Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Image Analysis Approaches for Parasite Detection on Honeybees

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Stefan Schurischuster, BSc
Matrikelnummer 1026040

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: PD DI Dr. Martin Kampel
Mitwirkung: PhD Sebastian Zambanini
PhD Beatriz Remeseiro

Wien, 1. August 2018

Stefan Schurischuster

Martin Kampel

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Image Analysis Approaches for Parasite Detection on Honeybees

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Medical Informatics

by

Stefan Schurischuster, BSc
Registration Number 1026040

to the Faculty of Informatics

at the TU Wien

Advisor: PD DI Dr. Martin Kampel
Assistance: PhD Sebastian Zambanini
PhD Beatriz Remeseiro

Vienna, 1st August, 2018

Stefan Schurischuster

Martin Kampel

Erklärung zur Verfassung der Arbeit

Stefan Schurischuster, BSc
Braunhubergasse 12/1/1, 1110 Vienna, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. August 2018

Stefan Schurischuster

Danksagung

Ich möchte mich bei all jenen bedanken, die mir bei der Durchführung dieser Arbeit zur Seite gestanden sind. Dabei möchte ich unabhängig von der Reihenfolge, folgende Personen hervorheben:

Vielen Dank an meine Betreuer Martin Kampel, Sebastian Zambanini und Beatriz Remezeiro für die ausgiebige Hilfe bei der Durchführung dieser Arbeit.

Ein herzliches Dankeschön geht auch an Benjamin Lamp und Kerstin Seitz vom Virologie Institut der Veterinärmedizinischen Universität Wien. Ich erinnere mich gerne an die ausgezeichnete Zusammenarbeit und ständige Unterstützung beim Prototypendesign und der Datenaufzeichnung.

Danken möchte ich auch den Mitarbeitern des Computer Vision Labs der TU Wien, sowie der Firma Cogvis, die bei der technischen Umsetzung zur Hilfe gestanden haben.

Danke an meine Familie und Freunde für die Geduld und den Rückhalt den ihr mir geboten habt. Besonderer Dank gilt meinem Bruder Rene Schurischuster, der mich in der finalen Phase dieser Arbeit tatkräftig unterstützt hat.

Kurzfassung

Das rapide Wachstum von Parasiten wie, der Varroa Milbe (*Varroa destructor*), ist einer der Hauptgründe für die erhöhte Sterblichkeitsrate von Bienenkolonien. Bienenzüchter müssen händisch zeitaufwendige Proben und Behandlungen durchführen um diesem Sterben entgegenzuwirken. Der Befallstatus einer Kolonie wird durch visuelle Kontrolle von Stichproben durchgeführt. Die Ergebnisse basieren dabei auf statistischen Hochrechnungen der Proben zum Milbenbefall und werden teilweise durch invasive und zeitaufwändige Maßnahmen erbracht. Das Ziel dieser Arbeit ist es zwei unterschiedliche Klassifikationsansätze zu vergleichen und auf die Klassifizierung von Bienen anzuwenden. Hierfür wurde ein Kamerasystem entwickelt, das in der Lage ist kontinuierliche Aufnahmen des Einganges einer Bienenkolonie anzufertigen. Von dem gefilmten Videomaterial wurde händisch eine Datenbasis mit mehr als 13,000 Bildern von infizierten und gesunden Bienen erstellt. Diese wurde weiterfolgend zum Trainieren und Evaluieren der beiden Klassifikationsansätze verwendet. Verglichen werden ein "traditoneller" Machine Learning Ansatz mit einem Deep Learning Ansatz. Die finale Evaluierung zeigt, dass die automatische Unterscheidung von infizierten und gesunden Bienen mit Hilfe des präsentierten Kamerasystems möglich ist. Unter der Verwendung des Deep Learning Ansatzes, basierend auf dem erstellten Testdatensatz, ergibt sich eine Erkennungsgenauigkeit von 94.4% für gesunde und 85.5% für infizierte Bienen. Dieser neuartige Ansatz zur Bienenklassifizierung und -überwachung dient als erster Schritt in Richtung voll-automatisierter Überwachung von Bienenstöcken und deren Parasiten.

Abstract

Rapid growth of parasites like *Varroa destructor* is one of the main reasons for elevated mortality of bee colonies. Beekeepers have to perform time consuming manual sampling to enable treatment and avoid colony losses. Most existing sampling plans only produce rough estimates and can be invasive and costly. This can be a significant stress factor, when considering an average sample size of 300 bees per apiary, to get a significant test result. This yields the question, if it would be possible to automatically monitor the infestation status of a beehive, using a non-invasive method. This works provides a first step towards answering this question. Therefore a camera system capable of creating continuous recordings of the entrance of an apiary is designed with whom more than 7TB of video data is recorded. From the conducted video material, a ground truth dataset is created with more than 13,000 manually labeled images of infected and healthy bees. The dataset is used to train and evaluate two detection approaches: A "traditional" machine learning pipeline and a deep learning pipeline using convolutional neural networks. The final evaluation shows that distinguishing between healthy and infected bees is possible using the convolutional neural network approach, providing the proof of concept. A per-class classification accuracy of 94.4% for healthy bees and 85.5% for infected bees is recorded with an overall f-score of 0.82, calculated on the labeled test dataset. This work therefore provides a novelty approach for automatic parasite classification and represents as a first step towards automated parasite monitoring.

Contents

CHAPTER 1

Introduction

Each year, for over a decade now, beekeepers are recording high losses of their honeybee winter populations [STS+15]. This is a global phenomenon, reducing the worldwide population of honeybee *Apis mellifera* continuously [ESM+09]. Researchers and beekeepers are still struggling to find the root causes, but agree on a combination of responsible factors [vM10]. Examples are: The heavy use of pesticides, genetic mono-cultures of bee breedings, diseases particular to honeybees and global climate change [vM10]. A factor to highlight respective to sudden dying of bee colonies is *diseases* [GG15]. Gisder et al. show that there is a correlation between the sudden death of a beehive and infections with viruses and diseases induced by parasites [GG15]. These diseases are mainly transmitted and spread by parasites [BG08]. In particular the *Varroa mite* (*Varroa destructor*) is a world-wide threat, responsible for the deaths of millions of colonies [Mar01]. The effects of a *Varroa* infestation on colonies are also referred to as *Varroosis* which, in summary, makes up the most destructive disease of managed honeybee colonies world wide [BG08].

To counteract and maintain their colonies, beekeepers are performing high frequency manual monitoring of their hives. This is necessary to determine the general health status of each colony which includes identifying the parasite load [LMB+10]. Monitoring is done manually, which comes with a number of drawbacks. Manual testing works on the basis of drawing random samples from a bee population and perform statistical projections on it. These are prone to errors and depend heavily on the sample size taken and the individual procedure used. Additionally, methods are invasive leading to bees being killed during the testing procedure. This is a significant stress factor, when considering an average sample size of 300 bees per test and per hive, to get a significant test result [LMB+10]. Finally, because manual interactions are necessary for each hive, the process becomes time consuming.

There is an obvious need for precise and economic ways of monitoring the health of a beehive. Especially regarding manual sampling. A possible solution is to omit the time consuming and invasive procedure of manual sampling and replace it with an automated

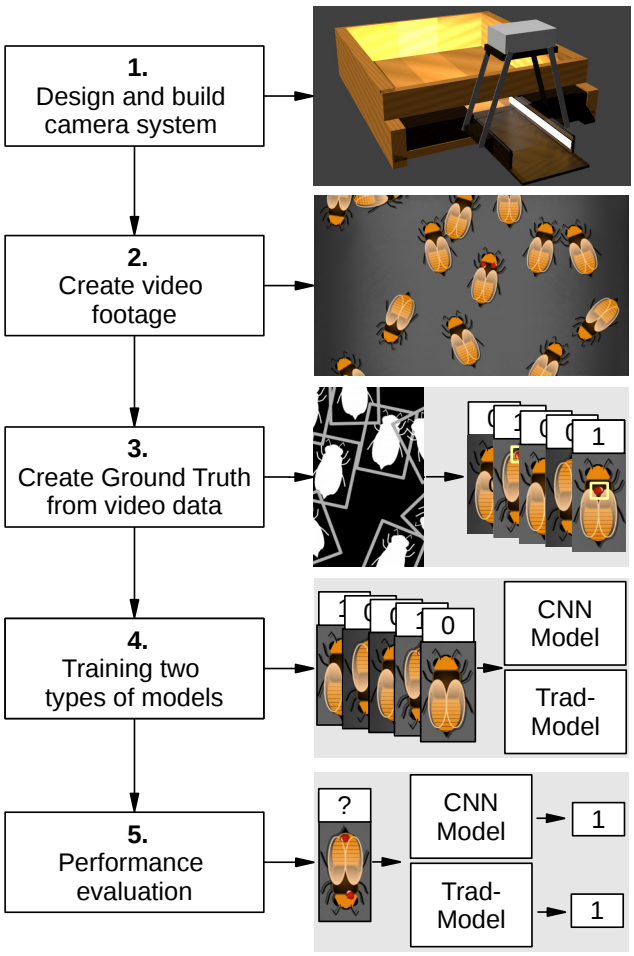


Figure 1.1: The overall project pipeline. The left side shows the five major steps. The right side shows schematic illustrations corresponding to each step.

non-invasive process. This work aims to answer the following question on this behalf: Is it possible to automatically monitor the infestation status of a colony using camera based observation techniques? In particular: Is it possible to detect, whether a honeybee is infected with the parasite Varroa destructor when passing through the entrance of a beehive.

To answer this question, the proof of concept is implemented using a classification approach. The necessary data is provided by using a video camera system which is placed on top of the entrance of a beehive. The idea is to monitor a colony at the hive entrance and film individual bees as they are entering or leaving the hive. For each observed bee a decision is made, whether the bee is infected with a parasite or not. Therefore two machine learning algorithms are trained to distinguish between healthy and infected bees.

The final results of the best performing algorithm provide the proof of concept.

The complete process is schematically explained in Figure 1.1, depicting four major steps. On the right side of each step a visual description is provided, illustrating the contents of this step. It ranges from designing and building the hardware recording system to evaluating the recorded data and algorithms. The individual steps of Figure 1.1 are further explained:

1. **Designing and building of the camera system**: First a video camera system is designed, which is placed in front of a beehive taking continuous recordings of the entrance. The systems design is heavily influenced by the designs of [CYJL12] and [KTP15]. It focuses on filming honeybees when they are entering or leaving the apiary. In the first step of Figure 1.1 the initial design of the proposed camera system is visualized with the bottom of a apiary in the back. It is a 3D - model designed using Blender[1]. The basic idea is to have a transparent tunnel acting as entrance to the hive, which is observed by a camera placed on top of it. The tunnel is further equipped with artificial lighting to maintain a constant and homogeneous background. Finally a micro processing unit is included, performing live computations on the recorded video data. The design is undergoing an evolutionary process yielding three different prototypes.

2. **Creating the video footage**: The second stage consists of testing the prototypes with the goal of recording video data of infected and healthy bees. Each prototype comes with an individual hardware configuration leading to different recording properties. Videos are recorded in laboratory and field setups. In the laboratory setup, different recording conditions as well as different camera sensors are tested. The video data, used to create datasets for training and evaluating the recognition models are recorded here. The field recordings target for experimenting with long term usage, as well as recording video data from beehives.

3. **Creating the ground truth from the video data**: For the training of a recognition algorithm, a manually labeled dataset is necessary. This approach requires datasets to train, validate and test the created models.

   Within the scope of this task a manually labeled dataset with more then $13,000$ manually labeled images is created. The videos acquired in the previous step are processed and images of individual honeybees are extracted. These images are then manually labeled by a human observer, marking “0” for healthy bees and “1” for bees infected with Varroa mites. In case of an infected bees, the position of the Varroa mite is marked on the image. The total labeled dataset is split into three distinct subsets: (1) train dataset, (2) test dataset and (3) validation dataset.

4. **Training two types of models**: The aforementioned datasets are now used to train two types of recognition algorithms: (1) “Traditional” machine learning

[1] `https://www.blender.org/` (last visited 06/2018)

algorithms and (2) Convolutional neural networks. The parameters of the individual models are evaluated using the validation dataset to find the best performing candidate models.

5. **Performance evaluation**: The final step is to apply the best performing trained models on the test dataset. Here the classification performances of both the best performing deep learning as well as "traditional" models are compared.

   Concluding with a final selection of the best performing model capable of differentiating between healthy and infected bees.

The contributions of this work are the following:

- Developing and evaluating three camera recording systems, providing details on how to design such a system. The complete pipeline from theoretical system design to implementation and building of a recording system is presented. This results in three novel camera systems which are implemented as prototypes and used for data recording.
- With these systems 7.01 TB of video data is recorded. Both laboratory and field setups are considered during this process. The detailed configuration of each test session is documented in a structured form, allowing the generation of a data report.
- From the recorded video data a labeled dataset with 13,464 manually labeled bee images is created. For this task, an objection detection and extraction pipeline is presented. Additionally a supporting labeling software tool is implemented using Python, easing the task of manually labeling the extracted images. Both the recorded data, as well as the labeled dataset qualify to be used in future projects and different research questions.
- A comparison between "traditional" machine learning and deep learning technology is applied for classifying the health status of honeybees. The results of this comparison are presented providing the proof of concept by showing the capability of distinguishing between healthy and infected bees.

1.1 Motivation

The main contributions of this work are based on creating an automated monitoring system and the comparison of segmentation and classification methods in this veterinarian context. This context is provided by real world implication of elevated honeybee colony losses that started in the year 2000 [Mar01]. This is still an ongoing problem as recent studies show [NLC16]. This vanishing of honeybees does not only affect the global supply of honey, but in reality global human food supply. The honeybee *Apis mellifera* historically developed to be the most used insect pollinator for fruits and vegetables [M$^+$76], [DMM00].

This is due to their self sustainability and workforce. Also the workforce can be stimulated by feeding artificial diets to adjust to pollination needs. [vM10].

Pollination is important for both wild plants as well as agricultural productivity, affecting global human food supply. Fifty-two of the 115 leading food commodities depend on honeybee pollination [KVC+07]. Putting this into figures results in an estimate of around 35% of the human diet made possible by animal pollination [KVC+07]. Imagining a scenario with all animal pollinator service declined estimates in around 1.42 million additional human deaths per year from malnutrition-related diseases, as recent studies show [SSMM15]. This is more than the total population of Cyprus[2]. These statistics need to be regarded with care, since honeybees are not the only animal pollinators. Nevertheless, they propagate the relevance of managed honeybee colonies for the global food supply.

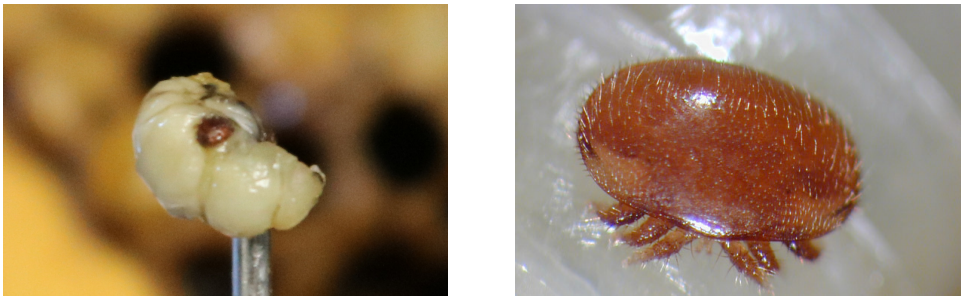
From an economic point of view insect pollination is estimated to have a global value of around 212 billion USD in the year 2009 [GSSV09]. This makes around 9.5% of the total value of agricultural production.

1.1.1 Colony Losses and Colony Collapse Disorder

In the years 2006 to 2008 a significant drop in the number of managed honeybee colonies is observed in the US [ESM+09]. This sudden vanishing of colonies is titled *Colony Collapse Disorder* [vHJUP10]. The keywords "phenomena" and "disorder" emphasize the fact that scientist and beekeepers are at the mercy of this anomaly. Within these years, research focused on finding reasons for this disorder, with the goal of identifying the root cause and possibly stopping the process. Despite the efforts no single root cause is identified and instead a combination of the following stress factors is found responsible [ESM+09], [PBK+10]:

- **Diseases and parasites**: There are more than 23 viral diseases known that specialized on honeybees, not including bacterial infections. These are transmitted and spread via parasites, like *Varroa destructor*.
- **Pesticides**: The increased use of pesticides in modern agriculture leads to a frequent contact of honeybees with these chemicals during foraging activity. With many of the used pesticides having unknown effects on honeybee populations.
- **Genetic mono cultures of honeybee colonies**: Genetic variety is getting sparse within the honeybee genome due to selective breeding. This also leads to higher susceptibility to pathogens and pests.
- **Increased industrialization**: Natural habitats of pollinators are shrinking and bacteria and fungi deadly to honeybees are spread easily because of increasing global transport.

[2]Population: 1,176,598, taken from `https://esa.un.org/unpd/wpp/`



(a) (b)

Figure 1.2: *Varroa destructor* mites. (a) A grown female Varroa mite on honeybee larvae after opening a brood cell. (b) A microscopic record of a female Varroa mite.

- **Weather and climate change**: Changing climate adds additional stress in warmer parts of the world.

This list is far from complete and does not show to complex interactions between different stress factors. It is undeniable that the parasite *Varroa destructor* is one of the driving factors in this field [LMB+10]. *Varroosis* is entitled the single largest threat for honeybee colonies world wide. [BG08], [LMB+10], [FNK13], [Loc16]

1.1.2 *Varroa destructor*

*Varroa destructor (*V. destructor*)* or the Varroa mite is a parasitic mite specialized on bees. Their original natural host is the Asian bee *Apis cerana*. In the age of globalization and international trade opportunities for displacement of parasites beyond their natural barriers is high [Hul09]. This lead to *V. destructor* adapting to new hosts like the European honeybee *Apis mellifera*. Because of this modification it successfully spread throughout the world, leaving only a few isolated islands and the continent Australia unaffected [RAZ10]. There exists a stable host-to-parasite relationship between the Asian hive bee *Apis cerana* and the parasitic mite. This symbiosis is missing with the European honeybee *Apis mellifera*. This is largely due to the fact that Varroa adapted their live cycle enabling them to reproduce not only in drone brood cells but worker brood cells as well [Loc16]. Also the Asian hive bee has developed behavioural defenses against *V. destructor*.

Grown female *Varroa destructor* are of round shape, approximately 1mm of diameter. They have six legs to move and jump onto their hosts and tubular feeding tools to suck the bodily fluids. Figure 1.2 shows examples of grown female *V. destructor*. Their life cycle is structured in three major phases [RAZ10], [LMB+10]: In the first phase, adult female mites invade brood cells of worker or drone bees just before they are closed with a wax capping. The reproduction and second phase is starting here. Inside this secure environment the female mite is feeding from the growing honeybee larvae while

reproducing and mating. After around three days the first male egg is laid followed by around 4 female eggs. The bee larvae is not killed during this procedure and later frees grown mites from the cell.

This releases 1 to 2 additional female mites, which spread to other hives or look for new brood cells directly. This is also the period, when the presented system can detect the presence of *V. destructor*, because they are transported through the entrance. Leaving a hive untreated causes an epidemic infection of surrounding hives, because of this transportation aspect.

The mite feeding on the bee results in a reduced life span and general weakening of the host. But this becomes a neglectable factor when compared the transition of viruses during the feeding. Up to 23 different viruses are transmitted by Varroa mites resulting in different symptoms like disoriented or crippled bees unable to fly because of deformed wings [MG15].

1.1.3 Monitoring and Treating Infestations

Untreated colonies exceeding an infestation rate of 30% during the summer months, are most likely to not survive the following winter [FHIR03]. Therefore regular monitoring and treatment is necessary to sustain honeybee colonies for more than one season [RAZ10]

In the year 2010, Lee et al. suggested standardized sampling plans to better control the global growth of parasites like Varroa destructor [LMB+10]. The procedures are based on statistical extrapolation from a minimum sample size of 300 bees per test and colony. The following three categories are identified:

- **Sampling from living bees**: A sample of 300 up to 1000 living hive bees is drawn and tested for the presence of Varroa mites. All samples are taken manually with either of the two known methods: *sugar shake [MWE02]* and *alcohol wash [DJDARG82]*. In both methods the sample is put in a jar and shaken, with the goal of separating the Varroa mites from their hosts. A fine scaled grid then separates the parasites from the rest and then they are counted manually. When using alcohol, the bees die during the procedure with the advantage of getting more accurate results compared to the other methods. Using powder sugar instead of alcohol produces less accurate results, but the sampled bees survive the procedure.
- **Sampling from the brood**: This method is time consuming, but is used to determine the infestation status in the first phase of the reproduction cycle. One or more apiary combs holding brood cells need to be removed from the hive and the cells are observed individually. They need to be opened, removing the covering wax capping, with the purpose of looking for occurrences of parasites. The bee larvae is killed during this procedure. In Figure 1.2a a brood with Varroa infestation is displayed. The more cells are observed, the higher the accuracy of the test. Advantage of this method is, that Varroa mites can be detected in the early stages of their life cycle, while they are still in their reproductive phase.

- **Non-invasive sticky board**: This method is non-invasive in the sense that no bees are being harmed during this procedure. To get an estimate of the Varroa population inside a hive, a board covered with Vaseline or other sticky material is put under the hive. In regular intervals the boards are recovered and the amount of dead mites sticking on the board are counted. The boards can be equipped with a printed grid to ease the task of manual counting.

All methods result in a count of parasites from a sample. The amount of parasites in relation to the total sample size gives the relative infestation level in percent. Here again the need for efficient methods of monitoring become apparent.

1.1.4 Comparison of Image Processing and Classification Methods

As part of this work, honeybees need to be segmented and extracted from consecutive video frames for processing. Five different methods qualify as potential solutions to the segmentation problem. This offers the possibility of comparing and evaluating these methods, which are best suited for the application in this veterinarian context.

In the next stage of the algorithmic pipeline, the question about manual and automatic feature extraction is applied to honeybee classification. Manual feature extraction describes the task of hand crafting features to be extracted from the input data to transform from the input space to the features space. This translation is necessary to reduce the dimensionality of the input data and enable feature based classification. This approach is also referred to as a "traditional" machine learning approach.

A different way of regarding the classification problem is to let the model learn the feature representation from the input data without manual intervention. This is the case with deep learning and convolutional neural networks. So the "traditional" machine learning approach consists of a manually engineered transfer function, followed by a "shallow" classifier. A "shallow" classifier denotes a model that is not using deep learning technologies [YYPH14]. Both methods qualify for the presented classification task and are also used for insect classification [GM16], [MCR+17]. A detailed comparison is provided by applying both methods to the same problem using the same datasets. This allows for making a decision about which method is better suited for the classification task in the video surveillance setup presented in this work.

1.2 State of the Art

The standard treatment against parasites involves frequent manual monitoring. This process is time consuming and possibly inaccurate depending on the individual procedure and sample size taken [LMB+10]. The major thread is imposed by a parasite called *Varroa destructor* creating the need for efficient and accurate parasite detection. In the best case, automating as much of this process as possible up to complete automation.

Ref	Year	Objective	Method	Dataset	Conclusion
[GVAG16]	2016	Creating a wireless ad hoc sensor network for bee monitoring	Ad hoc nodes are implemented using Raspberry Pis with temperature, humidity, gps and camera sensor	Not applicable	Power consumption on raspberry pi doubles with external sensors, but long term application is possible using all sensors
[ZKA+16]	2016	Smart apiary management: Monitoring the health of colonies	Combining sensors attached to the hives with a cloud based centered monitoring service	Not applicable	Internet communication technologies are necessary for precision agriculture minimizing manual intervention
[GCZ+15]	2015	Smart apiary management: Monitoring the health of colonies	Remotely monitoring: Temperature, humidity, sound, carbon dioxide and weight	Not applicable	A system for helping beekeepers improving their knowledge of the colonies by collecting environmental data is presented
[QAHL14]	2014	Using acoustic signals to detect Varroa infestations	SVM is trained with features extracted from acoustic signals from healthy and infected hives	5 sounds of healthy and 1 sounds of infected bees	Prototype for monitoring acoustic signals is presented, missing a detailed evaluation
[RH14]	2014	Measuring the flight activity	Human observers are manually counting bees at the apiary entrance	60 one-minute counts	Factors for influencing the flight activity like foraging availability are identified
[SMA+94]	1994	Measuring the flight activity	Rod arms triggering a mechanically driven pen	Not documented	Flight activity can be monitored mechanically
[Wri28]	1928	Measuring the flight activity	Human observers are manually counting bees at the apiary entrance	Not documented	The count of bees is used for deriving information about the honeybee colony
[Gat14]	1914	Manually measuring the hive temperature	Human observers are regularly measuring temperature of bee colonies	Not documented	The temperate gives insights into the status of a beehive

Table 1.1: State of the art for electronically enhanced apiary management.

To the best of my knowledge, no system providing this service exists today. The methods and ideas for the created detection pipelines are grouped in the following Sections. Each is explained in detail as well as summarized in the Tables 1.1, 1.2 and 1.3.

1.2.1 Electronically Enhanced Monitoring of Bees

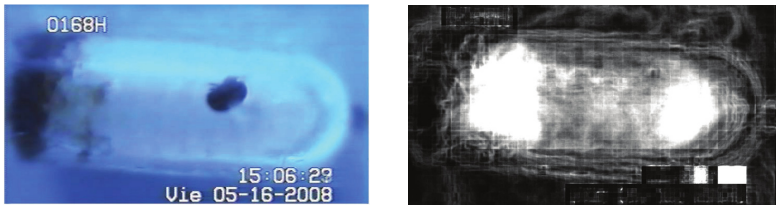
The idea of regularly collecting data about honeybees dates back to the year 1914, where Burton N. Gates recorded hourly temperature data from a beehive [Gat14]. A first

attempt for automatically recording the flight activity of honeybees is documented by Wrigth et al. in the year 1928 [Wri28]. Bees moving through the hive entrance are tripping a balance arm, which produced electrical impulses driving a printer. Their efforts show that even back then the relationship between measurable parameters and the activity or health of an apiary is suspected. Using micro processors to estimate flight activity is first presented in the year 1994 by Struye et al. [SMA+94]. They built a machine equipped with tunnels and infrared sensors for each tunnel to successfully count the number of bees flying in and out. In the year 2015 Meikle et al. state that multiple sensors are used to derive information about the health of bee colonies [MH15].

The trend of computer technologies entering the beekeeping profession is driven by the fact that electronic sensors are becoming increasingly cheaper and more accurate [MH15]. These topics are referred to as *precision beekeeping*, *precision apiculture* or *smart apiary management*. Main goal is to increase the efficiency of beekeeping and develop real time observation tools to continuously monitor bee colonies, without creating additional stress [ZBMS15], [ZKA+16]. The process is grouped into a three-phase-cycle: 1. *data-mining* using sensors, 2. *data interpretation* and 3. *application* based on the measured results. Examples for sensors which are used in the first phase are: Weight-, temperature-, oxygen-level-, humidity-, sound-, vibration- and optical sensors [MH15], [YLG11]. In the second phase, the collected information is statistically processed. This is then used to predict the colonies health state and apply treatment or other processes in the third phase [ZSM12].

