SAPIENZA
UNIVERSITÀ DI ROMA

Dataset Augmentation with Generative Adversarial Network for Crop/Weed Segmentation for Precision Farming

Faculty of Information Engineering, Informatics, and Statistics

Master's in Artificial Intelligence and Robotics

Candidate

Ibis Prevedello

Thesis Advisor

Prof. Daniele Nardi

Co-Advisor

Ciro Potena

Academic Year 2018/2019

Dataset Augmentation with Generative Adversarial Network for Crop/Weed Segmentation for Precision Farming
Master's thesis. Sapienza – University of Rome

Version: October 15, 2019

Author's email: ibiscp@gmail.com

To my father for giving me support to follow my dreams.

Abstract

The world is advancing towards automation of every task, including agriculture, such as selective weeding, and this will have a huge impact on the farm productivity on profitability. The most powerful tools to let autonomous robots to properly perceive the environment are neural networks. Unfortunately, such methods are data driven, meaning that they rely on a large labeled datasets, and the collection and preprocess of this data is expansive, time consuming, and depend heavily on manual labeling.

This project proposes a pipeline for image dataset augmentation generating synthetic data using generative adversarial networks. It starts generating the mask of the crop from random noise, and from the mask generates the final RGB and NIR images. For the mask generation it uses DCGAN network and for the final images uses SPADE network. The images are generated in a two step process in order to achieve the RGB, NIR, and semantic segmentation of each plant. One possibility would be generating the RGB and NIR images directly from random noise, but depending on the project that this dataset will be used, the semantic segmentation may also be needed.

The process is validated using metrics that compare the quality of the generated images with the original dataset and also a semantic segmentation model that evaluates the difference between using the original dataset compared with the synthetic and augmented dataset. The metrics show that the values of the synthetic images converge to the values of the original ones as the quality improves over the training and also that the segmentation performance can indeed be improved by using the new augmented dataset.

Acknowledgments

First of all, I wish to thank my family for all the support during my life and my career, without them I would not be able to achieve my goals in life and be the person that I am today, they did what they could to give me opportunities and freedom to decide what I thought would be better for my future and I am happy and glad for showing them that their effort has been worth it. Also my girlfriend, Pamela Biazon, for being by my side and supporting me during this time.

A special thanks to Ciro Potena for sharing his knowledge, advising me and discussing ideas of the work presented here, his help was essential to the results achieved. Also for everyone in the lab, mainly Mulham Fawakherji and Carlos Carbone for revising my thesis.

Thanks to Professor Daniele Nardi, for giving me this great opportunity to perform this project with him and for accepting me for the master at "La Sapienza" University of Rome. Professor Nardi was the first contact that I had with this university and it is good to be finishing my journey here also with him.

A big thanks, not to the university itself, but to the environment that the institution represents, to the good professors that I had during this two years in Italy and all the friends that I had the opportunity to make. All these knowledge and friendships are now part of what I am and I hope to be able to wisely represent you all in my future life.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Presentation

Recently, the interest in robots for precision agriculture has been growing. The use and study of robots in this field is increasing because it presents a series of advantages when compared to the traditional methods, not only in terms of harvesting and planting, but mainly in weed and pests control.

In case of weed removal, for example, the traditional method mainly depends on manual weed eradication or on using big and expensive agricultural machinery to apply herbicides on the whole field. Therefore the herbicide is uniformly spread all over the field, even though not all the parts in the field require the same amount of chemicals. This might lead to a higher cost.

The idea of precision agriculture is to employ the latest technologies to automatize the work in the field, reducing the application of chemical treatments. For example, instead of applying herbicides in the whole field, it would be possible for an autonomous robot to go through the field, taking pictures of every single plant, recognize which plants are not desired, in this case the weeds, and kill them individually. By developing this technology, it could be applied for different crops, also for pests, water and fertilize the whole field, but only where it is needed. In this way, resources could be used more efficiently, improving the quality of the crop, increasing production and making not only the cultivation, but also all the derivative products cheaper.

In order to develop this kind of new technology for autonomous agricultural vehicles some tools are needed, and the most powerful tools present today are neural networks. However, such methods are data driven, meaning that they rely on a large labeled datasets to understand the sensory data from the field, for example, classifying and segmenting crops, weeds and soil. The key point is that, besides the need of the robot with specific tools to collect all the data, humans are also needed to create this segmentation map of each image, selecting what is every object in the image at pixel level. This process, besides being manual, demands a lot of time and money, and depending on the project, can consume a significant amount of budget.

The goal of this project is to develop a method able to learn how to generate completely new random crop images similar to the ones provided by the agricultural robot dataset [1]. Such method needs to be able to learn how to generate new images that pass as real pictures when a human looks at and also improves accuracy when used for training other algorithms, in this case a semantic segmentation network.

By using the proposed data generation pipeline described in this thesis, it becomes possible to reduce the effort in gathering and annotating datasets on the field. Moreover, the proposed method have several other applications, like: creating simulators, training classification and segmentation networks, as well any other uses that requires a large image dataset.

1.2 Proposed Solution

In this project we use a dataset with limited samples of Sugar Beets plants collected in a real farm field. After preprocessing the dataset, it is possible to extract three different images, shown in Figure 1.1, a plant mask (semantic map), a RGB and a NIR image, which will be explained in Section 2.2. Both the images are taken by the camera, while the plant mask is taken by an additional manual work.

The focus of this work will firstly be the preprocess of the full dataset in order to extract only the three types of images as shown in the Figure 1.1. After that, the idea is to train Generative Artificial Networks in order to learn how to randomly generate these kinds of images, that means images that are not present in the original dataset, but are realistic enough both for humans as for training learning algorithms. In this way, we are able to augment the dataset with an arbitrary large number of new images and the user will have no further need to manually annotate a large number of real images.

The work here presented was created and tested with the goal to use it for precision agriculture, however its use is beyond this context. In fact, the inspiration for this work was the paper 'ARIGAN: Synthetic Arabidopsis Plants using Generative Adversarial Network' [2] and most recently 'Synthetic Medical Images from Dual Generative Adversarial Networks' [3], where the training was divided in two stages, the first one for the mask generation, and the second one for the RGB image generation, having the mask as an input of the network. Our implementation, besides changing the networks used, is also different because there is no interest in only generating the RGB image, but also the NIR in the same training. It means that the algorithm presented here can be extended to other datasets for other applications, while keeping the general idea of first generating a mask and from the mask generating the final desired image.

The network used for the first stage in order to train the masks is a DCGAN

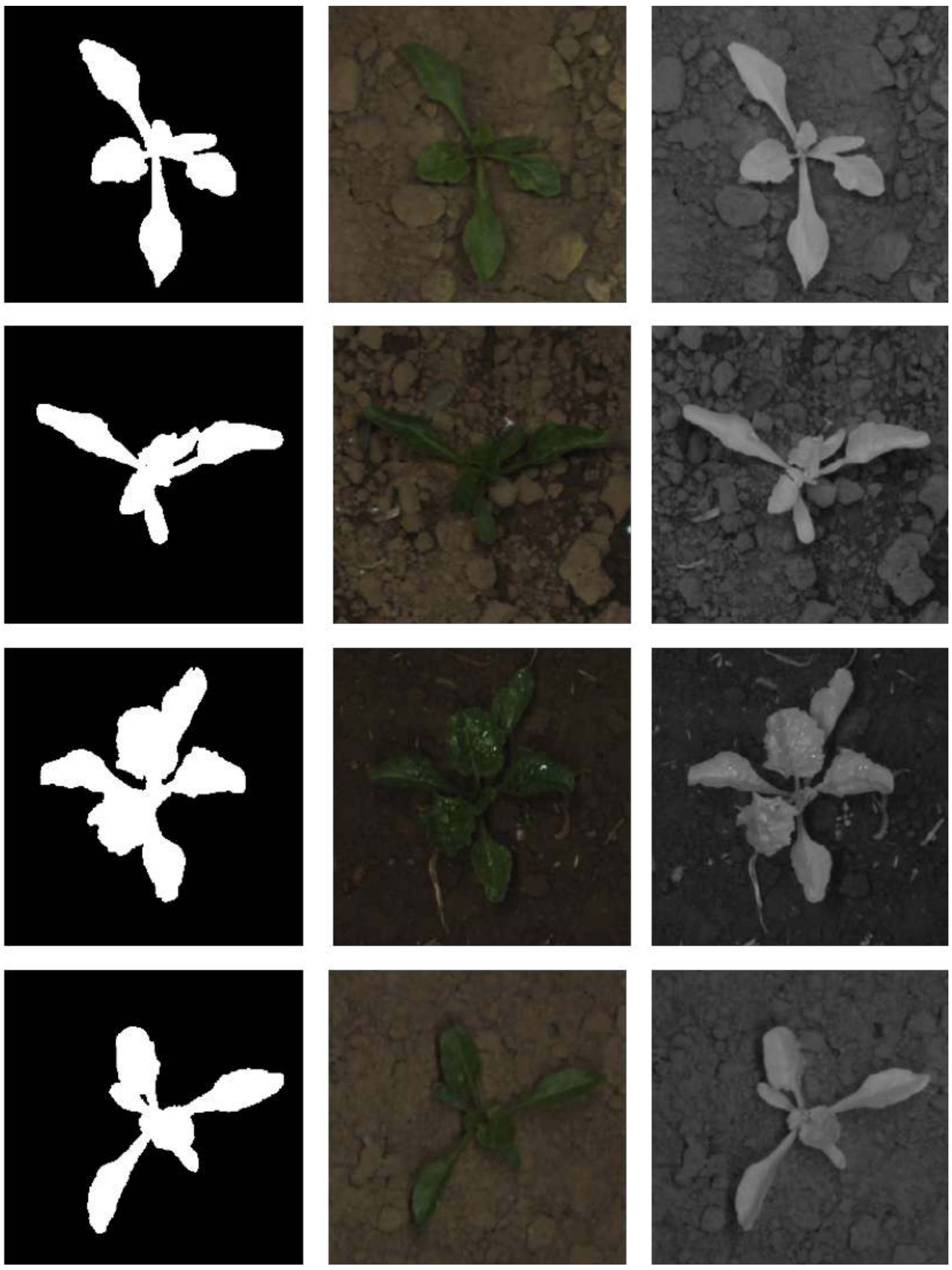

Figure 1.1. Dataset image examples.

network, based on the paper ’Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks’ [4] and for the RGB and NIR images is based on the paper ’Semantic Image Synthesis with Spatially-Adaptive Normalization’ [5]. Here will be shown that these two networks together can provide images with realistic enough quality for dataset augmentation.

Usually this kind of dataset augmentation is done by generating directly the desired image, in this case it would be a one step process, generating the RGB image directly from random noise, without having to generate the mask in the first stage. We chose to make the generation in a two step process, because it gives more freedom for future use of this dataset. For example, assuming that only the RGB images are available in a synthetic dataset and it is necessary to train a segmentation network, the images first need to be manually segmented. However, if the semantic segmentation images are also available this work is unnecessary.

The networks generated in this project will be evaluated using metrics that will analyse how the synthetic images being generated are compared to real images. And, at the end, also test how a segmentation training will performed in three different scenarios: using only the real dataset, a real dataset augmented with synthetic images, and a dataset composed of only synthetic images.

Chapter 2

Dataset

The first issue to address when developing and training any model is the dataset that will be used to feed the network. This chapter describes not just what is the dataset used in this thesis, but also how it was collected and gives some examples and an overview of the processing necessary to have the final cleaned dataset that will be used to feed all the models created.

2.1 Robot for data collection

The robot used for collecting the dataset can be viewed in Figure 2.1, it is called BoniRob platform and it is a multi-purpose robot by Bosch DeepField Robotic [6]. BoniRob is developed for precision agriculture applications, for example, mechanical weed control, selective herbicide spraying, as well as for plant and soil monitoring. It provides structure for installing different tools for these specific tasks, BoniRob is equipped with four wheels which can be steered independently of each other, which allows for flexible movements and navigation on rough terrains.

Figure 2.1 illustrates the locations of all sensors mounted on the BoniRob, an overview of the robot and some data collected by its sensors. They deliver visual, depth, 3D laser, GPS and odometry data.

Specifically, in this configuration, the robot carries the following equipment:

- JAI AD-130GE camera that collects RGB and NIR images.
- Kinect One that provides RGB and depth information of the scene.
- Velodyne VLP16 Puck, which is a 3D lidar sensor that provides distance and reflectance measurements obtained by a rotating column of 16 laser diodes.
- Nippon Signal FX8, a 3D laser range sensor that provides distance measurements up to a maximum range of 15 meters.

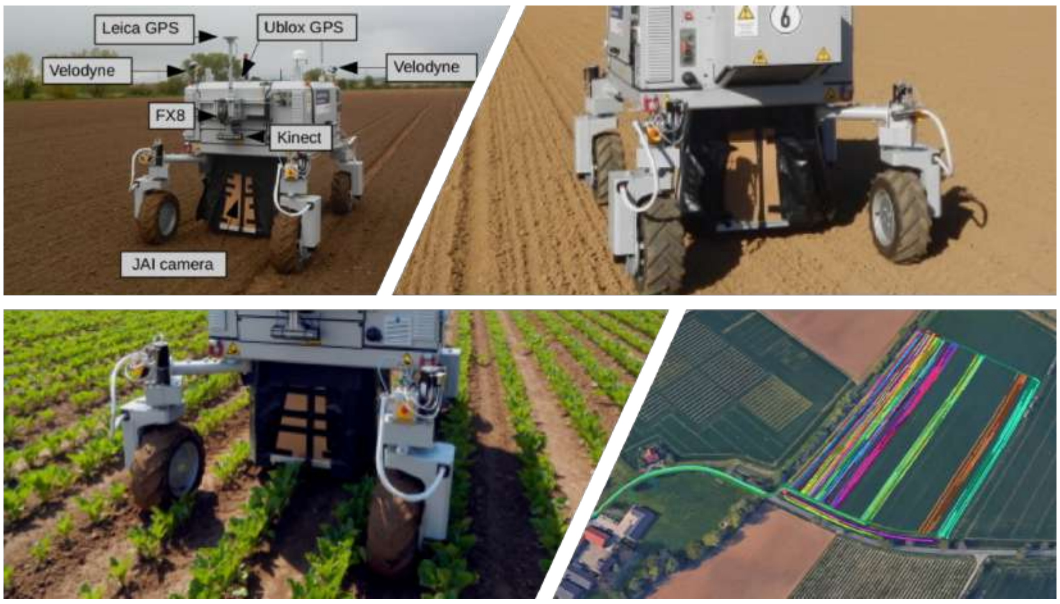

Figure 2.1. Robot used for data collection.

- Leica RTK GPS, which provides accurate position estimates tracking the signal of the satellites and additionally obtaining observations from a nearby base station with known location.
- Ublox GPS, which is a low-cost GPS and needs only one receiver.

2.2 Camera

The most known camera for normal uses are RGB cameras; this kind of cameras record the red, green and blue bands of light, which are sensitive to the human eye. And with only this three bands it is possible to recreate almost exactly what our eyes see.

This kind of cameras are normally cheaper and easier to find, however, there are different kind of cameras in the market for specific uses, depending on the applications. For example, RGB-D cameras, which record also the depth of each pixel, high speed cameras, that record videos with a higher number of frames, 3D cameras, and so on.

For the agricultural field one specific type of sensor is more common, they are called Near-infrared (NIR); they capture light invisible to the human eye, in our case, plant leaves exhibit high reflectivity in the NIR spectrum due to their chlorophyll content. NIR channel is useful in general for almost all agricultural tasks. For example separating vegetation from soil and other background data [7], detecting ill leaves, water stress, and so on.

This kind of cameras are very effective tool for evaluating soil productivity and analyzing crop health, having benefits such as the following:

- Identify weeds, disease and pests. Optimize pesticide usage and crop sprays.
- Refine fertilization detecting nutrient deficiencies, providing data on soil fertility.
- Help with land management, rotating crops or whether to take agriculture land in or out of production.
- Count crops to determine population and estimate crop yield.
- Determine spacing issues.
- Measure and control crop irrigation by identifying areas where water is needed. Help making improvement to land areas such as install drainage systems and waterways based on the multispectral data.
- View damage to crops from farm machinery and determine necessary repairs.
- Survey farm buildings and fencing.
- Monitor livestock.

