Using Simulated Data for Deep-Learning Based Real-World Apple Detection

Dylan Hasperhoven[1], Maya Aghaei[1], and Klaas Dijkstra[1]

NHL Stenden University of Applied Sciences, Rengerslaan 8-10, 8917 DD Leeuwarden, the Netherlands
maya.aghaei.gavari@nhlstenden.com

Abstract. Object detection using CNNs requires a large amount of data to achieve decent performance in real-world scenarios. The creation of traditional datasets involves acquiring numerous images and manually annotating them. In this paper, we introduce a method for simulating apple orchards utilizing the Unity 3D engine. We created a tool that uses this simulator to generate fully bounding-box annotated (simulated) datasets. We trained YOLOv5 models of different sizes on simulated data, real-world data, and a combination of both, and later tested the models on a real-world dataset to evaluate the suitability of our generated dataset. Our experiments show that object detection models trained on simulated data can achieve results on real-world images that are very similar to results of models trained solely on real-world data. We demonstrate that simulated data has the potential to eliminate the need for real-world datasets, thus saving a substantial amount of time. In this research, we focused our tests on a real-world dataset acquired under controlled settings, future work can be dedicated to evaluate the generalization ability of models trained on simulated datasets on more challenging real-world datasets.

Keywords: Simulated data, Apple detection, Apple orchards, Visual inspections, YOLOv5

1 Introduction

An estimated amount of 245 million kilograms of apples were harvested in the Netherlands in 2021 according to the CBS [1]. When making the assumption that an apple weighs 100 grams on average, it can be estimated that there were 2,45 billion apples harvested in the Netherlands in 2021. In addition to fruit picking being a labour intensive and repetitive task, it is also seasonal. All of this leads to farmers having trouble finding workers to pick the large quantities of apples in their orchards. For this reason, automating this process using e.g. robots could prove invaluable [2].

Perhaps the most apparent and complex issue with automating such a task is the ability for robots to detect, localize and manipulate apples. Recent advancements in the area of computer vision technology have shown the ability of Convolutional Neural Networks (CNNs) to perform object detection [3] and Multiple-Object Tracking and Segmentation (MOTS) [4] on videos of apples in orchards. CNNs do so with significantly higher speed and accuracy than traditional methods [3]. The output of these networks can be used to automate robotic tasks.

To train a CNN with high performance, a large quantity of annotated data is needed. For apple detection and tracking specifically, the necessary amount of data is amplified by the fact that apples are homogeneous and relatively small objects, which makes them harder to detect and track [5]. This means that a large dataset is necessary. Composing such a dataset is a time-consuming task, as all apples in consecutive frames of a video require manual annotation [6]. Moreover, a CNN trained using a dataset containing images of particular varieties of apples and specific types of environments might not have the expected performance when inferred using images of a different variety of apples or orchards. As such the dataset should be representative for the orchard the robot will be used in.

A possible approach to overcome aforementioned obstacles is to train the CNN with simulated data. Recent studies evaluating this method with pointcloud data have found similar performance can be achieved using simulated data as when using real data [7,8]. The creation of simulated datasets requires less human labour than traditional datasets. Moreover, simulated datasets can be adjusted and generated in a relatively small timespan. This paper explores an approach that can be used to generate simulated data, which then can be used to train a CNN performing object detection on apples. It does so by utilizing the Unity 3D engine[9]. We are also particularly interested in the performance that can be achieved when training on a simulated dataset as opposed to a real-world dataset.

In this paper, we aim to answer the question *Can a simulated dataset of apple orchards generated in a 3D engine be advantageous for the purpose of training CNNs for apples detection in real-world orchards?* To help answer this, we introduce the following subquestions:

- How can a representative simulated dataset for object detection of apples be generated in a 3D engine?
- How does the performance of an apple detector trained on simulated data compare to one trained on real-world data, when tested in a real-world scenario?
- Can simulated data be helpful for training a more robust apple detector when combined with real-world data?

2 State of the art

Using simulated data for the purpose of training deep networks is a relatively new concept. There have been a few publications on this subject. One of them is SqueezeSeg, where they used a modification for the video game *Grand Theft Auto V (GTA-V)* to attach a LiDAR scanner to a car in the game. They then drove the car around in the game and collected point-cloud data to supplement their dataset of real LiDAR scanner data. The authors goal was to create a state of the art semantic segmentation model for pointclouds. They concluded that a CNN trained on a dataset supplemented with simulated data has significant performance increases compared to a CNN trained exclusively on real data [10].

A similar research was conducted in 2021 [7] where the self-driving vehicle research platform CARLA (Car Learning to Act) [11] was used to collect simulated LiDAR data. They then applied noise to the simulated data to make it more representa-

tive of real-world data. A CNN architecture for performing semantic segmentation was trained on the simulated data and compared to a traditional machine learning algorithm trained on real-world data. The results showed that the CNN trained on simulated data performed significantly better than the CNN trained on real-world data. This shows that when not enough data is available to train a deep learning algorithm, providing more data using simulation is a better performing option than using traditional machine learning models with only the original data [7].

Different from aforementioned research, this paper focuses on the simulation of image data. At the time of writing, there has been little research on the usage of simulated image data. This research focuses on CNNs for object detection of apples in orchards, but further research can be done to determine whether the techniques presented in this paper are applicable to different use cases.

