Image Recognition For Tomatoes Based On Virtual 3D Plant Models

tomato 0.83
tomato 0.83
tomato 0.75
tomato 0.62
tomato 0.74
tomato 0.73
tomato 0.68

Willem van de Kraats
Under supervision of Qingzhi Liu

https://github.com/WURwillem/Image-Recognition-For-Tomatoes-Based-On-Virtual-3D-Plant-Models

1 ABSTRACT

Greenhouses can be benefitted with modern techniques to regulate the crops that are grown in them. Important factors include temperature, humidity, CO_2 concentration, daylight length and nutrient availability. An important next step is the direct monitoring of the crop and further automation in the growing and harvesting process. This is traditionally done manually. Identification of pests and diseases, ripeness and potential nutrient deficiencies are examples of factors that can be determined by examination of the crop by the farmer. Detection and identification are key words in this process. Image recognition is especially suited to perform these tasks via digital means. To aid the development of novel greenhouse techniques we are benefitted by an accurate model representation of the plants we want to grow. By digitalizing the test environment we have a larger amount of control over variables in our experiments and a practically infinite amount of test data.

This thesis consists of two main parts. The first is adapting code from a model that generates randomized virtual 3D tomato plants. We optimize for adaptability of certain parameters to real life measurements and increased workability. Second is assessment of the usability of the model plants in the development of image recognition networks for tomatoes. By doing this we also quantify the degree of realism of these virtual plant models.

The code of the tomato plant model was simplified and properly documented in order to increase workability in future projects. Attention was paid to the incorporation of easily manipulatable factors that capture the proportions of the plant. Measurements from specific varieties can be obtained and used in custom models that are more true to life than the ones generated by the original script. In order to detect tomatoes, a crucial step in many applications, we used image recognition. We trained two types of networks in order to assess the usability of the model generated tomatoes in the development of image recognition applications. One was provided with a train set with real life tomatoes and the other with one containing our computer model generated ones. We were especially interested in the performance of the network trained on modelled tomato images when confronted with pictures of real tomatoes. This is a variation on the principle that when a computer model is not able to distinguish between real and synthetic data, the model that provided the synthetic data is practically perfect. When performance is similar or high enough we can conclude that the model generated tomatoes are suitable to aid in the development of image recognition networks that recognize real tomatoes.

Although performance of the two networks was not similar, the tomato model proved useful for the development of image recognition networks that are supposed to locate and identify tomatoes in an image. Performance was higher when a network was trained on real tomato pictures and tested on the corresponding test set, which was expected. However, the performance of the model tomato trained network was still sufficient on the real tomato test set. Also, it required a little amount of fine tuning to match the performance of the purpose made network.

2 CONTENTS

3 INTRODUCTION

Automation processes for greenhouses are currently researched and developed at Wageningen University and Research. Growing conditions can already be monitored and influenced to reach an optimal balance between production efficiency and costs. The next large step in automation is the monitoring of the crop itself via digital means. Central to this is detection of objects of interest. Examples include detection of ripe fruits, targeted spraying of pesticides and identifying plant diseases.

Wageningen University and research is especially interested in Digital Twins[1]. It is one of the three main investment areas in the strategic plan of 2019 to 2022. Making a digital twin allows for a better understanding of the subject that is being researched, which in turn allows for better control over the growth of the crop. Having a virtual environment modelled exactly after a real life counterpart would be ideal for the development of a greenhouse system. In this digital environment it is easier to monitor and manipulate the variables at play. However, a digital twin for tomato plants is not yet available. Therefore we need to turn to simulations of tomato plants. A simulation is modelled in such a way that it resembles reality, in contrast to a digital twin that is supposed to copy reality. The model environment can still be used to run experiments and test novel techniques that would be too costly to perform in real life.

This thesis consists of two main components:

1. Adapting the code for a model that generates randomized 3D tomato plants to improve readability and workability. This should enable further development to achieve more realistic and true to life tomato plant models.
2. Assessing if the tomato plant models are sufficiently “real” to aid in the training of image recognition networks.

In the first component we work with a script provided by András Polgár[2] that generates randomized tomato plants. His results look good and resemble tomato plants to a large degree. A shortcoming is that there is no underlying theory provided on which these tomato plants are based. They only seem to resemble tomato plants to the human eye. In this thesis we will set out to adapt his script in such a way that data from real life measurements can be easily inserted into the script in order to make more true to life models. In the process we simplify the script and add commentary code to make the script more easy to work with in follow up research.

In the second component we test if the generated tomato plants are indeed usable as a substitute for real life data. We use image data of the virtual plants in a train set for an image recognition network. If this trained network can also predict tomatoes in real plants we can conclude that the modelled tomato plants resemble reality enough for the purposes of developing image recognition networks for tomatoes.

Considering the strong practical character of the first component of this thesis a lot of advice for future developers is included. This is done in the form of commentary code, code descriptions and examples scenarios of desired adaptations. The script can then be used to generate custom tomato models. The commentary code can be found in the script that is provided with this thesis. The code description can be found in chapter 6. It details how the main parts of the script function and lists easily manipulable parameters for the user. Examples of how to change the plants structure with these parameters are given as well.

The models from the first component are used in the second. Aside its use here, the model also lays a groundwork for the development of more realistic tomato crop models. Being able to replicate tomato plants contributes to our fundamental understanding of the way the plant grows. In potential future versions we can also input growth related parameters, like nutrient availability, in order to run more advanced digital experiments. This can save time and money that otherwise had to be invested in real life trials.

The second component of this thesis has a more research oriented approach. Namely if digital tomato plants could potentially add value to the train sets for image recognition networks for tomatoes, or even replace them. This can be determined by model performance. If a neural network trained on model plants performs equally well on real plants as network that was trained on real plants, we can conclude that the plant model is mimicking reality sufficiently enough.

3.1 RESEARCH QUESTION

The main research question for the second component of this thesis is as follows:

- Can tomato pictures generated from the 3D image model be leveraged to train DL models for tomato recognition?

We will also explore and answer other related questions. Namely:

- What are the effects of the size of the train set for the model generated tomato network?
- What are the effects of the incorporation of decoy images?
- How suitable is fine tuning the network to bridge the gap between model environment and real world application?

It is important to consider the context in deep learning applications. Deep neural networks are highly efficient in specialized tasks. However, slight variations might confuse the network. Our image recognition network for tomatoes is a good example. We expect our model tomato plants to mimic real tomatoes enough that at least a sizeable proportion of real tomatoes can be recognized by training the network on the virtual ones. However, performance should be at least slightly lower due to the small variations.

4 RELATED WORK

For this thesis project the tomato generator script from the Ignition-Gazebo repository will be adapted to fit our needs. Then, we will train an image recognition network on these generated tomato plants. We will assess the usability of the tomato generator script as synthetic data. Synthetic data as a substitute for real data has been a topic of interest in different fields. We will discuss some examples to show that this is a promising approach for image recognition of tomatoes as well.

4.1 IGNITION-GAZEBO

This thesis builds on previous work done by András Polgár[2]. In the fields-ignition project he provides a way to generate randomized tomato fields for Ignition Gazebo[3]. Ignition Gazebo is part of Ignition Robotics, a set of libraries that is used to develop robot and simulation applications. Using the fields-ignition repository we can create a tomato field that consists of randomly generated tomato plants. We can then use this 3D environment in Ignition Gazebo to let a small virtual robot drive around between these tomato plants using controller input.

In order to install the repository a specific version of Ubuntu is needed in order to avoid dependency issues. When installing the repository, please use the install manual provided with this thesis. Note that RVIZ and Ignition Gazebo need to run simultaneously. Gazebo is the simulated environment where the robot drives around and RVIZ visualizes the camera view of the robot.

Included in the repository is a tomato detector script as well. However, this detector is a simulation of how a real detector would function. The locations of the tomatoes are already known and it is only determined if these tomatoes are in the line of sight of the robot. A prefixed maximum detection distance of 3 is given and whether a tomato is detected or not is determined by the use of masks. In practice, this is similar to how an image recognition network would perform. More on this detector can be found in the scripts folder of the repository in the simple_detector.ipynb file. In the image below we see a visualization of how the detector works. However, it is not of further relevance to this thesis.

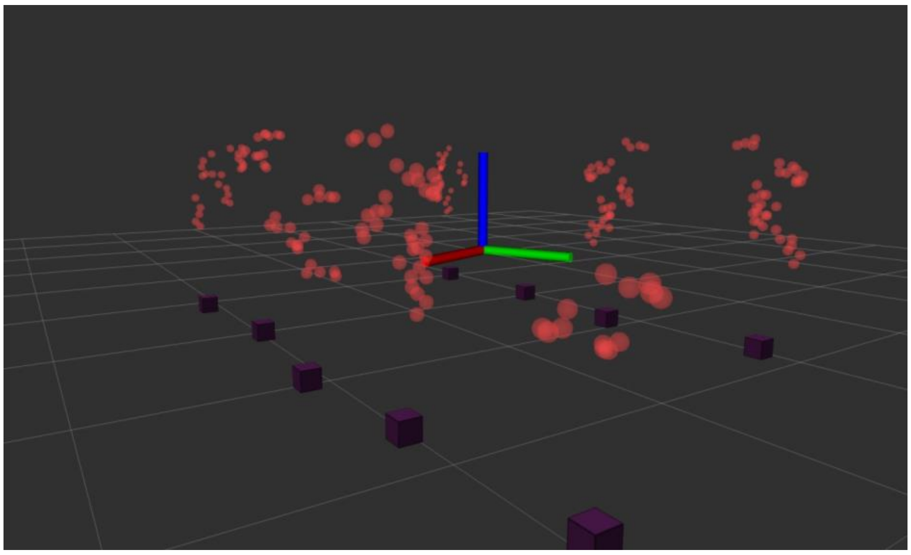

Figure 1: Tomato detector script in working. The purple blocks are the base of tomato plants and the red circles are detected tomatoes. The red, green and blue XYZ axis represents the camera of the robot.

A Playstation 3 and an XBOX controller were confirmed to be compatible with the software and were detected automatically via a USB connection. A downside that was encountered was the low resolution that the camera view from the robot provided in RVIZ, which would be too pixelated for use as image recognition train data. No solution was found to increase the resolution.

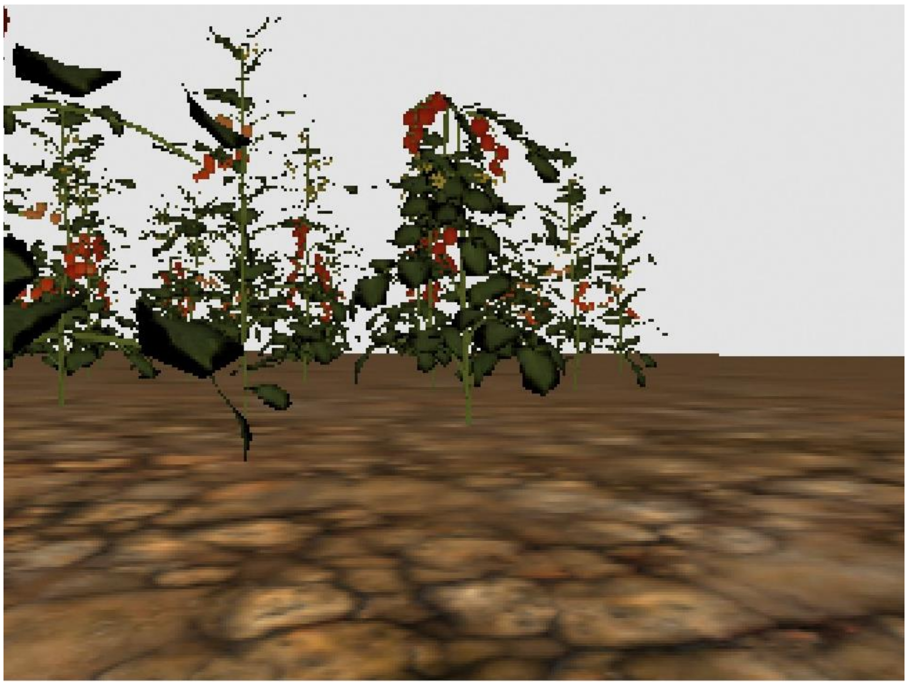

Figure 2: Pixelated camera view from the virtual robot.

