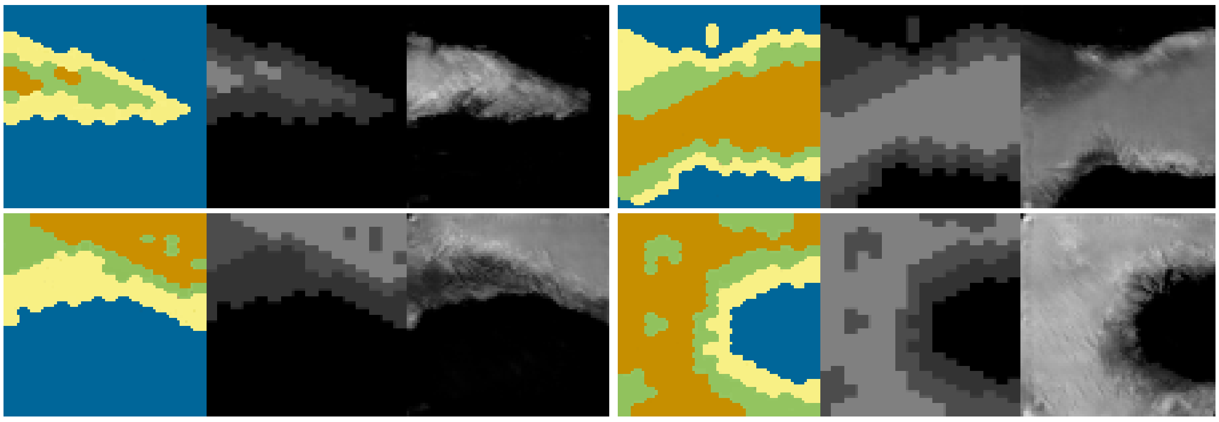


Dual Critic Conditional Wasserstein GAN For Height-Map Generation

Nuno Miguel Ramos

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Pedro Alexandre Simões dos Santos
Prof. João Miguel de Sousa de Assis Dias

Examination Committee

Chairperson: Prof. António Manuel Ferreira Rito da Silva
Supervisor: Prof. Pedro Alexandre Simões dos Santos
Member of the Committee: Prof. Francisco António Chaves Saraiva de Melo

November 2022

Acknowledgments

I would like to start by thanking my family - you made me who I am and without you this work would not only be impossible, it would never even have been pursued.

I would like to thank my friends and colleagues for all the moments, good and bad, we shared, I wouldn't have gotten here without you.

I would also like to acknowledge the professors who oriented me through this dissertation: Prof. Pedro Santos and Prof. João Dias, for their insight, knowledge, and support, which went above and beyond, and for which I am thankful.

Lastly, I want to thank this work's predecessor, Vasco Nunes, for all the help provided: others in your position would have ignored my asking for help, but you were always there for all the questions and assistance I required.

From the bottom of my heart: Thank you.

Abstract

Traditionally, video-game maps are either made by hand, which is a very inefficient process requiring many man-hours, or made using Procedural Content Generation (PCG) techniques, which rely on a predetermined algorithm to generate every feature of the map. This approach is flawed in a multitude of ways: creating the algorithm is an arduous process, the results lack realism and it's hard to create more complex geographical structures, such as bays, peninsulas, or diverse archipelagos. More recent studies have tried an approach using Deep Learning algorithms, which have their own limitations. Most importantly, these algorithms take away the creative freedom of the designers. To circumvent this problem we propose a system that transforms low fidelity sketches into realistic height-maps through a Deep Learning model we call the Dual Critic Conditional Wasserstein Generative Adversarial Network (DCCWGAN), thus providing high visual quality without removing control from the user.

Keywords

Height-map; Deep Learning; Image-to-Image Translation; GAN; Conditional GAN.

Resumo

Tradicionalmente os mapas para videojogos são feitos à mão, um processo que é ineficiente e custoso em termos laborais. A outra forma comum de fazer estes mapas é através de métodos algorítmicos, com técnicas como Procedural Content Generation (PCG), esta abordagem, no entanto tem múltiplas desvantagens: criar o algoritmo que gere estes mapas também é um processo custoso, os resultados são, comparativamente, pouco realísticos e é especialmente díficil criar estruturas geográficas mais complexas, tais como baías, penínsulas ou arquipélagos. Recentemente, alguns estudos focaram-se em usar técnicas de Deep Learning para reproduzir as caraterísticas geográficas presentes na Terra. Esta abordagem, no entanto, tem também as suas desvantagens, principalmente, o facto de retirar controlo ao designer, que deixa de conseguir especificar o conteúdo presente no mapa. Para remediar esta desvantagem propomos um sistema que transforma rascunhos de baixa fidelidade em mapas realísticos através de um modelo de Deep Learning que chamamos de Dual Critic Conditional Wasserstein Generative Adversarial Network (DCCWGAN), proporcionando desta forma resultados de boa qualidade visual sem sacrificar o controlo do utilizador.

Palavras Chave

Mapa de alturas; Aprendizagem; Tradução Imagem para Imagem; GAN; GAN Condicional;

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ANN	Artificial Neural Network
cGAN	Conditional Generative Artificial Network
CNN	Convolutional Neural Network
DCGAN	Deep Convolutional Generative Artificial Network
DCCWGAN	Dual Critic Conditional Wasserstein Generative Adversarial Network
GAN	Generative Adversarial Network
IMR	Intermediate Map Representation
PCG	Procedural Content Generation
ProgGAN	Progressively Growing Generative Adversarial Network
VAE	Variational Auto-Encoder
WGAN	Wasserstein Generative Adversarial Network

1 Introduction

Contents

1.1 Motivation

Maps are a crucial part of most video-games: in exploration games different areas can provide interesting challenges for the player, in strategy games more defensible geography can be considered a critical asset. If a video-game contains a map it is a reasonable assumption that the quality of the player experience is somewhat tied to the quality of this map.

In the video-games industry maps are made in one of two ways: either by hand, which is a very time-consuming task, whose results depend both on the prowess of the designer, who designs the challenges present in the map, and the artist, whose job is to implement those ideas; the other method is to procedurally generate those maps, which involves designing an algorithm to create a map from a set of parameters and random values; this approach has it's own problems, namely that it's results have sub-par quality and take control away from the designer, who, for the most part can no longer specify exactly which geographic formation appears where. The disadvantages incurred by these procedural content generation algorithms mean that they are used mostly for tile-based maps, or maps that are, in practice, infinite, meaning that it's impossible for them to be hand-crafted. Recent studies [Nunes et al., 2022] have tried to overcome the limitations of traditional Procedural Content Generation (PCG) techniques by implementing machine learning algorithms, using real-world geographic information to train networks created for this specific purpose. While this approach produced promising results it has a fundamental flaw: similarly to PCG techniques, it removes control from the developers. In order to create a level with a specific layout a developer would have to sieve through a large number of images generated by the network until the right one was found, and even then it could differ from the initial vision.

While there is a big focus in creating more and more realistic maps there is a variable that is not visible in the map itself, which is how closely the final result resembles what the designers imagined, and how much work must be put into it to reach that level of similarity. Our vision for the ideal way in which maps are created would be a way for the designer to be able to quickly create a low-fidelity sketch, from which a realistic map would be created. This would keep the level of realism and reduce the work necessary from the designer without sacrificing creative freedom.

1.2 Problem

The problem addressed by this thesis is the following:

"How to create realistic maps from low fidelity sketches?"

Realistic as a measure of how similar they appear to real-world geography. The process should also create a map similar to what's pictured in the low fidelity sketch, and allow a wide range of features.

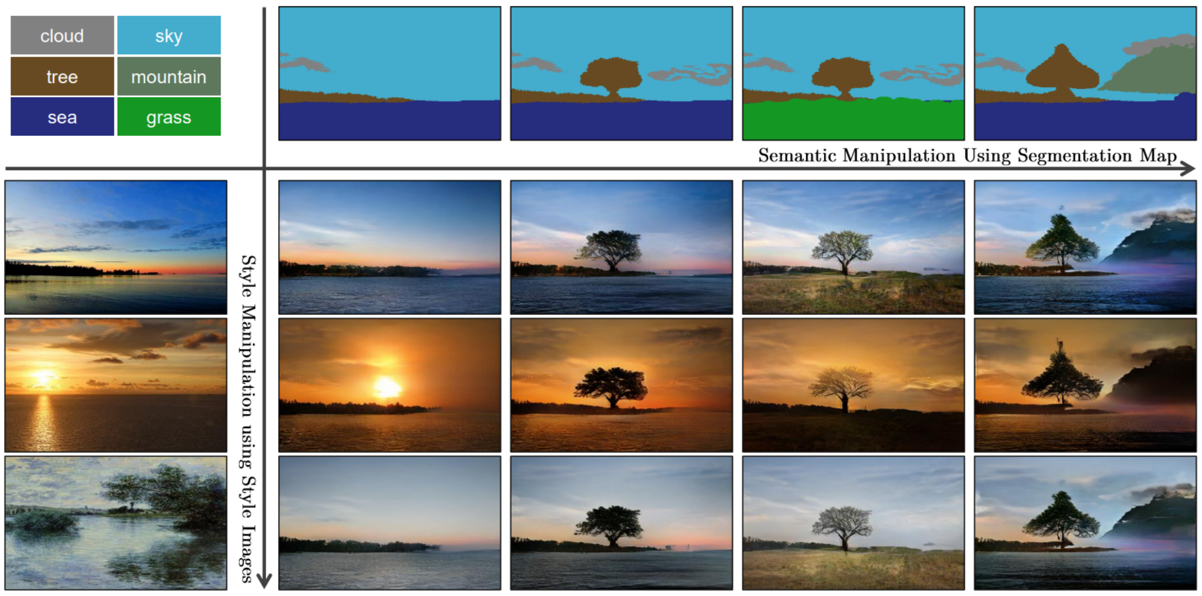

cloud
sky
tree
mountain
sea
grass
Semantic Manipulation Using Segmentation Map
Style Manipulation using Style Images

Figure 1.1: Tool that creates realistic images from low-level sketches. Top row contains the rough sketch while left column contains reference style. Image from [Park et al., 2019].

1.3 Hypothesis

Map generation for video-games is an area that has not yet been solved entirely. Recently, researchers have been trying to generate maps using machine learning algorithms. This research is still in it's infancy, but shows great promise, especially in regards to the visual quality of the maps generated. Where this new research falls short is in regard to the developer experience, which involves the developer giving up some measure of control of the process.

