Master of Science Thesis in Media Technology
Department of Electrical Engineering, Linköping University, 2022

A smart surveillance system using edge-devices for wildlife preservation in animal sanctuaries

Johan Linder and Oscar Olsson

LINKÖPING UNIVERSITY

Master of Science Thesis in Media Technology

A smart surveillance system using edge-devices for wildlife preservation in animal sanctuaries

Johan Linder and Oscar Olsson

LiTH-ISY-EX--22/5514--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

The Internet of Things is a constantly developing field. With advancements of algorithms for object detection and classification for images and videos, the possibilities of what can be made with small and cost efficient edge-devices are increasing. This work presents how camera traps and deep learning can be utilized for surveillance in remote environments, such as animal sanctuaries in the African Savannah. The camera traps connect to a smart surveillance network where images and sensor-data are analysed. The analysis can then be used to produce valuable information, such as the location of endangered animals or unauthorized humans, to park rangers working to protect the wildlife in these animal sanctuaries. Different motion detection algorithms are tested and evaluated based on related research within the subject. The work made in this thesis builds upon two previous theses made within Project Ngulia. The implemented surveillance system in this project consists of camera sensors, a database, a REST API, a classification service, a FTP-server and a web-dashboard for displaying sensor data and resulting images.

A contribution of this work is an end-to-end smart surveillance system that can use different camera sources to produce valuable information to stakeholders. The camera software developed in this work is targeting the ESP32 based M5Stack Timer Camera and runs a motion detection algorithm based on Self-Organizing Maps. This improves the selection of data that is fed to the image classifier on the server. This thesis also contributes with an algorithm for doing iterative image classifications that handles the issues of objects taking up small parts of an image, making them harder to classify correctly.

Acknowledgments

We would like to thank everyone involved in Project Ngulia who made this thesis possible. A special thanks to our examiner and project leader Fredrik Gustafsson and our supervisor Magnus Malmström at Linköping University. Your input and advice regarding all different parts of the thesis has been very helpful. Thanks to Martin Stenmarck for helping us with system and server related matters. Thanks to Carlos Vidal at Linköping University for your valuable input regarding hardware, networks and electrical engineering. Thanks to Kolmården Zoo for letting us spend time at their Savannah for testing our devices. Thanks to Kenya Wildlife Service and the rangers of Ngulia Rhino Sanctuary for the help during deployment in the Ngulia Park. And a big thanks to Sara Olsson, Amanda Tydén, Pontus Arnesson and Johan Forslund for the valuable work produced during previous theses that made it possible for us to take the system all the way to Kenya for deployment.

Norrköping, June 2022
Johan Linder and Oscar Olsson

Contents

Notation

Abbreviations

Abbreviation	Meaning
COCO	Common Objects in Context
CNN	Convolutional neural network
MAP	Mean average precision
R-CNN	Regions with convolutional neural network features
RT-SBS-V2	Real-Time Semantic Background Subtraction v2
SOM	Self-Organizing Map
API	Application Programming Interface
UI	User Interface
VOC	Visual Object Classes

1

Introduction

Thanks to the advancement of algorithms for object detection and classification for images and videos, a camera surveillance system automatically provides valuable information to the user of which images are of interest. This leads to that there are now smart and sophisticated surveillance-systems all over the world with different areas of use.

These systems can often be found in urban environments, especially in areas where a lot of people are passing through. There is however, sparsely populated and remote areas that could benefit from the use of information provided by a smart surveillance system. One example being animal sanctuaries that provide shelter for exposed and endangered species. These areas are often patrolled by park rangers to ensure the safety of the animals and protect the area from unauthorized people. The rangers work could become more effective by the use of a smart surveillance system that sends information to the rangers in real time about activities in the sanctuary.

Edge-devices are microcomputers that generally are used to collect, process and forward information. These devices are being used to a greater extent in for example units like smart cars, speakers and home automation. Also, the inclusion of cameras in these devices is becoming more popular. Edge-devices with cameras as sensors can be used to create a smart surveillance system in an environment where a traditional solution is not available. The edge-devices can provide its own power supply and connection, making it possible to place them in an animal sanctuary for surveillance in these remote areas.

1.1 Background

The black rhino is a rhino species with a population that has reached a critical level and is today on the verge of extinction, although the last few years have shown a slow regrowth. Ngulia is a sanctuary in Kenya with a population of around 130 black rhinos. Project Ngulia is a project for developing technical support for park rangers in the area, to monitor the rhinos and to prevent poaching [1]. The project is a collaboration between different companies and organizations and some of the major ones are Linköping University, HiQ, Kolmården Zoo and the Ngulia park. The role of Linköping University is mainly to do research on how different technical solutions can be developed and what sensors and hardware that are best suited for the task. In collaboration with Kolmården Zoo in Norrköping, sensors have been placed around the zoo to be able to simulate a test environment that is somewhat similar to the Ngulia park.

The capacity of today's edge-devices is rapidly increasing, which makes it possible to run better models for object-detection and even classifications on these devices. The Ngulia Project has been up and running for some years, hence the amount of training data have also increased, which makes it easier to train models for detection and classification of objects of interest for the park rangers.

During two earlier periods of thesis work, 2020 and 2021, students have been working with developing technical solutions for the project. During 2020, a thesis was made where focus were on integrating machine learning on edge devices like raspberry pi, to detect animals and especially rhinos in real time [15]. Different learning models were also evaluated during this thesis work to find out which was best suited for the task. During 2021, further work was made in the project where the focus was switched to detect humans inside the sanctuary [8].

1.2 Aim

The aim is to investigate methods to build a surveillance system for a remote environment. The surveillance system will consist of a number of Edge-devices that demand minimal maintenance, the devices will be scattered over a large area hence physical availability to the devices will be limited. The main objective will be to investigate the accuracy and efficiency of different motion detection algorithms on micro controllers. By increasing the accuracy and efficiency of the detection algorithm, the amount of data sent from the micro controller is minimized, which also lowers the power consumption of the device. It will also be investigated how the reliability of the Edge-devices can be increased when these are deployed in the field. Previously produced models for object classification will be combined and evaluated on its ability to classify both humans and animals. When the different parts of the system are implemented and evaluated, they can be connected into a surveillance network. The surveillance network should not be limited to only a specific type of camera-sensor, but any type of

camera that can upload images to a server should be able to plug in to the surveillance network. The design of the network should also be modular so that it can be easily applied to other areas of interest with smaller configurations, for example surveillance of wild animals in Sweden in the same way as for the animals in Ngulia.

1.3 Research questions

Preliminary focus areas:

1. How can Self-Organizing Maps be used for motion detection on the ESP32 and how does it perform compared to Frame differencing?
2. How can the performance of an image classifier be increased for low scoring objects which cover a small area of the image?
3. How can a robust and smart surveillance system be designed, implemented and deployed with a remote target area? How can the challenges that come with developing from afar be treated?

1.4 Limitations

The target environment of the project is located in Kenya and therefore not accessible during the major part of the project. Therefore, the deploy environment needs to be simulated in Kolmården Zoo to be able to test and evaluate the system. Also, the internet connection in the target area is of varying stability. However, for the purpose of this project, it will be assumed that internet connection is available most of times. The amount of available annotated training data for an image classifier is also a limitation for this project.

2
Theory

2.1 Self-Organizing Maps

Introduced by Kohonen in 1982, the Self-Organizing Map, SOM, is an unsupervised artificial neural network [12]. It can be seen as a projection of the input data onto a grid of prototype neurons, which is commonly two-dimensional. A sample from the input gets associated with the neuron which it is the most similar to using a suitable metric, for example euclidean distance:

$$c(t) = \underset{i \in |\mathcal{P}|}{\operatorname{argmin}} \| s(t) - n_i(t) \|, \tag{2.1}$$

at time step t, $\mathcal{P}$ is the set of prototype neurons, $|\mathcal{P}|$ is the number of neurons in $\mathcal{P}$, c is the index of the most similar neuron n and s is the new sample from the input.

The neurons in the grid are then adjusted towards the input sample, hence the name "Self-Organizing". Each neuron is adjusted according to a learning rate and smoothing kernel. A neuron in the network is updated according to:

$$n_i(t+1) = n_i(t) + \alpha(t) \cdot \eta_{i,c}(t) \cdot [s(t) - n_i(t)], \tag{2.2}$$

where n_i is the neuron to update, α is the learning rate and η is the smoothing kernel. The smoothing kernel depends on the distance from the current neuron i to the best matching neuron c from (2.1). Neurons which are closer to the best matching neuron is adjusted to a higher degree than neurons which are further away. See Fig. 2.1 for a visual representation of the training process.

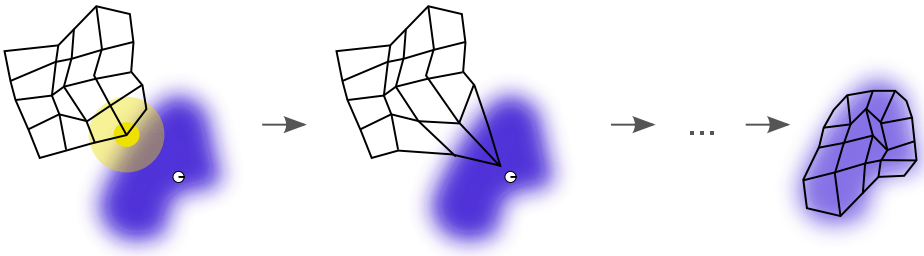


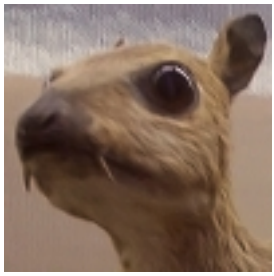
Figure 2.1: *SOM adjusting to a cluster of two dimensional inputs. The white dot is a first input sample, the neuron marked in yellow is the best matching neuron and the neurons are then adjusted towards the input sample. After processing all the inputs the resulting SOM can be seen on the right, with all the neurons fitted over the input data. Image from Dan Stowell under CC BY-SA 3.0 [20].*

After the SOM has adjusted the neurons from input samples, it can be used to classify anomalies in new samples. If the similarity between a new sample and any neuron in the grid is not under a certain threshold, the input sample can be treated as an anomaly. The SOM can be implemented to use online learning, meaning it can adapt to anomalies in the input if the same anomaly is consistently over multiple samples.

