Consultants in Quantitative Methods

Generative modeling for digital twins, a use-case

Project number **CQM internal**

Project name **ASIMOV**

Author **Rutger van Beek**

Date **02-11-2023**

Reference **RAP/23185/vBe/rb**

Summary

This report contains research executed by CQM as part of the ASIMOV project [1]. ASIMOV is collaborative research project running under the ITEA4 programme within the Eureka framework. ASIMOV stands for AI training using Simulated Instruments for Machine Optimization and Verification. The projects consists of research on building digital twins of Cyber-Physical-Systems and using AI in these digital twins to optimize the systems. This project is part of and made possible by ASIMOV.

In this research, the use of deep generative models for building digital twins is investigated. Generative modelling is the counterpart of a discriminative model. It models the data-generating process and is often used to generate new data. Examples of generative modelling are fitting probability distributions as a model, but also chatGPT that is generating text. A generative model is called deep when it uses neural networks. Digital twins are digital models of physical systems that get input from and give output to a physical system. Digital twins have many use-cases for the optimization of physical systems.

In this report, two deep generative models are explained: Generative Adversarial Networks (GANs) and Variational AutoEncoders (VAEs). Both models are tested in a use-case. The use-case consist of a machine operating on leaves. Several algorithms determine how the machine operates on these leaves. One can make a digital twin of this machine, but one would need many test leaves to test the algorithms. By using deep generative models, an infinite amount of test leaves can be generated.

It is hard to measure the performance of a generative model in general, which also became apparent in this research. There are many ways to measure the performance of a generative model. For the use of a (deep) generative model in a digital twin, the most important way seems to be to identify key properties through which the samples influence the digital twin and to check whether the distributions of these properties are similar to those obtained from the data.

On the use-case, the GAN seemed to outperform the VAE in terms of the quality of the samples. However, the training process of the GAN is much more involved. Tuning the hyperparameters of the GAN is hard and requires many trials. Training a VAE on the other hand is relatively straightforward. These results seems to be general and have been observed many times in the literature.

Contents

1. Introduction

Generative modeling is a type of machine learning that involves training an algorithm to generate new data samples that are similar to existing data [2]. Unlike traditional machine learning algorithms that are designed to recognize patterns and classify data, generative models aim to create new data that is similar to the input data. This approach has numerous applications, ranging from generating realistic images and videos to creating natural language text and music [3].

Generative modeling is becoming increasingly popular due to recent advancements in deep learning algorithms, which have enabled the creation of highly complex and realistic generative models. These models use large amounts of data to learn the underlying patterns and structures that define a particular dataset, and then use that knowledge to generate new data that closely resembles the original data. This has led to significant breakthroughs in a variety of fields, including computer vision, natural language processing, and audio synthesis. Overall, generative modeling is a powerful. By learning from existing data and creating new data that closely matches it, generative models offer a promising way to generate highly realistic and diverse content in a variety of domains.

ASIMOV [1] is a collaborative research project running under the ITEA4 programme within the Eureka framework. The project is aimed at developing a simulation-based platform for training AI models using simulated instruments for machine optimization and verification. The project focuses on using digital twin technology and simulation models to provide a realistic and safe environment for training and validating AI models.

A digital twin is a virtual replica of a physical system or process that can be used to simulate, monitor, and optimize its performance. Digital twins are becoming increasingly popular in a variety of industries, including manufacturing, aerospace, and healthcare, as they allow engineers and operators to gain insights into the performance of a system without having to physically interact with it [2]. By using sensors, data analytics, and machine learning algorithms, digital twins can provide real-time information about a system's behavior and predict its future performance. This information can be used to optimize the system, improve its efficiency, and reduce maintenance costs. Overall, digital twins offer a powerful tool for improving the performance and sustainability of physical systems in a wide range of applications. In ASIMOV, the focus is on digital twins of High-tech Cyber-Physical Systems (CPSs). The digital twins are used to optimize the systems and train AI models for operating them.

One way generative models can be used in a digital twin is by generating synthetic data to augment real-world data. In many cases, obtaining real-world data can be challenging and/or expensive. However, by using generative models, it is possible to generate synthetic data that is similar to real-world data, allowing for more comprehensive and efficient training of AI models. Another application of generative modeling in digital twins is the generation of scenarios that can be used to test the performance of the physical system. For example, a generative model could be used to simulate a variety of weather conditions, allowing engineers to test how the system performs under different weather scenarios. In general, with generative modelling, more variation can be introduced into the Digital Twin. This means that algorithms and AI models are more robustly optimized.

This report summarizes the application of generative modelling for a digital twin in theory and for a specific use-case. In the use-case, the goal is to control a machine processing leaves. Leaves go into this machine and several algorithmically controlled operations are applied to the leaves. These algorithms dynamically depend on the properties of the leaves. In this situation, only a small amount of digitized leaves is available. This small set is not enough to test the algorithms robustly. Generative modelling is used to generalize this small set to a set with more variation. Having a more varied input for the algorithms and observing the results, leads to more robust testing, analysis and optimization of the algorithms.

In this report, one can find the background for two generative modelling methods that are suitable for modelling distributions of images. The two methods are Generative Adversarial Networks (GANs) and Variational AutoEncoders (VAEs) Also, the ideas on applying generative modelling for digital twins are elaborated. The specifics of training the models and the results are enumerated in the second part of the report. Finally, a conclusion is drawn on general advice for using generative modelling techniques for adding variation to digital twins.

2. Generative Modelling Theory

2.1 Generative modelling vs discriminative modelling

Generative modelling is the counterpart to discriminative modelling. Machine learning models can be divided into these two approaches, generative modeling and discriminative modelling [3]. Generative modelling is concerned with how data is generated and models the distribution of the data. Discriminative modelling is focused on relations between pairs of data, via either classification or regression.

Generative modelling is used to model the distribution of the training data. The model learns the probability distribution of the data and can generate new samples that have similar statistical properties. This can be used for inference or sampling new data points. A generative model is a joint probability distribution:

$p(X, Y)$ or $p(X)$ when there are no labels

In contrast, discriminative modelling is used to predict the output variable given the input variables. For example, in image classification, the goal is to assign a label to an image based on its features. The model learns to identify patterns in the data that correspond to different classes, allowing it to make accurate predictions. A discriminative model captures a conditional probability:

$$p(Y|X)$$

2.2 Generative modelling

Overall, generative modelling is a powerful tool for creating new data and exploring the underlying probability distribution of the data. While discriminative modelling is focused on classification and prediction, generative modelling can be used for tasks such as data augmentation, image and text generation, and anomaly detection.

There are some well-known applications of generative modelling. One can generate human-like faces of humans that do not exist even taking into account-desired characteristics [4]. These models have learned the general characteristics of human faces and can generate more. Music and text can also be generated. Currently, a lot of hype is around chatGPT, which also is a generative model

One popular technique for generative modelling is the Generative Adversarial Network (GAN). GANs consist of two neural networks, a generator and a discriminator. The generator learns to create new samples that are similar to the training data, while the discriminator learns to distinguish between the generated samples and data samples. The two networks are trained simulataniously in an iterative process. During the process, the generator tries to create more realistic samples to fool the discriminator network. The discriminator tries to correctly classify the samples as from the data and generated.

Another technique for generative modelling is Variational AutoEncoders (VAEs). VAEs are a type of neural network that learns to encode and decode data in a low-dimensional space. The network can be used to generate new samples by sampling from the learned distribution in the low-dimensional

space. Both the GAN and the VAE use neural networks and are therefore called ‘deep’. The models are part of the class of Deep Generative Models (DGMs).

In Section 2.32.3 and Section 2.4 both classes of generative models are discussed in more detail. Later on, in Chapter 4the application of these models to our use-case will be shown. There, the caveats in trainings these models will become apparent.

2.3 Generative Adverserial Networks

Generative Adversarial Networks, or GANs, are a class of deep generative models that have gained significant attention in the machine learning community. Ian Goodfellow and his colleagues introduced GANs in 2014, and they have been used to generate realistic images, videos, and music, among other things [5].

2.3.1 Generative Averserial Networks, overview

GANs consist of two neural networks, which are trained simultaneously: the generator network and the discriminator network. The role of the generator is to produce synthetic samples that are practically indistinguishable from the samples present in a given training dataset. This network starts with a random noise vector, e.g. a random sample from a Gaussian distribution, as its initial input. It then transforms this noise vector through a series of neural layers into a synthetic sample. The primary objective of the generator is to learn the underlying patterns and structures of the samples in the training dataset, such that it can produce new generated samples that are indistinguishable from data samples. On the other hand, the role of the discriminator is to evaluate samples and predict whether they originate from the training dataset (data samples) or are produced by the generator (synthetic samples). In summary, by training both neural networks simultaneously, in each iteration the generator becomes better in producing samples that deceive the discriminator and the discriminator becomes better in distinguishing between datal and generated samples produced by the generator.

This process is often compared to a police officer and a counterfeiter in an arms race. The counterfeiter can be compared to the generator. The counterfeiter tries to create fake money in a way that is in distinguishable for the police officer. The police officer tries to detect the fake money. Both learn from their mistakes. The analogy differs from the GAN training that in a GAN training afterwards all information is shared for optimal learning. The analogy has even been implemented into real imitation-money-detection systems [6]. Technically this is a mini-max game.

The generator and the discriminator can be any kind of neural networks. The generator should map from a random vector to the desired data structure. The discriminator should map from this data structure to a probability. A common application is to generate images, meaning that the samples are now images. Than the generator usually consists of transposed convolutional layers and the discriminator consists of convolutional layers [7]. These models are often referred to as deep convolutional GANs (DCGANs). In the use-case,see paragraph 3.2, we are also dealing with images and therefore will make use of the DCGANs.