4.2 Image recognition with YOLOv7

In its most basic form, image recognition is a classification issue, but more hard and meaningful is object localization and recognition, semantic and instance segmentation, and key point detection[4]. In image classification a model is trained to distinguish between certain types of images. A basic question is answered, namely if an image contains a certain object, like a cat, a bicycle or a tomato. Multiple classes might be assigned to the same image. More advanced is object localization and recognition, also called object detection. In this case a square bounding box is drawn in the image. Multiple bounding boxes can be drawn around objects, allowing the network to count the amount of instances of that object within the image. In semantic segmentation we assign classes to the pixels in an image. In other words, does this particular pixel of the image belong to a certain type of object? If a network is trained to recognize people it should assign each pixel that is part of a person to the object type person. This is regardless of the amount of people in the image. In instance segmentation the amount of people is also taken into account. Each instance is segmented and thus we can identify the object and count the times it occurs within the image. Lastly, key point detection identifies important parts of the object that is being detected. In the case of humans these parts could be the head or the hands. Between these key points lines can be drawn to compose a rough figure that represents a human body and its movements.

For detection of tomatoes in greenhouses it can be sufficient to use a bounding box approach. Using this we can estimate the amount of tomatoes and determine their position. However, semantic segmentation offers the benefit of knowing the exact position of the tomatoes contours. In general it is advisable to simplify the problem that the network you are training is faced with as much as possible. Machine learning and AI are usually only good at very specific tasks and can get confused when confronted with data that deviates from the norm. The norm being all the images in the train set. Because most novel greenhouse applications should function using a bounding box approach, we would recommend to only focus on this, unless there are good reasons to do otherwise.

That being said, modern image recognition networks are perfectly capable to perform basic instance segmentation tasks. If enough train data is available and performance is therefore at least equal, instance segmentation might be preferable. Notice how in figure 3 from section 4.3 semantic segmentation is performed simultaneously with a bounding box approach.

YOLOv7[5] was chosen as image recognition network. It was released recently. It had the best real time performance when compared to other frequently used state of the art detection models as shown by the authors. They reported an average precision of 56.8% at 30FPS on the MS COCO[6] dataset. In a greenhouse setting a good real time performance is valuable. A quick detection leads to less time being wasted before continuing to a next plant. However, an accurate detection remains essential to the process. In practice this might result in the choice of a slower, but more accurate model.

The first version of YOLO[7] initially became popular due to its efficiency. As opposed to other networks at the time they used an approach which only assessed each part of an image once, hence the name You Only Look Once. The picture is divided into a grid and for each box in the grid it is decided if it contains the center of an object or not. If so, it will also give the coordinates of the center, the corresponding width and height, and the identification label of the object. The sizes of the grid used can differ between networks. In later version it is also possible to do multiple passes over the image with different grid sizes. The predictions for each time the image is scanned need to be averaged or used as input for a new layer in the network that does this via an additional convolutional layer.

YOLOv7 improved its predecessors on several fronts. Namely the use of E-ELAN, model scaling and bags of freebies. These improvements are discussed below.

4.2.1 E-ELAN (Extended Efficient Layer Aggregation Network)

E-ELAN is an improvement on ELAN. During the release of the YOLOv7 the paper containing details about ELAN was not yet released. However, a reference to the paper was given with an anonymous author. The paper, titled "Designing Network Design Strategies Through Gradient Path Analysis"[8], was later released under the same lead author as the YOLOv7 paper.

Model scaling is often done by adding computational blocks that are aggregated to make a prediction. ELANs main purpose was to solve the problem of deteriorating convergence when performing model scaling. Normally, convergence is the moment a model is stable and further training won't significantly affect the parameters. Here it means that when adding blocks, the deeper model eventually becomes less accurate than its shallower precursors with less blocks. When adding more blocks the models accuracy gains become smaller, until the amount is too high and this critical point of deterioration is reached. The authors refer to YOLOv4[9], where in the p7 version of the model gains only little accuracy, despite being significantly more intensive to train. By taking into account the shortest and longest gradient path in each layer of the network ELAN was designed. It is a layer aggravation network specifically designed to connect the computational blocks in a network. It takes features from VoVNet and CSPNet.

ELAN will reach a stable state according to the authors. But adding additional layers might destroy this stable state and decrease the utilization of parameters. Therefore E-ELAN was introduced. It does not affect the gradient path. It expands, shuffles and merges the cardinality of the network, which enhances the ability of the network to continuously learn. In this context the cardinality refers to the number of parallel paths in the network. Further information on the exact architecture on ELAN and E-ELAN is irrelevant to this thesis, but can be found in the previously mentioned papers.

4.2.2 Model scaling for concatenation-based models

Different models have different requirements. For example inference speed, accuracy and resolution. The depth and width of the model are to be adjusted depending on the requirements. Depth refers to the amount of layers and width is the amount of neurons in those layers. When upscaling by adding an additional p6 layer for example, the width increases, as well as the depth. This makes it difficult to analyze different scaling factors, since the network is also affected in other aspects that might affect hardware usage. The authors propose a design that maintains both optimal structure and design. This is achieved by scaling the width of the transition layers according to the change of the depth of the computational block.

4.2.3 Trainable bag of freebies

Yolov7 makes use of a so called trainable Bag of freebies. This is not part of inference (the use of the trained model) but it only affects training. A freebie can be as simple as data augmentation, as demonstrated in YOLOv4. A freebie increases model performance, without affecting training cost.

One of these techniques is model reparameterization. This technique averages the trained model weights to get a better model. We can distinguish between model level reparameterization and module level parameterization. In the first approach multiple identical models get trained with different train sets, the average model is the final model with increased performance. It is also possible to take the average weights from different epochs. In module level reparameterization the training gets split in different modules. Again, the average is the final model. The authors of YOLOv7 employed multiple ways of model reparameterization.

The authors also introduce "coarse for auxiliary and fine for lead loss" to YOLO. What they mean by this is that different labels are used for the different heads of the network. The final output of the model is the lead head of the network and this uses fine labels . The auxiliary heads, called P3 to P6, use coarse label assignments that fit their grid size.

Other freebie techniques were used as well during training for the original paper. But since these were not novel techniques they were elaborated on in the appendix and only mentioned in the main paper. The first of these techniques was connecting the batch normalization layer directly to the inference convolutional layer. This results in the incorporation of the mean and the variance from the batch normalization into the weight and bias parameters of the inference stage. The second is implicit knowledge as described in YOLOR (You Only Learn One Representation)[10] in combination with a convolutional feature map. Explicit knowledge is defined as knowledge derived from an input that is presented as the problem to solve. For example a classification problem; does this image contain a cat or a bike? An input image goes through the deep neural network and leads to a classification. Implicit knowledge is compared to human memory and experience. We use past experience to solve the current problem. In a neural network this can be implemented through rough feature extraction. By using implicit knowledge it is easier for a network to adapt to a new task. The third technique is Exponential Moving Average (EMA) as shown in the Mean Teacher method[11]. Considering that the Mean Teacher itself is not implemented in YOLOv7 we will not expand on it too much. The main take away is that two models are trained simultaneously, a student- and a teacher model. The student is optimized for classification and consistency with the teacher. The EMA is the measure of the change of the parameters in the student model. EMA is introduced to smoothen out noise in the data. In YOLOv7, EMA is employed in the final inference layer.

4.2.4 Final remarks on YOLOv7

It is important to note that YOLOv7 only takes square images as input. If different dimensions are used the model will resize the picture so that the longest side will be reduced to 640p and the edges of the shorter sides will be filled with grey[12].

There are different version of YOLOv7, that each have a slightly different architecture. We use the default YOLOv7 P5 version. The main difference with the P6 version is the size of the output feature maps and the input size of the images. P6 supports input of 1280x1280p images. The stride of the P5 version is 32, as can be seen in line 10 of the yolov7.yaml file from the official YOLOv7 Github repository. The numbers 5 and 32 are related to each other, as 2^5 is 32. Similarly, P6 has a stride of 64 and P4 a stride of 16. The P5 version of the YOLOv7 model includes P3 and P4 feature maps as well, but P6 is reserved for YOLOv7 P6. With a stride of 64 in a 640x640p image we get a grid of 10x10. This is likely too large for an image of this size, as it was not included in the P5 model.

The main code in the official Github repository is for object detection, but instance segmentation is also available in the u7 branch. Instance segmentation defines more accurately where the object is.

4.3 Pretraining on MS COCO

Microsoft COCO (Common Objects in Context)[6] is a collection of annotated images. It can be used to benchmark new designs of image recognition networks and to pretrain a network. Pretrained networks speed up the development of networks significantly, since the network has already learned to identify important features. When visualizing inference of the MS COCO pretrained network of YOLOv7 on tomato plants we see that the tomatoes are sometimes recognized as apples or oranges. Below we see such an example. Here we did not train the model ourselves, but used the weights from pretraining only. Presumably it is easier for the network to adapt its predictions based on new information in the train set then when it is trained from scratch. It is important to be careful with drawing premature conclusions, since the inner workings of a deep neural network are complicated and not necessarily predictable.

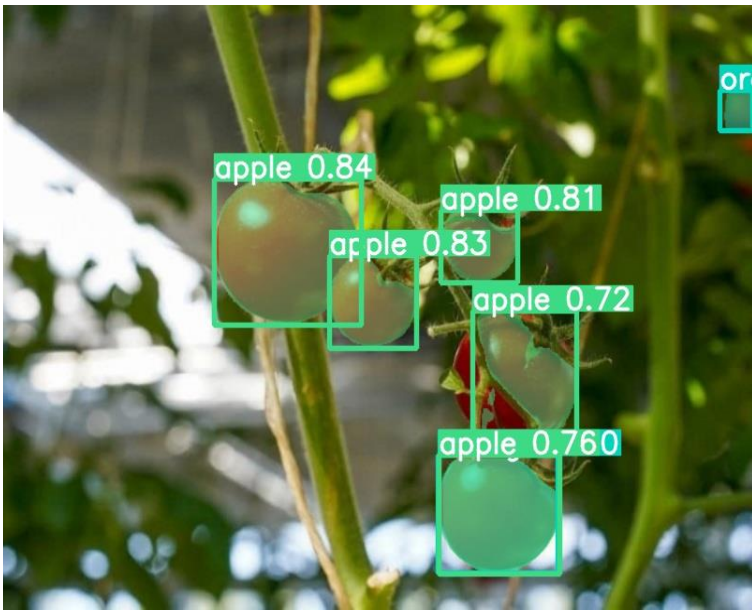

Figure 3: Inference of the MS COCO trained YOLOv7 model on tomatoes. Tomatoes are not included in this train set and therefore the tomatoes are wrongly identified as apples.

It is out of the scope of this thesis, but it is possible to learn more about these inner workings. An example is a visualization technique that is called the Grand Tour[13]. Here the behavior of the neural network is visualized to learn more about how it changes epoch to epoch. Another interesting approach is the use of an activation atlas[14]. This visualizes learnable features that the network uses to distinguish between objects. These techniques, among others, can help us to understand the network better. Without the substantiation these tools can provide we have to regard the neural network largely as a black box.

4.4 Model generated data as a replacement for real life data

Our assumption is that if an image recognition network trained on our virtual tomatoes performs equally well on its corresponding test set, as on the real tomato test set, then our virtual tomato model is realistic enough for its intended purpose. This is a loose variation on the principle that if machine learning cannot distinguish between real and synthetic data, a flawless model is created that mimics reality.