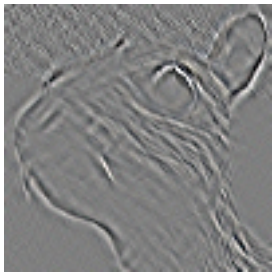
The use of online learning together with the ability to detect anomalies makes the network a suitable candidate for motion detection in video. The simplicity of the network also makes it suitable to implement on hardware which has limited computational power. SOM's have been used in motion detection algorithms on hardware with low computational power with promising results, see e.g. Ortega-Zamorano et al [16].

2.2 Neural network classifiers & transfer learning

Deep neural networks utilizing convolutional operations for feature extraction are the current state of the art when it comes to classification problems for images. Pre-trained networks, which have already been trained on large datasets, are commonly utilized when trying to solve new classification tasks. The pre-trained networks can be used with the help of transfer learning. Transfer learning means to use knowledge which the network learned previously to solve the new task. This is a good alternative compared to training a network from scratch because it significantly reduces the amount of time and power needed to achieve similar performance.



(a) *Original image.*



(b) *Output from convolution.*

Figure 2.2: *Edge detection on an image through a convolutional operation. Images by Michael Plotke, CC BY-SA 3.0 (a) and Crisluengo, CC BY-SA 4.0 (b) [17] [5].*

2.2.1 Convolutional neural networks

Convolutional operations are a common tool in image processing. Different features or properties can be extracted from an image depending on the kernel used in the convolution. See Fig. 2.2 where edges has been extracted from an image.

These convolutions can be used to produce features which describes the content of an image and features which describes the objects in an image.

One of the earliest implementations of a convolutional neural network, CNN, for object detection resulted in a great improvement in performance [9]. The implementation by Ross Girshick et al. improved the mean average precision, MAP, by more than 30% relative to the previous best result on the VOC 2012 challenge at the time [7]. CNN's have since been used as the backbone of well performing network architectures, for example the Centernet architecture, see e.g Duan et al [6].

Convolution layers

Given an input tensor, in this case an image or a batch of images, the convolutional layer performs a convolution on each image. The convolutional computation uses a kernel, also called filter or filter kernel. In this case the kernel is a two-dimensional matrix containing learnable weights. The kernel is placed over the image and the dot product is calculated between the kernel and the pixels which it currently covers, see Fig. 2.3. The kernel is then moved with a specific stride-length, but often just a single step, and the process is repeated until the kernel has moved over the whole image. The borders of the image can be padded to produce an output which has the same size as the input image. For example, the image in Fig. 2.3 is not padded and the 3×3 kernel can only be placed in nine different places, resulting in a 3×3 output from a 5×5 input.

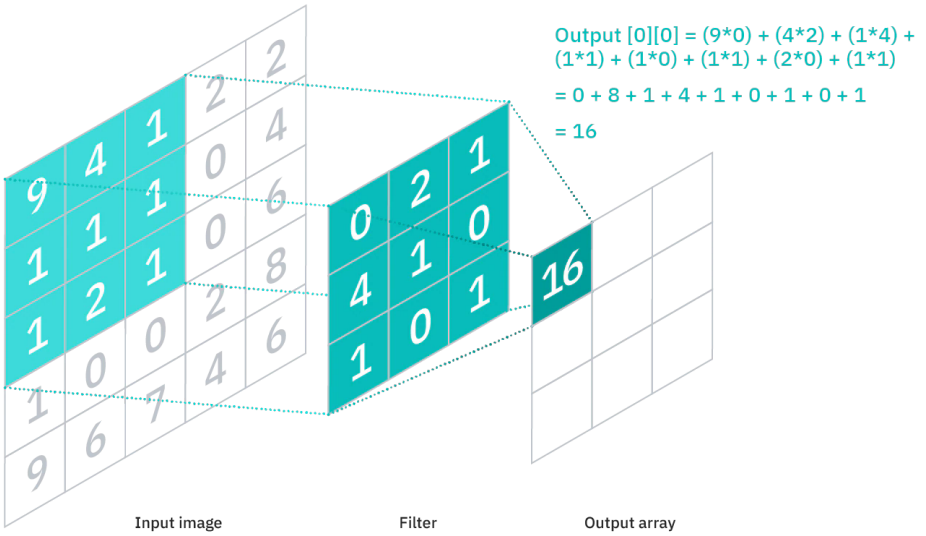


Figure 2.3: *Convolution in 2D. Image from IBM Cloud [4].*

The result of the convolution is a feature map where some features of the image is produced. The type and the values of the filter kernel used in the convolution decides what features will be highlighted from the image. A single convolutional layer consists of several different filter kernels with different weights which detects different features.

Pooling layers

A pooling layer reduces the dimension of the input to the layer by combining several data points to one. A simple two-by-two kernel combines four adjacent values into a single value. Common methods to combine the values are to use the average of the values inside the kernel (Average pooling) or to use the maximum value inside the kernel (Max pooling).

2.2.2 CenterNet

CenterNet is a deep neural network architecture for object detection which represents each object in an image as a keypoint triplet [6]. The triplet consists of a center point, a top-left and a bottom-right corner points which is then used to generate bounding boxes. The overall architecture can be seen in Fig. 2.4.

A stacked hourglass network is used as the convolution backbone to generate feature maps from the image [14]. Cascade corner pooling is then applied to the feature map to produce two corner point heatmaps with corresponding offsets and embeddings for each corner. The offsets are used to map the points from the heatmap back to the original image and embeddings help match corners which

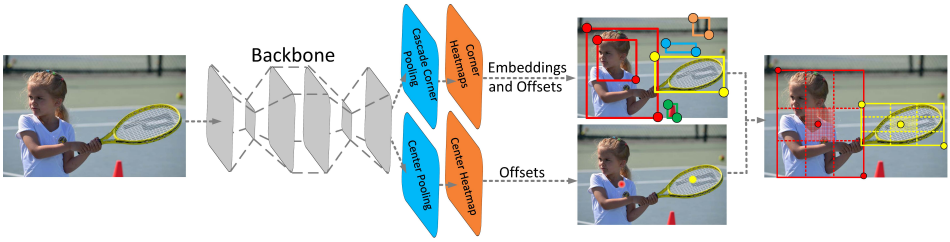


Figure 2.4: *CenterNet architecture overview. Image by Duan et al [6]. © 2019 IEEE.*

are from the same object. Center pooling is also applied to the feature map to produce a center point heatmap with corresponding offsets. Corner points together with their embedding are used to initially predict bounding boxes and then if a center point is present in the central region of a bounding box it is added to the final result. Bounding boxes which have no associated center point in their central region is not considered a valid prediction.

To train a model using the CenterNet architecture a dataset with images and labelled bounding boxes is used.

2.2.3 Transfer learning

Transfer learning in a general sense can be defined as "Transfer of learning means the use of previously acquired knowledge and skills in new learning or problem-solving situation" [19]. In the context of deep neural networks, transfer learning means the usage of a neural network which has been trained for a task and reuse the network for a new task. The amount of knowledge which can be transferred depends on the similarity between the tasks. In the context of image classification, the task often differs regarding the types of classes which the network tries to classify.

Classifying the type of an object is in general the last step in an image classification task. Region extraction and feature extraction are essential steps in a classification task and these steps can be generalized for arbitrary classes and objects. This means that a majority of knowledge related to these two steps, region extraction and feature extraction, can be transferred between image classification tasks where the classes differ.

In practice, this means that neuron weights in some of the layers of the pre-trained network are reused for the new task. The layers responsible for classifying the type of the object is the very last convolutional and fully connected layers in the network. These can be unfreezed and then retrained using appropriate input data for the new task [21].

2.3 ESP32 based microcontrollers

The use of microcontrollers in home automation and embedded systems are increasing all the time. Some of the main advantages with the ESP32 are the portability and the cost effectiveness as summarized in the article "Using ESP32 Microcontroller for Data processing" by Babiuch, Foltýnek and Smutný [18]. The ESP32 chip is developed by Espressif Systems and comes in different versions. All of the versions are microcontrollers with integrated WiFi and 4MB of flash memory. Some versions also come with external SPI PSRAM (Pseudo static RAM), which increases the amount of data that can be processed at the same time. The ESP32 also includes two CPU cores that can be used separately for parallel computing.

There are some different developing platforms that can be used when implementing code on a ESP32 microcontroller, for example the Arduino platform or the Espressif IoT Development Framework. For developing a more optimized embedded system, the most native way is to use the Espressif framework. Some of the ESP32 microcontrollers have the option to connect a camera module, turning them into powerful camera sensors that are power efficient, have the ability to process data and then provide information to a system with more computational power.

2.3.1 M5Stack Timer Camera

The M5Stack Timer Camera is an ESP32 based camera module with 8MB of PSRAM and a 3MP camera [2]. With the ability of going into "sleep mode" when not active, the current consumption is 2μA, making the module power efficient and it can be powered by a connected battery in environments where no electricity is available. The module also has the ability to transfer images with WiFi and has a USB-port for debugging and to flash software to the unit. In Fig. 2.5, a size comparison can be seen with the Timer Camera next to a regular pen.