2.3.2 Generative Adverserial Networks, training

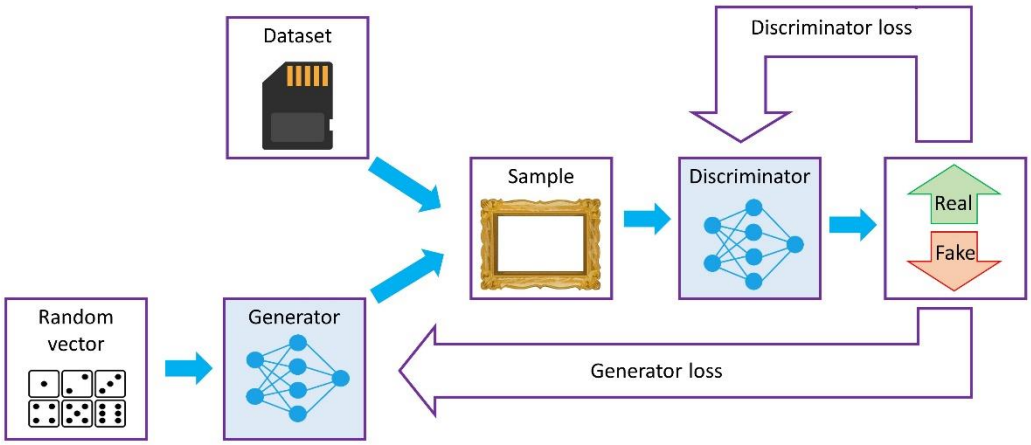

Figure 1: overview of training a GAN. There are two neural networks, the generator and the discriminator. The training starts with drawing random vectors and letting the generator create samples from this. These samples are mixed with samples from the dataset and fed to the discriminator. Based on the evaluations from the discriminator, the discriminator loss and generator loss are determined and fed back to the corresponding models.

Neural networks are usually trained via a version of stochastic gradient descent, where the gradient is determined from a loss function and related to all model weights via backpropagation. Each training step this gradient is determined and slightly followed (determined by the learning rate). It differs from gradient descent in that in each training step the gradient is determined based on a random batch of samples. In training a GAN we do the following each iteration (epoch). First, random vectors are generated and the current version of the generator is used to create new fake samples. Next, these samples and samples from the dataset are fed to the discriminator, recording the output probabilities. Finally, the weights of both the generator and discriminator are updated via backpropagation using the corresponding loss functions. This process can also be done in batch form, where we match the amount of fake samples to the amount of real samples each batch.

The loss function for the generator is the binary cross entropy loss on correctly finding the real images. This is a standard classification task. This can be compared to classifying images of either cats or dogs. Instead, the labels are not cats and dogs, but real or generated. For the generator we only look at the output of our generated samples. For these samples, a loss is assigned for the amount that these images are classified as fake.

2.3.3 Generative Adverserial Network, mathematical formulation

The game between the discriminator and generator can be formulated as a min-max game [5]:

$$\min_G \max_D V(D, G)$$

Where G is the generator, D is the discriminator and $V(D, G)$ is the value of the game. Minizing over G and maximizing over D means that the minimizing and maximizing over the parameters withint the

neural network structure. The value of the game is the ability of discriminator to differentiate between the data and generated samples. This can be represented by an adapted binary cross entropy.

$$V(D,G) = E_{x\sim p_{data(x)}}[\log D(x)] + \ E_{z\sim p_{z(z)}}[1 - \log D(G(z))]$$

$E_{x\sim p_{data(x)}}$ represents the expectation over the data, which can be approximated by summing over data samples. $E_{z\sim p_{z(z)}}$ represents an expectation over some random distribution. $D(x)$ is the probability the discriminator neural network assigns to the data sample. $D(G(z))$ represents the probability the discirminator network assigns to the generated sample produced by the generator based on a random sample.

The result is an arms race between the generator and the discriminator. The generator getting better at producing generated samples and the discriminator getting better at recognizing these generated samples. The goals is to find a Nash equilibrium where the generator produces samples that are indistinguishable from the data samokes. One can input random vectors into the generator to produce new generated samples.

2.4 Variational AutoEncoder

A Variational AutoEncoder (VAE) is another generative model [8]. First, AutoEncoders, the basis of VAE's, will be discussed. Next, the adaptions for making it a generative model are outlined.

2.4.1 AutoEncoder

An AutoEncoder is a type of neural network that is commonly used for unsupervised learning and dimensionality reduction tasks. It is designed to learn a compressed representation, or encoding, of the input data and then reconstruct the original data from this encoding.

The structure of an AutoEncoder consists of two main components: an encoder and a decoder.

1. Encoder: The encoder takes the input data and maps it to a lower-dimensional representation. It consists of one or more layers that progressively reduce the dimensions of the input. The final layer of the encoder is usually a bottleneck layer that represents the encoded representation. The encoder can be compared to the discriminator network of the GAN as it takes in a sample and maps it to a much smaller representation. In the discriminator, this is only one value. These networks often consist of similar layers.
2. Decoder: The decoder takes the encoded representation produced by the encoder and reconstructs the original input data. It mirrors the structure of the encoder but in reverse, gradually increasing the dimensions of the representation until it matches the dimensionality of the original input. The decoder is often a similar network as the generator network of a GAN. Both map a small-dimensional representation to a full sample.

The goal of an AutoEncoder is to learn a compressed representation of the input data in the bottleneck layer, capturing the most important features of the data. By learning such a representation, the AutoEncoder can be used for tasks like data compression, denoising, anomaly detection, and dimensionality reduction.

During the training process, the AutoEncoder aims to minimize the reconstruction error, which is the difference between the original input data and the reconstructed output. The loss function used for training is typically a measure of dissimilarity between the input and output, such as mean squared error (MSE).

2.4.2 Adding the Variational to the AutoEncoder

Given a learned decoder, one could insert a random vector. One would like the result to be a valid sample. However, this is only the case when this random vector is close to encoded samples from the dataset. As the decoder has only learned to decode these. The encoded representations should be a good representation from some space, to be able to reliably input random vectors in to the decoder. In a Variational AutoEncoder, this is part of the objective.

Usually, the multivariate normal distribution of some dimension is chosen as a distribution. Next to the reconstruction loss, another term is added to the loss function. This term is the KL-divergence of the encoded samples with the chosen distribution. When the KL-divergence is low, the set of encoded samples are a good cover of the chosen distribution. The decoder will therefore correctly deal with a new random sample from the distribution. A Variational AutoEncoder is thus an AutoEncoder, whose latent space is organized such that is suitable for generating new samples.

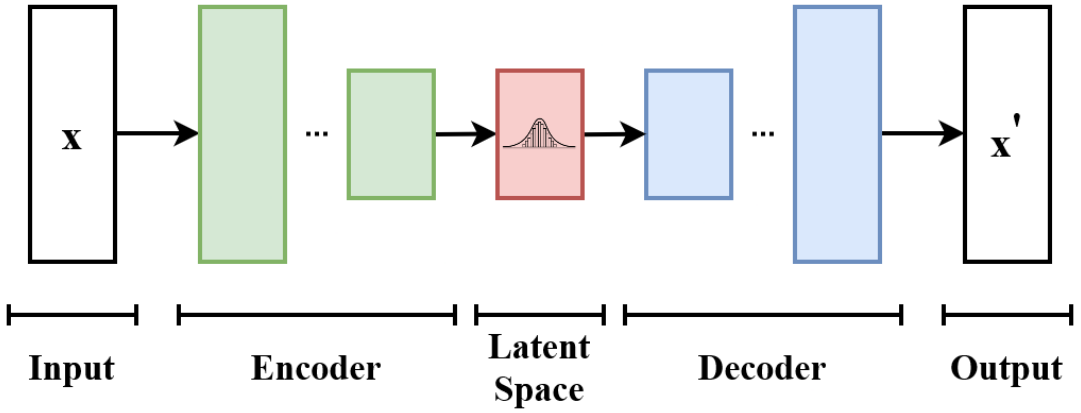

Figure 2: Overview of a Variational AutoEncoder with the different layers [10]

The loss function is a combination of the reconstruction loss and the KL-divergence. This function is not easily differentiable. This is easier after the reparemetrization trick [9]. The output of the encoder is used to parametrize independent normal distributions. A sample from these distributions forms the realization of the latent space. As the loss function consists of two parts, we can use a parameter to balance these [10].

The loss function can then be written as follows:

$$\mathcal{L}(\theta, \phi; \boldsymbol{x}, \boldsymbol{z}, \beta) = \mathbb{E}_{\{q\ \phi(x,z)\}}[log\ p_\theta\ (\boldsymbol{x}|z)] - \beta\ D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})) \quad (1)$$

With $\boldsymbol{x}$ an observation, $\boldsymbol{z}$ a latent factor, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ the distribution of the latent factor (reparametrization trick), $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is the reconstruction of the model with parameters θ. In this equation, we recognize the reconstruction loss in the first part, the KL-divergence in the second part and the balance-parameter β.

2.4.3 Reparametrization trick

Usually the latent space is supposed to follow a multidimensional normal distribution with independent components, with unknown mean and variance. To make the mean and variance of this distribution learnable a gradient is necessary. One cannot take the gradient of a random variable. Using the lineairity of the normal distribution, we can write

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma}\,\boldsymbol{\varepsilon}$$

Where $\mathbf{z}$ is the sample from the latent space, $\boldsymbol{\mu}$ is the (learnable) vector of means, $\boldsymbol{\sigma}$ is the (learnable) vector of variances) and $\boldsymbol{\varepsilon}$ is a multidimensional sample from standard normal distributions. The trick is that the random part is separated from the learnable part. There is a gradient trough the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, which does not depend on the random part. The random part as a constant distribution. As a result of this, the encoder can be trained with backpropogation to learn the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ for the correct parametrization of the latent space.

In this section, the reparametrization trick is explained for the multivariate normal distribution of the normal space, but many other (parametrizable) distributions are possible. A distribution can be used, when it can be parametrized such that there exists a gradient from a sample to the parameters.

