# Synthetic Data at Scale: A Paradigm to Efficiently Leverage Machine Learning in Agriculture

Jonathan Klein[a], Rebekah Waller[b], Sören Pirk[c], Wojtek Pałubicki[d], Mark Tester[b], Dominik L. Michels[a]

[a] *Visual Computing Center, KAUST Campus, 23955-6900, KSA*
[b] *Center for Desert Agriculture, KAUST Campus, 23955-6900, KSA*
[c] *Adobe Research, San Jose, 95110-2704, USA-CA*
[d] *UAM, Poznan, 61-614, EU-Poland*

**Abstract**

The rise of artificial intelligence (AI) and in particular modern machine learning (ML) algorithms has been one of the most exciting developments in agriculture within the last decade. While undisputedly powerful, their main drawback remains the need for sufficient and diverse training data. The collection of real datasets and their annotation are the main cost drivers of ML developments, and while promising results on synthetically generated training data have been shown, their generation is not without difficulties on their own. In this contribution, we present a paradigm for the iterative, cost-efficient generation of synthetic training data. Its application is demonstrated by developing a low-cost early disease detector for tomato plants (*Solanum lycopersicum*) using synthetic training data. In particular, a binary classifier is developed to distinguish between healthy and infected tomato plants based on photographs taken by an unmanned aerial vehicle (UAV) in a greenhouse complex. The classifier is trained by exclusively using synthetic images, whose generation process is iteratively refined to obtain optimal performance. In contrast to other approaches that rely on a human assessment of similarity between real and synthetic data, we instead introduce a structured, quantitative approach. We find that our paradigm leads to a more cost efficient development of ML-aided computer vision tasks in agriculture.

*Keywords:* Artificial Intelligence, Data Generation and Annotation,

Disease Detection, Greenhouse Farming, Machine Learning, Synthetic

Data, Tomato Plants, Unmanned Aerial Vehicles.

## 1. Introduction

Agriculture globally is more challenged now than ever before, needing to produce more food for a growing human population in the context of accelerating climate change, resource scarcity, and loss of biodiversity. These challenges will require smart, adaptable, and cost-effective technologies, which can maximize yields with minimal resource inputs. To this end, farmers are replacing traditional management practices with highly automated systems. High-tech greenhouses enable precise control of growing conditions, and computerized combine harvesters remove most of the manual effort required in the open field, significantly increasing yields per man-hour.

These advanced systems have made farmers increasingly reliant on information and communication technology for management, including wireless environmental monitoring and control systems, remote sensing via unmanned aerial vehicles, and cloud-based farm management software. The usage of these digital tools has produced large amounts of data that must be efficiently processed, analyzed, and interpreted by the farmer. To address this need, AI has emerged as an essential but still underutilized tool in modern agriculture. Indeed, the practical integration of smart systems powered by AI, and in particular by ML, will be essential to enable agriculture to be maximally resource-use efficient (FAO22).

ML is a sub-field of AI which is based on fitting various models to training data and comprises algorithms such as decision trees, support vector machines, logistic regression, expectation maximization, and neural networks. In the last ten years especially, neural networks have been tremendously successful in applications like search algorithms, suggestion systems, translation between natural languages, text-to-speech synthesis, playing logic games such as chess and Go, and a wide range of image analysis and generation algorithms including optical character recognition, face recognition, image segmentation, image style transfer and text-to-image synthesis. In all of these cases, ML-based approaches have surpassed previous non-ML based approaches. This was made possible by the availability of increasingly more powerful hardware that enabled training so-called *deep* neural networks (*deep learning*) on large datasets.

Apart from faster hardware, another factor contributing to the success in ML has been the availability of large, annotated datasets. Backends of

modern websites employ a wide range of tracking and clustering technologies to find the most relevant advertisements for each individual visitor, smart phones analyze billions of pictures and record the users movement, and several aspect of every day life is in one form or another digitized. If neural networks are considered as powerful rockets, data has to be considered as their fuel.

### *1.1. The Hard Thing about ML*

The goal of using deep neural networks is to obtain a model of a given dataset (i.e., a representation of the data far more compact than the actual amount of data). Consequently, successfully training a ML model requires three components: (1) the right architecture (i.e., the right type of network for the task and the right way to train it); (2) huge computational resources (depending on the task, whole computer clusters running for several days); and (3) an extensive amount of training data.

Thanks to the continuous research efforts and an active user community [1], many problems (such as image classification or segmentation) have established architectures that can be readily used. Although computation costs can be high for certain projects, service providers exist that provide those with a high flexibility. In practice the biggest remaining factor determining the final performance is the availability of sufficient training data.

Training data consists of examples of what the network should learn. For an image classification task, the training dataset may contain a huge number of images, each accompanied by its class. During training, the network learns the relationship between input images and output classes, and is afterwards capable of predicting a class for any given input image.

Collecting training data from the real world is very costly. Not only are many images required, they also need to be diverse and should cover all the variance that the network should learn. For example, if a network is trained on photos taken outside in the summer, it may later perform very poorly on pictures taken in the winter, where everything is covered in snow. As another example, if an additional use case is added in a later stage (e.g., also operating during the night time), a large set of new images have to be captured, making these adjustments very costly.

After collecting the data, the training data has to be labeled with ground

[1] See https://en.wikipedia.org/wiki/Kaggle .

truth information. For some tasks this may be cheap (e.g., assigning the correct class out of a selection of limited choices to each image), but it can still require the work of an expensive expert that can correctly determine the class. For other tasks, such as when pixel-precise segmentation masks should be inferred, the labeling may get very expensive. For the popular *Cityscapes* dataset widely used in autonomous driving, e.g., the labeling time required for a single image by an expert ranges from 4 minutes up to 1.5 hours, depending on the density of annotations (COR$^+$16). In some cases, data labeling can be outsourced to regions with less expensive labor costs (and service providers like *Zuru*, *Cogito Tech*, or *iMerit* offer a smooth integration of the process), but in several cases, (e.g., distinguishing between different kind of diseases on plants), domain experts may be required for reliable results and outsourcing becomes infeasible. To some degree, labeling can be speed up through specialized tools (e.g., *LabelBox*), however, these can only mitigate the cost. Even if costs can be halved, this does not change their order of magnitude.

The most promising candidate for overcoming the excessive cost of obtaining training data is the usage of synthetic training data: instead of taking photos and labeling them manually, a virtual scene is automatically generated by a computer program and then rendered into a photorealistic image. The correct label is known from the generation process and requires no additional work while being a hundred percent correct. In contrast, manually labeled data almost always includes mistakes caused by human error; generating the labels in the least amount of time using automated strategies sacrifices accuracy.

Here, the research field of computer graphics (CG) comes into play, which researched over the last decades the generation of synthetic renderings including realistic interaction between light and objects (HVDMG$^+$13). Using these algorithms, it is possible to generate images that cannot be distinguished from real photographs by humans any more. Using parameterized models, an infinite amount of different images can be created without additional human work. In the simplest case, the selection and position of objects in the scene is randomized, but in general every aspect of an object can be randomly changed while maintaining realistic results.

Plant modeling has a long history in the CG community. Their recursive structure often maps well to recursive algorithm such as *L(indenmayer)-systems* (PCFH18) and the elegance of their implementation makes them a very common topic in many introductory computer science lectures (PL90).

Individual plants have been simulated with biological precision to study different phenomena, such as the influence of the canopy to light levels (CNK14). On a larger scale, the interaction between large collection of plants and the environment they grow in has been addressed as well (MCLA$^+$17; MHS$^+$19).

In the early years of neural networks, convolutional layers inspired by biological processes where used to classify images of digits (Fuk80). The skip connections in the common *U-Net*, a fully convolutional network architecture, were initially used to segment biomedical images (RFB15). Region-based convolutional neural networks (R-CNN) were a breakthrough in instance segmentation and have been trained for general datasets (GDDM14).