There exist also commercial bee monitoring systems. The systems allow beekeepers to keep track of parameters like weight, humidity or sound inside the hive. Example systems are BeeHiveLab [GCZ+15], ARNIA [Eva18], HOBOS [Tau18], the Intelligent Beehive [HHDV16], or Melixa [Mel18]. All of these commercially available products facilitate all three phases of precision beekeeping in their products. The HOBOS project also incorporates visual information in the first phase. A surveillance camera is mounted vis-à-vis the entrance giving a visual clue about the activity of the hive. The data is not further processed but directly streamed to give a live camera feed of the hive.

Qandour et al. are focusing on the effects of Varroa infestations to remotely detect infestations of such [QAHL14]. They claim to detect pest infestations, including Varroa infestations, solely by remotely monitoring the sounds created by honeybee colonies. By using directed electronic microphones, sounds of healthy as well as infected beehives are recorded. Features like frequency and bandwidth of the sounds are then extracted and used to train a Support Vector Machine (SVM) classifier. The classifier is trained to discriminate between ”infected-” and ”healthy- hive” sounds. Unfortunately no detailed results or data sets are provided to compare their results. In their conclusion they state that the classification works better then random guessing and that future work is necessary. Nevertheless acoustic features can be used in combination with visual sensors, to detect parasite infestations.



(a) **(b)**

Figure 1.3: Tracking Varroa mites inside a brood cell. (a) A Video frame of a beehive brood cell infected with female Varroa mite. (b) The heat map derived from the tracking the mite inside the cell [RPT+12].

1.2.2 Monitoring using Visual Sensors

Meikle et al. state that technological progress, creates cheaper sensors [MH15]. This also applies to cameras and video sensors with advantages over other sensor technologies. They provide a rich amount of information and enable machine learning and image processing technologies. A camera facing the entrance of a beehive records all events occurring at the entrance. One can later set the actual issue to focus on. The same video can be used to count bees, track bees, perform behavior analysis, or to measure the pollen foraging activity [BPRM16]. Further, camera sensors allow non-invasive monitoring of honeybees [BPRM16]. This is why cameras are also used for behavior analysis and to understand the bee waggle dance [CMS08], [THKA16]. Finally, the collected video material acts as a video archive to be manually inspected by a beekeeper in the case of anomalous activity.

Camera sensors are also part of research related to Varroa destructor. In the year 2012, Ramirez et al. are detecting and tracking Varroa mites inside honeybees brood cells, with a follow up work in 2017 [RPT+12], [RBPRFM+17]. In a laboratory setup, a single mite infected brood cell is observed by a video camera, displayed in Figure 1.3a. Image processing is used to extract the Varroa mite from the background and track the mite inside the cell. They use a technique termed *adaptive background subtraction* to separate the mite from the background. This is part of a group of methods called *change detection*, which can be found in latter works dealing with bee detection and tracking. The idea is to extract the background from the image by averaging over previous frames and subtracting this background to extract the foreground. This is based on the assumption that the background is static and only changing slowly while the foreground is changing quickly. The information provided by the tracker is used to derive a heat map of the movements inside the brood cell, as shown in Figure 1.3b.

1.2.3 Monitoring the Hive Entrance

There exists a steady and balanced flow of bees entering and leaving a healthy hive which, is partly explained due to the fact that bees rarely die inside their hive. [KTP15]

Ref	Year	Objective	Camera Setup	Segmentation	Test dataset	Conclusion
[RMA+18]	2018	Detecting pollen bearing honeybees	Exposed camera setup protected by sun cover over hive and camera	Not mentioned	142 samples for the test dataset (80/20 split from total 710)	A two-layer CNN outperforms deeper pre-trained CNNs and "traditional" models with 96.4% accuracy
[BPRM16]	2016	Detecting pollen bearing honeybees	Box with artificial lighting and Raspberry Pi camera model	Background subtraction using MOG and color segmentation in LAB color space	354 images of honeybees extracted from 50 video frames	Using a SVM to classify pollen bearing bees with 92.14% accuracy running on a Raspberry Pi
[KR16]	2016	Counting the number of bees at Langsthroth beehive entrance	Exposed Raspberry Pi camera attached to a Raspberry Pi	Color thresholding in HSV	7,948 bees taken from 1781 frames	Counting works with an accuracy of 85%
[THKA16]	2016	Measuring the in-and-out activity at hive entrance	Box setup including LED lighting and processing unit	Background subtraction using average gray scale value	150 manually labeled frames from 30s of recordings	Predictive tracking and counting of honeybees shows high correlations with manual counts
[KTP15]	2015	Comparing image segmentation methods for bee detection	Exposed camera setup, without artificial lighting	Adaptive background subtraction and object detection using cascade classification and sliding window	Unknown	Background subtraction outperforms cascade classification for bee segmentation
[TG15]	2015	Estimating the number of bees at the entrance of a beehive	Exposed CCTV camera with time stamp relying on natural daylight	Adaptive background subtraction using 11 previous frames	Bee count manually evaluated on 20 frames over 12 hours	The bee estimation from the signal-to-noise ratio matches the manual count
[YC15]	2015	Tracking honeybees and splitting merged groups of honeybees	Exposed action camera without artificial lighting	Combination of background subtraction and color thresholding	300 video frames manually annotated	99% of flying bees and 70% of merged bees is tracked correctly
[CGKM13]	2013	Tracking bees at the hive entrance using stereo vision	Exposed stereo camera setup without artificial lighting	Combination of depth and motion intensity segmentation	80 manually annotated trajectories in 1000 frames	Tracking achieves a detection rate of 79.46%
[CYJL12]	2012	Identifying and recognizing honeybees using paper tags attached to bees	Box with artificial lighting and infrared sensitive CCTV camera	Hough transform for detecting the paper tag followed by OCR	30 min recording of in and out activity in laboratory	Bee detection and recognition rate of 86%
[CMS08]	2008	Measuring the in-and-out activity at hive entrance	Exposed camera setup, without artificial lighting	Adaptive background subtraction with template matching for detection	600 video frames manually annotated in the field	Tracking works in real-world condition with precision 0.94 and recall 0.79

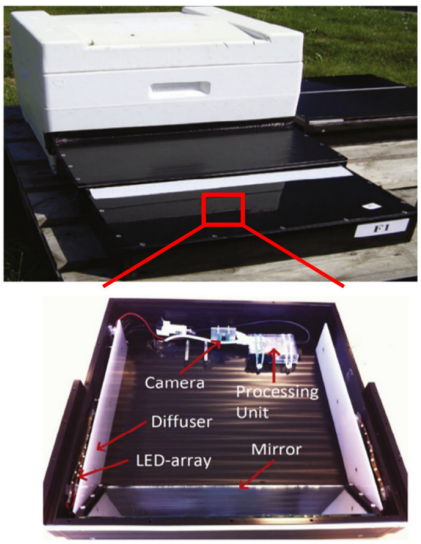
Table 1.2: State of the art for monitoring the entrance to a beehive using visual sensors. All camera setups use a similar top-down view on the landing platform of the beehive.

This flow allows for monitoring regular and irregular behavior which might lead to conclusions about the general health state [KTP15]. The behavior can be recorded by creating entrance-traffic statistics of bees entering and leaving the hive. This traffic is used to estimate the flight activity which can be used to determine colony intrinsic factors like foraging activity or general health, as well as assessing a colonies response to environmental changes [RH14]. This is conducted after manually monitoring a beehive by human observers.

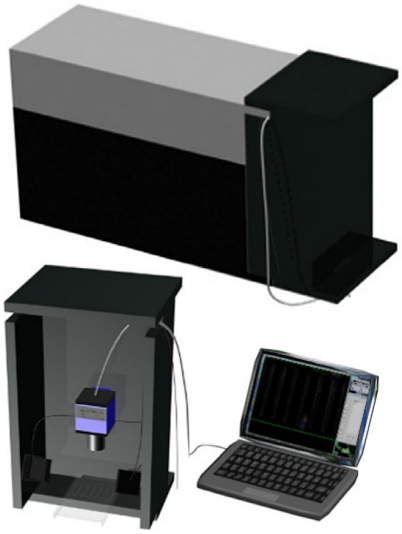
A possible setup for video based monitoring is to replace these observers by cameras mounted outside the hive facing the entrance. This setup has the following favorable properties: (1) Natural daylight can provide the required light for creating photo or video footage, or artificial lighting can be added easily. (2) The elevated humidity and temperature inside a beehive must be considered when deciding where to put a camera sensor. Placing the sensor outside the hive can avoid fogging of lenses and over-heating of electrical compounds. (3) The entrance to the hive offers a central point observation, providing an organized scenery when compared to sceneries found inside the hive. Nevertheless attempts for visually tracking and analyzing bees inside a beehive are presented by [WWRL17], [KOOI11], [KM09]. Main goal is to record and understand interactions and forms of communication between bees, like the waggle dance [RGS+05]. To deal with the chaotic scenery both [WWRL17] and [KM09] use sticky tags attached to the bees bodies for identification and tracking. In a laboratory environment Wario et al. achieve a detection accuracy of over 90% on the manually marked bees [WWRL17]. Kimura et al. are achieving a 72% detection rate of total of 500 bees in a dynamic honey comb scenery, without the need for manually tagging individual bees [KOOI11]. They use a data compression method similar to k-means named *vector quantization (VQ)*, to find centroids of individual bees and groups of bees respectively. Tracking of individual centroids is done by calculating the trajectory of previous frames and using a nearest neighbor approach to assign centroids to previous frames. The system is providing a proof of concept, but does not deal with continuous application issues. The maximum processing time is limited to 3 minutes due to memory limitations. With a maximum resolution of 720×480px and the in-hive setup, no parasites are detectable.

Campbell et al. are the first to publish a video monitoring system mounted at the hive entrance using the preferable setup with a camera mounted outside the hive, facing down on the entrance [CMS08]. Their goal is to measure the flight activity of a honeybee colony by tracking individual bees on a frame-by-frame basis. Figure 1.4a shows the camera setup, where the camera is mounted outside of the hive facing down on the landing platform. In this setup the camera is exposed to environmental factors and relies on natural day light as a light source. The exposed setup introduces additional challenges like constantly changing light conditions or light obstructions by falling leaves or other animals. The authors still argue in favor of this setup reasoning to be as least invasive to the bee colony as possible, preserving there natural behavior. For detection of individual bees in the scenery they use *adaptive background subtraction*. This technique is used for defining a mask separating foreground from background pixels. In this case the

(a) [CMS08]



(b) [THKA16]



(c) [CYJL12]

Figure 1.4: Apiary entrance monitoring. (a) Exposed camera setup with camera housing mounted on top of the entrance. (b) Closed camera setup with constant lighting conditions. All electrical parts are included. (c) Box setup similar to (b) but using external hardware and a infrared CCTV camera.

word *adaptive* refers to averaging the recent 300 video frames to predict the background. This compensates for slow changes in the background like changing day light conditions. Tracking is done by assigning each detected bee a probability of different behaviors like

"crawling" or "flying", to better predict their position in the next frames. In their results they show a high false-positive-rate due to shadows cast by approaching bees. Still they claim a detection rate of 94% for their bee counter measured on 600 manually labeled frames. This is followed by a work of Tashakkori et al. in the year 2015 [TG15].

Tu et al. present an efficient way for estimating the number of bees in the image, based on the amount of foreground-pixels in relation to the total amount of pixels [THKA16]. This relation is fairly constant, because the camera is mounted in a box with artificial lighting, only allowing bees to crawl into the hive. The size of individual bees only differ slightly, which allows estimating the amount of bees in each video frame. This count is then passed as parameter k for k-Nearest-Neighbors algorithm, to find the center points of the individual bees in the video-frame, which are tracked over time. For tracking each individual center point is modeled as the centroid of an ellipse with its main axis modeled using the moments of the foreground object. Using this ellipse approximation can be used to predict movement direction and improve the tracking results. The ellipse fitting step is used in this work as well, to improve the bounding box orientation extraction for the extraction of the ground truth.

The hardware setup used by Tu et al. is depicted in Figure 1.4b. This setup, in comparison to an exposed camera setup visible in Figure 1.4a, offers a constant background and therefore eases the task of identifying and tracking the objects of interest. Also the authors manage to get real time analysis using low cost hardware and present an all-in-one solution, with no external hardware necessary. In principle a similar goal is targeted in this work, although computational efforts are higher when processing frames for classifying individual bees.

The idea of using a cascade classifier, facilitating a windowing function for bee detection, is introduced by Kale et al. [KTP15]. They focus on creating segmentations of honeybees for later tracking, comparing two different methods for segmentation: *Background Subtraction* and *Cascade Classification*. Cascade classification is originally introduced by Viola et al. [VJ01] in the year 2001 and is used for face detection in real time videos. The technique is targeted at real time computation of features on low cost hardware. A *sliding-window* is used to scan the input image. For each window, easy to compute features are calculated, in a cascade manner, starting with simple to complex features. The complex features are only computed, once the simpler features show the possibility of detecting the object of interest. Only if all stages provided by the weak-learners are met, the object of interest is classified as detected. Another advantage of this method is, that not only the foreground is estimated but also the positioning of a bounding box for the detected object is provided by the sliding window. Despite these favorable properties experimental results show that background subtraction generally outperforms cascade classification in terms of segmentation. The sliding window technique is used in this work to extract sub-window-patches from bee images, before classification.

Using a closed environment, such as a box, fitted with artificial lighting to monitor the entrance is first presented by Chen et al. in the year 2012 [CYJL12]. Figure 1.4c shows

a schematic representation of the recording system. This work and the work by Tu et al. inspired the first prototype design.

Their goal is to create accurate statistics about the bee-traffic and flight time of individual bees. To distinguish between the individuals, they fitted each bee with a specially designed paper tag holding a unique code. The tags are detected by the camera and the code is extracted and identified using optical character recognition. For recognizing the tags a close up view of the bees backs is necessary. To get enough detail, the entrance is narrowed to 80mm of width and the bees have to pass a transparent tunnel of 6mm height. This tunnel is further divided into sub-tunnels of 8mm width, to separate the stream of bees and pre-set the possible walking directions. This inspired the tunnel design of the second prototype. They record detection rates up to 98% using this setup. Despite these good results, their work is contradictory. The authors refrain from using visible light and instead use infrared light, arguing to create less interference for the bees. On the other hand they physically manipulate bees by attaching tags on their backs and introducing tunnels which can get jammed easily. Experiments performed using the second prototype showed, that using the tunnels with the dimensions presented in this paper, dead bees can not bee carried out of the beehive. So the natural drop of bees is disrupted leading to jammed tunnels, if not cleared manually. Also, due the tunnel height of 8mm, bees are walking upside down through the tunnel hiding the tags with their bodies.

1.2.4 Visual Insect Classification

Detecting parasites on honeybees can be handled as a classification task rather than a detection task. Therefore a classifier is trained to discriminate between healthy bees and infected bees. Respectively, it is trained to distinguish between image patches showing Varroa mites and image patches not showing Varroa mites.

The first approach for automated bee classification is by Arbuckle et al. in the year 2001 [ASSW01]. This is also one of the very first attempts to automate insect classification in general [MCR+17]. The authors introduce a system named ABIS, short for: Automatic Bee Identification System, which consists of a suite of software tools for identifying and monitoring bees. Different bee species are determined using microscopic images of the bees wings. The venations and cell structures within bees wings are unique to their species and therefore offer the information needed to classify individuals. By only utilizing 20 samples per class for training a classifier, recognition rate of over 95% is achieved not mentioning the size of the test dataset. This work and the methods used for general insect classification or classifying butterfly species, are providing insights into possibly useful features for parasite detection.

Martineau et al. give an overview of existing image-based insect classification techniques [MCR+17]. They survey 44 papers and group them based on three sequential phases: (1) Image capture, (2) Feature extraction and (3) Classification. The first phase differentiates between Lab-based and Field-based input data. Since the camera

Ref	Year	Objective	Model	Classifier	Test dataset	Conclusion
[MCR+17]	2017	Survey of the state of the art for insect classification	Comparing features: SIFT, color and shape as well as hierarchical methods	SVM, Artificial Neural Networks, Decision Trees, Non-parametric models	33 different datasets are identified and compared	Most used features are shape, color and texture in combination with SVM or neural networks
[GM16]	2016	General insect classification with 277 different species	Convolutional neural network based on VGG-16	ConvNet	217,657 insect images organized in 277 classes	Top-5-miss-classification rate of 22.54% using hierarchical deep CNN
[KK14]	2014	Butterfly species classification with 14 different species	Five texture and three color features with "shallow" neural network classification	2-Layer ANN	95 images of butterflys	94.74% classification accuracy showing the value of texture and color features for butterfly classification
[ASSW01]	2001	Developing ABIS (Automatic Bee Identification System) using microscopic images of bee wings	Deformable template matching based on cells of wings of bees	SVM	Unknown	95% accuracy not mentioning the size of the test dataset nor the amount of classes.

Table 1.3: State of the art for visual insect classification.

setup followed in this work, provides a constant environment similar to the laboratory environments, the Lab-based methods offer similarities. Meaning that the lighting and background conditions can be manipulated and reconstructed over different recordings. The second phase reviews feature extraction techniques used for insect classification. One fourth of the 44 works compared show, that segmentation of background and foreground is done manually. In the remaining 33 works, thresholding and background subtraction techniques are used.

Feature extraction is further categorized in global and local features. The main global features used are color and shape features, with Histograms and Local Binary Patterns (LBPs) as examples. The first attempt of using LBPs as a feature does not produce satisfying results, which is why this method is discarded. Local features are also used, with Scale-invariant Feature Transformation (SIFT) and Speeded Up Robust Features (SURF) prevailing.

Three quarters of the analyzed works use discriminative classifiers, such as Random Forests (RFs), Support Vector Machines (SVMs), or Neural Networks (NNs), instead of generative methods like Naive Bayes. Nevertheless, the task of adding additional classes to an existing classification system requires more effort with descriptive methods than compared to generative methods. This makes descriptive methods less favorable for general insect classification tasks. Considering a static class structure this is not true and descriptive methods are to be preferred. Nevertheless both a discriminative and a

generative classifier are used in the evaluation.

Finally they mention that Convolutional Neural Networks (CNNs) and other Deep Learning structures are the state-of-art for insect classification tasks. But only one of the 44 papers considered, written by Glick et al., uses CNNs as a classifier [GM16]. They are applying Convolutional Neural Networks to differentiate between 277 distinct classes of insects. The complete dataset used, consists of around 200,000 images, which are split in 90% training, 5% validation and 5% testing. The network architecture follows the well known VGG-16 introduced by Simonyan et al. in the year 2014 [KA14]. It is designed as a web service, allowing users to upload images of insects taken via a phone camera to a cloud service and in return get the highest prediction scores per class. This creates a dynamic use case and the authors present a satisfying best top-5-accuracy of 23.03%.

1.2.5 State of the Art Summary

The following inspirations are drawn from the presented works:

- The ideas for the hardware setup of the prototypes are taken from Tu et al. [THKA16]. They manage to implement a real-time tracking algorithm on low cost hardware.
- The tunnel dimensions and camera position as well as the box setup of the prototypes is taken from Chen et al. [CPMR05].
- Ideas combining different foreground detection techniques in combination with morphological image operations is inspired by Yang et al. [YC15].
- Also the idea for approximating the bees shape with an ellipse is taken from Tu et al. [THKA16] In the context of this work it serves the purpose of aligning the extraction bounding boxes with the main axes of the bee.
- Martineau et al. [MCR+17] as well as Arbuckle et al. [ASSW01] provided ideas for possible features when classifying bee images.

Eight of the 22 related works deal with tracking and behavior analysis which is not the main focus of this work. No comparative work dealing with parasite monitoring using visual information are identified. This is why the novelty monitoring system is developed in a evolutionary process.

1.3 Structure of Work

This work is organized in five main chapters: Chapter 1 starts with an introduction to the significance of the problem in Section 1.1. Followed by the state-of-art, for honeybee monitoring and analysis in Section 1.2.

Chapter 2, covers all necessary steps for creating the ground truth dataset. First, the characteristics of the final dataset are presented. Next the prototypes are presented in

Section 2.1, followed by the necessary data processing and extraction steps in Section 2.2 to form the labeled ground truth. The chapter is finalized with post processing steps applied to the dataset in Section 2.3.

In Chapter 3 the machine learning models used for classification are presented and explained. Section 3.1 gives an overview of the classification problem followed by detailed explanations for both the "traditional" machine learning approach (Section 3.2), as well as the deep learning approach (3.3). This chapter sets the theoretical basis for the evaluations presented in the following Chapter 4.

In Chapter 4 the datasets are explained in Section 4.1, followed by a definition of the used performance metrics in Section 4.2. The best configuration of each aforementioned approach is determined in Sections 4.4 and 4.3. Then a final comparison of the best performing models is presented in Section 4.5 followed by a conclusion in Chapter 5.

CHAPTER 2

Data Acquisition

All data is acquired for the purpose of performing automated decisions. In case of this work, the decisions are, whether a bee walking through the entrance of a beehive is infected with a parasite or not. To perform these decisions a pipeline that expects input data from the sensor and outputs a decision is needed. This pipeline processes the data and passes it to a decision model, which decides about the infection status of a honeybee.

2.0.1 Dataset Background

The decision model is designed using *supervised learning* [LBH15]. This is the process of finding a function f^* that describes a predictive relationship between input and output to the function. Goal is to predict the values of unseen inputs based on the already seen inputs. This pool of known inputs is composed of data specific to solving the problem at hand. The idea of supervised learning is to create a database of correctly classified samples and use subsets of these samples to *train* the framework and another subset to *evaluate* the performance. This database is referred to as *ground truth*.

The review of exiting insect ground truth datasets revealed sparse availability. The only available dataset related to honeybees is the Honeybee Dance Dataset [ORBD08], which features video material and tracking data of honeybees inside the beehive. No labeled dataset for honeybee classification or parasite detection like MNIST for digits recognition [KB04] or the Stanford Dogs dataset for classifying dog breeds [KJYL11], exists. This implies that a new ground truth dataset needs to be created from scratch. For this purpose the designed recording system is used. The advantage of not having an existing dataset is the ability to fully design and create the ground truth to fit the needs of this project.

In this work, the original input for creating a ground truth is based on video streams coming from a video sensor. This stream is composed of a series of images put in a temporal context. An individual image is referred to as video-frame or simply frame.

Labeled Dataset Total	
# Total Samples:	13,464
# Negative Samples (class 0):	10,372
# Positive Samples (class 1):	3,092
Ratio:	1:3.35

Table 2.1: Total amount of manually labeled data, before splitting into subset.

Dataset	# of samples	% of total	Ratio
Train dataset:	8,028	59,6%	Class 0: 6,219, Class 1: 1,809, Ratio = 1:3.4
Test dataset:	3,433	25,5%	Class 0: 2,680, Class 1: 753, Ratio = 1:3.5
Validation dataset:	2,003	14,9%	Class 0: 1,473, Class 1: 530, Ratio = 1:2.8

Table 2.2: Ground truth datasets after manually splitting into three subsets.

Each frame shows an entrance tunnel to a beehive and covers a field-of-view of around 10×4cm. In this observed area more than 200 bees can be captured in a single frame, passing through the entrance to a beehive. The automated decisions about a parasite infestation need to be performed for each individual bee and in each observed video frame. Detecting parasites on honeybees in each video-frame can be reformulated to: Detecting honeybees in each video-frame and classifying these, as infected or not. This way a complex problem gets separated into two simpler ones: (1) Detection of honeybees in the video frames and (2) Classification of bees.

For creating the necessary ground truth first the detected bees extracted from the video frames. Then the individual extracted images of bees can further be examined for the presence of Varroa mites, assigning them to one of the two classes. This way of regarding the problem is beneficial for this labeling task. A crowded scenery with up to 300 bees in one video frame, create a challenging labeling task for human observers. This can be eased by first extracting the detected bees inside the video frames, before passing the data to the observer.

(a) Negative data samples without parasites.

(b) Positive data samples with at least one visible parasite per image.

Figure 2.1: Data samples of the manually labeled ground truth.

2.0.2 Dataset Properties

Three distinct datasets are necessary: First a *train dataset*: Used for training a model. Second a *test dataset*: To evaluate the final performance. The test dataset is solely used for this purpose and remains untouched until the final evaluation step. Performance evaluation can only be considered unbiased, if the algorithm is tested with data it has

never seen before. Therefore a third dataset, the *validation dataset* is created. It is derived from the training dataset, e.g. in 80/20 split, so 20% of the training dataset is taken to form the validation dataset.

In total 13,464 bee images are manually labeled, looking for occurrences of Varroa mites. Examples of both classes are displayed in Figure 2.1. The dataset distribution is given in Table 2.1.

The labeled data is split into the aforementioned three distinct datasets illustrated in Table 2.2. This is done before training, to allow the same datasets to be used for the training of different models. For performing the dataset splits, the goal is to keep the ratio of positive to negative samples evenly spread, while taking into account that a split can only be made between images of different videos. Otherwise the same bee would be visible in different dataset introducing a bias to the data. After the manual split, the datasets are stored in different sub-directories to avoid a mix-up of data samples.

2.1 Data Recording and Prototyping

First step is data recording. Here the goal is to create a system capable of capturing honeybee traffic at the hive entrance with enough detail to be able to detect parasites like *V. destructor*. The basic idea for this system is provided by the works of [THKA16] and [CYJL12]. Their setups include a closed box environment for the camera, which is put in front of the apiary entrance filming the bees from a top down view when they are entering or leaving.

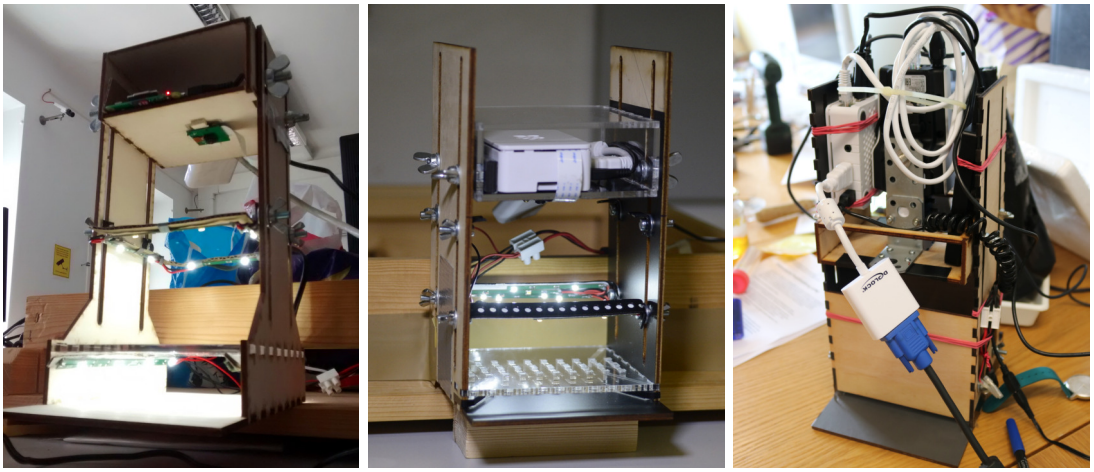
The purpose of the box setup is to create a constant environment with homogeneous lighting conditions, which is not provided in exposed setups similar to [CMS08] or [KTP15]. This way, effects of environmental factors like rain or extensive sunshine are reduced and a static background is provided.