The camera used to collect the dataset was JAI AD-130GE multi-spectral camera, which is a prism-based 2-CCD multi-spectral vision sensor. The color and the monochrome CCD of the JAI camera have both a size of 1/3", providing an image resolution of 1296 pixel $\times$ 966 pixel. One key feature of this camera system is its prism-base design, because the optical path of the RGB and NIR channels are identical, the image seen by both sensors are the same, and it can be treated as one 4-channel image, as shown in Figure 2.2.

The camera is mounted at the bottom of the robot chassis at a height of around 85 cm above soil, looking straight downwards. Using a Fujinon TF8-DA-8 lens with 8 mm focal length; this setup yields a ground resolution of approximately 3 px/mm and a field of view of 24 cm $\times$ 31 cm on the ground. The main purpose of the JAI AD-130GE camera is to capture detailed visual information of the plants for the perception system of the robot. Also, in order to be independent of natural light sources, an opaque shroud is mounted on the bottom of the robot chassis providing controlled high-performance artificial light sources.

2.3 Image dataset presentation

The part of the dataset used for this project is composed of both the RGB and NIR images, that are taken from the same camera, and some other annotation files that help to classify and segment between the different plants found in the field.

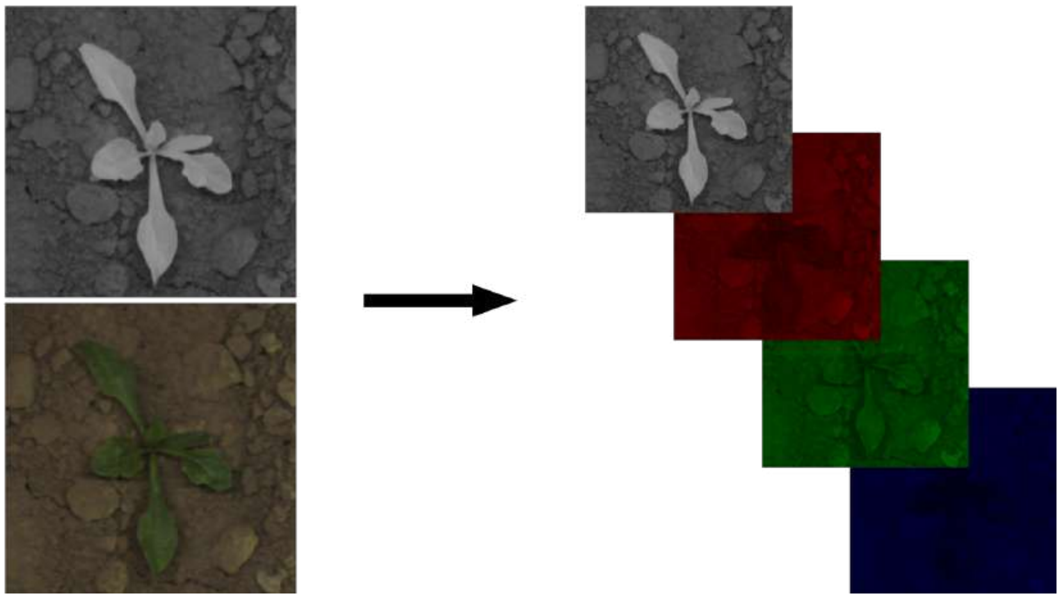

Figure 2.2. Image channel decomposition.

One of this is a *yaml* file that lists for each image the contour of the objects, its type (if it is a sugar beet or a kind of weed) and the stem position (x and y coordinates of the plants' center position). The stem position is a valuable information because it allows to center the plant in the image, it also helps the network to learn better how to generate the plants.

The images provided by the dataset are presented in Figure 2.3, which shows the RGB and NIR images in the first line and the masks for both RGB and NIR in the second line. In the mask image for the RGB image it is possible to see the sugar beet plants in green and some small weeds in red.

It is important to notice that not all the sugar beet images are used for the training. Only images bigger than the final image desired size can be used and reduced to compose our dataset. Figure 2.4 shows the number of images of each size giving an idea of the amount of data available for training depending on the goal of the project. For example, considering that is necessary to train a network to generate images of 512 pixel × 512 pixel, the total amount of examples available is more limited, because only images higher than this size can be used for training. So, the smaller the desired image size, the more samples are available for use.

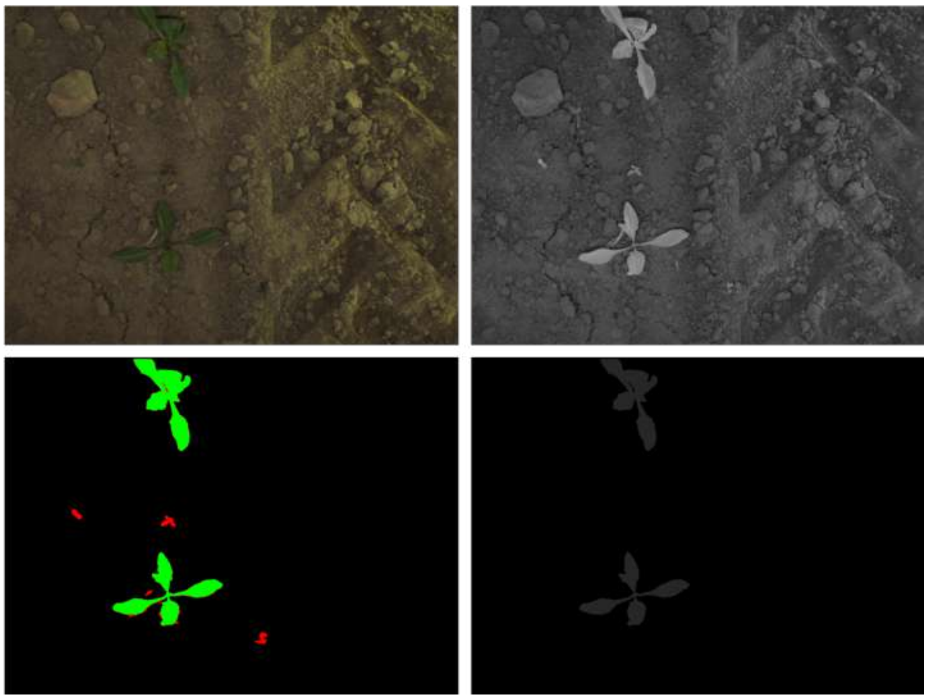

Figure 2.3. Dataset images example - First line: RGB and NIR images taken from the camera. Second line: RGB and NIR masks.

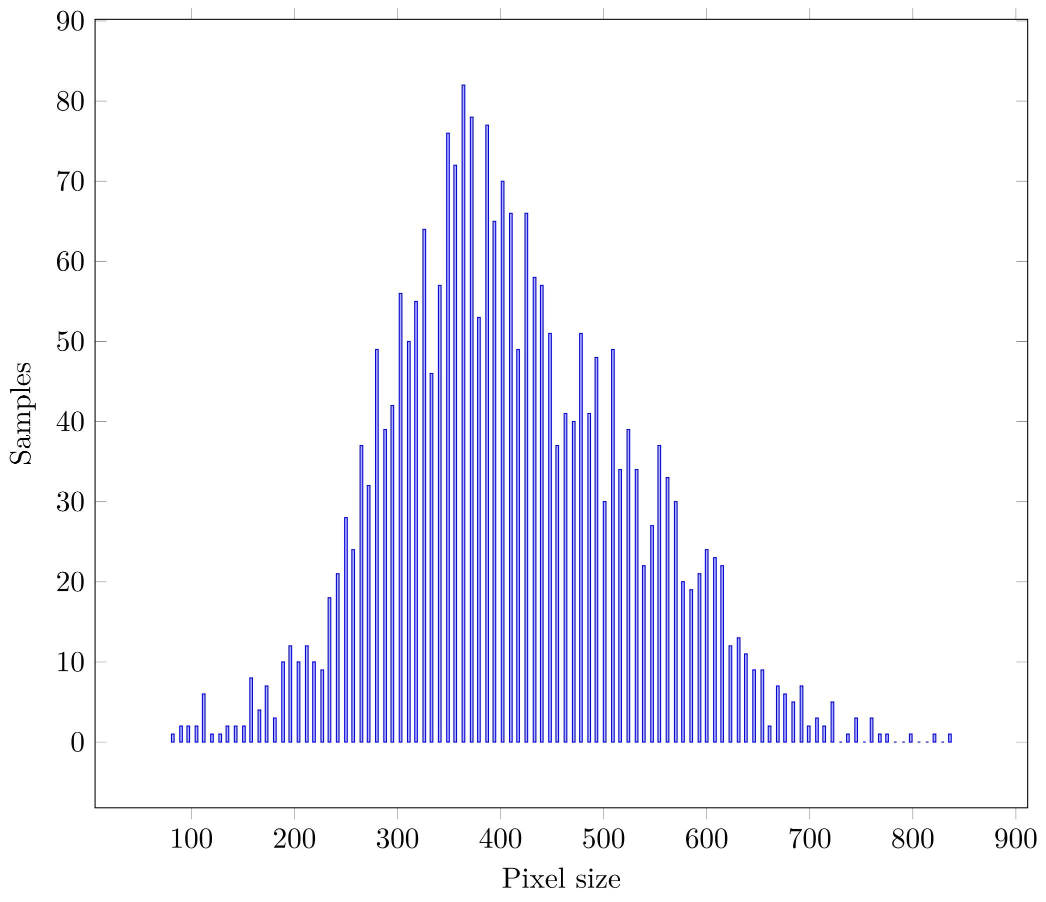

Figure 2.4. Number of samples distribution per image size.

Chapter 3

Generative Adversarial Networks

Generative Adversarial Networks (GANs) are deep neural network approach comprised of two nets. One net is known as the Generator, and the other known as the Discriminator, this two networks learn together in a process where one tries to fool the other. The generator learns how to generate better data, similar to the real ones, and the discriminator learns how to become better differentiating generated from the real data.

This chapter will give an overview of what actually is a GAN, how it works, and an explanation on the GANs used to develop this project.

3.1 GANs

GANs were introduced in a paper by Ian Goodfellow and other researchers at the University of Montreal in a paper titled 'Generative Adversarial Nets' [8]. The possibility to mimic any data distribution makes of GANs a network with a huge potential. They can be taught to create data impressive similar to the real ones in any domain: images, music, speech, text, and so on.

3.1.1 Supervised vs. Unsupervised Learning

Most of the machine learning algorithms uses Supervised Learning. It is called supervised learning when the goal is to map a function from an input to an output variable, this way both variables need to be known in advance.

The basic idea of a supervised learning process is that the answer of what is needed is already known, so the algorithm iteratively tries to make a prediction when training in order to minimize a cost function. Some known examples of Supervised Learning are classification, where the network learns how to classify an input between different classes, and regression, when the output is a real value.

On the other hand, Unsupervised Learning is characterized when only input data is used, and there is no output variable, so the idea becomes to model the structure or distribution of the data. Some examples of Unsupervised Learning can be clustering, where the model discover how to group the data or association, where the idea is to find rules that describe large portions of the data.

GANs are a special case, because they are unsupervised learning algorithms that use a supervised loss as part of the training. The data has no label and the idea is not to generalize any kind of prediction to new data. The goal is to learn to model how the data looks like and generate new samples that follow the same data distribution. However, it uses a supervised learning classification in order to classify if a sample is from the generator or real image.

3.1.2 Discriminative vs. Generative Modeling

In the same way, there is the discrimination between Disccriminative and Generative Modeling. Discriminative Modeling is in a sense that it tries to classify the input data into some class, given the feature of an instance of data, it predicts a label or category to which the data belongs. The opposite of this behavior would be to predict a feature from a given label, which can be called Generative Modeling.

3.1.3 How GANs Work

GANs are basically composed of two networks, a generator and a discriminator. The work of the generator is to create new instances of the desired data, while the discriminator evaluates for authenticity, it tries to discriminate the samples given to it between real ones (taken directly from the training data) and synthetic ones (generated from the generator).

The neurons of this two networks start with random weights, so at the beginning the generator does not know anything about the data and the same way, the discriminator also cannot tell the difference between the real and the noise images generated by the generator. With time, both networks start learning, one trying to fool the other, the generator needs to generate more and more realistic images and the discriminator need to learn better and better how to differentiate a synthetic from a real one.

Figure 3.1 shows how a GAN is structured, a noise is fed to the Generator, which generates fake images, these images are fed to the discriminator together with real ones, and it needs to learn how to distinguish them. It means that the discriminator is in a feedback loop with the ground truth of the images (Supervised

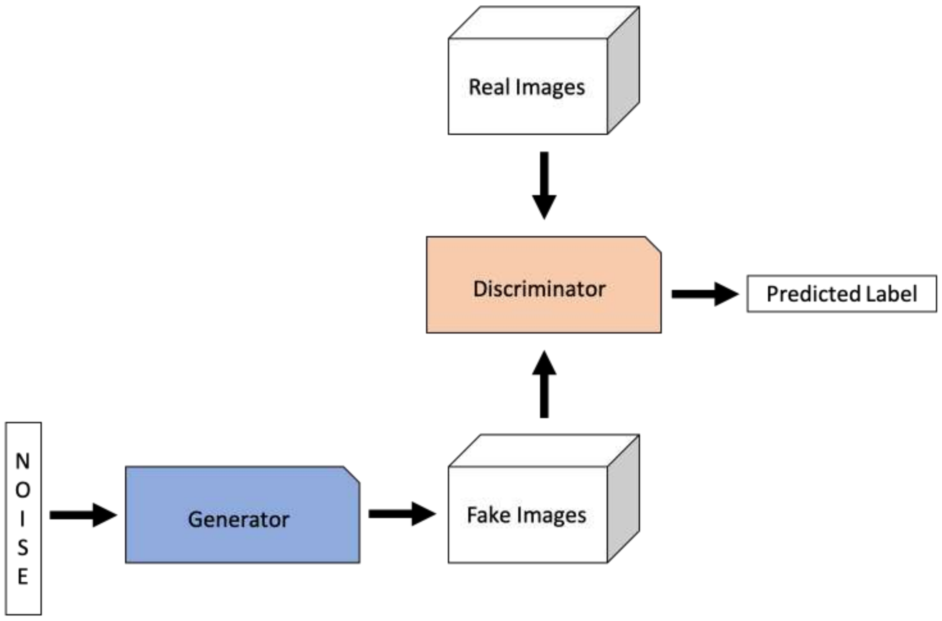

Figure 3.1. GAN structure representation.

Learning 3.1.1), and the generator is in a feedback loop with the discriminator (Unsupervised Learning 3.1.1).

In this project there will be two of this networks, one for the mask model with the noise as an input of the generator and another one for the RGB and NIR images, using the mask for the generator.

Both networks, generator and discriminator, are being trained at the same time, but it is important to notice that each should train against a static adversary, so when the discriminator is being trained, the values of the generator should be held constant, the same way, when training the generator, the discriminator values should also be held constant.

Another important factor when training a GAN is that both networks should evolve together, it means that no network should overpower the other. For example, if the discriminator is much better than the generator it will easily be able to distinguish the images, and the generator will have problems to read the gradient. The same happens if the generator is much better, it will persistently exploit weaknesses in the discriminator that lead to false negatives.

3.1.4 Mathematical Formulation

Mathematically speaking, for the Generator, an input noise variable $p_z(z)$ is defined in order to learn the generator's distribution p_g over data x, this mapping is represented as $G(z; \theta_g)$, where G is a differentiable function that represents a multilayer perceptron with parameters θ_g.

For the Discriminator, the multilayer perceptron $D(x;\theta_d)$ outputs a single scalar (real or synthetic), so $D(x)$ represents the probability that x comes from the data rather than p_g. This way, D is trained in order to maximize both the probability of assigning the correct labels to the training examples as well as samples generated by G. The same way, G is trained to minimize $\log(1 - D(G(x)))$.