Creating powerful parametric models is an expensive task in itself. But their true power is shown when a scene is adjusted for different scenarios. Modeling four different crops at day and night and in summer and winter requires 8 different models but allows for the generation of 16 different combinations. Thus, the cost for increasing the diversity in the training set grows only linearly rather than exponentially.

One way to increase the realism of synthetic renderings is to compute full global light transport using ray tracing. However, this can immensely increase the computational power required, which directly translates to added costs in hardware and power. In practice, rendering farms can be rented which support parallel rendering on thousands of computers.[2]

### *1.2. Specific Contribution*

While using real data is hard and costly, using synthetic data also comes with significant challenges. In order to get the most out of synthetic data, the cost delta compared to using real data must be maximized.

In this contribution, we present a paradigm addressing this problem. We formulate the generation of synthetic data as an iterative process where each step is guided by a human expert. The task is to estimate in each step what aspects of the renderings have to be improved in order to meet a given target quality without wasting resources on expensive but ineffective improvements. In other words, the goal is to find synthetic datasets that meet the minimal requirements to train successful deep neural network models, as this is the most cost effective solution.

[2]For comparison, the 2013 movie *Gravity* would have required 7 000 years of rendering time on a single personal computer available at that time; see https://creativechair.org/chris-parks/ .

Using this paradigm, the potential of synthetic data can be leveraged and significant cost savings reached. After a formal definition of our paradigm we demonstrate its application and effectiveness addressing a practical use-case of training a neural classifier to distinguish between healthy and diseased tomato plants (*Solanum lycopersicum*) grown in a greenhouse.

## 2. Related Work

ML has demonstrated a wide range of applications in the agricultural domain, including the management of crops, livestock, soil, and water. A comprehensive literature review of ML applications in agriculture shows that research has primarily focused on crop management (BTD$^+$21). Within this domain, ML techniques have been applied extensively to yield prediction (vKC20), crop recognition such as in (HLC20) and harvesting such as in (WBHvT$^+$17), as well as weed detection (WZW19).

A large body of ML research in crop management focuses on disease detection in plants (BTD$^+$21). This focus on disease detection is well-justified, as pests and diseases are a major challenge for agriculture and food security globally, causing an up to 40% loss in yields each year (SWP$^+$19). Early disease detection in agricultural crops enables earlier interventions that can prevent spread, saving substantial amounts of time and resources. Mitigation measures are generally more effective if applied at the early stages of disease, which also results in less pesticide used for management of the pathogen. Commercial agriculture currently relies on skilled human scouts for disease detection. Ideally scouts do daily walk-throughs, but due to costs and limited personnel, walk-throughs are typically much less frequent in practice. Manual detection methods are neither quick nor failsafe – detecting symptoms in crops requires careful attention, especially in the early stages, and costly errors are sometimes made.

Considering these challenges in manual disease detection, much attention in the past two decades has been directed to automated methods of detection, which utilize optical sensors to survey the crop and support in detection and diagnosis of plant diseases (Mah16). Such tools as RGB, multi- and hyperspectral, thermographic, chlorophyll fluorescence, and 3D imaging sensors are able to measure changes in plant physiology as the plant experiences biotic stress from disease. Common symptoms of disease in plants include leaf malformation, discoloration, and wilt. These can be detected via changes in plant or leaf temperature, reflectance, and fluorescence.

Despite advances in sensing technologies in recent decades, there are numerous challenges which limit the scope of automated disease detection applications. A main challenge is the selection of the appropriate image features (i.e., texture, color, and/or shape) which has to account for the complexity of various symptoms as well as the capturing modality that can be performed throughout the growing area. Another challenge is the development of accurate and efficient learning algorithms. Accurate classification of diseased and healthy plants in real conditions with varying light levels, shading, and complex surroundings can be extremely difficult. In addition, large image datasets in a diversity of conditions are needed to train the algorithm. *PlantVillage* is the largest and most widely studied repository of real images of diseased and healthy leaves (HS15), but its usefulness is limited by the fact that all of the images are segmented leaves with a homogeneous dark background.

Gathering real images of diseased plants at different stages of infection, but particularly at early stages of infection, is a often a challenge because of lack of available data. This challenge can be seen in (WM16), who developed a system for early detection of powdery mildew disease in greenhouse tomato in a natural setting using a camera setup with varying light settings, and Hough forests as the detection algorithm. According to the authors, the study was limited by the size of the dataset (60 images in total) that could be used for training and testing the classifier model.

Synthetic data is a promising solution for the lack of sufficient and high-quality, real training data for ML tools, and in recent years has been explored for agricultural applications. Augmentation (i.e. applying various geometry and color transformation) of real images can be understood as proto-synthetic approaches. (PKS17) use an image dataset of grass with and without weed incidence to train a neural network on weed detection. The authors apply a custom software to augment real images of a lawn, which were captured with a smartphone mounted on a robotic vehicle.

*Generative adversarial networks* (GANs) can be used as an even stronger form of augmentation. (AKS$^+$19) use a GAN to supplement traditional augmentation techniques to create an image dataset called *PlantDisease* for leaf diseases in more real-life conditions, as an alternative to the *PlantVillage* dataset. (AJGV21) utilize a *conditional* GAN (C-GAN) for image augmentation of diseased and healthy singular tomato leaves (also called "leaflets") from the *PlantVillage* dataset. Their model achieved high accuracy ($> 97\%$ mean average precision), but improving it further is made hard by the limited

amount of available input data for the GAN.

To the best of our knowledge, (WBHvT$^+$17; BIHH18) demonstrated for the first time the use of fully synthetic training data in a computer vision task in the agricultural domain when they created a synthetic image dataset of sweet peppers in a greenhouse. The authors used a few real images captured by a harvesting robot as a template to build a model based on *PlantFactory* that generates randomized instances of the plants, fruits, and backgrounds. These scenes are then rendered using *Blender*, requiring about 10 minutes of rendering time per scene on state-of-the-art hardware. With these images, it was for the first time possible to train a neural network for the segmentation of anatomical plant components without relying on excessive real data. However, considering the high amount of computation time necessary to generate the synthetic dataset, the authors point out the need for a more optimized process. Later, (TOI$^+$20) implemented a method for high-throughput crop seed phenotyping trained with synthetic images.

## 3. The Synthetic Data Pipeline

We now describe and formally define a pipeline which contains the relevant steps to train a neural network using synthetic training data. For a more thorough explanation of training neural networks in general, we refer to the literature (e.g., (GBC16)). For simplicity, we now focus on classification networks, that take a 2D image as input and outputs a discrete image class. However, our solution can also be extended to other neural networks and tasks, such as segmentation networks.

Neural networks are trained by supplying them with many examples, e.g., pairs of images and corresponding classes. During the training, they extract statistical properties from the inputs to derive the correct output. Training data can be captured from the real environment (e.g., photographs or point clouds) or synthesized (e.g., renderings). This distinction, however, is not exclusive, as real datasets usually contain some synthetic augmentation (e.g., (AJGV21)), or the synthetic renderings are generated using real images. For example, real photographs can be used as background textures as part of the rendering process (SK19).

Our method is described in Figure 1: we first define procedural models that generate the geometry of plants and accompanying textures required for rendering. We then use the procedural models to generate scenes of tomato plants in a greenhouse setting and render photorealistic images. Each image

is associated with a label, that defines whether or not a plant is diseased. A set of rendered images along with their labels is then used as a dataset for training the classification neural network. The network is trained on mini-batches of data from the dataset for a number of epochs. During training, various image-based operators are applied to individual images (e.g. brightness, contrast, etc.) - this process is called data augmentation. This results in greater variance of images in the training data. After the classification network has been trained, we validate its performance based on a dataset of real images. The resulting performance is analyzed qualitatively and quantitatively by a human expert to determine how to improve the procedural models for the next training cycle. The generation process of synthetic datasets is complex and goes along with the repeated training of the network.