Object detection for apples has been researched extensively in the past. Recent CNN architectures are able to achieve high detection performance. One research makes a comparison between the performance of different networks for the detection of apples [12]. Networks such as Faster R-CNN [13] and R-FCN[14] are able to achieve high mean Average Precision (mAP) [15,16], but do so at the cost of detection speed. Networks such as YOLOv3 [17] and YOLOv5 are an effective compromise. The mAP scores are slightly lower, however the detection speeds are significantly faster [18,12]. This means these networks are better suited for usage in UAV inspection.

In a recent study [19], a detection accuracy of 85.5% was reached on a particular dataset. This is 11% higher than previous studies. For the object detection, they utilized the YOLOv5 CNN architecture [20]. The researchers concluded that the algorithm is particularly good at detecting apples under the influence of complex occlusion. The algorithm can also reach a speed of 20FPS on a experimental platform which meets requirements for UAV inspection [19]. No studies so far have used simulated data to train apple detection networks however, which will be the contribution of this research. Our research will focus on using the YOLOv5 network.

In a NVIDIA research paper, a model named GET3D [21] is proposed. It is a generative model that is able to synthesize fully textured 3D meshes that can be used by 3D rendering engines such as Unity [9]. When trained on 2D images, the model can produce 3D shapes with high-fidelity textures and complex geometric details. This means that users of the model can generate a large number of 3D objects in a small amount of time. NVIDIA's GET3D, or a similar model could be used to generate objects to populate a 3D environment. This could be exceptionally useful for the creation of a simulated dataset of image data. In this research for example, it could be used to generate a large number of varying apple or apple tree objects. Since there were no trained models available at the time of this research, we chose not to use it.

3 Materials and Methods

For the systematic generation of simulated datasets, we propose a tool made in the Unity 3D game engine [9]. This software system can be configured to adapt the resulting dataset to specifically fit the application. After the configurations are chosen, the software will proceed to automatically create a dataset to be used as training data

for object detection networks. In this study, we specifically designed the simulation to mimic real-world apple orchards.

This section also describes the training and testing pipeline used to conduct our experiments.

3.1 Simulating apple orchards in Unity

To generate realistic data, a simulator that mimics an apple orchard is needed. This should reflect the scenario in the real-world. Our tool achieves this by using components supplied by the Unity 3D game engine. The camera component is used to simulate an RGB camera in a 3D scenario.

To simulate the 'orchard' component, our simulator uses a 2D plane with an image of a real-world orchard projected on it. This strategy allows us to simulate a representative orchard without the need for a true-to-life 3D reconstruction of an orchard, which would take a considerably larger amount of time and effort to realise. An example of this can be seen in Fig. 1.

Fig. 1. Simulated apple orchard with 2D plane with projected image and 3D textured apple models.

The simulator uses textured 3D models that are positioned between the 2D plane and the camera to simulate apples. The placements of the 3D apples are randomly determined. A selection of apple models that represent a wide range of varieties was used. The models used in the simulator are shown in Fig. 2.

3.2 Generating data in Unity

For object detection networks to achieve good results on real-world scenarios, datasets must be diverse. Our tool achieves a large variance in data by transforming the camera, giving the 3D apples new positions, and applying augmentations to the simulator. A new camera transform is generated for every data point. The number of data points after which the apples get augmented can be configured. The total number of data points can also be configured.

Fig. 2. Textured 3D apples used in the simulation.

Camera transformation By altering the camera transform for every data point the data will be generated from different angles and distances. Our tool does this by generating pseudo-random transforms. Because our simulator uses a 2D background plane with 3D apples positioned in front of it, there is a limited space of transforms for the camera that would produce valid data. In this case, valid data is data where:

- Every pixel on the RGB image is projected from the 2D background or an apple.
- The camera is facing the front-side of the 2D plane.
- Apples are clearly visible (i.e. not occluded, fully in frame of the camera, sufficiently sized to be recognizable).

Fig. 3 shows some examples of invalid camera transforms.

(a) Camera too close **(b)** Camera oriented too far downwards **(c)** Camera too far

Fig. 3. Invalid camera positions: (a) Camera is too close to the 2D plane, causing the apples to be not clearly visible. (b) Camera is oriented downwards causing pixels not projected from the 2D background plane or an apple to appear in the camera image. (c) Camera is too far from the 2D plane, causing the apples not to be clearly visible.

Limits of orientations and positions can be configured in the tool. The generator then generates random values to calculate transforms on the linear interpolants between these limits. Furthermore, our tool includes a minimum and maximum distance between the 2D plane and the camera. This guarantees the generated camera transforms are 'valid' according to the conditions described above.

Apple positions New 3D positions are generated for apples after a certain number of data points. This number can be configured.

The first method uses a random uniform distribution to generate x-coordinates and y-coordinates, where the limits are the target resolution of the image data. Back-projection and Unity's raycasting system are used to determine the 3D position on the 2D background plane that projects onto the x-coordinates and y-coordinates on the camera frame. If the smallest distance between the generated position and other generated positions is lower than the diameter of the apples (minus a maximum overlap value), the generated position is discarded. This ensures that the apples have a minimal amount of overlap.

For the second method 2D coordinates of the center of all apples are extracted from the instance segmentation masks of the APPLE_MOTS dataset [5]. Only segmentation masks from the scenarios with a line of apple trees perpendicular to the camera were considered. The extracted positions are stored per segmentation mask. When the tool generates new positions it will randomly pick a segmentation mask to use the positions from. Similar to the first method, back-projection and Unity's raycasting system is used to determine the 3D position on the 2D background plane that projects onto the x-coordinates and y-coordinates from the extracted positions on the image frame.