It is characterized by D and G playing a two-player minimax game given by the following value function $V(G,D)$ shown in the Equation 3.1:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{3.1}$$

The training process can be represented by the algorithm 1 below, taken from [8] and how the gradient is used in Figure 3.2:

Algorithm 1: Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter.

```
1 for number of training iterations do
2     for $k$ steps do
3         • Sample minibatch of $m$ noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior
            $p_g(z)$
4         • Sample minibatch of $m$ examples $\{x^{(1)}, ..., x^{(m)}\}$ from data
            generating distribution $p_{data}(x)$
5         • Update the discriminator by ascending its stochastic gradient:
              $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$
6     end
7     • Sample minibatch of $m$ noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior
        $p_g(z)$
8     • Update the generator by descending its stochastic gradient:
          $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))$
9 end
```

3.1.5 GANs Problems

Developing and training GANs is not an easy task, there are several problems that can happen in this process and it is good to keep in mind some of the challenges that may be found, such as:

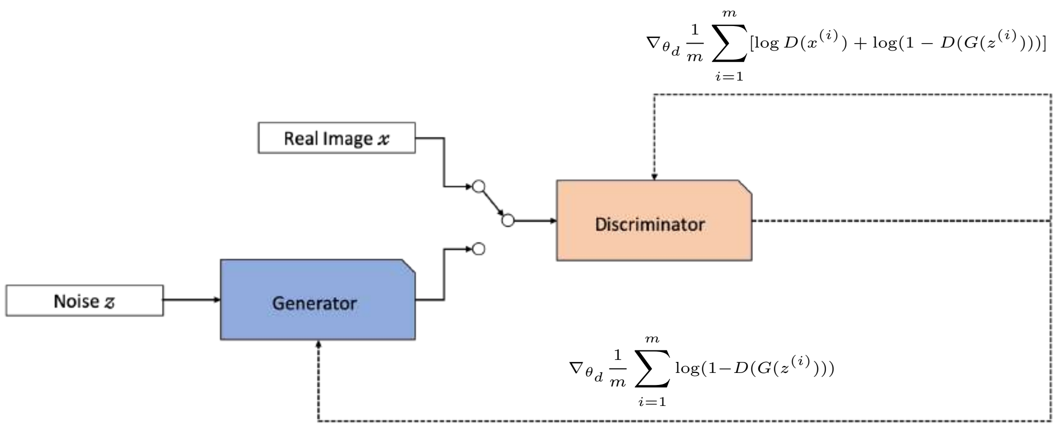
$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$
Real Image x
Discriminator
Noise z
Generator
$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))$

Figure 3.2. GAN gradient propagation.

- **Non-convergence**: it can happen when the model parameters oscilate, destabilize and never converge
- **Mode collapse**: the generator collapses and end up producing a limited variety of samples
- **Diminished gradient**: the discriminator becomes much better, causing the generator gradient to vanish and not be able to learn anymore
- Overfitting caused by unbalance between generator and discriminator
- Highly sensitivity to the hyperparameters selection

3.2 DCGAN

One of the reasons why the generator and the discriminator need to be trained in sync is to avoid the overtraining of one with respect to the other. If this unbalanced training status happens, the training becomes unstable, and this is in fact one of the problems of the first GAN model suggested in the original paper [8]. Aiming to solve this issue, the Deep Convolutional GANs (DCGANs) [4] introduced in 2016 a set of constraints on the architectural topology in order to make the training stable in most settings.

The first and most important thing modified is that all the downsampling and upsampling are substituted by convolutional stride and transposed convolution [9]. It allows the network to learn its own spatial downsampling and upsampling.

Some other modifications include the elimination of fully connected layers [10], the reason is that it was found that using global average pooling helped increasing stability, however hurting convergence speed.

Also, the use of batch normalization with exception of the output layer for the generator and the input layer of the discriminator [11]. It helps stabilizing learning

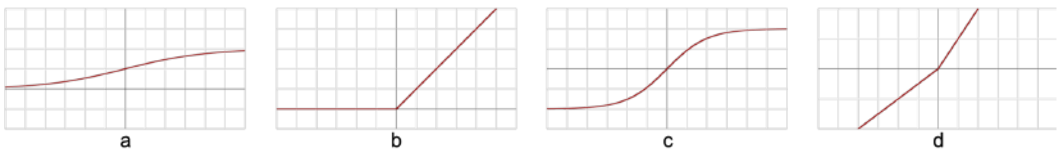

Figure 3.3. Activation functions: a) sigmoid funcion; b) Rectified Linear Unit (ReLU); c) Hiperbolic Tangent (tanh); d) Leaky Rectified Linear Unit (Leaky ReLU).

by normalizing the input to each unit to have zero mean and unit variance.

And lastly, use Rectified Linear Unit (ReLU) as activation, except for the output which uses Hiperbolic Tangent (tanh) for the generator, and use Leaky Rectified Linear Unit (Leaky ReLU) as activation for the discriminator [12]. For the generator, it was observed that the model learned quicker to saturate and cover the color space of the training distribution, and for the discriminator it worked better, especially for higher resolution modeling. It differs from the original GAN where the generator used a mixture of ReLU and sigmoid activation, while the discriminator used maxout activations, what means that it takes the maximum value of the pre-activations and reshapes it into a vector containing only this value. Figure 3.3 presents a comparison of the different activation functions used on GAN and DCGAN.

Figure 3.4 shows an example of a DCGAN, the generator is represented in the first line of the image, having as an input a random noise vector and an image in the output. In the second line is the discriminator representation, having an image in the input and a binary classification in the output.

Summarizing, the differences from DCGANs to GANs that make its training stable are the following:

- For the discriminator, pooling layers are replaced with strided convolutions
- For the generator, pooling layers are replaced with fractional-strided convolutions
- Use batch normalization in both the generator and the discriminator
- Remove fully connected hidden layers for deeper architectures
- ReLU is used as activation in the generator for all layers except for the output, which uses tanh
- Leaky ReLU is used as activation in the discriminator for all layers

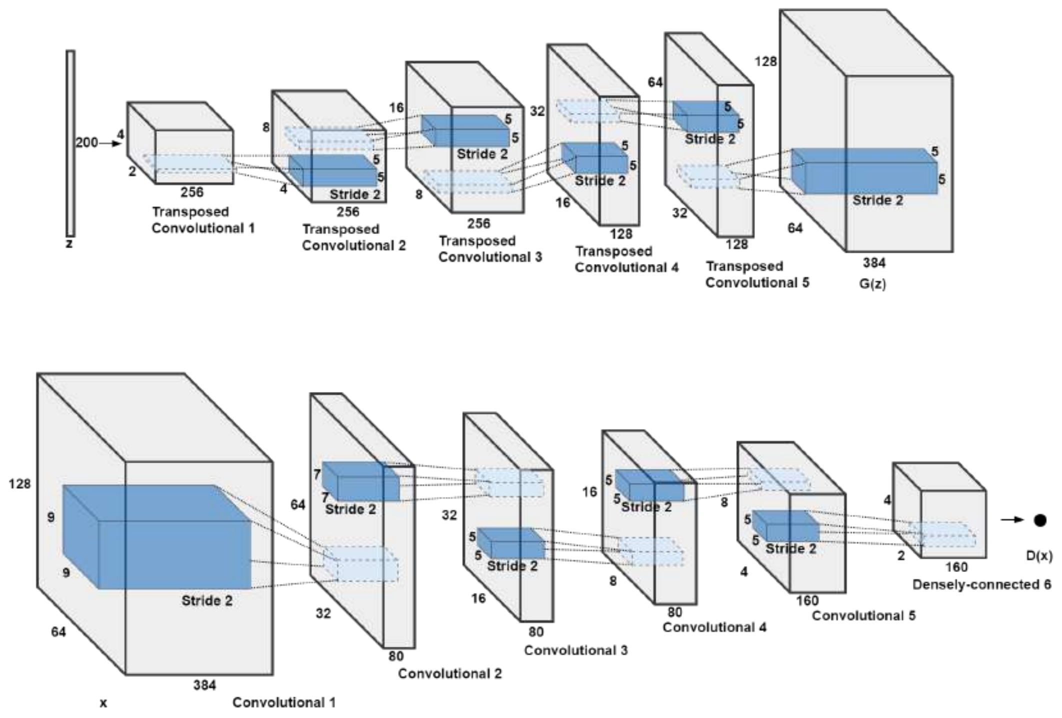
z
200
Transposed Convolutional 1
Transposed Convolutional 2
Transposed Convolutional 3
Transposed Convolutional 4
Transposed Convolutional 5
G(z)
Stride 2
x
Convolutional 1
Convolutional 2
Convolutional 3
Convolutional 4
Convolutional 5
Densely-connected 6
D(x)

Figure 3.4. DCGAN structure representation. Generator in the first line and Discriminator in the second one.

3.3 SPADE

SPADE stands for SPatially-Adaptive (DE)normalization [5] and is one type of conditional GAN (cGAN). The cGANs are a specific type of GAN that instead of receiving only a random noise as an input, it receives also a condition, which can be an image, a number, a class label, a mask, and so on.

This type of network is a Semantic Image Synthesis that is converting a semantic segmentation mask to a photorealistic image. Its goal is basically the opposite of an image segmentation network. There are a lot of publications presenting networks with impressive results [13] [14], however, this network is considered now the state-of-the-art for this specific use.

SPADE architecture is presented in Figure 3.5, the Image Encoder encodes a real image to a latent representation for generating a mean and a variance vector, with this it will be possible to change the style of the image. The generator uses the segmentation map in each SPADE ResBlk, explained below, and the discriminator takes a concatenation of the segmentation map with the original or generated image by the discriminator and aims to correctly classify as real or fake.

The biggest different from SPADE to the previous works is in the batch normalization layer. Previous works used the segmentation map only in the input of the network, and because it was only available in the first layer its information

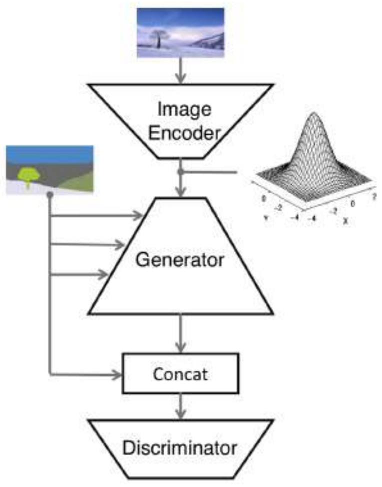

Figure 3.5. SPADE architecture.

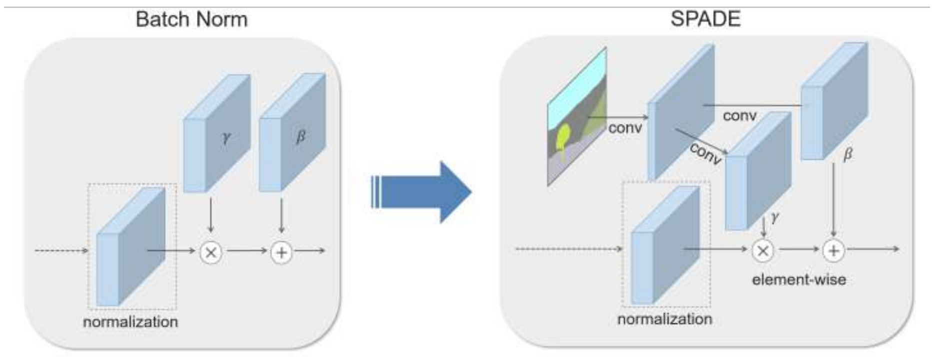

Figure 3.6. Comparison between common batch normalization and SPADE batch normalization.

content used to disappear in deeper layers after many batch normalizations. SPADE uses the semantic segmentation mask to learn the affine layer, this affine parameters are spatially-adaptive, it means that it uses different scaling and bias for each semantic label.

Figure 3.6 shows the difference between a normal batch normalization and SPADE, SPADE first passes the segmented map through convolutional layers to produce feature tensors and then, after normalization, element-wise operations is done using the feature tensors.

3.3.1 Generator

In Figure 3.7 is presented the SPADE Generator, it is based on pix2pixHD [14] and starts with a random noise in the input and uses the semantic map at every SPADE ResBlk upsampling layer. Also, with this network, it is possible to have separation between semantic and style control. Changing the semantic map it is

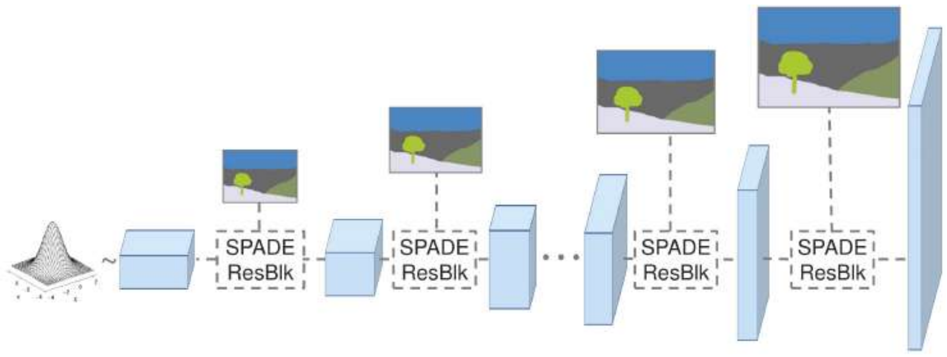

Figure 3.7. SPADE Generator.

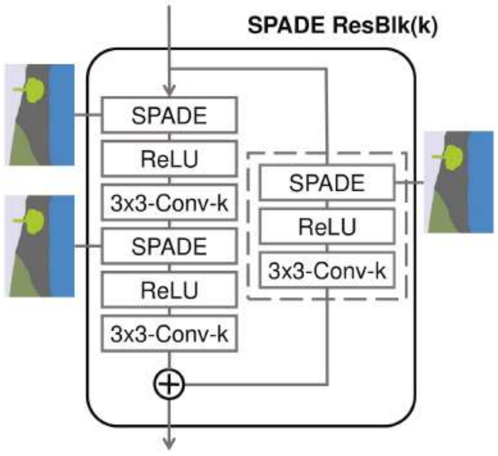

Figure 3.8. SPADE ResBlk.

possible to change the final content, and changing the random vector it is possible to change the style of the image.

The SPADE ResBlk is the residual block presented in Figure 3.8, it follows the implementation in [15] and [16], where each SPADE block refers to the one presented in Figure 3.6.

3.3.2 Discriminator

The discriminator architecture also follows pix2pixHD, using a multi-scale design with instance normalization, with the difference of applying spectral normalization to all convolutional layers of the discriminator. Figure 3.9 shows the discriminator model, it is base on Patch-GAN [13] and it takes the concatenation of the segmentation map with the image.

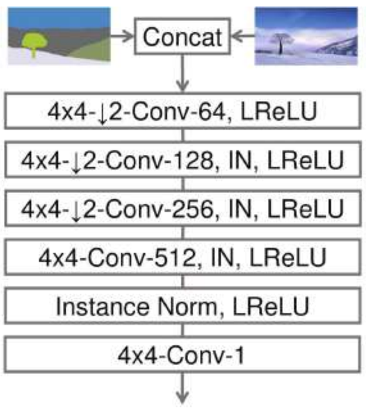

Figure 3.9. SPADE Discriminator.

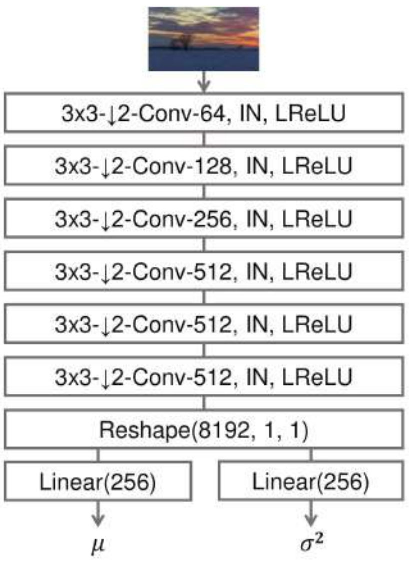
3x3-↓2-Conv-64, IN, LReLU
3x3-↓2-Conv-128, IN, LReLU
3x3-↓2-Conv-256, IN, LReLU
3x3-↓2-Conv-512, IN, LReLU
3x3-↓2-Conv-512, IN, LReLU
3x3-↓2-Conv-512, IN, LReLU
Reshape(8192, 1, 1)
Linear(256)
Linear(256)
μ
σ^2

Figure 3.10. SPADE Encoder.

