University of Southern Denmark

Master's Thesis

Applying Generative Adversarial Networks for plant stress detection

June 2018

Ahmad Gheith
ahghe13@student.sdu.dk

Supervisor: John Hallam
john@mmmi.sdu.dk

Declaration

I hereby declare that I have produced this work by myself. All used sources are listed in the bibliography.
This work has not been submitted to any other board of examiners.

Signature, 1st June 2018

Abstract - In this work, I aim to predict the soil moisture of Campanula plants based on multispectral images by using the discriminative model of Supervised Generative Adversarial Networks (sGAN).
The data set containing multispectral images of Campanula plants with different levels of soil moisture, was sliced into smaller images in the spatial domain.
Generative Adversarial Networks (GAN), which make up the baseline of sGAN, were implemented as two different sizes of network pairs named Mini Size GAN and Full Size GAN. After training, the Mini Size GAN had successfully learned to imitate data images produced by a fixed algorithm. The Full Size GAN was trained to imitate a specific channel of the plant images and has succeeded in producing images that look authentic.
Both GANs were modified to become sGANs. The Mini Size sGAN has successfully learned to identify brightness of specific pixels in data images produced by a fixed algorithm. However, the Full Size sGAN did not succeed in learning to identify the soil moisture when trained using all channels of the multispectral images, which gave reason to investigate whether excluding some channels would yield better results.
Four subsets of the channels were selected. The member channels of the first subset were selected based on a correlation-based approach, where the least correlated channels were included. The member channels of the second subset were selected based on the reflection of the Campanula. The third and the forth subsets were selected using the Sequential Forward Feature Selection algorithm fed with the first and the second subsets.
For each of the selected subsets, an sGAN was trained. Unfortunately, none of them were able to predict the soil moisture.
The obtained results indicate that the information required for predicting the soil moisture is not present in the multispectral images of the Campanula.

Contents

List of Figures

Chapter 1

Introduction

This chapter introduces the thesis by presenting the problem it deals with. Its objectives are listed and elaborated and the achievements are shortly mentioned. Furthermore, the overall structure of the thesis is explained.

1.1 Introduction

In the recent years, the popularity of Artificial Intelligence (AI) has increased significantly. In industry, there are countless of success stories related to AI, where the technology has been able to increase productivity and profitability.
Danish garden centres are interested in taking part in this development as well, in order to ensure competitiveness and improve their business.
This project originates from a group of Danish garden centres' interest in automating the detection of stressed plants. The Danish garden centre, PKM, is at the forefront of this project. At the moment, the job of detecting water-stressed plants is carried out manually by experts, which is expensive and suboptimal due to human error.
This thesis is conducted in cooperation with the Danish Technological Institute (DTI). DTI was responsible for the data collection and has offered an introduction to the project in the preliminary phase. The data collection was conducted by DTI, in a cooperation with PKM.

1.2 Motivation

As mentioned earlier, detecting stressed plants is carried out manually by experts. The labour of the experts is usually expensive. Moreover, the job is intensive, as the greenhouse areas are usually extremely large.

At PKM, for instance, the greenhouse area makes up 190,000 m^2 [1]. Automating the job of detecting stressed plants will reduce the expenses of the companies significantly. In addition, it will improve the quality of the products, as it will ensure a better environment in the greenhouses. These factors will contribute to making the companies competitive in the international market.

1.3 Problem background

Plants can become stressed for several reasons, such as water drought, excess light, pathogens etc. The response of the plants differs depending on the species and type of stress. The most common responses of stressed plants are color shifting and wilting.
Detecting stressed plants requires expertise, as there are numerous factors to look for. Plants and stress can be categorized into four categories: healthy plant, stressed recoverable plant, stressed unrecoverable plant, and dead plant. The term recoverable is to be perceived as "recoverable and worth recovering", as recoverable plants that do not pay off are categorized as unrecoverable plants. Stressed plants have to be detected as early as possible. The detection is optimal when it happens in the stressed recoverable phase, which requires an even higher level of expertise. Detecting stressed unrecoverable plants is beneficial as well, as it allows the gardener to prevent further infection of plants in the area.
For simplicity, water-stressed Campanula is the only species considered in this thesis. The Campanula plants were intentionally water-stressed. Images of the plants were then captured with a multispectral camera and provided for this project, along with information about the soil moisture of the plant at the time when the images were captured.
Water-stressed Campanula plants are usually not detected visually. Experts need to lift them and compare their weight and size. If the plant feels too light, it means the soil is dry and the plant is thereby water-stressed.
According to the Production Advisor of PKM, Niels Erik Andersson, it is not possible to detect water stress of Campanula plants based on the images provided for this project. This makes the task very difficult, as the information required to detect water stress might not be present in the images at all.

1.4 Objectives

The main objective of this thesis is to construct a discriminative model that is able to classify the level of water-stress in Campanula plants based on multispectral images. The model must be an Artificial Neural Network trained using the setup of Generative Adversarial Networks (GAN). This objective is decomposed into the following sub-goals:

- Construct a GAN and adapt it to be capable of handling an arbitrary number of channels.
- Implement an appropriate training algorithm.
- Evaluate and confirm the performance of the GAN.
- Construct the GAN-variety, sGAN, that is able to identify certain attributes in images.
- Adapt the training algorithm for the sGAN.
- Evaluate and confirm the performance of the sGAN.
- Explore the possibilities of constructing a Discriminator based on the sGAN.

In order to achieve these goals, two computers were at the disposal of the project: A Samsung laptop and the supercomputer, ABACUS 2.0. Their specifications are shown in Appendix A.1.

1.5 Achievements

In this project, I have successfully implemented two GAN architectures, which are able to handle images of two different dimensions. The implementation is able to handle an arbitrary number of prespecified channels by automatically constructing well-suited GAN architectures. The GAN is trained using the Adam optimization algorithm. After training, the GANs were able to produce images that look similar to the genuine data. Both GAN models have been modified to sGAN models. An sGAN is able to control and identify attributes in images. The sGAN that handles the smaller images has successfully been able to learn the desired image attribute. However, the second sGAN has failed to learn to predict the soil moisture in the provided multispectral images of Campanula plants. In an attempt to make sGAN work, the data has been compressed by producing a fingerprint for each image. An sGAN was constructed and trained based on the these fingerprints. Unfortunately, no useful results were achieved.

In a second attempt to make the sGAN work, two subsets of the channels of the multispectral images were used for constructing new sGAN models. The first subset was selected based on the least correlated channels. The second subset was manually selected based on the transition of the Campanula from non-reflective to reflective wavelengths. Neither approach yielded useful results.
Furthermore, the Sequential Forward Feature Selection algorithm was implemented and applied to find a subset of the selected subsets, that optimizes the performance. However, none of them yielded useful results. Thereby, I infer that the information needed to predict water-stress in Campanula is in fact not present in the multispectral images.

1.6 Structure of thesis

The Introduction of the thesis introduces the problem and objectives of the project. Next, the Theory chapter provides the necessary background knowledge to understand the methods being used. This mainly includes Artificial Neural Networks and Generative Adversarial Networks. Afterwards, the Data chapter presents a detailed overview of the data and explains how it was preprocessed. In addition, the chapter explains how the fingerprints were produced. The Generative Adversarial Networks chapter contains a description of the architecture of the Mini GAN and the Full Size GAN. The results obtained are described. Afterwards, the Supervised Adversarial Networks chapter introduces the concept of sGAN. Furthermore, the results obtained by it are described. In the Feature Selection chapter, the methods Correlation-Based Feature Selection and Transition-Based Feature Selection are explained and their results based on sGAN are presented. Next, Sequential Forward Feature Selection is explained and its results are described. Finally, an overall conclusion is given and future work is proposed.
The the source code produced in this thesis is publicly available at: `github.com/ahghe13/master_thesis_ahghe13_2018`

Chapter 2

Theory

This chapter provides the necessary background theory required to understand the methods applied in this thesis. Artificial Neural Networks and their most significant submodels are introduced, in addition to their most common building blocks, behaviour and training algorithms. Moreover, the concept of Generative Adversarial Networks (GAN) is introduced along with the Adam optimization algorithm.

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a big family of biologically inspired learning models that constitute a substantial part of Artificial Intelligence and Machine Learning. An ANN consists of a collection of nodes which corresponds to the biological neurons. The nodes are interconnected using weighted connections, corresponding to biological synapses. The nodes can be interconnected in a variety of different ways, which hugely affect their behaviour and learning ability, and thereby classifies them into submodels.

2.1.1 Submodels of Artificial Neural Networks

The way of connecting the nodes falls mainly into a two categories: Recurrent Neural Networks (RNN) and Feedforward Neural Networks (FFNN). In RNNs, the nodes are structured in a directed graph, where nodes can be connected to any other nodes in the network. In FFNNs, the nodes are structured in layers. Nodes belonging to a given layer can only be connected to nodes from the next layer. The first and the last layers are referred to as input and output layer, respectively. This way of connecting the nodes steers the flow of data toward the output layer. Thereby, for each input the network will produce a deterministic out-

put. In addition, it disallows previous data processing to affect future processing of data, when not training.
FFNNs are again divided into two submodels: Single-Layer Perceptron (SLP) and Multi-Layer Perceptron (MLP). In a SLP, the input and output layer is all there is. In simpler cases, this model might be sufficient. On the other hand, MLP consists of at least three layers: an input layer, an output layer, and one or more layers in between, usually referred to as hidden layers. In MLP as well as SLP, it is a common practice to connect all nodes in a given layer to all the nodes in the next layer. By the time the network is trained, it will learn to suppress the unnecessary connections and enhance the crucial ones.

2.1.2 Behaviour of Neural Networks

The input layer perceives the input of the network and forwards it to the first hidden layer, which thereafter forwards it to the next layer and so on, until the output layer is reached. The output layer provides the output or the prediction of the model. When the data is forwarded, each layer, other than the output layer, adds a bias. The bias can be perceived as an extra node in the layer, that forwards the value 1.
Inputs received by a node are initially multiplied by the specific weights of the connections and thereafter summed up. This sum is called activation, a. Each node has an activation function associated with it, which is a (usually non-linear) function applied to the activation before forwarding it to the next layer. It is a common practice to have identical activation functions for all nodes belonging to the same layer. The action of the nodes is summed up in the following equation:

$$y_i = f_i(a) = f_i(\Sigma(weight \cdot input) + bias) \tag{2.1}$$

where y_i is the output of the node i and f_i is the activation function of the given node. Note that the nodes in the input layer do not behave as Equation 2.1 explains. They just forward the input of the network.

2.1.3 Activation functions

The activation functions play a significant role in the network and must therefore be chosen carefully. Two function properties are relevant to consider when choosing the activation functions: linearity and differentiability. Applying linear activation functions in all layers will only allow the network to predict linear relations. It corresponds to composing multiple linear functions. The result will never become nonlinear. Differentiability is required by the gradient-based training algorithms.

Non-differentiable functions, such as the step function, do not have useful gradients. The gradient of the step function is zero at any point, except the transition point, where it is undefined. In other words, the gradients do not provide useful information about the training direction. The most popular activation functions are the Sigmoid, Tanh, ReLU, and Leaky ReLU, seen in Figure 2.1.

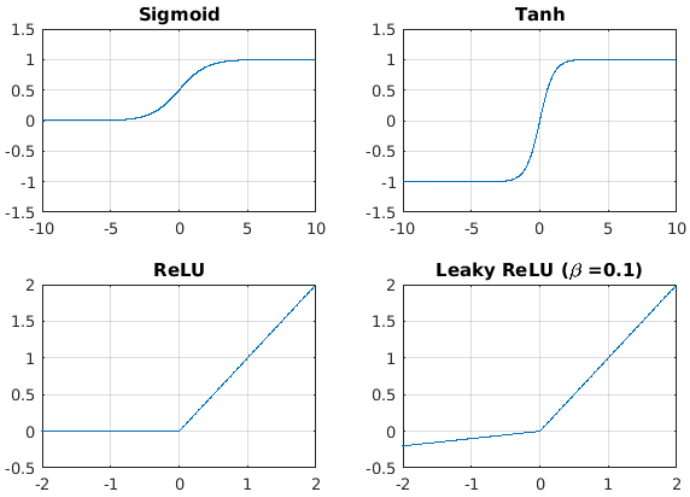

Figure 2.1 – The figure illustrates the activation functions: Sigmoid, Tanh, ReLU and Leaky ReLU.

Sigmoid is described by:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

It is a non-linear function, that restricts the output range to $(0, 1)$. The latter contributes to a more stable network. Moreover, around $x = 0$, the gradients of the function are strong, which enhances learning. However, when x is far from zero, it suffers from vanishing gradients, which is its main drawback.
Tanh is basically a scaled Sigmoid, that restricts the output range to $(-1, 1)$. It is described by:

$$tanh(x) = 2 \cdot sigmoid(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1 \tag{2.3}$$

The similarities between Sigmoid and Tanh are significant and all the pros and cons mentioned regarding Sigmoid apply to Tanh as well.
ReLU deviates from the previous functions. It is given by:

$$ReLU(x) = max(0, x) \tag{2.4}$$

In words, if the activation is negative, the node will forward 0, otherwise it will forward the activation unchanged. Its limits are $(0, \infty)$, which makes it unsuitable for the output layer. The main advantage of ReLU when compared to Sigmoid and Tanh is that it does not have the vanishing gradient problem. Additionally, it is more efficient computationally. ReLU is not differentiable at 0. For training, the gradient at that point is usually defined as 0. ReLU has one major drawback: if the weights are updated in such a way that the activation of the node is always negative, it will cause the output of the node to be zero forever. The real problem though is that the gradients of negative activations are also zero, which means the problem is irreversible and cannot be fixed through training. In order to fix this issue, LeakyReLU was introduced, see Figure 2.1. LeakyReLU is given by:

$$LeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \beta x & \text{otherwise} \end{cases} \tag{2.5}$$

where β is a small positive slope for negative values of x. This ensures non-zero gradients despite negative activations. Empirically, ReLU and its derivations have proven better performance than traditional activation functions in hidden layers.

2.1.4 Training Neural Networks

Neural Networks can be trained to become function estimators. This is achieved by iteratively adjusting the weights of the network based on the error between the actual output, $\hat{y}$, and the expected output, y. The function used to compute the error is called the loss function. A loss function must satisfy two requirements: firstly, if $\hat{y} = y$, the error should be zero and secondly, the larger the difference between $\hat{y}$ and y, the larger should the error be. The most common loss function is the Mean Squared Error (MSE), seen below:

$$MSE = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 \tag{2.6}$$

where m is the number of data samples fed into the network.
Due to the nature of MSE, it performs well when applied for linear regression problems. However, for logistic regression problems, Binary Cross Entropy (BCE) outperforms MSE. BCE is given by:

$$BCE = -\frac{1}{m} \sum_{i=1}^{m} (y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)) \tag{2.7}$$

The training process is an optimization problem that is equivalent to the problem of minimizing the loss function, $\mathcal{L}$. This is achieved by computing the partial derivative of $\mathcal{L}$ with respect to each weight in the network, which provides the gradients for the weights that point in the direction that maximizes the error. As we aim to minimize the error, the weights are updated by subtracting the gradients, multiplied by a scaling factor, α, also referred to as the learning rate. This procedure is called gradient descent and is described as:

$$w^{t+1} = w^t - \alpha \frac{\partial \mathcal{L}}{\partial w} \tag{2.8}$$

where t is the current iteration of weight update [2].
An influential factor to consider when training is the batch size, n, which is the number of data samples fed into the network each iteration. There are mainly three approaches for handling batch size while training: stochastic, batch, and mini-batch gradient descent [3]. Stochastic gradient descent is where each weight update iteration is based on a single sample. No gradients are summed up before updating the weight. This approach is efficient, but depending on the problem manifold, it might get stuck in local minima and never find a good solution.
Batch gradient descent is where all the data samples are fed into the network at once. All the gradients are summed up and applied at once. This approach is computationally heavy, as each weight update might require a huge amount of computations, depending on the size of the data set.
The mini-batch gradient decent is a middle ground. A mini-batch, which is a subsample of the data samples with a predefined size, is fed into the network and the weights are updated based on that. In most cases, the mini-batch approach gives the best result.