Figure 2.5: *M5Stack Timer Camera size comparison next to a pen.*

3
Method

This chapter describes how the different parts of the surveillance system were implemented and how the different features were used.

3.1 System overview

To make it easier to evaluate the system, the initial work consisted of putting together a modular end-to-end system. This such that each part could be evaluated as well as the whole system from camera to UI. In Fig. 3.1, an overview of the surveillance system is illustrated where each part of the system is displayed and the paths of communication is drawn out. The green parts are the core system and can be used for different types of camera sensor, FTP and User Interface. The yellow parts are interchangeable parts so that the system can be used as a template and then modified for different purposes.

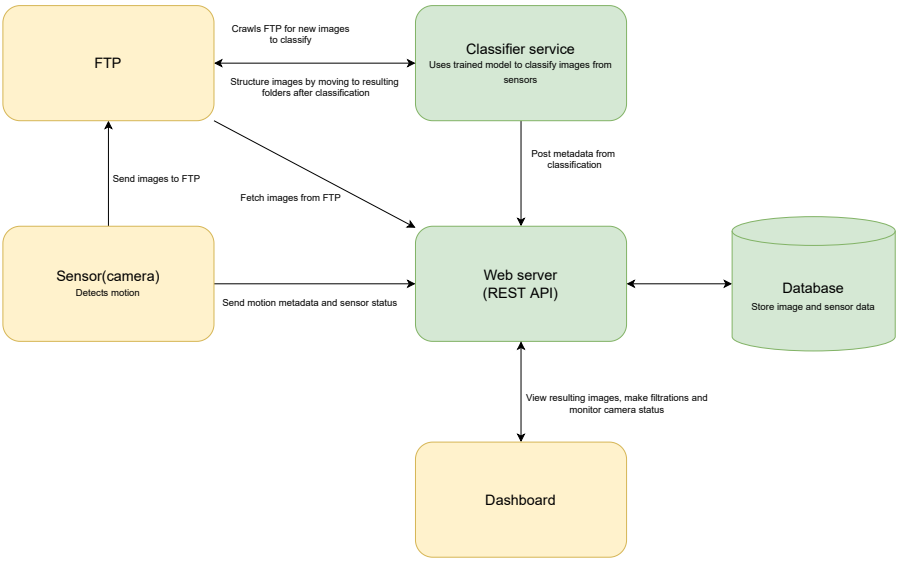


Figure 3.1: *Block diagram of the system.*

3.2 Camera sensor

The target hardware for the development of the camera software was the ESP32 based "M5Stack Timer Camera" which was described in Sec. 2.3.1. The work produced by Arnesson and Forslund in 2021 [8], was used as a baseline for the camera software. They implemented a motion detection algorithm with the use of background subtraction and comparing pixel changes between images. Arnesson and Forslund also implemented functionality for sending images to a FTP-server when motion was detected. This was used as a baseline to start with and from which other useful functionalities could be added to. In Fig. 3.2 the different components of the camera sensor system can be seen with the ESP32 Timer camera in the top of the image, the solar power management board to the right, the battery and battery gauge to the left.

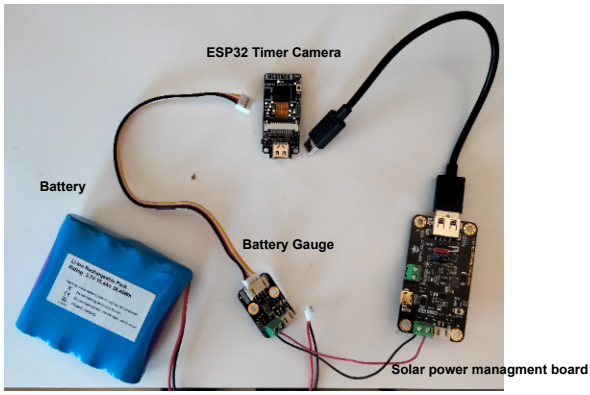


Figure 3.2: *The different components of the camera sensor system.*

3.2.1 Timestamp

To keep track of the images in the database and also make it possible to filter images based on different time periods, functionality was added to take a timestamp when motion was detected on a photo. This timestamp was then used as the filename of the image and later stored in the database.

3.2.2 Motion detection using SOM

The SOM described in Sec. 2.1 can be used as a motion detection algorithm by using an image or a representation of the image as input to the network. Meaning a pixel or an area of pixels can have a set of neurons which represent the most common samples for that pixel or area of pixels. However, images are sensitive to noise and having a set of neurons for each pixel is computationally heavy. The image can be divided into blocks, where the value of each block B is the average of all the pixels within that block. This helps reduce the computation needed and reduce the amount of noise in the input to the SOM. For example, this approach was used by Ortega-Zamorano et al [16].

A sample to the SOM looks like:

$$\boldsymbol{s}_B = \frac{1}{P_B} \sum_{\boldsymbol{p} \in B} \boldsymbol{p} \tag{3.1}$$

Where $\boldsymbol{s}_B$ is the new sample for block B, P_B is the total number of pixels in each block and $\boldsymbol{p}$ are pixels belonging to block B. This simply produces an average of the pixels in a block in the image to present to the network.

To decide if a sample is an anomaly and therefor not part of the usual background in the frame, the quantization error can be used. The smallest quantization error

among all the neurons is the interesting one in this case, since it is the neuron which is the best match to the sample i.e.,

$$q_B = \min_{i \in |\mathcal{P}_{block}|} \| \boldsymbol{s}_B(t) - \boldsymbol{n}_{Bi}(t) \|, \tag{3.2}$$

where, for block B, $|\mathcal{P}_{block}|$ is the number of neurons in each block, q_B is the smallest quantization error found between the sample $\boldsymbol{s}_B$ and the neurons $\boldsymbol{n}_{Bi}$ of the block. If the smallest quantization error is larger than some threshold the sample can be considered an anomaly:

$$anomaly_B = \begin{cases} \text{true}, & \text{if } q_B > T \\ \text{false}, & \text{otherwise}, \end{cases} \tag{3.3}$$

where T is the given threshold. A block which is considered an anomaly implies that something has moved in the block and caused a change in the pixel values of the block relative to their normal values.

3.2.3 Evaluation

The motion detection algorithm described above has been tested on all the "baseline" and "intermittent object motion" sequences from the CHANGEDETECTION.NET dataset [10]. The algorithm was compared to the original motion detection implementation on the following measurements: *Recall*, *Precision*, *Specificity*, False positive rate (FPR), False negative rate (FNR) and *F-Score*. The measurements are defined as:

$$Precision = \frac{TP}{TP + FP} \tag{3.4a}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.4b}$$

$$Specificity = \frac{TN}{TN + FP} \tag{3.4c}$$

$$\text{FPR} = \frac{FP}{TN + FN} \tag{3.4d}$$

$$\text{FNR} = \frac{FN}{TN + FN} \tag{3.4e}$$

$$F\text{-}Score = 2 \cdot \frac{Precision * Recall}{Precision + Recall} \tag{3.4f}$$

Here TP denotes a "True Positive", a correctly detected foreground pixel. TN denotes a "True Negative", a correct rejection. FP denotes "False Positive", a false

alarm and *FN* denotes a "False Negative", a false rejection. A summary of the measurements and their meaning can be seen in Table 3.1.

Table 3.1: *Summary of measurements and their meaning.*

Measurement	Question the measurement tries to answer
Precision	"What proportion of detected foreground pixels were actually correct?"
Recall	"What proportion of true foreground pixels were detected correctly?"
Specificity	"What proportion of true background pixels were rejected correctly?"
FPR	"What proportion of true background pixels were detected incorrectly?"
FNR	"What proportion of true foreground pixels were rejected incorrectly?"
F-Score	Harmonic mean between precision and recall

All measurements ranges between 0–1 and higher score is better in every case except for FPR and FNR, where lower is better.

At the time of this report the highest ranked unsupervised method is "Real-Time Semantic Background Subtraction v2" (RT-SBS-V2) [3]. The average result of RT-SBS-V2 over the entire dataset can be seen in Table 3.2.

Table 3.2: *Average evaluation scores for RT-SBS-V2 on all sequences of the CHANGEDETECTION.NET dataset.*

Algorithm	***Precision***	***Recall***	***Specificity***	**FPR**	**FNR**	***F-Score***
RT-SBS-V2	0.793	0.836	0.994	0.006	0.1639	0.805

The ground truth for the dataset is on a pixel level, while the output from both SOM and frame differencing algorithms are on a block level, meaning all the pixels within a block are marked as foreground pixels if the block is detected as foreground. The algorithms will therefor also be evaluated on adjusted ground truth images. The adjusted ground truth is on a block level, and if any pixel inside a block is marked as foreground in the original ground truth, the whole block is marked as foreground in the adjusted version.

Practical usage

If anomalies are detected in any block the image is sent from the camera for evaluation and classification, together with metadata about the the movement. The SOM can detect where the movement occurred in the frame on a block level. This

information can be used to crop around the detected movement in the image before running object classification which has shown possible increase in accuracy for the classification model by Forslund & Arnesson [8].

3.2.4 Device status

Since the deployment site of the camera modules is in a remote environment where manual support is limited, it is valuable to get information about the status of each device. Therefore, a system for sending status updates to the server was implemented on the camera. From the camera unit, information can be sent about which WiFi access point that is connected as well as the strength of the connection. The camera unit is powered by a battery that is connected to a solar cell. It is important that the camera system does not run out of power and that the solar panel can deliver enough power to charge the battery during the day. To make it possible to monitor the battery status of the devices, a battery gauge was integrated in the power cycle of each camera unit. With this battery gauge, it is possible to read the battery percentage and send that information to the server. With this feature, users are able to get an overview of each camera's status. This gives an understanding of how the power is consumed and recharged during the day when the camera is active, but also during the night when it is asleep. The battery gauge also creates an opportunity for the camera to manage its own sleeping cycle based on the battery status. For example, if the battery level goes below a certain threshold (20%), the camera unit goes to sleep for four hours to make time for the battery to recharge.

