Master of Science Thesis in Media Technology
Department of Electrical Engineering, Linköping University, 2021

Edge Machine Learning for Wildlife Conservation

Detection of Poachers Using Camera Traps

Johan Forslund and Pontus Arnesson

Master of Science Thesis in Media Technology

Edge Machine Learning for Wildlife Conservation: Detection of Poachers Using Camera Traps

Johan Forslund and Pontus Arnesson

LiTH-ISY-EX--21/5380--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

This thesis presents how deep learning can be utilized for detecting humans in a wildlife setting using image classification. Two different solutions have been implemented where both of them use a camera-equipped microprocessor to capture the images. In one of the solutions, the deep learning model is run on the microprocessor itself, which requires the size of the model to be as small as possible. The other solution sends images from the microprocessor to a more powerful computer where a larger object detection model is run. Both solutions are evaluated using standard image classification metrics and compared against each other. To adapt the models to the wildlife environment, *transfer learning* is used with training data from a similar setting that has been manually collected and annotated. The thesis describes a complete system's implementation and results, including data transfer, parallel computing, and hardware setup.

One of the contributions of this thesis is an algorithm that improves the classification performance on images where a human is far away from the camera. The algorithm detects motion in the images and extracts only the area where there is movement. This is specifically important on the microprocessor, where the classification model is too simple to handle those cases. By only applying the classification model to this area, the task is more simple, resulting in better performance. In conclusion, when integrating this algorithm, a model running on the microprocessor gives sufficient results to run as a camera trap for humans. However, test results show that this implementation is still quite underperforming compared to a model that is run on a more powerful computer.

Acknowledgments

We would like to thank our examiner Fredrik Gustafsson for the support, both in terms of guidance through the project and generous hardware supply. We would also like to give appreciation to our supervisor Magnus Malmström, who continuously has given valuable feedback on the thesis. Furthermore, we would like to thank Martin Stenmarck at HiQ for setting up the FTP server, Sara Olsson and Amanda Tydén for inspiration and annotated images, Sara Svensson for great insights, HiQ for supplying office space and finally Kolmården for allowing us to use their zoo as a testing area.

Norrköping, June 2021
Johan Forslund and Pontus Arnesson

Contents

Notation

Abbrevations

Abbreviation	Meaning
ADABOOST	Adaptive Boosting
ADAM	Adaptive Moment Estimation
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
COCO	Common Objects in Context
HOG	Histogram of Oriented Gradient
MCU	Microcontroller Units
RELU	Rectified Linear Unit
ROI	Region of Interest
SNTP	Simple Network Time Protocol
SSD	Single Shot Detector
SVM	Support Vector Machines

1 Introduction

Poaching is one of the reasons why many wildlife populations are becoming extinct. However, it is tough to combat poachers since it is simply too much area to cover and protect. Animals of extinction species are put in sanctuaries where park rangers can patrol the area and report any illegal activity. An interesting addition to the park rangers would be to install camera traps with smart algorithms that can automatically detect humans that are not allowed to enter the sanctuary, and also communicate this information to a backend service that alarms park rangers. Multiple machine learning solutions exist that can detect humans in images, though many focus on optimizing the performance on mid- to high-end machines. To apply this in sanctuaries, there is a natural need to use power sufficient machines where carefully crafted algorithms have to be used due to computational limits.

1.1 Background

Black rhinos are on the verge of extinction. From 1970 to 1995, the number of black rhinos in the world decreased from 65000 to 2400 [27]. Since then, there has been a slow increase to about 5600 in 2012 [34], but the species is still endangered. Therefore, the Ngulia park in Kenya has, in collaboration with other actors, been trying to stop the rapid decrease in the population of these animals. The Ngulia park is a rhino sanctuary located in the south east part of Kenya. The total area of the park is about 100 km^2 and home to about 80 black rhinos.

Project Ngulia is a collaboration between Linköping University, HiQ, Kolmården Djurpark and Ngulia park that attempts to assist the park rangers in the sanctuary with technical solutions, allowing the rangers to have more control over the large landscape and therefore reduce the threat of poachers. Linköping University is mainly responsible for researching how technical solutions can be applied to solve the problem, mostly focusing on how different sensors can be applied. HiQ has created an online service to allow the park rangers to report traces of activity in the park via a dashboard that can be accessed from their mobile phones. In the dashboard, the park rangers can create reports containing pictures of the traces together with the current location and a description. To more easily test the product, Kolmården Djurpark has allowed sensors to be set up in their park to monitor animal activity and evaluate the technology.

The recent increase of available data and computational power has made it possible to use deep learning methods effectively. This has been widely used in the computer vision field for tasks such as object detection, where it outperforms traditional computer vision techniques. A previous work within Project Ngulia with the aim of automatically detecting the rhinos using deep learning has been done [29]. By sending the location of the rhinos to the dashboard, the park rangers get a better overview of the park. There has, however, not been any project with the primary objective to locate poachers.

1.2 Aim

The aim is to research methods for detecting hostile human activity on the savanna, and applying those methods to reduce the risk of animals being poached. The information acquired from the detections will then be used to alert the park rangers via a backend service. In this work, the main focus is on evaluating and comparing object detection algorithms for both low- and high-end devices. Additionally, these algorithms will be incorporated in a pipeline stretching from capturing images to sending reports to a backend service. The solution should be suited for the conditions on the site in Kenya.

1.3 Research Questions

1. How can the performance of an image classifier be increased when the target object covers a small area of the image?
2. To what extent can a microcontroller be used to perform image classification using deep learning?
3. How to avoid that objects that look similar to humans get misclassified as humans?

1.4 Limitations

Since the camera traps are placed in the middle of the savanna, power-efficient devices are necessary. The placement of the camera traps also means poor internet connection which limits the amount of data that can be sent. Thus, the images that are processed will be reduced in size from their original resolution. Furthermore, it is necessary to adapt the hardware to bad weather conditions that may occur. Lastly, there will be limited human maintenance possibilities which makes it harder to perform software updates. This will also cause problems if one of the devices stops working because it cannot be instantly physically accessed.

2 Related Work

The task of detecting humans in images has been widely researched in the computer vision field. Initially, such algorithms required handcrafted feature extractors to detect human presence in images. In recent years, as the deep learning field has progressed, it has become more common to utilize neural networks as a classifier for this purpose. Furthermore, it has also become possible to utilize deep learning in edge devices. Edge devices are resource-constrained units, for example microcontrollers, single-board computers such as a *Raspberry Pi* or smartphones [28]. This opens up for running relatively accurate human detectors on small and lightweight devices anywhere. In this chapter, related work on these topics is presented and briefly described.

2.1 Human Detection

Before deep learning, there have been many attempts to detect humans in images with traditional computer vision techniques. These techniques are still highly usable and have the advantage of not needing a huge set of training data. Most of the traditional techniques however requires some sort of manually handcrafted feature extractor to detect human features, which can make these techniques less versatile in different settings and environments.

Viola and Jones [37] revolutionized the field in 2001 as they created the first real-time detector for human faces, achieving speed that was 10-100 times as fast as other algorithms with similar detection accuracy. Their algorithm has later been referred to as the Viola-Jones detector. The algorithm uses Haar-like features to extract important information from the input image, such as edges and lines, which is useful to detect common features in a human face. Since the process of extracting Haar-like features involves calculations over rectangle areas in the image, it is important to be able to run through each area efficiently. Therefore, the input image is first converted to an integral image, which is a data structure for generating the sum of values in a rectangular subset of a grid. The integral image allows the algorithm to compute Haar-like features at any scale or location in constant time. The *adaptive boosting* (ADABOOST) algorithm is then used to extract the best subset of features from all possible features. The subset of features are then used to create the classifiers that build up a cascade classifier. The input image is then fed through a sliding window method where each block of the image is processed by the cascade classifier. This method uses a multi-stage process where the input is fed through increasingly stronger classifiers where the input is immediately discarded if any of the classifiers output a negative result. By using this approach, the algorithm will generally spend little time on processing background areas, and more time on processing the areas where faces are present. Thereby, this algorithm is highly efficient and is still used for detecting human faces in modern software.

Dalal and Triggs [9] suggested another method that is specifically designed for human detection that considers the entire body, not only the face. They used locally normalized *histogram of oriented gradient* (HOG) descriptors as the feature set and proved that this gave excellent performance compared to Haar-like features in the case of human detection. The HOG representation was chosen as it is invariant to local geometric and photometric transformations. As classifier, they used support vector machines (SVM). Chao Mi, et al [26] improved the speed of this algorithm by an optimized algorithm that avoids a large number of repeated calculations.

Songmin Jia, et al [33] uses a template matching technique to perform human detection in video sequences. The input images are first processed by a background subtraction model to extract a mask of foreground objects. A head-shoulder model is then applied to each object to extract only the part from head to shoulders. The head-shoulder masks are then compared to a template via varying template scale matching, which is the final detection step.

The field of object recognition entered a new era as Yann Lecun, et al [23] presented a method for object recognition using *convolutional neural networks* (CNN). This started the wave of deep learning where much focus has been put into image-based tasks. Szegedy, et al [36] took this one step further by presenting a way to do object detection using CNN, in a method where the algorithm finds the actual positions of the recognized objects in an image. Further advances have since then been made in this field, as described in Chapter 3.

When classifying humans, the different characteristics of the person might play a big role. Joy Buolamwini and Timnit Gebru [7] shows that both gender of the subject, and the color of the skin can have a huge difference when classifying the gender of faces. Joy and Timnit present results showing that error rates as high as 34.7% can occur for dark-skinned females when using a commercial tool for gender classification, compared to light-skin males where the maximum error rate was as low as 0.8%.

2.2 Non-urban Environment

A lot of related work on human detection contains a majority of data from urban environments. Since this thesis aims to detect humans in a very different setting than what is presented in most other work, several considerations have to be made. Zachary Pezzementi et. al [30] highlights some areas which can have a great impact when comparing off-road and urban environments for human detection. These areas include the color and texture of the environment, poses and occlusion of the object as well as other natural factors. Even though there is a variance in these areas within urban environments as well, off-road human detection introduce even more dissimilarities which can make a detector pre-trained on urban environments less accurate when applied to off-road images.

2.3 Object Classification on Edge Devices

Since edge devices such as Raspberry Pi or microcontroller units (MCU) have a very limited amount of memory and computational powers, as well as often restricted power usage, the full classification pipeline has to be adjusted accordingly. In [13], Nikouei et. al presents real-time human detection on the Raspberry Pi with the use of L-CNN. By narrowing down the classifiers search area to focus on human objects, the L-CNN algorithm is able to perform pedestrian detection with affordable computations on the Raspberry Pi. However, microcontrollers have even more constraints than Raspberry Pi. In [11], Liberis and Lane focuses on how to deploy a neural network on MCUs with as little as 512KB SRAM by minimizing peak memory usage of a neural network through changing the evaluation order of its operators.