### *3.1. Network Training*

Since networks are mathematical objects, the input image has to be encoded into a vector of real numbers first. This high-dimensional vector is then processed by the network and transformed into a low-dimensional output vector which can be decoded into the classification. We call the vector space of encoded input images $I$, and an individual input image $i \in I$. $R \subset I$ is the subspace of real input images, while $S \subset I$ is the subset of synthetically generated input images. $L$ is the space of all possible labels, e.g., $L = \{\mathsf{healthy}, \mathsf{infected}\}$ in case of a binary classifier distinguishing between healthy and infected plants. Such a classifier is illustrated in Figure 2.

In a dataset $D := D(J) = \{(i, G(i)) | i \in J)\} \subset \mathcal{D} := I \times L$, each image $i \in J \subset I$ is assigned a ground truth label $G(i)$ via $G : I \to L$. While $G$ maps any image (whether real or synthetic) to its correct label, $D$ consists only of a limited amount of images and their labels. The number $|J|$ of images in $J$ is typically in the range between a few thousands and a multiple of ten thousand. A network $N_w : I \to L, N(i) \mapsto p$, which is parameterized by its weights $w$, similarly maps images to predictions $p$. If $p = G(i)$, the prediction is correct.

The network consist of several linear and non-linear layers that each perform simple mathematical operations (such as computing weighted linear combinations of its input or selecting the maximum value amongst its inputs), each parameterized by the network parameters denoted as the weights. The complexity and power of the network is a result of the large number of these operations, for instance the network used in our use-case described in Section 5 has a total of 2 960 514 weights. Due to the sheer amount of

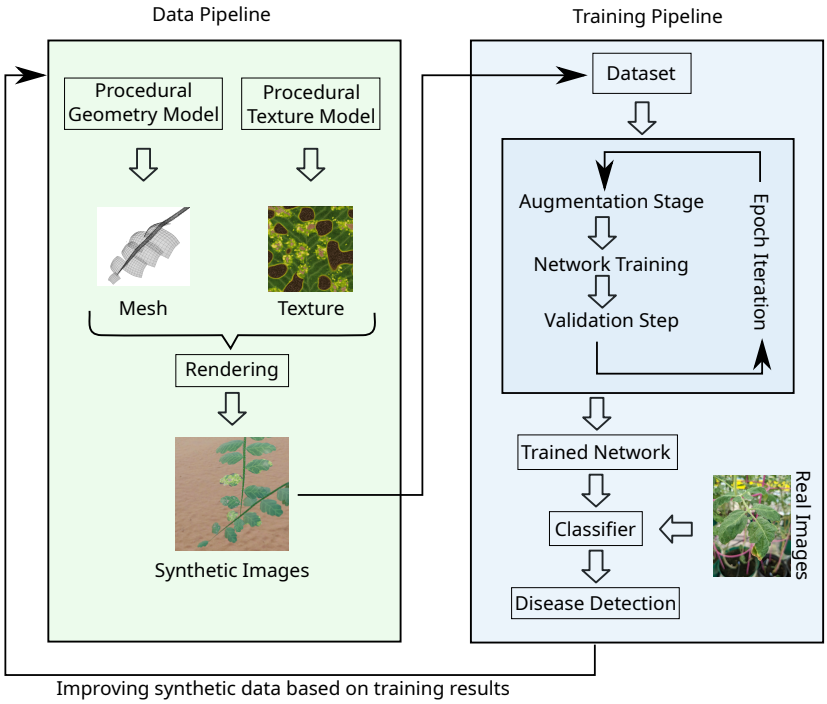


Figure 1: Illustration of our synthetic data pipeline. Left: Geometry and textures are generated, and used to render synthetic scenes. Right: A dataset of synthetic renderings is used to train a network. Based on the evaluation of the resulting classifier, the data generation is improved and the dataset regenerated for the next iteration.

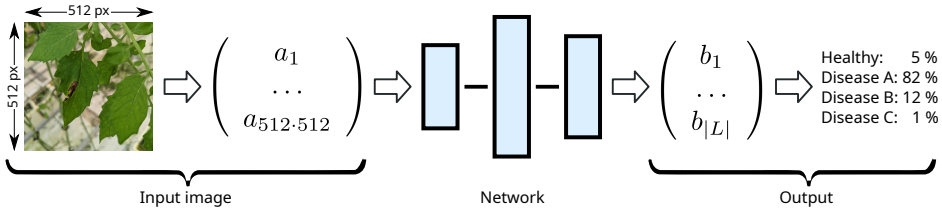


Figure 2: Illustration of ML driven classification. The input image is encoded in an input vector (which can have a very large number of components) that is then can be processed by the neural network. The resulting output vector is decoded (e.g., from a negative log-likelihood encoding), resulting in a probability per class.

weights, they cannot be chosen by hand but have to be determined through numerical optimization within the process of training the network.

During training, the task is to find the optimal weights to map the input images to the correct labels. The training is influenced by several parameters as well, called the hyper parameters $h \in H$. Note, that $h$ denotes a vector containing all hyper parameters and accordingly $H$ is the set of all possible hyper parameter combinations. The hyper parameters include the learning rate and batch size of the stochastic gradient descent optimizer, and also data augmentation parameters (see below) which have a significant impact on the learning success.

The training function $T : H \times \mathcal{D} \to (I \to L), T(h, D) \mapsto N_w$ then maps a combination of dataset and hyper parameters to a trained network with weights $w \in \mathbb{R}$. The mapping $\|\cdot, \cdot\| : L \times L \to \mathbb{R}$ is a measure of similarity between two labels. We can express $T$ as

$$T(h, D) = N_w \text{ , where } \quad w = \underset{w' \in \mathbb{R}}{\arg\min} \sum_{i \in D} \|N_{w'}(i), G(i)\| \ . \tag{1}$$

While the weights define the network parameters, its general structure is given by the so-called network architecture. The architecture describes the concrete combination (topology) of different layer types. Over the last years, suitable architectures for many types of problems (such as classification or segmentation) have been established. While the architecture could also be considered as a part of the hyper parameters, we assume it to be static for a given problem. We note however, that our framework could be adjusted for variable network architectures as well. An example of such an architecture is shown in Figure 5.

Multiple well established toolkits exist for the implementation of neural networks, such as *Keras*[3], *PyTorch*[4], and *TensorFlow*[5]. They also provide stochastic gradient algorithms than can be used to solve Equation 1 in an iterative way.

It is important to note, that the task is not to find the correct mapping $J \to L$ for images in $J$ contained in the dataset $D$ but rather a general mapping $I \to L$ for all possible input images. This is called generalization.

[3]See https://keras.io/ .

[4]See https://pytorch.org/ .

[5]See https://www.tensorflow.org/ .

If a network does not generalize well it is said to overfit on the training dataset. In the extreme case of overfitting, each $i \in J$ is predicted correctly while all $i' \in I \setminus J$ result in random labels. For a well generalizing network however, it is usually fine if some $i \in J$ are classified incorrectly, as long as $i' \in I \setminus J$ yields similar good results. The original dataset is therefore split into a training and a validation dataset.[6] While only the training dataset is used to optimize $w$, the validation dataset is used to monitor the performance on new images. Figure 11 (bottom right) shows an example of a successful generalization. While the performance on the validation dataset oscillates a bit, it is on average very similar to the performance on the training dataset.

Overfitting is often the result of a lack of diversity in the training data. This can either mean too few input images, or images that are too similar to each other (e.g., showing different objects always from the exact same angle). A common strategy to mitigate overfitting is data augmentation. During data augmentation, random alterations are performed on the image, such as geometric transformations (e.g., mirroring, rotating, zooming), color adjustments (brightness, contrast, hue), or adding noise. More advanced augmentation methods use neural networks to transform an input image into an entirely new, but similar one (AJGV21). A thorough overview of augmentation strategies is found in the literature (SK19). Since augmentation increases the diversity, it can actually reduce the network performance on the training dataset. This is however acceptable, since at the same time the performance on new images is increased. In summary, two key factors are important for a successful training: A training dataset with a large variety and the correct hyperparameters for the training. The framework presented in this paper optimizes both of them.

