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

The study report is submitted in partial fulfillment of the requirements for the degree of Master of Science

in the

Research Group Computer Vision
Institute of Cognitive Science

April 3, 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

TODO: 0.1 Structure of the project:

Some introductory sentences about the project and its contributors.

TODO: 0.2 Structure of the report:

Short overview of every chapter.

Contents

Chapter 1

Introduction

TODO: Introduction to the project and motivation behind it.

1.1 The project

TODO: Rough summary of the idea of the project. How did the idea come up? What is the project about?

1.2 Background on computer vision based classification tasks

TODO: Short introduction into the fields of machine learning, computer vision, and to artificial neural networks. Prospects and challenges of both on a general basis with respect to our issue. This part is kept brief.

1.3 Background on sorting asparagus

TODO: What is the main focus during the sorting process, i.e. why do you need to sort asparagus and in which classes do you sort it?
What problems and challenges will be met - including the difference of challenge for humans vs. machines.
Including the description of the labels as sorted by Hof Gut Holsterfeld (Figure 1.1).

1.4 Expected outcome vs. actual outcome of the project

Based on the literature review, we aimed to improve the current sorting performance of the XX machine at the local asparagus farm "Spargelhof Gut Holsterfeld". In this report, we investigated techniques from computer vision, both classical and deep learning based approaches. We expected to reach a result which is able to classify asparagus images into 13 different classes better than the current standard. For the initial performance of the sorting machine, no reliable accuracy of correctly sorted asparagus pieces is available, but between three and six workers are employed to re-sort wrongly classified asparagus pieces. The farmer himself assumes an accuracy of not more than 70%.

An advantage of our project is that it was directly supported by the local asparagus farm, providing training data and allowing us to evaluate our proposed solutions in a real environment during the asparagus harvesting season 2020.

However, there was a misunderstanding between us and the supporting asparagus farm about the kind of data we need. The existing images were too few, and also

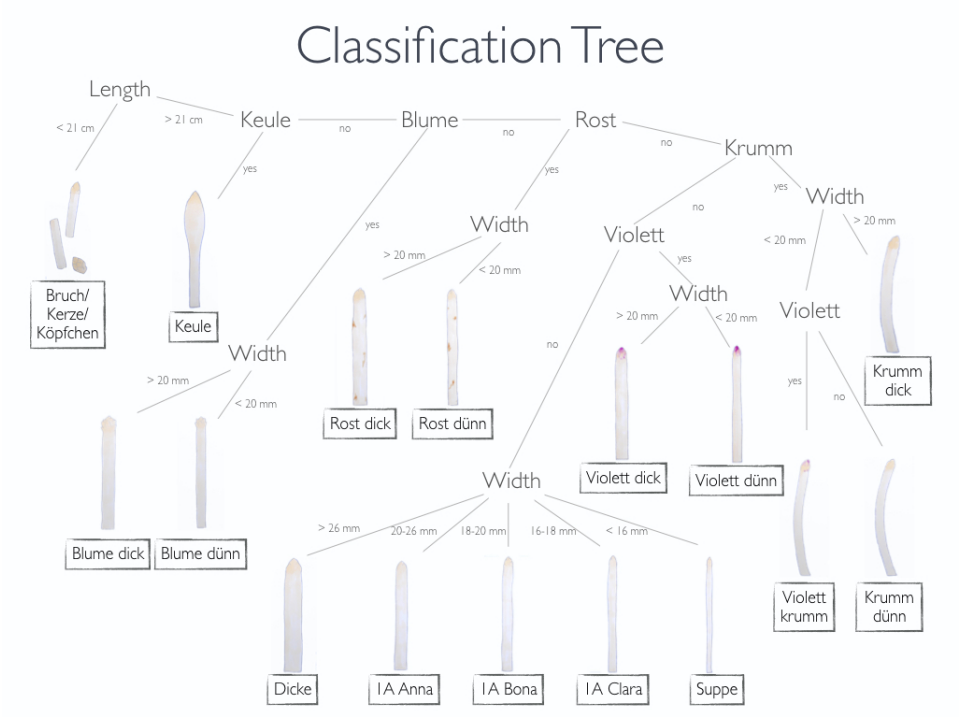


FIGURE 1.1: The decision tree for attributing a label to the asparagus as to the sorting rules of the asparagus farm "Gut Holsterfeld". Starting from the upper left corner of the image binary decisions are made until a label is reached (except for Width).

unlabelled. Therefore, we spent the first two and a half months with data acquisition instead of starting with preprocessing as we originally planned. Throughout the harvesting season 2019, we continuously went to the asparagus farm and collected unlabelled asparagus images during the normal harvesting procedure. For a small amount of asparagus pieces (xx in total), we collected labelled data by taking images of pre-sorted asparagus pieces. Pre-sorted in this context means that the asparagus pieces were sorted by the sorting machine and if needed re-sorted manually by professional workers.
The number of labelled images is insufficient to learn classes using deep learning approaches (Russakovsky et al., 2013) (Russakovsky and Fei-Fei, 2010) (https://petewarden.com/2017/12/14/how-many-images-do-you-need-to-train-a-neural-network/).
Therefore, we spent six months preprocessing and labelling the data manually. Preprocessing involved: organizing the large number of images, renaming the files, so that the three images of one asparagus piece can be accessed together, and performing automatic feature extractions (ref to preprocessing).To label the images by hand, we wrote a custom application (reference hand-label-assistant). The final labelled data set contains over 10.000 (genaue Zahl an stangen) labelled asparagus pieces. Next, we worked on numerous classical and deep learning approaches (ref to chap 4). We reached several very interesting results which will be discussed in Chapter 4. Although we developed an end-to-end prototype, we did not deploy it onto the sorting machine for the harvesting season 2020. Even though some of our results are highly promising, it is therefore difficult to compare it to the current performance of the machine.

Hier noch ein Absatz mit den Inhalten aus der Conclusion – sobald diese steht.

Chapter 2

Data acquisition and organization

The value and purpose of a study project does not only lie in the implementation and realization of a long-term scientific study - in our case, computer vision-based label prediction to improve an agricultural sorting machine - but also in learning to organize, schedule, and distribute tasks for a larger team over the period of one year.

To accomplish the objective of the project, a working structure had to be established. The project members had to learn each other's strengths and weaknesses to form a robust and efficient unit. Self-organization and team building became key to the success of the project. Thus, the secondary focus of the report lies on the management of the study group.
The process of team building and structuring is captured in the first half of this chapter. A timeline of the different working stages can be traced in a roadmap created at the start and updated at the end of the year. It reveals how well the team succeeded in predicting the duration of the various tasks.
Further, the communication between the members is addressed which includes regular meetings, agreements on different issues, or the working platforms that were used. Additionally, the teamwork is evaluated which comprises task distribution, the democratic working structure, and the (structural) integrity of the team.
After the organizational part of the project, the second half of this chapter is concerned with the collection of the data and the literature research. The sorting machine is described as well as the process of saving and retrieving the recorded images. It is followed by a summary of the literature which was thought to be relevant to our topic regarding the visual detection of agricultural products or images with low variance with machine learning approaches. There was no specific paper, however, which could be used as a basis for our project.

The first section of this chapter, 2.1 Timetable (roadmap) of the project, gives an overview of the time management of the study group.
It is followed by the chapter 2.2 Organization 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 is described in more detail.
The last section, 2.4 Literature research, presents the collected literature concerning our issue.

2.1 Timetable (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. Near the end of the year, this map was evaluated

and updated to mirror the actual time spent at each project stage and to check how accurately the project had been planned from the start. If there were great discrepancies between estimation and reality, the map helped to acknowledge the wrong judgement of time and which working stages had taken longer than others.

The following timetables in Figure 2.1 give a broad outline of the major stages of the project. In the left figure, it was estimated how much time for a specific phase is needed, whereas in the right figure the time spent for the stage is given. Both figures are structured to display the year in a circle, starting in April of 2019, and running clockwise through the year until April 2020. The months are represented by the inner circle while the outer circle marks the different working stages.
The project comprised four to five major stages. The following stage descriptions are shortened for the purpose of simplicity. A more detailed representation of the single tasks attributed to each stage can be found in the ensuing roadmaps in Fig. 2.2 and Fig. 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 labelling the image data. This phase is split into two phases for the right figure displaying the actual time, 'Preprocessing' and 'Labelling'. 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. It is noteworthy that the different stages overlapped to a certain degree, because minor tasks of a previous stage were still in progress while tasks from the subsequent stage had already started. For the purpose of this figure the start and end time is displayed as a hard boundary.

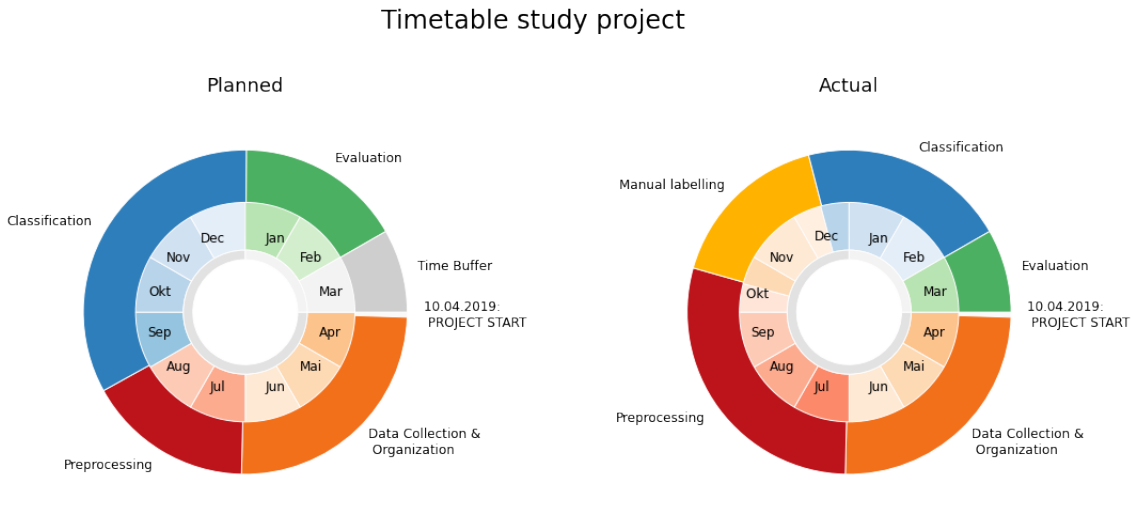


FIGURE 2.1: On the left side is a timetable with the estimated time of the study project during the year from April 2019 to April 2020. To the right, the timetable displays how the time was spent.

When comparing both figures, some distinctions can be recognized. The figure to the left shows that - while it was noted that the Preprocessing step of labelling the data would take some time - it was not estimated to take longer than until the end of August. Compared to the Preprocessing step in the right figure, it is observed that the phase continued until October. Furthermore, the time for labelling

a sufficient amount of images was underestimated. In the right figure, the labelling process received its own stage to highlight the fact that it demanded much more effort and time compared to the initial estimation. This exchange required that the time buffer and parts of the time calculated for the later stages of classification and evaluation had to compensate for the time loss. The time difference can best be seen when comparing the respective colours (that is, the brighter colours of red, orange and yellow to the darker colours of blue and green). While the main focus of this project was supposed to be the application of different machine learning techniques to classify the data, the data generation and preprocessing posed to be the most time-consuming stages.
As a result, both timetables differ in that the year was more optimistically planned than realized. A major factor was the lack of experience of the participants concerning not only the conduction of a larger project with many co-workers but also the general implementation of the preprocessing stage for machine learning classification. Another factor influencing the shifted timeline was the working structure of the team. It was re-evaluated during October to better fit the needs of the group members and avoid distributing so-called 'bottleneck' tasks (that is, tasks that are decisive for the continuation of the next steps) to single members only. All in all, a valuable lesson learned during the project was to not underestimate a preprocessing phase.

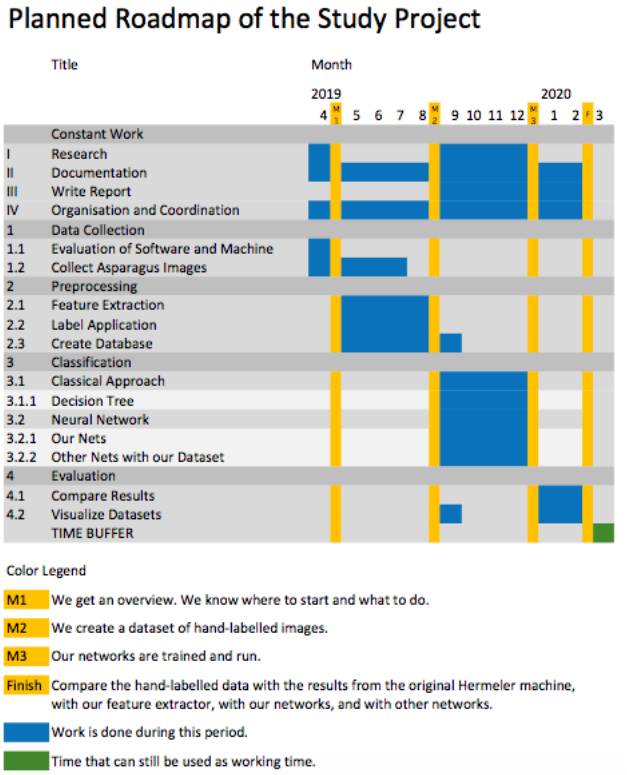


FIGURE 2.2: The figure shows the planned roadmap of the study project. It reveals how the time needed for each task was estimated in the beginning of the project.

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 the division into the major stages except for the first heading, 'Constant Work', which shows the tasks that demanded continuous attention

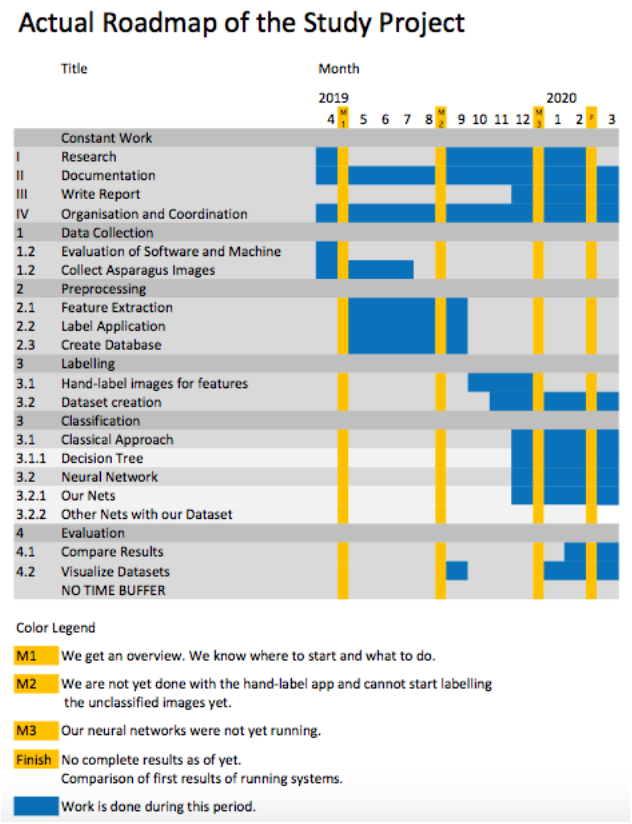


FIGURE 2.3: A roadmap that shows how the time was spent.

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

Comparing both roadmaps, the shift of focus is seen more clearly and can be associated with the duration of single tasks. Especially the prominent drift of the classification stage mirrors the fact that the time estimation is worth improving. The roadmap helped to better assess the time needed for task completion. In the following chapter, the management of the work distribution and the communication are taken into focus.

2.2 Organisation of the study group

In this chapter, we will describe how we have organized ourselves to work together on the project. For this we will elaborate on the tools we used for communication and organization as well as on the structure of our group work and our teamwork in general.

2.2.1 Communication

Our main way of communication were frequent meetings in which we discussed our process work period and distributed tasks for the next one. In addition to those meetings, we used different platforms, which worked with varying degrees of success. After a presentation about possible strategies of cooperation, we initially decided to use Asana, GitHub and Telegram to facilitate the communication outside of our meetings. The different means of communication will be described

and evaluated in the following abstract.

