OSNABRÜCK UNIVERSITY

STUDY PROJECT

Computer Vision Based Asparagus Classification

Authors:
Maren Born,
Michael Gerstenberger,
Katharina Groß,
Richard Ruppel,
Sophia Schulze-Weddige,
Malin Spaniol,
Josefine Zerbe

Supervisors:
Dr. Ulf Krumnack
M.Sc. Axel Schaffland

Research Group Computer Vision
Institute of Cognitive Science

October 5, 2020

OSNABRÜCK UNIVERSITY

Abstract

School of Human Sciences
Institute of Cognitive Science

Master of Science

Computer Vision Based Asparagus Classification

by Maren Born, Michael Gerstenberger, Katharina Groß, Richard Ruppel,
Sophia Schulze-Weddige, Malin Spaniol, Josefine Zerbe

In the agricultural industry, the process of sorting food products into different quality classes is nowadays primarily achieved through sorting machines using classical, rule-based methods. One food product with many possible quality classes is white asparagus.
In this project, our team tackles the issue of asparagus classification by exploring different classical and state of the art machine learning and computer vision techniques. For this purpose, a range of supervised, semi-supervised, and unsupervised learning approaches are tested and evaluated. The data set is based on the output of the asparagus sorting machine Autoselect ATS II, which uses image data for classification. The result of the project shows that different approaches are suitable for solving the issue. For comparison with classical methods used in the industry and to provide a practical application, further testing of the approaches is needed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An essential step in the commercial asparagus cultivation is the sorting of the harvested asparagus into different quality classes. Depending on size, shape, and color, a label is assigned to the individual asparagus which determines the quality class of the asparagus (United Nations Economic Commission for Europe (UNECE), 2017). Nowadays, this task is usually performed automatically, with modern asparagus sorting machines achieving a throughput of up to twelve asparagus spears per second (Ting, 2015). However, the accuracy of such machines can be unsatisfying. Manual resorting is usually necessary and thereby causes substantial effort.

The aim of this study project was to investigate how techniques from computer vision, both classical and deep learning-based, can be applied to improve classification results for asparagus sorting machines. Asparagus is sorted by quality. The quality is defined based on several features that represent differences in color, shape, or texture. Hence, improving sorting algorithms for the classification of asparagus requires reliable means of measuring these features. As such, we were interested in the question of how the quality-defining features of asparagus can be estimated using state of the art computer vision and machine learning techniques. The project was supported by a local asparagus farm and a company specializing in asparagus sorting machines. They provided training data and expertise on the classification of the asparagus. The idea to apply new machine learning approaches to a commercial problem of the agricultural industry seemed promising and challenging at the same time. As the data received from the farm was unlabeled, a larger focus on the preprocessing of the recorded image data became inevitable. Further, the variance in quality classes and the subjective view of human sorting behavior toughened the perspective on a practical application into the actual sorting machine. Nevertheless, the original goal of studying different techniques in computer vision and testing their usability in a fixed hardware setting for asparagus classification gave valuable insights into the practical application of previously theoretically assessed knowledge.
The hands-on experience on a long-term project gave us the chance to improve our project management skills, learn how to distribute work, and how to organize ourselves in a larger team. Over the course of one year, the team members worked on the implementation of different approaches to tackle the issue of image classification. After intense engagement with the subject, the methodologies and the project outcome were documented and critically reviewed. On the following pages, the different stages of the project together with the results of the applied computer vision approaches are described in detail. The report chapters are mostly in chronological order, each with the focus on a different stage of the study project throughout the year.

In chapter 1 Introduction, the idea of the project is presented together with an introduction to the current standards of image classification in machine learning as

well as a general background on the quality classes, the sorting process, and classification of asparagus in the agricultural industry.
Chapter 2 Organization and data acquisition comprises the process of data acquisition and organizational background to the project. This includes task management and teamwork, as well as a thorough evaluation of our planning abilities as a group. Further, the process of data collection and a first inspection of the sorting machine are elaborated, with an emphasis on the first issues of unlabeled data. There is not much literature on applying deep learning to agriculture. Still, our literature research gave an insight into practical possibilities and a rough guideline for our endeavor. The collected literature that was found to be close to our topic is reported at the end of this chapter.
In chapter 3 Preprocessing and data set creation, the preprocessing phase is captured with the building of the data set at its end section. The first classical machine learning techniques for automatic feature extraction on image data are explored, followed by the process of manual feature extraction with the usage of a labeling application. The resulting outcome is reported, evaluated, and the labeled image data is transformed into a data set for the training of the chosen models.
Chapter 4 Classification constitutes the different approaches that were chosen to tackle the issue of image classification. All approaches are critically reviewed and discussed in the subchapters. An emphasis was put on the application of deep learning techniques for the classification of the asparagus. As the collected data includes labeled and unlabeled samples, methods from the range of supervised learning, semi-supervised learning, and unsupervised learning were tested.
Chapter 5 Summary of results contains the overall results of the classification approaches. The outcome of each approach is described and compared to the other methods as well as to the original classification accuracy of the sorting machine at the asparagus farm. Further, the overall results are discussed with an evaluation of their practical application in the food industry in chapter 6 Discussion. The overall outcome of the project is judged as a scientific study and as a team management experience.
In Chapter 7 Conclusion, the findings are set into a broader perspective. The project and its results are summarized on a scientific basis as well as on an organizational level while the future prospects of the project are assessed.

Before analyzing the specific context of software development for the application in classification tasks further, a thorough insight into the idea of the study project is given in the following chapter. Additionally, background on the addressed topics of machine learning and the sorting of asparagus is provided. In 1.1 The project, the objective of the study project is introduced. The next section 1.2 Classification based on computer vision, gives an overview of the machine learning techniques that explore how image classification can be implemented and how these approaches can provide a solution to our issue. The first chapter concludes with 1.3 Background on sorting asparagus, where the process of asparagus classification in the commercial food industry is illustrated and the different quality classes of asparagus are defined.

1.1 The project

The objective of the study project was to find both, conventional and deep learning computer vision-based approaches that can be tested for their practical application in the commercial sorting of white asparagus. The different methods were implemented based on the image data that was received from the automatic sorting

machine Autoselect ATS II employed by the asparagus farm "Gut Holsterfeld" [1]. It is explored whether approaches from the fields of machine learning and computer vision can be applied to improve the classification behavior of the asparagus sorting machine. Further it is investigated, if these approaches can be used in a more industrial than scientific setting for the specific task of asparagus classification. The initial intention to directly implement new software into the machine was postponed to allow for intensive research on the different approaches and on fine-tuning the received data to build a practical data set to train neural networks.
The idea for the project was formed by one of the students of the study group. She has a familial relationship to the asparagus farm Gut Holsterfeld and had received note of the unsatisfying classification performance of its sorting machine. The practical application to a real-world problem sparked the interest of the group and the general curiosity towards computer vision, in particular neural networks, further inspired to deal with the sorting issue in a deep learning context, as well as with classical methods.
All project members are students in the field of Cognitive Science at the University of Osnabrück. The study project is part of the Master Program in Cognitive Science at the University of Osnabrück. It is supervised by Dr. Ulf Krumnack and Axel Schaffland.
The study project intends to confirm the ability of its participants to independently formulate and solve an unknown problem from the scientific context of one subject area using the methods and terms they previously learned (Universität Osnabrück, 2019a; Universität Osnabrück, 2019b). This includes the documentation and presentation of the results, the methodologies as well as the reflection on the work process. Within the scope of the project, for example, the development of software, analysis, and interpretation of statistical data material is practiced. A further aspect of the study project is to deepen the communicative and decision-making competence of its participants (Universität Osnabrück, 2019a). The idea is to train independent project work in groups of students under conditions that are common for research projects in science or industry.
As the project took part in the scope of an examination for the Cognitive Science Master Program at the University of Osnabrück, most of the work was developed at the university, that is, at the Institute of Cognitive Science (IKW).
Additional image data was received from the asparagus farm "Querdel's Hof" [2]. Further cooperation existed with the mechanical engineering company HMF Hermeler Maschinenbau GmbH [3] that developed the sorting machine Autoselect ATS II and provided valuable expertise on the sorting issue.
All associated software is stored online, in our GitHub repository, and at the institute internal storing system.[4]

1.2 Classification based on computer vision

In this chapter, the current standard of computer-based image classification is described. The main focus will be on Artificial Neural Networks (ANNs) and classical

[1] see the website of Gut Holsterfeld at `https://www.gut-holsterfeld.de/`
[2] see the website of Querdel's Hof at `https://www.querdel.de/`
[3] see the website of HMF Hermeler at `www.hmf-hermeler.de`
[4] The documentation to the project can be found at `https://asparagus.readthedocs.io/en/latest`. The GitHub repository can be found at `https://github.com/CogSciUOS/asparagus`. Until 31/07/2020, the internal storing folder to the project at the University of Osnabrück could be accessed via /net/projects/scratch/winter/valid_until_31_July_2020/asparagus. Since then, it can be accessed via /net/projects/scratch/summer/valid_until_31_January_2021/jzerbe, until 31/01/2021.

computer vision techniques. A broad overview of relevant topics will be given and their importance for this project will be underlined.
Computer vision is a field of computer science that aims to automatically extract high-level understanding from image data and provide appropriate output. It is closely linked with machine learning, which describes the ability of a system to learn and improve from experience rather than being specifically programmed. Machine learning is frequently used to solve computer vision tasks.

Image classification is one of the main subfields of computer vision which gains a lot of attention in the scientific as well as the economic world. Besides the classical computer vision techniques that use algorithmic approaches to determine patterns, edges, and other points of interest that can help to classify images, artificial intelligence was introduced to the field in the 1960s (Szeliski, 2010). Since then, more and more creative and complex artificial neural networks were introduced to solve numerous classification tasks, including the recognition of letters, faces, and street signs (Mirończuk and Protasiewicz, 2018; Balaban, 2015; Stallkamp et al., 2011).
Some experts claim that artificial neural networks revolutionized the field of image classification, yielding better results than ever before (He et al., 2016a; Krizhevsky, Sutskever, and Hinton, 2012). More and more challenges, for example ImageNet (Russakovsky et al., 2015), were introduced and computer scientists all over the world implemented creative solutions for the proposed problems. Additionally, influential companies like Google have a deep interest in finding solutions for image-based classification problems and push the research on these and related topics even further. In the era of optimization, computer-based classification became indispensable in many industries and also found its way into agriculture which is the field of interest in this study project. In the following, a short introduction in both artificial neural networks and classical computer vision techniques is given which will span a bridge to our current classification problem.

Neural networks are used for image classification in many domains. In contrast to the algorithmic approaches, it is not determined by the programmer how and what exactly the neural network learns. A large amount of data is provided to the model, which then extracts relevant information to learn the classification task. However, what is relevant to the model is not necessarily relevant or interpretable to a human observer. This is one of the biggest disadvantages compared to classical computer vision approaches, which are understandable and, therefore, interpretable and adjustable. Another problem that comes along with this is that the bias for those approaches often does not lie in the code itself but in the data, which is by far more difficult to detect and surpass. For this reason, many people see artificial neural networks as a black box, which may yield great results but cannot be fully understood. Recent advances try to tackle that issue by systematically researching artificial intelligence with the aim of making it more explainable (Tjoa and Guan, 2019; Gilpin et al., 2018).
In image classification, usually Convolutional Neural Networks (CNNs) are used (Géron, 2019; LeCun and Bengio, 1995), which are loosely inspired by the visual cortex of the brain. The idea is that highly specialized components or filters learn a very specific task, which is similar to the receptive fields of neurons in the visual cortex (Hubel and Wiesel, 1962). These components can then be combined to high-level features which, in turn, can be combined to objects that can be used for classification (Géron, 2019; Bishop, 2006; LeCun and Bengio, 1995). In CNNs this concept is implemented by several successive convolutional layers in which one or more filters are slid over the input, generating so-called feature maps. Each unit in one feature map looks for the same feature but in different locations of the input.

In recent years, CNNs improved so much that they outperform humans in many classification tasks (Russakovsky et al., 2015; Karpathy, 2014).

Although machine learning exhibits very promising results and a lot of research and literature is available on the topic, many branches of industry still rely on traditional computer vision techniques in their implementation of image classification. This also applies to the asparagus sorting paradigm. To the best of our knowledge, no asparagus sorting machine is currently on the market that uses artificial intelligence for its classification algorithm.
Many classical computer vision algorithms aim to detect and describe points of interest in the input images that can be generalized to features. For these features, low-level attributes such as rapid changes in color or luminance can be used. In contrast to the features learned with the help of machine learning, these features are not specific to any training data set and therefore do not depend on it being well-constructed. Further, they are usually created in a way that is interpretable by humans which makes them very flexible and easily adaptable to specific use cases. This is why in some cases, traditional computer vision techniques can solve a problem much more efficiently than deep learning approaches.
One of the big advantages of deep learning algorithms is that they can extract underlying features from the data. In machine learning, it is essential to know about the components constructing a problem, whereas artificial neural networks have the ability to learn the features of the data, combine and correlate them and thus enable faster learning without an explicit command how to do so (LeCun, Bengio, and Hinton, 2015).

In summary, both approaches have interesting implications for computer-based image classification tasks and provide promising techniques for our problem of asparagus classification. While we rely mostly on machine learning for the classification task itself, traditional computer vision algorithms are used to detect important features from the images.

1.3 Background on sorting asparagus

In this section, a background on sorting asparagus is given with a focus on the quality classes assessed by the asparagus farm Gut Holsterfeld. The owner of the asparagus farm, Mr. Silvan Schulze-Weddige, and the CEO of the engineering company HMF Hermeler Maschinenbau, Mr. Thomas Hermeler, were consulted on this issue.

While asparagus accounts for a fifth of the area used for vegetable cultivation in Germany, the harvesting season of white asparagus only spans over a period of two months, beginning in April and ending on the 24th of June (Statistisches Bundesamt (Destatis), 2017; Weber and Quinckhardt, 2018). During this period, the asparagus is harvested, classified, and sold in accord with a price range that is defined by the quality class of the asparagus.
In the European Union, there is a uniform system for the sorting of asparagus into quality classes (Europäische Komission, 1999; United Nations Economic Commission for Europe (UNECE), 2017).[5,6] However, supply and demand usually determine the number and accuracy of these classes. One of the first defining features

[5]see https://mlr.baden-wuerttemberg.de/de/unser-service/presse-und-oeffentlichkeitsarbeit/pressemitteilung/pid/nationale-handelsklassen-fuer-frisches-obst-und-gemuese-abgeschafft-1/
[6]see https://www.bzfe.de/inhalt/spargel-kennzeichnung-5876.html

Label	Description
I A Anna	Quality class Extra is represented by I A Anna, I A Bona, and I A Clara. I A is defined as asparagus that is perfectly straight and white. There are no large pressure marks, no rust, no violet color, and no curvature. The identification label Anna marks that the width (diameter) of the spear is in a range of 20 - 26 mm.
I A Bona	Quality class Extra asparagus with a width of width 18 - 20 mm.
I A Clara	Quality class Extra asparagus with a width of 16 - 18 mm.
I A Krumme	The asparagus fulfills all criteria for quality class Extra while it shows a slight curvature.
I A Violett	The asparagus is of a violet complexion, while it complies with all guidelines for quality class Extra. The color change is due to the spear having contact with sunlight before being harvested.
II A	The asparagus is both curved and violet.
II B	The asparagus is curved, rusty, and/or in any other way damaged or classified as a defective product.
Rost	The asparagus shows traces of rust or has rusty parts that occur when the roots of the plant were injured in a preceding year. This should not be confused with a fungal disease that can also be referred to as rust.
Dicke	The asparagus exceeds 26 mm in width.
Hohle	The asparagus is hollow from the inside.
Blume	The head region of the asparagus is about to bloom or it shows clear outlines of a flower.
Suppe	The asparagus has a width of less than 16 mm.
Bruch	Any asparagus that is below a length of 210 mm. However, another distinction in this class label is made between asparagus that has an intact head and a length of at least 100 mm (Kerze), asparagus without head (Bruch), and an asparagus head alone (Köpfchen).
Keule	The upper end of the asparagus is thicker than the lower end. The shape is similar to a club, hence the name of the class. This class label was of no concern to the project because no image data of this class label could be recorded.

TABLE 1.1: **List of Asparagus Quality Classes** In this table, 14 quality classes for asparagus categorization are listed and described, according to the asparagus farm Gut Holsterfeld. Except for the class label Keule, all 13 quality classes were used as the goal label for the asparagus classification.

is the color of the asparagus which comprises four categories: white, violet, violet-green, and green (Europäische Komission, 1999). For this project, only the first two colors are of relevance. A further distinction is made between the quality classes Extra, class I, and class II. The class Extra defines the product as perfect quality, class I defines it as good quality, and class II includes products that do not qualify for the other classes but satisfy the minimum requirements for commercial distribution (Europäische Komission, 1999). The last distinction is made for the characteristics of length and width. White and violet asparagus may not exceed 220 mm in length. The minimal length of the asparagus varies but should be above 170 mm for long asparagus. Additionally, there is some level of tolerance accepted for the quality classes. According to the quality class, 5% - 15% of wrongly sorted asparagus is tolerated in a package or bundle (Europäische Komission, 1999).

Depending on how carefully the individual farmer further categorizes the asparagus, additional classes can be established. Regional differences to those classes make the challenge for the manufacturers of sorting machines even more complicated. The class labels as stated in this report depend solely on the sorting conduct of the asparagus farm Gut Holsterfeld and do not necessarily apply to any other classification system of the asparagus industry.
In Table 1.1, 14 quality classes are shortly described, of which 13 classes are relevant to this project. The classification tree in Figure 1.1 illustrates the decision process that underlies the categorization of the relevant classes.

One of the challenges of asparagus classification with a machine lies in the human sorting error. The machine can assess exact calculations about, for example, the length or the width of a spear, while the human observer might not be able to detect minor differences in these features. Thus, a threshold has to be defined if a spear is sorted into a certain class. However, the data on which the machine calculates its features to characterize an asparagus spear was previously labeled by a human. The sorter might miss subtle differences in color. For example, the machine might sort a spear as the class label Violett while the spear would be judged as the class label I A Anna by the human sorter. Since human sorting behavior is subjective, the same asparagus can be categorized differently by two independent

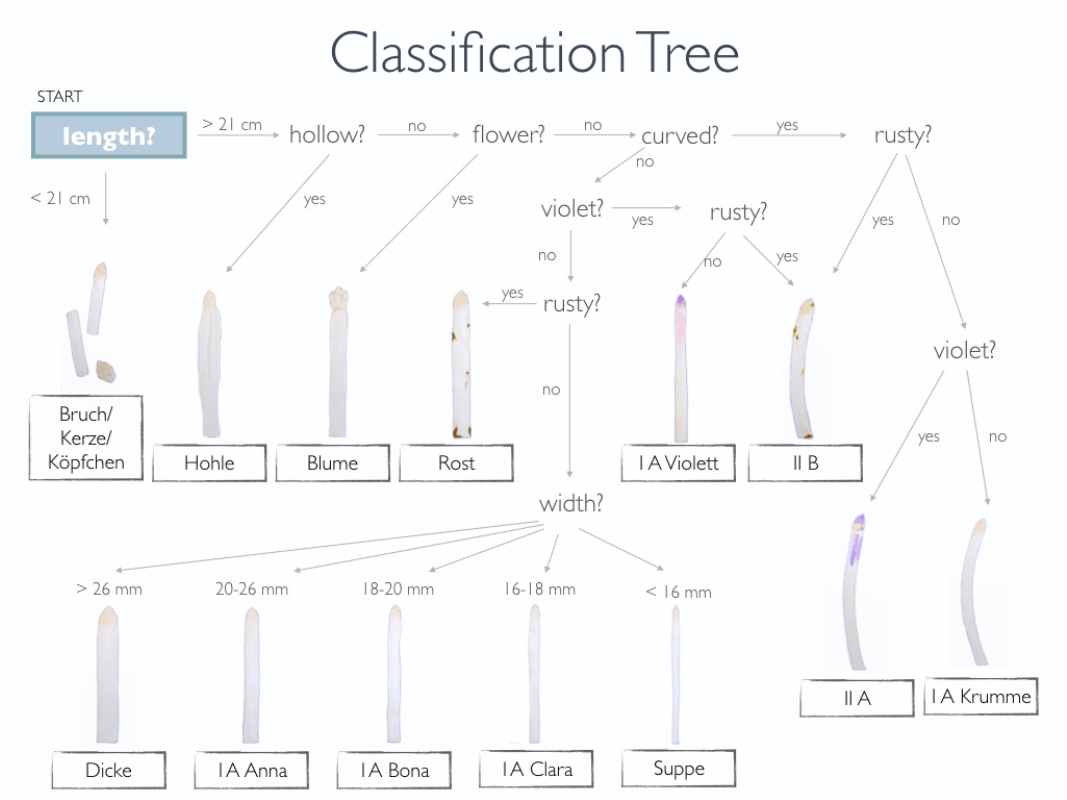


FIGURE 1.1: **Asparagus Classification Tree** The classification tree shows how each asparagus is attributed with a label of its quality class. It was drawn by the project group and follows the sorting rules of the asparagus farm Gut Holsterfeld. Starting from the upper left corner of the image, mostly binary decisions are made until a label is reached. Any further damaged or defective asparagus automatically belongs in category II B.

sorters. Furthermore, asparagus sorted twice by the same person at different time points might be assigned with two different class labels.
A second problem for the sorting machine poses the interpretation of colors. The color can be perfectly recognized by its program, however, the structure of the color is also important for correct classification. The machine might not be able to distinguish whether the asparagus was, e.g. of brownish color because it is very rusty or simply very dirty in certain cases.
A third factor for classification difficulties is caused by the restricted view of the product. Asparagus can look perfectly shaped from one angle but might be damaged on the side that is not exposed to the camera of the sorting machine.
Another complication poses the question of demand and supply. During some seasons there is more asparagus of a certain class label and less of another one available. The farmer usually distributes the number of spears belonging to a quality class according to the seasonal conditions. For example, during a season with less high-quality asparagus of the classes I A, the sorting threshold will shift. Spears that are usually sorted into, e.g. a lower class will now belong to a higher quality class. The challenge for a sorting machine will not only be to sort for the class criteria but also to provide a flexible and individual solution in accordance with the preferences of the farmer.
These challenges are not impossible to overcome but they make it more difficult to find a solution to the sorting task and compromises in the implementation (f.e. the precision of classification) might be necessary.

Chapter 2

Organization and data acquisition

In the second chapter of the report, the first stage of the study project is discussed, namely organizing, researching, and collecting the data.
The first section of the chapter 2.1 Roadmap of the project gives an overview of the time management of the study group. It is followed by the subchapter 2.2 Organisation of the study group in which communication and teamwork are assessed. In 2.3 Data collection, the acquisition of the data from the sorting machine Autoselect ATS II is described in more detail. The last section 2.4 Literature on food classification using computer vision presents the retrieved literature concerning the issue of agricultural product classification with machine learning based approaches.

2.1 Roadmap of the project

At the beginning of the project, a roadmap was created to structure the year into different working stages as well as to have an overview of the tasks and problems that needed to be addressed.

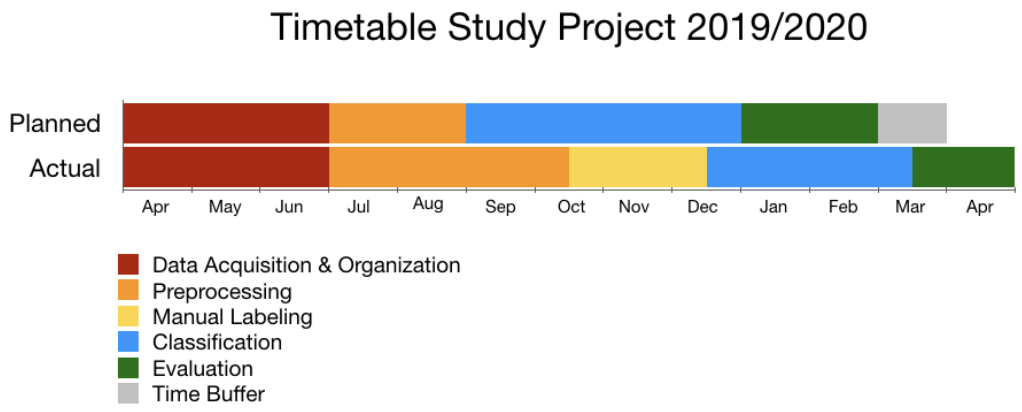


FIGURE 2.1: **Timetable of the Project** The upper timeline shows the estimated time of the study project from April 2019 to April/Mai 2020. The lower timeline displays how the time was spent. Both timelines differ in that the year was more optimistically planned than realized. A major factor was the lack of experience of the participants concerning the conduction of a larger project with many co-workers as well as concerning the general implementation of the preprocessing stage for machine learning classification. Another factor influencing the shifted timeline was the appearance of a fifth major stage, the *Manual Labeling*.

The timetable in Figure 2.1 gives a broad outline of the major stages of the project. In the upper timeline of the figure, it was estimated how much time for a specific

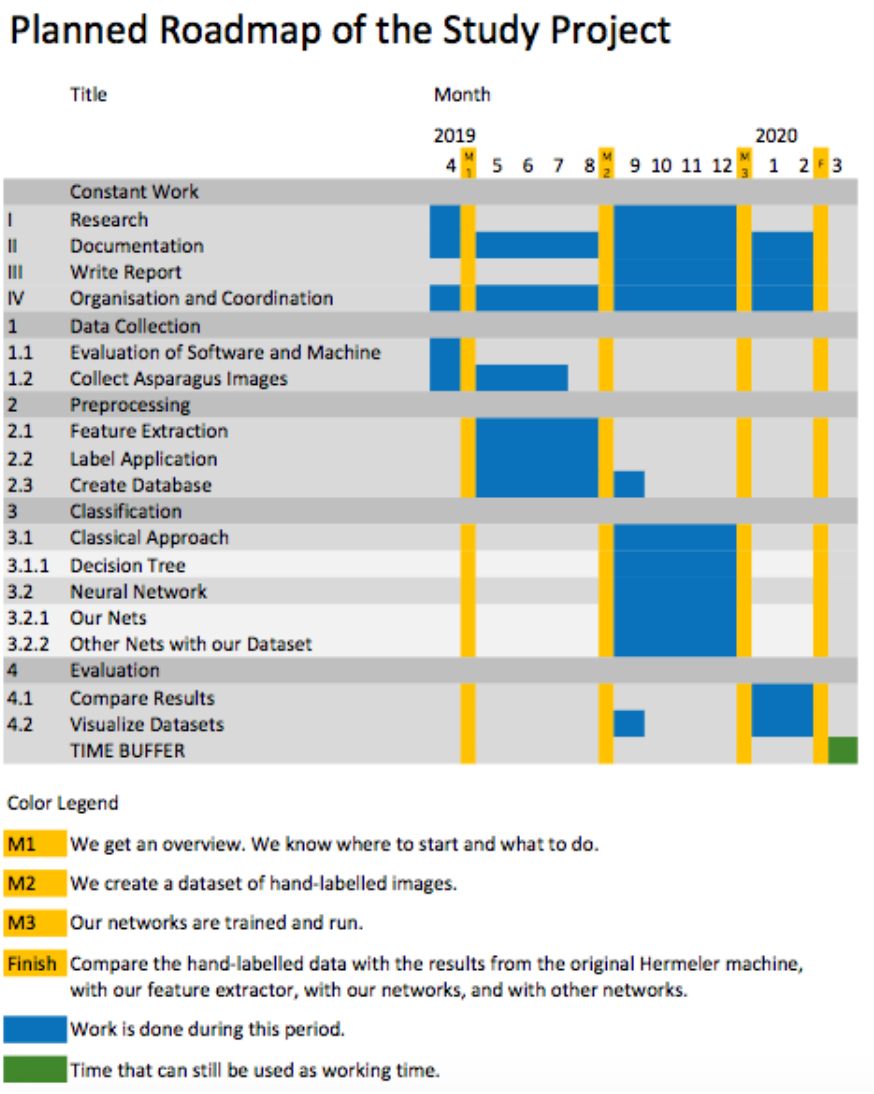


FIGURE 2.2: **Planned Roadmap** The figure shows the planned roadmap of the study project. It reveals how the time needed for each task was estimated in the beginning of the project.

phase is needed, whereas in the lower timeline the real time spent for the stage is given. Both timelines are structured to display the project year, starting in April 2019 and ending in April/Mai 2020. The months are represented by the x-axis while the colors mark the different working stages.
The project comprises four to five major stages: *Data Collection & Organization*, *Preprocessing*, *Manual Labeling*, *Classification*, and *Evaluation*. A detailed representation of the single tasks attributed to each stage can be found in the roadmaps in Figure 2.2 and Figure 2.3. The project started with the data collection and the organization of the study project. During the first stage, the images were recorded with the sorting machine, while the major planning and research for the project took place. In the second phase, most of the preprocessing happened, that is, preparing and labeling the image data. The classification stage includes the time spent on the machine learning approaches which were implemented and trained on the asparagus data. In the last stage, the approaches were evaluated and their results were compared. The different stages overlapped to a certain degree. For the purpose of this figure, the start and end time is displayed as a hard boundary.

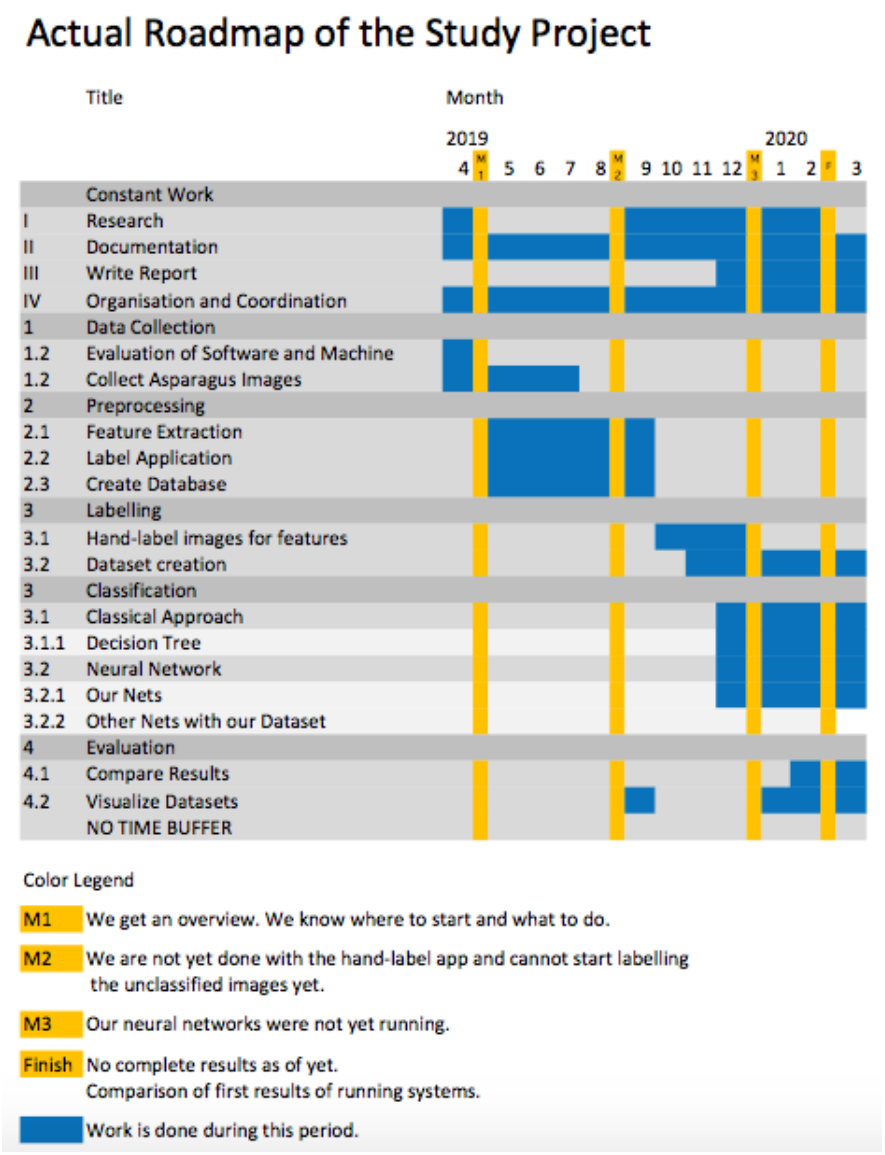


FIGURE 2.3: **Actual Roadmap** A roadmap that shows how the time was spent.

When comparing both timelines, some distinctions can be recognized. The upper timeline shows that preprocessing was estimated to be done by September. However, the phase continued until October, as can be observed in the lower timeline. Furthermore, the time for labeling a sufficient amount of images was underestimated, resulting in adjustments of the time attributed to this task. More specifically, it led to the *Manual Labeling* of the image data receiving its own phase in the timetable, independent of the preprocessing phase.
The differences are depicted with different color codings. While the main focus of this project was supposed to be the application of different machine learning techniques to classify the data (color-coded in blue and green), the preprocessing phase and the data set creation/manual labeling posed to be most time-consuming (color-coded in orange and yellow).
In Figure 2.2 and Figure 2.3, the stage specific tasks can be seen in more detail. Again, both figures display the estimated time and the actual time, respectively. The headlines serve as a division into the major stages except for the first heading, *Constant Work*, which shows the tasks that demanded continuous attention and

effort throughout the year. The duration of tasks is represented in blue, while the yellow lines mark milestones that are explained in the legends.

2.2 Organisation of the study group

In this chapter, the management of the work distribution and the communication are taken into focus. For this the tools that were used for communication and organization will be examined as well as the structure of the group work.

2.2.1 Communication

The main communication consisted of weekly meetings in which the working process was discussed and new tasks were distributed. In addition to those meetings, different platforms were used, which worked with varying degrees of success. Used platforms were Asana, GitHub and Telegram to facilitate the communication aside from group meetings. The different means of communication will be described and evaluated in the following section.

Regular meetings made up the core organization of our project. During the meetings a discussion leader and a protocol writer were picked.[1] The meetings were characterized by long discussions about how to approach the upcoming project step and how to tackle the next challenge. The project's supervisors were usually present at the meetings, to bring in their expertise and to give the opportunity to ask concrete questions. During the first half of the project, tasks were distributed at the end of each meeting. In the subsequent meeting the working progress was discussed. This procedure was changed in the second half of the project. A schedule was used that described the different tasks and deadlines in detail. Additionally, we regularly gathered for co-working. The organizational meetings were continued, during which everyone gave precise and structured reports on their area of responsibility. This helped us to spend less time discussing and have more time for task-relevant work.
The majority of important information was exchanged via Telegram [2]. Starting from the first meeting, we had a constant conversation in a group chat on Telegram, in which we informed each other about the status of the project as well as support each other by answering questions. The group chat also created space for mutual motivation when needed.
Additionally, Asana was used in the beginning of the project. The communication platform is usually used to distribute tasks and to communicate about them. Many integrations of other applications, such as Slack, can help to achieve this. However, the tasks were easier to distribute in direct consultation at physical meetings and results could be easier demonstrated or discussed. If we had relied on communication with Slack or other agreed services or applications, it might have made more sense, but Asana alone has proven to be inefficient in our use case.
During the project, it was further learned how to work with GitHub [3]. Git allowed us to work from anywhere, which facilitated the workflow.

[1] The protocols were saved for review in the GitHub project at https://github.com/CogSciUOS/asparagus

[2] Telegram is a cloud-based instant messaging service for the use on smartphones, tablets and computers.

[3] GitHub is a web-based popular platform using the version control system Git that helps developers to store and manage their code, and track and control changes to their project.

Furthermore, we were able to automatically create documentations via Sphinx. This means that by adhering to the style conventions, the protocols, work schedules, manuals, and code comments were automatically included in our documentation.[4]

2.2.2 Teamwork

This section starts by introducing the team members and their previous experiences. It is followed by a description of the practical aspects of teamwork, the working structure, and the distribution of project-relevant tasks.

The project was an initiative of one of the students. A large part of the project members knew each other in private but had not yet worked together. Further students joined the project after its public announcement to complete the team. Thus, the group consisted of members with varying degrees of knowledge about each other. The team was initially made up by Josefine Zerbe, Katharina Groß, Malin Spaniol, Maren Born, Michael Gerstenberger, Richard Ruppel, Sophia Schulze-Weddige, Luana Vaduva, Thomas Klein, and Subir Das. None of the members had yet worked together as such on a project of this scope. During the course of the project, three members left the team for various reasons. Thomas left in July due to a change in his study program. Further, Luana and Subir left in October to pursue different study projects.
The members brought a wide variety of backgrounds into the team through different bachelor programs or different majors in the broader field of Cognitive Science. In the beginning of the project, the team members had little to no experience in the application of computer vision or neural networks. The motivation of most students was to pursue new and interesting tasks in these fields. Four students had a theoretical background in computer vision, six students had gained some experience with neural networks through the course "ANNs with TensorFlow", taught at the University of Osnabrück. Some had also taken machine learning classes during their study program. Git was previously only used by three students, but none of them were experts on its usage. Further, the team had neither experience with the Grid system of the IKW, nor with running jobs on different machines. None of the members had prior knowledge about project management or task organization on a broader level.

In the beginning, the team lacked some structure and a clear distribution of individual roles. One reason for this could have been the harmonious atmosphere between team members. Further tasks such as the trips to the asparagus farm strengthened the team spirit and the social interactions. Thus, the task distribution was very dynamically structured by making every decision democratically. Most tasks were performed in smaller teams of two to three people. During meetings, possible next tasks were formulated but without assigning them to specific members or working groups. This resulted in a lot of unassigned tasks and a discontinuous workflow.
In August, the organization was restructured. On the one hand, a new structure for task distribution seemed more appropriate, instead of a democratic distribution. On the other hand, the strengths of the individual team members should be used more efficiently. Some team members had less programming experience than others. They had difficulties realizing certain tasks in an equal time period and with the same precision as others. Although they had good ideas in terms of concept, these were not implemented quickly enough to include them into the project. Nevertheless, it gave the opportunity to acquire new programming skills. To integrate

[4] see our documentation at `https://asparagus.readthedocs.io/en/latest/`

more of the strengths that the single team members brought and to tackle the issue of time management, it was decided to write a work schedule that distributed the work more appropriately, gave an overview of the tasks that still had to be done and showed how much time was left to do them.
The supervision of the work was divided into manager roles, which means that the work was split into different main fields. Each member was responsible for managing their assigned area, distributing tasks and keeping an overview of the relevant work inside their working field. The manager could be consulted for questions, when in need of discussion, or for feedback. The meetings became more effective due to the new structure, and there was less discussion concerning task distribution.
Further, common working hours on campus were introduced. The common working hours ensured that questions and decisions that arose could be discussed in person. This was especially helpful when different tasks overlapped and required communication and agreement.

In conclusion, the team structure and the distribution of work changed over the course of the project. The strengths of single members were used more efficiently and the supervision of working areas led to a more structured time management and task distribution.