Augmenting data Data from the generator should be representative of the real-world scenario. For this reason so-called augmentations have been implemented in the generator tool. These augmentations can be turned on or off to enrich the variety in the datasets.

In the baseline scenario (i.e. without any augmentations) all apples in the simulator have the same 3D model, texture, scale, rotation and coordinate on the z-axis. The lighting in the scene stays the same throughout the dataset. The positions and number of apples as well as the camera's transformation are the only variables which change throughout the dataset.

Table 1 shows all augmentations that have been implemented. When generating a dataset, new apple positions are loaded and augmentations are applied after a specified number of datapoints.

Generating annotations With our tool, we propose a technique that allows it to calculate perfectly accurate bounding boxes in Unity 3D.

The algorithm works by first calculating an instance segmentation mask. It does this by using back-projection to calculate the direction vector from the camera origin, through an x-coordinate and y-coordinate on the image plane. In the 3D scene, Unity's raycasting system is used to cast a ray from the camera origin to the direction calculated in the step before. The algorithm in Unity finds the first object the ray intersects, and determines if it corresponds to the background or an apple. If the intersection corresponds to the background, it will insert a value of '0' in the mask. If the intersection corresponds to an apple, an instance identifier belonging to the specific apple in the mask will be inserted into the map. Bounding boxes can then be calculated by determining the minimum and maximum x-coordinates and y-coordinates for each instance identifier. Fig. 4 shows and overview of this process.

Table 1. List of all augmentations for generating Simulated-Orchards datasets.

Augmentation	Description
rotation	Gives a random orientation to every apple.
scale	Gives a random scale to every apple.
depth	Gives a random z-coordinate (depth) to every apple.
lighting	Changes the lighting of the scene.
color_per_apple	Gives a random (realistic) color to each apple.
color_per_scene	Generates a random (realistic) color for all apples in the scene.
model_per_apple	Chooses a random 3D model for every apple from the set of 3D models shown in Fig. 2.
model_per_scene	Chooses a random 3D model apple from the set of 3D models shown in Fig. 2 and gives it to all apples in the scene.

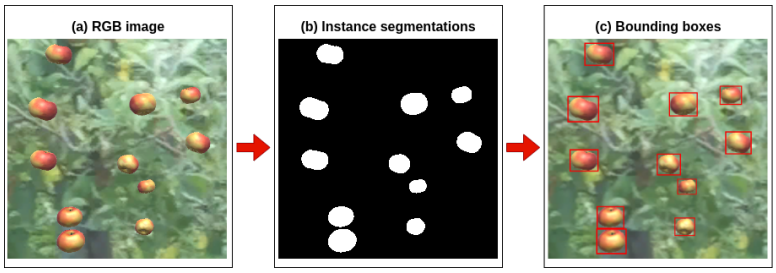


Fig. 4. Process of calculating bounding boxes. (a) RGB image from the virtual camera. (b) instance segmentation mask created using back-projection. (c) bounding boxes (overlayed on original RGB image) calculated by finding the minimum and maximum x-coordinate and y-coordinate for every instance.

3.3 Real-world datasets

A real-world dataset was used to evaluate and compare the performance of models trained on simulated data.

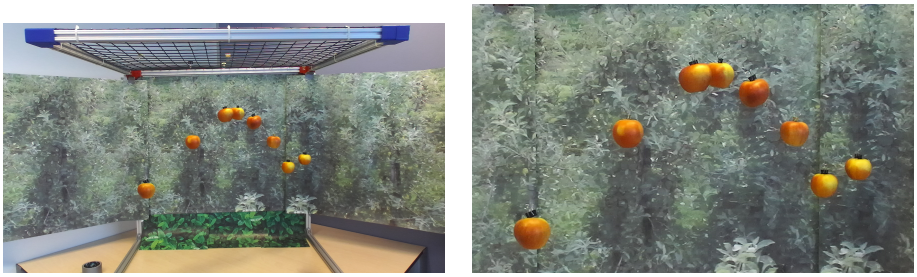
Mini-Orchards Mini-Orchards is a dataset created within our lab and consists of images of apples suspended by string from a frame. The apples hanging in front of a background emulate a real orchard. The nature of the images makes it a rather simple dataset, since apples are visible mostly without obstruction from branches and leaves. There are however cases where apples are obstructed by other apples.

The camera was placed at a fixed distance from the frame holding the apples. RGBD images were taken using a ZED Stereolabs stereo camera. However, since this research focuses on RGB data, the depth channel was discarded. The resulting images have a 1920x1080 resolution.

Mini-Orchards has 500 images that are divided into a train, validation, and test set. The train set contains 300 images, while the validation and test sets each have 100

images. The complete dataset has roughly 3000 apples that are annotated with bounding boxes.

In the original images, too much of the surrounding environment is visible in the frame of the camera. Objects like the grid structure holding the apples can be seen in the images. We cropped the images to discard the irrelevant background section of the images and used these cropped images for our experiments. Fig. 5 shows an example of a cropped and uncropped image from the dataset. The cropped images have a resolution of 1920x576.



(a) original image **(b)** cropped image

Fig. 5. Example of (a) original and (b) cropped image from the Mini-Orchards dataset.