3.3.3 Image Encoder

The encoder, shown in Figure 3.10 is responsible for producing the mean (μ) and covariance (σ^2), it is composed of six convolutional layers with stride 2 followed by two linear layers.

Chapter 4

Metrics

Evaluating GANs is a very challenging task, and several things need to be thought as necessary condition when defining metrics that can produce meaningful scores, such as distinguishing generated from real samples, detecting overfitting, and identifying mode dropping and mode collapsing. For most of the GANs development nowadays, the evaluations depends on visual inspection to check the fidelity of the generated images. This kind of evaluation is still considered the best way, but it is time-consuming, subjective and most of the times can also be misleading. Given all this limitations of qualitative evaluations, the need of proper quantitative metrics are crucial to guide the design of better models and speed up the automatic parameters search on the development of GANs. The metrics described here are based on the paper titled 'An empirical study on evaluation metrics of generative adversarial networks' [17].

The methodology used for evaluation follows the setup illustrated in Figure 4.1. The idea is to use the samples generated by the network an samples collected from the dataset, extract features from this two sets of samples and then calculate the metrics. Six methods will be described, Inception Score [18], Mode Score [19], Kernel MMD [20], Wasserstein distance [21], Fréchet Inception Distande (FID) [22] and 1-nearest neighbor (1-NN) [23].

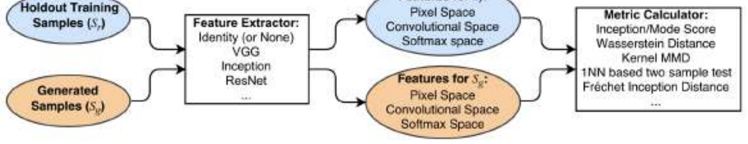

Figure 4.1. Typical sample based GAN evaluation methods.

4.1 Inception

This is probably the most popular metric, it is capable of measuring not only the quality but also the diversity of generated images using an external model, the Google Inception network [24], trained with the ImageNet dataset [25].

The Formula 4.1 shows how the Inception Score is calculated, considering a pretrained model $\mathcal{M}$, $p_{\mathcal{M}}(y \mid \mathrm{x})$ refers to the label distribution of x predicted by $\mathcal{M}$, and $p_{\mathcal{M}}(y) = \int_{\mathrm{x}} p_{\mathcal{M}}(y \mid \mathrm{x}) d\mathbb{P}_g$, which means the marginal of $p_{\mathcal{M}}(y \mid \mathrm{x})$ over the probability measure $\mathbb{P}_g$. The expectation and the integral in $p_{\mathcal{M}}(y \mid \mathrm{x})$ can be approximated with independent and identically distributed samples from $\mathbb{P}_g$.

For this metric, a higher IS means that $p_{\mathcal{M}}(y \mid \mathrm{x})$ is close to a point mass, that happens when the Inception network is very confident that the image belongs to a particular class, and has $p_{\mathcal{M}}(y)$ close to uniform, which means that all categories are equally represented. It suggests that the GAN has both high diversity and quality.

$$\mathrm{IS}(\mathbb{P}_g) = e^{\mathbb{E}_{x \sim \mathbb{P}_g}[KL(p_{\mathcal{M}}(y|\mathrm{x}) \| p_{\mathcal{M}}(y))]} \tag{4.1}$$

4.2 Mode Score

Represented in Equation 4.2, is an improved version of the Inception Score, but different from it, it is able to measure the difference between the real distribution $\mathbb{P}_r$ and the generated distribution $\mathbb{P}_g$ through the term $KL(p_{\mathcal{M}}(y) \parallel p_{\mathcal{M}}(y^*))$. The term $p_{\mathcal{M}}(y^*) = \int_{\mathrm{x}} p_{\mathcal{M}}(y \mid \mathrm{x}) d\mathbb{P}_r$ is the marginal label distribution for the samples from the real data distribution.

$$\mathrm{MS}(\mathbb{P}_g) = e^{\mathbb{E}_{x \sim \mathbb{P}_g}[KL(p_{\mathcal{M}}(y|\mathrm{x}) \| p_{\mathcal{M}}(y))] - KL(p_{\mathcal{M}}(y) \| p_{\mathcal{M}}(y^*))} \tag{4.2}$$

4.3 Kernel MMD (Maximum Mean Discrepancy)

Defined by the Equation 4.3, it measures the difference between $\mathbb{P}_r$ and $\mathbb{P}_g$ for some fixed kernel function k. The empirical MMD between two distributions is computed using finite sample approximation of the expectation. A lower MMD means that the two given samples of $\mathbb{P}_r$ and $\mathbb{P}_g$ are close to each other.

$$\mathrm{MMD}^2(\mathbb{P}_r, \mathbb{P}_g) = \mathbb{E}_{\substack{x_r, x'_r \sim \mathbb{P}_r \\ x_g, x'_g \sim \mathbb{P}_g}}[k(x_r, x'_r) - 2k(x_r, x_g) + k(x_g, x'_g)] \tag{4.3}$$

4.4 Wasserstein distance (EMD)

The Wasserstein distance between $\mathbb{P}_r$ and $\mathbb{P}_g$ is defined as shown in Equation 4.4, where $\Gamma(\mathbb{P}_r, \mathbb{P}_g)$ refers to the set of probabilistic couplings (all joint distributions) whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$, and $d(\mathrm{x}^r, \mathrm{x}^g)$ represents the base distance between the two samples. The Wasserstein distance will be lower when the two distributions are more similar.

$$\mathrm{WD}(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Gamma(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(\mathrm{x}^r, \mathrm{x}^g) \sim \gamma}[d(\mathrm{x}^r, \mathrm{x}^g)] \tag{4.4}$$

In case of discrete distributions with densities p_r and p_g, Wasserstein distance is usually called Earth Mover's Distance (EMD), and corresponds to the formula in Equation 4.5.

$$\begin{aligned} \mathrm{WD}(p_r, p_g) &= \min\nolimits_{w \in \mathbb{R}^{n \times m}} \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} d(\mathrm{x}_i^r, \mathrm{x}_j^g) \\ \text{such that } \sum_{j=1}^{m} w_{i,j} &= p_r(\mathrm{x}_i^r)\ \forall i,\ \sum_{i=1}^{n} w_{i,j} = p_g(\mathrm{x}_j^g)\ \forall j \end{aligned} \tag{4.5}$$

4.5 Fréchet Inception Distance (FID)

Shown in Equation 4.6, it calculates the Fréchet distance between the two Gaussian distribution. It uses a feature function ϕ (Inception network's convolution feature, by default) to model $\phi(\mathbb{P}_r)$ and $\phi(\mathbb{P}_g)$ as Gaussian random variables with means μ_r, μ_g and empirical covariance C_r, C_g.

$$\mathrm{WD}(\mathbb{P}_r, \mathbb{P}_g) = \parallel \mu_r - \mu_g \parallel + \mathrm{Tr}(\mathrm{C}_r + \mathrm{C}_g - 2(\mathrm{C}_r \mathrm{C}_g)^{1/2}) \tag{4.6}$$

4.6 1-Nearest Neighbor classifier (KNN)

This kind of metric is used in two-sample tests to evaluate if two distributions are identical or not. For two sets of samples $S_r \sim \mathbb{P}_r^n$ and $S_g \sim \mathbb{P}_g^m$, with $|S_r| = |S_g|$ it is possible to compute the leave-one-out (LOO) accuracy of a 1-NN classifier that was trained on S_r and S_g with positive labels for S_r and negative labels for S_g.

This metric should reach $\sim 50\%$ LOO accuracy when $|S_r| = |S_g|$ is large, what happens when the two distributions match. The LOO will be lower than 50% when the GAN overfits $\mathbb{P}_g$ to S_r, and if the GAN, for example, memorize every sample in S_r and re-generate it, $S_g = S_r$, and the accuracy would be 0%, because every sample from S_r would have it nearest neighbor from S_g with zero distance.

Chapter 5

Semantic Segmentation

Besides the metrics used for evaluate the generated images, it is also important to use a dataset augmented with the synthetic images and use it for a specific task in order to confirm that the network is learning better the task because of the augmented dataset. For this task a Semantic Segmentation network is trained to segment the RGB images collected by the robot. This chapter will describe what a Semantic Segmentation is and what particular architecture will be used for it.

5.1 Image Understanding

When talking about understanding images, there are various levels of granularity in which computers can perform its task, and each of these levels is a problem in the Computer Vision domain. These tasks can be:

Image Classification

This is the most fundamental task in Computer Vision, given an image the computer needs to output the label of the main object in the image. For image classification it is expected that there is only one main object in the image.

Classification and Localization

This is one step further of classification, in this case the computer also needs to localize exactly where the object is present in the image. In this case the localization is given by a bounding box and is also expected that there is only one main object in the scene.

Object Detection

Object Detection extends Classification and Localization for images with more than one main object. In this case the task is to classify and localize all the objects in the image also with the bounding box.

Semantic Segmentation

The goal of Semantic Segmentation is not just classify and localize the objects in the image with a bounding box but classify each pixel of the image with a specific label. In the literature this task is also commonly referred to as dense prediction.

For the Semantic Segmentation, the input is the full image of the scene and the output is a high resolution image (usually of the same size as the input image) where each pixel is labeled with a particular class.

Instance Segmentation

Instance Segmentation is one step ahead of Semantic Segmentation, and in this case, besides classifying each pixel of the image it is also expected to know to which instance of a class each pixel belongs to.

For example, in a picture with a group of people (n instances of the class person), a Semantic Segmentation would classify all the people as person, but in Instance Segmentation it is desired to know if a specific pixel belongs to person number one or person number two.

5.2 Applications

At first, it seems that classifying and identifying objects in an image is useful and important only for papers publication, however this is something that is very important and will be more and more present in people's lives in the future. Some of the places where it is required are:

Autonomous Vehicles

Autonomous driving rely a lot on computer vision, it is a complex robotics task in an evolving environment that requires perception, planning and execution. It is used for detecting lanes, pedestrians, vehicles, traffic lights and any error can be fatal.

Medical Image Diagnosis

Computer are more and more substituting and outperforming doctors in analysing medical images. And for this task image understanding is very important for applications such as: cell classification, X-ray, MRI, antigen detection, lung segmentation, and blood vessel quantitative measurements, for example.

Geo Sensing

Another good application is for Geo Sensing, where Semantic Segmentation can be used processing satellite images to recognize type of land cover, such as urban areas, agriculture, water, monitoring areas of deforestation or traffic management, city planning and road monitoring.

Precision Agriculture

And last, the subject of this work, Precision Agriculture, as already said, can be used to determine production, monitor field and reduce thee amount of herbicides used.

5.3 UNET Architecture

For the segmentation task, it was used the network published in the paper titled 'U-Net: Convolutional Networks for Biomedical Image Segmentation' [26]. It was developed by Olaf Ronneberger et al. for Bio Medical Image Segmentation.

As shown in Figure 5.1, the architecture looks like a 'U', justifying where the name comes from. The architecture is divided in three sections, the contraction (also called encoder), the bottleneck, and the expansion (also called decoder).

The contraction is made of many blocks that applies two 3×3 convolution layers followed by a 2×2 max pooling. After each block the number of kernels or feature maps doubles, making it possible for the architecture to learn the complex structures effectively. The layer between the contraction and the expansion section, the bottleneck, uses two 3×3 CNN layers followed by 2×2 up convolution layer.

The big contribution of this network is the expansion section, it works similar to the contraction section, each expansion block passes the input to two 3×3 CNN layers followed by a 2×2 upsampling layer, where after each block number of feature maps used by convolutional layer get half to maintain symmetry. The trick is that at every block the input is concatenated with the corresponding feature map of the corresponding contraction layer, this way, the features learned while contracting the image are used to reconstruct the segmentation map.

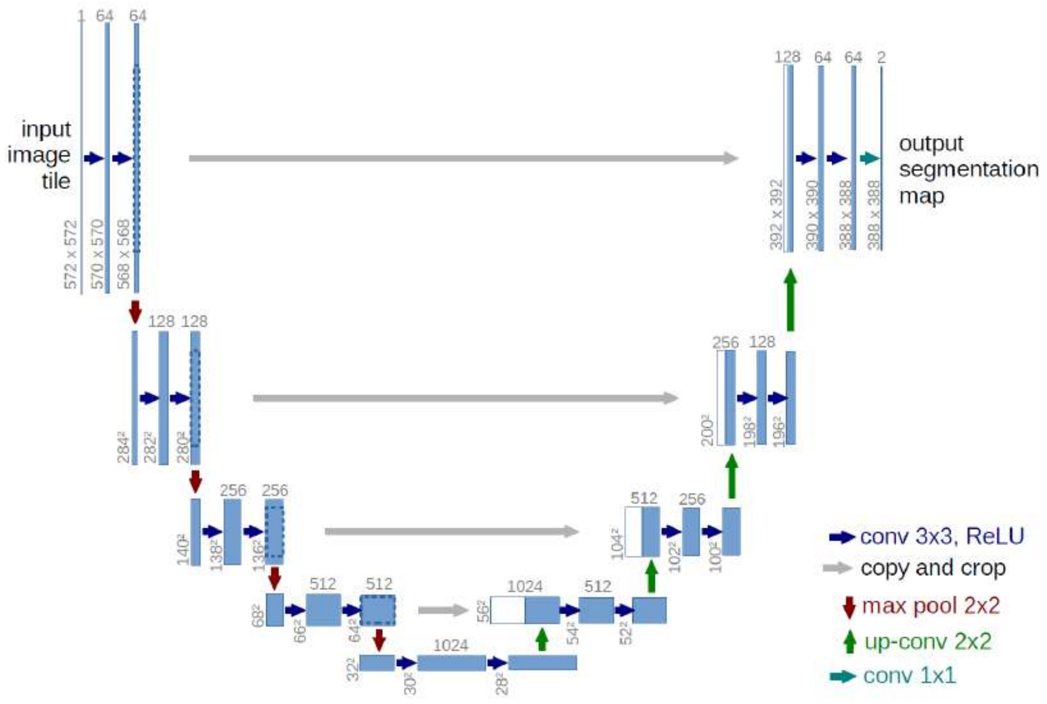

Figure 5.1. UNET Architecture.

The number of contraction and expansion blocks are the same, and at the end, the resultant map passes through another 3×3 CNN layer with the same number of feature maps of the segments desired.

Chapter 6

Implementation

This chapter will present all the implementation developed in order to create and evaluate the synthetic images. The first step is the dataset processing, where the images are organized in a way the GANs are able to use it. The second is the metrics evaluation during training. The third step is the mask generation, this is the first step for the creation of the synthetic dataset. The forth is the final RGB and NIR generation, where the synthetic dataset is complete. And lastly, the segmentation network that will be able to validate the results achieved.

The code was entire developed using Python language [27] an TensorFlow library [28] for development and train the Machine Learning models, and can be found in the repository [29].

6.1 Dataset Processing

Before starting training GANs, it is necessary to process the dataset in order to organize the images in a way that the networks can consume it. The dataset used is the Agricultural robot dataset [1], it provides the images as presented in Figure 2.3, composed by RGB and NIR image and a segmentation for each one.

The algorithm used is the one presented in Algorithm 2 and the process starts in line 1, where the list of all *yaml* files is acquired for processing. One *yaml* example file is shown in Listing 6.1, in this file it is possible to see the name of the *png* image, the plant id, type, stem position, contour and some other parameters. Each *yaml* file may contain more than one plant and all of them need to be processed. The algorithm then goes through each of this files, it reads the *yaml*, Line 4 and all the images necessary for the dataset construction, RGB, NIR, RGB and NIR mask, Line 5.

The RGB and NIR images are taken from the same camera, but because of small distortions, they first need to be undistorted using the camera matrices, provided by the dataset in Line 6. It was noticed that even after undistorting the images they were one or two pixels off when superposing both, this displacement is not relevant for most applications, but in this case it was necessary to have a perfect match, because of this, it was decided to also apply a perspective transformation, Line 7.