Regular meetings have been the most helpful in organizing our project. During the meetings, which usually lingered for one or two hours, there was a discussion leader and a protocol writer. The protocols were saved for review in our GitHub project. The meetings were characterized by long discussions about the best approach for the following steps and the possibilities to tackle the next challenge. Our supervisors were almost always present at the meetings, they brought in their expertise and gave us the opportunity to ask concrete questions. During the first half of the project, tasks were always distributed at the end of each meeting and in the next meeting the progress of the work was discussed. The communication was changed in the second half of the project, where we started to use a schedule that described the different tasks and deadlines in detail, and to gather for co-working. The organizational meetings were continued and used to discuss the current status with Ulf and Axel. Everyone reported about his area of responsibility. This helped us to spend less time discussing and debating and more time working on our tasks.
Telegram is a cloud-based instant messaging service for the use on smartphones, tablets and computers. Since the first meeting, we had a constant conversation in a group chat on Telegram, in which we arranged us for the weekly meetings, planned hand-overs of keys/transponders for the room or informed each other about the status of the project. The group chat also created space for mutual motivation when needed. The majority of important information was exchanged via telegram.
Asana is used to distribute tasks and to communicate projects successfully. Many integrations of other applications, such as Slack, can help to achieve this. We used a board view where we listed tasks in different sections. But this function alone did not help us much in our project. The tasks were easier to distribute in direct consultation at physical meetings and demonstrated or discussed after completion. So it happened that Asana was not used enough to be helpful. If we had relied on communication with Slack or other agreed services or applications, it might have made more sense, but asana alone was proven to be inefficient in our use case.
GitHub is a web-based popular platform using a version control system using Git that helps developers store and manage their code, and track and control changes to their project. During the project, we learned how to use it. Initially, a presentation was given by Katharina on how to use Git, because there are a few rules to follow to keep it straightforward. Git allowed us to work from anywhere, which was very helpful for us. We also automatically created a documentation with Sphinx.

2.2.2 Teamwork

This subchapter outlines the team forming, the team members and the cooperation in our study project. It starts by introducing the team members and their previous experiences. This is followed by a description of the practical aspects of teamwork, the work structure, and the distribution of project-relevant tasks.

First, the team members and their respective backgrounds are illustrated before delving into further work-related task distributions.
The project was an initiative of one of the students and, hence, a large part of the project members are friends that were inspired by her ideas. Other 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, which had an influence on the dynamics of the teamwork. The team was initially made up

by Malin Spaniol, Maren Born, Katharina Groß, Josefine Zerbe, 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. This was an unfortunate occurrence because he posed a valuable source of knowledge in the field of computer vision. Further, Luana and Subir left in October to pursue different study projects.

By the start of the project, all except two of the members were in their master's degree in Cognitive Science at the University of Osnabrück. 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 'ANN with Tensorflow', taught at the University of Osnabrück, while some had also taken machine learning classes during their study program. None of the members had prior knowledge about project management or task organization on a broader level. Git was previously only used by three students so far, but none of them were experts on its usage. Further, no participant had any experience with the Grid system of the IKW, not to mention running jobs on different machines.
Thus, the project started with 10 members, where each of the participants brought a different level of experience in the most relevant fields of machine learning, that is computer vision and neural networks, into the team.

After introducing the team members and their backgrounds, this section continues with elaborating the structural organization of the team and the work distribution. As none of the members had any previous experience with team formation, in the beginning, the team lacked some structure and a clear distribution of single roles inside the team. One reason for this was the harmonious atmosphere between the single members. Many of the team members had a friendly relationship already from the start of the project. Also, further tasks at the beginning, like the trips to the asparagus farm, strengthened the team spirit and the social interactions positively. So we started with a very dynamic structuring of tasks by making every decision democratically. As described in more detail in the next chapter, we first used the joint meetings to distribute the tasks. As an example, we had to prepare a schedule for organizing the data collection and already started with preprocessing tasks. These two main activities were mostly done in smaller teams of two to three people. Rather few tasks were done alone. In our meetings, we often formulated possible next tasks, even for the distant future, but as the project progressed we did not assign them to a specific person or working group.
In August, we restructured our own organization. On the one hand, this was due to the fact that the tasks changed and thus a new structure was more appropriate. On the other hand, also the weaknesses of the individual team members were a reason for the restructuring. Some team members had less programming experience than others and therefore had difficulties realizing certain tasks at the same time and with the same precision than others. Even if they had good ideas in terms of concept, it was not possible for them to implement these ideas quickly enough

so that they could be included into the project. In addition, the democratical self-organization and difference in programming expertise led to a distortion in the time management of the group and some tasks were not finished in time (with other tasks). To integrate more of the strengths that the single team members brought and to tackle the issue of time management, it was decided to write a roadmap that distributed the work more appropriately, gave an overview of the tasks that still had to be done and how much time was left to do them. Further, common working hours on campus were introduced as well as a division of responsibilities (work-related and related to the supervision of tasks). The common working hours ensured that questions and decisions that arose could be discussed directly. This was especially helpful when different tasks overlapped and required communication and agreement. The supervision of the work was divided into manager roles, which means that the work was split into different main fields where each member was responsible for managing their assigned area, distributing tasks and keeping an overview of the relevant work inside their working field. Furthermore, one knew who to turn to for questions and when in need of discussion or feedback. The meetings became more effective due to the new structure, and there was less discussion concerning task distribution. As we distributed roles, we were also more responsive to the strengths and weaknesses of the group members.
Therefore one can say that the team structure and 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 supervision and manageable task distribution.

In summary, we can say that we have not only learned new scientific skills and techniques of data acquisition, preparation, and analysis but also gained valuable new insights into the organization of a large project with many members. We understood how to organize ourselves more successfully and purposefully. First and foremost, we learned that this includes excellent communication. The whole team agrees that we would structure the next project in the same way as we did in the second half from the beginning on. Communication is important, but it was discovered that there is a right way to balance the amount and effectiveness of communication. Not everyone has to discuss or listen to all the details in every area, often it suffices when all team members have a broad overview. Additionally, it turned out to be helpful when, one or two members exchange some of the task-related work in favour of more management-related work. This helps to gain a better team structure, time management, and, in the end, make the team work more effectively and efficiently towards its goal.
Specifically, the experience of two different working structures gave us the ability to better judge how good teamwork is achieved, how each member can better include themselves into the team, and what each member can improve for future teamwork. As the main intention of the study project was understood as a learning unit, we wanted to seek out tasks that we are motivated to do and that bring us new experiences and skills, and not just practice what we already know.

In the course of the project and to sum up the focus of this chapter, namely the organisation of our study group, it can be said that we had a harmonious team with two different working structures, one cultivated at the start of the project and the other one at the end of the project, respectively, which happened due to the tasks amount and the optimisation of our teamwork. We have all learned that teamwork is not a trivial issue, even for members who already know each other well.

2.3 Data collection

TODO: How we collected the data and what the data looked like.

INTRODUCTION

- Driving to asparagus farm "Gut Holsterfeld"
- First look at the data.

MAIN

- First problem: no labelled data.
- The asparagus sorting machine
  - How is the data recorded?
  - How are labels calculated? Could we use them?
- Second problem: image saving stop after 1000 images.
- First solution: Teamviewer sessions.

SECOND SOLUTION: File moving service

The requirements for the software are described below. Because of the problems described, a solution was requested that runs in the background and does not disturb the workflow during the sorting process. The operating system on which the sorting machine runs is Windows. Consequently, after some internet research on background processes and programs, the decision was made to use a service. Windows was used, therefore, the development was done with the DOTNET framework in the programming language C. As discovered during the internet research, the package provided is called "Topshelf" (https://github.com/Topshelf/Topshelf). Topshelf is a service hosting framework for building Windows services using .NET. This makes it 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. The function of the service is based on the FileSystemWatcher object (https://docs.microsoft.com/en-us/dotnet/api/system.io.filesystemwatcher?view=netframework-4.8) from the System.IO namespace. In the main program a list of files in the source folder is kept. Files 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. After a short time the service was adjusted. Then, we introduced the described hourly waiting time, because the program of the sorting machine itself still accesses the images indefinitely.

2.4 Literature research

TODO: Previous literature research concerning food classification.

- Searching for background literature close to our project, e.g. automatic CV-based sorting of other food products.
- Researching semi-supervised and unsupervised learning approaches.

- Re-reading on potential ANN structures that we could use for sorting.
- What was the result?
- Could we rely on a certain paper/process? Did it work?

Chapter 3

The dataset

Before implementing any approach that lets us predict a label to a respective asparagus, the recorded image data has to be preprocessed and arranged in a sensible format making it accessible (or even simpler to work with) for any computer algorithm. We are going to describe in detail how the raw images were processed into a dataset that is ready to use and how we obtained labels for the supervised learning approach. Further, we will report the sorting criteria that we decided on and the evaluation of our label agreement. Also, we are going to outline the classical computer vision approaches that we used to detect certain features from the image data without a machine learning process.

As a first step, the images were loaded and saved on the IKW internal Grid, where the data was put in a fitting style and format. Then, the images had to be checked for any general errors that might have occurred during the recording and ordered in an understandable and structured way in the single folders on the Grid.
It was followed by different preprocessing steps where the three images of one spear were gathered at one location, and the removal of the background and any other unnecessary information in the images. Unfortunately, most of our collected data was not labelled and a solution to the issue of how to attribute a fitting label to each spear was needed. This proved to be no trivial task for the accumulated amount of data.
The first approach was to automatically extract the single features in the image which are responsible for a spear's label. The idea was to use classical machine learning approaches, f.e., a threshold for the RGB pixel values in the images, to obtain the individual features. As it was not possible to completely extract all the details from the pictures (especially features like the detection of a flower posed great difficulties) it was decided to label the images by hand. For that purpose, all implementable feature extraction functions were combined in an application (app). The hand-label application has a graphical user interface where the user can click through the different feature categories, thereby indicating the presence of a feature. Some of the automatic feature extraction scripts were not able to classify an image on their own but could be used as an assistance to the user of the app.
The second approach was to manually label all images with the application, in that every member of the group had to look through a batch of asparagus images and note whether a certain feature from a set of previously defined features is present in the image. The manual label process bore its own difficulties as vague images led to indecisiveness in the feature detection and a variance in the overall labelling agreement of the single members. The sorting agreement was therefore derived and used as a measurement of the overall goodness of the manual sorting process.
Finally, after all the different preprocessing and labelling steps were concluded, the dataset was built. To have a unified version of all labelled images and their respective labels, a reading function was created that enabled the later approaches presented in chapter 4 to assess the data.

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

3.1 Preprocessing

An important, often underestimated, step of any machine learning project is the preprocessing. After collecting the data, it is mostly not in the right format yet, but needs to be altered for the particular purpose.

In our case, the first step of the preprocessing was to combine the three perspectives of each asparagus piece. The sorting machine, that we are working with, takes images that show three compartments at a time and takes an image each time a new asparagus piece is in the center. That means, each asparagus can be found in three pictures, one in each of three positions - left, center and right. The names of the images can be used to combine the three relevant images and determine in which position the asparagus was captured. This way the images can be cut into three pieces and renamed in a way that makes it clear which images belong together. Each asparagus gets a unique identification number and the three perspectives are denoted with "a" for left, "b" for middle and "c" for right. For example, the image 42_b.png is the middle image of the asparagus with the identification number 42.
The second step was to remove the background to get rid of as much of the unnecessary information as possible. As the conveyor belt that transports the asparagus pieces is blue in color and there is a high contrast to the bright asparagus pieces it could easily be removed. First the asparagus piece is masked using the color hue of the HSV representation of each image. Second, very dark regions are added to the background based on a mask retrieved by thresholding the value component. This is particularly important for the automatic feature extraction as one can see in the following chapter. Additionally, the asparagus was rotated upwards for some of the applications to reduce the variance that is due to differences in the angle, in the hope of making the machine learning task (that has to be accomplished by the classifiers) more simple and hence reducing the required number of samples. This is especially important as labels had to be generated manually. Moreover, reducing variance in dimensions that are irrelevant with respect to the properties of interest is a prerequisite to apply unsupervised methods such as PCA or autoencoders and the semi-supervised approaches that are based upon the earlier as. This is achieved by binarizing 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. The estimate for the angle are retrieved using arcus tangens 2 and the depictions are rotated accordingly. Thereby, another dataset was created.

During preprocessing the raw images are collected, sorted by name and the three perspectives of each piece are identified. Then, the images are cut into three parts and the left, middle or right part is kept, respectively. Afterwards, the background is removed, which means pixels that are not part of the asparagus are set to zero. This is done by masking all pixels with a blue hue in HSV color space and all pixels that are not bright enough as background and set them to zero. On top of that, all connected regions that remain after removing background pixels by hue and brightness are also removed except for the largest one. By doing that, any distortions like reflections or single noisy pixels are set to zero and only the asparagus piece itself remains. The only faulty areas that can be left over with this approach are areas that are connected to the asparagus. But as this is only rarely the case, its effect is neglectable. The position of the asparagus within each cutted image was further improved by centering the fragments of the image around the center of mass. This way the asparagus is always in the middle. If the new cutting line falls out of the image boundaries, the missing area is filled with zeros, too.

It should be noted that, although it is an advantage to have several perspectives on the asparagus, as some features might only be visible from a certain side, it also bears some problems. Namely, the lighting and reflection are slightly different, which can alter the color values, and the images can be a little distorted. Hence, the asparagus might appear wider or smaller depending on the perspective. This makes it harder for classical computer vision algorithms to determine some features accurately.

Another version of the dataset was computed by converting the depictions to color palette images. An appropriate palette and the respective mapping of RGB values to palette colors was determined using clustering in color space. First a set of RGB tuples is collected by adding pixel values for the depiction of 10.000 asparagus pieces. Second the resulting list of RGB tuples is converted to an image and third a palette is determined using an algorithm 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 and represent the color pallette (Zarandi, Davari, and Sisakht, 2011) (https://github.com/python-pillow/Pillow/blob/55e5b7de6c41b0386660b0bee7784ac04f412f4b/src/libImaging/Quant.c). Each image of the downscaled version of the dataset was transformed to the palette representation. Visual inspection showed little quality loss such that it can be assumed that the relevant information for image classification is well preserved.

Several additional datasets were computed based on the background removed version. This holds for downscaled versions as well as for a version that contains the asparagus heads only. To compute the latter the images were padded to avoid sampling outside of the valid image boundaries and the uppermost foreground row was detected. Subsequently the center pixel was determined and a snippet of the image was cropped such that this uppermost central pixel of the asparagus is contained as the center of the uppermost row of the snippet. The resulting snippets of asparagus heads are rotated using the centerline regression approach described above. The approach has proven reliable and the resulting depictions were used to train a dedicated network for head related features (see XXX).

3.2 Automatic feature extraction

Beside the main aim of this study project, to use machine learning algorithms to classify asparagus automatically, also classical computer vision methods were implemented for comparison. We used these methods to write algorithms that take the images as an input and return a prediction for the different features as an output. The goal was to predict the features reliable enough to deduce the corresponding class from them.

According to the producer of this and other sorting machines, the softwares currently on the market use very similar methods, which gives us reason to believe that the classical methods might not be adequate to yield a better classification than the current status quo. Nonetheless, it is interesting to see, which features are harder to detect and which are easier.
The results vary greatly between the different feature extraction methods. While the functions to detect the width and length and whether the asparagus is violet or bended were good enough to be integrated into the hand label app to help us label, other features turned out to be more difficult.

In the following, the different automatic feature extraction methods will be described, alongside with the results that were achieved and future steps that could be taken to improve the results further. For each feature detection method, the images with removed background were used.

3.2.1 Length

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 background pixel and therefore not zero. The asparagus was rotated upwards, as described in chapter XX, in order 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 piece. But in reality, there are not a lot of asparagus pieces close to the decision boundary between broken and a whole piece. Usually, the asparagus is picked 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 while in the sorting machine. And if they break they generally break closer to the center of the asparagus rather than at the ends of it. Therefore, the difference in length detection does not matter for our classification.

All in all, the length detection yields good results that are very helpful for the 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 broken and not broken. 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 or learned with a machine learning approach.

3.2.2 Width

The width detection uses a very similar approach as the length detection as 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 different rows from which the mean width was taken. For the label app we decided that three measurements are sufficient, because asparagus spears usually do not show drastic changes in width at several positions. Hence, if there are larger changes at all, three evenly distributed positions should capture those changes well.

