ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

SCS
super computing systems

Spring Semester 2020

Smart Catflap
Convolutional Neural Network based Prey Detection on Cat Images

Semester Thesis

Nicolas Baumann, Michael Ganz
nibauman@student.ethz.ch, mganz@student.ethz.ch

26. June 2020

Advisors: David Gschwend, david.gschwend@scs.ch
Martin Bachmann, martin.bachmann@scs.ch
Professor: Prof. Dr. Anton Gunzinger, guanton@ethz.ch

Abstract

Every cat owner knows the problem of his or her cat returning into the house from outside with prey. This leads to a time consuming cleaning effort. A solution to this problem is, to employ state-of-the-art Computer Vision (CV) techniques and utilising Convolutional Neural Networks (CNN) to detect, if a cat wants to enter the catflap with prey. However, cats are only expected to enter the catflap with prey 3% of times, which leads to a largely imbalanced classification problem.
A custom and scalable image data gathering network has been built, to simplify and maximize the collection of training data. It features multiple distributed Camera Nodes (CN), a centralized master archive and a custom labeling tool. As a result of the data gathering network, 40 GB of training data have been amassed.
This thesis exploits Transfer Learning (TL) methods to generalize the problem, such that it is scalable and applicable to any cat in any environment. Further, the implemented system manages to run on an off-the-shelf Raspberry Pi 4 (RPI4) at an average detection frame rate of 1 FPS. This is achieved by an asynchronous queue that dynamically adjusts the processing rate of the queue, if a cat appears on an image. The mean time, that a cat is expected to wait for the cascade to evaluate, if the cat has a prey in its snout, is 9.6 s.
We show, that by arranging a cascade of TL models a recall rate of 93.3% with a False Positive Rate (FPR) of 28.5% can be achieved. Meaning that the cascade will correctly identify 93.3% of the cats prey entries, while falsely locking out the cat 28.5% of the times that it enters without prey.
Consequently this thesis covers; a CNN cascade with its data gathering and training process, that is scalable and able to generally classify any cat, if it has a prey in its snout; at a high recall rate with low FPR; while being edge implemented on an off-the-shelf processing unit with minimal time-overhead.

Acknowledgments

This semester thesis was a joint work between Super Computing Systems (SCS) and the Swiss Federal Institute of Technology (ETH). We would like to thank our supervisors David Gschwend and Martin Bachmann for all the efforts spent during weekly meetings and helpful feedback throughout the thesis. Additionally we would like to thank Pascal Kaiser, for his numerous and highly valuable advises regarding machine learning tasks. We thank the SCS with its joint Professor Dr. Anton Gunzinger, that provided us with powerful computational resources which were essential for the training of the multiple Neural Networks.
Further, we would like to thank all the friends and family members that supported this thesis by letting us install a Camera Node in their catflap, such that we were able to gather enough image data. And lastly, we of course want to thank our feline probands for their patience and elevated nuisance levels that they had to experience throughout this thesis.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Smart Catflap: Convolutional Neural Network based Prey Detection on Cat Images

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Baumann	Nicolas
Ganz	Michael

With my signature I confirm that

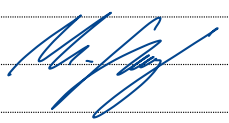
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

8716 Schmerikon, 22.06.2020

Signature(s)



For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

Chapter 1

Introduction

Cats like to hunt mice, birds and everything else that moves. One of the most common problem for cat owners is, that cats devour their prey inside the house. This occurs a few times per month and always leads to an annoying cleanup mission, or the owner hunting down a foul smell, as the prey has been tossed underneath the sofa. A solution to this problem would be a smart catflap, that is able to detect if the cat has prey in its snout. This can be done via Computer Vision (CV) techniques by utilising state-of-the-art Convolutional Neural Network (CNN).
The goal of this thesis is to implement and develop a system that can detect if any general cat wants to enter the catflap at any general environment with a prey in its snout. By generalizing the problem to any given cat, the system must be able to generally learn how a cat, with and without prey in its snout, looks like. This can be done by utilising the broad knowledge gained by pretrained models. This extensive knowledge can then be transfered via Transfer Learning (TL) techniques and be refined to solve a specific problem such as general prey detection in cat snouts. Thus allowing us to fulfill said complex task.
Interestingly enough, there is a lot of literature in the CV field regarding cats. As their large variance of shape and texture make them more challenging in CV as, for example humans [1] [2]. Hence there exist CV methods to detect cat faces [3], or even cat classes directly integrated in object detection frameworks [4]. Recently there has been a Verge article published of an Amazon employee, tackling an, at first glance, very similar sounding problem [5]. The article describes how Ben Hamm used a CNN to keep his cat from entering his catflap with prey. He uses an end-to-end solution and simply trains it with thousands of images of his cat containing *prey* or *no_prey*. This is a very specified problem, which aims to detect the prey in only one specific cat snout and with the exact same environment. This is the key difference to this thesis, as we developed a system capable of prey detection of any cat with any kind of background.

Chapter 2

Background

2.1. Machine learning basics

This section is meant to give the reader a brief overview and understanding of the most important Machine Learning (ML) terms to understand the further sections of this thesis.

2.1.1. Convolutional Neural Network

A CNN is the go-to model for most image based machine learning tasks. The duty of the CNN architecture is to provide a structure and sequence of filters and pooling layers [6], such that the model can minimize the given loss function. Usually the base layers of the CNN handle the general image processing steps, such as edge detection and further, while the deeper layers are more specialised and abstract. Therefore CNN architectures can be very complicated, and as such, nowadays it is usually the state-of-the-art approach to utilise a pretrained model which will be further discussed in Section 2.1.3.

2.1.2. ML Metrics

In order to measure the performance of a ML network, the output has to be evaluated against the ground truth; the quantisized measurement tools are called metrics. Choosing the right metric is very important, since not every metric judges the same way and therefore may not be suited for a specific application. Let us assume a binary classifier which separates cats and dogs. In Fig. 2.1 an output of this classifier is illustrated. The left part of the square represents the "ground positives" (cats) and the right side the "ground negatives" (dogs). The big circle illustrates the positive output of the network; all samples inside this circle are labeled as positive (cats), and the outside as negative (dogs).

The following points describe the confusion matrix seen in Fig. 2.1.

- **True Positives**: These are the samples labeled as positives, which are actually positive, such as cats that are actually correctly labeled by the network.
- **False Positives**: These are the samples, that are labeled as positive, which are actually negative, such as dogs which are labeled as cats.
- **False Negative**: These are the samples labeled as negative, which are actually positive, such as cats which are labeled as dogs.
- **True Negative**: These are the samples labeled as negative, that are actually negative, such as dogs which are labeled as dogs.

The goal of a network would now be to correctly separate the "ground positives" and the "ground negatives".

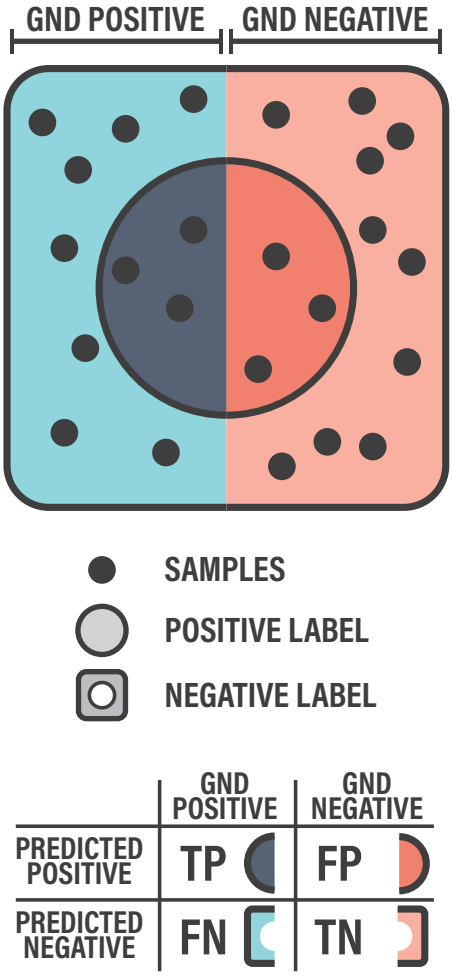


Figure 2.1.: Metrics Overview and Confusion Matrix.

The following four metrics are used to evaluate the relations between the entries of the confusion matrix. Since not every metric is equally useful for a specific application, they are explained further in Fig. 2.2.

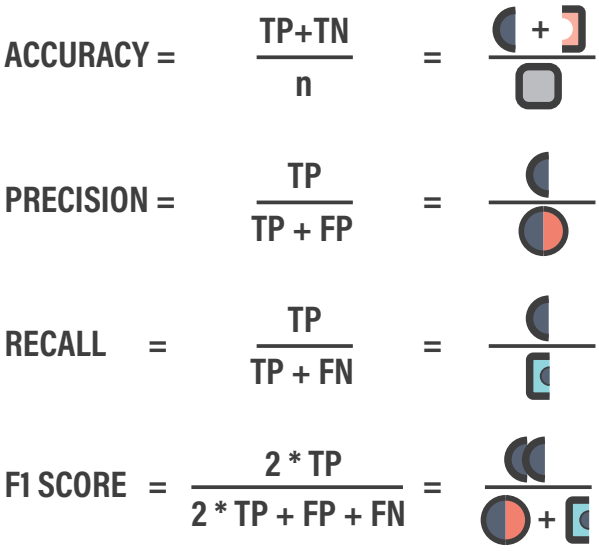


Figure 2.2.: Metrics for evaluating a network.

- **Accuracy**: This metric is simply showing the fraction of labels that are correct over all samples. It is often used for balanced dataset, where the samples are equally distributed over the classes.
- **Precision**: This metric describes the relation of the True Positive (TP) to all positive labeled samples. This shows, how safe it is to say that a sample is positive, when it is labeled as positive. It can be used for both classes, by just replacing positives with negatives. It then shows the strength of the network for a specific class. This is used, when the dataset is not balanced. Let us for example assume, that we have 900 cat images, but only 100 dog images. By classifying every image as cat, we would still have an accuracy of 90% by doing almost nothing. But the precision of the dog class would be 0% since no image has been labeled as dog.
- **Recall**: This metric is as well used for imbalanced sets. It describes the relation of correctly labeled positives to the ground positives (and vice versa for negatives). Recall is also known as the True Positive Rate (TPR), and the Inverse-Recall as the False Positive Rate (FPR) (recall for the other class, exchanging positive with negative).
- **F1-Score**: In imbalanced datasets recall and precision are often both very important. The F1-Score combines those to metrics.

A classifier such as the cat-dog classifier has an output of just one single value between 0 and 1. The higher the value, the more likely the sample was a cat. One can now set a threshold at which the labels are separated. All samples with a value higher than the threshold are labeled as positive; all lower or equal are labeled as negative. All metrics are depending on said threshold. There are multiple ways of quantising and evaluating the performance of a binary classifier for a given threshold, we will shortly summarise two prominent methods for doing so:

ROC Curve

A Receiver-Operating-Characteristic (ROC) curve plots the TPR (recall) and the FPR for all tresholds between 0 and 1. An example ROC curve can be seen in Fig. 2.3. Every point in the plot represents a threshold, at which all values of the samples are evaluated against this threshold. The blue dotted line represents the *no skill line*; a perfect classifier would have a point at (0,1). The performance can now also be evaluated by the Area under Curve (AuC) of this plot. The AuC is an accumulated value of all possible thresholds and is therefore threshold invariant. But since a network is mostly operated at one specific point on the ROC curve (which means at a specific threshold), other metrics would represent the performance better. One has now to find a point on the ROC curve where the trade-off between TPR and FPR are optimal for the specific application. If one wants to never miss a positive sample, a value further to the right should be chosen (which corresponds to choosing a low threshold).

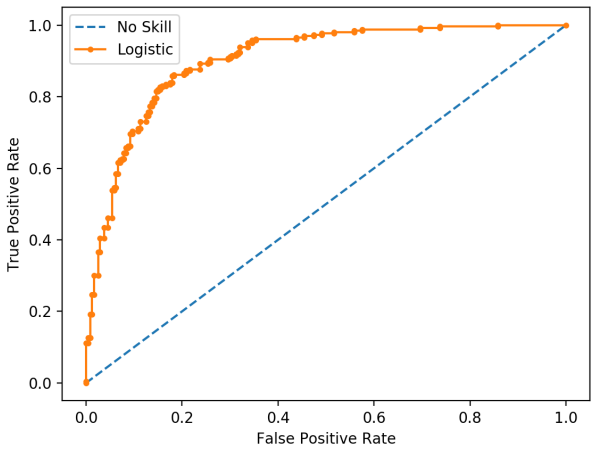


Figure 2.3.: ROC Curve sourced from [7].

Precision-Recall Curve

Similar to the ROC curve, this curve uses multiple probability thresholds to plot how precision and recall are influenced by the choice of said threshold. The probability

threshold once again refers to; at which threshold is the prediction classified as class 1 or 0? A model that has no skill at all, will predict a horizontal line on the precision axis at the ratio of the class split. For a balanced problem, this would result in a horizontal line at 50% precision. If the model produces a point above said mark, it implies that it has skill. The closer the curve is to the point (1,1), the more skill it possesses. The AuC in this case, gives a representation of the models skill invariant to the thresholds, similar to the ROC AuC. As the F1-score is a weighted average for precision and recall at a given threshold Section 2.1.2, the F1 score can also be incorporated in the Precision-Recall plot as an isobar for a given threshold. As both precision and recall are metrics valuable for imbalanced problems, a Precision-Recall curve is appropriate to evaluate an imbalanced models performance.