3.4 Apple detection using YOLOv5

YOLOv5 [20] is one of the newest deep learning models in the YOLO (You Only Look Once) family of object detectors. One of the prominent features of YOLO is fast inference, enabling the models to be used for real-time object detection. YOLO achieves great results in object detection tasks by using CNNs. The YOLO architecture consists of 3 parts: backbone, neck and head. The backbone is responsible for feature extraction. The neck is used for feature fusion. Lastly, the head performs the final predictions. YOLOv3 uses the 53 layer network Darknet-53 as the backbone, and stacks 53 more convolutional layers on top as the neck. The model then performs detection by applying 1x1 convolutional filters on feature maps at three different places in the network [17]. The network predicts four coordinates for each bounding box, t_x, t_x, t_w, t_h.

YOLOv5 achieves better results by adding improvements to the architecture. The backbone and neck are changed, while the head remains the same. YOLOv5 uses CSP-Darknet as the backbone and Path Aggregation Network (PANet) [22] as the neck. These new components enhance the speed and detection performance of the network. YOLOv5 has different variants: the n, s, m, l and x versions have different network width and depths which influence the number of trainable parameters, where the n version is the smallest and the x version the largest. This means that the YOLOv5 x version has the potential to learn more complicated tasks and achieve the best detection perfor-

mance, provided the model is trained on enough data of sufficient quality. The larger models also have the slowest inference and require the most data to train.

3.5 Evaluation metrics

To quantify the performance of trained networks, a number of metrics can be used. We chose to use the most commonly used metrics specific for the task of object detection. This section explains how the metrics are calculated and how they can be interpreted in the context of apple detection.

Intersection over Union (IoU) represents the intersection ratio of the predicted bounding box and the ground truth bounding box. Predictions with an IoU and probability score above a certain threshold are considered to be a true positive (TP) sample. Predictions below these thresholds are considered as a false positive (FP) sample. Cases of a ground-truth sample not being successfully detected are considered false negative (FN).

- *True Positive (TP)* are the cases that have been predicted as an apple, and there is an apple in the ground truth.
- *False Positive (FP)* are the cases that have been predicted as an apple, and there was no apple in the ground truth.
- *False Negative (FN)* are the cases that where not predicted as an apple, but there was an apple in the ground-truth.

Precision is a metric that shows the percentage of predictions by the network that are correct. To calculate precision, the number of correct predictions is divided by the total number of predictions. The precision can also be calculated using the number of TP and FP cases.

$$Precision = \frac{Correct\ predictions}{Total\ predictions} = \frac{\#TP}{\#TP + \#FP} \tag{1}$$

Recall is a metric that shows the percentage of apples in the ground truth that were predicted correctly by the network. To calculate recall, the number of correct predictions is divided by the total number of apples in the ground truth. Recall can also be calculated using the number of TP and FN cases.

$$Recall = \frac{Correct\ predictions}{Total\ instances\ in\ ground\ truth} = \frac{\#TP}{\#TP + \#FN} \tag{2}$$

Recall and precision scores complement each other. The *F1-score* is used to find a balance between the two. It can be seen as a harmonic mean of the precision and recall scores.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{3}$$

We denote the probability threshold used to calculate the precision, recall, and F1 scores with the @ symbol followed by the threshold value (e.g. $F1@0.5$).

Average Precision (AP) is a metric that summarises the precision recall curve in one value. The precision recall curve shows the relation between recall and precision, and

is calculated by evaluating the model at different probability thresholds. When given a precision recall curve, the AP is calculated using equation 4:

$$AP = \frac{1}{|r|} \sum_{r \in \{0, 0.01, \ldots, 1\}} p_{interp}(r) \tag{4}$$

In this equation $p_{interp}(r)$ denotes the *interpolated* precision at a recall level of r. We calculate this at 101 equally spaced recall values between 0 and 1, which is the standard used by the COCO challenge [23]. The precision is interpolated by taking the maximum precision corresponding to recall levels greater than r:

$$p_{interp}(r) = \underset{\tilde{r}:\tilde{r} \geq r}{max}\, p(\tilde{r}) \tag{5}$$

Where $p(\tilde{r})$ denotes the precision value evaluated from the precision recall curve at $\tilde{r}$.

4 Experiments & Results

This section describes the experiments that were conducted during the research, the relation among them and and their corresponding results. Since we are aiming for these models to run on robot systems with a limited number of resources, we chose the three smallest YOLOv5 sizes: *n*, *s* and *m*. All models were trained on a *NVIDIA A40* GPU with 16GB of available VRAM.

4.1 Experiment 1: performance of augmentations

With our first experiment, we intend to find how to generate a representative simulated dataset for the object detection of real-world apples in orchards. The dataset generator tool described in Section 3 is used to create the training sets. For this experiment, we created one training set for every augmentation described in Table 1. This means that we have eight separate training sets, each with exactly one unique augmentation. Additionally, one training set is created without any augmentations, which will be referred to as the 'baseline' training set. Each dataset is created with 1500 images, to ensure that there is enough data. This value has been chosen empirically. During the creation of the dataset, new apple positions and augmentations will be loaded every 5 images. This was done to introduce more variety into the dataset.

We trained three different sizes of YOLOv5 models for every dataset described above. The validation set from the Mini-Orchards dataset was used for validation during training. On every epoch, the model with the best AP score on the validation set was saved. All saved models were then tested on the testing set from the Mini-Orchards dataset, after which the metrics would be calculated.

All models were trained using a learning rate of 0.001 and the highest batch size that could fit in the memory of our GPU. The pretrained YOLOv5 checkpoints provided by Ultralytics [20] were used. Furthermore, all models were trained, validated and tested using 576x576 tiling, as this is the biggest tile size that fits the cropped images from the Mini-Orchards dataset. The models were trained for a maximum of 30 epochs. Furthermore, we used automatic mixed precision to speed up the training process.

