Advanced wildlife camera trapping using embedded AI machine vision

Nathan van Dijk

July 2023

Contents

List of Figures

List of Tables

Abstract

Conservation of biodiversity is important as a part of safeguarding earth's ecosystems. Collection of biodiversity data is usually done with wildlife cameras equipped with simple sensors based on movement or body heat at remote areas and limited resources regarding power and image storage. A downside of these systems is that they generate many false positive thereby overloading the image storage capacity. In this study we examined the potential of embedded artificial intelligence models to replace traditional camera sensors. To do this, different computer vision models were tested after conducting various experiments to optimize performance. The AIR sensor, which served as a baseline for wildlife cameras, resulted in the worst performance and a lot of false positives. A simple motion sensor and two different image classification models were tested on a data-set, which was collected by a video camera inside a nestbox. These achieved higher performance, but still contained false positives. A final trigger method was designed where an image classification model was run on images that were determined to contain significant amounts of movement. This resulted in the best performing model, which could identify nearly all true positives, while reducing the false positives to nearly 0 at the cost of a higher power consumption.

Chapter 1

Introduction

The earth's ecosystems, species and biological traits are dismantling at an alarming rate [11]. When too many species disappear, human life will not be sustained. Therefore, it is necessary to understand what actions are suitable to conserve and/or restore biological diversity. One way to collect data for research on this topic is through the usage of wildlife cameras that can record the behavior of animals in an unobtrusive manner. Wildlife cameras record useful species information at a large scale which helps in managing and analyzing biodiversity data. Many wildlife cameras record continuously or use triggers to capture footage when an animal of interest is in front of the camera. For biodiversity monitoring, these cameras are deployed in remote areas and are powered by batteries or solar-power.

Most traditional wildlife cameras use infrared sensors as triggers to detect motion. This works well in detecting wildlife, but there are downsides to this approach. It is biased towards more false positives due to triggering of the sensor by something that is not of interest, for example through the movement of a leaf or a branch. Furthermore, detecting a large variety of animals can be very difficult as species differ in many aspects such as size, color, behavior and more. Besides differences in animals, the weather itself can cause many visibility problems for both the sensor and camera. This combination of variables make it more difficult to detect animals, which leads to inaccurate data due to missing animals when monitoring biodiversity. The goal is therefore to reduce the amount of false triggers from such variables and to improve overall performance.

The current state of Artificial Intelligence (AI) now opens up the possibility to use sophisticated algorithms on low-power systems. This is referred to as Embedded AI, which is the application of machine- and deep learning in software at device level. With this, the objective is to find and tune different designs for an intelligent camera trigger system that can detect wildlife using low-power computer vision methods on resource constraint monitoring devices. Such systems would be able to evaluate their surroundings and ideally filter out false positives. A literature review is performed to find different technologies and methods for triggering a camera. The focus will be on finding the best performance, while also minimizing power consumption. Trade-offs between performance and power consumption are explored due to power being very limited in remote areas. This will be approached by collecting data from an infrared sensor as reference and video camera for annotating the ground truth. This data-set is used to determine the performance for infrared based wildlife cameras.

Chapter 2

Literature Review

2.1 Methodology

For this literature review, the sources used to search for related works are Scopus, ACM Digital Library and Google Scholar. The different keywords that were used, consist of the following: Machine vision, embedded AI, edge computing, wildlife detection, camera trap, low power, object detection, motion detection, animal detection, presence detection, accuracy, thermal imaging, microwave, infrared, sensors, millimeter wave, radio waves, sound, radar, sonar, lidar. These keywords were initially found, or found through other research and investigated further. Additionally, relevant filters were used in order to focus on recent articles that were published in the past 5 years. Besides scientific articles, occasionally another filter was added to focus on literature reviews.

2.2 Wildlife camera

A wildlife camera, also known as a trail camera, is designed to capture images or videos of animals and their habitat. They typically use motion sensors or heat sensors to detect animals which then activate the camera to capture images. As the name suggests, this is often done in the wild on remote locations where animals are likely to appear. These cameras are usually weather-resistant as they are exposed to the environment, often attached to a tree and need to be camouflaged in order to stay unnoticed by animals. Wildlife cameras are often used by researchers, hunters, and nature enthusiasts in order to study animal behavior, monitor population, or simply to enjoy the beauty of wildlife and nature. Because these cameras are often placed in remote areas, without power, the camera is made to be very low power and is equipped with a battery that can last for at least a couple of months. The reason behind this is to avoid having to go up and down to recharge the wildlife camera every couple of days.

2.2.1 Triggering a camera trap

Any camera that is triggered by the presence of an animal can be classified as a camera trap. These cameras can include a whole range of trigger methods such as trip-wire, pull-wire, pressure plates, lasers or body heat sensors [50]. To simplify these different methods, most forms of wildlife camera triggers can be divided into two categories: active sensing and passive sensing. These distinguish between direct detection of animals (active) and indirectly sensing of animal presence (passive).

With active sensing (e.g. ultra-sonic sensors), the device will actively send out a signal which gets measured again once this signal reflects back into the device and gets picked up be a sensor. Passive sensing is similar to active sensing as it can measure the same thing, but does not rely on first sending out a signal. Passive sensing implies the sensing of the surrounding environment such as ambient heat, sound or light. A passive trigger has a sensor which is continuously measuring and will only trigger the camera once certain thresholds are reached or patterns are recognised.

2.2.2 Traditional camera trap

One of the most commonly used trigger type is through the usage of infrared (IR) light. There are two ways that IR is used to detect the presence or movement of an animal; Active Infrared (AIR) and Passive Infrared (PIR). AIR sensors work with radar technology where it emits and measures IR radiation, see figure 2.1. This radiation reflects back from an animal and travel back into the receiver of the device. With this technology, the sensor can measure the distance between the animal and itself. Changes in this distance will indicate that

something is moving, which will then trigger the camera as it has detected movement. However, AIR sensors will not only detect animals, but also any other object that passes in front of the sensor that can reflect IR light. Vegetation moving in the wind can falsely trigger the camera trap into thinking there is movement from an animal. The sensor will trigger the camera even when there might not be anything of interest in front of the camera. These faulty camera captures are counterproductive when trying to lower the amount of energy used by the device.

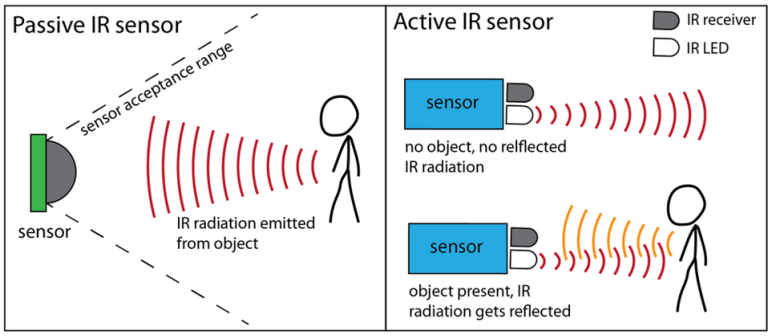


Figure 2.1: Passive infrared sensing and active infrared Sensing [10]

PIR sensors use a pair of pyroelectric sensors that are placed besides each other to detect heat energy emitted from an animal. The detection zone consists out of multiple detection windows. When an animal walks across a detection window, there will be a difference in the amount of IR radiation measured between both pyroelectric sensors. Once the signal differential between the two sensors change, the trigger will engage and activate the camera. This is further illustrated in figure 2.2. As it only measures the amount of heat energy, it cannot determine the distance between object and itself unlike AIR sensors. False detections occur less frequently compared to an AIR sensor because it is not easily triggered by movement from vegetation. Some problems do occur once the ambient temperature reaches the same temperature which animals emit. Another downside to this is that animals of interest whom are cold-blooded will be virtually impossible to detect. The reason why such a sensor is still widely used is because of its extreme low-power design. It can passively detect movement, where it has an resting current draw below a 1 mA. Most other forms of detecting movement do not come close to this level of efficiency.

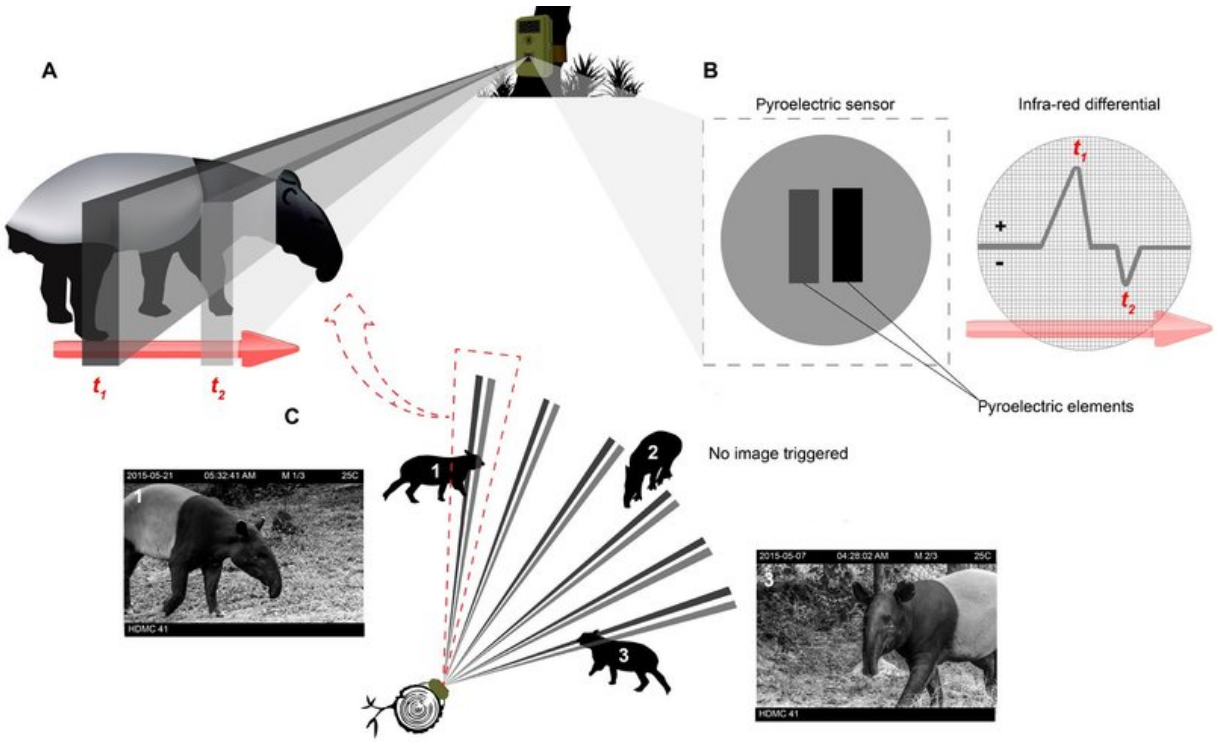


Figure 2.2: A) Detection window. B) Differential in IR radiation between pair of pyroelectric sensors. C) Multiple detection window through the usage of a Fresnel lens. [50]

2.3 Different trigger methods

2.3.1 Electromagnetic spectrum

The AIR sensor sends out infrared light, which is a form of light, but has a lower frequency compared to visible light. Both infrared and visible light are forms of electromagnetic (EM) radiation and are on the same electromagnetic spectrum. On this spectrum (see Figure 2.3), there are many more kinds of EM waves that have different frequencies; radio waves, micro waves, infrared, visible light, ultraviolet, X-ray and gamma rays. These EM waves could be used in a similar way to how AIR sensors use radar technology to function. Depending on the frequency, these waves have certain properties that can be useful for wildlife detection. The (dis)advantages of these EM waves will be discussed in how they can be used as a trigger.

However, not all EM waves will be discussed as some are dangerous for organic life. Those with very high energy such as extreme ultraviolet, X-rays and gamma rays will not be used to detect wildlife. This is because they are classified as ionizing radiation due to their high frequencies and exposure to this can be a health hazard. Besides the fact that they hurt animals, it would also require a lot of power to use such a high energy form of electromagnetic radiation.

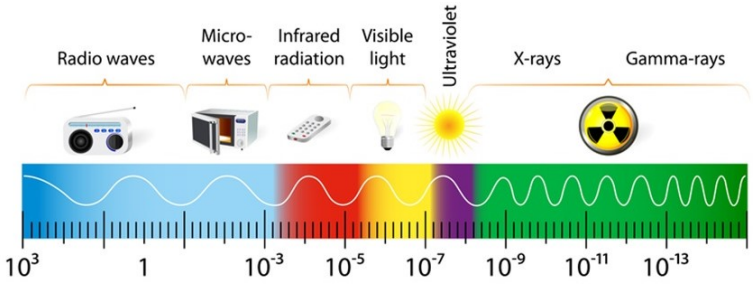


Figure 2.3: Electromagnetic spectrum [5]

Radio waves

Radio waves have the lowest frequency on the EM spectrum with a wavelength of hundreds of meters. Because of this, these waves are far too large and do not even get reflected by an object the size of house, let alone an animal. However, through the usage of RFID tags or radio telemetry it is possible to locate and track movements [25]. These tags do not need power to run and allows the collection of useful information, but do require large and power hungry equipment. The downside of this tactic is that the animal first has to be captured and tagged in order to use this technique. This technique is more suitable to track labeled previously captured animals using an array of detectors in a area.

Micro-waves

Micro-waves are smaller wavelengths compared to radio and are capable of being partially reflected by animals unlike radio waves. These waves can pass through objects such as glass, paper, plastic and other non-conductive materials. Micro-wave sensors can detect motion through the usage of the Doppler effect and are capable of detecting large animals [43]. Portable microwave radar systems can provide high motion detection sensitivity [29].

With microwaves, animals can be detected through low visibility conditions such as snow, fog or complete darkness. Not only this, but animals could also be detected through bushes or even behind trees. This has the potential to detect hidden animals, but currently results in very vague and unintelligible images [12]. Furthermore, if a hidden animal is detected behind bushes, the triggered camera would not be able to see it and capture an empty image.

Microwave sensors use more power and work in intervals. This means that an animal could be fast enough to evade detection. In a comparative study by Glen *et al.* [19] the microwave trigger performed significantly worse compared to the PIR trigger, with a false trigger ratio of 90%. The sensor often failed to detect an animal moving in front of it and was also frequently triggered by small movement such as rain or foliage movement in the background.

Millimeter waves

Millimeter waves are similar to microwaves, but are a factor of 10 smaller between 1 mm and 10 mm. Millimeter wave sensors have more precision compared to microwave sensors and offer excellent depth resolution even at long

ranges [37]. With computer vision techniques it can provide accurate depth estimations, even in highly cluttered environments. Furthermore, due to the smaller wavelength it can detect very slight movements. Because of this, it is capable of recognising subtle hand gestures from up close [30]. This does rise the problem of increased false positives as the sensor is easily triggered by subtle movements which can occur in the background.

Infrared waves

Besides PIR and AIR, infrared light can also be used as a laser. With this method an IR laser is directly pointed into a receiver and once this light beam gets interrupted, it measures a drop in IR light and gets triggered. This is called a break-beam IR sensor. Another purpose for IR lasers is to measure the distance from an object with LiDAR technology. This allows the ability to map the environment around itself. This technology is widely used in self-driving cars in order to scan the area and detect obstacles, see Figure 2.4. The advantage of this is that bad weather conditions do not reduce visibility, with the exception of temperature, and can even scan and detect sea-life underwater [15]. However, as IR light is absorbed by water, this technique has its limitations.

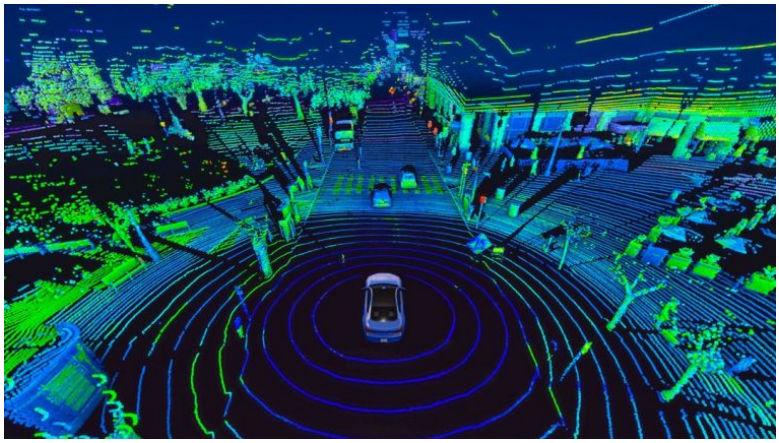


Figure 2.4: Electromagnetic spectrum [26]

Visible light waves

Visible can also be used in the same manner as AIR sensors. However, this would of course emit visible light which is easily noticed by animals in a very obtrusive way. Visible light can be used as a passive sensor, which is essentially a camera, where differences in between frames can be found. More about this will discussed later on. Compared to infrared, weather conditions such as fog, rain or snow do have impact on the visibility. One advantage is that it is not affected by temperature and generally creates detailed images.

Ultra-violet waves

Cameras that can measure UV light show interesting details about animals. Some animals have visual patterns in their appearance which can only be seen with a UV camera, see Figure 2.5. Using UV sensors can reveal these patters and identify animals based on their bio-fluorescence [49]. Normal cameras have difficulties in detecting camouflaged animals, but might be seen with a UV camera if the animal reflects UV light. This bio-fluorescence is common in reptiles, fish, birds and some mammals. Furthermore, UV cameras can also illuminate fluorescent tags used in tag-recapture studies.

2.3.2 Sound

Ultra-sonic Ultra-sonic sensors work very similar to AIR sensors, but send out a pulse of high frequency sound rather than light. These sensors can detect motion from animals with limited distance [41]. The advantage of ultra-sonic sensors is that they do not rely on visual variables or weather conditions. The main limitations are false triggers from other high frequency noises (30kHz - 500KHz), soft materials which do not reflect ultrasonic sound very well and the fact that some animals can hear ultrasonic sound. Sonar is very similar as it is also an ultrasonic sound, this is mainly used in scanning and mapping the ocean floor.