3.2.5 Energy consumption and online time

When deployed in the field, the camera system needs to be able to run continuously for a longer period of time so that it is self sufficient with the power from the battery and charging from the solar panel. To monitor the battery consumption for a camera, the status logs described in Sec. 3.2.4 could be used to plot the battery status for different time periods and cameras. Since the camera sensors are dependent on internet connectivity to be able to send images and status updates, the data could also be used to give insights on the up time of the sensors, since they were set to send updates at set intervals.

3.2.6 Weatherproof housing

Since a camera sensor with its belonging battery is deployed outside with varying weather conditions, the different parts need to be protected from rain and other factors that could risk breaking the unit. For this purpose, a 3D-printed case was created to protect the camera sensor together with the battery and solar power management board. In Fig. 3.3 the weatherproof housing containing the sensor system with connected solar power can be seen mounted in a tree.

Figure 3.3: *A tree-mounted esp-32 camera with weatherproof housing and connected solar panel.*

3.3 Web server & database

Since images from different cameras are stored on a FTP server, an efficient way of storing related metadata and classification results were needed. For these tasks, a MySQL database and a Node.js web server were implemented.

3.3.1 Database

A MySQL database was implemented to store all the necessary information about a classified image. In Fig. 3.4, a class diagram of the most significant parts of the database can be seen. With significant data stored about each image, queries can be made for images with a specific class property, or from a specific period of time.

3.3.2 Web server

To make the data stored in the database accessible from different parts of the system, a REST API in form of a web server was implemented using Node.js with the framework Express. The API was designed so that the functionality of the database is accessible both by a user interface and the classification service. From a UI, image metadata can be fetched through the API and with the filename from the database that holds the path to where the image is stored on the FTP-server, an image can be displayed to the user.

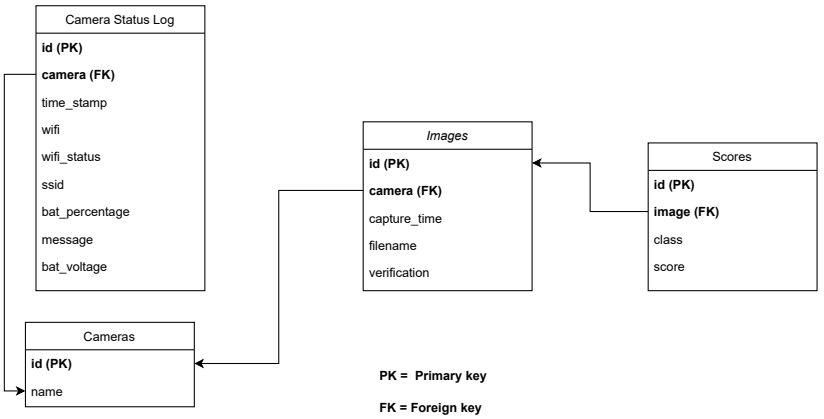


Figure 3.4: *Class diagram of the database holding information about the cameras and related images with scores.*

3.4 Classification service

The classification service consists of two major parts: the training phase and the classification phase which are described further in this section. The implementation of this service was made to be modular in a way so that the script could be applied to other areas of interest with small configurations. For example a different model could be trained with other types of animals and then used with the same logic implemented for classification of the Ngulia animals.

3.4.1 Training of model

A pre-trained CenterNet model supplied by Tensorflow was used as the classifier [6]. The model was pre-trained on the Common Object in Context, COCO, 2017 dataset [13]. The model was fine tuned for the task of classifying eight classes which are present in the Ngulia Sanctuary: buffalo, elephant, giraffe, human, leopard, lion, rhinoceros and zebra.

Fine tuning the pre-trained model for the new task was straight forward with the use of the Tensorflow Object Detection API [11]. The object detection API requires some input: a pre-trained model, a configuration file describing the architecture of the model together with training parameters and finally the dataset it should train on. The training is then started and when the desired number of training steps is reached it exits. The model was trained on *Google Colaboratory* which supplies GPU-supported enviroments to run the training steps on for free. The dataset which the model was fine tuned on consisted of a total of 3843 images distributed over the eight classes according to Table 3.3. The images were collected from multiple sources and labelled by Olsson & Tydén and Arnesson & Forslund [15] [8]. No further images were added to the dataset during this work.

Table 3.3: *Distribution of images for each class in the dataset.*

Class	Quantity
Buffalo	247
Elephant	327
Giraffe	426
Human	432
Leopard	586
Lion	499
Rhinoceros	881
Zebra	445
Total	**3843**

3.4.2 Classification

The classification service was built as a Python script, where the ability to load different trained models was implemented. The service is set to crawl the FTP-server to check if there are any new images to classify. As before, the previous work of Arnesson and Forslund [8], was used as a baseline to start from. When a new image is found, the service downloads the image and the model runs inference on the image. The result is a prediction of bounding boxes, classes and scores of objects in the image. Bounding boxes, with corresponding classes and scores, can then be drawn on the image. After the classification, the results are saved in the database through the API described in Sec. 3.3. Only classifications above a certain threshold will be drawn out on the image and saved to the classified image in the database. However, for better insights in the performance of the classifier, also the scores below the threshold were saved in a special debug folder on the FTP-server and in the database.

Image cropping

Since the model performs better on images where the object is closer to the camera and takes up a larger area of the image, an algorithm was implemented to crop out those certain areas where the model "sees" an object. This was done through an iterative classification where the classifier first makes a classification on the original image. If the result contains one or more bounding boxes, these coordinates are used to crop the image into one or more sub-images. A new iteration of inference then runs on these images and if the score is higher than the initial one, the new bounding box is drawn out on the cropped image and then saved in the FTP-server and database with a relation to its original image. In Fig. 3.5, a block schema illustrates how the image classification pipeline works.

This algorithm was evaluated on a set of 500 images to measure how many objects saw an increase in score after cropping around the original prediction and running inference again.

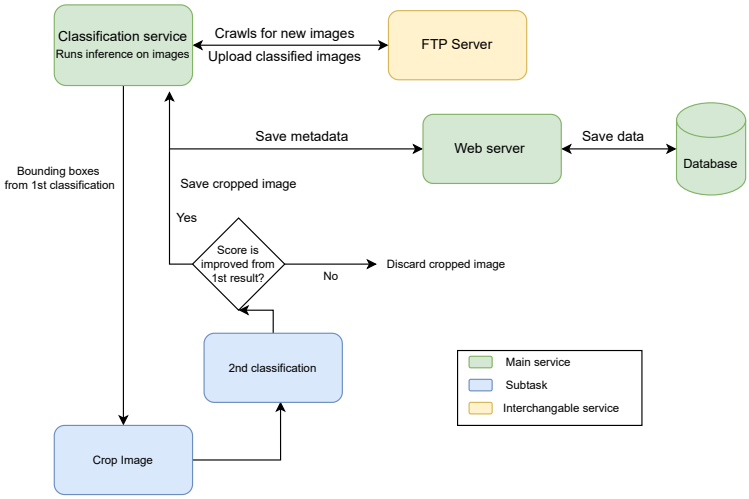


Figure 3.5: *Image classification pipeline.*

3.5 Developer dashboard (UI)

To be able to interpret the results of the classification pipeline from a camera to a processed output image, a developer dashboard was created where these resulting images could be displayed. Filtering functionality was also implemented to make it easier to query specific time periods and animal classes from the different cameras used in the project. In Fig. 3.6, an overview of the dashboard can be seen with its different functionalities, which are described further in this section.

3.5.1 Verification

A tool for verifying the classification results in an image was implemented in the UI where a user can determine if the classification made by the model is as expected. This tool can be seen within the dashboard in Fig. 3.6. With this function, it is possible to also filter displayed images to only show those that are verified as a correct or partly correct classification. By adding this functionality, it also created the possibility to use the images that are classified correctly and re-train the model to improve the accuracy of the classifications.

3.5.2 Debug

To get insights of how the classifier interpreted different types of images, a debug mode was added where the user can see all the different classifications made on an image, even those with a low score that is below the set threshold. In Fig. 3.7, an example of the debug mode of an image is shown. The image to the left shows the resulting image with classification scores above a set threshold (40%). In the right image, all bounding boxes and results are drawn out in the image.

Figure 3.6: *An overview of the user interface with its different functionalities.*

Figure 3.7: *Debug mode with resulting image to the left and all classifications scores including those below the set threshold to the right.*

3.5.3 Camera status monitor

To monitor the different camera sensors in the surveillance system, status updates from each camera were fetched through the API and displayed in the UI. By keeping and displaying logs from each camera, valuable information can be retrieved and displayed from a camera sensor. In this way, it is possible to see which cameras are online and if not, when the last status report was made and what the current status of the sensor was at that moment. With the implementation of a battery gauge, as described in Sec. 3.2.4, the battery percentage could be monitored from the UI. In the right column of Fig. 3.6, the overview of the cameras in the project are displayed with a green dot for online cameras and red for offline as well as the battery percentage for each camera.

3.6 Deployment of camera sensors

Since the targeted deployment site for the camera sensors are located in a remote area (Ngulia, Kenya), the deployment had to be made in two different phases, the testing phase in Sweden and the final deployment in Kenya.

To be able to test everything in a more stable and accessible environment, a twin station of the one planned in the Ngulia park was set up at Kolmården Zoo in Norrköping. There everything could be tested and evaluated before the final deployment in Kenya. Cameras were put up at the Kolmården "Savannah" and left there for some weeks, collecting data and images that could be evaluated. After that, updates could be made to the sensors to ensure their reliability and continuous power cycle.

