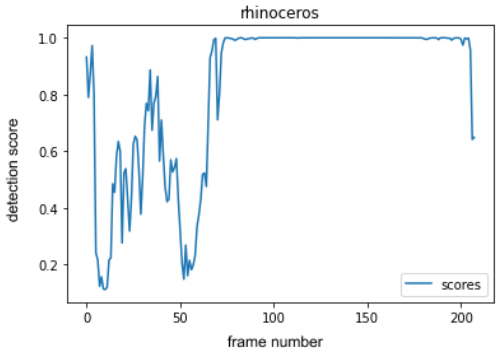


Figure 3.8: *Example of detection scores over time for a tubelet which tracks a rhinoceros. Notice how the score is sporadically low, which the tubelet need to have accommodation for to continuously follow the object.*

3.4.2 Real-time tracking aspects

As the application is to be run continuously, the inference time plays an important role. Given the FPS for the input video and the model inference time, it could be determined how many frames to skip to keep up in real-time. Consequently, the more frames that are skipped, the harder it became to match detections due to object movement. On the other hand, a model with longer inference time generally reflects higher detection accuracy. Therefore, a part of the method to produce a tracking solution was to save results for both model architectures to compare.

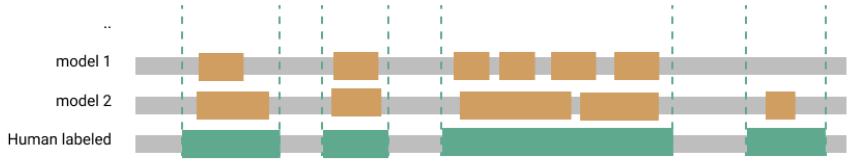


Figure 3.9: The objective of tests on recorded videos is to get a measurement of how well the different models can extract events in relation to the ground truth.

Tracking tests

Events in a set of video sequences were manually annotated by specifying a set of time intervals and the number of rhinoceros visible. Each time a new rhinoceros entered the frame, it was considered as a new event. Likewise, an event finishes when the rhinoceros leaves the frame. During evaluation, the event extraction was approved if it produced an event in that time interval. Figure 3.9 shows an example of how an event extraction of a video sequence is compared with the ground truth events.

To further test the models in tracking they were run simultaneously on a real-time video stream from Kolmården Zoo for 8 hours. The events that the application judged as interesting were sent up to a Dropbox server for manual control. The false positives for the respective model were counted since it is not desired to receive a lot of false alarms.

3.5 Edge application

The edge application is a Python program, with instructions as follows:

Before main loop
Initiate video stream with OpenCV, load the detection model from the tflite-file and initiate server connection, for example, to Dropbox. The program parameters are set, as which classes to look for and how often to upload events.

Main loop
Read a frame and prepare input tensor (resize+normalize), invoke the model and receive output tensor. This output tensor contains the resulting bounding boxes, classes and scores which are used to annotate the frame, and are sent to the update function of the tracking component. Extracted events are stored to a temporary event log until the next server upload.

4

Results

Results for a set of CNN models are presented, both for still images and also in the setting of real-time detection where the tracking component is added. While all models are suitable for edge-device applications, the impact of inference speed versus model accuracy for object tracking is presented.

4.1 Insights on qualified training data

Research insights in combination with evaluations with different datasets gave recommendations for multi-class detection as follows:

- It is vital to split the data such that the distribution for a given class is even between train and test. If not, validation results will give an incorrect measurement of the actual performance of the model.
- As expected, the model which was trained without transfer learning shows slow training progress. After the same number of training steps as when the pre-trained model converges, this model still has mAP values less than 1 % for all classes. With transfer learning, an acceptable result was achieved with only 300-500 images per class after 80 epochs.
- Given the small training set, the addition of augmented data increased the model mAP from 0.76 to 0.82. Figure 4.1 shows this results, as well as the AP comparison for each class. This result was achieved with the only difference being that the augmentation techniques *horizontal flip* and *gray-scale* were randomly applied and generated more training examples.
- Model accuracy increases as similar object classes are included in the training dataset. Figure 4.2 shows how the model trained without buffalo and

elephant generates incorrect detections of rhinoceros, as it is the most similar class (in comparison to giraffe, leopard, lion, human and zebra). By including the buffalo and elephant classes in the training dataset, the model accuracy was significantly better. Note that this model was trained with assumption of a closed set of animals.