Passive acoustic

Passive acoustic monitoring involves surveying and monitoring wildlife and environments using sound recorders such as a microphone. This can pick up the sound that animals make. Some microphones do not require power

Figure 2.5: UV patters on animals [46]

to operate and produce data much smaller compared to a camera. This results in a very low power trigger mechanism. The location of the animal can be determined using sound source localization based on a binaural model [41]. With this the camera can be turned towards the animal in order to capture images. The drawback with this is that animals do not always make sound when walking by and gets falsely triggered by other sounds too.

Zhao *et al.* [54] used hydrophones to detect underwater audio and even generate enough electricity from it in order to run a lightweight deep neural network. This was done with an ultra-low-power micro-controller and on-board sensor, which performs local inference on sensed measurements.

2.3.3 Non-trigger

Time-lapse

A time-lapse camera trap is not triggered or influenced by anything except for an internal timer. A time-lapse camera trap captures images in intervals, for example every minute. This results in many empty images, as the camera trap has no method to determine if there is an animal present. Time-lapse is generally the most easy way to collect a lots of data, especially when the interval becomes shorter such as single second between images. This will have a nearly 100% success rate in capturing all animals that pass in front of it, but will create a greater amount of empty images. This will rapidly fill the storage on the camera and the data-has to be automatically or manually filtered from these empty images. This will serve as a ground truth when evaluating performance during experiments.

2.4 Camera trap performance

Even when different camera models use the same methods of triggering, they can still vary a lot in their ability to detect animals [16]. The performance of a camera trap does not solely depend on the method of triggering. The environment can hinder the trigger and the camera can result in poor data quality. General features for a camera trap, that have influence on its performance in detecting animals, will be discussed in the following points.

2.4.1 Detection difficulties

The environment surrounding the camera has great influence in the decision making of what kind of sensors are needed. Weather conditions can change over time, which needs to be taken into consideration. A strong outer case is necessary for protection against blunt forces which can arise from storms or attacks from animals. In locations where there is rain, the camera trap needs to be water-proof. This is helpful in situations where the goal is to monitor aquatic life where the camera trap can placed under water. In some occasions the weather can obstruct the sight of both the trigger mechanism and the camera. A normal camera cannot see very var in the dark or in foggy climate. Heat cameras will have difficulties when ambient temperatures are too high. Motion detectors can get falsely triggered by treas and leaves moving in the wind.

Nature can provides shelter such as trees, bushes, burrows or tall grasses for animals to stay hidden and protected from possible danger. This can make it difficult to detect animals, especially those with color and patterns on their fur that are well camouflaged and blend into the environment. Furthermore, animals that are of small of size or travel at high speeds will be more difficult to detect as the trigger mechanism might not pick them up.

This can be circumvented by taking care that the camera is positioned in such a way that it is protected from environmental factors (a.o. covered to protect against sun, rain and wind) and ideally placed to spot animals emerging from the vegetation.

2.4.2 Data quality

A well performing camera trap needs a good trigger, a long life time and also have great image quality. Poor image quality can lead to useless data due to animals not being clearly visible on the camera. Furthermore, it is preferred by many scientists that the camera trap has a long lifetime [33]. This is because camera traps will collect more data as they monitor over longer periods of time. There are plenty of methods and adjustments that can be taken into consideration when trying to improve image quality.

Trigger specifications

- **Trigger speed**: How fast the camera responds to the detection of wildlife from the trigger. This is called the trigger speed and is defined in the amount of seconds from the moment when the trigger is activated to the moment an image is captured. Slow trigger speeds can result in missing faster moving animals. A trigger speed of 1.5s is often sufficient to detect 80% - 90% of animals [19][50].
- **Recovery time** How long it takes for the trigger to recover from the initial detection until it is able to activate a second time. This is important when multiple images are required. The recovery time is measured in seconds and defines the shortest time possible between two activation's of the trigger.
- **Detection zone** The area in which an animal can be detected by the trigger. In this zone animals can be detected from far distances or in some cases only from close distances. This is called the **detection distance** where there is a limit in how far an animal can be for it to be still detectable by the trigger. The detection distance is often in the range of 10m up to 30m [50]. The other variable that determines the size of the area besides detection distance, is the **detection angle**. This angle is reported to be between 15° and 75°, this depends on the vegetation and the kind of animal that is monitored [50].

Camera specifications

- **resolution** The detail which an image holds where a higher resolution means more image detail. The term resolution is often considered to be the number of pixels in an image, counted in megapixels. However, this is not an accurate way of measuring visible resolution as many cameras will 'up-scale' images using interpolating algorithms. An image can have many pixels, but still appear as poor image quality. Poor image exposure, motion blur and poor focus are also causes for a lower visible resolution.
- **Sensor size** The size of the inner hardware that captures light and turns it into an image. Large camera sensors can contain more pixels that can lead to less noise and better exposure. Compact camera sensors will generally require less power, but result in lower image quality. Because of the constant change in technology, there is no simple relationship between the sensor size and image quality.
- **Media type** The form of media in which the camera captures data when triggered. This could be an image, video or both. Sometimes sound is also recorded along with a video.
- **Field of view** The area which the camera can image is called the field of view (FOV) and is similar to how the detection zone works for a trigger. This is usually the same size or larger compared to the detection zone and have to overlap in order to capture the animal that is detected.
- **Images per trigger** The amount of images captured from a single trigger activation. A burst of images is taken where if the camera takes pictures fast enough (>1 fps), it can create a "near-video" effect. This can record animal movement and behaviour, but uses less memory and little power compared to taking video (30 fps).

Lifetime expectancy

- **Power consumption** The amount of electricity used by the camera trap in order to function. Power consumption is depended on the efficiency of the camera and trigger. The average infrared wildlife camera consumes around 0.15 milli-amperes (mA) at rest and roughly 500 milli-ampere seconds (mAs) during an entire day.[1].

- **Battery type** The three most common used battery types used in wildlife monitoring are Lithium, Nickel-Metal Hydride (NiMH) and Alkaline. Lithium batteries have high power, but are rather expensive and only have one life. Alkaline batteries are most commonly used, but discharge quicker than Lithium and NiMH batteries. NiMH batteries are rechargeable and allow multiple uses, but have lower voltage compared to Lithium and Alkaline batteries. Most wildlife cameras use premium batteries with a battery capacity up to 5000 mah and are either 6V, or more commonly 12V.
- **External input** With an input jack it is possible to attach an external battery pack. This increases the battery capacity and increases life time of the camera trap. Rechargeable batteries can also be charged with solar panels or hydrophones which generate electricity from light or sound respectively.
- **Data generation** The camera captures images which are stored as data in a memory file. The amount of data that gets produced depends on the media type and quality of the camera captures. Video files are much larger compared to an image and thus generate much more data from a single trigger activation. A higher resolution and more detail also result in larger file sizes for the same amount of images or video.
- **Memory storage** The amount of images that can be taken depends on the maximum capacity of the memory storage. More memory storage means that more images can be saved until it is full. Common types of memory cards used in camera traps are SD cards and SDHC cards, which typically have a capacity of 2-32 GB [33]. For reference, a 5 megapixel camera with only 1 GB of memory is estimated to hold almost 600 images. An uncompressed video that films at the same resolution on 30 fps will roughly last a minute before it reaches 1 GB. Compression algorithms can greatly decrease the file sizes of images and videos, which further allows more capacity for storing data.

Additional improvements

- **Flash** The usage of a flashlight can help in capturing a clear image, especially during the night. LED White-flash create sharp and colorful images, but are noticeable for animals. Xenon white-flash outperforms LED white-flash where it produces a more powerful flash that create much sharper images. These bright flashes improve image quality, but can also result in animal behaviour change that leads to animals becoming "camera shy" [50]. LED infrared-flash produces an invisible light which can only be seen by an infrared camera. This camera will capture images in monochrome colors. The advantage of LED infrared-flash is that it is mostly invisible to animals with only a few exceptions. Moreover, there are "no-glow" IR LED flashes that minimize the red glow produced in normal IR LED flashes to further reduce the camera shyness in animals[40]. Due to the relatively low energy provided by the IR flash, it is needed to have a longer exposure time which can result in blurry images [21]. There seems to be no significant difference between IR flash and white flash in the probability of detection [21].
- **Multiple cameras** Having more camera traps spread over an area will increase the chances of detecting animals. Furthermore, having more than one camera on the same location allows the capturing of an animal from multiple angles.
- **Setup support** The position of a camera trap determines what the camera trap will monitor. Through the usage of a compass, the camera trap can be angled in a way to face away from direct sunlight. A tiny screen on the camera or small laser pointers can help in seeing what the FOV is of the camera. This allows the user to adjust the camera in order to fully cover the desired area that needs to be monitored.

2.4.3 Evaluating performance

The ideal camera trap should reliably get triggered 100% of the times when there is any animal in front of the camera. In reality this does not happen, some animals are detected and some are missed. The evaluation of the performance is based on the counts of animals that are correctly identified and incorrectly identified. In evaluation metrics, there are 4 classification metrics that help in finding the performance of a model (see Figure 2.6) [31]:

- True Positive (TP): Animal is present and the camera is triggered.
- False Positive (FP): Animal is absent and the camera is triggered.
- True Negative (TN): Animal is absent and the camera is not triggered.
- False Negative (FN): Animal is present and the camera is not triggered.

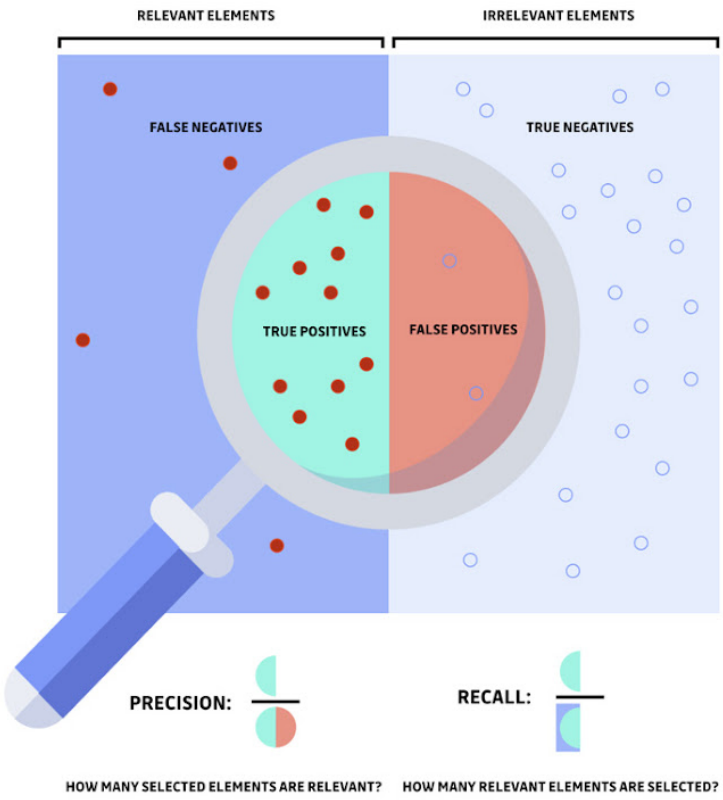


Figure 2.6: Diagram explanation of evaluation metrics

The performance of a camera trap can be defined in its accuracy, precision, recall and F_1-score. Accuracy is defined as the ratio of correct detections and correct non-detections compared to all (in)correct detections and non-detections. A higher accuracy means that the trigger makes more correct identifications and fewer false identifications:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

The precision is defined as the ratio of correctly identified animals compared to all instances where the camera trap was triggered. The more false positives that are found, the lower the precision:

$$Precision = \frac{TP}{TP + FP}$$

Recall (or sensitivity) is defined as the ratio of correctly identified animals compared to all animals that have walked in front of the camera. The more false negatives that are found, the lower the recall:

$$Recall = \frac{TP}{TP + FN}$$

The F_1-score combines both the precision and recall into one formula in order to evaluate the performance of the camera trap. The F_1-score provides a single score that represents the balance between precision and recall. A high F_1-score means that there are few false positives and few false negatives. A perfect F_1-score of 1 means that the camera trap is fully reliable and makes no errors. Calculating the F_1-score is done with the following formula:

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Besides the 4 different metrics that have been mentioned, there is one other scenario that can occur in detecting wildlife. It is possible to have a situation where a camera is falsely triggered for something that is not of interest, but coincidentally there happens to be an animal in front of the camera. This is called a False True Positive (FTP)[28]. For this research, FTP will simply be considered as being a true positive.

Another method which is used for evaluating the performance of a model is by creating a Receiver Operator Characteristic curve (ROC). This curve is a graph that shows the performance of a model at all confidence score thresholds, which plots the True Positive Rate (TPR) against the False Positive Rate (FPR). To quantify the performance, the Area Under Curve (AUC) is calculated, which is defined as the entire area underneath the ROC curve. This metric provides an aggregate measure for performance across the sensitivity scale of a model.

2.5 Artificial intelligence

Artificial intelligence (AI) is a program that can sense, reason, act and adapt. This section introduces a certain sub-field in AI called Machine Learning (ML), which are algorithms whose performance improve as they are exposed to more data. Furthermore, it outlines the steps necessary to implement embedded artificial intelligence.

2.5.1 Machine learning

Machine learning (ML) is a sub-field in artificial intelligence (AI) that involves creating algorithms and models, which enables computers to make predictions or decisions based the input data. In other words, this is the process of training a computer with provided data to recognize patterns, make predictions and take actions based on those patterns. When new data is inputted in a deployed model, the label prediction process is also called inference. ML algorithms are designed to learn from data, rather than being explicitly programmed to perform a specific task. It uses statistical techniques and algorithms to automatically identify patterns in data and make predictions or take actions based on those patterns. As the model learns and progresses through the data, the model's parameters can be adjusted accordingly until it is able to accurately make predictions or decisions based on that data.

Large ML algorithms are already used to rapidly classify animals in images collected from camera traps [44]. Most ML algorithms have a high computational demand and require a lot of power to function, this makes the adaptation of embedded AI much more difficult [9]. With current technology advancements, ML algorithms are possible on much lower power, small micro-controllers [42]. This allows for intelligent computations and decision making on these small devices such as a camera trap.

model training

There are many different types of ML algorithms, the most common ones are shown in Figure 2.7. The process of training a model with ML can be done supervised or unsupervised. This involves providing data that is either pre-classified / labeled with specific outcomes, or completely unlabeled. This data is often split into 3 parts where the largest part is to actually train the model (70%), where the remainder is for validating the model during training(20%) and the very part is for testing the trained model (10%). Supervised learning is useful for small data-sets and if the data-set is large it is better to use unsupervised learning as this generally performs better, but requires more data [32]. With truly huge amounts of data it is possible to use deep learning (DL) techniques. This is a sub-field within ML where multi-layered neural networks (NN) learn from vast amounts of data. These neural networks are similar to human brains and consist out of an assortment of algorithms used in ML.

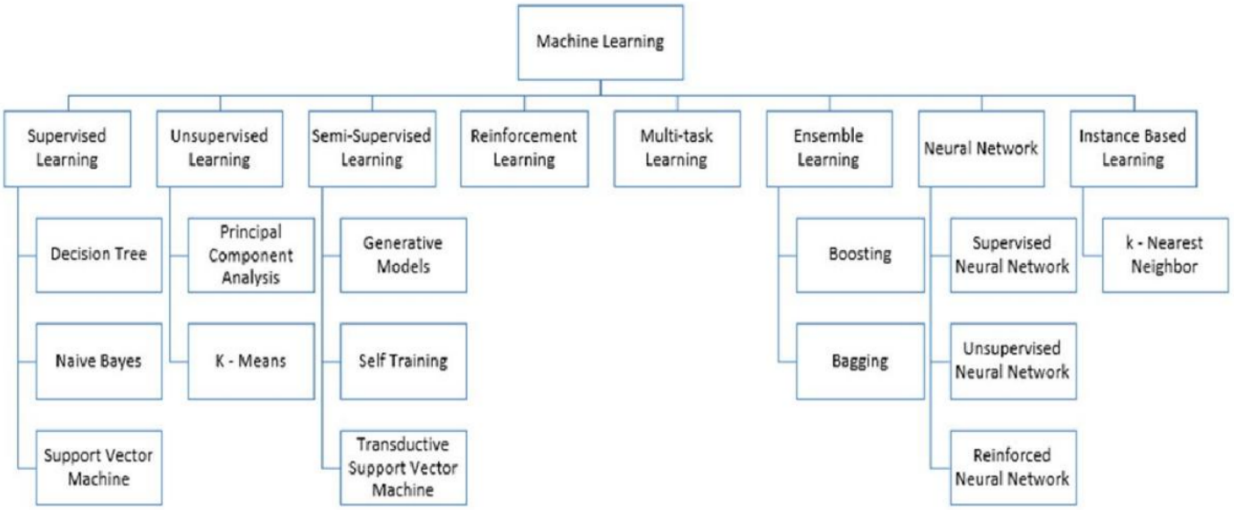


Figure 2.7: Machine learning algorithms [32]

model compression

ML models that perform well, especially the more sophisticated DL models, are big in size and take up a lot of memory and storage space. This will also result in longer inference time and higher power consumption. Reducing the size of such models with AI compression methods will make the model much smaller, run faster

with very minimal loss in accuracy. There are many different compression methods that can be utilized for model size reduction. The most popular techniques in NN compression are: Pruning, Quantization, Knowledge distillation, and low-rank factorization [35]. These techniques can remove redundant parameters, reduce bit size of values and have a larger, more complex NN pass its knowledge to a smaller NN. In some cases, these compression methods can be responsible for compressing model size up to 10 to 40 times and even speed up inference by 3 to 5 times compared to its original speed [35].