After several weeks of testing and fine tuning of parameters, the sensors were deployed in the Ngulia Park in Kenya. Also there, some changes had to be made to the software to adapt to the different deploy environment which had some issues with internet connectivity. Because of these unforeseen issues, a third deployment was made at the Kilaguni Resort, located in another area next to the Ngulia Park. There the internet connectivity was more stable and a more continuous flow of data and images were received.

4

Results

The results of the work on this thesis are presented in the following chapter. The work described in Method resulted in an end-to-end smart surveillance system with a modality that makes it suitable to use in other fields of animal surveillance and not only in the context of Project Ngulia. The results from the different parts of the system are both quantitative and qualitative and will be presented accordingly. In Fig. 4.1, the pipeline of the resulting system is visualized, from camera to dashboard.

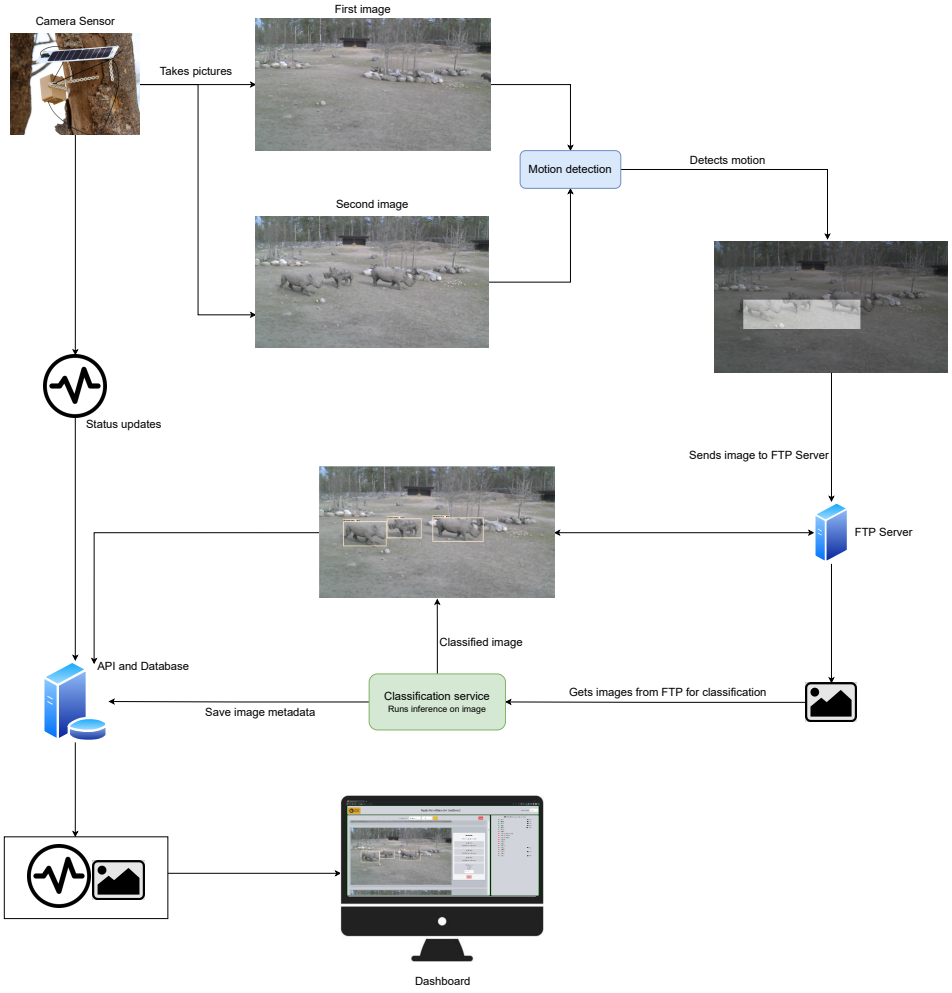


Figure 4.1: *The end-to-end pipeline of the surveillance system developed in the project.*

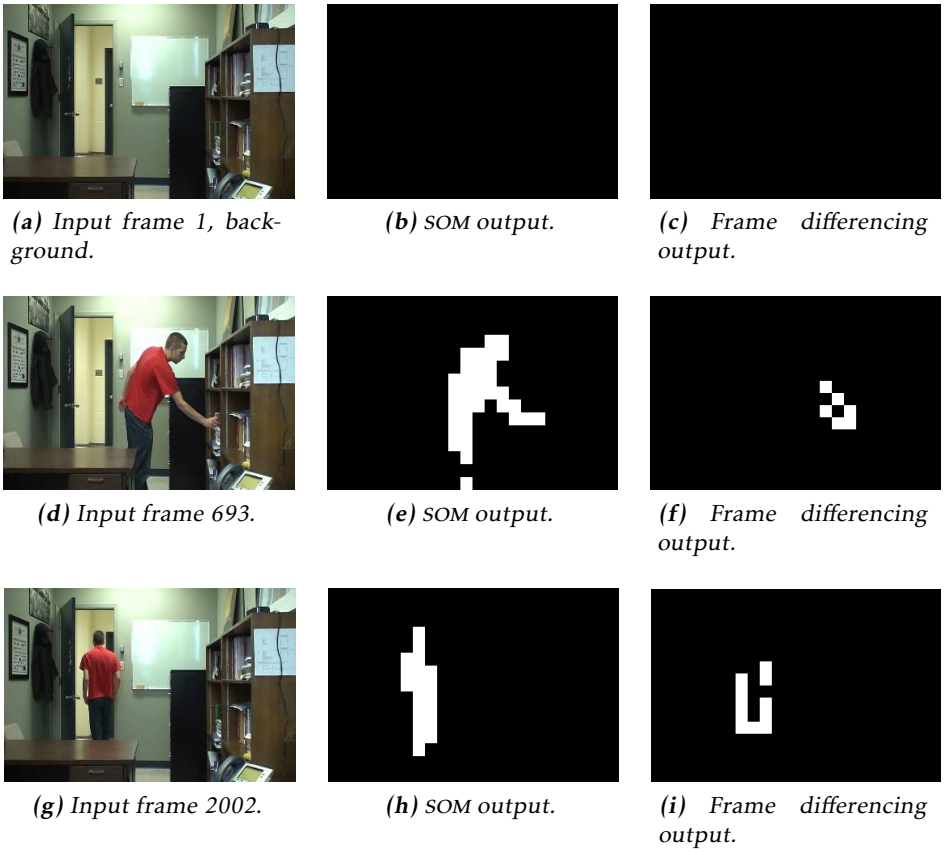
Table 4.1: *Overall evaluation scores for motion detection algorithms on the baseline and intermittent object motion sequences of the* CHANGEDETECTION.NET *dataset.*

Algorithm	***Precision***	***Recall***	***Specificity***	**FPR**	**FNR**	***F-Score***
SOM	0.592	0.477	0.904	0.096	0.523	0.476
Frame differencing	0.352	0.237	0.930	0.070	0.762	0.232
Difference	+0.240	+0.240	-0.026	+0.026	-0.239	+0.244
(In favor of SOM)	**+68%**	**+101%**	**-3%**	**+37%**	**-67%**	**+105%**

4.1 Motion detection

The SOM implementation was benchmarked against the baseline median filtering algorithm and the results can be seen in Table 4.1. As the table shows, the SOM implementation performs better than Frame differencing on four out of six measurements. The algorithms have used the same threshold and block size. No parameter tuning has been done between sequences, meaning the same threshold and block size has been used over all sequences. Neither algorithm performs exceptionally well on the benchmark which is as expected considering the simplicity of both algorithms. The improvement in *Precision*, just over 68%, means that when SOM detects movement it is more likely to be correctly predicted than Frame differencing. The improvement in *Recall*, just over 100%, means that SOM detects more of the true movement than Frame differencing. Lastly the overall *F-Score* was improved by more than 105%.

Sample results from the "office" sequence of the benchmark can be seen in Fig. 4.2. The most noticeable qualitative difference that can be seen when reviewing the output images from both algorithms is that the Frame differencing adapts to changes in the image too fast. Therefore incorrectly rejects true foreground pixels which has been stationary in the image for only a single frame.



(a) Input frame 1, background.

(b) SOM output.

(c) Frame differencing output.

(d) Input frame 693.

(e) SOM output.

(f) Frame differencing output.

(g) Input frame 2002.

(h) SOM output.

(i) Frame differencing output.

Figure 4.2: *Sample motion detection results on the "office" sequence from changedetection.net*

4.2 Camera sensor status

As a consequence of the remote location of the deployment site in the Ngulia Park, Kenya, one important aspect of the project was to ensure the reliability of the camera sensors. Also, the sensors needed to be self sufficient enough to keep running on its battery and connected solar panel. To achieve this, the camera sensor status system was implemented as described in Sec. 3.2.4. This created the opportunity to get an overview of all cameras and their status on a regular basis. All the cameras are set to send status updates when they wake up, goes to sleep and every 10 minutes. By doing so, it is possible to keep track of how a camera is operating and get insights on its battery status, wifi connection and sleeping cycle. If a camera goes offline, the log can be used to find out what's wrong with that specific sensor. In Fig. 4.3, an example of a camera's status log can be seen. This log shows that the camera went to sleep at 19, woke up to early

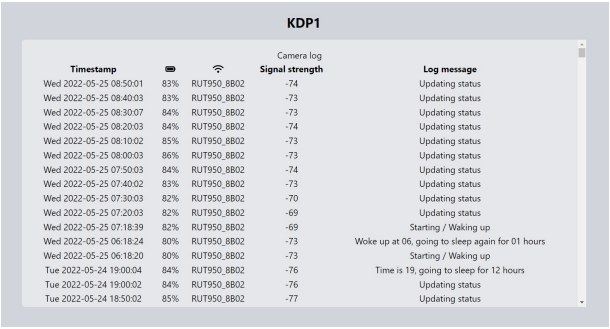


Figure 4.3: *An example of a camera's status log.*

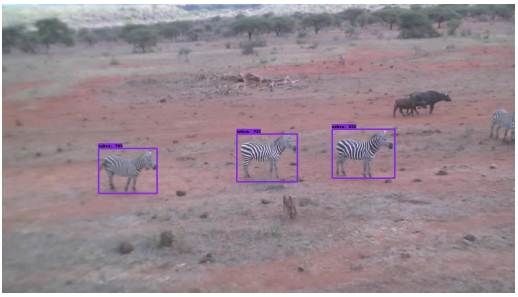
and went back to sleep again for 1 hour. After that, it starts the daily routine of taking images, running motion detection and sending status updates every 10 minutes.