2.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) is a branch of FFNNs that usually deals with imagery data. A CNN is similar to an FFNN in several ways, as it is composed of nodes, trainable weighted connections, and activation functions. Its components are also arranged in an input layer, one or more hidden layers, and an output layer. Moreover, it is trained using a loss function by applying the same optimization techniques [4]. Usually, the input of a CNN is an image and the output is one or more scalars, which makes it suited for image classification [5].
The raw imagery input is fed into the network as a set of pixel values.

Each pixel in the image is fed as an input for a specific node in the input layer. These are then forwarded through the network, as explained previously, until the processed data reaches the output layer.
What uniquely defines CNNs is the convolution layer, that convolves the input and forwards the result to the next layer. The size of the kernels applied when convolving is a predefined hyperparameter, and the weights of it are the trainable parameters. This allows the convolution layer to learn key patterns in images automatically. A CNN can include one or more convolution layers.
CNNs can also include deconvolution layers. As the name suggests, it deconvolves the input and forwards the result to the next layer. CNNs consisting of deconvolution layers will be able to produce images. The size of the kernels of a deconvolution layer is a hyperparameter, and the weights are the trainable parameters. This allows the network to learn key patterns and repeat them in the output image.
CNNs usually also include batch normalization layers. These layers are used to scale the input from the previous layer based on trainable weights.

2.3 Generative Adversarial Networks

Generative Adversarial Networks were introduced in 2014 by Ian Goodfellow [8]. It is a setup of two networks that compete with each other to improve their performance. The networks are named the Generator, G, and the Discriminator, D. G is capable of producing images and the D attempts to distinguish between genuine images from the data set and those produced by G. Figure 2.2 illustrates the way GANs work.

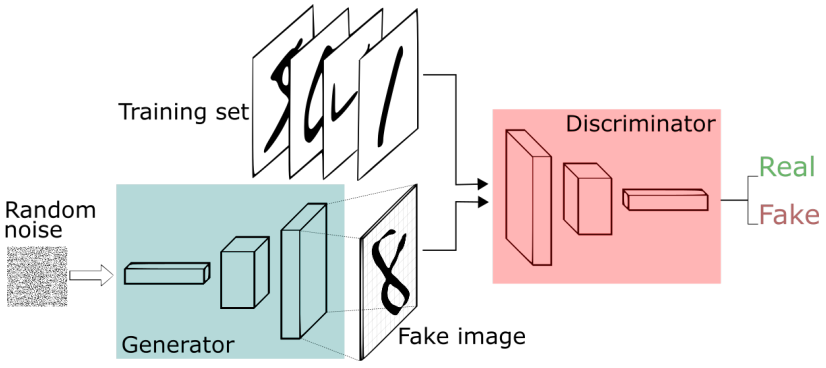

Figure 2.2 – GAN setup. The Generator attempts to produce images from the same distribution as the genuine images. The Discriminator attempts to distinguish genuine images and those produced by the Generator. *Retrieved from [7].*

The Generator is a Deconvolutional Neural Network, that maps a noise vector, z, from a predefined distribution, p_z, to an image, $G(z, \theta_g)$, where

θ_g is the parameters of G.

The Discriminator is a Convolutional Neural Network, that maps an image, x, into a scalar value, $D(x, \theta_d)$, where θ_d is the parameters of D. The output scalar is the probability of x coming from the genuine data distribution, p_{data}, rather than the distribution from which G produces its images, p_g.

The GAN is trained by training D to assign correct labels for both genuine data and samples produced by G, where:

$$D(x) = \begin{cases} 0 & \text{if } x \sim p_g \\ 1 & \text{if } x \sim p_{data} \end{cases} \tag{2.9}$$

where $x \sim p$ denotes that x comes from distribution p.

In other words, D is trained to maximize $log(D(x))$ when $x \sim p_{data}$ and to minimize $log(D(G(z)))$. Simultaneously, G is trained to maximize $log(D(G(z)))$, which means it will attempt to fool D to think that $G(z)$ comes from p_{data}, rather than p_g. In order to do so, G must produce images that look more and more genuine to D as training goes on. This is achieved by updating θ_g in a way that makes the distribution p_g more similar to the distribution p_{data}.

The competition can be expressed as:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}[log(D(x))] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{2.10}$$

The training procedure goes as explained in Algorithm 1.

Algorithm 1: Training algorithm for GAN. n is total size of the data, m is the size of the mini batch, and e is the number of training epochs. f is a function that updates the parameters.

Result: Optimizes the parameters of the GAN: θ_g and θ_d.

$itr \leftarrow floor(\frac{n}{m})$;
for i *in* e **do**
  Shuffle data;
  for j *in* itr **do**
    $x \leftarrow \{x_1, x_2, ..., x_m\} \sim p_{data}$;
    $z \leftarrow \{z_1, z_2, ..., z_m\} \sim p_z$;
    $\Theta_d \leftarrow \nabla\theta_d \frac{1}{m} \sum_{i=1}^{m} [logD(x_i) + log(1 - D(G(z_i)))]$;
    $\theta_d \leftarrow f(\theta_d, \Theta_d)$;
    $\Theta_g \leftarrow \nabla\theta_g \frac{1}{m} \sum_{i=1}^{m} logD(G(z_i))$;
    $\theta_d \leftarrow f(\theta_g, \Theta_g)$;
  end
end

After computing the gradients, the parameters of the network should be updated. There exist several optimization algorithms to improve learning. An algorithm that has proven itself exceptional in various applications is the Adaptive Moment Estimation (Adam) Optimization Algorithm [9]. Adam Optimization is widely used in GANs and will be applied in this thesis as well. The Adam optimization algorithm is described in Algorithm 2.

Algorithm 2: Adam optimization algorithm. θ is the parameters of the network, Θ is the first derivative gradients of θ, and Θ^2 is the second derivative gradients in the main diagonal of the hessian matrix of θ. β_1, β_2, and ϵ are hyperparameters. α is the learning rate.

Result: Improved updated network parameters, θ.

Initialize the algorithm by:
$m_0 \leftarrow 0$;
$v_0 \leftarrow 0$;
$t \leftarrow 0$;
Function $Adam(\theta, \Theta)$
    $t \leftarrow t + 1$;
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \Theta$;
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \Theta^2$;
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$;
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$;
    $\theta \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
end

Adam Optimization Algorithm has four hyperparameters: β_1, β_2, ϵ and α. β_1 and β_2 must be within the range $(0, 1)$. As default, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. ϵ is a small value that prevents the algorithm from dividing by zero. Its default value is $\epsilon = 10^{-8}$. The learning rate, α, is the hyperparameter to adjust to optimize the learning. Therefore, it does not have a default value.

Theoretically, as the training goes on, the distribution p_g will become more and more similar to p_{data}, as the parameters θ_g are updated. Similarly, D will become better at distinguishing between p_g and p_{data}, until a point, where they become semi identical. At that point D will no longer be able to distinguish between genuine images and those produced by G, and will therefore output 0.5 for both inputs. The theoretical results are illustrated in Figure 2.3.

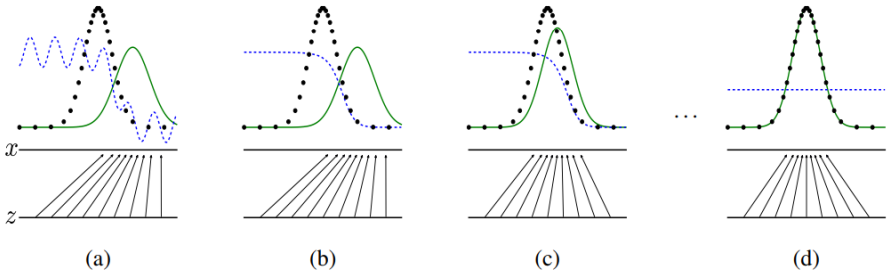

Figure 2.3 – The random noise, z, is mapped to an image, x, by a mapping function, G. The arrows illustrate the mapping. The green curves illustrate the mapping distribution, p_g. The black dotted curves illustrate the distribution of the data, p_{data}. The blue dashed curves illustrate the distinguishing function, D. In an early phase, (a), p_g and p_{data} are separated and D can distinguish between them. After training, (b) and (c), p_g and p_{data} starts overlapping and D's job gets more difficult. In the end, (d), the distributions overlap completely, and D can no longer distinguish between them. *Retrieved from [8].*

After the training has finished, the GAN can be evaluated in several ways depending on the purpose of it. The evaluation is further explained when needed according to the purpose.

Chapter 3

Data

In this chapter, the data acquisition is discussed, including the circumstances under which they were captured. Furthermore, the preprocessing of the data is elaborated which includes the algorithms applied to crop and slice the data. In the end of the chapter, an alternative representation of the data is suggested.

3.1 Introduction

The obtained data set consists of a number of multispectral images of Campanula plants. The quality of these images is very crucial, as they will be used for both training and evaluating the produced networks.
It is well-known that the spectral reflectance of the leaves of some plant species change based on the level of water-stress [6]. Assuming this applies for Campanula plants, we should be able to extract this piece of information based on the multispectral images.
The level of water stress in the leaves is advanced to extract. Therefore, the level of soil moisture will be used as ground truth due to its high correlation with water stress.
The Campanula plants in the images were intentionally water stressed at different levels, hoping that their reflection rate will vary and expose information about how stressed they are.

3.2 Data acquisition

The data collected for this project was captured by the company Danish Technological Institute in cooperation with Gartneriet PKM. The horticulture company, PKM, put 100 Campanula plants at DTI's disposal.

The plants were brought to a special room at PKM, where the temperature, brightness, and humidity were carefully controlled, in order to avoid leaving these factors to coincidence.
DTI set up a cell for data collection. The cell consisted of a semi-closed box, a conveyor belt, a Specim FX10 Camera, and a number of halogen lamps. The conveyor belt brought objects inside the box and the halogen lamps shed light on the input object in order to allow the Specim FX10 Camera to capture multispectral images.
The reason behind using halogen lamps inside the cell is that the spectral band of the light radiation produced by these lamps is very wide. It usually ranges from 400 to 1100 nm, see Figure 3.1.

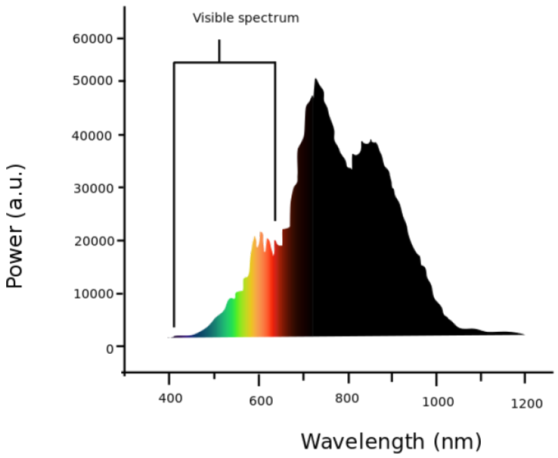

Figure 3.1 – Wavelength of the spectral radiation of Halogen Lamps. *Retrieved from: wikipedia.org/wiki/Halogen_lamp.*

The Specim FX10 Camera is an industrial camera that captures images with 224 channels in the spectral band 400 to 1000 nm. It produces images with the dimensions 1024x301x224.
The data were collected over 5 days. During this period, none of the plants were watered. Each day, 20 Campanula plants were transported to DTI and fed into the cell, where a multispectral image was produced for each plant, see Figure 3.2.

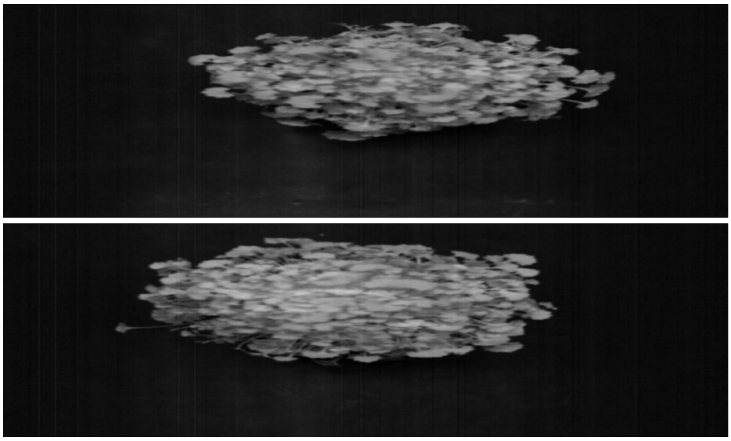

Figure 3.2 – Channel 163 of two multispectral images produced by Specim FX10 Camera is showed. The upper plant has not been watered for 1 day. The lower plant has not been watered for 5 days.

In addition, the soil moisture for each plant was measured at the time of data collection and associated with the respective multispectral image.

3.3 Data preprocessing

DTI provided 100 multispectral images of different Campanula plants. Besides the plants, a considerable area of the images was conquered by the conveyor belt in the background, as observed in Figure 3.2. Therefore, the images had to be cropped before going any further, in order to optimize the following processes by saving computation time and reduce the influence of the background noise.

3.3.1 Image cropping

The multispectral images were cropped using a simple algorithm. In words, the algorithm can be explained as follows: The user selects a high contrast channel, and passes the index of it to the algorithm. For this channel, the plant is supposed to appear bright, while the conveyor belt in the background is supposed to appear dark. The algorithm classifies the pixels as bright or dark according to a given intensity threshold. Now, it does a brute force search for the bright pixels that are located farthest to each of the four directions. These pixels make up the borders of the cropped image.
In addition to the image itself, the cropping algorithm requires two input arguments, i.e. the channel index and intensity threshold. These parameters were hand-tuned. The channel index 163 was used due to its relatively sharp contrast. The intensity threshold is a number between 0 and 1, and 0.4 seemed to work well. Using these parameters, all the

multispectral images were cropped, see Figure 3.3.

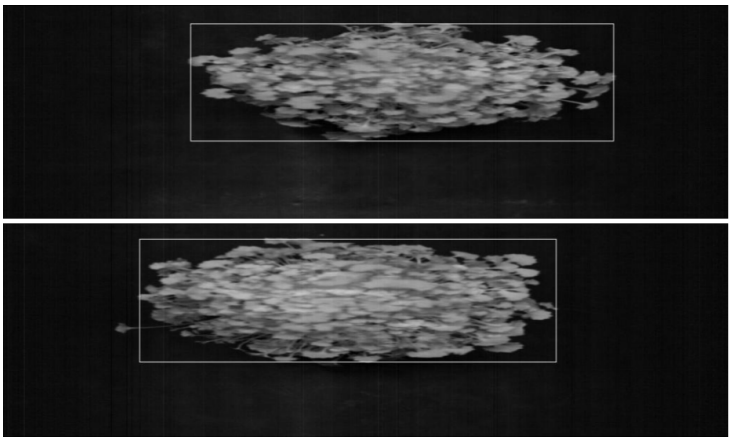

Figure 3.3 – Channel 163 of two multispectral images, where the cropping rectangles suggested by the cropping algorithm is shown. The boundaries were found using channel index 163 and intensity threshold 0.4.

As seen in the figure above, the algorithm performs well. Even though small parts of the plants may be excluded, such as the leaf on the left side of the lower image, the performance is still considered sufficiently good. For this case in particular, expanding the rectangle to include the leaf will lead to adding much more of the background and it might therefore be better to exclude it. Furthermore, perfecting the algorithm will cost much more effort without adding much value. For these reasons, this algorithm was applied as it is.

3.3.2 Image slicing

As the size of the plants varies, the size of the cropped images varies as well. This is actually an issue, as different image sizes require different GAN architectures. In addition, the data set only contains 100 images, which is far too small to train a GAN. Both issues were tackled by slicing the images.