3 Theory for Deep Learning

Deep learning is a sub-field of machine learning that has gained a lot of attraction in recent years. The algorithms that drive deep learning are loosely guided by the function of the brain [16] and are generally called *artificial neural networks* (ANN). Similar to other machine learning methods, an ANN is a black-box model that is used to predict future outputs, which for this project is information about if and where a human is in an image. The model between input and output depends on the internal weights of the network. The training process is done by feeding manually labeled training data to the network and then adjusting the internal weights of the network such as the network is tuned to fit the input data to the ground-truth labels.

3.1 Components of a Neural Network

A neural network is composed of multiple nodes (sometimes called neurons). The nodes are interconnected through different layers as can be seen in Figure 3.1, which illustrates a small network with an input layer, a hidden layer and a output layer. Normally, neural networks have a large number of hidden layers instead of only one as in this example.

Each connection is weighted according to the learned weights w_k of the network and these are initialized randomly with a weight initialization method. A commonly used weight initialization method is *Xavier initialization* [15], which was then refined into *Kaiming initialization* [19]. Each layer also includes a bias weight b_k that makes it possible to shift the output with a constant. The input data goes into the first layer of the network and then flows through the subsequent layers, where each layer applies a nonlinear mapping, also called activation function.

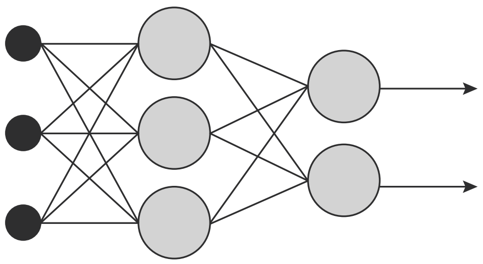
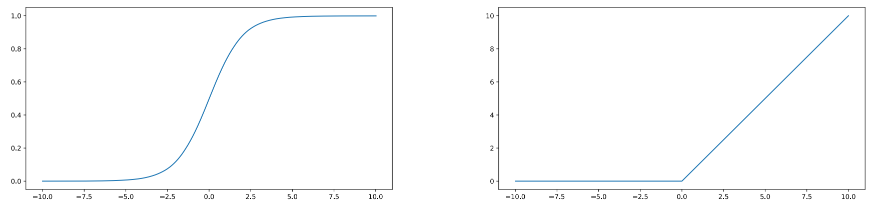


Figure 3.1: *Neural network with vertically aligned layers of nodes.*



*(**a**) Sigmoid activation function.* *(**b**) RELU activation function.*

Figure 3.2: *Two commonly used activation functions.*

Without the nonlinear activation functions, the network would only be able to learn problems that are linear in the input. Two common activation functions to use are the *sigmoid* and *rectified linear unit* (RELU) functions, illustrated in Figure 3.2. For the rest of this thesis, the output after applying the activation function is defined as a_k.

3.2 Gradient Based Learning

The basis of deep learning is gradient based learning. That is the mathematics behind learning the weights of the network in a supervised manner. Below is a mathematical description for a simple network, though the theory still applies to more complex networks.

For a simple network with one input layer and one output layer, the output of the node in the output layer can be defined as

$$z_1^{(l)} = b_1^{(l)} + \sum_{k=1}^{n_{(l-1)}} a_k^{(l-1)} w_k \, , \tag{3.1}$$

where each exponent is the index of the layer and the subscript is the index of the node. Thus, the final output of the network is

$$a_1^{(l)} = \sigma(z_1^{(l)}) \, , \tag{3.2}$$

where σ is the chosen activation function. When the input data has been propagated through the network, an error function E is used to measure how close the network output is to the true output y that is supplied during training. One such error function is the square difference function

$$E(a,b) = (a-b)^2\,. \tag{3.3}$$

The output of the error function determines the error of the network on the input data, which becomes

$$E = (a^{(l)} - y)^2\,. \tag{3.4}$$

To optimize the performance of the network, the output of the loss function should be minimized. This can be done with a method called *gradient descent*. The idea is to adjust the weights of the network in relation to the gradient of the error with respect to the weights. To find the partial derivatives of E with respect to w the chain rule is applied as

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial z^{(l)}}{\partial w}\,, \tag{3.5}$$

where each factor is expanded below for the last layer in the network.

$$\frac{\partial E}{\partial a^{(l)}} = 2(a^{(l)} - y)\,, \tag{3.6}$$

$$\frac{\partial a^{(l)}}{\partial z^{(l)}} = \frac{\partial \sigma(z^{(l)})}{\partial z^{(l)}} = \sigma'(z^{(l)})\,, \tag{3.7}$$

$$\frac{\partial z^{(l)}}{\partial w} = \frac{\partial (b + a^{(l-1)}w)}{\partial w} = a^{(l-1)}\,. \tag{3.8}$$

By combining (3.6), (3.7) and (3.8) the final expression becomes

$$\frac{\partial E}{\partial w} = 2(a^{(l)} - y)\sigma'(z^{(l)})a^{(l-1)}\,. \tag{3.9}$$

The weights of the network are then updated as

$$w^{(l)} = w^{(l)} + \alpha\frac{\partial E}{\partial w^{(l)}}\,, \tag{3.10}$$

where α is a set learning rate that determines how large steps to take in the direction of the gradient. Using a too large learning rate can make the optimization diverge while a too small learning rate will slow down the training. The weight updates are done multiple times until the solution starts to converge.

The method described above is called backpropagation and is fundamental in the learning process of a neural network. A more general description of backpropagation is described by Magnus Malmström [25] and Ian Goodfellow, et al [17] that takes into account networks that are more complex than this one-input-one-output network.

3.3 Optimizers

The task of minimizing the loss is an optimization problem where different optimization algorithms have been suggested. Equation (3.10) is known as *vanilla gradient descent* and is the most basic optimization algorithm used for training neural networks. In the most basic form, this algorithm tries to optimize the weights after running the full dataset through the network, which may take a very long time if the dataset is large. A better solution is to use *stochastic gradient descent*, where the dataset is fed through the network in smaller batches, and the weights are updated after each batch.

Another technique to stabilize and fasten the training process is called *momentum*. This technique dampens the effect of noisy gradients by adding a *momentum term* γ. The weights are now updated as

$$\begin{aligned} v_t &= \gamma v_{t-1} + \alpha \frac{\partial E}{\partial w} , \\ w &= w + v_t , \end{aligned} \tag{3.11}$$

where v is a velocity term. γ is usually set to 0.9 or a similar value.

This can be further extended by using momentum of both first and second order, which is the case in *adaptive moment estimation* (ADAM), which can give significant improvement on sparse gradients [38]. The authors Diederik P. Kingma and Jimmy Ba [21] show empirically that this optimization algorithm gives good results in practice and compares well to other optimization algorithms. ADAM has become the standard optimizer to use within the deep learning field.

3.4 Convolutional Neural Networks

A class of deep learning networks is CNNs which exclusively processes array data such as images. The weights in such networks are represented in a set of learnable filters. The filters are used to find important features and create maps representing the main characteristics of the input. Hence, spatial dependencies can be learned, which is suitable when dealing with images.

As opposed to the standard ANN, the input does not have to be flattened to 1 dimension, making it easier for the network to learn on images since the spatial structure of the images is preserved [22]. Another important feature of CNN is weight sharing. A filter (representing the weights) acts on a certain receptive field of the image, and the filter does not change as it moves through the image. Thus, if the filter has learned to detect a certain feature in one part of the image, it will still keep that knowledge for other parts of the image.

3.4.1 Filters

A filter can be visualized as a 2D grid where each position in the grid represents one weight in the network, see Figure 3.3 for an example filter with size $3x3$. The

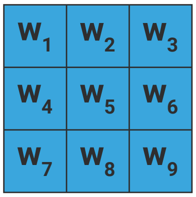


Figure 3.3: *Filter with corresponding weights w_k.*

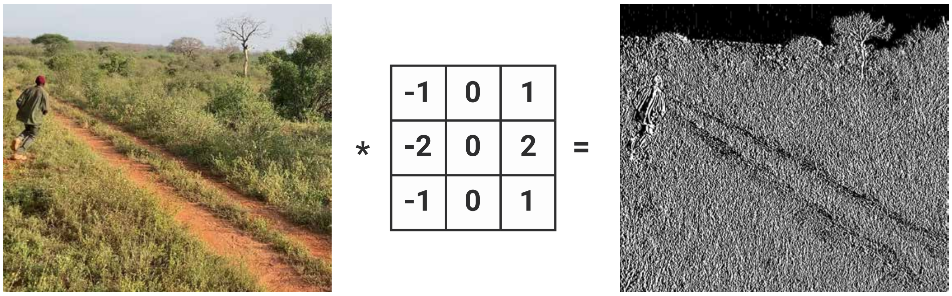


Figure 3.4: *Filter applied to an image to extract vertical edges.*

size may vary but is most often square.

The filters of the network are trained to detect different features in the input image. For example, one filter could extract vertical edges in the image, as seen in Figure 3.4. Generally, after the network has been trained for a while, the filters in the beginning layers will extract common features such as edges while the filters in the last layers will extract more use-case specific features [5], for example the shape of a nose when the goal is to detect humans. To actually extract the features, the filters are applied to the input image via a convolution process.

3.4.2 Convolutions

As explained above, the filters are applied to the input image with the use of convolutions. This is a common technique in image processing and the operation is highly optimized on modern computers, hence why it is suitable for deep learning.

Mathematically, if the input is a sampled two-dimensional signal, the convolution is defined as

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)\,, \tag{3.12}$$

where I is the input signal and K represents the filter. In the case of images, this process can be visualized by sliding the filter over the input image and multiplying the overlapping values element-wise, see Figure 3.5.

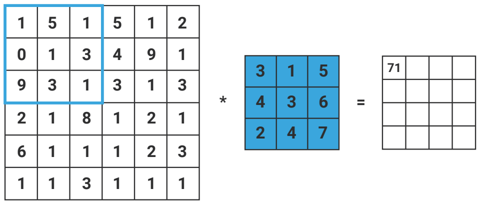


Figure 3.5: *Convolution between image and filter.*

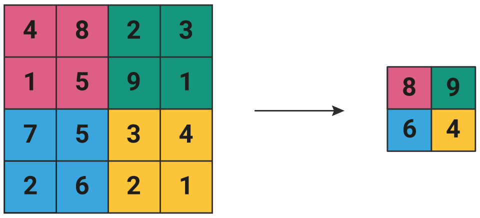


Figure 3.6: *2x2 max-pooling.*

3.4.3 Pooling

Pooling is a down-sampling method to reduce the dimensions of the feature maps. The most common form is max-pooling, where the feature map is divided into patches where only the maximum value of each patch is kept, thereby shrinking the size of the feature map. This is visualized in Figure 3.6.

3.4.4 Stride

Another alternative to reduce the dimensions of the feature maps is to use *stride*. Stride is done at the convolution layer, whereas pooling is done in a subsequent pooling layer. The way stride reduces the size of the image is to shift the filter a certain amount of steps each time the convolution filter is applied. With a stride of one, the filter is shifted one unit and therefore not reducing the dimensions. Using a stride of 2 for example, the filter skips one unit and therefore produces a smaller output dimension.