There have also been many recent breakthroughs in using machine learning algorithms for image generation [Brock et al., 2019, Karras et al., 2020, Miyato et al., 2018] and image-to-image translation [Isola et al., 2017, Park et al., 2019, Richardson et al., 2021, Wang et al., 2018], particularly in transforming low-level images into realistic depictions of the same content. Coupled with the fact that machine learning techniques themselves have also been in a steady state of evolution we believe that it is possible to leverage this technology to develop a tool that allows developers to draw a rough sketch of a map that will be transformed by deep learning model into a realistic and visually appealing version of the terrain depicted in the sketch, thus maintaining the visual quality while reducing effort without having to sacrifice control.

1.4 Contributions

The work done for this thesis contributes in a new paradigm for generating height-maps, not only through the way the maps are generated, but also on the workflow permitted by our system. We also contribute to existing knowledge by proposing a more advanced version of a conditional Generative Adversarial Network (GAN), using two critics, both of which based on the Critic of the Wasserstein GAN [Arjovsky et al., 2017], instead of the original GAN's Discriminator. This new Deep Learning model was thus named the Dual Critic Conditional Wasserstein Generative Adversarial Network (DCCWGAN). We have also implemented the necessary tools to test this proposed workflow in its entirety, including a script to translate from an existing hex map creating software into the format used by our system. We also offer an analysis of the results obtained, both through empirical observation of the conversion, but also of the realism of the maps generated, through user testing.

1.5 Organization of the Document

This document is divided into six chapters, we will start by giving the necessary background knowledge to fully understand both the problem and our suggested solution. After these theoretical bases have been covered we will explain in detail the architecture of our proposed solution, and important notes regarding its implementation. We will then analyse the results acquired, both in how realistic they appear and how well they represent the given input. We will conclude by pondering the impact this approach could have on the industry and what work is left to do.

2

Related Work

Contents

2.1 Background

Over the course of this section we will explain all the relevant concepts necessary to fully understand this research. We will start by discussing basic concepts regarding video game maps, then moving to traditional PCG techniques and their limitations. We will then describe various Machine Learning algorithms that are the most appropriate to implement this system, including Generative Adversarial Networks (GAN), Convolutional Networks and Variational Autoencoders (Variational Auto-Encoder (VAE)), all of which have gained more widespread use in the recent past.

2.1.1 Hexagonal Maps

Hexagonal maps are very common within video-games and board games alike. Typically each tile of the map, commonly referred to as a hex is of a given terrain type, and can also contain additional modifiers. A hex's terrain type usually includes biomes such as "forest", "desert", "ocean", "snow", etc... while the additional modifiers dictate whether there are trees or other vegetation, or structures such as castles.

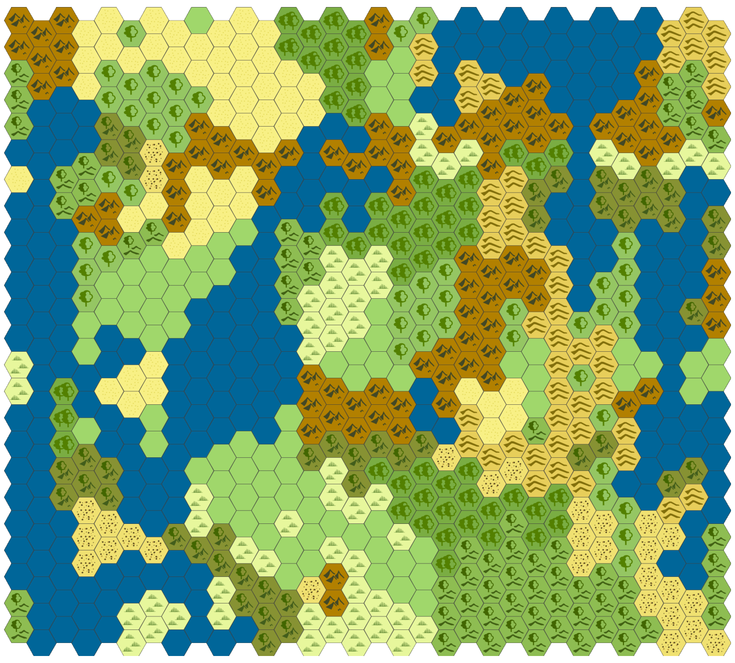


Figure 2.1: Example of a hex map, generated from `https://www.hexographer.com` .

Figure 2.1 contains an example of a hex map. It should be noted that a major reason for the popularity of these types of maps is both their ease of reading and the simplicity with which they can be created. The tool used to create Figure 2.1 is an example of a free editor for hexagonal maps, allowing maps to be created and edited within minutes.

2.1.2 Video-game maps

Video-game maps are made up of multiple different components, or layers [Millington, 2019]. The more notable inclusions in this category are:

- Height-map: contains the height of the terrain.
- Hydraulic map: contains the water in the terrain, rivers, lakes, etc...
- Vegetation map: determines which areas have trees and other vegetation.
- Biome map: determines to which biome each region belongs to.

The above is list is by no means comprehensive, different games have different requirements and may use other types of maps to satisfy those requirements. Of all the components listed above the most complex is by far the height-map, this makes it the hardest to recreate in a believable way, additionally, elevation tends to play a large role in how the player interacts with the game - high ground tends to be more valuable as it is easier to defend or rocky mountains may provide a direction unlikely to be attacked through. This creates a difficulty for the developers and designers of the game: on one hand the map must appear realistic, on the other it must provide interesting gameplay; a realistic map with no interesting features will oversimplify the game, while a map with varied features but that appears unrealistic will ruin the players' immersions.

2.1.3 Traditional PCG techniques

PCG techniques are by far the most common techniques used to generate maps in video games. These techniques function by generating pseudo-random numbers which are then put through complex functions that create the resulting map. The most common algorithm for generating these pseudo-random numbers is Perlin noise, which, unlike random noise, guarantees some level of coherence between the numbers generated.

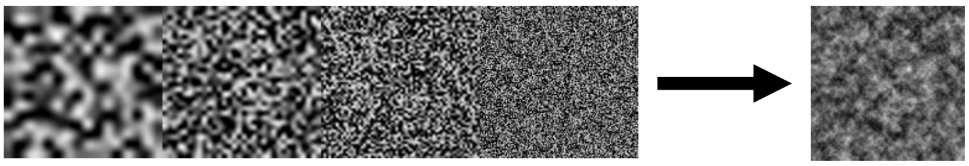


Figure 2.2: Different octaves of Perlin noise added together. Image from [Johanson, 2004].

In the particular case of creating elevation maps a common technique is to combine Perlin noise maps at different scales (usually each twice as big as the last), called octaves in such a way as to decrease the influence of smaller octaves. One can imagine that larger octaves with a large impact on the final result represent hills and valleys, while smaller octaves represent boulders or similar, smaller objects.

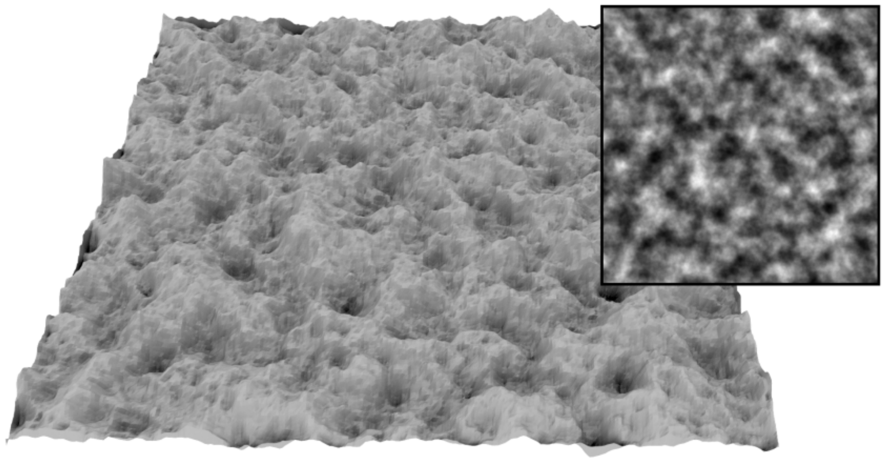


Figure 2.3: Height map created from Perlin noise, combining different octaves. Image from [Millington, 2019].

To create a map from the noise simply take the pixel value and map that to the height of that same coordinate, white pixels (i.e. with higher value) equate to higher elevation. Doing this for every coordinate creates a height-map as seen in Figure 2.3.

2.1.4 Deep Learning

Most programming is done explicitly by the programmer, even when a program takes a given input and returns an output based on the information it was given this mapping has to be coded step by step by the programmer; this approach has drawbacks when it is difficult to translate the problem into code: some things are easy for humans, and should be possible for computers to do, but it's not trivial to translate them into code, one such example is object recognition, which is intuitive for humans, there's no reason a computer shouldn't be able to recognize objects within a picture as it contains all necessary information, yet, it is in practice impossible to program this behaviour explicitly. To program computers to do this sort of tasks the most common approach is by using a technique known as Machine Learning [Pedrycz and Chen, 2020].

Machine Learning techniques all have one thing in common, they have a process of learning; instead of programming each step individually, the programmer implements an algorithm that will learn the problem through examples. One of the most common of these structures, and the building block to other, more complex architectures is the Artificial Neural Network (ANN).

$$o = \varphi\left(\sum_{j=0}^{m} w_j x_j\right) \tag{2.1}$$

ANN's are made up of neurons organized in layers, depending on the number of layers the network may be considered a deep neural network. Formula 2.1 states that the output of a neuron (o) is the output of the activation function (φ) of the sum of all inputs (x) multiplied by a weight (w) specific to each input. A fine point that may be overlooked from this expression is that it also contains another term, called the bias, which is added to the product of x_j and w_j. In practice, this term is prepended to w, and the value 1 is prepended to x, giving the same end result, but simplifying the training loop. In neural networks, individual neurons are arranged in sequential layers, (typically) with all neurons in a layer sharing the same input but different weights and connecting the outputs of neurons from one layer as the input of the next, such a structure can be seen in Figure 2.4. A key detail that's worth noting is that the choice of activation function can be the difference between a successful model and an unsuccessful one. Activation functions change the network from a simple sequential vector product and bias addition to non-linear models capable of much more complex behaviour.