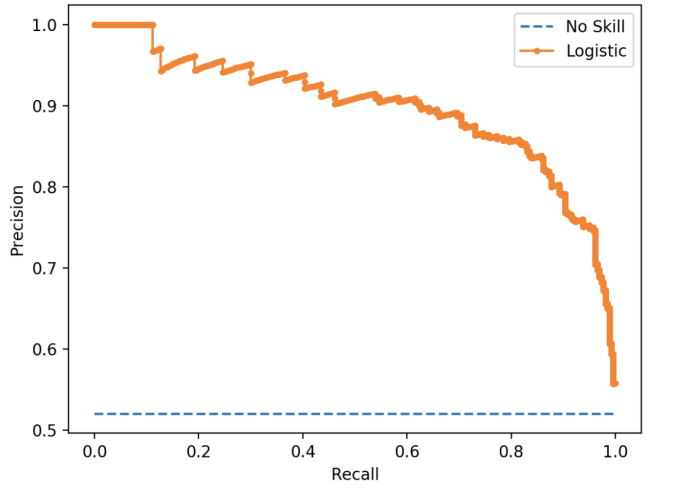


Figure 2.4.: Precision-Recall Curve sourced from [8].

2.1.3. Transferlearning

TL is used, when there already exists a trained Neural Network (NN) which solves a problem that is similar to the desired one. This model can then be used as base for the new model, including the pretrained weights. This is especially useful if there is not a lot of training data, or the task is very complex. It also saves a lot of time, if a large part of the model is already trained. We used two kinds of TL techniques:

- Feature Extraction
  Only the top layers (often the dense layers) get removed from the base model and replaced by new ones which are adapted to the desired problem. This often means that the dense layers are customized, such that they match the desired output. Subsequently only the new layers are trained, the base layer weights stay the same.

- Fine Tuning
  A whole pretrained model, which already does a fair job is used as base. All

the weights are used as initialization. Then some of the layers are selected for training, while the weights of the others are frozen. They are trained with a lower learningrate, to further improve the performance of the model. With Fine Tuning one often goes deeper in to the network than with Feature Extraction, where only the new layers are getting trained.

We frequently used both of these tools together. First, Feature Extraction was performed onto state-of-the-art networks such as MobileNet V2 [9] or VGG16 [10]. And after some epochs of training with a high learningrate, Fine Tuning was applied to the whole model with a lower learingrate.

2.1.4. Classfication Task

In a ML based classification task, one aims to predict if an input sample belongs to a certain class. Where the possible classes that we can categorize, are given by the labels of the training data. The mathematical formalities of the adaptation of the classification loss function are outside of the scope of this report. However, they can be understood by [11]. In our case for example, we want to perform image classification on the snouts of cats, to classify, if they belong to a *prey* or *no_prey* class.

2.1.5. Regression Task

The regression task works in a similar way to the classification task in Section 2.1.4. However, instead of predicting discretized class values, the regression task predicts continuous values. Meaning, that the model architecture used in a classification task, can be the same as in a regression task, *iff* the appropriate loss function is chosen, for example the Mean Squared Error (MSE) or L2 loss function.
In this project we used regressions tasks, in form of a Bounding Box (BB) finder. Here one cannot just label a number as positive or negative.

2.1.6. Object Detection

An Object Detection (OD) task, can be viewed as a combination of the classification and regression task and is usually always based on images. One aims to classify an object in the image; is this a cat? While simultaneously performing a regression task by predicting BB's around the targeted object; Where is the Cat? Thus performing the detection of the image. Such model architectures are very broad and differ from functionality to complexity over a wide spectrum. In this thesis, we are bound to real time constrains, where we have to trade accuracy for computational complexity. Therefore we are interested in architectures such as Single Shot Detector (SSD) [12] and You Only Look Once (YOLO) [4]. Both architecture types promise high accuracy while providing low inference times and there are pretrained models of both architecture types, trained on the Common Objects in Context (COCO) and ImageNet datasets. As these datasets aim to cover many general object classes, these models can be used out of the box in certain applications.

2.1.7. IoU

In order to evaluate an output of a BB model against the *ground-truth*, we used a way to map the relation between the *predicted* and the *groundtruth* BB to a value between 0 and 1 which then can be used such as the threshold of a classifier. The IoU (Intersection over Union) is calculated by the intersection of the predicted and the ground BB, devided by their union. If the IoU exeeds a given threshold, the sample is labeled as correct, otherwise it will be false. With this approach, only the accuracy can be evaluated.

2.2. Related Work

2.2.1. Amazon AI powered catflap

Recently the Verge published an article of an Amazon employee tackling the problem of using CV and deep learning to stop his cat from bringing home dead animals [5]. This project was a large motivator for this thesis. Ben Hamm, the Amazon employee behind this article, used an own CNN trained from scratch with thousands of images of his cat entering with and without prey, over a duration of multiple months. In his project, he provides data that can be used to calculate the ratio of prey and no prey entries, which amounts to roughly 3% vs 97% *prey* vs *no_prey*; and therefore we are dealing with an imbalanced data problem.

Image Type	No Cat	Cat not on approach	Cat on approach	Cat with prey
Count of Images	6,542	9,504	6,689	**260**
Example				

Figure 2.5.: Amazon AI powered catflap data overview, sourced from [5]. Prey vs No_Prey ratio can be infered to roughly 3% of prey probability.

At first glance, the problem sounds very familiar to the one that this thesis aims to solve. Yet there is a crucial difference; the task in the Amazon AI powered catflap aims to stop *his* cat in *his* environment from entering with prey, while this thesis aims to stop *any general* cat in *any* environment from entering with prey. Therefore a completely different approach is needed, which will be further explained throughout the following sections of this thesis.

2.2.2. Cat Hipsterizer

The Cat Hipsterizer is a GitHub project, that aims to hipsterize cat images by augmenting glasses onto the a cats face. As trivial as it may sound, this project involves state-of-the-art CV and CNN techniques. The author of this project uses a transfer learned CNN based on Googles MobileNetV2 and retrains the model such that it can perform a regression task onto the facial landmarks of cats, sourced from the CAT_DATASET [13]. Subsequently the angle of the landmarks are calculated and an image of spectacles can then be projected into the image.

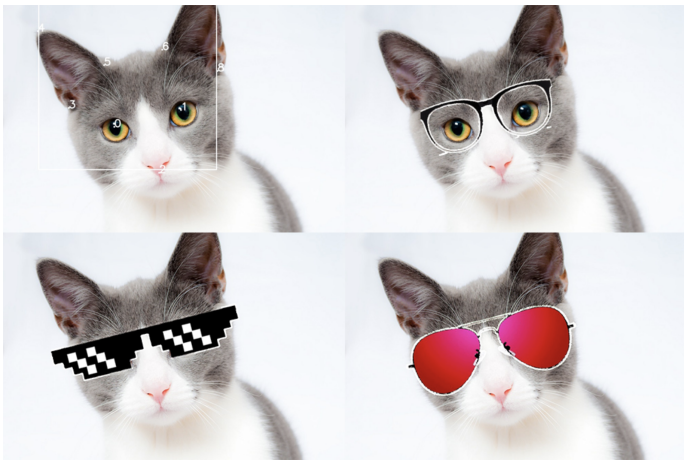


Figure 2.6.: Cat Hipsterizer GitHub project, sourced from [14]. Involves a CNN to infer facial landmarks of cats and projects spectacles into the image.

This project immensely aided our understanding of a simple BB generating model based on a CNN. In the end, we diverged from the solution presented in this project. However, it provided highly valuable information, that later evolved to the final models of this thesis.

Chapter 3

Implementation

3.1. Data Gathering

As in most ML projects, the problem falls or stands with the training data. Therefore it was essential for this thesis, to have a backbone of multiple cameras at different locations, that will perform the task of gathering images of cats with and without prey. As stated in Chapter 2 the ratio of expected prey images is roughly 3% (in our case it turned out to be much lower at 0.8%). This means, that it is crucial to have a very reliable system such that the highly valuable prey images are all captured, see Section 3.1.1. Thus the other approximately 97% of captured images will be *no_prey* images, therefore labeling has to be performed by hand with a custom labeling tool (Section 3.1.4) to streamline training data gathering. The interaction of multiple Camera Node (CN)'s and Label Graphical User Interface (LGUI) has to be handled by a master, which we call MotherShip (MS) and is further described in Section 3.1.4. The master additionally acts as the master archive for the training data. The following subsections further explain the mentioned data gathering structures in detail.

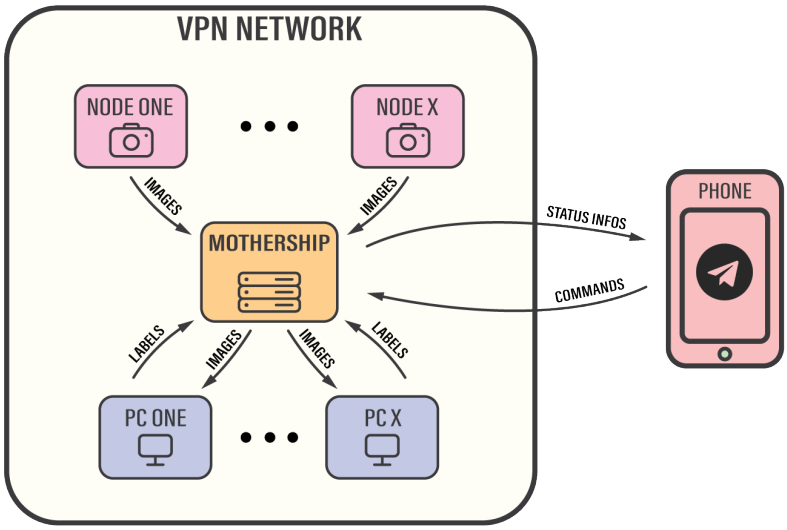


Figure 3.1.: Network overview with CN, MS and LGUI interactions.

3.1.1. Camera Node

The CN is used in the first place to gather image data for training, but as well for the actual edge implementation. The first task is described here, the later in Section 3.3.

The CN should be able to take images of approaching cats and send them to the MS. The aim is, to not miss any cat going through the catflap, since we need as many images as possible.

Hardware

In order to capture images and perform processing on it, we decided to build the CN with a Raspberry Pi 4 (RPI4). An infrared camera by Joy-IT [15] is used to take pictures, even in the night. It has a built in infrared filter which can be turned off by a GPIO pin. This allows us to use infrared LED's in order to brighten the image in darkness. For motion detection, a PIR Sensor [16] is used. In the first prototype, a hall sensor was installed on the flap itself, to trigger when the catflap physically opens.

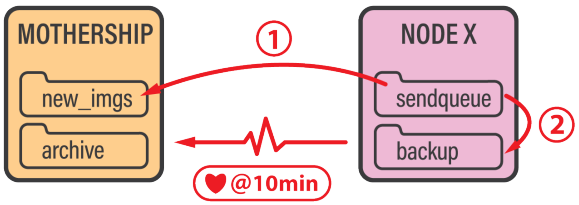


Figure 3.2.: CN and MS interaction; 1: sending images; 2: back up images; Heartbeat every 10 minutes.

Software

When the PIR Sensor is triggered by a motion, the camera starts taking images with a framerate of 2 fps. The two prototype versions handle the next step differently:

- Version 1:
  The camera takes images until the hall sensor is triggered (meaning the flap opened), or a timeout of 30s is reached. If the hall sensor is triggered, the images are fed into the send queue.
- Version 2:
  The camera takes a batch of 15 images. These are then processed by a different thread. The images will be fed through the MobileNetV2 [9], which performs an image classification on cats. If a cat is found on one of the images, the whole batch is fed into the send queue. This process has to be done asynchronously to the image capturing part, since it can happen that a cat needs longer than the 15 frames to enter, and then frames would be missing.

Each image gets a unique name, composed by node number, event number, image number and a timestamp. If the send queue contains images, the node tries to send them to the MS (Fig. 3.2 step 1). It can happen, that the node cannot reach the MS due to connection problems, so the images remain in the queue. The RPI4 has a 16GB SD card, which is also used for backing up the last 1300 images, which had a cat on it for redundancy reasons (Fig. 3.2 step 2). Every 10 min the CN sends a heartbeat to the MS. It also checks if the nodes backup is full, such that it has to delete the oldest images.

3.1.2. Mothership

The MS tasks are the following:

1. Serve a buffer for the LGUI, that holds the unlabeled images which the CN's upload.
2. Hold the master archive of the labeled data and allow other computers to synchronize their local archives for redundancy.

3. Allow easy Human Machine Interface (HMI) such as alerts and answering status requests.

From a Hardware perspective the MS is very simple, it does not require high computational power. However it needs to be very reliable with a high storage capacity. Therefore we chose to utilise a RPI4 with an external Solid State Disk (SSD) as storage medium for the master archive. To serve the buffer and allow other computers to synchronize to the master archive, the MS must be reachable and have a high up-time.

Task 1: Therefore the MS hosts a lightweight Flask server [17] such that the CN's can upload their images via a HTTP POST request. This is visualized in Fig. 3.2. The uploaded images will then be stored inside the *new_imgs* directory. Additionally the MS features a basic heartbeat interaction with the CN's, such that the network status can be monitored via the Telegram-Bot in Fig. 3.4. Further the MS acts as a Virtual Private Network (VPN) server, as the CN's can be placed outside of the MS home network, thus connectivity is guaranteed.

Task 2: The master archive has to be populated with images and labels which have been validated by a human, through the LGUI. Consequently the LGUI and MS must follow an interaction protocol, as can be seen in Fig. 3.3. Through the MS Flask server, the LGUI can download the raw/unlabeled data buffered in the *new_imgs* directory (step 1). Subsequently the MS will receive a *delete request* together with an *archive request* after the LGUI has labeled the data (step 3). With the *delete request* the MS now knows which images inside the *new_imgs* directory can immediately be deleted. With the *archive request* the MS receives the labels in form of a .csv file (step 3) and can now move the matching images together with its labels into the master archive (step 4). Lastly the Flask server allows an interface such that the file content of the MS master archive can be requested (step 1). From the reply of such a *sync-request*, the local archives can check, if they are any missing files in the master archive and then download the missing images together with the labels to mirror the master archive and assure redundancy (step 1/2).