Table 2. Results of experiment 1: various sizes of YOLOv5 models, trained on simulated data with different augmentations, and tested on the Mini-Orchards test set. Values are the mean and standard deviation calculated from five runs. The last row shows the average of all trained models per YOLOv5 size.

YOLOv5s				
Trainset	AP	Recall@0.1	Precision@0.1	F1@0.1
baseline	0.397 ± 0.315	0.400 ± 0.317	0.795 ± 0.444	0.507 ± 0.354
color_per_scene	0.948 ± 0.004	**0.953 ± 0.006**	**0.985 ± 0.012**	**0.969 ± 0.009**
model_per_scene	**0.949 ± 0.009**	0.956 ± 0.008	0.974 ± 0.007	0.965 ± 0.007
model_per_apple	0.490 ± 0.355	0.493 ± 0.355	0.795 ± 0.444	0.589 ± 0.371
color_per_apple	0.447 ± 0.342	0.531 ± 0.343	0.590 ± 0.383	0.550 ± 0.349
lighting	0.075 ± 0.099	0.089 ± 0.115	0.328 ± 0.330	0.133 ± 0.158
rotation	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
scale	0.094 ± 0.197	0.106 ± 0.194	0.257 ± 0.433	0.150 ± 0.268
depth	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Average	0.378 ± 0.147	0.392 ± 0.149	0.525 ± 0.228	0.429 ± 0.168

YOLOv5s				
Trainset	AP	Recall@0.1	Precision@0.1	F1@0.1
baseline	0.927 ± 0.007	0.934 ± 0.009	0.992 ± 0.006	0.962 ± 0.004
color_per_scene	**0.962 ± 0.004**	**0.965 ± 0.004**	0.984 ± 0.017	**0.975 ± 0.009**
model_per_scene	0.951 ± 0.011	0.958 ± 0.009	0.987 ± 0.005	0.972 ± 0.007
model_per_apple	0.942 ± 0.004	0.948 ± 0.006	0.988 ± 0.014	0.967 ± 0.006
color_per_apple	0.913 ± 0.035	0.920 ± 0.037	**0.996 ± 0.004**	0.956 ± 0.019
lighting	0.449 ± 0.335	0.489 ± 0.376	0.934 ± 0.147	0.543 ± 0.380
rotation	0.326 ± 0.401	0.366 ± 0.433	0.458 ± 0.453	0.364 ± 0.406
scale	0.316 ± 0.304	0.386 ± 0.314	0.761 ± 0.243	0.432 ± 0.327
depth	0.328 ± 0.349	0.357 ± 0.329	0.839 ± 0.348	0.441 ± 0.350
Average	0.679 ± 0.161	0.703 ± 0.169	0.882 ± 0.137	0.735 ± 0.168

YOLOv5m				
Trainset	AP	Recall@0.1	Precision@0.1	F1@0.1
baseline	0.937 ± 0.019	0.946 ± 0.014	0.954 ± 0.046	0.950 ± 0.027
color_per_scene	**0.964 ± 0.006**	**0.970 ± 0.006**	0.985 ± 0.005	**0.978 ± 0.005**
model_per_scene	0.952 ± 0.016	0.960 ± 0.011	0.985 ± 0.004	0.972 ± 0.007
model_per_apple	0.942 ± 0.018	0.952 ± 0.017	0.982 ± 0.004	0.967 ± 0.010
color_per_apple	0.953 ± 0.010	0.959 ± 0.007	0.990 ± 0.002	0.974 ± 0.004
lighting	0.928 ± 0.009	0.942 ± 0.013	0.944 ± 0.097	0.940 ± 0.048
rotation	0.934 ± 0.012	0.940 ± 0.010	0.991 ± 0.006	0.965 ± 0.008
scale	0.938 ± 0.012	0.946 ± 0.011	**0.992 ± 0.004**	0.968 ± 0.007
depth	0.928 ± 0.056	0.944 ± 0.037	0.957 ± 0.087	0.950 ± 0.062
Average	0.942 ± 0.018	0.951 ± 0.014	0.976 ± 0.028	0.963 ± 0.020

Because smaller YOLOv5 models can be quite noisy, all models were trained and tested five times. As a result, 135 models were trained and tested. This allowed us to

calculate the mean and standard deviation for the tests. The results of the tests can be seen in Table 2.

The results show that on average, the *m* YOLOv5 models perform the best, with *s* models following, and then *n* models. Given that *m* models are the largest and *n* the smallest of the tested models, this is to be expected.

For the *m* models, most of the augmentations showed an improvement over the baseline models. With the *depth* augmentation being the only exception to this. Of all the augmentations, *color_per_scene* showed the best improvement compared to the baseline model. The model trained on *color_per_scene* augmented data achieved the highest AP, F1 and recall scores of all models.

For the *s* models, more augmentations showed a decrease of performance when compared to the baseline models. The *color_per_apple*, *depth*, *lighting*, *rotation* and *scale* augmentations all performed less than the baseline model. Again, the *color_per_scene* augmentation showed the most performance increase over the baseline model.

For the *n* models however, most trained models were not able to consistently detect and localize apples, thus achieving very low scores. The exception can be seen for models trained on the datasets with *color_per_scene* and *model_per_scene* augmentations, which both achieved good scores, only slightly worse than the bigger models trained on the same datasets.