2.5.2 Hardware requirements

For a ML algorithm to be embedded on a device and be functional, 3 main components are necessary. Processing units such as a CPU or GPU for computational power, RAM modules for memory space and lastly, SD cards or SDHC cards for storage space [9] [33]. Training the model uses a lot more resources compared to when the model is deployed. Once the algorithm is fully trained, it processes data much faster and less computationally intensive compared to the training phase. Depending on the complexity of the task, the device needs to be equipped with more memory and higher performance processing capabilities.

2.6 AI applications

2.6.1 Machine vision

Traditional camera traps are often equipped with IR sensors to detect movement or animal presence, but this technique is incapable of evaluating whether images contain animals or not [50]. However, the evaluation of animal detection and recognition is possible with the usage of machine learning algorithms on image feature extractions [6]. This technique of image evaluation is called Machine Vision (MV), or also known as computer vision (CV) and is defined as a way to automatically inspect and analyze images. Such an algorithm can be trained in order to find and recognize patterns in animals from images, which can then be used in the decision making process to determine animal presence [6]. Different learning algorithms can be explored to find advantages / disadvantages and which model is most suitable as trigger for a wildlife camera [32]. Once said models are trained on a sufficiently large data-set, it should ideally identify animal-specific features with relative high accuracy [32]. This should then in turn result in improved animal detection performance and fewer false triggers.

Machine vision based on RGB imaging

Red, Green, Blue (RGB) are the three colors in which we perceive the world in, which is the same for a standard camera that creates images in the visible spectrum. RGB cameras have advanced a lot and are capable of producing very high quality images, even from far distances with the right lens. This allows for very fine feature recognition and animal identification in detailed images. The simplest form of motion detection, where the difference in between frames is measured, still results in many false triggers. This is because a high resolution camera does not only pick up small details from animals, but also all the small details and movements from the background. Riechmann *et al.* [39] found a way around this by using motion vector algorithms on images (see Figure 2.8). Using a threshold that is set on the amount of 'random' movement or noise. A larger animal recognition algorithm was used to analyze the image when motion was detected and resulted in an 85% accuracy with 4W of total power consumption.

An alternative method proposed by Riechmann *et al.* [39], is based on 'learning' the background which over time will exclude subtle movements from the background. By subtracting this learned background, a mask is created in black and white with the latter indicating the foreground. Subsequent animal detection is than based on determining the size of the white area or simply counting white pixels.

Detecting animals in the first place does prove to be a bit harder as animals can be (partially) hidden or camouflaged. Furthermore, RGB cameras works very well during the day, but fail at night or with low light such as during dusk, dawn or fog. This can be solved with the usage of an additional flash light, but is an obtrusive method and uses a lot of power if used continuously . The average large flashlight uses 3W or more continuously. Another possible solution to RGB cameras having poor visibility in the dark, is a residual light amplifier. This creates night vision through the usage of an electron tube that amplifies the available light in the dark [4].

Machine vision based on thermal imaging

Machine vision is not limited to captured images from 'normal' cameras that record visible light, but is also used to analyze thermal images [8]. Generally, any form of imagery can be used as input for Machine vision. Thermal vision is a camera that can see infrared radiation or heat energy and create heat maps. Thermal vision

Figure 2.8: Motion detection through machine vision on RGB image [39]

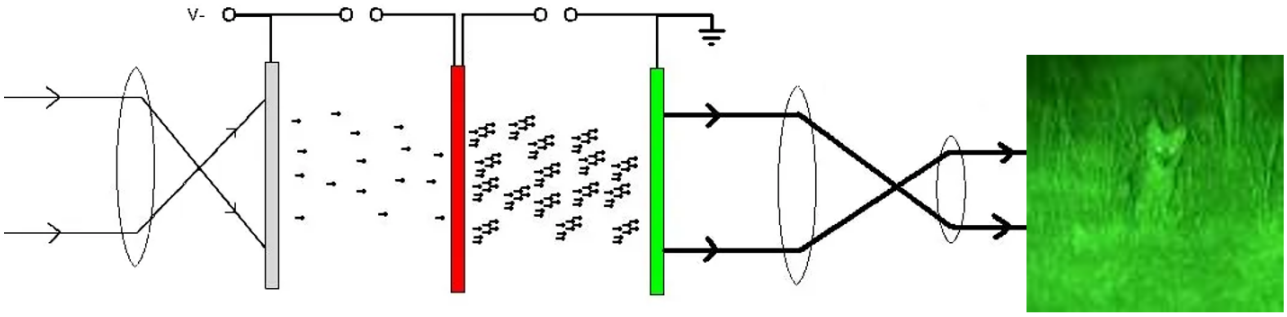


Figure 2.9: Residual light amplifier [4]

is similar to PIR, but can make coarse images with infrared array sensors [18]. This has the advantage over RGB cameras as it still works during the night, rain, fog, snow and other low visibility weather conditions with underwater monitoring being an exception.

When an animal has a different temperature compared to the background, it is significantly easier to detect the presence and location of the animal compared to RGB cameras. However, the analysis of the animal becomes more difficult for thermal images as they leave out small details such as color, fur patterns or material structure [18]. This is because most infrared sensors are accurate within 2 degrees Celsius, which results in materials of similar temperatures to appear as an unintelligible blob.

Cold blooded animals adopt the temperature of their surroundings, this makes thermal imaging useless in finding animals such as reptiles, amphibians, insects and more. A new camera has been developed by University of Central Florida, which exposes pythons for hunters with special wavelength of light 2.11. This camera uses near infrared light to detect snakes, which reflect this light differently compared to the vegetation found in South Florida [34]

Machine vision based on UV imaging

Ultra-violet images can reveal hidden animals that appear well hidden to the naked eye. An example of this is with polar bears and seal pups [27]. Due to their white fur, they are difficult to see in the snow where they blend in. However, this fur absorbs a lot of UV light which would show up as a very dark spot on a UV camera. This is makes it much easier to detect certain camouflaged animals as they stand out more in the UV spectrum.

Smaller details were also found in the study from H.Matthew [49] where the effectiveness of machine learning algorithms on classifying ducks using UV imagery was tested. Not only did the study successfully identify ducks, but also accurately determined age, sex, and species with up to 80% accuracy by using an UV spectrometer. This was combined with a thermal image to detect animal presence. Some limitations with this technique is that it is only effective during the day when the sun emits UV radiation. During the night it would be necessary to add an UV lamp, which can be seen by some animals.

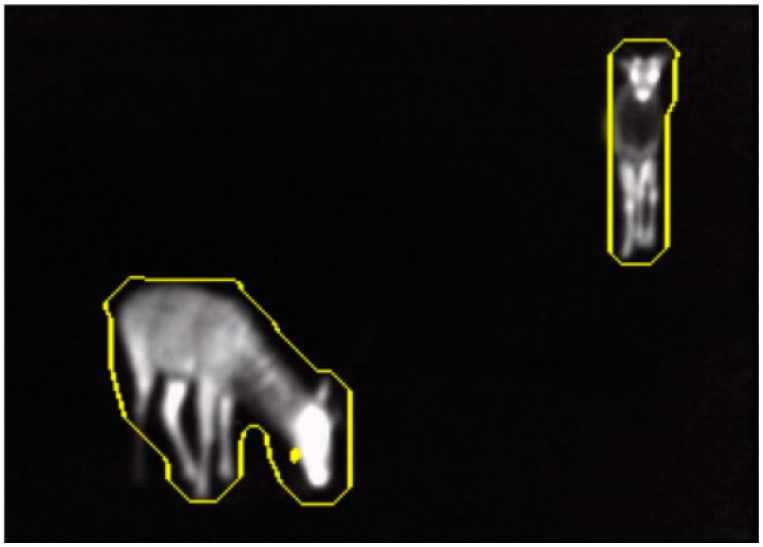


Figure 2.10: Region Proposal algorithm in thermal image [36]

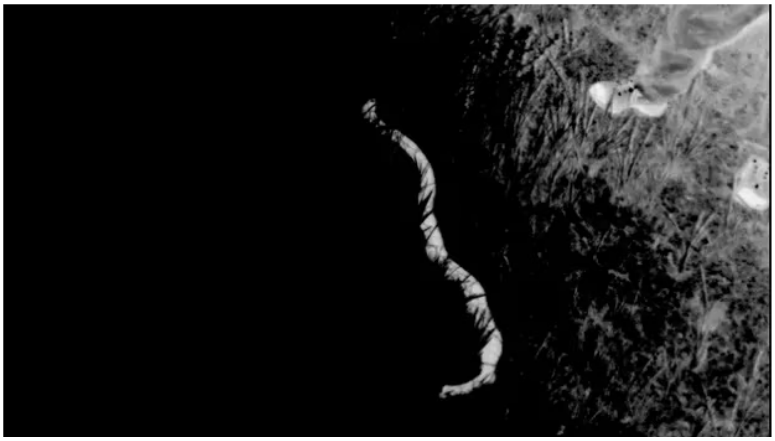


Figure 2.11: Detecting pythons with near infrared light reflections [34]

2.7 Tiny Machine Learning

Deep Neural Networks are an example of very powerful classifying tools and can reach an accuracy of more than 95% [48]. However, these algorithms take a lot of computational power, where the larger kinds need 40 billion operations for classifying one image and can consume up to 10W or more. With compression techniques, this can be taken down to a couple million operations [20]. By doing this, the training time will increase, which also requires a larger data set.

Tiny machine learning (TinyML) is a sub-field of machine learning which pursues enabling ML applications to be embedded on small, low-power, cheap devices. This is found to only have a power consumption between 0.1W - 0.3W [7]. Because TinyML is so low-power, it needs appropriate software, hardware, and algorithms. Most available hardware platforms for tinyML applications have a processing speed below 100Mhz and on average less than 1MB of flash and SRAM [29]. With minimal processing and memory capabilities, this is not suitable to run full-fledged ML models [38].

Some low power software that are applicable for tinyML are TensorFlow Lite, uTensor PyTorch, mobileNet-V1 and more, which support image classifications [38]. Wong *et al.* [51] introduces AttendNets, a low-precision, highly compact deep neural networks tailored for on-device image recognition [51]. In terms of lowest architectural and computational complexity, AttendNet achieved significantly higher accuracy, but requires fewer parameters, operations and lower weight memory requirements. Similar to Riechmann *et al.* [39] who succeeded in low power animal recognition embedded in camera traps using YOLOv4-tiny and obtained an 85% accuracy in animal detection.

2.8 Challenges

Camera traps are relatively small devices which do not have large computational abilities, large memory or abundant energy to use. This is because of the power constraints from being installed in remote locations, near animals. Using computer vision techniques introduces the problem of requiring the computer to 'see' the world with a continuously running sensor, such as a video camera. Furthermore, processing units that can process data faster, often draw more energy. This further reduces the lifetime of the camera trap and data collection when it only relies on batteries. Another constraint is from the memory which the ML algorithm takes up during computation. This can run into problems with small memory sizes on these devices. Furthermore, this also results in difficulties when implementing different algorithms for motion detection or feature recognition due to these energy, memory and computational power constraints.

2.9 Overview

The development of advanced algorithms and hardware technologies has allowed for implementation of low-power animal recognition models on small devices. Machine vision has enabled researchers to collect larger amounts of data on animals which were previously not detectable or hard to detect with fewer false triggers. Other methods for triggering a camera traps contained many false triggers and would lead to different animals not being detectable, such as fish, distant or small animals (See Table 2.1).

Video cameras have difficulty seeing through fog, rain, or in the dark, but create better quality images with high detail that can be analyzed. Besides visible light, IR and UV lights are visible to some animals. A combination of RGB imagery with thermal imagery would result in the ability to detect nearly every animal. This allows better detection rates for small, large, cold-blooded, underwater and camouflaged animals. Sound sensing has similar advantages and disadvantages, but results in data containing less information. In some cases, an animal can be detected behind a bush, but the camera would not be able to see it. Electromagnetic sensors do not suffer from environmental conditions with the exception of temperature for IR sensors, but are generally very sensitive to background movement, consume more power and are prone to false triggers. Furthermore, small EM sensors generally create relatively vague data, which is hard to evaluate for animal detection.

The use of machine vision is found to be an effective tool in wildlife monitoring by providing a non-invasive, continuous, and accurate method for animal detection. Machine vision on UV imagery has its potential, but lacks in visibility at night. Thermal imaging does have night vision unlike UV and RGB cameras, but runs into problems with temperature conditions and sunlight. A combination in different imaging methods would help minimizing these limitations, but is not feasible for this project. Therefore, the usage of a small RGB camera for computer vision is chosen to be further explored as method of detection for wildlife animals.

Trigger methods overview			
Trigger method	Undetectable animals	Causes of false triggers	Power consumption
Micro-wave proximity sensor [12][19][29][43]	Sea-life	Extremely sensitive to movement	1.1W - 1.5W
MM-wave [30][37][47]	Sea-life	Sensitive to movement	12.5W
AIR [50]	Distant, small, hidden, cold-blooded or sea-life	Temperature, very sensitive to movement	0.1W
LiDAR [13][15]	Small or hidden animals	Bad weather	2.5W - 8W
PIR [47]	Distant, small, hidden, cold-blooded or sea-life	Temperature, sensitive to movement, bad weather	0.325W
Thermal imaging [49]	Cold-blooded, hidden or sea-life	Animals within 2° C of background, bad weather, hot spots	1.5W
RGB imaging [39]	Camouflaged or hidden	Sensitive to movement, bad weather	0.4W - 1.4W
UV imaging [49]	Similar UV absorption to background or hidden	Other UV reflective objects	1.25W
Ultra-sonic [41][47]	Distant, hidden or Woolly animals	Sensitive to movement	0.075W
Acoustic [41][54]	Silent animals	Unrelated sounds	0.05W

Table 2.1: Trigger methods overview

Chapter 3

Research questions

Considering that the original focus of this thesis is on the evaluation of a camera trap/AI system suitable for use in remote areas, a proper trade-off between accuracy and power consumption must be realized, enabling detection of a large variety of animals leading to the following research questions:

> *How can Embedded AI be utilized to reduce false triggers and improve performance of animal detection in camera traps?*
>
> *What are the trade-off boundaries between performance and power consumption?*

With trade-off boundaries, it is referred to finding how suitable certain methods are for wildlife cameras depending on their power consumption or performance.

Chapter 4

Ideation

4.1 Experimental Setup

The first step to answering the research questions, is to have the most common method of triggering a camera trap as a reference point for standard wildlife monitoring performance. This serves as a reference point to compare to when evaluating other detection methods. The goal was at first to evaluate the performance of a PIR sensor, as this is the most used trigger method for standard wildlife camera traps. Unfortunately, the camera with PIR sensor could not be delivered in time by the manufacturer. This lead to the need of an alternative method, where the PIR is to be substituted by another sensor and the experiment setup needs to be slightly altered. As an alternative plan, it was decided to use raw video footage, provided by the supervisor of this project, that was obtained from a nestbox monitor equipped with an AIR sensor.

For this new experiment setup, the camera was placed inside the nestbox and positioned in a way where it monitored the entrance and a small part of the nest (Figure 4.1). The AIR sensor was positioned in front of the entrance on the outside of the nestbox to ensure that birds had to pass through the IR beam in order to get in the nestbox. The AIR sensor, an infrared break-beam sensor from AdaFruit, can detect any opaque object that blocks the IR beam. When obstructed, the sensor is triggered and will record the relative and absolute timestamps to a text file. The absolute timestamp refers to the actual time the trigger took place whereas the relative timestamp is relative to the individual video files. This allows for synchronisation of the AIR sensor and video camera, enabling evaluation whether events were false triggers or not. Besides evaluating the performance of the IR sensor, the video camera also served as a medium for computer vision models. The video camera footage was annotated appropriately and acted as ground truth data for evaluation of the AIR sensor and other models.

Figure 4.1: View from the video camera inside the nestbox

A fully working physical prototype has not been pursued, but rather theorized on how eventually such a product performs and, most importantly, how much power it consumes. All models are simulated locally rather than in the field, where a detection algorithm has been configured and tested on the data-set from video camera footage. The results, such as model power consumption and accuracy, were calculated or measured after and during simulations. Different detection methods vary in size and complexity, which influences the amount of computational power required to detect if a bird is present within an image. To test the potential of embedded AI to improve wildlife capture performance within a low power consumption system, different computer vision

models were created. Three different methods were tested for their usability to work as embedded AI in wildlife camera traps: two different ML models and one much smaller model with the intention of being the most basic forms of computer vision. Besides training ML models that can detect birds, the more simplified version of this helps in finding a truly low-power detection mechanism that can essentially replace the IR sensor in front of the nestbox.

One thing that does need to be kept in mind is the fact that this is experiment is placed in a highly controlled environment from the camera being placed inside a nestbox. The only thing that moves or passes in front of the camera are exclusively birds. This limits the problems of having to deal with obstacles that wildlife cameras often face in nature.

4.2 Preparations

4.2.1 Annotation

The video footage was annotated based on the presence of a bird, where frames containing a bird were tagged with a certain label. These annotations were used as ground truth data and were adapted to fit with the evaluation metrics later on. Four different labels were used as it was necessary to distinguish between the background and three different bird positions for the evaluation metrics. These labels were: Bird_enter, Bird_exit, Bird_in_nest, and Empty. The act of entering the nestbox was annotated from the moment a bird was visible at the entrance until it touched the ground/nest. Vice versa for exiting, where the annotation started from the moment the bird left the ground until it had exited the nestbox and was not visible anymore. The Bird_in_nest label was used when there was any bird (partially) visible in its nest and was touching the ground. The presence of the fourth label for empty images was useful for training a model to recognise and learn the background. However, this could still reasonably be filtered out from the annotation file without a necessity to actually label it.

Annotating the video footage is done in Label Studio, which is an open-source data labeling tool for labeling and annotating all kinds of data. It allows for easy annotation on videos and can export all annotations into a single JSON file. This file contains all key frames along with a label, timestamp and frame number. These is then be read by a python script and used for evaluating the performance of all models.