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 upper-most 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 below the uppermost pixel in order to bypass the head area with certainty. As described in the length detection, also in the case of the width detection curved asparagus pieces might lead to slightly different outcomes than the true values, but again this difference is regarded as irrelevant in our case.

3.2.3 Rust

The 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. But as 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 blue value between 30 and 55 are considered as rust.

Visual inspection shows that the rust detection algorithm works well to detect rusty areas and barely misses any rusty parts. But it is difficult to set a threshold for the number of pixels needed to be classified as rusty, because many pixels with a brown color that are distributed over the whole piece are not supposed to be classified as rust. Only clusters of brown pixels are a reliable indicator for rust. To make this intuition more clear, it could be the case that two asparagus have the same number of brown pixels, but in one case they are all connected building a cluster and in the other case they are evenly distributed over the whole asparagus piece. That would mean that, although both yield the same output value, only one of them should be considered as rusty. However, it should be mentioned that this is merely an artificial example to display one draw-back of the implementation. In reality, it is very

unlikely that an asparagus has a large number of brown pixels that are not in fact rust.

Nevertheless, it remains unsolved to set a robust threshold that works well on the whole dataset. 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 as 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 easily, but have to be communicated well to the farmers.

3.2.4 Violet

Especially when exposed to light asparagus pieces tend to change in color. An initial shift from the desired white appearance to a slightly rose or violet tone can be observed first. This change in colour, that is considered a flaw according to German quality standards, typically affects the head of asparagus pieces. However in some cases a non-uniform distribution of more or less violet areas can be observed. Moreover, it has been shown in practice that colour impression is highly subjective which manifested in discussions of edge cases during labeling. Effects of meta contrast potentially even affect the colour impression on an individual level. The lack of a formal definition for violet asparagus pieces also poses challenges to approaches of measuring this feature.

In a simple approach to measure whether an asparagus piece is violet or not, colour hues are evaluated. More precisely, this strategy is based on evaluating histograms of colour hues that are calculated for foreground pixels of the asparagus images after converting them to the HSV colour space. Pale pixels are removed from the selection by thresholding based on the value component of the HSV representation. Finding the optimal threshold was proven difficult and corresponds to the abovementioned question of what it means for an asparagus spear to be violet. A value of 0.3 was considered a good compromise. All three perspectives were taken into account to compute one histogram per asparagus piece. A score was calculated by summing up the number of pixels that lie within a violet colour range. A second threshold was used as the decision boundary for violet detection. The direct and intuitive feedback in the label app showed the relation between varying thresholds and the prediction. It has shown that lowering thresholds also means the feature extractor becomes more sensitive at the price of a reduced specificity. Best overall matches (accuracies) with the subjective perception were found for very low thresholds. In many cases, however, measurements based on this definition of "violet" did not match the attributed labels.

Hence, another sparse descriptor was derived from the input images. However, instead of thresholding pale values and calculating the histograms of colour hues this approach relies directly on the colours that are present in the depiction of an asparagus piece. As the 24 bit representations contain a vast amount of colour information in relation to the number of pixels it is, however, unfeasible to use these as an input. Instead the colour palette images can be used. Histograms of pallete images can serve as the basis to define the feature "violet" in a way that captures more of the initial colour information while being simple and understandable enough to allow for customizations by users of sorting algorithms or machines. As a consensus regarding such a definition is arguably hard to achieve and somewhat arbitrary the descriptor was used to learn implicit definitions of the feature (see XX).

It has been shown that explicit ways of directly measuring, in how far an asparagus piece is violet can be implemented but heavily depend on the definition of this feature. As colour perception is highly subjective across and even within subjects machine learning approaches that are trained on human labelled data appear to be promising. Using them can help generalizing a definition for the degree to which an asparagus piece is violet.

3.2.5 Curvature

Multiple curvature scores can easily be computed based on regression fits to the centerline of an asparagus piece. For example, the parameters of linear or polynomial regression can be interpreted as a description of the curvature of an asparagus spear. However, the question remains if a scalar descriptor that matches with the subjective definition can be derived.

Deriving sparse descriptions is based on a two stage approach. In the first stage, the centerline of an asparagus piece is computed because it is considered to be a good description of the curvature of asparagus pieces. Preprocessed images are used as the input where the background is removed and replaced by black pixels. Each asparagus piece is approximately oriented vertically. 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 the simplemost 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 pieces and increasingly large for bent ones. As an S-shaped piece 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 (e.g. at least one value indicates curvature) for the results for all three perspectives yields a rule to measure curvature. However, it has again proven difficult to set thresholds appropriately to reliably capture the visual impression. Hence, another sparse representation was calculated by fitting linear regression to each of six segments of each depiction of an asparagus piece. An MLP was trained on the resulting 18 angles per piece (see XX).

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 asparagus curvature. Curvature scores can serve as an input to a machine learning approach that uses the result of feature engineering (see XX).

3.2.6 Flower

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. On the one hand, the shape of the head was used as an indicator for a flower. The idea was that asparagus pieces 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. On the other hand, the structure within the head was examined. Supposedly, asparagus with flowery

heads exhibit more edges and lines in the head area. In both cases, it was 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 one to four, 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. There are newer versions of the machine available on the market that have an additional camera which solely takes images of the head of the asparagus piece. This way the inspection of the head improves considerably and the detection of flowery heads should be facilitated. Unfortunately, no such machine is available to us at the moment.

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

Providing a sufficiently large dataset that contains information about the ground truth of target categories is one of the major non-algorithmic challenges in the application of machine learning for classification tasks. In many cases, however, the respective labels are missing. This is especially problematic if traditional supervised learning methods such as simple feed forward CNNs are employed. Depending on the variance in the input images a very large number of samples are required. While suitable means of preprocessing may help to reduce the variance in the source data the amount of variance in images arguably remains high. Strategies on the algorithmic domain such as the use of pretrained networks as well as manual feature engineering (by means of traditional computer vision or machine learning) may help to retrieve sparse representations without the requirement for training on labeled datasets. Similarly semi-supervised learning aims at extracting a sparse representation of images and tying the contained features to the desired target categories. As such these approaches promise to reduce the requirement for very large labeled datasets. However, without sufficiently large labeled datasets also these approaches fail. In addition it is essential for the evaluation of machine learning algorithms to draw on labeled data. If target labels are unknown it is hence inevitable to manually generate them.

Generating labels manually is a common practice, however it requires plenty of effort. Human performance is commonly acknowledged as the baseline or "gold standard" that image classifiers are evaluated by. Hence in many scenarios data is labeled by human coders such that machine learning algorithms can be fitted on the training subset of the resulting hand labeled datasets and evaluated on a test subset. This holds especially for image classification tasks. In the present case some features were considered to be reliably measurable by means of computer vision (e.g. the width or the length). For features such as a flowering asparagus head or the evaluation whether or not a piece is affected by the rust fungus this has proven to be difficult. Considering the amount of data that could potentially be labeled, an interface is required that allows for efficient attribution of labels: For this purpose an app with graphical user interface is developed that allows for efficient attribution of labels.

The hand label app comprises two user interfaces. A startup window that allows for a preview of asparagus images and the attributed labels (represented by Ui_Asparator) as well as the main labeling interface (Ui_LabelDialog). 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 input path and output path is set and a dictionary of indices to images can be parsed from filenames and the minimum indice can be determined. As such the label dialog and the respective controller class always resides in a valid initial state. For labeling the user answers questions that are displayed alongside the images that depict each asparagus piece from three perspectives. To facilitate this process the arrow keys can be used. The result is saved and the next question will automatically show up upon answering.

In addition, automatic feature detection can be selected for specific features. The result is displayed and saved to file. The user is not asked to answer questions that target named features and attribute labels automatically. This flexible approach was chosen as it was initially unclear and disputed in how far automatic feature extraction yields results that meet up with the individual perception. It also allowed to improve automatic feature extraction methods and to develop a direct intuition for the relation with the data. On top of that it has proven to be useful for debugging automatic feature extraction methods that failed for some images.

The development of the app was accompanied by three major challenges. First handling a large dataset of several hundred gigabytes that is accessible in a network drive. Second changing requirements especially due to group decision processes related to the automatic feature extraction and unforeseen necessities in (parallel) preprocessing that made substantial changes of the initial architecture necessary. Third the related question of the handling of internal states of the app. The latter may be further explained shortly.

Internal states of the app were handled such that it is possible for the user to navigate into invalid states (i.e. set the asparagus ID accordingly) where no images are accessible. Note that preprocessing was done such that each asparagus piece has a unique Integer-ID and a specifier for perspectives a, b and c in it's filename. While generally IDs are in a continuous range from 1 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 few missing indices although jobs were started with. In addition, the large amount of data did not allow to save all files in a single directory. This means one could not simply iterate over asparagus pieces, determine the file path and display the related images. Instead parsing filenames from a slow network drive is necessary which requires limiting the number of selected pieces. In addition standard GUI elements such as spin boxes and keyboard controls allow to increment the asparagus ID. Employing them means, however, allowing the app to have an inconsistent state where the current ID is invalid (because there is no respective data). Hence, the cascade of methods that require the respective input was adjusted such that they can handle this case.

The app was implemented using the PyQt5 framework while coarsely following the model, view controller principle. No separate database and the respective model class is used. The data model is implicitly parsed from the filenames and administered in attributes of the view. The labels are administered 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 was mounted on the level of the operating system. QtDesigner was used to design the Views. Four manually coded classes are essential for the architecture of the app. The class HandLabelAssistant

where the PyQt5 App is instantiated, the controllers MainApp and LabelingDialog as well as Features which is member class of the latter, 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 was required to display images with the correct aspect ratio. The figure below 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 for more than 10.000 labels in a manageable amount of time. Details of the labeling process are described in the next section.

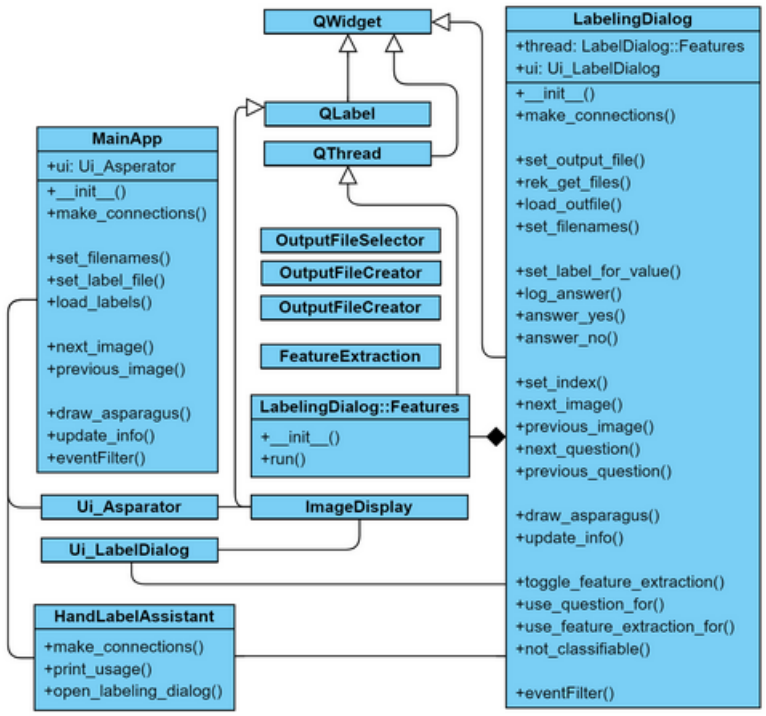


FIGURE 3.1: ??

3.4 Manual labelling

In this section of chapter 3, the process of manually labelling the data with the help of the hand-labelling app is laid out. It includes the sorting criteria which allocate each spear to a single asparagus class, the practice of manually labelling the preprocessed data for its features, the outcome of the labelling process, and the approach to measure the agreement of the manual labelling are described in more detail.

The images were classified by all members of the group. As none of the group participants are experts of labelling asparagus, a general guideline for the labelling had to be established which is elaborated in detail in the following subsections. The guideline is written in accordance with the owner of the asparagus farm of Gut Holsterfeld, Mr. Silvan Schulze-Weddige. He was consulted in all questions regarding the sorting of the asparagus. General challenges in the manual sorting in front of

a computer screen, including the respective image quality, and especially the variance in the agreement of the project members were expected from the start. As the task relies on the subjective view of single humans, opinions about the affiliation to one asparagus class can diverge in critical or vague cases. This is demonstrated in that the seasonal workers who usually sort the asparagus by hand do not agree on every case. Further, a classification session with Mr. Schulze-Weddige was held in which the project members learned to distinguish the different features that decide the class of an asparagus spear. During the session, again it emerged that some images containing critical spears - that is, where the features are not apparent - were hard to classify even by the expert.
To tackle the issue and to have an overview of the general agreement of the sorting between group members of the project, a measurement unit was researched and applied, namely the Kappa Agreement. It was applied on a smaller subset of presorted asparagus pieces in the beginning of the manual labelling and a second time at the end of the manual sorting process. The Kappa Agreement was used to assess the degree of accordance in sorting between the single members and monitor how the sorting agreement developed during the manual labelling process.

The first subsection, 3.4.1 Sorting criteria, expands on the guideline used for categorizing each spear of asparagus according to its features (into a single class).
In the second subsection, 3.4.2 Sorting outcome, the process of the manual sorting is shortly described and the achieved results are presented.
The ensuing third subsection, 3.4.3 Agreement measures, describes the theoretical background on the search to find a measurement for the group's agreement.
3.4.4 Agreement measures, focuses on the Kappa Agreement, which we chose as an evaluation measurement for the accord of our sorting, and the results of its application.

3.4.1 Sorting criteria

The class of an asparagus spear is decided by several factors, ranging from its shape to its colour. Put together, these single features give us the label for the spear.
As it was decided to label the images for their features, the features were checked by the labellers. The images were displayed with the hand-labelling app and the features are divided as follows: 'fractured', 'hollow', 'blooming', 'rust on body', 'rust on head', 'bent', 'violet', 'very thick', 'medium thick', 'thick', 'thin', 'very thin'. Further, images that could not be sorted thoroughly were sorted as 'not classifiable' (e.g. when the spear is cut off the image). To some extent, the manual feature categorization was influenced by the price that each category of asparagus earns. High quality pieces (i.e. 1A Anna) should be thoroughly examined and sorted more conservatively.
In the following text, the different criteria for manually labelling the data images for their corresponding features are described.

Fractured
An asparagus spear includes the feature 'fractured' when it is broken, or if it does in any other way not fulfill the required length of 210 mm.
First, the feature was labelled manually but the procedure was abandoned shortly after because it can be derived more precisely from the computer-vision based feature extraction 'length', which is automatically calculated within the labelling app. It is then discerned between a fractured piece with a head and a fractured piece without a head. Whenever a spear was fractured but with the head part still attached, it was sorted like a usual, intact asparagus. However, if the head part of the

spear was damaged or missing completely, no other features were chosen but the asparagus was labelled as 'not classifiable'.

Hollow
The feature 'hollow' indicates that the spear is hollow 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 piece looks like two spears fused together, forming a single asparagus. 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 which you can see when turning the spear around. Unfortunately, this cannot be done when only looking at images from the side. The feature hollow sometimes even occurs without showing a clear line or obvious bulge at its center. Therefore, there is a risk of wrong classification.

Blooming
The asparagus piece is sorted as 'blooming' when the head part forms a recognizable flower.
In a blooming asparagus, a jagged pattern (slightly resembling a crown) can be seen. Also, the tip of the head part can be an indicator if it resembles a pointed cap and the petals are visible. When a spear is in full bloom, it is clearly observable. However, the distinction between a spear with clear-cut but closed petals and a spear which just started to develop a flower can be quite difficult.
The feature was sorted less strict in uncertain cases where the flower is not clearly distinguishable. One argument is that the flower does not develop further after the sorting process (e.g., as it happens with the feature 'violet'). Another reason for a less conservative approach regarding the feature 'blooming' was to lessen any agreement errors in between the manual sorters. It was decided to sort a spear without a rather clear flower as not blooming.