A recent survey study from October 2022[15] described how deepfake images and videos are almost indistinguishable from real ones to the human eye. This gives rise to a demand in techniques or tools that enable us to distinguish between them. However, at the time of writing it was concluded by the authors that these techniques were not up to par. A common technique to identify deepfakes is to train a network using real and fake imagery to give a binary classification, but these networks are not reliable enough and improvements were deemed necessary. The article warns that deepfakes can pose threats if they are used maliciously. The example of falsified evidence in courtrooms is given, where even experts are unable to distinguish between real and fake. It is also noted that a battle between those that use advanced techniques to create deepfakes and those that try to detect them is growing.

Another example of creating fake data was given in 2021[16]. Researchers used advanced techniques to generate artificial human genomes. Protecting privacy was one of the main goals of this research. By generating artificial genomes based on real genomes, certain research can still be performed without compromising privacy. The artificial genomes were deemed realistic based on Principal Component Analyses of genomes of a real and artificial population. No artificial genome was overfitted by the model. This would have indicated a too large similarity to the train data and a privacy risk.

More comparable to our own project is a proof of concept from 2016[17]. Video game imagery from Half Life 2 was used to train a fully convolutional segmentation network. This video game has a photo realistic art style and thus mimics reality to a certain degree. It was shown that for certain instances in image segmentation tasks on real data, the performance was similar to that of a network trained on real data. This was mainly with very distinct looking objects. Depth estimation was also successful. Additionally it was shown that model performance can be increased by pre training on the synthetic video game dataset.

From the examples given above we can learn that reliability of models can indeed be assessed using machine learning and that we are not the first to try this. The overall principle is that if humans and machines cannot distinguish between real and fake, a high quality model is created. It is important to note that in the last example photo realism was not assessed directly. If we were to train a network that needs to distinguish between video game screenshots and real life pictures, we expect the model to be able to do this. Deepfake software is far more advanced and detailed then the outdated video game graphics from the game that was used in the research. They have relatively low detail textures and polygon counts, which makes them easy to distinguish as fake. Therefore, recognizing computer generated images is far more difficult for the deepfake images then for the video game images. Similarly, we expect an image recognition model to be able to distinguish between virtual tomatoes and real tomatoes as well. As long as the model is specifically trained to do this. The experiment using video game images is highly similar to our application where we want to make digital tomato models. Ideally we will be able to generate tomato plant models that approach photo realism as well as deepfakes, but this is not a realistic goal. For our application this does not seem necessary either. The goal is to make a tomato model that is realistic enough for the purposes of testing and developing novel greenhouse systems. Photorealism is not necessary as long as performance is still equal.

4.5 TOMATO PLANT ANATOMY

There are many different varieties of tomatoes, making a single description of the plant not sufficient to capture all the differences between the different cultivars. However, we can follow general descriptions of the plant that most varieties adhere to.

The Encyclopaedia Britannica[18] states that the plant are generally branches and can become 60 to 180 cm in width, but can also be more compact. Its pinnate compound leaves are moderately hairy, emitting a strong odor and can be up to 45 cm long. The yellow flowers are small with a diameter of about 5 cm. They hang downwards and contain 5 petals. The flowers grow in clusters and later form berries that can be 1.5 to 7.5 cm in diameter. They can be spherical, oval and even pear shaped. Their color is usually red or yellow, but green and purple varieties occur as well.

The OECD (Organisation for Economic Cooperation and Development)[19] states that the plant can have an indeterminate or determinate growth, meaning that it can either have a genetically programmed end to its growth or not. It further states that the plant can be up to 3 meters in length. Its primary root can be several meters as well. The stem is angular, meaning it is not. The stem is also described as being hairy and emitting a particular smell through its glandular trichomes. These trichomes can also be found on the leaves, which corresponds to the description of the Encyclopaedia Britannica. The OECD further adds that the leaves are alternately arranged on the stem with a 137.5 degrees phyllotaxy. This means that the angle between two leaves is 137.5 degrees, which is common for plants. Leaves can be compound, consisting of 5 to 9 leaflets that are petiolated and dentated, and lobed with pinnately arranged segments. However, larger lobed leaves exist as well.

Upon inspection of a tomato plant of an unknown variety we can notice a distinct main stem with side branches. This is in correspondence to how the tomato plants are generated in the ignition-gazebo repository.

It appears that the repository generates a medium sized tomato plant with a stem that does not need support. It is important to keep in mind that not all tomato varieties will look like this sample. Furthermore, not all details that have been described are necessarily important for a tomato model. As an example, in the context of greenhouse applications it likely redundant to include the hairs on the leaves.

5 WORKFLOW AND SYSTEM DESIGN

The goal of this thesis is to develop the groundwork for a realistic tomato model that can aid the development of image recognition models for tomatoes. We can distinguish two main components. The first is generating randomized tomato plants as 3D models. This was done using a preexisting project for Ignition Gazebo where test fields of tomato plants are generated. These simulated tomato plants will then be used in the second step where we test the usability of the tomato models for the purpose of aiding the development of image recognition models for tomatoes.

Initially, the tomato generator was adapted in order to increase workability and make the proportions of the plant more easy to manipulate in the future based on real life measurements. This would demand a lot of time, as the repository lacked documentation. The part responsible for the generation of tomato plants will be extracted and included with this thesis.

Testing the performance of the two models(one trained on images of computer generated tomatoes, the other on real ones) on their corresponding test set gives a baseline performance. Then we test each model with the test set of the other. We are mainly interested in seeing if a network trained on simulated tomatoes is able to identify real tomatoes. Additionally we will also test if the size of the train sets influences the model performance and if the inclusion of the decoy images has a benefit.

Furthermore we will test the usefulness of the images containing generated tomatoes for pre training. We will fine tune the model trained on the generated tomatoes for one epoch on the dataset containing real tomatoes. If this increases model performance drastically the potentiality to use the generated tomatoes in a pre-training session is sufficiently demonstrated. This can greatly reduce train time in larger data sets. It also bridges the gap between virtual and real better. A system could be fully designed in a virtual environment before being tested in the real world. There should only be minor fine tuning required to make the system work in an actual greenhouse.

5.1 COMPONENT 1: TOMATO PLANT MODELS

From the fields-ignition repository we isolated the files that were responsible for the random generation of tomato plants. These were a Blender file, an accompanying python script and a folder containing textures. When opening the Blender file, we can run the python script and generate a tomato plant. The python script was simplified and adapted to increase workability. The end goal being that data collected from real life plants is easier to incorporate in the model. And that the script was more understandable for future users. The adapted tomato generator script is explained in detail in chapter 6. In section 6.1 a list of all manipulatable parameters is included as well.

How the softwares in the fields-ignition repository relate to each other is visualized below.

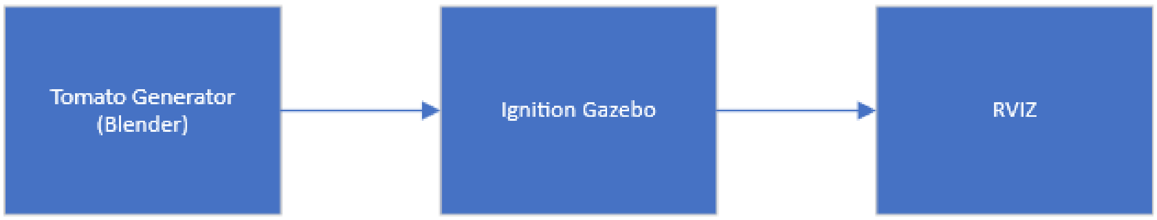

Figure 4: Software interactions tomato field simulator. The tomato generator creates randomized virtiual tomato fields. These are loaded in a digital environment through Ignition Gazebo. Here a virtual robot drives around, of which we can see the camera view via RVIZ.

The tomato generator is one of the two main parts of the project. Through a Jupyter notebook the script can be run and it generates a tomato field with randomized plants. This tomato field can then be loaded in Ignition Gazebo. This is the program that hosts the virtual test environment. Ignition Gazebo in turn communicates directly with RVIZ. This program visualizes the environment from the perspective of the robot that drives around in it. Ideally YOLOv7 is used to predict the position of the tomatoes from the perspective of the robot that is provided by RVIZ. Sadly, this part proved to be difficult to implement. The resolution of the robot camera is low, which results in a very pixelated image. Even though ripe tomatoes are clearly distinguishable to the human eye due to being red, unripe tomatoes that are green are difficult to locate and identify. The low amount of detail is not optimal either when images from the robots view are used for training a network that eventually is supposed to locate and identify real tomatoes that have a high degree of detail. Furthermore, connecting the live footage from RVIZ to YOLOv7 is not straight forward in implementation and thus not possible due to time restrictions. We can still provide a good proof of concept without this direct connection. We can run the blender file separately to generate individual tomato plants in a 3D environment. So even though Ignition Gazebo and RVIZ are used in the original fields-ignition project, we only need the Tomato Generator script from Blender.

We increase workability of the script by simplifying the code and adding commentary, something that was missing in the original repository. We also provide detailed code descriptions. Furthermore, a clear list of parameters that the user can manipulate is provided at the start of the script. These were previously hard coded throughout different sections of the script.

The manipulable parameters are listed below and further explained in section 6.1.1:

- avg_node_length
- sd_node_length
- avg_node_count
- sd_node_count
- first_node_multiplication_factor
- DIV
- avg_radius
- sd_radius
- node_radius_reduction
- lower_section_cutoff
- sd_lower_section_cutoff
- middle_section_cutoff
- sd_middle_section_cutoff
- chance_tomato_or_flower

5.2 Component 2: Image recognition

In order to measure the usability of the tomato model we rely on image recognition. As explained in section 3.3, if computer models can’t distinguish between real and fake, the model is approximating perfection. We train two YOLOv7 image recognition networks. One will be trained using images from the generated tomatoes and one will be trained on picture data of real tomato plants. We will then use the trained networks to assess performance on both their own test set and that of their counterpart. In other words, we will use the network trained on generated tomatoes to detect real life tomatoes, and vice versa. We are interested if an image recognition network that is trained on generated tomato plant image data performs well when confronted with real tomatoes. If performance is equal to that of the network that was trained on real tomatoes, we can conclude that for the purposes of tomato detection with image recognition, the tomato generation model is accurate. The models can then partly act as a replacement of real life data for this specific purpose. Although an equal performance is too optimistic due to visual differences in the tomatoes, it is expected that the network will be able to perform to at least a certain degree, as opposed to no performance at all. Despite the differences there is still a large amount of visual similarity, but the amount is unknown and difficult to quantify. Therefore we do not expect the model to be suitable as a replacement, but it can certainly aid development if performance is sufficient. The model performance can act as a form of quantification in usability. When determining realness it is more appropriate to train a network that labels images as “real” or “model generated”. Even is the resemblance is sufficient and the model is robust enough to use generated tomato images in its train set to recognize real tomatoes in a test set, there can still be obvious visual differences to the human eye, or even smaller details regarding anatomy can be off. Furthermore, there is more variety in natural plants as opposed to our computer generated ones. For example, there are only four different tomatoes, two different leaves and one flower petal. The branches are pre modelled manually and therefore there is only a limited number available, whereas in nature there are a lot more formations possible. In other words, there are only limited textures and meshes available. Meshes are the 3D shapes of objects, in our case tomatoes and branches with leaves. The textures are overlayed on these shapes and contain the colour pattern of the object.

