TECHNISCHE UNIVERSITÄT WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Modular Strawberry Detection using YOLOv5 and ROS

BACHELORARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Bachelor of Science (BSc)

supervised by

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. M. Vincze
Univ.Ass. Dipl.-Ing. M. Hirschmanner

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by

Stefan Lechner
Matr. Nr. 01608096

Wien, im März 2023

Vision for Robotics Group
A-1040 Wien, Gusshausstr. 27, Internet: http://www.acin.tuwien.ac.at

Preamble

I want to thank Matthias Hirschmanner, who guided and supported me through this thesis. I learned more than I could ever imagine from him and grew closer to scientific writing. Additionally, I want to thank my project partners, Gerwig Weiss and Matthias Leonhartsberger, for our collaboration and the excellent time we had together. Finally, I am grateful for my family, who always supported me positively throughout the writing part of this thesis.

Wien, im März 2023

Abstract

We present an approach incorporating YOLOv5 and SORT into ROS for crop detection and tracking. Strawberries are classified in their different growth cycles (red, green and flower) and tracked over several video frames with unique identifiers. Previous work designed systems tailored from the ground up for the robot hardware and did not fully exploit software backbones such as ROS. Instead, we used its divisibility into nodes thoroughly to separate data generation and processing. As a result, the hardware components are also separated and can be changed as required. To be independent of open-source datasets, we produced our own with 905 examples. After training YOLOv5 detects and classifies red strawberries with a probability of 94 %, green strawberries with 77 %, and flowers with 76 % correctly. The speed of processing per frame is between 46.5 ms and 69.3 ms for 0 and 20 found instances. SORT assigns a unique identifier to classified objects across multiple video frames. It works perfectly while the robot stands still but fails to identify small objects while moving. The system's modularity allows easy hardware upgrades to increase frame rate and reduce processing time. Our system introduces an alternative way of running fruit recognition software on a robot that can be adapted to specific needs.

Kurzusammenfassung

Wir stellen einen Ansatz vor, der YOLOv5 und SORT in ROS zur Ernteerkennung und -verfolgung integriert. Erdbeeren werden in ihre verschiedenen Wachstumszyklen (rot, grün und Blüte) eingeteilt und über mehrere Videoframes mit eindeutigen Identifikatoren verfolgt. Frühere Arbeiten entwarfen Systeme, die von Grund auf auf die Roboterhardware zugeschnitten waren, und nutzten Software-Backbones wie ROS nicht vollständig aus. Stattdessen nutzen wir die Möglichkeit der Aufteilung in mehrere Nodes konsequent, um die Datenerzeugung und -verarbeitung zu trennen. Dadurch sind auch die Hardwarekomponenten getrennt und können bei Bedarf gewechselt werden. Um unabhängig von Open-Source-Datensätzen zu sein, haben wir unseren eigenen Datensatz mit 905 Beispielen erstellt. Nach dem Training erkennt und klassifiziert YOLOv5 rote Erdbeeren mit einer Wahrscheinlichkeit von 94 %, grüne Erdbeeren mit 77 % und Blüten mit 76 % korrekt. Die Verarbeitungsgeschwindigkeit pro Frame liegt zwischen 46.5 ms und 69.3 ms für 0 und 20 gefundene Instanzen. SORT weist klassifizierten Objekten über mehrere Videoframes hinweg eindeutige Identifikatoren zu. Es funktioniert perfekt, wenn der Roboter still steht, erkennt aber keine kleinen Objekte, während er sich bewegt. Die Modularität des Systems ermöglicht einfache Hardware-Upgrades, um die Bildwiederholrate zu erhöhen und die Verarbeitungszeit zu verkürzen. Unser System stellt eine alternative Möglichkeit dar, Fruchterkennungssoftware auf einem Roboter auszuführen, welches an die jeweiligen Bedürfnisse angepasst werden kann.

Contents

List of Figures

List of Tables

1 Introduction

A vital part of the future will be a resource-efficient economy to reduce the impact we humans have on the planet. Agribusiness is one of the sectors most affected by changes in global climate and needs to adapt to the changing environment while reducing its own impact. Since the start of the Neolithic Revolution, many processes have been made more efficient but the need for human labour still persists. Although machines can be integrated into various systems and processes, they still require skilled human operators. There is also some filigree work where hand-picking is still the standard. Even in industrialized countries, poor working conditions make these jobs increasingly unattractive, so flying workers in is not uncommon. Industry 4.0 is currently on the way to changing this system [1]. Therefore, human interaction is decreasing steadily with the introduction of unmanned machines.

This has created a research area, with private companies and universities actively working towards developing efficient solutions. At the Viennese Faculty of Electrical Engineering and Information Technology, we operate a small-scale strawberry field for a robotic test environment, as shown in Figure 1.1. This Bachelor's Thesis deals with implementing a real-time capable software solution to detect strawberries with the help of a state-of-the-art Convolutional Neuronal Network (CNN). Figure 1.2 shows the used mobile platform.

Figure 1.1: Growing environment spaced for robot operation

Figure 1.2: Robot setup with camera and laptop while operation

1.1 Motivation

The food demand will increase due to the growing population. Cole et al. [2] identified three factors to support this growth: reduce the food production demand, increase food production or avoid losses in production. In other words, reduce over-consumption, increase land usage or keep the plants healthy. We try to tackle the last problem with

robotic solutions, where weeds, pests and diseases are the primary concern. They are currently addressed with herbicides, pesticides and antimicrobials. They are added to the whole field without considering if the crop needs support. A more targeted approach would benefit the plants, the farmers and the environment.

Furthermore, those chemicals are under intense surveillance because of human health risks. The state of the research differs for all three of them. For certain types of pesticides exist, statistical linkages to the incidence of some diseases [3]. Weed killers are used in huge volumes and were found in the urine of many humans and animals. Van et al. [4] analysed glyphosate and mentioned worrisome accumulations in the environment, plant products and animal organs. There is currently no definite answer if it is dangerous and further research needs to be done. Antimicrobial resistance exists, and efficient deployment control is strongly advised [5]. If it is possible to reduce or eliminate them, research suggests it should be done.

Many countries use low-skilled workers as the foundation for food production. Developed nations import seasonal workers, and developing nations exploit people from lower social standing [6] [7]. Most immigrant workers earn little pay and have exploitative contracts. On top of that, they have to endure slave-like working conditions. If housing is provided, it is primarily substandard. Hazardous tasks are often done without insurance or safety equipment [4]. In poorer regions, the exploitation of children, pregnant women or the elderly is nothing unusual. As mentioned, those people suffer from low social standing. Food production should be designed such that the need for exploitation is diminished.

1.2 Challenge

In agriculture, a specific treatment can be chosen if the crop's health can be determined during production at all times. This is managed by precision agriculture (industry 4.0). It depends on a combination of the Global Positioning System (GPS) and timekeeping. For example, image data or already executed treatments are assigned a timestamp and a position. This information will be fused to generate a database in a three-dimensional map. Seamless communication between several data streams is then managed by wireless technology. Precision agriculture technology can be made intelligent such that the requirements for environmentally friendly and sustainable production are implemented in real-time in crop treatment and fertilizer equipment [8].

A cheap and efficient way to gather diverse information is to use cameras. A Convolutional Neural Network (CNN) is able to analyse health indicators and the life-cycle status of a given crop (e.g., disease, pest infestation or fruit parameters). Successful CNNs require good datasets (i.e., large, diverse, and accurate), as a small number of images cannot reflect real-world conditions. A strong computational unit is needed to facilitate real-time processing.

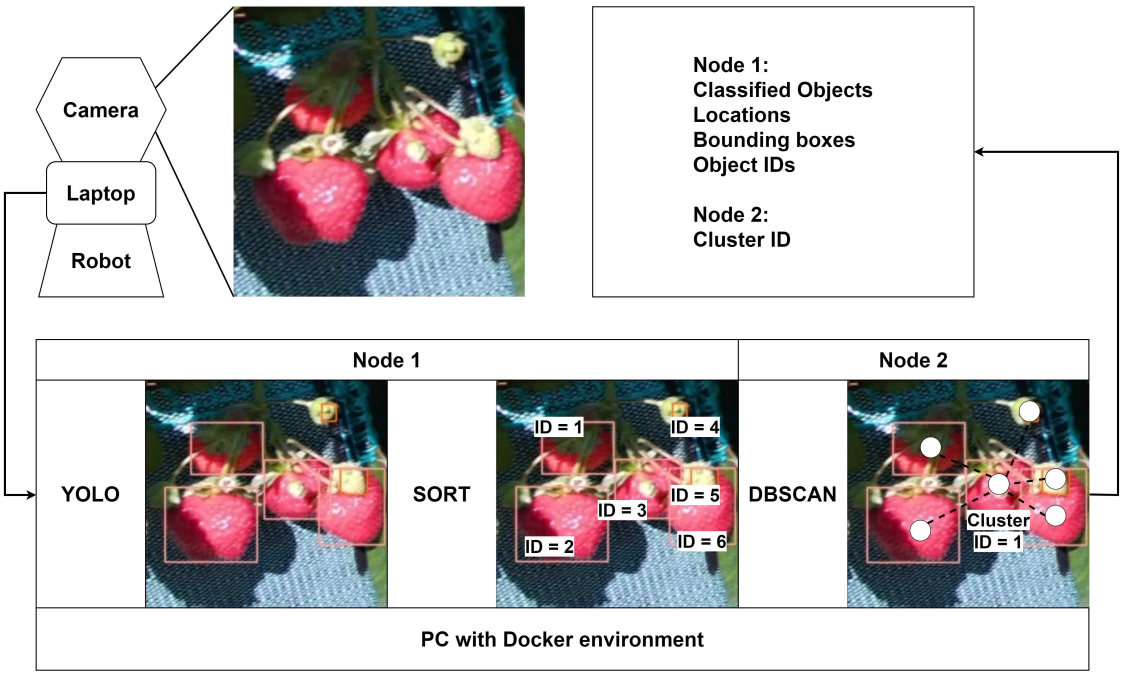

Figure 1.3: Processing pipeline from image taking with the camera over the developed detection system to generated data

1.3 Solution

In this work, a system was created from existing software packages. This includes YOLOv5 (You Only Look Once version 5) [9], SORT (Simple Online and Real-time Tracking) [10] and DBSCAN (Density-based spatial clustering of applications with noise) [11]. The programming language of choice is Python, which runs in the Robot Operating System (ROS) environment. Based on this, two nodes were developed. One deals with the detection and tracking of strawberries using IDs. The other helps to showcase the expandability of the system. It processes the generated data for further usage with the help of a clustering algorithm to generate cluster IDs. Figure 1.3 shows how this pipeline fits into the full setup.