Figure 2.2 shows the three major prototypes used for data recording. Each image shows the interior parts with the rain cover removed.

2.1.1 Design Considerations

The following considerations where taken into account when designing the proposed camera systems [SSMB16]:

- **Minimal interference:** This system should not interfere with the efficiency of the honeybees in their daily tasks. Using a box setup in front of the beehive possibly interferes with the harvesting activity [CMS08]. But neither [THKA16] and [CYJL12] are mentioning any negative effects after applying their systems. During the field testing phases, quick adaption to the new hive entrance is noticeable, with no visible decrease in efficiency.



(a) Prototype I **(b)** Prototype II **(c)** Prototype III

Figure 2.2: Prototypes in chronological order. Rain covers have been removed to show internal parts.

- **Easy implementation:** The overall dimensions of the system should allow easy mounting on beehives, as well as being compatible with different types of beehives.
- **Flexibility during prototyping phase:** Designing and building the system is an experimental process. A flexible design with modular mounting points is required to allow different hardware configurations such as: Different cameras, lighting, backgrounds etc.
- **Easily workable materials:** Materials like wood or plastic are to be preferred over metal to keep the costs low and make prototyping easier.
- **Weatherproof:** The system should be able to cope with different weather conditions, such as extensive sunshine, rain or wind. For this reason all electronic components must be kept save from water and temperature management might be necessary. This is accomplished by covering top part up to the tunnel entrance with a plastic rain cover.
- **Support independence/ autonomy:** Low power consumption hardware is to be preferred as well as wireless networking techniques, to enable the system to work even in remote places. Also all necessary hardware from storage unit to artificial light sources are to be included in the box setup.
- **Simplicity:** Utilizing the principle of simplicity, the proof of concept is provided using only one camera sensor. Additional sensors can be added to enhance the final detection performance, but are not dealt within this work.

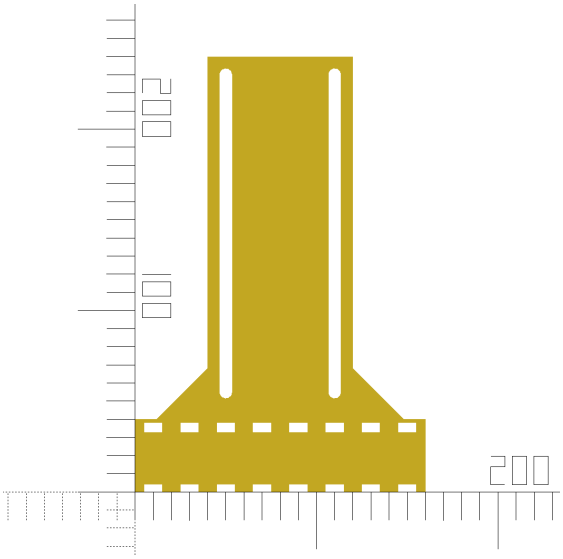


Figure 2.3: The OpenSCAD rendering of a wooden side panel of the Prototype I illustrating the bumps and dents design.

Figure 2.4: Calculating the necessary camera lens properties for a targeted field of view of 100×40mm and a camera to image plane distance of 200mm. The same calculations can be used for different fields of view.

To give the system stability and ease the assembly, all junction points of the individual parts of the prototype are equipped with a regular pattern of bumps and dents. These fit seamlessly into each other, making screws unnecessary for assembling the main parts. The design of a wooden side panel of the first prototype is displayed in Figure 2.3, showing the regular patterns of dents and holes. This design technique is common to all created prototypes, because of the satisfying results. All prototypes are designed using OpenSCAD[1], a 3D solid geometry modeling tool. It allows the design of complex objects by combining simple objects like cubes, cylinders or spheres.

When deciding about the overall dimensions of the system two primary factors are taken into account: (1) The overall dimensions of the system are chosen to allow easy mounting on industrial standard beehives. Also it allows mounting of more than one camera system on one hive. (2) The outer dimensions follow the requirements created by the sensor lens properties. Goal is to cover a minimum field of view of around 100×40mm by the camera lens. With this field of view and a maximum distance of 200mm from the image plane, the resulting viewing angles need to be at least 29° horizontally and 10.2° vertically. The necessary calculations are depicted in Figure 2.4. The final distance to the image plane depends on the camera lens used, which is why all prototypes are equipped with flexible

[1] `http://www.openscad.org/` (last accessed 04/2018)

mounting points for the camera. Using the same calculations depicted in Figure 2.4, an image plane distance of 150mm would result in a minimum viewing angle of horizontally 37° and vertically 15°. Further reducing the height of the prototype, while keeping the same field of view, requires a lens with wider viewing angles.

2.1.2 Lab Recordings and Field Recordings

Since the system is mounted at the entrance to a beehive a certain level of flight activity is necessary for creating video data. This limits the period of time available for recording data roughly to March to September (CET) where the average temperature is above 15 degrees Celsius. Further this becomes prominent when considering the fact that not only bees are to be monitored but also Varroa mites. This limits the final time span to a rough period of May to September, based on the observations made. Two different recording setups are used, which both depend on the availability of honeybees and Varroa mites to produce data.

1. **Field recording**: The camera system is mounted on beehives in an open-air environment, continuously recording video data during daytime. Long term application of the system is tested and test data is recorded. Using this setup, more than 820 hours of video material are conducted. The longest continuous field recording spans 52 consecutive days and produces 1.2 TB of video data.

2. **Lab recording**: Parallel to recording in the field, recordings are performed in a controlled laboratory environment. Labeling the field data is a difficult task, due to the amount of bees passing through the entrance at the same time. This leads to the idea of using a controlled laboratory setup to create the video data for the ground truth. The standard procedure includes: (1) Extracting parasites and honeybees from a beehive. (2) Artificially infecting bees with the parasites so each bee has at least one parasite sitting on its body. (3) Lock the infected bees inside the prototype by closing the tunnel entrances and perform continuous recordings. Figure 2.6a shows the laboratory setup with the first prototype. In Figure 2.6b the data recording process is shown, with bees locked inside the prototype.

   The task of manual labeling benefits from this setup. By introducing the prior of at least one Varroa mite per bee visible in the scenery less effort has to be put in looking for occurrences of parasites. Also since the amount of bees locked inside at once can be controlled. Making the scenery less chaotic when compared to videos from field recordings. Using this method more than 420 hours of video material are recorded.

2.1.3 The Prototyping Process

The process for creating a prototype is illustrated by a flowchart in Figure 2.5. It starts with the *Design* phase on the left side of Figure 2.5. Here new design ideas are implemented and overall dimensions are varied and tested in virtual form. Hence

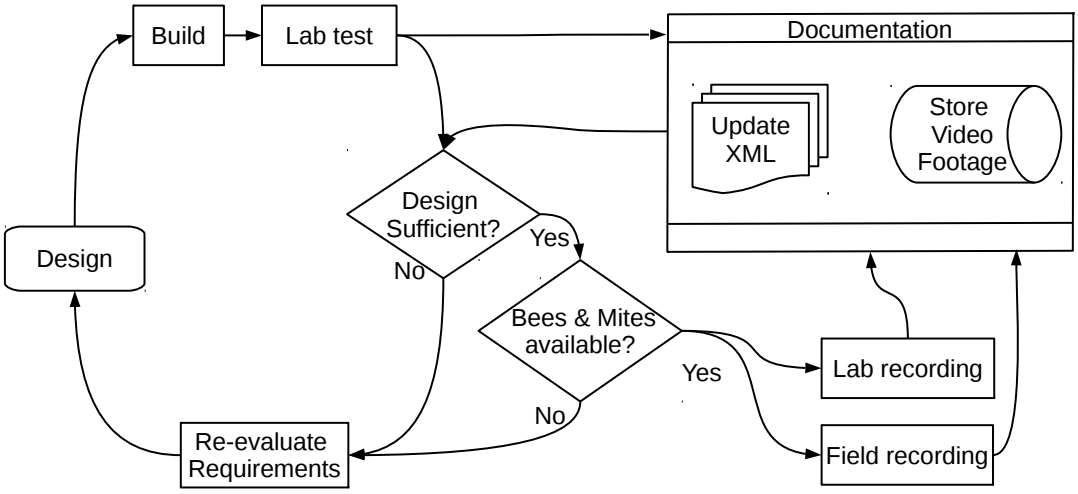


Figure 2.5: The data recording process. It shows a circular information flow starting in the *Design* phase leading to *Documentation* and back to *Re-evaluate Requirement* and *Design*.

the aforementioned program OpenSCAD is used. A resulting 3D model of the second prototype is presented in Figure 2.7a.

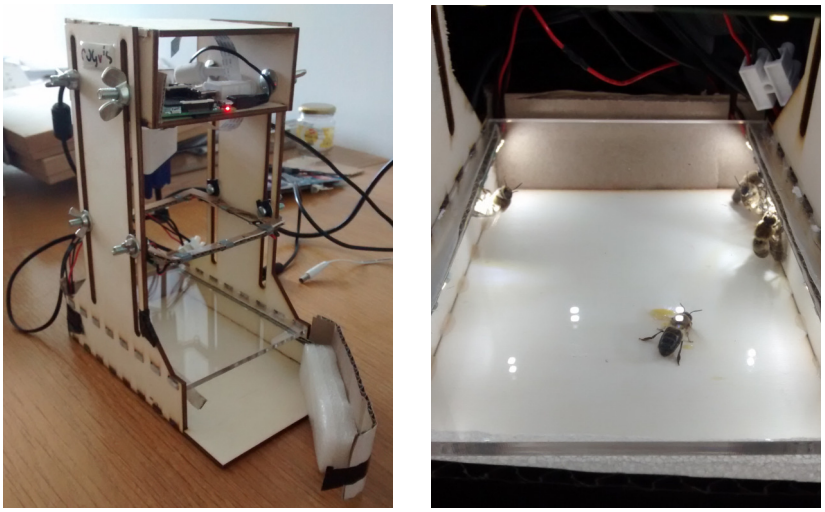
After the *Design* phase comes the *Build* phase. It incorporates all tasks related to physically creating the new prototype. The 3D model from the previous step is mapped onto a 2D plane and cut out of wood or plastic by using computer driven cutting machines like CNC or a laser cut. Such a machine is capable of creating precise cuts (< 0.2mm) through the materials wood and acrylic plastic, which is necessary to enable precise builds. The individual parts are assembled and the camera hardware is integrated to form a new prototype.

The next step is the *Lab test*, here the prototype is tested with different light and camera configurations. Varying these parameters is the main focus of this phase. The observations are documented and test video data is stored. If the design does not work as expected or produces insufficient results, it is re-evaluated and goes back to the *Design* phase.

If it works according to expectations the next steps are *Lab recording* and *Field recording*. Goal of these is to generate video data with the new prototype. This is only possible if honeybees and Varroa mites are available for recording, which is not the case during winter. The data is recorded in test sessions and is documented in a machine readable XML format, consisting of meta-information about the test and its configuration. This allows for the automated generation of data reports. Both the documentation as well as the recorded video data are archived on two mirrored 8TB hard drives in RAID 1. After each recording session, the design is re-evaluated and if still sufficient then further videos are recorded.

	Prototype I	**Prototype II**	**Prot III_Waveshare**	**Prot III_Axis**	**Prot III_Toshiba**
Processing unit	Raspberry Pi 2 Model B				Dell Workstation
Processor	Arm7 Quad Core Processor 900MHz				Intel i7
RAM	1 GB				8 GB
Camera	Raspberry Pi Camera Modul		Waveshare - RPi Camera (F)	AXIS M1125 Network Camera	Toshiba BU238MC
Sensor	1/4“ CMOS (Sony IMX219)		1/4” CMOS (OV5647 sensor)	1/3“ RGB CMOS	1/1.2” CMOS (Toshiba IMX174)
Pixel size	1.12 x 1.12 μm		1.4 x 1.4 μm	2.7 x 2.7 μm	5.86 x 5.86 μm
Resolution	1920 × 1080px				
Fps	30				165
Infrared filter	yes		no	yes	no
Sensititvity	680 mV/lux-sec			0.25 lux @ F1.4	7 lx @ F1.4
Min shutter	10ms			0.015ms	0.03ms
Shutter type	Rolling shutter			Global shutter	
Lens	Fixed focus: 3.04 mm, f2.0		Variofocal lens: 3.6mm	Variofocal lens: 3-10.55mm, f1.4	12mm Pentax TV lens 1:1.4
Sensor connection	MIPI Camera Serial Interface (CSI-2)			RJ45 Network Connector	USB (USB3 Vision Protocol)
Sensor size	25 x 23.86 x 9mm		25 x 23.86 x 9mm	148 x 70 x 44mm	29 x 29 x 42mm
Data interaction	Open Source Python library available			ffmpeg and Axis webinterface	C# with Toshiba drivers
Light setup	2700k LEDs 12V 16 LEDs (279lm/m)		Natural White (2700k) LEDs 24V: 60 LEDs (1249lm/m)		
Tunnel dims (len, wid, hei)	163 × 112 × 32mm	140 × 128 × 8mm	120 × 142 × 8mm		

Table 2.3: Technical properties of all three prototypes compared. Prototype III is represented with different configurations.



(a) **(b)**

Figure 2.6: Prototype I. (a) Lab recording setup for recording infected bees. (b) Lab recordings in action, with bees locked inside the prototype.

2.1.4 Prototype I

The first fully implemented prototype is displayed in Figure 2.2a and 2.6. The process of creating this prototype is published at the Visual observation and analysis of Vertebrate And Insect Behavior (VAIB) Workshop in the year 2016 [SSMB16]. The process starts with gathering the required information from experts and implementing the following ideas:

Tunnel entrance setup: All prototypes use a tunnel as central point for monitoring. The dimensions of the tunnel, functioning as new entrance for a beehive, are chosen to be as big as possible while allowing enough details to record Varroa mites. These dimensions are 163 × 112 × 32mm (length x width x height) for the first working prototype. The comparison to later models can be drawn by looking at Table 2.3, which compares all technical parameters between the different models. The original entrance with an opening of 370 × 40mm is narrowed to 112 × 32mm.

This change to the beehive raises the question of affecting the daily lives of a bee colony. While this concern cannot fully be eliminated it is mitigated by the fact, that beekeepers are narrowing the entrance to a beehive regularly during winter months. They considerably narrow the entrance, to protect the bees from predators entering the hive. Also apart from industry standard beehives, there exist alternative build types, like the Warre beehive [War07]. These are more economical and bee-friendly hive types, where new boxes are put under existing frame-less boxes to allow bees to

naturally grow their honey combs. The build plans[2] arrange the entrance with 120mm width and 15mm height, which comes close to the upper mentioned dimensions.

LED Lighting inside tunnel: The challenge of light reflections induced by the artificial lighting is dealt with, by placing the light sources inside the tunnel. Test recordings with dead bees showed little shadow casting and no reflections.

Low-cost hardware: The total price including all building materials for this prototype resulted in less than 150€. A Raspberry Pi model B micro computer is used as the main processing unit. It offers build flexibility and additional hardware with little costs of around 30€. The camera sensor used is provided by Raspberry Pi's official camera module which offers the desired resolution of 1920×1080px, at a minimum cost of around 30€. Both elements take up little space and therefore offer the possibility of fully integrating them inside the prototype. This is true for the camera module with outer dimensions of $30 \times 30 \times 5$mm (length, width, height).

The design already incorporates many of the above mentioned features like the transparent roof and the flexible camera mounting but suffers from poor design choices. Figure 2.6b shows the first test session with actual living bees in a laboratory setup. Up to six bees are taken from an apiary and locked inside the tunnel of the prototype to enable continuous recordings. The observed design flaws are: (1) The tunnel features the same height as the original apiary entrance. This is done to offer as much original entrance space as possible, but is unfavorable because it allows bees to fly through the tunnel. (2) The main light sources are placed inside the tunnel to reduce reflections on the acrylic glass. This is indeed lowering the reflections, but creates inhomogeneous light conditions when bees are passing by the light sources. In Figure 2.6b one can see the light occlusions produced by honeybees gathering at the light sources.

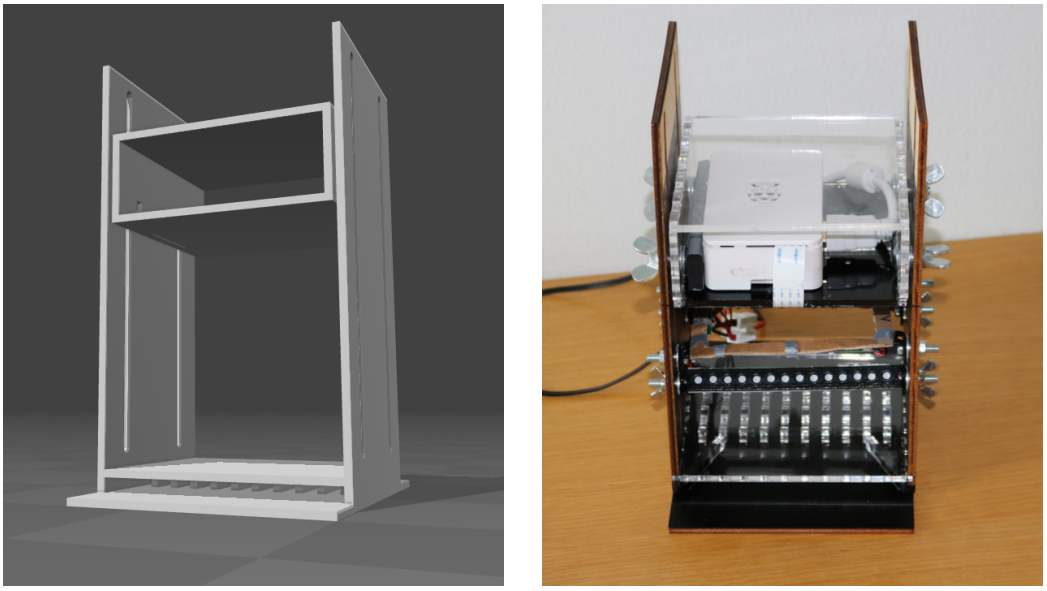
Figures 2.8a to 2.8c show examples of frames from videos taken with this prototype. Additional to the main light sources inside the tunnel a ring of LEDs is mounted from a top down view. This creates undesired reflections on the glass roof, but is necessary to raise the overall brightness of the videos. Despite the additional lighting the overall scenery looks dark compared to video frames from later prototypes such as Figure 2.8i. The faults in the initial design soon became apparent, leading to only little data being recorded using this prototype.

2.1.5 Prototype II

Main purpose of this prototype is to remove the design weaknesses of the first prototype and introduce sub-tunnels (Figures 2.2b, 2.7b). The implemented design features are summarized in the following description:

Subtunnel design: The overall tunnel height is lowered from 32mm to 8mm, to tackle the problem of bees flying through the tunnel. This way, bees can only crawl through

[2] `http://warre.biobees.com/plans.htm` (last accessed 04.2018)



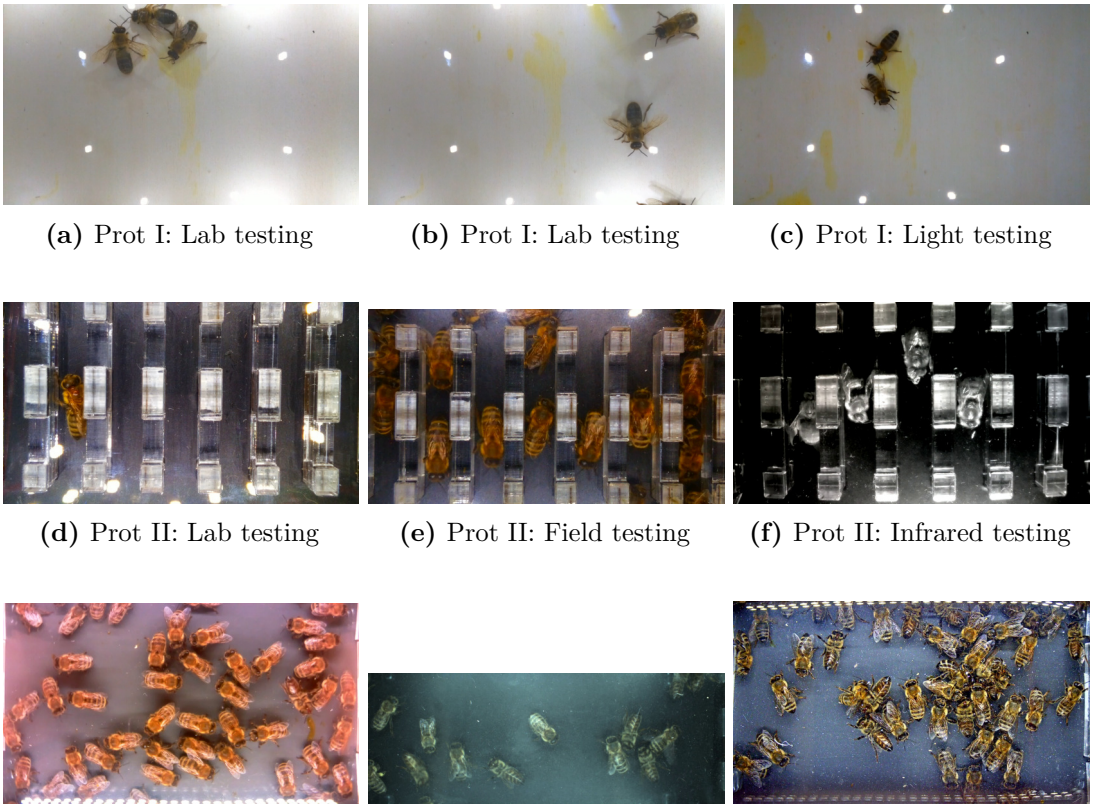
(a) **(b)**

Figure 2.7: Prototype 2. (a) Rendering of OpenSCAD 3D model. (b) Final realization with all electronic parts included. The entrance tunnel is equipped with sub-tunnels and re-positioned light sources.

the tunnel, one by one. The idea and tunnel dimensions are taken from [CYJL12], where a similar degree of detail is necessary to detect characters on attached paper tags. They also incorporate 8mm wide sub-tunnels, just wide enough to allow individual bees to pass through one at a time. This pre-sets possible walking directions, aligning all bees in one of two directions. Also they get horizontally separated. The sub-tunnels can be observed in Figure 2.7 and are visible in the videos taken (see example frames from Figure 2.8d to 2.8f).

The sub-tunnel design gives promising results, while running experiments in the laboratory environment. In the field setup it turns out unfavorable, because the entrance is easily jammed with dead bees. The tunnels do not allow dead bees to be dragged outside the hive. Bees trying to move the dead bodies jam the tunnels one by one, leading to fully sealing the entrance. This leads to the conclusion that the sub-tunnel design proposed by [CYJL12] does not work in practice and is removed from future prototypes.

Repositioning of the camera: The field of view is narrowed from 160×100mm to 100×40mm, to gain pixel density. This is done by re-positioning the camera closer to the image plane and rotating it to enable a tunnel width similar to the previous prototype. Bees are now passing through the field of view from top to bottom and vice versa, when before they moved from left-to-right and vice versa.



(a) Prot I: Lab testing **(b)** Prot I: Lab testing **(c)** Prot I: Light testing

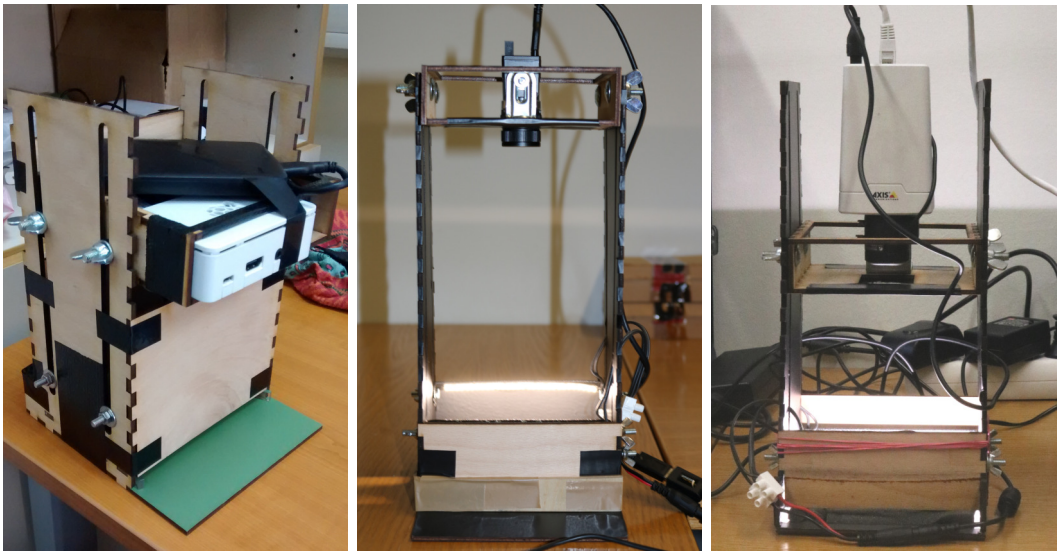
(d) Prot II: Lab testing **(e)** Prot II: Field testing **(f)** Prot II: Infrared testing

(g) Prot III: Raspberry Pi Camera Module **(h)** Prot III: Toshiba BU238MC **(i)** Prot III: Axis M1125

Figure 2.8: Example frames of recordings taken with Prototype I, II and III.

Repositioning of the light sources: By turning the camera 90° the top and bottom of the tunnel is not visible in the field of view of the camera. The light sources are re-positioned to these "blind-spots" outside the tunnel facing down on the acrylic glass. This led to the intentionally undesired reflections on the glass roof, but only occurring on the edges of the field of view. The center and the tunnels remain free from reflections (2.8e).

Experimenting with camera parameters: Due to setting the camera closer to the image plane, there is less time available for capturing objects of interest, since they pass by the sensor more quickly. From the cameras perspective the object are now bigger and move quicker through the image. This leads to motion blurring when looking at individual frames. Motion blur is reduced by testing different parameter combinations of the parameters frame rate, lens shutter speed and sensor light sensitivity. Figure 2.8d shows a scenery with un-adjusted parameters, whereas Figure 2.8e shows a different scenery with the above mentioned parameters adjusted.



(a) Raspberry Pi Camera Module **(b)** Toshiba BU238MC **(c)** Axis M1125

Figure 2.9: Prototype III. (a) Setup with Raspberry Pi Camera Module and Raspberry Pi including all necessary hardware. (b) Setup with Toshiba BU238MC camera sensor during a laboratory test session. (c) Final setup with Axis M1125 camera sensor during a laboratory test session.