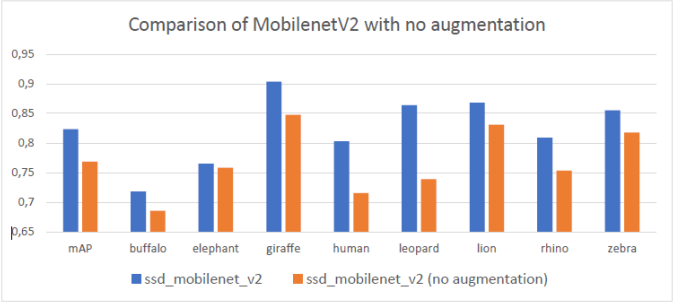


Figure 4.1: *Improvements in AP with the addition of augmented data*

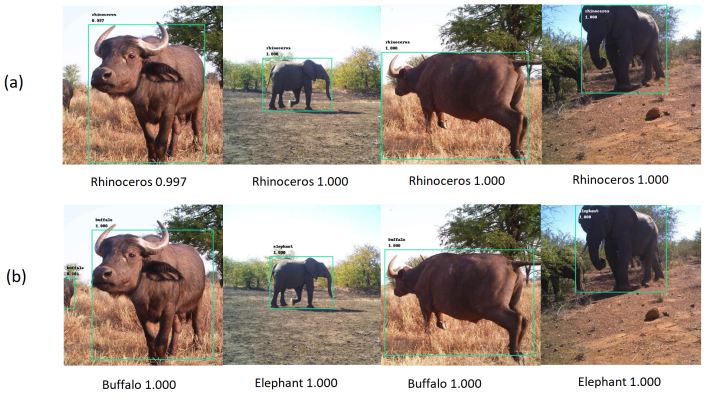


Figure 4.2: *Comparison of models trained with and without buffalo and elephant classes, in a closed set setting. The detections in row (a) were generated with a model trained without a buffalo nor elephant class, and all are incorrectly detected as rhinoceros with score values close to 1. Likewise, row (b) shows results from a model trained with all the same model parameters, except for the addition of the two classes, and it correctly detects all four images*

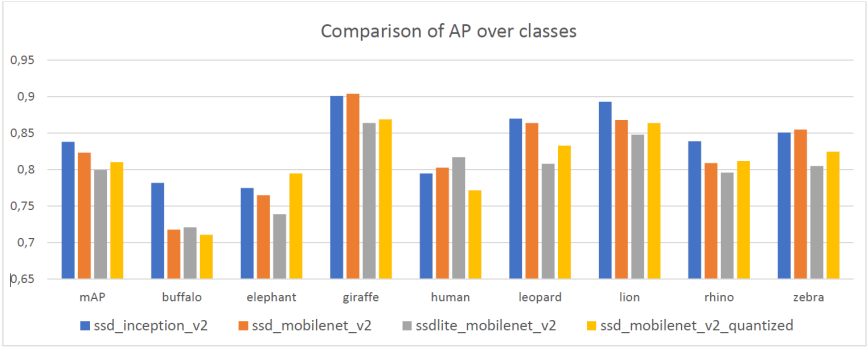


Figure 4.3: *Comparison of mAP per class for the different models.*

Table 4.1: *Inference time per model.*

Model	Raspberry Pi (s)	CPU (s)
ssd_inception_v2	0.804	0.267
ssd_mobilenet_v2	0.314	0.183
ssd_mobilenet_v2_quantized	0.316	0.174
ssdlite_mobilenet_v2	0.314	0.177

4.2 Results from training

From the training some quantitative and qualitative results could be found. Figure 4.3 shows a comparison of mAP and AP over classes for the different models. The inference time for each model is presented in Table 4.1.

4.2.1 Qualitative results

Figure 4.4 presents successful detections on the rhinoceros class, from evaluation data. These examples include cases where the rhinoceros is partially occluded and is present at a relatively far distance. For the two images where the rhinoceros are farthest away, their detections are only 27×27 and 30×48 pixels respectively. These sizes fall within, and slightly above, the scale "small" in COCO's detection evaluation which is 32×32 pixels.

The detector has some difficulties in certain situations, for example partial occlusion, objects far away or when the object is camouflaged by its surrounding. Figure 4.5 shows a set of images that were difficult to detect correctly.