1.4 Outline

First, similar projects are analysed in Chapter 2 to understand this project's goals. Then we introduce the used materials and methods in Chapter 3 Those consist of descriptions of the hardware, operation systems, convolutional neural networks and the implemented algorithms to post-process data. Subsequently, we discuss the proposed solution for our system in Chapter 4. This consists of integrating the introduced systems from material and methods into a ROS environment. The performance evaluation is then found in the next Chapter 5. We describe the experiments carried out and the results. The experiments address the training process of YOLO and the speed and accuracy of the implemented solution. The final Chapter 6 concludes and discusses the system's findings and future possibilities for research.

2 Related Works

Understanding how strawberries are cultivated in the current economy is the first entry point for robotic harvesting. In addition, the feasibility analysis of a robotic system is of paramount importance. Woo et al. [12] analysed virtual greenhouses based on robot compatibility and efficiency increase. The strawberries are planted in waist-high beds within reach of humans and robots, as seen in Figure 2.1. These beds are narrow so the berries can hang down on the side. With path planning, the robot traverses the greenhouse, recognises and picks the ripe strawberries. This reflects the reality of the real world. When comparing the operating speed, the robot is five times slower than a human. However, operating time will not be disrupted as there are no shift change restrictions. This gives the robot an advantage, which diminishes the difference in productivity. If we only consider the process of collecting data and ignore the subsequent steps of picking the crops, machines outperform humans in terms of speed. In order to be able to decide what growth phase the culture is in and when it can be picked, a quality assessment must be carried out. Quality is related to the visual characteristics of each plant and fruit. Therefore, image recognition software plays a vital role in automated robot systems.

Finite resources often constrain systems; engineers must make trade-offs between various performance metrics to achieve the desired balance between speed, accuracy and cost. Tu et al. [13] developed a passion fruit detection system with multiple scale faster R-CNN. The architecture consists of two parts. The first is a regional proposal network (RPN) that generates object proposals. The second stage classifies them and regresses a bounding box around them. They fuse the RGB image data with the depth data in this project. Depth data shows more robustness against illumination changes which leads to a higher probability of recognising crops. In addition, comparisons to older versions of YOLO were made to classify its effectiveness. They conclude that R-CNN is considerably slower but more precise. Nevertheless, the used YOLO versions are outdated by today's standards

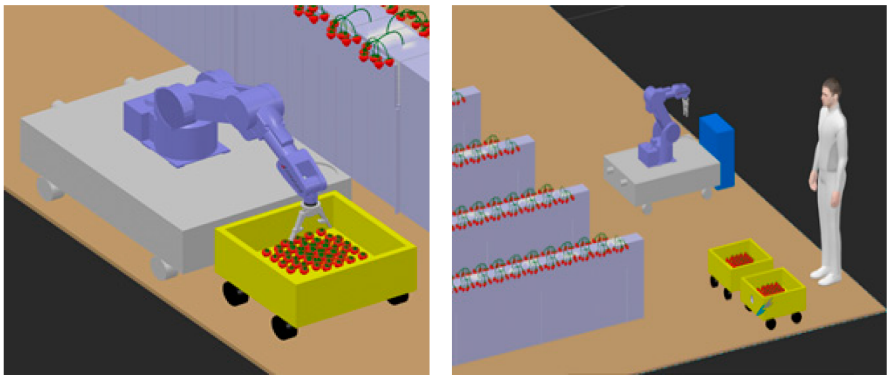

Figure 2.1: Virtual test environment for strawberry harvesting [12]

and were improved since then. Additionally, the depth data increases the complexity of the dataset. Exploring the potential benefits of utilizing an advanced version of YOLO could be an interesting research direction.

In addition to the correct architecture of the image recognition software, the dataset also affects the accuracy. Gai et al. [14] analysed different sizes of cherry datasets, ranging from 10 to 400 images per class. They found that the increase in accuracy flattens out with rising samples when using a transfer-learned network. Finding a good balance is significant because collecting data is very time-consuming. Producing a dataset with 400 samples per class should be in the realm of possibility because project-specific images reflect an environment more precise. However, some projects profit from generalisation, which can be learned from open-source datasets. If there are none of them, the production of a large dataset project is crucial. Therefore, setting up a suitable dataset for robust tools is very challenging [15].

Crop detection researchers have extensively studied strawberries due to their colours, form and popularity. The systems produced differ in their power consumption requirements and, thus, in their optimization effort. Nikolas Lamp and Mooi Choo Chuah [16] used a Raspberry Pi 3B as their main computing unit, which only has a power consumption of up to 12.5 W. Under this constraint, they implemented several optimizations to the image recognition software. The basic framework is a Single Shot Multibox Detector (SSD) similar to YOLO. They used image compression, colour masking, image tiling and network compression. The finished system can predict with 0.842 average precision (AP) at 1.63 frames per second (FPS). Especially if the system is planned to be mounted on a battery-powered robot, each component's power consumption should be considered. Depending on the image recognition software, the calculations consume considerable energy. If the main computing unit can be relocated away from the robot, the constraints can be neglected because of the connection to the power grid

The setup of the robot platform determines the versatility of the deployment. For example, Smitt et al. [17] build the system around an existing track in a greenhouse. This made robust construction possible because the weight is evenly distributed into the ground. Onboard are several RGB-D cameras, a robot arm, an onboard PC, a Wi-Fi router and a manual control panel. With the huge camera array, tall sweet pepper plants can be detected and analysed. This arrangement constrains the robot's versatility and adaptability because all those things make it heavy. Therefore, a wheel-based robot will have problems moving the equipment. Reducing weight and shrinking the structure is going to lessen this problem. For example, the PC could offload the calculations to a remote workstation. They implemented ROS to do so but will not send the crop data over Wi-Fi because of security reasons. It is only used for state inquiry and controlling. Combining those three things can lead to a smaller overall processing unit.

A more compact design was developed by Khort et al. [18]. The base consists of a custom four-wheeled car architecture, with a manipulator placed on top of it. Strawberries can be recognised and picked with the robotic arm that includes a camera. Similar to [16], the system is controlled by a Raspberry Pi. What differentiates the two is the type of image recognition software. In this system, a Hough Transform is used to detect

ripeness based on colour. This is not as computationally intensive as CNNs. This means local calculations are possible without major restrictions. The disadvantage is that this approach is overengineered and should be easier to solve with neural networks.

Xiong et al. [19] also encountered problems with traditional image recognition. They changed the hue and saturation of RGB images to find berries. A big challenge for them was the change in ambient illumination, especially its effect on the preferred saturation values. They found and implemented a correlation equation. This system is simple and fast but reaches its upper limits when the berries are not perfectly captured. For example, two nearby berries are recognised as one or if a stem occludes, one berry is recognised as two. Deep learning is suggested for further improvements [19].

In conclusion, these agriculture projects are built for specific use cases and are fitted to their hardware, which hinders them from being deployed in a general manner. This means, they do not represent a plug-and-play solution for software and hardware. The primary objective of this thesis is to develop a comprehensive software solution for crop detection that can easily be adapted and deployed to other environments. Similar to Smitt et al. [17], we use ROS because it is modular and platform-independent. Additionally, it is open-source and has a large community which makes it a good choice for the software ground structure. Through this, hardware components can be added and controlled efficiently and easily. The generated data is not outsourced via a network connection, only status messages. We implement ROS and use its message system to outsource the data processing to circumvent energy restrictions on a deployed robot.

Image recognition software requires high amounts of computational resources to detect crops. In contrast, YOLO is a good compromise between R-CNNs and handcrafted algorithms based on accuracy, power consumption and ease of use. Since it is a neural network, it can be tuned to different crops. Strawberries stand out because of their colour scheme, cultivation and popularity. Combined with automation and computer vision, the tedious task of recognising and monitoring them can be addressed. We use a custom dataset based on RGB image data to train the YOLO model. The testing hardware is kept simple, comprising of only a mobile robot, a laptop, and a camera. The feasibility and speed of the system are then evaluated through analyses to understand if it is possible to be implemented in a real-life situation.

3 Material and Methods

The developed system consists of hardware, software subsystems and algorithms. To understand their purpose, we will explain their functioning and structure in this chapter. The hardware consists of the environment, robot and camera. The systems are ROS and Docker. Finally, convolutional neural networks, SORT and DBSCAN are showcased.