### *3.2. Data Generation*

Synthetic dataset are created by means of CG methods. CG is an extensive research field with a rich history (HVDMG$^+$13) dealing with the modeling and rendering of virtual scenes. The three main components that need to be modeled are geometry, materials, and scene composition (object positions, lighting, camera). Examples are shown in Figure 3.

[6] To avoid overfitting the hyperparameter $h$ to training and validation datasets, a third dataset is commonly used at very end of the training procedure to measure the unbiased performance of the network.

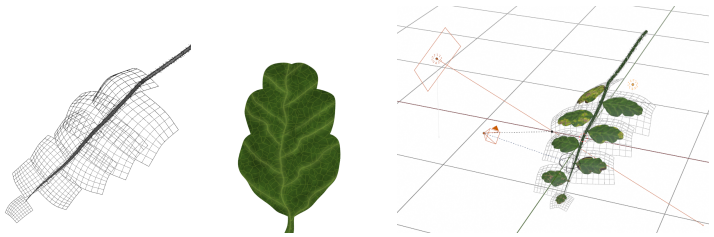

Figure 3: Illustration of the main components of a synthetic scene. Left: Object geometry. Middle: Materials. Right: Scene composition and lighting.

All these components can either be created by an artist by hand or automatically generated through a procedural model. Instead of defining properties (such as the outline of a leaf or the branching structure of a plant) by hand, procedural modeling defines rules that depend on various parameters, and can be instantiated to create geometry. An example of such a procedural model in the agricultural context are L-systems for plant geometry and node-based texture synthesis of materials (Pai19). By varying the input parameter vector, and endless amount of images can be created. However, this does not mean that the diversity is sufficiently high. The instantiation of procedural parameters essentially resembles an interpolation. If a larger portion of the total image space $I$ should be covered, additional parameters have to be added.

## 4. Using Synthetic Data at Scale

The main bottleneck of the procedure described in the previous section is the generation of a suitable training dataset $S$ and the finding of the correct hyperparameters $h$ at minimal development cost. The cost factor deserves special attention here since the main argument for the use of synthetic training data is their cost effectiveness. In order to use synthetic data at scale, i.e., being able to apply the previously described procedure for a large number of automation tasks in agriculture, we introduce a paradigm that addresses this problem by taking a holistic approach. Like the network training itself, finding $S$ and $h$ is formulated as an iterative optimization scheme where in each step $S$ and $h$ are gradually improved.

The first step is to determine a suitable target quality $q_{\min}$. We measure the performance $q = [N(R), G(R)]$ of a network $N$ on a real dataset $R \subset I$

using a measure $[\cdot,\cdot] : L^{|R|} \times L^{|R|} \to \mathbb{R}$ which is, e.g., defined as the $F$ score or the $P_4$ metric in case of binary classification (MRS09). Without loss of generality, we assume that larger values of $q$ are better. Note, that the network is trained on annotated synthetic data $D = D(S)$ but evaluated on real data $R$. Our goal is then to find a pair $(h, S) \subset H \times I$ such that

$$[T(h, D(S))(R), G(R)] =: q \geq q_{\min}.$$

Without consideration of any cost, $h$ could be found by a brute-force parameter sweep, while for $S$ renderings with the highest degree of photorealism and variability could be used. But if we assume that the quality is proportional to the invested cost, the goal becomes finding the worst pair $(h, S)$ which still satisfies $q \geq q_{\min}$. We address this iteratively. An iteration step

$$\mathcal{I}_k : H \times D \to H \times D, (h_k, S_k) \mapsto (h_{k+1}, S_{k+1})$$

refines the hyperparameters and the dataset (though it is not required that both change in each iteration). Each iteration $\mathcal{I}_k$ is associated with a certain cost measured by the cost function $C(\mathcal{I}_k)$. The overall optimization problem is then to find the sequence $(\mathcal{I}_1, \ldots, \mathcal{I}_n)$ of iterations with minimal cost that yields the desired quality:

$$\underset{(\mathcal{I}_1,\ldots,\mathcal{I}_n)}{\arg\min} \sum_{k=1}^{n} C(\mathcal{I}_k) \text{ , subject to } \; q \geq q_{\min}. \tag{2}$$

Here we see why a reasonable choice of $q_{\min}$ is important: According to the Pareto principle, if $q_{\min}$ is too large, this results in an excessive amount of iterations with exponentially growing costs. Knowing what quality is acceptable is crucial to minimizing cost. Solving Equation 2 cannot be performed automatically through naive numerical optimization. Rather, every iteration step $k$ requires the guidance of a human expert. The solution is typically obtained by maximize the quality gradually at every step.

Taking a closer look at $C_k := C(\mathcal{I}_k)$, shows that it consist of several components:

$$C_k = C_k^{\mathrm{E}} + C_k^{\mathrm{M}} + C_k^{\mathrm{R}} + C_k^{\mathrm{T}},$$

where $C_k^{\mathrm{E}}$ is the cost of evaluating the previous iteration required for deciding on the next changes, $C_k^{\mathrm{M}}$ is the modeling cost to improve the generator for the synthetic images (performed by an artist), $C_k^{\mathrm{R}}$ is the required rendering

costs for the new dataset (often outsourced to a rendering farm and payed per core minute) and $C_k^{\mathrm{T}}$ is the cost of training a new network with the improved hyperparameters and dataset. The cost for changing the hyperparameters is entirely contained in $C_k^{\mathrm{E}}$, since they are a simple vector that requires no modeling time. For a brute force search of the best hyperparameters the total cost is dominated by $C_k^{\mathrm{T}}$ since the dataset remains the same ($C_k^{\mathrm{M}} = C_k^{\mathrm{R}} = 0$) and $C_k^{\mathrm{E}}$ is minimal (as it only consist of a sampling strategy for $h$). Often the most expensive step is to improve the dataset since $C_k^{\mathrm{M}}$ and $C_k^{\mathrm{R}}$ are typically large. It can be beneficial to split such an iteration into multiple sub-iterations, which introduces additional $C_k^{\mathrm{T}}$, but gives an overall better understanding of the required changes.

## 5. Early Disease Detection for Tomato Plants

The previously described paradigm is now applied in order to develop a neural classifier for early disease detection of tomato plants (*Solanum lycopersicum*). This use-case is not only well suitable to demonstrate our paradigm but also addresses an important practical problem. Especially in monocultures found in greenhouses, diseases can spread rapidly and can quickly become uncontrollable (SWP$^+$19). Detecting them as early as possible greatly increases the chance of such a catastrophical crop failure but requires constant and expensive monitoring. Any step toward automatizing this process is therefore a great benefit.

We use a UAV patrolling through rows of the greenhouse complex in order to capture images of the tomato plants as illustrated in Figure 4. As the tomato plants grow up to remarkable sizes (typical values are up to 10 m), we prefer to use UAVs instead of self-driving vehicles patrolling through rows. This also comes with low hardware costs as the price of our *DJI Mini 3 Pro* is below USD 1 000. This UAV is also sufficiently small in size to fly through the rows of our greenhouse. For larger greenhouse complexes, multiple UAVs can be used, e.g., a single drone per row.

In this section, we focus on the binary classifier which groups pictures of the leaves into two classes containing healthy and infected leaves ($|L| = 2$). Potential infections can then be reported to a human overseer who can confirm or reject them. Reducing the need for manually checking the entire greenhouse complex to checking only a few candidates greatly reduces cost even if the detection rate is not perfect. Overall we aim for an accuracy of $q_{\min} \approx 90\%$.

Figure 4: Illustration of the image collection process in the greenhouse complex hosting tomato plants (*Solanum lycopersicum*). Left: An autonomously flying UAV patrols through each line, taking photos. Middle: Example of a healthy leaf. Right: Example of an infected leaf.