4.3 Results from event extraction

Two main aspects to evaluate an event are 1) to look at the confidence score, and 2) to measure the overlap with the ground truth event. Ideally, a detected event

would cover the full time span of which the animals are in the field of view. Results regarding how the models split, miss to detect events were extracted are presented in this section.

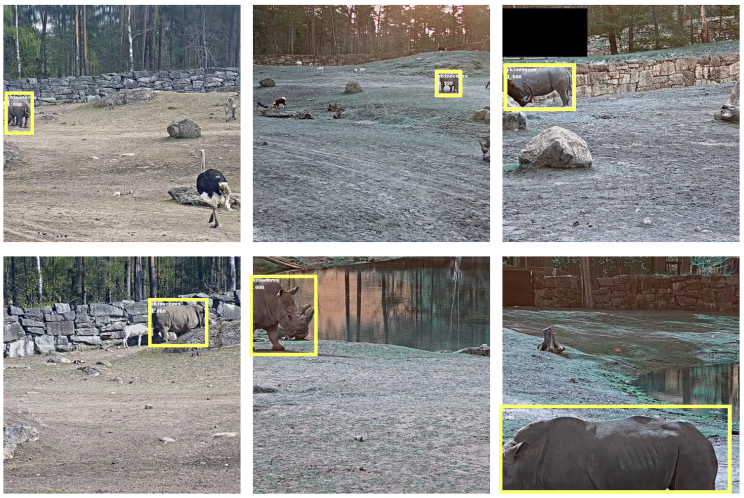


Figure 4.4: *Examples of successfully detected rhinoceros on unseen data.*

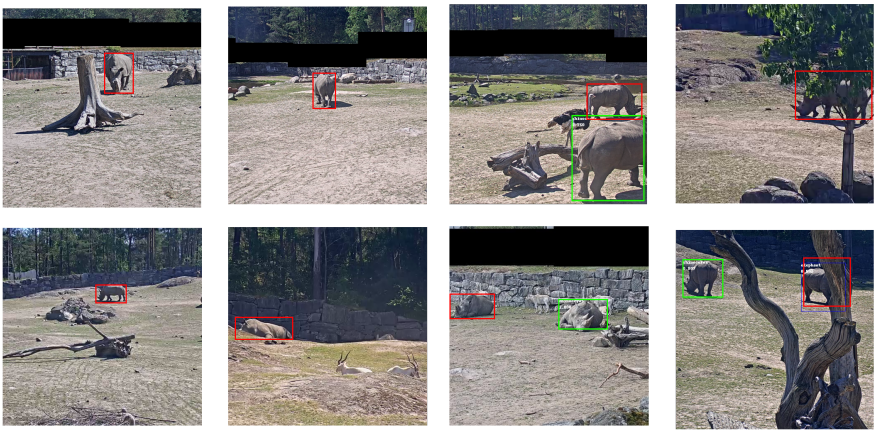


Figure 4.5: *Examples of images that were not properly detected. Red boxes mark incorrect or missed detections while green boxes represent correct detections. The images show that occlusion and and objects far away are difficult to detect.*

4.3.1 Results from test with human labeled videos

Three videos were annotated manually by defining time spans for where an event should occur, as well as the number of animals present. Inference was run 3-5 times per second for the MobileNet models and 2-3 times per second for the Inception model. The relatively low FPS may be caused by running the tests on videos instead of live-stream, due to time spent on decoding. The results from the test with the manually labeled videos showed that MobileNet V2 quantized was the only model that missed an event, a false negative, see Test 3 in Figure 4.6. The network that splits the event the least amount of times is MobileNet V2. SSDLite MobileNet V2 and MobileNet V2 Quantized split events the most.

To get an estimate how confident the models are the mean accuracy for those images with the correct classification are compared with the ground truth, together with the standard deviation it can be seen in Figure 4.7.