Rust on body
If a spear has rust on its body, it is visible as a dark brown colour.
It often starts at the tips of the leaves or at the bottom part. In severe cases it spreads over the whole asparagus piece. The colour is not light brown but of a dark shade. The colour is not to be confused with bruises, pressure marks, or a slightly yellow complexion (which can occur in a mature asparagus) which are of no further concern to us. The feature was sorted more strictly to facilitate the decision process during the manual sorting. Therefore, 'rust' was set to be present even when only the tip of a leaf showed a dark spot. Other brownish bruises were not classified as rust.
We decided to split the feature 'rust' into the sub-features 'rust on body' and 'rust on head'. In the case of rust being only on the body, it might still be removable by peeling. If there is rust on, or very close to the head, however, it is not removable.

Rust on head
If there is a dark spot on or close to the head region recognizable as rust, it is captured by this feature. The head part is usually distinct in shape and colour from the body part of the asparagus.
In principle, the same guidelines applying to 'rust on body' can be transferred here, with an explicit focus on the top part (until around 1 cm below the collar) of the asparagus. Rust at the head can be confused with shadow caused by the petals and

the luminance of the sorting machine. Also, it might be that the head is actually violet and not rusty. As rust at the top part of the spear cannot be removed without damaging the head, it is more decisive for the later categorization into a price class, if the head has rust.

Bent
A piece is categorized as bent, if the shape of the asparagus is curved and not straight. Further, a spear counts as bent whenever it changes the growing direction - that is, it resembles an S curve. If it is only slightly round but can otherwise be thought of as straight and fitting next to other straight pieces without standing out, it is sorted as not being bent. If the spear looks close to the same on all three pictures, it might indicate that it is heavily bent and therefore cannot be turned on the machine's conveyor belt.
Like the feature 'blooming', though, the feature bent has a broader range of shapes where the piece is not obviously deformed but also not completely straight. In an indecisive case, it was sorted less conservative. exception holds for S-shaped pieces, which always count as bent.

Violet
The feature 'violet' indicates whether an asparagus piece is of violet colour. The shift of colour 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. As the violet complexion arises because the asparagus came into contact with sunlight, it explains why the violet colour is usually present at the top half . However, the piece can also be violet on the whole body.
Here, it is crucial to sort thoroughly because the spear will darken further after the sorting. Thus, even a slightly pink spear was sorted as being violet.

Thickness and length
The thickness and the length were features we did not consider recognizable by view alone (except for extreme cases). Therefore, both features were measured automatically with scripts written and implemented into the manual sorting app. The division into different categories of thickness can be inferred by the overall thickness of the spear.

Not classifiable
Whenever an image was damaged, the spear was unrecognizable, the head part of the spear was severed, two separate spears were present on one picture, the spear was cut off by the image, or other unusual circumstances on the image occured, it fell into the category of not being classifiable. The image was not further checked for its features, i.e. as bent, violet, etc., but only marked as being not classifiable.

3.4.2 Sorting outcome

In this section, the process and the results of the manual sorting are presented. During the overall sorting period no major problems occurred that led to any breaks or even the abortion of the labelling process.

Whenever an image was manually sorted in the application, the chosen labels were saved in a csv-file. As csv files are plain text files, all stored values are strings arranged in a table. The first line of the table serves as the heading, attributing each entry in the first column to a feature. As seen in Figure 3.2, the features are noted

in the first column, with the first entry being the image ID. Every feature is separated by a semicolon and can be of value 0.0, 1.0, or empty. Hence, whenever a feature was present in an image, the value was set to 1.0, while, if the feature was not present, it was marked as 0.0. However, if there was no choice in which to decide if a feature is present, no value was given to the feature. Such occurred for the case that an image was labelled as not classifiable, where all other manually selectable labels are left empty. The image path to every of the three images (a, b, and c) for one spear was also saved in the label file in the first, second and third column, respectively.
After the labelling process, the single csv-files with the labels were merged into one large combined_new.csv file which was later used for the classification of the data with the different computer-vision approaches. It can be found in the study project's Github repository under XX.

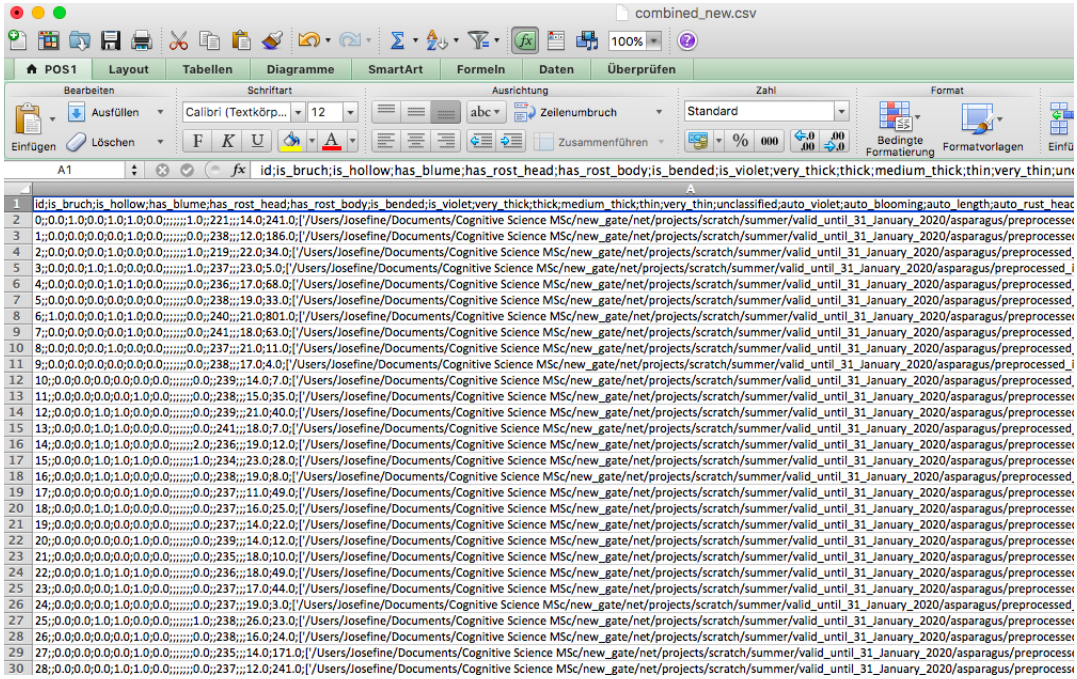


FIGURE 3.2: The feature labels extracted by the manual sorting process were saved in a csv file. This image shows the beginning of combined_new.csv in which all label files were later combined.

The manual sorting 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 from the same perspective. A session of sorting 500 images took around two to four hours. One minor factor sometimes influencing the time spent for sorting were difficulties with the external connection to the IKW store. Another factor was, of course, the large number of pictures, with some spears obviously incorporating certain features, while others needed more careful examination. The average time spent for sorting one piece (i.e. noting all features present) was around 27 - 48 seconds per spear.
All in all, 13319 triples of images were labelled for their features. There is a large variation in the presence of features per image because some features were occurring more often than others. Of the acquired 13319 images, the individual features are represented as follows: 3.5% fractured asparagus pieces, 3.3% hollow, 12.9%

blooming, 14.7% rust on head, 45.5% rust on body, 40% bent, 7.9% violet, and 2.1% not classifiable images. Further categories that were not manually assessed but calculated from the automatic thickness detection included 4% very thick (i.e., above 26 mm), 29.1% thick (20 - 26 mm), 18,7% medium (18 - 20 mm), 17.9% thin (16 - 18 mm), 30.3% very thin (below 16 mm). All images that were not classifiable are excluded from the calculation of the thickness in the stated numbers. As can be seen, many features are only sparsely present in the data. This poses an imbalance that might be relevant for later classification tasks and the usage of the dataset.
Further, every member of the group participated in the sorting but not everybody sorted the same number of images. Due to this circumstance, the sorting preferences of certain members are more present in the data than of others. The manual sorting was stopped when the amount of classified data was judged to be enough by the group.

As different people were sorting the asparagus and because the decision boundary for sorting a spear is flexible to the human eye, a measurement was needed that explores how well the members agreed in the labelling process. Thus, at the beginning and at the end of the manual labelling, images from pre-labelled classes were taken and scrutinized for their features. This was done to train the labellers but also to have a test set for the agreement measure. Every category was therefore sorted by two labellers separately and their agreement was compared. More details about the method that was chosen to assess the group's sorting accuracy can be found in the following sections.

3.4.3 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, were used. The first was Cohen's Kappa. It was chosen, as it is seen as a more robust measure than a simple agreement percentage, such as a measure of true positive and true negatives, which was traditionally used for those cases (Cohen, 1960). It is more robust, as the rate of agreement occurring by chance is included in the calculation. This method is applicable to compare the rating of two raters on a classification problem. This measure of agreement always lies between -1.0 and 1.0, inclusively. The higher the Kappa value, the higher the agreement. Values around 0 indicate no agreement and negative values indicate negative agreement, so to say 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. Everything below 0.4 is interpreted as not acceptable (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/).
Another statistical method used to measure agreement is the F1 score. The F1 Score is used for the evaluation of binary classification. The F1 score relies both on the precision, as well as the recall of a test's accuracy. A F1 score value lies between 0 and 1, the higher the F1 score, the higher the agreement.

Lastly, we looked at the accuracy measure. For a normalized accuracy score, the values lie between 0 and 1, and the best performance is 1. This measure returns the fraction of correctly classified samples. It is a less robust measure than Cohen's Kappa score.

3.4.4 Reliability

In order to evaluate the degree of agreement of our data, we made agreement measures at two points in time (for our API documentation see: https://asparagus.readthedocs.io/en/latest/api/measure_agreement.html). The first time, six different annotators labelled images out of each asparagus group (13 groups). We ensured that always two different annotators labelled the same set of images. Results were not as good as hoped for. 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. It seemed untypical to us, that the agreement scores were so low, even though the raters give the same label to many of the asparagus pieces. This is a known problem (Powers, 2012) (Sim and Wright, 2005) (Feinstein and Cicchetti, 1990)(https://www.sheffield.ac.uk/polopoly_fs/1.404095!/file/RSS_Poster_Laura_Flight_Final.pdf).

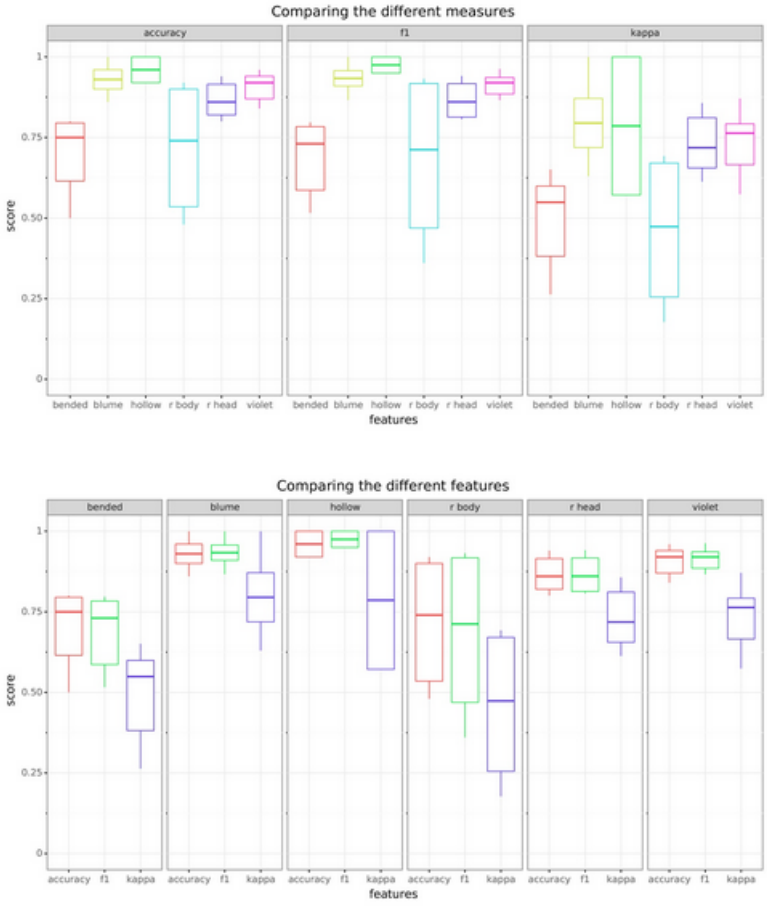


FIGURE 3.3: All scores are aggregated scores over all annotator pairs. The first plot shows the results agreement measure-wise, the second plot feature-wise.

One reason for our results could be that we compared the agreement group-wise. This means that we already knew in advance that for certain features, we have almost only 1s or almost only 0s for all images of one group. For Kappa scores, if the distribution of 0s and 1s is not balanced, disagreement of the underrepresented value is punished more heavily. Therefore, we decided to repeat our agreement

measure and do it this time feature-wise on non-labeled images, so that the annotators could not anticipate a specific group label. In order to better understand the reliability of our data, we also decided to look at the accuracy score and the F1 score, additionally. Before we did so, we labeled another 50 images all together, clarified classification boundaries again and discussed unclear images.

The second time, 50 images were labeled by 4 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 also between annotator pairs. However, the results are much better than the first time. The highest aggregated kappa score over all annotator pairs is reached for the feature flower (0.79), then hollow (0.79), 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 IQR is quite small (), whereas the IQR for hollow, bent and rusty body is much larger ().

The agreement scores accuracy and F1 yielded very similar results. Results were slightly better than the Kappa scores, in total and for each feature. The highest accuracy score is reached for the feature hollow (0.96), then flower (0.93), 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, one can therefore say that the agreement scores indicate moderate up to substantial agreement. Agreement is highest for the features flower, violet and rusty head.

3.5 The asparagus dataset

TODO: Any information on the data we worked with for our approaches.

3.5.1 Different datasets

TODO

What exists:

- Raw data (reading functions)
- Numpy Array
- TFRecord
- Tf.data.Dataset
- Other dataformats: https://github.com/tensorflow/io
- Publication -> Tf.data.dataset / tf.dataset

  Best practice:

  - Theoretical

  Used on our data:

  - Difficulties
    - Mulitlabel

    * Semisupervised dataset – not all labelled
    * Misunderstandings
- Results Discussion

Summary

- Theoretical usage on our data
- Implementation in our repo
- How to use in future work

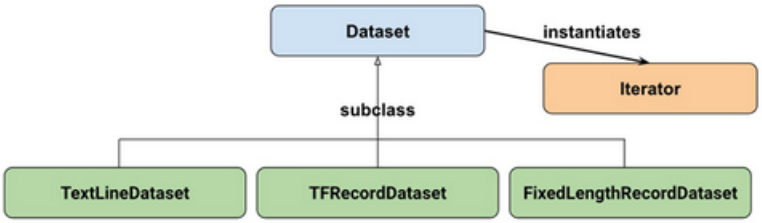


FIGURE 3.4: The tf.dataset.

```
# Use interleave() and prefetch() to read many files concurrently.
files = Dataset.list_files("*.tfrecord")
dataset = files.interleave(lambda x: TFRecordDataset(x).prefetch(100),
                           cycle_length=8)

# Use num_parallel_calls to parallelize map().
dataset = dataset.map(lambda record: tf.parse_single_example(record, …),
                      num_parallel_calls=64)

dataset = dataset.shuffle(10000)
dataset = dataset.repeat(100)
dataset = dataset.batch(128)

# Use prefetch() to overlap the producer and consumer.
dataset = dataset.prefetch(1)
```

FIGURE 3.5: ??

Additionally, we created a single numpy dataset for the labelled images that could be easily stored and loaded for training. Firstly, all the identification numbers from the labelfiles were selected and the corresponding images were found and copied to a separate folder. This folder contains all the images for which we have labels. Secondly, the images were grouped by their identification number, which means the three perspectives of each asparagus spear were combined. We decided to downsample the images for creating this dataset to facilitate the training process and reduce memory consumption. Initially, each image was downsampled by a factor of six, that means every 6th pixel was used in the reduced image, but this factor can be easily changed and a new dataset will be created for the individual needs of each approach. After that they were transformed to a numpy array and concatenated to a single file. The images were either concatenated horizontally, so that in the resulting image the three perspectives appear to lay next to each other, or vertically, so that the images are stacked one after another. Finally, all the concatenated arrays were combined to a single four dimensional dataset. The first dimension depicts the number of labelled asparagus spears, the second and third dimension represent the height and the width of the images, respectively. Further, the fourth dimension represents the depth, which is three for

the horizontally concatenated images, namely the RGB values, and nine for the vertically stacked images.