3. Generative Modelling and Digital Twins

3.1 Digital twins

A digital twin is a virtual copy of a physical system or process that serves as a tool for simulating, monitoring, and enhancing its performance [11]. Digital twins provide engineers and operators with valuable insights into system performance, eliminating the need for direct physical interaction. By leveraging sensors, data analytics, and machine learning algorithms, digital twins offer real-time information on system behavior and the ability to predict future performance. This information is utilized to optimize efficiency, reduce maintenance costs, and improve overall system functionality. ASIMOV specifically focuses on digital twins of advanced Cyber-Physical Systems (CPSs); employing them to optimize these systems and training AI models for their operation.

In the realm of creating digital twins, we explore the utilization of generative models as machine learning algorithms, as discussed earlier. These models offer two primary avenues for contributing to the development of digital twins. Firstly, when data is limited, generative models can generate more variations to enrich the available information. Secondly, these models can generate additional data to support the construction of a digital twin.

3.1.1 Creating variations

In a digital twin, there is some input into the system, with which the system interacts. These can for instance be sensors on a robot. A good digital twin is robust for all kinds of input. One is not only interested in scenerio's already played out, but also other scenaro's that have not happened yet, but might happenin the future. However, a digital twin is often created when physical experimentation is hard or expensive. Leading to little input data with little variations. To get good robust digital twin, there is a need to create artificially variations.

Generative models can be utilized for this. Based on limited input data, the data generating process can be learned. Creating a model for the input. When the digital twin is used to operate on new scenario's a sample from the generative model can be taken as input.

Below, in Figure 1, is the general overview of the contents of WP 2.2 of the ASIMOV project, *Methods and Tools for Training AI with Digital Twin* [12]. In this view of the AI training of digital twins, robustness is important and therefore variations need to be introduced. In WP 2.2 DGMs are also considered.

An example would be a robot vehicle operating in an environment. The robot has some sensors that perceive the environment. Each of these sensors should be simulated in a digital twin. One could use sample data already perceived by a physical robot, however there might not be enough variation in this dataset. A generative model can be applied to this dataset to learn the data generating process. In the digital twin, samples from this generative model can be used and there are new variations in this.

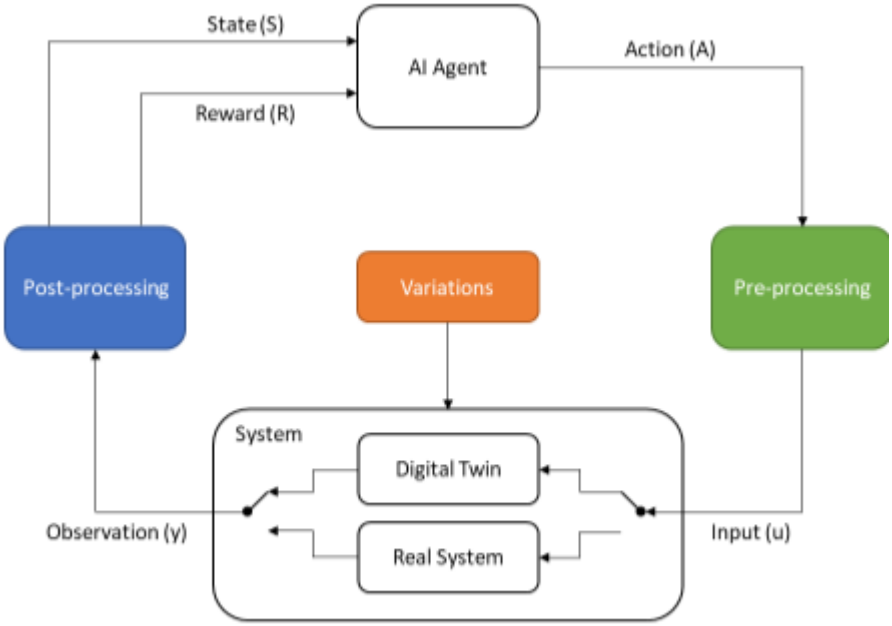

Figure 1: overview of ASIMOV WP 2.2 [12]. The graphic represents the interaction of an AI agent and a system, which can be a digital twin or a real system.

3.1.2 Generative model as building block.

A digital twin exist of a model or a combination of models that are a simplification of the physical world. Many models are interesting for different use-cases. In some cases, a generative model is a good way to create a model of a physical part of the system. It is a data-based modelling approach. Given input parameters a (part of) the physical world can be generated that is input to the decision maker. Generative models can become a building block for the digital twin [13].

This is mentioned in WP 2.3 of the ASIMOV project, *Architecture of optimized digital twins for AI-based training*. It is mainly mentioned as a method for modelling complex, unstructured data like text, images and audio. DGMs are very applicable.

3.2 Use-case

In this section, the use-case is presented that exemplifies the role of generative modeling in creating variations within a digital twin framework. The specific scenario involves a physical system consisting of a machine designed for processing harvested leaves. To assess the performance of the algorithms driving this machine, a digital twin is employed. At the beginning of the machine's operation, the leaves are scanned, generating digital copies in the form of images. Similarly, the digital twin starts with a digital representation of a leaf. However, due to the limited availability of scanned leaves, a more diverse set of leaves with variations is required to adequately test the robustness of the algorithms.

In this context, our aim is to create a generative model for leaves. We possess a small dataset consisting of 80 scanned leaf images, which are converted into geometric descriptions. These geometric descriptions are outlines represented by polygons, nerves represented by linear splines, and holes represented by polygons. The algorithms primarily operate on these geometric descriptions.

However, modeling these geometric properties directly using neural networks proves challenging. Hence, a decision is made to convert these geometric descriptions back into image format, allowing for the utilization of convolutional neural networks. Instead of using three color channels, two are used. One channel contains a binary variable whether the pixel is part of the leave or not and the other whether the pixel is part of a vein. Consequently, the generative model generates images of leaves, which are then subjected to a post-processing step to convert them into their respective geometric representations.

It is important to note that the generative modeling focuses on half-leaves, as the main nerve of the leaf should remain unaltered by both the algorithms and the physical machines. Therefore, operating on one side of the leaf at a time proves sufficient. In the dataset, all leaves are split and the left sides of leaves are mirrored to be a right side. The key leaf properties of interest include size, shape, the presence and size of holes, as well as the quantity and length of nerves.

In this work, two generative models are applied to this problem, the GAN and VAE as introduced in chapter 2. In the next sections, an overview of the resulting training process is given as well as the obtained results for each of these models.

4. Results on use-case

In this chapter, the results on the use-case are presented. For each of the two models, first the process of training the model on the use-case is shared. Next, the resulting samples from the final model are shared. The results are compared in chapter five.

4.1 Training a GAN

First, we apply a Generative Adversarial Network (GAN), which was introduced in section 2.3. In GAN training there are two neural networks, the generator and the discriminator. The learning process, the discriminator and the generator are all implemented, following the guidelines from the paper by Radford et al. [7]. In this paper, many choices in deep convolutional generative adversarial networks (DCGAN) are researched and evaluated. The discriminator is a binary classification neural network. For the classification of images, the convolutional neural network is a common approach, which was popularized after the success of AlexNet [14]. The discriminator for this use-case consists of eight layers of 2D-convolutions. In between the layers, the activation function is the leaky relu and there is a batchnorm layer [15], [16]. This network has 11.227.584 parameters.

The generator is in some sense the opposite of the discriminator. From a low-dimensional space, it maps to an image, a high-dimensional space. The generator consists of eight layers of tranpose 2D-convolutions. Instead of reducing the dimension and increasing the channels for each pixel, it reduces the channels and increases the dimension. In between there is the ReLU activation function and batchnorm layers for stabilization. At the end, the final activation function is the tangent. This model has 46.375.872 parameters and is still according to the guidelines from [7].

As was already discussed in section 2.3, it is hard to choose the right parameters in training a GAN. There is a fine balance of the complexity of the data, the complexity of the neural networks and the learning rates. The training process might derail in many ways. The following derailings were observed.

The situation can emerge that the discriminator can confidently classify all generated images correctly. Compared to the generator it has learned more. Suppose the generator would slightly adapt the parameters the generated images would still easily be classified by the discriminator. This means that there is no gradient for the loss function of the generator with respect to its parameters. This is called the vanishing gradient problem [17]. A problem that can occur in training a deep learning model with back propagation, and is also known to happen in GAN training [18]. The result is that it is hard for the generator to improve. This problem occurs when the learning rate of the generator is too low or of the discriminator too high and because of this the balance in their learning is broken. During the training run shown in Figure 2 and Figure 3, this phenomenon can be observed. Between the epochs 90 and 290 the discriminator predicts perfectly with absolute confidence. The average probability of images being from the dataset is close to one for real images and close to zero for generated images. When looking at the loss function for the generator during this period, we see that at first it does not improve at all. Once small improvements are made, the loss increases faster and faster, because better gradients are obtained each time. Learning would be more efficient if both models would learn simultaneously.

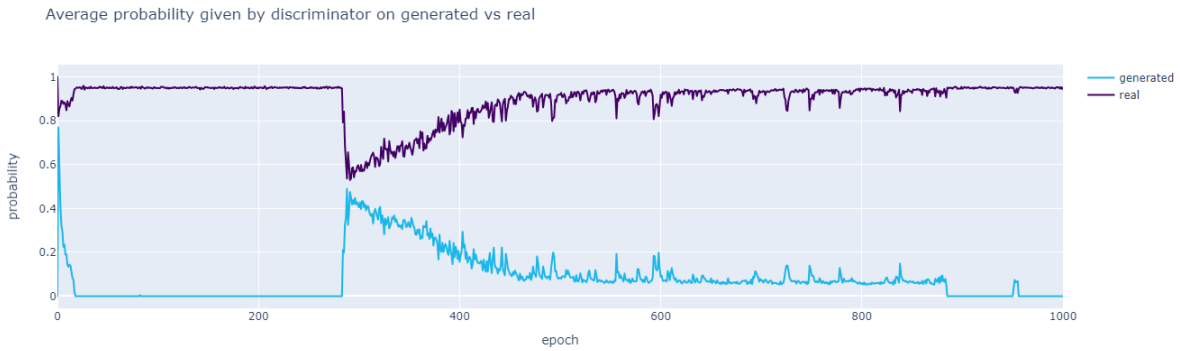

Figure 2: the average probability that an image is from the data as assigned by the discriminator to the images of two sources (generated and from data) over the epochs. The data in this graph is from the same training run as Figure 3.