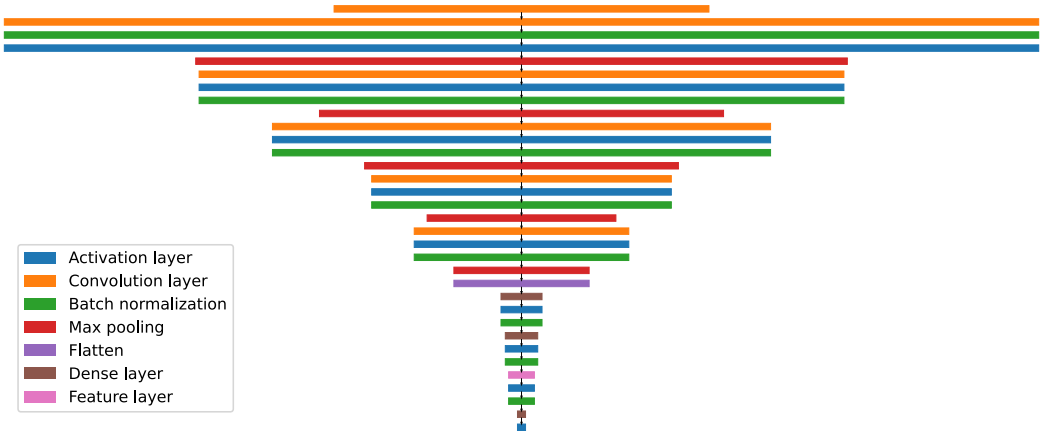


Figure 5: Simplified illustration of the layered architecture of our classification network. Each layers width corresponds to the cubic root of its dimensionality. The input image (top row) is expanded into multiple parallel filters and throughout the network their size consecutively shrinks until a single value denoting the classification remains. The total number of weight in this network is $|w| = 2\,960\,514$.

Figure 6: Illustration of geometry and textures of the first iteration of synthetic data. Left: The geometry of the branches is procedurally generated and two examples are shown. Middle: Textures for healthy and infected leaves are generated. The infected textures are generated from the healthy ones by adding typical patterns of dead leaf cells. Right: A final rendering of a textured branch in the scene.

We implement our neural classifier in *Keras* using a standard classification network architecture as shown in Figure 5. To measure the training loss $\|\cdot,\cdot\|$ we use the *categorical cross-entropy* loss function as implemented in *Keras*. As performance measure $[\cdot,\cdot]$, we divide the number of falsely labeled images by the total number of images. We start with an initial choice of $h_1$ and a simple initial dataset $D_1 := D(S_1)$ and refine it over the course of a total of $n = 6$ iterations generating $D_2 := D(S_2), \ldots, D_6 := D(S_6)$ and hyperparameter $h_2, \ldots, h_6$ to reach our target accuracy. After each iteration, an extensive evaluation is required to make an informed decision about the next changes in $h$ and $D$ (which is the reason why this evaluation is included as the cost $C_k^{\mathrm{E}}$). We monitor the achieved performance on the synthetic training and validation datasets as well as on a real dataset $R$. Moreover, we take a closer look at the performance on individual images which helps us to understand what additional features have to be modeled in the synthetic images.

### *5.1. Iteration $k = 1$*

The initial dataset is shown in Figure 6. To generate synthetic plant geometry, we have implemented a node-based procedural modeling system as commonly used L-systems for tomato plants (CNK14) do not aim for the level of realism required for our task. The model has a large number of parameters including the number, size, and orientation of leaves, as well as bending and length of the branch. The leaf textures are generated using *Adobe Substance*

*3D Designer*[7]. We generate the typical set of physically-based rendering (PBR) textures which include layers such as a diffuse albedo map, a normal map, an ambient occlusion map, and a height map (HVDMG$^+$13). With these layers, we do not only model the color of the leaves but also the physical interaction of light with the leaf material, which greatly enhances realism. The scene consists of a single branch with leaves and a random high dynamic range (HDR) panorama photo captured in the greenhouse as background. This panorama also illuminates the scene, meaning that it is illuminated by the same lighting conditions as the plants in the greenhouse. Although the generated renderings look plausible and detailed, they do not look completely photorealistic. A human may initially be fooled to think they are real images, but in comparison with actual photographs the differences become visible.

For the hyperparameter $h_1$ we chose values typical for a binary classification task: The input resolution is $256 \times 256$, the batch size is 16 and the learning rate is $10^{-4}$. For the augmentation, we chose a simple combination of zooming, brightness adjustment, flipping, and rotation. Examples of augmented images are shown in Figure 7.

The initial dataset consists of $|S_1| = 3\,400$ images, where half of them show healthy and half of them show infected leaves. Around 10% of the images are used for the validation dataset. Monitoring the performance on both synthetic datasets shows that the network does not overfit as shown in Figure 11, bottom right. This means, that the dataset is sufficiently large and by adding more images we likely would not see an increase in quality. This information helps to cap the cost $C_1^{\mathrm{R}}$.

We also evaluate the performance on real data as shown in Figure 11. We find that almost all images are classified as healthy regardless of their actual class which means that we learn almost nothing about the true class.

### *5.2. Iteration $k = 2$*

After reviewing the results we conclude that the size of the dataset, the network architecture, and training hyperparameters are fine (since the accuracy on the validation dataset is high), but that the domain gap between real and synthetic images is too large and the network cannot generalize to the real data. Since enhancing the realism of the procedural model is a very time-consuming task, we first try faster changes to shrink to domain gap.

[7]See https://www.adobe.com/ .