After having all the images in memory, it is possible to go through each plant present in the *yaml* file, that corresponds, obviously, to each plant in the images. The first thing is to get the stem position of the file in Line 10, the stem position makes all plants to be centered, what makes the learning process easier for the GANs.

Before starting processing the images it is useful to check if the stem is inside the boundaries of the image, Line 12, because for this dataset only images that are fully inside the picture are desired. Later it will be checked again if the crop is fully inside the image, but this check here avoids some computation steps.

The process of creating the mask itself is being shown in Figure 6.1. In Line 13 and 14 a blank mask is created and the contour is draw on it, this mask is used for bitwise operation with the RGB and NIR images, Lines 15 and 16, this two masks are at the end summed and the final crop mask is defined, Line 17.

Next, in order to calculate the cropping of the images, process shown in Figure 6.2, the maximum radius is calculated, for the radius calculation it is considered the most distance pixel from the stem, Line 18 of Algorithm 2. This radius is used to define the square, Line 20, so at the end the square will be of the size $2 \cdot radius \times 2 \cdot radius$ pixel.

Now, it is again checked to see if the square is fully inside the image and the radius is higher then the desired dimension, for example, if the goal is to generate 256×256 pixel images, the radius needs to be higher then 128. Then, the three images are croped, RGB, NIR and mask, Line 22, resized in Line 23, and augmented, Line 24.

For the augmentation, from the same image, it is possible to rotate and mirror, thus increasing the number of samples in the dataset. In Figure 6.3 is shown how the images were augmented, they were flipped horizontally, vertically and both at the same time, also for each one of these configurations, they were rotate 90 degrees. It would be possible to rotate at smaller angles, but in this case it creates a border interpolation problem, what can effect on the training.

Lastly, all the augmented images are saved in different folders in order to create the dataset for training, Line 25.

Algorithm 2: Image processing algorithm for dataset creation.

```
• Collect all yaml files

foreach yaml file do
    • Read yaml file
    • Read equivalent RGB, NIR, RGB mask and NIR mask
    • Undistort NIR and NIR mask
    • Applies perspective transformation to the NIR and NIR mask

    foreach crop inside yaml file do
        • Get stem_x and stem_y positions

        if $stem\_x > 0$ and $stem\_y > 0$ then
            • Create blank mask
            • Draw crop contour on mask
            • Bitwise operation with RGB mask
            • Bitwise operation with NIR mask
            • Build final mask
            • Calculate maximum crop radius
            • Define crop square

            if crop fully inside image and $radius > dim/2$ then
                • Crop mask, RGB and NIR
                • Resize
                • Augment
                • Save images
            end
        end
    end
end
```

Listing 6.1. *yaml* file example

```
filename: bonirob_2016-05-17-11-42-26_14_frame0.png
annotation:
- plant_id: 0
  type: SugarBeets
  level: 0
  stem:
    x: 440
    y: 751
  contours:
    - contour: 0
      x: [592, 590, 587, 585, 582, ...]
      y: [605, 604, 604, 604, 604, ...]
  params:
    threshold: 143
    minBrightness: 0
    fragmentSize: 63
- plant_id: 1
  type: SugarBeets
  level: 0
  stem:
    x: 418
    y: 287
  contours:
    - contour: 0
      x: [236, 235, 234, 233, 232, ...]
      y: [201, 202, 202, 203, 203, ...]
  params:
    threshold: 143
    minBrightness: 0
    fragmentSize: 63
image Id: 5425
time stamp sec: 1463478146
time stamp nsec: 284814296
frame: base_sensor_jai_camera_link
```

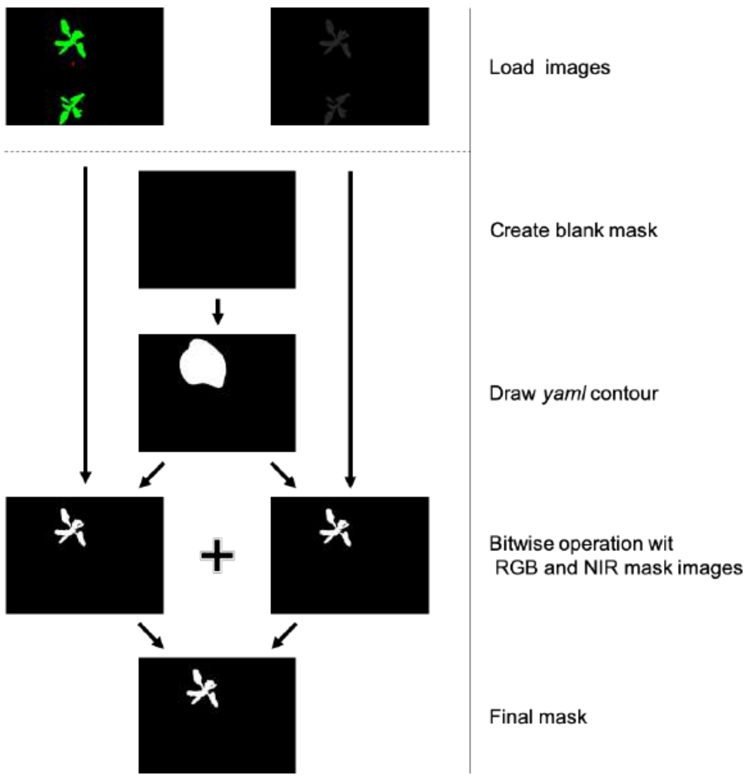

Figure 6.1. Mask processing.

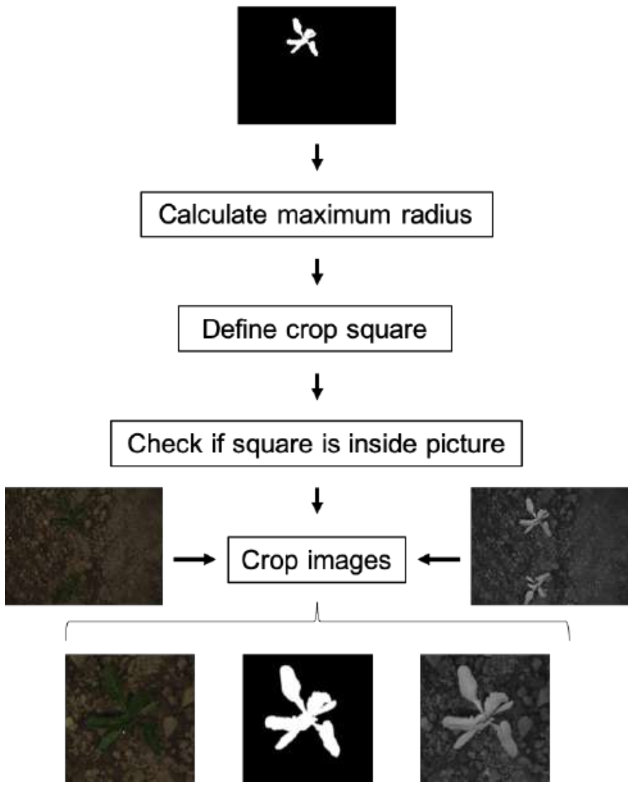

Figure 6.2. Crop process.

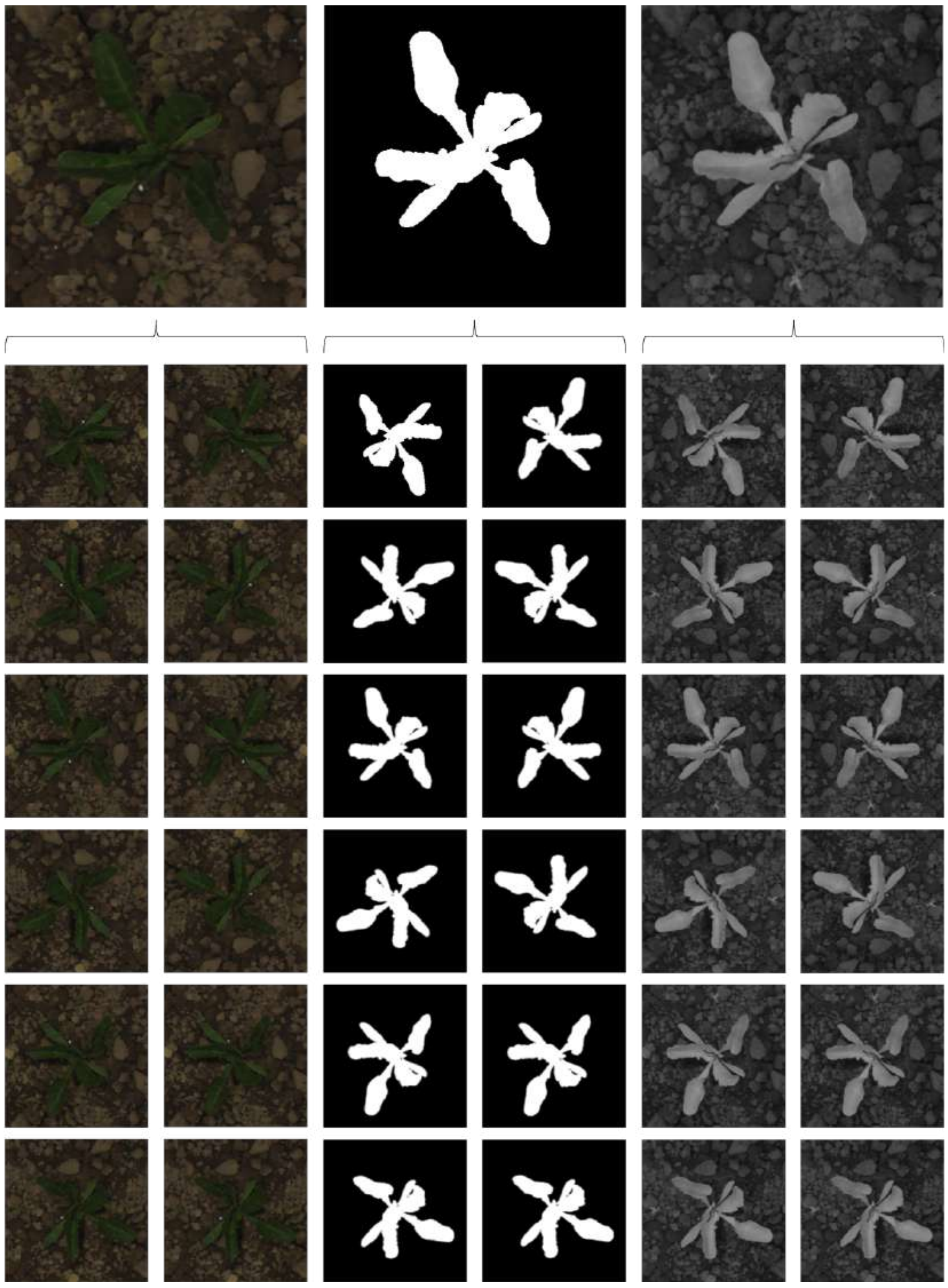
Figure 6.3. Augmentation process.

6.2 Metrics

As described in Chapter 4, the metrics were based on the paper 'An empirical study on evaluation metrics of generative adversarial networks' [17], and for this, it was used the code published on repository [30]. The code published with the paper is implemented in PyTorch [31], but the code was converted to TensorFlow, just in order to have everything standard with the same library. Later it was noticed that calculating the metrics with PyTorch was faster, so, for the results, it was used the original PyTorch implementation.

The metrics are calculate on every epoch and saved on a file for later use. It is used for the *gif* animation showing the training evolution, but the original idea of having metrics is the possibility to be able to evaluate the different models without the need of human inspection, this way it is possible to test different models and use parameters grid search to find the best network for this specific use.

6.3 Mask Generation

For the mask generation, the algorithm presented in Section 3.2 is used. The code was developed in a modular way, so the idea is to make it easier to simply plug another model, set the parameters and run it to test if the results are better or worse than the last implementations.

Listing 6.2 shows how the grid search is defined, any parameter can be defined as a list of parameters to be tested, not only the general parameters such as *epochs*, *latent_dim* (size of noise input vector) and *batch_size*, but also the specific parameters of each GAN, such as generator and discriminator learning rate, learning decay, and so on. In this example there are two GANs, DCGAN, and WGANGP [32], WGANGP was another network tried but at the end it presented worse results than DCGAN, that is why no attention was paid to it in this document.

6.4 RGB and NIR Generation

For the final image generator the network SPADE 3.3 is used, SPADE is originally implemented in Pytorch [33], but for this project it was used as a base, a modified version in TensorFlow [34].

The original code generates a three layer image in the output, but in this case it was necessary to modify the network in a way that it works with a four layer image, three layers for the RGB (red, green and blue) and one layer for the NIR image. When evaluating the metrics it is also necessary to separate the output and have again the two images, so the metrics are evaluated separately, even though being generated together by the same network.

Listing 6.2. Grid search code

```
# Load dataset list
train_dataset, shape = load_dataset_list(
                directory = train_directory, type = 'mask')
test_dataset, _ = load_dataset_list(
                directory = test_directory, type = 'mask')

# Define the base grid search parameters
base = {'epochs': [300, 500],
        'latent_dim': [100, 200],
        'batch_size': [32, 64]}

# DCGAN
DCGAN = {'g_lr': [0.0002],
         'g_ld': [0.001],
         'g_beta_1': [0.5],
         'd_lr': [0.0002],
         'd_ld': [0.001],
         'd_beta_1': [0.5]}
DCGAN.update(base)

# WGANGP
WGANGP = {'g_lr': [0.0002],
          'c_lr': [0.0002],
          'n_critic': [5]}
WGANGP.update(base)

# Train
grid = gridSearch(train_dataset = train_dataset,
                  test_dataset = test_dataset,
                  shape = shape,
                  parameters = DCGAN)
grid.fit()
```

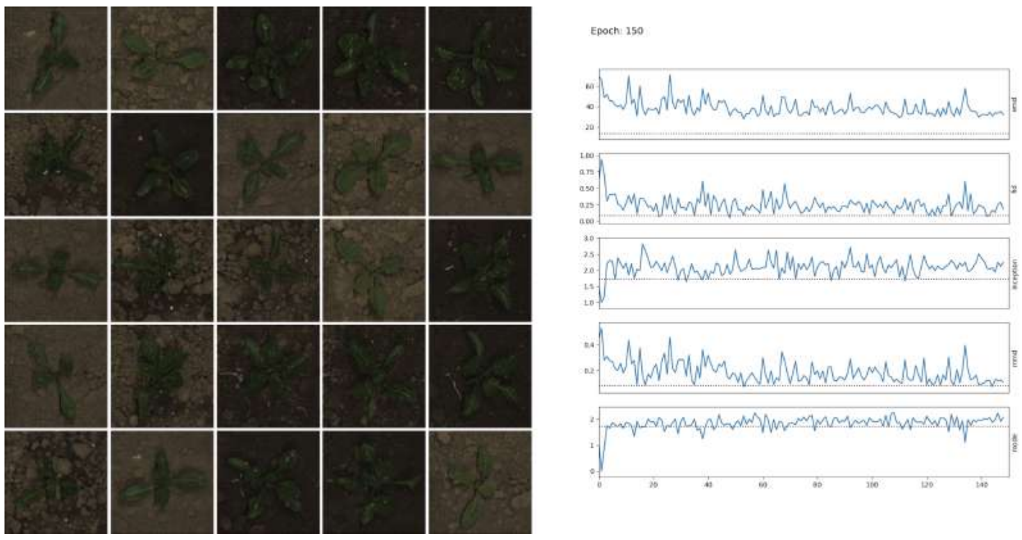

Figure 6.4. Gif example.

6.5 Gif Generator

The gif generator is something that seams to be only entertaining, but it turns out to be a very useful tool for humans to visually evaluate GANs. The frames of the gif are created at each epoch and the complete gif is created at the end of the training. It plots a sample of 25 images generated with the same seed with the current model of the network and evaluates the current model using the metrics.

Figure 6.4 presents one frame of the gif for the RGB model training. In the metrics plots, on the right side is presented the current epoch and the metrics evolution. The red line presented in the graph are calculated running the metrics on ten iterations calculating the metrics on two groups created with real images. The closer the metrics are from these lines, closer the images are from the real ones.