Motion blur is eliminated at the cost of image quality. Additionally, Figure 2.8f shows the results of testing the scenery with infrared light and a removed infrared light filter. No features of interest are detected using the infrared light spectrum.

In total 1.853 TB of video data is recorded using this prototype. This corresponds to 514.2 hours of recording. A comparison of the recorded data with each prototype is displayed in Table 2.4.

2.1.6 Prototype III

The bottle-neck in the design of Prototype II is represented by the camera sensor and its limited capabilities of creating records without motion blur. When choosing a setup with fast shutter speeds (5ms and faster), to reduce motion blur, it comes at the cost of image quality. Since less light is passing through to the image sensor, more noise and color artifacts are visible in the final recordings. Two major approaches are followed in the design of Prototype III to reduce these effects:

Brighter light source: Due to the fast shutter times less light reaches the image sensor. To enable shutter speeds greater 6ms brighter light sources are introduced. This

Prototype	Size (GB)	Time (hours)
Prototype I	0.2	0.7
Prototype II	1,853.0	514.2
Prototype III	4,176.0	1,022.9

Table 2.4: Amount of recorded video data with each prototype.

is done by switching from 16 LEDs @12V with 279lm/m to 60 LEDs @24V with 1249lm/m.

The frames in Figure 2.8e and 2.8g are recorded using the same Raspberry Pi Camera Module sensor with the same shutter speeds of 6ms. Figure 2.8g shows this difference and is brighter and more colorful, solely by changing the light source.

Testing different camera sensors: The Raspberry Pi Camera Module is identified as the bottle neck in Prototype II. To consolidate additional candidate sensors, the following properties are taken into account:

1. Sensor size: The greater the physical sensor size, the more light is handled by the sensor, resulting in increased image quality with unchanged lighting conditions.
2. Shutter speed: A minimum shutter speed of 6ms must be possible to allow recordings free from motion blur.
3. Camera outer dimensions: Build types with outer dimensions smaller than 10 cm are preferred over bigger cameras.

Two additional candidate cameras are tested. First, the *Toshiba BU238MC* (see Figure 2.9b). It features a 1/1.2" sensor at little overall size of the camera and is produced for industrial filming at high frame rates of 60-90fps. One example frame recorded with this sensor is displayed in Figure 2.8h. When comparing video recordings acquired using this sensor to frames acquired with the low-cost Raspberry Pi Camera Module (Figure 2.8g), small increases in image quality are noticeable. But this increase does not justify the about 20 times higher costs of the Toshiba sensor.

The *Axis M1125* is the second candidate camera. It is built for surveillance applications, which is why it is network based and connected via RJ45 instead of USB. Also it ships with a built in web interface allowing easy configuration of camera parameters and offering a live preview mode. By using this camera, the video quality is be increased significantly. The video frame displayed in Figure 2.8i, is produced during a field testing session using the Axis camera. Comparing this frame to all other frames displayed in Figure 2.8, one can observe the superior image quality. The camera is bulkier than the previously tested models with outer dimensions of 148 × 44mm. Additionally an external USB networking card is necessary to

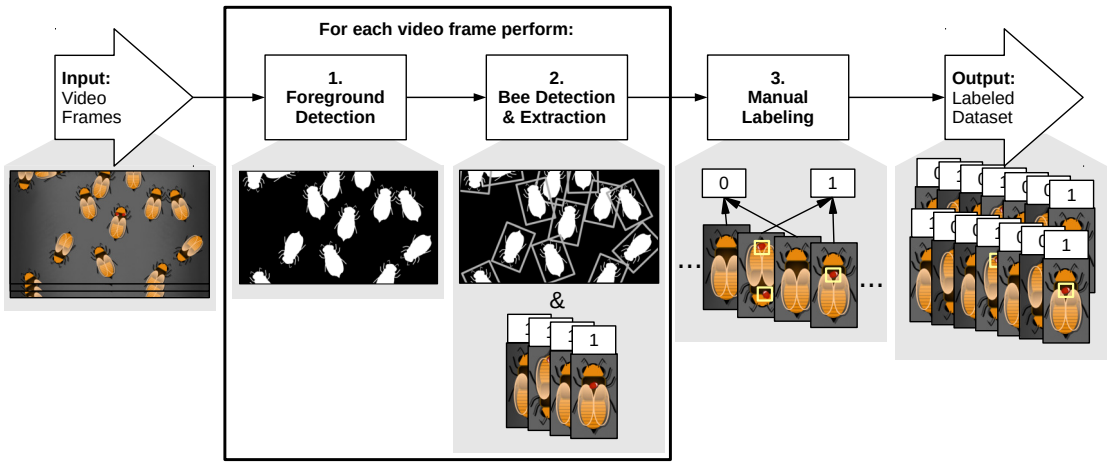


Figure 2.10: The pipeline for creating the ground truth, from input video frames to labeled data with three main stages: (1) Segment foreground (2) Detect, track and extract bees (3) Manually label the extracted images by marking the positions of Varroa mites.

compensate for the poor networking performance of the built-in network card of the Raspberry Pi. Nevertheless these drawbacks are disregarded in favor of the gained image quality.

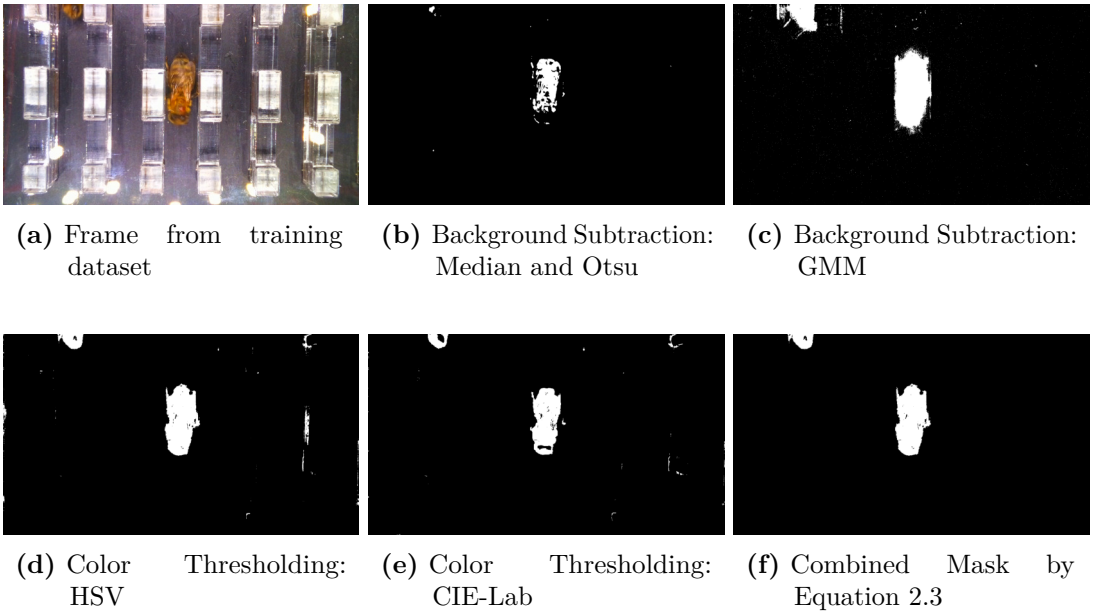
Parameter Tuning: Despite the aforementioned changes small adaptions are made to the overall dimensions. The tunnel is shortened to 120mm and widened to 142 mm (See Table 2.3 for comparison). Also new top parts visible in Figure 2.9 are designed to allow the testing of different camera sensors. The new dimensions are displayed in Table 2.3.

The data recorded with the Axis camera sensor outperformed all other camera sensors and prototypes. This is why the videos recorded with Prototype III are used as input for the next steps towards creating a ground truth dataset. In total 1.990 TB (505 hours) of lab recordings and 2.183 TB (517.9 hours) of field recordings are produced by using Prototype III with the Axis camera sensor.

2.2 Data Extraction and Annotation

After finishing the data recording phase, the data needs to be processed to allow the extraction of a ground truth dataset. All necessary processing steps are illustrated in Figure 2.10.

Up until the last step *Manual Labeling*, the pipeline works fully automated. The individual steps are further explained in the following sections.



(a) Frame from training dataset

(b) Background Subtraction: Median and Otsu

(c) Background Subtraction: GMM

(d) Color Thresholding: HSV

(e) Color Thresholding: CIE-Lab

(f) Combined Mask by Equation 2.3

Figure 2.11: Different foreground segmentation methods compared based on an example frame recorded with Prototype II.

2.2.1 Foreground Detection

In the first step, each video frame is separated into foreground and background regions. In this case, the foreground is defined by bees moving through the frames and the background as the rest respectively. The results of this step are black-and-white binary images assigning each pixel to one of the two classes.

The background in the videos is considered static due to a rigid camera position and the box setup with artificial lighting. This is why the *background subtraction* is used to segment the background from the foreground like presented in the works of [THKA16], or [CMS08]. Besides background subtraction, *color thresholding* in the color spaces RGB, HSV and CIE*L*A*B is tested. The color of the background is chosen freely, allowing contrast maximization between background and foreground. Also combinations of the two methods are tested, following the approach of [YC15].

Figure 2.11 compares the results of different foreground extraction methods. The example input frame is taken from the first lab recording session with Prototype II and displayed on the top left. In Figures 2.11b and 2.11c *Background Subtraction* is applied, to get a binary image. The idea is to subtract all intensity values belonging to the background (Equation (2.1)).

$$\boldsymbol{G}_i(x,y) = \begin{cases} 255 & \text{if} \quad |\boldsymbol{F}_i(x,y) - \boldsymbol{B}(x,y)| > t \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Each pixel of the output mask $\boldsymbol{G}_i(x, y)$ is assigned to the foreground only if its greater then a proposed threshold t after subtracting the background image $\boldsymbol{B}$ from the input video frame $\boldsymbol{F}_i$. This technique is also used in [KR16], [KTP15], [THKA16], using different approaches to calculate $\boldsymbol{B}$ and t.

Another approach is to calculate $\boldsymbol{B}$ based on a *Mixture of Gaussian* distributions (Figure 2.11c). For $\boldsymbol{B}$ all intensity values are modeled as linear combinations of k Gaussian distributions, calculated over the histogram of $\boldsymbol{F}_i$. Each pixel intensity value is assigned a probability of belonging to the foreground or background based on the k distributions. These distributions are constantly updated, making the background adapt to small changes [WPS02]. This, for instance, leads to bees slowly dissolving into the background if they are not moving.

Figure 2.11d and 2.11e, show the best results conducted with *color thresholding*. Color thresholding performs a simple thresholding operation on individual color channels c of the input video frame $\boldsymbol{F}_i$. To get the resulting output Mask $\boldsymbol{G}_i$, the Equation (2.2) is applied to each channel $\boldsymbol{F}_i^c$ defining the foreground for each channel with two thresholds t_{c1} and t_{c2}.

$$\boldsymbol{G}_i(x, y) = \begin{cases} 255 & \text{if} \quad t_{c_1} \geq \boldsymbol{F}_i^c(x, y) \geq t_{c_2} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Applying color thresholding in the default *RGB* color space gives unsatisfying results. One feature of the RGB color space is that brightness and color information represented by a single value for each color channel red, green and blue. Other color models exists like *CIE*L*A*B*, *HSV* or *HSL* which use a different representation of color information. HSV for instance, uses three color channels, but grouped into Hue, Saturation and Value. In this representation the brightness information (Value) is separated from the color information (Hue, Saturation) and therefore allows setting of independent thresholds. The Figures 2.11d and 2.11e, show the results of color thresholding. Finally a combination of functions, defined by Equation 2.3 is shown in Figure 2.11f.

$$\text{Combined}_{\text{mask}} = \boldsymbol{B}_c = (\text{HSV}_{\text{mask}} \vee \text{LAB}_{\text{mask}}) \wedge \text{Mog}_{\text{mask}} \tag{2.3}$$

Depending on which prototype is used for recording, a different approach gives best results. For Prototype I, color thresholding in CIE*L*A*B is used. For Prototype II, the $\boldsymbol{B}_c$ gives best results. For Prototype III, color thresholding in HSV color space in one single color channel works the best. Foreground pixels are then defined by all intensity values of the S channel falling into the region $[0.35, 1]$.

2.2.2 Bee Detection and Extraction

After successfully segmenting the foreground, the bees need to be identified and extracted. Goal is to retrieve individual images of bees, with and without parasites for labeling. The extraction follows the procedure illustrated by Algorithm 2.1.

Algorithm 2.1: Detecting and Extracting Bees from Videos

```
Input  : Video frame f and foreground mask m
Output : Extracted bee images
blobs = PerformConnectedComponentAnalysis(m);
forall blob in blobs do
    if blob dimension not in possible range then
        Discard blob;
    end
    else
        bee = PerformEllipseFitting(blob);
        if distance to previous bees < t then
            old_bee identified in new frame;
        end
        if old_bee has moved or is new bee then
            extracted_bee = RotateImageAndExtractBoundingBox(f, old_bee);
            SaveAsImage(extracted_bee);
        end
    end
end
```

It starts by finding connected regions in the segmentation mask using *Connected Component Analysis*. Each *blob* in the mask is assigned a label by scanning the image pixel-by-pixel and determining the connectivity to all adjacent pixels with the same value [BB16a]. All detected *blobs* are then scanned for possible bee candidates, based on their size and shape.

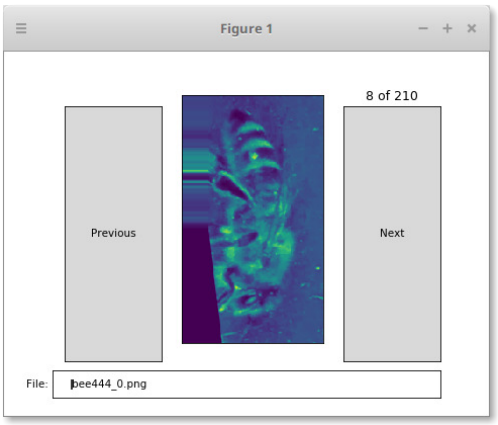
All candidate *blobs* are fitted with ellipses to determine their main orientation. This is done using the `fitEllipse` function provided by OpenCV [Bra00]. It calculates the ellipse that best fits a set of 2D points, in a least-squares sense using the algorithm proposed by Fitzgibbon et al. [FF95].

The next step is composed of a simple tracking logic. The center point of the detected ellipse is compared to the center points off all detected bees in the previous frame calculating the euclidean distance. This tracking is necessary to keep the dataset uncluttered from similar images. Bee images are only extracted if a new is found, or if the existing bee has moved a minimum of 150 pixels. This way it is ensured that non-moving bees are not extracted twice.

For extracting a rotated image patch of a bees from the video frame, the frame is rotated to align with the main axis of each detected ellipse. Each ellipse is fitted with a bounding box of size 160×280px and saved as an image file. This is done for each k^{th} video frame, where $k = 15$, so extraction is performed at two frames per second.

The resulting bee images can be observed in Figure 2.1. At this point it is noted that the

(a) Class: mite = 1 **(b)** Class: no_mite = 0



(c) Class: to_delete = -1

Figure 2.12: Screenshots from the labeling process. (a) A positive data sample, with the manually drawn selection. (b) A negative data sample not showing *V. destructor*. (c) An erroneous sample marked for deletion.

main goal of this work is not to detect honeybees in videos in a fully automated fashion. This task is subordinate and only follows the purpose of creating a functioning ground truth pipeline, which enables training of aforementioned image classifier. This is also the reason, why no effort is put into separating groups of bees.

Nevertheless for extracting images from video frames, this simple approach produces satisfying results.

2.2.3 Data Annotation

Arriving at this stage, manual intervention by users is necessary The extracted bee images need to be grouped in two distinct classes: “no_mite” and “mite” or “bee-without-parasite” and “bee-with-parasite” respectively. Data examples for each class are given in Figure 2.1. It is the users task to find the occurrences of Varroa mites in the images and mark the position in the images. By pre-determining the infestation status, a label can be pre-assigned to each extracted image. Bee images taken from a video with infected bees are assigned the pre-label “mite”, while sessions without parasites are labeled “no-mite”. This is also illustrated in the input field of the extraction pipeline (Figure 2.10). Two inputs are provided, first the video frames and second the pre-label.

During the lab testing phase it became evident, that pre-setting the infestation status does not fully correspond to the amount of extracted images actually showing Varroa mites. Looking at the final dataset (Table 2.1) around two thirds of the extracted images do not show Varroa mites, although each individual bee is originally infected. This is due to two major factors: First, the lowered tunnel height of 8mm, introduced when switching from Prototype I to Prototype II, allows bees to pass through the tunnel up-side-down. If a bee is infected with a parasite sitting on its back and is walking up-side-down, then the camera does not record the mite. Second, it is observed that Varroa mites tend to switch their host frequently. Interestingly, they tend to prefer weaker hosts, leaving their original hosts free and gather on the same weaker bees.

For the labeling task, a labeling software tool is implemented, allowing the user to make selections to mark the positions of Varroa mites and navigate through the dataset. Screen shots of this labeling tool are displayed in Figure 2.12. When a user switches to the next image, the label is set, depending on whether a selection is made or not. The label is saved into a text file with the same title as the image. Upon making a selection, the label “mite” (“1” in the file), as well as the position of the selected rectangles is saved in the text file. An example is displayed in Figure 2.12a. If no selection is made (see Figure 2.12b), the label “no-mite” (“0”) is written to the text file. A third class is added to the possible label set named “to-delete” (“-1”), which is assigned to erroneous images. These errors occur due to incorrect segmentation masks or artifacts coming from rotating and cutting of the bee images. Upon marking an image “to-delete”, the colors are inverted, visible in Figure 2.12c, to signal to the user that this image is not be used for the final dataset.

The benefit of this approach is that non-experts can be used for labeling. A person labeling the data, does not have to be a bee expert to detect the presence of Varroa in the images. Showing example images of both classes and a brief explanation of the labeling program is sufficient for this task. In total, an estimated 26 hours (around 4 working days) are consumed for creating the final dataset with $13,464$ manually labeled samples.

For efficient loading of the dataset, the individual text files resulting from the labeling application are aggregated into one single CSV file. Within the CSV, each data sample is represented by one line holding the path to the image, as well as the label and bounding box information. In this step, the images marked as “to-delete” are excluded from the

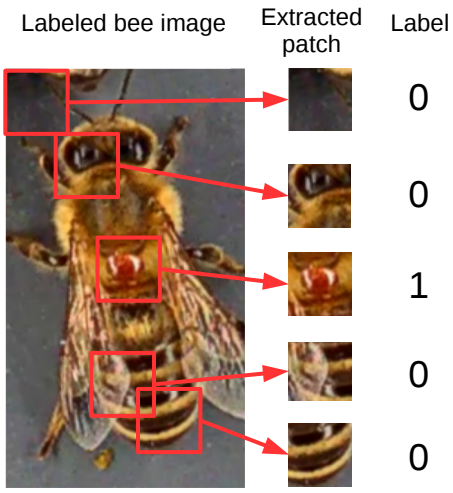


Figure 2.13: Sliding window visualized

process. The CSV and the data samples together form the final ground truth.

2.3 Dataset Post Processing

The previously explained steps are necessary for computing the ground truth. Starting from here, the ground truth is further processed to prepare the data for training a classifier. Two major processing steps are performed:

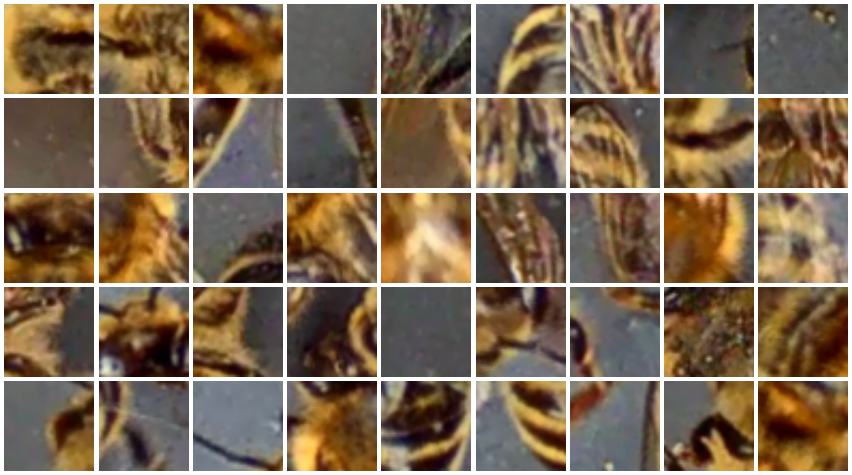
1. Patch Extraction, splitting the bee images into smaller sub-window-images, which are used as input for the recognition models.

2. Data Augmentation, performing translations on the data with the goal of creating more data samples.

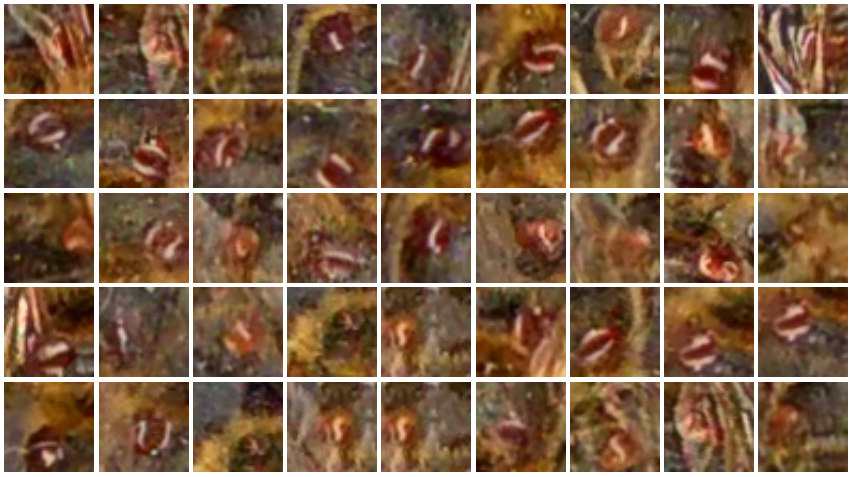
2.3.1 Patch Extraction

To reduce the risk of fitting the model to unwanted correlations in the data, it is trained and evaluated on sub-image patches instead of complete bee images. Examples for unwanted correlations are the orientation of the bee, or illumination changes in the background. This way, the amount of background information is minimized supporting the model in calculating features specific to classifying Varroa mites.

Patch extraction describes the process of extracting sub-image patches from the labeled bee images. These sub-image patches can be of arbitrary sizes smaller than the original image. At the full bee image resolution of 160×280px the average size of Varroa mites is around 25px in diameter. To grasp some of the background with a Varroa mite, a fixed window size of 42×42px is used for the sub-images.



(a) Examples of negative sub-image patches.



(b) Examples of positive sub-image patches.

Figure 2.14: Sub-image patches after patch extraction. (a) Negative data samples not showing *V. destructor*. (b) Positive data samples showing *V. destructor*.

To obtain the sub-image patches, a *sliding window function* is used, as presented in [VJ01]. A window is moved over the image, extracting sub-image-patches. Figure 2.13 visualizes the extraction, where the rectangle represents the sliding window. Each position the window resides in, the pixels are extracted and saved as a new image patch. Each extracted sub-image patch represents a data sample and is assigned a label from the ground truth. If the original image is a positive sample, each patch overlapping with the position of the Varroa mite, is marked as positive ("1"). All other samples are marked as negative ("0").

Dataset	# of samples	Class 0	Class 1	Ratio
Train dataset	954,967	951,507	3,460	1 : 275
Test dataset	509,643	508,186	1,457	1 : 348.8
Validation dataset	226,364	225,369	995	1 : 226.6

Table 2.5: Ground truth datasets after applying the sliding-window patch extraction.

The function is provided with four parameters: (1) Input image, (2) Border padding (3) Sub-window-size and (4) the stride to move the extraction window. The image size is constant at 160×280px and border padding is set to 5px, leaving a center cropped 155×275px for scanning. The window size is set to 42×42px to ensure that the mites are fully captured. The stride is set as big as possible while ensuring that a mite can always be fully represented in one patch. Setting the stride to $(14, 14)$, provides an overlap of one third between the windows ensuring full representation. This results in an extracted 153 sub-image patches from one input bee image. Table 2.5, shows an overview of the ground truth data, after applying patch extraction with the aforementioned configuration. Data examples of both classes after extraction are illustrated in Figure 2.14.

2.3.2 Data Augmentation

The generalization power of classifiers is increased by applying random transformations on the input data to effectively increase the amount of available data for training. By generating new samples from existing ones, a higher variability is represented inside the data, making the classifier less prone to overfitting. Enhancing the generalization power is not the main focus of using data augmentation in this work. Here it serves the purpose of reducing imbalances in the dataset that result from patch extraction. Augmentation is only applied to the positive class because a sufficient amount of negative samples exists already. Also augmentation is only applied on the training dataset.

To enable full comparison of the "traditional" and deep learning approach, both methods need to be trained and evaluated on the same datasets. This is why data augmentation is performed offline, before actual training rather than online during training. The augmented image patches are saved to disk to be loaded at a later stage.

For the augmentation the python library `imgaug` is used. It provides the function `SomeOf()`, which is used to apply a minimum combination of two of the following augmentation steps:

- `GaussianBlur(sigma=0.8)`: A gaussian filter-kernel function blurring the image with the intensity defined by *sigma*.
- `Fliplr()`: Mirror a data sample from left to right.

- `Flipud()`: Flip the image up-side-down. This way the orientation of the bee is randomized.
- `PiecewiseAffine(scale=(0.03, 0.05), mode='edge')`: Performs a piece wise scaling operation distorting the image using 2 randomly chosen points inside the image. The intensity of the distortion is managed by the *scale* parameter. Emerging holes are filled by replicating the edge pixels.

CHAPTER 3

Classification Architectures

The basis for creating a detection and classification algorithm is provided by the datasets which are now used to train classification models.

3.1 Classification Problem Description

The parasite detection problem can be split into two simpler problems: First a detection problem and second a classification problem.

The goal of the classification step is to predict the infestation status of a honeybee, as either healthy or infected with Varroa mite. For this *supervised learning* is used. It consists of a collection of algorithms that learn the parameters of a function f^* which associates an input $\boldsymbol{x}$ with an output y, given a set of training examples. The training samples consist of images $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, and corresponding class labels $y_1, y_2, \ldots, y_3$. In this work the classification problem is a binary problem: $y \in \{0, 1\}$, where bees are either infected with a parasites ($y = 1$) or not ($y = 0$). In general this can be formulated as [GBC16]:

$$y = f^*(\boldsymbol{x}) \tag{3.1}$$

Training a classifier means finding a function f^*, that maps the input $\boldsymbol{x}$ to the category y. $\boldsymbol{x}$ represents a bee image to be classified and y represents the true label.