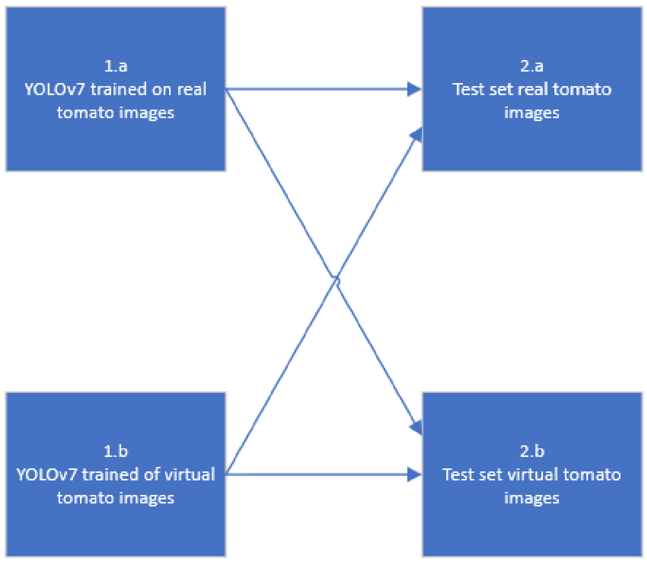
1.a
YOLOv7 trained on real tomato images
2.a
Test set real tomato images
1.b
YOLOv7 trained of virtual tomato images
2.b
Test set virtual tomato images

Figure 5: Main components involved with image recognition.

Above we can see a diagram that show the main components involved with the image recognition part of the project. Components 1.a and 1.b are the YOLOv7 networks trained on either real or virtual tomato pictures. These are then applied to a corresponding test set, as visualized with the horizontal arrows between 1.a and 2.a. And between 1.b and 2.b and the corresponding test sets, as visualized by the diagonal arrows. We will refer to these as:

- “Real tomato trained on real tomato tested.” (1.a on 2.a)
- “Virtual tomato trained on virtual tomato tested.” (1.b on 2.b)
- “Virtual tomato trained on real tomato tested.” (1.b on 2.a)
- “Real tomato trained on virtual tomato tested.” (1.a on 2.b)

The purposes of these tests are as follows:

- Testing a network with its corresponding test set provides a baseline of performance. These are “real tomato trained on real tomato tested” and “virtual tomato trained on virtual tomato tested”.
- Testing YOLOv7 network “virtual tomato trained on real tomato tested” tells us to which degree we can substitute real tomato images with images of model generated ones. This is a form of measurement of the “realness” of the tomato model.
- Testing the network “real tomato trained on virtual tomato tested” does the same as the point above, but in reverse. We measure how well a network trained on real tomatoes performs in a virtual environment.

We then assess how well the networks performed on these test sets. Ideally, the model trained on virtual tomatoes performs equally as well as the model trained on real tomatoes when applied to real tomatoes. This is the most important part of the performance assessment.

5.2.1 YOLOv7

Initially, the plan was to use Mask-RCNN[20] instead of YOLOv7. However, it proved difficult to implement. The tutorial from its original repository is outdated and the code is incompatible with the current version of Colab. Attempts to solve the many dependency issues were not successful. YOLOv7 on the other hand is well maintained and easy to implement due to many recent projects and practical tutorials. Shareability is high due to the dependencies being supported by the current version of Google Colab. YOLOv7 achieves the same end goal and is thus an easier and quicker way to generate a network for our purpose.

When training, image size was kept at the default 640p. Yolo can only work with square images, so the original picture is resized in such a way that the longer side of the image has a length of 640p and the remaining empty areas on the shorter side are then filled with grey. Batch size was set to 16. For the real tomatoes we saw that 25 epochs was enough for the model to stop learning, for the generated tomatoes this was 100. This is due to the smaller size of the dataset and presumably the lower degree of variety in the generated tomato plants. There are only 4 types of tomatoes to recognize, one for each texture, while there is a theoretically infinite amount of patterns on the tomatoes. There are also tomatoes of different varieties in the dataset containing real tomatoes, further increasing its variety in visuals.

We start training from the MS COCO checkpoint provided in the GitHub page of the original YOLOv7 article.

5.2.2 Datasets

There are two datasets being used in this thesis. The first one contains picture data from real tomatoes and was found in the Roboflow library. The second one was made by taking screenshots of model generated tomatoes. They were uploaded to Roboflow and annotated using the tools they provided on their website.

We can highly recommend using Roboflow. There is a large library of annotated images available for different kinds of applications, both for bounding box prediction as for semantic segmentation. Different ways of exporting the annotated images is possible to make your dataset compatible with a variety of popular image recognition models. Additionally, when annotating your own images the “smart polygon” function allows for a quick annotation. These tools can save the user a lot of time.

5.2.2.1 Real tomato image set

The real life tomato trainset[21] contained 1037 samples. The dataset the images were extracted from contained only 488 images, but due to augmentation steps the size of the train set was larger. Its validation set contained 93 samples and the test set 50. This is a more than sufficient size for a network that only needs to recognize one item. However, two shortcomings stand out in this dataset. Firstly, there is a lack of consistency in annotation. For some tomatoes the annotator included the part of the tomato behind an obstruction, imagining the rounding of the tomato where it was not visible, while for others they did not. Secondly, in a greenhouse we will see many unripe tomatoes. However, the dataset includes only few green tomatoes. The test set contains 88 images which contain 136 labels in total.

5.2.2.2 Generated tomato image set

The generated tomato image set[22] was developed for the purpose of this thesis and can be found on Roboflow as well. Pictures were obtained via screenshot from Blender in different image ratios. As explained earlier, it was not possible to use RVIZ due to the low image resolution. This would lead to a poorer performance when real tomato pictures are analyzed by the generated tomato trained network. Taking screenshots from Ignition Gazebo would give little more benefit as it also lacks background, only soil is visualized. Background might confuse a network if certain aspects are similar to objects that are being detected. By including images with background we can increase the robustness of the model. Taking screenshots in Blender proved more practical, as it is easier to isolate individual plants to annotate. The lack of soil as background noise should not make too much of a difference, since it only adds little noise. Instead, we can also use decoy images that only contain background and no labelled objects.

All screenshots of the generated tomatoes contain the same kind of background, and thus there is little for the model to get confused about. In order to mitigate this issue we could include decoy images. These are random pictures containing no tomatoes at all. It is hypothesized that the network will be forced to adapt to the background noise in these images. If a random object in a decoy image is predicted to be a tomato this will result in negative feedback and an adjustment of the parameters in the network. It was made sure of that some of the decoy parameters include vegetation, as this is likely abundant in the greenhouses the network would be employed at.

The obvious disadvantage of the dataset with model generated tomato plants is its lower degree of variety. There is only a relatively small number of side branches and the anatomy of each plant is highly similar. More importantly, there are only few textures available. The size of the dataset is also smaller. Furthermore, annotation proved difficult as well due to the same kind of inconsistencies encountered in the real life tomato set.

A relatively high degree of confidence could likely be achieved by simple color filtering. This is a technique that is used when working with satellite data. Satellites often make use of cameras that are able to capture multiple color bands. We see only three colors as humans, but these satellites also capture other colors like ultra violet and infra-red. Using these color patterns the cameras are able to identify different types of terrain and vegetation. Similarly, we can filter for red colors only and thus identify a large amount of tomatoes without the use if image recognition. When analyzing the weights in a custom trained network it is likely that there is a large emphasis on the color red. As opposed to simple color filtering however, a neural network is better suited to exclude background artefacts that just so happen to be red. Also, when dealing with partial obstructions it is more likely that a neural network can draw good bounding boxes.

It would be ideal if the entire tomato was clearly visible, but in reality there is often partial obstruction. These obstructions can be relatively easy to work around if only parts on the side of the tomato are covered. However, there are plenty of examples where a branch divides the tomato in two. Like is seen on the left. Here, the annotator needs to somehow incorporate the side of the tomato, but in such a manner that the least amount of unnecessary pixels get incorporated in the annotation. In Roboflow it is not possible to have more than one field count as the same annotation, so a small “bridge” needs to be made in order to incorporate the entire tomato. However, if only a very small part of a tomato is separated from the rest by an obstruction, the annotator might decide to leave that part out, since including the twig will add more non tomato pixels then tomato pixels to the annotation. In the middle we see an example where almost the entire tomato is obstructed. An approach could be to let the annotator estimate the rounding of the tomato. If done consistently the network will be trained to predict these roundings as well. Here it was eventually decided to leave the tomato out of the annotation. Most of the tomato was obstructed, there was no clear place where a bridge could be made and loosely predicting the rounding of the tomato was inconsistent with the rest of the annotations. A different case is presented on the right. Here, a flower obstructs the tomato. It is important to not accidentally include part of the flower in the annotation, since it is a relatively rare object that might confuse the network. Also, from early training we knew that the flower sometimes gets recognized as a tomato. Including it, even partly, might reinforce this mistake.

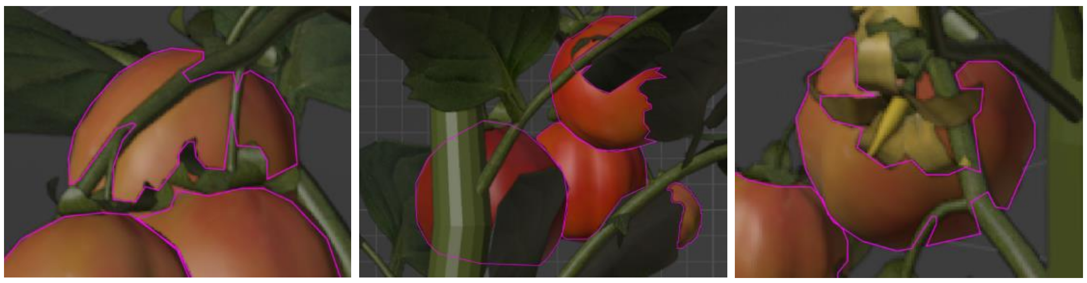

Figure 6: Common problems and consideration during annotation.

For the purposes of this thesis the way the annotation was done is sufficient. We are able to compare the networks fairly and learn something about the usability of the tomato model as a substitute for real life data. That being said, the previously mentioned inconsistencies in annotation are a shortcoming. For future research it is recommended to find a way to annotate pictures in such a way that two polygons can be attributed to the same object. If there is no support for this an alternative approach could be using bounding boxes instead of polygon annotation. Bounding boxes should provide a sufficient amount of information for tasks like crop counting or targeted spraying. For those tasks where a more precise localization is necessary an additional network could be used to annotate the pixels that belong to the tomato within the bounding box.

There are 4 different versions of the train set. Small, medium, large and large with decoy images. The amount of images in each train set was 30, 45, 60 and 70 respectively. Due to image augmentation the size of the train sets could be increased threefold (to 90, 135, 180 and 210). The options for augmentation were a flipping horizontally and vertically, rotations between -45 and +45 degrees, and noise up to 5% of the pixels. This can be verified in Roboflow. The test and validation sets for each version remained the same. They validation set contained 67 labels, the test set 54. Both consisted of 15 images.

6 RESULTS

6.1 COMPONENT 1: IMPLEMENTATION TOMATO PLANT GENERATOR

In this section we will describe the functionality of the tomato plant generator after adaptation. It is one of the two main parts of this thesis. The code was optimized for workability and potential inclusion of parameters derived from real life measurements. In our adapted version the code contains 322 lines, whereas the original contained of 247 lines. A lot of his is commentary code that was missing from the original script. Other parts are rewritten to decrease unnecessary complexity or expanded to comply more with our goals.

6.1.1 Main description

In the adapted script the tomato plants are generated in two steps. The first is generating a main stem consisting of nodes. In the next step a side branch is attached to each node. These side branches are taken from a collection of pre-modelled branches. The way these side branches look is therefore not random, only the selection is. In the figure below the main branch is clearly visible in the center vertical axis. A few nodes are marked with a blue circle. Notice how for each node there is a corresponding side branch.