Assuming that the information we are looking for resides in the leaves, we might be allowed to slice the images into smaller ones, where each slice contains a few leaves.

The images were sliced using a simple algorithm, which takes an image as input along with a desired slice size. The produced slices are always square. The algorithm figures out how many slices the image can contain and allocates the slices in the center of the image, in order to include as much of the plant as possible, see Figure 3.4.

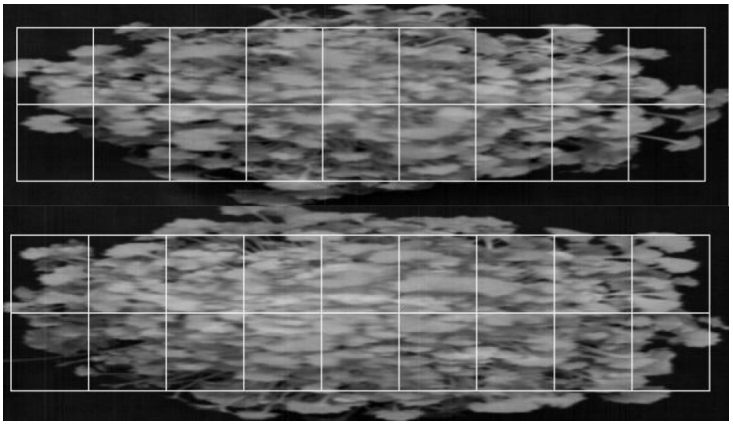

Figure 3.4 – Channel 163 of two multispectral images. The squares illustrate how the image is about to be sliced.

All the images were sliced after they were cropped into squares of dimensions 64x64x224. In addition, they were normalized to the range $(-1, 1)$, in order to comply with the networks to be implemented. Thus 1785 images were obtained. Figure A.1 contains all image channels of a single slice.

3.4 Alternative data representation

It is demanding to teach children to distinguish between the black-headed gull and the little gull. On the other hand, teaching them to distinguish a bird from a cat is much easier. The more obvious the difference between two classes, the easier it is to learn to distinguish between them. The same applies for machine learning.
The alternative data representation will hopefully highlight the water stress and thereby make it easier for the discriminative model to identify the level of water-stress in the plants.

3.4.1 Data fingerprints based on histograms

The data fingerprint is an alternative representation of the data, where each multispectral image is squeezed into a single one-channeled image, based on the histograms of the original image. For each channel in the original image, a histogram is produced. The histograms were then represented as a line of pixels, where the intensity of each pixel represents a score of a bin in the histogram. The number of pixels is therefore identical to the the number of bins. These histogram lines were then collected together, to form the one-channeled fingerprint image. The dimensions of the fingerprint image is the number of channels by the number of histogram bins. In order to get full control over the dimensions

of the fingerprints, they were scaled in the dimension that depends on the number of channels to be applicable for their use.
In order to improve the contrast of the image and attenuate exploded bins in the histograms, an upper limit has been sat for the histogram values. Afterwards, the images were normalized.
All the sliced images were used to produce fingerprint images of the dimensions 64x64. The fingerprints were also normalized to the range $(-1, 1)$ in order to comply with the networks to be implemented. A total of 1785 images were obtained, see Figure 3.5.

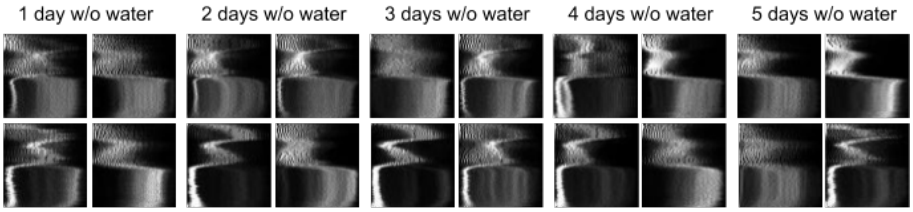

Figure 3.5 – Samples from fingerprints data set.

The fingerprints shown in the figure above do not highlight the water-stress as desired. However, the Production Advisor of PKM, Niels Erik Andersson, expressed the view that these fingerprints should definitely be studied further and used for training.

Chapter 4

Generative Adversarial Networks

This chapter concerns the implementation and evaluation of the Generative Adversarial Networks. Initially, the framework used to implement the GAN is validated by reproducing the results of a previous project. Next, a mini version of the GAN, called Mini GAN is presented along with a description of how its genuine data was generated. The performance of the Mini GAN is evaluated and presented.
Based on the Mini GAN, a Full Size GAN was implemented. The architecture of the Full Size GAN and the results obtained by it are presented.

4.1 Introduction

Training a GAN will produce a Generator and a Discriminator. As explained in a Chapter 2, the Generator will attempt to learn to produce images drawn from the same distribution as the images of the data set, and the Discriminator will attempt to distinguish between the genuine images and those produced by the Generator. Despite the fact that neither of these models are able to complete the main objective of the thesis, they are still very crucial, as they form the baseline of the methods to be applied later on. In this chapter, the main focus will be on demonstrating that training algorithm and the networks are working as expected.
In this chapter, the validity of the framework is discussed, the concepts of Mini GAN and Full Size GAN are presented, their architecture and implementation are explained, and their capabilities are demonstrated.

4.2 Framework validation

The software developed in this project is based on the framework Torch, which is an open source machine learning library based on the programming language Lua. It provides a number of features that facilitate a well-suited environment for developing and implementing deep learning models.

A common way of developing software in engineering is the step-by-step technique, which was also applied in this project. Before moving on to the next step, the previous step must be validated to ensure that everything works as expected. In this section, I will discuss the validation of the framework itself, which forms the foundation of the software.

Installing Torch framework is straightforward, as the library is well documented. The validation of it was achieved by reproducing the results of a previous work that is based on the same framework and, preferably, related to the software required by this project. If the reproduced results are similar to the original results produced by the project owners, it will be assumed that the framework is working properly.

In 2016, the researchers Alec Radford, Luke Metz, and Soumith Chintala released a paper with the title "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" [11]. Along with that paper, they released an open source implementation of a Deep Convolutional Generative Adversarial Network (DCGAN), based on Torch. They have trained 3 DCGANs using 3 different datasets. One of these datasets contains images of faces[1]. For the framework validation, I will attempt to reproduce the results they obtained from the face DCGAN, by training a similar DCGAN with the same dataset, using ABACUS 2.0 (see Appendix A.1).

The data consisted of, approximately, 200,000 face images of celebrities, and was preprocessed locally, in order to be compatible with the DCGAN.

The DCGAN consisted of a Generator and a Discriminator, as explained in the theory chapter. The Generator consisted of:

- Four deconvolutional layers, each followed by a batch normalization and a ReLU layer.
- A deconvolutional layer followed by Tanh activation function.

The Discriminator consisted of:

- A convolutional layer followed by a leakyReLU layer.

[1]Their code and dataset can be found at: `github.com/Newmu/dcgan_code`

- Three convolutional layers, each followed by a batch normalization and a leakyReLU layer.
- A convolutional layer followed by a Sigmoid activation function.

The developers applied Adam optimization algorithm to train the networks. After 25 epochs, the Generator produced the images shown in Figure 4.1.

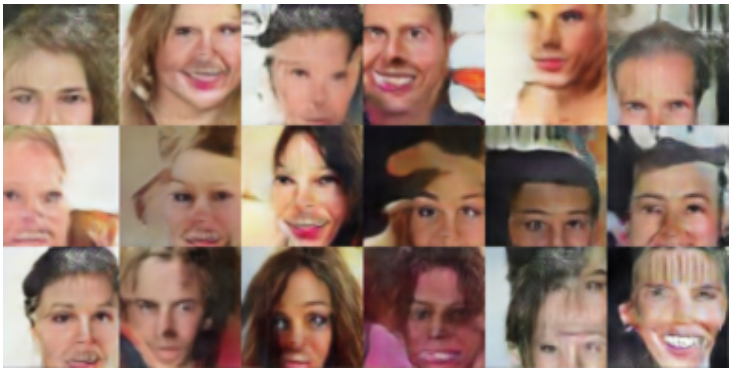

Figure 4.1 – Artificial images produced by Soumith's team by the Generator of their face DCGAN, after 25 training epochs.

After the same number of epochs and using the same implementation and their default options, I was able to produce the images shown in Figure 4.2.

Figure 4.2 – Artificial images produced using the implementation of Soumith's team by the Generator of the face DCGAN, after 25 training epochs.

It was not expected to achieve the exact same results, as they depend on the input noise vector.
However, the images produced by the Generators have several properties in common. First of all, both generators produce close to realistic images of faces. The eyes, nose, and mouth are placed correctly relative to each other. Furthermore, hair, hair color, skin and skin color are generated

mostly realistically. Still, a few details make the images look weird. In some of the images, the face is indistinguishable from the background. Moreover, in a few images, some features are repeated more than expected, such as more than two eyes and other times, the features are deformed.
Due to these similarities between the images produced by the two Generators, it will be assumed that the underlying framework is installed properly and works as intended.

4.3 Mini GAN

The Mini GAN is an ordinary GAN designed to handle very small images. By reducing the size of the GAN, the time required for processing and training the networks is likewise significantly reduced.
The purpose of it is to reduce the level of complexity of the GAN while implementing the setup and the training algorithm. Furthermore, it will be used for implementing and testing new features in the next chapter. After demonstrating the performance of the Mini GAN, it will be scaled up to the Full Size GAN, which is able to handle the images introduced in Chapter 3.
The Mini GAN was trained on the Samsung laptop, see Appendix A.1. In this section, the implementation and the achieved results of the Mini GAN are presented.

4.3.1 Implementation and architecture

The Mini GAN is designed handle images of the dimensions 4x4xC, where C is an arbitrary number of channels. The implementation allows the user to specify a sequence of channels to be used. For instance, if the data images are built up of RGB channels, the user can easily select the red and blue channels only. Any combination of the available channels is possible. C is then the size of that sequence, and the implementation will automatically build suited networks for the specified C-dimension.
The Generator of the Mini GAN consists of the following layers:

- *Devonvolution layer, batch normalization, and ReLU layer:* The deconvolution layer takes a noise vector of 100 elements as input. The size of its kernels is 4x4 and the stride is 2. Its input is padded with 1 pixel in each spatial dimension. The output of this part of the Generator is a matrix with the dimensions 2x2x4.

- *Devonvolution layer and Tanh layer:* The deconvolution layer has the same hyperparameters as the previous deconvolution layer, ex-

cept that it takes an input image of the dimension 2x2x4. The output is an image of the dimensions 4x4xC whose pixel intensities lie in the range $(-1, 1)$.

The architecture and the hyperparameters were partly hand-tuned and partly inspired by the literature. This applies for all the the networks constructed in this thesis, unless otherwise stated.
As explained in the architecture, the Generator takes a noise vector of 100 elements as input, which is the default size in [11]. The noise is drawn from a standard Gaussian distribution, as advised in [14]. The size of the noise vector is an influential factor, that has to match the number of Degrees Of Freedom (DOF) in the data images for optimal results. The DOF in images is an abstract concept and the size of it can be estimated empirically, but is outside the scope of this project. The size of it is somewhere between 1 and the number of pixels in the image. Theoretically, an overdose of noise would not affect the generative and discriminative power of the networks. However, it will affect the computation time, as there will be more data to process.
The Discriminator consists of:

- *Convolution layer and LeakyReLU layer:* The convolution layer takes an image of the size 4x4xC as input. The size of its kernels is 4x4 and the stride is 2. Its input is padded with 1 pixel in each spatial dimension. For the LeakyReLU, $\beta = 0.2$. The output is a matrix with the dimensions 2x2x4.

- *Convolution layer and Sigmoid layer:* The convolution layer has the same hyperparameters as the previous convolution layer, except that it takes an input matrix of the dimension 2x2x4. Thus the output becomes a scalar in the range $(0, 1)$.

The training algorithm for GAN is an implementation of Algorithm 1. The parameters were updated according to Adam, Algorithm 2, using the default values described in the previous Chapter. The learning rate was hand-tuned to 0.0005.
The parameters of the Generator's deconvolution layers and the Discriminator's convolution layers were initialized with noise values from a Gaussian distribution, where the mean is 0 and the standard deviation is 0.02. The batch normalization layers in both networks were initialized based on values drawn from a Gaussian distribution with a mean of 1 and a standard deviation of 0.02. These values were inspired by [11], and provide a good starting point for training.
Before training starts, the implementation shuffles the data and splits it

up in two parts: training set and validation set. The training set consists of 80% of the data set and is used for training. The validation set consists of the last 20% of the data set and is used for evaluating the trained networks.

4.3.2 Genuine data generation

As the Mini GAN is designed to handle small images, it requires a separate data set. The images in the data set must be compatible with the Mini GAN, which means they must be of the dimension 4x4xC. For simplicity, $C = 3$, and corresponds to RGB.
The data set for the Mini GAN was generated from a single ideal image. Every image in the data set was a manipulated copy of the ideal image. The manipulation was carried out by multiplying every entry in the copy of the ideal image with a random number from a Gaussian distribution with the mean, $\mu = 1$, and standard deviation, σ. Figure 4.3 provides results achieved using three different σ-values.

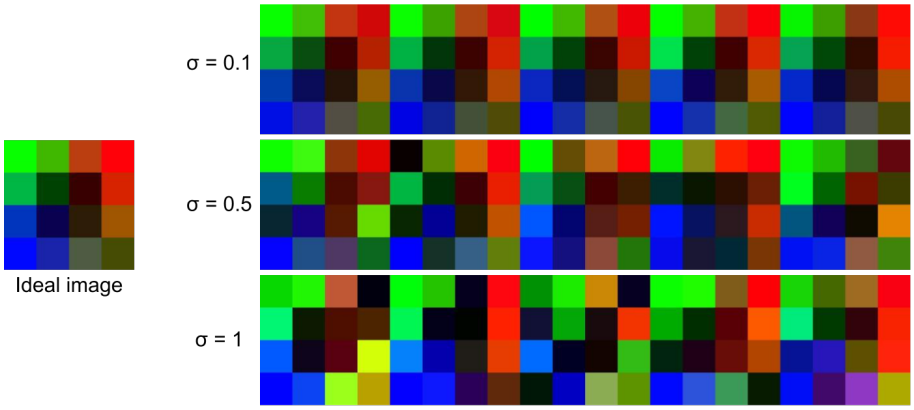

Figure 4.3 – The ideal image was used to produce the data by multiplying each entry by a random factor drawn from a Gaussian distribution, $X \sim \mathcal{N}(1, \sigma)$. The figure illustrates samples of data sets produced with the $\sigma = 0.1$, $\sigma = 0.5$, and $\sigma = 1$.

The final data set used for training the Mini GAN was produced by applying a standard deviation, $\sigma = 0.5$. A total of 2000 genuine images were produced, a number that is close to the size of the genuine plants data set. In principle, the procedure in which the genuine data has been generated does not restrict the range of the pixel intensities. The images were therefore normalized to lie in a range, that comply with the Mini GAN. As the Generator output is restricted to the range $(-1, 1)$, the genuine images were linearly normalized to that range.

4.3.3 Results

The Mini GAN was trained for 30 epochs. The capability of the Generator is illustrated in Figure 4.4.