3.1 Environment and Robot

To develop crop detection software, we use a simplified test planting area that allows easy access to the crops. It follows the scheme of the agriculture industry as shown in Figure 1.1. This means that plants grow in symmetrical arrangements for easier access and observation. Beneficial for us in relation to the robot's pathfinder. The floor is made of stone slabs and the space consists of a 2x2 grid where the robot can navigate. The strawberries grow in large containers with a height of around 35 cm and including the plants up to 60 cm. The paths between them provide enough space with a span of 135 cm for navigation. They are watered by a simple self-irrigation system which consists of a water hose with holes.

At the time of the final experiments, the strawberries were in their second growth cycle. In the first one, they grew less dense and lower in height after planting. This is perfect for object recognition because nearly every berry was easily visible with little occlusions. In the second year, they grew bushy, and only the ones overhung to the sides were obvious. Different plant species did better or worse. There are four types:

- Fragaria x ananassa ‘Ostara‘
- Fragaria x ananassa ‘Korona‘
- Fragaria x ananassa ‘Flamenco‘
- Fragaria vesca var. semperflorens ‘Rügen‘

The main robot for this project is a Pioneer 3-DX as seen in Figure 3.1. It was chosen because it fits between the grow beds, is compact and has a large mounting plate on top. Anything required for the respective application can be placed on this level. It consists of a two-wheel two-motor setup and one balancing wheel at the back. A battery and a sonar sensor are included in the base build. In addition, a rangefinder laser from Hokuyo [20] was mounted on the front for navigation. On top of the robot is a metal structure with a mounting port (1/4-20 UNC thread) for the camera. Depending on the use case and environment this setup can be adjusted in height and angle. It is mounted 55 cm above the ground and an angle of around 45° will be used. The angle has two advantages, the berries above and in front are recognized and the camera's automatic exposure adjustment

Figure 3.1: Front view of the Pioneer 3-DX robot with the mounted Hokuyo laser

is not confused with the bright sky. The robot's dimensions without any attachments are 50 cm × 40 cm × 24 cm and its wheel diameter is 19 cm. As a test platform, a laptop with a GTX 1660Ti from Nvidia is used to operate the hardware/software components.

3.2 Camera

A good camera system is a necessity for the proper functioning of fruit detection software. Image quality has a major impact on the functionality of the object recognition algorithm. Furthermore, the system can be more compact if a camera combines multiple sensors into one unit.

The first tests were conducted with a ZED camera by Stereolabs because it operates on image streams only and does not need an additional depth-sensor. Additionally, to avoid the fact that an infrared depth-sensor is disturbed by solar radiation. In retrospect, that was a premature conclusion! The Depth information is created through two stereo camera sensors as seen in 3.2 on the right camera. Because they are mounted at a certain distance from each other, the captured images will have a little disparity. It is largest when the object is near and goes to zero when far in the distance. This also implies that the accuracy is best in the near field since minimal changes in the far field cause large fluctuations in distance. To summarize, depth is inversely proportional to disparity. The ZED camera was unperformed during testing based on the depth accuracy and the camera quality. This was a related issue, when the sensor and lens produce images that are poorly defined and low in sharpness, good depth algorithms cannot compensate. Neupane et al. [21] compared the ZED camera to alternative options of which the Blaze 101 [22] and Azure Kinect [23] performed the best. They excel in daylight and at a distance of up to 2.5 m. The only difference is, that the Blaze is more robust. Luckily the Kinect was already in possession of the institute. Apart from that, this camera would be preferable

anyway because of the cost, availability and usability. The Azure uses a separate ToF (Time of Flight) sensor to calculate the depth information.

Figure 3.2: Stereo camera ZED and depth sensor camera Azure Kinect

Its significant components are a 12-MP-RGB camera sensor, a 1-MP depth sensor, an infrared projector, an accelerometer and a gyroscope (IMU). Additional things which are not decisive for image recognition, like the 7-way-microphone, will not be discussed. With a weight of 422 g, a solid mounting structure is necessary. On the backside, there is a USB-C Port for data transfer and a power jack with 5 Volt input. On the programming side, Microsoft provides an SDK with ROS capability for fast implementation. After installation, a simple user experience enables fast publishing of desired data streams. Several modes for image quality and depth type are available [24].

3.3 Operating System

Large parts of today's development happen in Ubuntu, what Microsoft Windows already excludes, especially in the field of research. Ubuntu 18.04 was chosen for the operating system because it was already in use in the institute and has an extensive library of well-supported packages. This decision saved time, and a new setup could be avoided on a shared computer with multiple projects working on it. This also resulted in the compatible version of our Robot Operating System (ROS Melodic) [25]. ROS Melodic only supports Python 2, while the used deep learning library requires Python 3. To avoid software compatibility issues, we executed the object detection algorithm inside a Docker container with ROS Noetic.

3.3.1 Robot Operating System

The above mentioned robot operating system is an open-source framework for building and programming robots. It provides libraries and tools for developing and managing robotic applications. Additionally, a big focus is on communication, perception, navigation and control. ROS is designed to work with a variety of robot hardware and software platforms, including those based on Ubuntu.

The here introduced fruit detection algorithm is part of a more extensive pipeline to which multiple people contribute. Therefore, a seamless connection is mandatory for smooth integration into a finished product. ROS acts as a distributed system where various

programs or nodes run concurrently and communicate with each other using standardized messages. These nodes publish and subscribe to data streams called topics. They have a specific identifier and build on the beforehand specified messages. For example, the camera publishes image data, point clouds and IMU data. This, in turn, is processed and published again for use by other software parts in real-time.

This framework can be used to develop autonomous systems. The supported programming languages are C++ and Python. Even if it is possible to use two languages in one ROS package, the focus has been shifted to Python-based development. Since the implementation and adaptation of the object detection model already require this language.

3.3.2 Docker

Docker underlies the same idea as a virtual machine (VM). The difference between them is how far virtualisation goes. For example, a whole operating system is emulated in a VM, which leads to a considerable performance loss. This is due to the fact that a complete system runs on a mother system. Therefore, hardware resources must be shared, and data flows must be emulated between the two. On the other hand, Docker uses the installed operating system to keep its abstraction level to a minimum[26]. If it comes to software version incompatibilities, a so-called Docker container is used to separate specific software environments. Those two benefits are incorporated into the system's real-time usage and version issues.

A neural network application has many software packages, and the version numbers constantly change. This presents an obstacle in the development and deployment phase. To cut it short, Docker is a controlled environment that can be easily shared and efficiently used on different machines.

3.4 Convolutional Neural Networks

In the domain of Computer Vision (CV) the aim is to extract information from images. Pieces of information are called features and represent a specific structure of an image. For example, finding edges, corners, or threshold extraction. These features are used to match similar images and understand what is depicted. They can be matched by classical algorithms that use hand-crafted features such as Scale Invariant Feature Transform (SIFT) [27] or Speeded Up Robust Features (SURF) [28]. Those are well-established CV techniques that focus on extracting specific features efficiently and quickly. Alternatively, one can use neural networks that learn to extract relevant features dependent on a specific task. The learning process is called training, which means that the network learns from ground-truth data to predict unseen data. Crafting specific features for classical algorithms can be a challenging and tedious task. In contrast, it is often easier to gather a representative dataset and achieve good results through this approach. Therefore, many algorithms from the past 20 years have been replaced with neural networks. However, it should be added that a combination of both approaches can bring clear advantages in further development [29].

3.4.1 Fully-connected Layer

The generic term neural networks can be divided into different types of networks. Two distinctions would be, e.g. feed-forward networks (FFN) and Recurrent Neural Networks (RNN) [30]. The FFN used in this project is a convolutional neural network (CNN). CNNs are designed for image recognition and classification, whereas RNNs are designed for processing sequential data such as language. Classification involves analyzing input data to determine the class to which it belongs.

A basic idea connects all neural networks, namely the presence of layers that are connected to each other visualised by directed acyclic graphs, where each element can be interpreted as a neuron. Those graphs are directed graphs with no directed cycles. An example of this is seen in Figure 3.4, where the connection between two layers is shown. We will focus on one element of the output layer to understand how the neurons are connected. The output neuron $y1$ is connected to all previews neurons. Every neuron has an additional factor: biases for outputs and weights for inputs. Expressed as a mathematical function, the representation is:

$$y_1 = \sigma(\sum_i (w_i \cdot x_i) + bias) \tag{3.1}$$

This structure is inspired by neuroscientific observations on how the activation of biological neurons works. Each output neuron y consists of a linear combination with all previous neurons x. In artificial neural networks, the stored values in the input layer neurons are multiplied with a weight w and a bias is added. After the calculation, the non-linear activation function σ is applied to it. Without σ, it would be a linear regression model that just represents linear combinations of the inputs, which can be reduced to one linear function. This does not allow the learning of complex relationships between inputs and outputs. A mathematical example of this would be a sinus function, without the activation function the layer is linear and a linear function cannot solve a non-linear function. These are important because they reflect real-life problems.

Two widely used are the rectified linear unit (ReLU) and the sigmoid function calculated as:

$$ReLU(x) = \max(0, x) \qquad SIGMOID(x) = \frac{1}{1 + e^{-x}}$$

Figure 3.3 shows the non-linearity and clear boundaries for the activation. These can then be shifted on the x-axis using the bias. This change is combined with the weight changeability to solve an optimisation task or, in other words, training. One that is often used is the gradient descent, which is explained in Section 3.4.5. The optimisation process is typically time- and resource-intensive.

3.4.2 Convolution Layer

The most critical layer of a CNN is the convolution layer. It generates a feature map of an input using the convolution operation. Figure 3.6 shows where this layer is deployed into an example network. The input is an image with width, height and three colour channels.