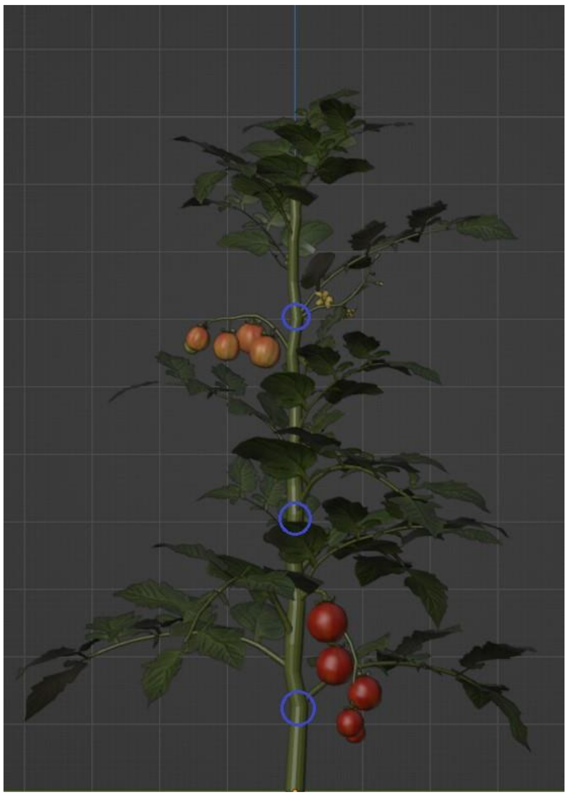

Figure 7: A generated 3D model tomato plant. Some nodes are encircled with blue.

In contrast to the original script, clear distinctions are made between different kinds of side branches. Namely tomato branches, flower branches, large leaf branches, small leaf branches (not used), and lastly normal branches called leaf branches. Small leaves are not used because they are only small and freshly sprouted buds. To include these in the model we would need to determine specific places where they would be appropriate to attach to the main branch. This would overcomplicate the script, without adding a lot because these buds would be badly visible in the final model. Also, there are only 3 small_leaves objects available and this does not add to the diversity of the plants if included. In the figure above different kinds of branches are visible. Some contain tomatoes or flowers, others only leaves. Notice how the lower branches are larger than the ones at the top of the plant.

In the script all parameters that can be changed are given in the beginning of the script. This makes changing the physiology of the plant easy. Examples include thickness of the main stem or at which height flower branches should occur. Based on real life measurements averages and standard deviations can be determined and used as model input to mimic the real life counterpart plants. Keep in mind that the plants that are being measured are of the same tomato variety and have to follow the same basic anatomy. They should also be grown in similar conditions for the same duration of time. If not, the average node lengths for example will definitely differ. There will not be a standard distribution for the averages and the resulting tomato plants will not be true to reality.

The parameters that can be adapted at the beginning of the script are listed below. They were chosen because they are easily adaptable and represent all the basic measurements of the plant. By providing an average measure paired with a standard deviation it is possible to mimic the natural variation in these measurements in the simulation. Also, this was all that could be done with this script in the given time frame to improve realness.

- avg_node_length = average node length
- sd_node_length = standard deviation of node height
- avg_node_count = nodes are places in the stem where a branch can appear
- sd_node_count = standard deviation of node count
- first_node_multiplication_factor = the first node can be longer to simulate the initial stretching phase of the sprout
- DIV = amount of edges the main stem has
- avg_radius = average radius of the first node
- sd_radius = sd of the radius of the first node
- node_radius_reduction = the reduction factor of how fast the stem becomes narrower towards the top
- lower_section_cutoff = the lower section of the plant contains only larger low_leaves
- sd_lower_section_cutoff = sd of which node is the cutoff for the lower section
- middle_section_cutoff = the middle section contains both tomatoes and leaves, the upper leaves and flowers
- sd_middle_section_cutoff = sd of which node is the cutoff for the middle section
- chance_tomato_or_flower = the chance that a node in the middle or upper section contains fruits or flowers

The textures and meshes of the side stems are not easily changed for someone unfamiliar with blender. In the original script we can recognize some of these variables, but often they are not static and not meant to be changed by the user. However, it is easily adaptable to other crops if the right meshes and textures are available. Meshes are 3D shapes and textures are the images that are overlayed on these shapes. Meshes can be seen as virtual statues and the textures as the paint. Together they form a 3D object. The new plants still have to grow in a similar way to tomatoes. Cauliflowers would not follow the basic physiology of the tomato plants, but bell peppers are likely similar enough. We can simply add side branches that we modelled after bell peppers to the main stem.

A clear shortcoming of the tomato generation model is the fact that the side branches are pre-modelled and not randomly generated. The limited number of side branches available restricts the potential diversity of the plants. This could be improved upon by including a separate generator for side branches in the script, but that is out of the scope of this thesis. Other examples of this lack of diversity is the fact that there is only one kind of flower, only four tomato textures for different degrees of ripeness, and only two textures for leaves. This can be seen in the textures folder.

The standard parameter sizes can be found in the tomato_gen.py script. They are not based on real measurements, but rather on what seemed to work well visually. In other words, does the plant resemble tomato plants when compared to images of tomato plants. It was not possible at the time of thesis to obtain good measurements due to tomatoes being out of season.

6.1.2 Tomato generator script in more detail

The tomato_gen.py script contains a large amount of commentary code to help the user. However, more guidance is provided here.

- If the user only wants to change basic parameters, they can change the values of the parameters between line 30 and 51. These are all the parameters that were given above.
- The script makes use of a custom class called Node. It contains the x, y and z coordinates of the node, its yaw (or angle) and a value called r which determines the radius of the node.
- Subsequently a list of nodes is generated using the basic input variables from the list above. The yaw can't be influenced by changing the parameters. If this needs to be changed the user can do that here. The reduction of the radius of the nodes towards the top is coded here as well.
- The next part is the create_ring_verts() function. Ring vertices determine the shape of the plant. The more vertices, the rounder the stem. If there are only 3, the stem is triangular, if there are 4, the stem is square. This can be influence by the user via the DIV variable.
- In the gen_main_stem() function the gen_nodes() and create_ring_verts() functions are called. It delivers a main stem to the user as an object in python. This part of the code is more complicated to work with and it is recommended to read into the Blender Python (bpy) packages in order to understand it. In this function a conical shape is also added. It is used to close the top of the stem. It uses DIV, but can't be manipulated by any other parameters. If the user want to change the height of the cone, the script can

be adapted here. The node length of each individual node is taken from a random distribution, as can be seen in line 75. The average node length and its standard deviation are the relevant parameters. The absolute height of the new node is calculated and added to a list, resulting in a list containing increasing numbers that represent the top of each node. Note that in line 76 and 77 the first node is multiplied with the first_node_multiplication_factor. This is to simulated the initial stretching phase of a seedling.

- In the PrebuiltMeshes class the get_end_stem_mesh() function is given. Here the selection of the premodelled side branch is determined. Between the line 196 and 209 this selection is carried out. If the user wants to change something about the selection of side branches, they can do this here. Examples may include adding own custom side branches, or adding the small leaf buds that were previously discussed.
- The following function is called gen_end_stem(). It adds the end_stem (or side branch as we refer to it) to Blender. This part of the code is also more difficult to work with. It is also not necessary, as all manipulation to the plant described in this thesis can be performed elsewhere.
- The last function is gen_plant(). It uses the functions described previously to generate the plant in Blender. A lot of Blender Python is used here. Again, it is recommended to know more about Blender and the Blender Python package before working with parts of the code that use this.

6.1.3 Examples of usage

When running the script as provided we get a default tomato plant. An example of a possible outcome can be seen in figure 8 . The script is run in Blender by clicking the play button at the top of the screen in the Layout:001 window from Blender. We see four perspectives at the bottom of the screen, but the Texture view gives the best rendering.

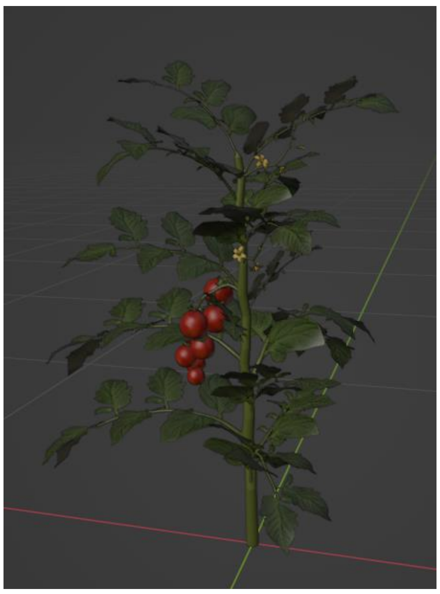

Figure 8: An example of a model generated tomato.

Suppose we want a more dense tomato plant. In order to accomplish this we need more nodes and thus open the tomato_gen.py file and change the avg_node_count parameter in line 36. We also need to take avg_node_length into account. If we do not change this we will end up with a higher plant that is equally as dense as it was previously. Suppose we change avg_node_count from 15 to 25 and avg_node_length from 0.05 to 0.03. On average this will result in a plant of 0.75 in length in both cases. We then run the model again. Do not forget to save the changed code before doing this. The result is the tomato plant displayed in figure 9.

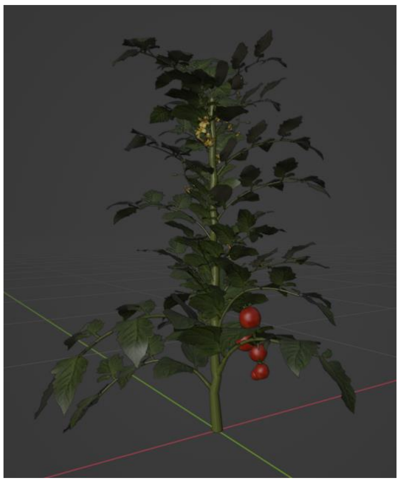

Figure 9: Example tomato plant with more nodes, but shorter internodes lengths.

The plant is more dense then it was previously. However, we only see tomatoes at the bottom of the plant and flowers start relatively low. If we want flowers to only occur at the top of the plant we can change the middle_section_cutoff value. We increase its value from 11 to 20. We also change the chance_tomato_or_flower value from 0.4 to 0.8 to increase the amount of fruits and flowers. The result can be seen in figure 10.

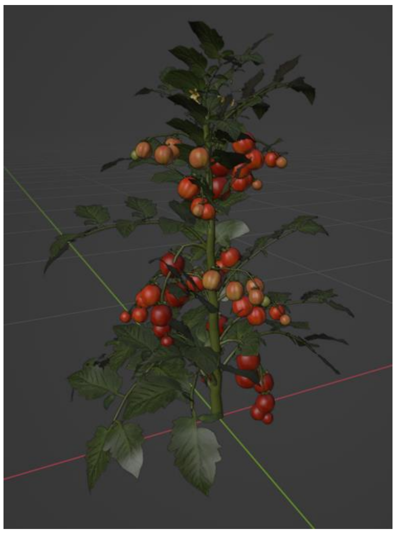

Figure 10: Example tomato plant with increased amount of tomatoes.

We end up with a plant that seems too dense in its amount of fruits. In its current state, we cannot change the script so that we only decrease the amount of fruits and not the flowers.

Instead we decrease the chance_tomato_or_flower variable to 0.6. We also want a slimmer main stem. In order to do this we change the avg_radius from 0.015 to 0.0075. Note that this is the measure for the radius of the first node. The other nodes become slimmer towards the top depending on the node_radius_reduction value. The end result can be seen in figure 11.