4.2.2 Evaluation metrics

As previously explained in Chapter 2, four different evaluation metrics were used; TP, FP, TN and FN. These evaluation metrics must be defined in a way where it fits appropriately in the current experiment and measure the performance of the IR sensor and the camera. Because the camera and IR sensor measure different things, the evaluation metrics have to be defined in specific manner such that they match:

- True positives occur when the sensor was triggered and the camera annotations confirmed that a bird indeed had entered.
- False positives occured when the sensor was triggered, but the camera annotations showed a bird exiting the nestbox instead of entering, or did not show a bird near the entrance at all.
- True negatives were the moments where the sensor was not triggered and no bird was present at the entrance, or the bird present was exiting the nestbox instead of entering.
- False negatives occured when the camera showed that a bird did enter the nestbox, but without it triggering the sensor.

The same evaluation metrics were used for the AIR sensor and computer vision models, enabling a performance comparison between machine learning methods and the traditional AIR sensor alone. It should be noted here that in case of a true negative it is unclear when a true negative precisely starts or ends. Time windows are used to help quantify the amount of true negatives, but this will also have influence on the other evaluation metrics as well. With this technique, a sensor or model was evaluated for each time window and results in outcome of one of the 4 evaluation metrics. This means that the evaluation metrics were again changed slightly in order to extract a single outcome from each time window.

- True positives occured when there was at least 1 frame/timestamp where the sensor got triggered and at least 1 frame/timestamp where the annotations confirmed that a bird had indeed entered.
- False positives occured when there was at least 1 frame/timestamp where the sensor got triggered, but the annotations did not show a bird entering the nestbox.

- True negatives occured when there were no frames/timestamps where the sensor got triggered and the annotations confirmed that there was no bird entering the nestbox.
- False negatives occured when there were no frame/timestamps where the sensor got triggered, but the annotations showed that a bird had entered the nestbox.

The CV models required an input in the form of a frame and resulted in a prediction on said frame as output. The AIR sensor followed a different approach where it continuously read IR levels and did not rely on an input frame. The sensor saved a timestamp of the exact moment where the IR light level breached the threshold instead of evaluating a video frame. Hence why the evaluation metrics included both frame/timestamp in case of evaluating either a CV model or the AIR sensor.

This time window needs to have an appropriate length in order not to distort the results from the evaluation metric. For example, if the time window was a single frame long, the amount of true negatives is drastically increased compared to the other metrics and increases the accuracy value to almost a 100%. The reason for this is because there are more than 100.000 true negative frames in the video footage and only a couple thousand frames worth of entrances/exits. Using longer time windows will solve this problem by looking at greater intervals with uninterrupted true negatives. However, using a time window that is too long might result in missed events. In a time window of 1 minute, a bird could enter the nestbox multiple times, but still count as only 1 true positive (or false negative). From the video footage, the average time for a bird to enter the nestbox and land in its nest was found to be around 1 second. With the fact that an event can overlap multiple time windows, the time window is chosen to be 2 seconds (double the average) in order to reduce the chance of an event overlapping more than 2 time windows. Thus, this time window counts as a single metric outcome and gives the ability to quantify the amount of true negatives.

From these evaluation metrics, the accuracy, precision, and recall can be defined in a similar way that apply to this project.

- Accuracy refers to the ratio of correct detections of birds entering (TP) and correct non-detections of birds not entering (TN) compared to all (in)correct detections (TP, FP) and non-detections (TN, FN).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Precision is the ratio of correctly identified birds entering (TP) compared to all instances where the sensor was triggered (TP, FP).

$$Precision = \frac{TP}{TP + FP}$$

- Recall is defined as the ratio of correctly identified birds entering the nestbox(TP) compared to all birds that have entered the nestbox (TP, FN).

$$Recall = \frac{TP}{TP + FN}$$

- The F1-score stays the same, being the harmonic mean of precision and recall.

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Recall and precision are very important, as it shows how sensitive a model is towards detecting a bird entering the nestbox and how 'correct' the model is towards the amount of false triggers. Both need to be as high as possible in order to not confuse a bird sitting in its nest or exiting the nestbox as a bird that is entering the nestbox. The IR sensor is expected to have a very high recall (90+%) and low precision (50%). This is because the IR sensor is very sensitive and will pick up any movement from both cases where a bird enters or exits the nestbox, but falsely triggers half of the times because of the equal distribution of enters and exits. It is further expected that the computer vision models will generally have a lower recall value as they are much more complex and will have more trouble to detect the presence of a bird to begin with. On the other hand, these models might result in a higher precision due their capability of differentiating between an entering or exiting bird and decreasing false positives.

4.2.3 Dataset

A dataset was needed to train the ML models. This dataset was collected from the video camera in the nestbox that recorded continuously each day for one week at the start of May. The camera only filmed during the day, when the birds were active, and created 15 minute long videos rather than one large video file. Figure 4.2 shows

bird activity over time, indicated by the number of triggers that were recorded by the AIR sensor from each video. Red lines indicate time frames when the camera did not collect footage, which happened mainly during the night. The camera filmed for 8 days, resulting in more than 400 15-minute videos, the equivalent of 100+ hours of video footage.

Annotating 100 hours worth of video was not realistic for this project and was reduced to 10 videos that added up to 2.5 hours total. This selection was based on the amount of times the IR sensor was triggered for each video. Videos with high amounts of activity were chosen for annotation due to it being more efficient compared to the use of videos with low- or no activity. This method of filtering videos did affect the kind of observable bird behaviour within the data-set. In videos where a bird enters and exits the nestbox multiple times, it is likely because the bird is searching and bringing back food to feed the chicks inside the nest. By the necessary omission of low activity videos, other behaviour which occur when the birds are resting might be overlooked . For example, when the eggs had not yet been hatched or during the night when the birds are asleep. The night of day 2 in Figure 4.2 was monitored, but resulted in no activity at all.

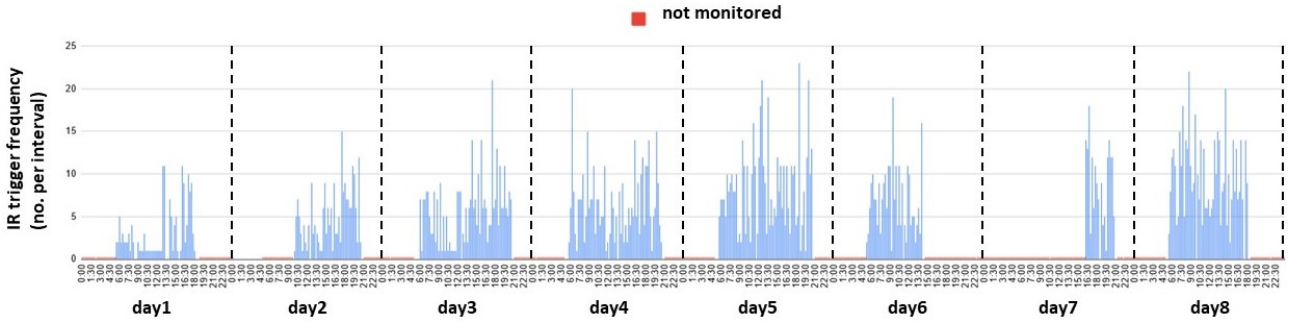


Figure 4.2: Amount of activity every 15 minutes during the full week of monitoring. Blue represents the number of triggers and red indicates that the birds were not monitored and the camera was not recording.

After annotating the videos, 76 Bird_enter events, 74 Bird_exit events, and 96 Bird_in_nest events were found. These events were spread out over all videos and resulted in a total number of 2111 Bird_enter frames, 3347 Bird_exit frames, 52419 Bird_in_nest frames and 185090 empty frames (Figure 4.3). All annotated frames that were either enter or exit were used for the data-set. However, using more than 200000 nest and empty frames would require a lot of storage because each frame is stored as a 1MB PNG image, resulting in more than 200GB worth of data. On top of already having stored the video files themselves, this was not feasible within the current setup and available memory storage. Both the Bird_in_nest frames and empty frames were reduced to a much smaller amount of images due to these resource constraints. The amount of Bird_in_nest frames and empty frames were greatly reduced to a total of 2500 images each, roughly the average of Bird_exit and Bird_enter. This reduction of the data-set was done randomly to avoid creating a bias. Having hundreds of empty frames instead of almost a hundred thousand empty frames did not really change the data-set, but did influence the training aspect for computer vision models. All empty frames were practically identical, but if the number of empty frames would be significantly higher than the non-empty frames, it would hamper training of the ML model in learning to identify a bird.

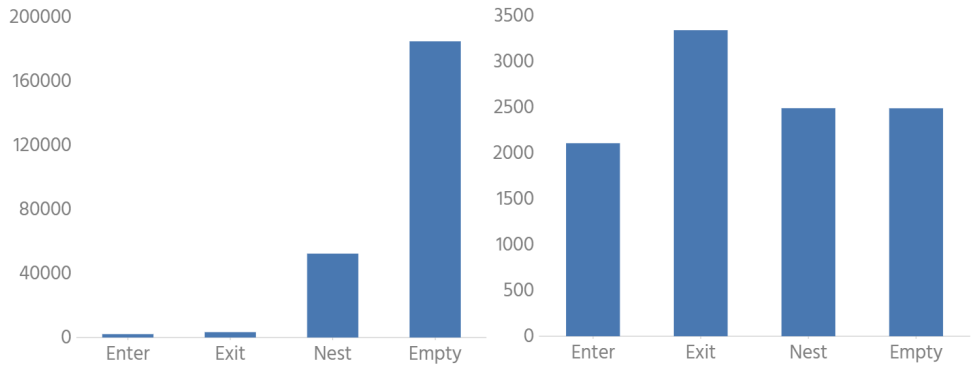


Figure 4.3: Distribution of labels from annotating video footage before and after reduction of nest and empty frames.

The annotation file was read by a python script which sarched for a frame number and label. This script then proceeded to save all video frames from the video footage that were indicated to be used for the data-set. These frames were then sorted into different folders according to their labels (exit, enter, nest, empty). This resulted

in 4 class folders containing a total of 8421 1080p RGB images, which then could be used for the training of ML models. For the purpose of training ML models, these class folders needed to be divided in 3 other folders: Training, testing, and validation. This was done in a 70%/20%/10% division which resulted in 5895 training images, 1684 test images, and 842 validation images in different folders containing similar amounts of each class. This ensured that different models were trained on the same data-set each time every time.

4.2.4 Power consumption

The measuring of power consumption is a complicated without a physical product. For the IR sensor, the power consumption is found online from the product description. The computer vision models are made in python and therefore are also measured within python. A python library called pyJoules returns the duration and amount of joules used from running encapsulated code. With duration and joules, the average wattage can be calculated over a certain time frame. This value does not represent the total power consumption of the complete trigger for a wildlife camera, but only the power consumed from running the code. This will be included in a final sum in power consumption from various components that are required for a functioning wildlife camera and to run this trigger. This final sum will contain components such as a camera, processing board and memory card.

4.3 Trigger techniques

4.3.1 Software libraries

All programming was done in python, which is widely used for ML purposes. The training and usage of computer vision models in particular can be done with the usage of different libraries such as TensorFlow, Keras, and pyTorch, which are user-friendly, open-source software libraries that provide a Python-interface for machine learning and artificial intelligence. These libraries can be used to customize, train, and test models which can also be evaluated be the same libraries. This makes it easier to track experiments, manage data and analyze training process of ML experiments. The evaluation of performance is then based on different evaluation metrics such as accuracy, precision, recall, and the f1-score, which is the harmonic mean of precision and recall.

4.3.2 Detection method

There are plenty of computer vision techniques that can be applied to detect, identify, segment or classify an object in an image. In table 4.1, an overview of different models can be seen along with their method of detecting, performance and footprint. These techniques all have similar use cases, where some can be very compact with low computational power. When looking for a low power system, the algorithm should not use any resources for creating data that serves no purpose in wildlife cameras. In other words, knowing what needs to be measured is the first step to creating an efficient AI algorithm.

With the goal of creating an algorithm that can simply detect the presence of a bird on camera, it is not essential to know exactly where the bird is in frame. When 'detecting' a bird, it is meant that it is only necessary to look if there is a bird present somewhere on camera. This means that the algorithm should not do anything more than simply knowing when an animal is on screen and activating the trigger. These conditions are perfect for image classification, which only requires a label along with the image instead of bounding boxes or segmented images. This is enough information to differentiate between the 4 classes alongside a lower computational power requirement. This is because image classification generalises more compared to object detection, as it looks at general features rather than countless little details for each image. A similar outcome is found when looking at Table 4.1, where most of the smaller models also use image classification as method of detection. Another advantage of image classification is not having to use bounding boxes, which decreases the amount of work for annotating videos. Manually selecting bounding boxes requires more time and precision compared to labeling images.

In case of image classification, there are a lot of widely used models that are publicly available. Matlab provides 20 'plug-and-play' pre-trained image classification models (Figure 4.4). This graph shows the relative prediction time using GPU against the accuracy of each model where the bubble size indicates the model size. The ideal model should be a small bubble towards the upper left corner with very low GPU usage, high accuracy and a small model size. In Figure 4.4, SqueezeNet is found to be the best competitor in terms of being compact and a very low power model, but with the downside of having a lower accuracy compared to other models.

In Table 4.1, it can be seen that most image classification models are trained on Deep Neural Networks (DNN) or Convoluted Neural Networks (CNN), which fall under the same DNN umbrella term. CNN models are great at finding patterns and are very compact. The neural networks chosen that is most likely to meet the criteria for the objective in this thesis is SqueezeNet (Figure 4.4, Table 4.1). Being the smallest network, SqueezeNet was chosen as a CNN image classifier.

Neural network models					
Model name	Goal	Method	Performance	Footprint	Publication
MobileNet-V1	Efficient CNN for Mobile Vision	OD, IC	64.5% accuracy	3260K parameters, 567.5M operations	Howard *et al.* (2017) [22]
AttendNets	Tiny DNN for Edge via Visual Attention Condensers	IC	71.7% accuracy	782K parameters, 191.3M opperations	Wong *et al.* (2020) [51]
YOLO v4-Tiny	Capture images when an animal passes camera trap without PIR	MD, IC	Processing rate / fps: 5.24	MD and IC: 0.4W	Riechmann *et al.* (2022) [39]
MOG	Camera that can monitor endothermic and ectothermic animals	MD	99% accuracy	Device Avg.: 2.65W - 4.13W depending in light levels. MD Algorithm: 1.1W	Corva *et al.* (2022) [14]
HOG	Human detection and tracking	OD	up to 89.59% accuracy	Not found	Seemanthini and Manjunath (2018) [24]
Tiny-YOLO-V2	Real-Time DNN Object Detection for Constrained Environments	OD	BFLOP/s: 6.97. mAP: 57.1%. FPS: 15.5	Model size: 63.4MB	Fang *et al.* (2020) [17]
Tinier-YOLO	Real-Time DNN Object Detection for Constrained Environments	OD	BFLOP/s: 2.563 mAP: 65.7% FPS:25.1	Model size: 8.9MB	Fang *et al.* (2020) [17]
AlexNet	CNN model that has very few parameters while preserving accuracy	IC	Top-1 and Top-5 ImageNet accuracy: 57.2%, 80.3%.	Model size: 240MB. Compressed: 6.9MB	Iandola *et al.* (2016) [23]
SqueezeNet	CNN model that has very few parameters while preserving accuracy	IC	Top-1 and Top-5 ImageNet accuracy: 57.5%, 80.3%.	Model size: 4.8MB. Compressed: 0.47MB	Iandola *et al.* (2016) [23]
SqueezeDet	Small, Real-time object detection for automous driving	OD	BFLOP/s: 9.7 mAP: 76.7% FPS: 57.2	Model size: 7.9MB, Activation memory: 117.0MB	Wu *et al.* (2016) [53]
TinySSD	Tiny CNN for real-time embedded object detection	OD	mAP: 61.3%	Model size: 2.3MB, 1.13M parameters, 572M opperations	Wong *et al.* (2018) [52]
EfficientDet-D0	Scalable and efficient object detection based on EfficientNet	OD	AP: 33.8%, AP50: 52.2%, AP75: 35.8%, BFLOP/s: 2.5	3.9M parameters.	Tan *et al.* (2020) [45]

Table 4.1: Neural network performance and footprint overview. *OD = object detection, IC = image classifier, MD = motion detection

4.3.3 CNN machine learning

The introduction to Convoluted Neural network (CNN) caused a major increase in classification accuracy compared to previous methods. These networks are especially good at processing grid-like data, such as time-series, audio and images. The architecture of a CNN model is created out of multiple layers of 'neurons', which each have trainable parameters and use a mathematical operation called convolution instead of general matrix