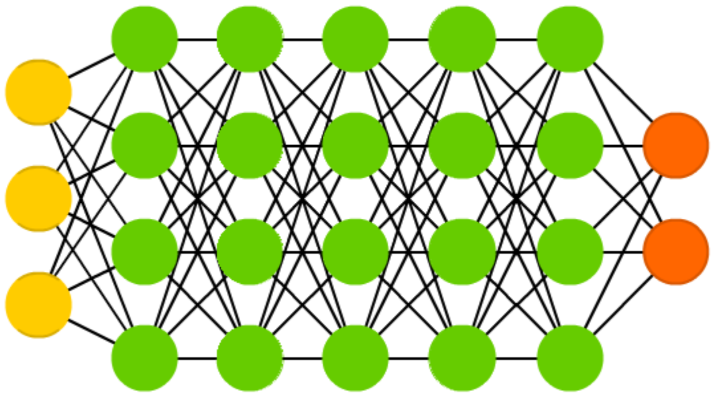


Figure 2.4: Structure of a Deep Feed-forward Network. Yellow neurons are input cells, green are hidden cells and orange are output cells. Note that both the size of the layers and their number may vary depending on application. Image adapted from [Veen, 2016].

For the network to learn a behaviour it must be trained, which is done most commonly through Backpropagation: the network is given an example and computes the output, this output is then compared to the real output. If the results match nothing happens, but if they don't match then the error of the network is calculated, which, together with the gradient of a loss function with respect to the weights is used to backpropagate a correction through the network, adjusting the weights. This process is repeated many times, either for a set number of passes through all examples, called epochs, or until it is noted that the network has reached a steady state, called convergence.

2.1.5 Convolutional Network

There is an important limitation when using Machine Learning algorithms for image processing; typically, the input of the neural networks is the color value of each pixel. This makes the network not learn examples correctly if, for example they appear in a different part of the image, or have a slightly different color. In essence, the raw values of each pixel are not particularly good indicators of the content of an image. Additionally this is not how human sight works [Mahapattanakul, 2019], these limitations led to the creation of Convolutional Neural Networks.

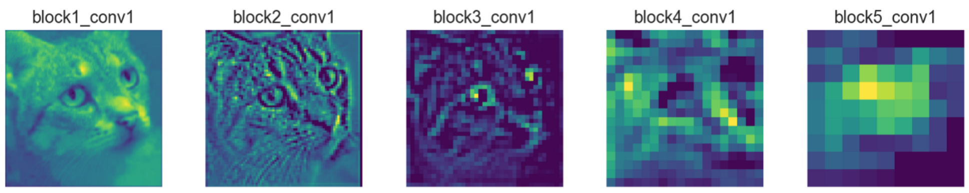


Figure 2.5: Visualization of a Convolutional Neural Network. Image from [Dertat, 2017].

Convolutional Networks are a special type of Feed-forward Artificial Neural Networks, with the addition of a new type of layer: a Convolutional layer [Pedrycz and Chen, 2020]. Convolutional layers work by sliding filters, known as kernels, over the image, extracting new features from the image. The kernels in the first convolutional layers extract low-level features, such as edges or corners while kernels in subsequent layers extract more complicated features, such as specific combinations of lines at specific angles. Convolutional Neural Networks also use pooling layers, which group pixels (usually 2x2 or 3x3) and output an aggregate function of these pixels, such as the mean value or the maximum value. An example of a possible structure for a Convolutional Neural Network (CNN) can be seen in Figure 2.6.

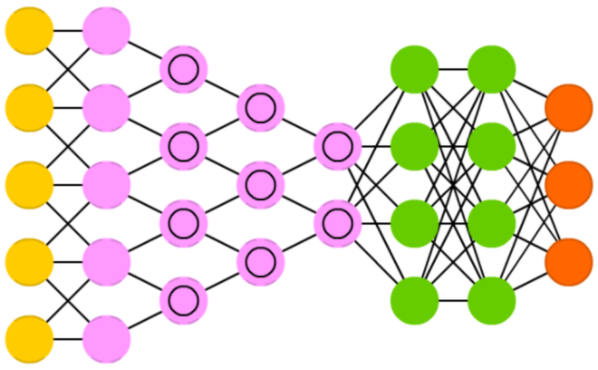


Figure 2.6: Structure of a CNN. Input cells (yellow), followed by kernels (pink), then convolutional or pooling layers (pink with circle), followed by a fully connected feed-forward network, hidden cells in green, output cells in orange. Image from [Veen, 2016].

The result of using convolutional and pooling layers is that the data contained in each dimension has been transformed, instead of the color values of each pixel, they now contain whether a specific geometric shape is present in sections of the image. This information is much more useful as it is more spatially independent: rotating, flipping, panning or any similar operation on the input image will wield similar values after all convolutional layers have been applied. This can be visualized in Figure 2.5 (which is an illustrative example and not necessarily based on reality), the input image is of a cat, and the purpose of this network is to classify whether this is the case or not. As the image goes through consecutive convolutional layers (and presumably pooling layers) the resolution decreases and the image gets more and more abstract. It is important to note that while the image looks more abstract it actually contains more pertinent information, particularly, it represents the features extracted, lines and curves and formations made from these low-level features. After these features have been extracted they are used as the input to a regular Feed-forward Fully Connected Artificial Neural Network, but in this instance the success rate of the network will be much higher [Mahapattanakul, 2019], as the question "Does this image contain a cat?" is a more direct function of whether these features are present, which was not the case if the input data were the value of individual pixels.

2.1.6 Autoencoders

An autoencoder is a type of Artificial Neural Network that does not require labels on the data, that is, it is trained through unsupervised learning [Pedrycz and Chen, 2020]. An autoencoder is comprised of two distinct parts, an encoder and a decoder (for this reason they are often called encoder-decoder networks). The encoder part of the network takes the original image as input and outputs a compressed version of it, known as code, or latent space, with the peculiarity that this compression is content aware, keeping the most important parts of the image while discarding superfluous information. The decoder's function is the opposite, taking the compressed image as input and outputting as faithful a recreation as possible.

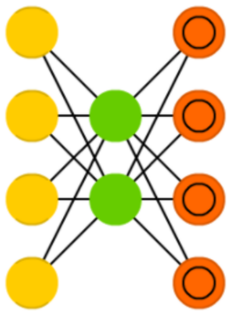


Figure 2.7: Structure of an autoencoder. Input cells in yellow, hidden cells representing latent space in green, output cells in orange. Image from [Veen, 2016].

Autoencoders are trained by providing an image that will then be encoded and subsequently decoded, with the difference between the original and the output being used to measure the error. Because the decoder only has access to the code and this code is determined by the network's specific architecture, as the network undergoes training the code will represent only the most important parts of the image.

2.1.7 Variational Autoencoders

Autoencoders, while a very powerful tool are not consistent enough to be able to perform content generation [Pedrycz and Chen, 2020]. The naive approach to force an autoencoder to generate content would be to randomly sample the latent space, with the assumption that every point in the latent space maps to something within the original domain. This assumption is not always true: some autoencoders do allow for this, but it is not a guarantee. Recalling how an autoencoder is trained there is no rule stating that every point in the latent space must map to a coherent image within the original space, in other words, it is possible that the autoencoder has over-fitted to the data in a way that random points in the latent space produce incoherent outputs. This problem could be fixed if the encoder could be trained to convert the data in a more regular way, which is what the variational autoencoder (VAE) attempts.

Training a VAE is almost identical to training a regular autoencoder, with the critical difference that the encoder maps the input not to a single point, but as a probability distribution over the latent space. Note that while the encoder's output has changed, the decoder's input has not; during training the input to the decoder is a randomly sampled point from the output of the encoder.

In practice, the output of the encoder changes to be a mean and the covariance matrix that describes that Gaussian distribution. This simple change means that it is impossible for the decoder to over-fit for the training data because more of this space is tested, instead of isolated discrete points. In turn, this also forces the encoder to give the latent space some organization, for the same reason.

2.1.8 Generative Adversarial Networks

A Generative Adversarial Network (GAN) is an architecture for machine learning that can create data similar to the input it is given. A GAN contains two distinct networks, a Discriminator (D) and a Generator (G), that function with opposite goals, hence the term adversarial [Goodfellow et al., 2017]. The Discriminator's job is, for any given input, to distinguish whether it came from the original data-set (ground-truth), or if it has been fabricated. The Generator, on the other hand creates random items to be evaluated by the Discriminator. In essence, because these two networks learn off of each other's outputs each one's improvements will force the other to improve. It is worth noting that while the theoretical architecture uses two separate networks this is not the case in practice, in which a stacked model of the two networks is created for ease of training.

GAN's are trained by creating a batch comprised equally of real images from the data-set and of

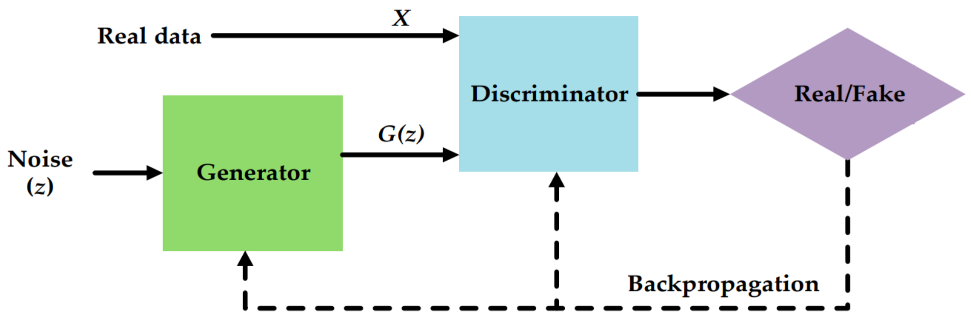


Figure 2.8: Generic structure of a GAN. Image from [Feng et al., 2020].

images created by the Generator. The Discriminator then assigns it's predictions to the given batch and is trained through back-propagation based on how correct each prediction was. After this step a second batch of images is created, though this time comprised entirely on predictions from the Generator. These predictions are then evaluated by the Discriminator and back-propagation is used to train the Generator, based on how close the images were to being classified as real images by the Discriminator.

A key detail about the GAN architecture is that the Generator allows for an input, a random vector, usually labeled z. This vector is necessary because there needs to be point of randomness within the system, otherwise the Generator, which is a deterministic system, would be able to create a single image.

2.1.9 Conditional GAN

While GAN's are one of the best deep learning architectures for image generation they lack a very important feature: it is impossible to guide their output within the space of possible results. Conditional Generative Artificial Network (cGAN)'s provide a way to remedy this limitation [Mirza and Osindero, 2014]. Traditional GAN's require a single noise vector input, but it is impossible to predict the output of the network given only the noise vector, other than performing the same calculations that will be performed by the Generator of the GAN.