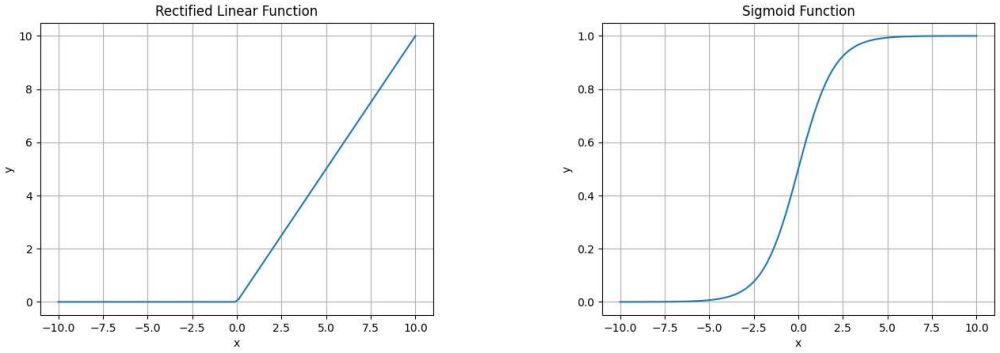

Figure 3.3: Representation of two non-linear activation functions, Rectified Linear Unit and Sigmoid

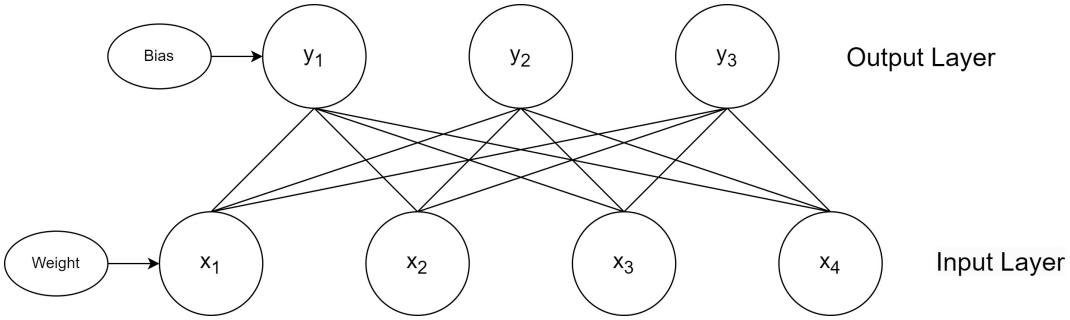

Figure 3.4: Connections between 2 rows of nodes

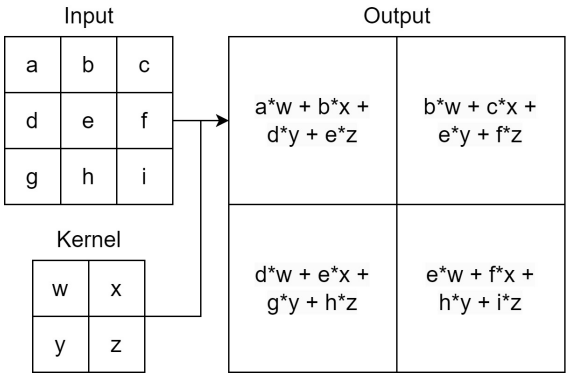

Figure 3.5: 2D Convolution represented by a kernel that moves over an input

Mathematically, this is a three-dimensional tensor. Tensors are multidimensional arrays of values related to a vector space. Convolution is applied to this using a spatially smaller kernel in width and height. Simply put, one can think of a three-dimensional matrix with a set of learnable parameters that move over the input matrix and performs matrix multiplication. Figure 3.5 shows a 2D example, where the input is 3x3 and the kernel 2x2. The completed calculation generates a feature map size of 2x2 [30]. Additionally, an increase in depth is possible by repeating the convolution with additional kernels. Each depth extension can be interpreted as a feature map for extracting features.

Discrete convolution with multidimensional arrays can be written as

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m,j-n)K(m,n). \tag{3.2}$$

A visualization of this is found in Figure 3.5. Where viewed together, K represents the kernel with the weights and I the input values. Compared to the equation 3.1 of the fully-connected layer, the weights are shared between multiple input values. This reduces the number of learnable parameters while enabling each neuron to learn distinct features from its receptive field. Convolution allows the layer to capture connections between adjacent elements of the input tensor. These represent the spatial features we want to find. Since convolution is a linear mathematical operation, activation functions are applied to its output.

3.4.3 Structural Layers

With the introduction of the basic principles in the last section, we have already outlined the groundwork for two simple types. Flatten layers are used to change the dimension of layers, and pooling layers are used in combination with convolution layers. Those two are not trained, but help adapt the preceding layers' output tensors. Flattening itself is pretty simple; it transforms a tensor into a 1D feature vector. The pooling layer reduces the dimension by combining the outputs of neuron clusters into a single neuron. Popular approaches use the maximum or average value of the neurones considered. This allows the network to select superior invariant features and improve generalisation [31].

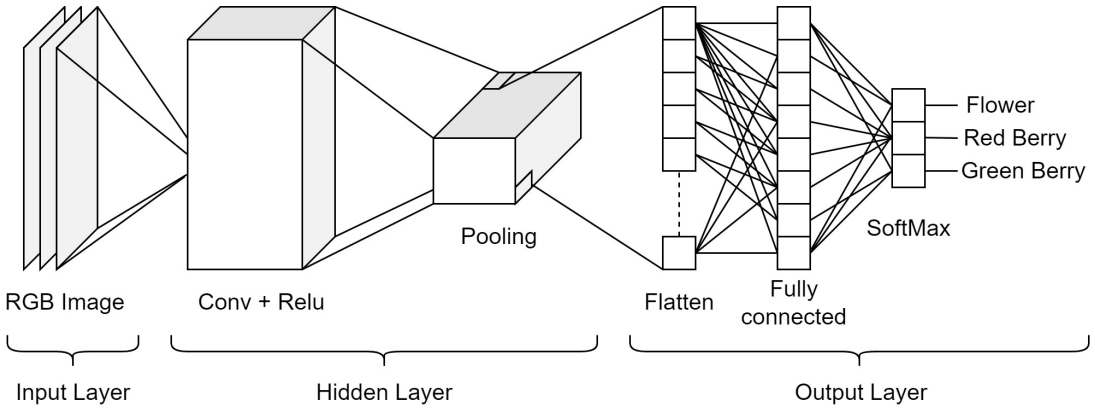

Figure 3.6: Network layer structure from image to classification

3.4.4 Network Structure

Figure 3.6 shows a simple network structure with the main layers. The input layer is the data that is fed into the network, the hidden layer extracts features, and the output layer generates the output in the required dimension for the specific task, e.g. n class scores for each class in a classification task.

In our example, the input data consists of an image with three colour channels that are converted to a tensor. After that, the prepared data are fed into a convolution layer for spatial feature extraction and passed on to a ReLU layer to introduce non-linearity. The following pooling layer reduces the width and height of the output tensor. A convolutional neural network often consists of several such layer combinations, each increasing the depth of the tensor while reducing the spatial dimensions of the feature maps. The depth correlates with the possibility of extracting more complex features [32]. Tracking what features are extracted precisely is challenging, so it is often called the hidden layer.

In the classification task, the output layer uses extracted features to calculate a score for each class. Flattening the multidimensional feature vector is the first step. This is done so the fully-connected layer can accept a consistent interface between layers. Figure 3.6 shows two fully-connected layers after the flatten layer. The first connects all the neurons one by one and uses ReLU activation, while the second generates the output in the required dimension for the specific number of classes. It uses a particular activation function per output neuron. The SoftMax function described in 3.4.5 is used to normalise the output of a network to a probability distribution over the output classes (three classes in this example). The softmax function rescales the output tensor of the last layer so that every element lies in the range of [0,1] and sum to 1. It is defined as:

$$SOFTMAX(x_i) = \frac{exp(x_i)}{\sum_j exp(x_i)} \tag{3.3}$$

3.4.5 Training a Network

A combination of specific layers is called a model. When the model is first created, its network structure is initialised with random weights and biases. As a result, the model has no knowledge or ability to classify input data accurately, and its output will be random. To make the model useful, it must be trained to learn the desired behaviour. It adjusts the weights and biases during training to minimise the error between predictions and the expected outputs. The optimisation task is solved by minimising the error which is calculated from the cross-entropy loss. In combination with gradient descent a local minimum of differentiable functions is tried to be found to minimise the error. Finally, the weights and biases are updated with backpropagation [33].

The distribution p from 3.3 is compared to the expected outputs y, a vector of zeros except for a one, indicating the expected class. This information is saved in a dataset. Calculating the errors is done by the cross-entropy loss for N detected classes defined as:

$$\mathrm{L}(p,y) = -\frac{1}{N}\sum_{i}[y_i * log(p_i) + (1 - y_i) * ln(1 - p_i)] \tag{3.4}$$

During training, the network minimises this loss function by adapting the weights and biases so that the model reduces the difference between its predictions and the actual labels. This is done by calculating the derivative of the loss functions, called the gradient. The parameters are iteratively adjusted toward the negative gradient, called gradient descent. In other words, gradient descent attempts to find the minimum of a function by taking steps in the direction of the steepest descent. The size of each step is determined by the learning rate; these training-specific values are called hyperparameters.

The goal is to find the weights which minimize the loss. In order to find these, all the linked derivations are gone through from the back to the front to calculate the gradients efficiently. That is called back-propagation and allows the information from the loss function to flow backward [33]. To understand what is the idea behind it, we will go through a short example by using a simple network consisting of three weights, three biases and four nodes, as seen in Figure 3.7. We describe the basics with the last two layers, but one can extend the math to several layers. The goal is to find how sensitive the loss function is to these variables and which one causes the most efficient decrease.