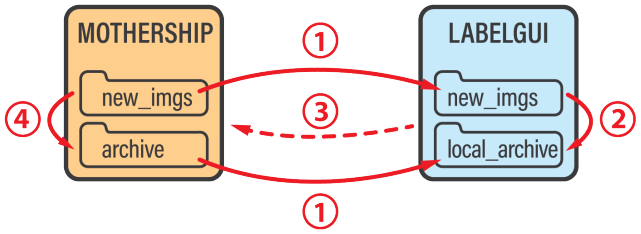


Figure 3.3.: MS and LGUI interaction. 1 represents the LGUI downloading images from the MS; 2 constitutes the transfer and/or the deletion of labeled image to the local archive; 3 represents the *delete request* and *archive request*; 4 constitutes the transfer and/or deletion of the post labeled images to the master archive.

Task 3: Lastly the MS implements a Telegram-Bot via [18], to facilitate the user

interaction and notify via the bot. This is the only interaction that leaves the MS-VPN network and is globally accessible to telegram users that know the bots id key, providing sufficient networking security. The users can now easily communicate via the bot, for example querying the *nodestatus* or receiving notifications of arrived CN images.

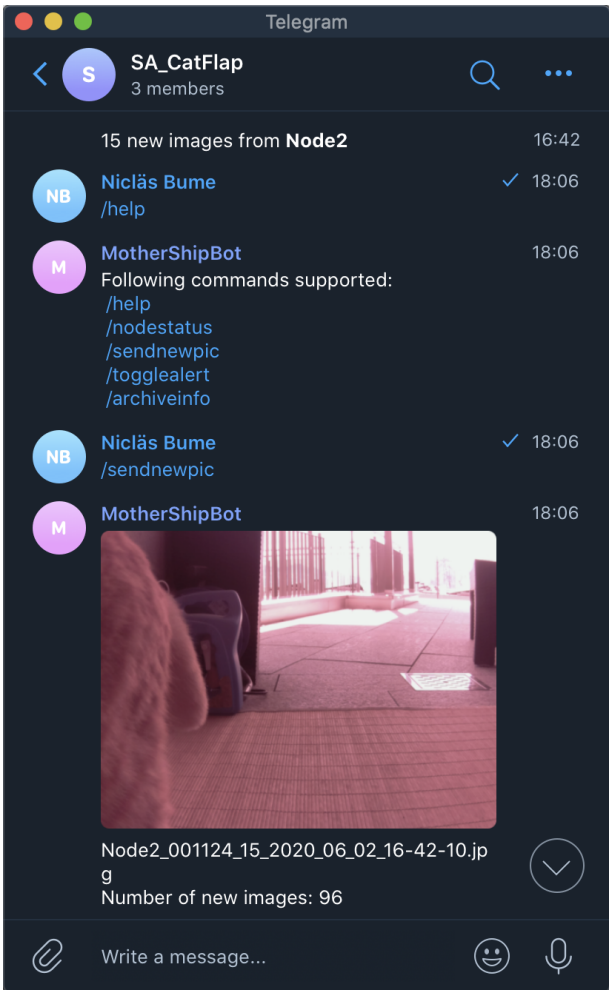


Figure 3.4.: Screenshot of the Telegram-Bot running on the MS. The top message is a notification that images arrived, while the later messages demonstrate the commands that the bot can handle.

3.1.3. Standalone Network

As can be seen in Fig. 3.1 and detailed in Section 3.1.2, the MS hosts a VPN server, while the CN's are the clients. We utilise WireGuard, a modern and lightweight VPN tunnel [19]. The VPN network is necessary, as the CN's are placed in different networks and therefore exist behind a Network Address Translation (NAT). A workaround to this problem would be to host the MS on as a public server, however from a network security

perspective this would not be suitable. Hence we chose to utilise the VPN network and secure the traffic trough the VPN tunnels and secure the MS by only being able to access it from within the secured network.

3.1.4. Image Label GUI

The LGUI is called from any UNIX based computer. it handles the following:

1. Download the images of the MS-*new_imgs* into the local-*new_imgs* directory.
2. Provide a user friendly UI such that the user can easily label the images.
3. Send *archive request* and *delete request* to the MS and move/delete the images accordingly to the local archive.
4. Mirror the local archive with the master archive for redundancy.

Task 1: This step is visualized in Fig. 3.3 and partly described in Section 3.1.2. The LGUI uses a HTTP GET request to the MS Flask server to query the content of the master *new_imgs* directory. Then it will download said images via a second HTTP GET request. Now the unlabeled images are inside the local *new_imgs* directory.

Task 2: Now that the raw images are inside the local *new_imgs* directory, the GUI will pop up. The user will see a collage of the 15 images that the CN uploaded, which we call an event as in Section 3.1.1. Firstly, the user must decide if the event shall be discarded. An event is deemed unusable if the human cannot infer if the cat has prey in its snout; one would then select the delete button and a delete request will be queued. If the event is usable, the user must label the cat's name and select the prey flag, if there is prey visible. This information is stored inside a *.csv* file and it forms the *archive request* which is queued as the user executes the *done* button. All buttons in the LGUI are mapped to keys to speed up the labeling process.

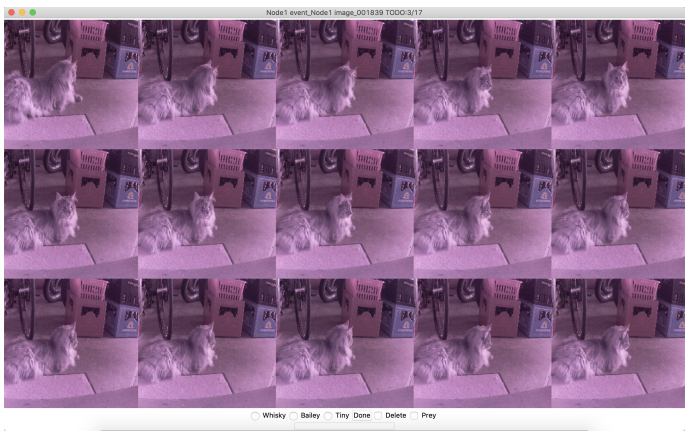


Figure 3.5.: Screenshot of the LGUI.

Task 3: A *delete request* and an *archive request* will be sent either after all images in the *new_imgs* directory have been labeled or after 30 labeling instances, such that one does not loose too much work, if an error occurs. After the requests have been done, the LGUI will move the labeled images in the *archive request* to the local archive and delete the images in the *delete request*.

Task 4: Lastly a *sync-request* can be sent to the MS. The MS will reply with the content of the master archive. The LGUI will compare the content of the local archive to said response and download the missing images with their labels, via a HTTP GET request, if necessary. Thus multiple computers can easily mirror the master archive and redundancy is achieved.

3.2. Convolutional Neural Network Cascade

It would have been a possibility to create an end-to-end system similar to [5], simply construct a CNN and train it with thousands of *prey/no_prey* images of your own cat. This approach would obviously work, however only with the cat used in training and only at the same environment/background. Since the model would only learn the difference between your cat with and without prey in the given environmental context. What we aim to achieve in this thesis, is that the model learns to identify if any cat has a prey in its snout, in any given environment. This requires that the system learns in general, what a cat is and how its snout should look like. Therefore, a good approach is to utilise pretrained models that have been trained for multiple weeks on a given problem and apply transfer learning to utilise the general knowledge from its exhaustive training process, in a more specified task. Yet with this method we run into constraints, that the pretrained models set, namely their input shape. CNN's usually cannot process images in arbitrarily high resolution, as this would result in too many parameters, which would need to be trained. Therefore the CNN's have a recommended fixed input shape (as they have been trained with said shape), requiring a resizing of the target image as a preprocessing step. In case of MobileNetV2 on Keras for example, the input shape is fixed to 224×224, meaning that an image has to be resized from its full resolution (in our case 2592×1944 pixels) to 224×224 pixels. By having a look at a prey image in full resolution Fig. 3.6, one can now easily see the problem; the CNN can only see the resized image of 224 pixels. Deciding if an image contains a cat with prey in that resolution would be very difficult for a human, which would of course provide a nearly impossible task for a NN since it does not only have to check about prey, but as well be able to find the face of the cat.

Figure 3.6.: Visualisation of the difference from a full resolution image resized to 224 × 224 and one which has the same area cropped at full resolution.

The solution that this thesis utilizes, is to use multiple different models, that we call stages, such that one can benefit from the high resolution of the image. We segment the problem into multiple stages. The first stage identifies the cat, called Cat Finder (CF), it crops the full resolution cat and provides it for the next stage. Said following stage tries to find the cats snout and again crop the snout from the full resolution image, it is called Snout Finder (SF). Lastly, the main stage, called the Prey Classifier (PC), will classify the cropped full resolution snout if there is a prey or not. All stages are called sequentially and the entire system is therefore a cascade of CNN models.

3.2.1. Cat Finder

The task of the CF, is to determine whether a cat is present in the image and if it detects a cat, it shall then return a BB of the detected object. Therefore the CF performs an OD task, as described in Section 2.1.6. Luckily for this thesis, both ImageNet and COCO datasets contain class instances of *cat*. As such, we can utilise a pretrained model more or less out of the box. The input is a full sized image and the output of the CF is a catBool, the certainty of the catBool, and the cropped image of the cat.

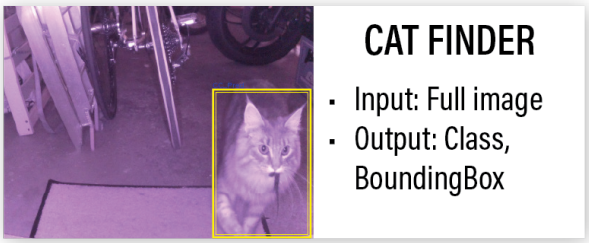


Figure 3.7.: Overview of the CF stage.

Implementation

Since we can utilise the pretrained models out of the box, we had to evaluate which architecture works best to find cats. Thus we implemented two versions of the CF, one with YOLOV3 and one based on Googles MobileNetV2 SSD architecture.

YOLOV3

The YOLOV3 model is at the moment the strongest YOLO based model, as stated in [4]. It is trained on the COCO dataset, which of course contains the cat class. YOLO is not specifically targeted to be used by Keras, however it can be done by using a weights converter script [20]. By downloading the model and its pretrained weights from [21] and applying the converter script, a h5 modelfile of YOLOV3 is created. Said model can then be manipulated just like any other Keras model.

Decodation of the YOLO prediction is a bit more complicated and involves the iteration of all BB's over every class instance. As the model is capable of recognizing 80 classes and due to the YOLO architecture, the amount of BB's it outputs is very high, this results in a large overhead. In our case, the overhead averaged to approximately 0.75s on a 2.3 GHz Quad-Core Intel Core i5 CPU, even though the authors of [4] claim that it is very fast. However their claim respectively covers GPU's and not CPU's. On the other hand, the accuracy of the model was astounding. Our custom evaluation set (as later in Section 4.2) yielded a 98% accuracy and a 97% IoU at a mean inference time of 0.32 seconds. To reduce the inference time, we also implemented a CF version using tiny-yoloV3, a fixed point model of YOLO. The results on the evaluation set however, showed that the lower inference time was traded with a high portion of accuracy: 45% accuracy and an IoU of 30% at an inference time of 0.06s. The accuracy could have been increased by retraining the tiny-yolo model, nevertheless due to time limitations we did not do so.

MobileNetV2

Googles MobileNetV2 [9] is a SSD designed to run on mobile devices. Throughout this thesis, the MobileNetV2 model is frequently used as base model, it is characterised by its very fast inference time and ease of implementation due to its integration into the Keras API. However the Keras MobileNetV2 model is only suitable for image classification and not OD. Fortunately, there exists a model zoo of pretrained OD models with MobileNetV2 as base model [22]. However said model could not be converted to a Keras h5 format. Therefore we had to implement the CF directly with the low level TensorFlow API. This is the only cascade stage that is not implemented in Keras directly, yet an interaction with the following stages can be achieved easily by using a wrapper as abstraction. On the evaluation set we achieved following results: 89% accuracy with an IoU of 78% at a mean inference time of 0.078s, again on the 2.3 GHz Quad-Core Intel Core i5 CPU. Thus the accuracy of cat detection is slightly lower than the YOLO version, however the inference time drastically decreased.

Figure 3.8.: Predictions and certainty of MobileNetV2 trained on COCO with real catflap images.

Cat Finder Stage Conclusion

The YOLO model provides very high accuracy and high quality BB, at the cost of a larger inference time. MobileNetV2 is slightly less accurate, but infers significantly faster. As this thesis is time constraint by the final edge implementation Section 3.3, we therefore opted to choose the MobileNetV2 model as base model for the CF stage. The output of the CF stage can be viewed in Fig. 3.7.

3.2.2. Snout Finder

In order to feed the PC with appropriate input images, the snout has to be located first. This is the task of the SF, which has to perform an OD task. As usual in the CV field, there are multiple methods of achieving this task. In this thesis we primarily investigated using a HAAR cascade and using a CNN. As both techniques have different strengths and weaknesses, we decided to use a wrapper logic to combine both methods, thus the SF profits from the strengths of both approaches.

HAAR Cascade

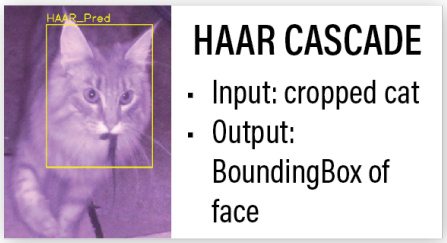


Figure 3.9.: Overview of the Haar stage.