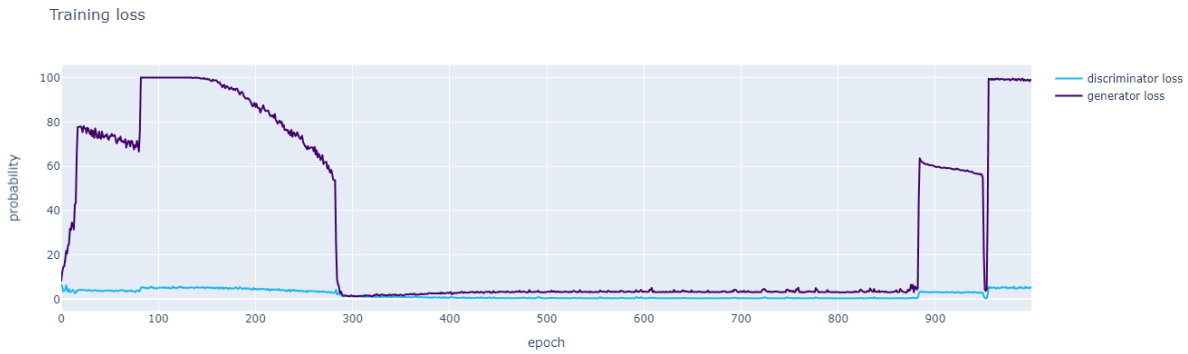

Figure 3: the losses of the discriminator and generator as developed over the epochs. The data in this graph is from the same training run as Figure 2.

Another problem in the training procedure is mode collapse [19]. The generator discovers patterns in the real data and tries to generalize these. However, a generator can also overfit. If it finds a pattern that successfully tricks the discriminator, because it occurs in one or a small number of data points. It will generate only images with this feature irrespective of the random input vector, because it seemed essential. The generator does not learn the full diversity of the distribution. This can be caused by a too high learning rate for the generator or a network that is not complex enough.

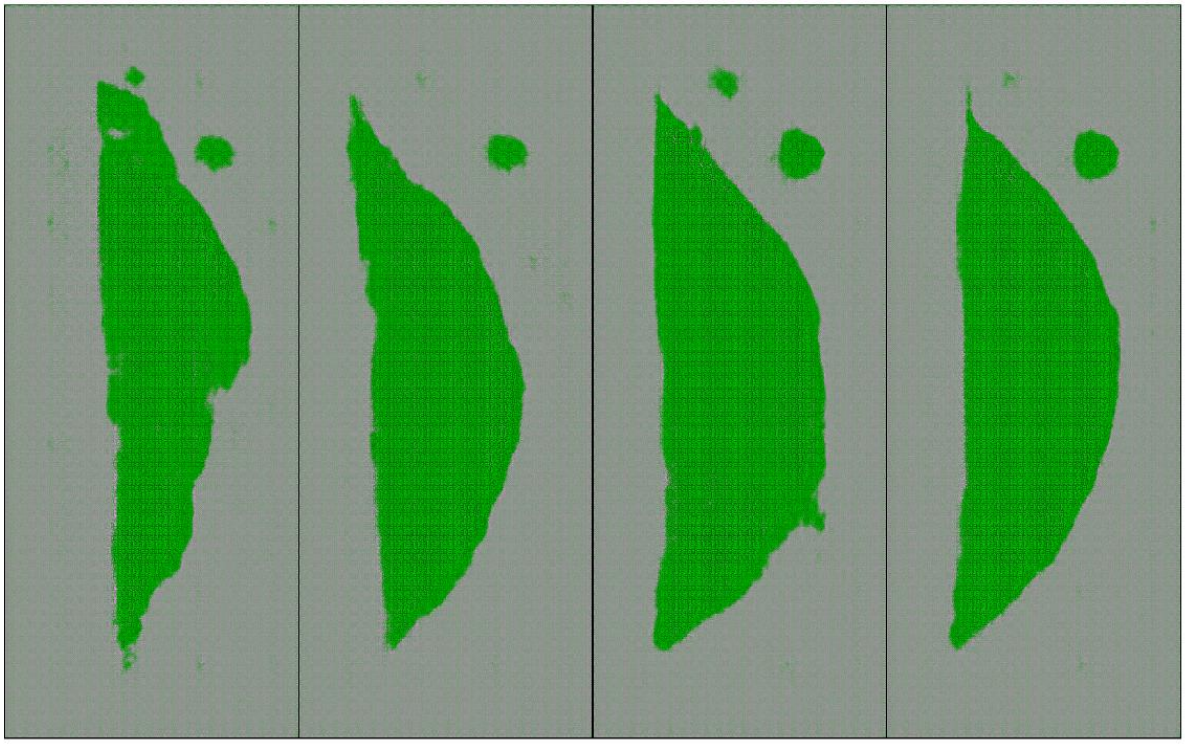

Figure 4: four images of leaves generated by a GAN, in which mode collapse is visible.

In Figure 4 one sees four generated half-leaves. These are all generated using entirely different random vectors. The images all look like leaves, but also all have some separate round piece at the top right. This might have happened in one leave, but is definitely not a general feature of leaves. The generator has overfitted and at this point generates leaves with these images irrespective of the random input

Finally, when the generator is sufficiently trained, the discriminator cannot distinguishes between data and generated samples. This gives a random signal for the generator to learn. The generator could become worse because of this phenomenon. This can also be seen in Figure 2 and Figure 3. At about 300 epochs, the average probabilities for the fake and real images are both about 0.5. This means that the discriminator cannot tell the difference at that point. The generator had no signal to learn and the discriminator slowly gets the upper hand again. When the loss of one is increasing, it does not necessarily mean it gets worse, but the counterpart could also become better.

The problems are mainly tackled by finding the right balance between both learning rates, balance of generator and discriminator learning and the complexity of the models duration of training. This is an art rather than a science. There is a lot of randomness in GAN training. In usual neural network training, the randomness is induced via the initialization of the parameters and the composition of the batches. However, during GAN training random input vectors for the generator are sampled each time, introducing much more variability. The right balance between hyperparameters can only be found through experimenting. In practice, there is only a small set that gives satisfying results. As an example, when tuning the learning rate for the generator, too small and you have vanishing gradients, too large and you risk mode collapse. In general, the large the learning rate the faster, but less stable the training procedure.

The eventually used hyperparameters are:

- Batch size 8
- Lr discriminator 0.0001
- Lr generator 0.0003

These hyperparameters are not necessarily optimal, but are found to give satisfying results by visual inspection. In Figure 6 and Figure 9, the statistics for the final training run are shown. One can clearly see the variability in the losses and probabilities. In addition, the generator collapses at about 550 epochs. Still this run gives satisfying results.

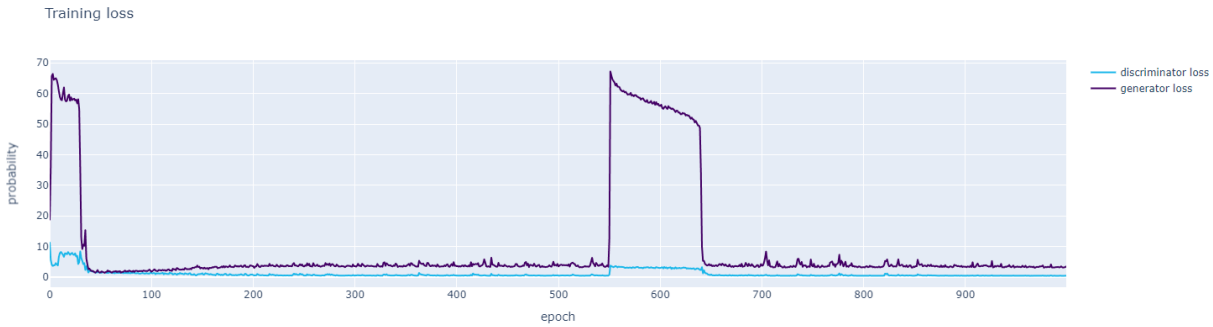

Figure 5: the average probability that an image is from the data as assigned by the discriminator to the images of two sources (generated and from data) over the epochs. This graph represents data from the final run and is connected to Figure 6.

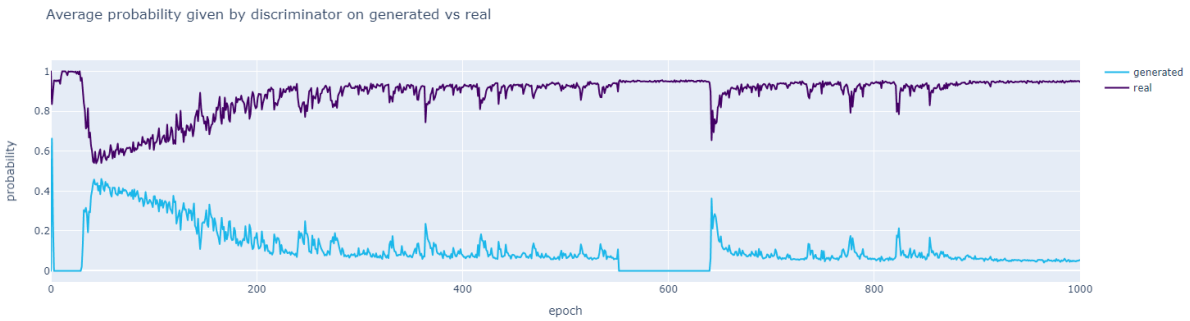

Figure 6: the losses of the discriminator and generator as developed over the epochs. The data in this graph is from the same training run as Figure 5.