In order to guide the image generation both the Generator and the Discriminator need to accept a new input, which is what will be used during inference to guide the image generation process. This additional information may be a class labeling, allowing, for example, the generation of specific digits when using the MNIST data-set. More relevant to our research is to use a smaller or simpler image as this additional information, which is one of the most commonly used methods of image-to-image translation.

To train a cGAN real samples are complemented with their respective additional information, when inputted into the Discriminator, with the goal that it will learn this relationship. When training with fake samples the Generator is given a noise vector, but also this additional information. The output of the

Generator is then used to train the Discriminator, along with the information the former used. The goal of the Discriminator is then, not only to discriminate between real and fake samples, but also to ensure that the image and the additional information display the same relation that is observed in ground-truth samples.

2.2 State of the Art of GAN

In this section we will summarize some of the more recent works that attempt to solve problems analogous to this research's, particularly in regards to solving the image-to-image translation problem and general improvements to machine learning architectures for images. We will also give an outline of the work of Nunes et al. [Nunes et al., 2022], which pioneered the specific area of research of generating maps from deep learning algorithms.

2.2.1 Progressively Growing GAN

The Progressively Growing Generative Adversarial Network (ProgGAN) is based on the work of Karras et al. [Karras et al., 2018]. ProgGAN's attempt to circumvent the problem that high-resolution images are harder to generate because they are easier to discriminate. The solution is to train a simpler model in lower resolutions, then increase both the model complexity and the resolution of the images being generated and discriminated. An important detail is that simply adding new layers would yield vastly different results from the already trained network, so the solution is to have a new parameter starting at 0 and increasing continuously to 1 that determines the influence of the new layers being added, with 0 being no influence (i.e. same as before the layer is added) and 1 being a normal network with the new layer, the influence of the new layers is controlled by use of skip connections over the new layer, the amount of which being inversely proportional to the influence of the new layer. Skip connections are connections between non-consecutive layers, passing the output of a given layer to the input several layers ahead.

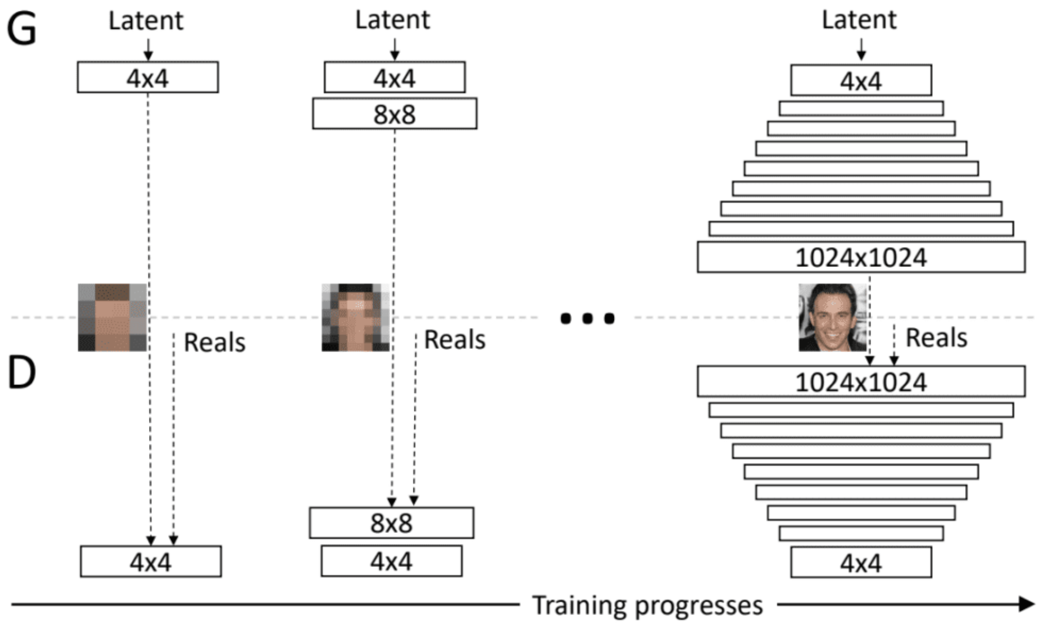


Figure 2.9: Training process of the ProgGAN: network complexity increases through training (left to right). Image from [Karras et al., 2018].

2.2.2 Wasserstein GAN

The Wasserstein GAN, or simply Wasserstein Generative Adversarial Network (WGAN) is a model suggested by Arjovsky et al. [Arjovsky et al., 2017] to combat a problem very common when training GAN's: because it is easier to differentiate between real and fake images than it is to create realistic looking images often the Discriminator becomes proficient too fast, ceasing to provide useful information to the Generator; one can imagine that a well-trained Discriminator can not only label fake images as such, but also every similar looking image, meaning that the Generator is unable to improve. WGAN's exist as a way to combat this issue: instead of training a Discriminator it uses what is known as a Critic (C). The Critic's job is to always provide information to the Generator, with the usefulness of this information increasing as the Critic is further trained, something that is not true for regular Discriminators. This is done by, instead of labeling images with either 1 or 0, the Critic rates the realness of the image, known as the Wasserstein estimate. In this model the loss functions are defined as such:

- Critic loss: (average critic score on real images) – (average critic score on fake images)
- Generator loss: -(average critic score on fake images)

This definition for the Critic's loss creates a function that doesn't saturate and converges to a linear function, providing cleaner gradients than a generic Discriminator. While this is the largest change there are other subtleties to this model:

- The Critic is trained n times more than the Generator. The authors suggest a value of 5 batches of training for the Critic for every batch the Generator is trained on. This is done as a consequence of the fact that results always improve for a better trained Critic, which would not be the case with a Discriminator.

- The weights on the Critic are clamped to a limited box after every batch. The range suggested by the authors is $[-0.01, 0.01]$. If the magnitude of this range is too high the Critic will take too long to converge, while a value too low will render the Critic unable to learn some features.

2.2.3 Map generation from GAN-based models

The work of Nunes et al. [Nunes et al., 2022] has been mentioned throughout this work, not only did this research pioneer the area of using deep learning algorithms to generate maps for strategy games, but it also had direct influence in the improvements this thesis aims to create.

This researched focused on experimenting different models of deep learning models to investigate which would perform the task of creating maps for strategy games better. In total, four different models were explored: Deep Convolutional GAN (Deep Convolutional Generative Artificial Network (DCGAN)), Wasserstein GAN (WGAN), ProgGAN, and a VAE model with adversarial loss. Of the four models tested, the DCGAN and VAE + GAN provided results inferior in quality to the ProgGAN and WGAN. These last two models provided results with high visual quality, with the ProgGAN being more efficient to train than the WGAN.

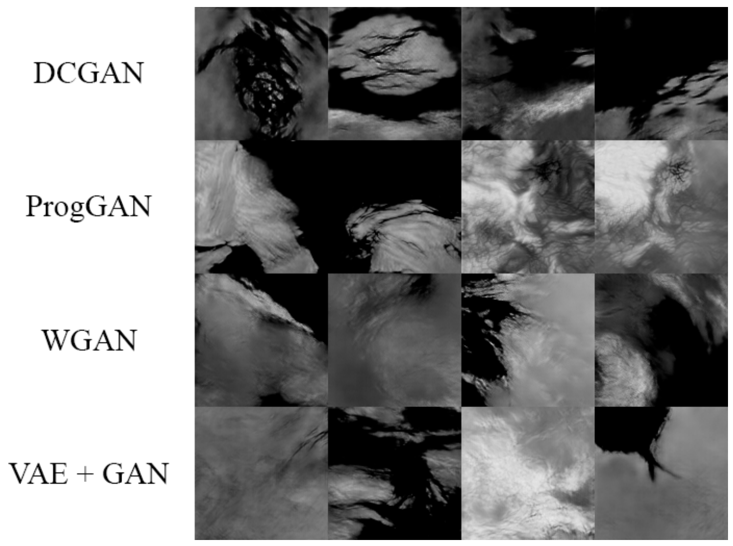


Figure 2.10: Sample of results obtained by [Nunes et al., 2022].

2.2.4 Auto-encoders for image-to-image translation problem

Lyu et al. [Lyu et al., 2017] proposed an architecture for a Machine Learning system that would take a generic font-style Chinese symbol and convert it to a hand-drawn depiction of the same symbol. Their work has great relevance to this research as the problems are similar - the font image is analogous to the low-level sketch and the hand-drawn symbol to the realistic height-map.

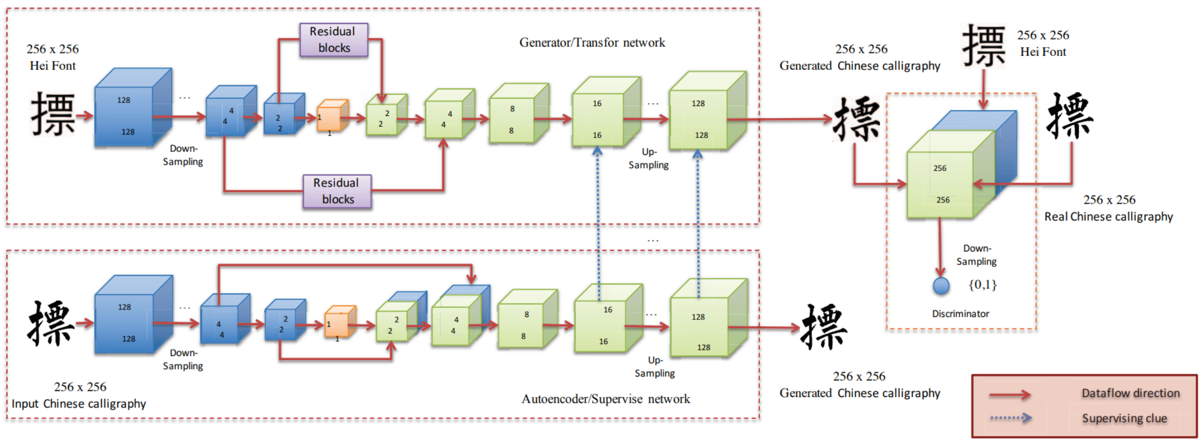


Figure 2.11: Model for synthesizing calligraphic Chinese characters from a generic font. Image from [Lyu et al., 2017].