3.5.2 Challenges

Problems and challenges during the creation of the datasets.

- What were the challenges in creating a general dataset?
- What were challenges in general?
- How well could we work with the datasets?
- What was used as training data, validation data, and test data?

Chapter 4

Classification

Given the structure of our dataset, namely image data with a sub dataset with corresponding class labels, and a sub dataset 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, multi-class and multi-label. Whereas a binary classification only distinguishes between two different classes and therefore classifies an image into one of the two classes, a multi-class classification distinguishes between multiple exclusive classes. A multi-label classification also works for multiple classes. However, a single image can belong to none, one, several or all of the classes. Those classes can be seen as a feature vector for each image. Each feature can be present or not, independently of the other features (vielleicht hier zitieren, seite muss noch rausgesucht werden) (Har-Peled, Roth, and Zimak, 2003).

Each of those classification methods comes with certain advantages and implications. There exist many classification methods which have been developed for binary classification problems, but less methods are suited for multi-label or multi-class classification. Therefore, the latter often work as a combination of binary classifiers. Moreover, multi-class and multi-label classification have the difficulty of sparser labels.

Binary classification can be used to decide whether a certain feature is present at one certain asparagus piece, or not. This is helpful for a first inspection of the data, but does not enable a full classification of one image into one of 13 classes, which are currently sorted at the Spargelhof Gut Hosterfeld. Multi-class classification solves this problem. It is fairly easy to apply this classification type on the prelabeled images, but increasingly difficult for the semi-supervised and unsupervised approaches. While it enables a clear identification of class belonging, it also does not enable to train variability within classes. As the class id results from a combination of the presence of certain features, and not others, it is therefore also reasonable to go for a multi-label classification approach.

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. In addition to classical computer vision-based approaches, our study group investigated several neural

network approaches.

During our group work, algorithms of all three different classification types (binary, multiclass and multilabel) as well as of all three learning types (supervised, semi-supervised and unsupervised) were applied for different working steps and different purposes.

In the long run, an integrated model was aimed which predicts all features of a single asparagus piece, and from which an additional class can be inferred. However, as intermediate steps towards that goal, the focus was to optimize models on identifying the presence of single features. Besides that, we only investigated a few multi-label classification tasks.

The following chapter aims to give 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. All algorithms were implemented in Python.

4.1 Supervised learning

TODO: Neural networks created to sort the samples for their features. Transfer to the specific model ideas that we worked on for supervised learning.

4.1.1 Prediction based on feature engineering

Besides approaches that directly use images as an input one may use high level feature engineering to retrieve sparse representations and apply classical machine learning classifiers such as multilayer perceptrons to predict labels. These approaches are comparatively simple, fast to train and only few hyperparameters have to be defined. As compared to deep learning this means that one has a large degree of control and if the learning task is simple, low accuracies are arguably rather due to incongruencies or missing information in the (sparse representation) of the data then a result of issues in the design and training parameters of the network. This classical machine learning approach was applied to predict features based on color and partial angles of asparagus pieces.

Violet and rust prediction based on color histograms
The initial approach of measuring the feature “violet” that is based on distribution of sufficiently intense color hues in the violet range faces at least two drawbacks: First it requires to define two thresholds and second the impression of a violet asparagus piece could potentially be affected by the mix of colors (combinations of colors) that are potentially outside the violet range or are too pale to be considered (see XXX). The same holds for the feature “rust”. Hence in a second approach histograms were computed for foreground pixels after transforming the images to palette images with 256 color hues (see XXX). The resulting representation is arguably a sparse descriptor that allows to predict color features using explicitly defined rules or trainable machine learning models.

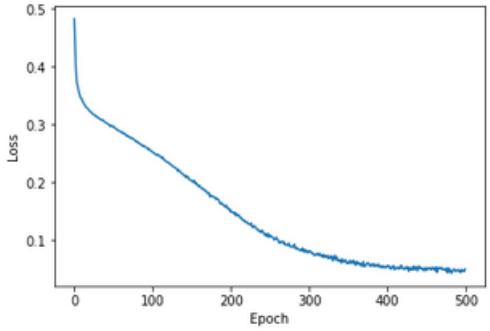


FIGURE 4.1: ??

	False positive	False negative	True positive	True negative	Sensitivity	Specificity
is_violet	0.04	0.03	0.05	0.88	0.62	0.96
has_rost_body	0.19	0.14	0.33	0.34	0.71	0.65

FIGURE 4.2: ??

Curvature prediction based on partial angles
Although the accuracies are far from perfect the results appear to be promising. The hit rate for curvature detection is high: 82% of all bended pieces were identified as such. In comparison, this holds for 71% of the pieces affected by rust and 62% of the violet pieces. In contrast, almost all pieces that are identified not to be violet are labeled accordingly (96% specificity) whereas the specificity for rust (65%) and curvature (67%) is lower. 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. Possibly this reflects that rather small brown spots were considered rust by some coders.

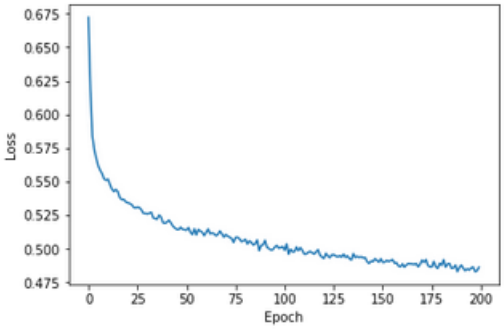


FIGURE 4.3: ??

Considering the low agreement in labeling, higher values for the specificity and sensitivity of the classifier were not expected. A likely explanation is that the model generalizes deviating understandings and incongruencies we had when attributing labels and affected the reliability of the data. Arguably only little information was discarded by computing the sparse representations that served as an input. Information about irregularities in the outline that a center line does not reflect but possibly contribute to the perception of

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

FIGURE 4.4: ??

curvature and the spatial distribution of colored pixels might contain additional information regarding rust and violet-detection. However, the major criteria are captured. As MLPs have little hyperparameters that are suitable for non-linear mappings and as the task of mapping high level features to human estimates appears to be rather simple, one could argue that there is little potential to improve the predictive quality using other techniques. By introducing a bias, the sensitivity (true positive rate) of the classifier can be adjusted at the cost of more false positives. 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 can be 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. The resulting behavior is reflected in the receiver operating characteristic that is calculated using different biases.

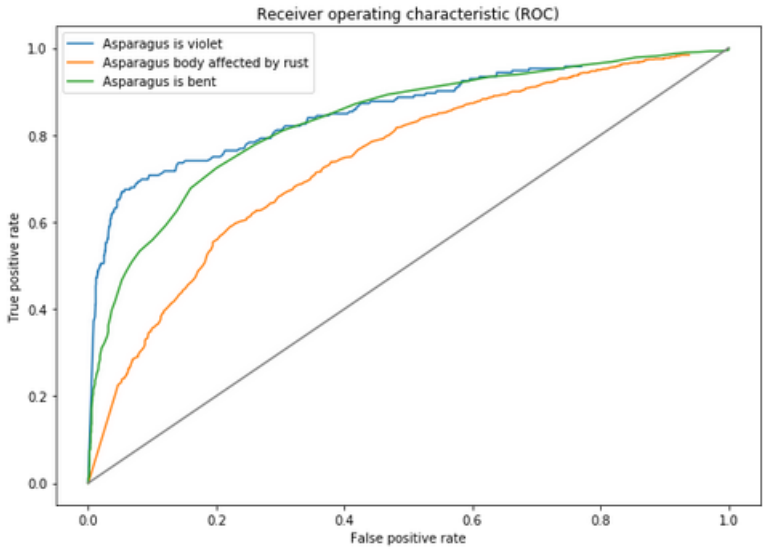


FIGURE 4.5: ??

4.1.2 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 convolutional neural network on depictions of the respective region of the asparagus. Therefore, a dataset that consists only of images of the head area was used. Images of all three perspectives were 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 some angles. The depiction below shows one sample of rust affected asparagus heads.

FIGURE 4.6: ??

A simple feedforward convolutional network was trained on the images. The features "flowering head" and "rust affected head" were chosen as target categories. The network comprises the input layer, ?? convolutional layers with kernel size ?? and ??, a fully connected layer with ?? neurons as well as the output layer. For the final layer the sigmoid activation function was 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 indicated convergence after 40 epochs.

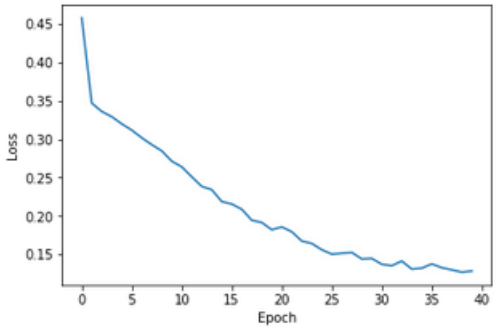


FIGURE 4.7: ??

The results for both features showed to be highly specific. In contrast the sensitivity is rather low. Only 55% of the asparagus pieces labeled as "flowering head" were identified as such whereas the true positive rate is only 19% for "rust head". Given the low labeling agreement for these criteria these mediocre results are not surprising.

	False positive	False negative	True positive	True negative	Sensitivity	Specificity
has_blume	0.04	0.06	0.08	0.82	0.55	0.95
has_rost_head	0.02	0.13	0.03	0.83	0.19	0.98

FIGURE 4.8: ??

The ROC curve indicates how the classifiers respond to the introduction of

a bias and shows the overall prediction quality. The area under the curve is small for the feature "rost head". Beside incongruencies in the labels this is possibly due the choice of the head region. It might be the case that brown spots in regions other than the cropped head were considered as an indicator for a rusty head when attributing labels. Improvements by increasing the cropped head region appear to be possible.

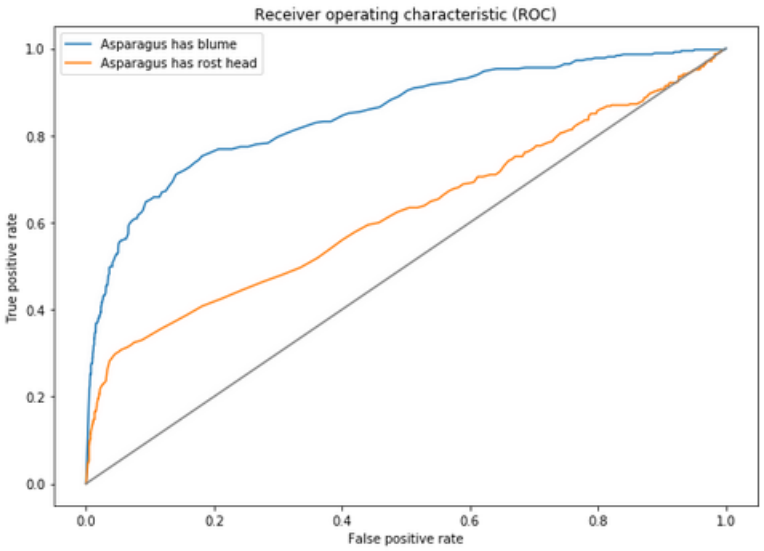


FIGURE 4.9: ??

4.1.3 Single-label classification

TODO

One supervised approach used for training a model on image data is a convolutional neural network (CNN). The approach was tested for our data with the application of a single-label classification CNN. A CNN uses trainable filtering kernels to extract features from an image and projects these features on feature maps (source XX). Based on these maps, the label is predicted by a standard multi-layer perceptron.

BACKGROUND
→ TODO

STRUCTURE
The model architecture is roughly based on AlexNet but was strongly simplified to the level of variability and complexity of the underlying data. 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. This input is trained on a set of binary labels containing information on whether the respective feature 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 0 and 1. The rounded integer values of this output give a prediction of the apparent label. For the training of the model, the Adam optimizer was used because of its general acceptance as the state of the art optimizer for backpropagation (source XX). As a loss function binary crossentropy was used as it promised good results for single-label classification tasks (source XX).

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. Hence, the idea was to start with the simplest form of a CNN and then gradually increase the complexity of the net. While AlexNet provides a good baseline for an image classification network, its architecture is still unnecessarily complex for the given task (source XX). 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 hyper-parameters were implemented and compared according to their performance. During this process, the data was split between training and validation data in order to have a reasonable overview on the possible test performance and to prevent direct overfitting. The course of the hyper-parameters is explained in the following.

Batch size
The batch size was initiated comparatively low with 64 samples per batch but soon increased to a value of 512 samples per batch. The large batch size was implemented in order to guarantee the convergence of the training data, since smaller batch sizes resulted in jumping gradients which were not able to converge into any minimum (source XX).

Learning rate
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 (source XX)) was soon found to be too large to guarantee convergence of the algorithm. Thus, the learning rate was gradually decreased and found to be most effective in the range of 0.000001 (= 1e-6) to 0.00000001 (= 1e-8). Learning rates as small as 0.00001 (= 1e-5) still seemed too large for training the net. In addition, a gradually decreasing learning rate was implemented in order to make the training more effective (source XX).
Layer/Deepness
Starting with the smallest layer size possible, more layers were added based on the amount of theoretical background (source XX), which includes filtering steps ... For example, for the feature fractured one layer for edge detection should have sufficed or for the feature violet a layer for colour detection. For other features, such as bent, at least two convolutional layers were expected to be more helpful (source XX). During the training process, it was settled to a model with two convolutional layers, one max pooling layer and one hidden dense layer.

Layer width/Number of kernels
A small number of kernels was thought to be enough compared to the kernel size of AlexNet (why? source XX). The kernel size resulted in 32 5x5 kernels

for the first convolutional layer and 64 3x3 kernels for the second convolutional layer. These sizes were assumed to be sufficient, especially since increasing the number of kernels did not lead to any better results.

Exploding gradients / Batch normalization
Both convolutional layers were built with batch normalization nearly from the start of implementation of the network. To guarantee that exploding gradients posed no problem, the gradients were inspected visually and the results gave no reason for assuming prolems with exploding gradients.

Epochs / steps
In most cases, 300 training steps were performed for the training.

Balancing the train data
A next step was to weight the loss function because of the largely unbalanced data. This ended with no better results, however, as the model still tended to make an unbalanced prediction by classifying all values as 0.0. Another idea was to reduce the dataset to make the number of images with the regarding feature even to the number of images where it is not present. This translates to throwing away valuable data, which can otherwise provide information about negative cases to support the model in it's training. Thus, it was decided to keep all data images and instead balance the data by multiplying the minority of samples to match the number of negative samples. As there was no feature positively exceeding a presence of 50% in the data, only positive labels were multiplied.

Data augmentation
To improve training performance, some kinds of data augmentation were implemented. First, the images were flipped horizontally. These seemed to be a valid enhancement to the training data since it resulted in similar images to the original data. Additionally, small changes in the image angle (up to 5°) were tested, however, this type of data augmentation was neither effective nor convenient to be computed on the grid since an additional library would have been necessary to install.

RESULTS
→ TODO

DISCUSSION
→ TODO

4.1.4 Multi-label classification

Building on the standard single-label classification we were further interested whether it would be possible to build a model that can predict several feature labels at the same time. A multi-label classification model hereby gets an image as the input and learns to predict the presence or absence of several classification objects, in our case the feature labels.
For our model we decided to use a small convolutional neural network as

described below and the features that we labeled by hand. Each of the six classes (hollow, flower, rust head, rust body, bended and violet) is encoded by a binary output in the target vector, indicating whether the asparagus exhibits the feature in question or not.

BACKGROUND
Multi-label classification is a useful tool that can be used in classification problems in which several classes can be assigned to a single input. In contrast to a multi-class 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 multi-class 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 also possible that all or none of the classes are present.
One could think of the multi-label classification task as consisting of different sub-tasks. Therefore, some multi-label classification problems can be transformed. There are two eminent ways to do so, either the problems are transformed into multiple binary classification tasks or into one multi-class classification task.
In the first approach, 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 or not and in many cases they are.
Therefore, we decided to not only use single-class classification as described in chapter 4.1.3 but to explore the possibilities of multi-label classification.

In the second approach, each possible combination of features is interpreted as one class. For a classification problem with 6 features that means there are 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 will be 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 multi-class problem.