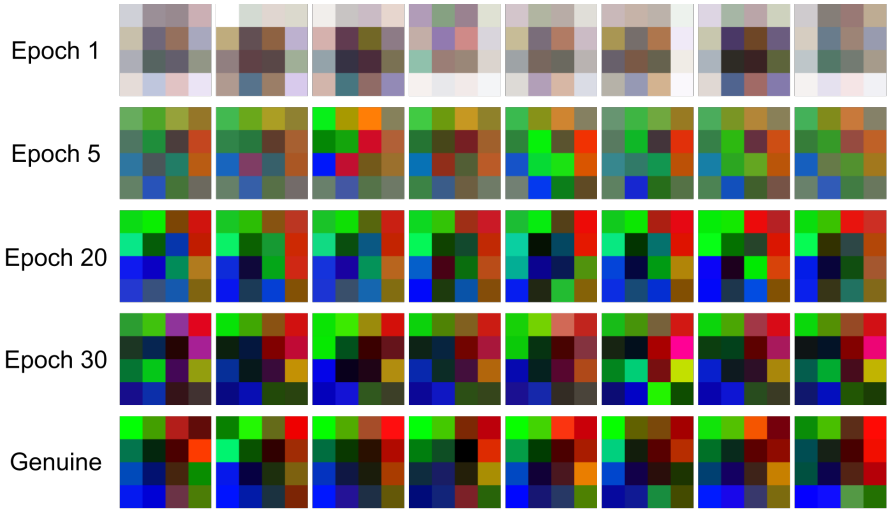

Figure 4.4 – Images generated by the Generator of the Mini GAN. Each row contains 8 images produced after training the Mini GAN for certain number of epochs.

The Generator has gradually learned to imitate the genuine data. Before training the GAN, the images produced by the Generator contained nothing but meaningless noise. After one epoch, the images starts getting shaped. After 5 epochs, a few pixels are taking the shade of the genuine images. After 20 epochs, the features and shades of the genuine data are very clear. After 30 epochs, the features start becoming a bit noisy, possibly because of overtraining.

As the data set for the Mini GAN was produced based on a single ideal image, that image can be exploited to evaluate the performance of the GAN.

Recall the parameters used for producing the genuine data set, $\mu = 1$ and $\sigma = 0.5$. Let μ_t and σ_t be the mean and the standard deviation of the ratios between the training set and the ideal image. The ratios are computed by dividing each entry value of the training set images by the corresponding entry value of the ideal image. Thus the factor applied to produce the entry value from the ideal entry value will be obtained. The values of the mean, μ_t, and standard deviation, σ_t, are computed based on the distribution of these factors. Ideally, $\mu_t = \mu$ and $\sigma_t = \sigma$, but the actual values are $\mu_t = 0.94$ and $\sigma_t = 0.65$, which means there is a slight deviation between the ideal and the actual values. However, the deviation expected and lies within reasonable limits.

Now, the performance of Mini GAN can be measured by computing the mean, μ_g, and standard deviation, σ_g, of the ratios between images produced by G and the ideal image, computed in a similar way as μ_t and σ_t. Ideally $\mu_g = \mu_t$ and $\sigma_g = \sigma_t$. Figure 4.5 illustrates how these parameters evolved while training the networks of the GAN.

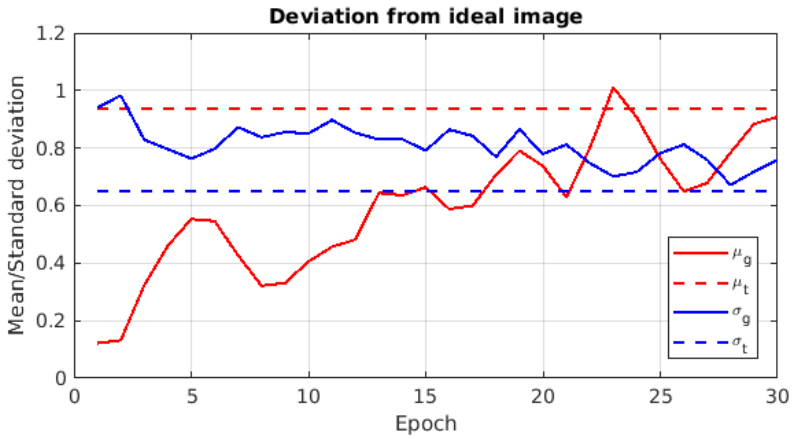

Figure 4.5 – These graphs were produced using 10,000 images produced by the Generator of the Mini GAN. The figure illustrates the mean, μ_g, and standard deviation, σ_g, of the error between the images and the ideal image. The graphs are compared to the mean, μ_t, and standard deviation, σ_t, of the error between the training set and the ideal image.

As seen in the figure above, μ_g and σ_g tend to go toward μ_t and σ_t respectively, which indicates that the training of the GAN is performing as expected. For a specific training epoch, Epoch 23, the both μ_g and σ_g appears to be closest μ_t and σ_t. Images produced at that epoch are shown in Figure 4.6.

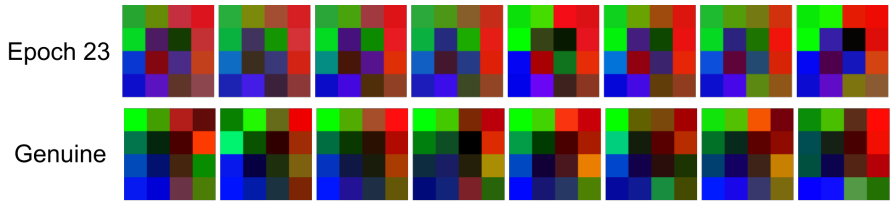

Figure 4.6 – Images produced by the Generator trained for 23 epochs, compared to samples from the genuine data.

Overall, the results are indicating that the setup of the GAN and the training algorithm are working properly. The Generator is learning the distribution of the genuine data, which also indirectly means that the Discriminator is learning to distinguish correctly. Otherwise, the Generator would not have learned to produce such images.

4.4 Full Size GAN

The Full Size GAN is an ordinary GAN, named according to the Mini GAN. It is able to handle the image slices produced in Chapter 3.
The Full Size GAN was trained on ABACUS 2.0, see Appendix A.1.

4.4.1 Implementation and architecture

The Full Size GAN is a scaled up version of the Mini GAN. It has the same setup and uses the same training algorithm. However, the architectures of its networks is differ, which allows it to handle images of the dimensions 64x64xC.
The Generator of the Full Size GAN consists of the following layers:

- *Deconvolution layer, batch normalization and ReLU layer:* The deconvolution layer takes a noise vector with 100 elements as input. The size of its kernels is 4x4 and the stride is 1. No padding is applied. The output is a matrix with the dimensions 4x4x512.
- *Deconvolution layer, batch normalization and ReLU layer:* The deconvolution layer takes a matrix as input with the dimensions of the output of the previous layer. The size of its kernels is 4x4 and the stride is 2. Its input is padded with 1 pixel in each spatial dimension. The output is a matrix with the dimensions 8x8x256.
- *Deconvolution layer, batch normalization and ReLU layer:* The deconvolution layer is similar to the previous one, except that it outputs a matrix with the dimensions 16x16x128.
- *Deconvolution layer, batch normalization and ReLU layer:* The deconvolution layer is similar to the previous one, except that it outputs a matrix with the dimensions 32x32x64.
- *Deconvolution layer and a Tanh layer:* The deconvolution layer is similar to the previous one, except that it outputs an image with the dimensions 64x64xC. The output is normalized by the Tanh layer, which causes the intensities lie in the range $(-1, 1)$.

The noise elements produced for the Generator is drawn from a standard Gaussian distribution. The Discriminator consists of:

- *Convolution layer and LeakyReLU layer:* The convolution layer takes an image of the size 64x64xC as input. The size of its kernels is 4x4 and the stride is 2. Its input is padded with 1 pixel in each spatial dimension. For the LeakyReLU, $\beta = 0.2$. The output is a matrix with the dimensions 32x32x64.

- *Convolution layer, batch normalization and LeakyReLU layer:* The convolution layer takes a matrix as input with the dimensions of the output of the previous layer. The size of its kernels is 4x4 and the stride is 2. Its input is padded with 1 pixel in each spatial dimension. For the LeakyReLU, $\beta = 0.2$. The output is a matrix with the dimensions 16x16x128.
- *Convolution layer, batch normalization and LeakyReLU layer:* The convolution layer is similar to the previous one, except that it outputs a matrix with the dimensions 8x8x256.
- *Convolution layer, batch normalization and LeakyReLU layer:* The convolution layer is similar to the previous one, except that it outputs a matrix with the dimensions 4x4x512.
- *Convolution layer and Sigmoid layer:* The convolution layer takes a matrix as input with the dimensions of the output of the previous layer. The size of its kernels is 4x4 and the stride is 1. No padding is applied. The output is a scalar. Due to the Sigmoid, it lies in the range $(0, 1)$.

The weights of the networks are initialized in the exact same way as explained for the Mini GAN.
For the Adam optimization algorithm, the default values have been used except for β_1, which was selected to $\beta_1 = 0.5$. The learning rate was 0.0002.
Before training starts, the implementation shuffles the data set and splits it up in a similar way as the Mini GAN: 80% training set and 20% validation set.

4.4.2 Results

The Full Size GAN was trained using only channel 163 of the multispectral images processed as explained in Chapter 3. The results are shown in Figure 4.7.

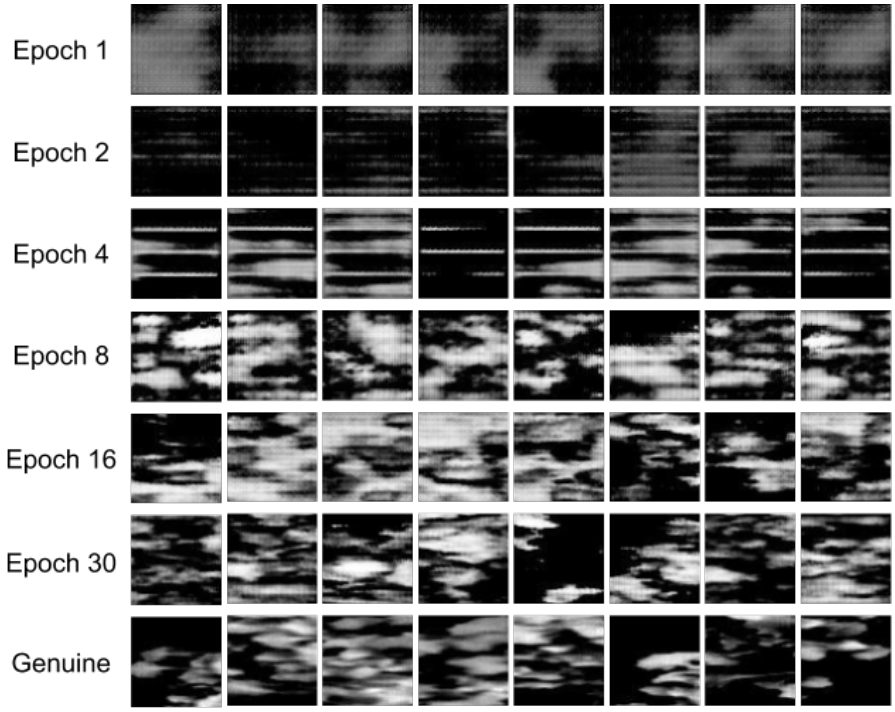

Figure 4.7 – Images produced by the Generator of the Full Size GAN.

After training the network for a single epoch, shades of plants are visible in the generated images. After the second epoch, the performance decays. After 4 epochs, the output images are dominated by strange horizontal lines. After 8 epochs, the performance is hugely improved, and the edges of plants become visible. After 16 epochs, it is unclear whether there is further improvement. After 30 epochs, the plants start looking very authentic.

The performance of the Generator can be evaluated based on the distribution of the Internal Image Distance (IID). The IID between a pair of images is defined as the MSE between the images, see Equation (2.6). In order to get a good feeling of the distribution, a couple of thousands of IIDs must be included. For each epoch, 150 images (corresponding to $11,175$ unique pair combinations) were used to compute IID distributions. Three of these are shown in Figure 4.8.

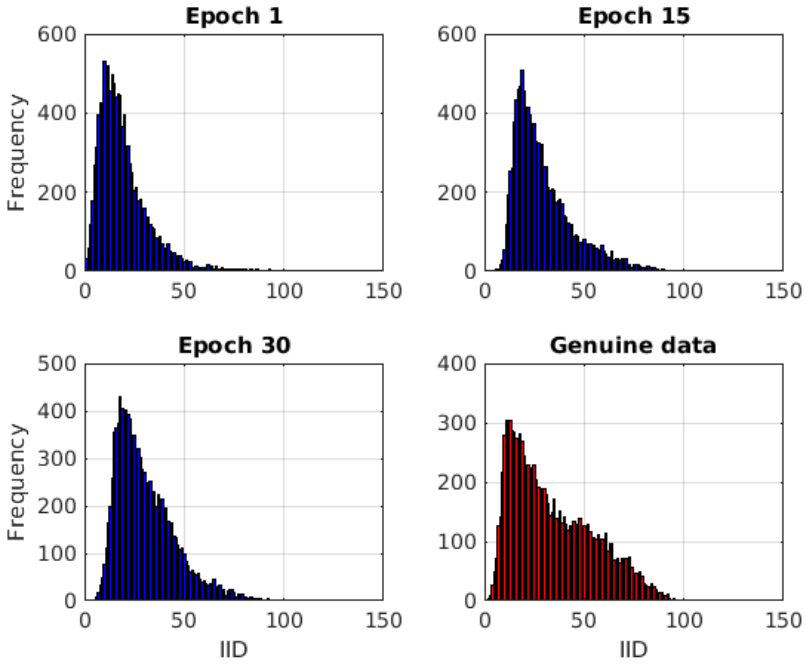

Figure 4.8 – Distribution of IID between pairs of 150 images produced by the Generator after training for 1, 15, and 30 epochs. The lower right diagram is the distribution of IID between pairs of 150 genuine images from the validation set.

All the histograms in the figure above have the same shape: An early peak followed by a tail. Common for the histograms of the generated images is that their peak goes higher than the histogram for the genuine data and their tail decays faster.

In order to assess how well the generated data histograms fit the genuine data histogram, Kullback-Leibler (KL) divergence will be applied. KL divergence is a method used to compute the distance between two distributions, explained in [12]. For two discrete probability distributions, P and Q, KL divergence is given by:

$$D_{KL}(P|Q) = \sum_{i=1}^{n_b} P(i) log \frac{P(i)}{Q(i)} \tag{4.1}$$

where $D_{KL}(P|Q)$ is the KL divergence (a scalar) of how well P fits Q and n_b is the total number of bins in both P and Q.

The closer D_{KL} is to zero, the better does Q fit P. Note that the KL divergence is not symmetric.

Equation (4.1) cannot handle empty bins in Q, e.i. $Q(i) = 0$, as this requires division by zero. In order to get around this issue, the equation

has been slightly modified to:

$$D_{KL}(P|Q) = \sum_{i=1}^{n_b} P(i) log \frac{P(i)}{Q(i)+1} \tag{4.2}$$

The distributions illustrated in Figure 4.8 have been computed for all epochs between 1 and 30. The KL divergence has been calculated, yielding the results shown in Figure 4.9.

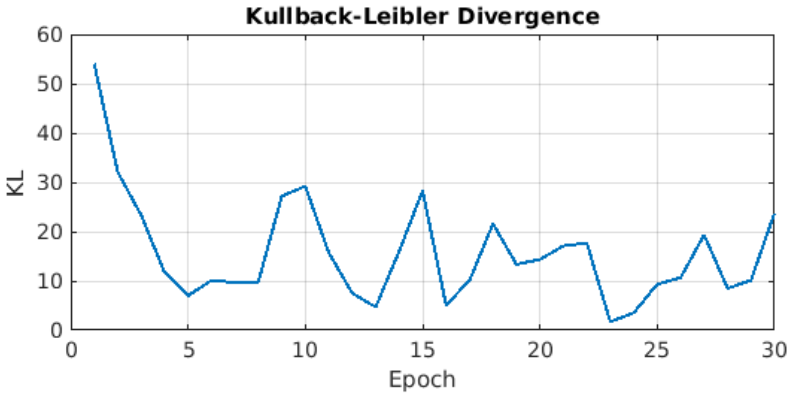

Figure 4.9 – KL divergence between the distributions of IID of generated and genuine images.