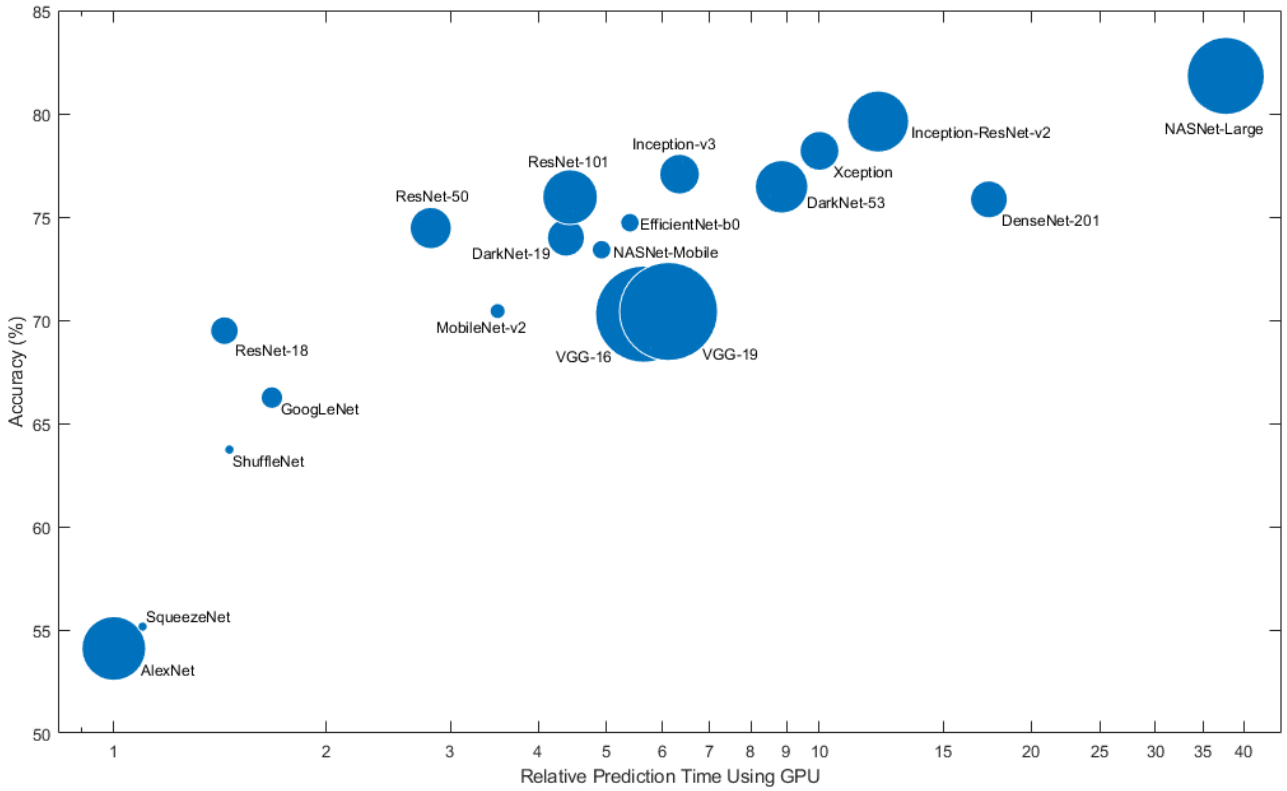


Figure 4.4: Plot of classification accuracy versus relative prediction time of different neural networks. Marker areas are proportional to neural network size on disk. [3]

multiplication. Creating a CNN model often includes the usage of several basic layers (See Figure 4.5), where the most used layers are as follows:

- **Input layer** This is a layer of neurons representing the raw data of an image that is to be classified. This matrix has three dimensions: width, height, and colour. Width and height simply specify the size of the image in pixels, whereas colour represents one or more channels for each colour value of the pixels in the image. With RGB images, there are three colour channels with 1 value for each channel (namely red, green and blue).
- **Convolution layer** In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size (e.g. 3x3 pixels). By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter. The convolution layer applies these filters to extract spatial features such as edges, corners, and textures.
- **Activation function** This is usually placed immediately after the convolution layer and serves as the last component of this layer. It applies a non-linear activation function (e.g., ReLU, Softmax, Sigmoid) to introduce non-linearity into the model. This helps the model to learn more complex patterns during training.
- **Pooling layer** In most cases, a convolution Layer is followed by a Pooling Layer. This layer aims to decrease the size of the output feature map from the convolution layer in order to reduce computational costs. This reduces the amount of connections between layers and independently operates on each feature map.
- **Flatten layer** The flatten layer is used to convert the multi-dimensional feature maps into a one-dimensional vector, which can then be fed into fully connected layers for classification.
- **Fully connected layer** The Fully Connected (Dense) layer consists of the weights and biases and is used to connect the neurons between two different layers. These layers are usually placed after several convolution and pooling layers before the output layer and form the last few layers of a CNN Architecture.
- **Output layer** The output layer is the final layer of the CNN model. It can consist of one or more neurons, depending on the number of classes. It applies an appropriate activation function (e.g., Softmax for multi-class classification) to produce the final output probabilities.

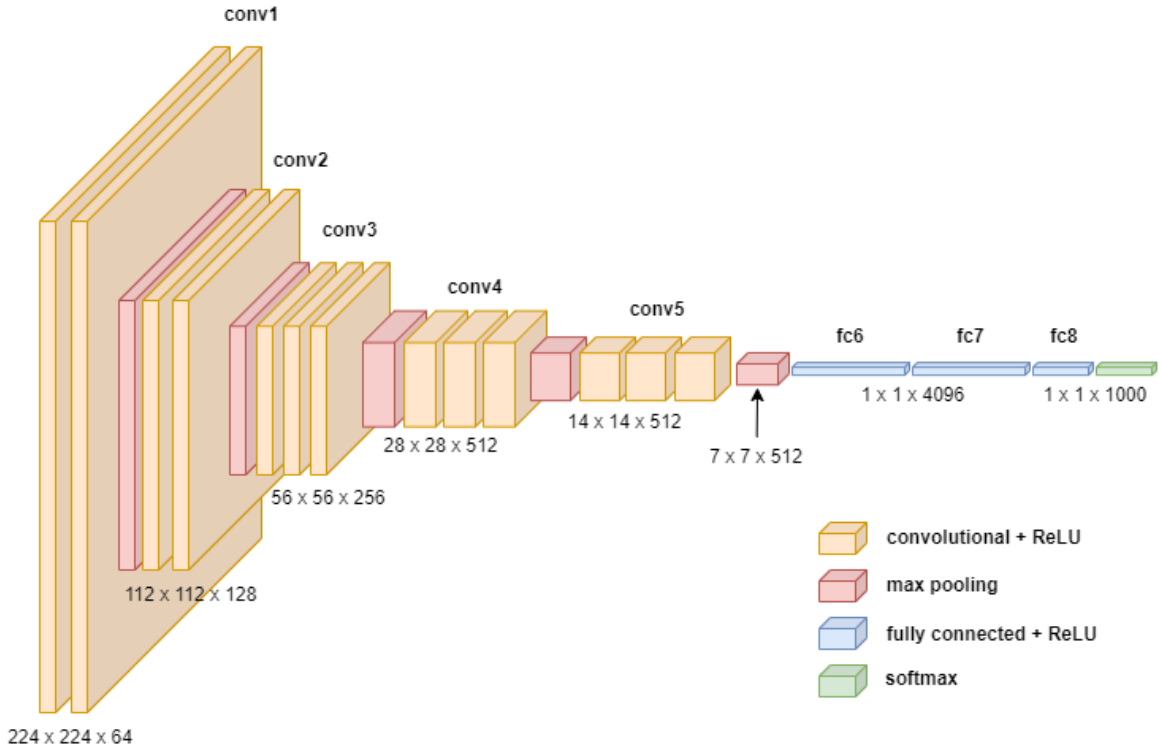


Figure 4.5: CNN model architecture

Another important tool that is used during training, but is not considered to be a layer, is the usage of optimization algorithms. These can be applied during the back-propagation process to update weights based on computed gradients. Optimization algorithms help to find the optimal values for parameters by adjusting them towards the steepest gradient descent. Adaptive Movement Estimation (Adam) and Stochastic Gradient Descent with Momentum (SGDM) are both widely used for optimisation in CNN training.

4.3.4 Low power detection method

It is expected that an AI image classifier will likely consume more power compared to the IR sensor and will not be able to replace IR sensors without lowering the lifetime expectancy. The camera necessary for image classification alone consumes more power than the IR sensor. A simple motion detection model has similar function compared to the IR sensor, which activates when a certain amount of movement is measured. Such a motion detection model will likely require less power compared to an image classifier. The goal is then to see if this minimal power motion detection model could replace the IR sensor based on power consumption and performance. A general problem with motion sensors is differentiating between object and background. However, this is not a problem within the experiment due to the static background that does not change or move.

Chapter 5

Methods

5.1 Detection of movement

5.1.1 Creating motion detection

Open-CV is a useful python library that allows the user to work with video files, which is necessary for constructing a motion detection model for a video camera. This tool allows the extraction of individual frames from videos in an image format. The idea of a motion detection model is to find and determine if there has been enough movement between two frames and the difference between pixel values. If a significant amount of movement has been found, the motion detector is activated.

The first step to create such a model, is to extract a video frame and prepare for pixel value comparison against the following frame. This video frame was resized to a smaller image of 500x280 pixels instead of the original size of 1920x1080. The image was then converted from RGB image to gray scale, where each pixel received a single brightness value between 0 and 255. The last step was to smooth the image with a Gaussian blur which filters out movements from very small objects. This frame got temporarily stored for comparison once the following frame also has been prepared in the same way.

Once two consecutive frames have been prepared, the absolute difference in pixel values was calculated between the temporarily stored frame and the current frame. This was done for every single pixel between these frames, where the the absolute difference between each pixel was translated to a new frame. This frame showed the amount of difference between both frames in terms brightness. Bright areas refer to a higher difference in pixel values between the two original frames and are the result from movement. A threshold filter was applied to this frame that set every pixel to either black or white instead of the gray scale gradient of 0-255. This threshold was set to a pixel brightness value of 25, meaning that each pixel higher than 25 got rounded up and changed to a white pixel (255). Pixels with a lower brightness value than 25 were rounded down and changed to a black pixel (0). This resulted in a black and white image where movement between two consecutive frames was projected in white pixels (see Figure 5.1). This entire process was then repeated for each frame in the video, where each time the current frame got temporarily stored in order for it to be compared again to the following frame (see Figure 5.2).

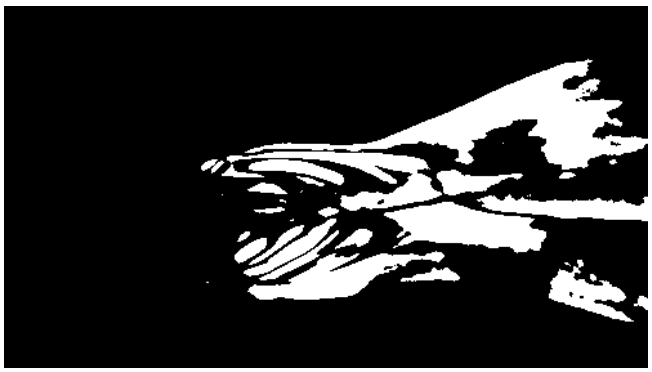


Figure 5.1: Perspective of motion detection model. White pixels show the difference in brightness between two consecutive frames that exceeded the threshold filter.

After completing the visualization of movement between two frames, the motion detection model needed to be finalized with optimal variables for triggering. Three variables were found to be important to optimize the motion sensor: threshold value, frequency of detection, and the application of a cool-down timer. An

appropriate threshold value was necessary in order to determine the sufficient amount of movement necessary to decide if a bird had entered the nestbox or not. Furthermore, the video footage was recorded at a frequency of 30 frames per second. Running the motion detection model at 30 fps seemed unnecessary, as no bird will enter or exit the nestbox under 33ms. Because the model would likely consume a lot more power when running at such a high frequency, lower frequencies were tested to reduce power consumption. Finally, the effect of a cool-down timer was explored as this was found to be present in the AIR sensor.

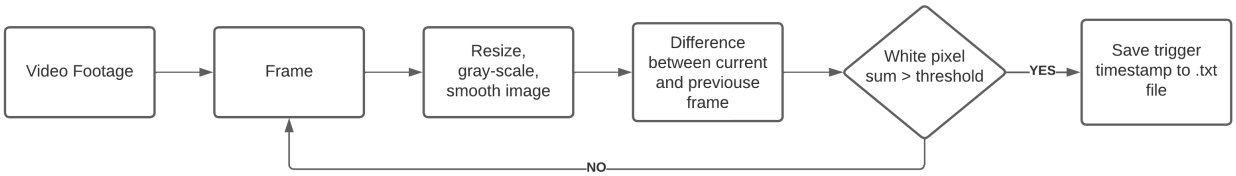


Figure 5.2: Flowchart of motion detection process.

5.1.2 Threshold value

Once the sum of white pixels in the constructed difference frame (Fig 5.1) exceeds a certain threshold, the motion detection model got triggered and saved the timestamp to a text file. This threshold value looks at the amount of white pixels within the frame, which was defined as a percentage of white pixels (movement) within the frame. To find the ideal threshold value, a small experiment was conducted on the nestbox data-set, where the motion detection model wast tested with different threshold values.

5.1.3 Detection frequency

The motion detection model is capable of keeping up with real-time motion detection on 30 fps videos. Running at full throttle, it even achieved motion detection at rates of 60 fps. Despite the model being small, it still consumes more power than needed. The model measured movement for each frame from the video file, which was recorded at 30fps. Similar to finding out what the optimal value is for the threshold value, another experiment was conducted to see if running the motion detection model at lower frequencies would indeed decrease the amount of power consumption. The goal of this experiment was to find the lowest detection frequency without significantly decreasing the performance of the model.

5.1.4 Cool-down timer

From reviewing the video footage, it was found that the AIR sensor uses a cool-down timer. This cool-down timer would either disable the writing of a timestamp or completely stop the sensor from working after every time the sensor got triggered for a small amount of time. During the annotation it became clear that for every trigger within 10 seconds of initial activation, the IR sensor would not detect the event. There were several moments where a bird that was detected entering the nestbox, was detected again when exiting the nestbox 11 seconds later. This provided sufficient evidence that a cool-down timer of roughly 10 seconds was used by the AIR sensor. To observe the effects of a cool-down timer, a third experiment was conducted with the motion detection model equipped with a cool-down timer of different lengths.

5.2 SqueezeNet

5.2.1 What is SqueezeNet?

SqueezeNet was chosen as an image classification model due to its small size and low relative prediction time using GPU compared to other existing models. This model classified individual frames into 4 different possible classes: Enter, Exit, Nest, and Empty. SqueezeNet is a pre-trained classification CNN model that has a total of 18 convolution layers with a model size of 5MB and contains 725K parameters [23](see Figure 5.3). With transfer learning, layers of the pre-trained neural networks are modified to obtain better accuracy for classification in a different training data-set. In other words, it is a machine learning technique where a model trained on one task is re-purposed on a second related task. A pre-trained model is often trained to recognise generic patterns or features, which is a great start for training a new model.

Figure 5.3: SqueezeNet model architecture

5.2.2 SqueezeNet setup

The SqueezeNet model was directly used from the Matlab Deep Network Designer tool package. It is practically a plug-and-play model where there are only two layers that needed to be changed for the model to work. The very last convolution layer has a weight- and bias learn rate factor, which had a value of 1 by default and was set to a higher value of 10 to increase the learning curve. This filter size in this layer was set to 1x1 instead of the default filter size of 3x3. The last classification layer was set to a 1000 classes by default, which should be changed to the correct amount of classes. However because this value could not be manually changes, the layer was replaced in its entirety by the exact same classification layer in order to set it to auto. This allowed the model to make predictions for all 4 classes that are found in the data-set.

Once the model had been created, the data-set containing the training, test, and validation folders was uploaded. Because the data-set had already sorted into these three folders, Matlab automatically recognized the correct data-subset for each class. The last step before training the model was to specify its training parameters (also known as hyper-parameters) appropriate for this data-set. In the training setup, Matlab provided the top 5 variables which are frequently used and altered for training. Only these 5 hyper-parameters were adjusted such that it fits the data-set size.

The training process used Adaptive Movement Estimator (Adam) as the training optimizer/solver, as this was found to be the most effective for small CNN models. Because this model was trained through transfer learning, only a small amount of epochs are required for the training process. The training ran for 10 epochs, where the validation data was calculated once every epoch with a validation frequency of 8. The initial learning rate was set to a smaller factor of 0.0001 (ten times smaller than default) to slow down learning in the transferred layers. The final hyper-parameter is the mini-batch size, which refers to how many images from the data-set to use in each iteration. To ensure that the whole data set was used during each epoch, the mini-batch size was set to evenly divide the number of training samples over each validation. With 7312 observations in the training data-set, the mini-batch size is set to 914. It was important that the amount of observations was exactly dividable by the validation frequency in order to have a whole number for the mini-batch size. If the ideal mini-batch size were to be a decimal number and got rounded to the nearest integer, the training process would result in validation tests not occurring exactly once every epoch.

5.2.3 Training

During training, Matlab provided an accuracy and loss graph that indicated the performance of the training process (Figure 5.4). This graph can give insights on why a model may not be performing very well. In some cases a model can suffer from under-fitting or over-fitting, where the model can become too generic or too specific on the training data. Due to the appropriately chosen hyper-parameters, the training process went well and resulted in a final validation accuracy of 95.05%. Once the training was finished, the model was exported to Matlab's workspace. Here it could be used freely, where the classification model can predict which of the 4 classes belongs to an image.

5.3 Custom CNN model

5.3.1 Minimal CNN architecture

SqueezeNet is made out of 18 convolution layers and contains 725K parameters. This is already very small, even when compared to most other small-scaled models. However, this could be further simplified by removing layers until the most basic form of a CNN model is left. This was possible because this experiment is on a much smaller scale with only a total of 4 classes, which is a lot smaller compared to the classification of 1000 classes for which SqueezeNet was originally designed for. The most simple CNN structure consists of just a couple of convolution layers, pooling layers and a fully connected layer at the end. Besides minimizing the amount of layers in the model, the most crucial size reducer was to down-size the input layer size from 227x227 to a much

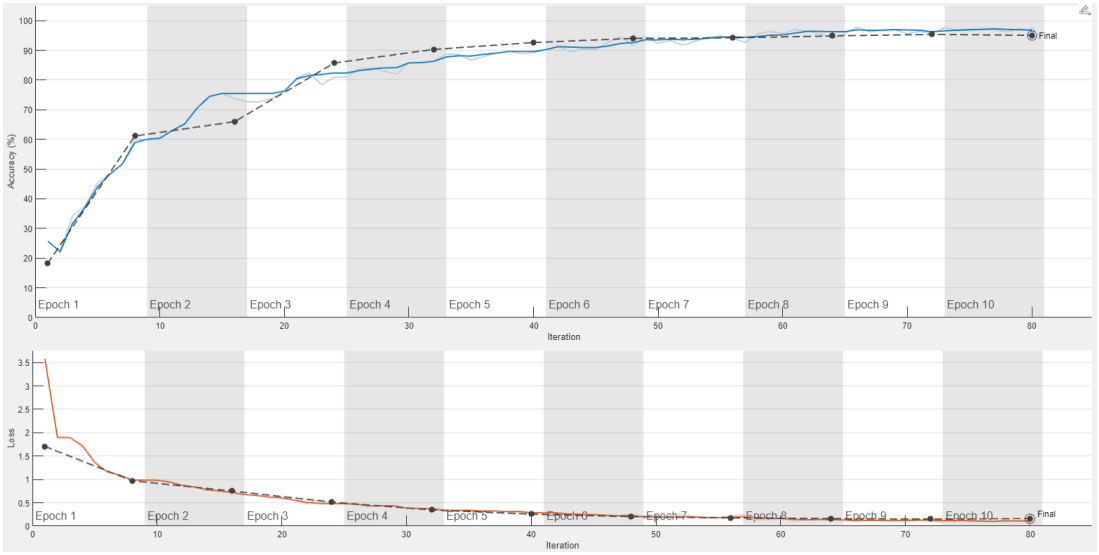


Figure 5.4: SqueezeNet training process with accuracy (Blue) and loss (Orange)

smaller size of 32x32. With these techniques, a new CNN model was created with a total amount of 36420 parameters (See Figure 4.5). This custom CNN model is 20x times smaller than SqueezeNet in model size.