A HAAR cascade is a method to find some specific patterns inside an image. It is often used to find faces, and has been used a lot in the beginning of auto focusing on faces in digital cameras. The HAAR cascade is a very well known and older technique for face detection, further it is not based on deep learning. Rather it relies on a few hand crafted features that can be run in thousands of different combinations [3]. In terms of HAAR cascades, one often refers to so called *weak classifiers*, as the features used in this technique are very primitive and therefore yield a high number of False Positive (FP), however this can be compensated by running the cascade in every numerous possible combination. This actually makes the HAAR cascade a very robust and low inference time model, which is a great strength of it. Combined with the fact that it is not a deep NN, which actually makes it independent of said deep NN's and can be used to run alongside. HAAR cascades are implemented inside OpenCV [23]. It requires an XML-file, which is the network. A few networks are already integrated inside the package, such as one for finding human faces, or even cat faces. And that is exactly what we used it for.
In comparison to Section 3.2.2 CNN-BB, it does not generate a bounding box on every image that is being fed. But if it finds a box, it is usually very accurate. It is extremely easy to implement compared to a CNN, which has to be trained with a lot of data.
So the HAAR cascade is not deep NN based, is very robust, has a low inference time and is easy to implement. The downsides of this approach however; are that it often fails to find the desired object and secondly that it allows for low flexibility, as the features are hand crafted. Yet luckily enough, there exist models for the use case of this thesis.

CNN Bounding Box

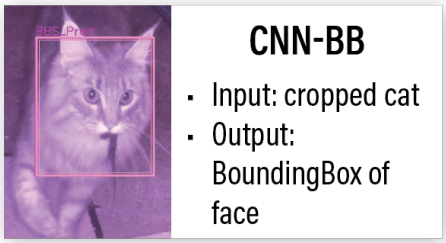


Figure 3.10.: Overview of the CNN Bounding Box.

Even tough the task of the SF performs an OD task, we model the CNN-BB task as a regression task of predicting coordinates in an image. To fulfill the OD requirements, we will sequentially run the output of the CNN-BB through the Face or Fur (FF) classifier, later described in Section 3.2.2 FF. Therefore we want to input an image of a cat and receive a BB around the snout of the cat.

Architecture

The CNN-BB is created by using TL techniques. We use MobileNetV2 as a base model. As feature extractor, we use a fully connected dense network as seen in Fig. 3.11. The last dense layer is composed of four neurons that use a ReLu activation function such that four continuous values can be predicted. These four values represent the x and y coordinates of the left and right eye of the cat. From the eyes we can then infer the BB with a simple scaling of the distance between the eyes. Important to note, as in Section 2.1.5 is, that we use the MSE loss function to enable the regression.

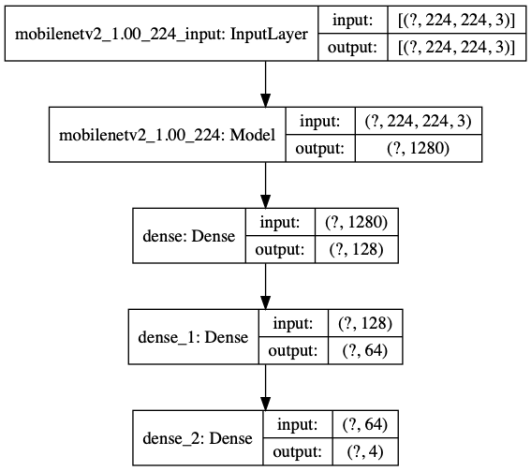


Figure 3.11.: Plot of the CNN-BB model graph.

Training

As is visible in Fig. 3.12, the used CAT Dataset [13] is very diverse and includes various different cats, angles and lighting. It contains over 9000 cat images together with nine landmarks each. Therefore it is safe to say that the CNN model is trained for any kind of cat. Initially we trained our network with similar labels as [14], meaning that we performed a regression on the top left landmark and bottom right landmark. This approach worked fairly well, however we noticed that the output of the model seemed to perform better, if both eyes were visible in the input image. This lead us to assume that the CNN-BB learns to detect the prominent landmarks and then rescale the regression output such that it represents a BB. Because of this, we also performed the task with all landmarks as labels and realized that the model performs best on the regression of the eyes.

Figure 3.12.: Sample of CNN-BB training data, sourced from the CAT Dataset [13]. The drawn landmarks are only to visualize the labels, the training images of course do not include the green dots.

Throughout the training process, we have also tried to compare the output of the landmark predictions to a Canny Edge Detection output [24]. The results of this can be seen in Fig. 3.13. Where we can see an overlay of the image with its landmark predictions and the canny edge detection. Throughout the evaluation of this technique, we realized that the landmarks always reside within a cluster of the canny transformation. Thus implying that the NN actually learned to perform a very similar transformation. Making an edge detection superfluous of course. Nonetheless it is an interesting feature to discover.

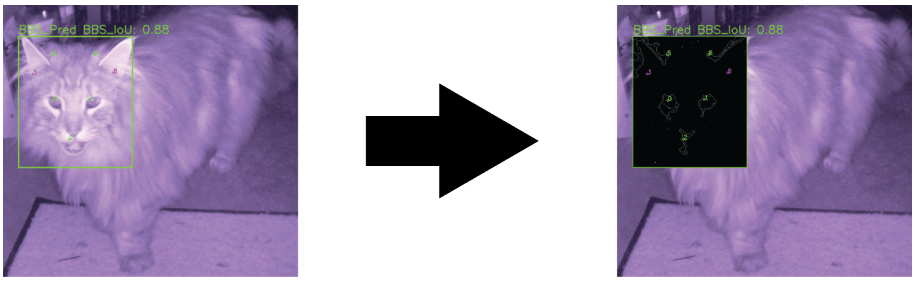


Figure 3.13.: Visualisation of the overlap of Canny Edge Clustering and the landmarks prediction.

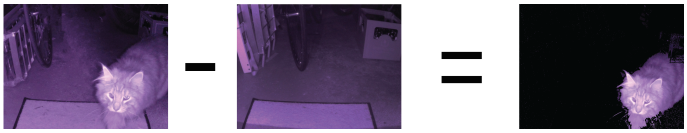


Figure 3.14.: Background substraction for preprocessing the input image. This did not have a significant impact on the BB prediction of the model.

Further we also tried to highlight the cat on the image by performing a background substraction, as in Fig. 3.14. However, the models predictions did not improve by applying this, see Appendix A. Given all these factors, we chose to use the model to predict the location of the eyes and then perform a hardcoded rescaling to output a BB resulting in Fig. 3.10.

Face or Fur

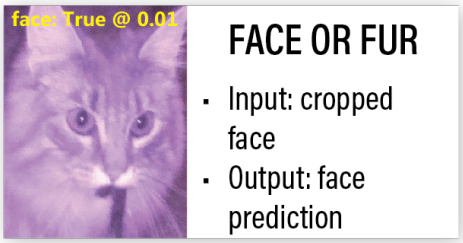


Figure 3.15.: Overview of the FF stage.

As previously mentioned in Section 3.2.2 CNN-BB, the output of the CNN-BB must run through a model that can return a certainty of the output actually being a cats snout. Therefore we have to create a second model that performs a classification on *Face* or *Fur*. Thus we will receive a BB with a confidence, hence attaining OD. Consequently the task of the FF model is to predict if an input image is actually a *Face*. Which is equivalent to a binary classification task.

Architecture

Once again we utilise MobileNetV2 as a base model and build a small dense network as feature extractor on top of the base. Important to notice in Fig. 3.15 is, the single output neuron with the sigmoid activation function. This is a typical output architecture for binary classification. Thus we obtain a value between 0 and 1 as an output, where 0.5 is the default threshold for evaluating to which class the input image belongs.

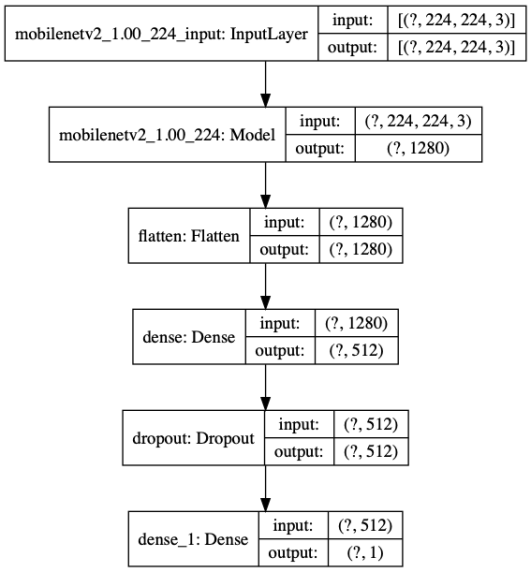


Figure 3.16.: Plot of the FF model graph.

Training

In Fig. 3.17 one can see that the images of the different classes are quite easy to separate as a human. Therefore we infer that the task should be fairly easy for the model as well (as a rule of thumb). The training is performed with 302 instances of the *Face* class and 306 instances of the *Fur* class. This results in 19 training batches at 32 images per batch and additionally we use 2 batches for validation. By using Keras *ImageDataGenerator* we additionally perform data augmentation on the training images, which increases robustness of the classification.

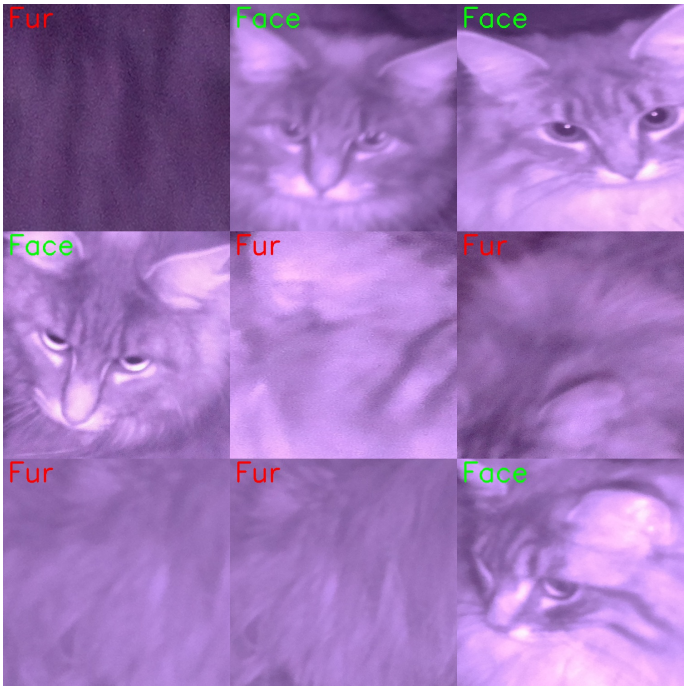


Figure 3.17.: Sample of FF training data, completely sourced by own data.

Snout Finder Stage Conclusion

As mentioned throughout this section, the two OD approaches featured in the SF, both have their advantages and disadvantages. The HAAR approach runs slightly faster than the sequential run of the CNN-BB and the FF. It also features a very low FPR and a reasonably high TPR, further evaluated in Section 4.2. This implies robustness, meaning that the HAAR approach does a good job, *iff* it triggers. The problem however, is that it only triggers roughly 50% of the time that it should. The CNN-BB + FF approach on the other hand, has a lower TPR and higher FPR, but it triggers 100% of the time.

Thus one can observe that their strengths can be accumulated, while minimizing their weaknesses. Therefore the SF as a stage, combines both HAAR and CNN-BB + FF, as can be seen in Section 3.2.5. This shows, that the SF has to use a wrapper logic that inferes a BB with the predictions of both, HAAR and CNN-BB + FF. This infered BB is represented as a white BB in Fig. 3.18. The wrapper logic is held very brief and simple: If the HAAR model predicts a large enough BB, then we trust it because of its low FPR, meaning that *inf_bb* = *HAAR_bb* and the CNN-BB + FF models do not have to be run. If the HAAR model fails to predict a large BB (it does not trigger), then *inf_bb* = *CNN-BB* if *FF_true* else *None*.

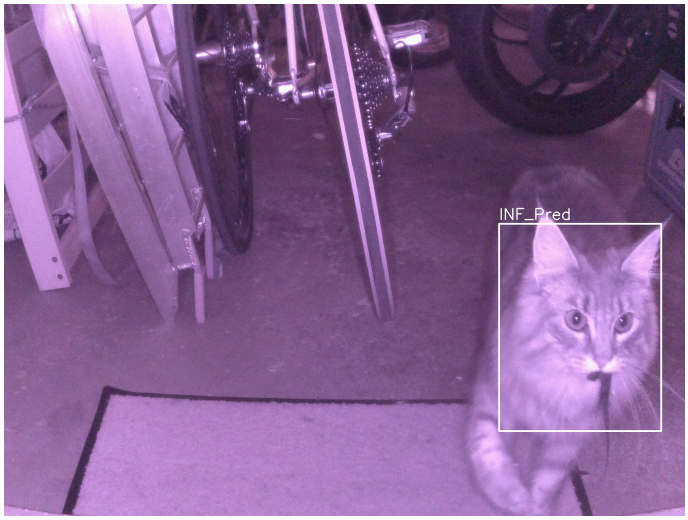


Figure 3.18.: Wrapper logic for *inf_bb* in white. In this case *inf_bb* = *HAAR_bb*, since the HAAR model triggered, as in Fig. 3.9.

3.2.3. Prey Classifier

The PC is the final step in our cascade. All the previous stages do their best to deliver a perfectly cropped catface, which now can be used to classify, if there is a prey in the snout of the cat. This is not an easy task, especially not on infrared images. A mouse for example often has a rather similar color to the fur of the cat. During the labeling process we discovered, that it is sometimes even hard for a human to determine if there is prey in the snout.

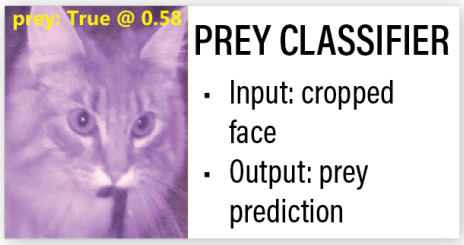


Figure 3.19.: Overview of the PC stage.

Architecture

During the implementation phase, we have iterated through every ImageNet pretrained architecture in the Keras API. The best result came from the VGG16 [10], which we used as base model. The top layers have been replaced by the following ones:

- tf.keras.layers.Dense(256, activation='relu'),
- tf.keras.layers.Dropout(0.5),
- tf.keras.layers.Dense(1, activation='sigmoid')

Since this is a classification task, the output is reduced to only one neuron.