2.3 Data collection

In this section the asparagus sorting machine at Gut Holsterfeld is described. Then the process of collecting labeled and unlabeled data is reported.
The machine Autoselect ATS II (2003) is designed for sorting white and green asparagus (see Figure 2.5) (HMF Hermeler Maschinenbau GmbH, 2015). The asparagus is arranged on a conveyor belt that runs it through the recording section of the machine. Here, a camera takes three pictures per asparagus spear (see Figure 2.4 and Figure 2.7). Small wheels on the conveyor belt rotate the asparagus in the meantime so that it can be photographed from several positions. In the best case, on each image a different side of the asparagus is recorded. The conveyor belt transports the spear further and it is sorted into a tray depending on the chosen class label by the machine. The sorting is based on the parameters for width, length, shape, curvature, rust, and color. A total of 30 criteria for classifying an asparagus spear are used to describe these parameters. The calculation of the single features is based on a classical analytical approach. For example, the parameter for color detection is composed of eight sub-parameters. Each spear is reviewed at different areas (the head of the asparagus, the area below the head, and the stem) and judged for its hue in percentage. The values are compared and, according to a threshold, the spear is sorted into a color category (e.g. white or violet). For all parameters, there is a minimal threshold and a maximal threshold. As another example, the parameter for width detection calculates at three points at the asparagus (top, middle, and bottom part). From these three values, an average value is calculated that decides in which category the asparagus is sorted. If an asparagus exceeds the maximal threshold for parameter detection, it is not recognized and cannot be sorted accordingly. The same holds for values below the minimal threshold. Thus, all parameters have to have an upper and a lower threshold, including parameters that decide the presence of features like shape, curvature, and color. When evaluating what parameter boundaries to choose, it is recommended to check that most asparagus spears tend to be in between the average value and the maximal threshold, with a larger tendency to accumulate around the average

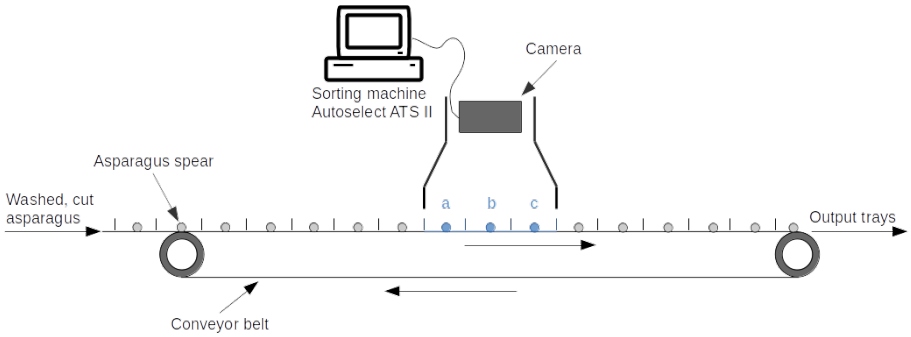


FIGURE 2.4: **Sketch of the Image Capturing Process with the Autoselect ATS II** The asparagus spears are transported on a conveyor belt. After being washed and cut, the spears pass the camera field. Images are taken of three compartments, so that each asparagus spear is photographed three times in each position (a, b, c). The camera system is connected to a computer on which the sorting software runs. Depending on the resulting classification, the spear is sorted into the corresponding output tray.

FIGURE 2.5: **Autoselect ATS II at Gut Holsterfeld** In the figure the asparagus sorting machine Autoselect ATS II at Gut Holsterfeld can be seen. The conveyor belt transports the asparagus from the left side of the image to the right side. It thereby passes the camera system. The display of the machine gives information on the parameters and the images. The machine is mainly controlled from here.

value. Reportedly, the parameters and their respective ranges can be freely chosen by the user and can in this way be fitted to the needs of the respective asparagus farm (HMF Hermeler Maschinenbau GmbH, 2015).
Before the first use of the machine, all parameters are selected after a calibrating charge of asparagus has run through the machine. Then, the user can adjust the thresholds accordingly.
According to the manual, the number of quality classes is selectable. The user can define the order of quality classes by choosing the arrangement of parameters. The manufacturer suggests to first sort for length and width, then use the parameters

FIGURE 2.6: **Output Trays of the Sorting Machine** Depending on the class label, the asparagus is sorted into one of the machine's 16 trays.

that sort for color, and the parameters for shape detection last.
The accuracy of the sorting machine is described to be as good as 90% best case by the manufacturer, while the farmer at Gut Holsterfeld reported it to be around 70% at best, with re-sorting being necessary by professional sorters. Especially categories like Blume or Hohle were considered to be inconsistent by both, manufacturer and farmer. Further information could not be given on the software of the machine. A meeting with a representative of the engineering company HMF [5] that manufactured the sorting machine was arranged. Unfortunately, the source code itself was not available to HMF as it was produced by another company.

In the following it is described how the image data was collected. It is possible to save images with the Autoselect ATS II, however, the storage space on the machine is very limited. Further, the selection of images to be saved is restricted to only 1000 images. One workaround to the problem is the installation of the Teamviewer software [6] on the machine and the connection of an external hard drive. After the installation, the process of image collection could be started remotely. This work was very ineffective and time-consuming. The data could not be directly transmitted to another computer because the internet speed at the farm is too slow. An automatic transfer of the images to the external hard drive was not possible until the installation of an automatic file moving service, for which the requirements are described below.
The file moving program needs to transfer the images to a new saving destination and has to run in the background without disturbing the workflow of the sorting

[5] see www.hmf-hermeler.de
[6] see https://www.teamviewer.com/en/

machine. After research on background processes and programs, the decision was made to use a service, that is, a system process running independently of any program. The service manages moving the newly generated image files, as described in detail in the appendix in subsection B.1.1.
The project members split in groups of two and exchanged the hard drive two times a week. The collected images were then transferred to storage capacities of the university.
The label that the machine attributes to each asparagus is not reliable. Therefore, sending the asparagus through the machine a second time would be the only way to gather labeled images. Unfortunately, a second sorting is not good for the quality of the asparagus. Further, at least one project member has to be involved in the re-sorting and, thus, has to be present at the farm. The sessions of exchanging the external hard drive and collecting labeled image data were combined.

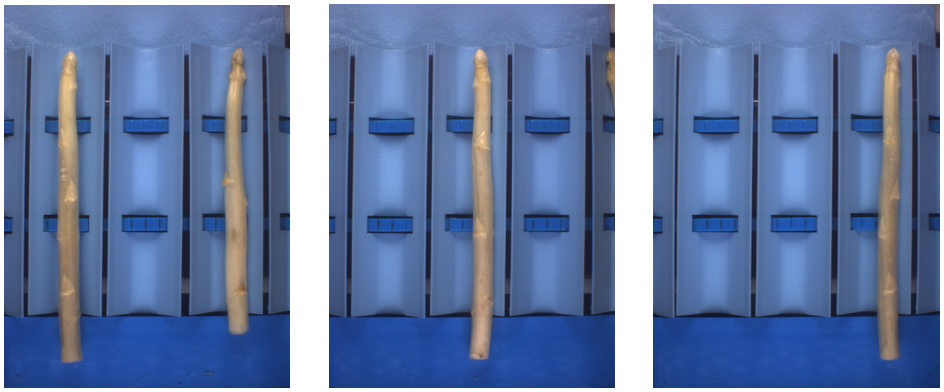
(A) left (B) middle (C) right

FIGURE 2.7: **Example Asparagus Images** Example pictures of the quality class I A Anna. The asparagus is arranged on a conveyor belt that runs it through the recording section of the machine, where a camera takes three pictures. In picture (A) the target asparagus is to the left, in (B) it is in the middle, and in (C) it is to the right. Small wheels on the conveyor belt rotate the asparagus in the meantime so that it can be photographed from several positions. In the best case, on each image a different side of the asparagus is recorded. The conveyor belt transports the asparagus further and it is sorted into a tray depending on the chosen quality class by the machine.

An example image of the received data can be seen in Figure 2.7. There are three pictures per asparagus. The image resolution is 1040×1376 pixel per image, with an RGB color space.
In total, 612113 images were collected, with each class label being represented with at least 309 images, corresponding to 103 asparagus.
At the asparagus farm Gut Holsterfeld, 591495 labeled and unlabeled images were collected with the Autoselect ATS II. The number of unlabeled data is 578226 images, thus, around 192742 different asparagus spears. Of the labeled data, the number of images that were collected per quality class can be found in Table 2.1. The image number does not represent the number of different asparagus spears, as each asparagus spear is represented by three distinct images.

Class Label	Nr. of Images	Class Label	Nr. of Images	Class Label	Nr. of Images
I A Anna	1005	II A	1051	Blume	1717
I A Bona	908	II B	1468	Suppe	1157
I A Clara	513	Rost	1169	Bruch	309
I A Krumme	936	Dicke	749		
I A Violett	1514	Hohle	775		

TABLE 2.1: **Collected Images with Class Label**
In this table, the number of collected images with a class label is reported. This was achieved by running pre-sorted spears a second time through the sorting machine



(A) left (B) middle (C) right

(D) upper part

FIGURE 2.8: **Example Images Querdel's Hof** Example pictures of the asparagus of the farm Querdel's Hof. In picture (A), the asparagus spear is to the left, in (B), it is in the middle, and in (C) it is to the right. A fourth picture (D) is taken with a second camera with focus on the upper part of the asparagus.

Additionally, a few images could be recorded at another asparagus farm, Querdel's Hof [7], in Emsbüren. The farm sorts the asparagus with an updated version of the Autoselect ATS II at Gut Holsterfeld, that is, it uses the same software but other hardware. In particular the resolution of the camera was improved and a second camera was installed that focuses on the head region of the asparagus. At Querdel's Hof, 20616 images were collected in total, 76 from the class label "normal", 152 from the class label "violet/flower", and 20388 unlabeled images. Each asparagus spear

[7] see https://www.querdel.de/

is represented by four images: three images show the asparagus from different perspectives and a fourth image depicts solely the head region. Example images for one asparagus can be seen in Figure 2.8. No internet connection could be established to the farm, thus, no further images were collected. Moreover, the data format of the images from Querdel's Hof is different to the data from the farm Gut Holsterfeld, due to the additional head image. Therefore a combination of both data sets was not convenient.

2.4 Literature on food classification using computer vision

In the classification of food products, there are numerous possibilities to apply classical machine learning and ANN approaches for classification tasks on image data (Bhargava and Bansal, 2018; Brosnan and Sun, 2002).
For the scope of this investigation, we decided to focus our literature search on fruit and vegetable quality evaluation using computer vision and machine learning. Compared to other fields, research and evaluation in agricultural classification shares many characteristics and faces similar difficulties.
The quality inspection based on computer vision is usually constituted into five main steps: image acquisition, preprocessing, image segmentation, feature extraction and classification (Bhargava and Bansal, 2018). Moreover, most data in agriculture is based on photographic images. Also the features of interest are similar for different kinds of fruit or vegetable. Frequently by traditional computer vision techniques inspected features concern color, shape, size, texture, and defect (Bhargava and Bansal, 2018). This makes other papers in the field of agricultural evaluation directly comparable to our case. Moreover, we hope to get an impression of the state of the art of how many images are needed in our case, how high the image resolution needs to be, what kind of computer vision approaches could be helpful as a starting point, and also to become aware of known challenges.

None of the found papers were suitable as blueprints for the asparagus classification project. However, some of them helped to get an idea of how to proceed with the project. For example, some papers show how the preprocessing phase could be structured (Mery, Pedreschi, and Soto, 2013), or they evaluate the machine learning methods that were already used on other food classification tasks (Bhargava and Bansal, 2018). Further, some of the literature is concerned with the classification of food products but not with differentiating between as many classes as 13 (Diaz et al., 2004; Kılıç et al., 2007). Often, the variance in the food products, that is, the quality as well as the type of food used is either too high (Zhang and Wu, 2012) or too low (Kılıç et al., 2007; Al Ohali, 2011) in comparison to the variance in our project data. One paper evaluates the sorting of asparagus, however, it only does so on a small data set with three categories of green asparagus (Donis-González and Guyer, 2016). Further papers on food classification are not detailed enough in their explanations and do not share the information needed for replication (Pedreschi, Mery, and Marique, 2016). Another paper is mainly about the implementation of a certain toolbox (Mery, Pedreschi, and Soto, 2013).
Even though no specific paper was used as guidance to our project, some specific papers inspires us to try out certain algorithms, such as PCA (Vijayarekha, 2008; Zhu et al., 2007)) or neural networks (Jhuria, Kumar, and Borse, 2013; Pujari, Yakkundimath, and Byadgi, 2014). Moreover, the literature review made us aware of the limiting fact that images of fruits and vegetables are captured mainly from one direction (Bhargava and Bansal, 2018).The literature suggests that performance

might improve, if more perspectives are taken into account. Moreover, the literature shows that different authors use different color spaces such as CIE Lab, RGB or HSI (Liming and Yanchao, 2010; Garrido-Novell et al., 2012; Kondo, 2010). This further inspired us to apply color quantization on our data.

As the available data was only sparsely labeled, further research was done to evaluate the use of a semi-supervised learning approach (Olivier, Bernhard, and Alexander, 2006; Zhu, 2005). Details about the corresponding literature can be found in section 4.3. In regards to deep learning-based approaches, classical neural networks – such as AlexNet (Krizhevsky, Sutskever, and Hinton, 2012), VGG16/VGG19 (Simonyan and Zisserman, 2014), GoogleNet (Szegedy et al., 2015), Capsule Networks (Sabour, Frosst, and Hinton, 2017), DenseNet (Huang et al., 2017), ResNet (He et al., 2016b) or Network in Network (NIN) (Lin, Chen, and Yan, 2013) – were assessed for better understanding of the range of possible pre-trained networks and ideas for network structures. Also, classical computer vision approaches were considered, like multiclass Support Vector Machines (SVMs) (Prakash et al., 2012).

Chapter 3

Preprocessing and data set creation

In this chapter, the different preparatory steps for the recorded data are described, including the creation of a data set which is usable for any machine learning or computer vision approach to analyze the image data.
In 3.1 Preprocessing the data is assessed and simplified for any further processing. The second section, 3.2 Feature extraction, deals with the creation of feature scripts that were researched and implemented for an automatic recognition of single features. The results were combined in an application which is described in detail in 3.3 The hand-label app: A GUI for labeling asparagus. In 3.4 Manual labeling, the process of hand-labeling the images for their feature class with the label application is described, followed by a section analyzing the results and comparing the overall agreement of the labelers. The last section 3.5 The asparagus data set concludes with the creation of the final data set, used for the later training of the neural networks and other approaches to detect the label of a spear from its three images.

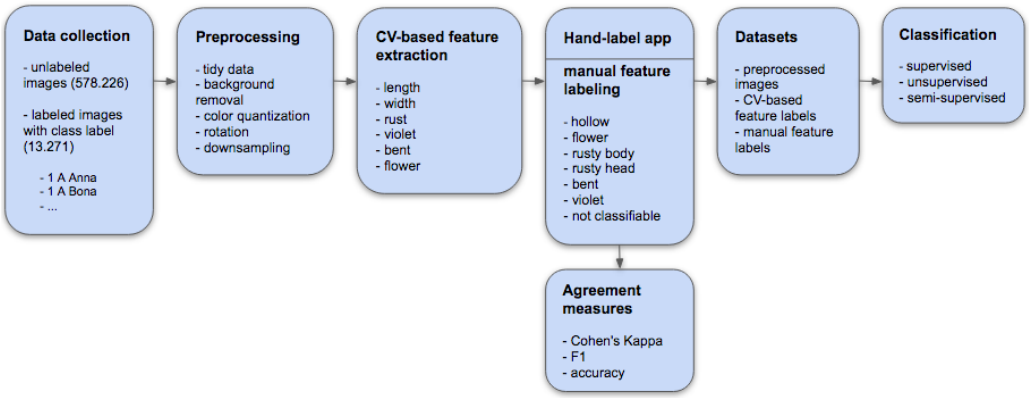


FIGURE 3.1: **Working Steps from Data Collection to Classification** All in all, 578226 unlabeled images were collected. Additionally, 13271 images with a class label were collected (see Table 2.1 for the number of images per class label). The collected images go through different preprocessing steps. The preprocessed images are then taken as input to the computer vision based feature extraction algorithms. The preprocessed images and the computer vision based, extracted measures are taken as input to the hand-label app. By the help of the application, features are manually labeled by seven annotators. Agreement measures are calculated to compare agreement across annotators. The preprocessed images, plus the computer vision based features, plus the manual labels are taken to create different datasets, which are further used for the different classification approaches.

3.1 Preprocessing

Before implementing any approach that allows to predict a label to an asparagus spear, the recorded image data has to go through multiple preprocessing steps.
The goal of this preprocessing is to reduce the variance by removing the background, shifting the asparagus spear to the center of the image patch and rotating it upwards. Correct orientation is especially important to make approaches such as PCA applicable and facilitates direct measuring of features. Background removal facilitates the measurement of features like width and height of the spears, because the position of foreground pixels per column can be easily evaluated (see section 3.2).
In the following, the different preprocessing steps are elaborated in detail.[1]

As described in section 2.3, each asparagus can be found in three pictures, one in each of the three positions – left, center and right. The image names are used to find the three relevant images and determine in which position the asparagus is captured. The images are cut into three pieces and renamed in a way that makes clear which images belong together. Each asparagus gets a unique identification number and the three perspectives are denoted with textita for left, textitb for center, and textitc for right. For example, the textttimage 42_b.png is the center image of the asparagus spear with the identification number 42.
Another step is to remove the background of the image. As the conveyor belt is blue, there is a high contrast to the bright asparagus spears which facilitates background removal (see Figure 3.3). Hence, it is possible to mask the asparagus spear using the hue of the HSV representation of each image. All pixels with a blue hue and very dark regions are marked as background through threshold limitation of the value component. This is particularly important for the automatic feature extraction (see section 3.2).

Subsequently, each triple of images that depict one piece from different angles is determined and the area that shows the correct piece is cropped. The cropping regions are set to three patches that are a little larger than the compartments of the conveyor belt. They are located such that they cover the spear of interest which can either be at the left, center or right (see Figure 2.7). As it has been found that some tilted spears span across the borders between the conveyor compartments, the cropping window in the images was moved. The new coordinates are determined by shifting the center of the current cropping box horizontally to the center of mass of the contained foreground pixels (see Figure 3.3). Repeating the procedure in a second iteration can further improve the result. As this is the case for few examples only we refrained from doing so to increase processing speed. Small parts of the neighboring depiction potentially end up in the region of interest as well. These remainders are removed by masking out all pixels that do not belong to the main blob of connected pixels.

To further reduce the variance the asparagus spears are rotated upwards to reduce the variance in angle. The rotation angle is achieved by binarizing the image into foreground and background pixels, calculating the centerline as the mean pixel location along the vertical axis, and fitting linear regression to the centerline pixels.

Another preprocessing step, which generates an additional set of images, is quantization using a common color palette. It was mainly employed to allow for the

[1] See https://github.com/CogSciUOS/asparagus/blob/FinalProject/preprocessing/perform_preprocessing.py

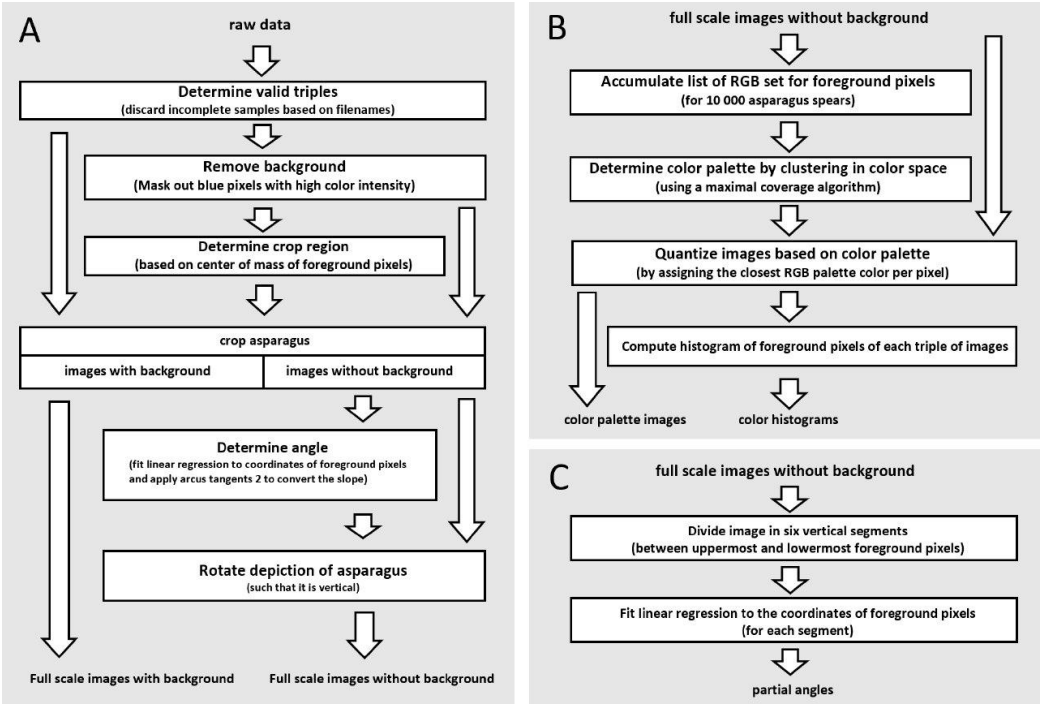


FIGURE 3.2: **Preprocessing Pipeline** The depiction shows the preprocessing pipeline that was used to generate different datasets. For each asparagus spear three images were saved. Downscaled versions were computed for each image dataset. A: The processing steps to retrieve full-scale images with and without background. B: The approach to retrieve color palette images and color histograms by feature engineering. C: Processing steps for the retrieval of partial angles. For details on B and C see also subsection 4.1.1.

computation of meaningful color histograms.[2] An appropriate palette and the respective mapping of RGB values to palette colors is determined using clustering in color space. First, a set of RGB tuples is collected by adding pixel values of 10000 asparagus spears. Second, the resulting list of RGB tuples is converted to an image such that a palette can be determined using standard tools for quantization. In the last step a clustering algorithm is employed that determines the position of cluster centers while maximizing the maximal coverage. The resulting cluster centers can be displayed as a list of RGB values which represent the color palette.[3] The color palette is used for quantization of the images: Each image of the downscaled data set is transformed to the palette representation. Visual inspection shows little quality loss such that it can be assumed that the relevant information for image classification is well preserved.

Several additional collections of preprocessed images are computed based on the

[2]The number of colors in the original 24 bit RGB colorspace is too large to use them as bins for the histograms: The low number of pixels per bin would not allow for meaningful statistics. By reducing the number of colors the problem is solved.

[3]Here we used a standard implementation for image quantization that employs a maximal coverage algorithm (Python Image Library, 2020). An optimal solution to the problem of maximum coverage relates to the challenge of distributing a given number of circular areas named facilities such that they cover the largest possible area of the sample space (Zarandi, Davari, and Sisakht, 2011). One can interpret the centers of named facilities as cluster centers. For each data point (here: pixel), the closest cluster center is determined and the respective value attributed. This means the data is quantized.

FIGURE 3.3: **Cropping Based on Center of Mass** The depiction shows the effect of moving the cropping window (red) to a new location (blue) by moving it's center horizontally towards the center of gravity (yellow arrow). The buckets on the conveyor belt are illustrated by white boxes. The approach is applicable for all three images given that the initial coordinates are set to the respective positions. For unprocessed images see Figure 2.7.

data without background. This holds for downscaled versions as well as for a version that contains the asparagus heads only. To compute the latter, the images are padded to avoid sampling outside of the valid image boundaries and the uppermost foreground row is detected. Subsequently, the center pixel is determined and the image is cropped such that this uppermost central pixel of the asparagus is the center of the uppermost row of the snippet. The resulting partial images of asparagus heads are rotated using the centerline regression approach described above. The approach has proven reliable and the resulting depictions are used to train a dedicated network for head related features (see subsection 4.1.4).

3.2 Feature extraction

The class label of an asparagus spear is decided by several features, ranging from its shape to its color. Put together, these single features give the class label that attributes an asparagus to a quality class (for a description of the quality classes, see Table 1.1, and further Figure 1.1 on how features lead to the class label).

We decided to label the images for their features rather than final class labels. The main reason for this was to make the labeling process easier for the inexperienced annotator. The boundaries between class labels are not always clear and can be difficult to detect from image data. Deciding whether a single feature is present or absent in an asparagus spear is more straightforward. Further, the training for the hand labeling and communication about special cases is facilitated. Another reason to label features rather than class labels was to break down the classification problem into smaller problems. Additionally, it is possible to detect which features are more difficult to learn than others, which provides meaningful insight into the classification task. Last but not least, deciding on the class labels after the features are detected reliably is a small step and can be easily done with a decision tree or rule-based approach.
Even though the chosen features closely resemble the class labels defined by Gut Holsterfeld, they are different and should not be confused with one another. The 6 features we labeled by hand are as follows: hollow, flower, rusty body, rusty head, bent and violet. Additionally, the length and width was automatically detected as described below and used to set supplementary labels for very thick, medium thick, thick, thin, very thin and for fractured. Further, images that could not be classified thoroughly were labeled as "not classifiable" (e.g. when the spear is cut off the image).

In this chapter, the different features as well as their extraction methods will be described. The results that are achieved by computationally extracting the features are reported alongside future steps that could be taken to improve the results further. For each feature detection method, the images with removed background are used as is displayed in a respective example image shown per feature. Additionally, it is described which features were hand labeled by us. The feature functions, that provide reliable predictions, were integrated into an application, which is described in the subsequent section 3.3, with which human annotators could manually label the unlabeled data.

3.2.1 Length

The length detection described in the following paragraph was later used to automatically calculate the presence of the feature fractured in an image. An asparagus spear includes the feature fractured if it is broken or if it does in any other way not fulfill the required, minimal length of 210 mm (see Figure 3.4).

The length detection uses a pixel-based approach. It counts the number of rows from the highest to the lowest pixel that is not a black background pixel and therefore not zero. The asparagus is rotated upwards, as described in section 3.1. This is done to improve the results, as the rows between the highest and the lowest pixel are counted and not the pixels themselves. This technique is a simplification, which does not represent curved asparagus very well, because it will have a shorter count than it would have if the pixels were counted along the asparagus spear.

However in reality, there are not a lot of asparagus spears close to the decision boundary between a fractured spear and a whole spear. Usually, the asparagus is harvested a few centimeters longer than necessary and then cut to the desired length. The only asparagus shorter than that length are the ones that break during the sorting process. Moreover, if they break, they generally break closer to the center of the asparagus rather than at the ends. Therefore, the difference in length detection does not matter for our classification.

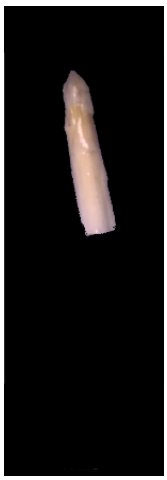


FIGURE 3.4: **Feature Fractured**

All in all, by visual inspection the length detection yields good results that are very helpful for the hand-label app. The next step would be to train a decision boundary that determines which number of pixels should be the threshold to differentiate between fractured and not fractured. At first, we tried to calculate this threshold by finding a conversion factor from pixel to millimeter, as we know the cut off in millimeters. But this approach appeared to be more difficult than anticipated, because the conversion factor varies in the different image positions. This problem only became apparent after the asparagus season had ended, for which reason we could not reproduce the camera calibrations in retrospective in order to take well-measured images, for example from a chessboard pattern. Accordingly, the threshold needs to be deduced from the data manually or learned with a machine learning approach.

3.2.2 Width

Like the length of a spear, thickness is a feature that is hardly recognizable by view alone. Fortunately, it could also be automatically extracted with a classical approach described in the following section.

The division into different ranges of width can be inferred by the overall thickness of the spear. The feature 'very thick' is attributed to asparagus that is more than 26 mm in width. The feature 'thick' corresponds to 20 – 26 mm, the feature 'medium thick' to 18 – 20 mm, and the feature 'thin' to 16 – 18 mm. Every asparagus with less than 16 mm in width is described with the feature 'very thin'.

FIGURE 3.5: **Feature Not Classifiable**
Example image for the feature not classifiable. Further, a difference in width of the asparagus is observable.

The width detection uses a very similar approach as the length detection. It takes the pixel count from the left-most to the right-most pixel in a certain row as a width measure. But in contrast to the length, the width was measured at several image rows from which the mean width was taken. Since the width detection works reliably, it is integrated in the hand-label app (see section 3.3).
The algorithm operates as follows: Firstly, the images are binarized into foreground and background,

which means setting all pixels that are not zero, and therefore not background, to one. After that, the uppermost foreground pixel is detected and the length is calculated with the length detection function as described above. The length of the asparagus is used to divide it into even parts. This is done by determining a start pixel and dividing the remaining rows that contain foreground pixels by the number of positions one wants to measure at. This way several rows are selected in which the number of foreground pixels is counted. One can interpret each row as a cross-section of the asparagus, therefore the number of foreground pixels is a direct measure for the width. Then, the mean of these counts is calculated and used as the final width value. As the head of the asparagus can be of varying form and does not represent the width of the whole asparagus well, it is excluded from the measure. This is done by selecting a start pixel below the head area instead of naively choosing the uppermost pixel. To be precise, the start pixel is chosen 200 pixels, which corresponds to roughly 25 mm, below the uppermost pixel in order to bypass the head area with certainty. As described in the section Length, also the width detection might lead to slightly different outcomes on curved asparagus spears than the true values. Again, this difference is regarded as irrelevant in our case.

3.2.3 Rust

The feature rust is split into the (sub-) features rusty body (see Figure 3.6) and rusty head (see Figure 3.7), because in the case of rust being only on the body, it is removable by peeling. Rust at the top part of the spear cannot be removed without damaging the head. Thus, rust on the head region is a decisive factor for the quality and later categorization into a price class.
If a spear has rust, it is visible as a dark brown color. It often starts at the tips of becoming leaves or at the bottom part. The color is not to be confused with bruises, pressure marks, or a slightly yellow complexion, which can occur in a ripe asparagus. The latter coloring is neglected. Rust is set to be present even when only the tip of a leaf shows a dark spot. Other brownish bruises are not classified as rust.
If there is a rusty spot on, or close, to the head region, it is captured by the feature rusty head. The head part is usually distinct in shape and color from the body part of the asparagus. In principle, the same guidelines that apply to the feature rusty body also apply to the feature rusty head, except for an explicit focus on the top part (until around 1 cm below the collar) of the asparagus. Any rust below this is labeled as the feature rusty body.

FIGURE 3.6: **Feature Rusty Body**

The automatic rust detection finds all pixels that fall in the range of RGB values that correspond to the color brown. Those pixels are counted and normalized by the maximal number of possible pixels that could be rusty, namely the number of foreground pixels. Since it is impossible that the whole asparagus is rusty and hence that all the foreground pixels fall into the relevant range of RGB values, this normalization yields small numbers as results. To give an example, an output value of 0.13 is already considered moderately rusty. The lower and upper bound for the RGB values are set to $[50, 42, 30]$ and $[220, 220, 55]$, respectively. That means, all pixels that have a red value between 50 and 220, a

green value between 42 and 220, and a blue value between 30 and 55 are considered to be rust.

Visual inspection shows that the rust detection algorithm works well to detect rusty areas and barely misses any rusty parts. The difficulty lies in setting a threshold for the number of pixels needed to be classified as rusty. Only clusters of brown pixels are reliable indicators for rust. Many pixels with a brown color distributed over the whole spear are not supposed to be classified as rust. It might be the case that a simple pixel count is not sufficient to set a classification threshold. More sophisticated approaches to detect clusters, such as morphological operators, could be beneficial for this feature detection. It remains unsolved to set a robust threshold that works well on the whole data set.

FIGURE 3.7: **Feature Rusty Head**

One problem that cannot be solved algorithmically is dirt in the sorting machine. If the machine is not cleaned thoroughly and regularly, dirt can be falsely classified as rust because it often falls in the same color range. Another problem can be a change of lighting when taking the images. Both issues can be controlled for, but have to be communicated well to the farmers.

3.2.4 Violet

The feature violet indicates whether an asparagus is of violet color. The shift of color from white to violet occurs most often around the head region – either at the tip of the head or just below the collar of the head region (see Figure 3.8).
As the spear will darken further after it is harvested, even a slightly pink spear is labeled as being violet.

According to the UNECE-norm asparagus of the highest quality grade may only be marginally violet or not violet at all (United Nations Economic Commission for Europe (UNECE), 2017). Hence it is crucial to sort asparagus pieces according to this binary attribute. In a simple procedure color hues are evaluated. More precisely, this strategy is based on evaluating histograms of color hues that are calculated for foreground pixels of the asparagus images after converting them to the HSV color space. Pale pixels are removed from the selection by thresholding based on the value component of the HSV representation. Finding the optimal threshold has proven difficult because of named subjectivity in color perception. A threshold of 0.3 for the value component is considered a good compromise: If applied, white and slightly rose pixels are masked out. All three perspectives are taken into account to compute a single histogram per asparagus spear. A score is calculated by summing up the number of pixels that lie within the violet color range. A second threshold is used as the decision boundary for violet

FIGURE 3.8: **Feature Violet**

detection. The direct and intuitive feedback in the hand-label app showed the relation between varying thresholds and the prediction. It could be seen that lowering the threshold also means that the feature extractor becomes more sensitive at the price of a reduced specificity. Best overall matches (accuracies) with the subjective perception are found for very low thresholds. In many cases, however, measurements based on this definition of violet do not match the feature label attributed by human coders.
Hence, another sparse descriptor is derived from the input images. Instead of setting thresholds for pale values and calculating the histograms of color hues, this approach relies directly on the colors that are present in the depiction of an asparagus spear. As the 24 bit representations contain a vast amount of color information in relation to the number of pixels, it is, however, unfeasible to use these as input. Instead, the color palette images can be used. Histograms of palette images can serve as the basis to define the feature violet in a way that captures more of the initial color information. At the same time it is simple and understandable enough to allow for customizations by users of sorting algorithms or machines. As a consensus regarding such an explicit definition is hard to achieve and somewhat arbitrary, the descriptor is used to learn implicit definitions of the feature through examples (see subsection 4.1.1).

The lack of a formal definition for violet asparagus spears has proven to be a major challenge to approaches of measuring this feature. It has been shown that directly measuring whether an asparagus spear is violet heavily depends on the definition of this feature. It is to mention that color impression is highly subjective across and even within subjects (Luo, 2000). Effects of meta contrast that make minor variations in color more visible arguably affected the attribution of labels when many similar spears were assessed in succession (Reeves, 1981). Using machine learning can help to find the definition that generalizes best over varying color perceptions and retrieve objective rules to measure the degree to which an asparagus is violet. In other words, the task of establishing a rule that is a good compromise for several human attributors is shifted to an optimization algorithm. Hence, machine learning approaches that are trained on human labeled data appear to be more promising.
The automatic detection of the feature violet was integrated into the hand-label app as a helper function for the human annotators.

3.2.5 Curvature

The curvature score of an asparagus image is expected to automatically detect the presence of the feature bent. The function was used as a help to the human annotators during the manual labeling with the hand-label app.

An asparagus is categorized as having the feature bent, if the shape of the asparagus is curved and not straight (see Figure 3.9).
If it is only slightly curved but can otherwise be thought of as straight – that means fitting next to other straight spears without standing out – it is labeled as straight. If the spear looks close to the same on all three pictures regarding its shape, it might indicate that it is heavily bent and therefore cannot be turned on the machine's conveyor belt.
The feature bent has a broader range of shapes where the asparagus is not obviously deformed but also not completely straight. An exception holds for S-shaped spears, which always count as bent.

In the following, the process for implementing an automatic function for curvature detection is described. Multiple curvature scores can easily be computed based on regression fits to the centerline of an asparagus spear. For example, the parameters of linear or polynomial regression can be interpreted as a description of how bent an asparagus spear is.

FIGURE 3.9: **Feature Bent**

Deriving sparse descriptions is based on a two-stage approach. In the first stage, the centerline of an asparagus spear is computed because it is considered to be a good description of the curvature of asparagus spears. In each image the asparagus spear is roughly vertically oriented. This means that also for bent spears the head relies within the top center of the image (see section 3.1). The centerline is computed by binarizing the image into foreground and background and computing the mean of pixel locations along the vertical axis (i.e. for each row). The resulting binary representation shows a single pixel line. It serves as the input to the second stage of curvature estimation.
In the second stage, curves are fit to the pixel locations of the centerline. For a simple score, linear regression is employed and the sum of squared errors is thresholded and interpreted as a curvature score. This score is small for perfectly straight asparagus spears and increases the more bent an asparagus is. As an S-shaped asparagus is arguably perceived as bent even when the overall deviations from the center line are small, a second descriptor was computed as the ratio between the error of a linear fit and polynomial regression of degree three. Thresholding values and employing a voting scheme for the results for all three perspectives yields a rule to measure curvature (e.g. at least one of the three perspectives indicates that the asparagus is bent). However, it has again proven difficult to set thresholds appropriately to reliably capture the visual impression. Hence, another sparse representation was calculated by dividing the spears into several segments and fitting linear regression to each segment. A Multilayer Perceptron (MLP) was trained on the resulting 18 angles per asparagus (see subsection 4.1.1).

Calculating a score for curvature is fast and efficient. While the respective approach is suitable to define curvature, it does not necessarily meet up with the subjective perception of how bent an asparagus appears. Just like histograms of palette images, curvature scores are the results of feature engineering: The use of extensive domain knowledge to filter relevant features (Zheng and Casari, 2018). They can serve as an input to a machine learning approach that maps this sparse representation to the target categories (see subsection 4.1.1).

3.2.6 Flower

An asparagus is graded as having the feature flower when the bud shape of the head is more developed, as in Figure 3.10.
When a bud is in full bloom, it is clearly visible. However, it can be quite difficult to distinguish between an asparagus with clearly cut but closed petals and an asparagus that has just begun to develop a flower. It was decided to label the feature as absent when the asparagus does not clearly show the characteristic flower. With this decision, we aimed to reduce the correspondence error between annotators.

The implementation of the flower detection function turned out to be difficult to realize. Several approaches have been tested, but none of them generated sufficiently good results. Two main notions were tried. The first approach uses the shape of the head as an indicator for a flower. The idea is that asparagus spears with a flowery head exhibit a less closed head shape. In other words, the head looks less round and has no smooth outline, but shows fringes. The second approach focuses on the structure within the head. Supposedly, asparagus with flowery heads exhibit more edges and lines in the head area. In both cases, it is challenging to find a way to discriminate between asparagus with and without flowery heads. One reason for that is the poor resolution of the camera that is installed in the sorting machine. With a pixel to millimeter ratio of around four to one[4], it is even difficult to detect flowers with the human eye. Likewise, the current software in the machine struggles greatly with the classification of this feature as well.

FIGURE 3.10: **Feature Flower**

3.2.7 Hollow

It was not possible to us to implement the feature hollow in an automatic, classical computer vision approach. Thus, only labels from the human annotators are available for this feature.