The dimensions of the raw pixel data of image $\boldsymbol{x}$ is generating the *input space*. For discriminating between classes, f^* needs to be insensitive to variations in the input data, such as illumination changes or background changes. But on the other hand, it needs to be sensitive to changes particular to the different classes of y. Working in the input space does not allow this kind of differentiation. This is why the input is transformed into the *features space* before passed onto f^*, depicted in Equation (3.2).

$$\boldsymbol{z} = \phi(\boldsymbol{x}) \tag{3.2}$$
$$y = f^*(\boldsymbol{z}) = f^*(\phi(\boldsymbol{x})) \tag{3.3}$$

This transformation is performed by a function ϕ, which produces a feature representation $\boldsymbol{z}$ of the input data $\boldsymbol{x}$ that better qualifies for discrimination. ϕ performs a non-linear transformation on the input data, which can be thought of as providing a set of *features* representing $\boldsymbol{x}$. This is beneficial when the feature space produced by ϕ is linearly separable. Then the function f^* can be a simple linear function, defining a hyper-plane in n-dimensional space.

In the following sections, two different approaches for choosing ϕ and f^* are represented by a "traditional" machine learning and a deep learning approach.

3.2 "Traditional" Machine Learning Pipeline

The "traditional" machine learning approach consists of a manually engineered transfer function ϕ, followed by a "shallow" classifier. ϕ is composed of handcrafted features that transform the input image into a feature vector. f^* is represented by a "shallow" classifier like *Random Forest* or *Naïve Bayes*. Nevertheless "shallow" classifiers in combination with feature selection is valuable to consider, due to the following:

- It offers the ability to engineer ϕ: The "traditional" approach requires the expertise of the designer to create ϕ. This creates a comprehensible workflow from input data to final classification result.
- Challenges when working with deep learning technologies: The progress in deep learning is achieved by exploring architectural variants on an experimental basis [AS16]. There exists no full understanding of how to choose structural parameters or how to efficiently tune hyper parameters. Solely rough guidelines and recommendations are available [AS16].

A pipeline for training "traditional" models is presented in Figure 3.1. It depicts the workflow of how "traditional" models are created, with all intermediate steps necessary. The input to the pipeline is the labeled train dataset from the ground truth and the output is a trained model, which is used for making predictions.

In the first step the data is preprocessed into sub-image-patches. The patch extraction results in $(42, 42, 3)$ dimensional sub-image-patches, which are vectorized for further processing. So each patch is transformed into a 1D vector with dimensions $(1, 5292)$. In the next preprocessing step, data augmentation is applied, to result in an evenly balanced dataset for training. This results in a total of n sub-window-patches, which are stacked to form a $(n, 5292)$ dimensional input array for training.

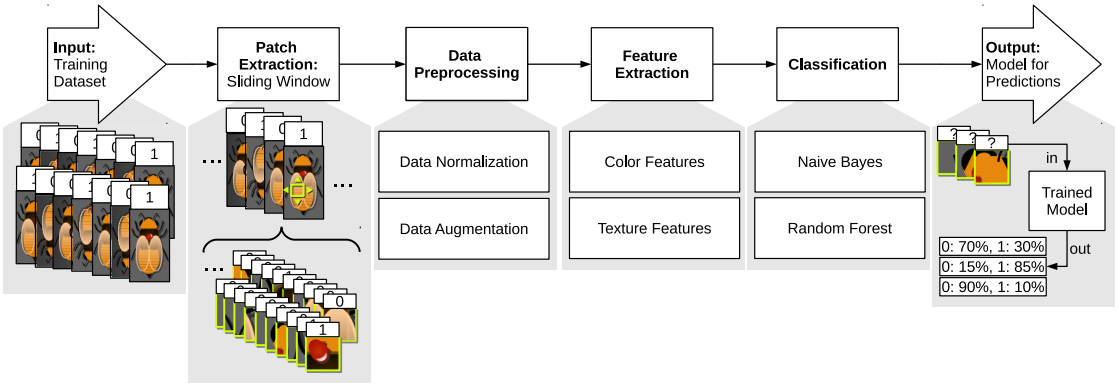


Figure 3.1: The pipeline for creating a “traditional” machine learning model. Input is the labeled training dataset. Output is a trained model, which can be used to predict unknown samples.

In the feature extraction step, the preprocessed input data is transformed into the feature space. The dimensions of this space depend on the combination of features, as well as on the individual configuration of a feature. For example, when extracting the feature “color mean”, the dimensions of the output array transforms to $(n, 3)$ with one mean value for each original color channel of the input images. A complete list of available features is displayed in Table 3.1, with “# of Params” giving the dimension of the feature space. The result of the extraction is an array with dimensions $(n, \text{len(feature space)})$, consisting of n feature vectors for each data sample.

In the next step, this array in combination with the corresponding ground truth labels is used for training a classifier . Two classifiers are tested in this work. A classifier is trained by calling the `fit()` method, passing the feature vectors and the true labels of each training sample.

The result of the pipeline is a trained classifier, which can be used to predict unseen inputs. Predicting unseen samples is done by following a similar pipeline:

Imagine an unseen bee image of size 160×280px. In the first patch-extraction step, the image is turned into an array of 153 sub-image-patches of 42×42px, which gives an input array of dimensions $(153, 42, 42, 3)$ or $(153, 5292)$. No data augmentation needs to be applied when predicting. The same feature extraction steps used for training are now applied on the $(153, 5292)$ dimensional input array. The resulting array of feature vectors is then passed onto the trained classifier by calling the `predict()` function of the pipeline. The output of this prediction is a $(153, 2)$ dimensional array, giving the probability estimation for every input patch belonging to one of the two output classes. To complete the prediction, the individual patch results are summarized and a final label is assigned.

The complete process is published in the year 2018 at the International 15th International Conference on Image Analysis and Recognition (ICIAR 2018) [SBPM18].

Feature	Description	# of Params
color hist k bins	Color histogram with k bins, for each color channel	k * 3
color mean	Mean value per color channel (1. statistical moment)	3
color std	Standard deviation per color channel (2. statistical moment)	3
color skew	Skew value per color channel (3. statistical moment)	3
texture SIFT	Scale-Invariant Feature Transformation extracted from center point	128

Table 3.1: The list of feature candidates. Four different color features as well as one texture feature qualify for representing ϕ for the "traditional" machine learning approach.

3.2.1 Feature Extraction

The goal of this step is to extract characteristics of positive and negative image patches taken from the training dataset, to distinguish between patches showing and not showing *Varroa destructor*. In the ideal case, the extracted features would make this classification problem trivial. They should be robust to irrelevant transformations in the data like translations, rotation and scaling but highly specific variations that are important for discrimination. Defining these features is a complex task requiring a high level of expertise [LBH15].

The input to the feature extraction is composed of images $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, which are spatially limited arrays encoding brightness and color information. Spatial information, is represented by groups of pixels forming visual objects or textures, while color information is represented by assigning each pixel a vector of color information. In the case of RGB-images, this vector is of length three and holds a value for each of the base colors red, green and blue.

Possible features are identified after observing the input data shown in Figure 2.14 leading to the following observations:

- Distinct parasite colors: When observing the comparison in Figure 2.14 the most prominent difference is the color. The reddish to brown color is visually different from other colors observed in negative image patches (Figure 2.14a).
- The position of a parasite varies: Parasites are not bounded to any specific spots on the bee, although they might prefer certain areas. Nevertheless they are potentially spotted everywhere on the bees corpus. This results in non-static environments close to the boundaries of parasites.

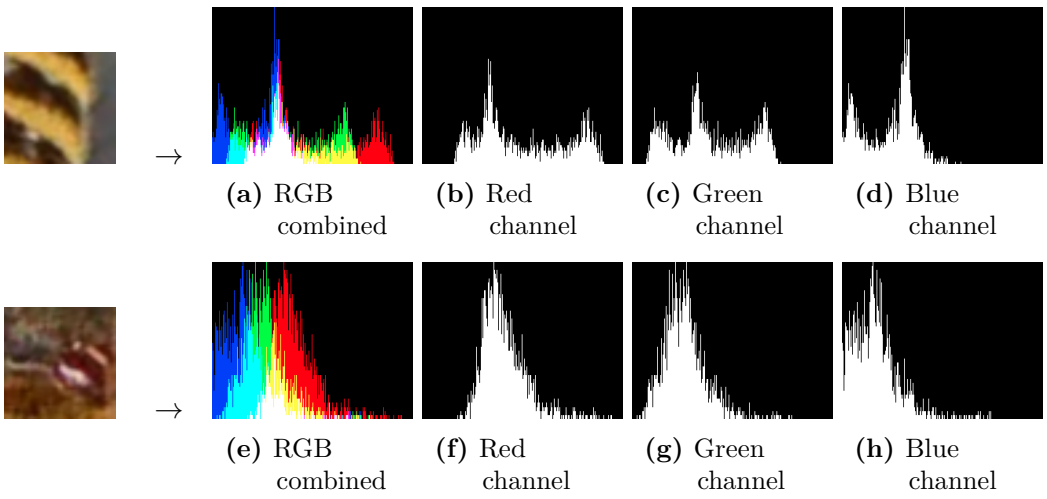


Figure 3.2: Color histograms with $k = 256$ bins for both a negative (top line) and a positive (bottom line) sample. (a) and (e) show a combined version of each histogram with the channels R,G and B summed up.

- Varroa mites are of round shape: This property can be extracted using shape descriptors like SIFT.
- Light reflections on the chitinous exoskeleton of Varroa mites: Due to the artificial lighting, reflections are visible in regular patterns on the backs of Varroa mites. This adds texture to the mites shape, which is not always present when looking at the input samples in Figure 2.1.

The observations led to a list of feature candidates described in Table 3.1. Two types of color features: Color Histogram and Color Moments are tested, as well as SIFT as a representative for texture and shape features.

Color Histograms

Color information is used to extract low-level features from images and is applied to different computer vision problems [SS01]. This is mainly due to their invariance to scaling, translation, rotation and partial occlusions [KS04], [AK11],.

Color is defined by three or more values that are referred to as *channels*. The standard format is RGB, but other representations like HSV or CIE-L*A*B* exist.

The most widely used color feature is the *color histogram* [KS04]. A histogram h_k defines a function m_i that quantifies the given space into k one dimensional sub-regions, also called *bins*. This is done by counting all pixel values falling into the disjoint regions $i = 1, \ldots, k$. A plot of a histogram shows a function of pixel intensity values with peaks and valleys corresponding to the amount of intensities. This is independent of

their position inside the image, which creates translation invariance. With $k = 256$ two histograms are displayed in Figure 3.2, which are calculated on a positive and a negative patch sample.

The best value for parameter k is found in an experimental process during hyper parameter search. The feature vector is yielded by creating a histogram for each available color channel separately. Hence, the length of this vector is $3 \cdot k$, for a three-channel color image.

Color Moments

Another group of color features is composed of *statistical color moments* computed on the intensity values of an input image. They describe a statistical representation of color by interpreting the color distribution as a probability distribution. These allow the computation of characterizing moments. In Stricker et al. [SO95] three main moments are calculated on the image's distributions: (1) *mean*, (2) *std* = standard deviation and (3) *skewness*.

The first moment: *color mean* is defined by Equation (3.4) and defines the average color value $mean_c$ for a specific color channel c.

$$mean_c = \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{x_{ij}^c}{m \cdot n} \tag{3.4}$$

x_{ij}^c is the color intensity of an image $\boldsymbol{x}$ at position ij of a color channel c. So after extracting this feature from a three channel color image ($c \in \{0, 1, 2\}$), the resulting feature vector has the length 3. This also applies to all following color moments.

The second moment, is represented by the standard deviation, defined by Equation (3.6).

$$variance_c = \frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij}^c - mean_c)^2 \tag{3.5}$$

$$std_c = \sqrt{variance} \tag{3.6}$$

It is interpreted as a measure of the color homogeneity in each channel. The higher the variance of color values within one color channel, the higher the *std*.

The third moment is called *skewness* and defined by equation (3.7).

$$skewness_c = \frac{\frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij}^c - mean_c)^3}{\sqrt[3]{\frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij}^c - mean_c)^2}} \tag{3.7}$$

The skewness can be understood as a measure of the asymmetry in the distribution of color values per channel. It is 0 for a perfectly symmetrical distribution. If the

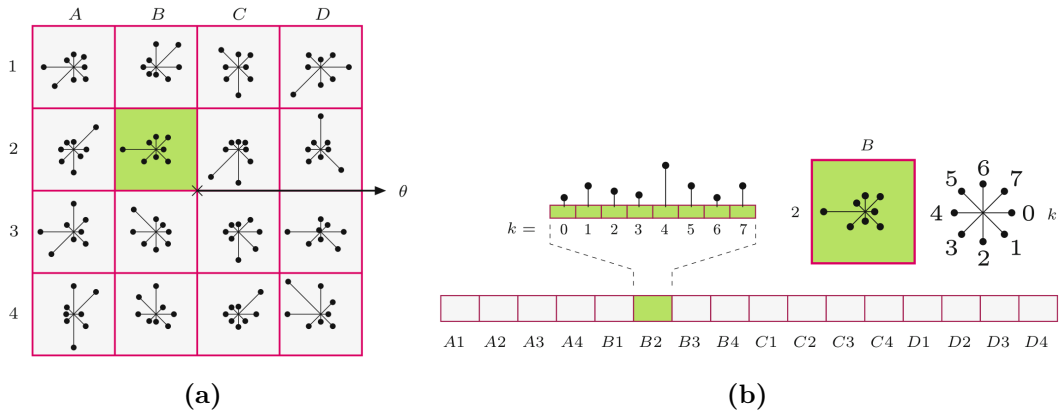


Figure 3.3: Calculating a SIFT feature vector. (a) The 4×4 sub-regions around a centered key point. (b) The final feature vector with the k computed gradients for each of the 16 sub-regions. [BB16b]

distribution is not symmetrical but tends towards the left side, it tends to positive values. If it tends to the right, the skew becomes negative.

Scale-Invariant Feature Transform (SIFT)

The next group of features is defined by texture and shape features. Here only one feature is extracted, namely *SIFT*.

The *Scale-Invariant Feature Transform (SIFT)* is proposed by D.Lowe in 1999 [Low99]. It describes a multi-scale corner detection algorithm, that is refined with a rotation-invariant feature descriptor attached to each key point. The main use of SIFT is to detect and recognize local key points in images, while tolerating transformations and scale changes [BB16b].

It consists of two major steps: (1) Efficient key point detection and (2) feature vector description. In the first step the goal is to find points of interest in the image. This key elements are identified using *Laplacian-of-Gaussian* filters, which detect corners and edges in the image. Basically the filter detects bright blobs surrounded by dark regions and vice versa. The filters are applied in different scales, followed by refinement steps to end up with a final set of interest points [BB16b].

In the second step local descriptors $\boldsymbol{k'} = (x, y, n_\sigma, \alpha)$ are created for each interest point. This is done by sampling the surrounding image gradients and creating a histogram of gradient orientations. (x, y) determine the key points spatial position. α is the angle of the dominant gradient vector at position (x, y). n_σ represents a scale parameter and sets the area of the surrounding pixels to incorporate for computing the gradients. It is set depending on the magnitude of the dominant gradient vector at the key points position. The area defined by $n_\sigma x n_\sigma$ is divided into 4×4 subregions, with 16 regions in total. In

Figure 3.3, these sub-regions are referred to by A-D and 1-4. For each sub-region, k gradients are computed. Usually $k = 8$ and determines 8 different directions, like depicted in Figure 3.3b. All the gradients are computed in a pre-set order shown in Figure 3.3b, which results in a final feature vector of length $16 * 8 = 128$.

The main use of SIFT is in Multi-view matching and image stitching [HZ03]. To use SIFT descriptors in the feature extraction pipeline, the output produced needs to have constant dimensions. Each key point descriptor has a constant length of 128 elements, but the amount of key points can vary between samples. This is avoided by extracting a fixed number n_p of key points, producing an output vector of dimension $(n_p, 128)$ for each sample. A simple implementation of this idea is to define a regular grid with n_p elements and compute $\boldsymbol{k}'$ at these positions. By pre-setting the position of key points, a dependency towards the location is created and translation invariance is voided. Nevertheless this dependency is tolerable due to the patchification pre-processing. By extracting sub-images from the original complete bee images, the input becomes invariant to translation. The majority of pixels of a sub-image patch either show a Varroa mite or not, which makes the key point search irrelevant. Also due to the small input image size of 42×42px, n_p is set to 1, with setting the scale parameter $n_\sigma = 42$ to cover the complete patch area. This way all pixel information of the image patch is incorporated when calculating the 128 element feature vector.

Another approach, not followed in this work, is to use Bag-of-Words (BoW) architecture to enable SIFT descriptors for classification [ATK+15]. Here the computed key points off all training samples are grouped in k clusters. Each sample is assigned to the closest cluster and represented in a histogram forming the final feature vector.

3.2.2 Classification

There exist two groups of classifiers: (1) generative and (2) discriminative models [MCR+17]. Generative models are also referred to as non-parametric models, because they do not rely on parameter tuning for computing probabilities. They estimate the likelihood of belonging to class C for a sample represented by its feature vector $\boldsymbol{z}$:

$$P(\boldsymbol{z}|C) \tag{3.8}$$

Which is the probability of the occurrence of feature vector $\boldsymbol{z}$, when considering the class C. A representative of this type of classifiers is the *Naïve Bayes* classifier.

The group of discriminative models is parameterized and is trying to set a boundary in a given feature space between different populations to separate them. In stochastic terms, they are calculating the probability of class C given a feature vector $\boldsymbol{z}$:

$$P(C|\boldsymbol{z}) \tag{3.9}$$

Examples are: *Linear Support Vector Machines (SVM)*, *Decision Trees (DTs)* or *Random Forests (RFs)*.

Naïve Bayes and Random Forest are used in this work, to provide a representative of both a parameterized and an non-parameterized model. Both classifiers expect an array of feature vectors with the dimension $(n, len(\text{feature vector}))$ as input, where n is the number of samples to classify. The output of each classifier is a predicted confidence for each sample of belonging to one of the two output classes. This allows for later setting a desired separation boundary between the classes, different from the default 0.5

Naïve Bayes Classifier

The Naïve Bayes classifier is a simple but effective classifier applying the *Bayes' Theorem* (Equation 3.11) to calculate the probability of an input $\boldsymbol{z}$ belonging to a certain class C [Ras14].

$$P(C|\boldsymbol{z}) = \frac{P(\boldsymbol{z}|C) \cdot P(C)}{P(\boldsymbol{z})} \tag{3.10}$$

$$\text{posterior probability } = \frac{\text{likelihood} \times \text{prior probability}}{\text{evidence}} \tag{3.11}$$

It proposes, that the *posterior probabilities* of observing an event can be calculated based on the prior probabilities and the likelihood of that event. In a classification scenario the posterior probabilities can be interpreted as the probability that a particular feature vector $\boldsymbol{z}$ belongs to a class C given its feature values.

The *likelihood* is the class-conditional probability for a class C to observe a feature vector $\boldsymbol{z}$, which is directly estimated from the training data (Equation (3.12)). This estimation can only be made under the assumption that the feature values $z_1, z_2 \dots z_n$ of $\boldsymbol{z}$ are statistically independent from another. This means that the probability of observing z_i does not affect the probability of another observation z_j with $i \neq j$. Further it (naively) assumes *conditional independence* of the features:

$$P(\boldsymbol{z}|C) = P(z_1, z_2 \dots z_n|C) = P(z_1|C) \cdot P(z_2|C) \cdot \dots \cdot P(z_n|C) = \prod_{i=1}^{n} P(z_i|C) \tag{3.12}$$

Under this assumption, the likelihood can be calculated as a product of the individual likelihoods for each feature. These are estimated using the maximum-likelihood estimate, which takes the frequency of observing z_i in C over the total count of all features in C.

The *prior probability* is also referred to as *class priors*, which describe the general probability of observing a particular class. It is also estimated from the training data, once more using the maximum-likelihood estimate.

The *evidence* is the probability of encountering a particular feature vector $\boldsymbol{z}$ independent from the class label. It can be interpreted as scaling factor and is calculated using Equation (3.13).

$$P(\boldsymbol{z}) = P(\boldsymbol{z}|C) \cdot P(C) + P(\boldsymbol{z}|\neg C) \cdot P(\neg C) \tag{3.13}$$

All parameters of the Naïve Bayes classifier can be calculated as presented and no hyperparameters need to be set. The condition of independent features is likely to be violated since the features can have unseen correlations.

Random Forest Classifier

Random Forest is the second classifier tested in this work. It is a representative of the group of discriminative classifiers and performs a non-linear mapping from input to output.

A Random Forest is a composition of simpler or weaker classifiers called *Decision Trees (DTs)*. This technique of having an ensemble of weak classifiers voting for the most popular class is called *ensemble learning*. It is a basic technique to reduce the risk of overfitting a training dataset [Bre01]. The forest is grown in a divide-and-conquer manner, also referred to as *bootstrap aggregation*: (1) Divide the data into disjoint subsets (bootstraps). (2) Grow a randomized tree predictor (weak classifier) for each subset. (3) Aggregate the results of the weak predictors together.

A weak learner in Random Forest is represented by a binary Decision Tree, which is acquired by recursively dividing the feature space into sub-regions. For building a tree, first a root node is chosen following a criterion. Each node consists of two child nodes (left and right) that further refine the division. Where to choose the optimum split is based on a decision rule, for instance by maximizing a "purity" criterion, in the case of classification. Each left and right child node are further split into two consecutive child nodes in a recursive manner. The procedure stops, if all samples in a leaf node belong to the same class, a maximum tree depth is reached, or all training samples have been used up.

For predicting the class of an unknown sample, the feature vector is passed to the root node of the individual DTs and traversed through the trees. Once a leaf node is reached, the value assigned to this node is used as the predicted output class. Each DT votes for one of the two available classes and the majority vote makes the final prediction.

The benefit of Random Forest is that they can be applied to a wide range of prediction problems, with only a few parameters to tune [BS16]. The only manually chosen parameter in this work is the amount of trees in the forest, which is left at a default value of 10.

3.3 Deep Learning Pipeline

A different way of regarding the classification problem is to learn the transfer function ϕ from the data, instead of manually engineering it. Simple linear models serve as basis for a learnable ϕ (Equation (3.14)).

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^T \boldsymbol{w} + b \tag{3.14}$$

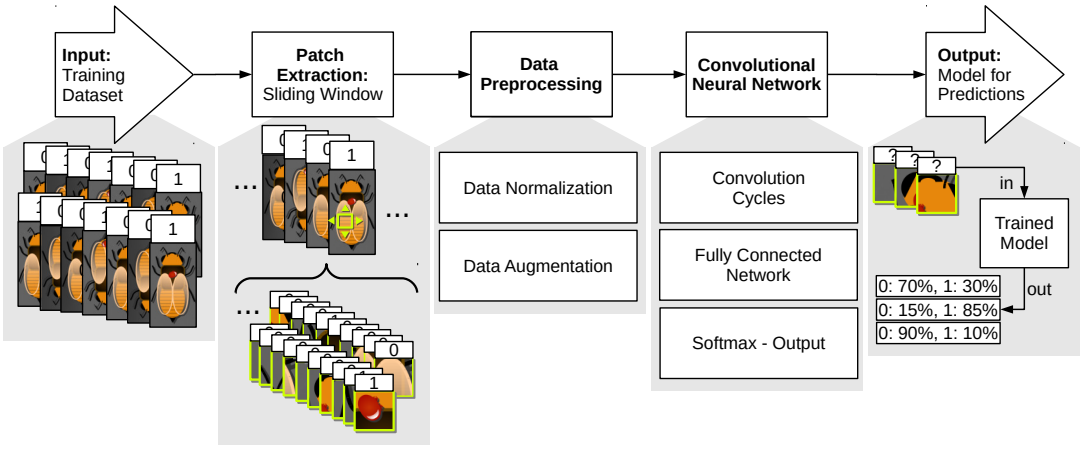


Figure 3.4: The pipeline for creating a ConvNet model. Input is the labeled training dataset. Output is a trained model, which can be used to predict unknown samples.

The generalization to non-linear problems is achieved by combining the results of the linear models with a non-linear function g, referred to as *activation function*. The linear models are referred to as *nodes* or *neurons*. By combining a multitude of these neurons in a non-cyclic way, one can create a network or neurons, also referred to as multi-layer perceptron or *Artificial Neural Network*.

This idea of combining simple entities to form complex models is inspired by actual biological neural networks found, for instance, in the human brain. Here the nodes are represented by biological neurons [AH17]. The interconnections of neurons are represented by the weight for each input connection to a neuron. If a stimulus reaches a certain threshold in a biological neuron, it fires an action potential. This activation of the neuron is modeled by the non-linear activation function g. The idea is founded by F. Rosenblatt in 1958 with the proposal of a single layer perceptron, which describes one linear entity of a network [Ros58].

The pipeline for creating a deep learning model is depicted in Figure 3.4. Up until the *Convolutional Neural Network* step, this pipeline is similar to the "traditional" pipeline. So the data undergoes the very same pre-processing steps before passed on to the deep learning model. In contrast to the "traditional" pipeline, no distinct feature extraction steps are necessary and the data is directly passed to the input layer of the network. The network is trained with a call to the `fit()` method, passing the pre-processed training dataset in the form of training batches. The output of the network is a trained classifier, that can be used to predict the label of unseen samples.

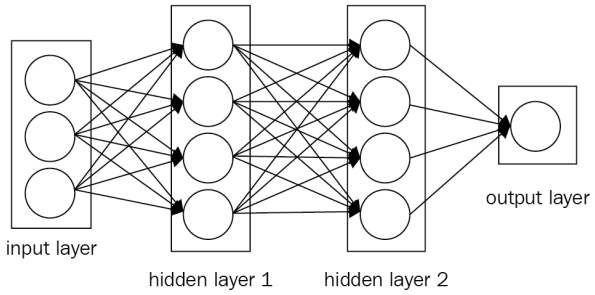


Figure 3.5: A three-layer fully connected feed forward artificial neural network. All connections are directed from input to output creating an acyclic graph. The input has three neurons, the two hidden layers have four neurons each and the output has one neuron [SKP18].

3.3.1 Deep Artificial Neural Networks

The neurons in a network are grouped in layers forming an acyclic graph, having all connections directed from input to output. In the case of a *fully connected network*, each neuron of a layer is connected to each other neuron in the previous layer (see Figure 3.5). This acyclic property does not comply with the biological model, but is necessary for efficiently training such networks. These networks are also referred to as *Feed-forward Networks*.

Figure 3.5 shows a very simple three-layer ANN. It is referred to as a three layer model, because the output layer is omitted when counting the number of layers. All layers between input and output are referred to as *hidden layers* of the network. In Figure 3.5 two hidden layers $\boldsymbol{h}^{(1)}$ and $\boldsymbol{h}^{(2)}$ are illustrated, with four neurons each, followed by the output $\boldsymbol{h}^{(o)}$ with a single neuron.