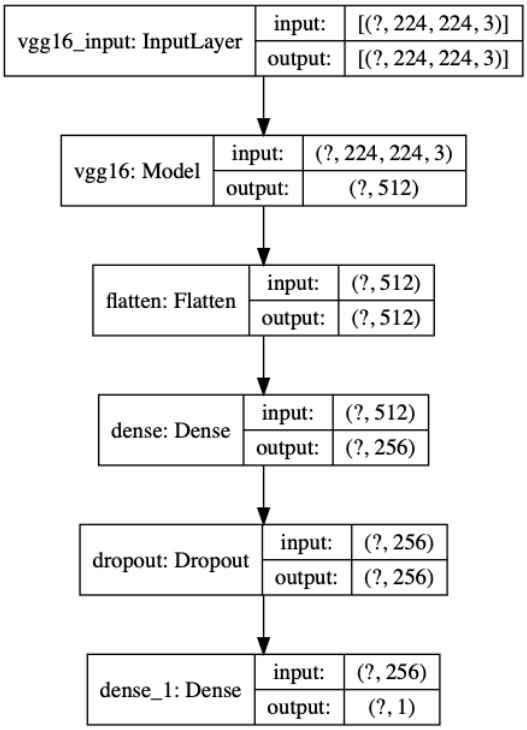


Figure 3.20.: Sample of PC model graph.

Training

The biggest challenge in training the PC was the lack of data. We mined the internet for cat images with prey, as we only had 19 own images with prey (at the time of training). Samples of the training data can be seen in Fig. 3.21. Another hurdle, was that our cats mostly brought lizards and slow worms. The cat images from the internet had only birds and mice. It is good to have such a variety, but since most of the training samples consists of birds and mice and our test set is only consisting of lizards and slow worms, the network will not perform as well on our images than on images with mice and birds.

On the other hand, it has been easy to get enough data without a prey. The realworld application of the catflap is a totally imbalanced task, with 3% prey images (seen in Section 3.1). If we would use a training split of 97% to 3%, the network would have an accuracy of 97%, if it just classifies every image as *no_prey*. There are multiple options to go against this behaviour. One could downsample the dataset, such that a lot of

no_prey images are getting thrown away, which would result in a proportion of 50:50 the downside of this method is, that a lot of data is lost.
The opposite procedure can be done, by upsampling the minority class. This means just duplicating the *prey* images. The disadvantage of this approach is, that the images of the *prey* class would be extremely similar, which would result in overfitting, since the network can just memorize the few different images.
Another solution would be, to penalize wrong decisions on a *prey* image harder than on a *no_prey*-image. This can be done by a customised loss-function, which can easily be implemented in Keras. At the time of the PC training stage, we managed to gather 150 internet-*prey*-images and 15 own-*prey*-images. 32 of the internet-*prey*-images have been used for the validation set. Therefore the minority class consists of 143 training images with 15 own-prey-images. As there is no problem of gathering enough samples of the majority class, we decided to go with a 2:1 *no_prey*:*prey* ratio. Which would result in a loss weights ratio of 0.75 : 1.5. With this approach we achieved the best results.
Since we only had 448 images, we applied data augmentation on it. This is crucial when the amount of training data is very sparse, as we can thus "create" multiple images from one original image. The Keras *ImageDataGenerator* provides an easily implementable framework to generate such augmented data on-the-fly:

```
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest')
```

Since this is a classification task with only one output, we went for the *binary_crossentropy* lossfunction. As optimizer we tried Adam, but got much better results in the end with RMSprop. A batchsize of 32 was used to train for 150 epochs.

Figure 3.21.: Sample of PC training data, sourced by prey images on the internet and own prey images.

Prey Classifier Stage Conclusion

At the beginning of this learning process we trained and verified solely on internet images, because at this moment we had even fewer images than the 19 own *prey*-images. First we applied feature extraction as described in Section 2.1.3. With this task and without our own data we achieved an accuracy of 81%. After that, fine tuning was applied, to further improve the model. This led us to an accuracy of 92%. At the point where we had 19 own *prey*-images, we added them, and 38 own *no_prey*-images to the dataset and repeated to whole process. After the fine tuning part, we settled at a lower accuracy of 86%, which was to be expected, since there are only internet images in the validation set. The new model (trained with our own data) as well seems to perform worse, but after the evaluation on our own data we could see, that it has actually improved significantly. The performance comparison can be seen in Table 3.1. The FPR has been reduced substantially, without decreasing TPR, which means the overall accuracy has improved. The goal of the catflap is to lock out all cats with prey, but still let through a cat without prey as often as possible. The False Negative (FN) stands for cats with prey, which would have been led in. The FP stands for cats without prey, which have been locked out by mistake. More information about the policy, when a cat has proved, if it is *clean* can be seen in Section 3.3.2

	only Internet	with own data
True Positive	3	3
False Positive	39	22
True Negative	21	38
False Negative	0	0

Table 3.1.: Performance comparison "only internet images" vs "all images" at same threshold.

In order to visualize the performance on our data, we took 3 images, where a cat is approaching the catflap. We photoshoped a mouse into the snout and classified it by the PC network, once with mouse and once without. The comparison can be seen in Fig. 3.22 and Fig. 3.23. One can see, that the PC value (on the top left) has increased strongly on the *prey*-images, compared to the *no_prey* ones. If the input image is a good cropped face, which is guaranteed by the previous stages, the PC works better than we expected with this little data.

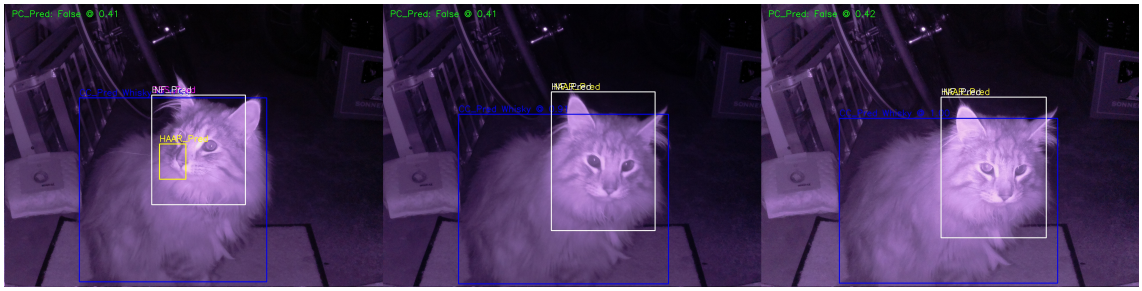


Figure 3.22.: PC on own images without prey, values: 0.41 / 0.41 / 0.42

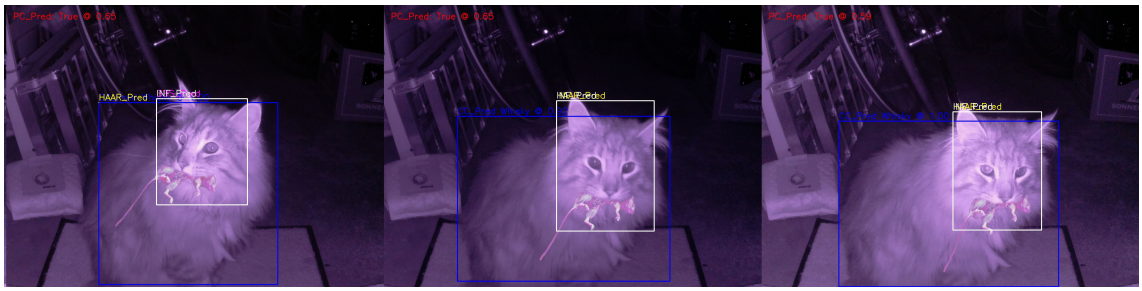


Figure 3.23.: PC on own images with prey, values: 0.65 / 0.65 / 0.59

3.2.4. Cat Recogniser

Technically a recognition task is different to a classification task. Classification is the supervised subset of pattern recognition. Nonetheless, in this stage we aim to find out:

Who is this cat? We can do so, by remodeling this task into a classification task, by classifying the cat on the image with a label of one of the training classes. Once again, there exist multiple different CV approaches. In this thesis, we will discuss a Principal Component Analysis (PCA) approach and CNN approach.

Principal Component Analysis Approach

Cat recognition works similar to human recognition, however cats have a much higher variance than humans, due to their different shapes and textures. This makes cat recognition significantly more challenging than human recognition for CV related tasks [1]. Nevertheless, we can have a look at the human face recognition techniques. One such approach is to utilise so called *Eigenfaces*, obtained by the PCA of a set of input images [25]. With the PCA we calculate the most defining eigenvalues of the input set, we call these eigenvalues the *Eigenfaces*. Now we know the transformation, such that an image of the input set can be perfectly reconstructed by a linear combination of the *Eigenfaces*. By knowing the PCA, we can apply the transformation to images outside of the input set, thus decomposing the input image into a linear combination of the *Eigenfaces*. In fact, this problem is actually roughly the same as a Fourier Decomposition, where one aims to reconstruct an input function with a linear combination of the sinusoidals with different frequencies [25].

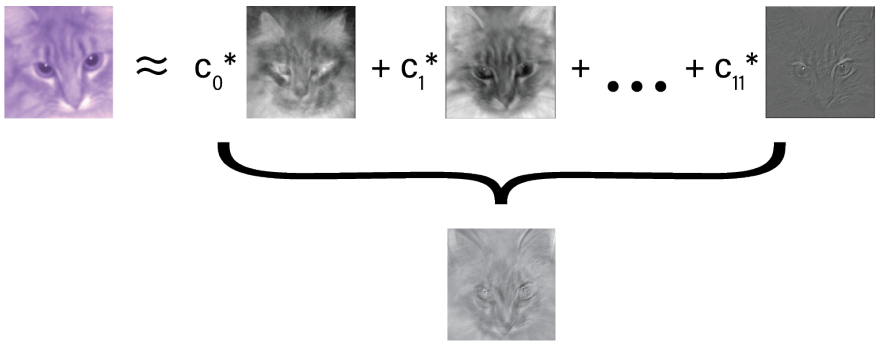


Figure 3.24.: Visualization of the reconstruction via the PCA transformation.

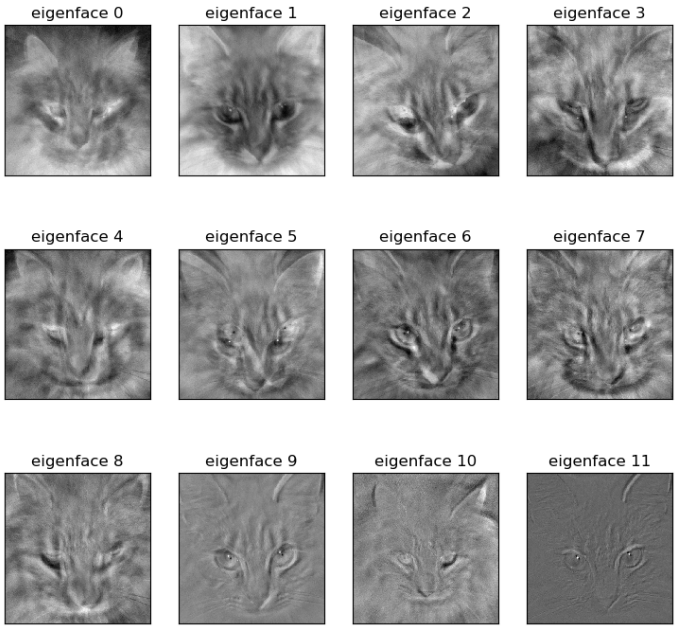


Figure 3.25.: The twelve most significant *Eigenfaces* of an input set of cat images.

By performing the PCA, one actually performs a drastic dimensional reduction of the data. Prior, the dimensionality of the data was the amount of pixels in the image, whereas now the dimensionality is the amount of coefficients of the transformation. This allows us to work in an abstract space, where a coefficient of the *Eigenface* is actually more meaningful than an image alone, as the *Eigenfaces* express abstract features that are (mathematically speaking) of higher significance. Thanks to the dimensional reduction, we also significantly decrease the amount of parameters needed to create a model that can handle the complexity of the problem. In fact, one does not even need a NN, as one can now solve this problem with a simple Support Vector Machine (SVM)! Throughout this thesis, we have even proven that the PCA approach for cat recognition is suitable for a Micro Controller Unit (MCU). We have run cat recognition on a STM32-Arm Cortex M4F, using only 43.06KB of ROM and 3.52KB of RAM (for the whole code) while having a run time of only 9.3ms at 80MHz (744000 cycles) clock speed. This has been done as a subtask for a separate course, Machine learning on Microcontrollers, at the Swiss Federal Institute of Technology (ETH) [26].

The PCA is a more traditional recognition approach, as it allows for unsupervised learning abilities, such as detecting a new face. It's shortcoming however, is, that it is not very flexible; a face can only be recognized, if the transformation is built for that case. Meaning, that if the transformation has only been instanced with full frontal views, then an image from any other perspective will not function properly. This is where a deep NN approach provides higher resilience.

Deep Convolutional Neural Network Approach

As previously mentioned, the resilience and robustness of a deep NN is a very significant strength in a real life usable image recognition system, as the images will not always be perfectly aligned. There exist multiple literature and state of the art recognition approaches, involving Triplet loss [27] or Siamese Networks. These approaches have the benefit of being able to compare image similarities and perform clustering. Unfortunately these implementations could provide a whole thesis on their own. Which is why we remodeled this task to a simple classification task consisting of the cat probands as classes.

Architecture

Once again the MobileNetV2 model is used as a base model and a small fully connected dense network is placed on top as a feature extractor. We created a Cat Recognizer (CR) model for each node, meaning that the number of neurons in the output layer differ from node to node. Since there are three target cats for Node1, two for Node2 and four for Node3. In any way, the number of output neurons is determined by the number of target cats. Meaning that we classify each cat with the output of the CF stage. A cat can then be viewed as recognized if it reaches a certain threshold, if it does not, it is an intruding cat. Therefore the CR stage takes the output image of the CF stage as input and outputs the cats class.