Figure 4.6: *Result from the event extraction tests on three videos, comparing the predictions from the models with the manually annotated ground truth.*

Figure 4.7: *Mean accuracy of correct identified images in event with standard deviation calculated with the n-1 method which gives an estimate of the mean deviation.*

4.3.2 Results from real-time video stream

Object detection and tracking were run on a video stream from Kolmården Zoo simultaneously for the four models during 8 hours. The events they extracted were uploaded to a server where they could be analyzed. The metric to look at here was how many true positives, false positives and false negatives the models produce. The results from this test can be seen in Table 4.2 where MobileNet V2 generates the least amount of false positives and MobileNet V2 Quantized the most. Inception and the stardard MobileNet miss to detect two more events than the quantized and SSDLite versions of MobileNet, which also split events more often. With the true positives, false positives and false negatives the precision and recall can be calculated.

Table 4.2: *Detection of true positives (TP), false positives (FP), false negatives (FN) and event splits with precision and recall calculated.*

Model	TP	FP	FN	Splits	Precision	Recall
ssd_inception_v2	19	6	13	1	0.76	0.59
ssd_mobilenet_v2	24	4	13	1	0.86	0.65
ssd_mobilenet_v2 (Q)	28	12	11	4	0.70	0.71
ssdlite_mobilenet_v2	26	7	11	2	0.78	0.70

4.4 Edge device compatibility

Examples of edge devices which the model can run on are smartphones and Raspberry Pi. Figure 4.8 shows a demonstration of the detector in a simple application. The setup on Kolmården Zoo can be seen in Figure 4.9, where a pan-tilt-zoom (PTZ) camera surveys the rhinoceros habitat. In the electrical cabinet which is self-sufficient, powered by a solar panel and running its own network, a Raspberry Pi receives the video stream and analyzes the frames, extracting events.

The events are uploaded to a Dropbox server and available for further usage, as for example to be fetched via the Dropbox API and displayed on a web-page as a surveillance monitor.

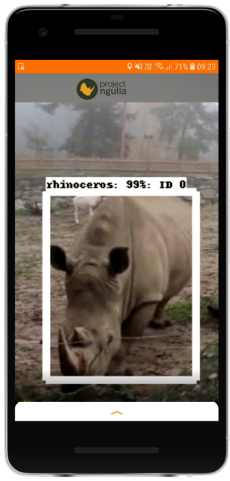


Figure 4.8: *A TFLite model can be integrated in an Android/IOS App for demonstration of the detection model.*

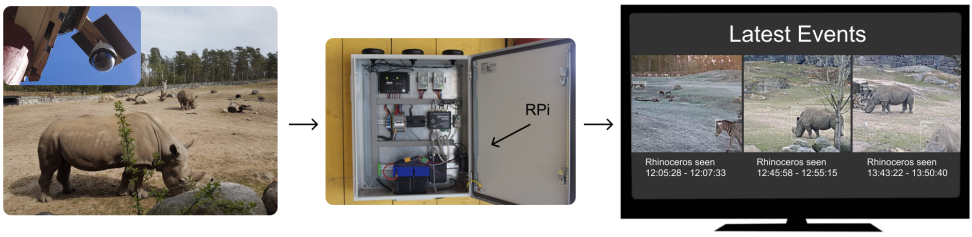


Figure 4.9: *The video stream from a camera is sent to a Raspberry Pi, powered by batteries charged by a solar panel. The video stream is processed and analyzed by the detector and the tracking algorithm. Events are continuously sent to a server and are, as one use case, displayed on a monitor.*

5 Discussion

This chapter contains discussions about the results and the method, as well as ethical and societal aspects related to the project.

5.1 Results

A contribution of this work is insights which are of importance for the deployment of a CNN model in a real-world setting. Annotated data is demanding in terms of resources and practical insights on what image characteristics needs to be represented in the training data has been presented. Moreover, a client-side solution for animal detection and tracking was proposed and evaluated.

5.1.1 Quality of detection models

Overall the four models achieved satisfying results with high mAP from training. Decisive for the quality evaluation was how likely the models were to produce false positives. The models all have some flaws regarding incorrect detections that appear occasionally, however, it happened most frequently for the quantized and the SSDLite model compared to the SSD MobileNet V2 and SSD Inception models.

MobileNet V2 was selected as the final model for the application. This due to its traits of generating a small amount of false positives and not splitting an event into smaller ones. Even though it missed to detect some events compared to the other models, the false negatives seen in Table 4.2, the low amount of false positives reflect that it has higher precision. It also performed well in the tests with labeled video events, given that it detected all events. Further, the model file size

is 25 MB for MobileNet V2 and 55 MB for Inception. MobileNet V2 is thus less than half the size of the Inception network which is an advantage considering the goal platform.