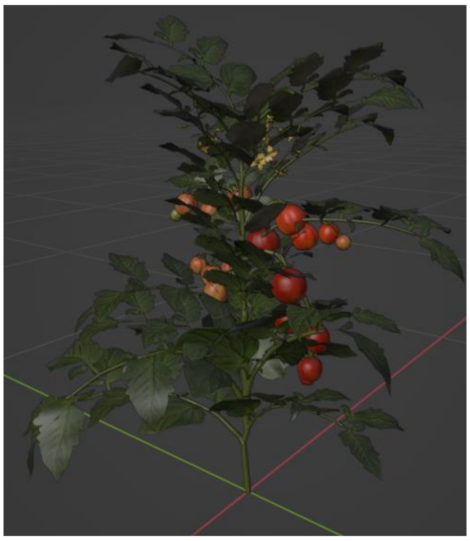

Figure 11: Example tomato plant with decreased tomato count and a more narrow stem.

These changes were applied to showcase the possibilities of the script and were not intended to result in a more realistic tomato plant. In order to do this it is recommended to base the values that can be adapted on real life measurements. That being said, it is still possible to use these images in a trainset for an image recognition model. The plants contain a lot of tomatoes and leaves, providing a lot of visual examples the model can use to learn. So despite not being based on real life measurements, the plant model could potentially be used for the purposes of training image recognition networks.

6.1.4 Future improvement tomato generator

In order to improve the realism of the tomato model it is recommended to take measurements from actual tomato plants. These measurements were not available. But when they are, it might be that an adaptation of the script is necessary. There are plenty factors that can be influenced by the parameters that were given, but it is possible that these are not sufficient to capture the entire physiology of the plants. For example, the first part of the stem towards the first node is longer than the rest. The lengths is multiplied with the first_node_multiplication_factor, to simulate the stretching phase of the plant. This is when a seed sprouts and elongates its stem to let the germ leaves capture as much sunlight as possible and potentially outcompete other plants. However, it might be discovered that internode length becomes increasingly shorter from the ground towards the top. If this is true it might be wise to formulate a function that captures this hypothetical phenomenon, in which the stretching phase could also be incorporated. An adaptation of the code in gen_nodes() would be necessary.

Improvement can also be accomplished by adding more side branches to the model. Ideally, these are based on actual side branches of real tomato plants or generated at random. When adding the side branches by pre-modelling this can be done by manual design or by 3D image scanning. If this process can be automated it might be possible to acquire a large library of side branches that negates the need for random generation. It would be necessary to determine the node or height of the side branches as well while taking samples. Usually, lower branches tend to be bigger and applying one of these to the top might result in an unrealistic looking plant.

However, if in a future project it is determined randomly generating side branches is preferable or more feasible, then the main stem generator can be reused. Similarly to how we add side branches to the main stem, we can add leaves, fruits and flowers to a side branch. If we change the parameters that were used to create the vertical main stem, we can turn it into a horizontal side stem and even determine a way to randomly attach additional side stems to the main side stem.

These improvements to the realism of the plant model might be redundant if its sole purpose is to serve as an aid in development of greenhouse applications. It depends on the exact application that is developed or tested. Where factors like leaf density might influence the ability to detect all tomatoes on a plant, the randomness of the side branches might not. For our method of assessing the realness of the model, namely by measuring image recognition network performance, it is unlikely that a realistic node length will be of added value. However, generated tomato plants that are as close as possible to their real life counterparts might add value to the novel greenhouse techniques that one might intend to test using these models. This added value might be important to potential customers. The concept of an essentially perfect model might be harder to sell if the customer in question clearly sees that the proportions of the tomato plants are off. In addition to this, it is possible that unforeseen issues arise when using an imperfect model.

6.2 Component 2: image recognition performance

Colab was used to train and test all networks. This makes the code easily sharable and reproducible. We used a base network that was pretrained on the COCO dataset. All networks were sufficiently trained, as no more improvement was seen in later epochs. This can be verified by examining the train data that can be viewed in the Colab environment.

6.2.1 Expectations

Here we discuss the performances of the YOLOv7 networks. Our main interest is whether or not a network trained on model tomato images is able to perform well when confronted with pictures of real tomatoes.

We are also interested in:

- The effects of train set size.
- The incorporation of decoy images.
- The advantages of fine tuning the pretrained network.

It is expected that a larger train set size will increase performance on a corresponding test set, meaning that if the train and test set both contain model generated tomato plant images, or both contain real tomato plant pictures, more training data will lead to better performance until a certain threshold is reached. However, when we want to apply the model tomato trained network to real tomatoes we might see a loss in performance due to a form of overfitting. In other words, the model might be more robust when fewer images are provided. This robustness can hypothetically be increased by the incorporation of the decoy images. These are chosen at random and do not contain tomatoes. It is difficult to predict the effects of these decoy images. The added robustness might make the model more hesitant to label a tomato as being one, which decreases performance. Especially in a greenhouse setting where most round objects are tomatoes. But it might increase performance as well due to a more generalized approach to recognizing a tomato. Lastly, we expect that finetuning is faster than training the model from the ground up. However, we do not know if this is worth the effort. We will attempt to quantify the benefits of finetuning the model tomato pretrained network with images of real tomatoes.

6.2.2 Performance metrics

Performances are measured with precision (P), recall (R), and mean average precision (mAP). Precision measures the amount of true positives and recall the amount of positives that were retrieved from the total amount of positives. In other words, precision measures the positive predictive value and recall the sensitivity. Mean average precision measures the average precision for a number of instances and multiple classes at a certain Inference over Union threshold. This is a measuremen for the correctness of the bounding box. Considering we only have one class, it gives the same value as the precision measure if it were to use the same IoU threshold. The IoU used to determine P and R was 0.65. Because the network was trained to optimize for an IoU of 0.65 we expect the best performance around this threshold. The value of mAP@.5:.95 refers to the mean average precision at IoU values ranging from 0.50 to 0.95.

Steps of 0.5 are used to calculate this metric. It is more informative, since performance where higher precision is demanded is also taken into account.

6.2.3 Results

The naming of the different trained networks is structured as follows:

- Real refers to the network trained on real images of tomatoes and tomato plants and generated to the network trained on the model generated tomato plant images.
- Small, medium and large indicates which train set size is used. There was one train set for the real tomato plants, so this is not applicable here.
- Decoy indicates if the decoy images were included. This is counted separately from the images of tomato plants in the train set.
- Best and last refers to which epoch is being measured. The last epoch does not always have the best performance, so we assess both.

6.2.3.1 Raw performance metrics

In the table below we can see the performance metrics from all test runs. Considering the information density, relevant results are visualized in the following sections.

Figure 12: Table containing the results of all test runs.

Network version	**Test set model generated tomatoes**			**Test set real tomatoes**		
	P	R	mAP@.5:.95	P	R	mAP@.5:.95
Real Best	0.947	0.821	0.681	0.925	0.912	0.839
Real Last	0.948	0.881	0.673	0.909	0.882	0.821
Generated Small Best	0.969	0.940	0.637	0.557	0.632	0.285
Generated Small Last	0.939	0.925	0.601	0.576	0.647	0.350
Generated Medium Best	0.969	0.925	0.652	0.674	0.639	0.370
Generated Medium Last	0.953	0.925	0.614	0.708	0.713	0.504
Generated Large Best	0.954	0.925	0.639	0.574	0.625	0.324
Generated Large Last	0.928	0.955	0.613	0.567	0.618	0.344
Generated Large decoy Best	0.867	0.879	0.607	0.683	0.698	0.473
Generated Large decoy Last	0.925	0.925	0.586	0.602	0.816	0.473

6.2.3.2 Model performances on corresponding test sets

The graph below shows model performances on corresponding test sets. They can also be found in the graph from the previous section. They are the “Real Best” and “Real Last” networks on the “test set real tomatoes”, and all the other networks (where the first part of their naming is “Generated”) on the “Test Set Model Generated Tomatoes”. The models trained on real tomatoes perform better on their own corresponding test set than the model generated ones on their test corresponding set. Especially if only mAP.5:.95 is considered. Precision and recall is overall good for all networks and about similar for both networks and their respective test sets. Precision and recall are optimized for by the network and have a threshold of 0.65 IoU. Despite the higher mAP.5:.95, we cannot say with certainty that the real tomato trained network outperforms the model tomato trained network. It is possible that the test set of one is easier to solve then the other.

There is no increase in performance due to a larger train set in the model tomato trained networks.

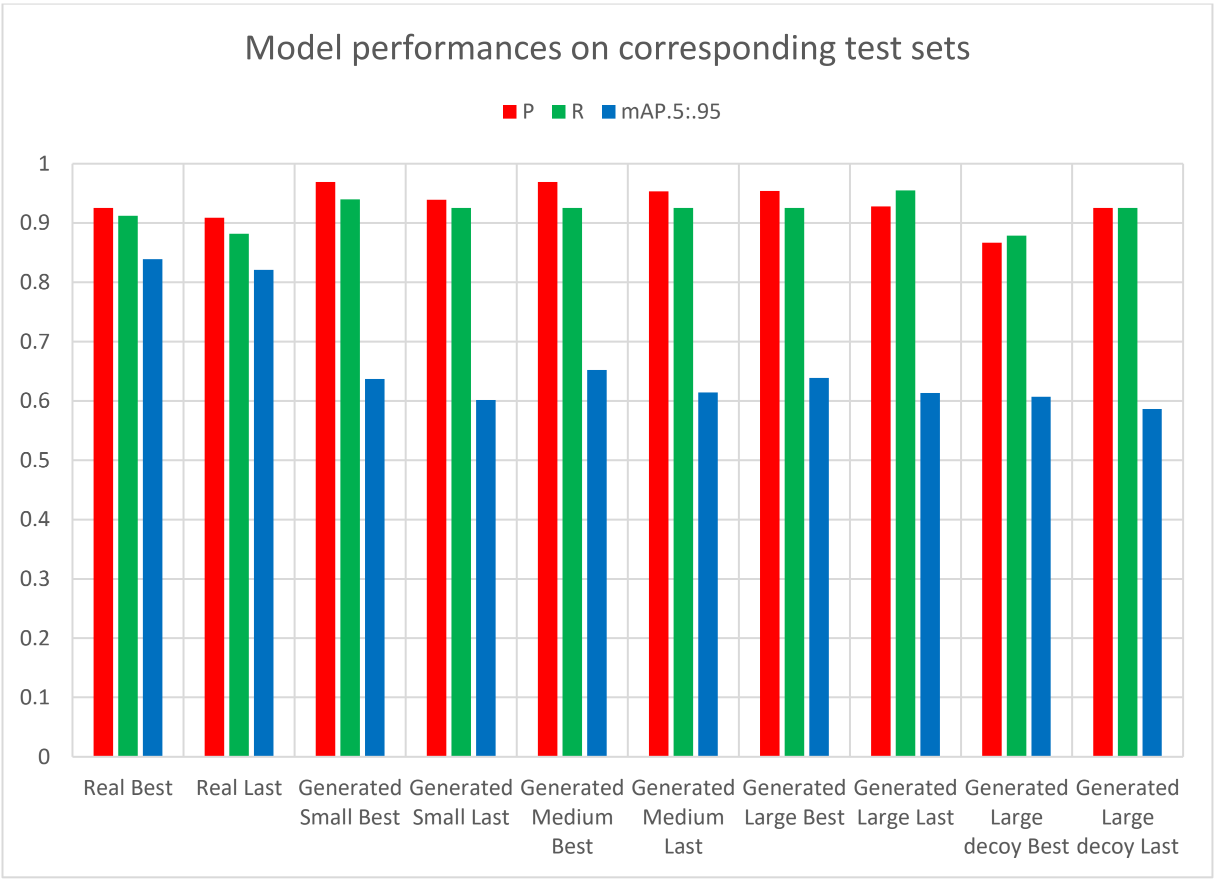

Figure 13: Results model performances on corresponding test sets. The exact values can be found in figure 12.

6.2.3.3 Performance trained networks on uncorresponding test sets

In the second graph we see the model performances when inferring on the uncorresponding test sets of the networks. Again, we see that the networks trained on real life tomato pictures perform better in comparison to the model tomato image trained networks.