The activations are represented as σ_N, where the indices represent the Nth layer. In combination with our ground truth in the dataset, the loss $L(\sigma_N,y)$ can be calculated. Here σ_N again consists of the activation before it, the same as the structure in 3.1, but now with only one preceding node:

$$\sigma_N = ReLU(w_N * \sigma_{N-1} + b_N) = ReLU(z_N) \tag{3.5}$$

To find the sensitivity between loss and a variable e.g. w_N a differential equation has to be solved. All variables in between are then taken into account with the chain rule. This states that the derivation can be divided into sub derivations which are multiplied together:

$$\frac{\partial L}{\partial w_N} = \frac{\partial z_N}{\partial w_N} \frac{\partial \sigma_N}{\partial z_N} \frac{\partial L}{\partial \sigma_N} \tag{3.6}$$

With this method, one can calculate all the derivatives of the variables from back to front. These are then changed by a certain amount, defined by the learning rate. After the update, this procedure is repeated with the rest of the training data split until everything is processed. This is called the train loop. Also worth mentioning is the validation loop, where the model is checked with the validation partition of the dataset to verify that it does not overfit. Overfitting means that the model is so well trained it only can detect the dataset and is not generalised for outside data. With this loop, one can detect problems in the dataset or training setup. One full iteration over the training and validation dataset is called an epoch. After this, the parameters are fully updated. Since gradient descent is an optimization procedure several iterations should be done to get the best result. How many epochs have to be performed depends on the model performance after n epochs.

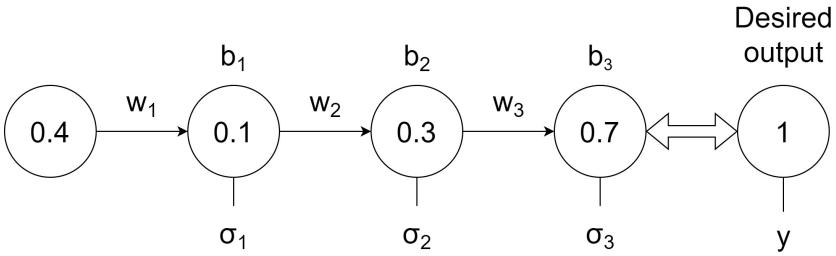

Figure 3.7: Simple network with 4 nodes

3.4.6 Datasets

To achieve a well-trained model, one needs enough data to train it. The data type can range from images over 3D human skeleton key points to words. Every data point in the dataset consists of some information and its ground truth. Therefore, false ground truth can be destructive in the training process because the model learns false classification. That is why setting up the dataset correctly is important. Additionally, if the dataset is small the risk of overfitting is there [34]. To overcome such restrictions with small datasets, data augmentation is important [35]. For example, simple techniques would be changing the brightness or colour of an image, but there are several more, as seen in Figure 3.8. In this image, one can see an example of the additional data augmentation applied by our implemented image recognition software. Data should also be captured from different perspectives, which means taking images from different angles. The last part is to split the data into training-, validation- and test datasets for performing the training task of the network. When considering all these things, a dataset can consume a huge amount of human labour, making these labelled data collections expensive to produce.

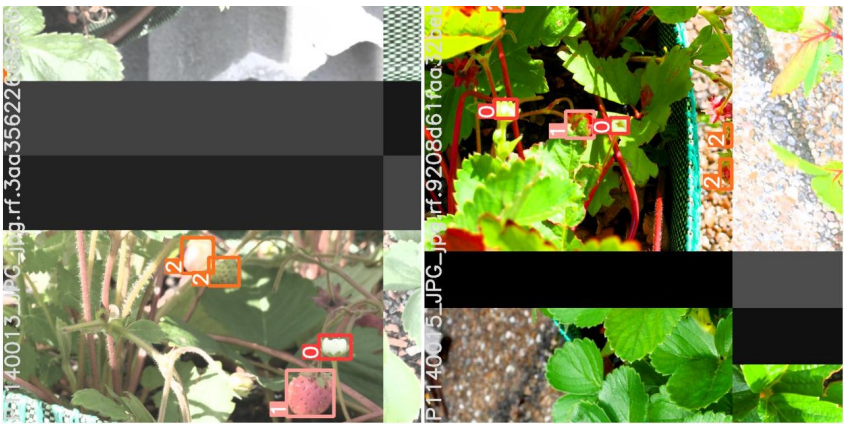

Figure 3.8: Additional image augmentation by YOLOv5 of the strawberry dataset

3.4.7 YOLOv5

A plant recognition system requires more than just classifying which object is present in an image, as it requires more properties, such as the location and size of the detected object. It is also essential to find several objects in one image and do this computationally fast, especially for real-time crop detection. Redmon et al. [36] developed "You Only Look Once" a unified real-time object detection system that meets these requirements. This means the network analyses the entire image and its objects simultaneously. Since this publication, several improved versions have been developed. Two contestants are YOLOv4 [37] and YOLOv5 [9]. The most significant difference between v4 and v5 is the used frameworks: Darknet and PyTorch. In comparison, the speed and accuracy are similar [38], but YOLOv5 is more user-friendly regarding setup and training [9]. YOLOv4 was mentioned for completeness as it has a published paper.

YOLO works by splitting the image into a grid as seen in Figure 3.9. For each cell, bounding boxes and confidence scores c are predicted. When the centre of an object is located within a specific grid cell, the cell is tasked with detecting the boxes. A box is represented as (x,y,w,h,c), where x and y are the locations in the image. Width and height correspond to w and h. The confidence is defined as the probability of how accurate the box is multiplied by the intersection over union (IoU) value. If there is no object present the confidence score will be zero, otherwise, it is equal to the IoU. IoU is calculated using the predicted box and the training data box as the intersection area divided by the union area. In addition, a cell also predicts class probabilities shown as the map in Figure 3.9. Regardless of the number of boxes, there is only one set of predictions of the class probabilities for each cell. When testing, class and bounding box probabilities are combined to:

$$Pr(Class_i|Object) * Pr(Object) * IoU = \text{class-specific confidence scores} \tag{3.7}$$

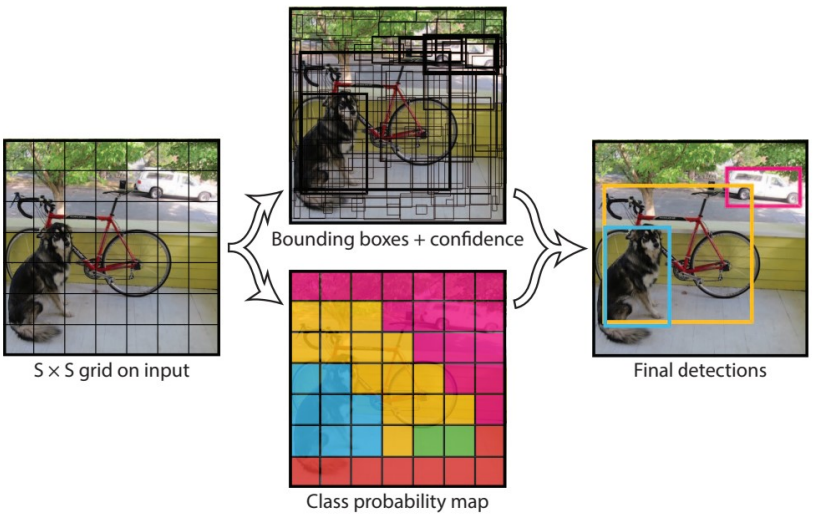

Figure 3.9: YOLO division into a classification grid pattern and best bounding box detection per cell [36]

3.5 Process generated data

The camera system and YOLO generate vast amounts of data that are only partially organized by the neural network. Additional algorithms are used to classify them even better or to create connections between the data points.

3.5.1 Creating unique identifier with SORT

When the camera moves past a strawberry, YOLO recognises it and marks it with a bounding box. The neural network only operates on a single frame. This means there is no connection between the frames which could be accomplished with 3D-CNNs. By connecting all images from an object the assessment is improved. To achieve this, another calculation must give the strawberry a recognisable identifier that only exists once for a specific box. It is further used for assignability in the system.

The SORT algorithm can solve this problem quickly and efficiently. We chose this algorithm based on real-time capabilities. Bewley et al. [10] shows that Sort is superior to other solutions in terms of speed and accuracy, without typical drawbacks. This tracker was developed for the MOT Challenge 2015 (Multiple Object Tracking), which is a framework for the fair evaluation of multiple people tracking algorithms [39].

The underlying functionality builds on the idea that a bounding box can be represented as a system of differential equations. This linear constant velocity model approximates inter-frame displacements. The state of a target is represented as:

$$x = [u,v,s,r,\dot{u},\dot{v},\dot{s}]^T$$

The centre's horizontal and vertical pixel locations are represented by u and v, respec-

tively. The scale and aspect ratio are denoted by s and r. When a target is detected, the bounding box is used to update the states with the help of a Kalman filter. If there is no detection, the linear model is solved.

To assign detections to existing targets, their bounding box location in the current frame is estimated by predicting their geometry, and then the IoU is calculated to determine the best bounding box match. A new tracker is initialised when a IoU_{min} threshold is triggered. With it, a new unique identifier is created and associated. When the tracker finds no new bounding boxes, it is destroyed after a specific time.