A good indication of the model quality is the versatility of detection capability shown in Figure 4.4. Rhinoceros are correctly detected both at close and far distance, as well as occluded, features which agree with the identified model objectives. These results are achieved with a model with a size far less than 100 MB. However, the inference speed of the models unfortunately reaches above the desired limit of 100 ms.

While the models perform acceptable results, they have potential to be further fine-tuned, and especially the optimization of hyperparameters could be explored more systematically. While several variations were tested and resulted in improvement, further hyperparameter tuning is encouraged before deployment of the final model. The network is not able to detect objects very far away with high reliability, which may be an objective to aim for.

5.1.2 Expected results in deployment

In a closed habitat like the one at Kolmården Zoo, there is a known number of species in the enclosure. To launch the system in such an area, the model could be extended to include a class with animals that are not in the current model. Such class could be broader and include goat-like species such as water buck, antelope and gazelle. In contrast to the case of an enclosure, false positives are expected in an environment with a more varying fauna and a larger number of unknown species. It is thus motivated to further look into the open set problem and ensure that unknown classes are rejected at detection and not classified as the class that it resembles the most.

5.1.3 Quality of event extraction

A hypothesis was that it is not crucial to look at perfect accuracy per image frame basis, rather it is of higher importance that it correctly identifies targets across a sequence of captured frames. As the events are filtered so that the mean accuracy for the events are higher than 0.5, it increases the possibility that events sent to the server are true positive events. By observing the real-time detection, it was noted how false positive detections sporadically appeared, but how they were successfully rejected by the event extraction. The test on a real time video stream, see Table 4.2, showed that for the time span of 8 hours only four false positives were extracted for the selected model SSD MobileNet V2. When analyzing the false positives it could be seen that they were all of a species that had not been included in the training set, more specific an antelope type of animal, which was thus an unknown class for the network.

5.2 Discussion of method

The method includes the whole chain from data collection and annotation, model training, and implementation of event extraction from object tracking, to finally testing of the system.

5.2.1 Data collection

While the supply of images with humans is quite unlimited, not as many are captured in savannah-like nature at far distance. At least images of this kind are rarely gathered as a dataset. A possible option is to re-train and update the model after data on the target location has been captured.

5.2.2 Annotation difficulties

In many cases it was not trivial how to annotate the images, especially animals that move in flock. Further, as an animal approaches or disappears towards the horizon, at one point it is no longer relevant to annotate it. That is, the same object could appear several times in the training data and at one time update the weights as it is a positive detection, and next time as a negative detection. On the other hand, it is hard task to collect a sufficient dataset without the use of this kind of sequential images which are produced by camera traps.

5.2.3 Detection approach

As it can be interpreted from section 2.2.1, there are several options whether how to detect and classify animals. A distinction can be made between models which perform image classification or object detection, for this project the latter was chosen since several species are expected to be present simultaneously. In retrospect, this path was never doubted during development or testing. In particular, the location attributes provided by object detection was highly beneficial for the tracking component.

5.2.4 Tracking approach

Since Kalman filters are commonly used for object tracking, it is relevant to discuss whether the choice of tubular proposal can be justified. The main advantage of using a Kalman filter would have been the possibility to estimate the new location of an object, based on previous observations. During development and testing, these estimations were concluded to rather complicate tracking, as the estimated location tend to drift incoherent with the animal movement. In comparison with, as for example, cars on a highway, one can conclude that mammals move more unpredictably. This amplifies as inference runs only a few times per seconds, and the animal has even more room to change direction. On the other hand, the Kalman filter would be helpful in situations where a single animal is quickly passing by. It may be the case that Kalman filters could add more

value then what was revealed during the tests, as further exploration of alternative state definitions or velocity models could be performed. However, for this use case where an exact position is not necessary, tubelet proposal is satisfactory to extract events for when specific object classes are present. Regarding the implementation details listed in Section 3.4.1, it is encouraged to further explore alternative settings, such as a more sophisticated approach of tubelet matching.

Currently, each model has a fixed FPS to ensure that the video stream is processed in real time. To improve this implementation, a more dynamic approach could be to fetch the latest frame as soon as the previous one is processed.

5.3 Work in a wider context

With the exploitation of Internet of Things, one can assume that edge machine learning applications will grow too. The real use of this solutions is not far away, and real-world aspects are discussed accordingly.