3.4.5 Padding

When applying the filters to the input image, the pixels in the corner of the image does not get covered the same number of times as the ones in the middle. This leads to both a reduction in size every time a convolution is performed, and a loss of information in the corners of the image. To prevent both of these, padding can be used. Often a version called *zero-padding* is used, where the extra pixels are set to zero. In Figure 3.7, zero-padding is applied to make sure the whole filter fits inside the image.

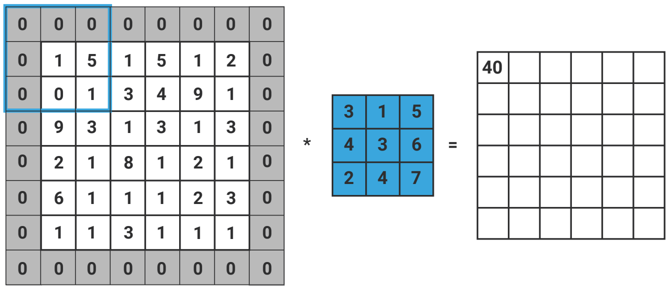


Figure 3.7: *Zero-padding applied to the input image.*

3.5 Network Architectures

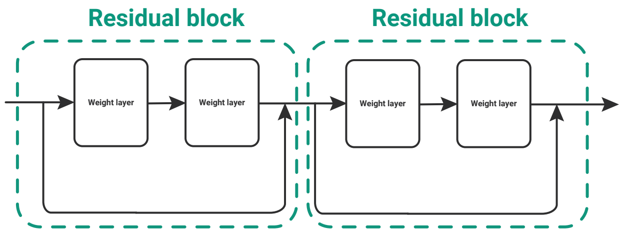
A neural network can be constructed in a lot of different ways. For example, one needs to decide how many layers the network should contain, how many nodes there should be in each layer, what kind of activation function each layer should have and so on. It could be easy to think that for example adding more layers would yield better accuracy, but that is not always the case [6]. Careful considerations has to be made when creating the structure of the network. There are however several popular network architectures that are known to be effective, and therefore commonly used in practice. These networks often have some extra features to make the network more efficient. A couple of these are ResNet, Inception and MobileNet.

3.5.1 ResNet

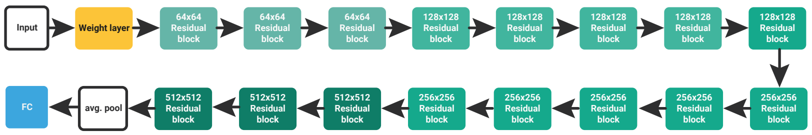
ResNet uses so-called "skip connections" to be able to handle neural networks with a larger amount of layers [18]. The skip connection takes the output of the activation functions of a layer and adds that output to the layer two or more steps ahead. This connection divides the network into several residual blocks, each block consisting of the two or more layers the connection encloses. These residual blocks make the result better or equally good as the shallower counterpart. This is because the weights can always be pushed towards zero to approach the identity mapping [18], which asserts that the residual block achieves at least equally good performance as if only a shallow network was used. An example of a residual block and the general structure of a ResNet can be seen in Figure 3.8.

3.5.2 Inception

The Inception architecture uses several different filters in parallel together with max-pooling layers, and then concatenates the result [35]. This collection of max-pooling and different filters is called an "Inception module". The network consists of multiple consecutive Inception modules, in order to increase the performance of the network. To reduce the computational cost of having all these different filters in the modules, 1x1 convolutional filters are applied before the larger fil-

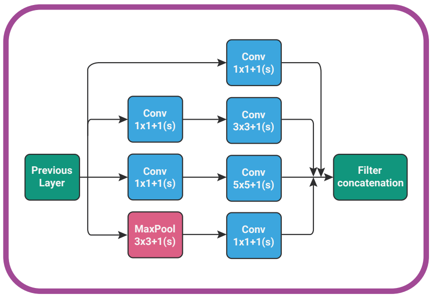


(a) Two residual blocks.

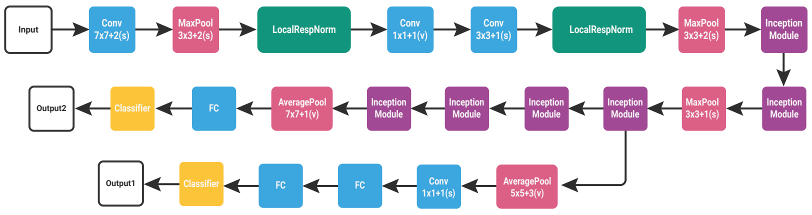


(b) The general structure of a ResNet network.

Figure 3.8: *The ResNet network.*



(a) An Inception module.



(b) The general structure of an Inception network.

Figure 3.9: *The Inception network.*

ters to reduce the computations needed [35]. The architecture also contains side branches with a final soft-max layer that predicts an output. These branches regularize the training by making sure that intermediate layers also have a reasonably good prediction capability. An example of an Inception network structure as well as an Inception module can be seen in Figure 3.9.

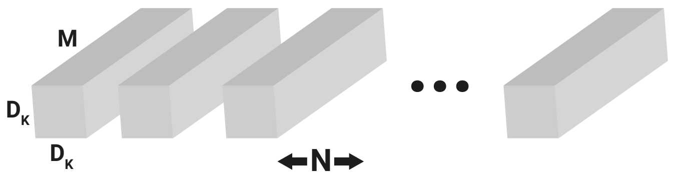


Figure 3.10: Standard convolution filters.

3.5.3 Mobilenet

Mobilenet is built on a streamlined architecture that uses depthwise separable convolutions [12]. It is a small, low-powered, low-latency model with the aim of meeting the constrains of several use cases. A depthwise separable convolution is a form of factorization that factorizes a standard convolution into a depthwise convolution and a pointwise 1x1 convolution. The depthwise convolution applies one filter to each input channel and the following pointwise convolution then combines the output from the previous depthwise convolution. This split into two layers, one layer for filtering and one layer for combining, drastically reduces the computation and model size compared to the standard convolution, where the filtering and combining of the outputs are performed in one step.

An output feature map for standard convolution assuming stride one and padding can be written as

$$G_{k,l,m} = \sum K_{i,j,m,n} \cdot F_{k+i-1,l+j-1,m} \,, \tag{3.13}$$

where F is an input feature map of size $D_F \times D_F \times M$ and G is the output feature map of size $D_F \times D_F \times N$. K is the convolution kernel of size $D_K \times D_K \times M \times N$.

The standard convolution filters can be seen in Figure 3.10 and gives the computation cost of

$$D_K^2 \cdot M \cdot N \cdot D_F^2 \,. \tag{3.14}$$

Depthwise convolution can be written as

$$\hat{G}_{k,l,m} = \sum \hat{K}_{i,j,m} \cdot F_{k+i-1,l+j-1,m} \,, \tag{3.15}$$

where $\hat{K}$ is the depthwise convolutional kernel of size $D_K \times D_K \times M$. The depthwise convolution filters can be seen in Figure 3.11. Here the mth filter in $\hat{K}$ is applied to the mth channel in F to create the mth channel in $\hat{G}$. Since the depthwise convolution only does filtering but no combination of the channels, the 1x1 convolution is applied to the output of the depthwise convolution to combine the channels and generate the new features. The computational cost of the depthwise convolution is

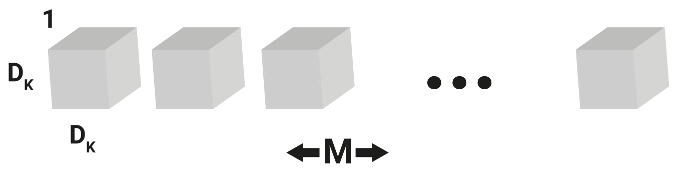


Figure 3.11: *Depthwise convolution filters.*

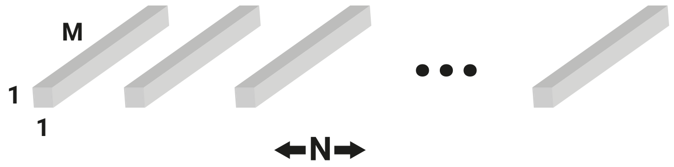


Figure 3.12: *Pointwise convolution filters.*

$$D_K^2 \cdot M \cdot D_F^2 \,, \tag{3.16}$$

and the computational cost of pointwise convoulution is

$$M \cdot N \cdot D_F^2 \,, \tag{3.17}$$

making the total sum

$$D_K^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2 \,, \tag{3.18}$$

for the separable depthwise convolution. The reduction of computational cost from depthwise separable convolutions compared to standard convolutions can then be expressed as

$$\frac{D_K^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2}{D_K^2 \cdot M \cdot N \cdot D_F^2} = \frac{1}{N} + \frac{1}{D_K^2} \,, \tag{3.19}$$

giving roughly 8 to 9 times less computational cost for a MobileNet network using 3X3 depthwise convolutions instead of standard convolutions, while only slightly affecting the accuracy.

The model also contains two global hyper parameters that can be controlled by the model builder to trade-off latency and accuracy, depending on the constraints for the use case. These hyper parameters are called the width multiplier α and the resolution multiplier ρ. With these hyper parameters, the already efficient MobileNet can get even smaller and faster. The width multiplier has the role of thinning the network uniformly at each layer by a factor α. This means that for a given multiplier α, the input size M will become αM and the output size N will be αN. The resolution multiplier ρ is applied to the input image, and the internal layers are implicitly reduced by the same multiplier.

With the width and resolution multipliers, the computational cost of the depth-wise convolutions for the core layers can be expressed as

$$D_K^2 \cdot \alpha M \cdot \rho D_F^2 + \alpha M \cdot \alpha N \cdot \rho D_F^2 \,, \tag{3.20}$$

where $\rho \in (0, 1]$, and $\alpha \in (0, 1]$ with common stops at 1, 0.75, 0.5 and 0.25. An α of 1 is the baseline model.

3.6 Object Detection

Convolutional neural networks can be used to detect objects in images. Such a network is trained to recognize a fixed set of classes and outputs the bounding box coordinates if any of these classes are visible in the image. Two of the most commonly used object detection algorithms are *region-based convolutional neural networks* (R-CNN) and *single shot detector* (SSD). These are described below.

R-CNN Ross Girshick, et al [14] created this algorithm where an input image is passed through three modules. In the first module, the algorithm uses a computer vision method called *selective search* to find potential objects in the image, called region proposals. The content in each region proposal is then passed through a CNN in the second module to extract features from each region. These features are finally classified as one of the classes using SVM, together with a score of how confident SVM is of the classification.

Since the first module might find many different potential objects in the image, the output of this pipeline can be cluttered with unwanted detections, for example parts of a human body when the desired output is only the full human body. To counteract this, *non-maximum supression* is used to remove regions that have a high intersection-over-union overlap with another region that has a higher confidence score. An illustration of non-maximum suppression can be seen in Figure 3.13.

Shaoqing Ren, et al [31] developed an improved version of this algorithm, called *Faster R-CNN*. In this faster and more accurate version, the region proposals are embedded in the neural network instead of using selective search. This avoids a multi-stage pipeline where data has to flow through three different modules, and instead it leverages the speed and accuracy of a neural network. The pipeline for Faster R-CNN can be seen in Figure 3.14. By taking the features extracted from the image using the CNN together with the selected region proposals, a region of interest (ROI) pool is applied and extracts the features that would correspond to relevant objects in the image.