In conclusion, many problems were encountered during the training of a GAN. It is known from theory that these problems can occur and it is part of search for the right hyperparameters. Finding the hyperparameters for stable GAN training is hard. However, once the right hyperparameters are found training is fast and results are often good.

4.2 Results GAN

In this section, the results after training a GAN are shown. There are three types of results, the raw generated images, the postprocessed images and distribution of the features of the generated images. It is hard to evaluate the performance of a generative model. On the one hand the generated data should resemble the dataset on the other it should have variations. There is not a single metric to capture these both. The easiest way to analyze the performance of a generative model is by inspecting the generated data. Next to that, we will be comparing the distribution of features with the original distribution of features. The idea is that when two distributions are equal the distributions of applying a function each of these distributions should also be equal. For this, we first convert the image to geometric shapes and then several features are extracted like, the size, the amount of veins and the sizes of the holes. All of these features are a function of the image.

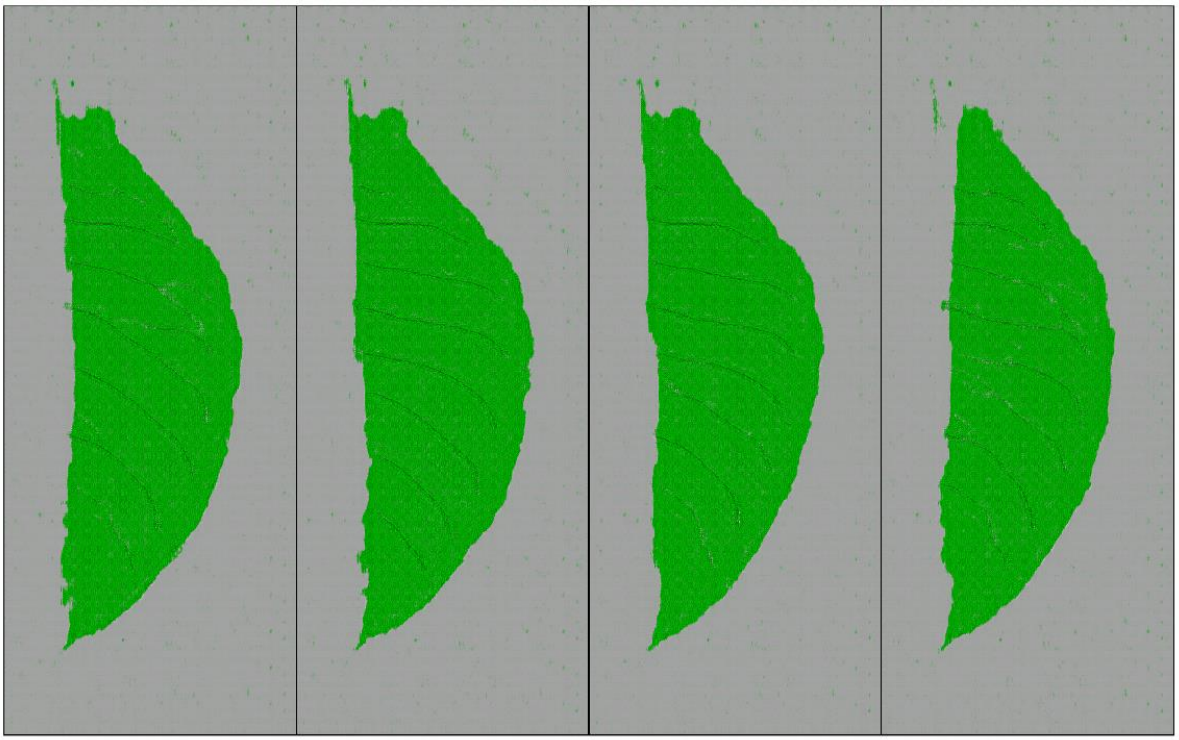

Figure 7: four images of leaves generated by a GAN. More of these samples can be found in the appendix, Figure 14Figure 14: 32 samples taking from the GAN..

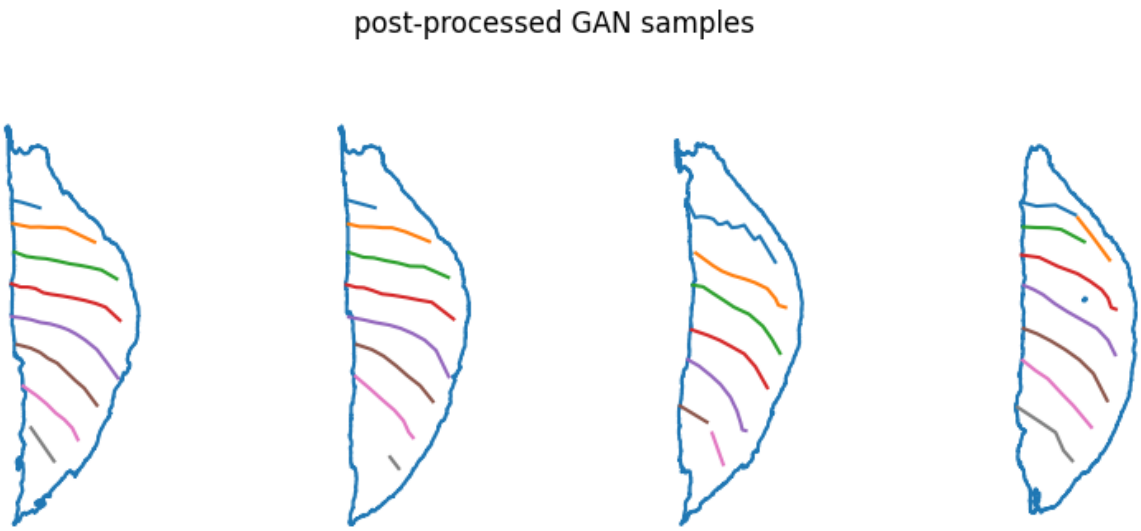

Figure 8: post processed version of the images in Figure 7. The blue lines represent the outline and the holes and colored lines represent veins.

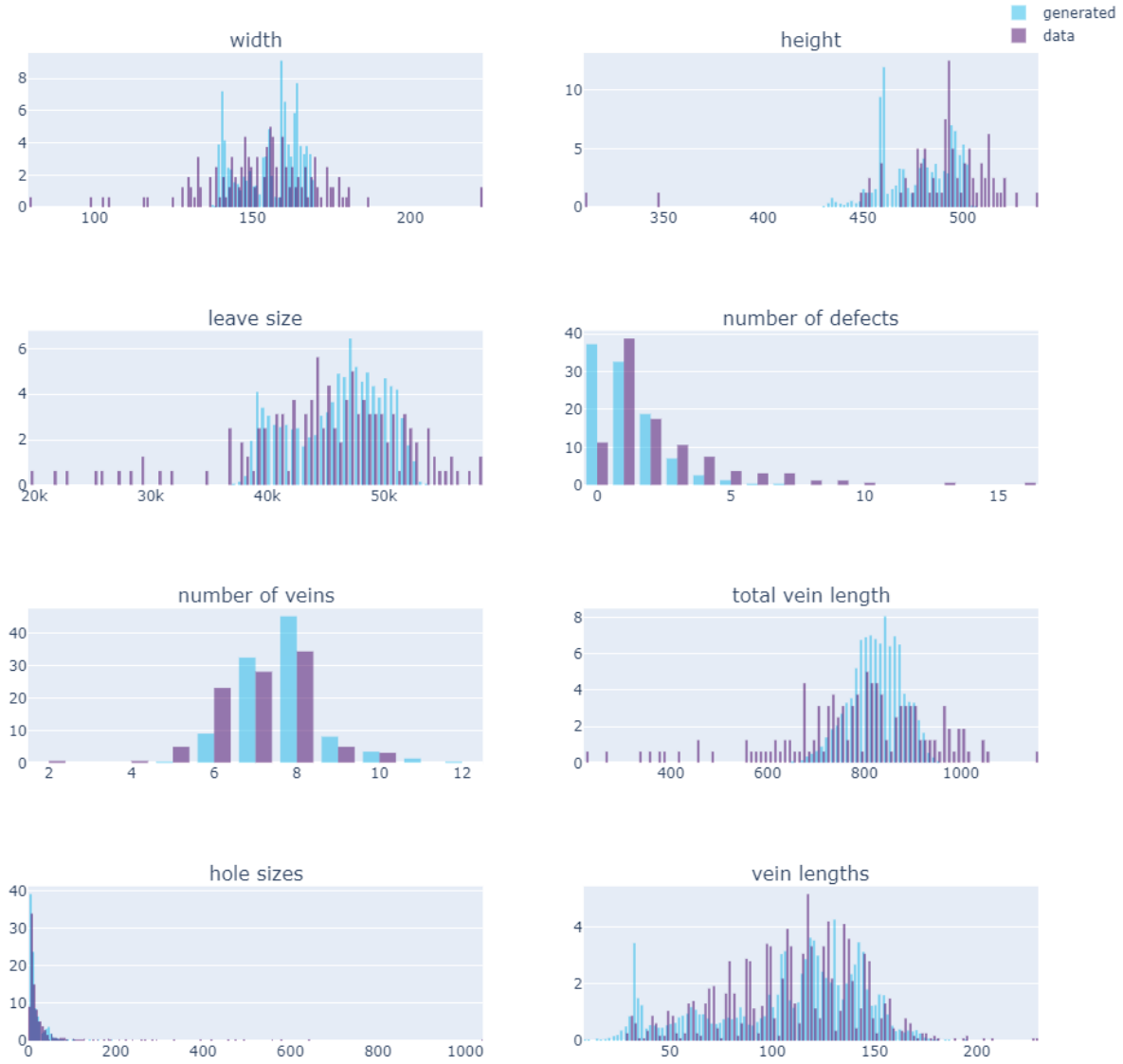

Figure 9: distributions of eight properties as taken from geometric distributions. There are 160 half-leaves taken from the dataset and there are 2000 half leaves generated by a GAN. The y-axis represents a percentage.