5.3.1 Ethical and societal aspects

Client-side solutions have the integrity aspect in its favor, since data is not sent to the server until an event occurs. Similar applications can be found in smart-home products, where voice activation component may be run on the client. Thereafter, the spoken language is to be processed on the server, when the user is aware about the recordings. That is, more than limiting data transfer, integrity is an advantage of client-side solutions which is relevant as soon as humans are involved in the video surveillance.

For the use case of surveillance of threatened species, information security becomes vital. Poachers are rarely those who fancy the finished ivory product, but rather workmen. Sadly, journalists have encountered cases where someone drifts between protecting the animals and hunting them, depending on how urgent their economical situation is. This raises the question of who should have access to the detection results.

5.3.2 Use case customization

The solution has been developed with generality in mind. For a specific application however, it can be further explored what functionality is provided in the camera of choice. As for example, some cameras has build-in motion detection which could be used to trigger the classification and tracking. Further, with the use of a PTZ camera, many possibilities open up. If the program has permission to control the camera, it could use tracking positions to zoom in and capture detailed images of an individual. As seen in section 2.2.1, classification of individuals is an on-going research topic. While the proposed system analyses only finished tubelets, it might be possible to zoom in on a tubelet which seems promising already while it is still active.

6

Conclusion

The aim to develop an object detection which can identify a set of African animal species was fulfilled. A tracking solution was further successfully set up. The research questions are discussed more in detail in this section.

6.1 Research questions

1. **How can a multi-class detector be designed to handle the open set problem, where just a few of the species are of interest? Should other species be included in the training data, and if so, to what extent?**

   As a conclusion, objects that share many visual features should be included in the training. The effort of additional data collection for a small set of classes would compensate for its benefit to distinguish the classes. Regarding to what extent classes should be added, one could decide to only include classes which are obviously similar. In the case of rhinoceros for example, it could be motivated to include a buffalo class since they are relatively similar, while a distinctly different animal such as an ostrich should not be needed. For this purpose, one should look further at the options to reject outliers, which currently seems to be an unsolved task for single shot detectors as SSD.

2. **Given the continuous output from object detection, what lightweight solution can be proposed to assemble events of interest?**

   A contribution of this work is a tracking solution using tubelet proposal, along with a simple event filtration. The solution produces meaningful events as long as the CNN model generates detections which are reasonably accurate. That is, the solution meets its goal. While related work has presented state-of-the-art accuracy with deep learning based tracking, it can be

concluded that for real-world usage, more light-weight tracking solutions can suffice. Especially, an advantage is its flexibility to track any object class.

3. **How will the trade-off between inference speed and accuracy for a CNN model affect the quality of real-time object tracking?**

   MobileNet V2 was selected, despite that it has neither the fastest inference time or the highest training accuracy compared to the other models. But in combination it results in the least number of false alarms and it is also conveniently small in model size. MobileNet V2 is a optimal trade-off, it is small, fast and accurate enough. As a conclusion, if model optimizations leads to a substantial decrease in inference time, it should be considered and tested against a more accurate model. In contrast, if the optimization benefit is only subtle and not crucial for the deployment platform, no particular advantage can be obtained in the task of event extraction in real-time detection.

6.2 Future work

This work has been developed towards a target application, and for this application in particular, some future adjustment are suggested. The performance of the detector might be limited due to the quality of the current training data, and thus a quick fix will be to gather more diversified data from the target location. It would also be interesting to further investigate how the system could be combined with other sensors, such as sound, LIDAR or IR sensors to only trigger to start the system when living creature is near. It would thus reduce the use of energy, and possibly also increase the system performance during the night hours when many animals, especially rhinoceros, are active. Additional sensor information could also be used together with the pan-tilt-zoom functionality to look more closely at the moving object.

An interesting spin-off of this work is to tackle the challenge of recognition of individuals. The question remains whether it is relevant to annotate images on a individual level, or rather rely on active learning. Another previously suggested solution is to use a database where representative images of individuals can be stored for comparison at inference time. Intuitively, recognition of individuals moves the scope of this work towards a less generalized domain, as the unique characteristic differs between animal species.

Bibliography

[1] X. Wu, D. Sahoo, and S. C. H. Hoi. Recent advances in deep learning for object detection, 2019.