The feature hollow indicates if the spear has a cavity inside. This might be expressed by a bulgy center and a line running vertically along the spear's body. Another, more distinct indicator is when the asparagus looks like two spears fused together, forming a single asparagus (see Figure 3.11). A hollow asparagus can be confused with a very thick asparagus.
The feature can be easily checked when you have physical access to the asparagus. If the asparagus is actually hollow, it will have a hole at its bottom that is noticeable when turning the spear around. Unfortunately, this cannot be done when only looking at the spears from the side. The feature hollow sometimes occurs without showing a clear line or obvious bulge at its center. Therefore, there is a high risk of wrong classification.

FIGURE 3.11: **Feature Hollow**

3.2.8 Not classifiable

The feature not classifiable is no feature to an asparagus per se. It is therefore not implemented as an automatic feature extraction approach. However, it is integrated into the hand-label application and can be selected by the human annotators if applicable.

[4]The ratio was not calculated by us but is an information provided by the manufacturer (HMF Hermeler Maschinenbau GmbH, 2015).

That is, whenever the spear is unrecognizable, the head part of the spear was severed, two spears were present in one picture (as in Figure 3.5), the spear was cut off by the image, or other unusual circumstances occurred, it falls into the category of being unclassifiable.

3.3 The hand-label app: A GUI for labeling asparagus

The previous section shows that some features are reliably measurable by means of classical computer vision. This holds especially for simple features such as length and width of an asparagus but to a certain degree also for curvature. In contrast, for features that relate to small details such as the feature flower direct measurements have proven to be difficult (see subsection 3.2.6). The application of machine learning for measuring these features appears as a promising alternative. However, the major part of our dataset consists of unlabeled data. Considering the amount of data that could potentially be labeled, a custom interface is required that allows for a time efficient attribution of labels.

3.3.1 Motivation

An efficient way of manually attributing labels becomes especially important if a very large amount of labeled data is required. Providing a sufficiently large, labeled data set is one of the major non-algorithmic challenges in the application of machine learning for classification tasks (Al-Rawi and Karatzas, 2018). Missing labels are especially problematic if supervised learning methods such as feed forward CNNs are employed. Depending on the variance of the data a large number of samples is required (Russakovsky et al., 2015).

The options to reduce the variance and hence the need to attribute labels to a very large number of samples are limited. We employed preprocessing and manual feature engineering to reduce the variance and tested strategies on the algorithmic domain such as unsupervised and semi-supervised learning as they promise to work with relatively few labels (see section 4.3 and subsection 4.1.1). Nonetheless for training and more importantly performance evaluation of machine learning models a substantial amount of labels are required. Otherwise quality metrics such as accuracies, sensitivity and specificity cannot be calculated. Hence labels had to be manually attributed.
Annotating labels manually requires plenty of effort. “Data set annotation and/or labeling is a difficult, confusing and time consuming task” (Al-Rawi and Karatzas, 2018, p. 2). Human performance is often acknowledged as the baseline or “gold standard” that image classifiers are evaluated by. Hence, in many scenarios data is labeled by humans such that machine learning algorithms can be applied. This holds especially for image classification.[5] In the present case some features were reliably measurable by means of classical computer vision algorithms (e.g. the width or the length). For features such as a flower or the evaluation whether or not a spear is affected by rust, this has proven to be difficult (see subsection 3.2.3). Considering the amount of data that could potentially be labeled, a custom interface is required that allows for time efficient attribution of labels.
We decided to attribute labels for each of the previously described features other than the width and the length to at least 10000 asparagus spears (and hence evaluate 30000 images). This means that several ten thousand judgements had to be

[5]For example the performance of GoogLeNet is compared to human level performance using the ImageNet data set (Russakovsky et al., 2015).

made which highlights the importance of a tool that allows to make this process as quick as possible.

3.3.2 The Labeling Application

A custom application was built aiming for a quick and intuitive labeling process.[6] Questions appear alongside depictions of an asparagus spear and the user answers them using the designated buttons or keys. Once all questions are answered the next asparagus appears automatically and the procedure is repeated. To assist the users in their judgement some automatically extracted features such as the length or width of the asparagus spear as well as a color histogram is displayed.

FIGURE 3.12: **The Labeling Dialog of the Hand-Label App** The depiction shows the main dialog used for labeling. In the left part you can see all three available images (perspectives) for the asparagus with the ID 40. The current question that targets at one of the features of interest is displayed in the area below the images. They are phrased such that the user can answer them with yes or no using the respective buttons or keyboard controls. On the right side you can see the results of the automatic feature extraction. The upper right panel shows the histogram of color hues.

The app comprises two user interfaces: A startup window that allows for a preview of asparagus images and the attributed labels (represented by Ui_Asparator) and the main labeling interface (Ui_LabelDialog) (see Figure 3.13). Using the labeling interface is possible only after the user selects the source folder of images and specifies or loads a file that contains the attributed labels. This ensures that the file paths for input and output are set. A dictionary that maps indices to images can be parsed from file names and the minimum index is determined. As such, the label dialog and the respective controller class always reside in a valid initial state. For labeling, the user answers questions that are displayed alongside the images that depict each asparagus spear from three perspectives. This can be done using the respective buttons or the arrow keys (see Figure 3.12). The result is saved and the next question will automatically appear upon answering.

[6]See https://github.com/CogSciUOS/asparagus/tree/FinalProject/labeling/hand_label_assistant

Automatic feature detection can be selected as an alternative for manual labeling for specific features. The result is displayed and saved to a file. This flexible approach was chosen as it is initially unclear and disputed in how far automatic feature extraction yields results that meet up with the individual, subjective perception. It also allowed to improve automatic feature extraction methods and to develop a direct intuition for the relation to the data. On top of that, it has proven to be useful for debugging automatic feature extraction methods that initially failed for some images.

The development of the app was accompanied by three major challenges. First, handling a large data set of several hundred gigabytes that is accessible in a network drive. Second, changing requirements that resulted from group decision processes with respect to automatic feature extraction as well as from unforeseen necessities in (parallel) preprocessing. This required substantial changes of the initial architecture and the reimplementation of parts of the code. The third challenge is related to the question of the handling of internal states of the app. The latter may be further explained in the following.
Internal states of the app are handled such that it is possible for the user to navigate into invalid states for which no images are available. Note that preprocessing was done such that each asparagus spear has a unique identification number and a specifier for perspectives textita, textitb and textitc in it's filename. While generally the identification numbers are in a continuous range from zero to n, some indices are missing. As preprocessing jobs were scheduled to run in parallel and preprocessing failed for few corrupted files, it has proven almost inevitable to end up with some few missing indices although a dictionary of input filename and output filename was passed to each grid job. In addition, the large amount of data did not allow to save all files in a single directory. In summary, this means that one could not simply iterate over asparagus identification numbers (represented by the state of a spin box in the user interface), determine the file path, and display the related images. Instead, parsing file names from a slow network drive is necessary which requires limiting the number of selected images. As GUI elements such as spin boxes and keyboard controls allow for setting an integer, and it was a requirement that this integer relates to the asparagus identification number, one ends up with the following situation: Either one prevents that the asparagus identification number is set or incremented freely to a value that does not exist, or one allows to navigate into an invalid state. The latter solution was considered to be easier and thus implemented.[7] Hence, all cascades of methods of the app including preprocessing functions that require the respective images as an input were adjusted such that they can handle this case.
The app is implemented using the PyQt5 framework[8] while coarsely following the model, view controller principle. Model and controller are not strictly separate and thus no distinct model class or database is used. Instead the labels are managed as a Pandas DataFrame and serialized as a csv-file. Upon state change (i.e. index increment), images are loaded from the network drive that is mounted on the level of the operating system. The views are designed using QtDesigner. Four manually coded classes are essential for the architecture of the app: (1) The class HandLabelAssistant in which the PyQt5 app is instantiated, (2) the controllers MainApp

[7] The earlier approach showed to have several implementation specific drawbacks. Note for example that upon entering multiple digits in an input field, an event is triggered multiple times. Upon entering the value 10 for the asparagus identification number, one ends up with the value being set to 1 before being set to 10 where 1 relates to a potentially missing identification number. This means the user cannot freely enter IDs because setting them to certain values is impossible.

[8] see https://pypi.org/project/PyQt5/

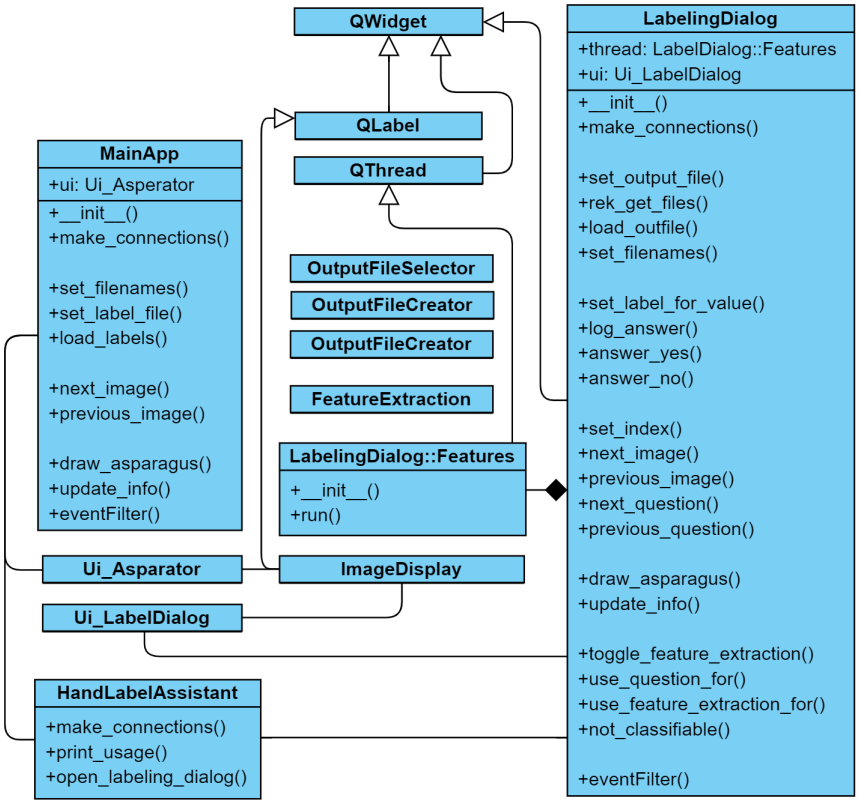


FIGURE 3.13: **UML Diagram for the Hand-Label App** The depiction shows the class diagram for the hand-label app in UML.

and (3) LabelingDialog as well as (4) Features which is member class of the latter. Features is of type QThread and represents the API to the automatic feature extraction methods in FeatureExtraction. Ui_Asparator, UiLabelDialog and classes for file dialogs represent the views. A class ImageDisplay is required to display images with the correct aspect ratio. Figure 3.13 shows the UML class diagram alongside methods and attributes that are considered relevant to understand the architecture of the app.

Developing a custom app for the labeling process required substantial time resources. However, it was found that existing solutions did not meet the specific requirements. Our custom hand-label app allowed us to attribute labels to more than 10000 asparagus spears in a manageable amount of time. Details of the manual labeling process are described in the next section.

3.4 Manual labeling

In this section, the process and the results of manually labeling the data with the help of the hand-label app is laid out. The labeling criteria which allocate each spear to a single quality class were explained in the subsections of the section Feature extraction. The outcome of the labeling process and one approach to measure the agreement of the manual labeling will be described in the following subsections.

The images were labeled for their features by all members of the group with the hand-label app (described in section 3.3). As none of the team members were experts in asparagus labeling, a general guideline for the feature labeling had to be

established (see section 3.2). The guideline was written in accordance with the owner of the asparagus farm Gut Holsterfeld, Mr. Silvan Schulze-Weddige. He was consulted in all questions regarding the labeling of the asparagus.

(A) bent? (B) flower? (C) violet?

FIGURE 3.14: **Examples of Critical Images** Three examples of asparagus are shown where the corresponding feature label is difficult to determine – thus, a critical decision has to be made. Image (A) displays an asparagus that shows a slight S-curve, however, not very strongly. It might be labeled as being bent or as not being bent by the annotator. Images (B) and (C) show asparagus with brown spots which can be judged as rust or as pressure marks (no rust), again depending on the annotator. Additionally, it is not obvious whether the asparagus in (B) exhibits a flower or whether the spear in (C) is of violet color at the head region.

The features labeled by the human annotators included the features hollow, flower, rusty head, rusty body, bent, and violet. Additionally, the feature not classifiable could be selected and attributed to spears where a feature detection was not possible (see subsection 3.2.8). The features fractured, very thick, thick, medium thick, thin, and very thin were extracted via the automatic feature detection functions described in the respective sections for Length and Width which were integrated in the hand-label app. Other features that were integrated into the hand-label app, and which helped as an orientation to the human annotators, were the feature extraction functions for feature bent and feature violet.

General challenges in the manual labeling in front of a computer screen, including the respective image quality and the variance in the agreement of the project members, were expected from the start. As the task relies on the subjective view of individual humans, opinions about the presence or absence of features can diverge. By consulting Mr. Schulze-Weddige on difficult decision-making cases, it became clear that some examples are difficult to classify even for experts (see Figure 3.14). To tackle the issue and to have an overview of the general agreement of the labeling between group members, a measure was applied, namely the Kappa Agreement. The Kappa Agreement was used to assess the degree of accordance in labeling between the single members and monitor how the labeling agreement developed during the manual labeling process.

3.4.1 Labeling outcome

In this section, the process and the results of the labeling with the hand-label app are described.
The labels of all labeled images are stored in a csv-file, as shown in Figure 3.15. The first entry is the image identification number. Every feature can be of value 0, value 1 or empty. Whenever a feature is present in an image, the value is set to 1. If the feature is absent, it is set to 0. For images labeled as not classifiable, all feature values remain empty. The image path to every of the three images for one spear is also saved in the label file in a separate column. After the labeling process, the individual csv-files with the labels are merged into one large `combined_new.csv` file. The content of this file is later used for the classification of the data with the different approaches (see chapter 4). It can be found in the study project's GitHub repository.[9]

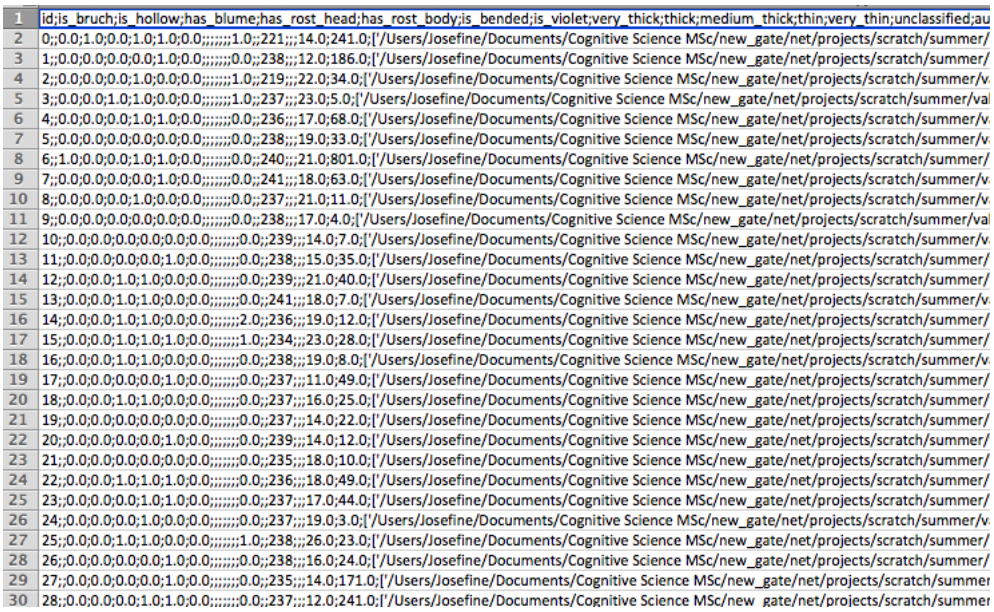


FIGURE 3.15: **Label File** The feature labels extracted by the manual labeling process were saved in a csv file. This image shows the beginning of the file `combined_new.csv` in which all label files were later combined as one.

[9] See https://github.com/CogSciUOS/asparagus/tree/FinalProject/preprocessing/get_data

The manual labeling lasted over the period of November 2019 to January 2020. A session usually consisted of 500 images, with an asparagus spear being viewed in three positions. A session of labeling 500 images took between two and four hours. One minor factor sometimes influencing the time spent for labeling were difficulties with the external SSH connection to the IKW storage, as all images are stored on the university servers. The average time spent for labeling one asparagus is around 27 – 48 seconds.
All in all, 13319 triples of images were labeled for their features. There is a large variance in the presence of the features in the data as can be seen in Table 3.1. Of the acquired 13319 images, the feature most present in the data is rusty body with 45.5%, followed by the feature bent with 40%. Features whose representation is below 10% include the features violet (7.9%), hollow (3.3%), fractured (3.5%), and very thick (4%). The feature not classifiable shows least presence with 2.1%.
It emerges that many features are only sparsely present in the data. This poses an imbalance in the data that is relevant for later classification tasks and the usage of the data set.
Further, every member of the group participated in the labeling but not everybody labeled the same number of images. Due to this circumstance, the labeling bias of certain members is more present in the data than of others.
The manual labeling was stopped when the amount of classified data exceeded 13000 samples. Reason for this is that we had reached our time limit for labeling samples and needed to begin with training our classification approaches.[10]

Feature	**%**	**Feature**	**%**
hollow	3.3%	fractured	3.5%
flower	12.9%	very thick	4.0%
rusty body	45.5%	thick	29.1%
rusty head	14.7%	medium thick	18.7%
bent	40.0%	thin	17.9%
violet	7.9%	very thin	30.3%
		not classifiable	2.1%

TABLE 3.1: **Feature Representation in the Data Set**
In this table, the representation of each feature in the manually labeled 13319 asparagus samples is reported in %.

3.4.2 Agreement Measures

When different annotators label data, it is indispensable to verify the degree of agreement among raters. In order to judge how consistently the data is labeled, several statistical methods (inter-rater-reliability) can be applied.

For the current purpose, different agreement measures, all implemented by scikit-learn, are used. The first is Cohen's Kappa. It is seen as a more robust measure than a simple agreement percentage, such as a measure of true positives and true negatives, which was traditionally used for those cases (Cohen, 1960). Cohen's Kappa is more robust, as the rate of agreement occurring by chance is included in

[10]To have an intuition of how much labeled data we might need, we found some of the suggestions in a blogpost helpful which can be visited at https://machinelearningmastery.com/much-training-data-required-machine-learning/. However, we did not find an exact number when to stop labeling.

the calculation. This method is applicable to compare the rating of two raters on a classification problem. The degree of agreement is always within -1.0 and 1.0 inclusive. The higher the Kappa value, the higher the agreement. Values around zero indicate no agreement and negative values indicate negative agreement which can be interpreted as systematic disagreement. Values between 0.41 – 0.6 are seen as moderate agreement, 0.61 – 0.8 as substantial agreement, and everything above as almost perfect agreement. All scores below 0.4 are interpreted as unacceptable (McHugh, 2012).
Another statistical method used to measure the agreement is the F1 score. The F1 score is used for the evaluation of binary classification. It relies on both precision as well as recall of a test. An F1 score value lies between 0.0 and 1.0 – the higher the F1 score, the higher the agreement.
Lastly, we calculated the accuracy measure. For a normalized accuracy score, the values lie between 0.0 and 1.0, and the best performance is 1.0. This measure returns the fraction of correctly classified samples. It is a less robust measure than Cohen's Kappa score (McHugh, 2012).

3.4.3 Reliability

In order to evaluate the degree of agreement of our data, we made agreement measures at two points in time.[11]
The first time, six different annotators assigned feature labels to images out of each class of the pre-sorted asparagus that we obtained by running the spears through the machine twice. We ensured that always two different annotators labeled the same set of images. The Kappa scores varied strongly between groups and features from -0.03 to 0.76, while the accuracy scores ranged from 0.49 to 1. We were surprised that the agreement scores are low, even though the raters gave the same label to many of the asparagus spears. This is an acknowledged problem (Powers, 2012; Sim and Wright, 2005; Feinstein and Cicchetti, 1990; Flight, 2018).
One reason for our results could be that we compared the agreement class-wise. The occurrence of 1s and 0s per class is therefore very unbalanced. For Kappa scores, if the distribution of 0s and 1s is not balanced, disagreement of the underrepresented value is punished more heavily.[12] Therefore, we decided to repeat our agreement measure feature-wise on non-labeled images, so that the annotators cannot anticipate a specific group label. In order to better understand the reliability of our data, we additionally decided to look at the accuracy score and the F1 score. Beforehand, the team labeled another 50 images all together, clarified classification boundaries again and discussed unclear images (see Figure 3.14).
The second time, 50 images were labeled by four annotators. The agreements were measured annotator-pair-wise, and then averaged. The results in the Cohen's Kappa score vary between and within features, and between annotator pairs as well. The highest aggregated kappa score over all annotator pairs is reached for the features flower (0.79) and hollow (0.79), then violet (0.76), rusty head (0.72), bent (0.55) and lastly for rusty body (0.47). For the features flower, rusty head and violet, the interquartile range (IQR) is quite small, whereas the IQR for hollow, bent and rusty body is much larger (see Figure 3.16 and Figure 3.17).
The agreement scores accuracy and F1 yield very similar results. Results are slightly better than the Kappa scores, in total and for each feature. The highest median accuracy score is reached for the feature hollow (0.96), then flower (0.93),

[11] for our API documentation see
https://asparagus.readthedocs.io/en/latest/api/measure_agreement.html
[12] see also
https://stats.stackexchange.com/questions/47973/strange-values-of-cohens-kappa

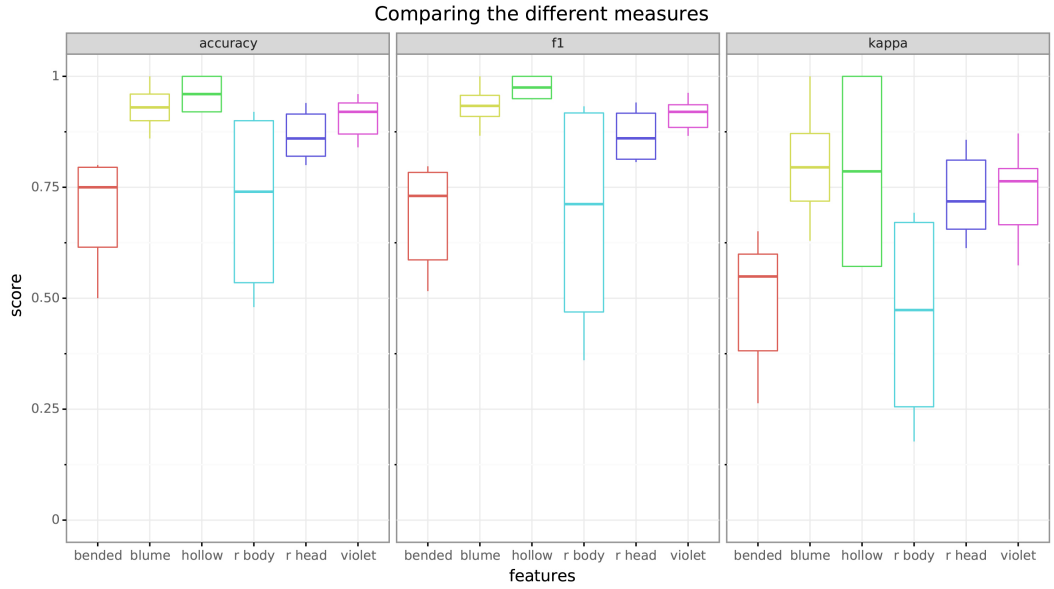


FIGURE 3.16: **Comparing Measures** The figure shows the agreement measures accuracy, F1 and Cohen's Kappa, separately for each manually labelled feature. Shown are the box-plots, so the middle line indicates the median, the box indicates the IQR. All scores are aggregated scores over all annotator pairs.

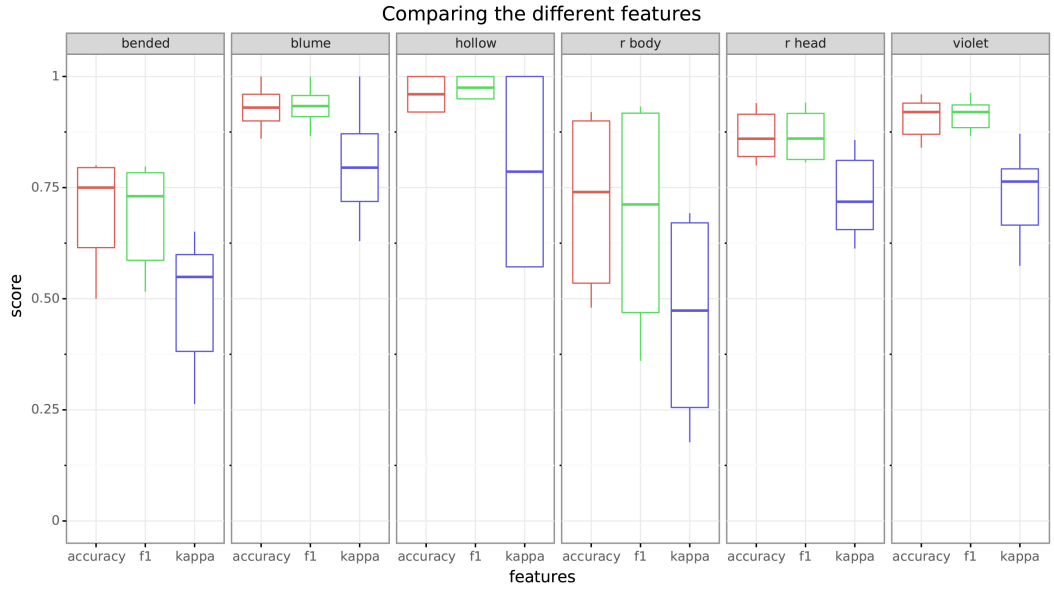


FIGURE 3.17: **Comparing Features** The figure shows each feature separately. For each feature, the corresponding accuracy, F1 and Cohen's Kappa score is given. Shown are the box-plots, so the middle line indicates the median, the box indicates the IQR. All scores are aggregated scores over all annotator pairs.

then violet (0.92), then rusty head (0.86), then bent (0.75) and then rusty body (0.74). The order is the same for the F1 scores. The median F1 scores lie between (0.71 and 0.97).

All in all, it can be said that the agreement scores indicate moderate up to substantial agreement. Regardless of the measurement, agreement is highest for the features flower, violet and rusty head.

3.5 The asparagus data set

In the following chapter, the asparagus data sets will be discussed. The data that we collected is restructured and preprocessed into several different versions which are used for the classification approaches. Further, the possibility to save a Tensor or a `TFRecord` file and the `tf.data` API and the `tf.data.dataset` API are introduced. These methods are compared and followed by a recommendation for further work with the data set.
One peculiarity about our data is its variance. The differences in the features which decide the class label of an asparagus are very small. All in all, the asparagus spears look very similar. This differentiates our classification task from other examples in the literature which usually have a high inter-class difference. We decided to reduce the variance within the images even further by removing all differences that are not relevant for the classification task (see Chapter 3.1). By doing so we facilitate the learning process and help the model focus on the features that matter.

3.5.1 Description of our data set(s)

With the help of the preprocessing as described in 3.1 Preprocessing, the raw images are transformed and distributed into folders which serve as input for the networks. The different data sets can be distinguished in three ways: the images have the original background or the background is removed, the asparagus spears are moved to the center of the image patch and rotated upwards or not and the images are down sampled to reduce memory consumption or in the original resolution.
Further, a data set is constructed with all the labeled images in a single numpy array, which can be stored and loaded at once. As the three perspectives of each asparagus spear are concatenated horizontally, they appear to lie next to each other in the resulting image. These concatenated images are then combined to the final file. The first out of four dimensions in this file depicts the number of labeled asparagus spears. The second and third dimension represent the height and the width of the images, respectively. Further, the fourth dimension represents the three RGB values. In addition, the images are downscaled to facilitate the training process and reduce memory consumption. Each image is downsampled by a factor of six, that means every 6th pixel is used in the reduced image. This factor can be easily changed and a new data set can be created.
An additional data set is generated which contains images of the head region of the asparagus spears exclusively. This data set was used to train a dedicated network for head related features (see subsection 4.1.4).

3.5.2 Data set creation with Tensorflow

In the following, TensorFlow's own binary storage format `TFRecord` is introduced. This approach facilitates the mix and match of data sets and network architectures. The large amount of data that was collected has a significant impact on our import pipeline and, therefore, on the total training time. The file format is optimized for images and text data. These are stored in tuples which always consist of file and label. In our case, the difference in reading time is significant, because the data is stored in the network and not on a SSD on the local PC. The serialized file format

allows the data to be streamed efficiently through the network efficiently. Another advantage is that the file is transportable over several systems, regardless of the model one wants to train.
Two data sets are created in this file format. One with all files that were preprocessed, and another one with the preprocessed and labeled data. The first binary file includes images with background in png format and has a size of 225 Gigabyte (GB). The `TFRecord` file does not only take up less memory capacity, but can also be read more efficiently. The second `TFRecord` file includes all labeled images and their labels.
Working with these files simplifies the next steps of transformations. With the `tf.data` API complex, input pipelines from simple and reusable components are created, even for large data sets. The preferred pipeline for our asparagus project can apply complex transformations to the images and combine them into stacks for training, testing and validating in arbitrary ratios. A data set can be changed, e.g. by using different labels or by transformations like mapping, repeating, batching, and many others. These dozens of transformations can be combined.
Besides the described functional transformations of the input pipeline under `tf.data.dataset`, an iterator gives sequential access to the elements in the data set. The iterator stays at the current position and allows to call the next element as a tuple of tensors. Initializable iterators go through the data set in parallel. In addition, different parameters are passed to start the call. This is especially handy when searching for the right parameters in parallel.
In summary there are two advantages using `tf.data`. On the one hand, it is possible to build a data set with different data sources. On the other hand, there are many functional transformations and an iterator with sequential access to the data.

Further we tried to add our data set to the `TFDS` [13]. `TFDS` enables all users to access the data set directly using the TensorFlow API. For this, we need to publish the data. However, the process of publishing the data set turned out to be too time consuming. Especially the large amount of data was a problem. Questions like: How do we deal with the fact that only a part of the images has labels? How should we pass the labels: each as a feature, in a list, or as several features? It would have been better, faster, and more helpful for the development of our network approaches, if we had continued to search and to integrate the `TFRecord` files with the `tf.data` API in our pipeline early.

[13] See `https://www.tensorflow.org/datasets/beam_datasets`

Chapter 4

Classification

Given the structure of our data set, namely image data which is partially annotated with corresponding class labels and partially hand-labeled with corresponding feature labels, different machine learning and computer vision methods were chosen to tackle the problem of image classification.

Image classification refers to the method of identifying to which category an image belongs to, according to its visual information. Classification problems can be divided into three different types: binary, multiclass and multi-label. Moreover, there are different methods on how to approach image classification. Those can be divided into three main groups: supervised learning, semi-supervised learning and unsupervised learning (Har-Peled, Roth, and Zimak, 2003).
Binary classification can be used to decide whether a certain feature is present in an asparagus spear, or not. Multiclass classification solves this problem as well, but is also applicable to data with more than two classes. As the class label results from a combination of the presence of certain features and the absence of others, it is also reasonable to go for a multi-label classification approach. In this approach, several features are learned simultaneously without being mutually exclusive.
During our group work, algorithms of the different classification types as well as of the different learning types were applied. In the long run, an integrated model was aimed at that is predicting all features of a single asparagus spear, from which the final class label can be inferred. However, as intermediate steps towards that goal, the focus was to optimize models on identifying the presence of the features described in section 3.2.

This chapter gives a general background of the different approaches chosen for our image classification problem, as well as a detailed overview of the concrete implementations of the models and the mechanisms of their hyperparameters.

4.1 Supervised learning

In machine learning, there are different approaches for an application to be trained on a set of data (Géron, 2019; Bishop, 2006). Depending on the level of supervision that the system receives during the training phase, the learning process is grouped into one of four major categories (Géron, 2019). One of these categories is supervised learning. For supervised learning approaches, the training data includes not only the input but also its corresponding target labels. The objective is to find a mapping between object (x) and label (t) when a set of both (x, t) is provided as training data to the application (Olivier, Bernhard, and Alexander, 2006). An advantage of supervised learning is that the problem is well defined and the model can be evaluated in respect to its performance on labeled data (Daumé III, 2012; Olivier, Bernhard, and Alexander, 2006). In other words, the labels are used as a direct basis for the model optimizing function during training.

Supervised learning spans over a large set of different methods, from decision trees and random forests, to Support Vector Machines (SVMs) and Artificial Neural Networks (Caruana and Niculescu-Mizil, 2006; Géron, 2019).
A classical task for supervised learning systems is the classification of received data and mapping it to one of a finite number of categories (Bishop, 2006). The disadvantage of supervised learning is the effort of receiving enough labeled data. It can be challenging to obtain fully labeled data because labeling experts are needed to classify the data which is usually a time consuming and expensive task (Zhu, 2005; Figueroa et al., 2012).

In the following sections, different supervised learning methods were chosen to solve the classification task using the data that was manually labeled for features as described in chapter 3. In 4.1.1 Prediction based on feature engineering, an approach using an MLP is described for feature classification. The section of 4.1.2 Single-label classification is concerned with labeling the input images with a CNN in a binary setup for their designated features. In section 4.1.3 Multi-label classification, a neural network is trained on the data to label it not only for one feature but all features at the same time. In the fourth section, 4.1.4 A dedicated network for head-related features, a CNN is used to train solely on the head image data for the features flower and rusty head. Finally, in 4.1.5 From hand-labeled features to class labels, a random forest approach is described to map the features of the image data to their class label.

4.1.1 Prediction based on feature engineering

Besides approaches that directly use images as an input one may use high level feature engineering. That is, one can retrieve sparse representations that contain relevant information in a condensed form and apply classical machine learning classifiers such as MLPs to predict labels (Zheng and Casari, 2018). These classifiers are comparatively simple, fast to train and only few network hyperparameters have to be defined. One may argue that this is one of the major benefits as compared to networks of higher complexity (e.g. deep CNNs). As a consequence, finding suitable parameters for MLPs (i.e. the number of hidden layers and neurons per layer) is comparatively easy.[1] Beside of that retrieving sparse representations highly reduces the amount of variance and hence the required amount of labeled data.
The simplicity of MLPs also means that the best suitable structure could be found more easily as compared to deepleaning CNNs. In CNNs for deep learning, suitable means are required to avoid the vanishing gradient problem (Wang, 2019). Further, kernel sizes, strides, the number of kernels and other parameters must be defined. Therefore, designing shallow MLPs appears to be easier: As less decisions must be made, there is less one could possibly do wrong. Underfitting can potentially be due to an unsuitable network design or because predictions are impossible due to incongruencies or missing information in the sparse data set (LeCun et al., 2012).

[1] The challenge of finding appropriate network parameters is well known in the deep learning community: "Designing and training a network using backprop requires making many seemingly arbitrary choices [...]. These choices can be critical, yet there is no foolproof recipe for deciding them because they are largely problem and data dependent" (LeCun et al., 2012, p. 9). The requirement to specify hyperparameters is a disadvantage of neural networks (including MLPs) as compared to parameter free methods (Scikit-Learn, 2019). Due to the combinatorial explosion, the challenge of finding suitable parameter settings is harder for more complex networks as more options must be considered.

An extensive search in hyperparameter space is practicable for simple MLPs. This is because of them having few parameters only and because training on sparse representations limits the number of neurons in the networks. Taken together this results in fast training that allows for many experiments. If the learning task is simple enough to be accomplished by MLPs (e.g. finding combinations of partial angles that correspond to the impression of curvature), one may hence speculate that underfitting can rather be explained by incongruencies or missing information in the labels than a result of issues in network design and training parameters.
This classical machine learning approach which relies on feature engineering is applied to predict features based on color and partial angles of asparagus spears because of these benefits.[2]

Violet and rust prediction based on color histograms
The feature violet is based on the distribution of sufficiently intense color hues in the violet color range. The initial approach of measuring the feature faces at least two drawbacks. First, it requires defining two thresholds. Second, the impression of a violet asparagus spear could possibly be affected by the combination of colors that are potentially outside the violet range or that are too pale to be considered (see subsection 3.2.4). The same holds for the features rusty body and rusty head. Hence, in a second approach histograms were computed for foreground pixels after transforming the images to palette images with 256 color hues (see section 3.1). The resulting representation is a sparse descriptor that allows to predict color features using explicitly defined rules or trainable machine learning models.

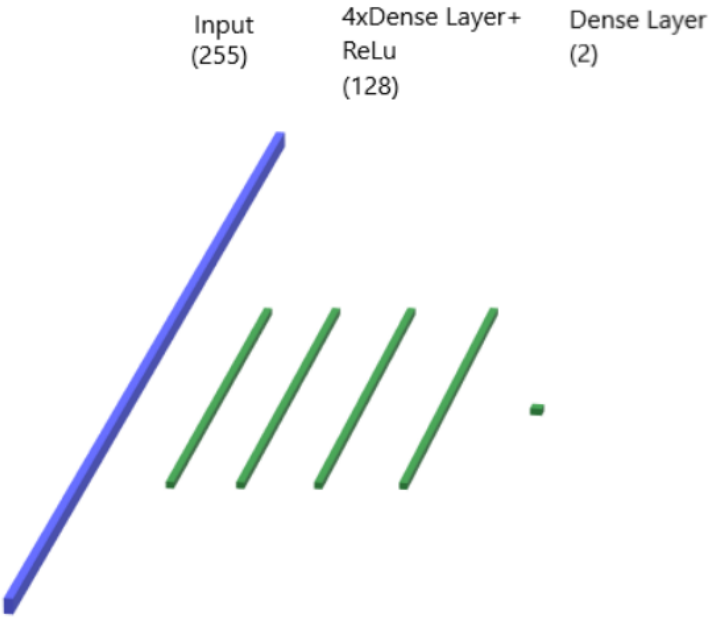


FIGURE 4.1: **Feature Engineering Net Structure Color** The depiction shows the network structure of the MLP for color-related features.

The sample set contains a histogram for each labeled asparagus. Twenty percent are randomly assigned to the evaluation set. A simple MLP with four hidden layers and 128 neurons in each of them was trained on the resulting normalized histograms of palette colors (ReLU activation / sigmoid activation in the final layer).

[2] See `https://github.com/CogSciUOS/asparagus/tree/FinalProject/classification/supervised/mlps_and_feature_engineering`

	False positive	False negative	True positive	True negative	Sensitivity	Specificity
violet	0.04	0.03	0.05	0.88	0.62	0.96
rusty body	0.19	0.14	0.33	0.34	0.71	0.65

TABLE 4.1: **Feature Engineering Color-Histogram-Based Prediction** Performance of color histogram-based predictions with a MLP.