Inspiration for the model gave a blogpost[1] which aims to classify images of the MNIST fashion dataset in the context of multi-label, rather than multi-class classification. The author altered the dataset in such a way that each input image contains four randomly selected items from the MNIST fashion dataset. The model then learns to predict which classes are present in the image. There is no restriction on the random selection, therefore the items can be all from the same class, from four different classes or anything in between. The target vector has 10 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. In both cases, the model is supposed to predict the presence or absence of different fashion items or labeled features, respectively, and the number of classes is similar, it is 10 in the fashion example and 6 for our model.

Secondly, the model uses a dataset of similar size, 9000 images were used for training and 1000 for validation, while we were training on 10800 images and validating on 1200. Despite the rather small dataset in comparison to many other machine learning problems, good results with an accuracy of 95-96% were reached. This leads us to think, it might be a model with a good complexity for our problem too, as it is complex enough to yield good results, but not too complex for the medium-sized dataset.

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

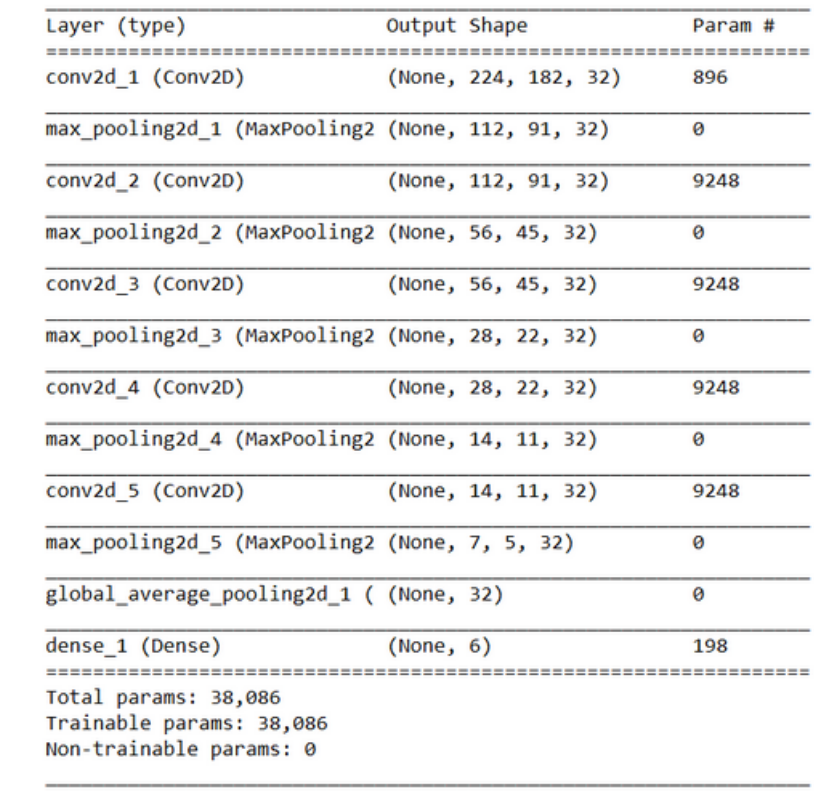
In contrast to multi-class classification models, where usually a softmax activation function is used in the last layer together with a categorical cross entropy loss, the multi-label classification model uses a sigmoid activation function and a binary cross entropy loss.

As the input of the model a concatenation of the three perspectives of each asparagus piece were used in order to maximize the information the model gets about each asparagus. This yields input images that look like the three asparagus pieces are laying side by side. Further the images were downscaled by a factor of 6 to facilitate training. The output, or target, of the model is a vector of length six in which each position encodes one of the six hand-labeled features (hollow, flower, rust head, rust body, bended and violet). Each feature can either be applicable to the input or not, which leads to a "1" or "0" in the target vector, respectively.

Several loss functions were tested to improve the models performance. For example, the hamming loss, which uses the fraction of the wrong labels to the total number of labels. Additionally to the in-built loss functions from keras, a custom loss function was implemented, that penalizes falsely classifying ones as zeros more than falsely classifying zeros as ones. 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 zeros than ones in many classes. Hence, the model can get a decent accuracy by labeling most features as zero. By penalizing this error more, we intended to counteract the unbalanced dataset. But at the end, the binary cross entropy loss remained 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 L1 or L2 regularization, respectively, to all five of the convolutional layers. Hereby, a kernel regularization was implemented with a value of 0.01.

L1 and L2 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 L1 regularization reduces the coefficient of irrelevant features to zero, which means they are removed completely. Hence, L1 regularization allows for sparse models and can be seen as a selection mechanism for features. As the inputs to our 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, it appears to be a good idea to reduce the number of features taken into account in the early layers. L2 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.



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

FIGURE 4.10: ??

RESULTS
As one can see in the figures XX-YY, all the different approaches explained above show a similar behaviour 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, L1 and L2 regularization are used to prevent overfitting, but in our case it did not improve the results, as one can see in figure XX.
When looking at the true positive rates, also called sensitivity, and the true negative rates or specificity, one can see 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 epochs, after which the change progresses slowly in the same direction with no greater disturbances. One exception is the model trained with the L2 loss, which does not show a large change in either of the rates.

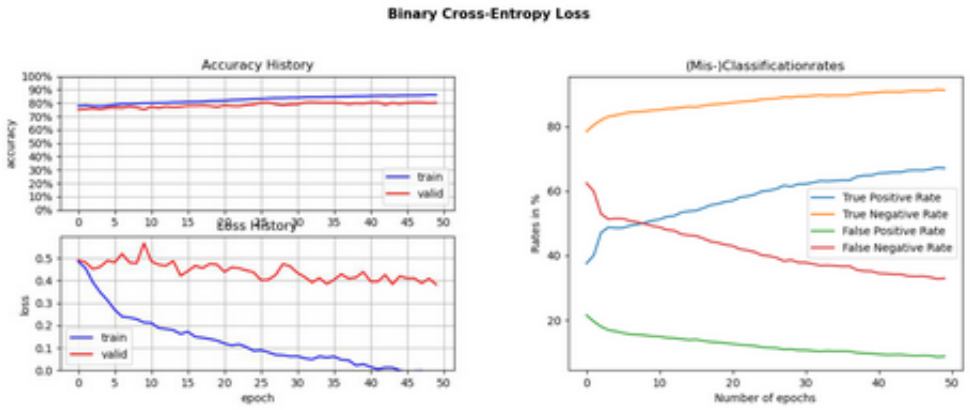


FIGURE 4.11: ??

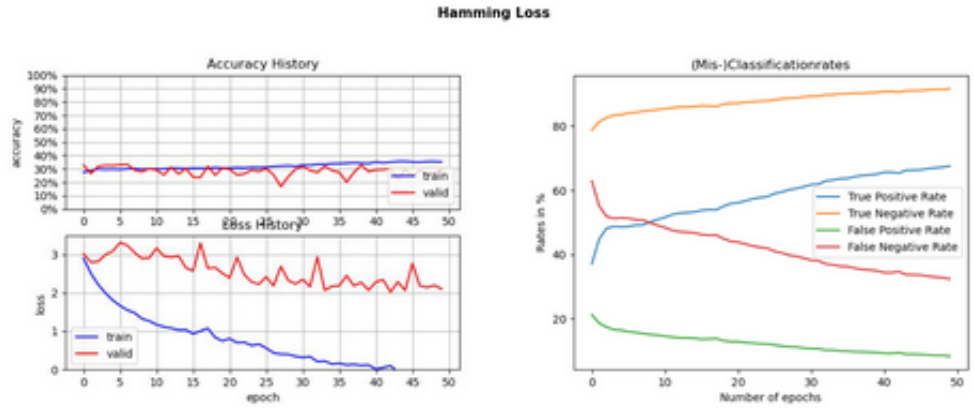


FIGURE 4.12: ??

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 costum loss. Its values start at 75%, while they start at around 30% for the other two loss functions. The behaviour of the curves and the (mis-)classification rates, however, are very similar in all three approaches. The true negative rates start off very high with values around 80% and increase further during the training. The true positive values start off lower, at around 40% to 45%, and increase rapidly in the first few epochs, after which the rates proceed to increase but with a narrower slope. As stated above, the false negative and false positive rates show the same slope but in the opposite direction.

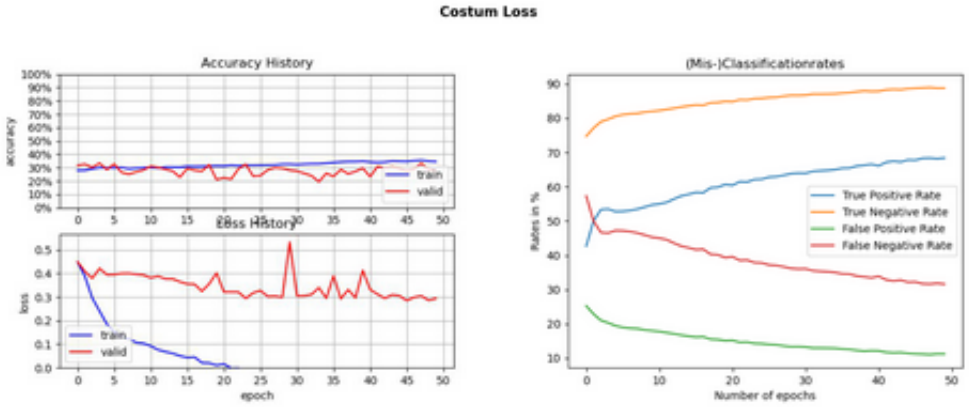


FIGURE 4.13: ??

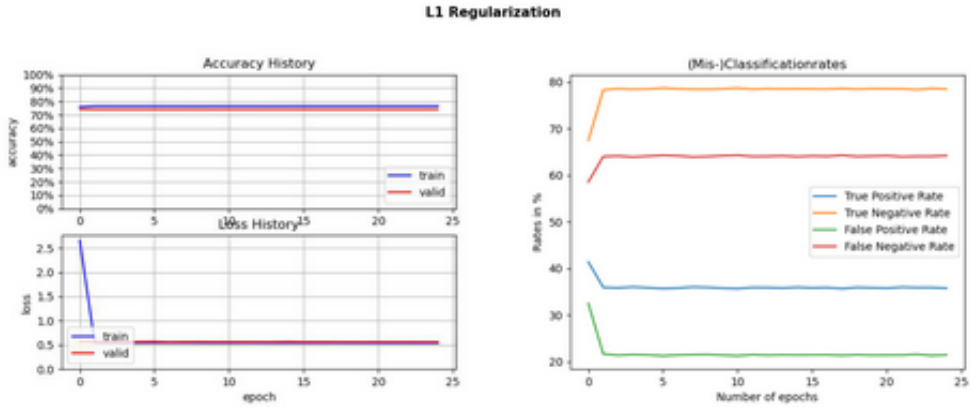


FIGURE 4.14: ??

The accuracy values of the models that were trained with L1 or L2 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 after that. While the (mis)-classification rates of the model trained with L1 regularization behave similarly to the ones trained with no regularization, the rates of the model trained with L2 regularization show a smaller increase and lack the fast change in the first epochs.

DISCUSSION

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. But 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 one can see in the result section neither L1 nor L2 regularization alone were able to prevent overfitting. Another common practice that could 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

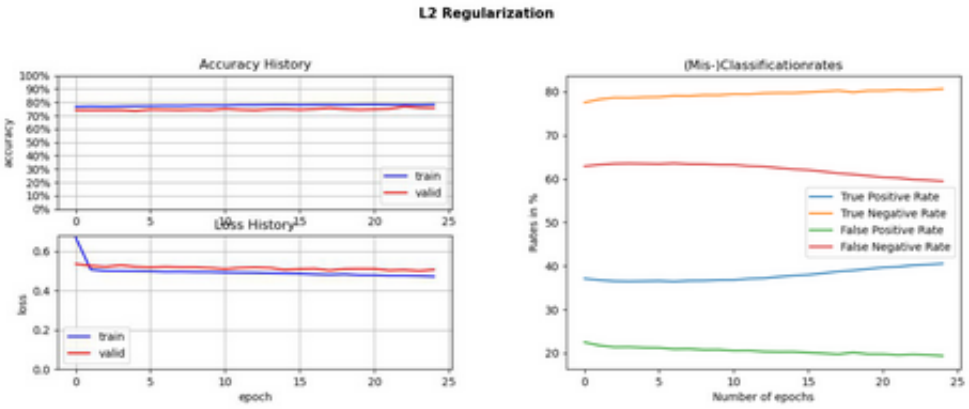


FIGURE 4.15: ??

a small number of nodes but distribute the information between all nodes available. Hence, the coefficients remain smaller. Another way 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 (QUELLE). 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, in fact, perform better than the other two models, because 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 true positive rates are a lot lower than the true negative rates. A reason for that might be that there are fewer positive values in the dataset and they are, thereby, more difficult to learn. The model might have learned that, if unsure, a zero is the more likely guess.

The L1 and L2 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 was 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 and after this problem is solved, it should be tested whether additional changes can improve the performance of

the model further.

4.1.5 From feature to label

Additionally to the feature classification tasks, we used supervised learning to predict the 13 commercial asparagus classes from the labelled features. As described in chapter XX we decided to label features rather than classes, therefore all the approaches described also learned to predict those features from the input images. To get to the final classes from the features is only a small step in theory and if we would be able to solve the problem of the feature classification task perfectly, the distribution in the commercial classes should not be problematic. Nonetheless, it is interesting to see how easily this translation can be done and especially how well the categories match with the ground truth that we established at the asparagus farm.

In order to build a ground truth, we collected images of the 13 classes of asparagus that were ready to be sold as the specific class. That means, the asparagus was not only sorted by the machine but resorted by the experienced staff at the asparagus farm and the images were collected by running those re-sorted spears through the machine a second time. We then labelled these images with the label app and the labelled features were used as the input for the following supervised learning approach, while the known classes were used as the output or target. Approximately 200 images per category were used.

Two different approaches were implemented and tested to predict the classes from the features. The first one is a simple model with X fully connected layers.
TODO: Model structure beschreiben

The second approach is a random forest, with X number of trees, X number of features per branch and a depth of X for each tree.
TODO: details/structure beschreiben

Random forests are a popular machine learning approach, because they reach good results in both regression and classification tasks. Further, they are based on decision trees, which are frequently used because of their intuitive interpretation for humans. Although decision trees themselves are powerful tools, they have the major drawback of being prone to overfitting. Random forests aim to avoid this problem, while still using the advantages of decision trees, namely that they are flexible, easy to use and fast to train. They do so by training several decision trees on different random parts of the data, after which a majority voting decides on the final output or class. An additional trick that can be used is a random selection of features to be used at each branch. This reduces the influence of highly correlated features and thereby makes the random forest more robust.

With this second approach, a score of 0.76 was reached, which is already very high compared to what the accuracy of 0.08 by random chance guessing would be. One way to increase the accuracy even further would be to increase the agreement on the labelled features. By reducing the amount of

noise generated by ambigues labelling the performance might be enhanced.

When comparing the results of the two approaches one can see...

- unified interface (labels + classes, scikit-learn keras model compatibility)
- metrics: look at classification report in streamlit multiple_models app
- simple keras model with fully connected layers etc., reaches an evaluation of 2.2
- confusion matrices for both as figures, compare the different results
- app: dataframe, diagrams, able to see the images and corresponding hand-labeled features and predicted categories

QUELLE (Friedman, Hastie, and Tibshirani, 2001)

4.2 Unsupervised learning

Unsupervised learning are all kinds of machine learning algorithms which are not supervised. More specifically, they work without a known goal, a reward system or prior training, and find a structure within the data, or some form of clustering of data points. In supervised approaches, the model is given both the input and the labels. In unsupervised learning approaches, the model is only given the input data. What this means is that unsupervised learning works without training samples, and without labeled data. One goal of unsupervised learning algorithms is tao find hidden structures or patterns in the data, without a given label.

Dimension reduction algorithms and clustering algorithms have been identified as the two main classes of unsupervised machine learning algorithms which are used in unsupervised image categorization (Olaode, Naghdy, and Todd, 2014).