Figure 2.11 depicts the model suggested, it contains a transfer network, which was originally based on an auto-encoder architecture, but later updated to a GAN architecture, and an auto-encoder based model called the supervise network, which is only used during training. The transfer network takes the generic font image and outputs a calligraphic depiction of that character, this network is, essentially, the Generator. The supervise network receives the calligraphic image and attempts to recreate it. This architecture forces the supervise and transfer network to have a similar model, such that, during training, loss from the supervise network is used to influence weight adjustment of the transfer network. It is important to note that the supervise network uses skip connections skipping over the bottleneck. What this does is give the decoder of the supervise network more information, as it can access features that were not present in the latent space, allowing it to better recreate the character, in turn making it more useful to train the transfer network.

$$\mathcal{L}_{\text{supervise}} = \mathbb{E}_{y \sim p_{\text{data}}(y)} [\|y - S(y)\|_1] \tag{2.2}$$

Equation 2.2 shows the loss function for the supervise network, which is a generic auto-encoder. It is defined as the L1 loss between the original calligraphic image (y) and the output of the supervise network of that image.

$$\begin{aligned}\mathcal{L}_{\text{reconstruct- }(k)} &= \mathbb{E}_{t_{(k)}} \sim_{p_{\text{data}}\left(t_{(k)}\right)} \left[\left\|t_{(k)} - s_{(k)}\right\|_1\right] \\ \mathcal{L}_{\text{reconstruct}} &= \lambda_1 \times \mathcal{L}_{\text{reconstruct }1} + \ldots + \lambda_k \times \mathcal{L}_{\text{reconstruct- }(k)}\end{aligned} \tag{2.3}$$

Equation 2.3 shows the reconstruction loss for the transfer network. This loss is defined as L1 loss between the low level features of the transfer and supervise networks, labeled as t_i and s_i, where i denotes the layer index. The authors suggest $\lambda_i = 1$ for all values of i.

$$G^* = \arg\min_G \max_D \mathcal{L}_{\text{adversarial}} + \lambda_s \mathcal{L}_{\text{supervise}} + \lambda_r \mathcal{L}_{\text{reconstruct}} \tag{2.4}$$

The final objective can be expressed through Equation 2.4: it is a weighted sum between adversarial loss, loss from the transfer network (called reconstruction loss) and loss from the supervise network. The weights for the sum, λ_s and λ_r have been tested by the authors, who suggest the value 100 for both. The reconstruct term in the loss equation is the reason for the supervise network; reconstructing the character is easier (and made even easier through the skip connections) than creating detail from nothing - which is the Generator's task - which is why this model is advantageous.

2.2.5 Semantic Image Synthesis

In recent years there have been large advances made in the area of image-to-image translation [Isola et al., 2017, Park et al., 2019, Wang et al., 2018]. This area encompasses every problem that can be described as taking an image from one domain to another, with a subset being semantic image synthesis, which is the specific process of taking an image made up of labels and generating a realistic image of the same content, a problem very similar to that of this research.

The most common approach to the problem of semantic image synthesis is to use a GAN where the Generator accepts the semantic map as an additional input [Isola et al., 2017, Park et al., 2019, Wang et al., 2018], this is commonly referred to as a Conditional Generative Artificial Network (cGAN). Another common solution is to use autoencoders, or variational autoencoders to generate the images. In this approach the autoencoder's job is not merely to reconstruct the semantic image map but to generate the realistic image, this is generally complemented with another network to provide adversarial loss, essentially rendering it into a GAN where the Generator is an autoencoder or variational autoencoder.

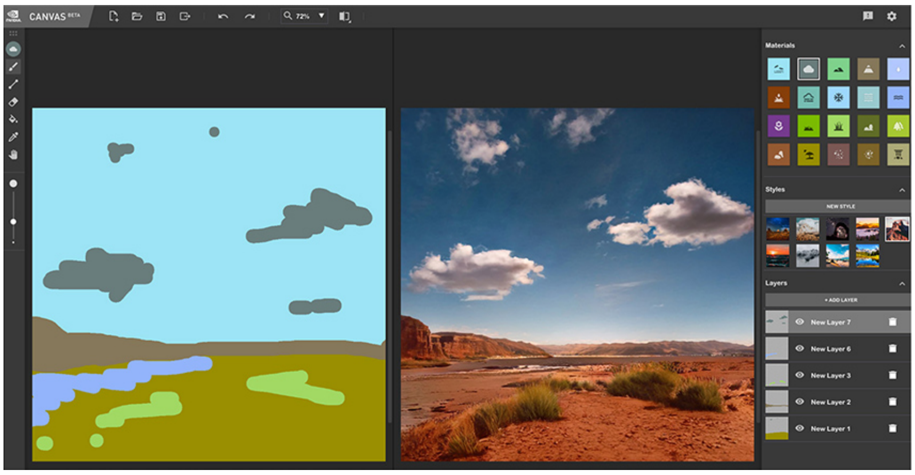


Figure 2.12: NVIDIA Canvas, a free tool that generates realistic images from semantic maps. Image from `https://www.nvidia.com/en-us/studio/canvas` .

2.2.6 High-Resolution Image Synthesis and Semantic Manipulation with Conditional GAN's

Wang et al. [Wang et al., 2018] have recently proposed many improvements to the semantic image synthesis problem, most relevant to this research is a way to deal with fine detail in high resolutions. The approach suggested to deal with this problem is to use multiple Generators at different resolutions, a lower resolution Generator called the global Generator and one or more local Generators. In their research, Wang et al. use a global Generator at a resolution of 1024x512 to create the image, with a local Generator, responsible for adding detail, at 4 times the resolution (2048x1024), this approach is not unlike that of an AI super-sampler. This same approach can also be applied to the Discriminator, using a lower resolution image as input to a Discriminator that guides global decisions, with finer detail being judged by Discriminators operating over higher resolution images. A key advantage of this approach is that Local Discriminators do not need to observe the whole image at once, allowing them to be a faster, shallower network than would otherwise be the case; it also means that in order to increase the resolution of the network these Discriminators do not have to be retrained, it's enough to create a new Discriminator and train it.

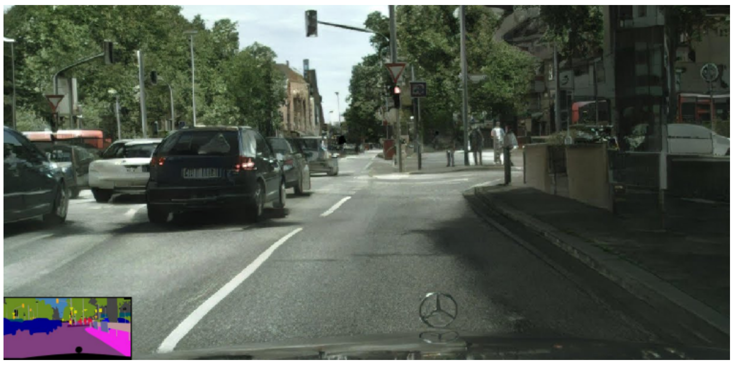


Figure 2.13: Sample of results of Wang et al.. Label map used to create image in lower left corner. Image adapted from [Wang et al., 2018].

2.2.7 Image-to-Image Translation with Conditional Adversarial Networks

Isola et al. [Isola et al., 2017] describe other improvements for the image-to-image translation problem, some of which very pertinent to this research. The first problem the authors solve is the fact that often during the training process of GAN's the Generator may learn that the correlation between the generated image and the noise vector is not evaluated by the Discriminator, in other words, the Generator may choose to ignore the noise vector, theoretically gaining better results. While this may be true, it is not wanted, as it is a side-effect of the Generator becoming deterministic, and thus, only able to generate a single image. The solution suggested by the authors is not to input the noise vector directly into the Generator, but instead to use random dropout[1] in any layer as a means of generating noise to create different images.

The authors also propose the PatchGAN, which differs from a regular GAN in that the Discriminator may function at a much smaller resolution and work over a single patch of the image at a time. This Discriminator calculates it's rating for the image over all patches of the image and averages the results, giving it the final output. The advantages of this approach are that the Discriminator may be a shallower, and thus faster network and becomes independent of the resolution of the image being generated, while the approach of Wang et al. [Wang et al., 2018] also provided a level of abstraction between the Discriminator and the resolution of images, this approach does not require any further training on the part of the Discriminator.

[1] Dropout is a technique in which random neurons are temporarily removed from the network.

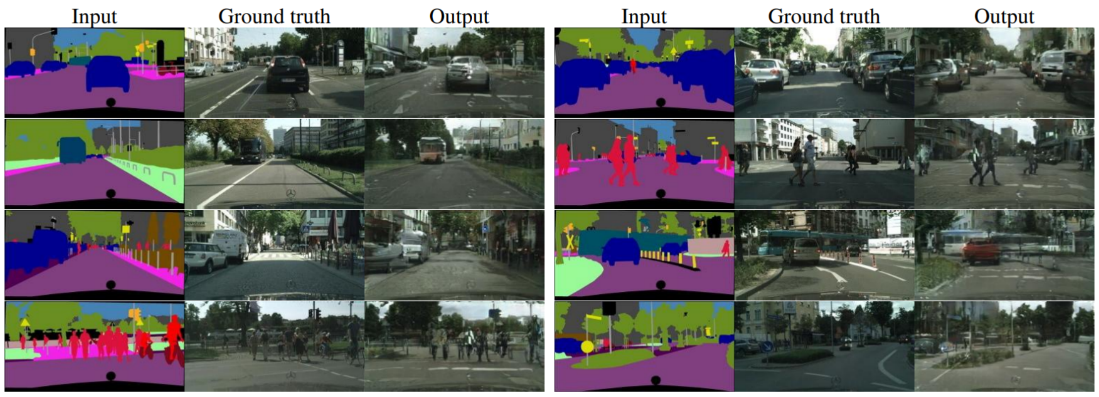


Figure 2.14: Sample of results of Isola et al.. Label map used to create image in lower left corner. Image from [Isola et al., 2017].

2.2.8 GauGAN: Semantic image synthesis with spatially adaptive normalization

The work of Park et al. [Park et al., 2019], used to create NVIDIA Canvas, shows very interesting results. The first major improvement the authors suggest is the SPatially-Adaptive (DE)normalization (SPADE), which is an improvement over batch normalization. Batch normalization is a technique in which the data-set being used to train a network is normalized such that every batch has mean equal to zero and standard deviation equal to one, this makes the network more stable and the training faster [Goodfellow et al., 2017]. The authors argue that the common way to perform batch normalization on images, normalizing according to the mean and standard deviation of each channel across the whole image, is wrong and can lead to information loss. SPADE attempts to correct this by normalizing each pixel location separately, in other words, after SPADE each pixel location has the same mean and standard deviation across all images whereas regular batch normalization would only guarantee normalization for an entire image.