3.5.2 Clustering data points with DBSCAN

Berries grow in vines. They can be interpreted as distinctive marks of a single plant or a whole grow bed. Grouping the found bounding boxes up into clusters helps to find them easier later in the growing phase. These collections of points do not form particular shapes but grow arbitrarily. Imagine two vines of berries growing over each other. They can create a lot of possible arrangements. Additionally, the information on how many clusters should be found is unavailable beforehand. It is conceivable that a cluster may be enveloped by another cluster. This rules out a lot of the existing cluster algorithms. We used DBSCAN to deal with these constraints. It is more effective than other well-known algorithms in finding arbitrarily shaped clusters.

The algorithm searches for core and border points. It needs two parameters, the minimum points for initiating a cluster and the allowed distance (epsilon) between them, as seen in Figure 3.10. On the right side is an example of two clusters. To initialise a new cluster, a random point is chosen and checked if there are enough neighbour points in epsilon distance to comply with the minimum points threshold. These are then also considered part of the cluster. From this on, only the epsilon distance will determine if a nearby point is part of the cluster. This step is repeated until no more new points can be added. After this, a new cluster will be initialised from a point that does not yet belong to anyone. Points that do not comply with the epsilon distance and minimum points threshold are recognised as noise. This simple tactic is the foundation of a fast and reliable cluster algorithm.

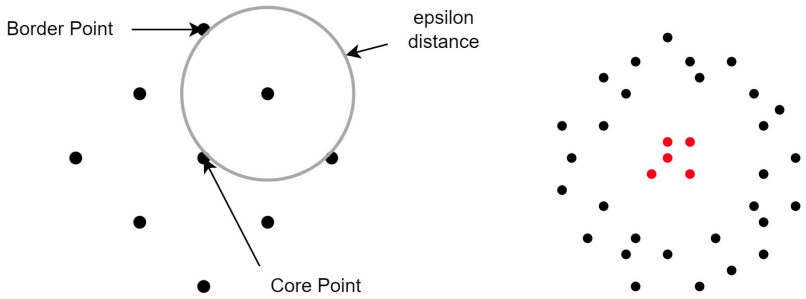

Figure 3.10: Minimum distance between points and detection of arbitrary clusters

4 Developed System

As written in Section 2, most crop detection systems are specifically built for their hardware and usually cannot be used in a general manner. We developed a system with ROS that joins hardware, data streams, YOLOv5 and SORT into a coherent package[1], which can be adapted and expanded. As a showcase, we implemented an additional node, which includes the DBSCAN algorithm. It further processes the generated data to find clusters of strawberries to better identify them in the growth phase. This represents just a proof of concept to test the expandability and is not further analysed.

The structure is based on the design principles of ROS. This means data generation and its processing can be split up into units that run on different PCs, which is possible through ROS nodes. Figure 4.1 shows how the hardware and data streams interact. The laptop acts as the computing unit that is connected to the camera and the robot. They exchange data between them. The camera sends image data as a 1280×720p 30 Hz video stream to the laptop and the robot can be operated with control commands. As indicated by the dashed line, a secondary PC is able to offload the processing of the data and make it available to other subscribers. These data streams can then be used on the laptop for additional tasks, such as controlling the robot. As described in 2, offloading energy-intense processing, makes the robot lighter and energy efficient. There is also the possibility of processing everything on the laptop but this limits the effectiveness of the entire system. More processing leads to more energy consumption and less runtime for an autonomous robot. Additionally, if the laptop is overloaded with processes the finite computing power leads to increased computation time.

4.1 Software backbone

As seen in Figure 4.2, our system is split up into two software environments. The first one is a Docker container based on Ubuntu 20.04 with ROS Noetic to run YOLO/SORT and the clustering algorithm. The second one is the main computer's operating system Ubuntu 18.04 which runs the Kinect SDK node to publish image data. The Kinect SDK is a package that allows the control of what the camera publishes. This split was needed because Ubuntu 18.04 only supports ROS Melodic, consequently the usage of Python 3 is not feasible. Python 3 is necessary for YOLO. Encapsulation not only solves the problem but also provides the solution for an easy-to-use system since a Docker container contains everything needed to run an application on a Linux system. The Kinect SDK node is not restricted by this and can run on the main operating system. This node can easily be switched by another one if there is a wish to use a different camera. The only thing to check is the name of the topic it publishes to. It should be the same as the tested Kinect

[1] https://github.com/St333fan/Strawberry_Detection_YOLOv5_SORT_ROS

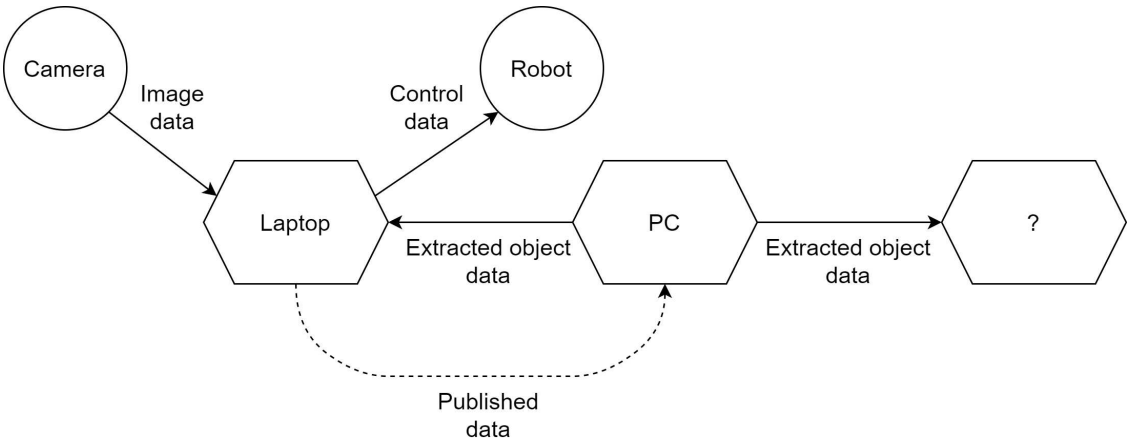

Figure 4.1: The laptop controls the camera and the robot, while the computationally intensive data processing is outsourced to another PC

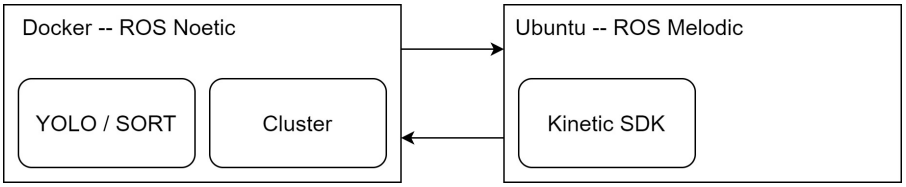

Figure 4.2: Distribution of software packages based on version dependencies

SDK Node.

To conclude, three nodes operate in two different environments that can communicate with each other via ROS. For YOLO, we forked the repository from Ultralytics [9] from March 2022 where we specifically use the YOLOv5m model. It was implemented in a ROS subscriber node which is written in Python, where it processes the images published by the Kinect node. The process resolution is set to (1280, 720) because this is a good compromise between finding small objects and processing time. When YOLO classifies found bounding boxes, those are initialized with an id by the SORT algorithm to track them. All the essential pieces of information about an image are then saved in a ROS message and published on a topic. Those include the location and size of the bounding boxes, classes, confidence scores, several identifiers and time data. The cluster node then subscribes to this topic and receives the messages. DBSCAN from scikit-learn [2] then further processes the information by clustering all bounding boxes and allocating them to each other. This additionally enhances assignability because it connects the growing locations of the objects. The DBSCAN node shows how important it is to be able to expand the system with additional nodes.

[2]https://scikit-learn.org/stable/modules/clustering.html

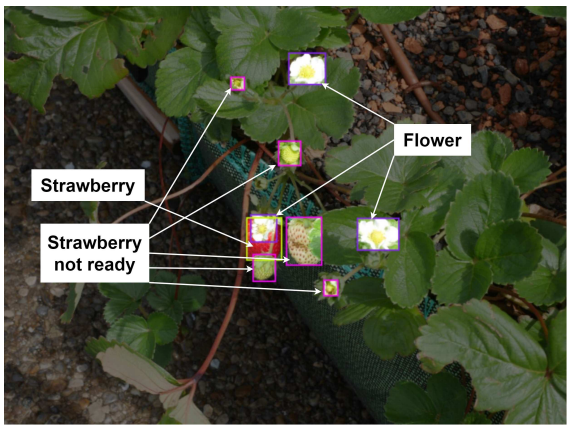

Figure 4.3: Annotations with bounding boxes from the three different classes

4.2 Dataset creation

Many different plant species are used in agriculture. Relying solely on open-source datasets can be a limiting factor because they do not always represent them sufficiently enough. Therefore, we explored the possibility of creating our own dataset [3]. The camera setup consists of a Panasonic G70 with a Panasonic Macro Lens of 30 mm. It was used to shoot 905 images of the strawberries in the growing beds, whereas 48 have no class labels. The camera's perspective range from hip high to close-up, with a downward angle to simulate the robot's view. In Figure 4.3 one can see an example of how the images are annotated. The different label types are flower, strawberry and strawberry_not_ready. Consequently, the bounding boxes are highlighted in different colours purple, green and pink. The additional 48 images represent therefore background classes.

Figure 4.4 shows how examples look after which were annotated. The labelling cutoff point was chosen as a quarter of the whole object, lower for ripe strawberries because finding them under leaves is the most crucial task (better a false classification than no classification). That should represent enough features of the object for detection. The first row consists of strawberries that are ripe for collecting. Leaves and other classes may occlude some examples. The second row represents the strawberry_not_ready. They range from strawberries that are not entirely red to the sight of the first berry right after the flower lost its leaves. The third row, representing flowers, starts with a counterexample, as seen on the left side, where one leaf is still present and labelled as a flower. To conclude, there are three difficult-to-annotate situations, occlusions, switch between flower to green strawberry and percentage redness of a strawberry.