The grouping of linear nodes into layers is described by equation (3.15), which shows the process for a single image $\boldsymbol{x}$:

$$\boldsymbol{h}^{(i+1)} = f^{(i)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}) = g(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{c}) \tag{3.15}$$

$\boldsymbol{h}^{(i+1)}$ is the output of the ith layer. $\boldsymbol{W}$ is an array of weight vectors $\boldsymbol{w}$ and $\boldsymbol{c}$ is the vector of biases b from the linear models presented in Equation (3.14). The function g is the non-linear activation function, that can be of various forms. A common way is to choose g as *rectified linear unit* [GBB11]. Only by applying this non-linearity, the model can generalize to non-linear problems.

The final representation of f and ϕ is created by chaining the layers. The output of the input layer is the input for the first hidden layer and so on:

$$\boldsymbol{y} = f(\phi(\boldsymbol{x})) = \boldsymbol{h}^{(o)}(\boldsymbol{h}^{(2)}(\boldsymbol{h}^{(1)}(\boldsymbol{x})) \tag{3.16}$$

In general this chaining of functions can be of arbitrary length, creating *deep neural networks*.

3.3.2 Convolutional Neural Networks (ConvNets)

A subset of deep neural networks is defined by *Convolutional Neural Networks*, also referred to as *ConvNets* or *CNNs*. When working with images as input to a network, the number of neurons and therefore parameters of the network can be very high even for "shallow" networks. This is due to images carrying a lot of information, which needs to be handled by the network. For example, assuming a 16×16px grayscale input image connected to a hidden layer with 7200 neurons already result in $16 \times 16 \times 7200 = 7,372,800$ different parameters. Imagining multiple hidden layers in combination with bigger and multi-channel color input images results in millions of parameters to train. The goal of ConvNets is to reduce the number of parameters allowing networks to be deep with much less parameters [AH17]. This is done by introducing new types of layers to the architecture:

Convolution Layers

Two principles are followed for reducing the amount of parameters: (1) Spatial information is encoded by groups of pixels (2) Weight-sharing between groups of neurons. The first principle is based on the assumption that a pixel in an image is highly correlated with its close neighbors and loosely correlated with its far neighbors. This is considered by defining sub-image regions of smaller sizes e.g. 5×5px, which are connected to a neuron in the next layer. This implies, that the network is not fully-connected anymore. Considering the example from before, the parameters can be reduced to: $(5 \times 5) \times 7200 = 180,000$. To further reduce the amount of parameters the second principle, *weight-sharing* is applied: The 7200 neurons in the hidden layer can be re-arranged to 50 blocks of 12×12 neurons. The neurons inside each block are now set to share its weights with all other neurons in the same block. This way further local information is encoded and the amount of parameters reduces to $5 \times 5 \times 50 = 1250$, which is a reduction of more than 99.98% compared to the fully connected network. Combining the two operations is similar to a convolution operation with a filter kernel in the spatial domain. This is why these layers are named *convolution layers* and the result of convolving a filter with the input image is called *feature map*.

Convolving a filter with a multi-channel input, always results in a single channel output [AH17]. Input images are three dimensional, with the third dimension equal to the number of channels in the image, for example $(42, 42, 3)$. The output of convolving a 5×5px filter with this image, results in a one channel image with dimensions $(38, 38, 1)$ So when applying 50 of these 5×5-convolution filters in a convolution layer the output is a 50-channel image $(38, 38, 50)$ with each channel representing the filter result of a different kernel.

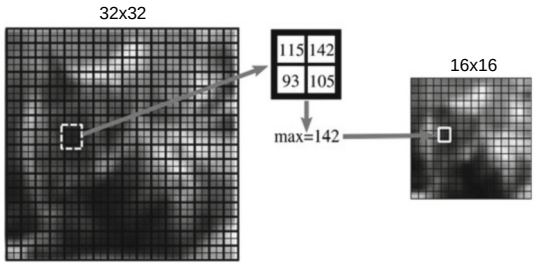


Figure 3.6: Max-pooling operation with kernel-size $d = 2$ and stride $s = 2$ applied on a 32×32px input image. The resulting image is 16×16px, which is a spatial reduction by the factor 2.

ReLU Activation

In case of this work, a convolution layer is always followed by a Rectified Linear Unit (ReLU) activation function. This is necessary to provide non-linear mappings. The function cuts of negative values at zero, depicted in Equation (3.17) for a random $d \in \mathbb{R}$.

$$g(d) = \max\{0, d\} \tag{3.17}$$

Different functions can be used to introduce non-linearity. Historically the sigmoid-function is used to train multi-layer-perceptrons [Gol17]. But this can lead to a problem with vanishing gradients, when the signal propagates from the last to the first layer during training [GBB11]. This is why activation functions of the ReLU family are mostly used nowadays [AH17].

Pooling Layers

Another way of reducing the amount of parameters in the network, is to use *Pooling*. This is done by spatially downsampling the resulting feature maps from convolution operations. Then use *max-pooling* with a kernel-size of $d = 2$ and a stride $s = 2$ (Figure 3.6). This extracts the maximum pixel value within a $d \times d$ grid across the image. As a result the output is spatially reduced by the factor s. Alternatively average-pooling can be used, but it is proven to perform worse [SMB10]. Pooling layers do not have any trainable parameters.

Dropout Layers

In the AlexNet [KSH12], a technique called *Dropout* is used, to reduce overfitting. Dropout sets the output of each hidden neuron to 0 with a probability of 0.5. This way, neurons drop out at the forward pass phase during training and as a conclusion do not participate in the backpropagation. As a result, more robust features are learned because uncertainty is removed [HSK+12].

Softmax Output Layer

The *softmax* layer is an output layer that applies the *softmax function* on a resulting vector. The softmax function is a logistic function, that treats the scores of a output vector $\boldsymbol{y}$ as unnormalized log probabilities [Kar18]. Exponentiating these gives unnormalized probabilities which are divided by the sum over all probabilities to sum to one (Equation (3.18)). This is a favorable property because the output gives the confidences of the classifier for observing each individual class C. This is similar to the maximum likelihood estimation performed in the Naïve Bayes classifier.

$$P(C|y_C) = \text{softmax}(y_i) = \frac{e^{y_C}}{\sum_j^{j \neq C} e^{y_j}} \tag{3.18}$$

The probability of observing class C given y_C is calculated by exponentiating y_C and dividing it by the sum over all exponentiated scores.

3.3.3 Training the Network

Training a network requires an *objective function* or *loss function* to measure the error between the calculated output and the actual output. Learning is then performed in the form of changing the weights of each node so that the error or loss is minimized.

This is based on the principle of *gradient decent* and applied by the *backpropagation* algorithm [AH17]. The idea is to compute the gradient of the loss function at each node and change the weights in the direction of the gradient to minimize the loss. Backpropagation is used for efficiently computing these gradients. First the output is calculated by a combination of all gradients of the previous nodes. Then the new weights are calculated in one single backward pass, using the already calculated gradients [AH17].

For efficiently training a neural network the following parameters need to be set accordingly:

Batch size: The batch size describes distinct junks of data, which are passed to the inputs of a network at once. Here it is set to 256, meaning 256 samples per batch. This results in an input vector dimension of $(256, 42, 42, 3)$ for each batch. A igger batch size of 512 samples is also tested but gives worse results on the validation test dataset.

Loss function: Different loss functions are found in literature [AH17]. The *binary cross entropy* is used as loss function in this work (Equation (3.19)).

$$\text{binary cross entropy}(p_i) = H(p_i) = -(y_i \log(p_i) + (1 - y_i) \log(1 - p_i)) \tag{3.19}$$

With $y_i \in \{0, 1\}$ being a binary indicator for the class C_i being the correct label and p_i denoting the predicted probability of the observation being of class C_i. It

is calculated on the outputs of a probability model with values normalized to 1: $0 \leq p_i \leq 1$, which result from the softmax function. The cross-entropy loss increases with the predicted probability for observing a label diverging from the actual label.

Optimizer and learning rate: For efficiently calculating the loss the*RMSprop* optimizer is used with a learning rate of 0.0001 and a decay of $1e^{-6}$. The RMSprop optimizer is a mini-batch optimizer developed by Geoff Hinton and used as one of the default optimizers in Keras [Hin18].

Training epochs: Training is performed in a maximum of 15 epochs. One *epoch* is defined by one pass of the full training dataset to the network. If the total number of epochs is consumed, the network has seen the complete dataset 15 times. In many cases the maximum number of iterations is not reached, because a *stopping criterion* is met. The stopping criterion used here is the gradient of the loss function becoming positive. So training is stopped, if the loss does no longer minimize.

To build the ConvNet of Figure 3.7, the library *Keras* is used [C+15]. It is a high-level neural networks API, built on top of the powerful deep learning library TensorFlow [AAB+18]. The model used here is a simple `Sequential` model, with the parameterized layers added one by one. Each layer consumes one line of code, so programming a neural network becomes highly efficient.

3.3.4 Network Architecture

All the previously presented types of layers are used by the convolutional neural network architecture presented in this work. It follows the main ideas proposed by LeCun et al. and their LeNet-5 [LBBH98] and Krizhevsky et al. [KSH12]. This is to reduce the spatial information by a series of convolution and pooling layers to finally result in a fully connected network with a softmax output layer. The architecture is illustrated in Figure 3.7.

The basis for the network is given by three convolution cycles followed by the fully connected network. Each convolution cycle (*Conv Cycle* in Figure 3.7) is composed of: (1) Two convolution layers (*Conv* (3×3)), which are each followed by ReLU activations and (2) A max-pooling layer followed by a dropout layer. Each convolution layer performs 3×3 convolutions and is parameterized with a number denoting the number of feature maps to generate determining the output dimensions (Section 3.3.2) After three of these cycles, the data is flattened and passed to a fully connected network. This is composed of an input layer with 1152 nodes, followed by a hidden layer with 512 nodes, followed by a ReLU activation and dropout layer to finally arrive at a binary softmax output.

This rather simple network, already has a total number of 878,370 learnable parameters, when working with image patches of size $42 \times 42 \times 3$px as inputs. The argument for choosing this structure, as opposed to more recent and more complex methods is, simplicity. A simpler model is easier to train and consumes less training data, because of

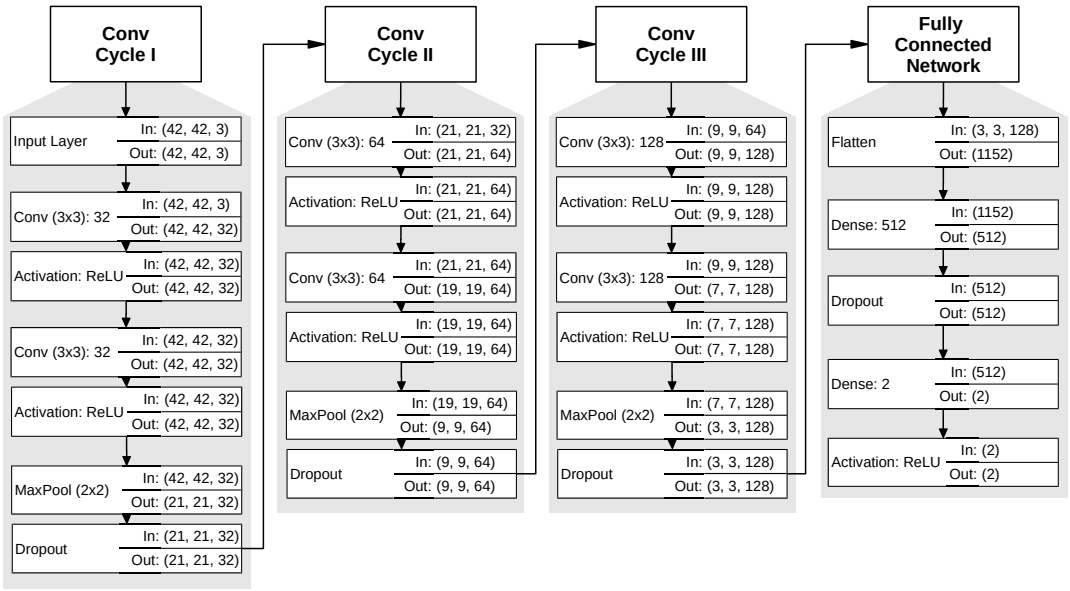


Figure 3.7: Visualization of the ConvNet model as a directed acyclic graph. It is grouped into three *Conv Cycles* and a *Fully Connected Network*. Each layer of the network is illustrated with a title and the dimensions of its inputs and outputs.

less weights being trained. If the classification works on a simple network, the prove of concept is provided. Further work can build up on this solution testing more complex and "deeper" models.

CHAPTER 4

Evaluation and Discussion

Goal of the evaluation process is to determine the performances of each individual detection approach, as well as finding out, which outperforms the other. This is done by answering the following questions:

- How well does the patch classification work?
- How well are complete bees classified?

To answer the second question the first question needs to be dealt with first, because the complete bee results are inferred from the patch classification results. The patch classification performance is of valuable information, because it provides detailed information about the classification process and can be used for parameter tuning.

The classification performance for the complete bee images is calculated from the patch prediction results. All predicted labels that belong to the same original bee image are gathered and a label for the original bee is inferred from the individual patch predictions. For the original bee image to be predicted positive, at least one of the sub-image patches must be classified as positive. Otherwise the prediction is negative.

4.1 Datasets for Evaluation

Applying the patch extraction using the proposed sliding-window function creates an imbalanced dataset with an imbalance of 285 : 1, negative to positive patches. This is why, for evaluating the patch classification performance, two types of datasets are used. These are created based on the idea that one can freely choose the number of negative and positive samples used for training and evaluation. Keeping all available positive patches and varying the amount of negative patches leads to the following two datasets:

1. **Sliding-window datasets**: For each bee image all patches are extracted using the windowing function. So for each positive image patch sample there exists 285 negative samples. While being unfavorable for hyper parameter tuning, this dataset is used to simulate the *in-action* performance. This simulation is possible because the same windowing function is used when the system is in action. So in later use the chances of observing an imbalance of negative to positive samples will be high which is why this type of dataset reflects a similar imbalance.

2. **Balanced dataset**: This is a subset of the sliding-window dataset, where each data sample is represented by one sub-window-patch only. If a bee image has the label "mite"(positive sample), the representing sub-window patch is being extracted from the position of the Varroa mite marked by the user in the ground truth. If the label is "no-mite" (negative sample), the representing patch is chosen randomly and also assigned the label "no-mite". This simulates a balanced dataset, which is used to fine-tune parameters and calculate performance measures like accuracy or confusion matrices.

Both types of datasets are applied to the existing test, as well as, a validation dataset. This results in a sliding-window and a balanced version of both datasets, which are used during the evaluation process.

The complete bee performance is computed based on a sliding-window dataset. This is because, the results of all predicted patches of the original complete bee image must be taken into account, when deciding for the predicted label of the complete bee. By using the sliding-window all patches belonging to a complete bee image are represented.

For evaluating the individual performances of each model, different pipeline configurations and sets of parameters are tested. This process is also referred to as *hyper parameter search*, where hyper parameters denote all configurations that can not be set automatically and need manual pre-selection. For hyper parameter search solely the balanced and sliding-window representation of the *validation* dataset are used, to avoid adding a bias to the fine-tuning process.

4.2 Metrics for Evaluation

For performance evaluation different measurements are used. For comparing different models, both Precision-Recall-Curves (PRC) as well as Receiver-Operating-Characteristic-plots (ROC) are used. The PRC plots precision over recall, which are computed for different threshold values to separate the predicted classes. In ROC the true-positive-rate is plotted over the false-positive-rate. For both curves, scalar representations can be computed. The area-under-the-curve (AUC) is one of theses scalar representations and is applied for the PRC ($PR\text{-}AUC$) curve as well as the ROC curve ($ROC\text{-}AUC$). The area is summarized from trapezoids fitted under the curves, applying the trapezoidal rule. For the PRC, additionally the average precision (AP) is computed, which can be interpreted

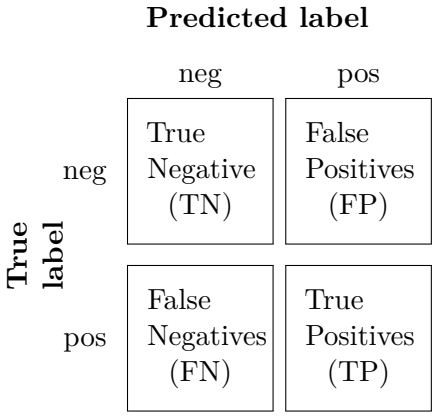


Figure 4.1: Binary confusion matrix explained

as the weighted mean of precisions achieved at each threshold. *Binary confusion matrices* are used to better visualize the per class-accuracy and give insights into false-positive and false-negative predictions. The schema used in this work is shown in Figure 4.1 and differs slightly from typical schemas, with the true-positives shown in the bottom right square. The confusion matrix gives detailed insight into the classification result and is used as basis for other scores like the f-score or precision and recall.

For illustrating the overall performance the harmonic mean between precision and recall is used (Equation (4.1)).

$$f1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{4.1}$$

This harmonic mean is called *f1-score* and is defined on the interval $[0, 1]$ with 1 as best score.

4.3 Evaluation Deep Learning Approach

Applying the patch extraction using the sliding-window function on the manually labeled dataset with imbalance $1 : 3$ creates a new dataset with imbalance $1 : 285$ positives to negatives. This dataset does not provide satisfactory results for training. This is because the imbalance has a significant impact on the classification performance and balanced datasets are to be preferred [SMD13], [HG09]. In this work a combination of the following two approaches is used to achieve a balanced dataset for training: (1) Under-sampling the majority class, in this case the negative class. (2) Over-sampling the minority class, the positive class.

Under-sampling is realized by randomly choosing a subset of negative data patches from the overall pool of negative patches after extraction. Therefore a parameter n_{neg}

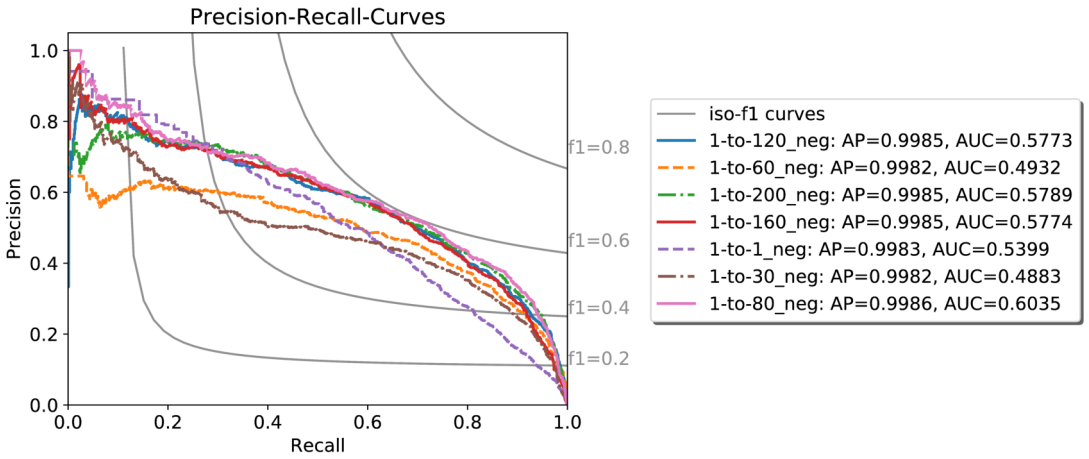


Figure 4.2: Evaluating different dataset for training the ConvNet, evaluated on the sliding-window validation dataset. The number in the title of each model indicates the number of patches extracted from the original training dataset.

is introduced, that represents the number of randomly chosen image patches that are returned from each negative sample. For instance, setting $n_{neg} = 80$, returns 80 negative patches from a complete bee image. The training dataset has: class_0: 6,219, class_1: 1,809 complete bee images. When applying patch extraction with $n_{neg} = 80$, the resulting training dataset has the form class_0: 497,520, class_1: 1,809, which is a subset of the total available patches of class_0: 951,507, class_1: 1,809. The minority class remains unchanged at this stage, extracting all available positive data patches.

To create a balanced dataset,the remaining $1,809$ positive patches are over-sampled, using data augmentation. For $n_{neg} = 80$, a number of 275 augmented images must be created from each positive image patch, to result in an evenly balanced dataset of around $500,000$ samples per class.

The results of the process of determining the optimal n_{neg} is displayed in Figure 4.2. The plot shows the comparison of precision-recall curves, resulting from models trained with datasets extracted with different n_{neg}. They are evaluated on the same sliding-window validation dataset to better examine the results. For numerical comparison, the average-precision (AP) as well as the area-under-the-curve (AUC) are provided. Best results can be achieved when extracting patches with $n_{neg} = 80$ with a maximum $PR\text{-}AUC = 0.6035$. But closely followed by 1-to-120_neg, 1-to-160_neg and 1-to-200_neg, with about 2% less performance. The model, which is trained with 1-to-1_neg performs better then the model trained with 1-to-60_neg. This indicates that the number of negative samples is of less impact than the number of positive samples and that the heavy data-augmentation for the minority class does not provide sufficient variation.

The best performing deep learning models are further referred to as ConvNet_80_neg and ConvNet_160_neg trained on the 1-to-80_neg and 1-to-160_neg dataset respectively.

4.4 Evaluation “Traditional” Approach

Both approaches “traditional” and deep learning use parameterization which influence the classification results. These parameters are *hyperparameters*. Setting these follows a logical approach, where possible values are defined in literature or by testing via a trial-and-error.

For the “traditional” approach the hyperparameters are represented by the configurations of the individual features and the combinations features. In cases where parameters can not be determined logically, *grid search* over possible values is performed. Each parameter configuration is tested on the previously mentioned *balanced validation dataset* with the goal of finding the best configurations.

The order in which the features are extracted is irrelevant to the final outcome, which is why no special regard is taken during parameter search. The functionality for combining different features is provided by the python library *scikit-learn*[1]. It contains the `Pipeline` facility, which allows the combination of data processing steps, feature extraction steps and final classification, into one pipeline. This library facilitates hyper parameter search. Features can be grouped into a so called `feature_union` which, in combination with a classifier, form a `Pipeline`. The classifier within the pipeline is trained, by a single call to the pipelines `fit()` method, simplifying the process of hyper parameter tuning.

4.4.1 Optimal Feature Selection

The grid search method mentioned above is used to find optimal parameter configurations and feature combinations. All parameters are computed on the balanced validation dataset. This dataset is used in favor of the sliding-window dataset, solely because it is smaller and therefore offers better performance. After grid search a list of relevant feature candidates is extracted, which are further tested on the sliding-window validation dataset.

The list of available features is listed in Table 3.1. Each feature allows for different configurations which are tested with two classifiers: (1) Random Forrest and (2) Naive Bayes. In total, 258 different feature combinations, 129 per classifier are tested. Table 4.1 shows a filtered version of these original grid search results. The top part shows the results for the individual features in all configurations tested using both classifiers. The lower part contains all feature combinations with an area-under-the-curve of the Precision-Recall-Plot ($PR\text{-}AUC$) greater than 0.8 to filter for the best performing combinations. For all results, additionally the AUC of the Receiver-Operating-Characteristics-Plot ($ROC\text{-}AUC$) and the average precision (AP) are listed. Based on the best results, highlighted bold, seven candidate pipelines are extracted, that qualify for further testing. So each highlighted result is represented by a candidate pipeline, summarized in Table 4.2

[1] `http://scikit-learn.org/stable/` (last accessed 04/2018)

feature combination	Naive Bayes			Random Forrest		
	roc-auc	pr-auc	ap	roc-auc	pr-auc	ap
color hist 32 bins	0.72	0.58	0.7	0.71	0.56	0.68
color hist 8 bins	0.72	0.57	0.7	0.72	0.57	0.68
color hist 64 bins	0.71	0.56	0.68	0.7	0.55	0.67
color mean	0.84	0.75	0.82	0.91	**0.86**	0.88
color std	0.75	0.61	0.72	0.67	0.54	0.63
color skew	0.85	0.77	0.82	0.84	0.74	0.8
texture SIFT grid=15	0.63	0.55	0.62	0.73	0.63	0.69
color hist 8 bins, color mean	0.8	0.67	0.77	0.89	**0.83**	0.86
color mean, color std	0.87	**0.8**	0.86	0.89	**0.82**	0.86
color mean, color std, texture SIFT grid=15	0.86	0.77	0.82	0.86	**0.82**	0.83
color skew, texture SIFT grid=15	0.85	**0.81**	0.82	0.83	0.74	0.79
color hist 32 bins, color mean, color skew	0.86	**0.8**	0.83	0.88	0.8	0.84
max	0.87	0.81	0.86	0.91	0.86	0.88

Table 4.1: Selection from the total feature grid search results for a maximum of three feature combinations computed on the balanced validation dataset. The top part show the individual features and their results. The bottom part shows the best performing combinations.

Pipeline label	Features	Classifier
Mean_RF	color mean	Random Forest
Hist8Mean_RF	color histogram with 8 bins, color mean	Random Forest
MeanStd_RF	color mean, color standard deviation	Random Forest
MeanStdSIFT_RF	color mean, color standard deviation, texture SIFT	Random Forest
MeanStd_Bayes	color mean, color standard deviation	Naive Bayes
Hist32MeanSkew_Bayes	color histogram with 32 bins, color mean, color skew	Naive Bayes
SkewSIFT_Bayes	color skewness, texture SIFT	Naive Bayes

Table 4.2: The pipelines resulting from grid search, with a $PR\text{-}AUC > 0.8$ on the validation dataset

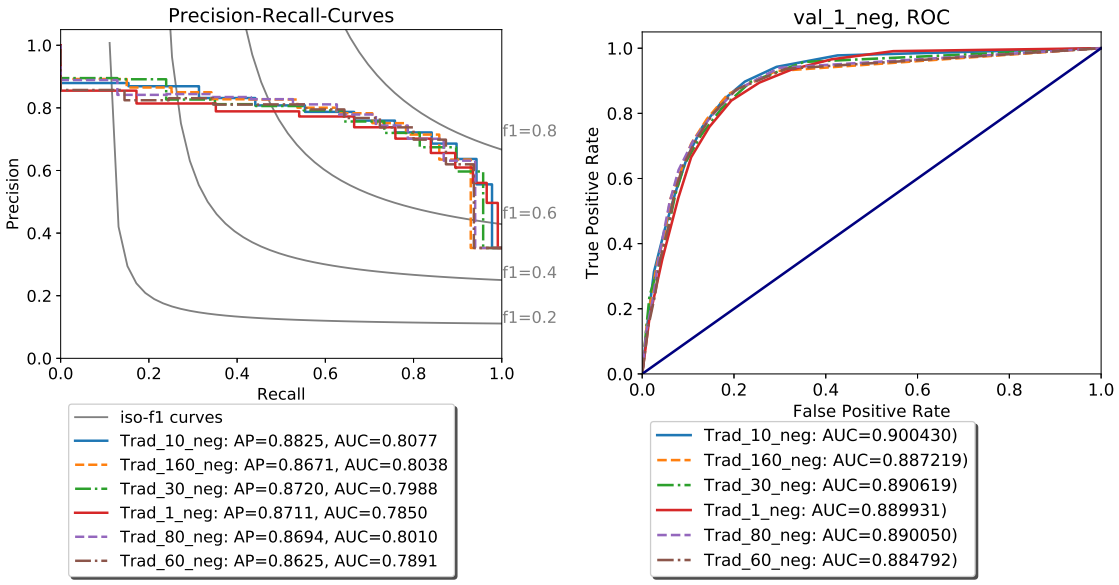
The overall best result with a $PR\text{-}AUC$ of 0.86 computed on the balanced validation dataset is achieved by the Mean_RF pipeline. This is a very simple pipeline with color mean as single feature followed by a random forest classifier. Nevertheless, all candidate pipelines of Table 4.2 are further tested, to ensure independence from the validation dataset.