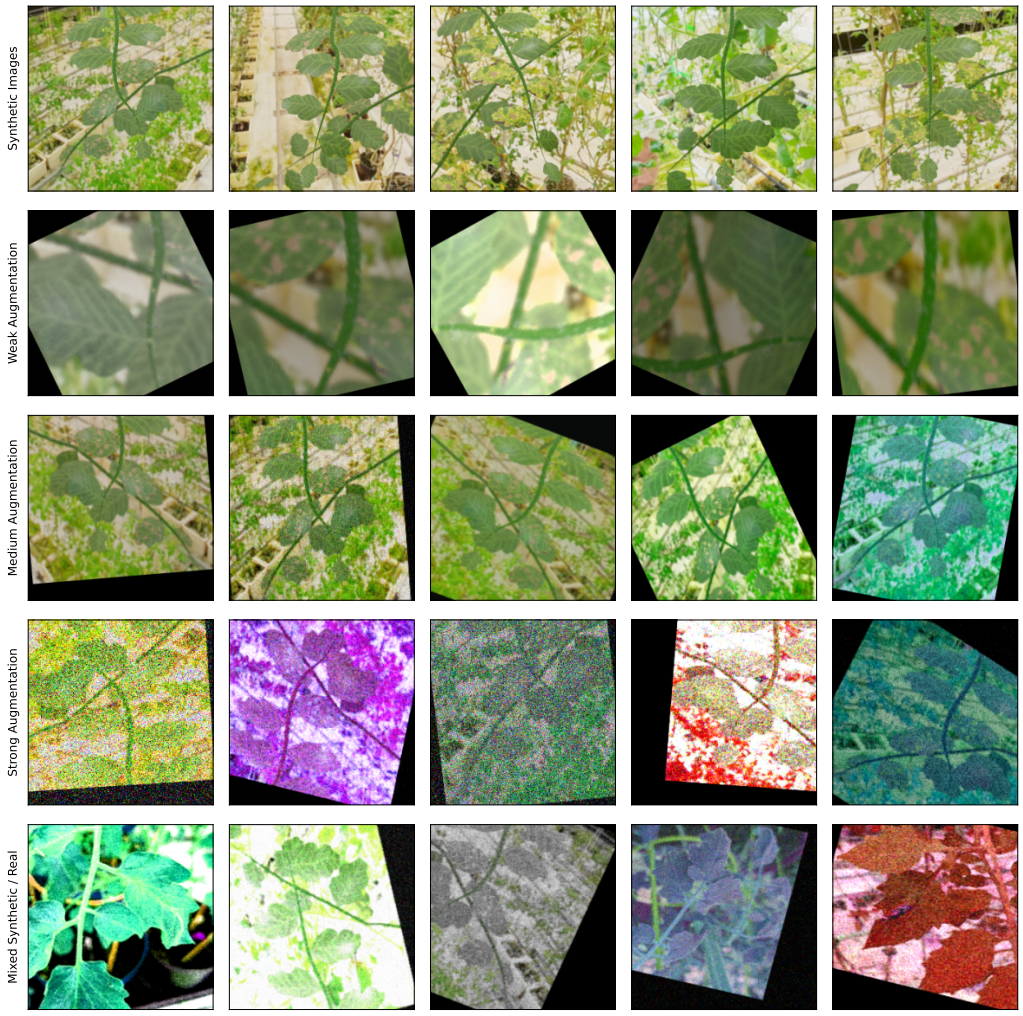


Figure 7: Comparison of augmentation modes. First row: Five different synthetic renderings. Second to fourth row: The first image of the first row augmented 5 times for each of the 3 augmentation modes (weak, medium, and strong). Fifth row: A random selection of real and synthetic images augmented with strong augmentation. Through the augmentation process, it becomes difficult to distinguish between real and synthetic images, thus the domain gap between both sets is reduced.

We add a second, slightly defocused branch to the background of each image without changing the branch generation itself. This is done to mimic the cluttered environment of the greenhouse and to make the network invariant to camera focus. In total, we render $|S_2| = 2\,472$ new images. We also increase the augmentation in $h_2$: Referring to Figure 7, we add Gaussian blur, contrast adjustment, hue shifting, and additive Gaussian noise, and retrain the network. This indeed improves the detected of infected leaves, but not by an sufficient amount as shown in Figure 11.

### *5.3. Iteration $k = 3$*

Since increasing the augmentation is cheap and gave good results in the previous iteration, we now increase it even further for $h_3$. We continue to use the same operations but with a larger variety of parameters. As seen in Figure 7, the look of the images is quite drastically altered now. The dataset remains the same as in the previous iteration, thus $|S_3| = |S_2|$. After retraining the network, we find that the accuracy improves only marginally (Figure 11), indicating that we have to proceed in a different direction.

### *5.4. Iteration $k = 4$*

It is now clear, that our synthetic images are too different from the real photos. However, many things could be improved about the renderings: We could have more variety in the branch geometry, increase texture details, or model more complex scenes (e.g., creating geometry for the whole greenhouse and a large amount of plants instead of a single branch in front of background panoramas). Implementing all of these improvements would be prohibitively costly, so instead, we perform a detailed analysis on which images are classified well. Since healthy plants are usually classified as healthy, we focus on images of infected plants.

Figure 8 shows the accuracy for 19 different input images across the iterations. We find that the distribution is extremely uneven: Some images are repeatedly classified correctly while others are almost never. Comparing the real input images with our renderings (see Figure 9), we find that infection can alter the leaf textures in many different ways. Infections resembling the type that we initially modeled are then classified correctly, while other types of patterns are not detected. We therefore improve our texture creation pipeline by adding additional disease types. The new synthetic disease textures are also shown in Figure 9. Since we have significantly increased the

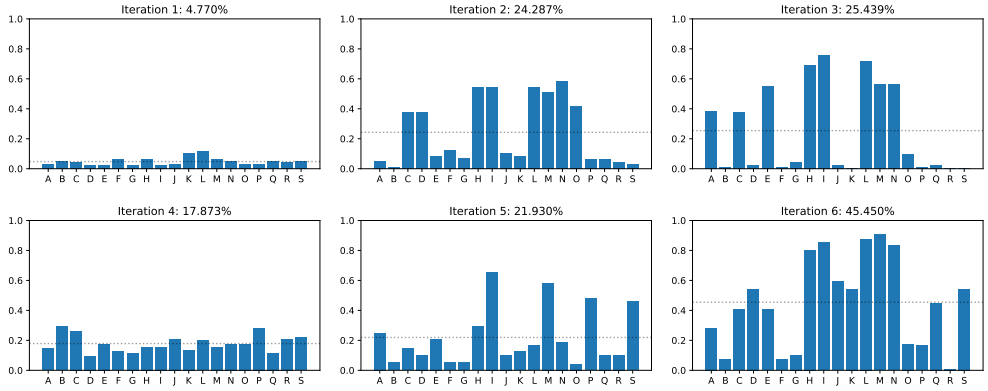


Figure 8: Network accuracy on real images of infected plants for each iteration. The letters on the horizontal axis denote the individual images. The dotted line shows the average accuracy across all images.

variety of the dataset, we increase to total number of images to $|S_4| = 6\,400$. The hyperparameters stay the same, thus $h_4 = h_3$.

While training the network we find that the accuracy even on synthetic data is very low. The increased diversity in the dataset makes the training significantly harder.

### *5.5. Iteration $k = 5$*

To make the training easier without reducing the diversity of the dataset, we reduce the image augmentation in $h_5$ again to the previous level as in $h_2$. The dataset stays the same as in the previous iteration, thus $|S_5| = |S_4|$. The training works and we get an overall accuracy similar to iteration 3 (Figure 11). However, when looking at the performance of individual images (Figure 8), we find that the distribution is more even than for iteration 3. This means, that the modeling of additional diseases has payed off.

### *5.6. Iteration $k = 6$*

We attribute the remaining inaccuracies in the classification to the different global look of the renderings and photos. This could be addressed by stronger augmentation, however in iteration 4 we saw that a too difficult dataset makes the training harder. We therefore change the training strategy in $h_6$ and employ a mixed training model, without changing the dataset

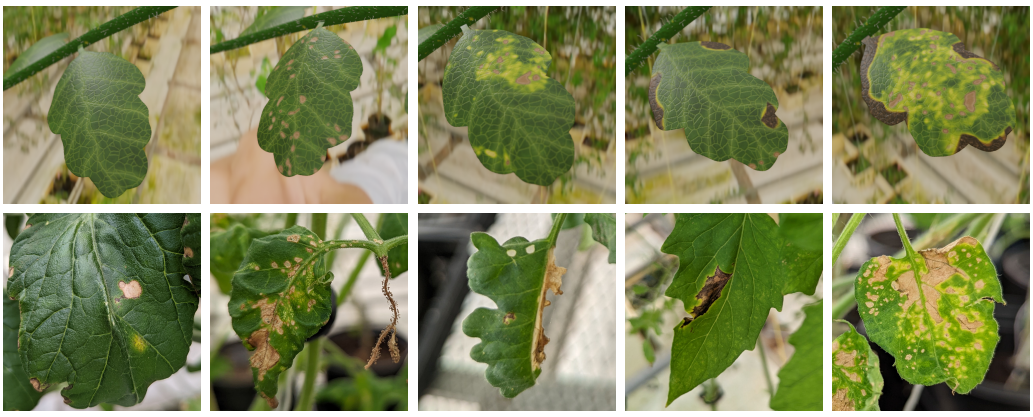

Figure 9: Different types of diseases affecting the leaf texture of these tomato plants (*Solanum lycopersicum*). Top row: Synthetic images. Bottom row: Photographs.

($|S_6| = |S_5|$). During the first half of the training, medium augmentation as in $h_2$ is used. Once the network works sufficiently well, we switch over the stronger augmentation of $h_3$. This results in an initial drop of the accuracy (since the problem became harder), but eventually the half trained network can adjust to the stronger augmentation and reach a high accuracy on them.