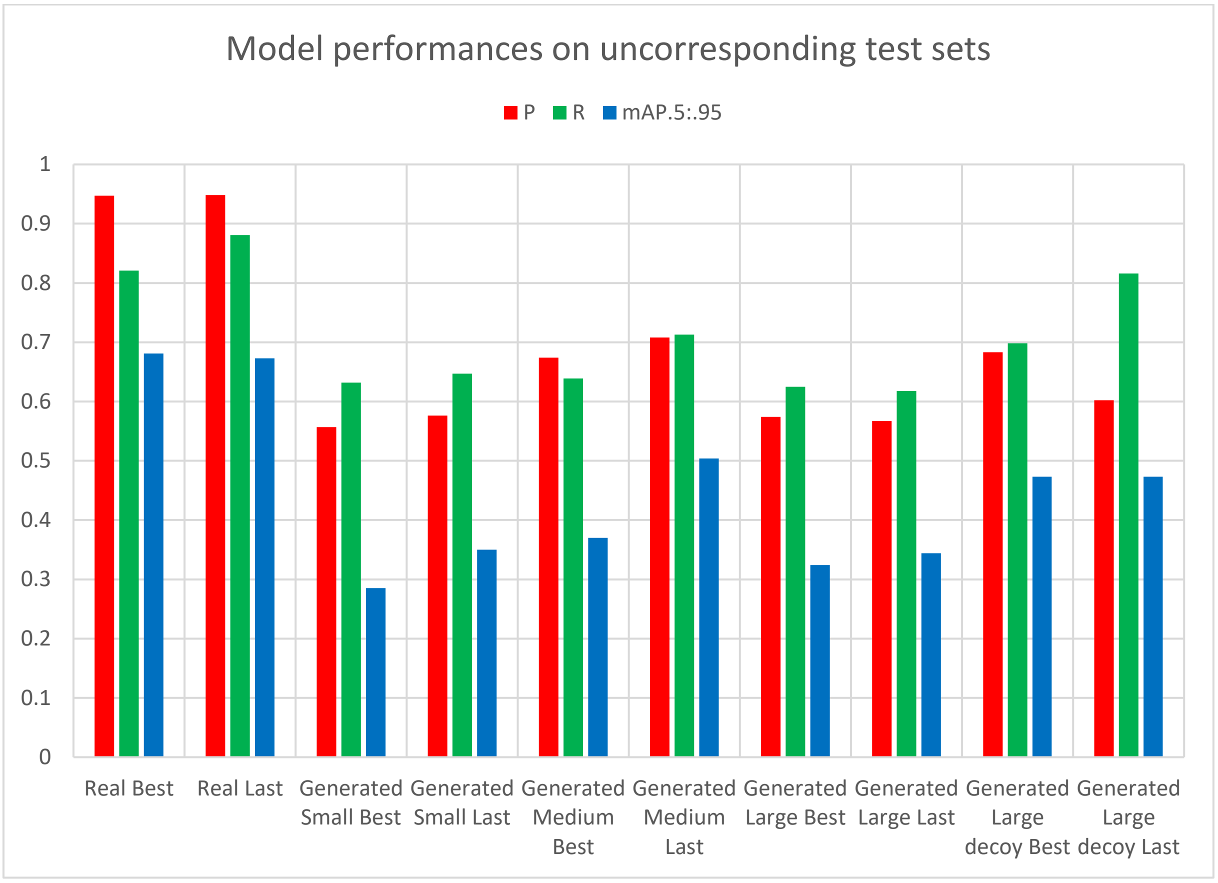

Figure 14: Model performances on uncorresponding test sets. The exact values can be found in figure 12.

6.2.3.4 Effect train set size on performance when using uncorresponding test set

We see that for the networks trained on generated tomatoes the increased size of the train set seems to affect performance. The medium test set outperforms the small one. However, performance drops in the large test set. In the graph from the previous section it can be seen that this holds true for the precision as well, while the difference in recall are less pronounced.

These effects are likely due to overfitting. Initially the performance rises due to the advantage of more train data. Because we test on an uncorresponding test set the network from the last epoch has seen more training and performs better than the network from the best epoch. The best epoch is the one where performance on the corresponding validation set was highest. Then performance drops, because the network is trained on generated tomatoes only and due to the abundance of train data the network is very strict in its judgement towards what a tomato looks like. For example blemishes on the fruit might confuse the model.

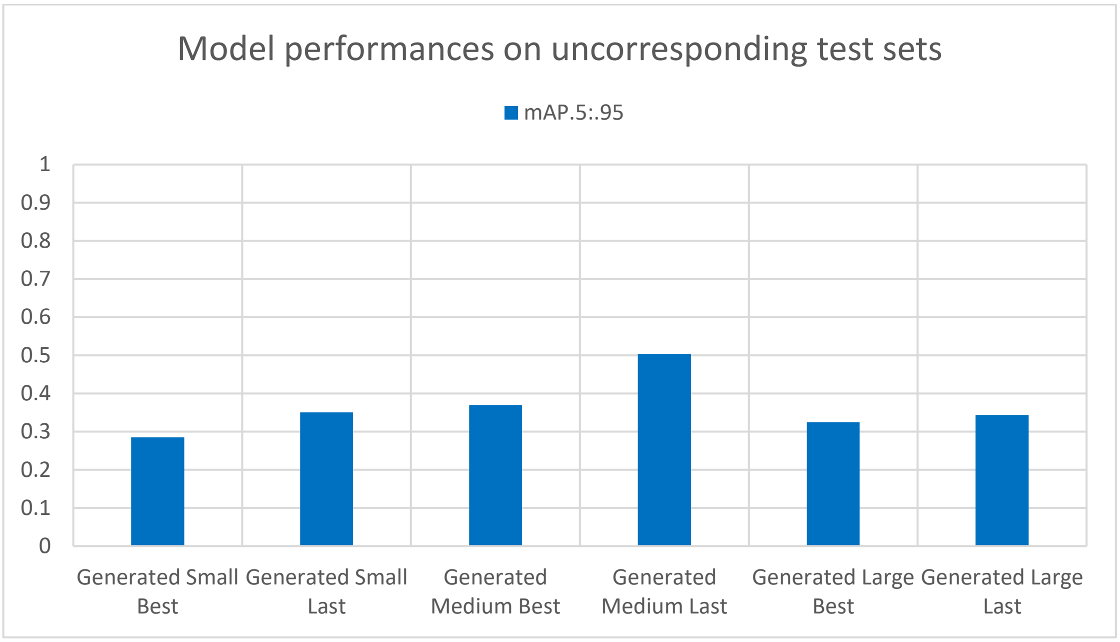

Figure 15: Model performances on uncorresponding test sets. Only mAP.5:.95 is shown for the networks trained on small, medium and large train sets to highlight the effect of train set size on performance. The bars represent the same values as the corresponding blue bars in figure 14 and the exact values can be found in figure 12.

6.2.3.5 Effect decoy images on network performance

In the graph below we see a clear rise in performance when decoy images are included. The mAP.5:.95 is higher, as well as the precision and recall, as can be seen in the table in section 7.3.1 or the graph in 7.3.3. The inclusion of decoy images makes the network more robust. Which leads to a better performance in the uncorresponding test set.

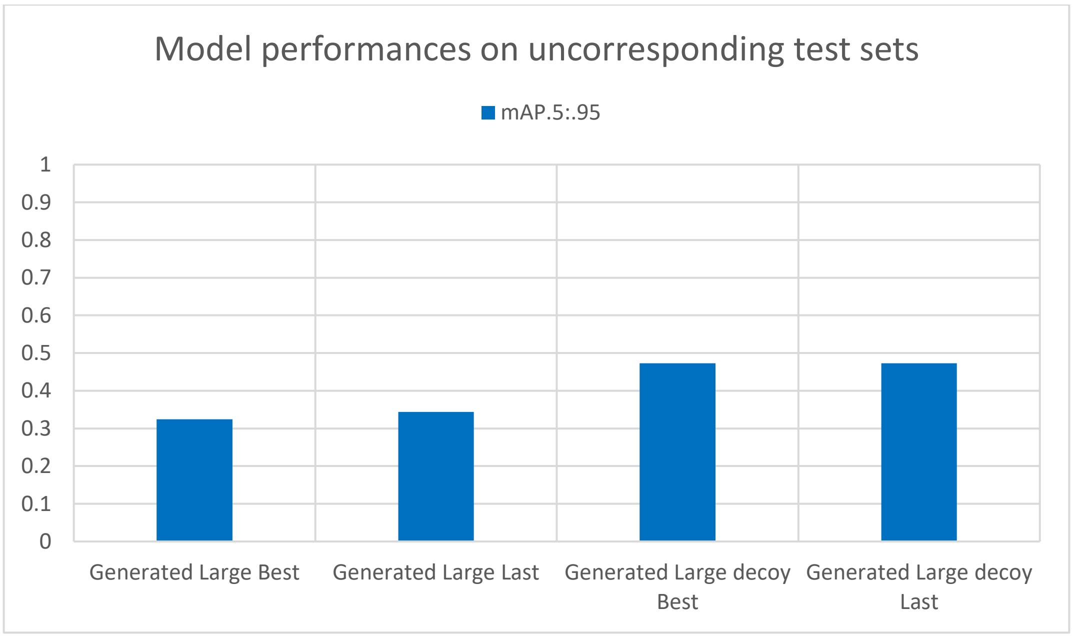

Figure 16: Model performances on uncorresponding test sets. Only mAP.5:.95 is shown for the networks trained on the largeand the large + decoy train sets to highlight the effect of the inclusion of decoy images on performance. The bars represent the same values as the corresponding blue bars in figure 14 and the exact values can be found in figure 12.

6.2.3.6 *Fine tuning best performing networks*

We want to determine if model generated images of tomatoes are suitable to aid the development of real world applications. An important step is bridging the gap between simulation and reality. In order to do this we use the pretrained networks from our previous steps and fine tune these. The pretrained networks were only trained using images of 3D model generated tomatoes. In fine tuning we train them for a few epochs on pictures of real tomatoes.

We fine tuned the network from the last epoch of the medium sized train set, as performance was highest here. We also fine tuned the network from the last epoch of the large train set including decoy images. This one was exposed to the most train data and we wanted to see if this gave any benefits.

In the table below we see the results of fine tuning the networks that were previously trained on images of generated tomatoes. Fine tuning was done for only 5 epochs.

Figure 17: Performances on the real tomato test set after fine tuning.

	P	**R**	**mAP@.5:.95**
Fine tuned medium last	0.922	0.875	0.735
Fine tuned large decoy last	0.875	0.882	0.719

The graph in figure 18 shows how the finetuning of these networks preceded in comparison to training from scratch. (Although this is not entirely true, the model was still pre trained on the COCO dataset at the start of training.) Here we see that when training the network from scratch we see a similar performance on the validation set after 5 epochs. However, we also see that the pre trained networks have a significant head start. Fine tuning generated medium last took only 0.171 hours, and generated large decoy last only 0.165 hours. Training the network from scratch on the real tomato train set took a total of 1.646 hours for 50 epochs, which is on average a similar duration per epoch. This is not surprising as the train sets are the same. For this smaller application it might seem like too much effort to pretrain on simulated tomatoes due to the relatively small amount of time that is needed, but for larger applications it might prove useful. Instead of saving 1.5 hours, training time can be reduced much more when larger datasets are considered and where a pretrained network is already available from a digital proof concept.