In the beginning of the curve in the figure above, a huge decay takes place, which was expected, due to training. The curve indicates that the learning speed is at maximum in that phase. This illustrates the development of producing random images (untrained Generator) to producing shades of plants.
In the range between 4 and 30 epochs, the divergence is oscillating, which is against the expectation. As shown earlier in Figure 4.7, there is a clear improvement in that interval, which was expected to be represented as further decay in Figure 4.9. According to the KL divergence, the IID distribution of images from epoch 4 matches the distribution of IID from genuine images better than epoch 30 does. This is obviously not true from Figure 4.7, which puts a question mark on whether the IID distribution is a representative measure of image distribution similarity.
The best match between Generator images IID distribution and genuine images IID distribution is seen at epoch 23. The KL divergence for this particular epoch is 1.69, and the performance of the Generator is illustrated in Figure 4.10.

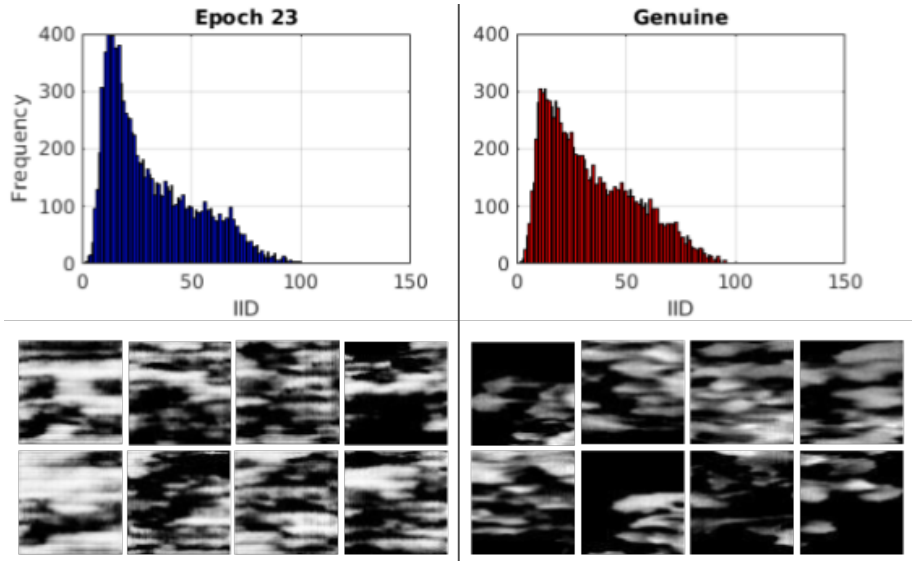

Figure 4.10 – Comparison between the generated images after 23 epochs and the genuine data. In the upper part, Distributions of IIDs of generated and genuine images are shown. In the lower part, generated and genuine images are shown.

The histograms shown in the figure above are clearly more correlated than any of those viewed in Figure 4.8. Here, they do both decay in a similar way. This indicates that KL divergence is a fine measure for distribution distances. Moreover, the images produced by the Generator of epoch 23 look reasonable. They are in many ways similar to the genuine images. However, they do have unclear horizontal lines, which make them look less reasonable than those produced after 30 epochs, shown in Figure 4.7, which again raises the question about the validity of IID as a performance measure.

Up to now, it has been demonstrated that the GAN setup works as expected. The Generator is able to produce images that look genuine, which confirms that the Discriminator, and the training algorithm are working properly. In the next chapter, a variety of GAN, based on these implementations, will be presented.

Chapter 5

Supervised Generative Adversarial Networks

This chapter concerns Supervised Generative Adversarial Networks (sGAN). Initially, the concept of this variety of GAN is explained. Next, the previously built GANs are adapted to it. The Mini sGAN is trained based on artificial images and the results are presented. Afterwards, two Full Size sGAN models are trained: The first one is trained based on the multispectral images and the second one is trained based on the plant fingerprint images. The obtained results are also presented in this chapter.

5.1 Introduction

The GAN setup is not capable of solving the main objective of the thesis. Therefore, it was necessary to introduce Supervised GAN (sGAN). An sGAN is capable of controlling certain attributes in images. If the multispectral images contain information about soil moisture, this variety might be able to learn to control this attribute. Thereby, the trained Discriminator will be able to predict soil moisture and thus complete the objective of the thesis.
The sGAN was proposed by [13], where it was successfully applied to control facial features such as smiling, moustache, sunglasses, and gender. The team was also able to construct unrealistic genuine looking images by combining attributes that, in reality, does not exist, e.g. woman face with mustache.
The sGAN is an interesting and promising way of training Neural Networks with great potential.

5.2 Concept

The sGAN is a variety of the traditional GAN explained in the theory chapter, proposed by [13]. In sGAN, every image, x, has a class vector, c, associated with it. Each element in the class vector represents an attribute related to the image. For face images, the attributes (or class elements) could for instance be: gender, sunglasses, moustache, and hair length.

In order to train an sGAN, every genuine image in the data set must be associated with a vector containing information about all the attributes. Considering the example with face images: a vector must be associated with each image providing information about the gender, whether the person wears sunglasses, has a moustache, and the length of his/her hair.

In a traditional GAN, the Generator produces images based on noise. In sGAN, the Generator produces images based on noise along with a class vector. The goal is to make the Generator produce images that hold attributes according to the class vector. Thus the generated images are associated with the class vector which the Generator was given at the time the image was produced.

The attributes described in the class vectors can be perceived as DOF in the image. The objective of the sGAN is to control those DOF, by training a Generator that is able to produce images that comply with the specified attributes, and train a Discriminator that is able to identify and distinguish them. Figure 5.1 illustrates how sGAN differs from traditional GAN.

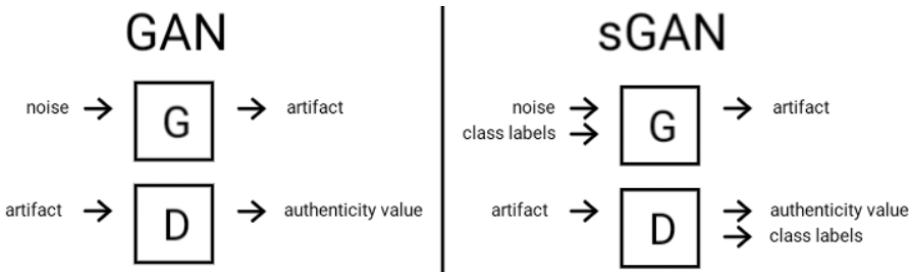

Figure 5.1 – Difference between traditional GAN and sGAN. GAN aims to produce and predict images that look realistic. In addition to that, sGAN aims to control some attributes and predict them as well. *Adapted from [10].*

The values of the class vector can both be discrete (such as sunglasses) or continuous (such as hair length).

The name, sGAN, was given to this variety of GAN by [10]. Therefore, the name is used here, even though it is not informative, as traditional

GAN is also supervised. In the following sections, the name sGAN will continue to be used, in order to distinguish it from the traditional GAN.

5.3 Mini sGAN

Similar to Mini GAN, the Mini sGAN is an ordinary sGAN designed to handle very small images. Its purpose is to speed up the training process while implementing the sGAN, expose errors in an early stage and thereby simplify debugging. Mini sGAN is also a proof of concept.
The architecture and hyperparameters of the Mini sGAN are similar to the Mini GAN explained in the Section 4.3. The only difference is regarding the input layer of the Generator and the output layer of the Discriminator. These two layers are adapted to the sGAN models by expanding the number of nodes with the number of class elements.
The class vector, c, must be normalized to a range that is compatible with the output of the Discriminator. As the last layer of the Discriminator is a Sigmoid layer, the output of the Discriminator is restricted to $(0, 1)$. Therefore, the values in c must be normalized to lie in the same range.
The implementation of the Mini sGAN shuffles the data and splits it up in a similar way as the Mini GAN.
The Mini sGAN was trained on the Samsung laptop, see Appendix A.1.

5.3.1 Genuine data generation

Training the Mini sGAN requires a data set containing images associated with one or more meaningful attribute values. The images in the data set were initially constructed in the exact same way as explained in Section 4.3.2. After producing each image, a set of attributes could be incorporated to into it.
For simplicity, a single attribute indicating the brightness of the center of the image was used. For each image, a random brightness value, b, in the range $(0, 1)$ drawn from a uniform distribution was generated. The brightness was incorporated by multiplying b and a constant scaling factor, s, by each center entry in all the channels. The center of the image is defined as the 4 entries in the center of each 4x4 channel.
The scaling factor is chosen to $s = 3$. The brightness value, b, is the actual class value. A total of 2000 images were produced for training the Mini sGAN. Figure 5.2 shows a few samples of the produced data set.

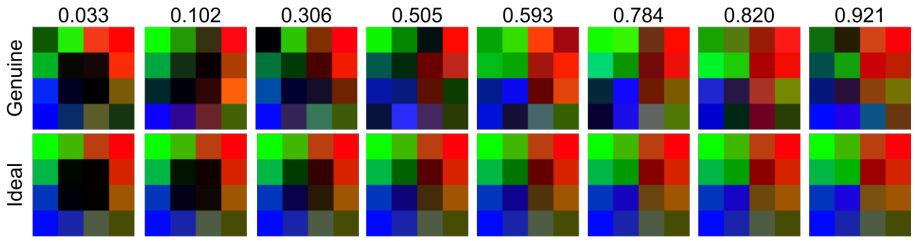

Figure 5.2 – Samples from the data set produced for the Mini sGAN (genuine images), compared to the ideal image, after the same brightness value was incorporated. The ideal image is shown in Figure 4.3. Numbers above each pair of images indicate their brightness value.

The figure above shows a few samples from the produced data set in the upper row. In the lower row, the ideal image is shown, after the same brightness value has been applied. The images are sorted according to their brightness value. It is clear that the brightness of the center is gradually increasing. However, in the genuine images, the brightness is sometimes misrepresented, due to the procedure in which the data was produced before incorporating the brightness value. For instance, the second last genuine image with $b = 0.820$ seems brighter than the last image with $b = 0.921$. However, this is not a problem, as there must be space for this kind of uncertainty. In face images, this could for instance correspond to a male face with female features.
The produced data set will be used to train a Mini sGAN, hoping that it will learn to assess the brightness of the center of similar images.

5.3.2 Results

The Mini sGAN was trained for 30 epochs, using the same algorithms applied for the traditional GAN. After each epoch, a Generator and a Discriminator were obtained. Figure 5.3 illustrates the performance of Generators trained for 1, 5, 20, and 30 epochs based on generated images with gradually increasing brightness values. The generated images are compared to a sample of genuine images from the validation set.

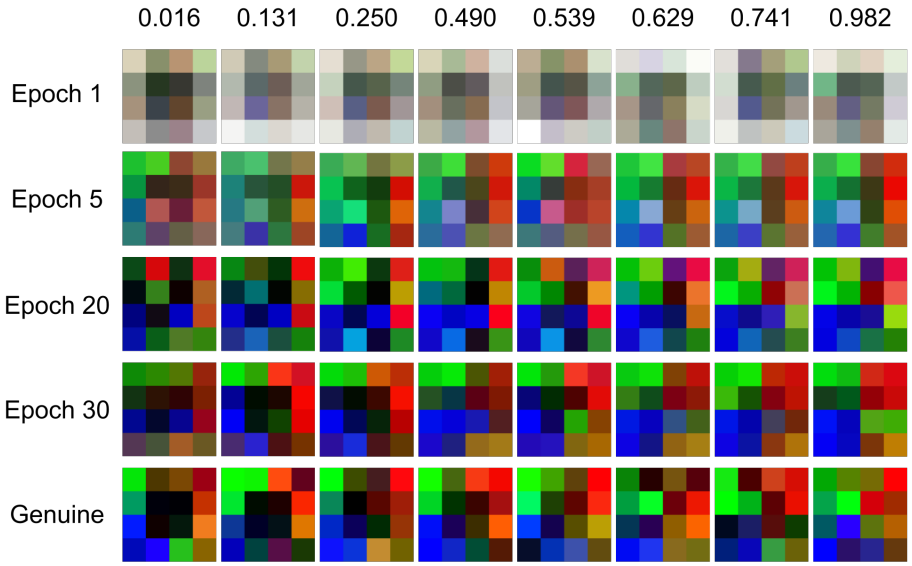

Figure 5.3 – Images generated by the Generator of the Mini sGAN. Each row contains 8 images produced after training the Mini GAN for certain number of epochs. The last row contains genuine data from the validation set. Each column is produced using a specific brightness value, shown above.

It is clear that the images are gradually evolving toward the genuine data. After 1 epoch, the images do not have the shades of the colors in the genuine images. After 5 epochs, the color shades are clear and they are correctly located. However, the blue color is weak in the lower left corner in most images. After 20 epochs, the images become messy, and some black pixels appear. After 30 epochs, the images look more or less similar to the genuine images.

In sGAN, the interesting part is the attributes we are attempting to control, which in our case is the brightness of the four pixels in the center. The images are sorted according to their brightness value in the figure, where the leftmost images have the lowest brightness value, and the other images have a gradually increasing brightness value, as we move to the right. In the genuine images, the brightness of the center pixels is clearly increasing in the range 0 to 0.6. For the last three images, it is not clear which has the lowest brightness value.

After 1 and 5 epochs, the center of the leftmost images is darkest. However, this is most likely coincidence, as the brightness of the center of the other images is not gradually increasing. After 20 epochs, the center of the images on the left half seems darker than the centers of the images on the right half. The difference is very clear when comparing the leftmost image and the right most image. After 30 epochs, the Generator seems to have learned the chosen attribute. The centers of the three leftmost

images are clearly the darkest, while the centers of the three rightmost images are clearly the brightest. However, the two images in the middle do not represent their respective brightness value well, but that is acceptable, as their brightness values are close: 0.490 and 0.539.
Even though the brightness of the centers of the images produced after training the Generator for 30 epochs indicates their brightness value in a reasonable manner, the brightness is not exactly as in the genuine images. It seems like the brightness is damped, meaning that the first image is less dark than the genuine image, and the last image is less bright.
It is tricky to measure the performance of the Discriminator in a GAN, because, according to Figure 2.3, it is supposed not to be able to distinguish between genuine images and those produced by the Generator, when training is finished. However, this does not apply to sGAN, as the performance of the Discriminator can be evaluated based on its ability to predict the class vector.
After training for 1, 4, 16, and 30 epochs, 200 genuine images from the validation set have been fed to the Discriminators. In Figure 5.4, the Discriminators' predictions of the brightness values are illustrated.

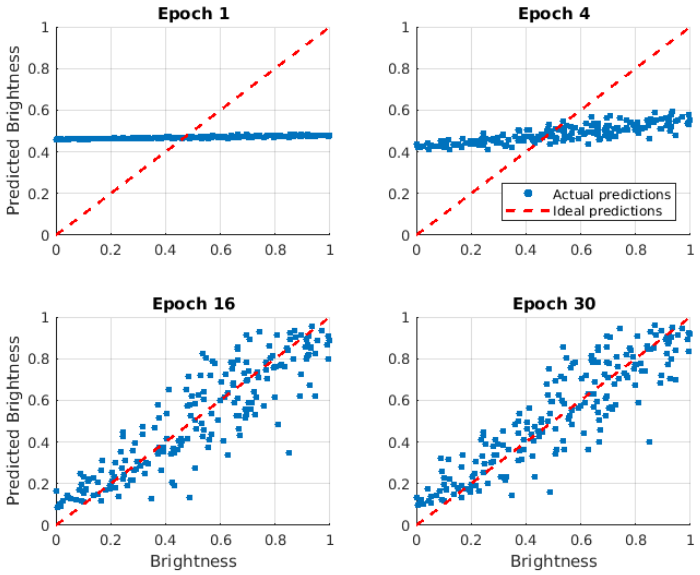

Figure 5.4 – Prediction of the brightness values made by Discriminators trained for 1, 4, 16, and 30 epochs. The red dashed lines are the ideal predictions.

Ideally, the predictions of the brightness values should fit the actual brightness values. The points would thereby be on a diagonal line, illustrated as a dashed red line.