4.4.2 Optimal Number of Training Samples

The dataset used for training changes depending on the amount of sub-window patches extracted from each original sample. Goal of this evaluation is to find the optimal amount of training-patches for training the “traditional” pipeline.

The MeanStd_RF pipeline is used for the comparison of the different configurations of the datasets. It is trained using different training dataset configurations, but validated on the same dataset.

Figure 4.3 shows both a PRC-plot and a ROC-plot, acquired after training MeanStd_RF with different training datasets. The datasets are titled: 1_neg, 10_neg, 30_neg, 60_neg, 80_neg and 160_neg. The number in the title of each individual dataset indicate the number of random 42×42px sub-window-patches, which are being extracted from each *negative* 160×280px complete bee image. Further all available *positive* sub-window-patches, showing Varroa mites, are extracted and data augmentation is applied to match the number of negative patches. This results in different sizes for the dataset. For instance, the training dataset 1_neg consists of a total number of 12,438 extracted



(a) PRC **(b)** ROC

Figure 4.3: Results of training the same "traditional" pipeline MeanStd_RF with different Trad_i_neg training datasets. i represents the number of negative patches extracted using a sliding-window function.

patches and 160_neg of 1,990,348 samples, so a broad range of inputs is covered.

Comparing the individual performances using the *AUC* values in Figure 4.3 shows a maximum difference between the *PR-AUC*s of less than 1% and 1.5% for the *ROC-AUC*s. This leads to the conclusion that the amount of training patches is of little impact to the classification result.

4.4.3 Final Pipeline Selection

In the previous steps, the candidate models are narrowed from 258 to 7 best performing "traditional" pipelines. Further the training dataset is set to 80_neg with a total of 1,258,260 patches for training.

The seven pipelines are further evaluated to select the best performing models from this group, which is compared with the best performing deep learning models. Therefore the sliding-window validation dataset is used which has an imbalance towards the negative label. This is similar to the imbalance of the test dataset. Each model is trained using the 80_neg training dataset and evaluated on the sliding-window validation dataset. The result of this evaluation is a comparative ROC plot displayed in Figure 4.4.

Again slight differences are noticeable between the performances of the different models. Closely examining the curves, a group of three pipelines of marginally better performance

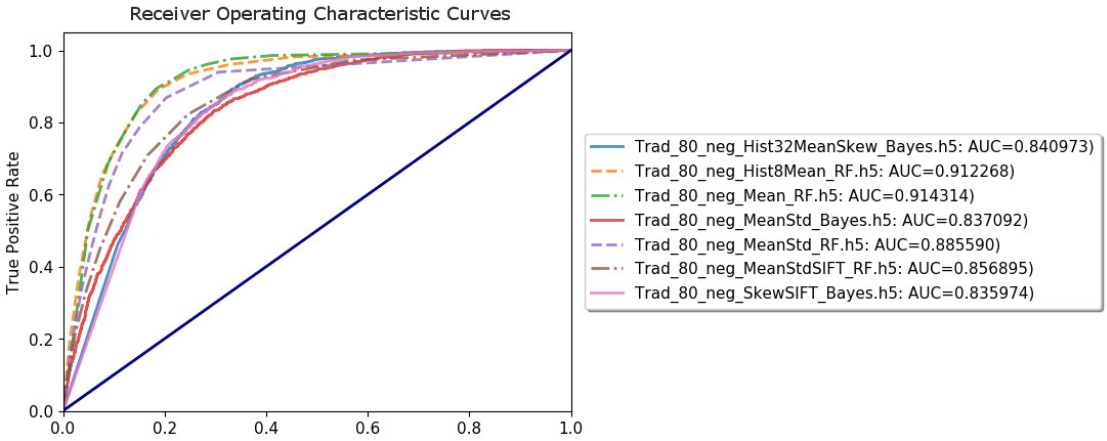


Figure 4.4: Results of the seven best performing "traditional" pipelines computed on the sliding-window validation dataset.

can be identified. These are Mean_RF leading with $AUC = 0.9143$, followed by Hist8Mean_RF ($AUC = 0.9122$) and MeanStd_RF ($AUC = 0.8855$).

4.5 Classification Performance Evaluation

All previous evaluations are performed on the validation dataset in different manifestations. To evaluate the actual classification performance, the best models of both approaches are applied on the *test dataset*. The models are listed in Table 4.3 and consist of three "traditional" and two deep learning models.

For a complete comparison, the patch classification performance is evaluated using the above mentioned *balanced test dataset* and the *sliding-window test dataset*. Then the *complete bee performance* is inferred from the patch classification results, to determine the in-action performance.

4.5.1 Results Patch Classification

The patch classification performance is identified using a similar approach as is used for the evaluation of the hyper parameters for the individual models. With the difference of testing the models on the *test dataset* instead of the validation dataset.

Balanced Test Dataset Performance

First, the results of applying the models on the balanced test dataset are summarized in Figure 4.5. Here the individual performances can be compared using a PR-plot and a ROC-plot. Additionally the confusion matrices for both the best performing "traditional", as well as, deep learning approach are presented.

Model label	Description
Trad_80_neg_Mean_RF	"Traditional" pipeline: Color mean with Random Forest, trained on dataset 80_neg
Trad_80_neg_Hist8Mean_RF	"Traditional" pipeline: Color histogram with 8 bins, color mean with Random Forest, trained on dataset 80_neg
Trad_80_neg_MeanStd_RF	"Traditional" pipeline: Color mean, color standard deviation with Random Forest, trained on dataset 80_neg
ConvNet_160	Convolutional Neural Network, trained on dataset 160_neg
ConvNet_80	Convolutional Neural Network, trained on dataset 80_neg

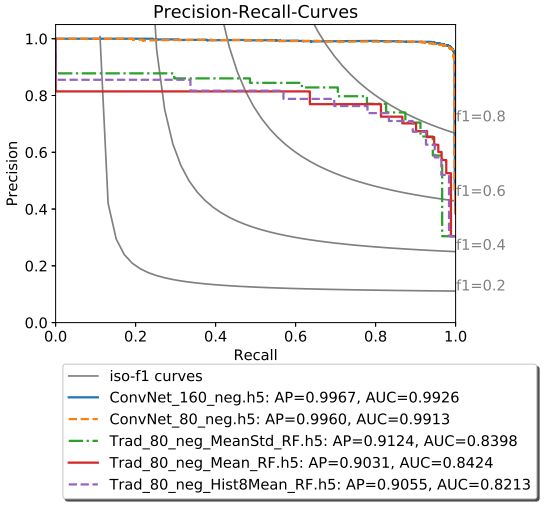
Table 4.3: The list of the final best performing models including a short description.

Observing the results depicted in Figure 4.5a, the PR-curves allow a clear separator into two groups: (1) Curves belonging to "traditional" models and (2) Curves belonging to deep learning models. Within the groups similar performances are observed with a clear separation between them. The same behavior can be observed in the corresponding ROC plot in Figure 4.5b. Here the two groupings become even more evident. Based on the *PR-AUC*, the best performing models for each group are: ConvNet_160_neg with $PR\text{-}AUC = 0.9926$ and Trad_80_neg_Mean_RF with $PR\text{-}AUC = 0.8424$. The best f-scores per group are achieved by ConvNet_80_neg and Trad_80_neg_MeanStd_RF. For these two models, confusion matrices are presented with Figure 4.5c showing the confusion matrix for the best performing "traditional" model and Figure 4.5d, with the best performing deep learning model.

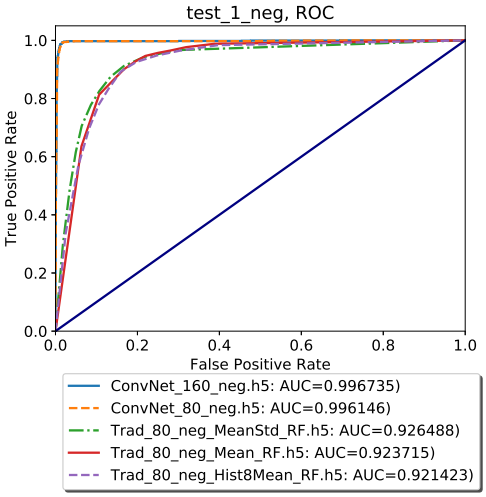
Both the PR-plot as well as the ROC-plot indicate a superior performance of the deep learning models, independent of the individual configuration. Closely observing the confusion matrices reveals that the accuracy for the negative class is about 20% higher with the ConvNet_160_neg model (Figure 4.5d), in comparison to the best performing "traditional" model (Figure 4.5c). Also the f-score = 0.77 of the best performing "traditional" model is around 20% lower than the f-score = 0.97 for the deep learning model.

Sliding-Window Test Dataset Performance

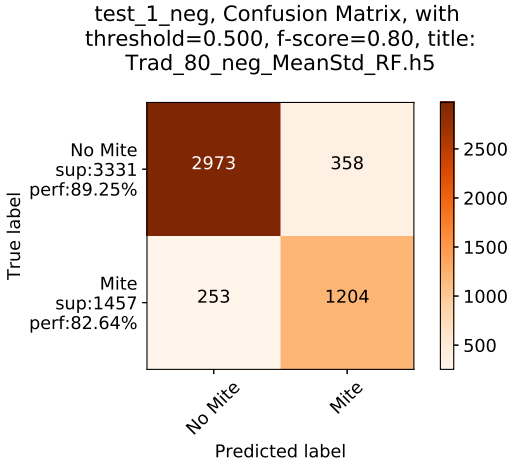
The gap between the two groups becomes more evident, when using the sliding-window test dataset with a class imbalance of $1 : 285$. So for each positive sample there exist 285 negative samples in the test dataset. The results for applying the models to this dataset are summarized in Figure 4.6. Both the performance of the "traditional" and



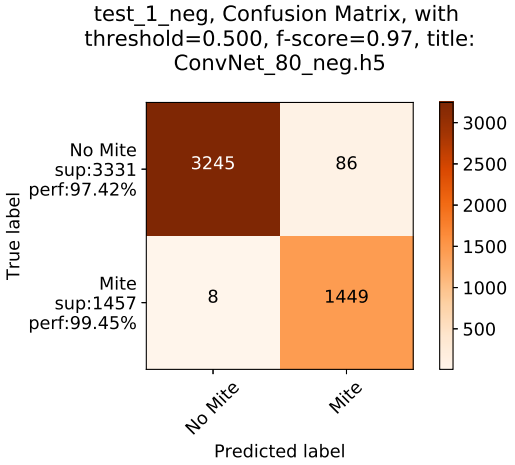
(a) PRC



(b) ROC



(c) Confusion Matrix of best "traditional" approach

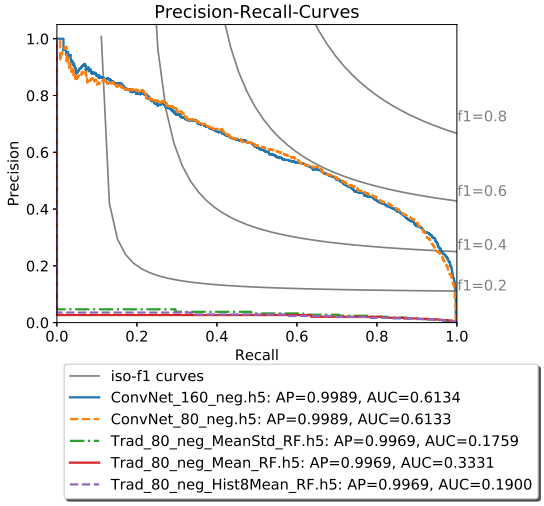


(d) Confusion Matrix of best deep learning approach

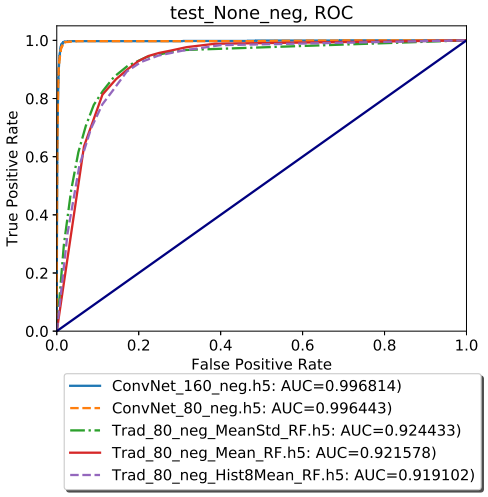
Figure 4.5: The overall patch classification performance of the balanced test dataset calculated by the best performing candidate models.

deep learning approaches are decreasing using this dataset.

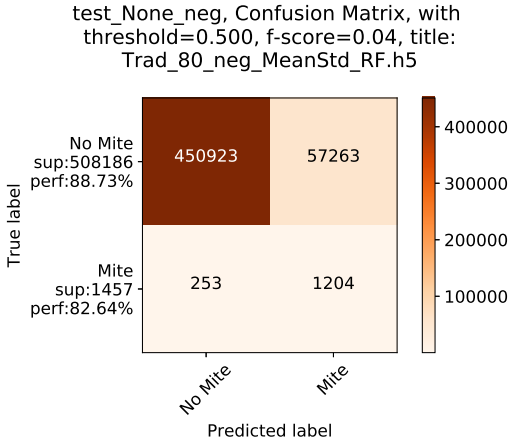
The PR-Plot in Figure 4.6a shows a similar performance of the ConvNet models of $AUC = 0.613$, which is a drop of about 38% compared to the best *PR-AUC* score of the balanced dataset in Figure 4.5a. Within the "traditional" models, again similar performances are observed, but showing a higher drop in their overall performance. The best model Mean_RF achieves a $PR\text{-}AUC = 0.1333$, making a total reduction of more than 60% compared to the balanced dataset with 0.8424.



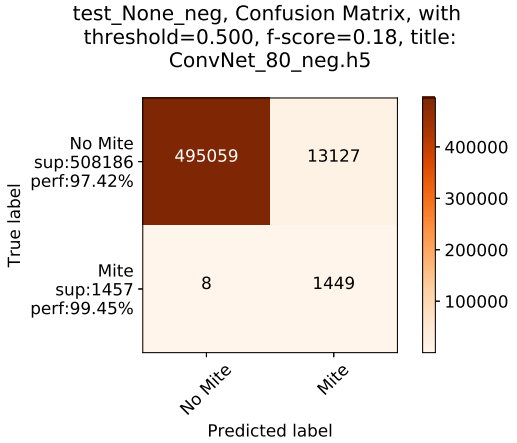
(a) PRC



(b) ROC



(c) Confusion Matrix of best "traditional" approach



(d) Confusion Matrix of best deep learning approach

Figure 4.6: The overall patch classification performance of the sliding-window test dataset calculated by the best performing candidate models.

This behavior is better explained after observing the confusion matrices for the best performing "traditional" (Figure 4.6c) and deep learning model (Figure 4.6d). Again the best f-scores are provided by the models Trad_80_neg_MeanStd_RF and ConvNet_80, which allows a direct comparison to the confusion matrices of Figure 4.5. Both models show a higher rate of false-positive predictions (top right square), where the actual label is negative but is predicted as positive, when compared to Figure 4.5. Which is why the f-scores of both models drops significantly. The f-score of the "traditional" model (Figure 4.6c) drops to a minimum of f-score = 0.04 (top description), which is explained by

the lower precision compared to the deep learning models. The ROC-Plot also indicates that the high rate of false-positives is responsible for the drop in performance, since the ratio of true-positives to false-negatives remains approximately unchanged. This is also shown by the per-class accuracies of the confusion matrices, which can be found on the left side of each plot, under the true label description. Similar per-class performances are reached.

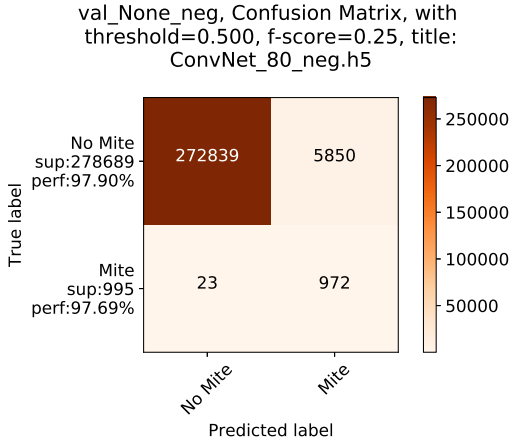
This shows that the per-class accuracy and ROC-plot are insufficient performance measures when working with imbalanced datasets. This is because both performance measures are sensitive to imbalances in the data. Looking at the test dataset, an accuracy above 90% could be achieved by predicting all samples as negative, solely because negative samples outnumber positive ones. In the ROC-plot, the true-positive-rate is plotted over the false-positive-rate. The false-positive-rate (FPR) is also sensitive to imbalances in the data, since it incorporates the total number of negative samples to compute: $FPR = \frac{\sum \text{False positives}}{\sum \text{Total negatives}}$. This is why performance measures based precision and recall are to be preferred when working with imbalanced data [SR15]. At the default threshold of 0.5 the best performing "traditional" model MeanStd_RF reaches a maximum f-score of 0.04 (Figure 4.6c) and the deep learning approach a maximum f-score of 0.18 (Figure 4.6d). This result can be improved by choosing a different threshold.

4.5.2 Choosing Optimal Thresholds

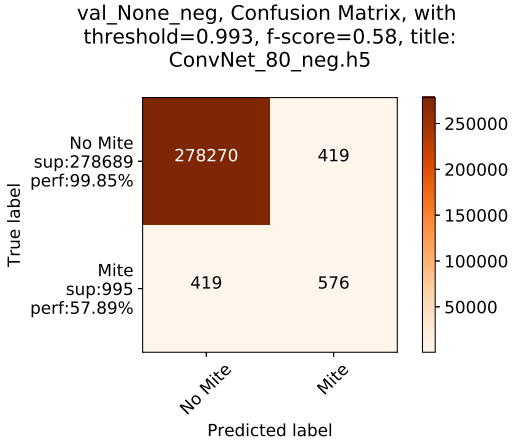
The output of each model can be interpreted as confidences for the predicted labels. In the binary case, two confidences result from a prediction, one for the positive and one for the negative class adding up to 1. Example predictions are: Class 0 = 0.7, class 1 = 0.3, or class 0 = 0.01, class 1 = 0.99. A natural separation of the two classes is provided by a threshold of 0.5. When interpreted as confidences a predicted confidence greater 0.5 is necessary to classify a patch as positive (class 1). This threshold can be varied to change the confidence of the predicted classes.

Two confusion matrices are given in Figure 4.7. Both are acquired after applying the same ConvNet_80_neg model on the sliding-window validation dataset. The only difference is the threshold used. In the confusion matrix on the left (Figure 4.7a), the threshold is set to default 0.5, separating the classes directly in the middle. Here a higher false-positive rate can be observed (top right square) with an overall f-score of 0.25 on the validation dataset. On the right hand plot (Figure 4.7b), the threshold is set to 0.993. With this new threshold, the corresponding f-score raises to 0.58. This lowers the false-positive error, but raises the false-negative error, so the classes get separated more evenly. At this new threshold the positive class is only assigned if the confidence of observing a positive sample is greater than 99.3%. This leaves the question for how to set this threshold in an automated fashion.

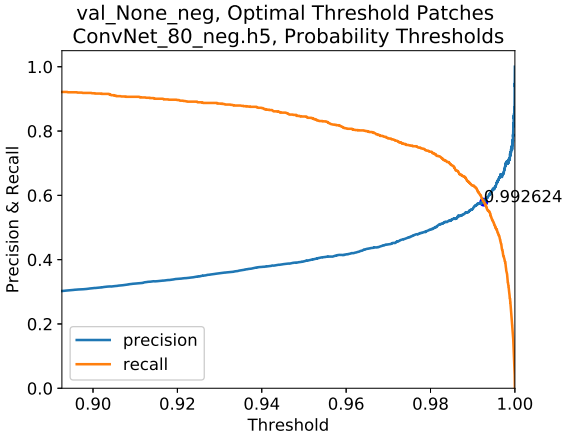
First, the optimal threshold depends on the model used, so it needs to be evaluated on a per model basis. Second, the threshold is calculated based on the validation dataset and not on the test dataset. In later application, this value is adjusted based on experience and



(a) Confusion Matrix @ threshold 0.5



(b) Confusion Matrix @ threshold 0.993



(c) Optimal Threshold for ConvNet_80 model

Figure 4.7: Finding the optimal threshold, where precision and recall are highest. (a) and (b) show confusion matrices computed using the same models, but different thresholds for assigning a predicted positive label. (c) shows the intersection point of precision and recall.

observations. The optimal patch classification threshold for the ConvNet_80_neg model is based on the separation point calculated in Figure 4.7c, where precision and recall reach a maximum value. The classification error in the confusion matrix in Figure 4.7b (top-right and bottom-left square) gets evenly spread across the classes.

The optimal threshold for each candidate model and the corresponding patch classification results, computed on the sliding-window test dataset, are displayed in the last column of Table 4.4. These thresholds are calculated using the same method as used for the ConvNet_80_neg model illustrated in Figure 4.7c. All candidate models use an optimal threshold above 0.95 again indicating the higher false-positive rate. The best

Model	Results @ 0.5	Results @ opt threshold
Trad_80_neg_Mean_RF	f-score = 0.02	@ 0.965: f-score = 0.05
Trad_80_neg_Hist8Mean_RF	f-score = 0.03	@ 0.960: f-score = 0.06
Trad_80_neg_MeanStd_RF	f-score = 0.04	@ 0.985: f-score = **0.08**
ConvNet_160_neg	f-score = 0.21	@ 0.990: f-score = 0.57
ConvNet_80_neg	f-score = 0.18	@ 0.993: f-score = **0.58**

Table 4.4: Patch classification performance calculated on the sliding-window test dataset. The results are given for the default threshold 0.5 and the respective optimal threshold, which is calculated based on the results on the validation dataset. Best Performing model is ConvNet_80_neg with a maximum f-score = 0.58.

Model	Results @ 0.5	Results @ opt threshold
Trad_80_neg_Mean_RF	f-score = 0.33	@ 0.800: f-score = 0.33
Trad_80_neg_Hist8Mean_RF	f-score = 0.33	@ 0.960: f-score = 0.36
Trad_80_neg_MeanStd_RF	f-score = 0.33	@ 0.985: f-score = **0.37**
ConvNet_160_neg	f-score = 0.52	@ 0.985: f-score = **0.82**
ConvNet_80_neg	f-score = 0.47	@ 0.985: f-score = **0.82**

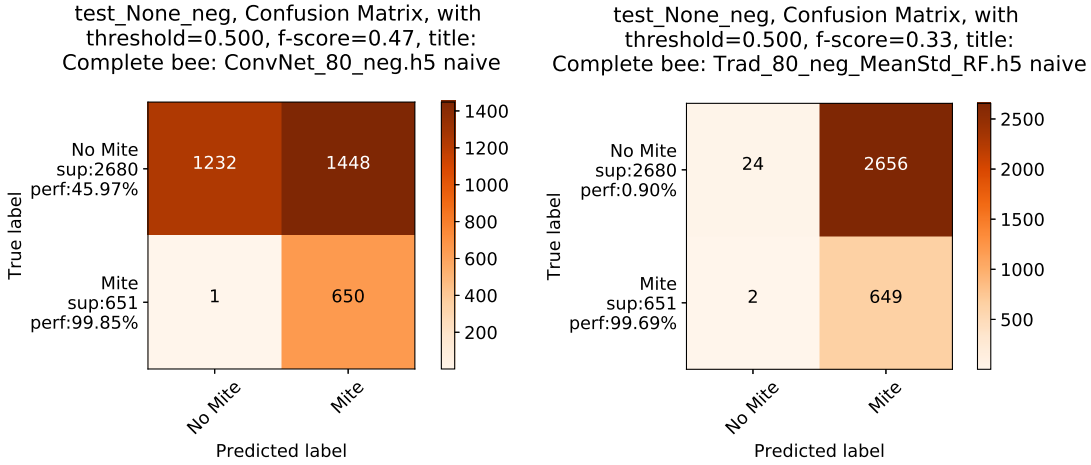
Table 4.5: The final classification results for complete bee images calculated on the sliding-window test dataset. The overall best performing model is ConvNet_80_neg with results highlighted in gray. All results are inferred from the patch classification results of the sliding-window test dataset.

patch classification result is achieved by the ConvNet_80_neg with a f-score = 0.58 at threshold 0.993.

This must however not be the optimal threshold for the complete bee classification, because the patch results are grouped for each complete bee image. A positive predicted class label of the complete bee image depends on the positive prediction of one single patch. So the same label is assigned to a complete bee even if more than one patch show a positive prediction and these predictions can be false-positives as well. When using the proposed threshold of 0.993 for complete bee validation on the validation dataset a f-score of 0.85 is achieved. Additionally a range of thresholds around this patch threshold are tested and a higher f-score of 0.88 are achieved with a threshold of 0.985. This threshold is chosen as the optimal threshold for the final bee classification. The list of optimal complete bee thresholds are illustrated in the last column of Table 4.5

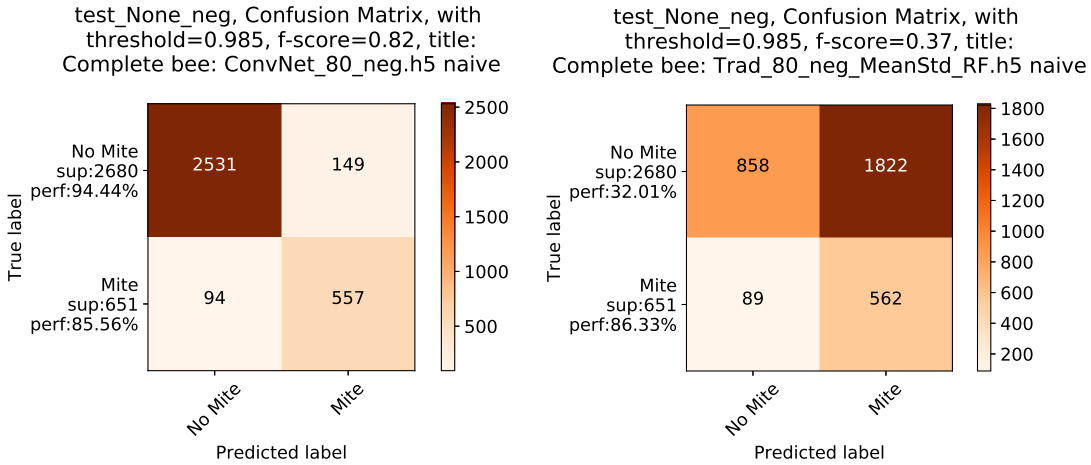
4.5.3 Results Complete Bee Classification

With the patch classification performance being necessary for parameter tuning and providing a detailed analysis of the classification process, it does not answer the question



(a) f-score = 0.47 **(b)** f-score = 0.33

Figure 4.8: Complete bee classification results of the best performing models at threshold 0.5, calculated on the test dataset. (a) Result of the best deep learning model ConvNet_80_neg (b) Result of the best "traditional" model MeanStd_RF.