4.3 Classification

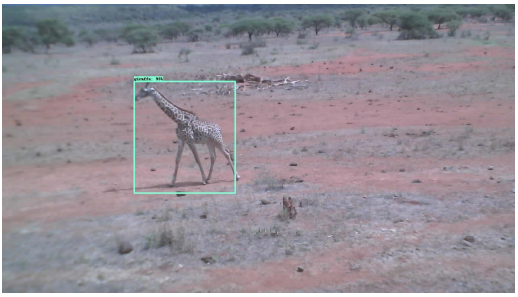
The classification step of the system is where the images get analysed and the objects found in the image get a class belonging with a score. The results from this step varies and are in many cases correct but there are also results where the classification is not correct. When the objects is further away from the camera and the image is less sharp, the classifier struggles more. It has problem to distinguish between animals that looks similar when their special features is not clearly visible (like the horns of the buffalo or the tusks and trunk of the elephants). However, the iterative classification process tends to correct many of the errors made in the first classification.

4.3.1 Qualitative results

In Fig. 4.4 some results can be seen from the image classification. The images are captured by the ESP32 Timer Camera with the software developed in this project. As can be seen in Fig. 4.4a and Fig. 4.4b, the classifier successfully detects the correct animals in the images. However, in Fig. 4.4c and Fig. 4.4d, the classifier does not manage to distinguish between animals that looks similar from afar and when the image is not clear enough. In this case it is the buffalo, elephant and rhinoceros that has a similar shape and color, making it harder to make a correct classification. Also in these cases, the error is often corrected by the iterative classification made in the next step.



(a) *Successful classification with Zebra 78%, 71% and 82% score.*



(b) *Successful classification with Giraffe 93% score.*

(c) *Classification error with rhinoceros 72%, but supposed to be buffalo.*

(d) *Classification error with rhinoceros 54%, but supposed to be elephant.*

Figure 4.4: *Different classification results.*

Table 4.2: *Score improvement from iterative classification on 500 images from cameras in Kilaguni. Objects with an initial score between 10–90% were considered.*

Number of objects	2155
Number of improved scores	773
Proportion of improved scores	35.9%
Average score increase among the improved scores	20.7%

4.3.2 Iterative classification

The iterative classification process resulted in increased accuracy of both the score and the correct class of the object in the image.

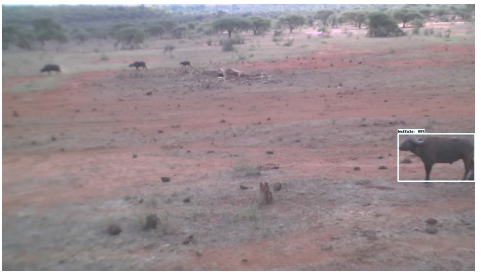
Quantitative results

The iterative algorithm was evaluated on 500 images from a camera which were deployed in at the Kilaguni Safari Lodge. The results can be seen in Table 4.2. The iterative classification resulted in an improvement in 35.9% of the objects which were found in the initial inference. The average increase was 20.7% among the increased scores. The algorithm is used as a complement to the initial inference and if the additional inference does not result in an increase in score it is simply discarded, which is why the results are focused around those scores that were improved.

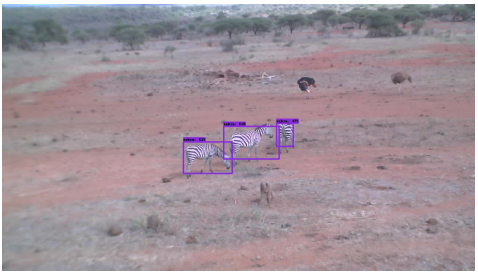
Qualitative results

Some examples of where the classification service has made a crop of the original image based on the first classification results and then run a new classification on that image can be seen in Fig. 4.5. These results show great improvement in many cases where the original classifications of the whole image do not reach a trustworthy score. However, it gives one or more ROI (Region of interest) that can be cropped and classified in one more iteration by the classification service.

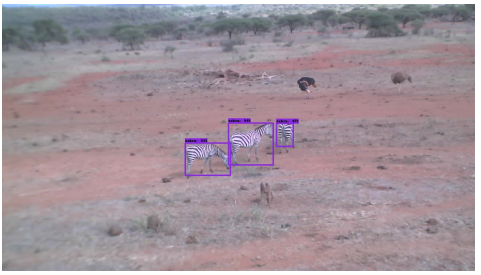
(a) Buffalo, initial score: 65%



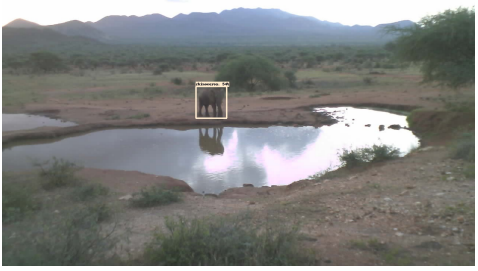
(b) Buffalo, improved score: 88%



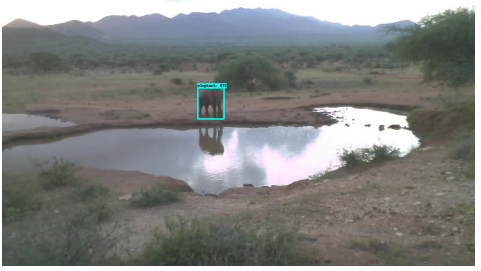
(c) Zebras, initial scores: 62%, 64% and 47 %. (From left to right)



(d) Zebras, improved scores: 93%, 94%, no improvement for the right most zebra.



(e) Elephant, initial score: 54% (with class rhinoceros)



(f) Elephant, improved score: 87% (with correct class)

Figure 4.5: *Improvements from the iterative classification algorithm.*

4.4 Developer Dashboard

The dashboard was implemented as a proof of concept of the capabilities of the system and how the data collected from the cameras could be used through the API to get a working end-to-end surveillance system. The functionalities implemented can later be integrated in the already existing dashboard that the park rangers in Ngulia uses on a daily basis. A user can select a specific camera and get all the latest images with the classification scores as well as the status of the

Figure 4.6: *Filter function applied to show images of rhinos from the last 24 hours.*

camera at a given point in time. The filtering functions also makes it possible to do more specific searches for a certain class in a certain time span. An example of a filtration made for rhinoceros from the last 24 hours can be seen in Fig. 4.6.

Another important feature was the ability to monitor the status of the cameras active in the system. Since they are put up at remote locations, it is valuable to know when a camera stops working and possibly why. The status overview is displayed in Fig. 4.7. For example, it can be seen that camera "KIL01" is online and has 99% of battery according to its last status update. If the user clicks on a specific camera in the list, the more detailed log is displayed like in Fig. 4.3.

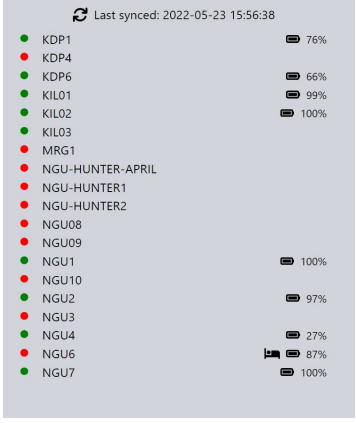


Figure 4.7: *Camera status overview.*

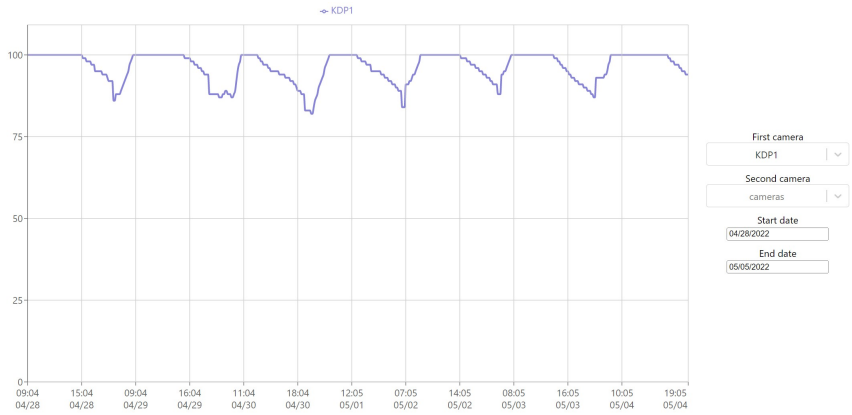


Figure 4.8: *Battery status plot for KDP1 during one week of the testing phase.*

4.5 Deployment

The cameras were deployed at three different locations in two phases, a testing phase in Kolmården Zoo, Sweden, and a deployment phase in Ngulia Sanctuary and Kilaguni Safari Lodge in Kenya. In Kolmården three cameras were put up for testing and evaluation. After two weeks, the cameras were updated with new software that had more stable sleep cycle and the battery fuel gauge was integrated. This resulted in valuable data about the units power cycle, internet connectivity and how they recharged during the day depending on the position of the sun. In Fig. 4.8, the battery status data for a camera in Kolmården is plotted during a week's time. The plot indicates that the camera charges well in the morning when the sun is shining directly at the solar panel, and after lunch it starts to drop when the solar panel lays in shadow from the wall it is mounted on. This gave valuable insights on the importance of the placement of the solar panel. But it also showed that even with limited solar power during the day, the battery was able to charge up to 100% during a few hours of direct sun. These were important insights for the work with the final deployment in Ngulia, where some cameras had to be placed in trees with limited sun-access and shadowing branches.