Multivariate datasets are generally also high dimensional. However, it is common that some parts of that variable space are more filled with data points than others. Large parts of high dimensional variable space are not used. In order to recognize a structure or pattern in the data, it is often necessary to reduce the number of dimensions. For this, both linear- as well as 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 would be Kernel PCA, (Scholkopf et al. – im oben genannten paper), Isometric Feature Mapping (ISOMAP), Local Linear Embedding, and Local Multi-Dimensional Scaling (Local MDS). For the current work, the linear dimension reduction algorithm PCA was chosen.

4.2.1 Principal Component Analysis

PCA was chosen, because it is one of the standard methods of unsupervised learning in machine learning, to analyze data. Moreover, it is a linear, non-parametric method and widespread application to extract relevant info from

high dimensional datasets. The hope is to reduce the complexity of the data, by only a minor loss of information (Shlens - A tutorial on PCA). Besides being a dimension reduction algorithm, PCA can also be useful to visualize the data, filter noise, extract features, or compress data.

Our initial aim was to reduce the dimension of our dataset for further models. However, as this was performed 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 dataset have a high quality, and instead of only reducing the pixel size, we aimed for reducing the information contained in named depictions by analyzing the principal components in a first step and projecting all relevant images into the lower dimensional space. This information could serve as the input for later machine learning algorithms in a second one. As we decided to use an approach for semi supervised learning that is based on neural networks later we focussed on the question what principal components reveal about the data.

PCA relies on linear algebra. It assumes that large variances are accompanied by important structure. When calculating the covariance matrix of the data, it 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. It further assumes 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 - Dim. Reduction and Cluster Analysis paper p.2). The covariance between each pair of principal components is zero, as they are uncorrelated. Generally, the higher the eigenvalues, the more useful it is for the analysis.

The result of a PCA is a representation of the data in a new lower coordinate system, which depends on the axes of the largest variance. The dimensionality of this lower coordinate system should depend on the size 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 a consensus in the literature, 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, it has to be decided if the PCA should be performed on a black and white image, or if a more complex approach on colorized images is needed. When working on black-and-white images, only one data point per pixel is given, therefore, performing a PCA is less computationally expensive, and also finding structure is easier. However, as we need to be able to recognize violet as well as rust, it was important to us, to be able to differentiate between color nuances, which cannot be represented in a black and white image.

There are at least three different ways on how to perform a PCA. First, it can be performed on images of different classes at the same time – similar as to capture several images of several people in one database, on which a PCA is performed. In our case this would mean that a PCA is applied to a

dataset of several input images of all 13 classes of asparagus images. Second way would be to perform a PCA separately on each group. This way, an "Eigenasparagus" of each group would be calculated, and distances between Eigenasparagus of different groups could be measured. Third PCA can be employed feature-wise. In this case, the dataset would consist of a collection of images with a certain feature present vs a collection of the same size with the feature absent.

We calculated the PCA for black-and-white images, as well as on images without background, and also tried to work on all groups at the same time, group-wise as well as feature-wise. We decided to perform our final PCA on sliced RGB images with background, that are labeled with features by us with the hand-label-app, as this seems to yield the best results. An amount of 400 pictures per feature was considered to later on perform a binary PCA for each feature (either the feature is absent or present).
The 200 pictures where the certain feature is present as well as the 200 pictures where the certain feature is absent are extracted through a function in code/pca_code/feature_ids.py. This function loops over the combinded_new.csv csv-file, where all hand-labelled 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]3 $\times$ 4 img_shape[1]3 $\times$ 4 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, broken, flower, rusty head, rusty body, bent, violet, length and width.

For all these features, a PCA is calculated by first standardizing the matrix pixelwise (dimensions: 13403 $\times$ 4 3463 $\times$ 4 3), calculating the covariance matrix and then extracting the ordered eigenvalues. The principal components are calculated multiplying these eigenvectors with the standardized matrix. The highest 10 eigenvalues were plotted to visually decide where to set the threshold of how many principal components will be further included. The feature space, the principal components and the standardized matrices are saved to later perform a recognition function.

The recognize function is a control function, which performs on unseen images and tries to predict if a feature is absent or present. This function is also performed feature-wise. It reads in a new picture of one asparagus, which is not part of the asparagus database, so not within the 400 images. Then, it searches for this picture the most similar asparagus in the feature matrices (which are 200 pictures with the feature, 200 pictures without).
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 eigen asparagus and the 400 pictures in the feature space, by using the cdist function of the SciPy software. 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 the feature is absent. By comparing this to the information of the single asparagus piece, 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.

All preparation, calculation and analysis for the final PCA was performed in python (version x). All scripts ran in the virtual environment dataSet. The main packages used were cv2, SciPy and Matplotlib.

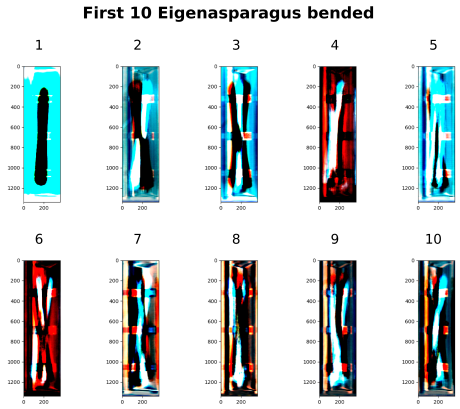


FIGURE 4.16: ??

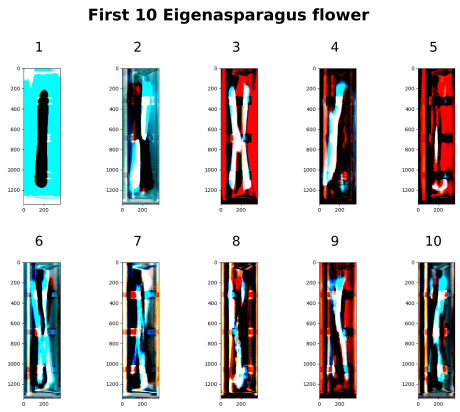


FIGURE 4.17: ??

The final results of our PCA are given feature-wise and were not translated back on the initial 13 groups. The features on which results are given are: hollow, flower, rusty body, bent, width and length. All features are binary. For the first five features, the manually hand-labeled information was taken, for the last two features, the information, which was extracted by the algorithms used in the hand-label app were taken. The decision boundary for the first five therefore depends on our predefined criteria on labeling. The decision boundary for width was narrower or equal to 20 mm or wider than 20 mm, for length shorter or equal to 210mm or longer than 210 mm.

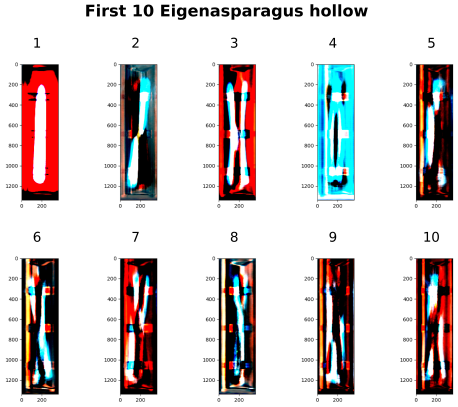


FIGURE 4.18: ??

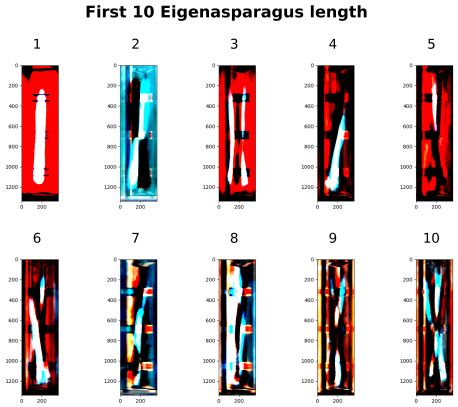


FIGURE 4.19: ??

There are no results for the feature broken, even though it is one of the initial main features, as there were only five labelled pictures for this category. Therefore, it was excluded for further analysis.
For the feature rust head, there was a problem in the calculation. For an unexplainable reason, there were complex values in the calculation of the principal components. Due to time constraints, this problem was not solved, yet. Therefore, plotting of the feature rust head was not possible.

By applying the feature_pca.py file on each feature, the eigenvectors, eigenvalues, and principal components for each feature were computed and stored. In all cases, the first principal component is quite high (between X and X), and drops down rapidly afterwards. The values of the principal components after the 4th principal component are very low (below X).

After inspection of the graph of the first 10 principal components, we decided that only the first four principal components are used for the analysis, as there was either a sharp drop in the slope after the first 4 or to maintain continuity between the different features. Following, the corresponding feature space is

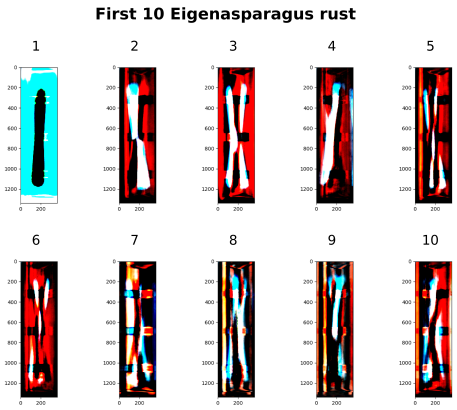


FIGURE 4.20: ??

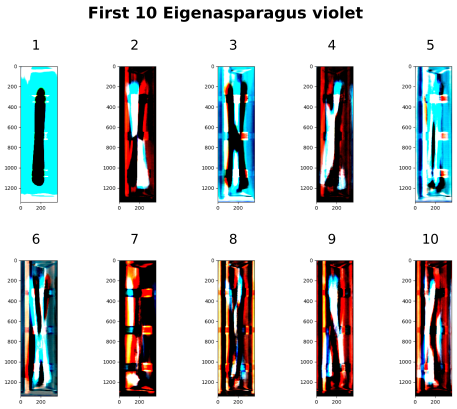


FIGURE 4.21: ??

that space spanned by the first four principal components.

We plotted the four-dimensional data in the feature space as scatterplots in three-dimensional space and the fourth dimension as color, but as it was difficult to interpret and visually understand, we decided to show plots of only the first two dimensions, here.
The following scatterplots show the data of each feature lined up along the axes of the first two principal components of each feature.

For the recognition function, we used the same 10 images, to test each feature. The classification worked best for the features length and hollow (10/10 classified correctly) and then width (8/10 classified correctly). It performed 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).

DISCUSSION
The results we got for the final PCA approach, of calculating the principal

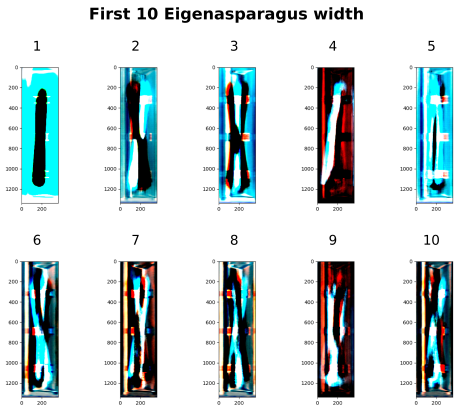


FIGURE 4.22: ??

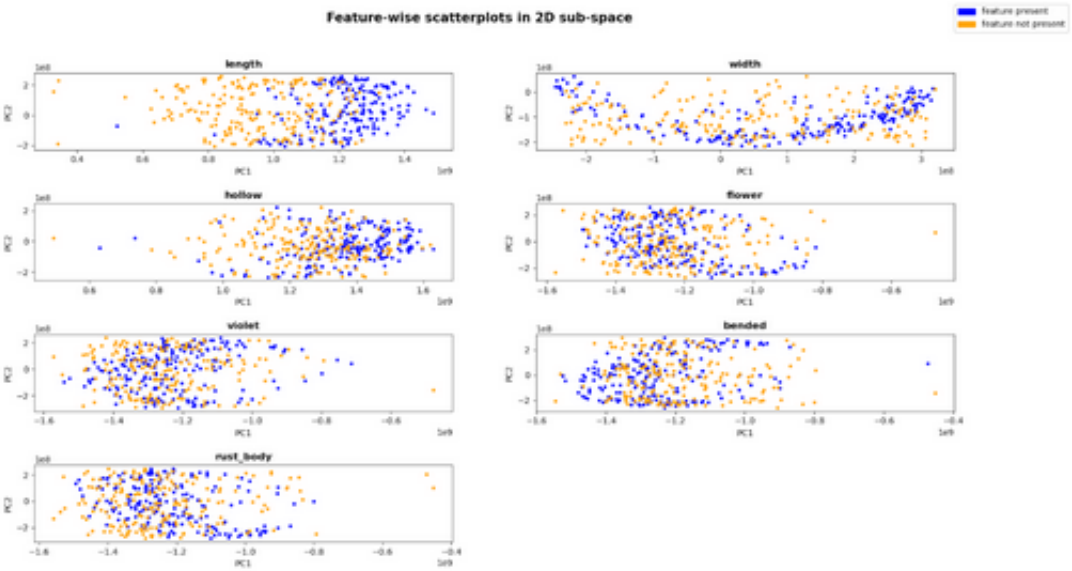


FIGURE 4.23: Plotting the data in the 2D PC space for each feature.

components for each feature separately led to better results, then the first two approaches.

From the images of the first 10 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 piece can be seen. The following images leave a lot of room for interpretation, about what information is contained there.
We performed the PCA on each feature separately, to extract the principal components of each feature. It is interesting to see that the pictures of the different features are all very similar. One reason for this might be that many of the 400 pictures for each feature are overlapping between features. Another reason might be, that even though the images vary between features, the general information of all asparagus images is very similar.

From the results of the recognition function, one can see that there are large differences between the features on 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 also chapter 3.4.3 (agreement measures)), and that the results are due to inconsistencies within the data. One indicator that this is a considerable reason is that the performance of the width and length features, which is information which is not hand-labeled, is very high. Moreover, the poorest results can be observed for the features bent and rust-body. Those are the features, for which the agreement measures showed the largest discrepancies between annotators (see 3.4.3).

Another reason why the results are partly only moderate, is that RGB image data 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 on 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 image data (Olaode, Naghdy, and Todd, 2014).

4.2.2 Autoencoder

Beside PCA, there are further techniques for dimensionality reduction. One alternative machine learning approach that can be employed to deduce sparse representations and automatically extract features by learning from examples are autoencoders. Simple autoencoders, where the decoder and encoder consist of multilayer perceptrons, were already proposed as an alternative to PCA in early days of artificial neural networks 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 that employed convolutional autoencoders in various domains. Examples range from medical images to areal radar images (Chen et al., 2017). These include not only shallow networks but more recently the benefits of deep autoencoders were demonstrated (Geng et al., 2015). In addition, more complex architectures like ones that combine autoencoders with generative adversarial models were proposed recently (SOURCE). In many cases the purpose of autoencoders is dimensionality reduction and feature extraction or in other words the learned latent space of the bottleneck neurons. In other cases autoencoders are used to map images from one domain to another, for example camera recordings to label-images such that after training a labeled image can be retrieved from the decoder's output layer. In short, there are many possible ways to apply autoencoders and many architectures to realize them.

This motivates the question of how autoencoders work. As mentioned all autoencoders have a bottleneck layer. If applied for dimensionality reduction autoencoders are usually used to predict the input, in this case the image, with the input itself. Autoencoders consist of an encoder that contains the initial layers as well as the bottleneck layer and the 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 side product of the 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 ANNs such as the encoder are non-linear function approximators. Networks with multiple layers are especially well known for establishing named nonlinear correlation. Autoencoders allow for non-linear mappings to the latent space. 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. In this case, a convolutional variational autoencoder was used. In variational autoencoders a special loss function is used that ensures features in latent space are mapped to a compact cluster of values. This allows for interpolation between samples by moving on a straight line because regions between points in the latent space lie within the data and hence reconstructions of the decoder are more realistic. The most important feature of autoencoders for dimensionality reduction is preserved. The location of a point in latent space refers to a compressed representation of the input.These can be interpreted as features of the samples.

Different variational autoencoders were tested in the realm of the project. First, a simple variational autoencoder with a multilayer perceptron as decoder was implemented. The second approach was a comparatively shallow convolutional variational autoencoder. And the third was an autoencoder with a deeper encoder that was later used to design the networks for semi supervised learning. As already with a simple convolutional autoencoder some properties of asparagus pieces can be adequately mapped to a two latent asparagus space the results for the second of named networks are described here. It was derived from a standard example for convolutional variational autoencoders (SOURCE Keras example). The presented results are for a network that comprises two hidden convolutional layers in the encoder and two deconvolutional layers in the decoder.