After training for 1 epoch, the predictions are always close to 0.5, as the Discriminator has not learned the prediction yet. All the points fit a horizontal line. After 4 epochs, the predictions seem to improve. The predictions lie between 0.4 and 0.6, where images with low brightness values are predicted around the lower limit, and images with high brightness values are predicted around the upper limit. After 16 epochs, the predictions start following the diagonal line with a significant deviation. The same applies for epoch 30.
In order to evaluate the performance of the Discriminator through all epochs, the MSE has been computed for predictions, similar to those shown in Figure 5.4. The results are shown in Figure 5.5.

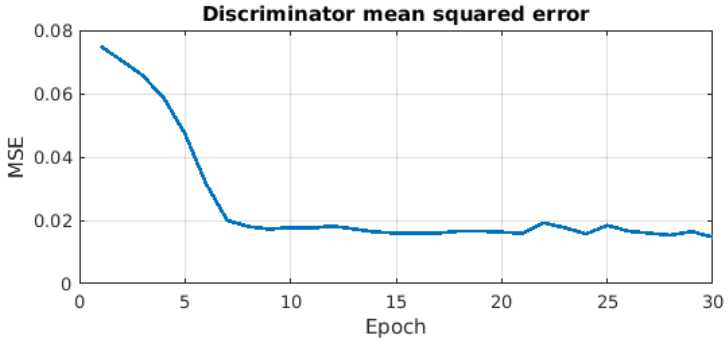

Figure 5.5 – MSE computed based on actual brightness values vs predicted values by Discriminators trained for 1-30 epochs. 200 genuine images from the validation set have been used for these computations.

As expected, the MSE is decaying as the training goes on. However, after 7 epochs, the MSE stabilizes which indicates that the learning is saturated.
The achieved results confirm that the Mini sGAN works as expected. It is possible to control certain attributes of images by controlling an input element of the Generator and training the Discriminator to predict it. The results achieved with the Mini sGAN are satisfying. The real challenge is to make it happen with the real problem.

5.4 Full Size sGAN

The Full Size sGAN is a scaled up Mini sGAN. It is built in a similar way as the Full Size GAN meaning that it consists of the same layers. Its hyperparameters are also the same as the Full Size GAN. The only difference is that the input layer of the Generator and the output layer of the Discriminator are expanded with the number of class elements. In addition, the size of the noise vector is doubled.
Initially, the data set that will be used is the one containing sliced images

of plants. For each image, two pieces of information were given: the number of days it has not been watered and the soil moisture. Both items have been independently linearly normalized to the range $(0, 1)$ and are used as attributes for the Full Size sGAN. However, the main focus will be on predicting the soil moisture.
The severity of the task of the Full Size sGAN is very high, as soil moisture is not predictable by experts based on these images. However, the sGAN might have a chance, as it considers the channels in different way compared to humans.
The implementation of the Full Size sGAN shuffles the data and splits it up in a similar way as the Full Size GAN.
All Full Size sGANs were trained on ABACUS 2.0, see Appendix A.1.

5.4.1 Results based on multispectral images

The Full Size sGAN was trained for 30 epochs, using all channels of the data set images. Figure 5.6 illustrates the results.

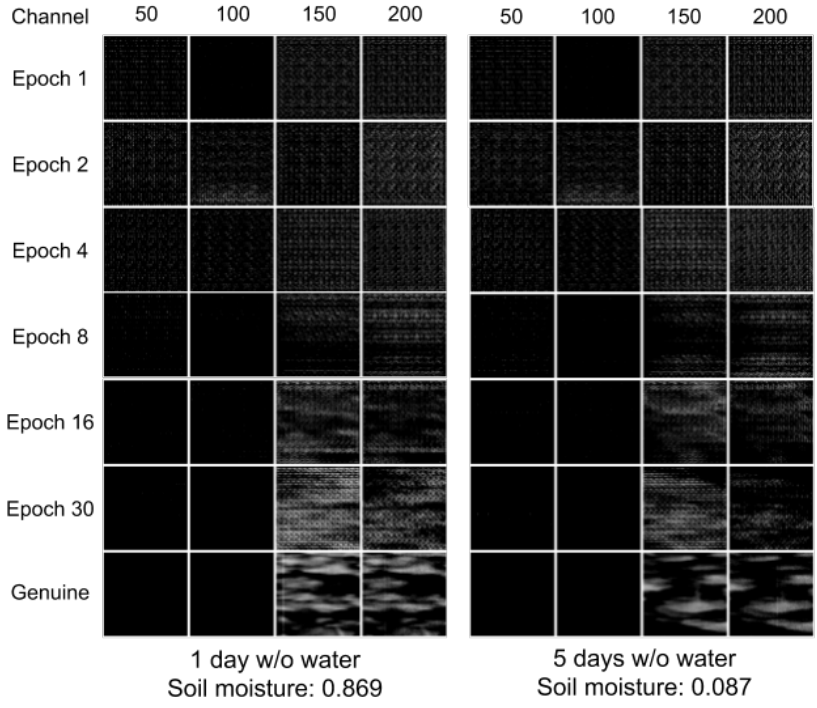

Figure 5.6 – Images produced by the Generator of the sGAN, compared to genuine samples from the validation set. For each of the selected epochs, two images are shown. The images on the left represent recently watered plants with high soil moisture. The images on the right represent water stressed plants with low soil moisture. Each image is represented as four single-channeled images, whose indices are shown above the columns.

The figure above illustrates images produced by the Generator, after

training for 1, 2, 4, 8, 16, and 30 epochs, in addition to genuine images from the validation set. Each multispectral image is represented as four single-channeled images, whose indices are given at the top of the figure. The genuine multispectral image to the left represent a plant that has been watered only 1 day before the image was taken and thus has a high soil moisture of 0.869. The genuine image to the right represent a plant that has not been watered for 5 days, and thus has a low soil moisture of 0.087.

For each of the selected epochs, the Generator was fed with the same class values as those describing the genuine images. The images produced by the Generator are shown above the genuine images in the figure.

All the images produced by Generators trained for 1, 2, and 4 epochs are very noisy and contain uninformative patterns, which is characteristic for untrained Generators. However, after 8 epochs, the noise starts becoming horizontal and more structured. Moreover, channels 50 and 100 start becoming dark, similar to the genuine images. After 16 epochs, channels 50 and 100 are even darker (less noisy), and channels 150 and 200 start taking shape. After 30 epochs, the images are slightly improved, but they still do not look very similar to genuine images. Perhaps more training will improve the performance.

The bad performance of the Generator is unfortunate, but what really matters is the performance of the Discriminator. The Discriminators trained for 1, 4, 16, and 30 epochs were evaluated by feeding them with approximately 200 genuine multispectral images from the validation set. Their predictions for soil moisture are shown in Figure 5.7.

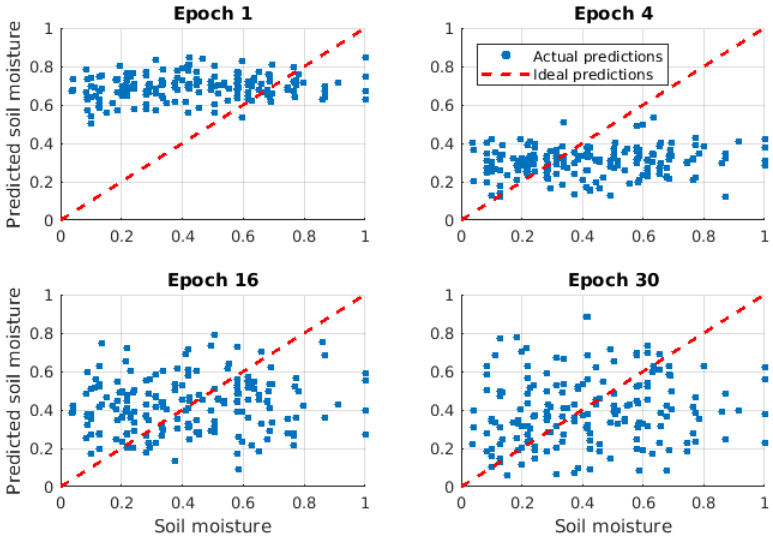

Figure 5.7 – Soil moisture predictions for approximately 200 multispectral images, after training the Discriminators for 1, 4, 16, and 30 epochs. In ideal case, all the points should lie on the red dashed line.

After 1 and 4 training epochs, the soil moisture predictions of the Discriminator tend to fit a horizontal line, which indicates that it has not been trained sufficiently. After 16 and 30 epochs, the predictions are spread all over, which indicates that the training was unsuccessful.
This evaluation procedure was repeated for all epochs and the MSE was computed for every epoch, yielding the results shown in Figure 5.8.

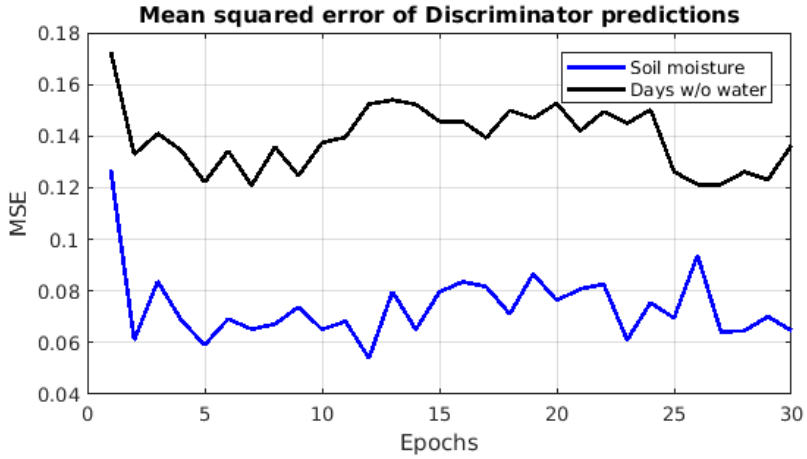

Figure 5.8 – The MSE of approximately 200 predictions made by Discriminators trained for 1-30 epochs.

According to the figure above, the number of days the plant has been without water is predicted worse than the soil moisture.
The best performance for soil moisture prediction was observed after 12 training epochs. The predictions of that Discriminator are illustrated in Figure 5.9.

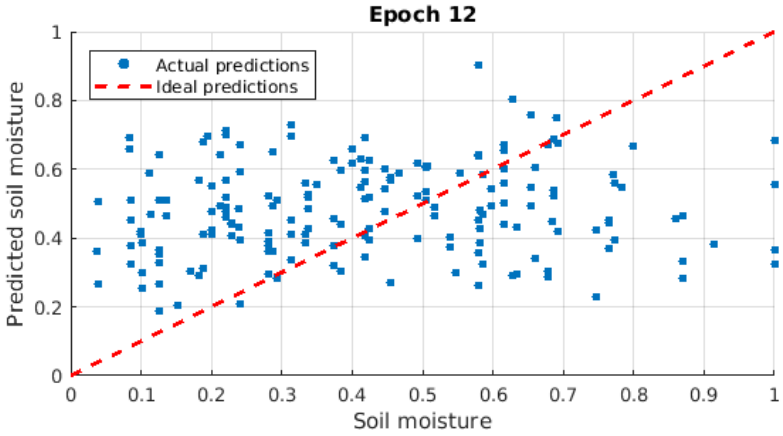

Figure 5.9 – Soil moisture predictions for approximately 200 multispectral images after training the Discriminator for 12 epochs.

Unfortunately, the predictions deviate too much from the red dashed line. The Spearman's Rank Correlation [15] was computed for this point

cloud yielding 0.124. These results indicate one of two things: either the training has failed or it has not been sufficiently complete.
According to the images viewed in Figure 5.6, the performance might be improved by further training the sGAN. This will hopefully improve the quality of the output of the Generator, and thereby provide higher quality images for the Discriminator, which might improve its performance. However, this is too unsure and due to the excessive computational power required, this option is not possible. It is therefore necessary to find an alternative solution.

5.4.2 Results based on image fingerprints

Based on the fingerprint images produced in Section 3.4.1, a Full Size cGAN was trained for 30 epochs. The obtained results are shown in Figure 5.10.

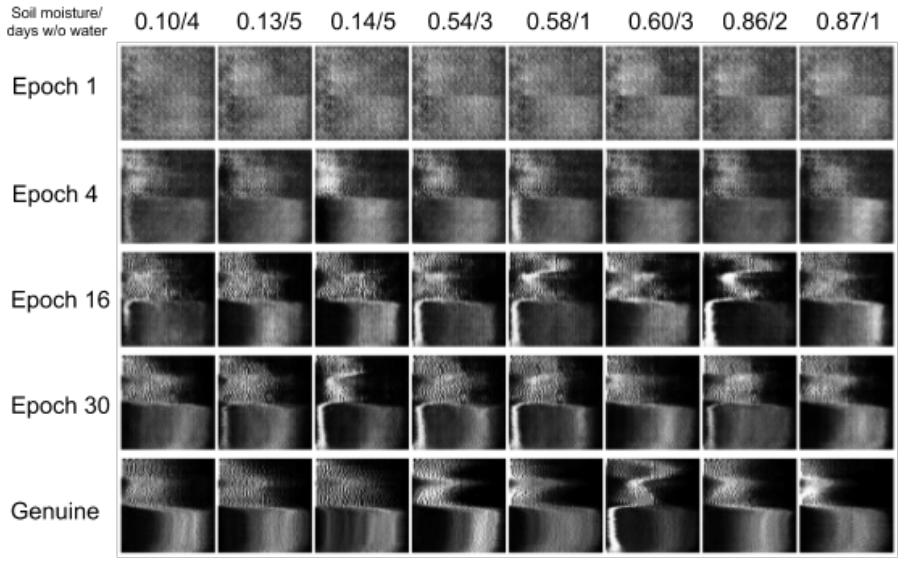

Figure 5.10 – Images produced by the Generator of the sGAN, compared to genuine samples from the validation set. For each of the selected epochs, 8 images are shown. The images are sorted according to the soil moisture of the plant they represent.

In the figure above, images produced by Generators trained for 1, 4, 16, and 30 epochs are shown. For each epoch, 8 images were generated using class vectors of 8 genuine images from the validation set.
After 1 epoch, the images are very noisy but the shade of the genuine fingerprint images is visible. After 4 epochs, there is a significant improvement. However, the noise is still present. After 16 epochs, the images look more like the genuine images, but the noise makes it easy to distinguish between them. After 30 epochs, some of the images are indistinguishable from the genuine images. The noise is further reduced.

As learned from Figure 5.10 and Figure 3.5, there is no obvious way to read the soil moisture based on the fingerprint images. Therefore, it is not possible to assess whether the produced images comply with the class vector or not. However, it seems like the two leftmost and rightmost images at epoch 30 contain the same shape as their corresponding genuine images, which is a positive indicator.
Each Discriminator trained for 1-30 epochs was fed with approximately 200 genuine fingerprint images and the predictions of the Discriminators were evaluated based on MSE. The results are shown in Figure 5.11.

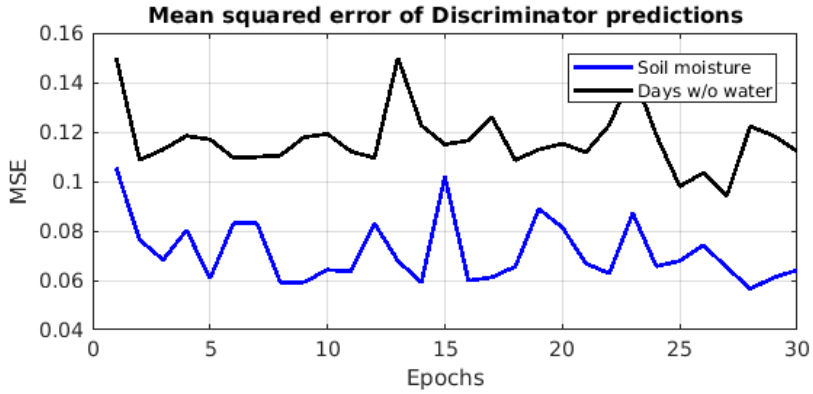

Figure 5.11 – MSE of the predictions of Discriminators trained for 1-30 epochs based on approximately 200 images from the validation set.