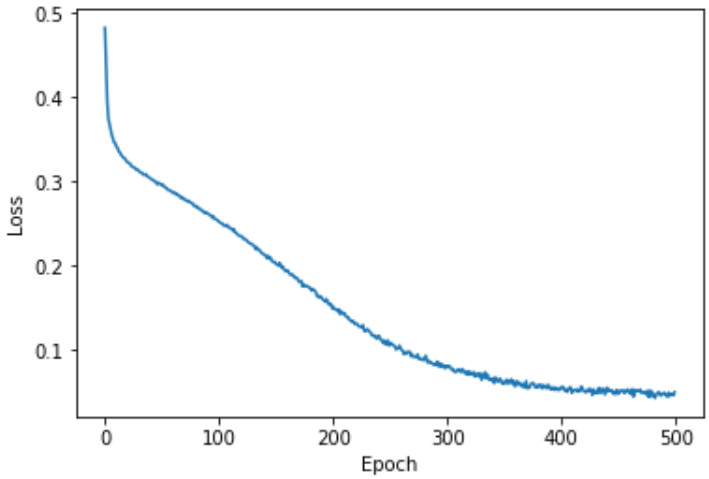


FIGURE 4.2: **Learning Curve for Palette-Color Histogram based Prediction** The depiction shows the loss per training episode for the MLP trained on histograms of palette images.

Hyperparameters were optimized and the network was trained for a total of 500 epochs as the learning curve indicated convergence at this point.

Curvature prediction based on partial angles

The sensitivity for curvature detection is high: 82% of all bent spears were identified as such. In comparison, this holds for 71% of the spears affected by rust and for 62% of the violet spears. In contrast, almost all spears that are identified as not to be violet are labeled accordingly (96% specificity). However, the specificity for rust (65%) and curvature (67%) is lower. Although the accuracies are far from perfect the results appear to be promising.

	False positive	False negative	True positive	True negative	Sensitivity	Specificity
bent	0.19	0.07	0.34	0.4	0.82	0.67

TABLE 4.2: **Curvature Prediction** Performance of curvature prediction based on angles.

The receiver operating characteristic reveals that the prediction is of better quality for violet and curvature prediction as compared to rust prediction which is reflected in a smaller area under the curve for the latter. Possibly this reflects that rather small brown spots were considered rust by some raters but not by others.
Considering the low agreement in labeling, high values for the specificity and sensitivity of the classifier are not expected. A likely explanation for rather low values is that the model generalizes deviating and potentially contradicting perceptual color-concepts that we had when attributing labels which in return affects the reliability of the data. Arguably, only little information was discarded by computing

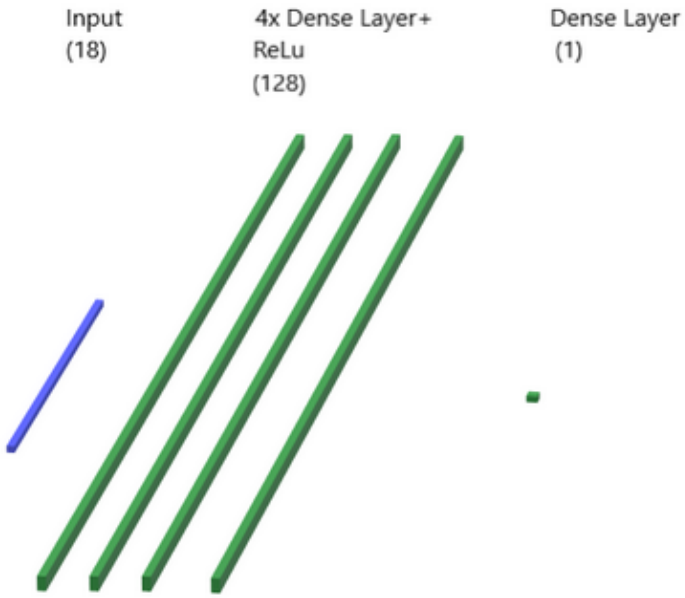


FIGURE 4.3: **Feature Engineering Net Structure Curvature** The depiction shows the network structure of the MLP for the feature bent.

the sparse representations that served as an input for training. However, information about irregularities in the outline are not reflected in the partial angles but might contribute to the perception of curvature. The same holds for the spatial distribution of colored pixels which might contain additional information regarding the detection of rust and violet. Nonetheless, the major criteria are captured. As MLPs are suitable to establish non-linear mappings and as the task of mapping high level features (such as partial angles) to human estimates appears to be rather simple, one may speculate that there is rather little potential to improve the predictive quality using other techniques. By introducing a bias, the sensitivity of the classifier can be adjusted at the cost of more false positives. Here, introducing a bias means that the threshold that is used to convert the floating point outputs of a neural network to booleans that indicate whether a feature is present or not is set to values other than 0.5. The possibility of making the classifier more or less sensitive appears to be a good option to be implemented as a feature for customization by the user in asparagus sorting machines.

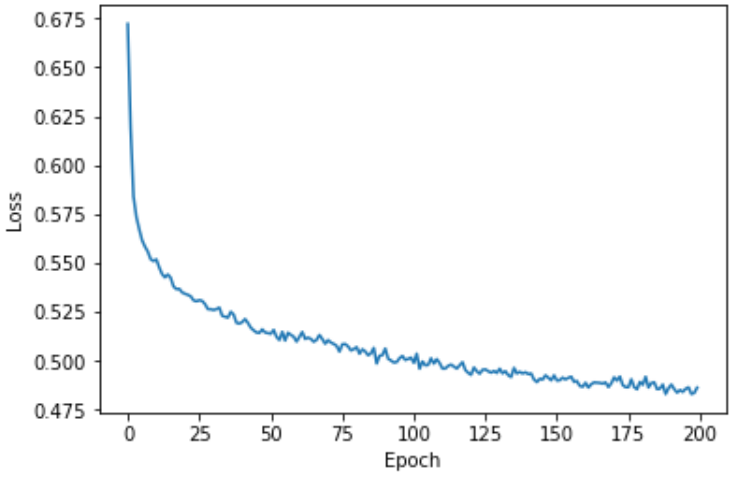


FIGURE 4.4: **Learning Curve For Angle Based Prediction** The depiction shows the loss per training episode for the MLP trained on partial angles of the centerline of asparagus spears.

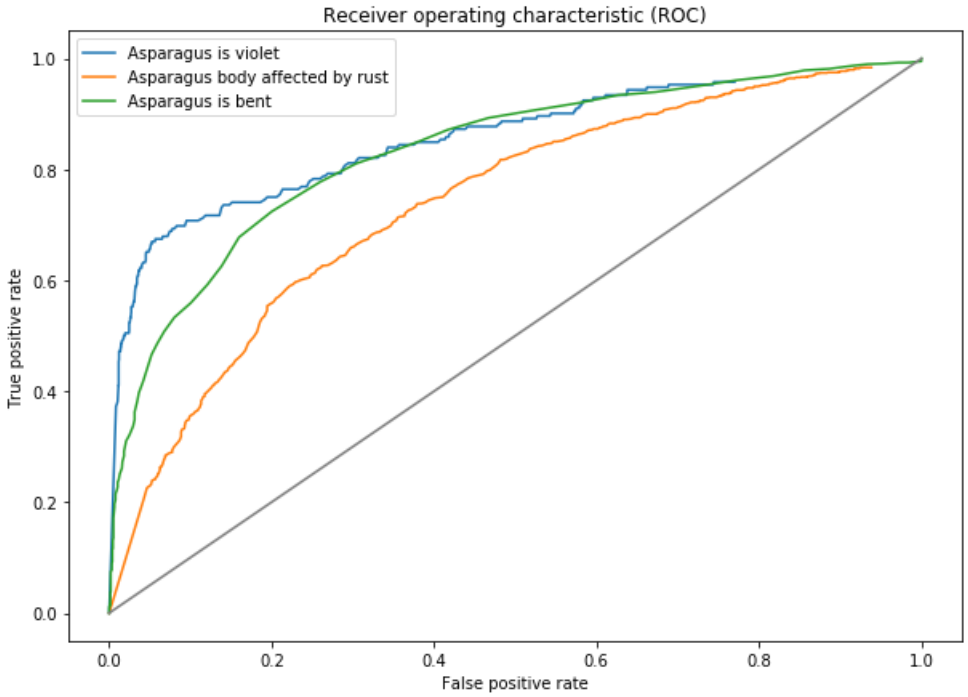


FIGURE 4.5: **ROC for MLPs Trained on High Level Features** The depiction shows the Receiver Operating Characteristic (ROC) for the classifiers that were trained on the features retrieved via feature engineering with MLPs. This allows to compare the performance. A larger area under the ROC curve indicates better performance while a curve close to the diagonal line indicates poor results.

4.1.2 Single-label classification

In the following chapter, a Convolutional Neural Network is described, which is used for single-label classification on features.[3] The approach was tested on the 13319 hand-labeled data images. 13 models were created, each predicting one feature. [4]
A general model structure was needed as inspiration for the CNN. For example, the Visual Geometry Group (VGG) networks with varying depth seem to be a good choice for image classification as their VGG16 won the ImageNet challenge of 2014 and is often implemented for image classification tasks (ul Hassan, 2018c; Simonyan and Zisserman, 2014). However, there are two major drawbacks to using them. That is, they are slow to train and in need of a lot of memory storage due to their depth and the amount of fully-connected nodes (ul Hassan, 2018c; Zhang et al., 2015). Part of these problems also arise with even deeper networks like ResNet (He et al., 2016b; ul Hassan, 2018b). Thus, AlexNet is chosen as a blueprint for the CNN because it is small in relation to other networks while still performing comparatively good (ul Hassan, 2018a; Krizhevsky, Sutskever, and Hinton, 2012; Géron, 2019). As the variance in the data images is relatively small it is assumed that not as many layers are needed as employed in deeper networks like VGG (Géron, 2019).

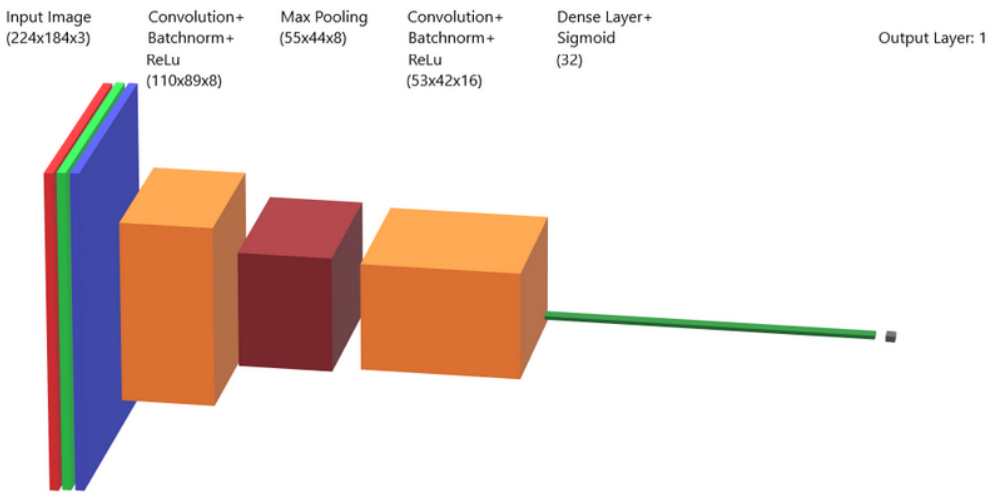


FIGURE 4.6: **Single-Label CNN Net Structure** The depiction shows the network structure of the simple feedforward CNN.

The network comprises four hidden layers: a convolutional layer, followed by a pooling layer, a second convolutional layer, and a dense layer. The input is an array of multiple horizontally stacked images with no background removed and reduced by a factor of six. This input is trained on a set of binary labels containing information on whether the respective feature label applies to the current image. The output of the network gives a prediction on each entered image gated by a sigmoid function on a range between zero and one. The rounded integer values of this output give a prediction of the apparent feature label. For the training phase of the model, the Adam optimizer is used because of its general acceptance as the state of the art optimizer for backpropagation (Bushaev, 2018; Kingma and Ba, 2014). As

[3] It can be found under https://github.com/CogSciUOS/asparagus/tree/FinalProject/classification/supervised/single-label-CNN
[4] The 13 features to predict are: fractured, hollow, flower, rusty head, rusty body, bent, violet, very thick, thick, medium thick, thin, very thin, and not classifiable.

a loss function, binary cross-entropy is used as it promises good results for binary single-label classification tasks (Géron, 2019; Godoy, 2018; Dertat, 2017).
When training Artificial Neural Networks, it can be difficult to find clear guidelines on how to implement an architecture such that an optimal training performance is given (Heaton, 2015; Géron, 2019; Bettilyon, 2018). Hence, the idea was to start with the simplest form of a CNN and then gradually increase the complexity of the network. While AlexNet provides a good baseline for an image classification network, its architecture is still assumed to be unnecessarily complex for the given task. First, the architecture was reduced to the minimum number of layers and parameters needed for a CNN. Over the period of training optimization, various processing steps and hyperparameters were implemented and compared according to their performance. During this process, the data was split between 12000 samples for training data and 1319 samples of validation data in order to have a reasonable overview on the possible test performance and to directly check for overfitting. The data used as test data was randomly chosen from the whole data set.[5]

In the following, the development of the hyperparameters over time is explained.
The batch size was initiated comparatively low with 64 samples per batch but soon adjusted to a value of 512 samples. The larger batch size was implemented in order to guarantee the convergence of the training data, since smaller batch sizes result in jumping gradients, which are not able to converge into any minimum (Bengio, 2012).
Various learning rates were tested during the optimization process. The initial learning rate of 0.003 (which is the standard learning rate for Adam optimizers in TensorFlow (Kingma and Ba, 2014; Géron, 2019)) was soon found to be too large to guarantee convergence of the algorithm. Thus, the learning rate was decreased and found to be most effective in the range of $1e^{-5}$ to $1e^{-8}$. A gradually decreasing learning rate was tested in order to make the training more effective (Bengio, 2012). [6] It does not yield better results and thus a constant learning rate of $1e^{-5}$ is implemented.
Starting with the smallest layer size possible, in the course of time more layers were added. For example, in order to detect the feature fractured one layer for the edge detection is considered to be sufficient. For other, higher-level features, such as bent, additional convolutional layers were expected to be more helpful (Géron, 2019). During the training process, it was settled to a model with the architecture described above.
A small number of kernels is used compared to AlexNet since for single-label classification fewer kernels are expected to be needed (Géron, 2019). The kernel size is picked to be eight 5×5 kernels for the first convolutional layer and 16 3×3 kernels for the second convolutional layer. The hidden dense layer consists of 32 neurons. These sizes are assumed to be sufficient, since increasing the number of kernels does not lead to any better results but to overfitting on the training data.
Both convolutional layers of the model are built with batch normalization. The gradients were inspected visually and the results give no reason for assuming exploding or vanishing gradients (Pascanu, Mikolov, and Bengio, 2012).
A next step was to weight the loss function because of the largely unbalanced data (He and Garcia, 2009; Batista, Prati, and Monard, 2004). However, this did not lead to any changes. The model still tended to make an unbalanced prediction by classifying all values as negative samples. Another idea is to reduce the data

[5] Only the models trained on detecting the features hollow, flower, rusty head, rusty body, bent, and violet use the same validation set. Like this, a better comparison can be made between them. For the other features, the validation set was always randomly composed anew before model training.
[6] In theory, the Adam optimizer already manages learning rate decay (Kingma and Ba, 2014).

set to make the number of images with the regarding feature present even to the number of images where it is absent. This would mean to throw away valuable data, which can otherwise provide information about negative cases to support the model in its training (Batista, Prati, and Monard, 2004). It was decided to keep all data images and instead balance the data by multiplying the minority of samples to match the number of contrary samples. As there was no feature positively exceeding a presence of 50% in the data, solely positive labels were oversampled. The balancing was only performed on the training data, while the test data was not changed.
To prevent overfitting of the negative data samples, L_2 regularization was applied. To improve training performance, some kinds of data augmentation were tested (Brownlee, 2019) like horizontal flipping or small changes in the angle (up to 5°) but they are not used in the final version.

Around 2700 to 5400 training steps are performed for the training, translating roughly to 120 epochs (with an exception for the model of feature fractured, which is trained for 60 epochs). Due to the balancing of the data, the number of training data (and therefore the ratio of training steps to epochs) varied when training the CNN on the different features.

	Sensitivity	Specificity	Validation Accuracy	Balanced Accuracy	Training Steps
fractured	0.8846	0.9976	99.32%	94.11%	2690
hollow	0.7308 (0.7692)	0.9821 (0.9831)	97.58% (97.77%)	85.65% (87.62%)	5390 (4270)
flower	0.5603 (0.6983)	0.8975 (0.8288)	85.96% (81.41%)	72.89% (76.36%)	4790 (2900)
rusty head	0.4311 (0.5210)	0.8695 (0.8106)	79.86% (76.38%)	65.03% (66.58%)	4670 (3320)
rusty body	0.6420 (0.6400)	0.7767 (0.8049)	71.15% (72.51%)	70.94% (72.25%)	2990 (2270)
bent	0.6541 (0.6753)	0.7533 (0.7500)	71.25% (71.93%)	70.37% (71.27%)	3230 (2470)
violet	0.4643 (0.5119)	0.9652 (0.9452)	92.45% (91.00%)	71.48% (72.86%)	5150 (3550)
very thick	0.9778	0.9939	99.32%	98.59%	5390
thick	0.9525	0.9688	96.41%	96.07%	3950
medium thick	0.8917	0.9249	91.87%	90.83%	4550
thin	0.9399	0.9280	93.13%	93.40%	4190
very thin	0.9733	0.9579	96.13%	96.56%	4310
not classifiable	0.5217	0.9961	98.79%	75.89%	5390

TABLE 4.3: **Single-Feature Label Classification Results** In this table, the sensitivity, specificity, validation accuracy, balanced accuracy, and the number of training steps are given for each feature model after training on 12000 original hand-labeled data images. The numbers in brackets indicate the best result for the model in relation to its balanced accuracy. For most features, training lasted for 120 epochs. However, the number of data samples (and thus training steps) varies between features because of the data balancing.

The CNN is trained on 13 features separately, resulting in 13 trained models. For some features, there are no labels in the csv-file but they can be calculated from the parameters length and width of an asparagus. That is, the features very thick, thick, medium, thin, and very thin are all calculated from the thickness measured by the automatic feature extraction algorithm for width, according to the boundaries for each class label (for reference, see Table 1.1 and Figure 1.1 in section 1.3, and also subsection 3.2.2). For the feature fractured, the length was set to a threshold of 210 mm, with all asparagus of smaller length labeled as fractured. Additionally, asparagus labeled as not classifiable was included for training the model that is

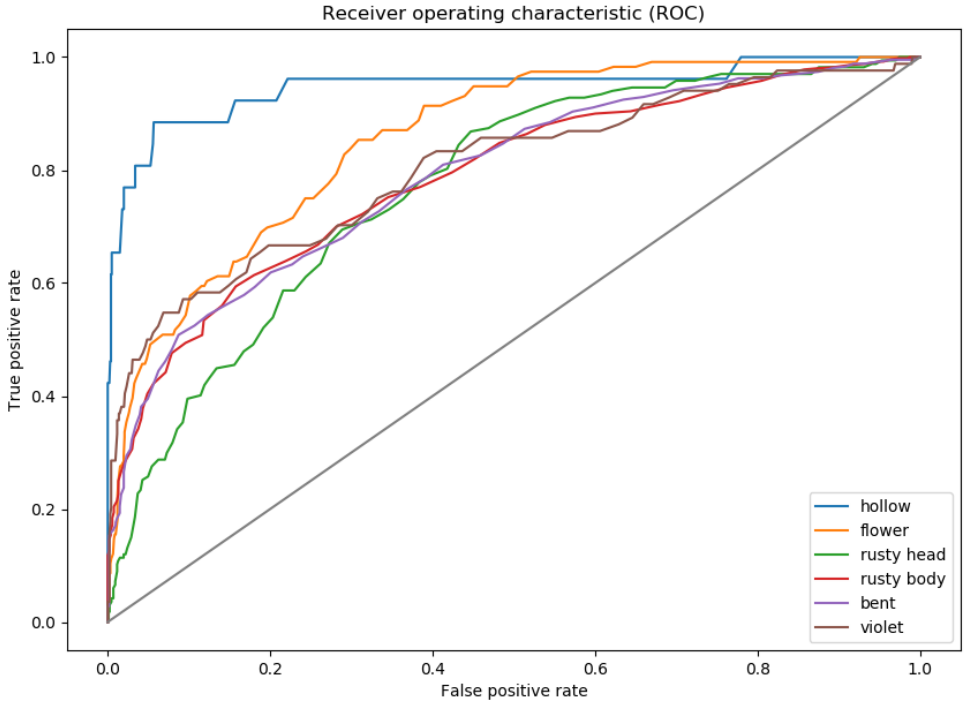


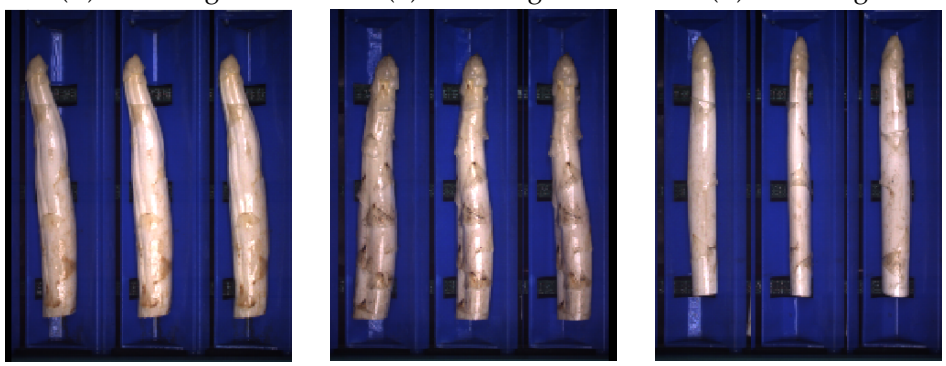
FIGURE 4.7: **ROC Curves for Six Feature Models** The figure shows the ROC curves for the six models trained to detect the features hollow, flower, rusty body, rusty head, bent, and violet, respectively. For the features rusty body and bent, the curve is comparatively smooth as their representation is more balanced. The curves are calculated from 1319 data samples.

detecting the feature fractured. The reason is that fractured asparagus without a head part was previously labeled as not classifiable (see the feature descriptions in subsection 3.2.1). The feature not classifiable was also trained on separately. Further, for all other features, not classifiable samples were removed before training to prevent a bias in the occurrence of false positives.[7]

In Table 4.3 the 13 features are listed on which the CNN architecture was trained 13 times. It further shows the results for the sensitivity, specificity, validation accuracy, and balanced accuracy of each feature after a certain number of training steps, indicated in the last column of the table.
Sensitivity is a measure to assess the performance of the model in labelling positive samples correctly, while the specificity shows how correctly the model predicts negative samples. The validation accuracy is the accuracy of the validation set which is a representative subset of the entire data set. The balanced accuracy of a feature label is the mean of the sum of its sensitivity and specificity. It represents the accuracy of the feature label if positive and negative samples in the data set were evenly balanced. Additionally, the performance of six models is calculated in a

[7] Not classifiable asparagus was not sorted for the presence of any other features (see subsection 3.2.8). If a sample is sorted by a model, e.g. that is detecting the presence of feature bent, and the sample shows the sorting criteria, i.e. it is bent, the model will classify it as positive but its feature label will be negative. This will disturb the model and, thus, it was decided to exclude not classifiable samples with an exception for training on feature fractured and for feature not classifiable.

(A) False Negative (B) False Negative (C) False Negative



(D) False Positive (E) False Positive (F) False Positive

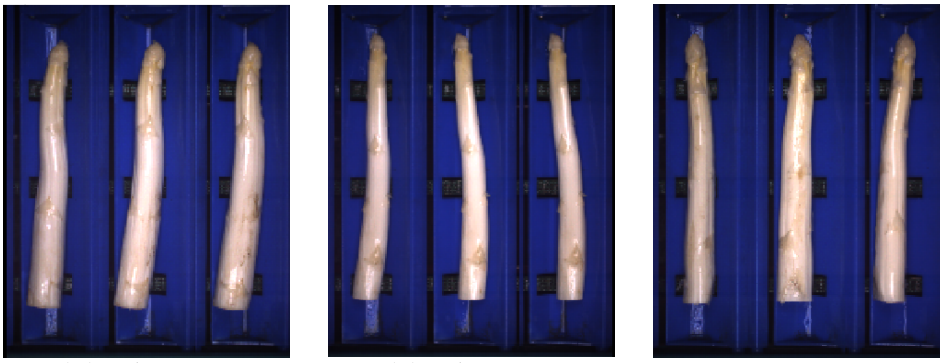
FIGURE 4.8: **Feature Hollow** Randomly chosen example images of false negatives and false positives of the feature hollow after 5400 training steps. Image (A) and (C) might have been sorted wrongly as the presence of the feature hollow is not obvious. Of the false positives, image (E) might have also been sorted wrongly, as the vertical line can be seen on the lower part of the asparagus which is a sign for a hollow asparagus (for reference, see the section Hollow). The thickness of the asparagus might have been an indicator to the model that the feature is present. This can be seen in image (D) which shows a thick asparagus, and image (F) where the asparagus' thickness varies depending on its position.

ROC curve in Figure 4.7.[8] Models that determine features which indicate the thickness and length, as well as the feature not classifiable are excluded from the curve. In the following discussion, it is referred to the best result of a model (depicted in brackets in Table 4.3) and not the last result.

The results reveal that for every feature, the sum of sensitivity and specificity exceeds 1. This corresponds to a balanced accuracy over 50%, which is better than chance level. For all features, the balanced accuracy is above 65%. Best results are achieved for features that indicate the thickness of an asparagus. The feature very thick reaches the best results with 98% sensitivity, 99% specificity, and a balanced accuracy of 98.5%. Besides features for thickness, the best prediction is observed for the feature fractured, which relies on the parameter length. It reaches a sensitivity of 88%, a specificity of 99.8%, and a balanced accuracy of 94%. On average, for the hand-labeled features hollow, flower, rusty head, rusty body, bent, and violet a balanced accuracy above 72% is reached. Feature rusty head performs worst with 52% sensitivity, 81% specificity, and 67% balanced accuracy. In general, the specificity of all features is relatively high. Most features reach a specificity above 90%, except for the features flower (83%), rusty head (81%), rusty body (80%), and bent (73%). For all features, the sensitivity is above 50%. After visual inspection of test

[8] For an explanation of ROC curve, see Figure 4.5 in the previous section.

(A) False Negative (B) False Negative (C) False Negative



(D) False Positive (E) False Positive (F) False Positive

FIGURE 4.9: **Feature Bent** Randomly chosen example images of false negatives and false positives of the feature bent after 3240 training steps. In all cases the presence of the feature bent is disputed. This becomes evident when comparing false negative and the false positive examples. The difference of both is not clearly evident (see the section Curvature for a description of feature bent). It gives the impression that the human labelers did not sort with a strong threshold regarding the feature.

loss and training accuracy, none of the models showed any form of overfitting in their respective range of training steps. Further, example images of wrong classification are shown for feature hollow in Figure 4.8 and for feature bent in Figure 4.9. Additional example images of false negative and false positive classification for the other features can be found in the appendix in subsection B.2.1. For feature fractured, training and test loss as well as training and test accuracy are plotted as an exemplar in the appendix in Figure B.13.[9]

The results indicate that the CNN architecture is able to learn every feature. Features relying on the parameters of length and width achieve a good performance, with both validation accuracy and balanced accuracy above 90%. The prediction of features like hollow or flower was expected to be more difficult. However, the balanced accuracy of both is above 75%, with feature hollow even reaching a balanced accuracy of 88%. This shows that the model predicts them relatively well. Features that depend on color (like rusty body, rusty head, and violet) do not reach equivalent results. It should be further tested whether an increase in model depth might lead to better results for features depending on color or for features depending on a more complex shape (like the feature bent).

[9]The models and their log files for accuracy and loss can be found at https://github.com/CogSciUOS/asparagus/tree/FinalProject/classification/supervised/single-label-CNN/asparanet.

An inspection of false positive and false negative images at the end of the training process suggests that training performance might be influenced by mislabeled data to a certain extent. The random images for feature hollow in Figure 4.8 propose that the model might use the thickness of an asparagus as an indicator. Some of the samples look like hollow asparagus that could have been labeled incorrectly. For the feature bent, the randomly selected examples in Figure 4.9 might reveal a labeling bias by the human annotators. When the feature is only slightly present, it seems to become a random choice whether the sample was labeled as positive or negative by the human labelers. If true, this can make it difficult for the machine to perform above a certain level of accuracy. It might also be that the difference between the samples is not prominent enough to the model. Again, these images are just randomly chosen examples and can mean that the model is simply not able to label them correctly. However, the results can also suggest that correct labeling might sometimes be difficult for the model because of an inconsistency in the labeling behavior of the human labelers (see subsection 3.4.3).
On a general note, the architecture of the model is very flexible. It can be applied to many tasks (i.e. predicting different features) without much preprocessing of the image data beforehand. Further, the model is quite small, which makes it fast and robust for practical applications. However, instead of having the same architecture for all features, more precise adjustment of each model to its feature is needed.

In conclusion, the results of the single-label CNN seem promising. Due to a very long debugging phase, there was not enough time to further test and improve the performance of the model. In turn, this means there is still a lot of fine-tuning and possible improvement. A restriction poses the labeling bias of the manually labeled images.

4.1.3 Multi-label classification

Building on the standard single-label classification we were further interested in how well a model, that predicts several feature labels at the same time, performs. A multi-label classification model hereby gets an image as the input and learns to predict the presence or absence of the feature labels. For this model, we use a small CNN as described below and the features that we labeled by hand. Each of the six features (hollow, flower, rusty head, rusty body, bent and violet) is encoded by a binary output in the target vector, indicating whether the asparagus exhibits the feature in question or not.

Multi-label classification is a useful tool for classification problems in which several classes are assigned to a single input. In contrast to a multiclass classification, where the model is supposed to predict the most likely class for an input, the multi-label classification makes a prediction for each class separately, determining whether the class is present in the image or not. While the different classes are mutually-exclusive in the multiclass classification, they can be related in the multi-label classification. Further, there is no limit on how many classes can be depicted in one image. It is possible that all or none of the classes are present.
Multi-label classification tasks can be thought of as consisting of different sub tasks. Therefore, the problem can be transformed to multiple binary classification tasks. In this transformation, a new model for each feature is trained, which are then combined to give a single output. That means that all features are independent of one another because they are learned separately. This can be seen as one of the major drawbacks as it is not always clear whether features are related, but in many cases they are. Therefore, we decided to not only use single-label classification as

described in subsection 4.1.2 but to explore the possibilities of multi-label classification.
A second approach to transform a multi-label classification task is to interpret each possible combination of features as on class. Hereby, the problem is redefined as a multiclass classification task. For a classification problem with six features that means there are $2^6 = 64$ classes to be learned. The problems with this approach are, on the one hand, the exponentially increasing number of classes, and on the other hand, the sparsity of samples per class. In many cases, some of the classes are highly underrepresented or even empty. For that reason, we decided not to elaborate this approach further and implement a model for multi-label classification without transforming the task to a multiclass problem.

Inspiration for the model gave a blogpost (Franky, 2018) which aims to classify images of the MNIST fashion data set in the context of multi-label, rather than multiclass classification. The author altered the data set in such a way that each input image contains four randomly selected items from the MNIST fashion data set. The model then learns to predict which classes are present in the image. The target vector has ten values, one for each class, which are either 0 or 1 depending on whether that class can be found in the input image or not.
This model was chosen as inspiration for two main reasons. Firstly, it tackles a similar problem as ours and the number of classes is similar. Secondly, the model uses a data set of similar size. Despite the rather small data set in comparison to many other image classification problems, good results with an accuracy of 95 – 96% were reached (Franky, 2018). This leads us to think, it might be a model with a good complexity for our problem too, as it is complex enough to model the underlying distribution, but not too complex for the medium-sized data set.

A classical CNN was chosen for the multi-label classification task. It consists of five blocks of convolution layers with max pooling layers each followed by a global average pooling layer and a dense layer.

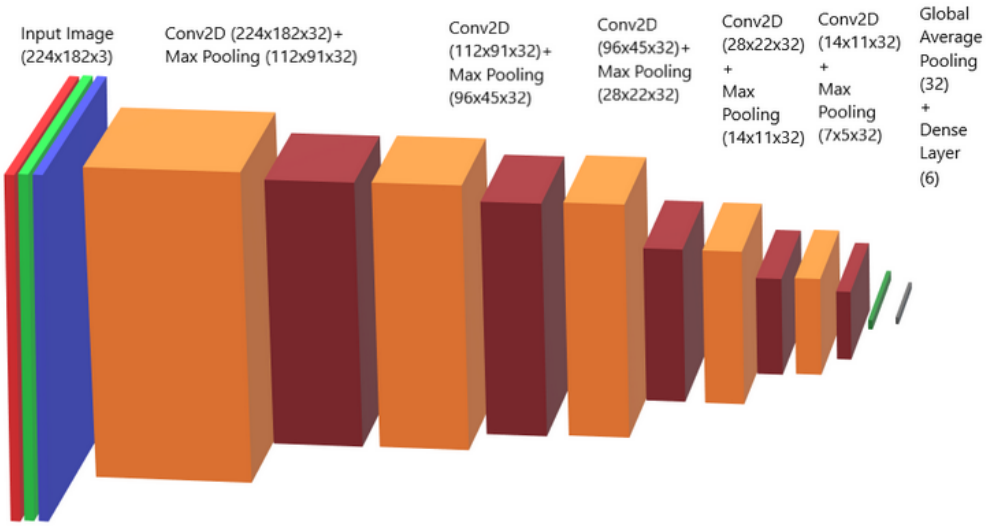


FIGURE 4.10: **Multi-Label Net Structure** The depiction shows the network structure of the multi-label CNN.

In contrast to multiclass classification models, where usually a softmax activation function is used in the last layer together with a categorical cross-entropy loss, the

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 182, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 112, 91, 32)	0
conv2d_2 (Conv2D)	(None, 112, 91, 32)	9248
max_pooling2d_2 (MaxPooling2	(None, 56, 45, 32)	0
conv2d_3 (Conv2D)	(None, 56, 45, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 28, 22, 32)	0
conv2d_4 (Conv2D)	(None, 28, 22, 32)	9248
max_pooling2d_4 (MaxPooling2	(None, 14, 11, 32)	0
conv2d_5 (Conv2D)	(None, 14, 11, 32)	9248
max_pooling2d_5 (MaxPooling2	(None, 7, 5, 32)	0
global_average_pooling2d_1 (	(None, 32)	0
dense_1 (Dense)	(None, 6)	198

Total params: 38,086
Trainable params: 38,086
Non-trainable params: 0

TABLE 4.4: **Multi-Label Model Summary** The summary of the multi-label classification model is shown. It describes which layers are implemented, how the output changes in each layer and how many parameters are trained in each layer and in total.

multi-label classification model uses a sigmoid activation function and a binary cross-entropy loss.
As the input of the model, a concatenated image of the three perspectives of each spear is used in order to maximize the information the model gets. This yields input images that look like the three asparagus spears are laying side by side. Further, the images are downscaled by a factor of six to facilitate training (see section 3.5).
The output of the model is a vector of length six in which each position encodes one of the six hand-labeled features (hollow, flower, rusty head, rusty body, bent and violet). Each feature can either be present in the input or not, which leads to a 1 or 0 in the target vector, respectively.
Three loss functions are tested to improve the model's performance. The first two losses are in-built functions from keras, namely binary cross-entropy loss and hamming loss. The latter uses the fraction of the wrong labels to the total number of labels. Additionally, a custom loss function was implemented, that penalizes false negatives stronger than false positives. The motivation for this custom loss was the fact that the two labels 0 and 1 are highly unbalanced. As previously stated, there are noticeably more 0s than 1s in many classes. To be more precise, the model can reach an accuracy of 77% by labeling all features as 0. By penalizing this error more, we intended to counteract the unbalanced data set. But at the end, the binary cross-entropy loss remains the one with the best results.
Further, it was tested whether regularization would improve the performance of the model on the validation data by preventing overfitting. For this, the model

was trained by adding L_1 or L_2 regularization, respectively, to all five of the convolutional layers. Hereby, a kernel regularization was implemented with a value of 0.01.
L_1 and L_2 regularization can both be interpreted as constraints to the optimization that have to be considered when minimizing the loss term. The main difference between the two is that L_1 regularization reduces the coefficient of irrelevant features to zero, which means they are removed completely. Hence, L_1 regularization allows for sparse models and can be seen as a selection mechanism for features. The inputs to this model are images that consist of a large number of pixels, and additionally a large portion of those pixels are black, because the background was removed. Therefore, it appears to be a good idea to reduce the number of features taken into account in the early layers. L_2 regularization, on the contrary, does not set coefficients to zero, but punishes large coefficients more than smaller ones. This way the error is better distributed over the whole vector.

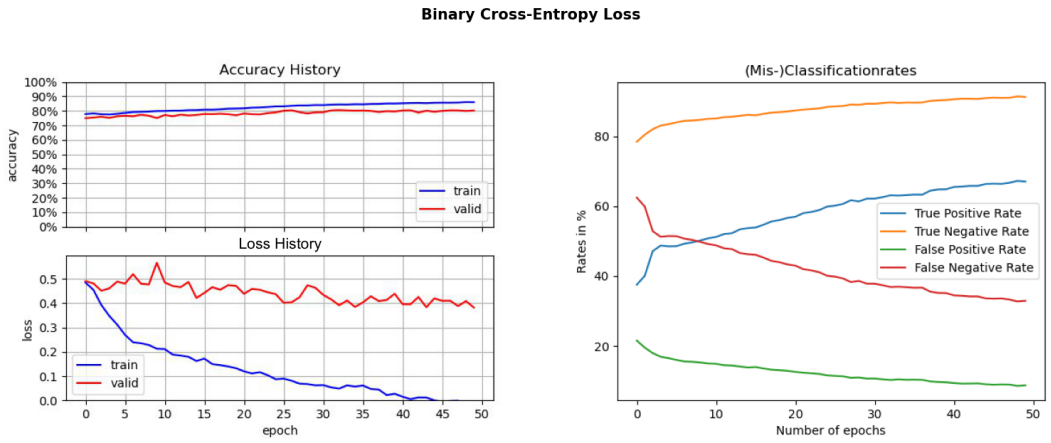


FIGURE 4.11: **Binary Cross-Entropy Loss** These graphs show the evaluation of the training with binary cross-entropy loss. The model was trained over 50 epochs and the accuracy and loss was measured. Further, the false/true positive/negative rates were determined.