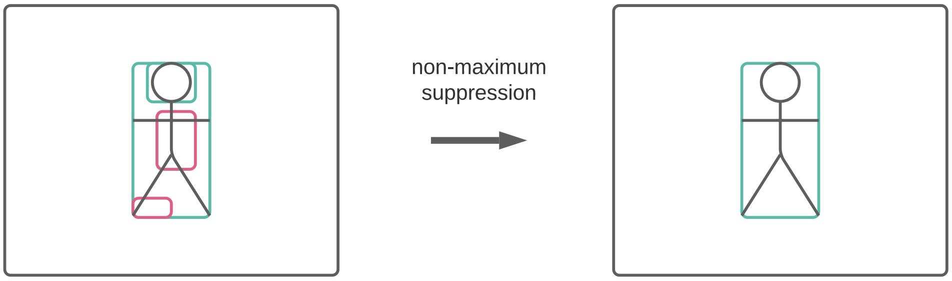


Figure 3.13: Some of the detected boxes are removed using non-max suppression.

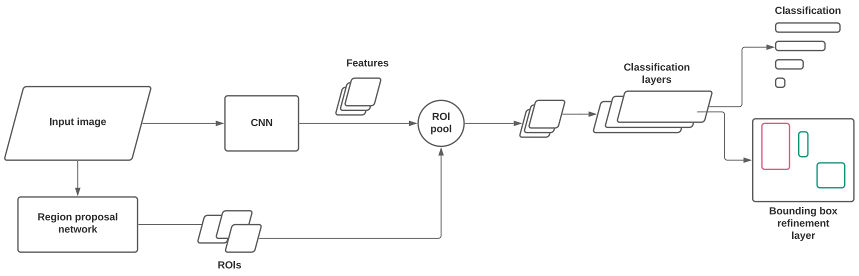


Figure 3.14: Pipeline for Faster R-CNN.

SSD This algorithm only needs one shot to pass through the architecture, compared to R-CNN that needs two shots (one for region proposal and another for classification). Thus, the SSD-algorithm is faster than R-CNN [24] and more suitable for real-time object detection.

In the training phase, the input to the network is not only the images but also ground-truth bounding boxes. Each image is divided using a grid where each grid cell is responsible for detecting objects in that specific region, meaning the class and location of the respective objects. This input is fed through a CNN that consists of a backbone model and a SSD head. The backbone model is usually some pre-trained image classification network, excluding fully connected layers, that extracts features. These features are then processed in the SSD head which is another set of convolutional layers that outputs bounding boxes with the associated classes, see Figure 3.15.

3.7 Transfer Learning

There are many publicly available pre-trained object detection networks that have been trained for a huge amount of steps on very large datasets. These models have learned everything from finding simple structures in the images, to complex features that belong to the specific classes it has been trained to detect. Trans-

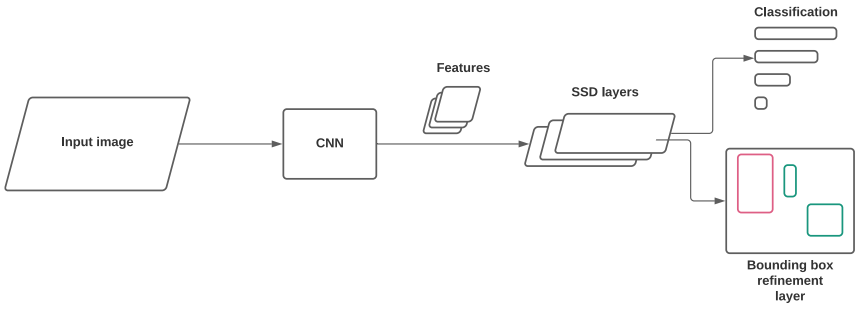


Figure 3.15: *Pipeline for SSD.*

fer learning is when one of these pre-trained model is used as a base, and then tweaked to be better suited for a different but similar task [39]. This reduces the training data needed since the model already has a lot of knowledge of detecting similar objects.

3.8 Edge Devices

Edge devices have, as previously mentioned, a lower amount of memory and computational resources than a traditional laptop or desktop and can therefore not be used in the same way when working with demanding machine learning algorithms. To solve this, alternative deep learning network architectures that are adapted to to these types of devices have been presented. One architecture that does this is MobileNet which is described in section 3.5.3. By using a fast model like MobileNet, even edge devices like microcontrollers can perform image classification in almost real-time. In [32], Voghoei et. al highlights several approaches that can be used to more efficiently be able to run the deep learning models on the edge device. Some of the techniques are, for example, *pruning* and *quantization*.

3.8.1 Pruning

Pruning is a process of removing unnecessary, less relevant or sensitive links from the network with the goal of creating a smaller and less complicated model. The effect of this is reduced computational cost and reduced memory and storage usage. It does this while still preserving, or at least only having a minor impact on, the performance [32].

3.8.2 Quantization

Quantization is a technique used to reduce the size of the model and increase the computation speed [4]. The concept of quantization is to reduce the precision of the numbers used to represent the parameters of a model. The default precision for a model is often 32-bit floating point, and by reducing that to for example 16-bit or even 8-bit integer representation, the model can be made much more efficient in terms of both model size and inference speed with a minimal loss in accuracy.

4 Method

The main part of the work has been to develop and evaluate algorithms for object detection. Since there is some uncertainty about the internet connectivity in the sanctuary, it may or may not be possible to send images to a remote server. This will determine whether the detection algorithm has to be run on the resource-constrained edge devices or if it can be run on a much more powerful computer in the cloud. To cover for all possible scenarios, detection models for both micro-controllers and desktop computers have been created.

4.1 Data Collection

Relevant data have been collected to use for training the neural networks. Most effort has been put into finding images of humans in environments that resemble the production site. Some of these have been collected from the internet, and some images have been captured at the actual production site. Furthermore, the network has also been trained with images of animals to help the network distinguish between humans and these animals. The animal images were given from the authors of [29] as they had already collected and annotated this data. When the system finally runs at the production site, it will be possible to collect more training data simultaneously.

The human images also had to be annotated with class names and bounding boxes for the networks to be able to learn from them. This was done manually using the tool *LabelImg* [2], shown in Figure 4.1.

Some of the networks used in the project are pre-trained on some of the largest public datasets available, the *common objects in context* (COCO) dataset [1] and the *ImageNet* dataset [10]. The COCO dataset has over 200,000 already annotated

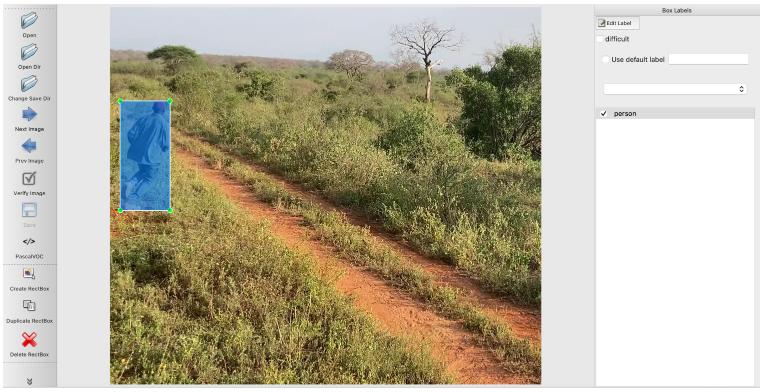


Figure 4.1: *Person has been annotated with a bounding box.*

images with bounding boxes spread over 90 classes, including a class *person*. The ImageNet dataset has over 1,000,000 annotated images and 1000 classes, but not any class with *person*.

A separate dataset was created to use for pure classification without localization. These images only needed to be labelled with the class name, no bounding boxes. This dataset was generated by extracting all images from the COCO dataset where a human is present and covers at least 5% of the total image area. All other images that did not match this criterion were labelled as background. This dataset is commonly called the *Visual Wake Words* dataset, originally invented by Chowdhery et. al [8].

4.2 Motion Detection

A motion detection algorithm has been implemented to avoid running the model when no foreground objects are present in front of the camera. This reduces the battery usage since it is computationally expensive to run model inference. It is also a relatively easy task to implement motion detection in this project since the cameras will always be stationary.

The first step in the detection algorithm is to downsample the image. This reduces the effect of noise and also makes the detection step more efficient because of the reduced resolution. The downsampling is done by dividing the image into smaller blocks and computing the average pixel value for all pixels inside each block. These averages build up the new image. The smaller the block size is, the more sensitive the algorithm is to motion from small objects.

The next step is where the motion detection is run. This is done by first creating a foreground mask. Each pixel value in the downsampled image is compared to the same pixel value from the previous image, and if the difference is larger than a threshold τ_d, that pixel is marked as foreground. If a pixel has not reached

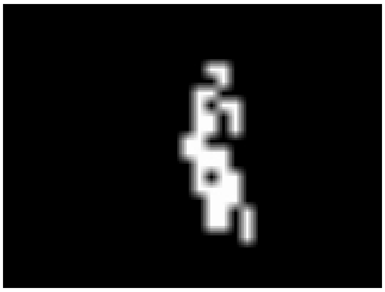


Figure 4.2: *Foreground mask of a walking person.*

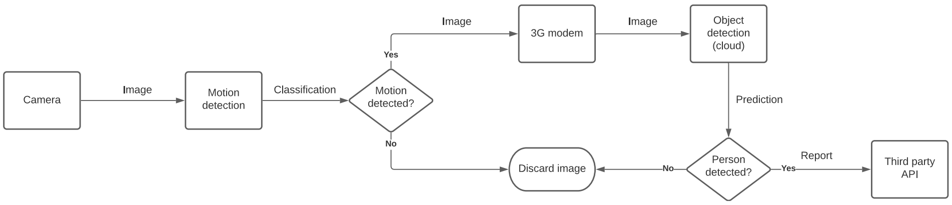


Figure 4.3: *Pipeline for the first alternative that requires 3G modem.*

the threshold for N number of frames, it is marked as background. A larger N creates fewer holes in the foreground mask but introduces more noise. Figure 4.2 shows a generated foreground mask of a walking person where the white pixels are foreground and the black pixels are background. In each frame, motion is detected if the percentage of foreground pixels is above a set threshold τ_m.

4.3 Preparing for the Savanna Conditions

Due to the uncertainties of the internet connection at the production site, two different pipeline alternatives have been tested, described in Section 4.3.1 and 4.3.2. Both of these alternatives will use multiple camera-attached microcontrollers spread over the savanna, utilizing a simple motion detection algorithm to determine which images to process. Furthermore, both alternatives will go into sleep mode at night to save battery since no camera with the ability to take night photos will be used.

4.3.1 Pipeline 1

In this pipeline, when something moves in front of the camera, the motion detection described in 4.2 triggers and the microcontroller sends the image to a backend server. This server will then run the object detection algorithm on the image to determine if the moving object is a human. If the image contains at least one human, that image is uploaded to the dashboard, including bounding boxes around all humans. This pipeline can be seen in Figure 4.3.