[2] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, Apr 2020.

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems. 25. doi: 10.1145/3065386*, January 2012.

[5] R. Girshick. Fast r-cnn. *In Proceeding 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.*

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations, San Diego, 2015*, 2014.

[7] L. Torrey and J. Shavlik. Transfer learning. *In E. Soria, J. Martin, R. Magdalena, M. Martinez A. Serrano, editor, Handbook of Research on Machine Learning Applications. IGI Global. (chapter 11). doi: 10.4018/978-1-60566-766-9.ch011*, 2015.

[8] S. Schneider, G. W. Taylor, and S. C. Kremer. Deep learning object detection methods for ecological camera trap data. *pp. 321-328. doi: 10.1109/CRV.2018.00052*, 2018.

[9] G.S. Cheema and S. Anand. Automatic detection and recognition of individuals in patterned species. *In: Altun Y. et al. (eds) Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2017. Lecture Notes in Computer Science, vol 10536. Springer, Cham*, pages 27–38, 2017.

[10] M.S. Norouzzadeh, N. Joshi N. Jojic D. Morris, S. Beery, and J. Clune. A deep active learning system for species identification and counting in camera trap

images. *Proceedings of the National Academy of Sciences Jun 2018, 115 (25) E5716-E5725; doi: 10.1073/pnas.1719367115*, 2019.

[11] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2009.

[12] M. Wilber et al. Animal recognition in the mojave desert: Vision tools for field biologists. *In: 2013 IEEE Workshop on Applications of Computer Vision (WACV), pp. 206-213.*, pages 206–213, 2013.

[13] N.M. Badrinath, N. Bharat, S. Madhumathi, and B.S. Kariyappa. Pattern recognition using video surveillance for wildlife applications. *International Journal of Electronics and Communication Engineering Research and Development (IJECERD),Volume 4, Number 2, April-June (2014)*, 2014.

[14] C. Geng, S.-J. Huang, and S. Chen. Recent advances in open set recognition: A survey. page 1–1, 2020. In Proceeding of IEEE Transactions on Pattern Analysis and Machine Intelligence.

[15] M. Hassen and P.K. Chan. Learning a neural-network-based representation for open set recognition, 2018. In book: Proceedings of the 2020 SIAM International Conference on Data Mining, pp.154-162.

[16] P. Bodesheim, A. Freytag, E. Rodner, M. Kemmler, and J. Denzler. Kernel null space methods for novelty detection. In *Proceeding of 2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3374–3381, 2013.

[17] D. Miller, L. Nicholson, F. Dayoub, and N. Sünderhauf. Dropout sampling for robust object detection in open-set conditions, October 2017.

[18] W. Liu, D. Anguelov, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Sdd: Single shot multiple detector. *In Proceeding of the European Conference on Computer Vision (ECCV) (2016). doi: 10.1007/978-3-319-46448-0_2*, 2016.

[19] P. Hu and D. Ramanan. Finding tiny faces. *1522-1530. doi: 10.1109/CVPR.2017.166.*, 2017.

[20] C.R. Banbury et. al. Benchmarking tinyml systems: Challenges and direction. In *Proceedings of the 3rd MLSys Conference, Austin, TX, USA, 2020*, March 2020.

[21] M. Betke, E. Haritaoglu, and D. Larry. Real-time multiple vehicle detection and tracking from a moving vehicle. *Mach. Vis. Appl.*, 12:69–83, August 2000.

[22] M. Price, J. Glass, and A. P. Chandrakasan. 14.4 a scalable speech recognizer with deep-neural-network acoustic models and voice-activated power gating. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 244–245, February 2017.

[23] Deploy machine learning models on mobile and IoT devices. https://www.tensorflow.org/lite. Accessed: 2020-03-25.

[24] P. Warden and D. Situnayake. *TinyML - Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, Inc., first edition, 2019. ISBN: 9781492052043.

[25] L. Liu, H. Li, and M. Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking, Los Cabos, Mexico (MobiCom '19)*. Association for Computing Machinery, New York, NY, USA, 2019. ISBN: 9781450361699.

[26] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[27] R. Huang, J. Pedoeem, and C. Chen. Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. *In Proceeding of 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 2503-2510, doi: 10.1109/BigData.2018.8621865.*, 2018.