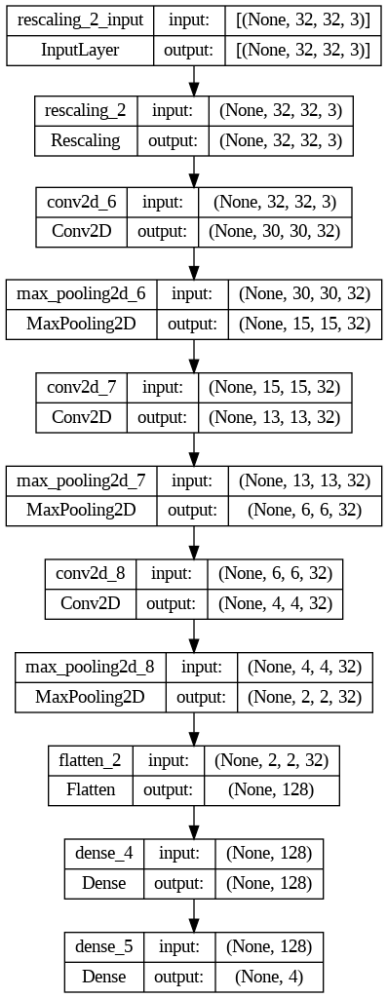


Figure 5.5: Custom CNN model architecture

5.3.2 Custom CNN model setup

The model was designed and created in python with the usage of the TensorFlow library and Keras utilities. The first step was to import the image data-set from directory for training, validating, and testing. Here, the image width and height were defined and set to 32x32. A batch size of 20 was chosen as this was said to be the standard for small CNN models by TensorFlow. The model itself was created with the sequential function that groups a linear stack of layers into a TensorFlow model with training/inference features. This was done according to Figure 5.5 where once the model was created, it was further configured with a loss function and metric in the compile function. In this function, the Adam optimizer was used along with a sparse categorical cross-entropy loss function and accuracy as metric. After the configuration of the model, the model was complete and ready for training.

5.3.3 Training

The model was trained on both the train data-set and validation data-set with a total of 20 epochs. The amount of epochs was increased compared to the training of SqueezeNet because it was found to be more effective. To avoid continuously re-training the model when running the script, the fully trained weights and biases were saved in a separate file and could be called to load the model back in again. This could then also, in a similar way, be exported and loaded onto other platforms or embedded systems. Furthermore, an early stop callback was created to stop the model during training if it were over-fitting. This allowed the model to train just enough that it did not cause over-fitting. Once the training was finished, the model is complete and fully functional. It can classify images with the built-in prediction function that outputs the confidence value of each class for an image. With the usage of the evaluation functions and the Matplotlib library, the accuracy and loss for both the training and validation were plotted in Figure 5.6. The final validation accuracy after training came out to be 88.89%.

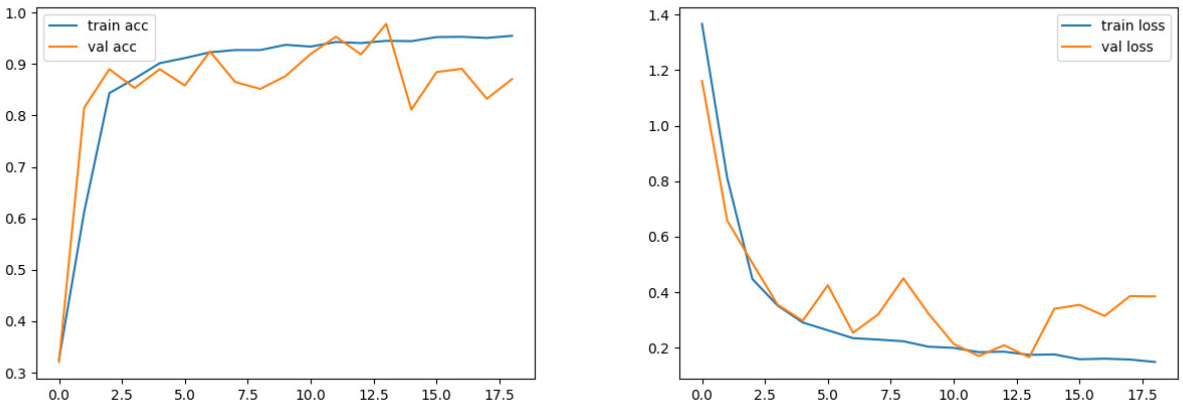


Figure 5.6: Custom CNN accuracy and loss during training and validation

5.4 Improvements

5.4.1 Rate of detection

Similar to the motion detection model, both image classification models could be used to perform inference on every frame. Doing predictions at 30 fps was very likely to require a lot of power without much change in performance compared to running the model at a somewhat lower frequency. Both models were evaluated based on time windows of 2 seconds and thus it was decided to perform image classification at least once every 2 seconds, once for each time window. Rather than doing 30 classifications per second, 0.5 classifications per second was the required minimum in order to have each classification model evaluated with time windows. It was found that the time of a bird entering the nestbox was 1 second long on average. With only 1 prediction per 2 seconds, some events could be missed. This would likely result in a larger amount of false negatives. Using other frequencies, that are slightly higher than the minimum, could be tested in order to find an optimal value that maximizes performance while minimizing power consumption. However, such an experiment was not conducted due to time constraints, but mainly because of the fact that there is a more sensible method of deciding how often an image classification model should be used.

5.4.2 Dual trigger

A solution to the problem of finding the most optimal rate of detection, was by using the motion sensor as input rather than using a time interval. Instead of measuring at a certain fps, the image classification models could make predictions on those frames that were found to contain movement. This would not only increase performance, but also reduce power consumption as the image classification would only be used once there is motion rather than every 2 seconds. The motion detection model was expected to have a high recall, which should pick up nearly all true positives with the downside of also resulting in plenty of false positives. The addition of an image classification can filter out these false positives by looking if there was indeed a bird entering the nestbox or not. This is a two-step process where the motion sensor ran continuously and detects movement whereas the image classification model served as a double check. This combination was expected to have the best of both worlds with the high recall value maintained from the motion detector and a high precision value from the double check of the image classification model.

For this to work efficiently and to further improve such a model, the motion sensor that serves as the initial trigger should have the highest possible True Positive Rate (TPR) without resulting in a too large amount of false positives. Even if the classification model could perfectly filter out all false positives, it was preferred to have the image classification model run as few times as possible to minimize power consumption. To optimize the motion sensor, another experiment was conducted to again find the best values for the threshold, detection frequency and cool-down timer.

5.5 Measuring power consumption

To find what model is most suitable for a wildlife camera to last long periods of time, the power consumption of each model was measured as close as possible. These models ran on the CPU of a Lenovo ThinkPad laptop instead of being embedded onto a separate piece of hardware such as an Arduino. A problem with this method is the fact that a laptop also has other applications running in the background that use the CPU and consume power. To estimate the power consumption of the model specifically, the consistent background power consumption was subtracted from total power consumption when running each model. To do this as accurate as possible, a useful tool called HWiNFO was used to find detailed information on the power consumption of the CPU.

This tool provided plenty information on the CPU, where the most important variable was 'CPU Package Power' and is defined as the total power consumed by the entire CPU package. By clicking on the variable, a graph of the current CPU power consumption can be seen. The minimum, maximum and average power consumption were read from this variable and are visible in the graph. The process of reading the power consumption from the model and background applications stayed consistent throughout the measuring of each model. This process consisted of a 30 second measurement of the background CPU usage, followed by the model running, and ended with another 30 second measurement of the background CPU usage. The purpose of the second 30 second window was to ensure that the background CPU stayed consistent throughout measurement. If the average background CPU usage significantly increased or decreased compared to the first interval, the measurement was discarded and measured again once the background CPU usage became stable.

Both the stand-alone motion sensor model and the motion sensor for the dual trigger were allowed to run for a maximum of 30 seconds at real-time speed. This was achieved with an appropriately set wait key such that it performed motion detection on a video as if it were played back at normal speed. This resulted in the measurement of the true power consumption, analogous to where it would be used as trigger in a wildlife camera. For the measurement of the image classification models, both models were allowed to run for a maximum of 60 seconds at full throttle, or until finished. The reason why the image classification models are allowed to go at full speed, is because these models would otherwise be too slow and not accurately measured with the background CPU still being present.

To estimate average power consumption that is analogous to where it would be used as a trigger in a wildlife camera, the amount of triggers that were successfully classified within a certain time was counted. This amount was then multiplied by the trigger frequency to find how long the model would have ran in real time. This was then divided by the amount of seconds the model actually ran, resulting in a ratio between actual run time and theoretical 'normal pace' run time. To calculate the true average power consumption, the average power consumption measured from the actual run time is divided by this ratio to find how much the model would have consumed on average if it were run on its normal speed. This was applied for both the CNN model and SqueezeNet model and for both continuous detection and dual trigger detection.

Chapter 6

Results

6.1 IR sensor

6.1.1 Evaluation

The IR sensor was expected to result in an almost equal amount of true and false positives. However, this proved not to be the case (Table 6.1), i.e. a total of 76 enters and 74 exits were recorded according to the ground truth, but the number of false positives was lower than the number of exits. This originated from the fact that the IR sensor has a 10 second cool-down after it is activated and caused some exits to be missed by the IR sensor, reducing the amount of false positives.

Furthermore, the amount of false negatives was also much higher than expected (Table 6.1). The AIR sensor was expected to result in a low amount of false negatives, as it is a very sensitive sensor. However, an increased amount of false negatives was found to be caused by the method of evaluating the sensor with time windows. There were 76 Bird_Enter events, whereas due to the usage of time windows, a total of 72 + 37 = 109 true positive events are expected for a perfect model.

	Actual	
Prediction	**Positive**	**Negative**
Positive	72	52
Negative	37	3889

IR sensor	
Precision	58.06%
Recall	66.06%
F1-score	61.80%

Table 6.1: Evaluation metrics on IR sensor performance.

With the values from Table 6.1, the precision, recall, and F1-score were calculated. Before conducting the experiment, it was hypothesised that the recall (or sensitivity) would be very high, but this was proven to be incorrect as mentioned above. On the other hand, the precision was estimated to be around 50% as there was an equal division between Bird_enter and Bird_exit, but resulted in a slightly higher value of 58%.

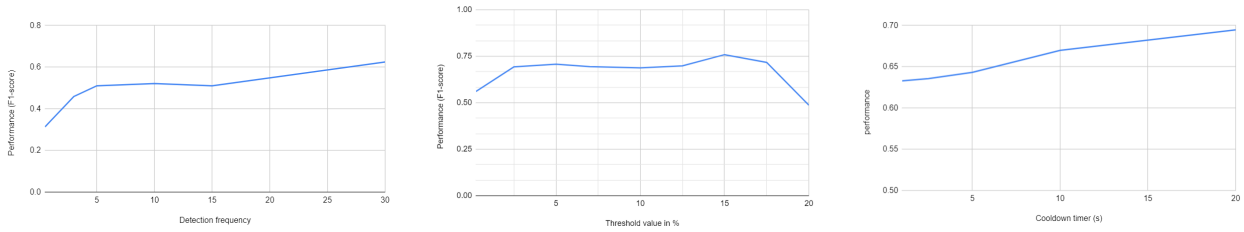
6.1.2 Power consumption

The power consumption was found from the product description on the online Adafruit web-shop [2]. The IR sensor is capable of operating on 10mA at 3.3V or 20mA at 5V, which is a power consumption of 0.033W - 0.1W.

6.2 Standalone motion detector

6.2.1 Configuration

Three separate experiments were conducted to find the optimal values for the detection frequency, threshold value (sensitivity), and the presence of a cool-down timer (See Figure 6.1). To find the best working motion detection model, each variable was chosen in such a way where it maximizes performance. A detection frequency of 30 fps, threshold value of 15%, and a cool-down of 20 seconds were found to be optimal values for each variable.



(a) (b) (c)

Figure 6.1: Graphed F1-scores for optimal detection frequency, threshold and cool-down timer for the motion sensor.

6.2.2 Evaluation

The motion detection sensor was expected to perform equally or even slightly worse compared to the AIR sensor. This reason behind this was that the motion detection sensor was expected to contain a lot of false positives from each time a bird exits the nestbox. However, this was not the case due to the presence of a relatively long cool-down timer (See Figure 6.2).

The optimal motion sensor ran at 30 fps. An alternative setting (5 fps) for the motion sensor was tested in an attempt to decrease its power consumption (See Table 6.3). Below 5 fps, performance would be significantly decreased (See Figure 6.8a). It was expected that lowering the detection frequency would have little impact on performance, with the exception of extremely low frequencies when a bird has enough time to enter or exit the nestbox without being detected. Surprisingly, the performance slightly increased, i.e. resulted in an F1-score of 69.47% (30 fps) and 70.94% (5 fps), a significant improvement compared to the AIR sensor. Furthermore, an AUC ROC curve was created by running the model multiple times at different threshold values, setting the cool-down to 0 seconds and resulted in an AUC of 0.973 (See Figure 6.2). This was necessary as the model would not reach a TPR (true positive rate) of 1 when a 20 second cool-down was applied.

	Actual	
Prediction	**Positive**	**Negative**
Positive	66	15
Negative	43	3926

Motion sensor	
Precision	81.48%
Recall	60.55%
F1-score	69.47%

Table 6.2: Evaluation metric on standalone motion sensor at 30 fps and 20 s cool-down.

	Actual	
Prediction	**Positive**	**Negative**
Positive	72	22
Negative	37	3919

Low-power motion sensor	
Precision	76.60%
Recall	66.06%
F1-score	70.94%

Table 6.3: Evaluation metric on standalone motion sensor at 5 fps 20 s cool-down.

6.2.3 Power consumption

The power consumption was measured for the motion sensor at 30 fps and at 5 fps (See Figure 6.3). Table 6.4 further shows the amount of wattage that was recorded for both scenarios where the sensor was active and when the sensor was idle. This shows that the models used 6-7 Watts on average, whereas the background CPU usage was found to be consistent around 3.5-4 Watts on average. By subtracting the average wattage of the idle sensor from the average wattage of the active sensor, the average power consumption of the motion sensor alone was obtained. This resulted in the average consumption of 3.778 W for the 30 fps motion sensor and 2.155W for the 5 fps alternative. Using a lower frequency did not have a negative effect on the performance of the sensor, but did decrease the power consumption by 43%. Even with the 5 fps model, the power consumption of the current motion detection sensor is still significantly larger than that of the AIR sensor, which was expected beforehand.

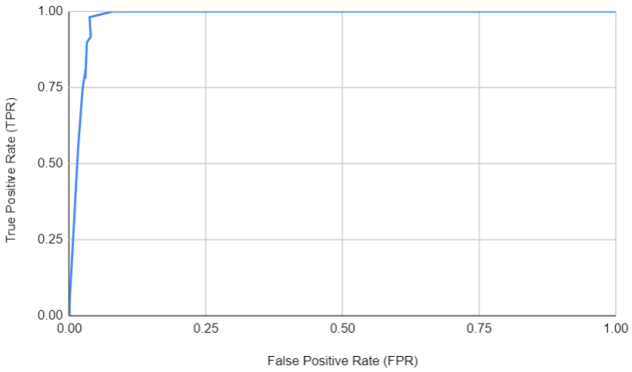


Figure 6.2: Motion sensor ROC curve at 5 fps without cool-down [AUC = 0.973].

(a) Motion sensor at 30 fps

(b) Motion sensor at 5 fps

Figure 6.3: Motion sensor power consumption comparison

Motion sensor at 30 fps		
	Sensor On	**Idle**
Min	5.351 W	2.986 W
Max	9.422 W	4.505 W
Average	7.196 W	3.418 W
Real-time average: 3.778 W		

Motion sensor at 5 fps		
	Sensor On	**Idle**
Min	5.168 W	3.283 W
Max	9.939 W	7.129 W
Average	6.094 W	3.939 W
Real-time average: 2.155 W		

Table 6.4: Power consumption of motion sensor set at 30 fps vs 5 fps.

6.3 SqueezeNet model

6.3.1 Evaluation

After the training was completed, the SqueezeNet model was tested on the test data-set. In Table 6.6, a confusion matrix and classification report are found. This table shows how good the model is at identifying each individual class, where the most important one is the Bird_Enter class. This test resulted in the model having a 92.79% F1-score in being able to detect if an image contains an enter event or not. Furthermore, Figure 6.4 shows a one-vs-rest multi-class AUC ROC curve of the model with AUC = 0.994.

The SqueezeNet model was used to classify every 60Th frame of each video, or in other words every 2 seconds. The image classification models were expected to result in a low recall and high precision. Figure 6.5 shows that this expectation was indeed correct in terms of high precision, whereas the recall was quite similar to what both the motion sensor and AIR sensor achieved. The resulting amount of false positives was very low due to the ability of the image classification model to evaluate and classify an image, which could identify and filter out most false positives.