4.2 Experiment 2: performance of simulated data versus a traditional dataset

With the second experiment, we aim to find out how the performance of an object detection CNN trained on only a simulated training set compares to that of a CNN trained on only real-world data.

Much like the first experiment, we used the dataset generator tool to create a training set. Due to its superior performance from experiment 1, we decided to generate the training set using the *color_per_scene* augmentation for this experiment. Since the training set of the Mini-Orchards dataset only contains 300 images, we chose to create the simulated training set with the same number. This removes the number of images used being a factor for causing a difference in performance.

We then trained YOLOv5 models on the simulated training set, and on the Mini-Orchards training set. The hyperparameters and the validation process were the same as in experiment 1, with the only difference being the number of epochs. Since these training sets have fewer images than the ones in experiment 1, the models took a longer time to converge. By using 80 epochs this experiment, we made sure that the models converged during training.

Like experiment 1, trained models were tested on the test set from the Mini-Orchards dataset, after which the evaluation metrics would be calculated. Training and testing was done five times. Table 3 shows the means and standard deviations of the metrics for all models.

As expected, *m* models have the best performance for both training sets, with *s* and *n* following after. When comparing models of the same sizes trained on the Mini-Orchards and Simulated-Orchards training sets, the results show that the ones trained on Mini-Orchards consistently perform better. Scores for the *m* size models are relatively

Table 3. Results of experiment 2: various sizes of YOLOv5 model, trained on the Mini-Orchards (training) dataset or 300 images from the Simulated-Orchards dataset. The models were tested on the Mini-Orchards (real-world) test dataset.

	AP	F1@0.1	Recall@0.1	Precision@0.1
YOLOv5n				
Mini-Orchards	**0.972 ± 0.004**	**0.987 ± 0.004**	**0.979 ± 0.003**	**0.995 ± 0.004**
Simulated-Orchards	0.789 ± 0.261	0.862 ± 0.199	0.799 ± 0.252	0.976 ± 0.040
YOLOv5s				
Mini-Orchards	**0.976 ± 0.006**	**0.977 ± 0.000**	**0.983 ± 0.005**	**0.970 ± 0.033**
Simulated-Orchards	0.845 ± 0.095	0.834 ± 0.146	0.906 ± 0.060	0.795 ± 0.222
YOLOv5m				
Mini-Orchards	**0.982 ± 0.004**	**0.983 ± 0.006**	**0.987 ± 0.004**	**0.979 ± 0.013**
Simulated-Orchards	0.955 ± 0.007	0.967 ± 0.009	0.964 ± 0.005	0.971 ± 0.019

similar, with the model trained on Simulated-Orchards scoring 0.03 and 0.02 less in AP and F1, respectively.

When looking at the smaller models, the difference between models trained on Mini-Orchards and Simulated-Orchards become greater. The models trained on Simulated-Orchards get 0.13 AP and 0.14 F1 less than the models trained on Mini-Orchards for the *s* models, and 0.18 AP and 0.13 F1 less for the *n* models. We can also see that the smaller models trained on Simulated-Orchards get inconsistent results, as can be seen by the increasing standard deviations.

4.3 Experiment 3: performance of a hybrid dataset

With the third experiment, our objective is to find out whether adding real images to a training set of simulated images can improve object detection on real data. We are particularly interested in understanding how much real data is needed to be added to simulated data for achieving the performance that training on only real data can offer. This is important, as the conclusion of this experiment can be used to compensate the shortage of real data, which is a common problem in training deep networks, with simulated data.

We created new training sets by adding different numbers of real-world images to a simulated training set. To do this, we used the Simulated-Orchards training set from experiment 2. This training set has 300 images and uses the *color_per_scene* augmentation. The Simulated-Orchards training set was then expanded by the first 100, 200, and 300 images from the Mini-Orchards training set, producing three new training sets. We then trained YOLOv5 models of different sizes, using similar hyperparameters and validation as experiment 2. The trained models were once again tested on the test set from the Mini-Orchards dataset. This process was repeated five times. Table 4 shows the means and standard deviations of the metrics that were calculated after every test.

The quantitative results show that all models trained on datasets with added real data have improved performance compared to models trained without real data. By looking

Table 4. Results of experiment 3: different sizes of YOLOv5 model, trained on hybrid datasets with a combination of real and simulated data. The models were tested on the Mini-Orchards (real-world) test set.