6.6 Semantic Segmentation

For the Semantic Segmentation, it was used an implementation of UNET Architecture [35], explained in Section 5.3. This UNET segmentation model uses RESNET-50 [36] encoder as the base model. For this evaluation, different tests were made, the first one using only the original dataset, another using the original augmented with the synthetic dataset and the last using only the synthetic dataset.

For the original dataset the number of training examples used was 9784, for the synthetic it was 15246 samples, and for the original with the synthetic the total number is 17407 (9784 from the original plus 7623 from the synthetic). The three training were performed for 5 epochs iterating throw the whole dataset with the same parameters.

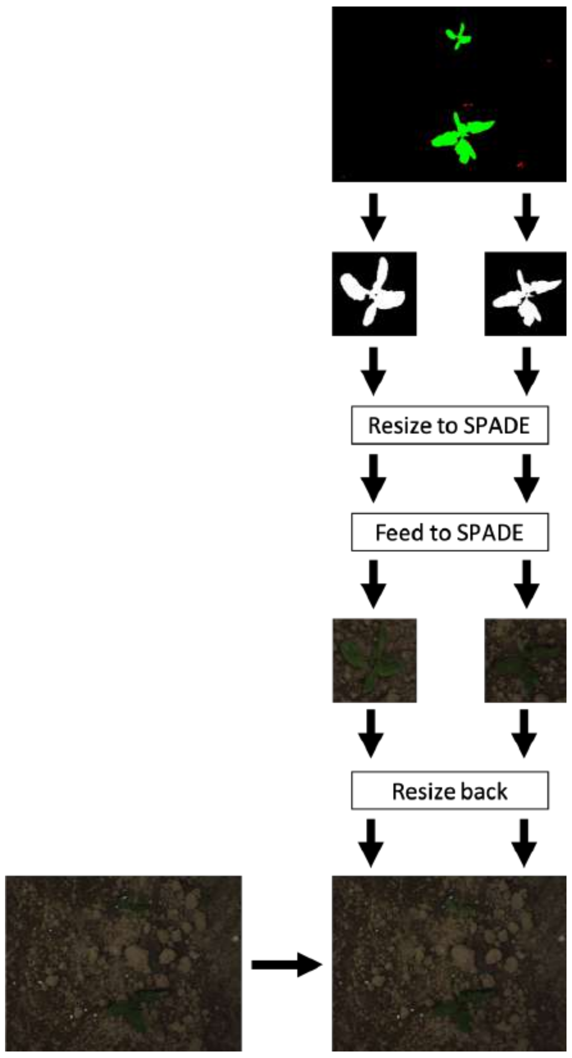

Figure 6.5. Dataset creation for segmentation training. First the crop masks are taken from the full mask, new RGB crops are generated from these masks and past back to the original image.

For this evaluation the ideal scenario would be to generate from scratch the scene of the field, with the soil, weeds and crops, however because there is only the network for the crops generation, the methodology needed to be changed a little bit. When generating the synthetic images, as shown in Figure 6.5, the process is first to get the crops mask from the full mask image, resized it for the SPADE network 256×256 pixels, generate the RGB image, resize again to the original size and paste on the original image. This technique is not optimal because some times it depends on increasing the final image to match the original mask, and when increasing the image it loses some details. The ideal would be to have trained generators on bigger images, for this case 512×512 pixels would be ideal.

Chapter 7

Results

This chapter will present the results of the training process, how the image quality improved over the training and how the metrics changed over time.

7.1 Mask Generation

The mask training seems to be the simplest problem, because the goal is to generate the mask directly from a random noise, however this is the one that presents more variables to be tested. For this first step there is no state-of-the-art model, each model presents different results depending on the application, and there are more possibilities of networks and parameters to test in order to find the one that presents the best results. As described in Section 6.3, only two networks were tried and DCGAN presented the best results.

One of the problems found with this model was the difficulty training a network able to generate black and white images that has an abrupt change in the border between the crop and the soil. Figure 7.1 shows how is the result training on a binary mask and on a blurred mask. It was discovered that with the blurred mask the network learns better what to do when switching from the crop to the soil. More examples with these two different training can be seen in Figure 7.2.

In figure 7.3 is shown the evolution of training using the same seed as input for different epochs. This network was trained for 300 epochs and it is possible to see that at the end the quality of the mask is very convincing to be real.

The metrics are shown in Graphs 7.4, it is possible to notice that values of the training are close to the desired value. In these graphs the red line represents the values when calculating the metric with two batches of real images.

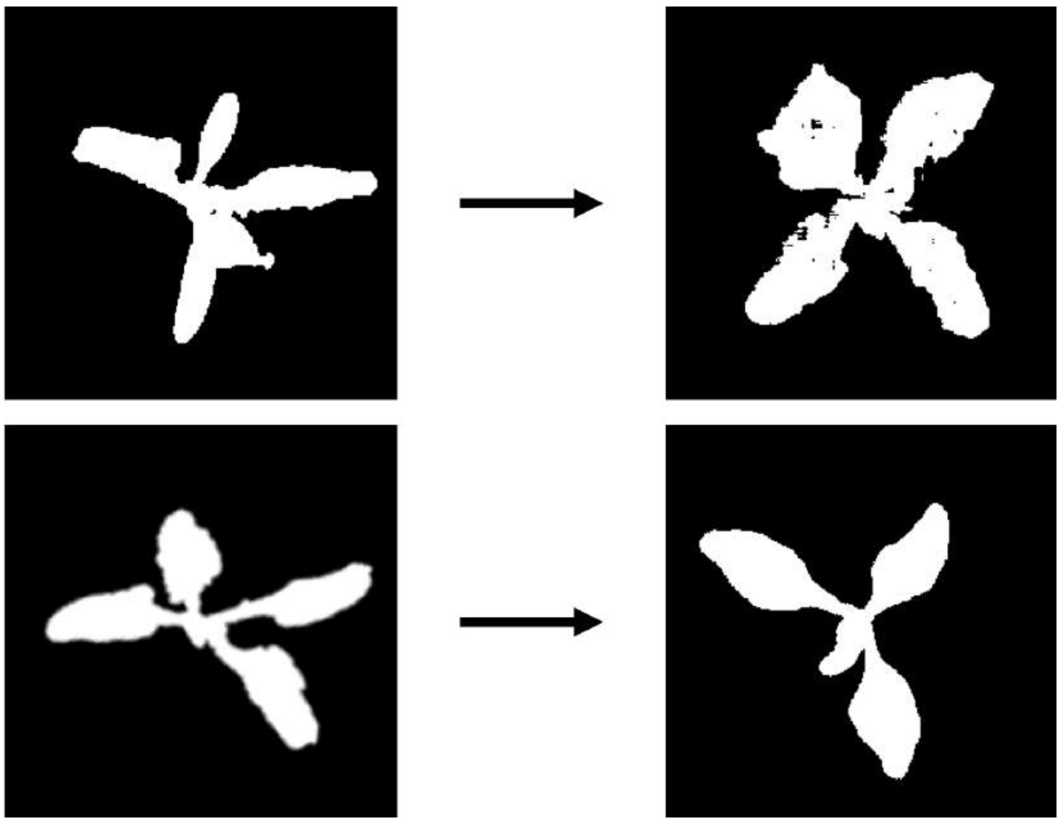

Figure 7.1. Comparison on the effect of training the network using blurred and not blurred masks.

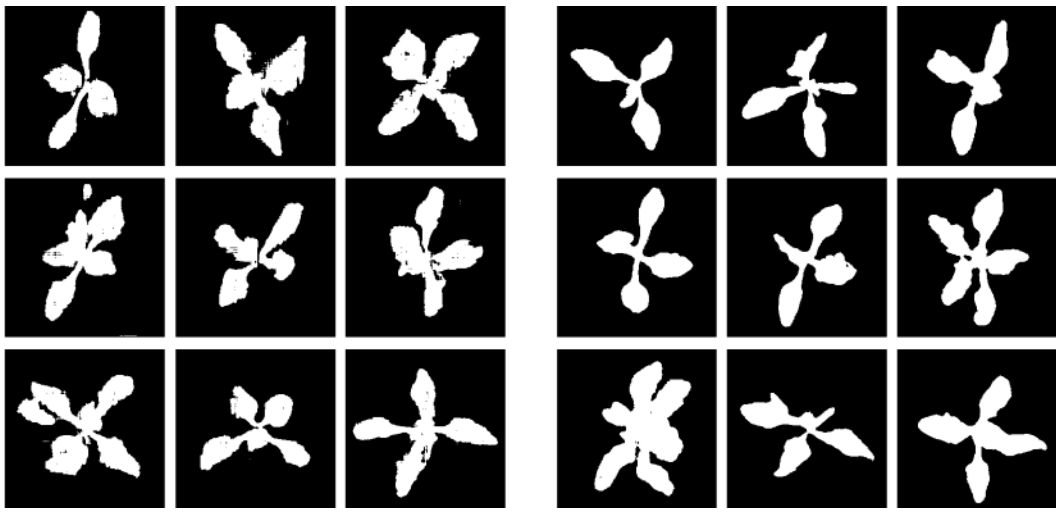

Figure 7.2. Image generation comparison with blurred and not blurred inputs, images on the left were generated with abrupt changes between crop and soil, and images on the right were trained using blurred images.

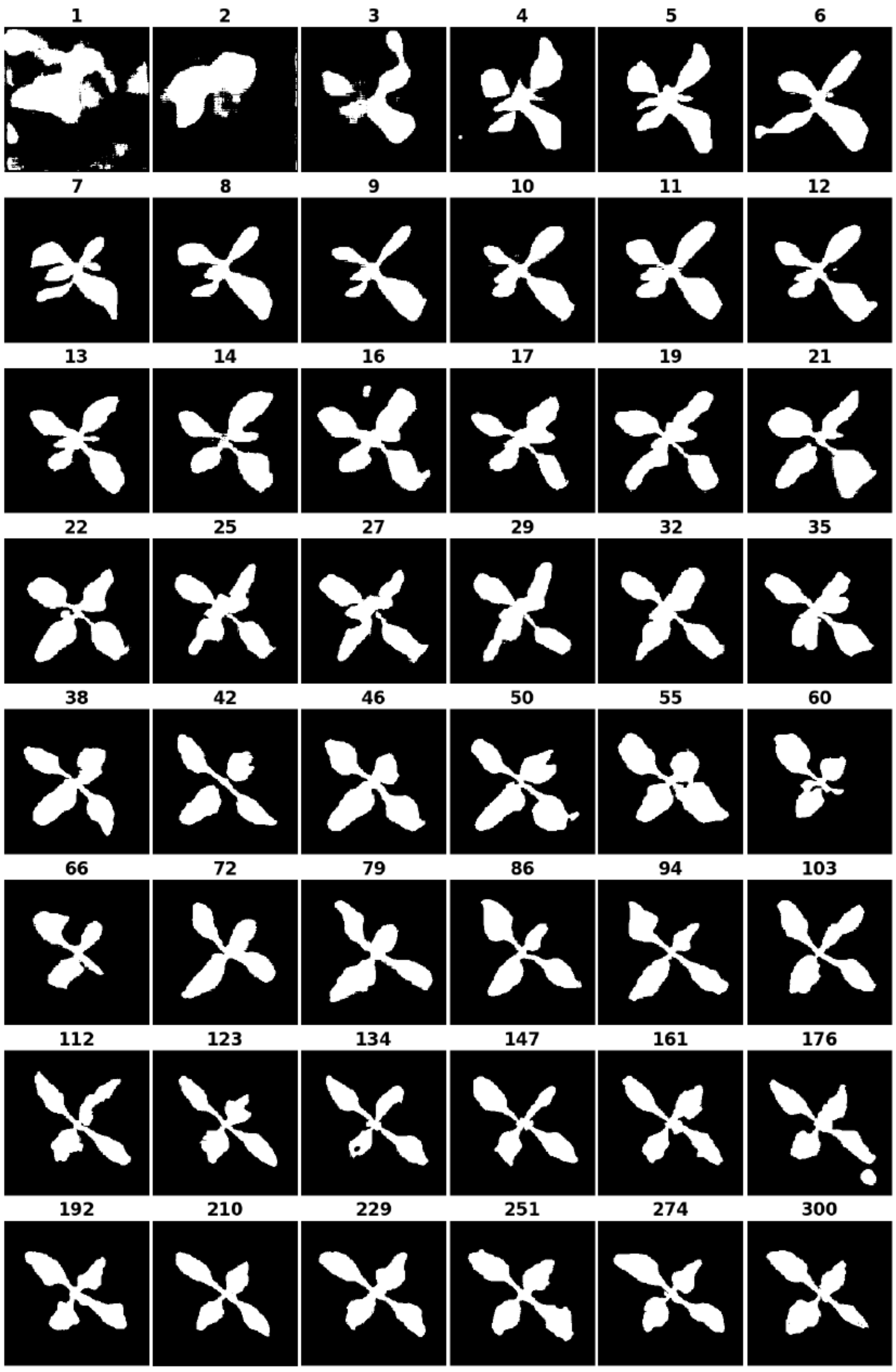

Figure 7.3. Mask generation, it shows the output of the same seed with different epochs (numbers above the images)

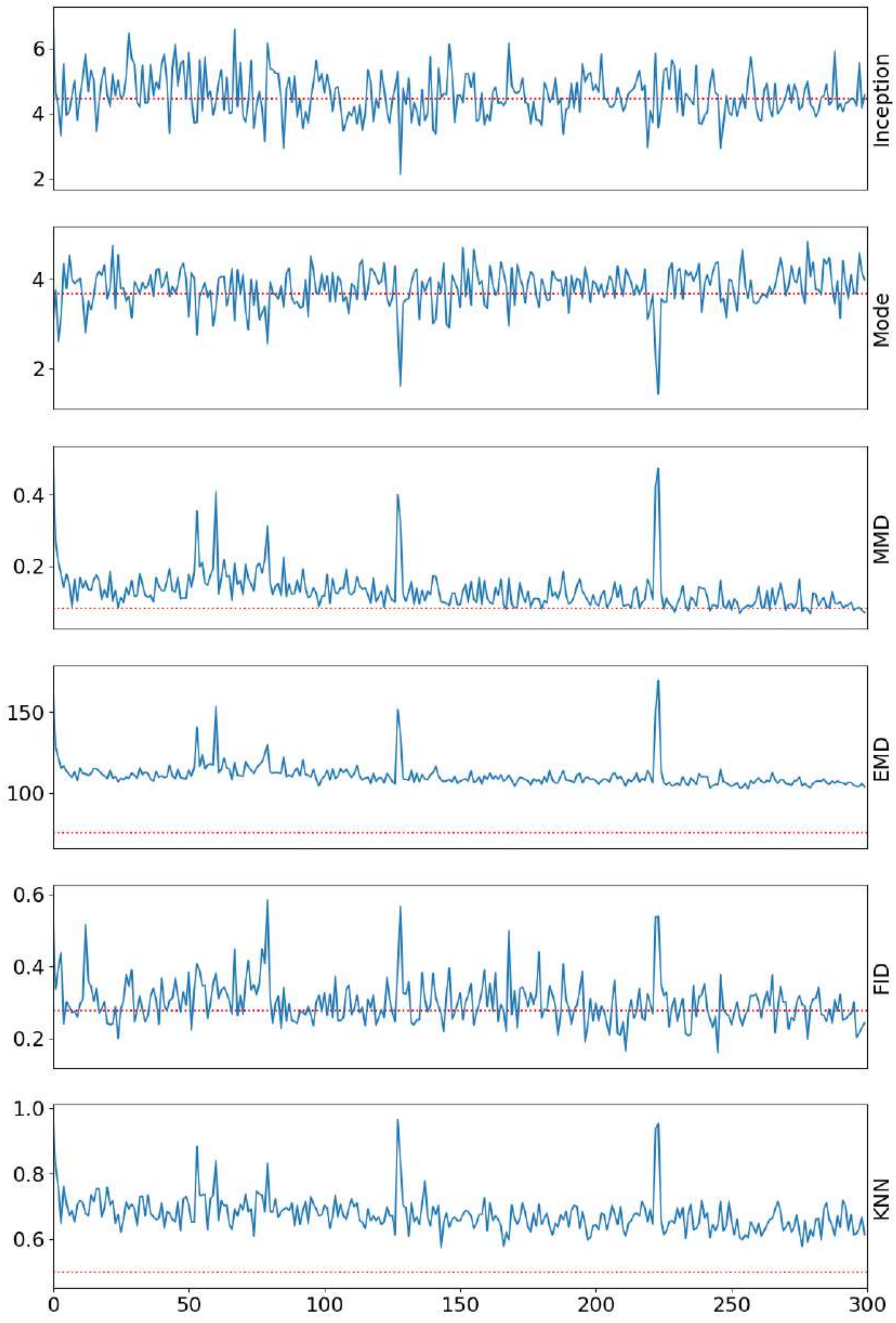

Figure 7.4. Metrics evolution over epochs for mask training.