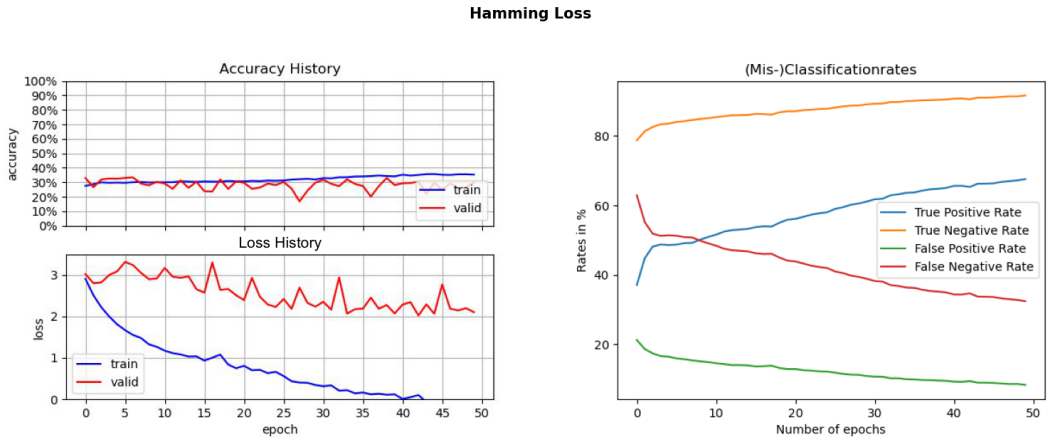


FIGURE 4.12: **Hamming Loss** These graphs show the evaluation of the training with hamming loss. The model was trained over 50 epochs and the accuracy and loss was measured. Further, the false/true positive/negative rates were determined.

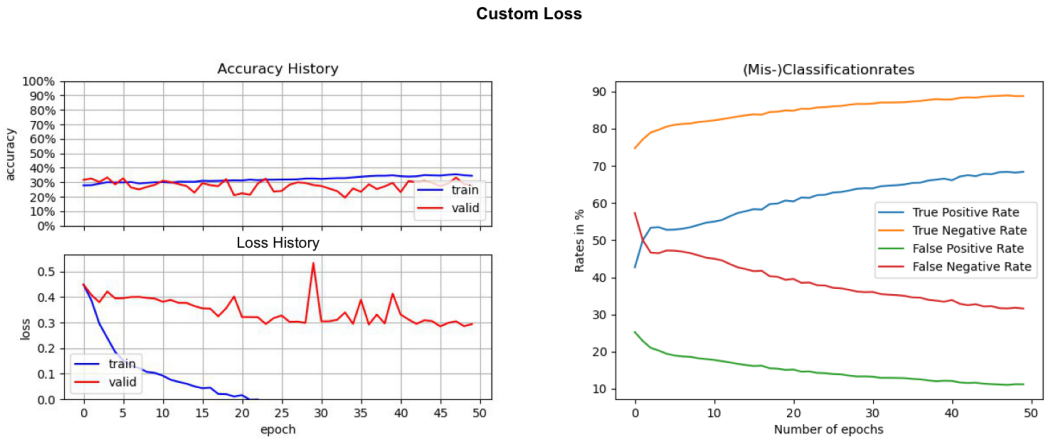


FIGURE 4.13: **Custom Loss** These graphs show the evaluation of the training with custom loss that punishes falsely classified ones more than falsely classified 0s. The model was trained over 50 epochs and the accuracy and loss was measured. Further, the false/true positive/negative rates were determined.

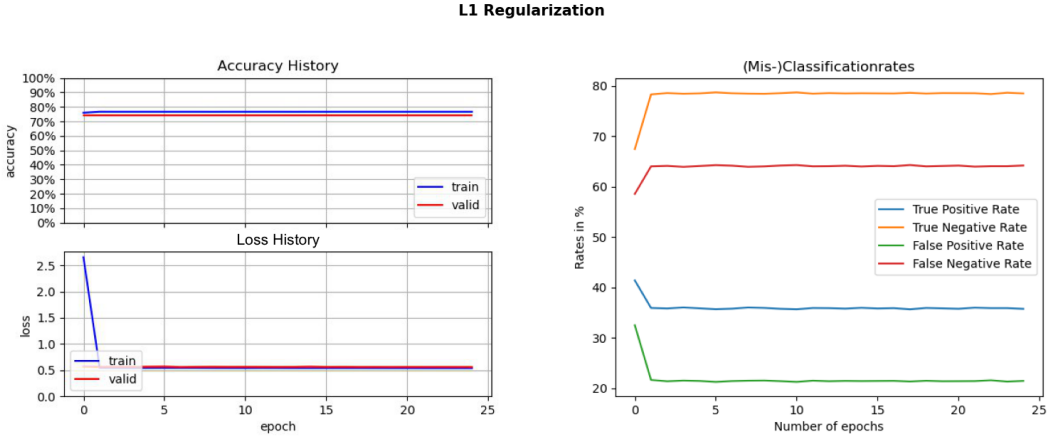


FIGURE 4.14: L_1 **Regularization** These graphs show the evaluation of the training with L_1 regularization. As the loss function the binary cross-entropy loss was used. The model was trained over 25 epochs and the accuracy and loss was measured. Further, the false/true positive/negative rates were determined.

As shown in Figure 4.11, Figure 4.12 and Figure 4.13, all the different approaches explained above show a similar behavior in accuracy and loss values. The training and validation accuracy increase slowly but steadily with the training accuracy always being a little higher than the validation accuracy. The training loss decreases rapidly, while the validation loss only decreases very little and shows random fluctuations. This can be an indicator for overfitting. Usually, L_1 and L_2 regularization are used to prevent overfitting, but in our case it did not improve the results, as shown in Figure 4.14 and Figure 4.15.
When looking at the sensitivity and specificity, it becomes apparent that both increase during the training process, while the false negative and false positive rates decrease with the same slope. The false positive and false negative rates are mirror images to the true positive and true negative rates with the mirroring axis at the 50% mark. It can be observed that the rates change rapidly in the first two to four

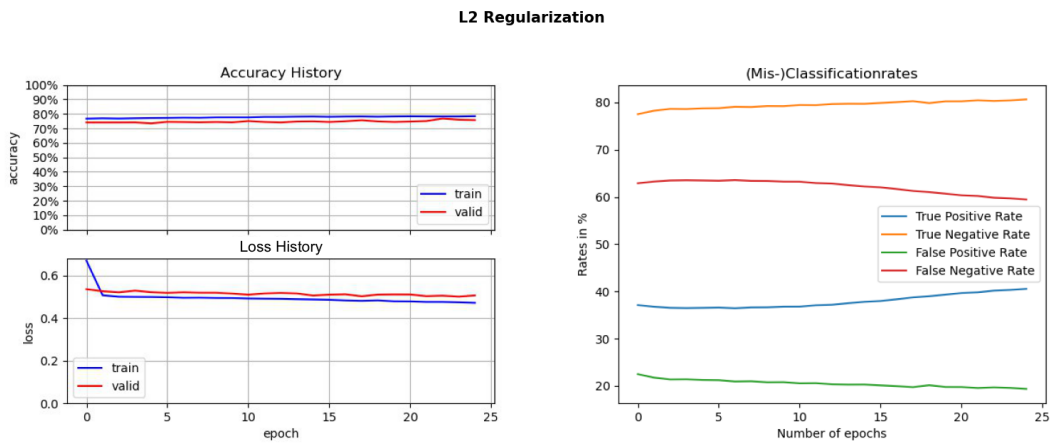


FIGURE 4.15: L_2 **Regularization** These graphs show the evaluation of the training with L_2 regularization. As the loss function the binary cross-entropy loss was used. The model was trained over 25 epochs and the accuracy and loss was measured. Further, the false/true positive/negative rates were determined.

epochs, after which the change progresses slowly in the same direction with no greater disturbances. The model trained with the L_2 loss is an exception as it does not show these large changes in either of the rates.

When comparing the three different loss functions, it is noticeable that the binary cross-entropy loss has significantly larger accuracy values than the hamming loss and the custom loss. The values of the binary cross-entropy loss start at 75%, while they start at around 30% for the other two loss functions. The behavior of the curves and the (mis-)classification rates, however, are very similar in all three approaches. The specificities start off very high with values around 78.46% for the binary cross-entropy loss, 78.73% for the hamming loss and 74.74% for the custom loss, and increase further during the training. The highest values are reached with the binary cross-entropy loss (91.41%), closely followed by the hamming loss (91.37%) and the custom loss (88.75%). The sensitivity values start off lower, at around 37% to 42%, and increase rapidly in the first few epochs, after which the rates proceed to increase but with a narrower slope. They reach values of up to 67.27% with the binary cross-entropy loss, 68.17% with the hamming loss and 68.17% with the custom loss. As stated above, the false negative and false positive rates show the same slope but in the opposite direction.
The accuracy values of the models that are trained with L_1 or L_2 regularization, respectively, do not change over the epochs. The same holds for the validation loss. The training loss decreases in the first few epochs and remains stable thereafter. While the (mis-)classification rates of the model trained with L_1 regularization behave similarly to the ones trained with no regularization, the rates of the model trained with L_2 regularization show a smaller increase and lack the fast change in the first epochs.
The slopes of all curves indicate that the model is learning, because they are increasing in the case of the accuracy, sensitivity and specificity and decreasing in the case of the loss, false positive and false negative rates until the end of training. Hence, one might think that a longer training period will lead to better results. However, the training loss decreases very rapidly while the validation loss does not. This suggests overfitting of the model, a problem which gets worse when increasing the

training steps. Therefore, a longer training period most likely will not increase performance unless overfitting is prevented. As shown in the result section, neither L_1 nor L_2 regularization alone were able to prevent overfitting.
Another common practice that can be tested is the drop-out, in which a certain amount of nodes are left out in different backpropagation steps. This way the model learns to not rely on a small number of nodes but distribute the information between all nodes available. Hence, the coefficients remain smaller. Another method to prevent overfitting is to reduce the model's complexity. A model with fewer parameters to train, is less prone to overfitting. A fitting degree of complexity should be found to model the data sufficiently good without losing the possibility of generalization.
Accuracy alone might not be a good indicator to evaluate a multi-label model (Gibaja and Ventura, 2015). As it highly depends on the loss function, it may have misleading results. This can be seen in the comparison between the three different loss functions. Although the sensitivity and specificity show similar values, the accuracy values suggest that the binary cross-entropy loss outperforms the other two loss functions by far. The accuracy of the model trained with the binary cross-entropy loss has an accuracy more than twice as high. But when looking at the slope of the curve, it appears that the model with the binary cross-entropy loss does not perform better than the other two models. All three have an increase of accuracy of roughly 10% and a similar sensitivity and specificity. This indicates that the slope of the accuracy function can be considered to evaluate the training process of the model, but the real values should be interpreted with caution.
One thing that comes to attention when looking at the (mis-)classification rates is that the sensitivities are a lot lower than the specificities. A reason for that might be that the positive values are more difficult to learn because they are only sparsely present. The model might have learned that, if the allocation to an output class is not clear, a 0 is the more likely guess.
The L_1 and L_2 regularization both seem to prevent the model from learning all together instead of only preventing overfitting. A reason for that might be that the regularization factor is too high. More experiments should be conducted with varying values to test this hypothesis.

In summary, the model improves its sensitivity and specificity, but it seems like it does so by overfitting the training data. Therefore, the next step should be to prevent the model from overfitting. Afterwards, it should be tested whether additional changes can improve the performance of the model further.

4.1.4 A dedicated network for head-related features

Some features relate to the asparagus heads only. Hence, it was assumed that classification is easiest when training a CNN on depictions of the respective region of the asparagus. Therefore, a data set that consists only of images of the head area is used. Images of all three perspectives are appended horizontally such that each sample contains the information from all available viewpoints. This is especially important as rust affected spots are sometimes only visible from certain angles. Figure 4.16 shows one sample of a rust affected asparagus head.

A simple feedforward CNN was trained on the images.[10] The features flower and rusty head are chosen as target categories. Hence, the model is an example for

[10] See https://github.com/CogSciUOS/asparagus/tree/FinalProject/classification/supervised/dedicated_head_network

FIGURE 4.16: **Training Sample** The depiction shows a sample for the preprocessed and vertically aligned asparagus head. Mind that images from all perspectives were stacked horizontally and used as a single input for the CNN.

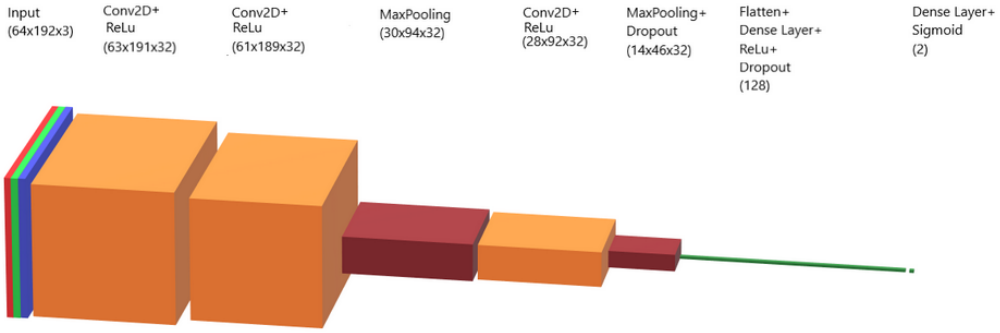


FIGURE 4.17: **Head Features Net Structure** The depiction shows the network structure of the CNN specifically aimed at head features.

multi-label classification. The network comprises the input layer, three convolutional layers with kernel size two, a fully connected layer with 128 neurons as well as the output layer. For the final layer, the sigmoid activation function is applied while the hidden layers have ReLU activations. A dropout layer was added to avoid overfitting. The network was trained using Mean Squared Error (MSE) as an error function. The development of loss in the learning curve indicates convergence after 40 epochs (see Figure 4.18).

The results for both features show to be highly specific. In contrast, the sensitivity is rather low. Only 55% of the asparagus spears labeled as flower are identified as such whereas the true positive rate is only 19% for rusty head. Given the low labeling agreement for these criteria (see subsection 3.4.3), these mediocre results are not surprising.
The ROC curve indicates how the classifiers respond to the introduction of a bias and shows the overall prediction quality. In Figure 4.19 the area under the curve is small for the feature rusty head. Beside incongruencies in the labels, this is possibly due to the choice of the size of the head region. It might be the case that brown

	False positive	False negative	True positive	True negative	Sensitivity	Specificity
flower	0.04	0.06	0.08	0.82	0.55	0.95
rusty head	0.02	0.13	0.03	0.83	0.19	0.98

TABLE 4.5: **Performance of Head Features CNN** Performance of the CNN trained on asparagus heads.

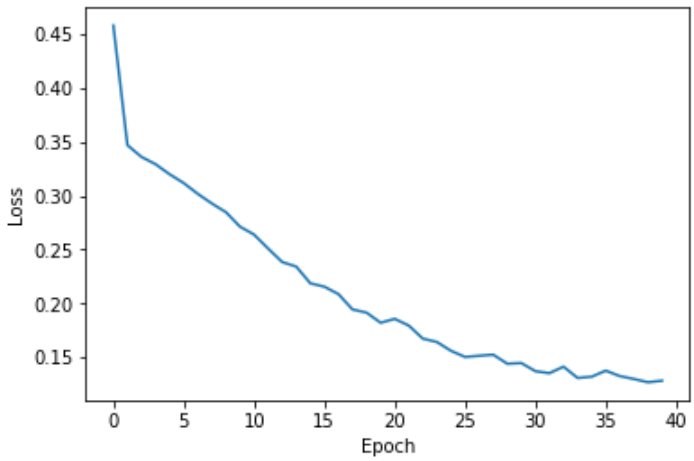


FIGURE 4.18: **Learning Curve for the Head Features CNN** The depiction shows the loss per training episode for the CNN trained on asparagus heads.

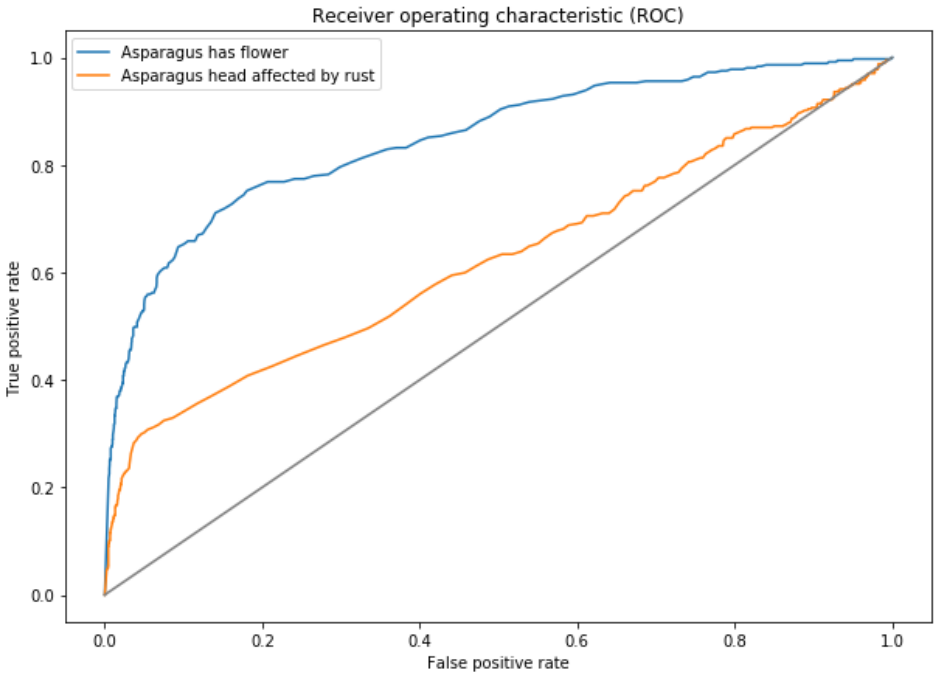


FIGURE 4.19: **ROC for the Head Features CNN** The depiction shows the Receiver Operating Characteristic (ROC) for the predictions of the CNN trained on asparagus heads. It allows us to compare the performance. A larger area under the ROC curve indicates better performance while a curve close to the diagonal line indicates poor results.

spots in regions other than the cropped part were considered as an indicator for a rusty head when attributing labels. Improvements by increasing the cropped head

region appear to be possible.

4.1.5 From hand-labeled features to class labels

Approximately 200 asparagus spears per class label are pre-sorted[11] and serve as ground truth mappings between input images and output class labels. We manually annotated the images with features (see section 3.4). This allows us to divide the classification process into two steps: In the first step we predict feature values from images and in a second step we predict class labels from those feature values.

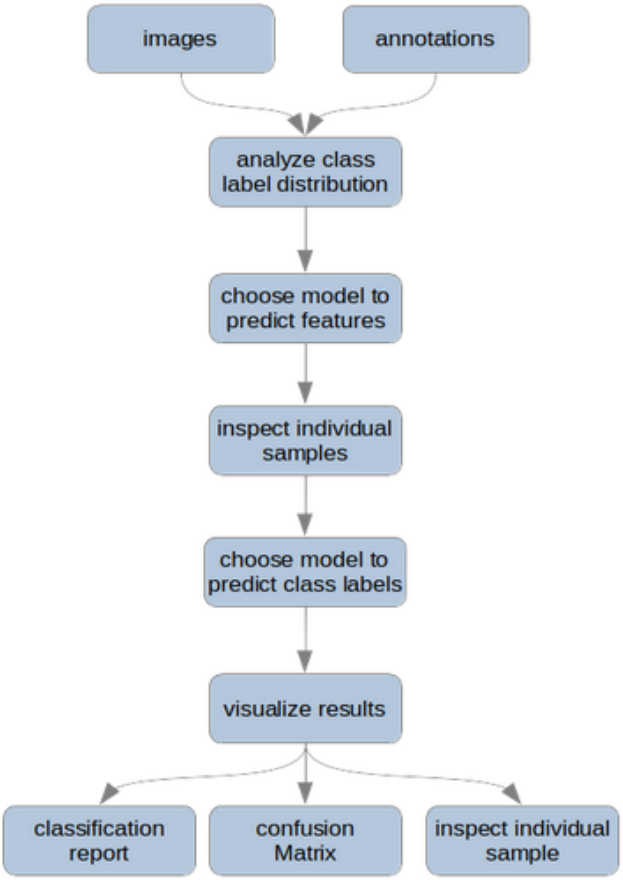


FIGURE 4.20: **Feature to Class Label Pipeline** The figure shows the flow of data and possible visualizations of our end to end solution app. We developed an app into which the user can load the asparagus images and the corresponding annotations. The app then visualizes the class label distribution. The user then can choose a model to predict the features of the asparagus pieces and inspect predictions for individually selected samples. Next, the user chooses a model that predicts class labels based on features of the asparagus. The app then presents a classification report, a confusion matrix and the possibility to inspect individual samples to visualize the prediction performance.

This chapter deals with the second step: Using supervised learning to predict 13 class labels, referring to the classes at the asparagus farm Gut Holsterfeld, based on the manually labeled features. We built a unified interface to load different models,

[11] Pre-sorted in this context means that the asparagus spears were first sorted by the sorting machine and, if needed, re-sorted manually by professional workers.

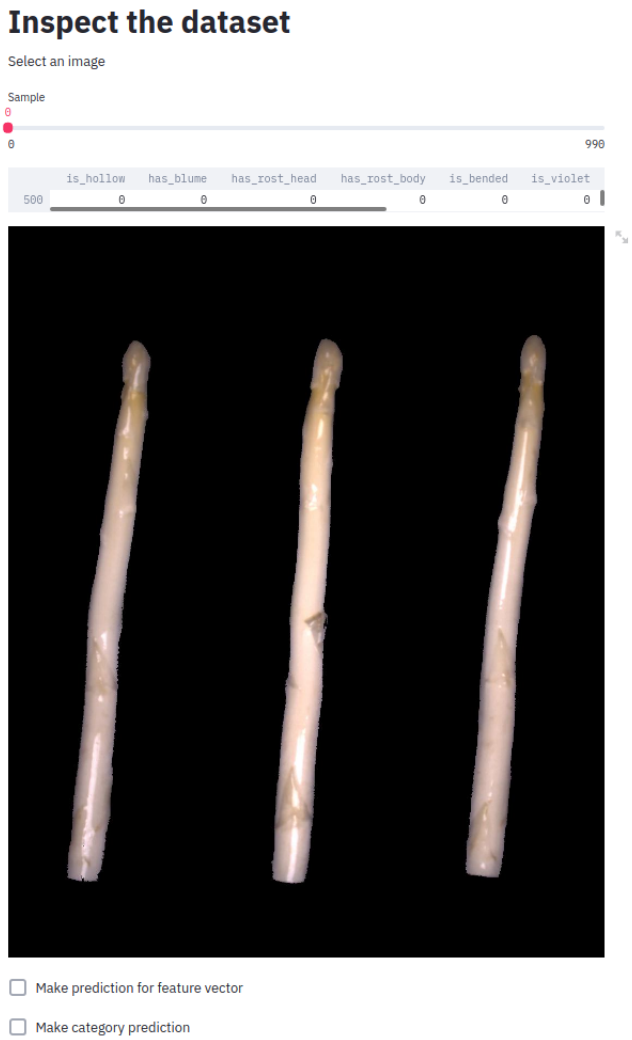


FIGURE 4.21: **Screenshot of Streamlit App** Screenshot of the streamlit app showing the three images of one asparagus spear with the corresponding labeled features.

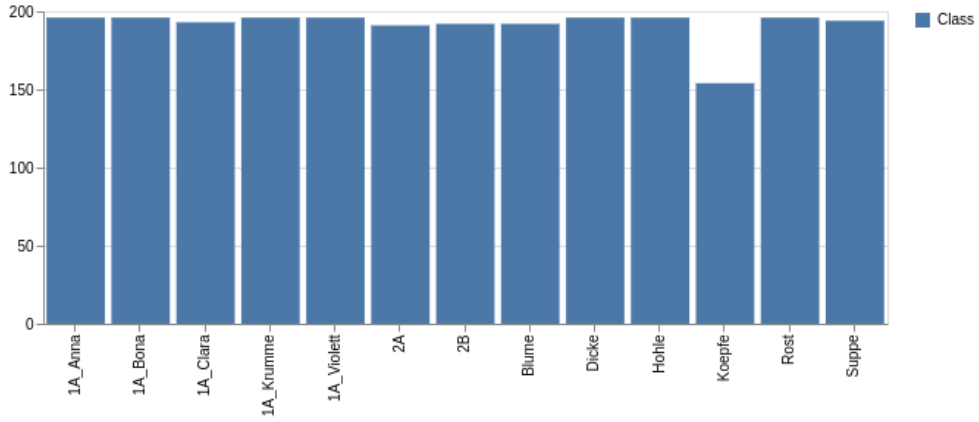


FIGURE 4.22: **Distribution of Class Labels** Absolute number of the asparagus spears for which a ground truth class label is available.

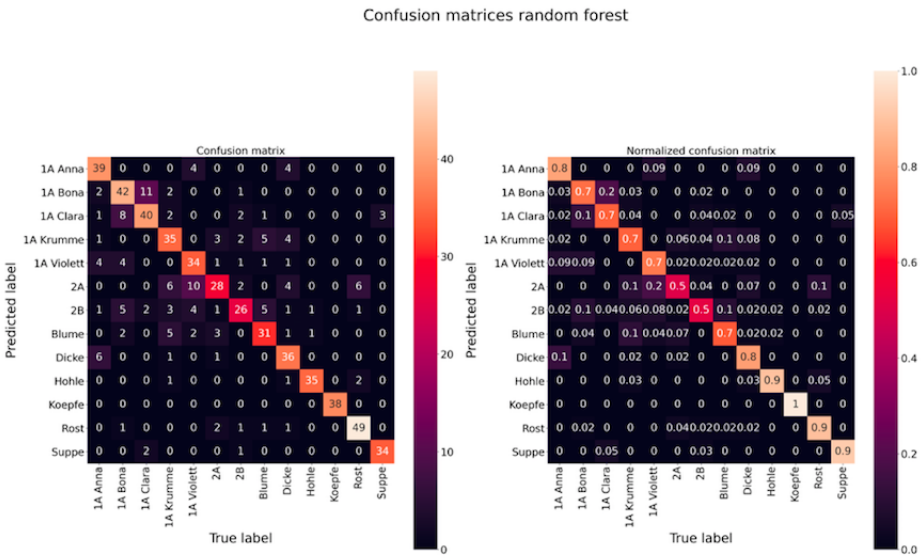


FIGURE 4.23: **Random Forest Classifier Confusion Matrices** Confusion matrices showing the absolute and relative number of true positives of the random forest model.

train them and analyze their predictions. It provides compatibility for scikit-learn as well as for keras models. To explore the data and visualize the predicted results, we built a streamlit app.[12] This has the advantage that the user can easily load and train a model, select an asparagus spear, and see the corresponding images as well as the predicted class label (see Figure 4.21).
The user can also inspect the distribution of the selected data (Figure 4.22) and the absolute and relative number of correctly and incorrectly classified asparagus spears in a confusion matrix (Figure 4.23).
The precision and recall are measures on how "useful and complete" the results are (Wikipedia contributors, 2020). Precision is the ratio between true positives and the set of all true and false positives.Recall is the ratio between the true positives and the set of all true positives and false negatives. The confusion matrix (Figure 4.23) gives us insight about which kind of errors the models make. We can observe that classes such as I A Anna, Dicke, Hohle, Rost, Köpfe, and Suppe can be recalled well (relative recall ≥ 0.8), while other classes, such as II A and II B have much lower recall ratings (relative recall ≤ 0.6). That means that II A and II B are the most commonly mislabeled classes (cf. Table 4.6).

Two exemplary models were implemented and tested to predict the classes from the features. The first one is a random forest (Breiman, 2001) with 100 trees.
The second model to predict classes from features is an MLP with six fully connected layers. It is only implemented to show how to integrate other models, but achieves a similar score as the random forest classifier when trained for 500 epochs (score of 0.76). However, it takes longer to train than the random forest classifier.
The score of 0.76 on the validation set is the accuracy of the random forest classifier, that means that 76% of the data is predicted correctly — random guessing would yield an accuracy of 0.08 for uniformly distributed classes.

[12] inside code/pipeline in the GitHub repository

	precision	recall	f1-score	support
I A Anna	0.7360	0.8936	0.8077	47
I A Bona	0.7000	0.7241	0.7119	58
I A Clara	0.7368	0.7368	0.7368	57
I A Krumme	0.6667	0.6400	0.6531	50
I A Violett	0.6481	0.7609	0.7000	46
II A	0.6842	0.4643	0.5532	56
II B	0.7179	0.5600	0.6292	50
Blume	0.6939	0.7556	0.7234	45
Dicke	0.7000	0.7955	0.7447	44
Hohle	0.9459	0.8974	0.9211	39
Koepfe	1	1	1	38
Rost	0.8305	0.8909	0.8596	55
Suppe	0.9444	0.9189	0.9315	37
accuracy	0.7588	0.7588	0.7588	0.7588
avg	0.7696	0.7722	0.7671	622
weighted avg	0.7591	0.7588	0.7549	622

TABLE 4.6: **Classification Report of the Random Forest Classifier** The classification report shows key metrics for the trained random forest classifier.

Although the results point into the right direction, it would be interesting to see how the selection of features and the agreement on the values of the manually labeled features influence the performance of the described classifiers. Further work is required to implement the decision process described by the local farmer (see section 1.3). One could test if the decision tree based on expert knowledge in Figure 1.1) can outperform the random forest trained on the training samples.

4.2 Unsupervised learning

Unsupervised learning is a class of machine learning techniques that deals with unlabeled data. More specifically, they work without a known goal, a reward system or prior training, and are usually used to find structure within data. Dimension reduction algorithms and clustering algorithms have been identified as the two main classes of unsupervised machine learning algorithms which are used in image categorization (Olaode, Naghdy, and Todd, 2014).

Multivariate data sets are generally high dimensional. However, it is common that some parts of that variable space are more filled with data points than others. A large part of the high dimensional variable space is not used. In order to recognize a structure or pattern in the data, it is necessary to reduce the number of dimensions. For this, both linear and non-linear approaches can be applied. Linear unsupervised learning methods for which also descriptive statistics can be acquired are e.g. Principal Component Analysis (PCA), Non-negative matrix factorization, and

Independent Component Analysis (Olaode, Naghdy, and Todd, 2014). Some examples for non-linear approaches are Kernel PCA (Olivier, Bernhard, and Alexander, 2006), Isometric Feature Mapping, Local Linear Embedding, and Local Multi-Dimensional Scaling. We employed the linear dimension reduction algorithm PCA as well as autoencoders for nonlinear dimension reduction.

4.2.1 Principal Component Analysis

Principal Component Analysis was chosen, because it is one of the standard unsupervised machine learning methods. Moreover, it is a linear, non-parametric method and a widespread application to extract relevant information from high dimensional data sets. The goal is to reduce the complexity of the data by only a minor loss of information (Shlens, 2014). Besides being a dimension reduction algorithm, PCA can also be useful to visualize or compress the data, filter noise, or extract features.

Our initial aim was to reduce the dimension of our data set for further models. As PCA was applied at the beginning of the data inspection, we also had the aim to visualize our data in a three-dimensional space, in order to get a better understanding of the data distribution. The images that comprise our original data set have a high quality, and instead of only reducing the pixel size, we aimed to reduce the information contained in named depictions. It was achieved by analyzing the principal components in a first step, followed by projecting all relevant images into the lower dimensional space. This information could serve as the input to supervised machine learning algorithms or as a simple lookup scheme to retrieve the label of the example with the most similar low dimensional representation.

PCA relies on linear algebra. A general assumption, especially with respect to images, is that large variances are accompanied by important structure (Shlens, 2014). The covariance matrix of the data reveals information about the overall structure and orientation of the data points in the multivariate space. The axis with the largest variance is set as the first principal component. An additional assumption is that the principal components are orthogonal to each other. Therefore, the second principal component is the highest variability of all directions which are orthogonal to the first one (Bohling, 2006). The covariance between each pair of principal components is zero, as they are uncorrelated. It generally holds that the higher the eigenvalues are, the more useful they are for the analysis. As eigenvalues specify the variance of the data, higher eigenvalues indicate more relevant features (Shlens, 2014).
The result of a PCA is a representation of the data in a new, smaller coordinate system, which depends on the axes of the largest variance. The dimensionality of this lower coordinate system should depend on the magnitude of the eigenvalues. When plotting the data along the axes of the principal components, it is often easier to understand and interpret the data, than in the original variable space.
It is widely agreed upon that PCA can be a good method to apply dimension reduction on images (Turk and Pentland, 1991; Lata et al., 2009). However, there are several different ways on how to do so. First of all, we performed the PCA on black and white images. When working on black and white images, only one data point per pixel is given. Therefore, performing a PCA is less computationally expensive, and finding structure is easier. However, as we need to be able to recognize violet as well as rust, and therefore be able to differentiate between color nuances, it was decided to work on colored images.

There are different possibilities regarding our project which can be considered as useful ways on how to perform a PCA. First, it can be performed on images of different classes at the same time – similar to capturing several images of several people in one database, on which one PCA is performed. In our case this would mean that a PCA is applied to a data set of several input images of all 13 class labels. The second way would be to perform a PCA separately on each class. This way, an "Eigenasparagus" in each class would be calculated, and distances between the Eigenasparagus of different classes could be measured. Thirdly, PCA can be employed feature-wise. In this case, the data set would consist of a collection of images with a certain feature present vs a collection of the same size with the feature absent.

After trying several different approaches, we decided to perform our final PCA on sliced RGB images with background. The images were labeled for their features with the hand-label app, as this yielded the best results. An amount of 400 pictures per feature was used to perform a binary PCA for each feature (either the feature is absent or present).
The 200 pictures where a certain feature is present as well as the 200 pictures where a certain feature is absent are extracted via loops over the csv-file, where all hand-labeled information is stored as well as the path to the labeled pictures. For each feature a matrix is created, storing 200 pictures with the present feature and 200 pictures without the feature. E.g., m_hollow is the matrix created for the feature hollow (shape = 400, img_shape[0] $\times$ img_shape[1] $\times$ img_shape[2]). The first 200 entries in the matrix are pictures of hollow asparagus. The last 200 pictures show asparagus, which is not hollow. These matrices were calculated for the features hollow, flower, rusty head, rusty body, bent, violet, length, and width. The data points of those 400 images in 2D space can be seen in Figure 4.32.

For all these features a PCA is calculated by first standardizing the matrix pixel-wise (dimensions: $1340 \times 346 \times 3$), calculating the covariance matrix, and then extracting the ordered eigenvalues. The principal components are calculated multiplying these eigenvectors with the standardized matrix. The feature space, the principal components, and the standardized matrices are saved to later perform a classification function. The highest ten eigenvalues are plotted to visually decide where to set the threshold of how many principal components will be further included. The first ten eigenvalues and the first ten Eigenasparagus for each feature can be seen in Figure 4.25 – Figure 4.31.
The classification function is a control function, which performs on unseen images and predicts if a feature is absent or present. It reads in a new picture of one asparagus, which is not part of the asparagus database, meaning not within the 400 images that were used to calculate the PCA. Then, it searches for the asparagus that is most similar to it in the feature matrices (which are 200 pictures of asparagus carrying the feature, 200 pictures of asparagus without the feature). This comparison is made with the reduced representation of the original pictures.
In greater detail, the input picture is first centered by subtracting the mean asparagus and then the picture is projected into the corresponding feature-space. That means the picture is translated into the lower dimensional space, in order to compare it to the known 400 pictures. The comparison is made through calculating the distance between the single centered Eigenasparagus and the 400 pictures in the feature space, by using the cdist function of the SciPy package. The smallest distance is considered as the most similar asparagus. If the index of the most similar asparagus is smaller than 200 we know that the feature is present, if it is above 200 the feature is absent. By comparing this to the information of the single asparagus

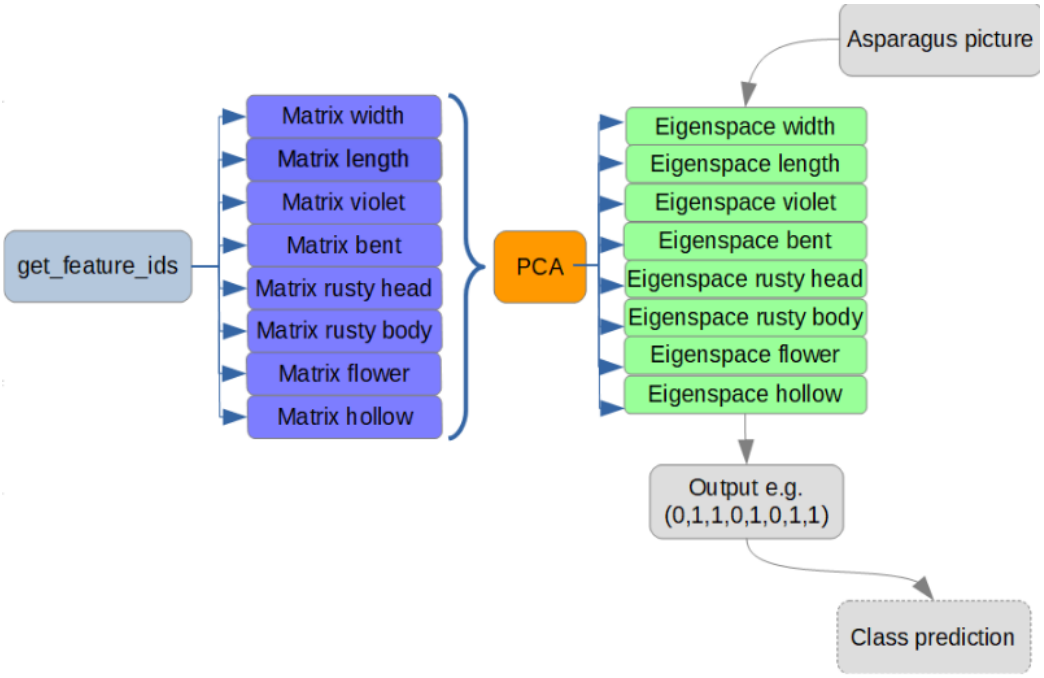
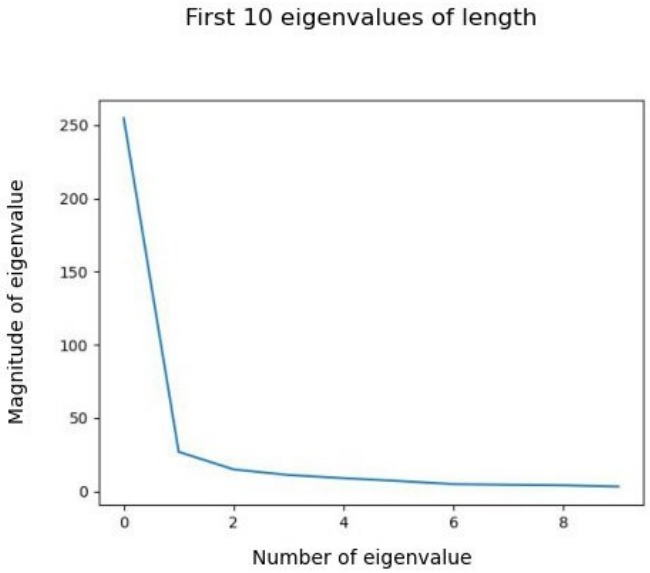


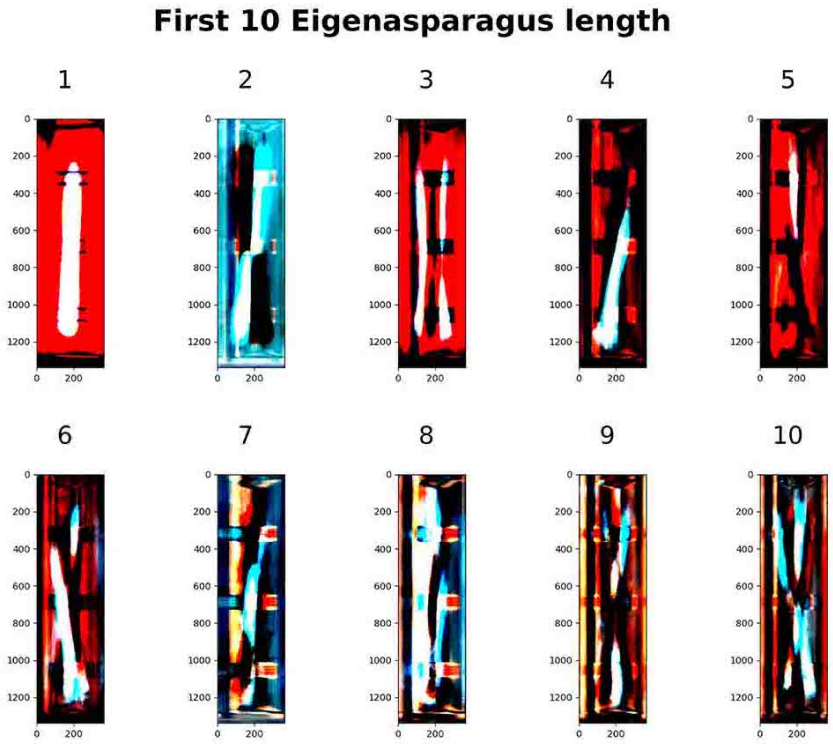
FIGURE 4.24: **PCA Process** Before the PCA can be conducted, 200 images per feature are identified by the `get_feature_ids` function. Then, a matrix for each feature is calculated. The PCA is conducted feature-wise. Thereafter, an Eigenspace for each feature is calculated. To evaluate the performance of the PCA, a verification process was performed. Therefore, new feature-labeled asparagus pictures are taken as input and by the help of all Eigenspaces, they evaluate if the feature is present "1" or absent "0". The result is compared to the label in order to evaluate the performance of the Eigenspace. To improve the algorithm, a pipeline of calculations can be implemented such that a feature vector is calculated. This feature-vector gives a class prediction. This is compared to the known label to the image.

spear, we know if the new asparagus has the same feature as its closest asparagus in the feature space, or not. By doing this for several images, we can already presume if the two features are likely to be easily separable or not. By evaluating this, we have a measure of how well our used principal components capture the distinguishing information of each feature.