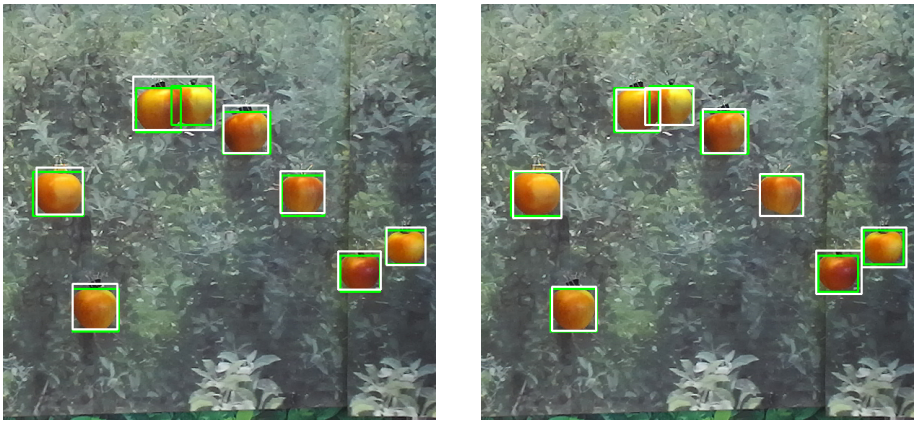
	AP	F1@0.1	Recall@0.1	Precision@0.1
YOLOv5n				
Simulated-Orchards	0.789 ± 0.261	0.862 ± 0.199	0.799 ± 0.252	0.976 ± 0.040
Simulated-Orchards + 100 real	0.950 ± 0.021	0.943 ± 0.031	0.957 ± 0.017	0.930 ± 0.052
Simulated-Orchards + 200 real	0.954 ± 0.006	0.961 ± 0.027	0.962 ± 0.006	0.961 ± 0.054
Simulated-Orchards + 300 real	**0.964 ± 0.005**	**0.980 ± 0.007**	**0.971 ± 0.004**	**0.989 ± 0.017**
YOLOv5s				
Simulated-Orchards	0.845 ± 0.095	0.834 ± 0.146	0.906 ± 0.060	0.795 ± 0.222
Simulated-Orchards + 100 real	0.940 ± 0.039	0.960 ± 0.022	0.958 ± 0.016	0.962 ± 0.034
Simulated-Orchards + 200 real	0.970 ± 0.007	**0.988 ± 0.003**	**0.978 ± 0.005**	1.000 ± 0.001
Simulated-Orchards + 300 real	**0.970 ± 0.000**	0.987 ± 0.000	0.974 ± 0.001	**1.000 ± 0.000**
YOLOv5m				
Simulated-Orchards	0.955 ± 0.007	0.967 ± 0.009	0.964 ± 0.005	0.971 ± 0.019
Simulated-Orchards + 100 real	0.966 ± 0.005	0.983 ± 0.007	0.972 ± 0.003	0.994 ± 0.013
Simulated-Orchards + 200 real	0.970 ± 0.000	0.984 ± 0.003	0.973 ± 0.004	0.996 ± 0.003
Simulated-Orchards + 300 real	**0.970 ± 0.000**	**0.986 ± 0.001**	**0.977 ± 0.001**	**0.996 ± 0.002**

at the results of models trained with 100, 200 and 300 real images, it seems that when adding more real images to the dataset, the performance gets higher.

The most significant improvements are seen in the smaller models. By adding 100 real images to the simulated dataset, the *n* model gets an AP of 0.950 and an F1 score of 0.943. This is only slightly worse than the larger model and an improvement of 0.16 AP and 0.08 F1 compared to the same model when trained without any real data.

However, when comparing to the models trained on only Mini-Orchards which can be seen in Table 3, it shows that none of the models with simulated and real data achieve better performance.

The qualitative results reveal that models trained with added real data are better at distinguishing overlapping apples, when compared to models trained without real data. An example of this can be seen in Fig. 6.



(a) trained without real images **(b)** trained with 300 real images

Fig. 6. Qualitative results of YOLOv5m models trained on (a) only simulated data and (b) simulated data with 300 real images. White boxes show the predictions from the trained model, while green boxes show the manually annotated ground-truth.

5 Discussion, Conclusion and Future Work

The goal of this research was to answer the question: *Can a simulated dataset of apple orchards generated in a 3D engine be advantageous for the purpose of training CNNs for apples detection in real-world orchards?* We created a tool to generate simulated data of apples together with a lab made dataset of imaged of hanging apples and conducted three different experiments. In this section, we discuss our findings and the conclusions we can draw from them. We also talk about the research that could be done in the future to further improve on our results.

5.1 Discussion

In this research, we introduced a way of simulating an apple orchard, using the Unity 3D game engine. We used this simulator to create a tool that is able to generate fully bounding-box annotated datasets. Moreover, the simulator and data generating tool are highly configurable and modifiable. This allows for the rapid generation of different types of datasets.

In addition to the baseline situation, we introduced eight different augmentations to the simulated scenario. These augmentations change the visual aspect of the simulation. This was done to find what techniques could make the simulated dataset represent the real-world scenario the best. The experiment show that our *color_per_scene* augmentation achieves the best results. This augmentation changes the colour of all apples in the simulation every time new apples are loaded in.

We compared YOLOv5 models trained exclusively on simulated data to models trained on real-world data from the Mini-Orchards dataset. We tested the models on

real-world data from the test set of the Mini-Orchards dataset and calculated the Precision, Recall, F1, and AP metrics. The metrics show that for the YOLOv5m models, models trained solely on simulated data can achieve results that are quite close to those of models trained on real data. The smaller YOLOv5 models show a bigger performance gap between the ones trained on simulated and the ones trained on real data.

We also did experiments where the training sets of simulated images were supplemented by different amounts of real data. Once again, we trained models on these training sets, tested them on the Mini-Orchards test set, and calculated the metrics. The results show that adding real images does improve the results, however the results would never be better than the models trained solely on real data. Most interestingly, the results of the smaller YOLOv5 models improved significantly, achieving almost the same results as those of the bigger YOLOv5 models.

However, it should be said that the real-world dataset used for testing the models was quite more simple than an actual orchard. Further experiments on more challenging real-world datasets have to be performed to demonstrate the generalizability of these findings.