The network suggested by the authors, GauGAN, also differs in architecture from others: the authors suggest for the Generator a decoder made up fully of up-sampling layers and groups of SPADE layers, ReLU activation layers and convolutional layers, called SPADE blocks. The Discriminator network follows a similar implementation, being made up fully of convolutional down-sampling layers, outputting a list of features, that then need to be averaged to differentiate between real and fake images.

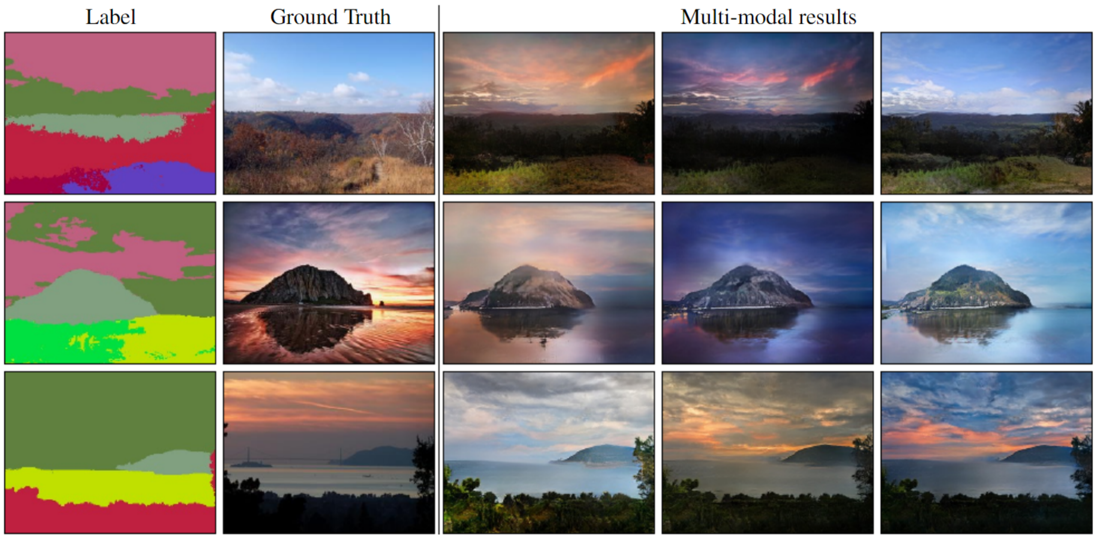


Figure 2.15: Sample of results of Park et al.. Label map used to create image in lower left corner. Image adapted from [Park et al., 2019].

3

Dual Critic Conditional Wasserstein GAN for Height-Map Generation

Contents

In this chapter we will discuss the overall architecture of the system we created, in particular our deep learning model, which we named as Dual Critic Conditional Wasserstein Generative Adversarial Network (DCCWGAN). In particular we will examine how the paired data-set, named the Intermediate Map Representation (IMR), is created, how the different deep learning models were created, and discuss three key architecture decisions.

3.1 System Overview

In this section we will give an overview of our system's architecture, examining the data flow and all the networks present in our deep learning model.

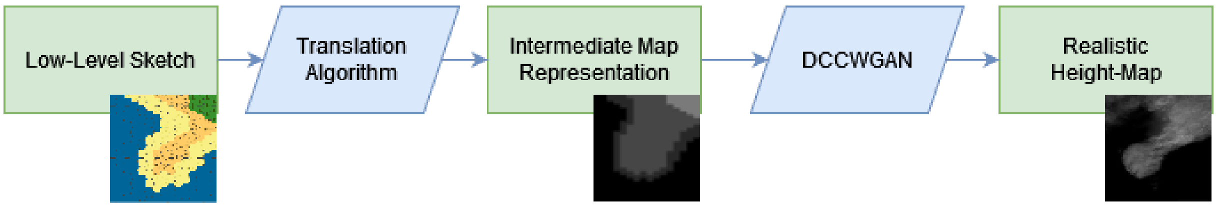


Figure 3.1: Data flow for the system, user creates low-level sketches (left) and the system outputs Realistic Height-maps (right).

The data flow for our system (illustrated on Figure 3.1) begins with the low-level sketch, which is created by the user and exists only during inference. While we illustrate a specific type of sketch it is important to stress that the system is very easily adapted to use a different tool to create sketches. The low-level sketch is then used as input to a translation algorithm, which transforms it into the Intermediate Map Representation (IMR) format.

For our system to function there needs to be a format that will be used as input by the user, but is also able to be created from the ground-truth data-set, so as to compare if the content matches during training, this format is the Intermediate Map Representation (IMR). While it would be possible for a user to create maps directly in the IMR format, we opted to create it from an existing map-creating tool, using a simple, deterministic algorithm. Users wanting to use a different map-creating tool need only to create this translation algorithm. The IMR format exposes the average height of each cell in a hexagonal grid. The IMR outputted by the translation algorithm is then used as input to our deep learning model, the Dual Critic Conditional Wasserstein Generative Adversarial Network (DCCWGAN), both during inference and training, resulting in the height-map.

3.1.1 The DCCWGAN

In this section we will describe our deep learning model, the DCCWGAN, in its entirety. In order to choose an architecture for our deep learning model we started by dividing the problem into two distinct learning processes:

- Create realistic maps.
- Create maps whose content corresponds to the given input.

Given this division we opted for an architecture with a Generator, but two distinct critics, one for each of the necessary learning processes. The first of these Critics evaluates only the visual qualities of the images generated, and forces the Generator to create more and more realistic images; for this reason we call this the Realism Critic. The second critic is responsible for evaluating how well the contents of the generated maps correspond to the input used in their generation; this critic is called the Conversion Critic. Training is done using one Critic at a time, depending on the current loss. The decision of which Critic to use is explained in further detail in Section 3.2.

This thesis' predecessors [Nunes et al., 2022] focused heavily on testing multiple Deep Learning architectures and comparing their results. In this work we use that knowledge and opt to build only upon one of the most successful models showed, the Wasserstein GAN [Arjovsky et al., 2017], iterating the model based on results achieved. The WGAN was chosen over the ProgGAN as, despite having a longer training period, it is more simple and therefore, in our opinion, more likely to maintain good results in spite of changes added.

3.1.1.A Generator

The Generator is the network responsible for generating images. It accepts two separate inputs: the IMR and a noise vector. The IMR controls the general layout of the final result, while the noise vector, similarly to other GAN's, provides a source of entropy, adding a layer of randomness to the output and allowing the same IMR input to generate multiple different results. A concern that is specific to cGAN's is that the noise vector must provide some randomness, but not so much that the results differ from the IMR content provided.

3.1.1.B The Realism Critic

This critic functions exactly as a critic in any generic WGAN. It takes as input either a real or fake image and attempts to differentiate between these two domains. Unlike the Conversion Critic, this critic does not receive the IMR, and therefore judges only the visual quality of the image, and not its accuracy.

3.1.1.C The Conversion Critic

The Conversion Critic, unlike a generic WGAN Critic, receives a paired input: a real or fake image and an IMR. The IMR given is either generated from the real data-set or, in the case of fake images, the

IMR used as input for the Generator. The purpose of this network is to discriminate whether the general content of the image matches that of the IMR given. In the case of real images the pair will match since the IMR is created from the image itself; in the case of fake images the network attempts to correct the generator to force this content matching.

The data flow for training the Conversion Critic can be observed in Figure 3.2. The left side of the image refers to training with fake images, while the right side of the figure refers to training with real images. In essence, the goal of the Conversion Critic is to learn the opposite of connection A, in other words, how to generate a map given only it's IMR representation.

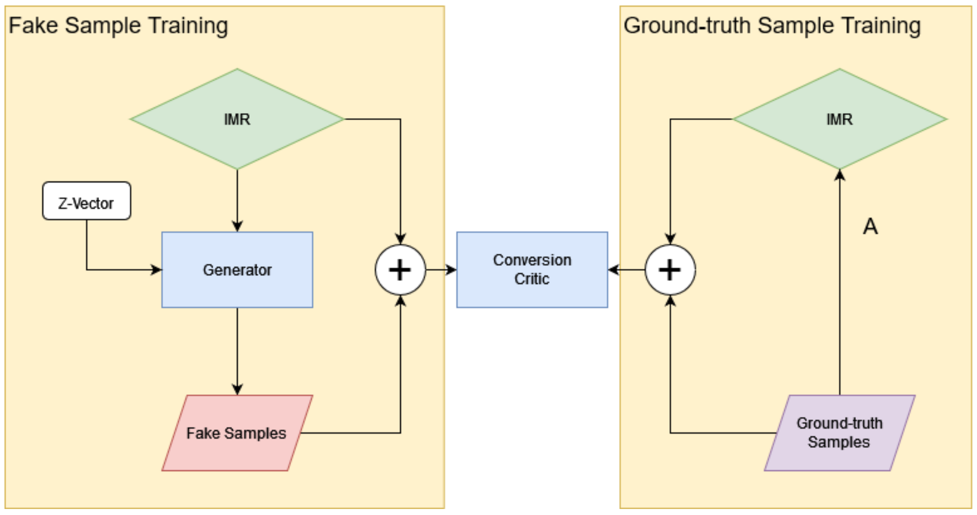


Figure 3.2: Training of the Conversion Critic.

3.2 Training

The training of our system, for each GAN, is identical to the algorithm used to train any generic WGAN, as presented in Algorithm 3.1, the only difference from this algorithm is that our system uses not one, but two separate GAN's, and as such, it is necessary to decide which GAN to train at any given point.

Since our architecture uses two separate Critics, and therefore, two separate GAN's one consideration to be had is how to choose which GAN to train. The tasks required of each of the Critics are of different scales of difficulty, with the Conversion Critic having a simpler task. Theoretically, this means it is possible to lower this latter Critic's learning rate and train both GAN's every epoch, however, this is neither easy to accomplish, nor is it optimal; a better strategy is to evaluate the current loss and choose a GAN to train accordingly. This approach has several advantages: it doesn't require a lower learning rate on the Conversion Critic since it won't be trained every epoch; it is flexible and adapts quickly to sudden changes in the loss function. The decision of which GAN to train is made as described in Algorithm 3.2.