According to the figure above, there is a slight improvement after the first epoch for both soil moisture and days w/o water predictions. However, from epoch 2-30, the performance is oscillating. Compared to Figure 5.8, the performance is roughly the same.
The Discriminator trained for 28 epochs is performing best in terms of soil moisture prediction, where $MSE = 0.0566$. Its predictions are illustrated in Figure 5.12.

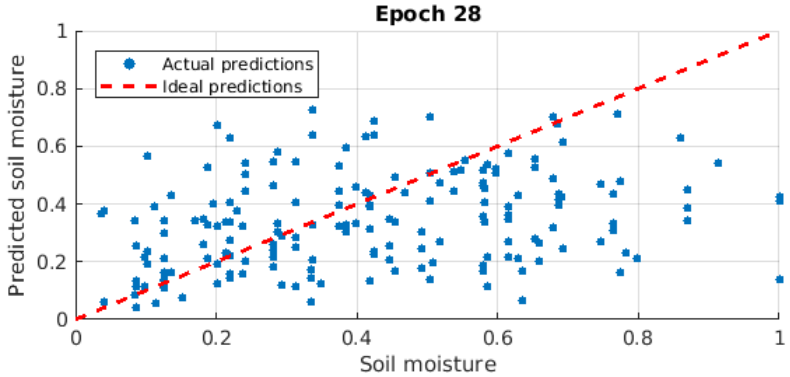

Figure 5.12 – Predictions of soil moisture of approximately 200 genuine images from the validation set made by the Discriminator trained for 28 epochs.

Unfortunately, the predictions deviate too much from the diagonal. The Spearman's Rank Correlation for this point cloud yielded 0.303, which is too low. Again, that means the training has either failed or is not sufficiently complete. These results give reason to further investigate an alternative input for the sGAN.

Chapter 6

Feature Selection

This chapter concerns selection of features for the sGAN. Initially, the Correlation-based and the Transition-based feature selection methods are applied. The selected features were then applied to train and evaluate two separate sGAN models.
The procedure of Sequential Forward Feature Selection algorithm is elaborated and applied based on the features selected by Correlation-based and Transition-based feature selection methods. The obtained features were then used to train and evaluate two new sGAN models.

6.1 Introduction

The data set provided for this project consists of multispectral images. Each image consists of 224 channels. These channels can be viewed as a set of features. In the previous chapter, all channels were included but no good result was obtained. In this chapter, an alternative combination of features will be explored, hoping that it will yield better results.
The objective of Feature Selection can be stated as follows: Given a set of features, F, find a feature subset, S, that maximizes a given criterion. Decreasing the number of features will mainly affect two factors: The discriminative power and computational requirement for both training and discriminating. The discriminative power of the model might be empowered, but might as well be weakened. However, the computational requirement will be reduced for sure. There is a trade-off between these two factors. In our case, the discriminative power is too weak and it is therefore unaffordable to compromise that factor. The objective of Feature Selection is therefore to empower the discriminative ability of the model.

6.2 Correlation-based feature selection

The idea behind Correlation-Based Feature Selection (CBFS) is to find a subset of features, S, that contains least-correlated channels, hoping that they will provide dense information. After finding S, it will be used to train a Full Size sGAN, evaluate it, and hopefully obtain a useful Discriminator.

6.2.1 Method

CBFS selects least correlated features based on a correlation matrix. In order to compute the correlation matrix of an image with m channels, the channels are initially reshaped as vectors. Now, for each vector pair i and j, the cosine of the angle between them is computed as:

$$c_{i,j} = cos(\theta(i,j)) = \frac{\langle i,j \rangle}{||i|| \cdot ||j||} \tag{6.1}$$

where $\theta(i,j)$ is the angle between the vectors i and j, $\langle i,j \rangle$ denotes the scalar product, and $||v||$ denotes the magnitude of the vector v.
The cosine of the angle between two vectors ranges between -1 and 1, where 1 indicates perfect correlation, 0 indicates complete uncorrelation, and -1 indicates perfect opposite correlation.
The correlation matrix of an image n, is given by:

$$C_n = \begin{bmatrix} c_{1,1} & c_{1,2} & ... & c_{1,m} \\ c_{2,1} & c_{2,2} & ... & c_{2,m} \\ ... & ... & ... & ... \\ c_{m,1} & c_{m,2} & ... & c_{m,m} \end{bmatrix} = \begin{bmatrix} 1 & c_{1,2} & ... & c_{1,m} \\ c_{1,2} & 1 & ... & c_{2,m} \\ ... & ... & ... & ... \\ c_{1,m} & c_{2,m} & ... & 1 \end{bmatrix} \tag{6.2}$$

Note that the diagonal is 1, as it denotes correlation between two identical vectors.
The correlation matrix of a data set with N multispectral images is defined as the mean of the correlation matrices of all the images:

$$C = \frac{1}{N} \sum_{n=i}^{N} C_n \tag{6.3}$$

After obtaining the correlation matrix of the data set, C, CBFS can be run as explained in Algorithm 3.

Algorithm 3: Correlation-based feature selection: Given a correlation matrix C of a set of features, F, the algorithm finds a subset of features, S, that are least correlated. n_S is a hyperparameter denoting the desired size of S. n_F is the size of F.

Result: S

Require F, C, and n_S;
$S \leftarrow F$;
$C \leftarrow diag(C) = 0$ (substitute the diagonal of C with 0s);
while *size of* $S \neq n_S$ **do**
  $i, j \leftarrow max(C)$ (max finds the feature pair with highest correlation);
  $c_i \leftarrow \sum_{p=1}^{n_F} C(f_i, f_p)$;
  $c_j \leftarrow \sum_{p=1}^{n_F} C(f_j, f_p)$;
  $f_{exclude} \leftarrow \begin{cases} f_i & \text{if } c_i > c_j \\ f_j & \text{otherwise} \end{cases}$;
  $S \leftarrow S \backslash f_{exclude}$ (exclude $f_{exclude}$ from S);
  $C(f_{exclude}) \leftarrow 0$ (substitute the row and column of $f_{exclude}$ with 0s)
end

6.2.2 Results

In order to select the least-correlated channels, the correlation matrix was initially computed for the full data set. The dimensions of the correlation matrix are 224x224, as each image consists of 224 channels.
The correlation matrix was fed into Algorithm 3 with $n_S = 10$. Figure 6.1 shows the channel indices selected by CBFS, and those channels in a few images from the data set.

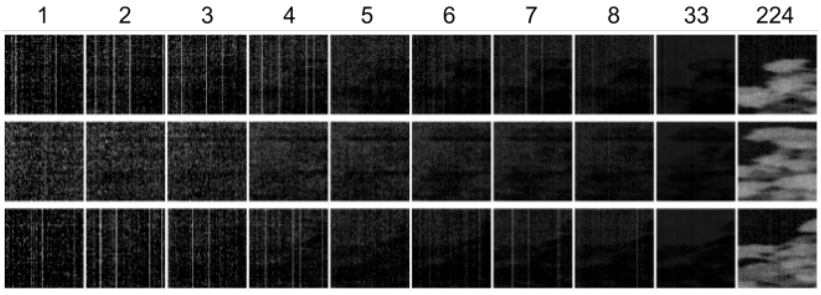

Figure 6.1 – Channels selected by CBFS. Numbers above each column indicate the channel index. First row: 1 day w/o water. Second row: 3 days w/o water. Third row: 5 days w/o water.

Based on the figure above and Figure A.1, it seems like the low indexed

channels are noisy. Apparently, the noise is uncorrelated, which is why these channels are being selected by CBFS. As these channels do not seem promising because they do not include much relevant information, CBFS was be repeated, where the first 10 channels were excluded before running the algorithm.
Selecting the same value for n_S, CBFS selected the channels shown in Figure 6.2.

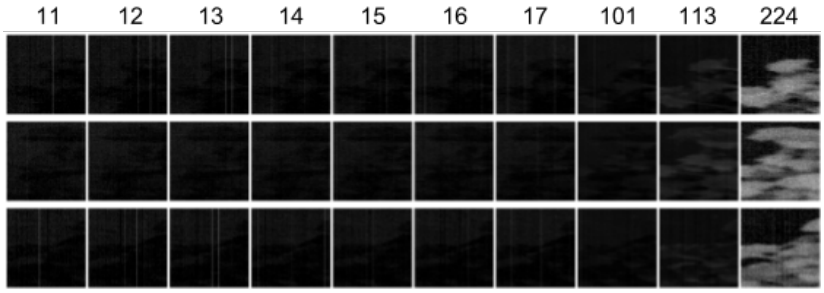

Figure 6.2 – Channels selected by CBFS. First 10 channels were excluded. Numbers above the columns indicate the channel index. First row: 1 day w/o water. Second row: 3 days w/o water. Third row: 5 days w/o water.

Unfortunately, the channels look correlated and many of them seem very similar. The first 7 channels are least correlated, according to CBFS, due to the vertical noise lines in the channels. Channel 101 is a clean version of the the first 7 channels. Channels 113 and 224 represent channels corresponding to wavelengths where the plants are partly and fully reflective.
The channels selected by CBFS were used to train a new sGAN for 30 epochs. The performance of the Discriminator was evaluated by predicting the soil moisture and the number of days w/o water of approximately 200 images. The MSE of these predictions are illustrated in Figure 6.3.

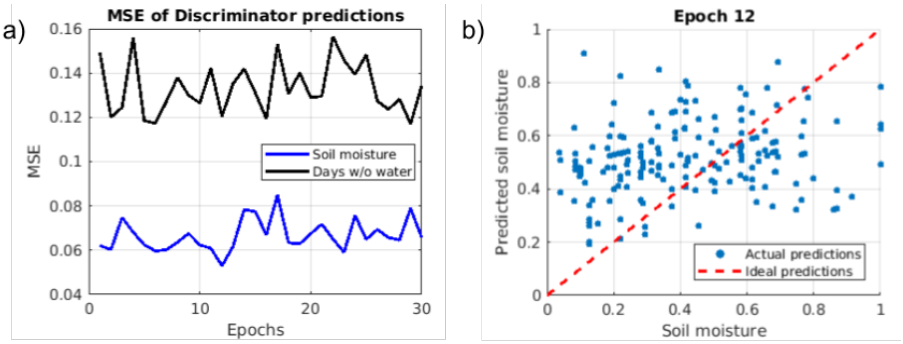

Figure 6.3 – a) MSE of predicted soil moisture and number of days w/o water by Discriminators trained for 1-30 epochs. b) the predictions of the best performing Discriminator in terms of soil moisture, i.e. after 12 epochs. The results were obtained based on approximately 200 images from the validation set.

Comparing the results obtained in the previous chapter with the results shown in Figure 6.3a, the performance is worse here. The MSE start oscillating from the beginning, without decaying initially. This indicates that the Discriminator is not able to learn anything about the specified attributes at all.
In Figure 6.3b, the predictions of the best performing Discriminator, in terms of soil moisture prediction, are presented. As the points deviate too much from the diagonal, we can conclude that the training has failed. The Spearman's Rank Correlation for this point cloud yielded 0.215, which confirms the achieved conclusion.

6.3 Transition-based feature selection

Transition-Based Feature Selection (TBFS) is a feature selection method based on the transition of the spectral reflectance in plants. In contrast to CBFS, it is not an algorithm, but a manual selection of features.
Based on Figure A.1, the main reflection transition takes place at the wavelengths corresponding to channels 111-130. In order to stick to $n_S = 10$, every second channel in that range is selected after. In Figure 6.4, the selected channels are shown for three images.

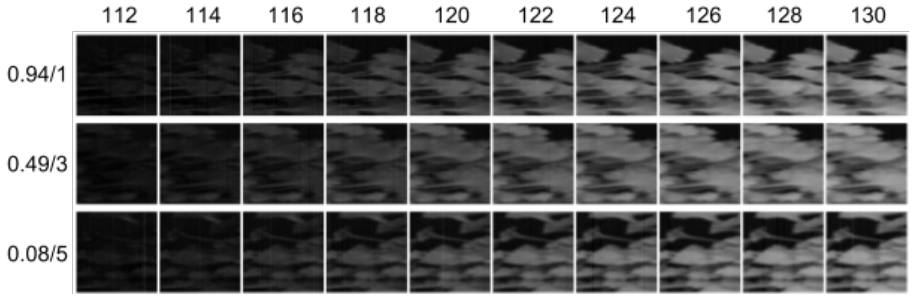

Figure 6.4 – Channels selected by TBFS. Each row represents an image with the attributes given in the format: soil moisture/number of days w/o water. The number above each column specifies the channel index.

TBFS is inspired by the idea that water-stressed plants reflect light in different wavelengths compared to non-stressed plants. By focusing the channels on the spectral reflectance transition, we might be able to obtain useful results.

6.3.1 Results

The channels selected by TBFS were used to train a new sGAN for 30 epochs. The performance of the Discriminator was evaluated by predicting the soil moisture and the number of days w/o water for approximately

200 plant images. The MSE of these predictions are illustrated in Figure 6.5.

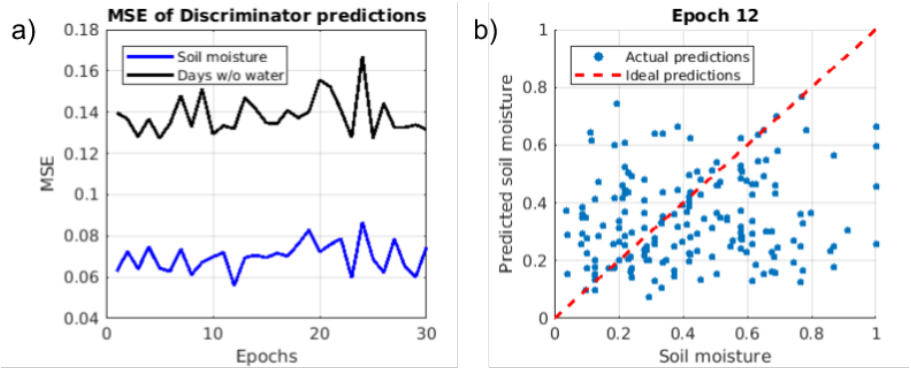

Figure 6.5 – a) MSE of predicted soil moisture and number of days w/o water by Discriminators trained for 1-30 epochs. b) The predictions of the best performing Discriminator in terms of soil moisture, i.e. after 12 epochs. The results were obtained based on approximately 200 images from the validation set.

The results obtained by TBFS are in several ways similar to those obtained by CBFS. The MSE of both methods oscillates from the beginning and do not decay initially. Moreover, both results are not promising and the best performing Discriminator was obtained after 12 epochs.
In Figure 6.5b, the predictions made by Discriminator trained for 12 epochs is shown. Unfortunately, the predictions deviate too much from the diagonal, which also indicates that the training has failed. The Spearman's Rank Correlation for this point cloud yielded 0.142, which also supports the achieved conclusion.

6.4 Sequential Forward Feature Selection

Sequential Forward Feature Selection (SFFS) is an algorithm that selects a subset, S, of a set of features, F, that optimizes the performance according to a given criteria [16]. It is a greedy algorithm that explores a subset of the search space and picks what apparently seems best. This makes it suboptimal, i.e. it does not guarantee to find the optimal result. Initially, SFFS constructs and evaluates sGAN models based on every feature in F individually. The feature that yields the best performance is added to S. Next, sGAN models are constructed and evaluated based on the remaining features combined individually with S. The feature that yields the best performance is again added to S. This procedure is repeated until the performance no longer improves or until $S = F$. See

Algorithm 4.

Algorithm 4: Sequential Forward Feature Selection: Finds a subset, S, of F that optimizes the performance.