Training the Network

The *Object Detection API* [3] from Tensorflow has been used to train the different object detection models, i.e., SSD and Faster R-CNN. All of them are pre-trained on the COCO dataset and then fine-tuned using transfer learning with the data described in Section 4.1. The training has been done on *Google Colaboratory* which is an environment to run Python code on GPU-supported machines for free.

The transfer learning has been done by changing the output of the network to output fewer classes than the 90 that are part of the COCO dataset. Both a 1-class network (human) and a 8-class network (human, elephant, giraffe, buffalo, leopard, lion, rhinoceros, zebra) has been trained and evaluated. The idea of the 8-class network is to make it easier for the network to differentiate between humans and common animals in the wild.

Communication Between Edge Device and Server

When the edge device has detected motion, it uploads images to a FTP server where the location of the device is encoded in the filename. A separate server continuously checks to see if any new files have been uploaded.

Another method that was tested was to use a *WebSocket* connection between the edge device and server to send the images instantly. This was suitable during development since the server could communicate back to the edge device to turn on a LED light on detection. However, this is not used in production because it is easier to perform updates to the server when using the FTP solution.

Running Inference

When the server has found new images on the FTP server, it runs inference with the trained model using the Tensorflow library. This returns a set of detected objects where each detection has a confidence score that represents how certain the model is of the detection. Even though it is using a full fledged object detection algorithm, the location of detected objects is not considered when determining if a report should be sent. Thus, this is a classification task where an image is considered to contain a person if one or more of the detected objects have a person confidence score larger than τ_P.

4.3.2 Pipeline 2

This pipeline focuses on reducing the bandwidth used, and therefore does all the image classification on the edge device instead of on the server. It is still assumed that a limited 3G connection is available to be able to send low resolution images to the dashboard in the event of human detection. An overview of the pipeline can be seen in Figure 4.4.

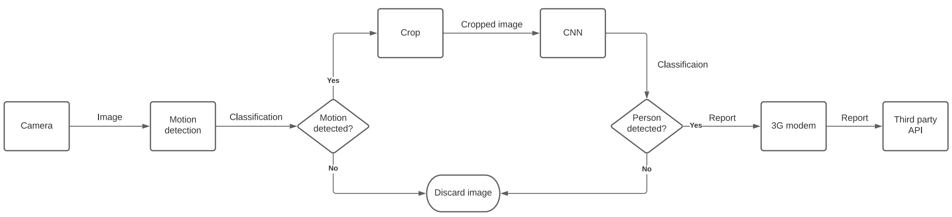


Figure 4.4: *Pipeline for the second alternative.*

Training the Network

Since microcontrollers naturally have a small amount of RAM, they do not have the computational capacity to drive a full-fledged object detection algorithm such as Faster R-CNN. Instead, only a classification model has been trained since it does not require to localize the position of the detected object. In this thesis, MobileNet was chosen as network architecture for the classifier since it is small (a few hundred kilobytes) but still has relatively good performance on human detection [20]. The network has been trained from scratch on the Visual Wake Words Dataset, once again utilizing Google Colaboratory. Furthermore, an attempt was also made to use a network that was pre-trained on the ImageNet dataset and fine-tune it on the Visual Wake Words Dataset using transfer learning.

Extracting Important Areas

Since this pipeline only uses a classification model to detect objects, it is important to feed the model with images where the foreground object fills a large portion of the pixels. However, as this project uses camera traps, the foreground object may be far away from the camera. To counteract this, an algorithm has been developed to extract only the region in the image where the foreground object is located.

The algorithm uses statistics from the foreground mask that is created during the motion detection step, described in Section 4.2. In the first step, the centroid of the foreground mask is found by calculating the mean position in x, m_x and mean position in y, m_y.

Next, the mean deviation from the centroid in x (v_x) and y (v_y) is found by summing the distances between each foreground pixel to the centroid, and then dividing by the total number of foreground pixels N_F

$$\begin{aligned} v_x &= \frac{1}{N_F} \sum_{x \in F} |x - m_x| , \\ v_y &= \frac{1}{N_F} \sum_{y \in F} |y - m_y| , \end{aligned} \tag{4.1}$$

where F is the foreground mask. v_x and v_y indicate how wide the foreground object is in each dimension, while allowing for some noise in the foreground mask. Since the model expects a square input, the largest value of v_x and v_y is chosen as the width for the extracted area. This area is then cropped from the original image, using $(m_x, m_y - \delta)$ as the center point, where δ is a shift in the vertical direction so that more of the upper body is centered if the foreground object is a human.

Running on Multiple Cores

To get good extracted areas, the motion detection algorithm described in Section 4.2 has to run at a decent frame rate. Otherwise, objects will have time to move far between frames, which creates an incorrect foreground mask. However, the model inference takes around one second which halts the motion detection algorithm, making it too slow. To work around this, the motion detection and the inference run on separate cores. In the code, semaphores are used to ensure that inference does not happen at the same time as the cropped image is written to.

4.4 Camera Trap Design

For the system to have any effect, it is important that the camera-equipped microcontrollers are not visually noticeable in the sanctuary. Therefore, a custom made case has been 3D-printed that can fit the microcontrollers, including a battery to power the device. Furthermore, a solar panel is attached to the battery to charge it during sunlight. The whole camera trap can be seen in Figure 4.5 and Figure 4.6.

At the moment, the system is not usable during night because the camera is not able to capture well in darkness. Therefore, the microcontrollers are put into deep sleep to save battery. For this to be possible, the microcontrollers must retrieve the time and date during startup, which is done via *simple network time protocol* (SNTP).

4.5 Report Creation

The final step in the system is to create the reports that get communicated to the park rangers. This is only done if the system has detected a human x times within a certain time frame. The reason for this is to eliminate sporadic false positives and avoid generating reports in such cases. Furthermore, after the system has generated a report, it cannot generate a new one for y amount of minutes to avoid reporting the same event twice.

When these conditions are fulfilled, an image is uploaded to a third party FTP server, separate from the FTP server in Section 4.3.1, where the third party handles the communication to the park rangers. In pipeline 1, this is a low resolution image of the detected human. In pipeline 2, it is a higher resolution image of the detected human, including a bounding box around the person.

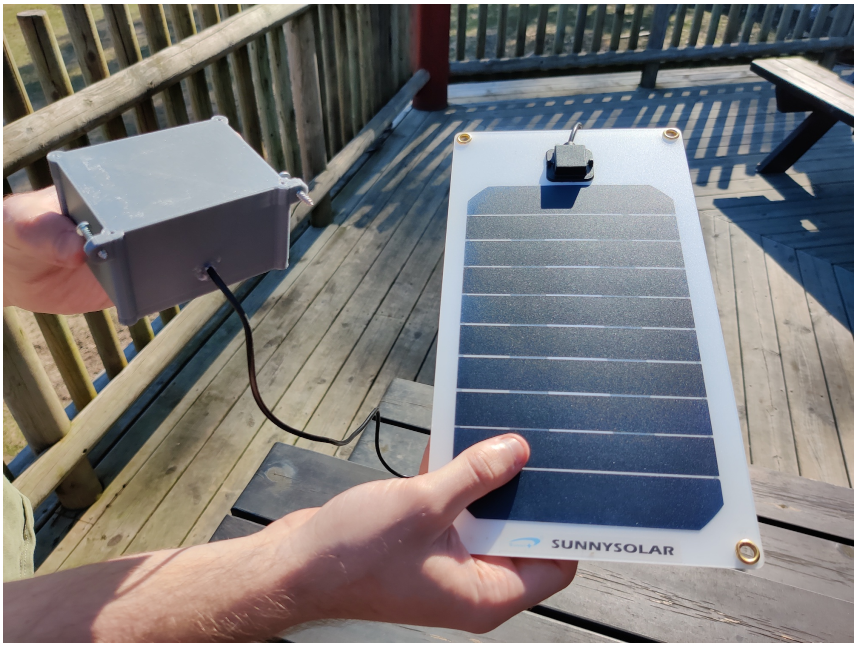


Figure 4.5: *The camera trap consisting of a solar panel and a custom made case with an ESP32-cam, a battery and a solar power manager.*

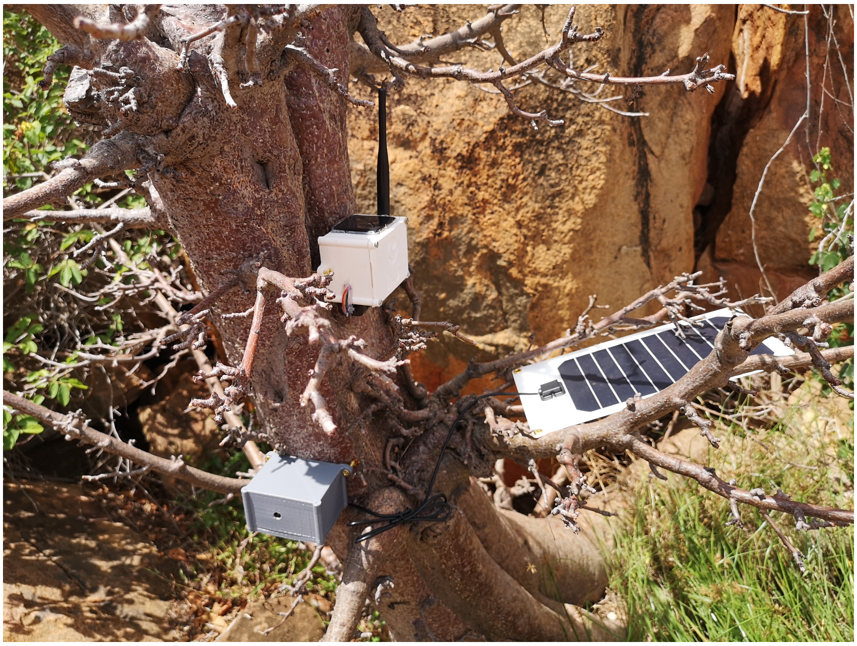


Figure 4.6: *The camera trap installed at the production site.*

4.6 Evaluation

The system has been evaluated to determine which model to run in production. To be able to compare between the two pipelines, both of them have been evaluated as a classification problem, even though pipeline 2 is actually using an object detection algorithm.

Two different metrics have been used during the evaluation. In the first case, the models are evaluated using standard metrics for image classification, which are *accuracy*, *precision* and *recall*

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

$$precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$recall = \frac{TP}{TP + FN} \tag{4.4}$$

where TP, TN, FP and FN are *True Positives*, *True Negatives*, *False Positives* and *False Negatives* respectively. This is evaluated over a set of test images where each image is labelled as either human or not.

To get a combination of the precision and recall, the *F1 score* was used. The F1 score is defined as the harmonic mean of the two, and the formula is expressed as

$$\text{F1 score} = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{4.5}$$

Precision and recall scale between 0 and 1, meaning that an optimal performing model will have precision and recall both being 1. Plugging these values into (4.5) gives $2 * \frac{1*1}{1+1} = 1$. In other words, the F1 score is also ranged between 0 and 1 where the best possible score is 1.

In the second case, the full pipelines are evaluated on image sequences, where each sequence contains a human walking past the camera. The evaluation counts how many of the sequences each pipeline generated a report.