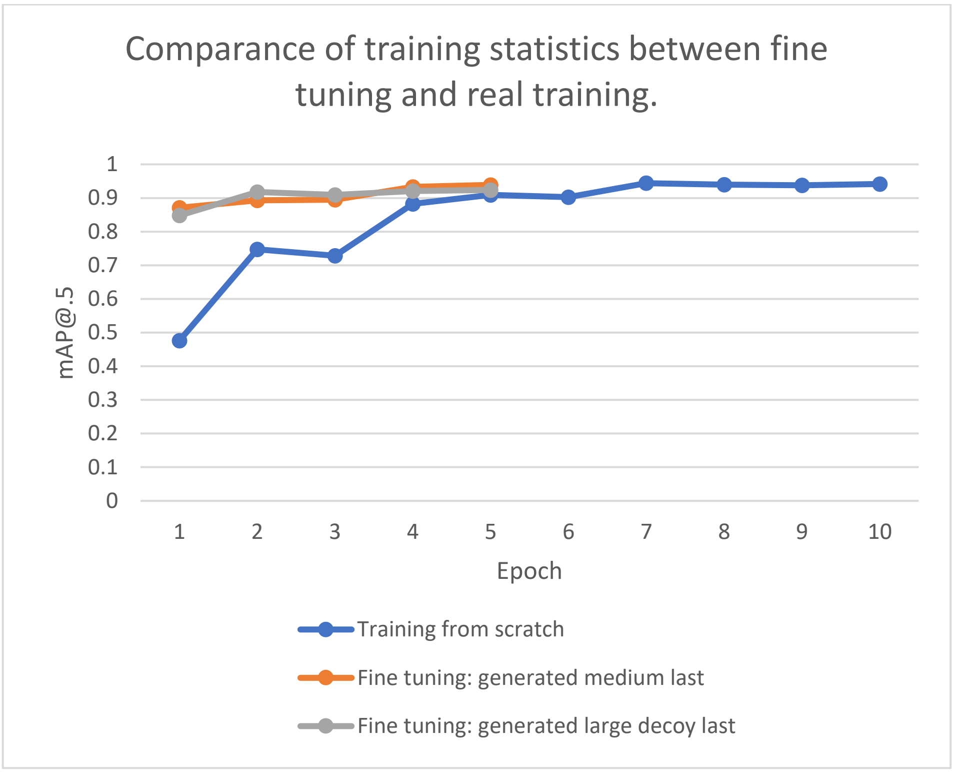

Figure 18: Performance comparison between fine tuning and training from scratch.

7 CONCLUSIONS

We can conclude that the script that was initially provided by András Polgár led to a reliable enough technique of creating digital tomato plants. Yolov7 recognized digital tomatoes based on a real life train set and vice versa. However, the model is far from perfect, as the networks trained on real and generated tomato plants are not interchangeable. This was evident from the performance statistics when testing a network on the test set of its counterpart. When developing novel greenhouse techniques where image recognition plays a central role this tomato model can be used to test concept before building prototypes.

The three sub questions were as follows:

- The effects of train set size.
- The incorporation of decoy images.
- The advantages of fine tuning the pretrained network.

We indeed observed a loss in performance when too many images of generated tomato plants were included in the train set, when the network was used to infer on the test set containing real tomatoes. This indicates a form of overfitting. We conclude that more is not always better when training a network that is expected to have more robustness. This is the case when digital models are used as a replacement of real image data. The models clearly represent tomato plants to the human eye, but are still dissimilar.

When incorporating decoy images we did not see an increase in performance when the corresponding test sets with model generated tomatoes were used. However, we did observe an increase in performance when inferring on the real tomato test set. We conclude that decoy images indeed add robustness and recommend using this in applications where virtual models are used in training a network which has the purpose of inferring on real images. At least for tomatoes this proved to be the case.

Fine tuning the pretrained model showed effective. In very few epochs a similar performance was reached in comparison to more epochs when training from the ground up. It should be noted that this network was also pretrained on the COCO dataset, which might reduce the amount of epochs necessary. It has been shown that pre training not always reduces the amount of epochs necessary, but that it can add to the models robustness[23]. We conclude that pretraining on model generated tomato images can indeed be a valuable and time saving step when developing image recognition networks.

8 DISCUSSION

We can still expect a lot of development with regard to computer performance[24]. Moore's law previously dictated that the amount of transistors that fit on a chip would double each two years, but this is a trend that slows down due to physical limitations. Instead we have to turn to smarter approaches. Therefore, a rise in computer efficiency in the coming years is still very much to be expected. Faster computing enables us to make even more expansive neural networks that we can train faster than before. And computer graphics will also become more detailed and realistic in the future.

As computer imagery becomes better, we can also expect a rise in seeming visual realism of future plant models. However, making model plants that fully and without fault represent their real life counterparts can realistically not be achieved. Similar to how other scientific models incorporate a statistical error margin, a plant model will also have a margin of error in regard to realism. Furthermore, the problem becomes more complex when more details are incorporated. A more accurate representation can be made when we model even the hairs on the leaves, but we need to ask if this is helpful. A similar problem that is a good analogy is the coastline paradox[25]. Here we see that when the scale of measurement becomes smaller, more detail is incorporated in the answer. Nevertheless, a practical application lacks. Therefore we should not strive for a perfect model for tomato plants either. We need to take context into account and produce a model that is sufficient enough for the application.

Areas where we can improve the model tomato plants are mainly with regard to variety and detail. More detail can easily be added by incorporating textures with higher resolution. But most important is the addition of more kinds of tomatoes. There are only four types of tomatoes in the model. Giving the model a very limited view of what a tomato can be. This is a typical case of overfitting. Similarly, when a network needs to be trained to recognize human faces in pictures, we want to incorporate a large variety of faces in the train set. When only pictures of four feminine women are used, the model might be thrown off when confronted by a picture of a man with a large moustache and beard. Just like how our models might be thrown off by a tomato of a different variety or one with blemishes. A larger variety in side stems is less important, because this is not what the model is optimized for to recognize. More variety and detail in this part of the model is of course welcome, but it likely will not add a lot to the networks' performances. If we also want to use the tomato plant models to develop other systems, this changes. For example if we want to automate pruning.

In addition to these considerations, future users should also be aware of the use of virtual models as a pre-training step. If an image recognition network trained on model data performs well in a virtual environment, then we can bridge the gap between simulation and real world by finetuning that network. This further diminishes the need for the network to be perfected. It is to be expected that our findings here can be applied to other similar projects as well. For example driving simulators could be used to aid the development of autonomous driving systems.

Lastly, the incorporation of decoy images, those that only contain random pictures of background noise, in the train set of the generated tomato network also increased robustness. From this we learn that a network can become better by adding unannotated pictures as well. In other projects where image recognition is being used we would advise to test the incorporation of random images as well.

We concluded that the 3D tomato model is realistic enough for the purpose of aiding the development of image recognition based systems, although a larger variety of tomatoes is welcome. However, the model also lays a groundwork for other purposes. Where we are benefitted with more detail and variety is in crop growth models. In order to further develop the model it needs to be perfected to correctly generate a main stem, based on real world measurements. Additional code can be written to also incorporate randomized side stems with tomatoes and leaves. These tomatoes and leaves can initially be selected from a large library of premodelled leaves and tomatoes, but ideally they are randomly generated as well. The addition of nutrition, sunlight hours, water availability and temperature parameters is also desirable. Based on these we can also incorporate varying growth rates, discoloration and wilting of the leaves, or varying degrees of fruit formation. This would be an ambitious project requiring a lot of data. However, our fundamental knowledge of plant growth would increase. Eventually also bringing us closer to a system of digital twins.

9 REFERENCES

1. Digital Twins. Accessed August 1, 2023. https://www.wur.nl/en/research-results/research-programmes/research-investment-programmes/digital-twins.htm

2. Polgár A. fields-ignition. https://github.com/azazdeaz/fields-ignition

3. Gazebo. Accessed October 20, 2023. https://gazebosim.org/home

4. Wu H, Liu Q, Liu X. A Review on Deep Learning Approaches to Image Classification and Object Segmentation. *Comput Mater Contin*. 2019;60(2):575-597. doi:10.32604/cmc.2019.03595

5. Wang CY, Bochkovskiy A, Liao HYM. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. Published online 2022. doi:10.48550/ARXIV.2207.02696

6. Lin TY, Maire M, Belongie S, et al. Microsoft COCO: Common Objects in Context. Published online February 20, 2015. Accessed August 1, 2023. http://arxiv.org/abs/1405.0312

7. Redmon J, Divvala S, Girshick R, Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. Published online 2015. doi:10.48550/ARXIV.1506.02640

8. Wang CY, Liao HYM, Yeh IH. Designing Network Design Strategies Through Gradient Path Analysis. Published online November 9, 2022. Accessed September 12, 2023. http://arxiv.org/abs/2211.04800

9. Bochkovskiy A, Wang CY, Liao HYM. YOLOv4: Optimal Speed and Accuracy of Object Detection. Published online April 22, 2020. Accessed September 12, 2023. http://arxiv.org/abs/2004.10934

10. Wang CY, Yeh IH, Liao HYM. You Only Learn One Representation: Unified Network for Multiple Tasks. Published online May 10, 2021. Accessed September 8, 2023. http://arxiv.org/abs/2105.04206

11. Tarvainen A, Valpola H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. Published online April 16, 2018. Accessed September 11, 2023. http://arxiv.org/abs/1703.01780

12. Skelton J. Step-by-step instructions for training YOLOv7 on a Custom Dataset. Paperspace. Accessed June 13, 2023. https://blog.paperspace.com/train-yolov7-custom-data/

13. Li M, Zhao Z, Scheidegger C. Visualizing Neural Networks with the Grand Tour. *Distill*. 2020;5(3):10.23915/distill.00025. doi:10.23915/distill.00025

14. Carter S, Armstrong Z, Schubert L, Johnson I, Olah C. Activation Atlas. *Distill*. 2019;4(3):10.23915/distill.00015. doi:10.23915/distill.00015

15. Nguyen TT, Nguyen QVH, Nguyen DT, et al. Deep learning for deepfakes creation and detection: A survey. *Comput Vis Image Underst*. 2022;223:103525. doi:10.1016/j.cviu.2022.103525

16. Yelmen B, Decelle A, Ongaro L, et al. Creating artificial human genomes using generative neural networks. Mathieson S, ed. *PLOS Genet*. 2021;17(2):e1009303. doi:10.1371/journal.pgen.1009303

17. Shafaei A, Little JJ, Schmidt M. Play and Learn: Using Video Games to Train Computer Vision Models. Published online August 15, 2016. Accessed June 5, 2023. http://arxiv.org/abs/1608.01745

18. The Editors of Encyclopaedia Britannica. Tomato. Encyclopaedia Britannica. Accessed August 28, 2023. https://www.britannica.com/plant/tomato

19. Tomato (Solanum lycopersicum). In: *Safety Assessment of Transgenic Organisms in the Environment, Volume 7*. Harmonisation of Regulatory Oversight in Biotechnology. OECD; 2017:69-104. doi:10.1787/9789264279728-6-en

20. He K, Gkioxari G, Dollár P, Girshick R. Mask R-CNN. Published online January 24, 2018. Accessed October 18, 2023. http://arxiv.org/abs/1703.06870

21. denismaestrox. vegetables Dataset. *Roboflow Universe*. Published online April 2023. https://universe.roboflow.com/denismaestrox/vegetables-fozzg

22. thesis WUR. Simulated Tomato Plants Dataset. *Roboflow Universe*. Published online June 2023. https://universe.roboflow.com/wur-thesis/simulated-tomato-plants

23. Hendrycks D, Lee K, Mazeika M. Using Pre-Training Can Improve Model Robustness and Uncertainty. Published online October 20, 2019. Accessed July 4, 2023. http://arxiv.org/abs/1901.09960

24. Leiserson CE, Thompson NC, Emer JS, et al. There’s plenty of room at the Top: What will drive computer performance after Moore’s law? *Science*. 2020;368(6495):eaam9744. doi:10.1126/science.aam9744

25. Steinhaus H. Length, shape and area. In: *Colloquium Mathematicum*. Vol 3. Polska Akademia Nauk. Instytut Matematyczny PAN; 1954:1-13.