7.2 RGB and NIR Generation

For the RGB and NIR images model, it was trained using the SPADE network [5] with a four layer image, allowing to generate the RGB and NIR image together, this way, the texture of the soil match between both images. Figure 7.5 and 7.7 shows the training evolution for both RGB and NIR images, in these images it is possible to notice that not only the crop generated have the same shape, but also the soil have the same texture.

Figures 7.6 and 7.8 shows the metrics evolution during the 300 epochs training. In these graphs it is possible to notice that all the metrics are converging to the red line, the values that were calculated when comparing real images with real images, meaning that the quality of the generated plants are approximate to the real ones.

Last, Figure 7.9 shows mask crops generated with different styles, the styles are presented on the column in the left and the masks in the top. From this figure it is possible to visualize that the network learned how to generate different styles, however the generated images do no match exactly the style desired, this may have happened because the time training was not enough and also the data was not well balanced between the styles, this way the network learned more the style that is more common in the dataset.

7.3 Segmentation

Besides the images seem to be realistic and all the metrics converge to the values of the real images, it is important to compare the effect of training another network with a dataset augmented with these images. Table 7.1 shows de IoU for each class, ground, weed, crop and also total IoU for the image.

It is possible to notice that the IoU increased drastically, 14.97% using the original dataset augmented with the synthetic one compared to using only the original dataset. Not only that, but using only synthetic dataset also showed a better performance when compared with only using the original one, an increase of 6.68%. The same effect is noticed when trained with a reduced number of original images augmented with synthetic images, the accuracy increased 5.51%, proving that it is possible to obtain a higher accuracy even with a smaller dataset of original images. Figure 7.10 shows one example of segmenting a random image from the test set using these three networks.

It is important to notice that by synthetic it means the substitution of all crops that are more than 50% inside the frame by synthetic ones. For the crops that are mostly out of the frame it was kept the original, this happened because it is necessary to have the root of the plant to center the mask for the synthetic image generation.

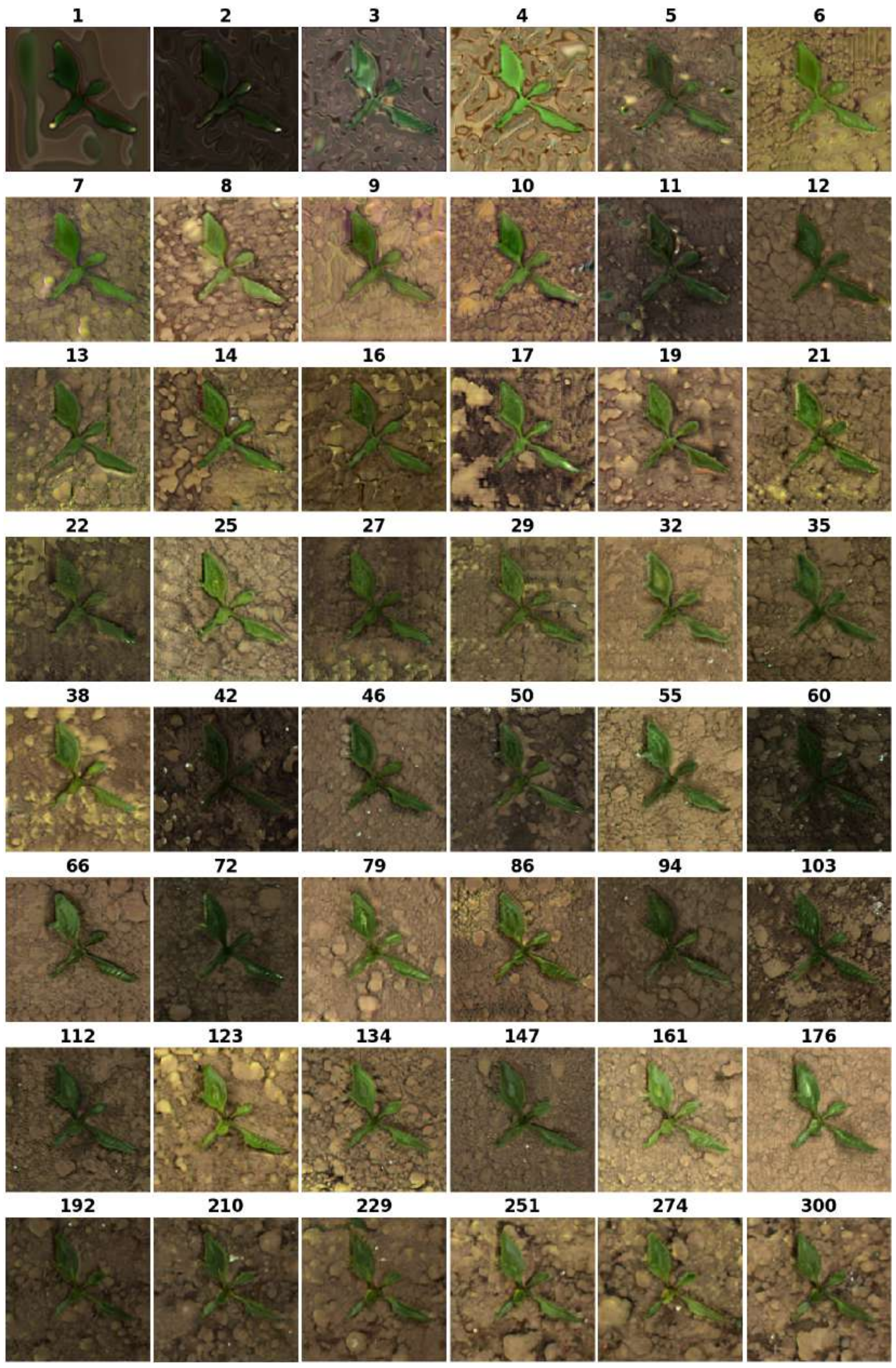

Figure 7.5. RGB generation, it shows the output of the same seed with different epochs (numbers above the images)

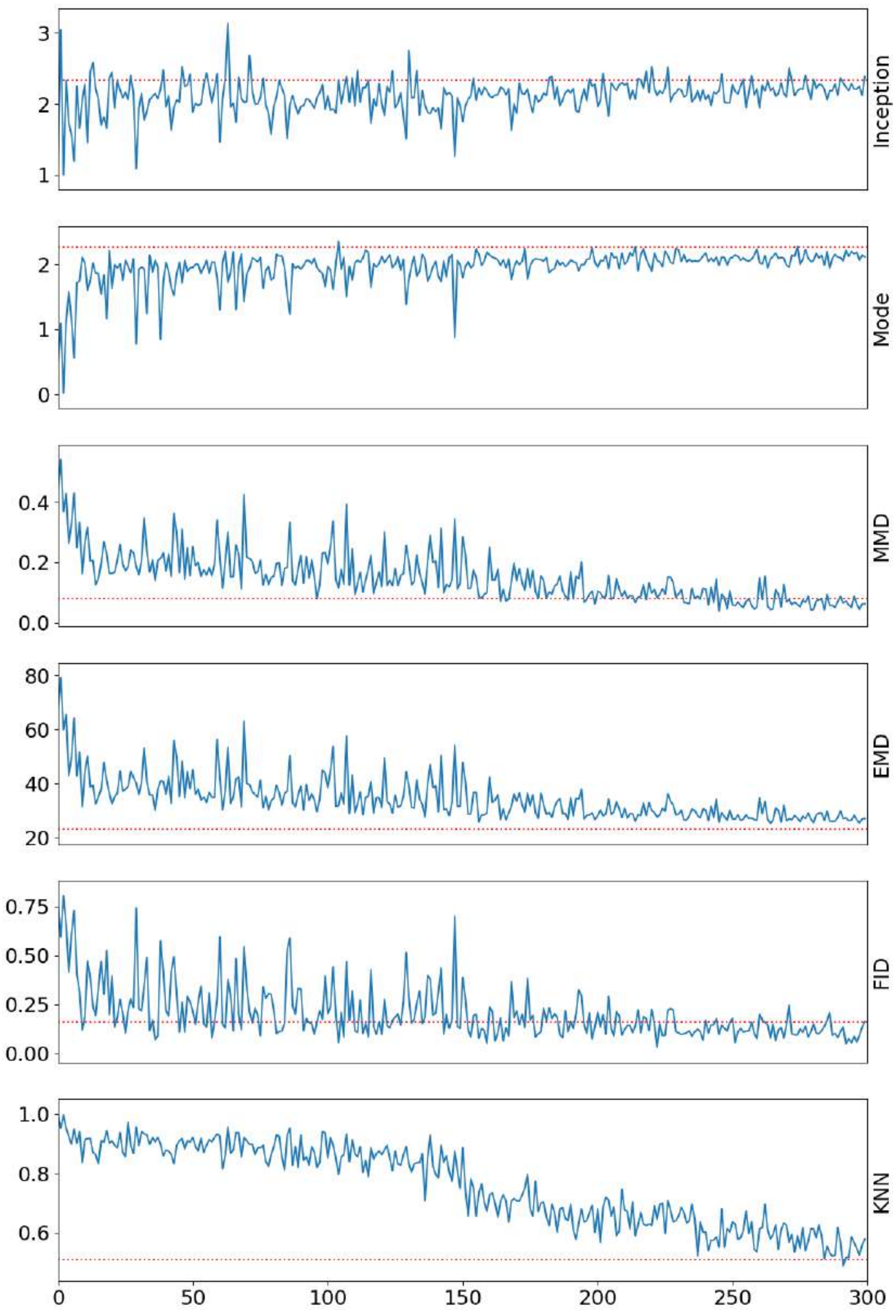

Figure 7.6. Metrics evolution over epochs for RGB training.

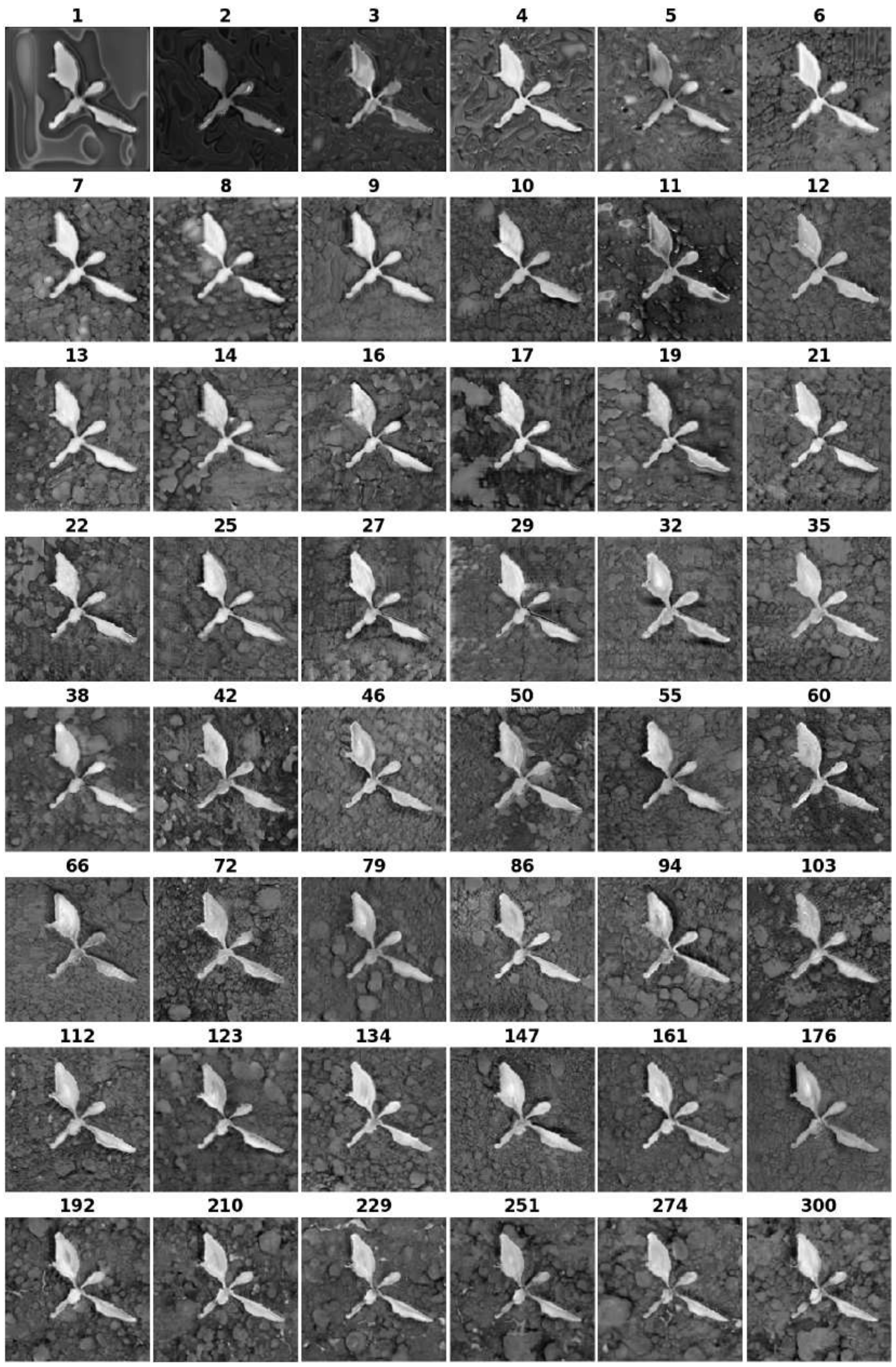

Figure 7.7. NIR generation, it shows the output of the same seed with different epochs (numbers above the images)

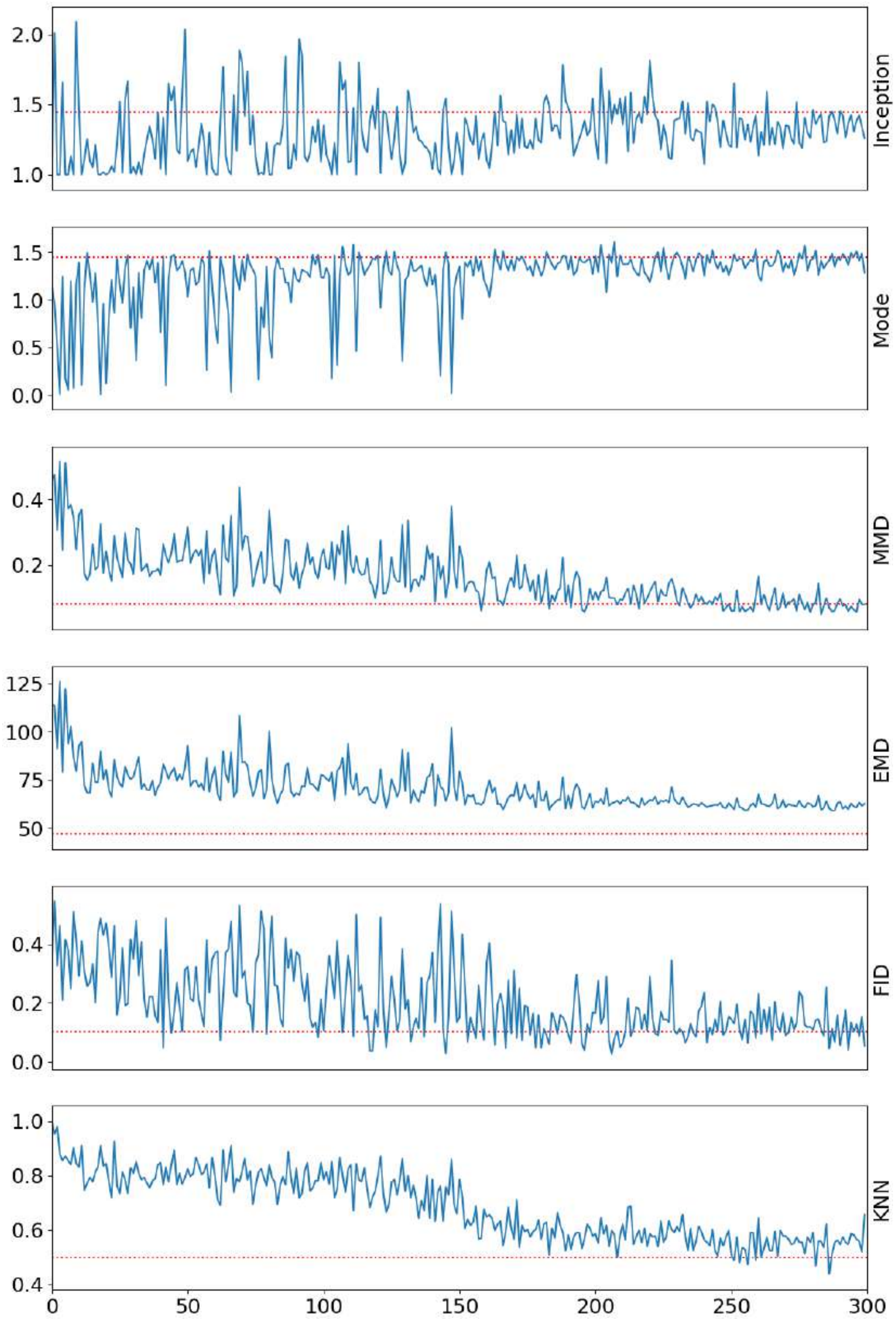

Figure 7.8. Metrics evolution over epochs for NIR training.