Result: S

Require: F;
$S \leftarrow \varnothing$;
$c_{best} \leftarrow \infty$;
for F **do**
    $R \leftarrow \varnothing$;
    for f *in* F **do**
        if $f \notin S$ **then**
            $sgan \leftarrow sGAN(S \cup f)$;
            $c \leftarrow C(sgan)$ (C returns the criterion to minimize);
            $R \leftarrow R \cup [c, f]$;
        end
    end
    $c, f \leftarrow min_c(R)$ (min_c returns minimum c and its associated f);
    if $c < c_{best}$ **then**
        $c_{best} \leftarrow c$;
        $S \leftarrow S \cup f$;
    else
        done;
    end
end

The validation set is split up in two parts of equal size. All the Discriminators of an sGAN are evaluated by computing the MSE of predictions based on the first part. Once the best Discriminator is found, the criterion to minimize is defined as the MSE of the predictions of the best Discriminator based on the second part of the validation set.
In order to save time and computational power, the sGAN models constructed by SFFS were only trained for 15 epochs. According to the evaluation figures shown previously (such as Figure 6.3), there is not much information to gain by training an sGAN for 30 epochs compared to 15 epochs.

6.4.1 Results based on CBFS

Defining F as the channels selected by CBFS, shown in Figure 6.2, the SFFS suggested channel 11 alone. Thus an sGAN based on that feature was trained for 30 epochs. The results are shown in Figure 6.6.

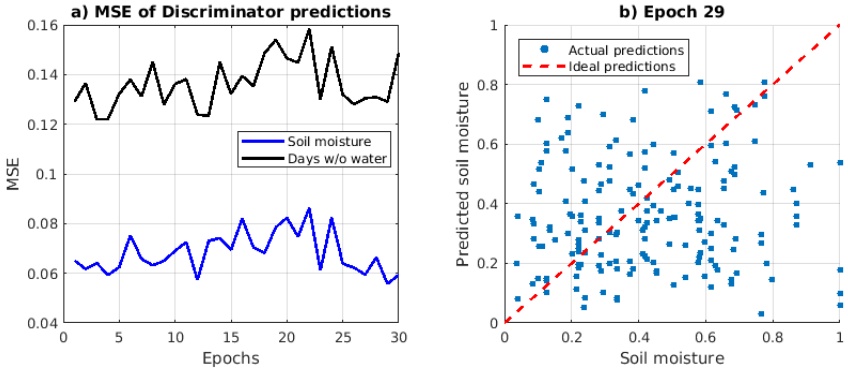

Figure 6.6 – a) MSE of predicted soil moisture and number of days w/o water by Discriminators trained for 1-30 epochs. b) predictions of the best performing Discriminator in terms of soil moisture, i.e. after 29 epochs. The results were obtained based on approximately 200 images from the validation set.

Unfortunately, according to Figure 6.6a, there is no significant learning, when only channel 11 is used. The MSE is oscillating and not even decaying in the beginning. The best performance is obtained after 29 epochs. It is only slightly better than the other training epochs. The predictions made by the Discriminator trained for 29 epochs are shown in Figure 6.6b. The predictions deviate too much from the red dashed line, indicating that the training has failed. The Spearman's Rank Correlation for this point cloud yielded the lowest value so far, 0.066, which confirms the failure of the training.

6.4.2 Results based on TBFS

Defining F as the channels selected by TBFS, shown in Figure 6.4, the SFFS suggested channel 130 alone. An sGAN based on that feature was trained for 30 epochs. The results are shown in Figure 6.7.

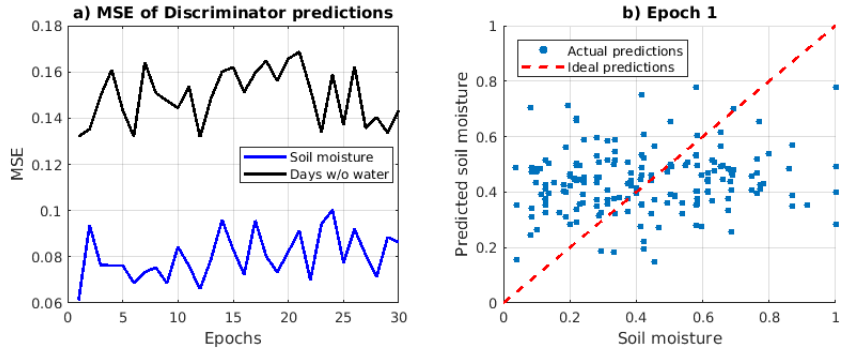

Figure 6.7 – a) MSE of predicted soil moisture and number of days w/o water by Discriminators trained for 1-30 epochs. b) the predictions of the best performing Discriminator in terms of soil moisture, i.e. after 1 epoch. The results were obtained based on approximately 200 images from the validation set.

Unfortunately, according to Figure 6.7a the training was unsuccessful. The MSE is oscillating without tending to decay. On the contrary, the MSE of the soil moisture prediction seem to increase after the first epoch. The first epoch had the best performance, which indicates that the training is meaningless, as the first epoch is highly affected by random factors. In Figure 6.7b, the predictions of the Discriminator trained for 1 epoch is shown. The predictions deviate significantly from the red dashed line, which confirms that the training has failed. The Spearman's Rank Correlation for this point cloud yielded 0.114, which unfortunately supports the achieved conclusion.

Chapter 7

Conclusion

In this chapter, the overall conclusion is presented and future work is suggested.

7.1 Conclusion

In this project, I have successfully implemented a Mini GAN that is able to handle images of the dimensions 4x4xC. The dimension C is an arbitrary number of channels specified before the GAN is constructed. The Mini GAN has been trained using Adam optimization algorithm. Furthermore, it has proven its ability to generate images similar to those in the data set. It has been evaluated visually, based on its deviation from an ideal image compared to the deviation of the training set from the ideal image. Results showed that the training was successful.
The Mini GAN has been scaled up to a Full Size GAN, that is able to handle images of the dimensions 64x64xC, which suits the multispectral genuine data images of plants after preprocessing. The Full Size GAN has been trained using the same optimization algorithm and based on a single selected channel of the multispectral images. Based on visual evaluation, the Generator proved its ability to generate images close to the genuine ones. The Full Size GAN was also evaluated by comparing the distributions of the internal image distances between genuine images mutually and between generated images mutually. Furthermore, the Kullback-Leibler divergence of the distributions was computed, which also indicated that the learning was successful.
The Mini GAN was modified to become a Mini sGAN, that is able to control and predict certain attributes in images. It was trained to learn the brightness of the center of genuine data using Adam optimization algorithm. After training, it was able to generate images with a center

brightness the comply an input parameter. Furthermore, given new genuine images, the Mini sGAN was able to predict the brightness values. It was evaluated based on the MSE of such predictions. A clear decay in the error indicated that the learning was succesful.
The Mini sGAN was scaled up to a Full Size sGAN, that is able to handle the data images of plants. Initially, all the channels of the multispectral data images were used for training. However, the training was unsuccessful. Based on visual evaluation, the training seems incomplete. Based on the MSE graph of the predictions, the learning seems to have failed, as the graph is oscillating and not decaying significantly.
Another Full Size sGAN was trained using data fingerprints. The Full Size sGAN has successfully learned to imitate these fingerprints, but has failed to use them for predicting water stress.
An algorithm has been proposed to select the 10 least correlated channels. These channels were used to train a Full Size sGAN. However, the training was also unsuccessful, as the Discriminator was unable to predict water stress in a satisfying manner.
Another Full Size sGAN has been trained based on 10 manually selected channels that represent the transition of the spectral reflectance of the plants. Unfortunately, that did not produce useful results either.
Sequential Forward Feature Selection (SFFS) was implemented and applied using the channels selected by the correlation-based and the transition-based approaches. For each set of channels selected by SFFS, a Full Size sGAN was trained. Unfortunately, none of them succeeded.
The repetitive failure of training indicates that the information we are looking for does not exist in the images. Experts do not evaluate water stress visually, and it might not be possible using the multispectral images in the range 400-1000 nm. Alternatively, the plants might not be seriously water-stressed.

7.2 Future work

The next intuitive step is to run the SFFS based on all channels. This step was not included in this project, due to the excessive computational power and time required.
The applied methods have not been able to predict soil moisture based on multispectral images captured in the wavelength range 400-1000 nm. Investigating the actual spectral reflectance behaviour of the Campanula plant would significantly benefit future work in this field. It is very likely, that there is a wavelength range, that clearly highlights the level of water stress in the plant. The range might also be beyond 400-1000

nm.

In case new data is going to be collected, it is important to include the gardener experts while capturing the data, in order to get their assessment on the plants. It is important that every plant is classified into one of the four categories presented in the beginning: Healthy plant, water-stressed recoverable plant, water-stressed unrecoverable plant, and dead plant.

As mentioned in early chapter, experts evaluate plants based on their size and weight. Information regarding these two factors might enhance the discriminative power of the sGAN. The sGAN could be extended to also operate with an information vector, which is an input vector for both the Generator and the Discriminator. In this way, the Discriminator will have an additional clue for predicting the water stress.

A total of 100 multispectral images of Campanula plants were provided. After cropping and slicing the images, 1,785 images were obtained. In order to ensure saturated training, it would be more optimal to increase the size of the data set. Moreover, the training process will be improved if the data set contained images of water-stressed plants in all phases, i.e. healthy plants, water-stressed recoverable plant, water-stressed unrecoverable plant, and dead plants.

Chapter 8

Bibliography

[1] Gartneriet PKM, *"About PKM"*, 2015.
Link: `pkm.dk/en/about-pkm`

[2] Machine Learning Mastery, Jason Brownlee, *"Gradient Descent For Machine Learning"*, 2016.
Link: `machinelearningmastery.com/gradient-descent`

[3] Machine Learning Mastery, Jason Brownlee, *"A Gentle Introduction to Mini Mini-Batch Gradient Descent and How to Configure Batch Size"*, 2017.
Link: `machinelearningmastery.com/gentle-introduction-mini`

[4] Ujjwal Karn, *"An Intuitive Explanation of Convolutional Neural Networks"*, 2016.

[5] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, *"ImageNet Classification with Deep Convolutional Neural Networks"*, 2012.

[6] Yiwei Jiang, Huifen Liu, Van Cline, *"Correlations of Leaf Relative Water Content, Canopy Temperature, and Spectral Reflectance in Perennial Ryegrass Under Water Deficit Conditions"*, 2009.

[7] Deep Learning for Java, *"GAN: A Beginner's Guide to Generative Adversarial Networks"*, 2017.
Link: `deeplearning4j.org/generative-adversarial-network`

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, *"Generative Adversarial Nets"*, 2016.

[9] Diederik P. Kingma, Jimmy Lei Ba, *"Adam: A Method for Stochastic Optimization"*, 2015.

[10] Christoffer Stougaard Pedersen, Martin Rønning Bech, Mikkel Larsen, *"Capabilities of Generative Adversarial Networks"*, 2016.

[11] Alec Radford, Luke Metz, Soumith Chintala, *"Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"*, 2016.

[12] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *"Numerical recipes - The Art of Scientific Computing"*, Third Edition, 2007.

[13] Martin Rønning Bech, Mikkel Larsen, Christoffer Stougaard Pedersen, Joel Lehman, and Sebastian Risi, *"Supervised Generative Adversarial Networks"*, 2016.

[14] Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu, *"How to Train a GAN? Tips and tricks to make GANs work"*, 2016.

[15] D. J. Best and D. E. Roberts, *"Algorithm AS 89: The Upper Tail Probabilities of Spearman's Rho"*, 1975.

[16] Machine learnining extensions, mlxtend, Sebastian Raschka, *"Sequential Feature Selector"*, 2018.
Link: `rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector`

Appendix A

Appendix

A.1 Hardware

The hardware used in this project are a Samsung NP355V5C-S08SE laptop and the supercomputer ABACUS 2.0.
ABACUS 2.0 has 14016 processor cores and a theoretical performance of 766 teraflop/s. Further information about it can be found on: deic.dk/Abacus
The Samsung NP355V5C-S08SE has the following specifications:

Table A.1 – Specifications for Samsung NP355V5C-S08SE laptop

Processor	AMD A8-4500M APU w. Radeon(tm) HD Graphics
Number of cores	4
Memory	8 GB
Operating System	Ubuntu 16.04.4 LTS

A.2 All channels of a sliced multispectral image

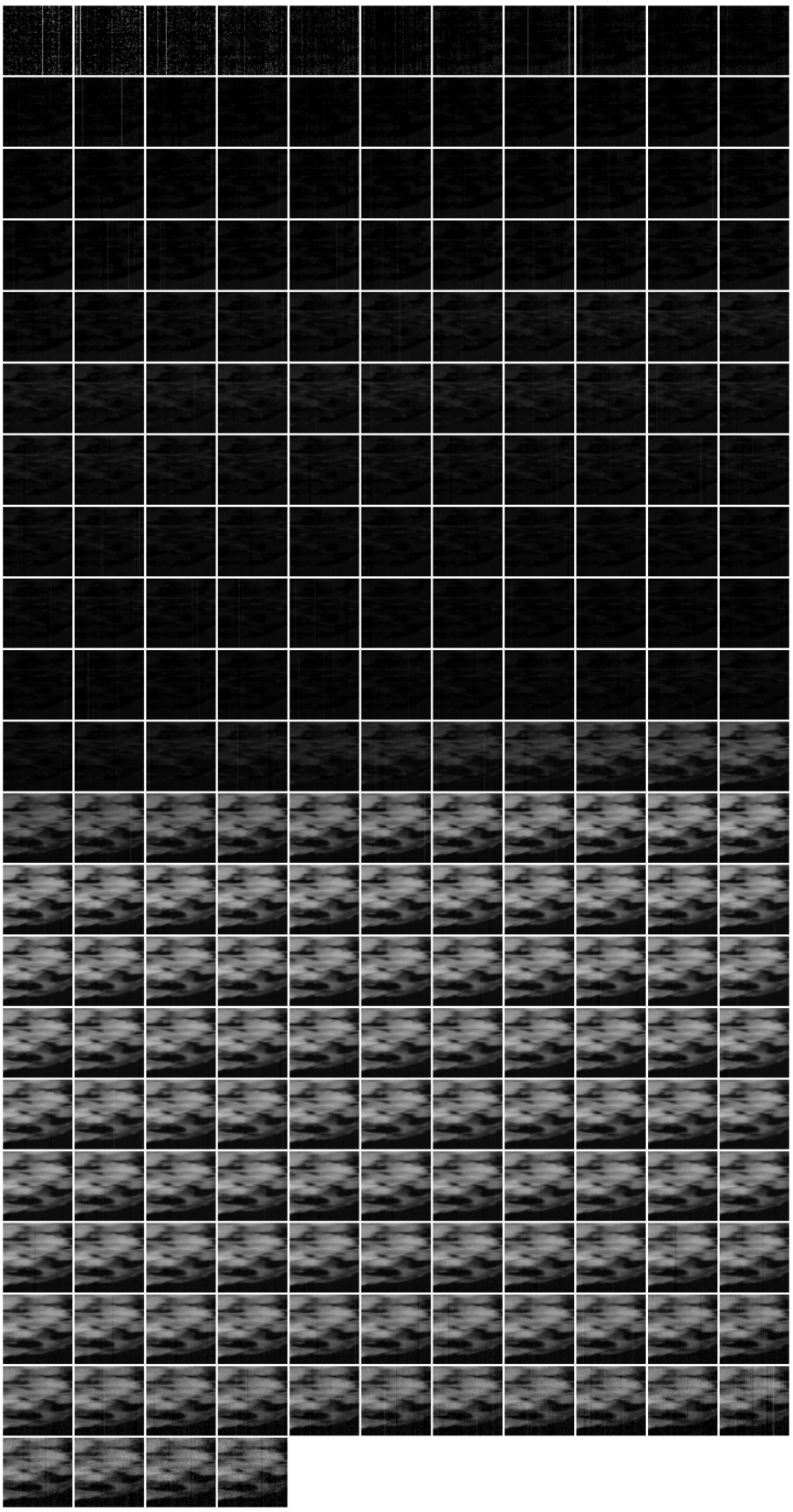

Figure A.1 – All channels of an image slice.