Algorithm 3.1: WGAN training algorithm with n_critic = 2 and $batch\_size$ = 64

```
Normalize $p\_data$ between $-1$ and $1$;
for $epochs$ do
    shuffle $p\_data$;
    $half\_batch = \frac{batch\_size}{2}$;
    $iterations = \frac{\text{number of images of } p\_data}{half\_batch}$;
    for $iterations$ do
        Choose $GAN$ to train according to algorithm 3.2;
        for $n_critic$ do
            $z\_vectors$ = $half\_batch$ samples from $\mathcal{N}(\mu, \sigma^2)$;
            $real\_images$ = $half\_batch$ images from $p\_data$;
            $fake\_images$ = $half\_batch$ samples from $G(z\_vectors)$;
            $y\_real$ = $half\_batch$ size vector of value $-1$;
            $y\_fake$ = $half\_batch$ size vector of value $1$;
            Train $C$ with $real\_images$ labelled as $y\_real$ using gradient descent with Wasserstein
              estimate;
            Train $C$ with $fake\_images$ labelled as $y\_fake$ using gradient descent with Wasserstein
              estimate;
        $z\_vectors$ = $batch\_size$ samples from $\mathcal{N}(\mu, \sigma^2)$;
        $y\_gen$ = $half\_batch$ size vector of value $-1$;
        Train $G$ with $z\_vectors$ and $y\_gen$ labels using gradient descent with Wasserstein estimate;
```

Algorithm 3.2: Choice of GAN to be trained

```
for iterations do
    if last_conversion_fake_loss$\times 2$ >last_realism_fake_loss then
        gan = conversion_gan
    else
        gan = realism_gan;
```

It is important to note two aspects of how this decision is made: it depends entirely on the loss of the fake images, and not in the critic's evaluation of the real images, this is because, in theory, the loss of the real images will continuously increase, therefore becoming a function of how much the network has been trained; the loss for the fake images of either network, on the other hand, is affected by the training of the other, thus, if the training of a network negatively impacts the loss function of the other, this will be corrected by changing which network is being trained. The second important aspect of how this decision is made is that it allows for a weight to be assigned, in the example shown, and the final value used is to give twice as much importance to the Realism GAN over the Conversion GAN, the reasoning being that the task of the Conversion GAN is less critical as the user will not notice if the content doesn't fully align, and the looser this restriction is, the more freedom allowed for the Realism GAN to improve the aesthetic aspect.

3.3 The data-sets

In this section we will explain the different data-sets, their creation, and their use. The first data-set is adapted from a topographic image of the Earth, and serves as the ground-truth for realistic height-maps. The second data-set, the IMR data-set, serves as a representation of the low-level sketches and is used as input to the system.

3.3.1 The Ground-Truth Data-Set

In order to train the deep learning models to create realistic height-maps we must start with something we can classify as such. The starting point for this realistic data-set is the Shuttle Radar Topography Missions (SRTM) data-set, created by NASA. The ground-truth data-set used in this work is a filtered and augmented version of the SRTM data-set, as adapted by the researchers of GAN-Based Content Generation of Maps for Strategy Games [Nunes et al., 2022], who were kind enough to provide it to us. In total, this augmented data-set contains 12640 images of various types of terrain present on Planet Earth. Figure 3.3 presents a few example images from this data-set.

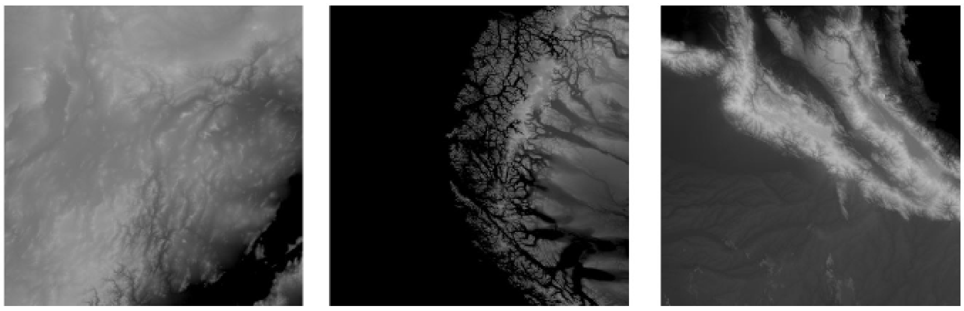


Figure 3.3: Sample images of the ground-truth data-set. Image from [Nunes et al., 2022].

3.3.2 Paired Data-set

The pair to the ground-truth data-set is called the IMR data-set, and is used both in training the Deep Learning model and is what the user will create as input to the final system. This format is a simple grid of values corresponding to the average elevation value in that area. It is important to note that these values belong to a limited number of options, the amount of which has to be carefully chosen: a number of classes too low will make the ground-truth data-set not match the IMR format as closely, giving an unpredictable output while simultaneously limiting user freedom; a number of classes too high will, in turn, difficult training as some values may become too rare in the ground-truth data-set. We give further detail on how the IMR format is created in Section 4.1.

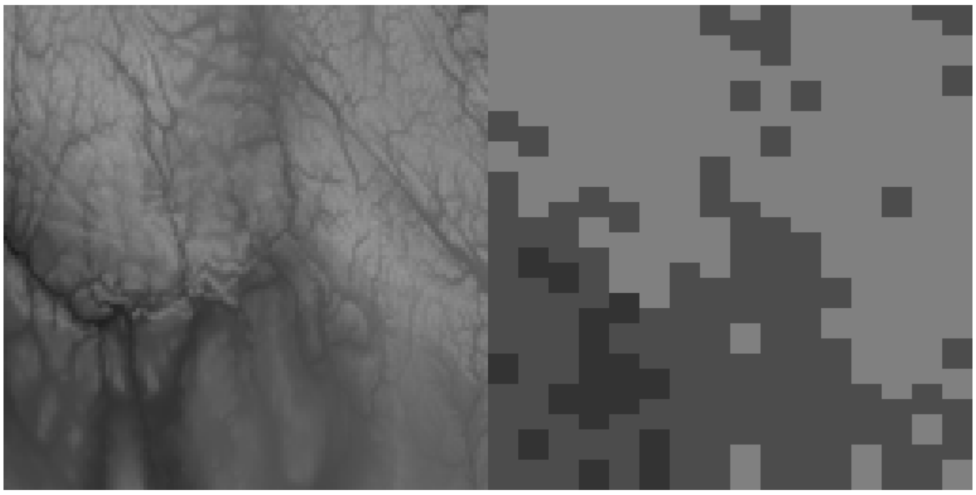


Figure 3.4: Left: Original image from ground-truth data-set. Right: Same image translated into the IMR format.

3.4 Architecture Variations

In this section we will overview three different architecture decisions that we deemed to have theoretical merit, regardless of their experimental results, which will only be discussed in Chapter 4. The three decisions to be discussed are the following:

- Bilinear interpolation: use bilinear interpolation in the Generator in an attempt to overcome possible limitations of the IMR format.
- Forced Conversion Critic Scheduling: schedule which GAN to train in such a way that no GAN is left untrained for too many consecutive epochs.
- Per-epoch scheduling: schedule which GAN to train on a per-epoch basis.

3.4.1 Bilinear Interpolation

One possible improvement to the system would be to use bilinear upscaling in the Generator. Recall from the structure of the Generator (Figure 4.5) that it uses upsampling layers to go from the resolution of the IMR format to the resolution of the final height-maps. The upsampling layers in the final model use nearest neighbour interpolation, however, there is a theoretical justification for using bilinear upsampling: such an upsampling method would make for smoother features, something already more aligned with the features present in nature than the rougher changes in altitude created by the nearest neighbour interpolation method.

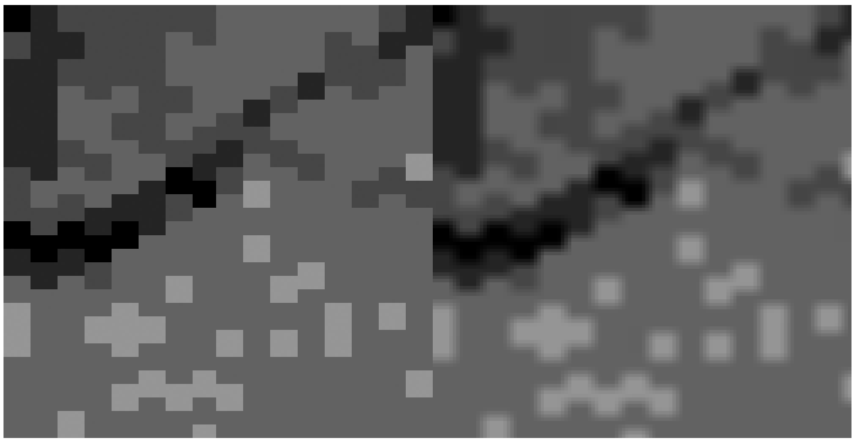


Figure 3.5: Comparison between original IMR and upscaled with bilinear interpolation.

3.4.2 Forced Conversion Critic Scheduling

The method for choosing which GAN to train, established in Algorithm 3.1 has a seemingly large drawback: it is not unreasonable to think that not training a network for various epochs would bring adverse results, given that the results of the Generator will be vastly different from the last time it was trained. While this is not necessarily implicit in the algorithm it is also not prevented, and through empirical experimentation we discovered that, for the optimal learning rates, it was common for the Conversion GAN to go hundreds of epochs without being trained once. To combat this issue we tested a maximum limit of consecutive epochs trained per GAN, therefore forcing the less trained GAN not to go too many epochs without training.

3.4.3 Per-epoch Scheduling

The algorithm for scheduling which GAN to train doesn't need to be run once per epoch: it may be run once for any arbitrary number of epochs, or, conversely, to any fraction of an epoch, down to a singular batch. We theorise that making this decision on a per-epoch basis or per-batch basis may have significant effects in the final results, as, the more granular this decision is made, the less each GAN may go without being trained, which may impact results positively. On the other hand, scheduling the training on a finer scale than a whole epoch may cause each GAN to train on a fraction of the data-set, theoretically making each GAN train on non-overlapping halves of the data-set, though this extreme case is statistically impossible.

4 Implementation

Contents

In this chapter we will describe in further detail how we implemented the system, as described in Chapter 3.

4.1 Implementation of the IMR format.

In this section we will explain in detail the implementation of the IMR format, including the algorithms for translating both the ground-truth data-set and low-level sketches originated from one specific tool.