An important goal within Project Ngulia has for many years been to be able to setup a camera surveillance network in the Ngulia park. With the contribution of the work made in this thesis combined with previous theses, the system could be deployed in the Ngulia park. Seven cameras were put up around a waterhole within the park where animals often come to drink. However, it was noticed that the internet connection was not stable at that area which lead to the decision to also put up three cameras at the Kilaguni Resort. There the connection was more stable and because of the similarity to the waterhole at Ngulia in terms of animals and environment, valuable data and insights could be collected from these cameras as well. In Ngulia, the cameras were placed in different locations, some

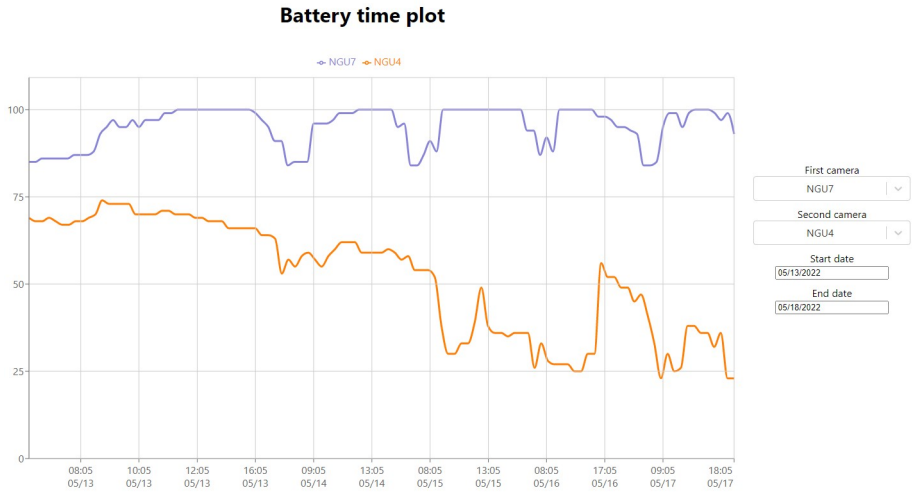
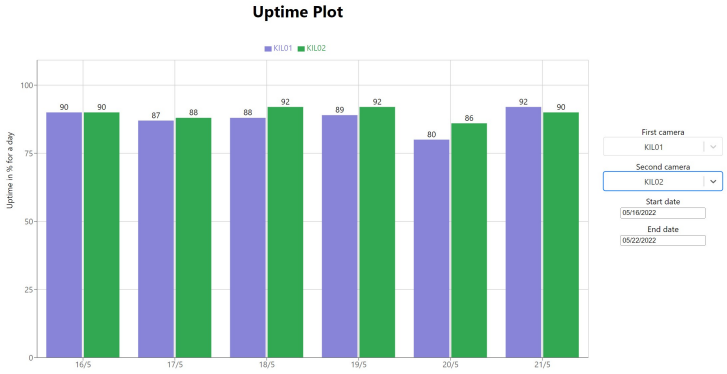


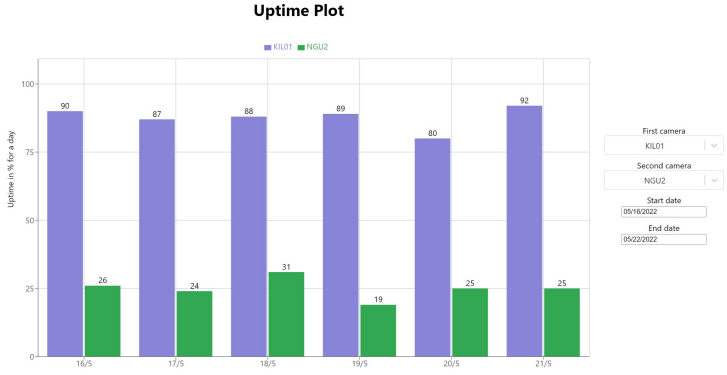
Figure 4.9: *Plots from two different cameras in Ngulia with different positions of the solar panel.*

in trees and some in a tree house overlooking the waterhole. In Fig. 4.9, plots from two different cameras show how the placement of the solar panel affects the battery percentage during the day of a week's time. The orange line shows the battery plot of the camera "NGU4" located in a tree that only has access to sunlight during some hours of a day, whilst camera "NGU7" plotted with the blue line, is located in a tree house with unobstructed access to sunlight. This shows that with uneven and obstructed access to sunlight, the battery level drops fast and becomes unstable. At the end of the week it already reached critical level of 20% or less, making it hard to keep a stable power cycle.

With the sensor data available from each camera, it was also possible to study the uptime for each device. Since the devices are set to send status updates every ten minutes when awake, i.e., six times per hour, it could be measured how many percentages of a day a camera is connected to the internet. In Fig. 4.10a the uptime percentage from two different cameras deployed at the Kilaguni Resort in Kenya can be seen during six days. The camera "KIL02" is placed close to the router whilst the camera "KIL01" is placed further away. This results in a slight improvement of uptime for the "KIL02", which can be seen in the plot. As mentioned, the internet connectivity around the waterhole in Ngulia was unstable at the time of deployment and in Fig. 4.10b, the uptime plot of the camera "NGU2", compared to "KIL01" can be seen. The plot shows that the camera is online around 25% of the day.



(a) *Uptime plot of two cameras in the Kilaguni Resort deployed with different distances to the router.*



(b) *Uptime plot of one camera in the Kilaguni Resort (KIL01) and one in the Ngulia Park (NGU2).*

Figure 4.10: *Uptime plots from different cameras.*

5
Discussion

In this chapter, the methods and results of the project are discussed and analyzed.

5.1 Smart surveillance system

The surveillance system implemented in this project provides valuable data to a user that a regular image feed can not do. The large amount of images provided from cameras mounted to surveillance such a large area as the Ngulia park would make it a time consuming task to manually go through them every day. The automated process of classifying images and saving data of interest developed in this thesis helps with this and provides the park rangers with extra "eyes" out in the field during their everyday work. By using micro-computers such as the ESP32, there are possibilities for custom modification of the software that is not possible on closed system of traditional trail cameras. With the ESP32, it would also be possible to implement two-way communication so that the cameras can be controlled and updated remotely. However, for the scope of this project, focus were to get as much valuable information as possible sent from the cameras to the server to study how they operate in a remote environment and how an image classifier will perform on the images sent. The use of a database and an API, makes it possible to query for specific data to make the selection presented to the end-user even more customized. This is a effective way of utilizing the data that can be provided by deployed sensors in the field.

5.2 Motion detection

The motion detection on the camera is the first step which filters images from the deployment site to the rest of the system. The goal with the motion detection is to remove as much non interesting images as possible while still keeping all the interesting images. This was the motivation behind the implemention of a motion detection algorithm which performed better than the previous baseline.

Sophisticated, state-of-the-art motion detection algorithms were not viable candidates for the ESP32 microcontroller because of its limited computing power. The constrained simplicity of the motion detection algorithm affects its performance and a trade off has to be made whether to filter more images, which increases the risk of discarding interesting images, or to send more images containing more noise and non interesting data. In this case the latter option is preferred since the images sent from the cameras will be further processed by a deep neural network on a machine with more computing power which will filter and remove even more of the non interesting images.

The SOM implementation provided an improvement on every measurement compared to the median filtering method while still being computational effective and fully usable on the ESP32 unit. The improved motion detection algorithm causes less overall load on the rest of the system since it will send fewer images for classification which is of greater importance as the number of cameras grow and the total number of images sent for classification increases.

When compared to RT-SBS-V2 the SOM implementation underperforms. That is unfortunately the current reality for algorithms which are meant to run on devices with limited computational power. Motion detection in video is a fundamental pre-processing step in computer vision and will continue to improve with time, even for devices with limited computational power.

5.3 Classification

Most of the neural network and classification parts of the system could be reused from Arnesson & Forslunds previous implementations. Since the same dataset used to train the model has not changed the overall performance of the model has not changed since their implementation.

The idea of letting the classifier run inference multiple times on the same image was inspired by the models ability to give more accurate predictions when an object covered a large part of the input image. The model can often find small objects in an image with a low probability for the class, but when the model is supplied a smaller, cropped image of the same object it gives much higher probability for a class. The evaluation of the iterative classification algorithm shows that it is a good complement to the classification process. The average increase of 20.7% among the 35.9% of improved scores is a welcome improvement. Because the iterative process is a complement to the overall classification process, scores

which is not improved is not interesting to look at since they would be discarded anyway. One theory of the high average improvement is because of the overall performance of the model and relatively low amount of data it is trained on. It is possible that the iterative classification would be redundant for well performing models trained on large datasets.