### *5.7. Improving the Classification Results*

After the network training, we now perform a deeper statistical analysis of the results. During the training with synthetic renderings, we used augmentation to increase the diversity and to mimic artifacts found in real images but not the renderings (blurring, noise, etc.). As seen in Figure 7, the augmentation can be quite strong. Therefore, when using the trained network for classification, the same augmentation as during the training should be applied before passing the images to the network. However, since the photos already contain artifacts mimicked by the augmentation, this would in some sense result in a double augmentation. To decide, which augmentation mode should be applied for real images, we perform an analysis, where each image is classified multiple times (since the augmentation parameters are chosen at random, every time). The results are shown in Figure 10.

We find, that there is no clear best performing augmentation mode for real input images but that the results rather depend on the input image. We further find, that healthy input images are classified as healthy in over

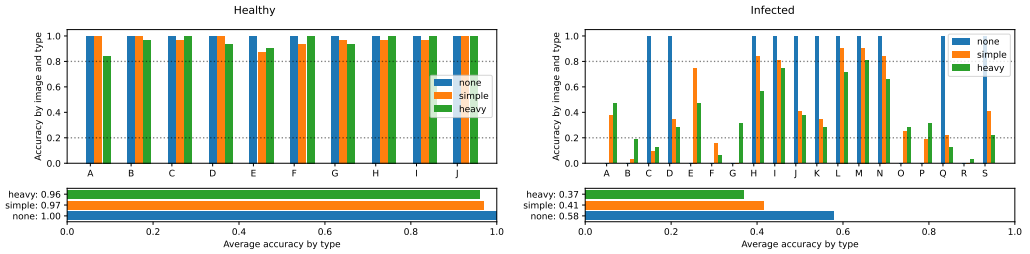


Figure 10: Network prediction performance by input image for different augmentation modes. For each mode, the input image was augmented 32 times with random parameters according to the given augmentation mode. The number of correct classifications are added up. The vertical scale shows normalized accuracy. Note, that since the *none* augmentation mode does not alter the image, either all or no instances of the image are correctly classified since the network is deterministic.

80% of the cases, independent from the augmentation mode. Furthermore, for 16 out of the 19 infected images, at least one augmentation mode lies above the equivalent threshold (20%). We conclude, that the network is biased towards the healthy case. But by taking the estimated accuracy into account, this bias can be corrected. If we consider for any input image a value of below 80% healthy score (equivalent to an above 20% infected score) as infected, then 26 out of 29 images are correctly classified. We therefore reach an overall accuracy of 89.6%, which is roughly equal to the initial $q_{\min}$, ending our optimization. Without this analysis, the naive threshold would be at 50%, leading to an accuracy of only 75%. In some sense, we apply a post-processing to the network's output to increase the accuracy – similar to the pre-processing of the inputs in the augmentation step.

### *5.8. Further Comments*

A summary of the cost and performance of each iterations is shown in Figure 11. It is important to note, that the cost types are separated into two distinct categories. $C^E$ and $C^M$ directly relate to working hours of an expert and are thus typically very expensive. In contrast, $C^R$ and $C^T$ relate to computational time of computers. For scenarios like our use-case, they can often run over night and thus do not stall the general development if scheduled carefully. However, for larger datasets and more complicated

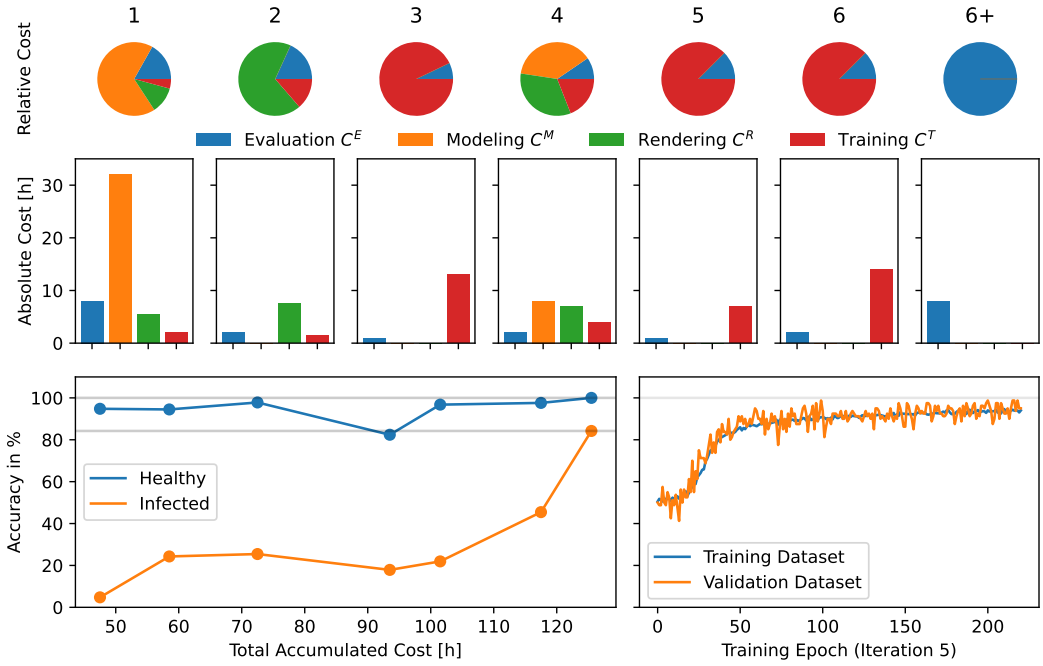


Figure 11: Development cost and performance of each iteration. Top row: Relative cost of the different cost types for each iteration. Middle row: Absolute values of the cost in hours. $C_1^E$ includes the initial development of the classifier framework. $C_{6+1}^E$ include the statistical analysis described in Section 5.7. Bottom left: The accuracy $q = [T(h, D(S))(R), G(R)]$ of the classifier plotted over its development time. The dots mark the individual iterations. Bottom right: The training process of iteration 5 shows, that the accuracy on the validation dataset closely follows the accuracy on the training dataset indicating that the network generalizes well. The lines show the average of the healthy and infected classes. Note, that the lowest accuracy for a binary classification is 50% which corresponds to random guessing of the class.

training $C^R$ and $C^T$ can also become very expensive, for instance, the training cost of the recently released *Stable Diffusion* network (RBL+22) was about USD 600 000.[8]

In this use-case we trained a network to the desired accuracy in only $n = 6$ iterations. Out of those, only one included a redesign of the dataset. We can see several key points here: Firstly, the dataset is not optimized for photorealism but rather for the distinction between healthy and infected leaves. Secondly, this crucial information became available through thorough evaluation. In other words, increasing $C_k^{\mathrm{E}}$ can over proportionally decrease $C_k^{\mathrm{M}}$, $C_k^{\mathrm{R}}$, and $C_k^{\mathrm{T}}$, leading to an overall lower cost $C$. And thirdly, a good understanding of the behavior of the trained network can be used to increase its performance.

Using the presented paradigm, total costs of about $C = 125.5$ h have been invested to develop our classifier comprising approximately $C^E + C^M = 64$ h of human work and 61.5 h of computation. For comparison, we estimate the total costs of human work without applying the presented paradigm. Based on our extensive previous work comprising plant modeling, simulation, and rendering, we estimate the development time of a fully photo-realistic plant generator to be at least three months for a single expert. In this unguided approach, no continuous, quantitative feedback based on the intermediate network performance would be provided during the development and thus all visual features would be addressed with equal importance. This lack of prioritization then severely impacts the efficiency, driving up overall development cost.

## 6. Conclusion

In this contribution, we have presented a paradigm for the development of synthetic training data in order to efficiently automatize agricultural tasks with ML. While it is to some extend straightforward to create "good" training data, it is much more difficult to do so in a cost efficient way. We have demonstrated, that using our paradigm, the desired goal can be achieved by a small amount of only six iterations in our use-case. Importantly, we find that photorealism (which is expensive to achieve) is not the main quality driver of the trained network. Rather, most iteration steps consist only of

[8] See https://twitter.com/emostaque/status/1563870674111832066 .

small changes that optimize the data for the distinction between the different classes, rather than overall realism. Naturally, our paradigm is driven by a human expert. It is therefore less of a plug-and-play solution but rather a development philosophy enabling the efficient and effective use of synthetic data. In future work, we aim to further automatize different steps within the development process to boost efficiency and reduce the time spent by the human in the loop.