(a) f-score = 0.82 **(b)** f-score = 0.37

Figure 4.9: Complete bee classification results of the best performing models at threshold 0.985, calculated on the test dataset. (a) Result of the best deep learning model ConvNet_80_neg, with the overall best result highlighted in bold. (b) Result of the best "traditional" model MeanStd_RF.

of how well the actual bee classification works. This is the goal of this final evaluation section. So given a new bee image composed of 153 unseen image patches, how well does the system classify the complete bee as infected or not. This evaluation is performed on the test dataset and not on a live-data stream, which makes it a simulation of the

in-action performance. The results are still representative for the in-action performance, because the same approach for extracting the image patches using a sliding-window is used, as for live-data. The prospect of working with the test dataset is, that it allows a detailed evaluation. Measures like f-score or per-class-accuracy can be calculated directly, enabling exact evaluation of the classification results.

The classification result for a complete bee image is inferred from the classification results of all sub-window patches extracted from this image. So the first step to the complete bee prediction is the predictions of all sub-window patches belonging to the same bee image. A bee image is assigned a positive label, if one or more sub-window patches are predicted as positive. Therefore a negative label is only assigned, if all sub-window patches are predicted as negative. This is due to the fact that a Varroa mite can be fully represented in one single image patch. Therefore, one patch alone holds enough information to classify the complete bee as positive.

The individual patch classification results depend on the threshold for the confidence of the classifier for predicting the positive class. This is why the patches are classified with two different thresholds, before summarizing the results: (1) The default threshold of 0.5 and (2) the optimal complete bee threshold for each model calculated on the validation dataset. With the default threshold (Figure 4.8a) 63.03% of the negative bee samples are miss-classified as positive samples using the deep learning approach. The classification performance is increased using the optimal threshold (Figure 4.9a). The complete bee classification results of all candidate pipelines are presented in Table 4.5. Each model is described with two f-scores, one for the 0.5 threshold and one for the optimal threshold. The highest scores and best performing models are highlighted in bold. These are Trad_MeanStd_RF with a f-score of 0.37 and ConvNet_80_neg and ConvNet_160_neg, with the latter two performing equally well, achieving a f-score of 0.82.

A final comparison of the best performing "traditional" and deep learning model is given in the form of confusion matrices in the Figures 4.8 and 4.9. Figure 4.8 illustrates the complete performance for both models at threshold 0.5 and Figure 4.9 the performance at threshold 0.985. At the optimal threshold of 0.985 the ConvNet_80_neg model shows a per-class accuracy of 94.4% for the negative and 85.5% for the positive class. This results in an overall miss-classification rate of less than 8%[2]. The best performing "traditional" model is Trad_80_neg_MeanStd_RF, with a per-class accuracy of 32% for the negative and 86% for the positive class, with a total 57% miss-classification.

4.6 Discussion of Classification Results

The results of the patch classification are summarized in Table 4.4. These show a significant performance differences between the two groups: "Traditional" and deep learning models with an f-score difference of 0.45. This is in contrast to the performance of the configurations within the same groups, which are little with a f-score variability of

[2] $\frac{fp+fn}{\text{total number of samples}} = \frac{149+94}{2680+651} = 0.072951$

less than ± 0.02 . Both groups show a high false-positive rate, explained by the strong imbalance of $1 : 285$ in the test data. The maximum patch classification performance is 0.58 using the ConvNet_80_neg model. This is reached after adjusting the confidence threshold from 0.5 to 0.993. Reasons for this average performance are: (1) The limited amount of training data, which does not provide sufficient variability in the data. (2) The strong imbalance in the test dataset, raising the chances for a miss-classification.

Similar to the results of the patch classification, a high number of false-positives is observed in the complete bee results with 64.03% false-positive classifications. Adjusting the confidence threshold is used to balance this error. It is set to the optimal threshold of 0.985 computed on the validation dataset. So to classify an image patch belonging to a complete bee image as positive, the model has to be 98.5% sure that it deals with a positive sample to output a positive prediction.

The ConvNet_80_neg is the overall best performing model. It achieves best performance both at patch-classification (Table 4.4), as well as complete bee classification (Table 4.5). Less resources are consumed for training this model compared to the ConvNet_160_neg, which shows similar performances.

With this model a maximum complete bee classification performance with 0.85 (f-score) is reached. This is a gain of 0.27 compared to the patch classification performance using this model. This is explained by the miss-classifications at patch level having less effect in the context of complete bee classification, where the individual results are grouped to form the final result. Both the false-positive and false-negative errors decrease when choosing the confidence threshold of 0.985. This indicates that before the threshold adjustment the classifier is more confident in correctly predicting negative complete bee images than correctly classifying positive images as positive. And that miss-classifications, leading to a higher false-positive rate, occurs with patches taken from positive samples. This is again explained by the imbalance in the test dataset.

CHAPTER 5

Conclusion and Future Work

This works presents a comparison of state of the art image classification methods as well as different segmentation approaches to detect and classify honeybees at the beehive entrance. It is shown that the classification of healthy and infected bees works with a f-score of 0.82 using optimal hyper parameters calculated on the validation dataset. First, three video recording systems were designed that enable recording of honeybees at the hive entrance. The designs were developed in an evolutionary process and resulted in three novel recording prototypes. The prototypes were applied both in field recordings as well as in the laboratory creating 7.01 TB of processable video data that represents $1,584.5$ hours of recording. Four different camera sensors were tested with the Axis M1125 as the best performing sensor. From the recordings with this sensor a labeled dataset with $13,464$ images of infected and healthy honeybees was created. This included the implementation of a bee detection and extraction software and the manual marking of the positions of Varroa mites in each extracted image using a custom designed labeling tool.

This dataset was used to train and evaluate two types of classification pipelines: A "traditional" machine learning, as well as, a deep learning pipeline. Each pipeline was tested in different configurations using the validation dataset, to find the respective optimal parameterizations. The best performing models of each pipeline were compared using the f-score metric and confusion matrices. The best performance is achieved using a convolutional neural network with a miss-classification rate of 7.2%. This model is capable of distinguishing between healthy and infected bees with a f-score of 0.82. The best performing "traditional" model achieves a minimum miss-classification rate of 57% and a maximum f-score of 0.37. This shows the superiority of the deep learning approach compared to the "traditional" approach. This work therefore provides a proof of concept depicting both the hardware as well as the software implementation.

The following future work is necessary before the presented system can be set in action:

(1) Touching bees leading to merged foreground masks need to be separated to enable a full tracking solution of individual bees. This is necessary to feed the classification pipeline with well prepared image patches of bees.

(2) Applying different deep learning techniques. For future development of the detection pipeline other deep learning models can be tested. They can be evaluated and compared using the created dataset. Following the suggestions from [Gol17], unsupervised pre-training or transfer learning could lead to better results. Here the idea is to use a ConvNet that is pre-trained on a large image dataset and fine-tune the top layers with the available dataset. The advantage of this method is that a small dataset can be used for the fine tuning step since the network already learned how to extract image features due to the pre-training. This methods can be applied without putting additional effort into data labeling.

Whether this system is capable of fully replacing manual sampling requires a long term field study, which is beyond the scope of this work. Nevertheless, the presented system shows the capabilities of detecting the parasites *Varroa destructor* and provides a potential hardware setup in the form of a functioning prototype. This system can further be used for other recognition tasks like identifying bees carrying pollen or estimating the in-and-out-activity using the data from the tracker. These results can be used to give further insight into the foraging activity and general health of bee colonies.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AAB+18] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. `tensorflow.org`, 2018. (last accessed: 02.06.2018).

[AH17] Hamed Habibi Aghdam and Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Springer Publishing Company, Incorporated, 1st edition, 2017.

[AK11] Fatemeh Alamdar and MohammadReza Keyvanpour. A new color feature extraction method based on quadhistogram. *Procedia Environmental Sciences*, 10:777–783, 2011.

[AS16] Plamen Angelov and Alessandro Sperduti. Challenges in deep learning. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 485–495, 2016.

[ASSW01] Tom Arbuckle, Stefan Schröder, Volker Steinhage, and Dieter Wittmann. Biodiversity informatics in action: identification and monitoring of bee species using abis. In *Proceedings of the 15th International Symposium Informatics for Environmental Protection*, 2001.

[ATK+15] Ryfial Azhar, Desmin Tuwohingide, Dasrit Kamudi, Sarimuddin, and Nanik Suciati. Batik image classification using sift feature extraction, bag of features and support vector machine. *Procedia Computer Science*, 72:24 – 30, 2015. The Third Information Systems International Conference 2015.

[BB16a] Wilhelm Burger and Mark J. Burge. *Regions in Binary Images*, chapter 10, pages 209–219. Springer London, London, 2016.

[BB16b] Wilhelm Burger and Mark J. Burge. *Scale-Invariant Feature Transform (SIFT)*, pages 609–664. Springer London, London, 2016.

[BG08] Otto Boecking and E. Genersch. Varroosis – the ongoing crisis in bee keeping. *Journal für Verbraucherschutz und Lebensmittelsicherheit*, 3(2):221–228, May 2008.

[BPRM16] Z. Babic, R. Pilipovic, V. Risojevic, and G. Mirjanic. Pollen bearing honey bee detection in hive entrance video recorded by remote embedded system for pollination monitoring. *Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences ISPRS*, III-7:51–57, 2016.

[Bra00] Gary Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001.

[BS16] Gérard Biau and Erwan Scornet. A random forest guided tour. *TEST*, 25(2):197–227, Jun 2016.

[C+15] François Chollet et al. Keras. `https://keras.io`, 2015. (last accessed: 02.06.2018).

[CGKM13] Guillaume Chiron, Petra Gomez-Krämer, and Michel Ménard. Detecting and tracking honeybees in 3d at the beehive entrance using stereo vision. *Journal on Image and Video Processing, EURASIP*, 2013(1):1–17, 2013.

[CMS08] Jason Campbell, Lily Mummert, and Rahul Sukthankar. Video monitoring of honey bee colonies at the hive entrance. *Visual observation & analysis of animal & insect behavior, ICPR*, 8:1–4, 2008.

[CPMR05] Junqing Chen, T. N. Pappas, A. Mojsilovic, and B. E. Rogowitz. Adaptive perceptual color-texture image segmentation. *IEEE Transactions on Image Processing*, 14(10):1524–1536, October 2005.

[CYJL12] Chiu Chen, En-Cheng Yang, Joe-Air Jiang, and Ta-Te Lin. An imaging system for monitoring the in-and-out activity of honey bees. *Computers and Electronics in Agriculture*, 89:100–109, November 2012.

[DJDARG82] David De Jong, D De Andrea Roma, and Lionel Gonçalves. A comparative analysis of shaking solutions for the detection of varroa jacobsoni on adult honeybees. In *Proceedings of the Apidologie*, volume 13, pages 297–306, 01 1982.

[DMM00] Keith S. Delaplane, Daniel R. Mayer, and Daniel F. Mayer. *Crop pollination by bees*. Cabi, 2000.

[ESM+09] Jay D. Evans, Claude Saegerman, Chris Mullin, Eric Haubruge, Bach Kim Nguyen, Maryann Frazier, Jim Frazier, Diana Cox-Foster, Yanping Chen, Robyn Underwood, et al. Colony collapse disorder: a descriptive study. *PLoS ONE*, 4(8):e6481, 2009.

[Eva18] Huw Evans. Arnia: Using remote hive monitoring data. `http://www.beeculture.com/arnia-using-remote-hive-monitoring-data/`, 2018. (last accessed: 02.06.2018).

[FF95] Andrew Fitzgibbon and Robert Fisher. A buyer's guide to conic fitting. In *Proceedings of the 5th British Machine Vision Conference*, pages 513–522, 02 1995.

[FHIR03] Ingemar Fries, Henrik Hansen, Anton Imdorf, and Peter Rosenkranz. Swarming in honey bees (apis mellifera) and varroa destructor population development in sweden. *Apidologie*, 34(4):389–397, 2003.

[FNK13] Roy M. Francis, Steen L. Nielsen, and Per Kryger. Varroa-virus interaction in collapsing honey bee colonies. *PLoS ONE*, 8(3):1–9, March 2013.

[Gat14] B.N. Gates. The temperature of the bee colony, united states department of agriculture, dept. *Bull*, (96), 1914.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GCZ+15] M. Giammarini, E. Concettoni, C. C. Zazzarini, N. Orlandini, M. Albanesi, and C. Cristalli. Beehive lab project - sensorized hive for bee colonies life study. In *Proceedings of the 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES)*, pages 121–126, October 2015.

[GG15] Sebastian Gisder and Elke Genersch. Special issue: Honey bee viruses. *Viruses*, 7(10):2885, 2015.

[GM16] Jeffrey Glick and Katarina Miller. Insect classification with heirarchical deep convolutional neural networks. *Convolutional Neural Networks for Visual Recognition*, 2016.

[Gol17] V. A. Golovko. Deep learning: an overview and main paradigms. *Optical Memory and Neural Networks*, 26(1):1–17, Jan 2017.

[GSSV09] Nicola Gallai, Jean-Michel Salles, Josef Settele, and Bernard E Vaissière. Economic valuation of the vulnerability of world agriculture confronted with pollinator decline. *Ecological Economics*, 68(3):810–821, 2009.

[GVAG16] Santiago González, Tito Raúl Vargas, Pau Arce, and Juan Carlos Guerri. Energy optimization for video monitoring system in agricultural areas using single board computer nodes and wireless ad hoc networks. In *Proceedings of the Signal Processing, Images and Artificial Vision (STSIVA)*, pages 1–7. IEEE, 2016.

[HG09] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, Sept 2009.

[HHDV16] Donald Howard, Gordon Hunter, Olga Duran, and Demetrios Venetsanos. Progress towards an intelligent beehive: Building an intelligent environment to promote the well-being of honeybees. In *Proceedings of the 12th International Conference on Intelligent Environments, IE2016*, pages 262–265. IEEE, 2016.

[Hin18] Geoffrey E. Hinton. Cs321 neural networks for machine learning-overview of mini batch gradient descent. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2018. (last accessed: 02.06.2018).

[HSK+12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR - Computing Research Repository*, 2012.

[Hul09] Philip E. Hulme. Trade, transport and trouble: managing invasive species pathways in an era of globalization. *Journal of Applied Ecology*, 46(1):10–18, 2009.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[KA14] Simonyan Karen and Zisserman Andrew. Very deep convolutional networks for large-scale image recognition. *Learning Representations*, abs/1409.1556, 2014.

[Kar18] A. Karpathy. Cs231n convolutional neural networks for visual recognition. `http://cs231n.stanford.edu/`, 2018. (last accessed: 02.06.2018).

[KB04] Ernst Kussul and Tatiana Baidyk. Improved method of handwritten digit recognition tested on mnist database. *Image and Vision Computing*, 22(12):971–981, 2004.

[KJYL11] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proceedings of the CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, volume 2, 2011.

[KK14] Yılmaz Kaya and Lokman Kayci. Application of artificial neural network for automatic detection of butterfly species using color and texture features. *The Visual Computer*, 30(1):71–79, 2014.

[KM09] Uwe Knauer and Beate Meffert. Evaluation based combining of classifiers for monitoring honeybees. In *Proceedings of the Workshop on Applications of Computer Vision (WACV)*, pages 1–6. IEEE, 2009.

[KOOI11] Toshifumi Kimura, Mizue Ohashi, Ryuichi Okada, and Hidetoshi Ikeno. A new approach for the simultaneous tracking of multiple honeybees for analysis of hive behavior. *Apidologie*, 42(5):607, 2011.

[KR16] Vladimir Kulyukin and Sai Kiran Reka. A computer vision algorithm for omnidirectional bee counting at langstroth beehive entrances. In *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)*, page 229. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.

[KS04] S. R. Kodituwakku and S. Selvarajah. Comparison of color features for image retrieval. *Indian Journal of Computer Science and Engineering*, 1(3):207–211, 2004.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[KTP15] David J. Kale, Rahman Tashakkori, and R. Mitchell Parry. Automated beehive surveillance using computer vision. In *Proceedings of the SoutheastCon 2015*, pages 1–3, April 2015.

[KVC+07] Alexandra-Maria Klein, Bernard E. Vaissiere, James H. Cane, Ingolf Steffan-Dewenter, Saul A. Cunningham, Claire Kremen, and Teja Tscharntke. Importance of pollinators in changing landscapes for world crops. In *Proceedings of the Royal Society of London B: Biological Sciences*, volume 274, pages 303–313. The Royal Society, 2007.

[LBBH98] Yann LeCun, L. Bottou, Yoshuga Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 86, pages 2278–2324, Nov 1998.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[LMB+10] K. V. Lee, R. D. Moon, E. C. Burkness, W. D. Hutchison, and M. Spivak. Practical sampling plans for varroa destructor (acari: Varroidae) in apis mellifera (hymenoptera: Apidae) colonies and apiaries. *Journal of Economic Entomology*, 103(4):1039–1050, 2010.

[Loc16] Barbara Locke. Natural varroa mite-surviving apis mellifera honeybee populations. *Apidologie*, 47(3):467–482, May 2016.

[Low99] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the 7th IEEE international conference on Computer Vision, 1999.*, volume 2, pages 1150–1157. Ieee, 1999.

[M+76] Samuel Emmett McGregor et al. *Insect pollination of cultivated crop plants*, volume 496. Agricultural Research Service, US Department of Agriculture, 1976.

[Mar01] Stephen J. Martin. The role of varroa and viral pathogens in the collapse of honeybee colonies: a modelling approach. *Journal of Applied Ecology*, 38(5):1082–1093, 2001.

[MCR+17] Maxime Martineau, Donatello Conte, Romain Raveaux, Ingrid Arnault, Damien Munier, and Gilles Venturini. A survey on image-based insect classification. *Pattern Recognition*, 65:273 – 284, 2017.

[Mel18] Melixa. Melixa hive monitoring systems. `http://melixa.eu/en/`, 2018. (last accessed: 02.06.2018).

[MG15] Alex Mcmenamin and Elke Genersch. Honey bee colony losses and associated virus. 107, 01 2015.

[MH15] W. G. Meikle and N. Holst. Application of continuous monitoring of honeybee colonies. *Apidologie*, 46(1):10–22, 2015.

[MWE02] Paula Macedo, J Wu, and Marion Ellis. Using inert dusts to detect and assess varroa infestation in honey bee colonies. 41, 06 2002.

[NLC16] Francesco Nazzi and Yves Le Conte. Ecology of varroa destructor, the major ectoparasite of the western honey bee, apis mellifera. *Annual review of entomology*, 61:417–432, 2016.

[ORBD08] Sang Min Oh, James M Rehg, Tucker Balch, and Frank Dellaert. Learning and inferring motion patterns using parametric segmental switching linear dynamic systems. *International Journal of Computer Vision*, 77(1-3):103–124, 2008.

[PBK+10] Simon G. Potts, Jacobus C. Biesmeijer, Claire Kremen, Peter Neumann, Oliver Schweiger, and William E. Kunin. Global pollinator declines: trends, impacts and drivers. *Trends in Ecology & Evolution*, 25(6):345 – 353, 2010.

[QAHL14] A. Qandour, I. Ahmad, D. Habibi, and M. Leppard. Remote beehive monitoring using acoustic signals. *Acoustics Australia*, 42:205, 2014.

[Ras14] Sebastian Raschka. Naive bayes and text classification I - introduction and theory. *CoRR - Computing Research Repository*, abs/1410.5329, 2014.

[RAZ10] Peter Rosenkranz, Pia Aumeier, and Bettina Ziegelmann. Biology and control of varroa destructor. *Journal of Invertebrate Pathology*, 103:S96–S119, January 2010.

[RBPRFM+17] Melvin Ramírez-Bogantes, Juan P. Prendas-Rojas, Geovanni Figueroa-Mata, Rafael A. Calderon, Oscar Salas-Huertas, and Carlos M. Travieso. Cognitive modeling of the natural behavior of the varroa destructor mite on video. *Cognitive Computation*, 9(4):482–493, Aug 2017.

[RGS+05] J. Riley, Uwe Greggers, A. D. Smith, Donna Reynolds, and Robert Menzel. The flight paths of honeybees recruited by the waggle dance. *Nature*, 435 7039:205–7, 2005.

[RH14] Guy Rodet and Mickaël Henry. Analytic partitioning of honeybee (apis mellifera l.) flight activity at nest entrance: adaptation and behavioural inertia in a changing environment. *Ecological Research*, 29(6):1043–1051, 2014.

[RMA+18] I. F. Rodriguez, R. Megret, E. Acuna, J. L. Agosto-Rivera, and T. Giray. Recognition of pollen-bearing bees from video using convolutional neural network. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV), 2018*, pages 314–322, 2018.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[RPT+12] M. Ramírez, J. P. Prendas, C. M. Travieso, R. Calderón, and O. Salas. Detection of the mite varroa destructor in honey bee cells by video sequence processing. In *Proceedings of the 16th International Conference*

on Intelligent Engineering Systems (INES), 2012, pages 103–108, June 2012.

[SBPM18] Schurischuster S., Remeseiro B., Radeva P., and Kampel M. A preliminary study of image analysis for parasite detection on honey bees. In *Proceedings of the 15th International Conference on Image Analysis and Recognition (ICIAR 2018)*, 2018.

[SKP18] M. Sewak, M.R. Karim, and P. Pujari. *Practical Convolutional Neural Networks: Implement advanced deep learning models using Python*. Packt Publishing, 2018.

[SMA+94] M. H. Struye, H. J. Mortier, G. Arnold, C. Miniggio, and R. Borneck. Microprocessor-controlled monitoring of honeybee flight activity at the hive entrance. *Apidologie*, 25:384–384, 1994.

[SMB10] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 92–101. Springer, 2010.

[SMD13] Yale Song, Louis-Philippe Morency, and Randall Davis. Distribution-sensitive learning for imbalanced datasets. In *Proceedings of the 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), 2013*, pages 1–6. IEEE, 2013.

[SO95] Markus A. Stricker and Markus Orengo. Similarity of color images. In *Proceedings of the Storage and Retrieval for Image and Video Databases*, 1995.

[SR15] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10(3):1–21, 03 2015.

[SS01] George Stockman and Linda G. Shapiro. *Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[SSMB16] Schurischuster S., Zambanini S., Kampel M., and Lamp B. Sensor study for monitoring varroa mites on honey bees (apis mellifera). In *Proceedings of Visual observation and analysis of Vertebrate And Insect Behavior (VAIB) Workshop*, 2016.

[SSMM15] Matthew R. Smith, Gitanjali M. Singh, Dariush Mozaffarian, and Samuel S. Myers. Effects of decreases of animal pollinators on human nutrition and global health: a modelling analysis. *The Lancet*, 386(10007):1964–1972, 2016/06/15 2015.

[STS+15] Nicola Seitz, Kirsten S. Traynor, Nathalie Steinhauer, Karen Rennich, Michael E. Wilson, James D. Ellis, Robyn Rose, David R. Tarpy, Ramesh R. Sagili, Dewey M. Caron, Keith S. Delaplane, Juliana Rangel, Kathleen Lee, Kathy Baylis, James T. Wilkes, John A. Skinner, Jeffery S. Pettis, and Dennis vanEngelsdorp. A national survey of managed honey bee 2014–2015 annual colony losses in the USA. *Journal of Apicultural Research*, 54(4):292–304, 2015.

[Tau18] Jürgen Tautz. Hobos (honeybee online studies). `http://www.hobos.de/`, 2018. (last accessed: 02.06.2018).

[TG15] Rahman Tashakkori and Ahmad Ghadiri. Image processing for honey bee hive health monitoring. In *Proceedings of the SoutheastCon 2015*, pages 1–7. IEEE, 2015.

[THKA16] Gang Jun Tu, Mikkel Kragh Hansen, Per Kryger, and Peter Ahrendt. Automatic behaviour analysis system for honeybees using computer vision. *Computers and Electronics in Agriculture*, 122:10–18, 2016.

[vHJUP10] Dennis vanEngelsdorp, Jerry Hayes Jr, Robyn M. Underwood, and Jeffery S. Lock Pettis. A survey of honey bee colony losses in the united states, fall 2008 to spring 2009. *Journal of Apicultural Research*, 49(1):7–14, 2010.

[VJ01] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001. CVPR 2001.*, volume 1, pages I–I. IEEE, 2001.

[vM10] Dennis vanEngelsdorp and Marina Doris Meixner. A historical review of managed honey bee populations in europe and the united states and the factors that may affect them. *Journal of Invertebrate Pathology*, 103, Supplement:S80 – S95, 2010.

[War07] Émile Warré. *Beekeeping For All*. David Heaf, 2007.

[WPS02] P. Wayne Power and Johann Schoonees. Understanding background mixture models for foreground segmentation. In *Proceedings of the Image and Vision Computing New Zealand*, 01 2002.

[Wri28] B. Wright. Lundie's "flight activities of the honey bee.". *Bee World*, 9(2):21–23, 1928.

[WWRL17] Fernando Wario, Benjamin Wild, Raúl Rojas, and Tim Landgraf. Automatic detection and decoding of honey bee waggle dances. *PLoS ONE*, 12(12):1–16, 12 2017.

[YC15] Cheng Yang and John Collins. A model for honey bee tracking on 2d video. In *Proceedings of the International Conference on Image and Vision Computing New Zealand IVCNZ2015*, pages 1–6. IEEE, 2015.

[YLG11] Ting Hua Yi, Hong Nan Li, and Ming Gu. Optimal sensor placement for structural health monitoring based on multiple optimization strategies. *The Structural Design of Tall and Special Buildings*, 20(7):881–900, 2011.

[YYPH14] X. C. Yin, C. Yang, W. Y. Pei, and H. W. Hao. Shallow classification or deep learning: An experimental study. In *Proceedings of the 22nd International Conference on Pattern Recognition, 2014*, pages 1904–1909, Aug 2014.

[ZBMS15] Aleksejs Zacepins, Valters Brusbardis, Jurijs Meitalovs, and Egils Stalidzans. Challenges in the development of precision beekeeping. *Biosystems Engineering*, 130:60 – 71, 2015.

[ZKA+16] Aleksejs Zacepins, Armands Kviesis, Peter Ahrendt, Uwe Richter, Saban Tekin, and Mahmut Durgun. Beekeeping in the future - smart apiary management. In *Proceedings of the 17th International Carpathian Control Conference (ICCC), 2016*, pages 808–812. IEEE, 2016.

[ZSM12] Aleksejs Zacepins, Egils Stalidzans, and Jurijs Meitalovs. Application of information technologies in precision apiculture. In *Proceedings of the 13th International Conference on Precision Agriculture*, 2012.