The features on which results are given are: Hollow, flower, violet, rusty body, bent, width and length. All features are binary. For the first five features, the manually hand-labeled information is used. For the last two features (length and width), the information, which is automatically extracted by the algorithms used in the hand-label app is taken. The decision boundary for the first five features therefore depends on our predefined criteria on labeling (see section 3.2). The decision boundary for width is narrower or equal to 20 mm or wider than 20 mm, for length shorter or equal to 210 mm or longer than 210 mm.
There are no results for the feature fractured, even though it is one of the initial main features, as there were only five labeled pictures for this category. Therefore, it is excluded for further analysis. For the feature rusty head, a calculation problem emerged during the process. For an unexplainable reason, there occurred complex values in the calculation of the principal components. Due to time constraints,



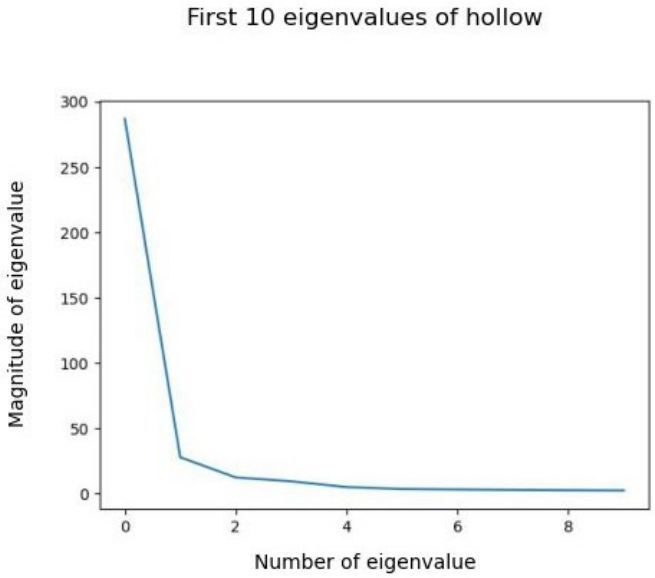
(A) This plots shows the magnitude of the first ten eigenvalues for the feature length.



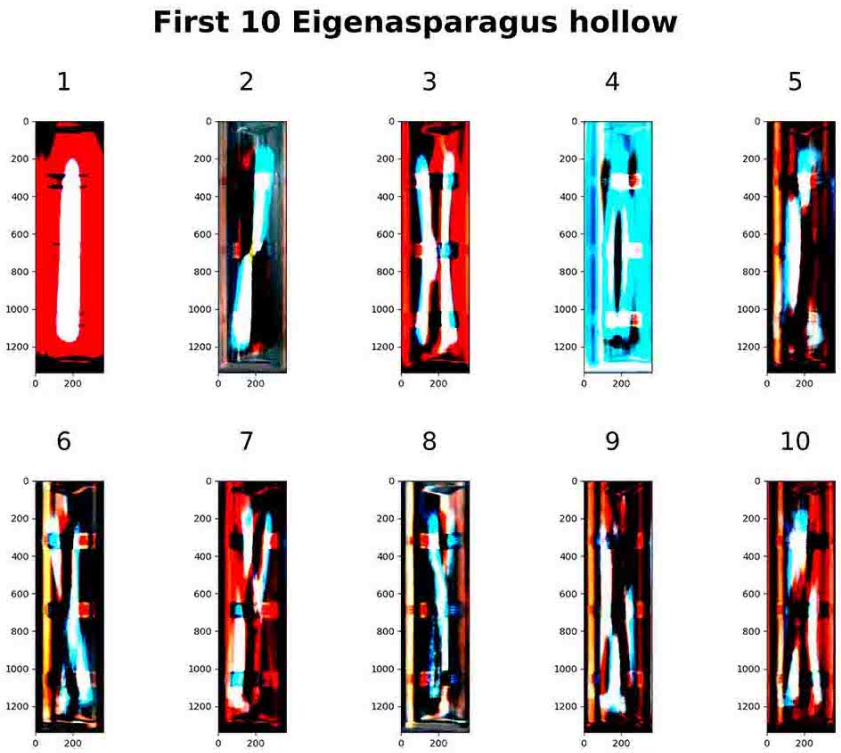
(B) These images show the first ten Eigenasparagus of the feature length.

FIGURE 4.25: **Eigenvalues and Eigenasparagus of Feature Length** (A) This plots shows the magnitude of the first ten eigenvalues for the feature length. (B) These images show the first ten Eigenasparagus of the feature length.

this problem is not solved, yet. Therefore, plotting of the feature rusty head is not possible.
By applying the approach for each feature, the eigenvectors, eigenvalues, and principal components for each feature are computed and stored. In all cases, the first



(A) This plots shows the magnitude of the first ten eigenvalues for the feature hollow.

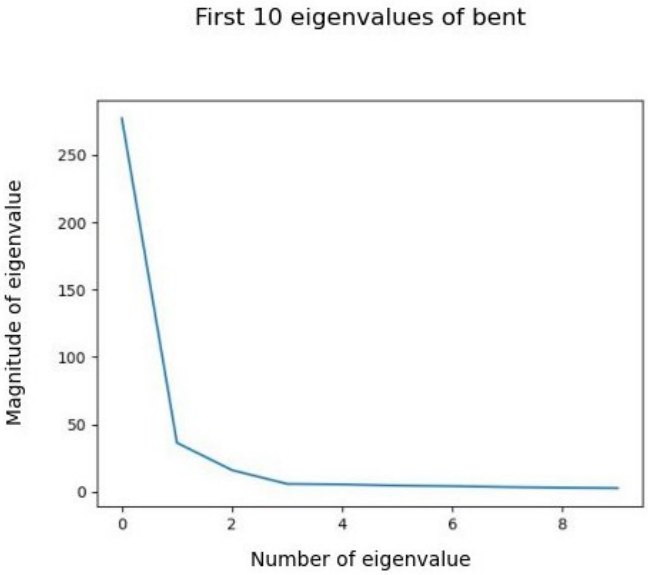


(B) These images show the first ten Eigenasparagus of the feature hollow.

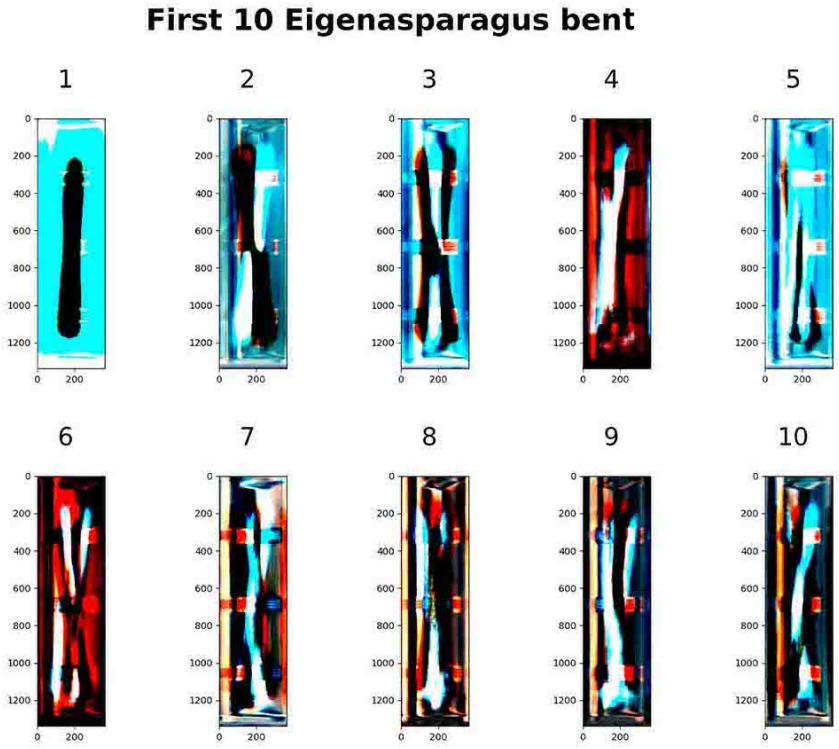
FIGURE 4.26: **Eigenvalues and Eigenasparagus of Feature Hollow**

principal component is quite high (between 286.94 and 254.78), and drops down rapidly afterwards. The values of the principal components after the fourth principal component are very low (≤ 9).

After inspection of the graph of the first ten principal components, only the first four principal components are used for the analysis, as there is either a sharp drop



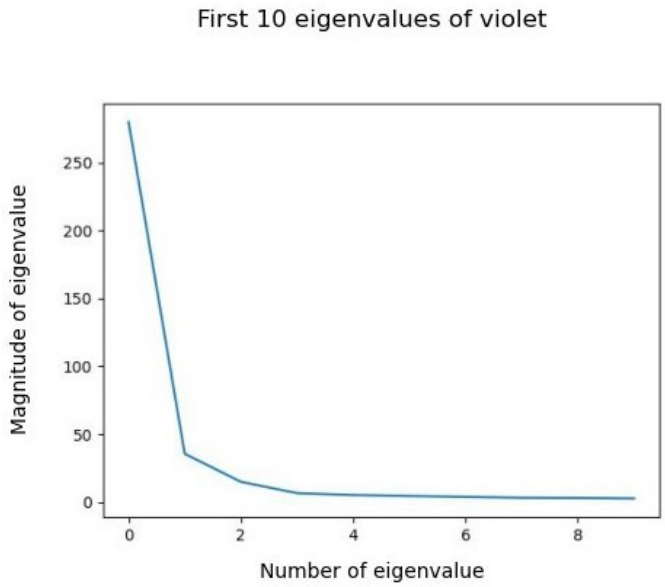
(A) This plots shows the magnitude of the first ten eigenvalues for the feature bent.



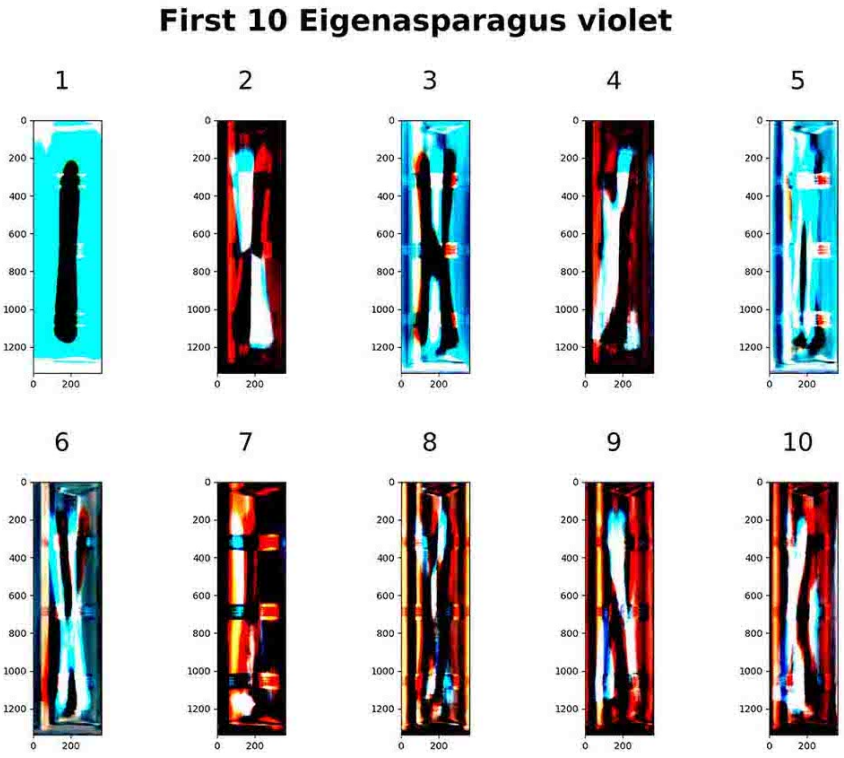
(B) These images show the first ten Eigenasparagus of the feature bent.

FIGURE 4.27: **Eigenvalues and Eigenasparagus of Feature Bent**

in the slope after the first four components or in order to maintain continuity between the different features. Following, the corresponding feature space is the space spanned by the first four principal components.
The scatterplots in Figure 4.32 show the data of each feature lined up along the axes of the first two principal components of each feature.
For the classification function, we used the same ten images, to test each feature. The classification works best for the features length and hollow (10/10 classified



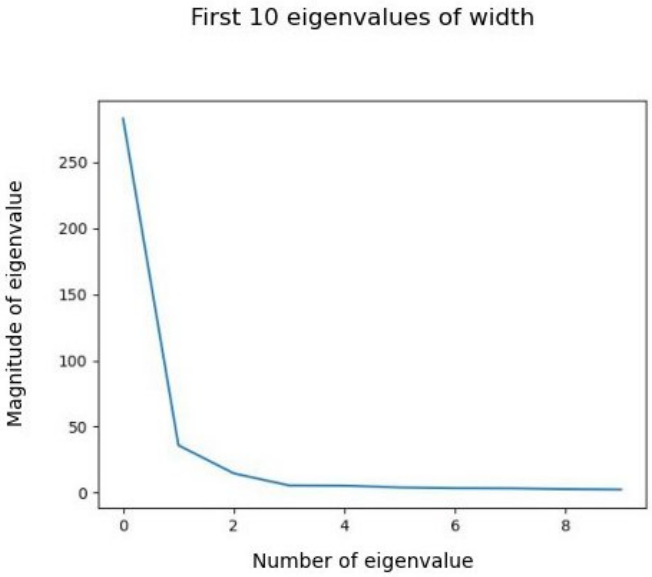
(A) This plots shows the magnitude of the first ten eigenvalues for the feature violet.



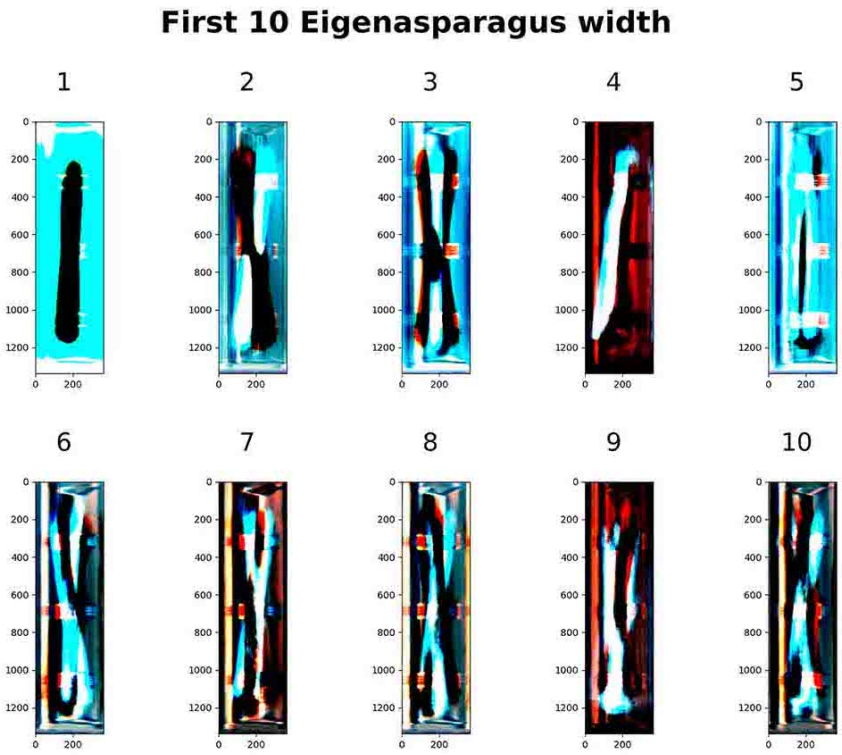
(B) These images show the first ten Eigenasparagus of the feature violet.

FIGURE 4.28: **Eigenvalues and Eigenasparagus of Feature Violet**

correctly) and then width (8/10 classified correctly). It performs around chance-level for flower (6/10 classified correctly), violet, and rusty body (5/10 classified correctly), and extremely poor for bent (2/10 classified correctly).
From the images of the first ten principal components, we can visually assume that there is information about the length and shape stored in the first principal component, as a clear asparagus spear can be seen. The following images leave a lot of room for interpretation, about what information is contained there.



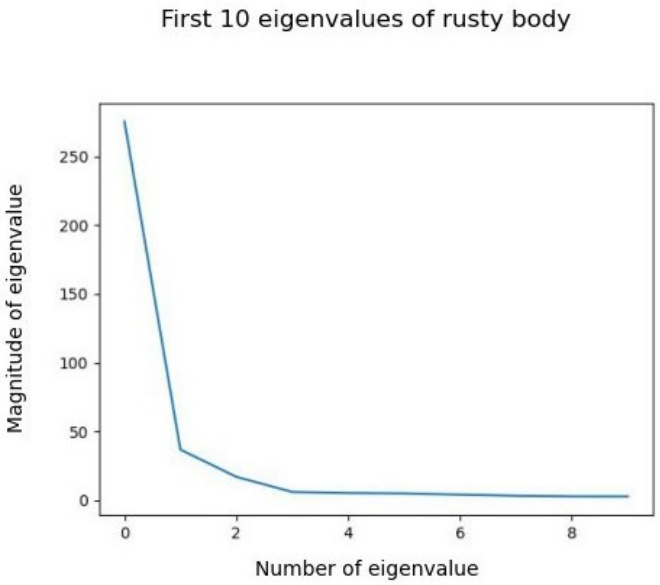
(A) This plots shows the magnitude of the first ten eigenvalues for the feature width.



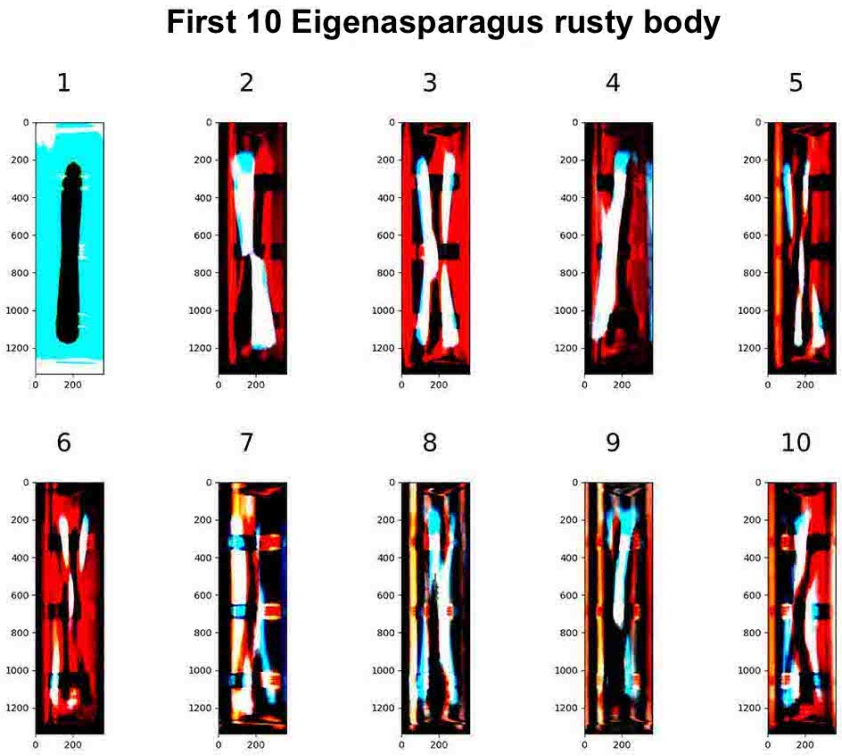
(B) These images show the first ten Eigenasparagus of the feature width.

FIGURE 4.29: **Eigenvalues and Eigenasparagus of Feature Width**

We performed the PCA on each feature separately to extract the principal components. It is interesting to see that the pictures of the different features are all very similar, see Figure 4.25 – Figure 4.31. One reason for this might be that many of the 400 input pictures for each feature are overlapping between the remaining features. Another reason might be that even though the images vary between features, the general information of all asparagus images is very similar.



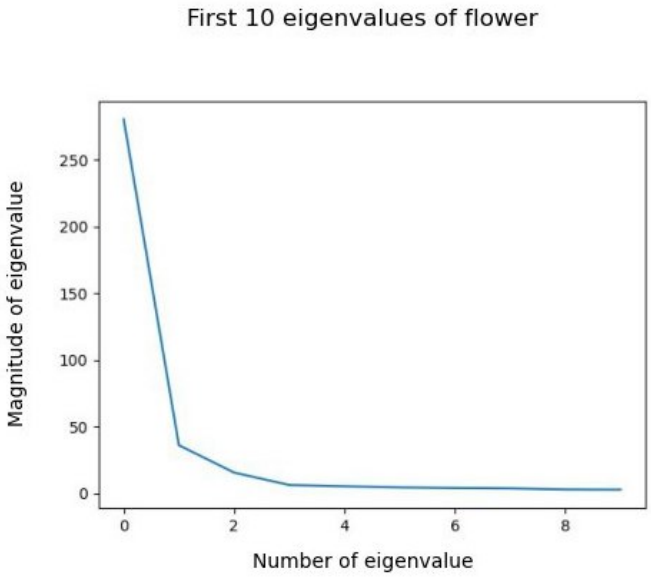
(A) This plots shows the magnitude of the first ten eigenvalues for the feature rusty body.



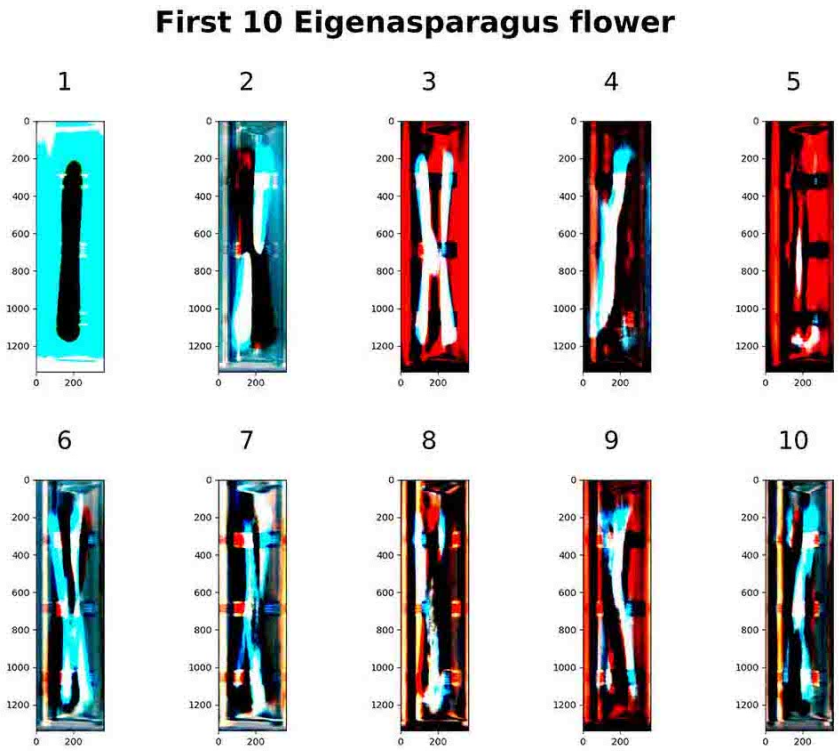
(B) These images show the first ten Eigenasparagus of the feature rusty body.

FIGURE 4.30: **Eigenvalues and Eigenasparagus of Feature Rusty Body**

From the results of the classification function, one can see that there are large differences between the features in how well our PCA performs (20% – 100%). One reason for this could be that certain features are simply more difficult to distinguish than others. Another reason for this large variation can be that certain features are also more difficult to label consistently (see subsection 3.4.2), and that the results are due to inconsistencies within the data. One indicator that this is a considerable



(A) This plots shows the magnitude of the first ten eigenvalues for the feature flower.



(B) These images show the first ten Eigenasparagus of the feature flower.

FIGURE 4.31: **Eigenvalues and Eigenasparagus of Feature Flower**

reason is that the performance of the width and length features, which is information that is not hand-labeled, is very high. Moreover, the poorest results can be observed for the features bent and rusty body. Those are the features, for which the agreement measures show the largest discrepancies between annotators (see subsection 3.4.3).

Another reason why the results are partly only moderate, is that RGB image data

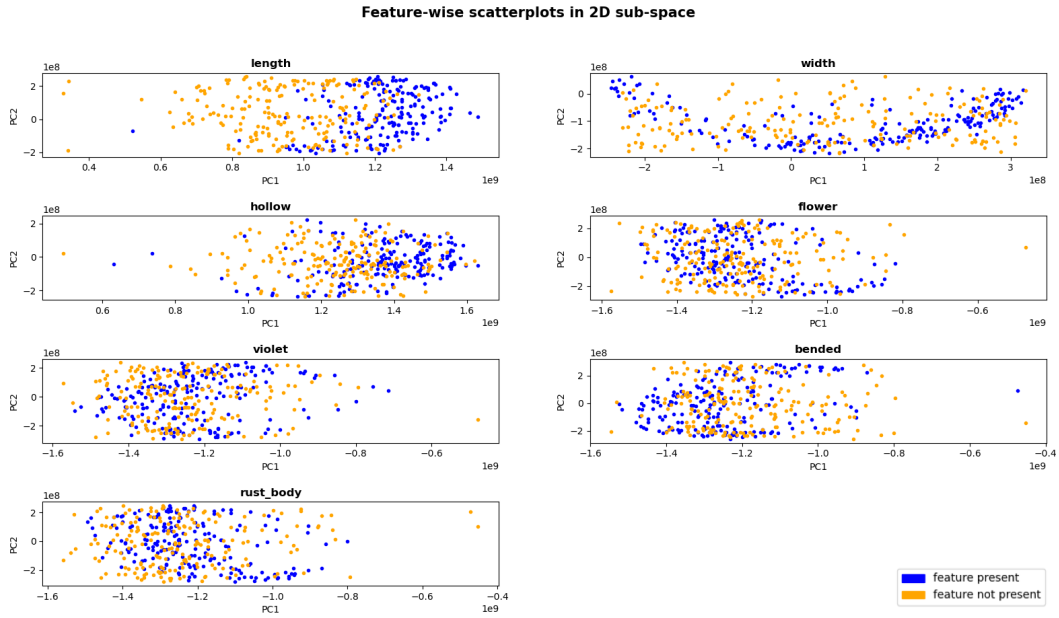


FIGURE 4.32: **Feature-wise Scatterplots** These scatterplots show the 200 data points where the feature is present in blue and the 200 data points where the feature is absent in orange. The data is projected into a 2D subspace, which is spanned by the first two principal components of each feature.

possesses complicated structures and by representing it in a linear, low dimensional feature space, it might be that simply too much information is lost. Even though there are papers reporting good results of PCA on image data (Turk and Pentland, 1991; Lata et al., 2009), there are other papers claiming that nonlinear dimension reduction algorithms are needed for this kind of image data (Olaode, Naghdy, and Todd, 2014).

4.2.2 Autoencoder

Beside PCA, there are further techniques for dimension reduction. An alternative that can be employed to deduce sparse representations and automatically extract features by learning from examples are autoencoders.[13]

Simple autoencoders, in which the decoder and encoder consist of MLPs, were already proposed as an alternative to PCA in the early days of Artificial Neural Network when computational resources were still comparatively limited (Kramer, 1991). Today one can choose from a multitude of network architectures and designs that all have one property in common: A bottleneck layer. For image classification it is common practice to use convolutional autoencoders. There are numerous papers about applications in various domains. Examples range from medical images to aerial radar measurements (Chen et al., 2017). The autoencoders employed include not only shallow networks but more recently the benefits of deep autoencoders were demonstrated as well (Geng et al., 2015). In addition, more complex architectures combining autoencoders with generative adversarial models were proposed lately (Bao et al., 2017). In many cases the purpose of autoencoders is dimension

[13]See https://github.com/CogSciUOS/asparagus/tree/FinalProject/classification/semisupervised/variational_auto_encoder

reduction and feature extraction. Hence the activation of the bottleneck layer neurons (the output of the encoder) is of main interest. In other cases, autoencoders are used to map images from one domain to another. For example, camera recordings can be mapped to pixel-wise labeled images. In this approach the labeled image can be retrieved from the decoder's output layer after successfully training the network (Iglovikov and Shvets, 2018). In short, there are many possible ways to apply autoencoders and also many possible architectures to realize them.
This motivates the question of how autoencoders work. As mentioned, all autoencoders have a bottleneck layer. If applied for dimension reduction, autoencoders are usually used to predict the input – in this case the image – with the input itself. The sparse representation corresponds to the activation of the latent layer for a given input. Autoencoders consist of an encoder that contains the initial layers as well as the bottleneck layer, and a decoder that maps the respective latent space back to the image. The desired mapping function of the input to a sparse representation is generated as a by-product of optimization in end-to-end training, as weights of the decoder are trained such that meaningful features are extracted. The main difference to PCA is that nonlinear functions can be approximated. Feedforward neural networks such as the encoder of an autoencoder are non-linear function approximators. Networks with multiple layers are especially well-known for establishing named nonlinear correlation. Hence, autoencoders allow for non-linear mappings to the latent space (Kramer, 1991). This means that in the latter, multiple features may be represented in a two dimensional space. It shows that compared to PCA, where one dimension typically corresponds to one feature, more information can be represented in fewer dimensions. Different properties of the input are mapped to different areas of the latent space.

For this project, we used a certain kind of autoencoder as basis, namely a Variational Autoencoder (VAE) for unsupervised data exploration. In VAEs, a special loss function is used that ensures features in the latent space are mapped to a compact cluster of values. This allows for interpolation between samples by moving on a straight line. Regions between points in the latent space lie within the data and hence reconstructions of the decoder are more realistic (Scikit-Learn, 2020). Other than that, VAEs share most properties with regular autoencoders. The location of a point in latent space refers to a compressed representation of the input. These can be interpreted as features of the input.
Different VAEs were tested in the scope of the project. First, a simple VAE with a MLP as decoder was implemented. The second approach was a comparatively shallow convolutional VAE. The third approach relates to a convolutional VAE with a deeper encoder that was later used as a basis to design the networks for semi-supervised learning. The third approach is more complex but does not improve the mapping of the properties of asparagus spears to a two dimensional latent asparagus space. Thus, only the results for the second of the three mentioned networks are reported in the following.
The network is derived from a standard example for VAEs (Scikit-Learn, 2020). It comprises two hidden convolutional layers in the encoder and two deconvolutional layers in the decoder.
Similar to the application of PCA, batches of images that contain only one perspective are used as input to the network. The downsampled data set is used. Images have to be padded as the implementation does not work with inputs of arbitrary shape. This is because the deconvolutional layers of the decoder can only increase dimensionality by an integer factor: The filters that are used for the deconvolution in the given network double the tensor dimensionality. An increase of the vertical dimension from 34 to 68 and finally to the desired 136 pixels is achieved in the

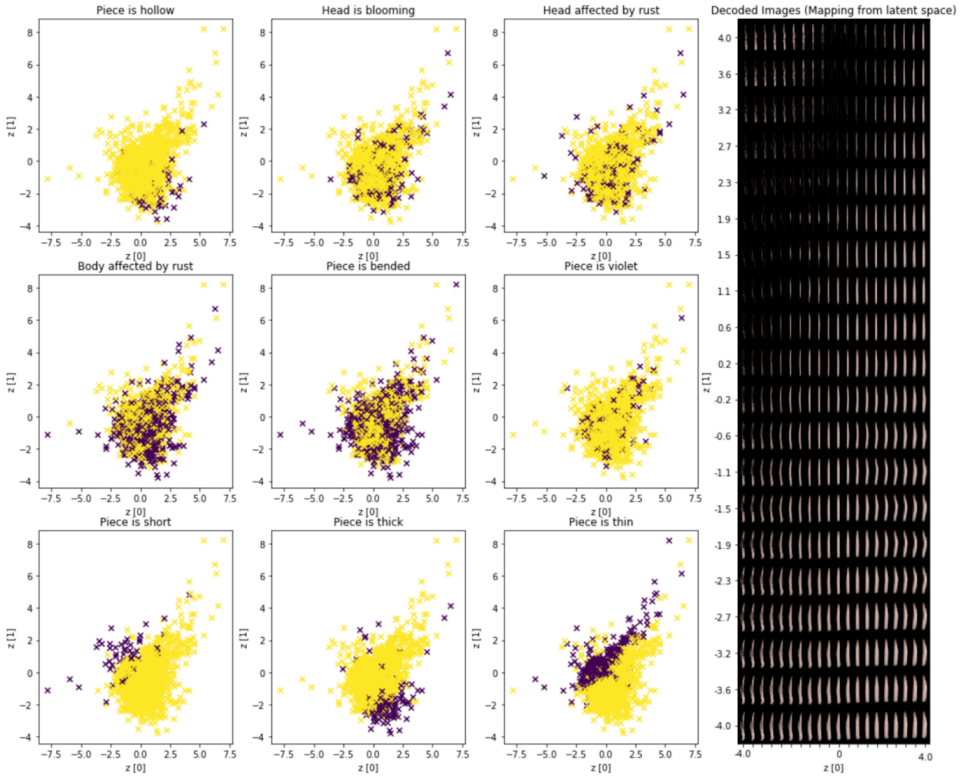


FIGURE 4.33: **Latent Asparagus Space for MLP-VAE and Reconstruction Manifold** The depiction illustrates the mapping by the encoder and decoder of the VAE: On the left you can see scatterplots illustrating the activation of latent layer neurons for the test data set (the mapping by the encoder). Image-features are mapped to the respective latent asparagus space. There is a scatterplot for each feature of interest where colors indicate positive (yellow) and negative (purple) samples. On the right a manifold of decoded images is shown. The axes relate to the points sampled in latent asparagus space that correspond to the reconstructions (mapping by the decoder).

last three layers of the network (which is impossible for the original height of 134 pixels). The input shape, which also defines the shape of the output layer, must be divisible by two without remainder twice.

Figure 4.33 shows the results. It demonstrates that the features short, thick, and thin are mapped to separable clusters. As a tendency the feature bent correlates with a region in the lower periphery, as indicated especially by the deconstruction depicted on the right side of Figure 4.33. The other features (violet, rust and flower) are not mapped adequately, thus, they are not visible in the reconstruction. This shows that only some features of interest are mapped to the latent space and used to decode images. Reconstructions of autoencoders are known to miss many details (Kramer, 1991). One may speculate that better results can be achieved using larger input images as we applied autoencoders on downscaled images.
The possibility to generate, for example, more or less bent asparagus spears may help to define a precise decision boundary and classify images accordingly. As a potential feature for asparagus sorting machines, this would allow the user to customize the definition of features such as bent to their own taste. For this approach

to be viable for all features, however, the network performance appears to be insufficient. Some features are poorly separated by the network.

4.3 Semi-supervised learning

We collected more than 100000 samples. Considering the uniform appearance of the images this represents a substantial amount. However, labels had to be manually generated. This was done for only around 10% of the samples. As a consequence, there is only a small subset of data with attributed labels. Smaller amounts of labeled data mean that predictions can be successful only if the variance in the source values is limited. Hence, for high dimensional data such as images, sparse representations are desirable. Extracting features automatically instead of relying on manual feature engineering is a strategy that is especially appealing if large amounts of unlabeled data are available.
In semi-supervised learning features are extracted from the input images in an unsupervised fashion. If labels are available for a sample they are used to ensure that the extracted features correspond to the target categories (Keng, 2020). Hence, semi-supervised learning promises better results by using not only labeled samples but also unlabeled data points of partially labeled data sets.

4.3.1 Semi-supervised autoencoder

In the previous chapter, methods and results for unsupervised learning are presented. One example is a convolutional autoencoder. In this section, it is shown how convolutional autoencoders with additional soft constraints in the loss function can be used for semi-supervised learning.[14] Instead of using unsupervised methods to compute another data set of sparse representations and use the latter to predict labels, in semi-supervised learning sparse representations are retrieved and mapped to latent features at the same time (Keng, 2020). Bottleneck layer activations represent automatically extracted features. For semi-supervised learning one tries to enforce that latent layer activations of autoencoders correlate with the target categories.