The generated images displayed in Figure 7 are good examples of leaves. There are more samples in the appendix, Figure 14. It seems that some features are more prevalent than expected, but also just a sample of four, For instance, there seem to be relatively small amount of holes in these four images. When looking at the other samples in the appendix, this seems to be a general feature. The images are well captured during the post-processing step. The veins are not always connected, but that is not a big issue for the use-case. Finally, the distributions of the eight properties are similar on average. For some there seems to be a bit less variation in the generated samples, especially for width, total size and vein length.

4.3 Training a VAE

Next to the GAN, also a Variational AutoEncoder (VAE) was trained, as mentioned in section 2.4 A Variational AutoEncoder consists of only one neural network, which has two parts. The decoder and encoder. The decoder is similar to the generator and the encoder is similar to the discriminator. That is why their architectures for these problems are quite similar. The encoder, similarly as the discriminator consists of eight layers of 2D convolutions with ReLU activation function [14]. In between the encoder and the decoder, there is the reparametrization trick. Finally, we end with the decoder. It consists of eight 2D transposed convolutions with ReLU activation. The network is thus very similar to a discriminator and generator combined together.

In the loss function of the VAE there is the parameter beta, see equation on page 11. It is unclear what the best beta is. Following the approach of [20], we use a cyclical beta schema. Beta goes from 10 to 100 in 1000 epochs and returns to zero. The models was trained for 5000 epochs with a learning rate of 0.001 and a batch size of eight.

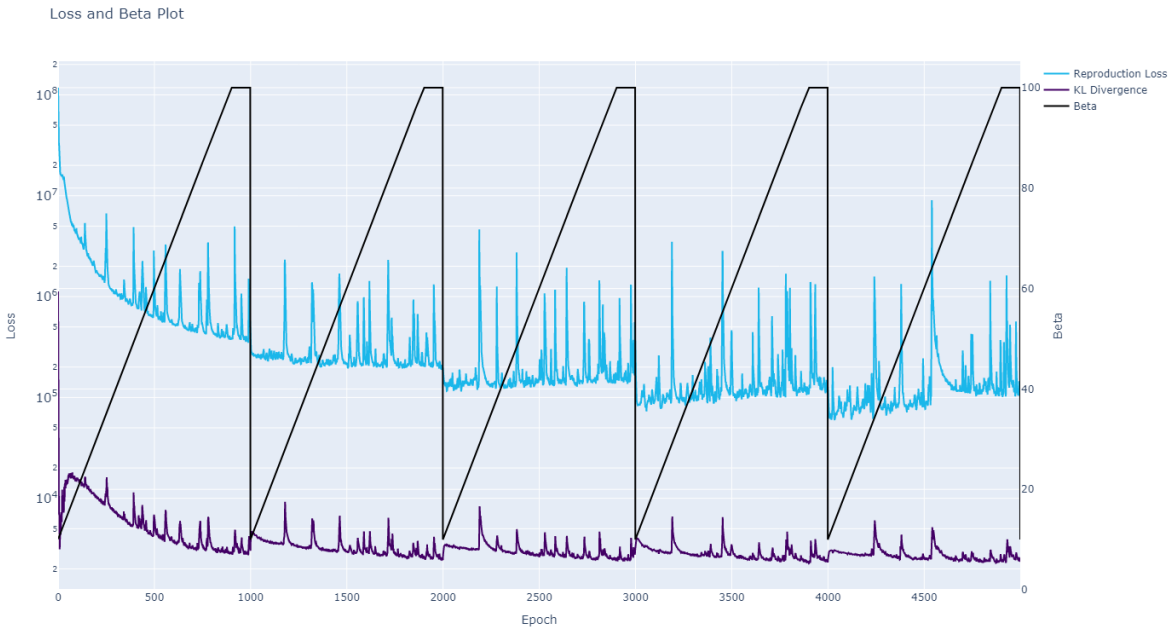

Figure 10: the statistics of the final VAE training run. The blue and purple lines correspond loss terms, respectively the reproduction loss, and the KL divergence. The values correspond to the left (log) axis. The black line represents the value of beta in each epoch and is connected to the right (linear) axis.

In Figure 10, we see that on average the Reproduction loss term is about 100 times as large, that is why we set the beta between 10-100. At the start, the KL divergence term even increases, because the total loss can be decreased. The learning seem to converge except for the influence of beta. There are many spikes that break the trend. In the reparametrization, trick random latent vectors are sampled. This causes randomness in the training and therefore the spikes. In the next section, we see the results of training a VAE.

4.4 Results VAE

In this section, we make the same visualizations as for the GAN in section 4.2. The results are compared in section 5.2. These are four samples generated by the VAE generative model. These look al like leaves. They are not too similar. The veins start at the edge, but there are holes in the veins. There also seems to be a lack of holes in the leaves. However, this is just a random sample of four.

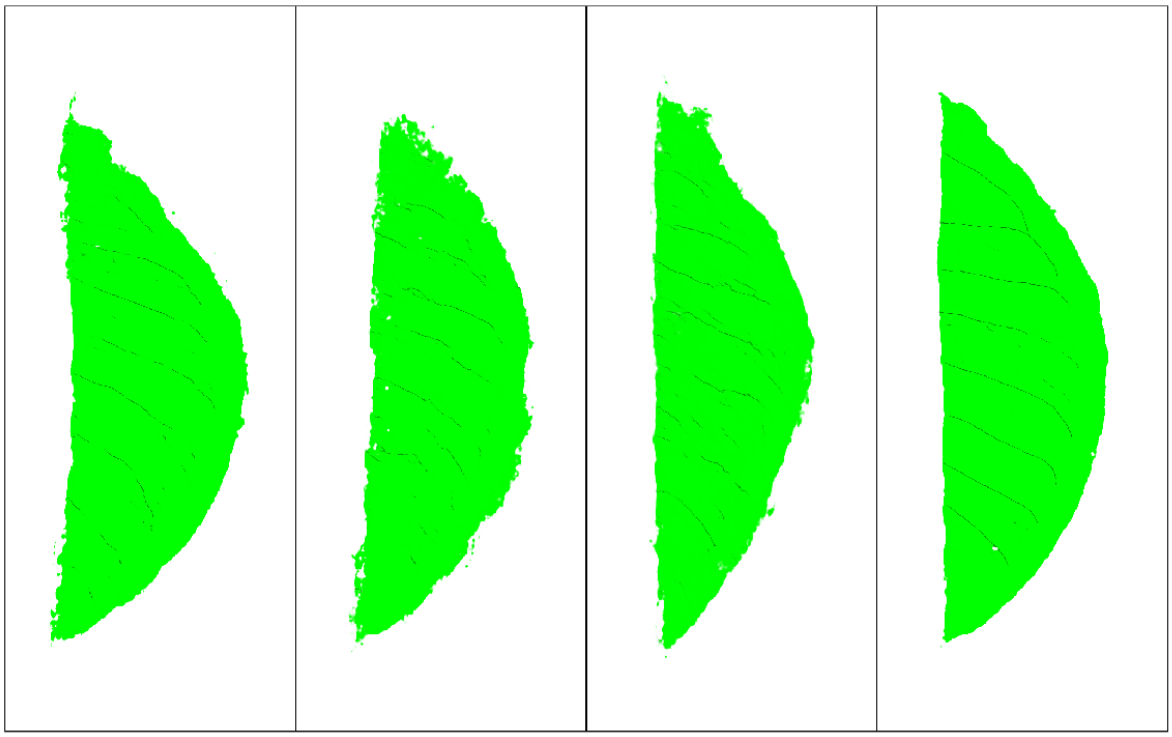

Figure 11: four images of leaves generated by a VAE. More of these samples can be found in the appendix, Figure 1Figure 14: 32 samples taking from the GAN..

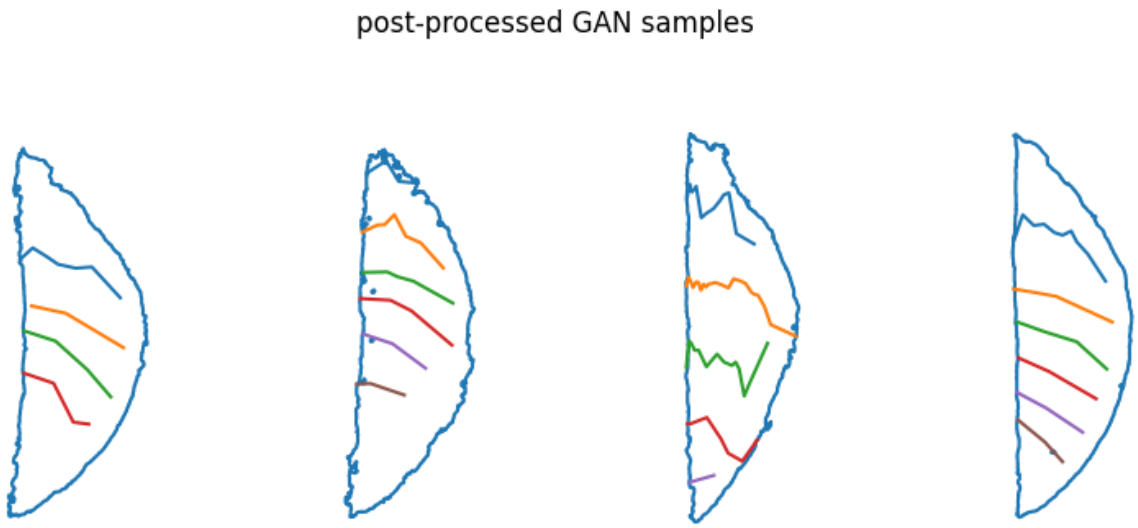

Figure 12: post processed version of the images in Figure 11. The blue lines represent the outline and the holes and colored lines represent veins.