[3] https://huggingface.co/datasets/St333fan/StrawberryRedGreenFlower

Figure 4.4: Annotation examples for the three classes in different growth stages

5 Experiments and Results

In this chapter, we evaluate our system through multiple experiments to identify its limitations and opportunities for improvement. We showcase the training setup and interpret how well the model works with our self-made dataset. In addition, the integrated software components YOLO and SORT are analysed for accuracy and speed.

5.1 Training of YOLOv5

The computer that was used for training consists of Ubuntu 18.04, 16 GB RAM, an Intel Core i5-6500 CPU and an NVIDIA GeForce GTX 1070 TI GPU with 6 GB of VRAM. During training, we applied the following data augmentations. During training, we applied the following data augmentations. An equal probability of one of the following 90-degree rotations: none, clockwise, or counter-clockwise. Furthermore, a random brightness adjustment of between -25 and +25 per cent was included. It was then split into an 80/20 distribution for the training and validation step. After preparing the data, the YOLOv5m model was trained using transfer learning with the train.py script from the git repository [9] for the crop detection task. The training parameters were set as followed: image resolution 1280×720p, batch size 4 and training epochs 150.

After the last epoch, we evaluated how well the model performed with our dataset. First, we analyse the whole dataset and then evaluate the model against the validation split. Figure 5.1 shows comparisons between the classes in the dataset. Those are unevenly distributed based on labelled instances per class. This means the captured images were taken when the plants had primarily strawberries which were not ready for harvest. The right graph shows that the objects do not evenly distribute spatially over the images. This is due to the close-up shots of the objects which naturally centre when they fill out most of the image. Both of those anomalies are based on a bias in the dataset generation and could be partly solved with model design or enhanced data augmentation.

Figure 5.2 represents a confusion matrix between the classes. It shows the predicted class compared to the true distribution. The values represent probabilities. The diagonal represents the comparison with the same class. Flowers are recognised with a probability of 97 %, red strawberries with 94 % and green strawberries with 94 % correctly. The values are close to 1, meaning the model finds almost all objects in the validation set. The absolute precision is 95.2 %. Misclassification comes up predominantly with the background class which is the environment. Flowers perform the best, followed by red strawberries, while green strawberries perform poorly. Section 5.2 further evaluates this. Between the three main classes is nearly no misclassification. It seems that the network mainly uses colour information for classification.

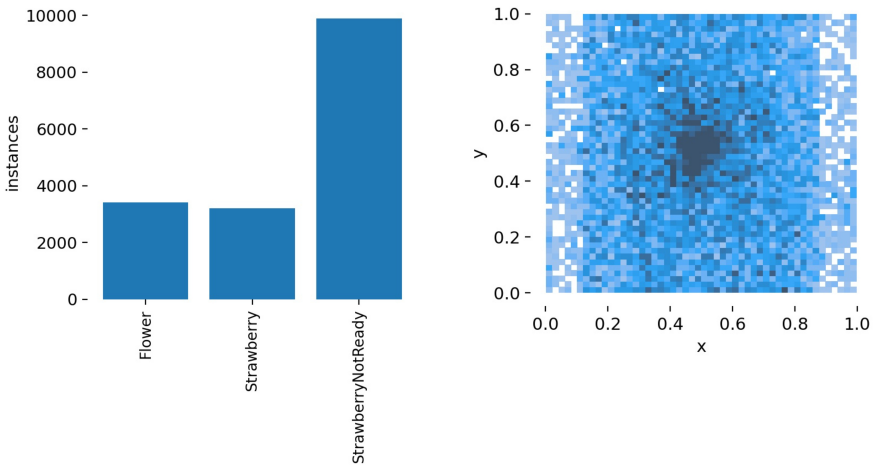

Figure 5.1: Class and spatial distribution of the dataset

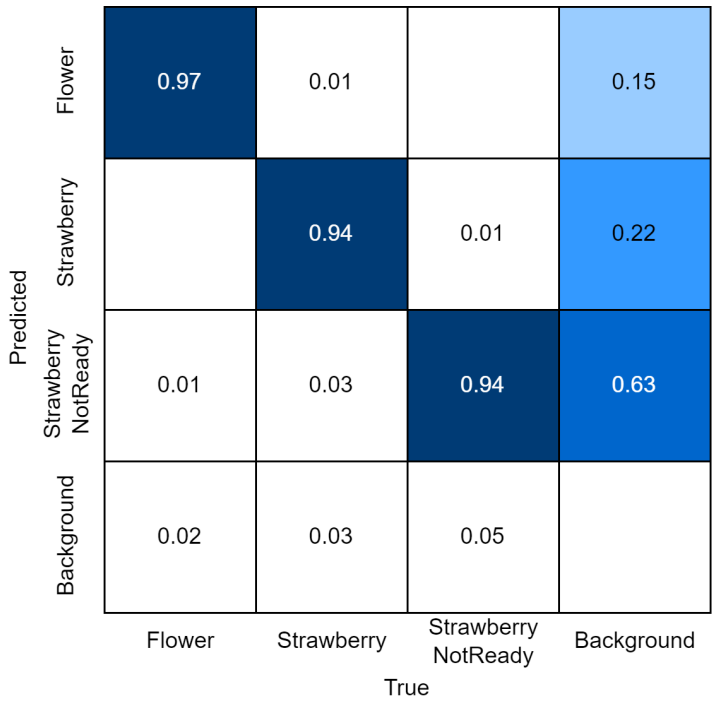

Figure 5.2: Confusion matrix between the classes with probability scores

Video	**Flower**	**Strawberry**	**Strawberry_not_ready**
Run 1	0.642	0.961	0.810
Run 2	0.744	0.94	0.721
Run 3	0.827	0.937	0.816
Average	0.76	0.943	0.77

Table 5.1: Probabilities of correct classification per video sample

5.2 Metrics of detection

Analysing the system based on metrics estimates how reliable the generated data is. We recorded three video streams from the robot's camera perspective driving through our testing environment. By doing this, the system is evaluated on specific camera properties and certain characteristics that occur during robot operation, such as motion blur. We analysed every seventh frame of the videos and compared them to the output of YOLOv5. This was done to have enough spacial difference between every analysed frame and not capture the same situation twice. The parameters for the inference are a confidence threshold of 0.4 and an IoU threshold of 0.5 Finally, 46 examples were produced and analysed.

Table 5.1 shows the detection accuracy per class. Accuracy means how many detections are classified with the correct class. The upper three represent, respectively, one video each. In the last row, we show the average accuracy of all videos. Compared to Figure 5.2, only the red strawberries showed similar detection results in the testing environment. Figure 5.3 shows how challenging the image data are. Shaking of the robot, overexposure and small objects dampen the detection accuracy of the flowers and the green strawberries. Figure 5.4 shows misclassification examples from the running system. In the first row are misclassified red strawberries with leaves and green strawberries. The next row shows leaves classified as green strawberries. The last one represents flowers, where the middle part is classified as a green strawberry and the last two show problems with light exposure. A perfect example of an image is shown in Figure 5.5. This does not represent the view from the robot and shows the plants from the first year. Additionally, there is a change of 8.4 % that the background is classified as one of the three classes. To conclude the accuracy is not necessarily based on the performance of YOLOv5, but is influenced by the environment and capturing conditions.

In addition to accuracy, other metrics allow further conclusions to be drawn because it is often inadequate in the case of imbalanced data. That is why we compared it to other standard detection metrics. In Table 5.2, we compare accuracy to precision, recall and F1 score. In terms of the values given, the conclusion differs if we compare the previous result to the new metrics. The model has high precision and F1 scores for all classes except strawberry, which may indicate that the network struggles to identify this class correctly. It seems that the misclassification with brown leaves reduces the precision score significantly.

Metrics	Flower	Strawberry	Strawberry_not_ready
Accuracy	0,76	0,94	0,77
Precision	1	0,81	0,93
Recall	0,92	1	0,99
F1	0,96	0,89	0,96

Table 5.2: Accuracy compared to other common detection metrics, divided into individual classes

Figure 5.3: Extracted frame of the driving robot

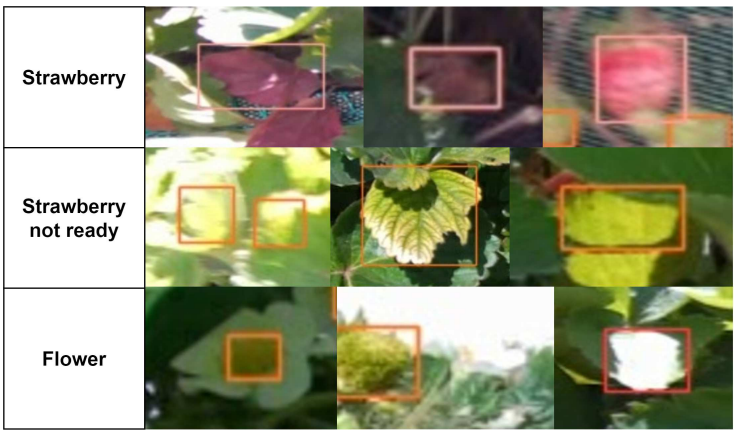

Figure 5.4: Classification issues because of leaves, misclassification with other classes and light exposure

Figure 5.5: Processed image from the first year of growth

Objects	Time in s
0	0.0465
20	0.0693

Table 5.3: Median processing time per found objects in one image

5.3 Speed of detection in ROS

To analyse the speed of the system, it was run in a controlled environment same as we did in section 5.2. As described in section 4.1 the developed software runs in a Docker container on the same computer. For the processing of each image, the time is measured and averaged after the end of the transmission. Table 5.3 shows the best possible time for an image with no detected object compared to 20 found objects. The times are acceptable because if we take, for example, a worst-case of 10 Hz (100 ms) the robot sees an object several times.