5 Results

In this chapter, the results from both pipeline implementations are presented. This includes quantitative results such as classification accuracy, as well as qualitative results from the model output. The models that are trained in pipeline 1 are denoted as *high-end models* while the models that are trained in pipeline 2 are denoted as *low-end models*.

5.1 Model Performance

To compare different pre-trained models, each model has been applied to test data and evaluated using a couple of classification metrics. Each test image is classified as either human or not and the output is then compared to the ground truth. The results are presented below.

5.1.1 Video Sequence

The pre-trained models have been evaluated on a video sequence containing a person walking in and past the camera view. The video has been captured at the production site, similar to the final setup. To give a fair comparison between each evaluation, the video frame rate is set to a fixed value, generating a total of 201 frames. Note that for the low-end model evaluation, this process includes the ROI extraction step. In other words, the images that are sent to the low-end model are cropped using the method described in Section 4.3.2.

The accuracy, precision and recall for the video sequence evaluation on high-end are presented in Table 5.1. The same metrics for the low-end model are presented in Table 5.2. The F1 scores for all models are presented in Figure 5.1, where blue bars are high-end models and green bars are low-end models. All

Table 5.1: *Classification metrics for high-end models on video sequence.*

Model	Accuracy	Precision	Recall
SSD ResNet-50	0,910	1,000	0,822
Faster R-CNN ResNet-50	0,968	1,000	0,964
Faster R-CNN Inception	0,973	1,000	0,970

Table 5.2: *Classification metrics for low-end models on video sequence.*

Model	Accuracy	Precision	Recall
MobileNetV1-0_25	0,353	1,000	0,274
MobileNetV2-0_35	0,430	1,000	0,360

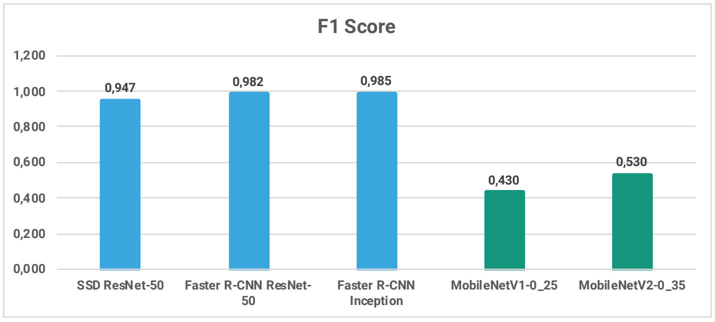


Figure 5.1: *F1 scores for all models on video sequence.*

metrics are ranged between 0 and 1. For example, an accuracy of 1 means 100% accuracy.

5.1.2 Evaluation on Static Images

The pre-trained models have also been evaluated on a test set containing:

- 102 images of humans, taken at the production site.
- 102 images of animals in the wild.

Similar to the previous section, the accuracy, precision and recall for the high-end test set evaluation can be seen in Table 5.3.

Since the static images are not a sequence of continuous frames, and the images come without any correlation to the previous image, the ROI extraction step can not be performed. To get a fair comparison between the high-end and low-end pipelines, interesting areas were cropped out manually to get a similar input

Table 5.3: *Classification metrics for high-end models on the static images.*

Model	Accuracy	Precision	Recall
SSD ResNet-50	0,936	1,000	0,873
Faster R-CNN ResNet-50	0,971	1,000	0,941
Faster R-CNN Inception	0,975	1,000	0,951

Table 5.4: *Classification metrics for low-end models on the static images.*

Model	Accuracy	Precision	Recall
MobileNetV1-0_25	0,637	0,912	0,304
MobileNetV2-0_35	0,686	0,932	0,402

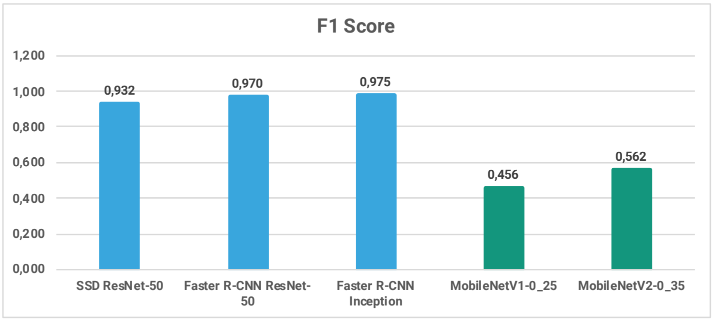


Figure 5.2: *F1 scores for all models on test set.*

image as if there would have been a sequence of frames with moving objects. An evaluation of both a pre-trained MobileNetV1, and a self-trained MobileNetV2 can be seen in Figure 5.4. Both models have been trained on the visual wake words dataset from COCO. The F1 scores for all models are presented in Figure 5.2.

5.2 Effect of Transfer Learning

The usefulness of transfer learning has been investigated, which was done by comparing the performance for the high-end model trained with and without transfer learning. Two different variants have been tested. In the first case, the network only has one potential output class, which is *human*. In the other case, the network has eight potential output classes, namely *human*, *elephant*, *giraffe*, *buffalo*, *leopard*, *lion*, *rhinoceros* and *zebra*.

ResNet-50 has been used as the base network and the transfer learning has been

Table 5.5: The difference in performance between models when using transfer learning.

Model	TP	TN	FP	FN
Pre-trained	83	102	0	19
1 class	87	95	7	15
8 classes	98	102	0	4

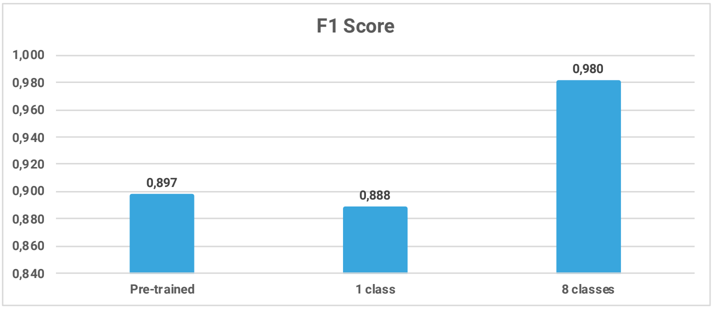


Figure 5.3: The difference in F1 score when using transfer learning.

done using the manually collected data that was described in Section 4.1. All evaluations have been run on the static images described in Section 5.1.2. The number of true positives, true negatives, false negatives and false positives for the base model and for the transfer learned models are presented in Table 5.5, and the F1 scores in Figure 5.3.

5.3 Pipeline Performance

The ultimate goal of the system is to send one and only one event for each time one or more persons enter the camera view. Therefore, an additional evaluation process has been done where 20 video sequences have been created. Each video sequence contains a person walking past the camera. The system has done correctly if it sends one event for each video sequence. In this evaluation, both pipelines have been compared and the number of correctly sent events are counted. Out of the 20 sequences, pipeline 1 managed to detect an event for **19** of them, while pipeline 2 detected **12**.

5.4 RoI Extraction

The performance of the low-end model is highly dependant on the ROI extraction. In the best case scenario, each camera frame is cropped so that only the moving object is sent to the model for classification. As described previously, the crops are done based on the foreground mask from the background subtraction algo-

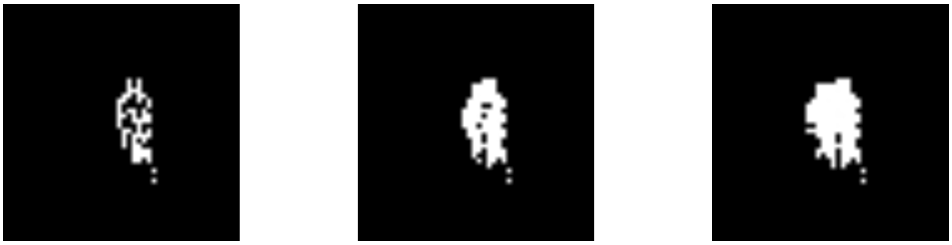

(a) N = 1 *(b)* N = 3 *(c)* N = 5

Figure 5.4: *The difference the parameter N makes for the background subtraction.*

Table 5.6: *RoI extraction performance on the low-end pipeline.*

RoI extraction	Accuracy	Precision	Recall
Yes	0,353	1,000	0,274
No	0,181	1,000	0,081

rithm. This algorithm has a few different parameters that affect the output. One of which is the parameter N, representing the number of frames it takes to mark a pixel as background (see Section 4.2). Figure 5.4 shows the difference between different values of N.

As can be seen, a larger value of N gives fewer holes in the foreground mask. However, this amplifies the effect of noise and also makes the foreground mask more smeared if the moving object moves across the scene. From qualitative tests, it was decided that $N = 2$ gave the best results in this thesis. The other parameters used for ROI extraction (see Section 4.2) in this thesis are presented below:

- $\tau_d = 0.200$
- $\tau_m = 0.003$

Furthermore, an evaluation was made where the performance of the low-end model with and without ROI extraction was compared. This evaluation was done on the video sequence described in Section 5.1.1. The accuracy, precision and recall are presented in Table 5.6. The F1 scores are shown in Figure 5.5.

As is shown in the results, the ROI extraction has high of impact on the performance of the model. It is clear to see why when inspecting the images that are sent to the model with and without ROI extraction, see Figure 5.6. The results show that it is a simpler task for the model to classify the foreground object when the image only contains the object without unnecessary background.

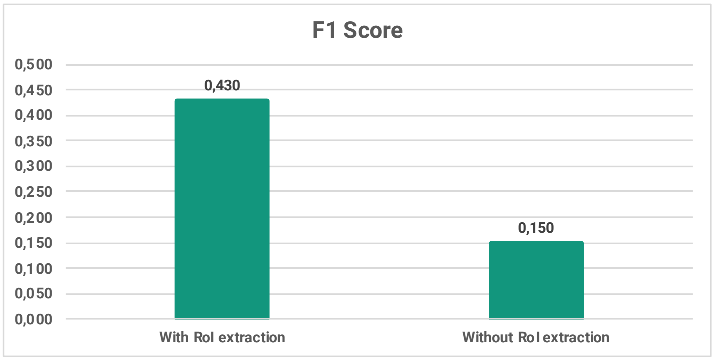
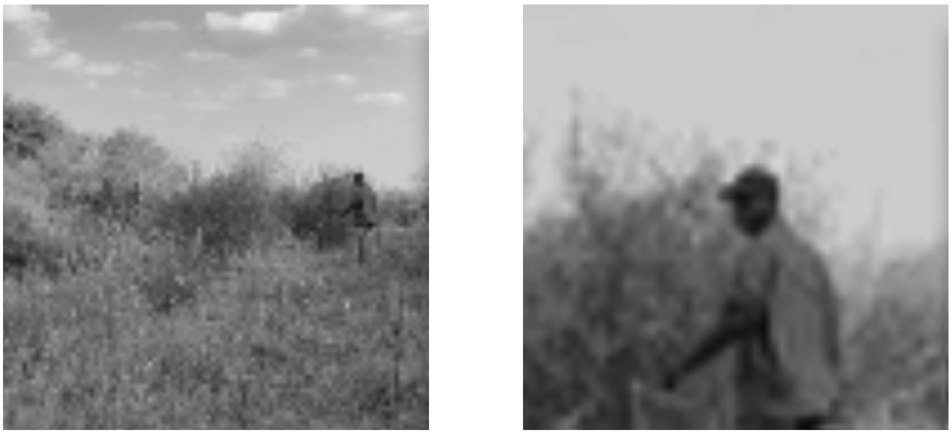