Besides the effect on false negatives from using time windows, the model resulted in an increased amount of false negatives compared to the AIR sensor. The reason behind this was because of the image classification model being run only every 2 seconds. Because a bird could enter or exit the nestbox in under a second, some events were missed and resulted in either false negatives or true negatives. The overall performance of the SqueezeNet model resulted in an F1-score of 71.96%, which is an improvement compared to the AIR sensor.

	Actual								
Prediction	**Empty**	**Enter**	**Exit**	**Nest**	**Classes**	**Precision**	**Recall**	**F1-score**	**Support**
Empty	495	0	0	4	**Empty**	0.9920	0.9322	0.9612	540
Enter	12	399	1	10	**Enter**	0.9455	0.9110	0.9279	498
Exit	8	31	622	8	**Exit**	0.9297	0.9952	0.9614	590
Nest	16	8	2	473	**Nest**	0.9479	0.9556	0.9517	461

Table 6.5: SqueezeNet model Confusion matrix

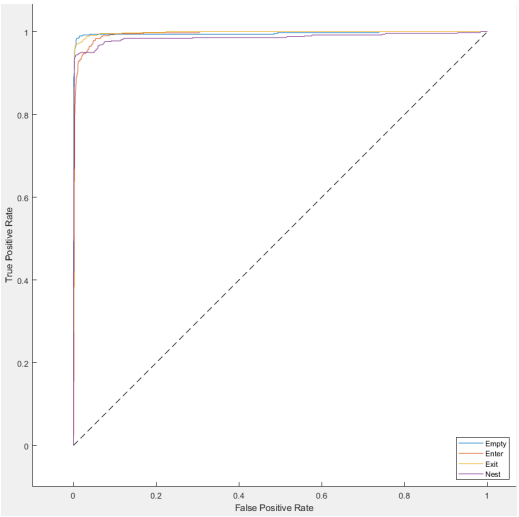


Figure 6.4: SqueezeNet one-vs-rest multi-class ROC curve. [AUC = 0.994]

	Actual	
Prediction	**Positive**	**Negative**
Positive	68	12
Negative	41	3929

SqueezeNet model	
Precision	85.00%
Recall	62.39%
F1-score	71.96

Table 6.6: Evaluation metric on SqueezeNet image classification performance

6.3.2 Power consumption

The SqueezeNet model was allowed to run for a total of 60 seconds, which resulted in a total of 911 images that were classified with an average power consumption of 17.901 - 3.408 = 14.493 Watts. If this model were to be slowed down to only one image classification every 2 seconds rather than 900+ per minute, the average power consumption would down to 0.477 Watts. With this calculation, it was assumed that the model does not consume power whilst idle.

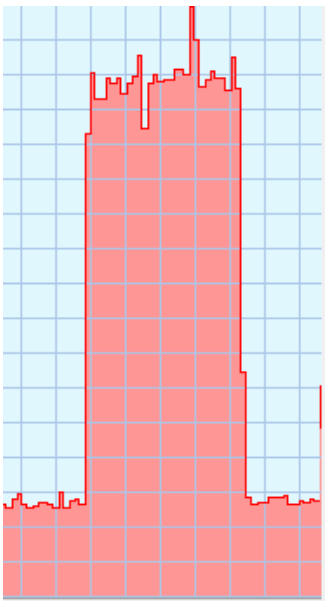


Figure 6.5: SqueezeNet power consumption experiment graph

SqueezeNet model		
	Model On	**Idle**
Min	15.904 W	3.040 W
Max	22.128 W	3.631 W
Average	17.901 W	3.408 W
Real-time average: 0.477 W		

Table 6.7: (SqueezeNet model power consumption table

6.4 Custom CNN model

6.4.1 Evaluation

The custom CNN was tested and evaluated in the same way as the SqueezeNet model. The custom CNN model was first tested on the same test data-set and returned the confusion matrix and classification report as shown in Table 6.8. This table shows how well the image classification model performed, where it achieved an F1-score of 89.69% for the Bird_Enter class. Furthermore, Figure 6.6 shows a one-vs-rest multi-class AUC ROC curve of the custom CNN model with AUC = 0.986. The custom CNN model also performed image classification on every 60Th frame, identical to the SqueezeNet model. The custom CNN model was expected to perform slightly worse than the SqueezeNet model due to its reduction in size and complexity. This was confirmed in Table 6.9, where the custom CNN model achieved an overall F1-score of 70.53%, whereas the SqueezeNet model achieved an overall F1-score of 71.93%.

	Actual								
Prediction	**Empty**	**Enter**	**Exit**	**Nest**	**Classes**	**Precision**	**Recall**	**F1-score**	**Support**
Empty	494	14	12	110	**Empty**	0.7841	0.9900	0.8751	630
Enter	1	387	32	21	**Enter**	0.8776	0.9171	0.8969	441
Exit	2	18	617	7	**Exit**	0.9581	0.9223	0.9398	644
Nest	2	3	8	361	**Nest**	0.9652	0.7234	0.8270	374

Table 6.8: Custom CNN model Confusion matrix

6.4.2 Power consumption

Similar to the SqueezeNet model, the custom CNN model was also allowed to run for a maximum of 60 seconds (See Figure 6.7). The large peak was caused by the model starting up, where the data-set was loaded in and sorted into folders, the CNN model structure got created, and the weights and biases are loaded in. This was

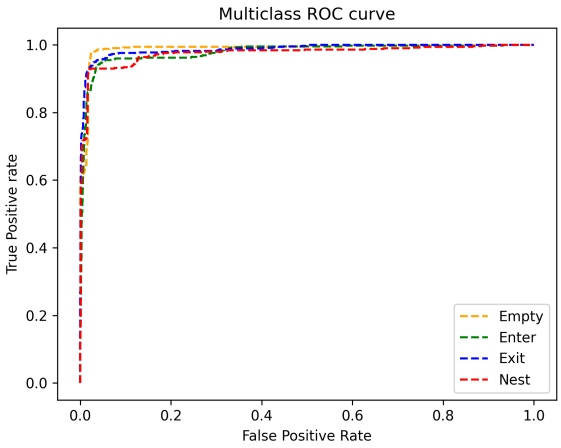


Figure 6.6: Custom CNN model one-vs-rest multi-class ROC curve. [AUC = 0.986]

	Actual	
Prediction	**Positive**	**Negative**
Positive	73	25
Negative	36	3916

Custom CNN model	
Precision	74.49%
Recall	66.97%
F1-score	70.53%

Table 6.9: Evaluation metrics on custom CNN image classification performance

therefore not included in the power consumption measurement of the model, as it does not reflect how much power it consumes when performing image classification.

The custom CNN model was able to perform image classification on 496 images where it used 13.210 - 3.194 = 10.016 Watts on average for a full minute (See Table 6.10). As this model was also doing image classification on every 60Th frame, slowing down the model to its 'real-time' speed would result in an average continuous power consumption of 0.606 Watts. It was expected that the custom CNN model would require less power compared to the SqueezeNet model due to the reduction of model size and amount of parameters. Surprisingly, the more compact CNN model resulted in a higher power consumption compared to the SqueezeNet model.

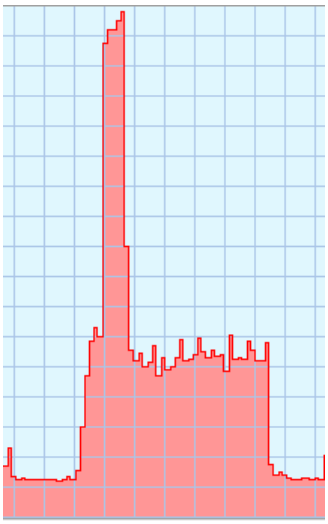


Figure 6.7: Custom CNN model power consumption

Custom CNN model		
	Model On	**Idle**
Min	11.187 W	3.075 W
Max	14.439 W	4.136 W
Average	13.21 W	3.194 W
Real-time average: 0.606 W		

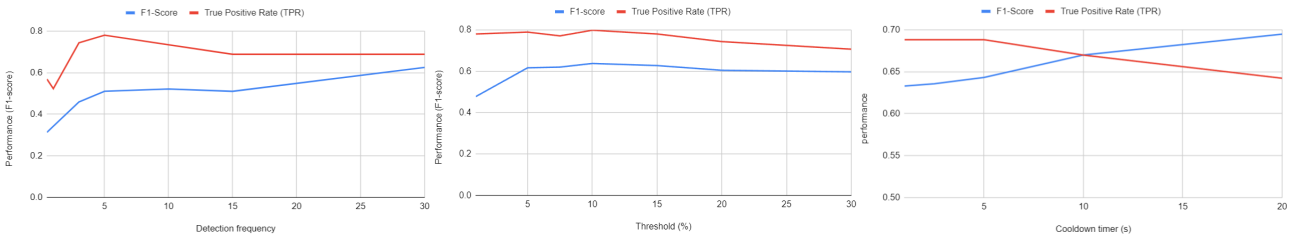
Table 6.10: (Custom CNN model power consumption table

6.5 Improved Motion Detection Sensor

6.5.1 Configuration

Similar to configuring the most optimal variables for the motion sensor, other values were chosen from the same experiments to optimize a newer motion detection sensor. This improved motion detection sensor served as the first step in a dual trigger, where it maximized its TPR to ensure optimal results when combined with an image classification model. This new motion sensor was chosen to run at 5 fps, with a cool-down of 5 seconds and a threshold value of 10% (See Figure 6.8).

A cool-down of 0 seconds would have resulted in a slightly higher TPR, but this was not chosen because of a very important benefit that comes from using a cool-down. The addition of a cool-down helps keeping the amount of triggers to a minimum by blocking the motion sensor from writing trigger timestamps to the text file. This would result in an image classification model having to do predictions on fewer images, further reducing run-time and power consumption. This motion sensor resulted in a total of 149 images from the 2+ hours of video footage that were found to contain significant amounts of motion.



(a) threshold 10% and 5 s cooldown (b) 5 fps and 5 s cooldown (c) 5 fps and threshold 10%

Figure 6.8: Graphed F1-scores and True Positive Rate (TPR) for optimal detection frequency, threshold and cool-down timer for the improved motion detection sensor.

6.5.2 Evaluation

Improved motion detection sensor

The dual trigger motion sensor was also tested to see what was being fed into the image classification models later on. The goal was to have the recall (TPR) as high as possible in order to collect most true positives. Table 6.11 shows the results of running the improved motion detection sensor on the data-set. It did indeed achieve a very high amount of true positives, resulting in a large TPR. The downside of having such a sensitive model is that it does not perform very well overall due to the large amount of false positives. This model had an overall F1-score of 63.74%, which is lower than the original motion sensor but remarkably similar to the AIR sensor.

	Actual	
Prediction	**Positive**	**Negative**
Positive	87	77
Negative	22	3864

Improved motion detection sensor	
Precision	53.05%
Recall	79.82%
F1-score	63.74%

Table 6.11: Evaluation metrics for the improved motion detection sensor performance.

Motion triggered SqueezeNet model

Instead of having SqueezeNet run on every 60Th frame, it ran on all images that were identified to contain significant amounts of movement according to the improved motion detection sensor. Doing this resulted in a significant increase in performance compared to having the model run at a consistent frequency. Table 6.12 shows that the precision drastically increased with minimal loss of recall. This was as expected, as image classification would serve as a double-check and so mostly would improve the input motion sensor. This dual trigger resulted in an F1-score of 87.31%, where the number of false positives was reduced from 77 to 2.

	Actual	
Prediction	**Positive**	**Negative**
Positive	86	2
Negative	23	3939

Dual trigger SqueezeNet model	
Precision	97.73%
Recall	78.90%
F1-score	87.31%

Table 6.12: Evaluation metrics on the improved motion detection sensor triggered SqueezeNet model performance.

Motion triggered custom CNN model

The exact same process was applied for the custom CNN model, where it was fed the output images of the improved motion detection sensor. It was expected that the custom CNN model would achieve similar performances compared to the SqueezeNet model. Table 6.13 shows that the custom CNN model achieves the exact same performance values as the SqueezeNet model. Both models classified the 149 motion images in the same way.

	Actual	
Prediction	**Positive**	**Negative**
Positive	86	2
Negative	23	3939

Dual trigger custom CNN model	
Precision	97.73%
Recall	78.90%
F1-score	87.31%

Table 6.13: Evaluation metrics on the improved motion detection sensor triggered SqueezeNet model performance.

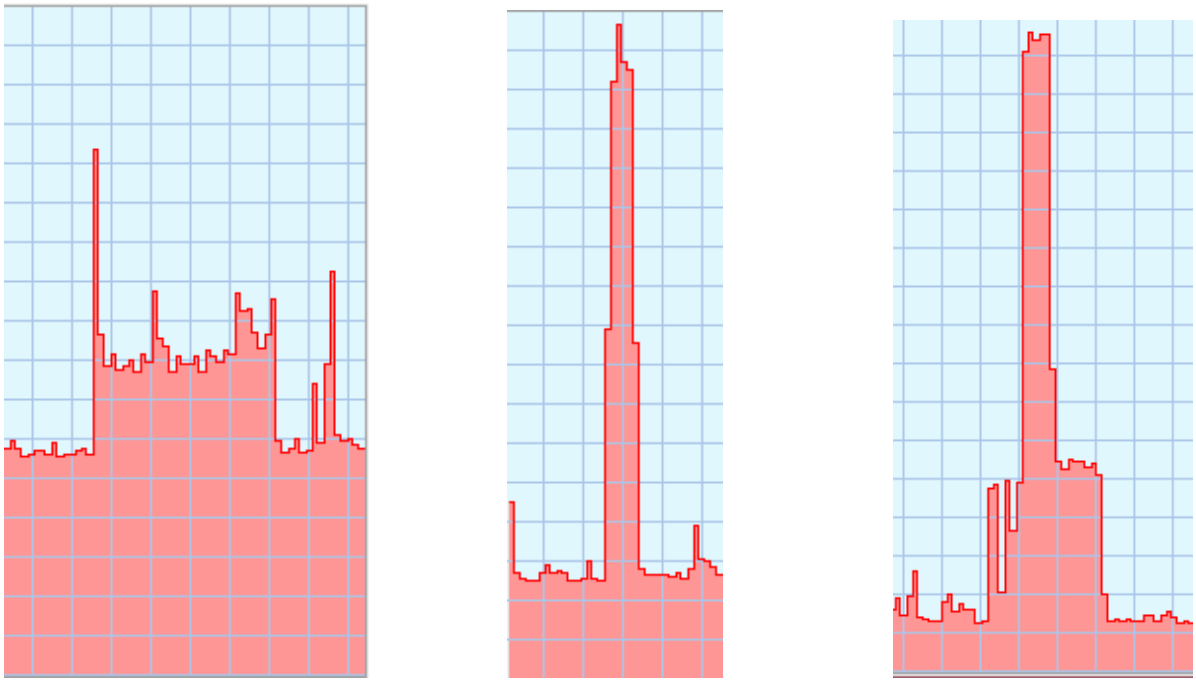
6.5.3 Power consumption

The continuous power consumption of the motion sensor was found to be 7.697 - 5.334 = 2.363 Watts on average (See Table 6.14). This also served as the minimum power consumption of the dual trigger model, as the motion detection sensor ran continuously. The motion detection model already includes the extraction of frames from the video files, therefore the image classification models did not need to use extra computational power and do this process again.

The SqueezeNet model was again only allowed to run for a maximum of 60 seconds, or until the model finished classifying all motion images. This process used an average power consumption of 17.899 - 3.549 = 14.35 Watts (See Table 6.14), where it was able to perform image classification on all 149 motion image in 10.19 seconds (See Figure 6.9b). Similar to slowing down the continuous SqueezeNet model, the 'real-time' average power consumption of this dual trigger model was also estimated with the same formula. This resulted in a true average power consumption of 18mW (see Table 6.15).

The same experiment was conducted for the motion sensor combined with the custom CNN model. This model was also only allowed to run for a maximum of 60 seconds, or until it finished classifying all motion images. It used an average power consumption of 12.856 - 3.807 = 9.049 Watts (See Table 6.16), where it performed image classification on the same 149 motion images in 17.51 seconds (See Figure 6.9c). With the amount of images it classified in a certain amount of time, the 'real-time' true average power consumption resulted in 20mW (See Table 6.16).

The custom CNN model was found to consume less power for the classification of an image, but performed much slower compared to the SqueezeNet model. This resulted in the real-time average power consumption of the custom CNN model being slightly higher compared to the SqueezeNet model.



(a) Improved motion detection sensor (b) Dual trigger SqueezeNet model (c) Dual trigger custom CNN model

Figure 6.9: Power consumption graphs of the improved motion detection sensor and improved motion detection sensor triggered image classification models.

Improved motion detection sensor		
	Sensor On	**Idle**
Min	6.881 W	5.067 W
Max	11.841 W	9.111 W
Average	7.697 W	5.334 W
Real-time average: 2.363 W		

Table 6.14: Improved motion detection sensor power consumption.

Dual trigger SqueezeNet model		
	Model On	**Idle**
Min	10.175 W	3.04 W
Max	19.675 W	5.401 W
Average	17.899 W	3.549 W
Real-time average: 2.363 W + 0.018 W		

Table 6.15: SqueezeNet power consumption on motion detected images.

Dual trigger custom CNN model		
	Model On	**Idle**
Min	12.198 W	3.184 W
Max	13.156 W	6.269 W
Average	12.856 W	3.807 W
Real-time average: 2.363 W + 0.020 W		

Table 6.16: Custom CNN model power consumption on motion detected images.

6.6 Overview

Table 6.17 shows an overview of the F1-score and average real-time power consumption for each model.

Performance summary		
Sensor	**F1-score**	**Power consumption**
AIR sensor	61.80%	0.1 W
Motion sensor	70.94%	2.155 W
SqueezeNet model	71.96%	0.477 W
Custom CNN model	70.53%	0.606 W
Dual trigger motion sensor	63.74%	2.363 W
SqueezeNet + motion	**87.31%**	**0.018 W**
CNN + motion	**87.31%**	0.020 W

Table 6.17: Performance summary of all methods tested on the same, full data-set

Chapter 7

Discussion

A known disadvantage of a camera AIR sensor is that it has a relative high detection rate of false positives. It was hypothesised that replacement of the AIR sensor by a pre-trained (CNN) image classification model and/or motion detection sensor algorithm may augment the animal detection performance by wildlife cameras.