While not the prime objective, the images received from the recently deployed cameras, both in the Ngulia Sanctuary and Kilaguni resort, can be used in the future to further increase the number of images in the dataset used for training and improve the performance of the model. This would result in a closed-loop system which supplies itself with new images for the model to train on. This is an important addition to the current system because one of the long standing issues for classification performance, as noted by both Olsson & Tydén and Arnesson & Forslund, is the lack of images to train the model on. The current available dataset of 3843 images are not enough to create a robust and accurate model. The steady stream of new images from the deployed cameras gives an opportunity to improve the model and in turn make the surveillance system more reliable as a whole. The new images will also be perfect candidates for the model to train on since they are captured from the target environment and have the same quality as the model will have as input when running inference.

5.4 Camera Sensors

The results presented in Sec. 4.5 shows that the cameras are self sufficient in their power cycle with the power of the sun, as long as the solar panel is placed in a good location. Also if the power gets low after a few days of bad weather, the unit can recharge and continue to send data, without any manual support needed. This was one of the main goal of this thesis, to enhance and ensure the reliability of the camera sensors, so that they could be deployed in a remote environment such as the Ngulia Park in Kenya. However, the units are still very reliable on sufficient internet connection to provide valuable data. In the cases where the units are only able to send updates a few times a day, the reliability when it comes to activity reports is not sufficient. The image quality from the camera sensors is also something that affects the possibilities for correct classifications. The cameras tend to struggle with objects that are far from the camera which becomes out of focus, making it harder for the classifier to detect and analyze the edges of the object. The concept of using edge-devices in a smart surveillance network is however proved to work well if the internet connection can be stabilized. And for the project in general, the implementations and studies made in this thesis have been an important contribution from where more improvements can be made for future work.

5.5 Deployment

The testing phase gave important insights of the reliability of the cameras. The three cameras which were installed were power self sufficient with the help of the solar panel. The number of sun hours for these cameras were limited because of the installation site. The cameras were installed such that two would charge through the solar panel from sunrise to about 13.00 mid day, the last one would charge through the solar panel from 12.00 mid day until sunset. The test deployment gave confidence that the cameras would be able to power themselves with the battery and solar panel in Kenya considering the Sweden has less sun hours than Kenya overall as well as the limitation from the installation site.

The deployment phase was mostly a success. Everything which could be tested in the testing phase also worked when deploying the cameras in Kenya. The only major issue was the unreliable network connection in the Ngulia Sanctuary. The router which was installed could not establish a stable connection which in turn hinders the cameras to upload images and send status updates. This was not something that was tested before since the network connection in Kolmården was stable and caused no issues. Kilaguni was more similar to Kolmården with stable network connection and the deployment had no issues.

6 Conclusion

The aim for this thesis was to investigate methods of building a smart surveillance system for a remote environment using Edge-devices as camera sensors for collection of data. The aim was fulfilled and resulted in a complete end-to-end surveillance system that can be tested and evaluated for a longer period of time since the deployment in Kenya. Further solutions for improvement have been tested and suggested as a result of the development and studies during this project. In this section, the research questions of the thesis are further discussed and answered.

6.1 Research questions

1. **How can Self-Organizing Maps be used for motion detection on the ESP32 and how does it perform compared to Frame differencing?**

   The image can be divided into equally sized blocks and each block is the representation of that area of pixels by calculating the average pixel value within each block, effectively low pass filtering the image. Each block can then have a SOM which represents the possible background values for that block. By downsampling a block of pixels to a single value the computational load of the algorithm on the ESP32 is decreased. Anomalies, which implies motion, can be detected by the quantization error when new samples are presented to the SOM. If the quantization error between the closest matching prototype neuron in the map and the new sample is larger than some predefined threshold, it can be considered an anomaly and motion is detected in that block of the new image. The self adjusting nature of the algorithm means it is adaptable to changing conditions in the frame over

time.

The SOM provided an improvement on most measurements when tested on sequences from the CHANGEDETECTION.NET dataset. Most notably a 105% increase in *F-Score* over the Frame differencing implementation. From the results it was established that SOM was better performing algorithm than Frame differencing. However, it is still far from the current state of the art unsupervised-methods, such as RT-SBS-V2.

2. **How can the performance of an image classifier be increased for low scoring objects which cover a small area of the image?**

   The bounding boxes from the low scoring objects can be used as regions of interest. Cropping around the bounding box results in a new image in which the object will cover a larger percentage of the image. Then inference can run on the smaller image and improve the score as well as correct the object class. This iterative classification process provided an improved score on 35.9% of object which were detected on an evaluation set of 500 images. Among all the improved scores, the average increase of score was 20.7%.

3. **How can a robust and smart surveillance system be designed, implemented and deployed with a remote target area? How can the challenges that comes with developing from afar be treated?**

   One of the contributions of this work is a smart surveillance system with ESP-32 Timer Cameras as sensors, which can be deployed and run in remote environments like the Ngulia Park in Kenya. The system also handles all the data provided by the sensors and shows different ways of presenting and using them in a user interface. The system is developed with a certain modality, making it suitable to integrate in the already existing system that is used by the park rangers in Ngulia. With the data logging system implemented, the reliability of the sensors power cycle and connectivity can be evaluated and improved. Also, the system can work as a template for other types of surveillance tasks and it is not limited to Ngulia Project. The classification model can be retrained on other types of animal so that the core structure can be reused for other areas where surveillance and protection is of importance. It can be concluded after this project that simulating the deploy environment as similar as possible is important to be able to face the challenges of unstable network, power loss and other circumstances that can occur when deploying technical solutions out on the African Savannah. The twin station built in Kolmården contributed to valuable insights that were later applied to the final deployment in Kenya.

6.2 Future work

The work in this thesis has given insights in the different parts that make up this smart surveillance system and what the major challenges are. Although the system is working in its current state, there are improvements that can be made. There is a need of more training data for the classification model to be more reliable, which would improve the end result as well. It would be interesting to investigate how the process of annotating images could be more automated, since it is a time consuming task. Also, a workflow for continuously re-training the model when more data gets collected, would improve the system over time.

There are also limitations with the M5Stack Timer Cameras being used at the moment. Although they are power efficient, the camera sensor itself performs quite poor when it comes to image quality and there are limitations with the sensor memory, making it hard to hold several images of higher quality during run time. It would be worth investigating which other options there are when it comes to hardware and if possible find the best mix of cost, memory, quality and power consumption to fit the needs in this kind of system. Since the sensors are dependent on internet connection they basically becomes "blind" when there is no connection. If the images could be stored on a memory card and then sent when internet comes back, it would still provide valuable data to the rangers, although with a varying delay. Also, like described in the work made by Arnesson and Forslund, running a lightweight classification model on the edge-device as a backup would be worth looking further into [8]. If a reliable classification could be made directly on the device, the resulting metadata could be sent with LoRa (Long Range Network) instead, which has longer reach than traditional internet protocols.

Bibliography

[1] Project ngulia. URL http://www.projectngulia.org/.

[2] M5stack timer camera. URL https://docs.m5stack.com/en/unit/timercam.

[3] Anthony Cioppa, Marc Van Droogenbroeck, and Marc Braham. Real-time semantic background subtraction. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3214–3218, 2020. doi: 10.1109/ICIP40778.2020.9190838.

[4] IBM Cloud. What are convolutional neural networks? https://www.ibm.com/cloud/learn/convolutional-neural-networks. Accessed: 2022-05-05.

[5] Crisluengo. Effect of applying the negated laplace operator, 2021. URL https://commons.wikimedia.org/wiki/File:Vd-Rige2.png.

[6] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6568–6577, 2019. doi: 10.1109/ICCV.2019.00667.

[7] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, Jun 2010. ISSN 1573-1405. doi: 10.1007/s11263-009-0275-4. URL https://doi.org/10.1007/s11263-009-0275-4.

[8] Johan Forslund and Pontus Arnesson. Edge machine learning for wildlife conservation: Detection of poachers using camera traps. *Master Thesis*, 2021.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. doi: 10.1109/CVPR.2014.81.

[10] Nil Goyette, Pierre-Marc Jodoin, Fatih Porikli, Janusz Konrad, and Prakash Ishwar. Changedetection.net: A new change detection benchmark dataset. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, 2012. doi: 10.1109/CVPRW.2012.6238919.

[11] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[12] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, Jan 1982. ISSN 1432-0770. doi: 10.1007/BF00337288. URL https://doi.org/10.1007/BF00337288.

[13] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10602-1.

[14] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 483–499, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46484-8.

[15] Sara Olsson and Amanda Tydén. Edge machine learning for animal detection, classification and tracking. *Master Thesis*, 2020.

[16] Francisco Ortega-Zamorano, Miguel A. Molina-Cabello, Ezequiel López-Rubio, and Esteban J. Palomo. Smart motion detection sensor based on video processing using self-organizing maps. *Expert Systems with Applications*, 64:476–489, 2016. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2016.08.010. URL https://www.sciencedirect.com/science/article/pii/S0957417416304031.

[17] Michael Plotke. Before any kernel convolution, 2013. URL https://commons.wikimedia.org/wiki/File:Vd-Orig.png.

[18] Pertab Rai and Murk Rehman. Esp32 based smart surveillance system. In *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–3, Sukkur, Pakistan, 30 - 31 January 2019. doi: 10.1109/ICOMET.2019.8673463.

[19] G. Steiner. Transfer of learning, cognitive psychology of. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*, pages 15845–15851. Pergamon, Oxford, 2001. ISBN 978-0-08-043076-8. doi: https://doi.org/10.1016/

B0-08-043076-7/01481-9. URL https://www.sciencedirect.com/science/article/pii/B0080430767014819.

[20] Dan Stowell. An illustration of the training of a self-organising map, 2010. URL https://commons.wikimedia.org/wiki/File:Somtraining.svg.

[21] Jason Yosinski, Jeff Clune, Y. Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NIPS)*, 27, 11 2014.