5.2 Conclusion

Based on this research, we can conclude that the use of simulated data for training object detectors appears to be promising. When testing on the simple testing set from the Mini-Orchards dataset, models trained on simulated data received high Precision, Recall, F1, and AP scores. While we were not able to achieve the same or better results as models trained exclusively on real data, we did achieve very similar results. The biggest advantage of simulated data, is that by using a 3D engine, perfect bounding boxes can be automatically generated. This eliminates the need for manual annotation of datasets, which is an exceedingly time consuming task. Simulated data also has the advantage of being very configurable, allowing changes like lighting, camera properties and much more to be made very quickly.

However, we were never able to achieve higher results using models trained on simulated data, than models trained exclusively on real-world data. We conclude that when the best possible results are needed, it is better to use real-world data.

Furthermore, we published both the *Mini-Orchards*[1] dataset and an example of a *Simulated-Orchards*[2] dataset with the *color_per_scene* augmentation.

5.3 Future Work

The results from our experiments are promising, however we believe future research should apply the techniques used in this research to train models for more challenging datasets: including more occlusion (e.g. leaves and branches blocking apples), complicated backgrounds, and various camera angles and distances. This could be done by manually annotating images taken in real orchards. The future work should include exploring which augmentations from the dataset generator tool perform the best on this

[1] https://www.kaggle.com/datasets/dylanhasperhoven/mini-orchards
[2] https://www.kaggle.com/datasets/dylanhasperhoven/simulated-orchards

particular dataset, and could even include implementing new, more complex, augmentations and backgrounds.

The Unity 3D engine can also be used to simulate a multitude of image modalities, such as RGB, depth and IR intensity. Future work could be done to investigate the impact of different image modalities on the performance of models trained on simulated data.

Finally, the techniques used to create the dataset generator tool are fairly generic: allowing the tool to be modified and updated, making it possible to simulate a wide range of scenarios. Future research could be dedicated to investigate the advantages of simulated data for object detection of other types of objects than apples. The tool could easily be modified to simulate other types of fruit orchards, but also contrasting scenarios other than orchards.

Acknowledgements

This project was financially supported by Regieorgaan SIA (part of NWO) and performed within the RAAK PRO project Mars4Earth. We would like to thank our collaborators at Saxion University of Applied Sciences for insightful discussions.

References

1. Centraal Bureau voor de Statistiek. Fruitteelt; oogst en teeltoppervlakte appels en peren.
2. Robert Bogue. Fruit picking robots: has their time come? *Industrial Robot: the international journal of robotics research and application*, 2020.
3. Fangfang Gao, Wentai Fang, Xiaoming Sun, Zhenchao Wu, Guanao Zhao, Guo Li, Rui Li, Longsheng Fu, and Qin Zhang. A novel apple fruit detection and counting methodology based on deep learning and trunk tracking in modern orchard. *Computers and Electronics in Agriculture*, 197:107000, 2022.
4. Stefan de Jong. Stepping towards real-time detection and tracking of apples using deep learning (mots), Jan 2021.
5. Stefan de Jong, Hilmy Baja, Karsjen Tamminga, and João Valente. Apple mots: Detection, segmentation and tracking of homogeneous objects using mots. *IEEE Robotics and Automation Letters*, 7(4):11418–11425, 2022.
6. Paul Voigtlaender, Michael Krause, Aljosa Osep, Jonathon Luiten, Berin Balachandar Gnana Sekar, Andreas Geiger, and Bastian Leibe. Mots: Multi-object tracking and segmentation. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pages 7942–7951, 2019.
7. S. Spiegel and J. Chen. Using simulation data from gaming environments for training a deep learning algorithm on 3d point clouds. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, VIII-4/W2-2021:67–74, 2021.
8. Sijun Niu and Vikas Srivastava. Simulation trained cnn for accurate embedded crack length, location, and orientation prediction from ultrasound measurements. *International Journal of Solids and Structures*, 242:111521, 2022.
9. Unity Technologies. Unity real-time development platform — 3d, 2d vr & ar engine, 2022.
10. Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893, 2018.

11. Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator, 2017.
12. Bin Yan, Pan Fan, Xiaoyan Lei, Zhijie Liu, and Fuzeng Yang. A real-time apple targets detection method for picking robot based on improved yolov5. *Remote Sensing*, 13(9), 2021.
13. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.
14. Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
15. Dandan Wang and Dongjian He. Recognition of apple targets before fruits thinning by robot based on r-fcn deep convolution neural network. *Transactions of the CSAE*, 35(3):156–163, 2019.
16. Longsheng Fu, Yaqoob Majeed, Xin Zhang, Manoj Karkee, and Qin Zhang. Faster r–cnn–based apple detection in dense-foliage fruiting-wall trees using rgb and depth features for robotic harvesting. *Biosystems Engineering*, 197:245–256, 2020.
17. Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.
18. Dean Zhao, Rendi Wu, Xiaoyang Liu, and Yuyan Zhao. Apple positioning based on yolo deep convolutional neural network for picking robot in complex background. *Trans. Chin. Soc. Agric. Eng*, 35(3):172–181, 2019.
19. Shaopeng Wang, Xiaodong Zhang, Haiming Shen, Minxuan Tian, and Mingyang Li. Research on uav online visual tracking algorithm based on yolov5 and flownet2 for apple yield inspection. In *2022 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, pages 280–285, 2022.
20. Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Kalen Michael, Jiacong Fang, imyhxy, Lorna, Colin Wong, (Zeng Yifu), Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Max Strobel, Mrinal Jain, Lorenzo Mammana, and xylieong. ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations, August 2022.
21. Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. In *Advances In Neural Information Processing Systems*, 2022.
22. Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation, 2018.
23. COCO Consortium. Common objects in context.