Based on our paradigm, a neural classifier could be efficiently developed for the early detection of infections in our greenhouse complex growing tomato plants (*Solanum lycopersicum*). Total costs of about 125.5 h have been sufficient to develop the classifier within our paradigm which only includes approximately 64 h of human work (evaluation plus modeling costs) and 61.5 h of computation (rendering plus training costs). Note, that these costs are only a very small fraction of the effort of the research project presented here as – next to the formalization of the paradigm which emerged from the experience with different use cases – we developed the corresponding technical routines to allow for an efficient workflow. Also, the training time of the developer who has to become familiar with these routines and working within the presented paradigm, is not included. Our classifier performs with an accuracy of about 90% significantly reducing the need for manual checking of the entire greenhouse complex. Using UAVs, our final early disease detection method for tomato plants can be implemented in greenhouse complexes at low costs. However, our classifier comes with limitations as infected leaf textures have been generated from healthy ones by adding typical patterns of dead leaf cells. If, e.g., a disease is mainly visible at an early stage by looking at the branches instead of the leaves, it is not sufficient to only focus on leaf textures, but instead more investments have to be made to model the implications on the branches. This could, e.g., require the modeling of wilting effects influencing the whole plant geometry and not only the leaves' textures. This is why, among others, we aim for an efficient simulator of plant wilting in future work addressing geometrical features of plant diseases in addition to those which could already be modeled by modifying leaf textures.

**Acknowledgements**

This work was funded by KAUST's Competitive Research Grant program.

## References


[AJGV21] Amreen Abbas, Sweta Jain, Mahesh Gour, and Swetha Vankudothu. Tomato plant disease detection using transfer learning with c-gan synthetic images. *Computers and Electronics in Agriculture*, 187:106279, 8 2021.

[AKS+19] Marko Arsenovic, Mirjana Karanovic, Srdjan Sladojevic, Andras Anderla, and Darko Stefanovic. Solving current limitations of deep learning based approaches for plant disease detection. *Symmetry 2019, Vol. 11, Page 939*, 11:939, 7 2019.

[BIHH18] R. Barth, J. IJsselmuiden, J. Hemming, and E.J. Van Henten. Data synthesis methods for semantic segmentation in agriculture: A capsicum annuum dataset. *Computers and Electronics in Agriculture*, 144:284–296, 2018.

[BTD+21] Lefteris Benos, Aristotelis C. Tagarakis, Georgios Dolias, Remigio Berruto, Dimitrios Kateris, and Dionysis Bochtis. Machine learning in agriculture: A comprehensive updated review. *Sensors 2021, Vol. 21, Page 3758*, 21:3758, 5 2021.

[CNK14] Tsu-Wei Chen, Thi My Nguyet Nguyen, and Katrin Kahlen. Quantification of the effects of architectural traits on dry mass production and light interception of tomato canopy under different temperature regimes using a dynamic functional-structural plant model. *J. Exp. Bot*, 65(22):6399–6410, 2014.

[COR+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[FAO22] FAO. In Brief to The State of Food and Agriculture 2022: Leveraging automation in agriculture for transforming agrifood systems. Technical report, Rome, 2022.

[Fuk80] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 1980.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

[HLC20] Gwo-Jiun Horng, Min-Xiang Liu, and Chao-Chun Chen. The smart image recognition mechanism for crop harvesting system in intelligent agriculture. *IEEE Sensors Journal*, 20(5):2766–2781, 2020.

[HS15] David. P. Hughes and Marcel Salathe. An open access repository of images on plant health to enable the development of mobile disease diagnostics. 11 2015.

[HVDMG+13] John F. Hughes, Andries Van Dam, Morgan Mc Guire, David F. Sklar, James D. Folez, Steven K. Feiner, and Kurt Akeley. *Computer Graphics - Principles and Practice, 3rd Edition*. Addison-Wesley, 2013.

[Mah16] Anne Katrin Mahlein. Plant disease detection by imaging sensors – parallels and specific demands for precision agriculture and plant phenotyping. *Plant Disease*, 100:241–254, 2 2016.

[MCLA+17] Amy Marshall-Colon, Stephen P. Long, Douglas K. Allen, Gabrielle Allen, Daniel A. Beard, Bedrich Benes, Susanne von Caemmerer, A. J. Christensen, Donna J. Cox, John C. Hart, Peter M. Hirst, Kavya Kannan, Daniel S. Katz, Jonathan P. Lynch, Andrew J. Millar, Balaji Panneerselvam, Nathan D. Price, Przemyslaw Prusinkiewicz, David Raila, Rachel G. Shekar, Stuti Shrivastava, Diwakar Shukla, Venkatraman Srinivasan, Mark Stitt, Matthew J. Turk, Eberhard O. Voit, Yu Wang, Xinyou Yin, and Xin-Guang Zhu. Crops in silico: Generating virtual crops using an integrative and multi-scale modeling platform. *Frontiers in Plant Science*, 2017.

[MHS+19] Miłosz Makowski, Torsten Hädrich, Jan Scheffczyk, Dominik L. Michels, Sören Pirk, and Wojtek Pałubicki. Synthetic silviculture: Multi-scale modeling of plant ecosystems. *ACM Trans. Graph.*, 38(4), jul 2019.

[MRS09] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge UP, 2009.

[Pai19] Hong-Yi Pai. Texture designs and workflows for physically based rendering using procedural texture generation. *IEEE Eurasia Conference on IOT, Communication and Engineering*, 2019.

[PCFH18] Przemyslaw Prusinkiewicz, Mikolaj Cieslak, Pascal Ferraro, and Jim Hanan. Modeling plant development with l-systems. *Mathematical Modelling in Plant Biology*, 2018.

[PKS17] Larry Pearlstein, Mun Kim, and Warren Seto. Convolutional neural network application to plant detection, based on synthetic imagery. *Proceedings - Applied Imagery Pattern Recognition Workshop*, 8 2017.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, 1990.

[RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention (MICCAI), Springer*, 2015.

[SK19] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deeplearning. *Journal of Big Data*, 2019.

[SWP+19] Serge Savary, Laetitia Willocquet, Sarah Jane Pethybridge, Paul Esker, Neil McRoberts, and Andy Nelson. The global burden of pathogens and pests on major food crops. *Nature Ecology  Evolution 2019 3:3*, 3:430–439, 2 2019.

[TOI+20] Yosuke Toda, Fumio Okura, Jun Ito, Satoshi Okada, Toshinori Kinoshita, Hiroyuki Tsuji, and Daisuke Saisho. Training instance segmentation neural network with synthetic datasets for crop seed phenotyping. *Communications Biology 2020 3:1*, 3:1–12, 4 2020.

[vKC20] Thomas van Klompenburg, Ayalew Kassahun, and Cagatay Catal. Crop yield prediction using machine learning: A systematic literature review. *Computers and Electronics in Agriculture*, 177:105709, 2020.

[WBHvT+17] C. Wouter Bac, Jochen Hemming, B. A. J. van Tuijl, Ruud Barth, Ehud Wais, and Eldert J. van Henten. Performance evaluation of a harvesting robot for sweet pepper. *Journal of Field Robotics*, 34(6):1123–1139, 2017.

[WM16] Patrick Wspanialy and Medhat Moussa. Early powdery mildew detection system for application in greenhouse automation. *Computers and Electronics in Agriculture*, 127:487–494, 9 2016.

[WZW19] Aichen Wang, Wen Zhang, and Xinhua Wei. A review on weed detection using ground-based machine vision and image processing techniques. *Computers and Electronics in Agriculture*, 158:226–240, 2019.