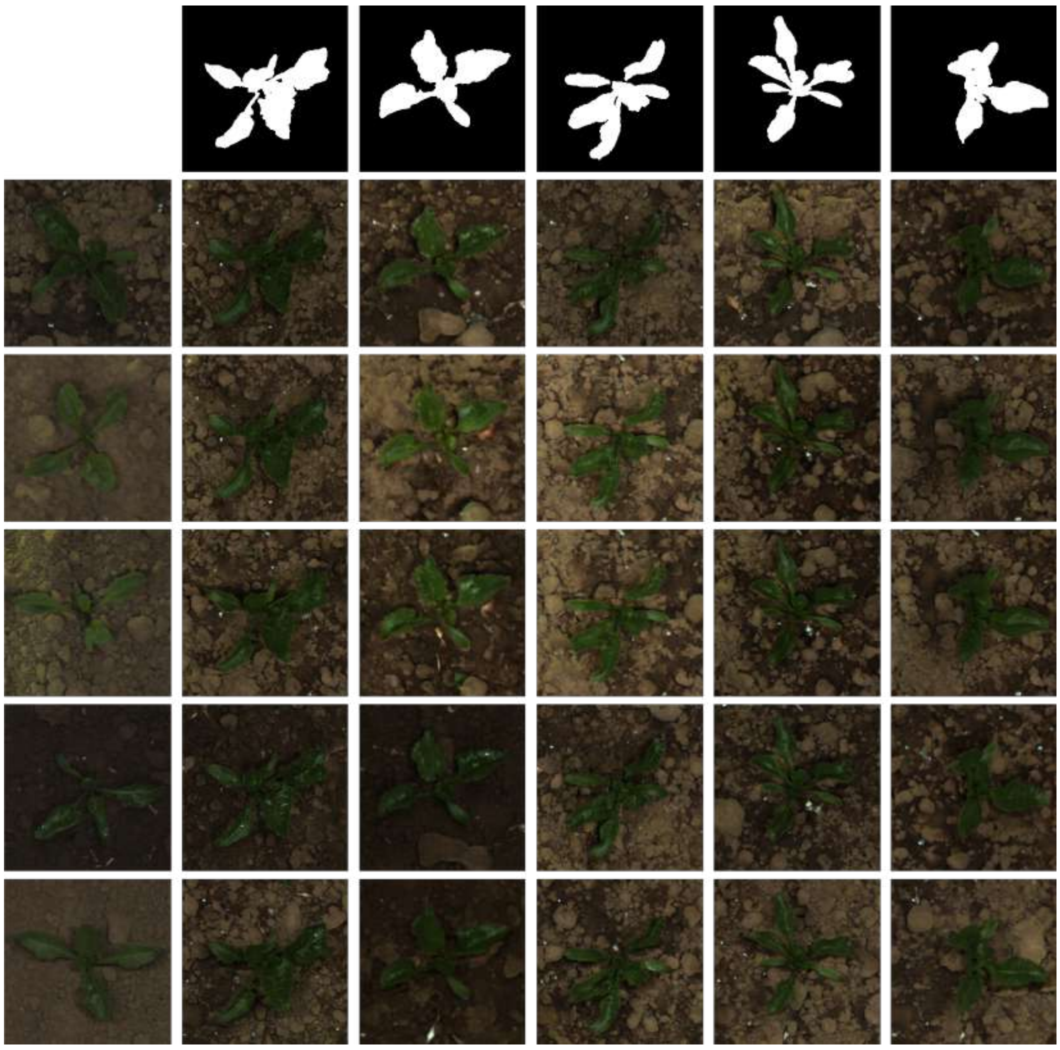
Figure 7.9. Crop comparison generated with different styles.

Number of Images	Ground	Weed	Crop	Total IoU
9783 Original	0.9896	0.1551	0.5121	0.5522
4898 Original + 10924 Synthetic	0.9574	0.1761	0.6882	0.6073
21844 Synthetic	0.9886	0.2359	0.6324	0.6190
9783 Original + 10924 Synthetic	**0.9898**	**0.3167**	**0.7993**	**0.7019**

Table 7.1. IoU segmentation results comparing training on three different dataset.

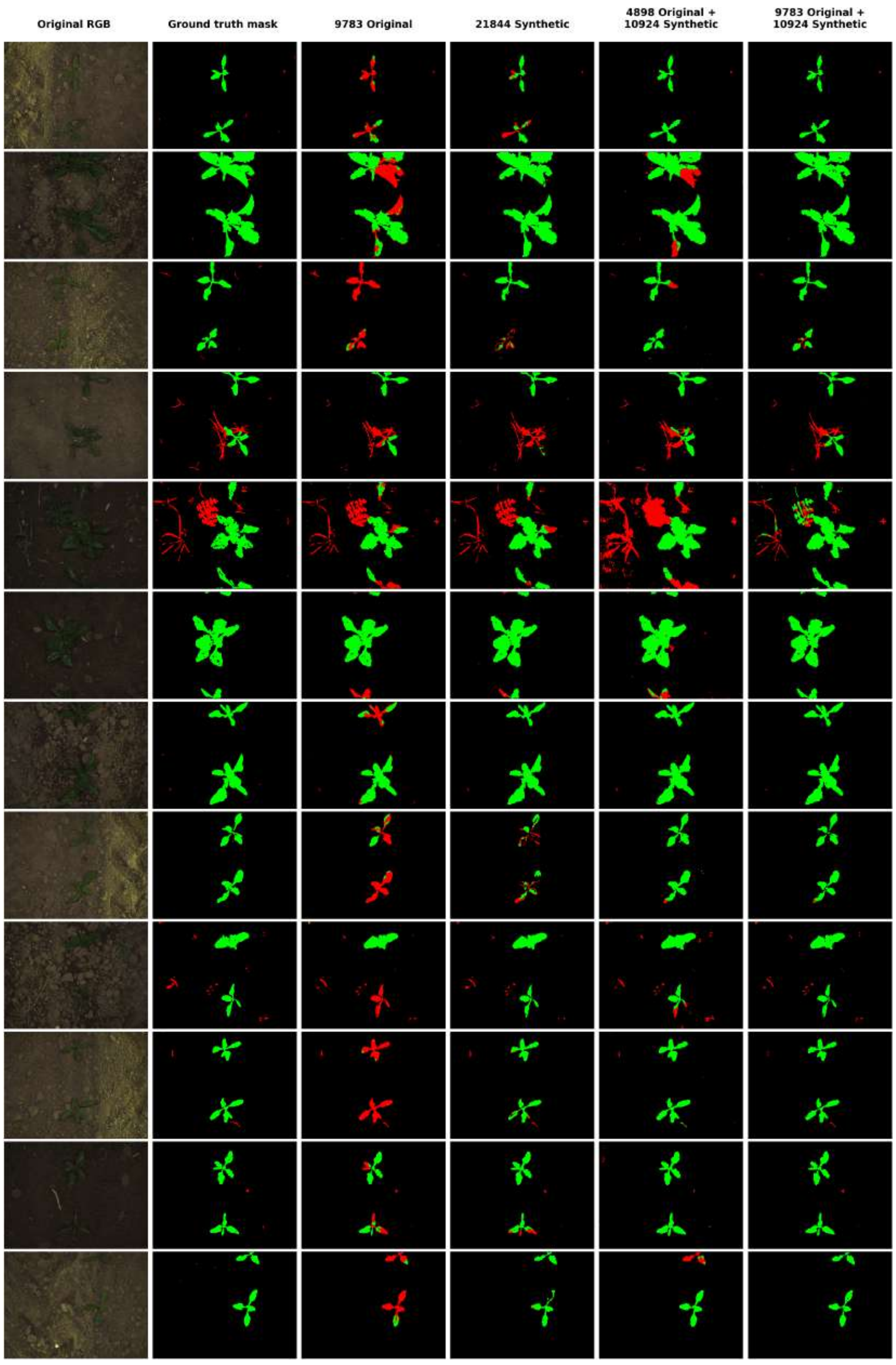
Original RGB
Ground truth mask
9783 Original
21844 Synthetic
4898 Original + 10924 Synthetic
9783 Original + 10924 Synthetic

Figure 7.10. Segmentation comparison training on different datasets.

Chapter 8

Conclusion

This work demonstrated the viability of using Generative Adversarial Networks pipeline for image augmentation and the results of using this augmented dataset compared with only using the original one for segmenting soil, crop and weed in an agricultural field. The full pipeline presented is capable of generating the semantic segmentation of the crops, the RGB and the NIR images as well as different ways to evaluate the quality of these synthetic data.

The results collected here prove that the series of networks are able to generate new plants different from the ones in the dataset, following the same data distribution, and realistic enough to improve the training of another semantic segmentation network.

It was shown that the metrics calculated on the synthetic images converge to values similar to the ones of the original dataset. The image quality of the generated images is also convincing to be real and when using this synthetic dataset with the original one for training a semantic segmentation network it was possible to obtain a difference of 14.97% compared to training only on the original dataset.

This way this approach proved to be indeed useful for training machine learning algorithms, saving time, money and avoiding manual labeling, becoming a vital technology for companies that do not have access to a big budget or tools for data collection.

Chapter 9

Future Work

This project took a long time to be developed and the results achieved are impressive, however, there are several parts of this project that could be improved in order to generate better images and achieve even better results evaluating the use of the synthetic dataset.

In respect to the dataset creation, the agricultural robot dataset [1] used for this project gives images of the field in different days, so it is possible to divide the plants by size (age). When selecting the days to use, only the last days were selected, in order to allow generating bigger images, but even when getting only the last days it is possible to notice difference between older plants, where the leaves are bigger and rougher, and newer plants, with smaller and smoother leaves. It would be interesting to try in the future to separate the dataset between days and train a conditional GAN, passing the age of the plant as the condition for the mask generation.

Besides this, some other improvements could also be done in respect to the mask generation, the code is structure to allow automatic parameters grid search with the help of the metrics implemented, but this was not done because the lack of computational power to process all this data. Only two networks were tested in this phase and DCGAN presented reasonably good results testing only a few parameters, but it would be advised to try different networks and do a proper parameter grid search for each network tested.

For the final image generation, the results are really impressing, the network used is the state-of-the-art, but it would be interesting to train for more epochs in order to improve the quality of the image when giving a style as an input. Also, it would be interesting to try doing transfer learning from a network already trained in a bigger or in another plants dataset.

Another improvement that would be interesting and was the goal of this project is to generate bigger images, 512×512 pixels, but it was also not possible because the number of parameters to train would be much higher and the graphics card used could not handle that.

For the evaluation, it could be interesting to find a weed dataset and train another set of networks to learn how to generate also weeds, with this, it would be possible to generate a completely synthetic field and use it for better evaluation using not only segmentation but also classification between crops and weeds.

Last, for the semantic segmentation tested here, some other networks could be tried and the results compared. Besides that, train a network that generates the output in the same size as the input, this way it would be possible to have a better detailed segmentation map of the field.

Bibliography

[1] N. Chebrolu, P. Lottes, A. Schaefer, W. Winterhalter, W. Burgard, and C. Stachniss, "Agricultural robot dataset for plant classification, localization and mapping on sugar beet fields," *The International Journal of Robotics Research*, 2017.

[2] M. V. Giuffrida, H. Scharr, and S. A. Tsaftaris, "ARIGAN: synthetic arabidopsis plants using generative adversarial network," *CoRR*, vol. abs/1709.00938, 2017.

[3] J. T. Guibas, T. S. Virdi, and P. S. Li, "Synthetic medical images from dual generative adversarial networks," *CoRR*, vol. abs/1709.01872, 2017.

[4] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[5] T. Park, M. Liu, T. Wang, and J. Zhu, "Semantic image synthesis with spatially-adaptive normalization," *CoRR*, vol. abs/1903.07291, 2019.

[6] "Deepfield robotics." `https://www.deepfield-robotics.com/en/`.

[7] G. Avola, S. F. Di Gennaro, C. Cantini, E. Riggi, F. Muratore, C. Tornambè, and A. Matese, "Remotely sensed vegetation indices to discriminate field-grown olive cultivars," *Remote Sensing*, vol. 11, no. 10, 2019.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[9] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[10] C. O. Alexander Mordvintsev and M. Tyka, "Inceptionism: Going deeper into neural networks." `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`. Accessed: 2015-06-17.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[12] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel* (J. Fürnkranz and T. Joachims, eds.), pp. 807–814, Omnipress, 2010.

[13] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CoRR*, vol. abs/1611.07004, 2016.

[14] T. Wang, M. Liu, J. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional gans," *CoRR*, vol. abs/1711.11585, 2017.

[15] L. M. Mescheder, "On the convergence properties of GAN training," *CoRR*, vol. abs/1801.04406, 2018.

[16] T. Miyato and M. Koyama, "Cgans with projection discriminator," *CoRR*, vol. abs/1802.05637, 2018.

[17] Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, and K. Q. Weinberger, "An empirical study on evaluation metrics of generative adversarial networks," *CoRR*, vol. abs/1806.07755, 2018.

[18] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *CoRR*, vol. abs/1606.03498, 2016.

[19] T. Che, Y. Li, A. P. Jacob, Y. Bengio, and W. Li, "Mode regularized generative adversarial networks," *CoRR*, vol. abs/1612.02136, 2016.

[20] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample problem," *CoRR*, vol. abs/0805.2368, 2008.

[21] I. Olkin and F. Pukelsheim, "The distance between two random vectors with given dispersion matrices," *Linear Algebra and its Applications*, vol. 48, pp. 257–263, 1982.

[22] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a nash equilibrium," *CoRR*, vol. abs/1706.08500, 2017.

[23] D. Lopez-Paz and M. Oquab, "Revisiting classifier two-sample tests," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[26] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[27] "Python." `https://www.python.org`.

[28] "Tensorflow." `https://www.tensorflow.org`.

[29] "Synthetic plants." `https://github.com/ibiscp/Synthetic-Plants`.

[30] "Gan-metrics." `https://github.com/xuqiantong/GAN-Metrics`.

[31] "Pytorch." `https://pytorch.org`.

[32] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *CoRR*, vol. abs/1704.00028, 2017.

[33] "Spade pytorch." `https://github.com/NVlabs/SPADE`.

[34] "Spade tensorflow." `https://github.com/taki0112/SPADE-Tensorflow`.

[35] "Image segmentation keras." `https://github.com/divamgupta/image-segmentation-keras`.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.