Similar to the presented way of applying PCA, batches of images that contain only one perspective were used as input to the network. The downsampled dataset was used. Images had to be padded as the deconvolutional layers of the decoder can only increase dimensionality by an integer factor. The filters that were used for the deconvolutions layers double the tensor dimensionality. Therefore, the input shape had to be divisible by four without remainder. The depiction below shows the results.

It demonstrates that the features short thick and thin are mapped to separable clusters. As a tendency curvature correlates with a region in the lower periphery as indicated especially by the deconstruction. The other features (violet, rust and blooming) are not mapped adequately and they are not visible in the reconstruction. This shows that only some features of interest were mapped to the latent space and used to decode images. Reconstructions of autoencoders are known to miss many details. Better results could potentially have been achieved using larger input images. The possibility to generate, for example, more or less curved asparagus pieces may help to define a clear cut decision boundary and classify images accordingly. As a potential feature for

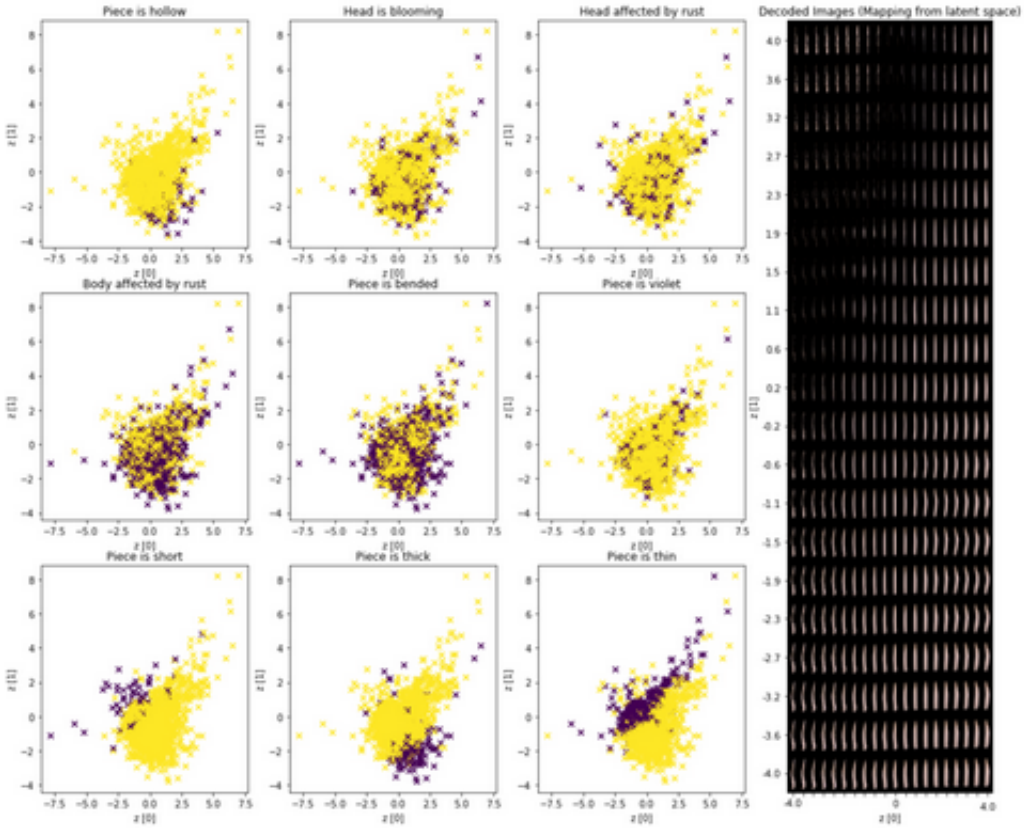


FIGURE 4.24: ??

asparagus sorting machines this would allow the user to customize the definition of curvature to his or her 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 our network that was employed on downscaled images.

4.3 Semi-supervised learning

We collected xxxx samples. With respect to the limited variance substantial amount of data. However, the target categories and information regarding the features present for each peace are unavailable. Labels had to be manually generated. Hence there is only a small subset of data where labels were attributed. A small amount of labeled data means that predictions can be successful only if there is little variance in the source values. Hence, for high dimensional data such as images sparse representations are desirable. A strategy that is especially appealing if large amounts of data are available is to automatically extract features instead of relying on feature engineering. In the previous chapter methods for unsupervised learning were explained and the results for the given data presented. One example are autoencoders. Here we used convolutional autoencoders with additional soft constraints in the loss function for semi supervised learning. Instead of extracting features and then using the resulting sparse representation to train a machine learning classifier named technique relies on doing both at the same time: The strategy in semi supervised learning with bottleneck layer neural networks is simultaneously

extracting features and enforcing that they (or at least some) correlate with the target categories.
The network: For the encoder a feedforward CNN was used that has proven to be suitable to detect at least some features with sufficient adequacy. ???????? layers.... NETWORK DETAILS. A challenge resulted from training with multiple inputs. As deep learning frameworks (here we employed Keras) usually require a connected graph that links inputs to outputs a trick was used. A dummy layer was introduced where all information that stems from the labels was multiplied by zero. The output vector was concatenated in the bottleneck of the encoder. As it contains no variance training and more importantly validation accuracies remain unaffected even though information about the categories that are predicted is added on the input side. The result is a few dead neurons only. For the decoder we chose the one of the variational autoencoder presented in the previous chapter. Two variations of the named network were tested: A convolutional variational autoencoder and a variational autoencoder for semi supervised learning.
A custom conditional loss function was used. If labels are present for the current batch a combined loss was used that comprised the reconstruction loss and the label loss. Here, reconstruction loss refers to the pixelwise loss that was used for the main task of the network - namely mapping of input images back to the same input while 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 error layer is low. The loss that is due to labels was multiplied by a custom factor (k). In addition it was defined such that it scales with the current pixel wise reconstruction loss and converges to a constant (c). These values were chosen aiming for an increase of the contribution of the label loss to the combined loss especially in late stages of training.

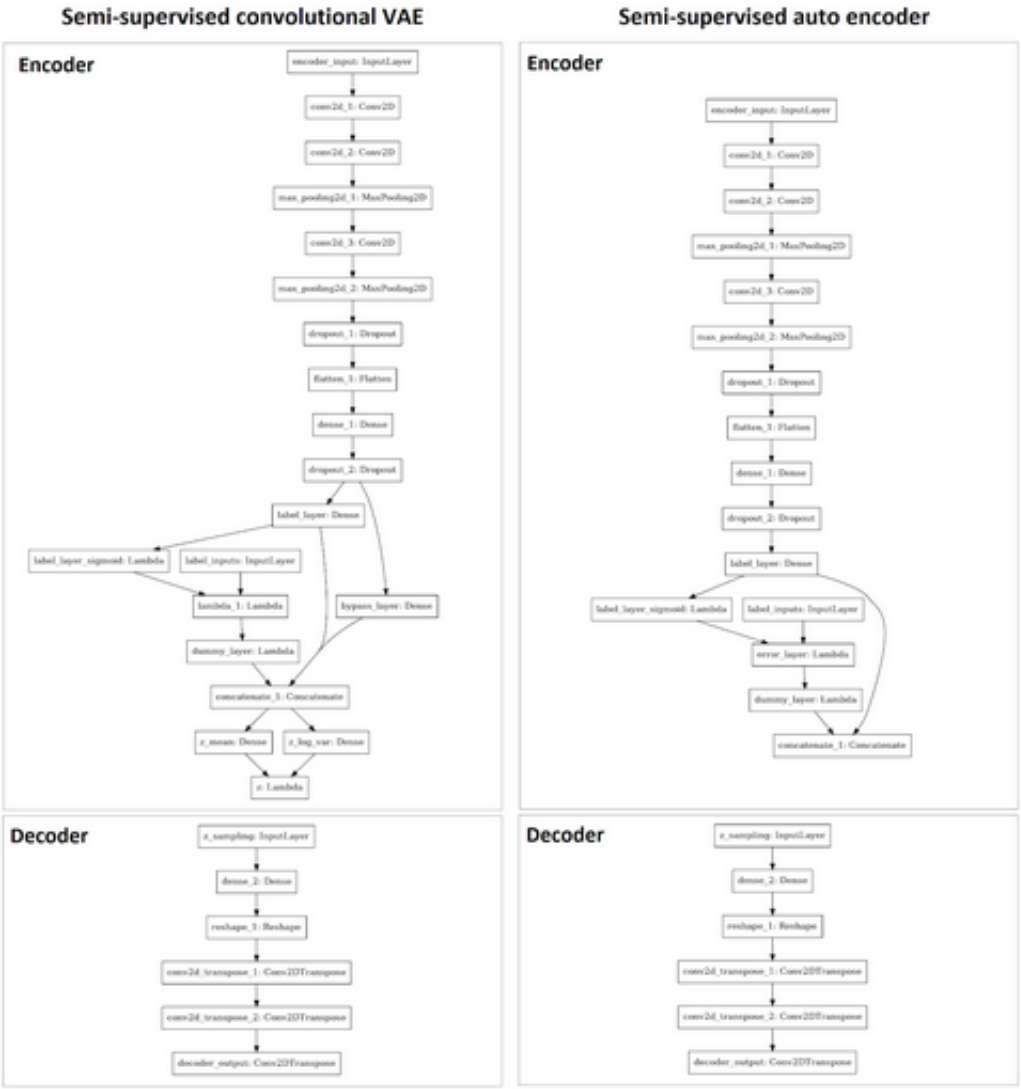

Semi-supervised convolutional VAE
Encoder
encoder_input: InputLayer
conv2d_1: Conv2D
conv2d_2: Conv2D
max_pooling2d_1: MaxPooling2D
conv2d_3: Conv2D
max_pooling2d_2: MaxPooling2D
dropout_1: Dropout
flatten_1: Flatten
dense_1: Dense
dropout_2: Dropout
label_layer: Dense
label_layer_sigmoid: Lambda
label_inputs: InputLayer
lambda_1: Lambda
bypass_layer: Dense
dummy_layer: Lambda
concatenate_1: Concatenate
z_mean: Dense
z_log_var: Dense
z: Lambda
Decoder
z_sampling: InputLayer
dense_2: Dense
reshape_1: Reshape
conv2d_transpose_1: Conv2DTranspose
conv2d_transpose_2: Conv2DTranspose
decoder_output: Conv2DTranspose
Semi-supervised auto encoder
Encoder
encoder_input: InputLayer
conv2d_1: Conv2D
conv2d_2: Conv2D
max_pooling2d_1: MaxPooling2D
conv2d_3: Conv2D
max_pooling2d_2: MaxPooling2D
dropout_1: Dropout
flatten_1: Flatten
dense_1: Dense
dropout_2: Dropout
label_layer: Dense
label_layer_sigmoid: Lambda
label_inputs: InputLayer
error_layer: Lambda
dummy_layer: Lambda
concatenate_1: Concatenate
Decoder
z_sampling: InputLayer
dense_2: Dense
reshape_1: Reshape
conv2d_transpose_1: Conv2DTranspose
conv2d_transpose_2: Conv2DTranspose
decoder_output: Conv2DTranspose

FIGURE 4.25: ??

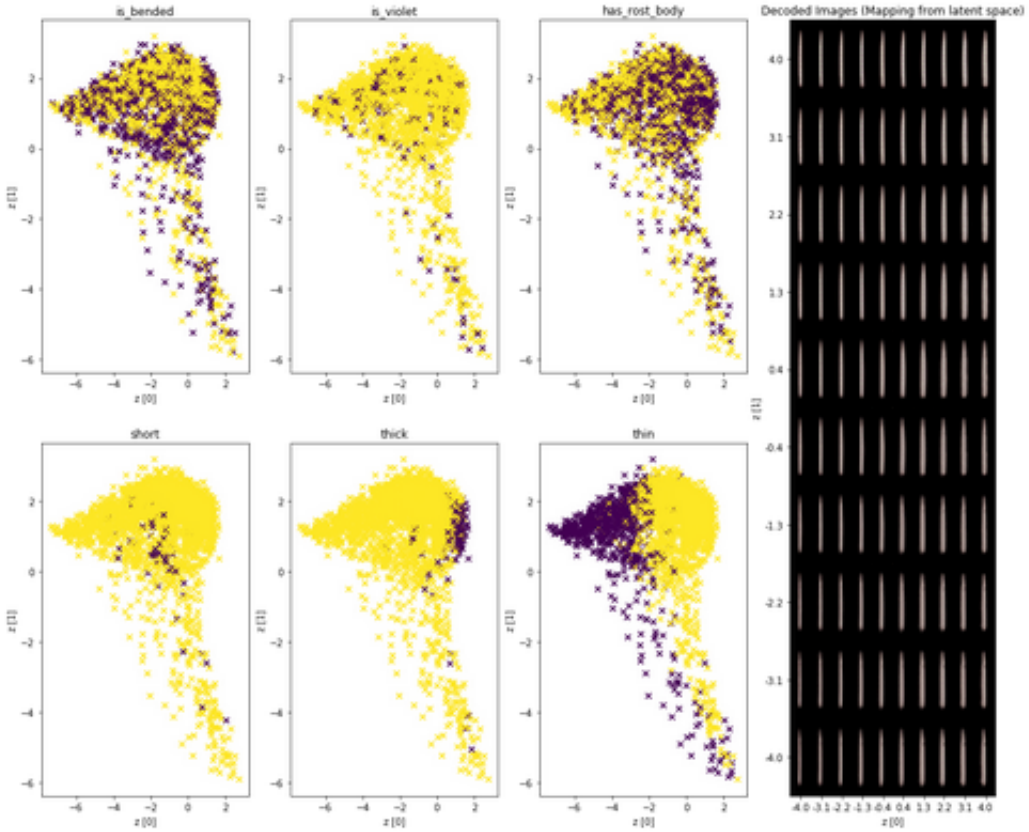


FIGURE 4.26: ??

	False positive	False negative	True positive	True negative	Sensitivity	Specificity
is_bended	0.04	0.37	0.04	0.55	0.10	0.94
is_violet	0.00	0.08	0.00	0.92	0.00	1.00
has_rost_body	0.14	0.27	0.20	0.39	0.42	0.73
short	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

FIGURE 4.27: ??

Chapter 5

Discussion

TODO

5.1 Comparison of classification approaches

TODO: In this section, we can compare the different approaches we used to classify our data.

5.1.1 Comparing architectures

TODO: Comparing the architectures of the approaches.

- Are they comparable?
- How is the structure different?
- Which parameters are different?
- Can we expect the results to be comparable?

5.1.2 Comparing results

TODO: Comparing the results of the different classification approaches.

- Which approach worked best?
- Which values do we want to use for comparison? e.g.
  - ROC curve (false positive (FP), true positive(TP))
  - validation accuracy
  - ...

5.2 Final result of the project

TODO: Our final conclusion on the project.

5.2.1 Scientific results

TODO: Short summary/evaluation of the project and its outcome on a scientific basis.

- What did we achieve in the end?

- What went well and what did not work so well?
- What would we do differently if we could start anew, what would we keep?
- What problems did we run into and how did we solve them?
- Were there any technical restrictions?

5.2.2 Organization

TODO: Short summary/evaluation of the project as a study project and teamwork experience.

- How did the organisation work?
- What did we achieve in the end?
- What went well and what did not work so well?
- What would we do differently if we could start anew - what would we keep?
- What problems did we run into and how did we solve them?

Chapter 6

Conclusion

TODO: We summarise again what we have learned and put the results and the project into a future perspective.

6.1 Summary

TODO: Short summary of the whole report.
What we aimed for, what we tried, and what we achieved (as best to our knowledge).

6.2 Next steps

TODO: What happens after the project is done?

6.2.1 Outlook of the project

TODO: How could the project continue?

6.2.2 Contribution to scientific landscape

TODO: How do our findings contribute to the ANN landscape/ open science? Can we publish the dataset?

Appendix A

Appendix A

A.1 Task list

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

- ...

A.2 Report list

TODO: Here, an overview of the report is given in chapters and subchapters. The name behind each chapter indicates who wrote the respective section.

- ...

Appendix B

Appendix B

TODO: Additional information will be given here.

Bibliography

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.

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.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2001). *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York.

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.

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.

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

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

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.

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.

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.

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.

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.

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.