5.4 Accuracy of SORT

The same objects will be seen in several consecutive frames. If it is possible to identify each object, the gathered information can be assigned to a unique object instance. For example, several perspectives will enhance the assessment of an entirely red strawberry. We analysed the SORT algorithm based on its ability to create unique identifiers and assign them to the same objects over several frames. An identifier is generated when an instance overlaps on two consecutive frames and is kept alive for two frames when the object instance is lost for re-identification. The minimum IoU is set to 0.05 because the objects are small. Figure 5.6 shows the found objects by YOLO with a unique identifier. This was tested while the robot stood still, which means that several consecutive frames show almost the same picture. SORT performed error-free and all 12 objects held their assigned identifier. Problems arise when the robot moves. 10 out of 12 lost their identifier

Figure 5.6: Unique identifier mapped to detected objects; while standing, only objects 11 and 12 can hold the identifier while moving because they are large enough

and received a new one. This behaviour was also the case for the rest of the video stream. Large objects performed best and could hold the identifiers for 7-10 frames. The green strawberry and flower with the identifiers 11 and 12 were the ones that held the identifier in our shown example. Identifiers are lost because the frames overlap too little; therefore, the IoU for an object is often zero. As a result, the identifier can no longer be assigned to the object.

6 Conclusion and Future Steps

The implementation of YOLOv5 and SORT into the ROS environment was successful. Especially the newer YOLO version is user-friendly for model training and implementation purposes, which sets it apart from other neural networks and handcrafted algorithms. The self-made dataset performed well and showcased the possibility of detecting crops where no data is available. Outsourcing the data processing to an external computer improves modularity and enables optimised construction for a robot.

Further research should be done by upgrading the system with more potent hardware to operate the camera with a framerate of 60 Hz. This should improve the performance of SORT because the consecutive frames will overlap more. Additionally, the shorter exposure time will lead to motion blur reduction. Alternatively, the camera structure could be adapted to enhance the closeup of the objects. A newer version of YOLO could be implemented with enhanced dataset augmentation and resolution. One possible augmentation for a driving robot would be simulated motion blur. Additionally, the dataset could be extended to include depth information for improved feature extraction and tracking.

The system in its current form can be used to operate plant recognition with a robot. It does not require any specific robot hardware used in this work and has the possibility to be adapted to different circumstances. In addition, the data can be distributed and processed location-independently with the help of ROS.

Bibliography

[1] Y. Liu, X. Ma, L. Shu, G. P. Hancke, and A. M. Abu-Mahfouz, „From industry 4.0 to agriculture 4.0: Current status, enabling technologies, and research challenges," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4322–4334, 2020.

[2] M. B. Cole, M. A. Augustin, M. J. Robertson, and J. M. Manners, „The science of food security," *npj Science of Food*, vol. 2, no. 1, pp. 1–8, 2018.

[3] K.-H. Kim, E. Kabir, and S. A. Jahan, „Exposure to pesticides and the associated human health effects," *Science of the total environment*, vol. 575, pp. 525–535, 2017.

[4] A. H. Van Bruggen, M. He, K Shin, *et al.*, „Environmental and health effects of the herbicide glyphosate," *Science of the total environment*, vol. 616, pp. 255–268, 2018.

[5] M. E. Velazquez-Meza, M. Galarde-López, B. Carrillo-Quiróz, and C. M. Alpuche-Aranda, „Antimicrobial resistance: One health approach," *Veterinary World*, vol. 15, no. 3, p. 743, 2022.

[6] K. Hoggart and C. Mendoza, „African immigrant workers in spanish agriculture," *Sociologia Ruralis*, vol. 39, no. 4, pp. 538–562, 1999.

[7] B. Sinha, „Working conditions of agricultural workers: A reflection of socio-economic status.," *Human Geographies–Journal of Studies & Research in Human Geography*, vol. 4, no. 2, 2010.

[8] J. De Baerdemaeker, „Precision agriculture technology and robotics for good agricultural practices," *IFAC Proceedings Volumes*, vol. 46, no. 4, pp. 1–4, 2013.

[9] Ultralytics, *Git yolov5*. [Online]. Available: `https://github.com/ultralytics/yolov5` (visited on 02/17/2023).

[10] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, „Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*, IEEE, 2016, pp. 3464–3468.

[11] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, „A density-based algorithm for discovering clusters in large spatial databases with noise.," in *kdd*, vol. 96, 1996, pp. 226–231.

[12] S. Woo, D. D. Uyeh, J. Kim, *et al.*, „Analyses of work efficiency of a strawberry-harvesting robot in an automated greenhouse," *Agronomy*, vol. 10, no. 11, p. 1751, 2020.

[13] S. Tu, J. Pang, H. Liu, *et al.*, „Passion fruit detection and counting based on multiple scale faster r-cnn using rgb-d images," *Precision Agriculture*, vol. 21, no. 5, pp. 1072–1091, 2020.

[14] R. Gai, N. Chen, and H. Yuan, „A detection algorithm for cherry fruits based on the improved yolo-v4 model," *Neural Computing and Applications*, pp. 1–12, 2021.

[15] J. G. A. Barbedo, „Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification,“ *Computers and electronics in agriculture*, vol. 153, pp. 46–53, 2018.

[16] N. Lamb and M. C. Chuah, „A strawberry detection system using convolutional neural networks,“ in *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 2515–2520.

[17] C. Smitt, M. Halstead, T. Zaenker, M. Bennewitz, and C. McCool, „Pathobot: A robot for glasshouse crop phenotyping and intervention,“ in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 2324–2330.

[18] D. Khort, A. Kutyrev, R. Filippov, N. Kiktev, and D. Komarchuk, „Robotized platform for picking of strawberry berries,“ in *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)*, IEEE, 2019, pp. 869–872.

[19] Y. Xiong, Y. Ge, L. Grimstad, and P. J. From, „An autonomous strawberry-harvesting robot: Design, development, integration, and field evaluation,“ *Journal of Field Robotics*, vol. 37, no. 2, pp. 202–224, 2020.

[20] Hokuyo, *URG-04LX-UG01*, `https://www.hokuyo-aut.jp/search/single.php?serial=166`, [Online; accessed 10-November-2022], 2021.

[21] C. Neupane, A. Koirala, Z. Wang, and K. B. Walsh, „Evaluation of depth cameras for use in fruit localization and sizing: Finding a successor to kinect v2,“ *Agronomy*, vol. 11, no. 9, p. 1780, 2021.

[22] Basler, *Blaze 101*. [Online]. Available: `https://www.baslerweb.com/en/products/cameras/3d-cameras/basler-blaze/blaze-101/` (visited on 01/26/2023).

[23] Microsoft, *Azure kinect dk*. [Online]. Available: `https://azure.microsoft.com/en-us/products/kinect-dk/` (visited on 01/26/2023).

[24] ——, *Azure kinect dk hardware specifications*. [Online]. Available: `https://learn.microsoft.com/en-us/azure/kinect-dk/hardware-specification` (visited on 01/26/2023).

[25] ROS, *Ros melodic morenia*. [Online]. Available: `http://wiki.ros.org/melodic` (visited on 01/29/2023).

[26] Docker, *Use containers to build, share and run your applications*. [Online]. Available: `https://www.docker.com/resources/what-container/` (visited on 01/28/2023).

[27] D. G. Lowe, „Distinctive image features from scale-invariant keypoints,“ *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[28] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, „Speeded-up robust features (surf),“ *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[29] N. O'Mahony, S. Campbell, A. Carvalho, *et al.*, „Deep learning vs. traditional computer vision,“ in *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1 1*, Springer, 2020, pp. 128–144.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[31] D. Scherer, A. Müller, and S. Behnke, „Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks–ICANN 2010: 20th International Conference, Thessaloniki, Greece, September 15-18, 2010, Proceedings, Part III 20*, Springer, 2010, pp. 92–101.

[32] J. D. Bodapati and N. Veeranjaneyulu, „Feature extraction and classification using deep convolutional neural networks," *Journal of Cyber Security and Mobility*, pp. 261–276, 2019.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, „Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[34] P. Thanapol, K. Lavangnananda, P. Bouvry, F. Pinel, and F. Leprévost, „Reducing overfitting and improving generalization in training convolutional neural network (cnn) under limited sample sizes in image recognition," in *2020-5th International Conference on Information Technology (InCIT)*, IEEE, 2020, pp. 300–305.

[35] C. Shorten and T. M. Khoshgoftaar, „A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, „You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[37] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, „Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[38] M. Sozzi, S. Cantalamessa, A. Cogato, A. Kayad, and F. Marinello, „Automatic bunch detection in white grape varieties using yolov3, yolov4, and yolov5 deep learning algorithms," *Agronomy*, vol. 12, no. 2, p. 319, 2022.

[39] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, „MOTChallenge 2015: Towards a benchmark for multi-target tracking," *arXiv:1504.01942 [cs]*, Apr. 2015, arXiv: 1504.01942. [Online]. Available: `http://arxiv.org/abs/1504.01942`.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, im März 2023

Stefan Lechner
Matr. Nr. 01608096