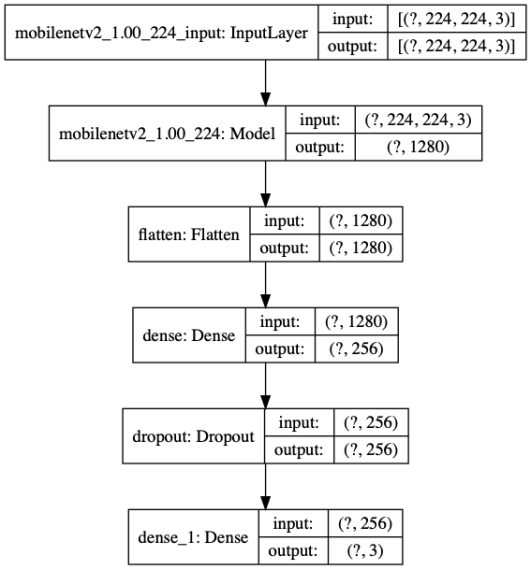


Figure 3.26.: Architecture of the CR model from Node1, there are three cats targeted for said node.

Training

The training section is fairly straight forward for a classification task. As already mentioned, for each node we train an own classifier, therefore the training data of course differs from node to node. However we paid attention to provide approximately 120 images of each cat/class as training instance. Further we utilise the Keras *ImageDataGenerator* to feed the NN and simultaneously perform data augmentation on the training images, which again increases noise resilience and robustness. In Fig. 3.27 one can see a sample of the training images for the CR stage (mix of Node1 and Node2). All training images have been obtained by the output of the CF stage and use the labels provided by the local archive, as described in Section 3.1.4.

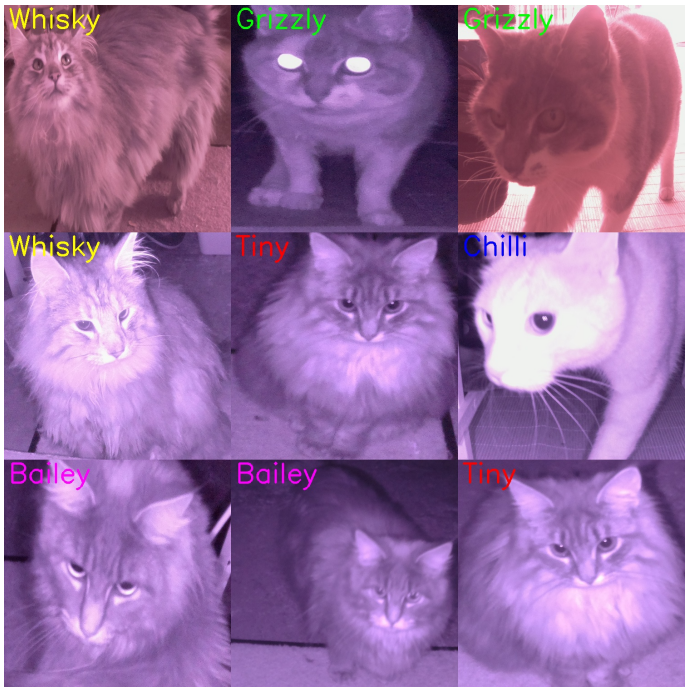


Figure 3.27.: Sample of CR training data, sourced on own data.

Cat Recognizer Stage Conclusion

With the CR we have investigated multiple recognition methods. In the end we opted for a CNN since in a real life deployment the images are not perfectly aligned and the CNN approach compensates this with its noise resilience and robustness. Again as mentioned, the chosen classifier approach is not perfect and not very elegant. It is of course not scalable and not ready for the market, if this smart catflap were ever to be sold, as a user would have to first train a model with approximately 120 images per cat! A solution would be to utilise the Triplet loss or a siamese NN, which are built exactly for said reason. On the other hand, we did not built this model to be ready for market,

and the complexity of the aforementioned approaches are very high. Further the model does not have to be scalable, as we expect the number of cat classes to remain. Therefore the capability of the model mirrors the usecase, but there would definitely be more room for optimisation.

3.2.5. Cascade

Now that all stages have been introduced, one can demonstrate the whole cascade in Fig. 3.28. The reasoning behind said cascade, has been described in the beginning of Section 3.2. The system is now capable of processing a full resolution image of any size, locating the cat, subsequently finding its snout and lastly predicting if the snout contains a prey or not. One can clearly view the different output states, that the cascade provides in Fig. 3.28, the *prey-state*, the *no_prey-state* and the *Don't-Know-state*. The *Don't-Know-state* has not been mentioned frequently in the description of the single stages. It is used either; When the CF stage does not find a cat, resulting in a path called *P1*, this occurs most of the time as the cat is not in front of the catflap all the time; When the CF stage finds a cat, but the SF stage fails to find the snout, resulting in *P3*, this occurs when a cat is exiting the catflap and only the rear is visible, or the cat is not looking at the camera directly; And of course the *Don't-Know-state* can result as an error of the CF and SF stages, this however does not occur often, as seen in Chapter 4.

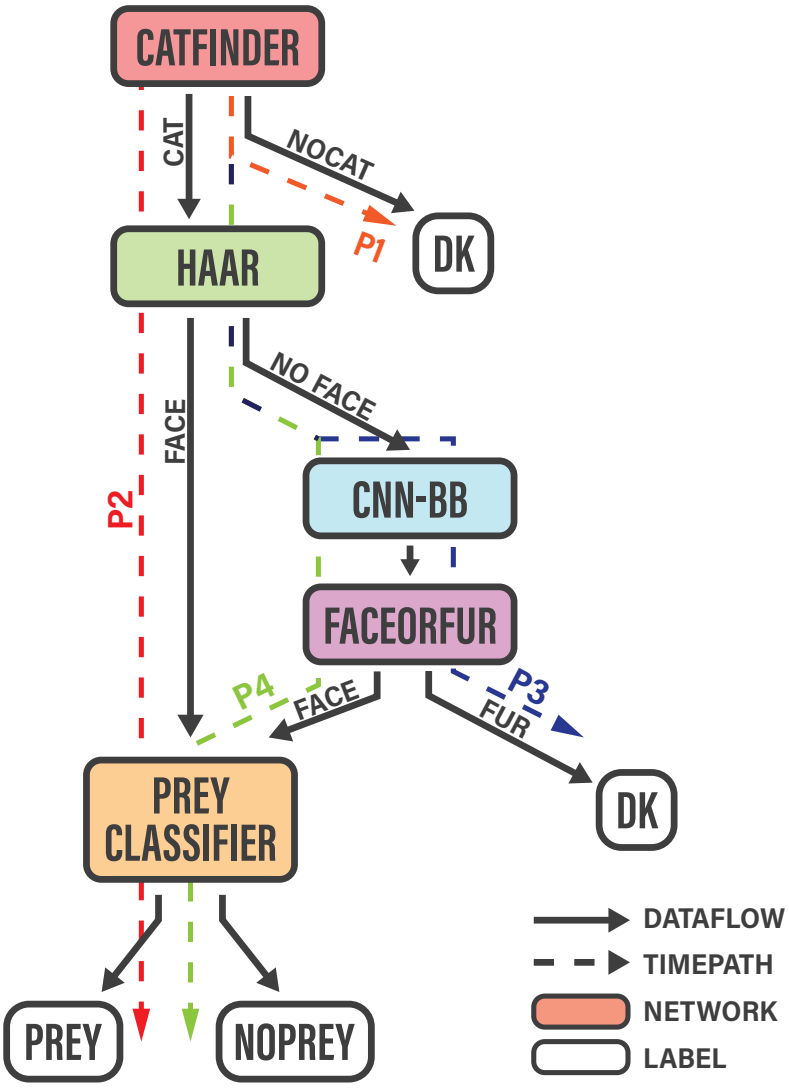


Figure 3.28.: Overview of the cascade.

Path	Processtime
P1	507 ms
P2	3734 ms
P3	2035 ms
P4	5481 ms

Table 3.2.: The mean inference time of the different paths on a RPI4 [28]

3.3. Edge Implementation

The cascade of the different NN is the core of this thesis, nonetheless the cascade has to be implemented in a catflap, otherwise it would not offer any functionality. To do so, one has virtually two options; run it in the "cloud", similar to an Amazon Alexa device; run it on the edge, meaning, that we predict locally on an edge computing device.
The "cloud" solution would be, that the catflap acts as an image provider for a back end GPU, however the transmission overhead of the multiple high resolution images must be taken into account. Yet this would have been a viable solution.
The edge solution running the cascade on a CPU, such as the aforementioned 2.3 GHz Quad-Core Intel Core i5, is not an appropriate solution, as this would imply the catflap to be powered by a full spec computer. Thus we intended to run the cascade locally on the catflap by modifying the CN from Section 3.1.1. This conveys handling of the runtime on a RPI4, which of course takes longer than on a state-of-the-art CPU. One now has to handle the trade off between latency and catching every event.

3.3.1. Asynchronous Queue

The goal is to process the images taken from the camera at a framerate as high as possible. In order not to miss any approaches from cats to the flap, a framerate of at least 2 images per second has empirically been determined. Feeding images through the whole cascade takes between 0.5 s and 5.5 s (including post model inference overhead) as can be seen in Table 3.2. This is too long, compared to the capturing interval of 0.5s. Since the RPI4 is not capable of processing the images at such a high framerate, a workaround has been implemented. The asynchronous queues seen in Fig. 3.29, illustrates this process. Only every second image gets fed through the CF (Step 1). If the output is negative (label: *no cat*) at Step 2, the old images from the camera queue are popped (Step 3). The process is repeated in Step 4. If the output is positive (label: *cat* at Step 5) a batch of the following images of the camera is getting processed (Step 6). The CNN-Queue takes images from the camera queue until it is sure to tell, if there is a prey or not (Step 7). At this point the whole camera queue gets cleared and it starts over. Therefore a cat detection of 1 Frame per Second (FPS) is guaranteed. As soon as a cat is detected, the cascade provides an inevitable average overhead of 9.6 s, that we deem acceptable for a cat to wait before a decision is established.

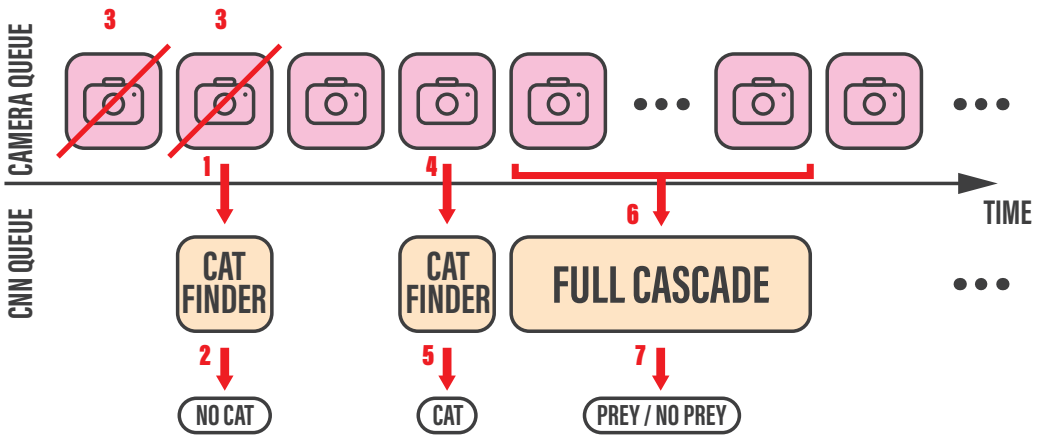


Figure 3.29.: Illustration of the asynchronous queue.

3.3.2. Policy

From a statistical view point, the problem that this thesis tries to solve, is the classical baysian probability paradox of detecting a very rare disease with a very accurate test. Imagine a disease that only 3% of people have and a medical test that has an accuracy of 86%. If one tests positive to the test, the probability of actually having the disease is only 19.5% Eq. (3.1).

$$\begin{aligned} \Pr(D) &= 0.03 \\ \Pr(T|D) &= 0.86 \\ \Pr(D|T) &= \frac{\Pr(T|D)\Pr(D)}{\Pr(T)} = 19.5\% \end{aligned} \tag{3.1}$$

The careful reader might have realized that said numbers are in fact the ones of the exact problem the thesis is trying to solve. According to [5] the probability of the cat having a prey is 3% and the accuracy of the PC is 86% on the validation set. Thus one can see that it is nearly impossible to clearly infer by just one single image. The difficulty arises through the nature of the rarely occurring prey event, namely $\Pr(D) = 3\%$. However, by performing a second test on an independent image, we drastically increase the probability; $\Pr(D|T) = 59.8\%$. And by applying a third test; $\Pr(D|T) = 90.1\%$. Therefore the logical approach is to perform multiple tests. Thus a policy has to be established on how to handle an incoming cat, as it is not always certain how many tests that can be performed, since the cat does not provide for a constant series of full frontal images. We chose the following policy: *A cat has to prove that it has no prey.* Thus we keep the door locked until we reach a certainty that deems us acceptable. To perform the wrapper logic over multiple images, we developed two approaches:

Bayesian Approach

In the bayesian approach, we utilise a bayesian network that calculates the probability of $\Pr(M = t|T = t)$ over multiple images/tests. The mathematical details can be viewed in [29]. Basically, we calculate what the probability of a prey is, when x images are classified as *prey* and y images are classified as *no_prey*. If the probability of $\Pr(M = t|T = t)$ is larger than 50%, then the wrapper logic predicts *prey*. The probabilities that have to be known for this approach are; the probability of a prey M (M for mouse) $\Pr(M = t) = 0.03$; the accuracy of the prey prediction $\Pr(T = t|M = t) = 0.86$ (value taken from validation accuracy).