Figure 13: distributions of eight properties as taken from geometric distributions. There are 160 half-leaves taken from the dataset and there are 2000 half leaves generated by a VAE. The y-axis represents a percentage.

In Figure 11: Figure 11 four samples generated by a trained VAE are shown; there are more samples in the appendix, Figure 15. A thing that stands out is that the background is sharper. This is probably because the VAE explicitly contains a reconstruction error, which makes the images look as much as possible as the real data. There seems to be some nice variation in the shapes of the leaves. The have veins are in the right places, but they seems interrupted in many places. Figure 12 shows these images after post-processing. The veins are not processed optimality as can be expected from the raw images. Some veins are missing after post-processing, because they are not apparent enough. Finally, in Figure 13 the distributions of the properties are shown. Indeed one can see that most properties have very similar distributions and the same variation. However, the number of veins and because of it the total vein length is lower on average, because veins are often not evident enough to be taken into account in the post-processing.

We have seen the training of the GAN and VAE in this chapter. Furthermore, the resulting leaves were shown. In the next chapter, we compare these two models.

5. Comparison GAN vs VAE

During this report, two models are mentioned continuously. The Generative Adversarial Network (GAN) and the Variational AutoEncoder (VAE). In this section, we compare these two models on three bases. First, we compare the core ideas and theory, as described in chapter 2. Next, we compare the models based on their results on the use-case as follows from chapter 4. Finally, they are compared based on their potential use in a digital twin, following their comparison on theory and in the use-case.

5.1 Theory

In a GAN and VAE, both use neural networks for generative modelling. The GAN consists of two neural networks and the VAE consists of one neural network. However, the VAE can be viewed as two neural networks, the encoder and decoder, joined together. In this view, for the same problem, both methods will have similar neural networks. The encoder will be like the discriminator and the decoder will be like the generator. GAN and VAE do not necessarily largely differ in neural network architecture, but mainly in training style.

The GAN is trained as a game between the discriminator and generator and alternatively updating them. Both the discriminator and generator are updated using stochastic gradient descent. The VAE is one neural network also updated with stochastic gradient descent. The main difference is in the loss functions. The problem with extreme flexible models like neural networks is that it is easy to overfit. The GAN evades this by never showing the data directly to the generator, but just the classification of the discriminator. The VAE evades overfitting by using the reparametrization trick and by requiring the organization of the latent space. Both these factors show up in the loss functions as listed in chapter 2.

The usage of the models is quite similarly. One draws a random vector from an appropriate distribution. This vector is inputted into either the generator or the decoder and the result is a sample of the desired data structure.

5.2 Use-case

The results obtained on this use-case coincide with what was expected from theory. First, GAN training proved to be extremely unstable. Many trials of hyperparameters were necessary for obtaining stables ones. Many forms of derailing were encountered including mode collapse and vanishing gradient. In contrast to the VAE training, which almost worked out of the box. After the initial guess of the hyperparameters a cycling beta was introduced and the learning rate was updated slightly. These changes are straightforward and the parameters hardly interact, making VAE hyperparameter tuning easier.

However, once the models were trained the GAN seemed to outperform the VAE. It is not trivial to find objective measure for the evaluation of the result of a generative model. It seemed that the VAE was not fully capturing the veins as well as the GAN did (see Figure 12 and Figure 13). This lead to the veins sometimes not being recognized during post-processing. Many leaves had only half the amount of veins as of what was generally in the data. The samples created by the models were inspected in three ways. It is still hard to create objective metrics that really cover the whole problem of evaluation a generative model. Evaluating relies a lot on inspection.

5.3 DT situation

As is been stated before, it is hard to estimate the quality of a generative model. It would be interesting if the quality can be measured with respect to the impact on the DT. This is very use-case specific. For the usage of the generative model some properties of the generated data can be defined that are the main channels through which the generated data interact with the digital twin. For instance in our use-case. The machine should cut out pieces from the leaves containing no veins. For this, it is important to know where the boundary of the leave is including potential holes and where the veins are. However, the shape of the circumference is of less importance. There might be use-cases where that property is more important.

In general, the best way to evaluate a generative model focusing on the impact it will have on a digital twin is to compare the distribution of these key properties. These distributions should be similar to their counterparts based on real data. These distributions should be both similar in location as in variation. This also means that models who create samples that seem worse (for instance by visual inspection in the case of images) might still be better for the digital twin, because they resemble the data more on key properties that are important for the digital twin. Nevertheless, it remains hard to draw objective conclusions on the performance of generative models.

6. Conclusion and Discussion

In this report, generative modelling, including two deep generative models, were introduced. Generative modelling was related to creating digital twins. Next to that, a use-case on processing leaves was mentioned. The two deep generative models were applied to this use-case and the results are in chapter four. In chapter five, the two models are compared based on their theoretical properties, their results on the use-case and their application in digital twins.

The first general conclusion is that it is hard to measure the performance of a generative model. One wants the samples to be varying, but also similar to the original data. There are several metrics on either of those, but their combination is hard to measure. Another evaluation method is inspection. This is possible for most data structures, like images, video and sound. This method is subjective and it is hard to evaluate a large sample. However, often with inspection all the facts can be taken into account. The last way of evaluation is by studying the distribution of properties. The high-dimensional samples are reduced to some lower dimension by studying some of their properties. For the leaves, these properties were for instance, their size, the height of the leaves, and the amount of holes and so on. These distributions should be similar to the distributions of these properties based on data. This also seems the most appropriate form of evaluation for the usage of deep generative model in a digital twin. Generally, a sample from a generative model influences the digital twin through a couple of properties or characteristics, but not all. When thinking about the leave use-case, the placement of the veins and holes is important, but the shape of the edge not. This means that the when the distribution of relevant properties is similar then the generative model is a sufficient replacement for the data. Irrelevant properties can be ignored.

When evaluating the Generative Adversarial Network (GAN) and Variational AutoEncoder (VAE) on the use-case the following results were obtained. By visual inspection, it could be seen that the samples created by the VAE did not capture the veins as well as the samples from the GAN. Analyzing the distribution of the important properties after post-processing, showed that the number of veins was off. Therefore, it seems that the GAN has shown better results on the use-case. It should be noted however that these results depend a lot on the hyperparameters of the models and the training procedure. An effort has been made to optimize these, but optimality is not guaranteed. The process of optimizing hyperparameters is a lot harder for a GAN than for a VAE. The learning process of a GAN is very unstable and good learning relies on a fine balance of the complexity of the data, complexity of the models and the learning rates for both models. It is therefore significantly harder to find a set of hyperparameters that actually makes the GAN learn. For the VAE, most sets of hyperparameters work, but some are somewhat better than others. On the use-case, the result is that the GAN produced better samples, but with more effort in the tuning of the hyperparameters and this is something that is found in general according to the literature.

Generative modelling using deep neural networks can be beneficial in developing a digital twin. When low-dimensional samples are needed one can probably construct some model based on probability distribution. For high-dimensional samples, one should resort to deep generative models. These can model high-dimensional data and return samples that are varying. The VAE is the most easy to train and therefore should be tried first to see whether it obtains satisfying results. In case the results of the VAE are not sufficient, a GAN can be tried. However, the training of a GAN can become quite involved. It is important to decide upfront on the evaluation on the generative model. Through what properties will the generated samples influence the digital twin? When successfully trained deep generative models can be an input source into the digital twin or one of the models making up a digital twin.

Both the GAN and VAE were introduced in 2014 and have since then been developed and tested to become the two mainstream deep generative models [5] [8]. There are many variations of both. Mainly in the domain of controlling the samples. Many variations allow for different amounts of control over the resulting samples. For the use-case, one could build a deep generative model and desire a sample

with a little amount, normal amount or large amount of holes. A relatively new idea is to combine both models in what is called a VAE-GAN [21]. A VAE model replaces the generator and the reconstruction loss term is replaced by the generator loss. Furthermore very deep generative models are rising in the case of large language modelling, most famously the GPT-series. In conclusion, this area is actively researched.

7. Bibliography

[1] "About ASIMOV," 08 08 2023. [Online]. Available: https://www.asimov-project.eu/.

[2] A. Ng and M. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes.," *Advances in neural information processing systems,* 2001.

[3] L. Ruthotto and E. Haber, "An introduction to deep generative modeling," *GAMM-Mitteilungen,* 2021.

[4] E. Negri, L. Fumagalli and M. Macchi, "A review of the roles of digital twin in CPS-based production systems.," *Procedia manufacturing,* pp. 939-948, 2017.

[5] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets.," *Advances in neural information processing systems,* 2016.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adverserial Nets," *Proceedings of the International conference on Neural Information Processing Systems (NIPS),* pp. 2672-2680, 2014.

[7] A. Togeer, S. Jan, A. Alkhodre, M. Naumann, M. Amin and M. S. Siddiqui, "DeepMoney: counterfeit money detection using generative adversarial networks," *PeerJ Computer Science,* 2019.

[8] A. Radford, L. Metz and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks.," *arXiv preprint,* p. 1511.06434, 2015.

[9] D. P. Kingma, D. J. Rezende, S. Mohamed and M. Welling, "Semi-Supervised Learning with Deep Generative Models," *Advances in neural information processing systems,* vol. 27, 2014.

[10] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv,* vol. preprint arXiv:1312.6114, 2013.

[11] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed and A. Lerchner, "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework," in *International Conference on learning representations*, 2016.

[12] M. W. Grieves, in *Virtually intelligent product systems: Digital and physical twins*, pp. 175-200.

[13] "Publications," November 2022. [Online]. Available: https://itea4.org/project/workpackage/document/download/8514/. [Accessed 08 August 2023].