Figure 5.5: *The difference in F1 score when using RoI extraction on the low-end pipeline.*



(a) *Without RoI extraction* ***(b)*** *With RoI extraction*

Figure 5.6: *The difference of using RoI extraction.*

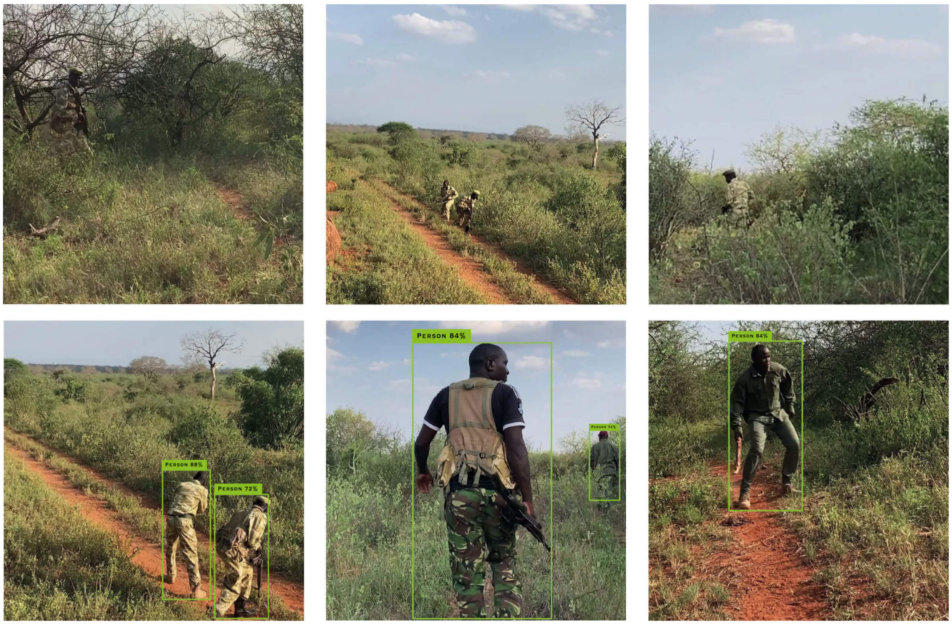


Figure 5.7: *Some examples of successful and unsuccessful human detection for the high-end pipeline. Upper row: unsuccessful human detection. Lower row: successful human detection.*

5.5 Qualitative Results

In Figure 5.7 and 5.8, some examples of successful and unsuccessful human detection for both pipelines can be seen. The images are captured in the production environment with a mobile phone camera and converted to the correct resolution and color for the corresponding pipeline.

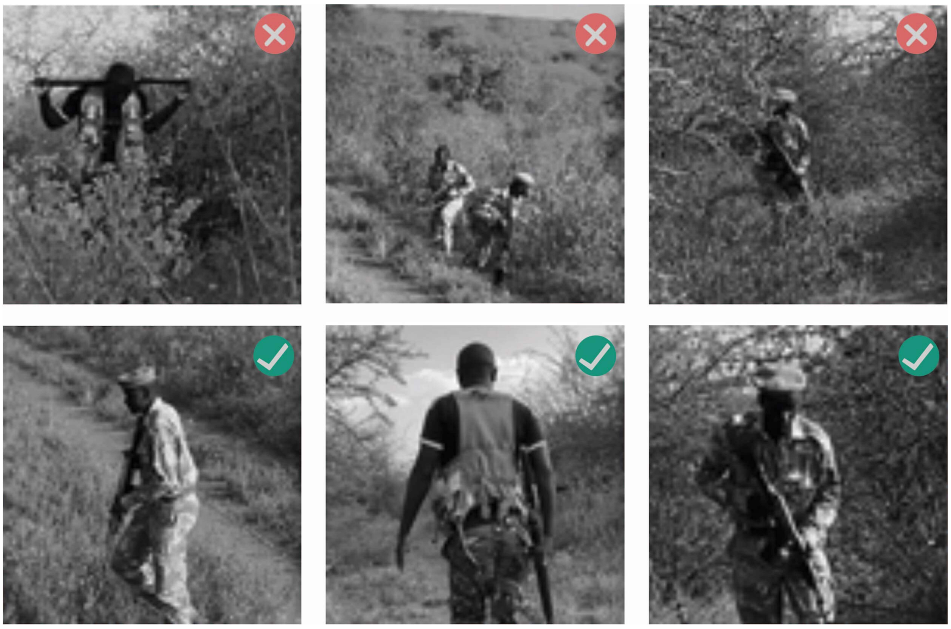


__Figure 5.8:__ Some examples of successful and unsuccessful human classification for the low-end pipeline. Upper row: unsuccessful human classification. Lower row: successful human classification.

6 Discussion

In this chapter, the thesis is discussed with regards to the method used and the results that were obtained. Furthermore, the chapter contains a discussion on ethical and societal aspects of the thesis.

6.1 Method

The chosen method was influenced by uncertainties of internet connection in the sanctuary. Due to this, two different pipelines were implemented and compared, which gave some interesting insights from a research and development perspective. This section includes a discussion on these insights.

6.1.1 Developing from Afar

One of the challenges in this thesis was to develop a system without having physical access to the production site. Though there are many images of humans in the wild available on the internet, they do not match the exact environment where the system will be set up. This makes it harder to train a network for the production environment. Furthermore, it becomes tricky to test and evaluate the system before deploying it. As a workaround, the system was set up at Kolmården Djurpark for analysis and evaluation. Even though Kolmården Djurpark has a specific area for animals of the savanna, it might not have conditions that are sufficiently similar to the production site. This can be improved by collecting training data after deployment and iteratively re-training the model. This is easy for the high-end model since it runs remotely. However, the low-end model could only be upgraded by having physical access to the device.

Since the microcontroller is not connected to the computer to be monitored while in the produciton environment, no logs are recorded if the system crashes for any reason. It can therefore be difficult to troubleshoot the reason why no images are uploaded from the camera trap. Either it can depend on internal or external problems, or just that nothing has happened in front of the camera. To solve this, logs could be saved to the SD card. Even if this does not give answers right away since the SD card is located elsewhere, it can still be useful to have when finally physically accessing the camera trap at a later time.

Another improvement that could be implemented would be to send regular status updates to the FTP server as a text file containing for example current battery life and wifi strength. This would, except for giving valuable numerical information, also act as an assurance that the system is still currently alive and running.

6.1.2 Detecting Motion

It is not trivial to implement a robust motion detector using image processing. Since the algorithm can only operate on changes in pixel values, this is affected by noise in the camera output. This is circumvented by downscaling (which effectively works as a low-pass filter) and setting high enough thresholds so that noise is not mistaken as motion. However, the amount of noise from the camera is heavily affected by external factors such as the amount of light in the scene. Therefore, it is hard to set the parameters so that the algorithm works robustly independently on the time of the day, and weather conditions. Furthermore, the different parameters are correlated, which makes it even harder to find good parameters. The solution was to set the thresholds so high that the algorithm works in the most noisy conditions. However, this affects the results because small movements may not be detected since it is considered as noise by the algorithm.

The motion detection algorithm could probably be improved by implementing an adaptive background subtraction algorithm, for example, by using *gaussian mixture models*. The question that arises is whether this makes the full pipeline too slow on a microcontroller.

6.1.3 Data Collection and Network Structure

A time consuming part of deep learning is to find and annotate train and test data. The model improves with the amount of quality train data collected but there is a trade-off with how much time to spend on this part. It was decided to collect enough data to make the transfer learned model have better performance than the pre-trained model. This could, however, be improved further by collecting more data at a later stage.

When training a network, the set of output classes has to be decided. One possible configuration is only to have two classes which are *human* and *background*. This could make the model focus on the primary task, which is to detect humans. However, objects that look similar to humans could be falsely classified as a human. This can be improved by introducing more output classes with obvious objects that may be falsely classified. In this thesis, it was decided to try a configuration where animals from the wild was put into the output classes. This can be a big decision to make since it requires lots of more training data. In this thesis, the additional animal data was already available which made it easy to make this decision, but it could be a huge task in other projects to collect the data.

6.1.4 Managing Multiple Pipelines

Having multiple pipelines to manage, both in the development phase and the production environment, is not the ideal solution. Apart from getting confused of which device is running on which pipeline, and the obvious time increase of the implementation, there will be more time spent training the models. Either both pipelines have to be trained on different computers and occupying more resources, or in sequence on the same computer and therefore taking double the time. However, when not knowing how much allowed bandwidth there will be and the speed of the internet, developing two different pipelines unlocks more possible alternatives to be chosen from when finally deploying the system. Depending on the internet connection, the more suitable pipeline will be chosen, and the other one will be a backup if the conditions would change in the future. Implementing two pipelines not only makes it possible to choose the most suitable one for the situation, but also gives an interesting comparison of object detection and classification on different types of hardware.

6.1.5 Communication

Before implementing the FTP solution, the system sent all images via WebSockets to the server. This had some advantages since WebSockets is a duplex communication protocol. For example, the server could communicate back to the microcontroller the classification results, which could trigger a LED blink. This was useful during development to be able to see directly if the camera trap triggered or not. However, the LED blink is not suitable during production, and hence the two-way communication was not needed anymore. The decision was then to use FTP instead, which has other advantages. The most important advantage is that the microcontroller and server do not need to maintain a continuous connection. This makes it more stable to do upgrades on the server because the microcontroller can keep uploading images to the FTP while the server is upgraded. In the case of WebSockets, this would mean that the system would miss all the camera frames during the upgrading process.

6.2 Results

The results show that the system could be used as a camera trap for humans. These results are discussed further in this section.

6.2.1 Model Performance

For obvious reasons, the high-end models give much better performance than the low-end models. The high-end models can correctly classify most images of humans except cases where the human is far away, has a low contrast to the background or is heavily occluded. The results show that the transfer learned model with one additional output class does not miss many humans. However, it tends to sometimes falsely classify animals as humans, and it therefore gets a lower F1 score than the pre-trained model. The transfer learned model with eight additional output classes also gives a high amount of correctly classified humans, and at the same reduces the amount of false detections. This shows that transfer learning with classes that are somewhat similar to the desired class is prefered to get a more robust detection algorithm.

The low-end models could work in cases where the input data is fairly simple. For example, if the camera is set up so that it captures a straight trail and a human is walking towards the camera. For more complex scenarios for example, if the human is occluded, the low-end models often miss the detection. However, as seen in Section 5.3, a low-end model correctly sent an event for 12 out of the 20 test video sequences. These video sequences were purposely created so that the classification task is hard, i.e., by choosing videos where the humans are far away from the camera and often occluded. Taking this into consideration, the low-end model completes its task impressively well and should be able to give good results in a production setting.