Cummuli Approach

What the baysian approach does not consider, is the certainty of the predictions, as it treats the probability distribution as a binomial distribution. Thus the cummuli approach accumulates "certainty-points". For example if the PC stage predicts a value of 48% and we set our *prey/no_prey* threshold at 50%, the cat earns 2 points; these points will accumulate (hence the name cummuli approach) until it reaches a threshold where the cat has proven not to have a prey. The same way, a cat can of course accumulate negative points. After evaluating the results in Chapter 4, the approach was reformulated such that it also weighs the cummuli points by the amount of images accumulated, transforming the cummuli approach to a weighted average of "certainty-points".

Chapter 4

Results

4.1. Data Gathering

Hardware

The third and final version of the CN is shown in Fig. 4.1. The case is designed in Fusion 360 [30] and then printed with an Anet A8 3D printer [31] using transparent PET Filament. The camera [15] performed well. The images from the infrared sensor were mostly bright enough, even during the night. Yet, the motion sensor [16] worked quite inconsistent. It did not always trigger at the same location, sometimes even too late, such that the camera only managed to capture one picture of an approaching cat.

Figure 4.1.: Final waterproof version of the Camera Node, including rain cover and tilt mechanism for the camera.

Network

The network and communication described in Section 3.1.3 was very stable, even though the nodes were deployed in different households. The Telegram-Bot reliably informed users as soon as connectivity was lost. The node itself saved the images on the SD card, until it managed to get an internet connection again and uploaded the images immediately to the MS. The whole system is extremely reliable and did not need a lot of maintenance.

Label GUI

The Label GUI works on every Unix based machine, is intuitive and handles all the data transfer by itself. It saved a lot of time during the labeling process.

Statistics

The following table Section 4.1, represent a sample of key statistics that characterised the data gathering network throughout the duration of this thesis.

Labeled cat images	50’610
Cats looking perfectly into camera	5’412
no_prey images	5’369
prey images	43
Network Traffic	442 GB
Final Archive	40 GB

4.2. Convolutional Neural Network Cascade

The evaluation of the single cascade stages has partially been discussed in Chapter 3. As a whole evaluation of the cascade, we chose to apply a black box approach. As discussed in Section 3.3, we implemented a policy that infers a prediction over multiple images, called an event. Therefore we will apply the black box approach on both, the single and the event based cascade. The black box takes a single image or an event as input, and outputs one of three states; either *prey*, if the cascade decided that the input is a prey; or *no_prey*, if the cascade decided that there is no prey; or *don’t know*, if the cascade could not find a cat or face.

Single Cascade

The evaluation of this single cascade is performed by running the full cascade on single images, that are sampled from 152 images containing 105 *no_prey* images and 47 *prey* images. These images represent ground truth of this evaluation. The network takes a single image as input and will return either *prey*, *no_prey* or *don’t know* as previously mentioned and viewed in Fig. 4.2.

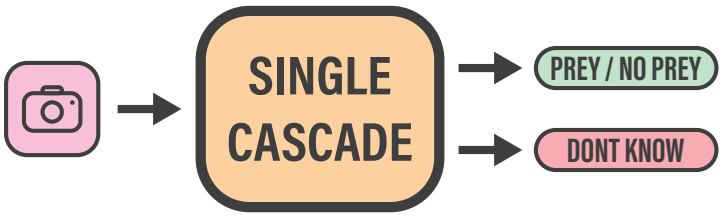


Figure 4.2.: Illustration of the single cascade.

First we will only consider the *prey* and *no_prey* outputs, these are plotted in Fig. 4.3. The left hand graph represents the Precision-Recall curve, that is ideally used to describe a models skill addressing an imbalanced problem. The threshold in this curve is the value of the PC, therefore the graph shows how precision and recall would vary on a given threshold. Now the operator would have to decide on how important which metric is; does one want a high recall (no prey would be let in)? How much does one care, that the cat is not falsely locked out (how much should one trust the prey classification)?

If the Precision-Recall curve does not yield a point at (1,1), then such a tradeoff must always be done. We chose the point depicted in the graph. It represents a PC threshold value of 0.47 and consequently a recall of 93.9% and a precision of 47.6%. Thus this system will detect 93.9% of all prey cases and the probability of a *prey* output being true will be 47.6%. Interestingly enough, the models skill seems to vary around the F1-isobar of 60% for all thresholds.
On the right hand side of the figure, the ROC curve is depicted. We decided to plot this graph, because it is a very well known metric. The depicted point represents the same threshold value as in the Precision-Recall curve and since TPR and recall are the exact same metric, the TPR of course has the same value of 93.9%. However FPR and precision describe different metrics. With this threshold, the FPR yields 56.6%, meaning that 56.6% of the time where the cat does not have a prey (ground truth), it will be falsely locked out. Thus, we will capture 93.9% of all preys and falsely lock out the cat 56.6% of the time. Keeping in mind, that the cat will only return with prey in 3% of the cases (which we would correctly classify 93.9% of times) and return 97% of the time without prey (which we would falsely classify 56% of the time), we would apply a great nuisance on the cat. Yet in both plots, the no skill level is represented by the blue dashed line. It is clearly visible that the models skill outperforms this threshold largely; it clearly shows skill, in this difficult problem.

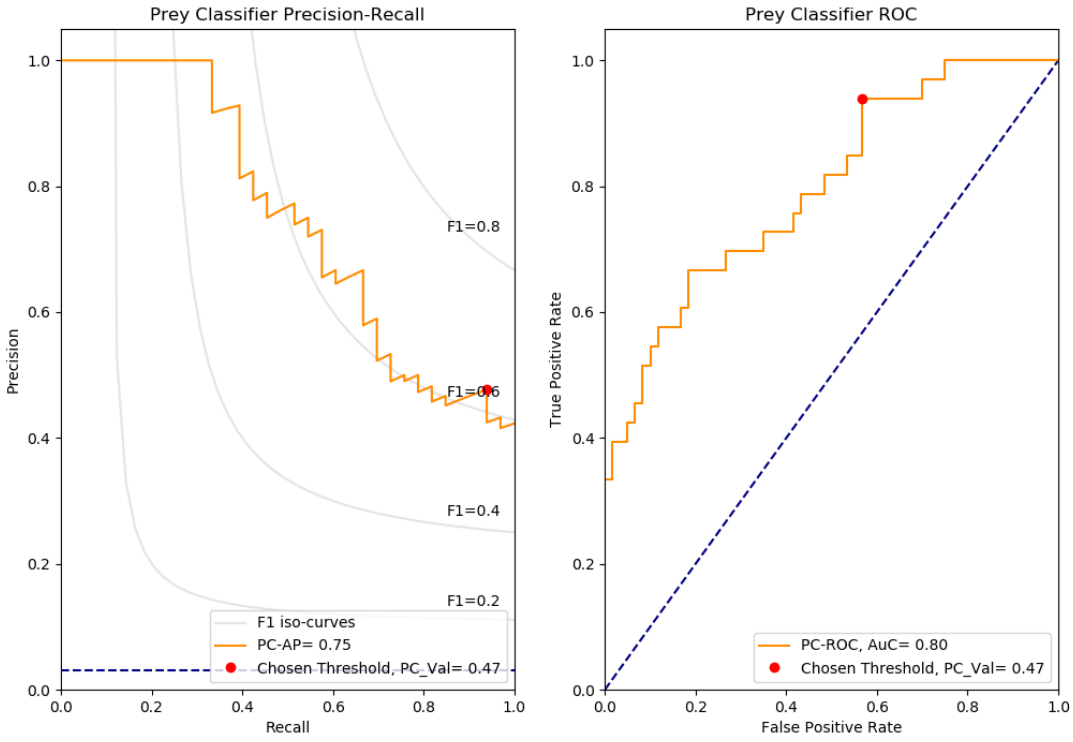


Figure 4.3.: ROC and Precision-Recall curve of the single cascade evaluation.

By applying the PC probability threshold gained from the chosen threshold of 47% in Fig. 4.3, one can now plot the confusion matrix of this evaluation. Lastly, as a result of this evaluation, the cascade returned 59 *don't know* states, meaning that the cascade failed to either detect the face or the cat on its own in these images. This results in a *don't know* ratio of 38.8%. Thus the single cascade will fail to detect the snout of the cat in 38.8% of the cases. In the other instances, it will correctly detect 93.9 % of all *prey* instances while falsely locking the cat flap 56.6% of the *no_prey* instances. This behaviour is to be expected, as is described in Section 3.3.2, as the single cascade can only probe a single image. The *don't know* ratio and the high FPR are both addressed by the event cascade in the following section.

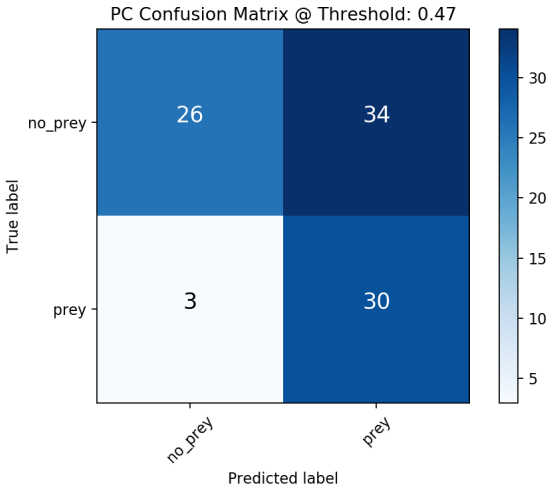


Figure 4.4.: Confusion matrix of the single cascade at a PC probability threshold of 47%.

Event Cascade

For the evaluation of this system, we sampled 58 random events with 15 images per event from the archive of the MS. It is ensured, that none of the cascade stages have ever used these images during their training. This set consists of 42 *no_prey* events and 16 *prey* events resulting in 870 images, these represent the ground truth values. The network is behaving as visualized in Fig. 4.5.

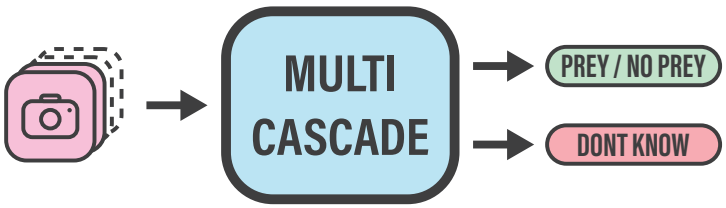


Figure 4.5.: Illustration of the multi cascade.

The output of this measurement is visualized in Fig. 4.6. The left side Precision Recall curve shows, that the cummuli approach yields a better result than the baysian approach. The cummuli approach slightly varies around the the F1-Isobar of 65%. Again, it is the operators choice to decide the trade-off between precision and recall. Does one want to prevent prey entries at any cost? Then a recall of 100% should be chosen. Does one rather care about keeping the number of false lockouts for the cat low? Then one has to consider a higher precision. In our case, we opted for a threshold that resembles the cummuli value of 2.96, meaning that at a cummuli value of 2.96 we deem it acceptable

for the cat to enter. This yields a recall of 93.3% and a precision of 53.8 %. Meaning that this system will keep out 93.3% of all prey entries and that a prey inference has a probability of 53.8% of being true. In any case, the no skill threshold is represented by the blue dashed horizontal line. This demonstrates, that the cascade is well above the random guessing threshold.
On the right hand side of Fig. 4.6, we have again visualized the same threshold point. The TPR at the given threshold, once again represents 93.3% (as TPR and recall are the same metric). Yet FPR and precision are not the same, as explained in Section 2.1.2. Within this graph, the FPR yields 28.5% at the given threshold and states that 28.5% of the time where a cat without prey enters, it will be falsely locked out, which is a huge improvement against the single cascade. Once again the no skill border is marked by the blue dashed diagonal in the ROC plot, thus this system clearly shows skill.

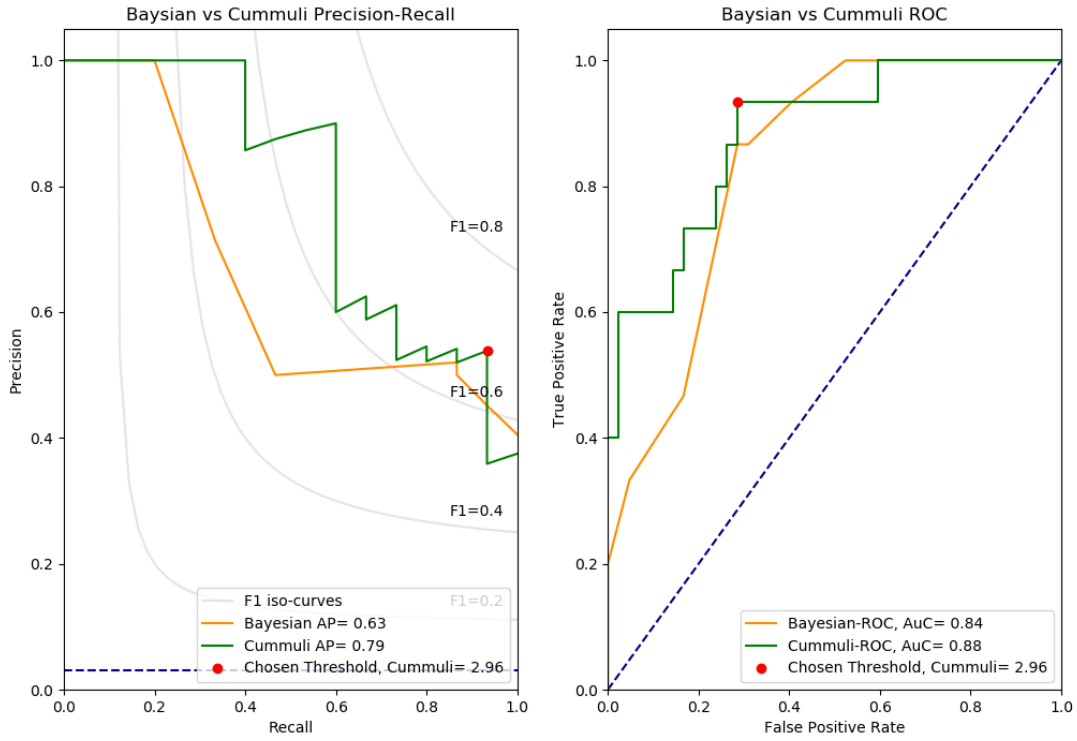


Figure 4.6.: Precision-Recall curve and ROC curve of the event cascade evaluation.