The use of the AIR sensor alone was expected to result in an almost equal amount of true and false positives, as there were a total of 76 enter- and 74 exit events recorded according to the ground truth. However, this proved not to be the case (see Table 6.1), as the number of false positives was significantly lower than the number of exits. This originated from the fact that the IR sensor has a 10 second cool-down after it was activated and caused some exits to be missed by the IR sensor, reducing the amount of false positives. When a bird entered, it often only dropped off some food or nest material and quickly left the nestbox again. This process of entering the nestbox followed by quick exits could happen within a couple seconds, much shorter than the 10 second cool-down. This resulted in the IR sensor being able to detect most birds entering, but unable to detect birds that exit the nestbox right before the cool-down timer has passed.

The motion sensors were expected to perform equally or slightly worse compared to the AIR sensor. Yet, similar to the reduction of false positives by the build-in delay (10 s) of the AIR sensor, the motion sensors, because of their 20 second cool-down resulted in even fewer false positives (See Table 6.2 and 6.3). This cool-down period did not affect the number of true positives to the same extent. Most likely because exits may occur very close after an entry, but not the other way around. When a bird left the nestbox, it did not return until it had found food, which generally took longer than the cool-down timer was active for. One thing that could cause the missing of an entrance, is when the entrance of a bird was immediately followed by the entrance of a second bird. This would result in the sensor only picking up the first event and missing the second entrance, leading to a false negative.

Furthermore, the amount of false negatives was also much higher than expected (Table 6.1). The AIR sensor was expected to result in a low amount of false negatives, as it is a very sensitive sensor. However, due to the method of evaluating the sensor with time windows, some enter events were overlapping two consecutive time windows at once. The evaluation script then looked at whether there was a Bird_Enter annotation present and if there was a sensor trigger timestamp present. Because the AIR sensor only produces a single timestamp for each event, the evaluation script would expect two true positives for the overlapping enter event, but instead resulted in a true positive for the first time window and a false positive for the following time window.

Both the SqueezeNet model and custom CNN model were expected to result in low recall and high precision. Tables 6.6 and 6.9 show that this expectation was indeed correct in terms of high precision, whereas the recall was quite similar to what both the motion sensor and AIR sensor achieved. The main difference in performance between the SqueezeNet model and custom CNN model was in their ability to differentiate between an enter and exit event (See Table 6.5 and 6.8). Furthermore, the CNN model also had trouble with identifying Bird_Nest images, it often that it was an empty image (See Table 6.8).

The amount of false positives became very low due to the ability of the image classification models to evaluate and classify an image in order to filter out any false positives. Both models achieved a better performances compared to that of the AIR sensor and a similar performance to that of the motion sensor. The advantage of the image classification models is that they require much less power than the motion sensor, while having a better performance.

Both image classification models resulted in a somewhat increased amount of false negatives compared to the AIR sensor. Besides the effect using time windows has on false negatives, another reason for this was found to be from the image classification model being run only every 2 seconds. A bird can enter or exit the nestbox in under a second, which lead to some events being missed and resulted in either extra false negatives or true negatives.

The dual trigger motion sensor was optimized to result in the highest possible true positive rate (TPR), which resulted in a significantly higher recall value of 79.82% at the cost of a high number of false positives and

a precision of 53.05% (See Table 6.11) and a slightly higher power consumption (See Table 6.9a). The remaining number of false negatives were mostly due to the time window effect, where some events were counted twice by the evaluation script. Both the SqueezeNet model and the custom CNN model achieved near perfect false positive reduction without significantly reducing the the amount of true positives (See Tables 6.12 and 6.13). Both models achieved the exact same performance, with an F1-score of 87.31% where it only increased the total power consumption by 18mW or 20mW in case of using either the SqueezeNet model or CNN model accordingly.

Chapter 8

Conclusions

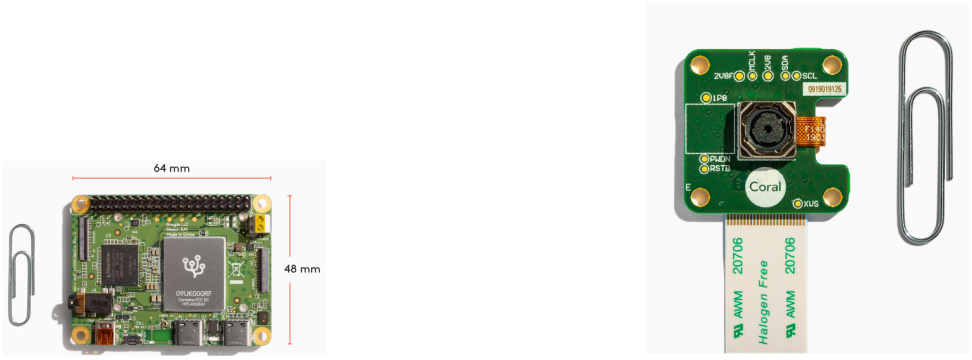
Different computer vision models were tested after conducting various experiments to optimize performance. The AIR sensor resulted in the worst performance, with an F1-score of 63.11% and plenty of false positives. The best performing model was the dual trigger model, which successfully outperformed the AIR sensor and achieved an F1-score of 87.31% with nearly no false positives. This trigger used an image classification model that was run on images which were determined to contain significant amounts of motion from another motion detection model. The down-side of this model is that it requires a continuous power consumption of almost 3 Watts, which is significantly more compared to the standard wildlife camera trap.

Chapter 9

Future work

9.1 Theoretical wildlife camera

Computer vision is required for a motion detection sensor or image classification model to work on an embedded system. This means that such a wild camera should preferable have a second, smaller, camera that runs continuously for the purpose of being used for computer vision models along with a power efficient, but powerful, processing board. An example of such a component is the Google Coral AI, which has a camera and Dev board mini suitable for such a project (See Figure 9.1). The Dev board mini is built specifically for running



(a) Dev board mini

(b) Camera

Figure 9.1: Images of low power camera and and processing board from Google Coral AI

machine learning models and should perform image classification at much greater speeds. It is equipped with a 'ML accelerator', with a peak performance of 4 TOPS (Trillion operations per second) at 0.5 watts of power consumption per TOPS. According to their website, the processing can run SqueezeNet image classification at roughly 25x times faster compared to a desktop CPU that is similar in performance to the CPU used in this project. Of course, when using the dual trigger combination, the processing board would not have to run constantly at full speed, consuming 2 Watts. However, one thing that remained unknown is whether the dev board mini could also perform efficient motion detection just like image classification. This would have to be further explored and tested on what kind of processing board would be sufficient for running multiple computer vision models.

Besides the processing board, a camera is also included that can be directly attached to the board itself (See Figure 9.1b). This is a 5MP camera that is powered by the processing board at 3.3V and 150mA, resulting in a consistent power draw of 0.495W according to its data sheet. This would add an additional 0.495 W tot the total power consumption of such a wildlife camera, which falls in the lower bound of the expected range from the literature review (See Table 2.1). The dual trigger that achieved the highest performance required roughly 2.9 Watts of continuous power consumption, 2.4 W from the dual trigger and 0.5 W from the camera.

9.1.1 Estimated life-expectancy

Most of wildlife cameras use premium AA batteries which can have a capacity of 5000 mah. These batteries are either 6V, or more commonly 12V. With the dev board mini requiring an input of 5V, drawing 3 watts continuously would refer to a current draw of 0.6 Amps. Even with a battery of 20000 mah, this would last for a little over 33 hours. Using the less effective model where the SqueezeNet model is run solely, it would draw $0.477 + 0.5 = 0.977$ Watts continuously which would be equal to a current draw of 195 mA. This would last

for roughly 102 hours, which is a little over 4 days of continuous detection. Having to use an LED during the night would even further decrease the life-time expectancy.

The largest power consumer is the motion sensor alongside a continuously running camera at almost 3 watts of power consumption. A standard PIR sensor only requires at most a couple milli-amperes of current, which is significantly lower compared to each model that was tested. It is highly unlikely that a motion detection software can be made on such a small camera where it can reach a current draw in the micro amperes.

9.1.2 Future improvements

A suggested alternative approach for future works on this project, is to keep the dual trigger method, but replacing the motion sensor with the already low-power PIR sensor. Rather than letting a small camera continuously record, having it only take pictures would be enough for an image classification model and should save a lot of power. Furthermore, having an actual wildlife camera gather data on wild animals would result in more meaningful data. This would create an experiment that is much closer to the goal of trying to improve wildlife cameras on a large variety of animals, rather than birds only.

Furthermore, having a physical prototype would also make the entire process of measuring power consumption much easier by not having to estimate how much power such a system would hypothetically consume.

Bibliography

[1] Battery consumption test archive.

[2] IR break beam sensors with premium wire header ends - 3mm LEDs.

[3] Pretrained deep neural networks.

[4] Residual light amplifiers.

[5] Types of electromagnetic radiation.

[6] William H. S. Antônio, Matheus Da Silva, Rodrigo S. Miani, and Jefferson R. Souza. A proposal of an animal detection system using machine learning. 33(13):1093–1106.

[7] Colby Banbury, Zhou, and Fedorov. MICRONETS: NEURAL NETWORK ARCHITECTURES FOR DEPLOYING TINYML APPLICATIONS ON COMMODITY MICROCONTROLLERS.

[8] Thomas M. Banhazi and Matthew Tscharke. A brief review of the application of machine vision in livestock behaviour analysis. 7(1).

[9] Amin Biglari and Wei Tang. A review of embedded machine learning based on hardware, application, and sensing scheme. 23(4):2131.

[10] Jon Bumstead. Selecting the right sensor for arduino projects.

[11] Bradley J. Cardinale, J. Emmett Duffy, Andrew Gonzalez, David U. Hooper, Charles Perrings, Patrick Venail, Anita Narwani, Georgina M. Mace, David Tilman, David A. Wardle, Ann P. Kinzig, Gretchen C. Daily, Michel Loreau, James B. Grace, Anne Larigauderie, Diane S. Srivastava, and Shahid Naeem. Biodiversity loss and its impact on humanity. 486(7401):59–67.

[12] Gregory Charvat, Andrew Temme, Micha Feigin, and Ramesh Raskar. Time-of-flight microwave camera. 5(1):14709.

[13] Jingrong Chen, Hao Xu, Jianqing Wu, Rui Yue, Changwei Yuan, and Lu Wang. Deer crossing road detection with roadside LiDAR sensor. 7:65944–65954.

[14] Dean M. Corva, Nathan I. Semianiw, Anne C. Eichholtzer, Scott D. Adams, M. A. Parvez Mahmud, Kendrika Gaur, Angela J. L. Pestell, Don A. Driscoll, and Abbas Z. Kouzani. A smart camera trap for detection of endotherms and ectotherms. 22(11):4094.

[15] Fraser Dalgleish, Bing Ouyang, Anni Vuorenkoski, Brian Ramos, Gabriel Alsenas, Benjamin Metzger, Zheng Cao, and Jose Principe. Undersea LiDAR imager for unobtrusive and eye safe marine wildlife detection and classification. In *OCEANS 2017 - Aberdeen*, pages 1–5. IEEE.

[16] Michael M. Driessen, Peter J. Jarman, Shannon Troy, and Sophia Callander. Animal detections vary among commonly used camera trap models. 44(4):291.

[17] Wei Fang, Lin Wang, and Peiming Ren. Tinier-YOLO: A real-time object detection method for constrained environments. 8:1935–1944.

[18] Rikke Gade and Thomas B. Moeslund. Thermal cameras and applications: a survey. 25(1):245–262.

[19] Alistair S. Glen, Stuart Cockburn, Margaret Nichols, Jagath Ekanayake, and Bruce Warburton. Optimising camera traps for monitoring small mammals. 8(6):e67940.

[20] Abhinav Goel, Caleb Tung, and Yung-Hsiang Lu. A survey of methods for low-power deep learning and computer vision.

[21] Daniel J. Herrera, Sophie M. Moore, Valentine Herrmann, William J. McShea, and Michael V. Cove. A shot in the dark: White and infrared LED flash camera traps yield similar detection probabilities for common urban mammal species. 32(1):1.

[22] Andrew G. Howard, Zhu, and Chen. MobileNets: Efficient convolutional neural networks for mobile vision applications.

[23] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5mb model size. Publisher: arXiv Version Number: 4.

[24] Seemanthini K and Manjunath S.S. Human detection and tracking using HOG for action recognition. 132:1317–1326.

[25] R. Kays, S. Tilak, M. Crofoot, T. Fountain, D. Obando, A. Ortega, F. Kuemmeth, J. Mandel, G. Swenson, T. Lambert, B. Hirsch, and M. Wikelski. Tracking animal location and activity with an automated radio telemetry system in a tropical rainforest. 54(12):1931–1948.

[26] Shahab Khokhar. How to choose the right LiDAR sensor for your project.

[27] D. M. Lavigne. Counting harp seals with ultra-violet photography. 18(114):269–277.

[28] Scott Leorna and Todd Brinkman. Human vs. machine: Detecting wildlife in camera trap images. 72:101876.

[29] Li, Changzhi, and Peng. A review on recent progress of portable short-range noncontact microwave radar systems. 65(5):1692–1702.

[30] Jaime Lien, Nicholas Gillian, M. Emre Karagozler, Patrick Amihood, Carsten Schwesig, Erik Olson, Hakim Raja, and Ivan Poupyrev. Soli: ubiquitous gesture sensing with millimeter wave radar. 35(4):1–19.

[31] Clare Liu. More performance evaluation metrics for classification problems you should know.

[32] Batta Mahesh. Machine learning algorithms - a review. 9(1):381–386.

[33] P. D. Meek and A. Pittet. User-based design specifications for the ultimate camera trap for wildlife research. 39(8):649.

[34] Kimberley Miller. New camera exposes pythons for hunters with special wavelength of light.

[35] Sabina Pokhrel. 4 popular model compression techniques explained.

[36] Łukasz Popek, Rafał Perz, and Grzegorz Galiński. Comparison of different methods of animal detection and recognition on thermal camera images. 12(2):270.

[37] Akarsh Prabhakara, Diana Zhang, Chao Li, Sirajum Munir, Aswin Sankanaryanan, Anthony Rowe, and Swarun Kumar. A hybrid mmWave and camera system for long-range depth imaging. Publisher: arXiv Version Number: 3.

[38] Partha Pratim Ray. A review on TinyML: State-of-the-art and prospects. 34(4):1595–1623.

[39] Miklas Riechmann, Ross Gardiner, Kai Waddington, Ryan Rueger, Frederic Fol Leymarie, and Stefan Rueger. Motion vectors and deep neural networks for video camera traps. 69:101657.

[40] Francesco Rovero, Fridolin Zimmermann, Duccio Berzi, and Paul Meek. 'which camera trap type and how many do i need?' a review of camera features and study designs for a range of wildlife research applications. 24(2).

[41] S Santosh Kumar, M Sushmitha, P Sirisha, J Shilpa, and D Roopashree. Sound activated wildlife capturing. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 2250–2253. IEEE.

[42] Kah Phooi Seng and Li-Minn Ang. Embedded intelligence: State-of-the-art and research challenges. 10:59236–59258.

[43] V. Shapoval, J. Lev, J. Bartoška, and F. Kumhála. Application of doppler radar for wildlife detection in vegetation. 49(2):136–141.

[44] Michael A. Tabak, Mohammad S. Norouzzadeh, David W. Wolfson, Steven J. Sweeney, Kurt C. Vercauteren, Nathan P. Snow, Joseph M. Halseth, Paul A. Di Salvo, Jesse S. Lewis, Michael D. White, Ben Teton, James C. Beasley, Peter E. Schlichting, Raoul K. Boughton, Bethany Wight, Eric S. Newkirk, Jacob S. Ivan, Eric A. Odell, Ryan K. Brook, Paul M. Lukacs, Anna K. Moeller, Elizabeth G. Mandeville, Jeff Clune, and Ryan S. Miller. Machine learning to classify animal species in camera trap images: Applications in ecology. 10(4):585–590.

[45] Minginx Tan, Pang, and Le. EfficientDet: Scalable and efficient object detection.

[46] Ausrys Uptas. Scientists show how differently birds see the world compared to humans.

[47] Max van den Berg. The estimation of rodent population in an area with the use of IR&RGB camera images and USV sounds.

[48] Gyanendra K. Verma and Pragya Gupta. Wild animal detection using deep convolutional neural network. In Bidyut B. Chaudhuri, Mohan S. Kankanhalli, and Balasubramanian Raman, editors, *Proceedings of 2nd International Conference on Computer Vision & Image Processing*, volume 704, pages 327–338. Springer Singapore. Series Title: Advances in Intelligent Systems and Computing.

[49] Matthew W. Helvey. Application of thermal and ultraviolet sensors in remote sensing of upland ducks.

[50] Oliver R. Wearn and Paul Glover-Kapfer. Camera-trapping for conservation: a guide to best-practices. Publisher: WWF.

[51] Alexander Wong, Famouri, and Javad Shafiee. AttendNets: Tiny deep image recognition neural networks for the edge via visual attention condensers.

[52] Alexander Wong, Mohammad Javad Shafiee, Francis Li, and Brendan Chwyl. Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. Publisher: arXiv Version Number: 1.

[53] Bichen Wu, Alvin Wan, Forrest Iandola, Peter H. Jin, and Kurt Keutzer. SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. Publisher: arXiv Version Number: 4.

[54] Yuchen Zhao, Sayed Saad Afzal, Waleed Akbar, Osvy Rodriguez, Fan Mo, David Boyle, Fadel Adib, and Hamed Haddadi. Towards battery-free machine learning and inference in underwater environments. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, pages 29–34. ACM.