[14] G. Tsialiamanis, D. J. Wagg, N. Dervilis and K. Worden, "On generative models as the basis for digital twins.," *Data-Centric Engineering 2,* p. e11, 2021.

[15] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks.," *Advances in neural information processing systems,* 2012.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift.," *International conference on machine learning,* pp. 448-456, 2015.

[17] B. Xu, N. Wang, T. Chen and M. Li, "Empirical evaluation of rectified activations in convolutional network.," *arXiv,* vol. preprint arXiv:1505.00853, 2015.

[18] S. Basodi, C. Ji, H. Zhang and Y. Pan, "Gradient amplification: An efficient way to train deep neural networks," *Big Data Mining and Analytics 3,* vol. 3, pp. 196-207, 2020.

[19] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE signal processing magazine,* vol. 35, pp. 53-65, 2018.

[20] S. Lala, M. Shady, A. Belyaeva and M. Liu, "Evaluation of mode collapse in generative adversarial networks," *High Performance Extreme Computing,* 2018.

[21] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz and L. Carin, "Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing," *arXiv,* vol. preprint arXiv:1903.10145, 2019.

[22] A. B. L. Boesen, S. K. Sønderby, H. Larochelle and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

8. Appendix

8.1 Generative Adversarial Network

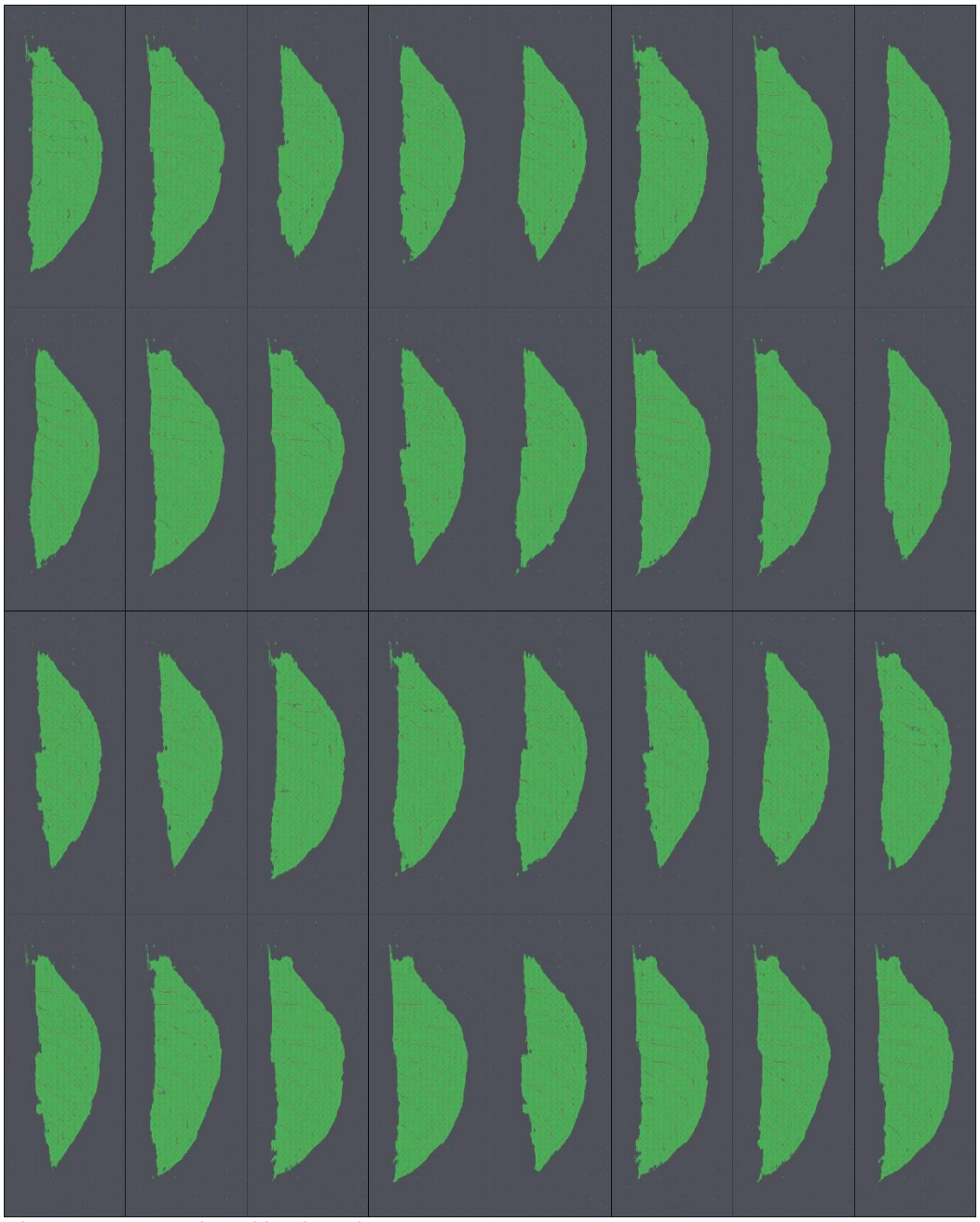

Figure 14: 32 samples taking from the GAN.

8.2 Variational AutoEncoder

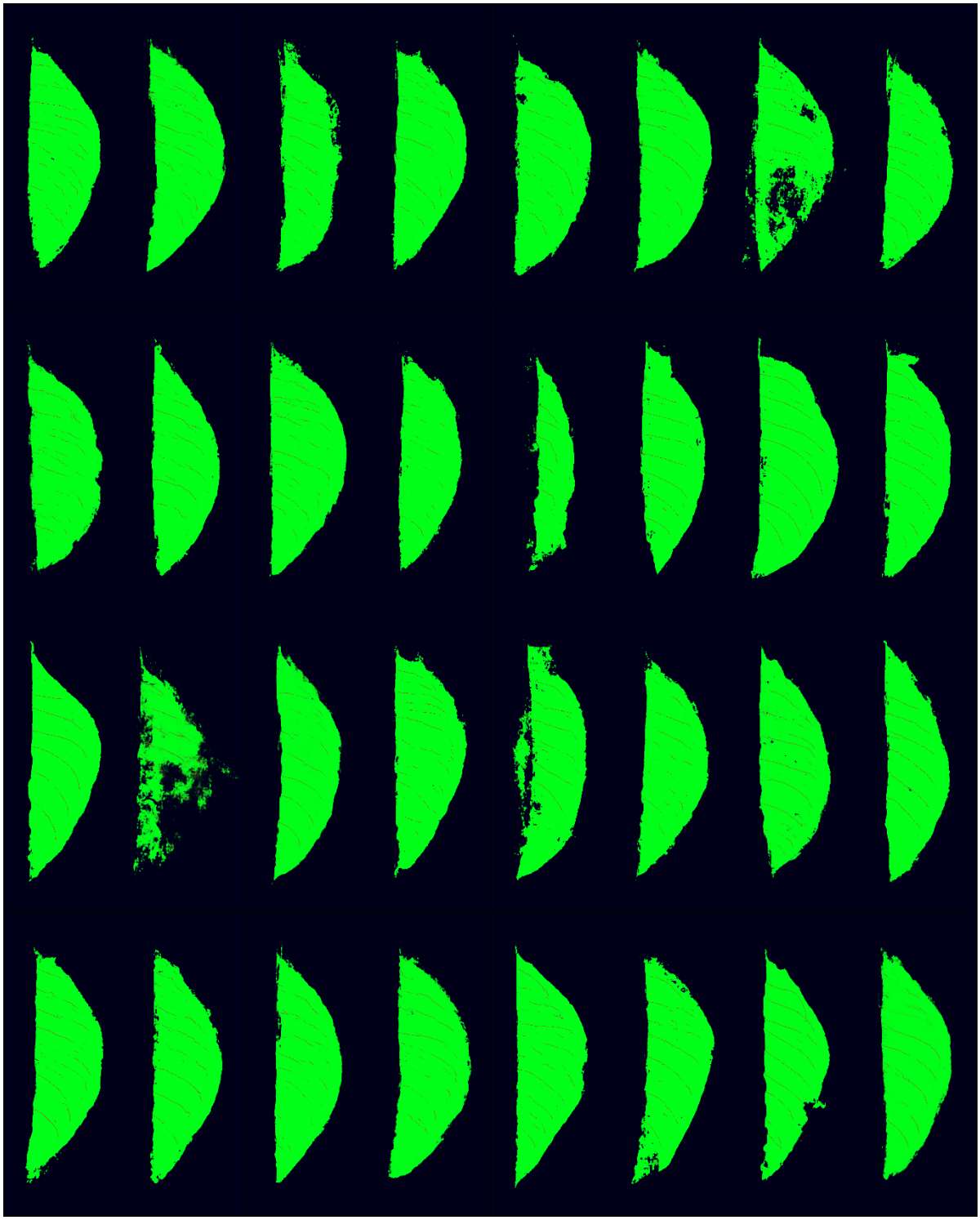

Figure 15: 32 samples taking from the VAE.

Mailinglist

CQM B.V.

Rutger van Beek Rutger.vanBeek@cqm.nl

CONSULTANTS IN QUANTITATIVE METHODS

Countless variables, innumerable x's. Are they coincidental or structural? How do they interrelate and what effect do they have? What is really important and what not? We help organizations make complex processes transparent. Using quantitative models, we create a framework to analyze processes and make decisions based on the facts, enabling you to optimize your planning and logistics, and improve your product and process innovation. Intelligence, that takes your organization to a lasting, higher level. We analyze and clarify, with a genuine understanding of the issues you face. That is the way we do things.

From x to u

CQM B.V.

T +31 40 750 23 23
F +31 40 750 16 99
E info@cqm.nl
I www.cqm.nl

Vonderweg 16
5616 RM Eindhoven
P.O. Box 414
5600 AK Eindhoven
The Netherlands

Trade register 17076484
IBAN NL61RABO0359340598
VAT NL801228505B01