By applying the threshold gained from Fig. 4.6, it is now possible to plot the confusion matrix of the evaluation. It is to mention, that the matrix represents the numbers of events, which constitutes 15 images per unit. The confusion matrix itself does not further express the models skill, it is simply shown for data transparency reasons.

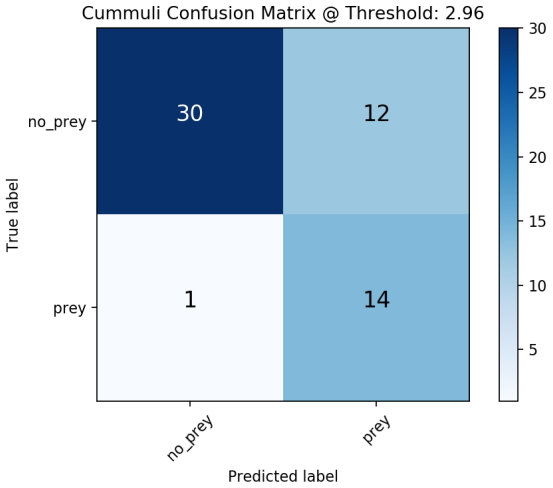


Figure 4.7.: Confusion matrix of the event cascade approach. The shown numbers represent events of 15 images.

Yet these plots only represent the *prey* and *no_prey* output states. The *don't know* state is applied, if there are no faces detected within an event. Due to the policy of the cat having to prove innocence, the door will still be locked. However, in event cascade, the ratio of *don't know* state is significantly lower as in the single cascade, as there are of course multiple images to evaluate. In this case, there has just been one single *don't know* output (the event cascade could not detect a snout in said event), resulting in a 1.7% ratio to the ground truth.
Thus the event cascade will be able to perform a *prey*/*no_prey* classification 98.3% of the times. It will correctly detect 93.3% of all prey entries and falsely lock out the cat 28.5% of the times, where it truly was a ground truth *no_prey* state. Therefore the event cascade provides a large improvement with the *don't know* ratio and lowers the FPR significantly compared to the single cascade.

4.3. Edge Implementation

As stated in Section 3.3, a capturing framerate of 2 images per second is guaranteed. A framerate of 3 fps would still be feasible, if the asynchronous camera queue would only feed every third image to the CNN queue in Fig. 3.29. In order to evaluate, if every cat gets recognised by the edge, we placed two nodes next to each other in front of the catflap. One is running the code of the data gathering part Section 3.1.1, the other CN is running the code from Section 3.3. We declared the "data gathering node" output as the ground truth, as it has no timing constraints and worked very reliably throughout the data gathering process. With this knowledge, we can test how often the edge is

missing an event, by comparing the events of the two nodes against each other. It has been empirically determinde, that no event is lost at the edge.

A time overhead is inevitable as the inference time of path P4 in Fig. 3.28 is larger than the capturing framerate. Yet the overhead from when the cat is first detected and when the event cascade gained enough cummuli points to let the cat in, is on average 9.6s. This means, that if a cat wants to enter, and the cascade infers a *no_prey* state, it has to wait 9.6s until it is let in.

Chapter 5

Conclusion and Future Work

In this thesis we have demonstrated a CNN based CV approach to detect, if a cat wants to enter a catflap with or without a prey in its snout. From a ML perspective this is a highly imbalanced problem, as a cat is expected to enter the catflap with prey only 3% of times. Our approach works with any cat, in any environment and is therefore generalizable, due to the TL training efforts taken. Consequently, an end-to-end approach is not feasible due to the resolution limitations of the TL base models and the lack of training data, which is why this thesis addresses a cascade of multiple transfer learned models. As an endeavor of image data collection for this highly specific task, a reliable, scalable and centralized data gathering network has been implemented, that provided 40 GB of training data. It has been evaluated that the optimum solution involves an event based cascade, capable of inferring on multiple images of the cat entering, due to baysian restrictions of the imbalanced problem. The event cascade is further capable of running on an off-the-shelf RPI4 at an average detection framerate of 1 FPS. This is achieved by utilizing an asynchronous queue between parallelized processes and dynamically adjusting the processingrate of said queue. The wrapper logic that finally concludes a verdict between multiple images, acts on the policy: *The cat must prove that it has no prey*. During an entry a cat will therefore accumulate trust points based on the certainty of the classifier. As soon as the cat amasses enough cummuli points, it has proven *no_prey*.
The evaluation of the event based cascade yields, that the cascade will find the cats snout in 98.7% of cats entries. If the cascade finds the snout, its *prey/no_prey* classification returns a recall rate of 93.3% and a FPR of 28.5%. Meaning that from the 3% of events, that a cat enters with prey, 93.3% of entries will be correctly detected as prey. While during the other 97% of instances, where the cat enters without prey, the cascade will falsely lock the cat out with a probability of 28.5%. These thresholds can be adjusted by a user according to the Precision-Recall curve and ROC curve of the cascade. Due to the edge implementation of the system on an RPI4, a time overhead is inevitable. Yet the implementation exhibits a mean overhead of 9.6 s until a conclusion is achieved. Consequently a cat must wait an average of 9.6 s before it is let in, if the cascade concluded in *no_prey*.

The event cascade might be scalable (apart from the CR stage), however it is not yet ready for series production. Especially the CR stage would greatly benefit from a state-of-the-art approach of face clustering via image similarity [27]. Another point for future work, would be to implement the PC stage on a less memory and computationally expensive base model than the VGG16 network. As this is the stage, that largely outweighs the other stages timewise. Additionally, zero/few shot techniques as in [32] could be further investigated, which could greatly improve performance, regarding the low number of prey images, obtained during the data gathering process.

Appendix A

Machine Learning Tips and Tricks

Imbalanced Classification

When posed with an imbalanced problem, there are three main methods:

- Undersampling: Use less of the majority class, such that one solves a balanced problem.
- Oversamplng: Duplicate the minority class, such that one again solves a balanced problem.
- Classweight: Use all of the data. Weight the changes during a minority class sample higher than the majority class sample. If you are using Keras this is a very good tutorial on this matter [33].

We recommend the classweight method, as it allows the network to train on the same amount of minority instances as in undersampling, while simultaneously learning even more about the majority class.

Transfer Learning

If you perform Feature Extraction as mentioned in Section 2.1.3, then it really makes sense to train the model with the base fixed, since the weights of the top layers are probably initialized randomly and need to be adjusted much stronger than the weights in the layers underneath. In our case it worked best, when cutting the model at the bottleneck. Most of the CNN's are built in blocks, which consists of a few convolutional and pooling layers. Between those blocks, most of the time there is a bottleneck. Start your Feature Extraction / Fine Tuning at a bottleneck, and not just at a random layer. Before you Fine Tune you should make sure, that your base has a good accuracy and is not improving any more with your learning rate.
Choosing the right dense layers in our case was just a trial and error task. But do not

try to add more neurons to a dense layer, than the flattened output of the previous layer provides. Make sure to use a non linear activation function on the dense layers, otherwise they all collapse in a linear combination of the output layer.

Base Model Choice

When applying TL methods, choose the base model according to your needs. If inference time is of importance, an SSD should be considered, otherwise experimenting with different architectures is always a valid answer. In the case of the PC we tried multiple different architectures from MobileNetV2 to ResNet, yet the only architecture that worked well was VGG16.

Optmizer

Sometimes during training of the multiple stages, we encountered that RMSProp can yield different and better results then Adam, contrary to [34] and [35]. Our advice would be to start with a vanilla optimizer such as SGD as a sanity check. Once this works correctly try the same approach with an adaptive learning method such as Adam or RMSProp.

Data Preprocessing

Train your model with as much high quality data as possible, try to avoid preprocessing the training data such that a human could infer clearer than before. For example, when training the SF stage we performed background substraction as in Fig. 3.14, as a preprocessing step such that the cat is easier to distinguish then before. However the pretrained models have never encountered such data and therefore did not perform well when dealing with those images. Also do not try to manipulate the images with filters. We encountered this problem during the training of the PC stage, where we applied a filter in Photoshop to every image of the internet dataset, in order to produce an infra red effect. But the results were much worse than without the filter.

Appendix B

Task Description

Image Classification of Cats with Prey to be used in a Smart-Catflap

February 7, 2020

1 Introduction

Cats like to hunt mice, birds and everything else that moves. One of the most common problem for cat owners is that cats devour their prey in the house. This occurs a few times per month and always leads to an annoying cleanup mission, or the owner hunting down a foul smell, as the prey has been tossed underneath the sofa. A solution to this problem would be a smart-catflap, that is able to detect if the cat has prey in it's snout. Similar to a recent article of an Amazon engineer [1]. This can be done via Computer Vision (CV) by utilising state of the art zero/few-shot-learning techniques [2].

2 Goal

The goal of this work is to implement an AI-based classifier for cats with and without prey. It will be divided into a data-collecting part and an implementation part for the AI.

3 Tasks

1. Design a platform to gather images and upload them onto a server. This platform could also be used later as the actual classifier, after the model is trained.
2. Making yourselves familiar with Transfer-Learning, Zero/Few-Shot-Learning [2] [3].
3. Implement a classifier; either by Transfer-Learning or by pure relying on our own images.

4 Kind Of Work

30% Theory, 70% Implementation

List of Figures

List of Tables

Bibliography

[1] N. Mukai, Y. Zhang, and Y. Chang, "Pet face detection," in *2018 Nicograph International (NicoInt)*, 2018, pp. 52–57.

[2] W. Zhang, J. Sun, and X. Tang, "Cat head detection - how to effectively exploit shape and texture features," in *Computer Vision – ECCV 2008*, D. Forsyth, P. Torr, and A. Zisserman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 802–816.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I.

[4] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[5] B. Hamm, "An amazon employee made an ai-powered cat flap to stop his cat from bringing home dead animals," https://www.theverge.com/tldr/2019/6/30/19102430/amazon-engineer-ai-powered-catflap-prey-ben-hamm, accessed: 2020-01-03.

[6] H. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[7] "Roc curve," https://mc.ai/roc-curves-and-precision-recall-curves-for-imbalanced-classification/, accessed: 2020-06-12.

[8] J. Brownlee, "How to use roc curves and precision-recall curves for classification in python," https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python, accessed: 2020-11-06.

[9] "Mobilenet v2," https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet, accessed: 2020-06-03.

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

[11] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," 2015, cite arxiv:1512.02325Comment: ECCV 2016. [Online]. Available: http://arxiv.org/abs/1512.02325

[13] Z. et al, "Cat-lmks-ds," https://archive.org/details/CAT_DATASET, 2008.

[14] kairess, "cat_hipsterizer," https://github.com/kairess/cat_hipsterizer, 2018.

[15] "Infrared joy-it camera," https://joy-it.net/de/products/rb-camera-IR_PRO, accessed: 2020-01-03.

[16] "Pir sensor," https://www.distrelec.ch/de/pir-bewegungssensor-12v-adafruit-189/p/30139067?channel=b2c&price_gs=11.4162&source=googleps&ext_cid=shgooaqchde-na&gclid=CjwKCAjw5cL2BRASEiwAENqAPpW1-cLZRwk3KMeXkVkZu0h-tBDS3XIBEsy5blTtPnmhmyjE_bGSjRoCZFUQAvD_BwE, accessed: 2020-05-29.

[17] "Flask," https://flask.palletsprojects.com/en/1.1.x/1, accessed: 2020-06-15.

[18] "Telegram bot api," https://core.telegram.org/bots/api, accessed: 2020-06-15.

[19] "Wireguard," https://wireguard.com/, accessed: 2020-06-12.

[20] qqwweee, "keras-yolo3," https://github.com/qqwweee/keras-yolo3, 2018.

[21] "Yolo: Real-time object detection," https://pjreddie.com/darknet/yolo/, accessed: 2020-06-03.

[22] "Tensorflow model zoo," https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md, accessed: 2020-06-15.

[23] "Opencv," https://opencv.org/, accessed: 2020-06-05.

[24] OpenCV, "Canny edge detection," https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html, 2008.

[25] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1991, pp. 586–591.

[26] M.Magno, "Machine learning on microcontrollers," https://iis-students.ee.ethz.ch/lectures/machine-learning-on-microcontrollers-page/, 2020.

[27] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering." *CoRR*, vol. abs/1503.03832, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1503.html#SchroffKP15

[28] "Raspberry pi 4 model b," https://www.raspberrypi.org/products/raspberry-pi-4-model-b/, accessed: 2020-06-09.

[29] "Bayes theorem to bayes networks," https://towardsdatascience.com/will-you-become-a-zombie-if-a-99-accuracy-test-result-positive-3da371f5134, accessed: 2020-06-20.

[30] "Fusion 360," https://www.autodesk.ch/de/products/fusion-360/subscribe?plc=F360&term=1-YEAR&support=ADVANCED&quantity=1, accessed: 2020-06-12.

[31] "Anet a8," https://www.3dmake.de/3d-drucker/anet-a8-bausatz/, accessed: 2020-06-12.

[32] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," https://arxiv.org/abs/1703.05175, p. 1, 03 2017.

[33] Tensorflow, "Classification on imbalanced data," https://www.tensorflow.org/tutorials/structured_data/imbalanced_data, accessed: 2020-15-06.

[34] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016. [Online]. Available: http://arxiv.org/abs/1609.04747

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[36] D. Wertheimer and B. Hariharan, "Few-shot learning with localization in realistic settings," http://openaccess.thecvf.com/content_CVPR_2019/papers/Wertheimer_Few-Shot_Learning_With_Localization_in_Realistic_Settings_CVPR_2019_paper.pdf, p. 1, 04 2019.

[37] W. Zhang, J. Sun, and X. Tang, "Cat head detection - how to effectively exploit shape and texture features," in *ECCV*, 2008.

[38] "Imagenet," http://www.image-net.org/, accessed: 2020-06-05.

[39] H. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.