The general network structure is derived from the convolutional autoencoder used for unsupervised learning in subsection 4.2.2. The feedforward CNN is replaced by a network that has proven to be suitable to detect at least some features with sufficient adequacy when trained on asparagus heads (see subsection 4.1.4). It comprises three convolutional layers with 32 kernels, respectively. The first layer has a kernel size of 2×2, and the two subsequent layers have a kernel size of 3×3. A max pooling layer with stride size two is added mainly to reduce the number of total neurons while maintaining a high number of kernels. Additionally, a dropout layer is added to avoid overfitting. In contrast to other implementations for semi-supervised learning (Keng, 2020) the same network is used for the prediction of labels (when they exist for the current batch) as for the encoder that retrieves the sparse representation for reconstructing images. We chose the same decoder as in the Variational Autoencoder presented in the previous subsection 4.2.2. The effects of a bypass layer that contains neurons not being subject to the label layer

[14] See https://github.com/CogSciUOS/asparagus/tree/FinalProject/classification/semisupervised/semi_supervised_autoencoders

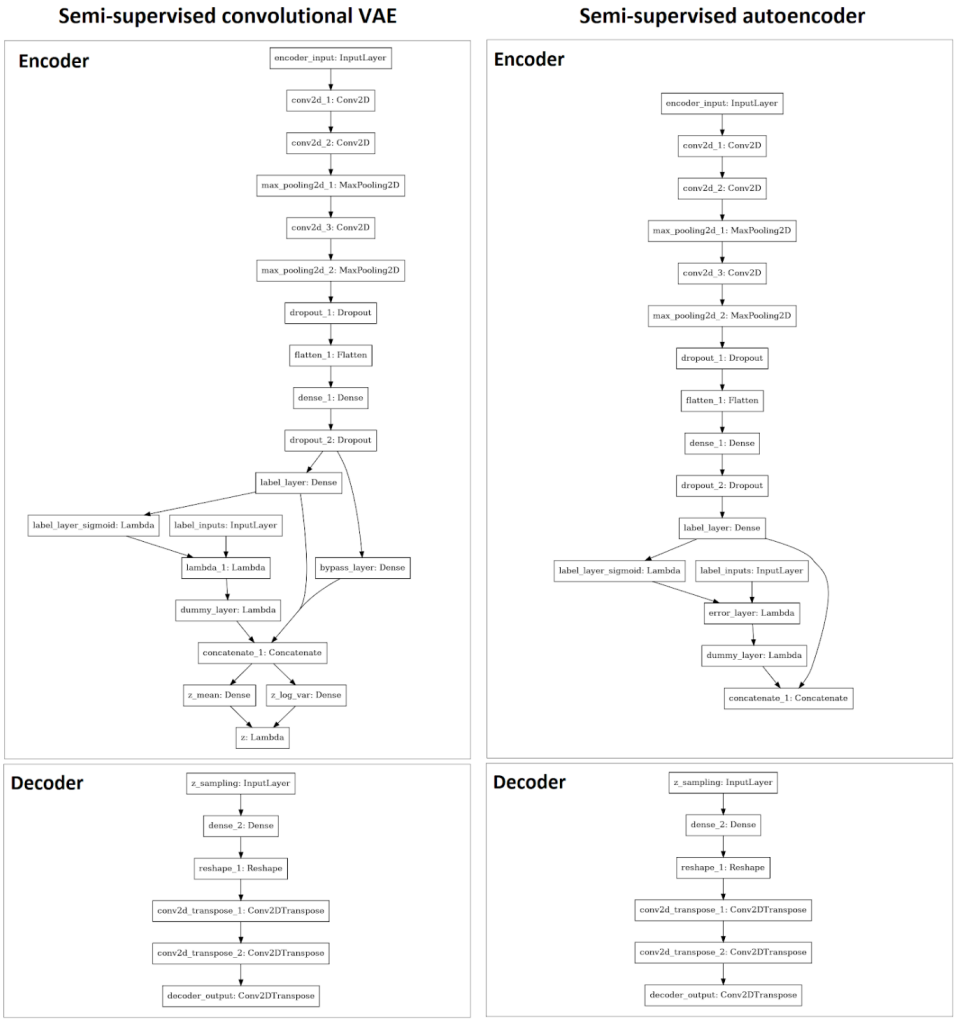


FIGURE 4.34: **Network Structures for Semi-Supervised Learning** The depiction illustrates the network structure for the autoencoders used for semi-supervised learning. Left: The structure for the semi-supervised convolutional VAE. Right: The structure for the semi-supervised convolutional autoencoder. Further explanations can be found in the text.

loss (see Figure 4.34) were tested. As accuracy did not improve, it was later dismissed. Two variations of the network were tested: A convolutional VAE for semi-supervised learning and a convolutional autoencoder for semi-supervised learning. The architecture for both networks can be seen in Figure 4.34.
A challenge results from training with multiple inputs. As deep learning frameworks usually require a connected graph that links inputs to outputs, a trick is used in order to handle that two tensors – images as well as labels – are given as an input. A dummy layer is introduced where all information derived from the labels is multiplied by zero. The output vector is concatenated with the bottleneck of the encoder. As it contains no variance, training and, more importantly, validation accuracies remain unaffected even though information about the categories to be predicted is added on the input side. Nevertheless, the labels are part of the network graph and could hence be used in the loss function.
A custom conditional loss function is used. If labels are present for the current batch, the custom loss is equal to a combined loss that comprises the reconstruction

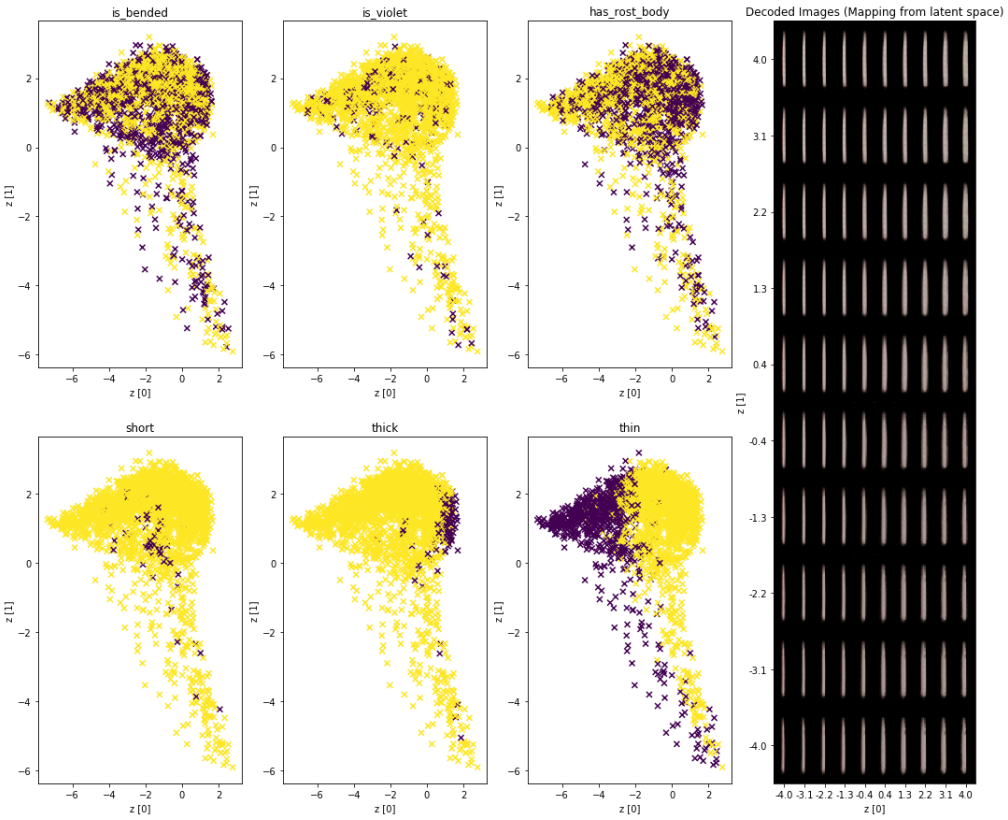


FIGURE 4.35: **Latent Asparagus Space for Semi-Supervised VAE** The depiction illustrates the mapping by the encoder and decoder of the CNN-VAE with additional constraints in the loss function for semi-supervised learning: On the left one can see scatterplots illustrating the activation of latent layer neurons for the test data set (the mapping by the encoder). Image-features are mapped to the respective latent asparagus space. Mind that there is a scatterplot for each feature of interest where colors indicate positive and negative samples. On the right, a manifold of decoded images can be found. The axis relates to the points sampled in latent asparagus space that correspond to the reconstructions (mapping by the decoder).

loss and the label loss. Here, reconstruction loss refers to the pixel-wise loss that is used for the main task of the network – namely, the mapping of input images back to the same images (fed into the network as target values to the output layer). The label loss is used with the goal of mapping label layer activations to the actual labels. It is low if activations in the sigmoid transformed label layer match the target values i.e. the sum of the error layer is low. The loss that is due to labels is multiplied by a custom factor (k). In addition it is defined such that it scales with the current pixel-wise reconstruction loss and converges to a constant (c). These values are chosen with the aim of increasing the contribution of the label loss to the combined loss, especially in late stages of training.

The results for the semi-supervised VAE are illustrated in Table 4.7 and visualized by Figure 4.35. As one can immediately see, the feature thin is adequately mapped to a decisive region in latent space. For the other features, no such clear cut clustering is visible. Reconstructions indicate that the main purpose of the network of

	False positive	False negative	True positive	True negative	Sensitivity	Specificity
bent	0.04	0.37	0.04	0.55	0.10	0.94
violet	0.00	0.08	0.00	0.92	0.00	1.00
rusty body	0.14	0.27	0.20	0.39	0.42	0.73
fractured	0.00	0.02	0.00	0.98	0.00	1.00
thick	0.00	0.07	0.00	0.93	0.00	1.00
thin	0.00	0.14	0.16	0.70	0.54	0.99

TABLE 4.7: **Convolutional VAE Performance** Performance of the semi-supervised convolutional VAE.

	False positive	False negative	True positive	True negative	Sensitivity	Specificity
bent	0.02	0.34	0.07	0.57	0.18	0.96
violet	0.00	0.08	0.00	0.92	0.00	1.00
rusty body	0.23	0.19	0.28	0.30	0.59	0.57
fractured	0.00	0.01	0.01	0.97	0.57	1.00
thick	0.00	0.06	0.00	0.93	0.04	1.00
thin	0.02	0.07	0.23	0.68	0.76	0.97

TABLE 4.8: **Convolutional Autoencoder Performance** Performance of the Semi-Supervised Convolutional Autoencoder.

predicting the input images is accomplished successfully although the reconstructions look rather uniform. Values for accuracy and sensitivity indicate only poor performance. Sensitivity is only above zero for the features rusty body (0.42), thin (0.54) and bent (0.1).
Compared to the variational autoencoder for semi-supervised learning, the simple convolutional autoencoder for semi-supervised learning performs better. However, there are substantial potentials for improvements. Table 4.8 shows a summary of results. Violet detection is not successful at all, as indicated by a sensitivity of zero. For the other features that the network was trained on, mediocre results are achieved. Thickness detection shows little sensitivity (0.04), however a high specificity (1.0). Better results in sensitivity exist for bent (0.18) rusty body (0.42), and short spears (0.6). The specificity for rusty body is lower (0.6) as compared to named other features (1.0 and 0.96). Thin spears are detected in 76% of all cases and few false positives characterize the detection of named feature (0.97 specificity).

The approach for semi-supervised learning presented here faces two challenges. First, the networks are trained only on one perspective although labels are attributed per asparagus spear – that is, for three concatenated images only one label exists. Information that might only be visible on one or two out of the three perspectives cannot be mapped to the desired target category. Image-wise labels would be desirable to improve the approach.
Second, reconstructions using convolutional autoencoders contain little detail. However, small details in the image, such as small brown spots that indicate rust, play an important role for classification. These features are not sufficiently reconstructed by the VAE. Arguably, they are hence not reflected in the sparse representation that corresponds to latent layer activations. One may speculate that a substantial increase of the network size would help to reconstruct more details and hence extract more features. As Generative Adversarial Networks (GANs) are known to generate more detailed images (Bao et al., 2017) they could possibly be adopted for semi-supervised asparagus classification with greater success. However, this is a question that must be answered empirically.

In summary, one may conclude that automatic retrieval of sparse representations with autoencoders appears as an alternative to manual feature engineering (rule based retrieval of sparse representations) if a large data set is available and only a subset contains labeled samples. However, more research is necessary to find best suitable network structures for asparagus classification.

Chapter 5

Summary of results

The study project was conducted to investigate the state of the art of asparagus classification aiming at the development of approaches that could help to improve the current sorting algorithm implemented in the Autoselect ATS II at the asparagus farm Gut Holsterfeld. Data was collected, preprocessed and then analyzed with seven different approaches. Out of our 591495 images, roughly corresponding to 197165 different asparagus spears, 13271 images were collected by re-sorting with the machine and 13319 images were manually labeled by us. This labeled data is considered for the supervised approaches. The semi-supervised approach is in addition based on approximately equally many unlabeled images, 20000 in total. The unsupervised learning approaches are based on roughly 5500 images. The results illustrate that classifying asparagus is not a trivial problem. However, the results also show that it is possible to extract relevant features that might improve current sorting approaches.

For supervised learning, we employed MLPs and CNNs. Whereas the former were trained on sparse descriptions retrieved by high level feature engineering, the latter were directly trained on preprocessed images. They include networks for single-label classification as well as multi-label classification. The feature engineering MLP for curvature prediction and the single-label CNN perform binary classification, whereas the CNN for the multi-label approach as well as head-related features network perform multi-label classification. All approaches aim to solve the same image classification problem, using supervised learning.
Each approach has drawbacks and benefits. The complexity and the requirement to specify many parameters has proven to be a disadvantage of relatively deep but also rather shallow CNNs as compared to e.g. MLPs. In contrast, stronger preprocessing or even feature engineering is required to successfully employ the latter. After all, however, the most important criterion to evaluate an approach is its predictive performance.
The heterogeneity of approaches with respect to the number of target categories and the variety of performance measures pose challenges for a direct comparison using the overall accuracies. Therefore, feature-wise evaluation appears most promising. As the distribution of some features (e.g. violet) has proven to be very unbalanced in our data set, even high accuracies might relate to poor predictions (e.g. when the feature is never detected). Hence, feature-wise accuracies are only a coarse indicator of the model's performance that may nonetheless give insights where difficulties lie and what features are more difficult to determine than others. However, for some promising approaches we computed the sensitivity and specificity per feature to reveal a more fine-grained picture of the predictive performance.

In the single-label CNN, very good results are achieved for features relying on the thickness and length of the asparagus (see subsection 4.1.2). All of these features

achieve a balanced accuracy above 90%, with best results for the feature very thick (98% sensitivity and 99% specificity). Of the solely hand-labeled features, feature hollow shows the best performance (77% sensitivity and 98% specificity). The feature rusty head has the least performance (52% sensitivity and 81% specificity).
The multi-label approach has an overall accuracy of 75% (see subsection 4.1.3). The performance of the CNN for the two head-related features is indicated by sensitivity and specificity values. Flower detection reaches 55% sensitivity and 95% specificity while rusty head detection attains only 19% sensitivity at 98% specificity. The overall accuracy of the multi-label CNN approach reaches up to 87%. For this model, accuracies are not calculated per feature.
In contrast, feature-wise accuracies for binary classification can be reported (see subsection 4.1.1). The same holds for feature-wise performance measures that were calculated for some of the other approaches. The feature engineering based approaches show good results on all of its three detected features, namely for bent (82% sensitivity, 67% specificity) and similarly for violet detection (62% sensitivity and 96% specificity) as well as for rusty body (71% sensitivity, 65% specificity).

The unsupervised learning approaches, namely PCA and the convolutional autoencoder, both deal with dimension reduction. Both were trained on the sample set for which labels are available. While the classification method based on PCA targets at binary feature prediction (absence or presence of a feature) (see subsection 4.2.1), the unsupervised autoencoder does not predict labels (see subsection 4.2.2). The accuracy of PCA is promising for length and hollow (100%) but extremely poor for bent detection (20%). It has to be mentioned that only very few samples were used for training and evaluation of the named approach. As such it is yet to be proven whether or not these results generalize.

A semi-supervised learning method was based on a partially labeled data set (see subsection 4.3.1). A semi-supervised autoencoder and a semi-supervised variational autoencoder perform multi-label classification. The more simple semi-supervised autoencoder performs better. Unfortunately, the predicted power is still rather poor, best for fractured (57% sensitivity, 100% specificity) and worst for violet as it does not detect any violet spears (0% sensitivity).

In the last approach, a random forest model that predicts the class labels based on the annotated features instead of features as the aforementioned approaches, delivers an average accuracy of about 75%. Our analysis shows that the model recalls some class labels like I A Anna or Hohle more reliably than class labels like II A or II B.

Chapter 6

Discussion

In our study project we pursued three main objectives. The main goal was to explore and implement different algorithms for asparagus classification. The second objective is closely linked to this and relates to best practices in relation to applied data science and big data. This included storage of data on remote servers and computationally expensive procedures that are required for training in the computational grid of Osnabrück University: The methodological aspect of our study project. As our work also served as a sample project to learn more about possibilities to effectively organize collaborative work, we also targeted a third objective, that is, the organizational aspect that is closely linked to project management. In the following, the core results with respect to these three objectives are shortly named and discussed.

6.1 Classification results

Asparagus spears have several features that we aimed to extract. Some features such as the length and width of asparagus spears were undoubtedly measurable using a pure computer vision approach that does not rely on machine learning. For others, direct filtering was not easily possible because no clear cut definition of features, such as bent or violet exists, that is precise enough to be implemented directly. Although relevant information can easily be extracted, the rules to infer the desired binary features are inaccessible: On one side, decision boundaries for binary classifiers have to be found. On the other side, the perception of the features color or bent has been shown to be very subjective. Moreover, filtering features such as flower has been proven difficult. Named attribute relates to details in a few pixels, comes in different forms, and highly depends on the perspective. These are the reasons machine learning must be employed to successfully classify asparagus. We designed several neural networks and applied them in different ways to analyze and classify a large-scale asparagus image data set. Some approaches worked better than others, as can be concluded from the Summary of results.

Training MLP classifiers on histograms of palette images has proven a promising approach to predict color features. Named histograms contain information about the fraction of foreground pixels that correspond to violet or rusty colors. As MLPs have few parameters, the design is rather trivial and the training process quick. The fact that predictions are far from perfect might be due to inconsistencies in the training data. One may assume, however, that the models generalize well and represent rules that relate to average opinions or definitions of highly subjective features such as color or bent.

The single-label CNN for the classification of 13 features promises a flexible and easy-to-use solution. Not much preprocessing is needed[1] and it is able to learn every feature. The network architecture is not aimed at a small, specific subset of only one or two features. Rather it is a basic network that could now be fine-tuned to the individual needs of each feature.
As already mentioned before (like for the feature engineering network from subsection 4.1.1), the lack of a clear threshold regarding certain features like rusty body or bent might be a factor that reduces the possible performance of the CNN. The smooth transition of the presence and absence of a feature makes it more difficult to categorize images close to named transition. In contrast, the performance of the model is well on features that were previously not labeled by humans but labeled automatically, like width and length. This difference between humanly labeled and automatically labeled features is striking in the network's results.
A drawback of the approach is the need of more labeled data with a feature being present to make the network more robust to outliers.

Feedforward CNNs were applied to predict individual features, for multilabel prediction, and predictions based on snippets that depict asparagus heads. In addition, effects of a custom loss function were tested. Promising in the multilabel prediction is that not only individual features, but also the relation between features can be considered in the learning process.
Our multilabel CNN reaches an accuracy up to 87%, which seems high. However, when looking at the accuracy and loss values over time, one can see that the model does not improve much. While sensitivity and specificity improve, and therefore indicate learning, the validation loss remains high, indicating overfitting. This model seems to be especially sensitive to the imbalance between 0 and 1 in the label vectors. Concerning this, there is still room to play around with our parameters to further improve the architecture.

Applying PCA on individual features and projecting the image information into a smaller dimensional subspace showed promising results. It revealed that the first principal components managed to capture most of the information. However, differences between most features seem to be too small to be adequately represented in the low dimensional space.
In this approach, the features width, length and hollow seem to be classifiable with high performance, and the features bent and rusty body seem to be most difficult. Width, length and hollow (as hollow asparagus is likely to be thick) are features that can be related to the shape and spatial appearance of the spear in the picture. This walks together with the findings that the first principle components (so the most important ones) only refer to the appearance of the asparagus in the picture and show the same picture for all asparagus. This leads to the assumption that the spatial appearance is counted as the most important feature, rather than taking the surface of the spears into account. This problem could be improved by generating pictures, where the possible asparagus positions are equally distributed over the pictures. Another reason for the similarities between the first principal component pictures is that one asparagus can have many features. Therefore the same pictures can be used for several feature PCAs. As asparagus with only one present feature is rather difficult to find, another solution to solve this problem needs to be found.

Similarly, Variational Autoencoders were used to derive a low dimensional representation using unsupervised learning. While some features such as the width

[1]The input images are only slightly reduced in pixel size to increase training speed, however, the background is not removed.

and length are mapped to clearly differentiable regions in latent asparagus space, this is not the case for many others. Only as a tendency, spears labeled as bent are for example mapped to regions in the lower periphery. Autoencoders are known for blurry reconstructions. This is a possible explanation for the lack of clusters in latent space for features that relate to details that are not sufficiently reconstructed.

Convolutional autoencoders were used for semi-supervised learning. However the results for this approach can be described as merely mediocre. One problem is arguably the mentioned insufficiency in reconstructing details. As details such as brown spots define target classes (e.g. rusty head) and they are not present in latent space. It is hard to establish a correlation of the respective latent layer activation and the target labels. Larger input image sizes or different network architectures that are suitable to reconstruct higher detail images could potentially help to improve performance of these semi-supervised learning algorithms.

Detecting the feature rusty head has proven rather difficult even though a dedicated network was trained on snippets that show asparagus heads in rather high resolution. This is potentially the case because details that are hardly visible even to the human eye have to be considered that occur in different locations. Although better results are achieved for the feature flower, the same most likely holds for this category as well. In contrast, better results are achieved for features that relate to the overall shape of asparagus spears instead of fine details. This holds for the category hollow as well as for bent. Color features are detected especially well based on histograms of palette images while CNNs have proven suitable to detect shape related features.
As previously noted (i.e. in subsection 4.1.1 and subsection 4.1.2), an obstructive factor for applications relying on labeled data proves to be the inconsistent labelling of certain data samples. Even for an expert in asparagus labeling like the owner of Gut Holsterfeld, setting a clear threshold for the absence or presence of specific features (and thus the attribution of a class label) becomes difficult in certain cases. This is partly evident in our classification approaches. Additionally, the agreement of manual annotators has to be better controlled during labeling. Thus, one suggestion for improvement would be to label the data a second time, with clear and consistent thresholds for feature presence and absence, then adapt and improve the supervised approaches.

Our work was specially directed at the Autoselect ATS II sorting machine. Whether we have succeeded in improving its currently running sorting algorithm could not yet be clarified systematically because of a lack of time and resources for a suitable comparison and evaluation. An idea for evaluating the current sorting method and our developed methods would be to run pre-sorted asparagus through the machine, test our approaches on the generated images, and then compare the performance of both.
During our meetings we discussed existing difficulties of evaluation. If our algorithm controls the sorting machine, new possibilities to create an evaluation are given. One method is to measure and compare the sorting of the harvesters who control the asparagus label after it was sorted by the machine. Further methods need to be considered. If necessary, the setup for automated procedures can be extended.
In cooperation with the local asparagus farm Gut Holsterfeld and the manufacturer of the sorting machine, a concrete realization of our approaches should now be developed and tested.

In summary, we successfully measured the width and height of asparagus spears and were able to develop detectors for the other features that performed surprisingly good, given the moderate inter-coder reliability that was partially due to unclear definitions of binary features such as bent or violet.

6.2 Methodology

Looking back, there are several methodological issues, which we would process differently now. We started our study project at the beginning of April. This is as well the beginning of the asparagus harvesting season. Therefore, we were able to start collecting data straight away. On the down-side, we had to start collecting data without a detailed plan in advance. Planning the data collection ahead could have made the data acquisition more efficient, and structured. Afterwards we could not change relevant parameters to answer a lot of organizational and methodological questions such as: How much data do we need? What format do we need it in? Is autonomous calibration possible? How exactly do we store the images effectively and efficiently? Is the hardware setup as we need it? And if not, how can we improve it? What kind of measurements or changes do we want to perform with the camera? Is the illumination as we want it? Could stereo cameras or other 3D viewing techniques such as depth cameras or laser grid be used? Would we integrate an additional camera taking pictures from the bottom of the spear or from the head region separately? What should our pipeline look like? How can we get labeled data right away?

As already mentioned in section 2.1, there was a misunderstanding between the group and the supporting asparagus farm about the type of data necessary. The already existing images were too few, and unlabeled. Therefore, we spent the entire asparagus season with data acquisition instead of starting with preprocessing as had been planned at project start. The number of labeled images that were collected by running pre-sorted classes though the machine is arguably insufficient to learn classes using the chosen deep learning approaches (Russakovsky et al., 2013; Russakovsky and Fei-Fei, 2010; Warden, Pete, 2017). Therefore, a lot of time was spent on preprocessing and labeling the data manually.
Another discussion point concerns our data. The image quality in terms of pixel size of our images is really high. Due to limited memory capacity and long runtimes of some of the tested networks, images needed to be down-sampled. Additionally, images of different file types were used (e.g. png, jpg), which also includes reduction in disc space. We should further investigate to what extent images can be down-sampled without losing critical information, in order to optimize the named difficulties.
Even though three images of every asparagus spear are given, they are all taken from the same angle – from above. In the ideal case, the asparagus spear rotates over the conveyor belt, such that each spear is depicted in the pictures from a different viewpoint. The better the asparagus rotates, the more reliable is a later judgement of the spear in terms of class labels or features. Since the rotation is often missing when the spear is too bent, an additional angle could improve the rating. Concrete ideas on how to improve the setup are given in the last chapter, 7 Conclusion.
As previously mentioned, our labels of asparagus features are partly achieved by computer vision algorithms, partly based on human perception. As previously

outlined, human performance is commonly acknowledged as the baseline performance in classification tasks. While the performance of our automatic feature extraction for length and width is really high, we decided that for the features violet, rusty head, rusty body, bent, hollow, and flower a human perception would be more accurate. Even though this is commonly used as the "gold standard", it can of course also bring more variation, and maybe even inconsistency between raters, than an algorithm.
As explained in the section Preprocessing, during preprocessing for the labeling procedure, the three images of one asparagus spear are joined together and then labeled. Therefore, labeling was faster because three perspectives are labeled at once. However, it could also be tried to use the single perspectives and conclude the classes from a combination of the recognized features.
We kept the features binary, as this is easier to label, and easier to use for our supervised classification approaches. The down side of a binary label is, however, that a clear boundary is set, where in real life there is a smooth transition. Even for our supervising farmer, it is sometimes difficult to decide on a boundary. This is arguably due to the small differences between class labels, and vague borders between positive and negative examples. While the binary representation makes certain analyses and classification much easier, it also brings restrictions.
Moreover, we observed difficulties concerning the labels in the communication between the group and the farmer. The communicated need is that the sorting algorithm works "better". But what does that technically mean? And what is technically possible? For the farmer, the sorting would already be "better", if the sorting mistakes would be more systematic. This would not necessarily mean that the overall accuracy of correctly sorted asparagus into one out of 13 class labels needs to improve, but that the overall impression of all spears sorted into one tray is more homogeneous.

6.3 Organization

This section summarizes and briefly discusses what each member has learned on an organizational level during the year.
The team did not only achieve new scientific skills and techniques of data acquisition, preparation, and analysis but also gained valuable new insights into the organization of a large project. It was learned how to structure the team more successfully and purposefully.

First and foremost, a successful project needs excellent communication. Not everyone has to discuss or listen to each specific detail in every working area. Instead, communication should be balanced. Often, it suffices when all team members have a broad overview.
Secondly, it turns out to be helpful when one or two members exchange some task-related work in favor of more management-related work. Democratic decision making does not necessarily exclude the role of a team leader or of a manager who has an overview of the tasks or of a certain task area. The whole team agrees to structure the next project in the same way as done during the second term, including manager roles and a stricter working plan. By this, better team dynamics are gained and time management can be improved.
Further, the strengths of the single members have to be evaluated before the project starts, so they can be used efficiently. Although, not everyone has to do the task he or she is best at. One should also have the opportunity to work on tasks that are

new, challenging and interesting. It allows each member to broaden their skills and it avoids discouragement.
Finally, the team agrees to have more focus on the overall goal than only think of what directly lies ahead. For this, concrete goals have to be formulated well. Milestones or intermediate goals should be defined and evaluated more rigorously. More time has to be taken into consideration when planning ahead as well as for including adjustments.

In conclusion, the experience of having two different working structures gave us the ability to compare and judge what is essential to successful teamwork. It also helped to understand how each member can contribute to the team regarding personal skills and interests, and what each member wants to improve for future teamwork.
As the main intention of the study project was to enhance our knowledge, we sought out a task that we were highly motivated to do. By that, we did not only practice theoretical knowledge but gained new experiences and sustainably improved our team skills for future work.

Chapter 7

Conclusion

In the scope of our project, we sucessfully adopted various classical as well as deep learning based computer vision approaches to classify asparagus spears according to their descriptive features or class labels. After collecting images, labeling, and processing the data, different trainable models were implemented from the fields of supervised learning, unsupervised learning and semi-supervised learning.

We could prove that computer vision and machine learning based techniques are practicable for the asparagus classification problem.
Our explorative study gave the possibility of a better overview on which techniques seem promising for asparagus classification and which might not need to be pursued in the future. As a next step, we would concentrate on using binary feed-forward CNNs because they offer a flexible and simple solution. Additional fine-tuning of the architecture to fit the individual needs of the to-be-predicted features (or class labels) and further preprocessing of the data will improve the network (as demonstrated in subsection 4.1.1). Considering the amount of labeled data that is available, unsupervised and semi-supervised approaches gave some insight into the data but, in the end, they were more effortful to implement while not promising much better results. If there had been less labeled data, these approaches might have been more useful than approaches relying on labeled data.
Whether we have succeeded in improving the currently running sorting algorithm can not be said, yet. In cooperation with the local asparagus farm Gut Holsterfeld and the manufacturer of the asparagus sorting machine Autoselect ATS II, a method for evaluation can now be developed.

Due to the direct start of the harvesting season at the beginning of the project, the sorting machine and the hardware setup had to be used as available (see also section 6.2). However, improvements were discussed with the manufacturer of the machine.
A second camera to capture the head of the asparagus is needed in particular[1] which is reflected in our results. Especially for features like flower or rusty head, an additional head camera helps greatly with the classification (as shown in A dedicated network for head-related features). Another camera taking an image from the bottom of the spear could improve the detection of the feature hollow.[2] Additional perspectives of the hole asparagus spears give more usable information to determine certain features more accurately. For example differences in lighting and reflection can carry useful information about the shape.
To further improve the setup, it is also conceivable that other sensor systems, such as laser technology, could help to find relevant properties for asparagus classification (Bhargava and Bansal, 2018). As mentioned in the Discussion, the most crucial

[1] This is already the case for newer versions of the Autoselect ATS II like the one at Querdel's Hof.
[2] These cameras are already part of new asparagus sorting hardware like the one mentioned here: https://www.neubauer-automation.de/uk/asparagus-sorting-machine-espaso-technicaldata.php

component for improving future asparagus sorting with the Autoselect ATS II is a further development of the setup. This requires hardware changes to the sorting machine and is therefore beyond the scope of our project.

Another claim that we started to address in the Methodology is that the asparagus classification varies between different farms. Further it also varies within one farm throughout the harvesting season. For example, one request of the farmer of Gut Holsterfeld is to have the possibility to sort asparagus in a higher category if weather conditions reduce the usual amount of high quality asparagus. In order to meet this requirement, manual adjustment of parameters and a smooth transition for several features is needed. At the moment this is impossible with most contemporary neural networks. Since there is the temporal, sequential component of the harvesting season, it may be worthwhile to consider LSTMs in combination with CNNs. According to the literature there are still many wide possibilities, for example applying bayesian networks learning.

In conclusion, we can confirm that modern approaches from computer vision and machine learning bear huge potential for the improvement of asparagus classification. Effective means of agile development allow for efficient collaboration in the production of the respective implementations. We demonstrated that the algorithms we selected could be used not only for scientific purposes but also in industrial applications. We strongly believe, machine learning approaches can help to improve the classification of asparagus into commercial quality classes.

Appendix A

Work Overview

A.1 Task list

Below the name of each project member, the tasks that were contributed by the member are listed.

Maren Born

- Organizational
  - File manager:
    - create manual on how to label
    - assign label tasks among team members
    - control and clean label files
- Theoretical background and research
  - Presentation on neural network architectures
  - Paper research
- Technical setup and problem solving
  - Organize unlabeled image folders on the university folder
  - Clean folders of pre-labeled images
  - Collect example images for label training with Silvan
- Programming
  - Curvature detection
  - Principal components analysis
- Report
  - Write and proofreading
  - Latex formatting
  - Figures plotting
- Other
  - Systematic testing of hand-label app
  - Collect and label images

Michael Gerstenberger

- Organizational

- Preprocessing coordinator
- Grid manager

◦ Theoretical background and research

- Background on autoencoders and feature engineering
- Presentation on neural network architectures

◦ Technical setup and problem solving

- Support team with Grid related questions

◦ Programming

- Preprocessing (background removal, rotation, centering, cropping)
- Curvature detection, including reimplementation to improve performance
- Violet detection, including reimplementation to improve performance
- Hand-label app using PyQt
- Create palette colors and heads-only data sets
- Feature engineering and MLP
- Python API to submit scripts
- Variational autoencoder
- Semi-supervised autoencoder
- Dedicated head network

◦ Report

- Write and proofreading

◦ Other

- Collect and label images

Katharina Groß

◦ Organizational

- Setup sphinx documentation build process
- Readthedocs setup and maintenance
- Converting protocols, handling rst files
- Virtual environment setup for rtd
- Manual on `ssh -Y` usage
- Proposed structural paper reading process
- GitHub project management:
  · Pull request reviews
  · Pull request merges
  · Issue management
  · Code quality assurance
  · Continuous integration setup and maintenance
  · Remove stale branches

        - Keep README up to date
    - Theoretical background and research
        - Research agreement measures
            - Kappa, f1, etc.
        - Presentation on:
            - Neural network architectures / multi-class SVMs
            - Git and GitHub
            - Virtual environments
            - Continuous integration
            - Structure of literature catalog
            - Background literature
    - Technical setup and problem solving
        - Setup virtual environments
        - Move files to new scratch folder and clean folders
        - Support on problems with
            - Virtual environments
            - Quota
            - X forwarding
            - Help others debugging their code
    - Programming
        - Pre-processing
            - Multiple perspectives as one image
            - Automatic merging of label files
            - Keras data set for efficient iteration
        - Tried to do Flower detection
        - Kappa agreement(Box plots for inter-rater agreement)
        - Pipeline for multiple models
        - From feature to class label with random forest and MLP
        - Confusion matrix visualization for class labels
        - Streamlit app from asparagus image to class label
        - Continuous integration setup with Travis
        - Grid training of TensorFlow/keras models
            - Write *.sge scripts to run training
            - Include continuation after wall time
            - Train models and store them
            - Use tensorboard for inspection
    - Report
        - Create LaTeX project
        - Write and proofreading
        - LaTeX formatting

        - plot figure of sorting machine
    - Other
        - Collect and label images

Richard Ruppel

- Organizational
    - Manual on hand-label app for remote usage
    - Manual on shared virtual environments
    - Distribute tasks among team members
    - Create work schedules and road maps
- Theoretical background and research
    - Presentation on neural network architectures
    - Research on services, APIs and frameworks
- Technical setup and problem solving
    - X11 forwarding for windows
    - SSL remote debugging
    - Create shared virtual environments and help with problems
    - Help with conda installation
    - Create input pipeline
- Programming
    - Service for image transfer on the sorting machine
    - Flower detection
- Report
    - Write and proofreading
- Other
    - Try to compile and reverse compile software of sorting machine
    - Collect and label images

Sophia Schulze-Weddige

- Organizational
    - Develop idea, prepare study project, find supervisors and group members
    - Distribute tasks among team members
    - Create work schedules and road maps
- Theoretical background and research
    - Presentation on neural network architectures

- Technical setup and problem solving
  - Familiarize with sorting machine
  - Problem solving on machine remotely and on site
- Programming
  - Preprocessing (background removal and combining perspectives)
  - Length detection
  - Width detection
  - Rust detection
  - Flower detection
  - Create data set of labeled images
  - Decision rules to get from features to class labels
  - Multi-label classification model
- Report
  - Write and proofreading
- Other
  - Collect and label images

Malin Spaniol

- Organizational
  - Manual on ssh-Y usage
  - Manual on hand-label app
  - Manual on running the hand-label app on the university server
  - ANN manager: Keep track on classification approaches of all members
- Theoretical background and research
  - Presentation on neural network architectures
  - Research agreement measures
- Technical setup and problem solving
- Programming
  - Length detection
  - Preprocessing of images for kappa agreement
  - Agreement measures (kappa, F1, accuracy)
  - Principal component analysis
- Report
  - Write and proofreading
- Other
  - Systematic testing of hand-label app

- Collect and label images

Josefine Zerbe

- Organizational
  - Manual on hand-label app for remote usage
  - Distribute tasks among team members
  - Create work schedules and road maps
- Theoretical background and research
  - Presentation on
    - neural network architectures
    - background literature
    - structure for literature catalog
    - semi-supervised learning
- Technical setup and problem solving
- Programming
  - Curvature detection
  - Single-label classification model
- Report
  - Main editor and supervision
  - Create structure
  - Write and proofreading
  - Main latex formatting
    - Continuously transfer Google Docs file to LaTex Project
    - Write LaTex formatting instructions
  - Create and update time table, update roadmap
- Other
  - Count all image data
  - Collect and label images

A.2 Report list

Here, an overview of the report is given in chapters and subchapters. The name behind each chapter indicates who wrote the respective section. The name in brackets indicates the first reviewer of the chapter. All chapters were reviewed by all team members.