[28] TensorFlow. Model optimization. https://www.tensorflow.org/lite/performance/model_optimization. Updated: 2020-04-27. Accessed: 2020-04-15.

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 2014.

[30] L. Weng. Object Detection Part 4: Fast Detection Models. https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html. Published: 2018-12-27. Accessed: 2020-04-15.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *In Proceeding of 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018, pp. 4510-4520, doi: 10.1109/CVPR.2018.00474.*, 2018.

[32] A. Krizhevsky. Convolutional deep belief networks on cifar-10, 2010.

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.*, 2014.

[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *In Computer Vision and Pattern Recognition 2016*, 2015.

[35] X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollar, and C.L. Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.

[36] Object Detection on COCO test-dev. `https://paperswithcode.com/sota/object-detection-on-coco`. Accessed: 2020-03-26.

[37] COCO Detection Evaluation. `http://cocodataset.org/#detection-eval`. Accessed: 2020-03-23.

[38] M. Everingham, L. Van Gool, C.K.I Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *Int J Comput Vis 88, 303–338 (2010), doi: 10.1007/s11263-009-0275-4*, 2009.

[39] L. Shang, Q. Yang, J. Wang, S. Li, and W. Lei. Detection of rail surface defects based on cnn image recognition and classification. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 45–51, 2018.

[40] H. Wu, Y. Chen, N. Wang, and Z. Zhang. Sequence level semantics aggregation for video object detection. In *Proceeding of 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9216–9224, 2019.

[41] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. Simple online and realtime tracking. *In Proceeding of 2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016.

[42] P. Emami, P. M. Pardalos, L. Elefteriadou, and S. Ranka. Machine learning methods for solving assignment problems in multi-target tracking, 2018. arXiv preprint arXiv:1802.06897.

[43] Y. Pei, S. Biswas, D. Fussell, and K. Pingali. An elementary introduction to kalman filtering. *Commun. ACM 62, 11 (November 2019), 122–133. doi: 10.1145/3363294*, 2019.

[44] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *In Proceeding 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.*, 2009.

[45] K. Kang et al. Object detection in videos with tubelet proposal networks. *In Proceeding of 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 889-897, doi: 10.1109/CVPR.2017.101*, Jul 2017.

[46] Google Colaboratory. `https://colab.research.google.com/notebooks/intro.ipynb`. Accessed: 2020-03-24.

[47] LabelImg. `https://github.com/tzutalin/labelImg`. Accessed: 2020-03-25.

[48] Y. Xian, C.H. Lampert, B. Schiele, and Z. Akata. Zero-shot learning - a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, no. 9, pp. 2251-2265, 1 Sept. 2019, doi: 10.1109/TPAMI.2018.2857768*, 2018.

[49] Animals with Attributes 2. https://cvml.ist.ac.at/AwA2/. Accessed: 2020-04-02.

[50] iNaturalist. https://www.inaturalist.org/. Accessed: 2020-04-02.

[51] Unsplash. https://unsplash.com/. Accessed: 2020-04-02.

[52] LILA BC. Wildlife Image and localization Dataset. http://lila.science/otherdatasets. Accessed: 2020-04-02.

[53] LILA BC. Snapshot Mountain zebra. http://lila.science/datasets/snapshot-mountain-zebra. Accessed: 2020-04-02.

[54] LILA BC. Snapshot Enonkishu. http://lila.science/datasets/snapshot-enonkishu. Accessed: 2020-04-02.

[55] LILA BC. Snapshot Kruger. http://lila.science/datasets/snapshot-kruger. Accessed: 2020-04-02.

[56] LILA BC. Snapshot Karoo. http://lila.science/datasets/snapshot-karoo. Accessed: 2020-04-02.

[57] A.B. Swanson, M. Kosmala, C.J. Lintott, R.J Simpson, A. Smith, and C. Packer. Data from: Snapshot serengeti, high-frequency annotated camera trap images of 40 mammalian species in an african savanna. *Scientific Data. 2. 150026. doi: 10.5061/dryad.5pt92*, 2015.

[58] LILA BC. Snapshot Serengeti. http://lila.science/datasets/snapshot-serengeti. Accessed: 2020-04-02.

[59] TFRecord. https://www.tensorflow.org/tutorials/load_data/tfrecord. Accessed: 2020-03-24.

[60] Tensorflow detection model zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. Accessed: 2020-03-24.