6.2.2 Durability of the Camera Trap

Placing a camera trap in the middle of the savanna requires several areas to be covered regarding durability. Some of those are the physical durability of the camera trap, energy usage and limited internet connection.

The hardware that can be seen in Figure 4.5 has to withstand everything from broiling hot sun to whipping rain over a longer period of time. The case was designed with this in mind, reducing the amount of holes to the minimum and placing the lid on the bottom to make it as water resistant as possible. Hopefully, this solution will handle all the conditions at the savanna and not let external factors prevent the trap from running.

As mentioned before, to increase the battery life and keep the camera trap alive for a long period of time, both pipelines go into sleep mode at night. This reduces the battery consumption of the camera trap by a huge amount, making it less likely to drain the battery when no sun can charge the solar panel. Without entering sleep mode, the camera trap would probably start with a rather empty battery every morning when the sun first comes up and starts charging it.

One specific power intensive operation on the low-end pipeline is the model inference step. It is no surprise that running inference on a deep learning classification model uses quite a bit of energy, and has to be performed as rarely as possible when battery life is an important factor. The preceding motion detection step therefore not only acts as a performance increasing procedure by allowing for ROI extraction, but also as an energy saving step in the pipeline. Furthermore, instead of having the WiFi on all the time, it could be turned off if it has not been used for a certain amount of time to save even more power.

When there is a hard limitation on the internet in terms of either the amount of data that can be sent, or the speed of the internet, the more durable solution is the low-end pipeline. When using the low-end pipeline, a noticeable reduction in internet usage is that all images that do not trigger motion detection are discarded, heavily reducing the bandwidth used. The second reduction in internet usage is the low-resolution images sent only once every time an event occurs, reducing the amount of data needed to be transmitted for each image.

6.2.3 Importance of RoI Extraction

One of the main discoveries in this thesis was that a ROI extractor could be implemented to increase the performance of the low-end model. This was possible since the cameras are always static, allowing for image based motion detection. It was quite time consuming to implement this on a microcontroller, but the rewards were well worth it, as was proved with a F1 score almost three times as good on the test data. It was shown during testing that the low-end models could not handle cases where the human was far away from the camera if the ROI extraction was deactivated.

However, this algorithm can only zoom in on one specific area. If there are multiple moving objects in the scene, and they are spread across the camera view, the algorithm will keep the image zoomed out because the spread of changing pixels are too wide. This could be managed by a more sophisticated algorithm where it can detect local changes within the image, extract multiple regions and send each region separately to the model.

6.3 Work in a Wider Context

Since the camera traps are placed outside in the wild without anyone monitoring them, integrity is an issue that can not be overlooked. The park rangers that patrol the area for example have a chance of, unwittingly or wittingly, being caught on an image. However, all park rangers need to explicitly consent to the possibility of being included in images captured by the camera traps in advance. The problem arises when there are people present who have not given consent. Even though no people should be in the park except the park rangers in normal circumstances, both people being there illegally to hunt animals, and legally to manage something in the park have a chance of being captured without giving consent.

By sending the images over the internet, there is a chance for them to be intercepted and end up in the wrong hands. While this can be an integrity issue, another eminent problem when sending images that include animals is that the intercepter can get an indication of where the animals are in the park. With this information, it can be easier for the poacher to locate and harm the animals, resulting in an opposite outcome than what was originally desired. This puts a lot of importance on security for the data transfer, which has not been prioritized in this thesis due to time constraints.

7 Conclusion

The aim of building a system that could be used for detecting poachers has been fulfilled. Results show that deep learning methods are suitable for this task and can be run even on a microcontroller. Furthermore, a ROI extractor was implemented which proved to increase the performance of the system greatly.

The research questions that were stated at the beginning of this report are answered below.

7.1 Research Questions

1. **How can the performance of an image classifier be increased when the target object covers a small area of the image?**
   While an object detection model can identify small objects in the image, an image classification model needs that the foreground object covers a large portion of the image. To solve this requirement, an algorithm was implemented that extracts the area of the image where there is movement. As a result, this removes a large portion of the image where there is only background, making it simpler for the model to focus on the foreground object. This method significantly increased the accuracy of a deep learning model. However, this is only possible if the camera is static so that the image based motion detector can work as expected.
2. **To what extent can a microcontroller be used to perform image classification using deep learning?**
   It was possible to run a low-end MobileNet model on the microcontroller trained to classify an image as either human or not. Compared to a high-end model, the accuracy was less than half of a high-end model run on a

powerful computer. However, when running the low-end model on a video sequence, it could often find a human in at least a few frames. This is enough to detect human presence in a video sequence which can be used to alert the park rangers. However, this requires that the camera is set up so that the visible humans are not too occluded or have too low contrast against the background.

3. **How to avoid that objects that look similar to humans get misclassified as humans?**
   While a model with only two output classes (human, background) can suffice to detect humans in images, it is quite sensitive to objects with visual features close to humans. This was improved by integrating more output classes in the model, in this case other animals that live on the savanna. By doing this, the number of false positives that occurred during the evaluation was decreased.

7.2 Future Work

The majority of the work done for this thesis was to implement and test the full pipeline for the microcontroller. Even though the system works in the current state, some improvements could be tried in future work. A simple improvement would be to collect and annotate more training data, which would increase the performance of the models. This is a time consuming task which is why the current training data is limited. The models could also be improved further by doing careful parameter search during training to fine tune the hyperparameters optimally. Another interesting attempt would be to integrate more humans into the training data by using public human datasets available on the internet. Even though it is hard to find datasets of humans in a wildlife setting, it could still improve human classification accuracy.

The ROI extractor was implemented to be as efficient as possible not to sacrifice the frame rate. It would be interesting to implement a more sophisticated algorithm that could be less sensitive to noise and be able to extract multiple regions in an image, for example if there are multiple moving objects.

Another significant extension of this work would be to handle nighttime images. Currently, the microcontroller is put to sleep during night to save battery since the camera can not capture in the dark, but in the future the cameras should be running at all times. The most obvious solution would be to use a night vision camera, but it could also be solved by supplementing the microcontroller with some other sensor (e.g. PIR or Lidar) that could trigger a camera flash on movement.

Bibliography

[1] Coco - common objects in context. https://cocodataset.org/. Accessed: 2021-02-02.

[2] Labelimg. https://github.com/tzutalin/labelImg. Accessed: 2021-05-03.

[3] Tensorflow objet detection api. https://github.com/tensorflow/models/tree/master/research/object_detection. Accessed: 2021-05-10.

[4] Model optimization. https://www.tensorflow.org/lite/performance/model_optimization. Accessed: 2021-04-23.

[5] Neena Aloysius and M. Geetha. A review on deep convolutional neural networks. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 0588–0592, 2017. doi: 10.1109/ICCSP.2017.8286426.

[6] Yoshua Bengio. Learning deep architectures for ai. 2009.

[7] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In Sorelle A. Friedler and Christo Wilson, editors, *Conference on Fairness, Accountability and Transparency, FAT 2018, 23-24 February 2018, New York, NY, USA*, volume 81 of *Proceedings of Machine Learning Research*, pages 77–91. PMLR, 2018. URL http://proceedings.mlr.press/v81/buolamwini18a.html.

[8] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. Visual wake words dataset, 2019.

[9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005. doi: 10.1109/CVPR.2005.177.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[11] Nicholas D. Lane Edgar Liberis. Neural networks on microcontrollers: saving memory at inference via operator reordering. 03 2020.

[12] Andrew G. Howard et. al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017.

[13] Seyed Yahya Nikouei et. al. Real-time human detection as an edge service enabled by a lightweight cnn. 2018.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition, Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 580 – 587, 2014.

[15] Xavier Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 9780262035613.

[18] K. et al. He. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision (ICCV 2015)*, 1502, 02 2015. doi: 10.1109/ICCV.2015.123.

[20] Chloe Eunhyang Kim, Mahdi Maktab Dar Oghaz, Jiri Fajtl, Vasileios Argyriou, and Paolo Remagnino. A comparison of embedded deep learning methods for person detection, 2019.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[22] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[23] Yann Lecun, Patrick Haffner, and Y. Bengio. Object recognition with gradient-based learning. 08 2000.

[24] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. 2015.

[25] Magnus Malmström. Uncertainties in neural networks : A system identification approach, 2021. ISSN 0280-7971.

[26] Chao Mi, Xin He, Haiwei Liu, Youfang Huang, and Weijian Mi. Research on a fast human-detection algorithm for unmanned surveillance area in bulk ports. *Mathematical Problems in Engineering*, 2014:1–17, 11 2014. doi: 10.1155/2014/386764.

[27] K. Adcock M. Brooks A. Conway M. Knight S. Mainka E. B. Martin T. Teferi N. Leader-Williams, S. Milledge. Trophy hunting of black rhino diceros bicornis: Proposals to ensure its future sustainability. *Journal of International Wildlife Law Policy*, 2005.

[28] Ogheneuriri Oderhohwo, Tolulope A. Odetola, Hawzhin Mohammed, and Syed Rafay Hasan. Deployment of object detection enhanced with multi-label multi-classification on edge device. *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), Circuits and Systems (MWSCAS), 2020 IEEE 63rd International Midwest Symposium on*, pages 986 – 989, 2020.

[29] Sara Olsson and Amanda Tydén. Edge machine learning for animal detection, classification, and tracking. 2020.

[30] Zachary Pezzementi, Trenton Tabor, Peiyun Hu, Jonathan K. Chang, Deva Ramanan, Carl Wellington, Benzun P. Wisely Babu, and Herman Herman. Comparing apples and oranges: Off-road pedestrian detection on the nrec agricultural person-detection dataset. 2017.

[31] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Pattern Analysis and Machine Intelligence, IEEE Transactions on, IEEE Trans. Pattern Anal. Mach. Intell*, 39(6): 1137 – 1149, 2017.

[32] Jason G. Wallace Hamid R. Arabnia Sahar Voghoei, Navid Hashemi Tonekaboni. Deep learning at the edge. 2018.

[33] Jia Songmin, Wang Shuang, Wang Lijia, and Li Xiuzhi. Robust human detecting and tracking using varying scale template matching. *2012 IEEE International Conference on Information and Automation, Information and Automation (ICIA), 2012 International Conference on*, pages 25 – 30, 2012. ISSN 978-1-4673-2236-2.

[34] Swara. Black rhino population increasing slowly. 45(2):8–8, 2020.

[35] C. et al. Szegedy. Going deeper with convolutions. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[36] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. pages 1–9, 01 2013.

[37] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. volume 1, pages I–511, 02 2001. ISBN 0-7695-1272-0. doi: 10.1109/CVPR.2001.990517.

[38] Hui Zhong, Zaiyi Chen, Chuan Qin, Zai Huang, Vincent W. Zheng, Tong Xu, and Enhong Chen. Adam revisited: a weighted past gradients perspective. *Frontiers of Computer Science*, 14(5), Jan 2020. ISSN 2095-2236. doi: 10.1007/s11704-019-8457-x. URL `http://dx.doi.org/10.1007/s11704-019-8457-x`.

[39] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.