0. Abstract – *Josefine* (*Katharina*)
1. Introduction – *Josefine* (*Maren*)

1.1. The project – *Josefine* (*Maren*)
1.2. Classification based on computer vision – *Sophia* (*Josefine*)
1.3. Background on sorting asparagus – *Josefine* (*Sophia*)

2. Organization and data acquisition – *Josefine* (*Richard*)

2.1. Roadmap of the project – *Josefine* (*Richard*)
2.2. Organisation of the study group – *Richard* (*Sophia*)
2.2.1. Communication – *Richard* (*Sophia, Josefine*)
2.2.2. Teamwork – *Richard* (*Sophia*)
2.3. Data collection – *Josefine, Richard* (*Malin*)
2.4. Literature on food classification using computer vision – *Josefine, Malin* (*Katharina*)

3. Preprocessing and data set creation – *Josefine* (*Maren*)

3.1. Preprocessing – *Sophia, Michael* (*Katharina*)
3.2. Feature extraction – *Sophia, Josefine* (*Katharina*)
3.2.1. Length – *Sophia* (*Katharina*)
3.2.2. Width – *Sophia* (*Katharina*)
3.2.3. Rust – *Sophia* (*Katharina*)
3.2.4. Violet – *Michael* (*Sophia*)
3.2.5. Curvature – *Michael* (*Sophia*)
3.2.6. Flower – *Sophia* (*Katharina*)
3.2.7. Hollow – *Josefine* (*Malin*)
3.2.8. Not classifiable – *Josefine* (*Malin*)
3.3. The hand-label app: A GUI for labeling asparagus – *Michael* (*Maren*)
3.3.1. Motivation *Michael* (*Maren*)
3.3.2. The Labeling Application *Michael* (*Maren*)
3.4. Manual labeling – *Josefine* (*Richard*)
3.4.1. Labeling outcome – *Josefine* (*Malin*)
3.4.2. Agreement Measures – *Malin* (*Katharina*)
3.4.3. Reliability – *Malin* (*Katharina*)
3.5. The asparagus data set – *Sophia* (*Richard*)
3.5.1. Description of our data set(s) – *Sophia* (*Richard*)
3.5.2. Data set creation with Tensorflow – *Richard* (*Josefine, Katharina*)

4. Classification – *Malin* (*Maren*)

4.1. Supervised learning – *Josefine* (*Maren*)
4.1.1. Prediction based on feature engineering – *Michael* (*Maren*)
4.1.2. Single-label classification – *Josefine* (*Katharina*)
4.1.3. Multi-label classification – *Sophia* (*Katharina*)
4.1.4. A dedicated network for head-related features – *Michael* (*Josefine*)
4.1.5. From hand-labeled features to class labels – *Katharina* (*Sophia*)
4.2. Unsupervised learning – *Malin, Maren* (*Michael*)
4.2.1. Principal Component Analysis – *Malin, Maren* (*Michael*)
4.2.2. Autoencoder – *Michael* (*Sophia, Josefine*)

4.3. Semi-supervised learning – *Michael* (*Maren, Sophia*)

4.3.1. Semi-supervised autoencoder – *Michael* (*Maren, Josefine*)

5. Summary of results – *Maren* (*Malin*)

6. Discussion – *Malin, Richard* (*Michael*)

6.1. Classification results – *Malin, Maren* (*Michael*)

6.2. Methodology – *Malin* (*Richard*)

6.3. Organization – *Josefine, Richard* (*Malin*)

7. Conclusion – *Richard* (*Josefine, Malin*)

Appendix B

Additional Material

Here, additional material can be found that was not included in the chapters of the report.

B.1 Supplementary Information

B.1.1 File moving service

In this section it is in detail described how the service of filemoving is constructed. As Windows is used as the operating system of the sorting machine, the development was done with the .NET framework[1] in the programming language C#. The package provided is called Topshelf.[2] Topshelf is a service hosting framework for building Windows services using .NET. With the package, it is possible to develop a console application in the development phase, compile it as a service, and install it later via the console. Previously, it was not possible to debug services during the development phase. The function of the service is based on the FileSystemWatcher object from the System.IO namespace.[3] In the main program, a list of files in the source folder is kept. Files that are older than one hour are moved to the target folder on the external drive. The selected files are moved by a function that is called, when an event is triggered. The event is triggered by the FileSystemWatcher after subscribing to different flags. Shortly after initialization, the service was adjusted because removing the images from the C disk straight away caused the sorting program to stop. The problem is solved by keeping the most recent 1000 images and moving older images to the external disk.

[1]For further information on the .NET framework, see https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview

[2]For further information on Topshelf, see https://github.com/Topshelf/Topshelf

[3]For further information on the SystemFileWatcher, see https://docs.microsoft.com/en-us/dotnet/api/system.io.filesystemwatcher?view=netframework-4.8

B.2 Additional figures

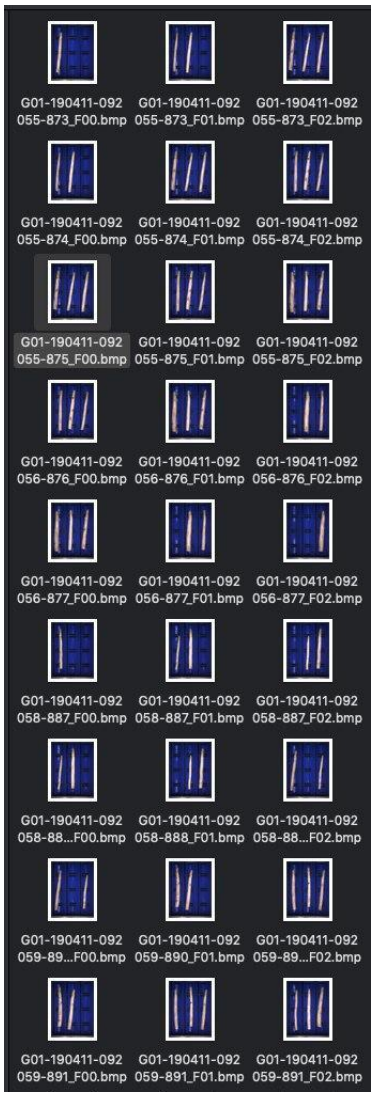
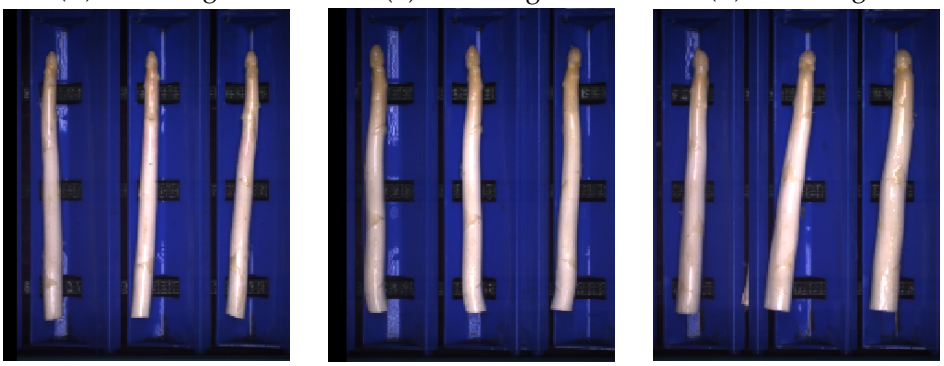


FIGURE B.1: **File Overview Example of Original Image Data** A view of the data files after data collection. The original file names can be seen. The images belonging to one asparagus are marked with F00, F01, and F02 respectively.

B.2.1 Single-label CNN figures

(A) False Negative (B) False Negative (C) False Negative



(D) False Positive (E) False Positive (F) False Positive

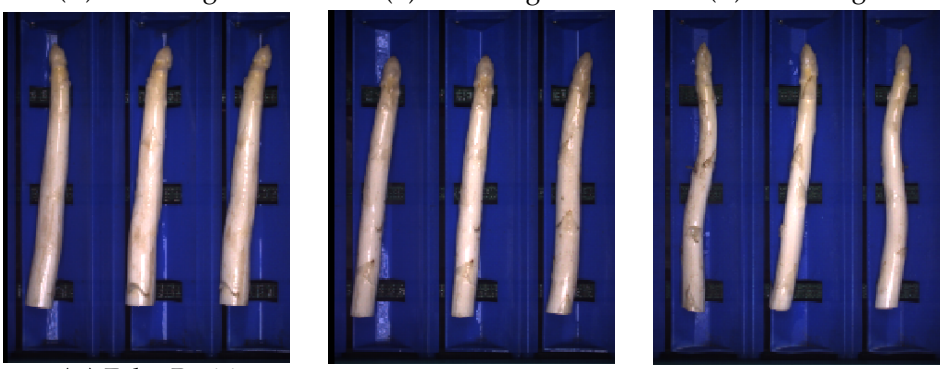
FIGURE B.2: **Single-Label CNN Example Images Feature Violet** Example images of false negatives and false positives of the feature violet after 5160 training steps.

(A) False Negative (B) False Negative (C) False Negative

(D) False Positive (E) False Positive (F) False Positive

FIGURE B.3: **Single-Label CNN Example Images Feature Fractured** Example images of false negatives and false positives of the feature fractured after 2700 training steps.

(A) False Negative (B) False Negative (C) False Negative



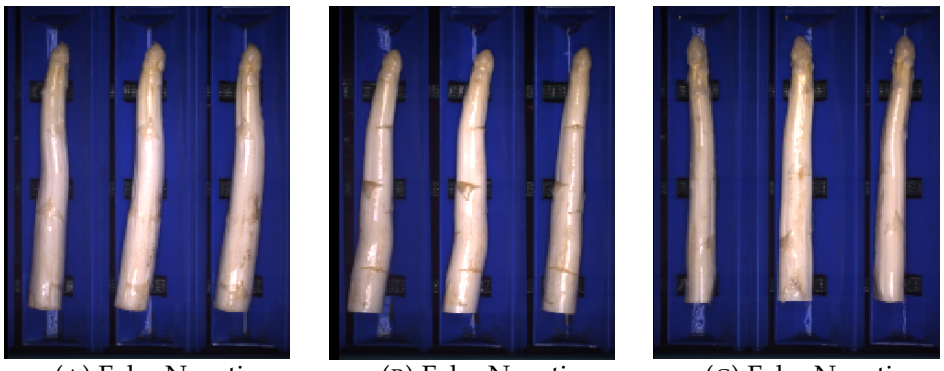
(D) False Positive (E) False Positive (F) False Positive

FIGURE B.4: **Single-Label CNN Example Images Feature Flower** Example images of false negatives and false positives of the feature flower after 4800 training steps.

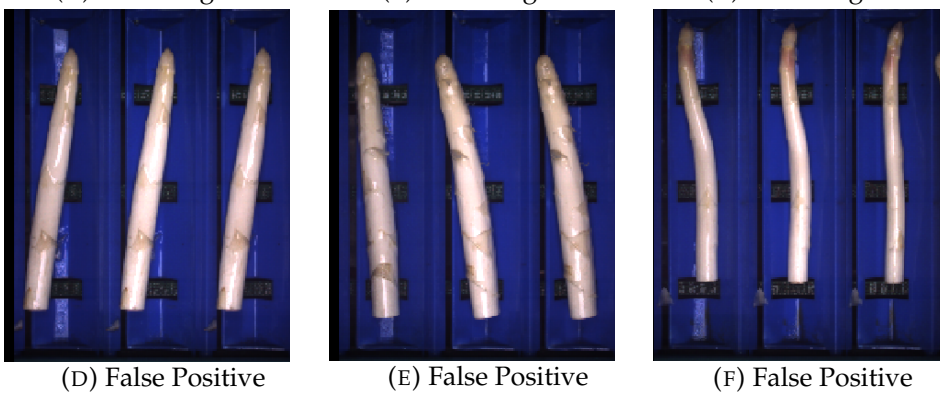
(A) False Negative (B) False Negative (C) False Negative

(D) False Positive (E) False Positive (F) False Positive

FIGURE B.5: **Single-Label CNN Example Images Feature Rusty Head** Example images of false negatives and false positives of the feature rusty head after 4680 training steps.



(A) False Negative (B) False Negative (C) False Negative



(D) False Positive (E) False Positive (F) False Positive

FIGURE B.6: **Single-Label CNN Example Images Feature Rusty Body** Example images of false negatives and false positives of the feature rusty body after 3000 training steps.

(A) False Negative

(B) False Positive

(C) False Positive

(D) False Positive

FIGURE B.7: **Single-Label CNN Example Images Feature Very Thick** Example images of false negatives and false positives of the feature very thick after 5400 training steps.

(A) False Negative

(B) False Negative

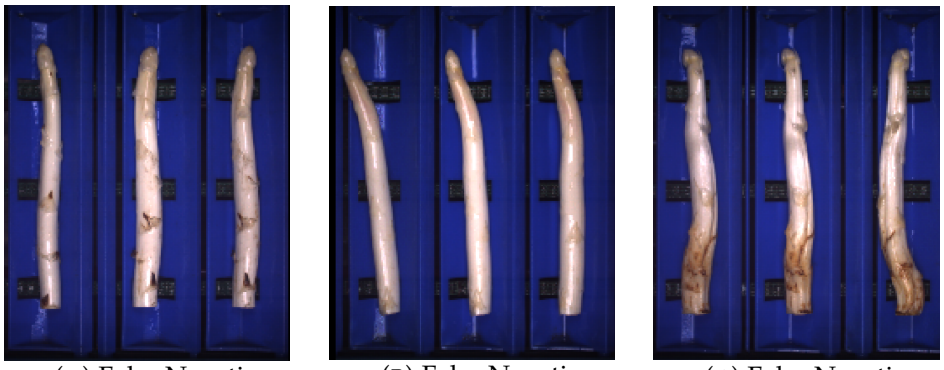
(C) False Negative

(D) False Positive

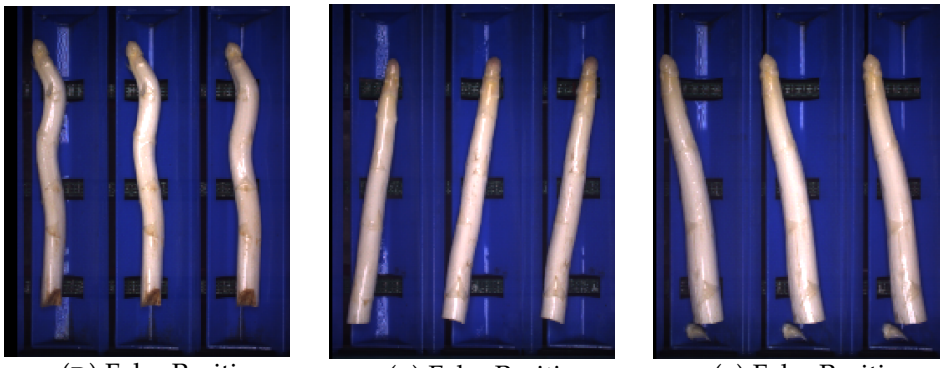
(E) False Positive

(F) False Positive

FIGURE B.8: **Single-Label CNN Example Images Feature Thick** Example images of false negatives and false positives of the feature thick after 3960 training steps.

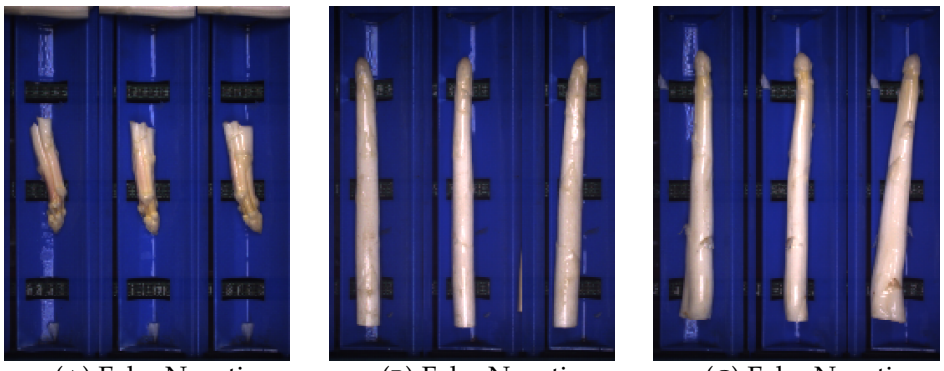


(A) False Negative (B) False Negative (C) False Negative

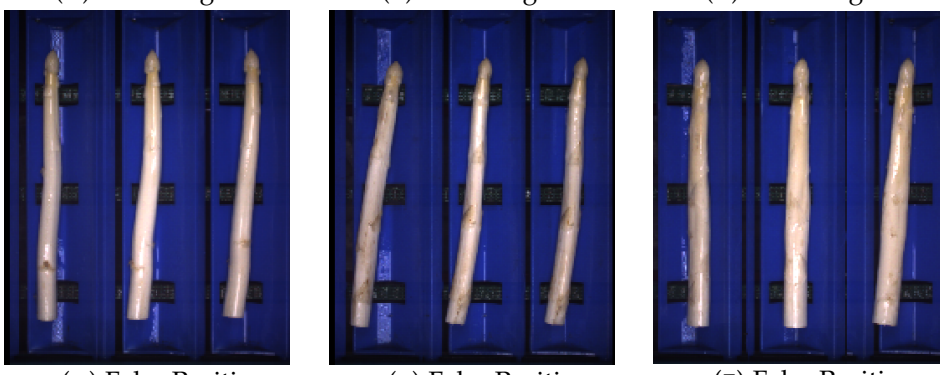


(D) False Positive (E) False Positive (F) False Positive

FIGURE B.9: **Single-Label CNN Example Images Feature Medium Thick** Example images of false negatives and false positives of the feature medium thick after 4560 training steps.



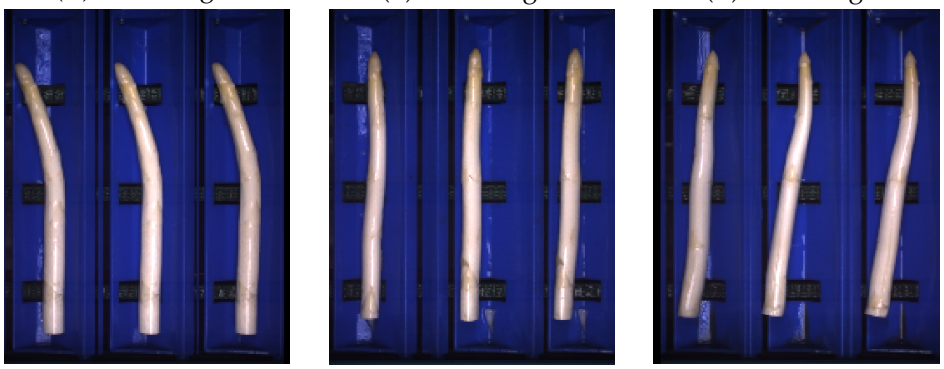
(A) False Negative (B) False Negative (C) False Negative



(D) False Positive (E) False Positive (F) False Positive

FIGURE B.10: **Single-Label CNN Example Images Feature Thin** Example images of false negatives and false positives of the feature thin after 4200 training steps.

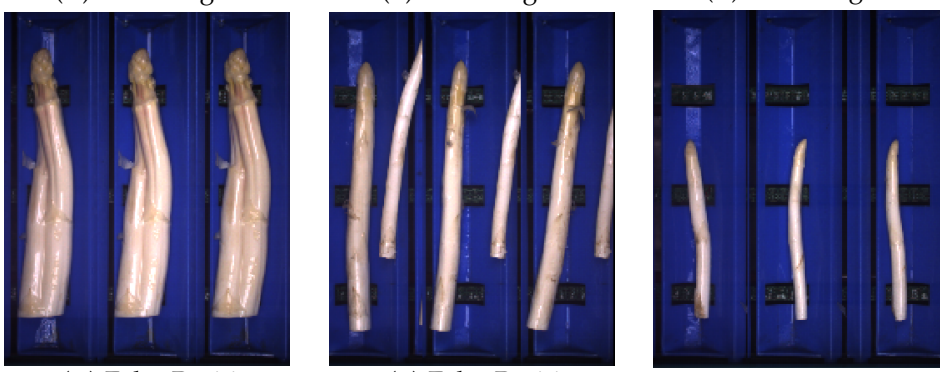
(A) False Negative (B) False Negative (C) False Negative



(D) False Positive (E) False Positive (F) False Positive

FIGURE B.11: **Single-Label CNN Example Images Feature Very Thin** Example images of false negatives and false positives of the feature very thin after 4320 training steps.

(A) False Negative (B) False Negative (C) False Negative



(D) False Positive (E) False Positive (F) False Positive

FIGURE B.12: **Single-Label CNN Example Images Feature Not Classifiable** Example images of false negatives and false positives of the feature not classifiable after 5400 training steps.

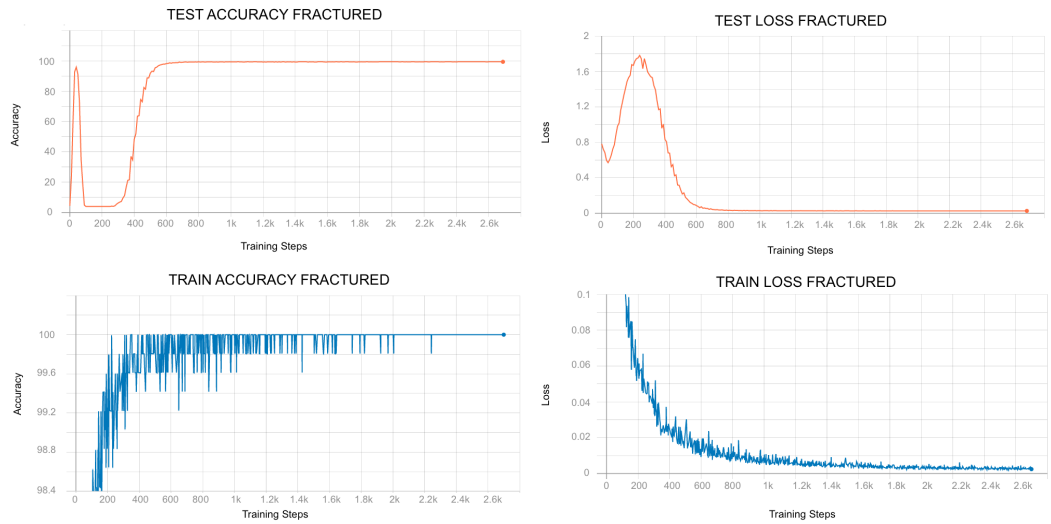


FIGURE B.13: **Accuracy and Loss of Feature Fractured** Test accuracy and test loss as well as training accuracy and training loss for the feature fractured are depicted above. It was randomly chosen to show an exemplary course of accuracy and loss during training.

Acronyms

ANN Artificial Neural Network.

CNN Convolutional Neural Network.

GAN Generative Adversarial Network.

IKW Institute of Cognitive Science.

LSTM Long Short-Term Memory.

MLP Multilayer Perceptron.

MSE Mean Squared Error.

NIN Network in Network.

PCA Principal Component Analysis.

ROC Receiver Operating Characteristic.

SSH Secure Shell.

SVM Support Vector Machine.

TFDS TensorFlow Datasets.

UML Unified Modeling Language.

VAE Variational Autoencoder.

VGG Visual Geometry Group.

Bibliography

Al Ohali, Yousef (2011). "Computer vision based date fruit grading system: Design and implementation". In: *Journal of King Saud University-Computer and Information Sciences* 23.1, pp. 29–36.

Al-Rawi, Mohammed and Dimosthenis Karatzas (2018). "On the Labeling Correctness in Computer Vision Datasets." In: *IAL@ PKDD/ECML*, pp. 1–23.

Balaban, Stephen (2015). "Deep learning and face recognition: The state of the art". In: *Biometric and Surveillance Technology for Human and Activity Identification XII*. Vol. 9457. International Society for Optics and Photonics, 94570B.

Bao, Jianmin et al. (2017). "CVAE-GAN: fine-grained image generation through asymmetric training". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2745–2754.

Batista, Gustavo EAPA, Ronaldo C Prati, and Maria Carolina Monard (2004). "A study of the behavior of several methods for balancing machine learning training data". In: *ACM SIGKDD explorations newsletter* 6.1, pp. 20–29.

Bengio, Yoshua (2012). "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, pp. 437–478.

Bettilyon, Tyler Elliot (2018). *How to classify MNIST digits with different neural network architectures*. URL: https://medium.com/tebs-lab/how-to-classify-mnist-digits-with-different-neural-network-architectures-39c75a0f03e3 (visited on 04/24/2020).

Bhargava, Anuja and Atul Bansal (2018). "Fruits and vegetables quality evaluation using computer vision: A review". In: *Journal of King Saud University-Computer and Information Sciences*.

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. Springer.

Bohling, Geoff (2006). "Dimension reduction and cluster analysis". In: *Scientist, no. March*, pp. 1–22.

Breiman, Leo (2001). "Random forests". In: *Machine learning* 45.1, pp. 5–32.

Brosnan, Tadhg and Da-Wen Sun (2002). "Inspection and grading of agricultural and food products by computer vision systems—a review". In: *Computers and electronics in agriculture* 36.2-3, pp. 193–213.

Brownlee, Jason (2019). *How to Configure Image Data Augmentation in Keras*. URL: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/ (visited on 04/24/2020).

Bushaev, Vitaly (2018). *Adam — latest trends in deep learning optimization.* URL: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c (visited on 04/24/2020).

Caruana, Rich and Alexandru Niculescu-Mizil (2006). "An Empirical Comparison of Supervised Learning Algorithms". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 161–168. ISBN: 1595933832. DOI: 10.1145/1143844.1143865. URL: https://doi.org/10.1145/1143844.1143865.

Chen, Min et al. (2017). "Deep features learning for medical image analysis with convolutional autoencoder neural network". In: *IEEE Transactions on Big Data*.

Cohen, Jacob (1960). "A coefficient of agreement for nominal scales". In: *Educational and psychological measurement* 20.1, pp. 37–46.

Daumé III, Hal (2012). "A course in machine learning". In: *Publisher, ciml. info* 5, p. 69.

Dertat, Arden (2017). *Applied Deep Learning - Part 1: Artificial Neural Networks*. URL: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6 (visited on 04/24/2020).

Diaz, R et al. (2004). "Comparison of three algorithms in the classification of table olives by means of computer vision". In: *Journal of Food Engineering* 61.1, pp. 101–107.

Donis-González, Irwin R and Daniel E Guyer (2016). "Classification of processing asparagus sections using color images". In: *Computers and Electronics in Agriculture* 127, pp. 236–241.

Europäische Komission (1999). "Verordnung (EG) Nr. 2377/1999 der Kommission vom 9. November 1999 zur Festsetzung der Vermarktungsnorm für Spargel". In: *Amtsblatt der Europäischen Gemeinschaften, OJ L 287, 10.11.1999, p. 6–11*. URL: https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:31999R2377&from=DE (visited on 03/25/2020).

Feinstein, Alvan R and Domenic V Cicchetti (1990). "High agreement but low kappa: I. The problems of two paradoxes". In: *Journal of clinical epidemiology* 43.6, pp. 543–549.

Figueroa, Rosa L et al. (2012). "Predicting sample size required for classification performance". In: *BMC medical informatics and decision making* 12.1, p. 8.

Flight, Laura (2018). *The Disagreeable Behaviour of the Kappa Statistic*. URL: https://www.sheffield.ac.uk/polopoly_fs/1.404095!/file/RSS_Poster_Laura_Flight_Final.pdf (visited on 04/21/2020).

Franky, Frank (2018). *Multi-Label Classification and Class Activation Map on Fashion-MNIST*. URL: https://towardsdatascience.com/multi-label-classification-and-class-activation-map-on-fashion-mnist-1454f09f5925 (visited on 04/21/2020).

Garrido-Novell, Cristóbal et al. (2012). "Grading and color evolution of apples using RGB and hyperspectral imaging vision cameras". In: *Journal of Food Engineering* 113.2, pp. 281 –288. ISSN: 0260-8774. DOI: https://doi.org/10.1016/j.jfoodeng.2012.05.038. URL: http://www.sciencedirect.com/science/article/pii/S0260877412002701.

Geng, Jie et al. (2015). "High-resolution SAR image classification via deep convolutional autoencoders". In: *IEEE Geoscience and Remote Sensing Letters* 12.11, pp. 2351–2355.

Géron, Aurélien (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.

Gibaja, Eva and Sebastian Ventura (Apr. 2015). "A Tutorial on Multi-Label Learning". In: *ACM Computing Surveys* 47. DOI: 10.1145/2716262.

Gilpin, Leilani H et al. (2018). "Explaining explanations: An overview of interpretability of machine learning". In: *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE, pp. 80–89.

Godoy, Daniel (2018). *Understanding binary cross-entropy / log loss: a visual explanation*. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a (visited on 04/24/2020).

Har-Peled, Sariel, Dan Roth, and Dav Zimak (2003). "Constraint classification for multiclass classification and ranking". In: *Advances in neural information processing systems*, pp. 809–816.

He, Haibo and Edwardo A Garcia (2009). "Learning from imbalanced data". In: *IEEE Transactions on knowledge and data engineering* 21.9, pp. 1263–1284.

He, Kaiming et al. (2016a). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

— (2016b). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Heaton, Jeff (2015). *AIFH, Volume 3: Deep Learning and Neural Networks*.

HMF Hermeler Maschinenbau GmbH (2015). "Bedienungsanleitung Automatische Spargelsortiermaschine Autoselect". In: HMF Hermeler Maschinenbau GmbH.

Huang, Gao et al. (2017). "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708.

Hubel, David H and Torsten N Wiesel (1962). "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160.1, pp. 106–154.

Iglovikov, Vladimir and Alexey Shvets (2018). "Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation". In: *arXiv preprint arXiv:1801.05746*.

Jhuria, Monika, Ashwani Kumar, and Rushikesh Borse (Dec. 2013). "Image processing for smart farming: Detection of disease and fruit grading". In: pp. 521–526. ISBN: 978-1-4673-6099-9. DOI: 10.1109/ICIIP.2013.6707647. URL: https://doi.org/10.1109/ICIIP.2013.6707647.

Karpathy, Andrej (2014). *What I learned from competing against a ConvNet on ImageNet*. URL: http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/ (visited on 04/21/2020).

Keng, Brian (2020). *Semi-supervised Learning with Variational Autoencoders*. URL: http://bjlkeng.github.io/posts/semi-supervised-learning-with-variational-autoencoders/ (visited on 03/01/2020).

Kılıç, Kıvanç et al. (2007). "A classification system for beans using computer vision system and artificial neural networks". In: *Journal of Food Engineering* 78.3, pp. 897–904.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kondo, Naoshi (2010). "Automation on fruit and vegetable grading system and food traceability". In: *Trends in Food Science & Technology* 21.3. Advances in Food Processing and Packaging Automation, pp. 145 –152. ISSN: 0924-2244. DOI: https://doi.org/10.1016/j.tifs.2009.09.002. URL: http://www.sciencedirect.com/science/article/pii/S0924224409002611.

Kramer, Mark A (1991). "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE journal* 37.2, pp. 233–243.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Lata, Prof et al. (Apr. 2009). "Facial recognition using eigenfaces by PCA". In: *SHORT PAPER International Journal of Recent Trends in Engineering* 1.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

LeCun, Yann, Yoshua Bengio, et al. (1995). "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10, p. 1995.

LeCun, Yann A et al. (2012). "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48.

Liming, Xu and Zhao Yanchao (2010). "Automated strawberry grading system based on image processing". In: *Computers and Electronics in Agriculture* 71. Special issue on computer and computing technologies in agriculture, S32 –S39. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2009.09.013. URL: http://www.sciencedirect.com/science/article/pii/S016816990900204X.

Lin, Min, Qiang Chen, and Shuicheng Yan (2013). "Network in network". In: *arXiv preprint arXiv:1312.4400*.

Luo, M Ronnier et al. (2000). "A review of chromatic adaptation transforms". In: *Review of Progress in Coloration and Related Topics* 30, pp. 77–92.

McHugh, Mary (2012). "Interrater reliability: The kappa statistic". In: *Biochemia medica : časopis Hrvatskoga društva medicinskih biokemičara / HDMB* 22, pp. 276–82. DOI: 10.11613/BM.2012.031.

Mery, Domingo, Franco Pedreschi, and Alvaro Soto (2013). "Automated design of a computer vision system for visual food quality evaluation". In: *Food and Bioprocess Technology* 6.8, pp. 2093–2108.

Mirończuk, Marcin Michał and Jarosław Protasiewicz (2018). "A recent overview of the state-of-the-art elements of text classification". In: *Expert Systems with Applications* 106, pp. 36–54.

Olaode, Abass, Golshah Naghdy, and Catherine Todd (Sept. 2014). "Unsupervised Classification of Images: A Review". In: *International Journal of Image Processing* 8, pp. 2014–325.

Olivier, Chapelle, Scholkopf Bernhard, and Zien Alexander (2006). "Semi-supervised learning". In: *IEEE Transactions on Neural Networks*. Vol. 20. 3, pp. 542–542.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2012). "Understanding the exploding gradient problem". In: *CoRR, abs/1211.5063*, 2, p. 417.

Pedreschi, Franco, Domingo Mery, and Thierry Marique (2016). "Grading of potatoes". In: *Computer vision technology for food quality evaluation*. Elsevier, pp. 369–382.

Powers, David MW (2012). "The problem with kappa". In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 345–355.

Prakash, J Suriya et al. (2012). "Multi class Support Vector Machines classifier for machine vision application". In: *2012 International Conference on Machine Vision and Image Processing (MVIP)*. IEEE, pp. 197–199.

Pujari, Jagadeesh D., Rajesh Yakkundimath, and Abdulmunaf S. Byadgi (2014). "Recognition and classification of Produce affected by identically looking Powdery Mildew disease". In: *Acta Technologica Agriculturae* 17.2, pp. 29 –34. URL: https://content.sciendo.com/view/journals/ata/17/2/article-p29.xml.

Python Image Library (2020). *Image quantization used by the Python Image Library (PIL)*. URL: https://github.com/python-pillow/Pillow/blob/55e5b7de6c41b0386660b0bee7784ac04f412f4b/src/libImaging/Quant.c (visited on 03/25/2020).

Reeves, Adam (1981). "Metacontrast in hue substitution". In: *Vision Research* 21.6, pp. 907–912.

Russakovsky, Olga and Li Fei-Fei (2010). "Attribute learning in large-scale datasets". In: *European Conference on Computer Vision*. Springer, pp. 1–14.

Russakovsky, Olga et al. (2013). "Detecting avocados to zucchinis: what have we done, and where are we going?" In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2064–2071.

Russakovsky, Olga et al. (2015). "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3, pp. 211–252.

Sabour, Sara, Nicholas Frosst, and Geoffrey E Hinton (2017). "Dynamic routing between capsules". In: *Advances in neural information processing systems*, pp. 3856–3866.

Scikit-Learn (2019). *Neural network models (supervised)*. URL: https://scikit-learn.org/stable/modules/neural_networks_supervised.html (visited on 03/01/2020).

— (2020). *Example of VAE on MNIST dataset using CNN*. URL: https://keras.io/examples/variational_autoencoder_deconv/ (visited on 03/01/2020).

Shlens, Jonathon (2014). "A tutorial on principal component analysis". In: *arXiv preprint arXiv:1404.1100*.

Sim, Julius and Chris C Wright (2005). "The kappa statistic in reliability studies: use, interpretation, and sample size requirements". In: *Physical therapy* 85.3, pp. 257–268.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

Stallkamp, Johannes et al. (2011). "The German traffic sign recognition benchmark: a multi-class classification competition". In: *The 2011 international joint conference on neural networks*. IEEE, pp. 1453–1460.

Statistisches Bundesamt (Destatis) (2017). *Gemüseerhebung 2016 - Anbau und Ernte von Gemüse und Erdbeeren*. URL: https://www.destatis.de/GPStatistik/receive/DEHeft_heft_00070882 (visited on 03/25/2020).

Szegedy, Christian et al. (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Szeliski, Richard (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

Ting, Marie (2015). "Eine Region im neuen Zeitalter". In: *Frankfurter Allgemeine Zeitung - Verlagsspezial*. URL: https://www.hering\-international.com/fileadmin/media/archive1/downloads/presseberichte/2015-09-23_FAZ_-_Verlagsspezial_Industrierregion_S\%C3\%BCdwestfalen_-_Textilbeton_f\%C3\%BCr_Dior.pdf (visited on 03/16/2020).

Tjoa, Erico and Cuntai Guan (2019). "A survey on explainable artificial intelligence (xai): Towards medical xai". In: *arXiv preprint arXiv:1907.07374*.

Turk, Matthew and Alex Pentland (1991). "Face recognition using eigenfaces". In: *Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition*, pp. 586–587.

ul Hassan, Muneeb (2018a). *AlexNet – ImageNet Classification with Deep Convolutional Neural Networks*. URL: https://neurohive.io/en/popular-networks/resnet/ (visited on 04/24/2020).

— (2018b). *ResNet (34, 50, 101): Residual CNNs for Image Classification Tasks*. URL: https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/ (visited on 04/24/2020).

— (2018c). *VGG16 – Convolutional Network for Classification and Detection*. URL: https://neurohive.io/en/popular-networks/vgg16/ (visited on 04/24/2020).

United Nations Economic Commission for Europe (UNECE) (2017). *UNECE STANDARD FFV-04 concerning the marketing and commercial quality control of asparagus*. URL: https://www.unece.org/trade/agr/standard/fresh/ffv-standardse.html (visited on 03/25/2020).

Universität Osnabrück, Fachbereich Humanwissenschaften (2019a). *Modulbeschreibungen Cognitive Science*. URL: https://www.uni-osnabrueck.de/studium/im_studium/zugangs_zulassungs_und_pruefungsordnungen/fach_master/cognitive_science_msc.html (visited on 03/21/2020).

— (2019b). *Prüfungsordnung für den Master Studiengang Cognitive Science*. URL: https://www.uni-osnabrueck.de/studium/im_studium/zugangs_

zulassungs_und_pruefungsordnungen/fach_master/cognitive_science_msc.html (visited on 03/21/2020).

Vijayarekha, K. (2008). "Multivariate image analysis for defect identification of apple fruit images". In: *2008 34th Annual Conference of IEEE Industrial Electronics*, pp. 1499–1503.

Wang, Chi-Feng (2019). *The Vanishing Gradient Problem The Problem, Its Causes, Its Significance, and Its Solutions*. URL: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484 (visited on 03/01/2020).

Warden, Pete (2017). *How many images do you need to train a neural network?* URL: https://petewarden.com/2017/12/14/how-many-images-do-you-need-to-train-a-neural-network/ (visited on 03/25/2020).

Weber, Karola and Katrin Quinckhardt (2018). "Heimvorteil Spargel - Selbst angebaut, selbst zubereitet! Anbautipps und Rezepte für Spargel". In: Landwirtschaftskammer Nordrhein-Westfalen. URL: https://www.landwirtschaftskammer.de/verbraucher/rezepte/spargelrezepte.pdf (visited on 03/25/2020).

Wikipedia contributors (2020). *Precision and recall — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-April-2020]. URL: https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=945184172.

Zarandi, MH Fazel, Soheil Davari, and SA Haddad Sisakht (2011). "The large scale maximal covering location problem". In: *Scientia Iranica* 18.6, pp. 1564–1570.

Zhang, Xiangyu et al. (2015). "Accelerating very deep convolutional networks for classification and detection". In: *IEEE transactions on pattern analysis and machine intelligence* 38.10, pp. 1943–1955.

Zhang, Yudong and Lenan Wu (2012). "Classification of fruits using computer vision and a multiclass support vector machine". In: *sensors* 12.9, pp. 12489–12505.

Zheng, Alice and Amanda Casari (2018). *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc."

Zhu, Bin et al. (2007). "Gabor feature-based apple quality inspection using kernel principal component analysis". In: *Journal of Food Engineering* 81.4, pp. 741 – 749. ISSN: 0260-8774. DOI: https://doi.org/10.1016/j.jfoodeng.2007.01.008. URL: http://www.sciencedirect.com/science/article/pii/S0260877407000489.

Zhu, Xiaojin (2005). *Semi-Supervised Learning Literature Survey*. Tech. rep. 1530. Computer Sciences, University of Wisconsin-Madison. URL: http://pages.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf (visited on 02/16/2020).