As explained in Section 3.3.2, one of the considerations necessary in the creation of the IMR format is the number of values present. The final value chosen is five, and their relative frequency over our ground-truth data-set is present in Figure 4.1. While these class values do not have definitive real-world equivalents we can still infer what they represent most commonly, note that the values present in the IMR format are in the range $[0, 1]$, the same range used by the height-maps during training:

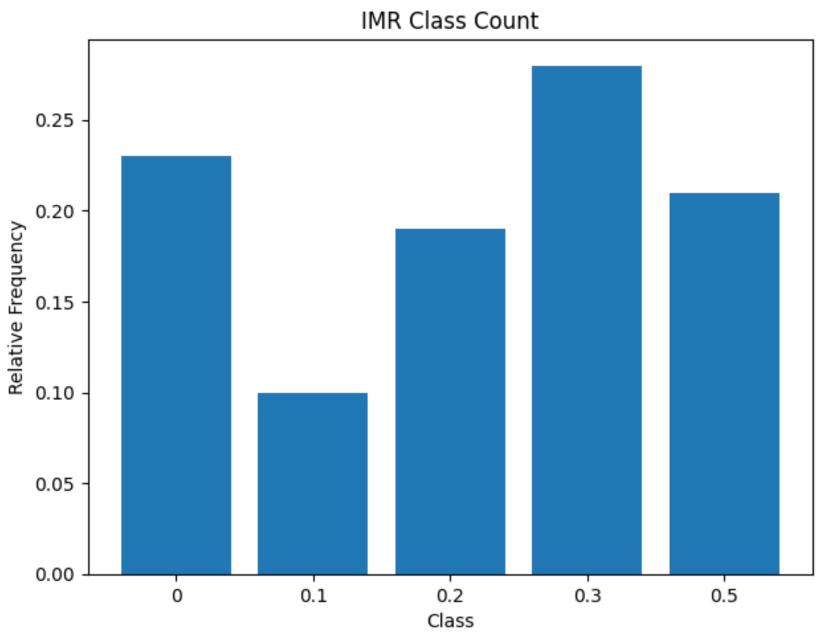


Figure 4.1: Absolute frequency of each class present in the IMR format.

- **0.0:** Ocean, this is the only class whose real-world counterpart is well defined.
- **0.1:** Coastlines, river deltas, swamps and other landmasses that tend to be partially submerged. While this class has the lowest absolute frequency it is very important as it is the lowest non-zero value and, as such, serves to distinguish landmasses regardless of how low they are.
- **0.2:** Plains, forests, deserts and other types of land of mid-low altitude.
- **0.3:** Plateaus, hills and other terrain of mid-high elevation.

- **0.5:** Mountainous terrain. It is worth noting that any sufficiently tall mountain on the ground-truth data-set will map to this value, regardless of its elevation, as such, it is impossible for the user to specify the height of a mountain. We consider this limitation to not be too detrimental to the end result, and the existence of classes with even higher values is difficulted by the fact that it would not have a sufficiently significant sample size for the model to train with.

To create the IMR format, we analyze each image in the ground-truth data-set and obtain the average elevation of all pixels that correspond to each hex cell. Each of these averages is then compared to the list of possible values for the IMR format and is attributed one of the two closest values randomly based on how close it is to either, according to Algorithm 4.1.

Algorithm 4.1: Mapping of area elevation to IMR class.

```
for hex do
    diff = highest_lower_IMR_class - lowest_higher_IMR_class
    r = random[0, 1] * diff
    if r + highest_lower_IMR_class < hex then
        class = lowest_higher_IMR_class
    else
        class = highest_lower_IMR_class
```

It is important to note that we do not simply map the area's elevation to the closest IMR class, but instead introduce some randomness to this conversion. This randomness makes the paired data-set less homogeneous, removing examples where the whole image would be a single class, a common occurrence with river deltas or plains. River deltas are particularly problematic as they could become the ocean class (zero elevation) despite containing land, which in turn would make the model generate similar land masses where only ocean was intended. In Figure 4.2 we can observe an example of this phenomenon: despite there being two separate landmasses, were the conversion to happen using only the closest class the top-most landmass would not be translated into the IMR, in turn the Generator may learn that ocean could occur with this type of landmass, which is not desirable.

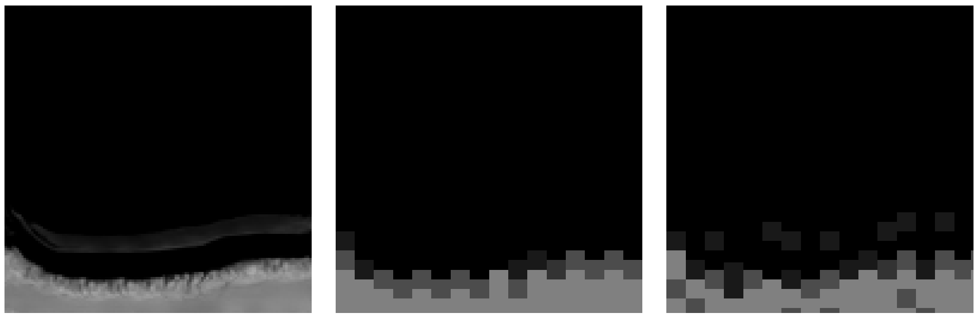


Figure 4.2: Left: Original image from ground-truth data-set (brightness has been increased for ease of readability); middle: IMR with no randomness; right: IMR using randomness.

4.1.1 Adapting the IMR for Neural Networks.

An important detail to note is that the topology of the IMR format is different to that of an image by the simple fact that it is based on hexagons, therefore having six neighbouring cells instead of the four present in a square grid. This presents a problem when implementing the system as convolutional operations are, usually if not always, only available for square grids. To circumvent this issue, whenever the IMR format is used in a network, we build a square grid where each cell is a 2x2 square, offset vertically by a single pixel every other row, thus maintaining the topology of the original hexagonal grid. Note that all depictions of the IMR format are using this offset square grid. As a consequence of having the this format as input the first convolutional layer must have a dilation rate of 2, as otherwise it would only be able to evaluate the current cell and two of it's neighbours; having a dilation rate of 2 allows the cell and four of it's neighbours to be evaluated. Note that this means that each original hex cell is evaluated 4 times, with the top and bottom cells being present in all evaluations and each pair of left/right cells only once.

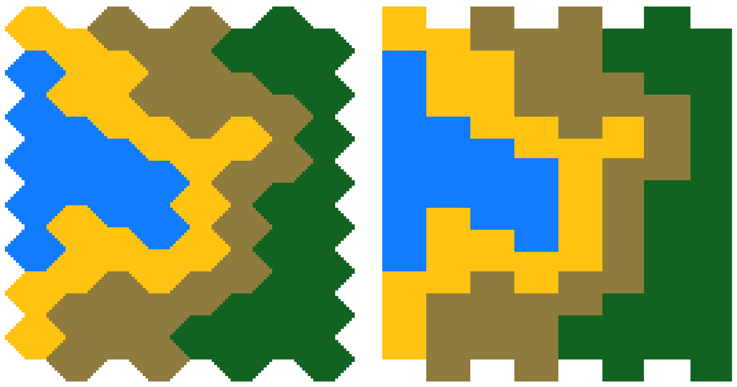


Figure 4.3: Comparison of hex and topologically equal square grid.

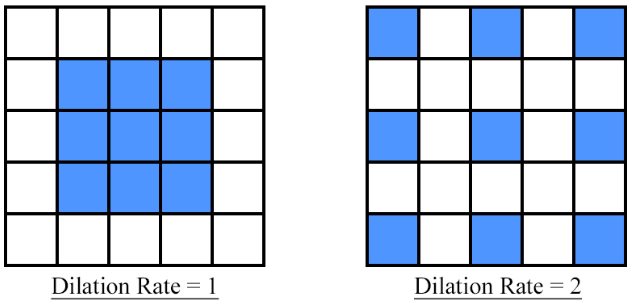


Figure 4.4: Placement of each kernel filter for dilation rates of 1 and 2.

4.1.2 Conversion of Hex Maps to the IMR format

Establishing a format such as the IMR provides a useful advantage to the user experience as it decouples the generation of the low-level sketch from the input of the networks, requiring only a script to translate from the output of the chosen tool to the IMR format. This means our system can easily be adapted to work with tools such as hex map generators, such as those referred in Section 2.1.1, to working with any generic image editing software and even to be ported into a game engine, and used as an in-game map editor.

While there is a wide variety of tools that may be adapted to create IMR format files, we implemented only one of these, as a way to test the system and the workflow. We created a script that converts from maps generated by a hex map generating tool called hexographer [1]. We chose this tool because it is free and easily accessible and generates maps in a cleartext format that is easy to read and convert from. To perform this conversion we simply read the biome of each hex and convert it to the corresponding IMR class using a look-up table, as can be seen in Table 4.1.

Hex Class	IMR Class
Ocean	0.0
Sea	
Bog	0.1
Beach	
Coast	
Wetlands	
Farmland	0.2
Desert	
Grassland	
Forest	0.3
Hills	
Plateau	
Mountain	0.5
Volcano	

Table 4.1: Conversion table from hex tool to the IMR format.

4.2 Structure of the Networks

In this section we provide visual representations of the structure of the three networks used by our system, along with a brief insight into the reasoning behind some decisions. Complete tables containing these structures are present in Appendix A.

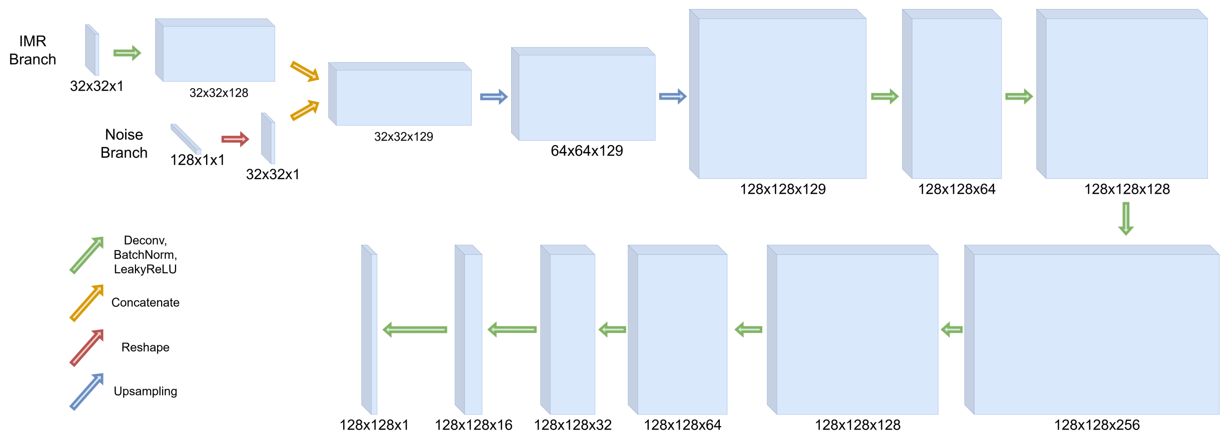


Figure 4.5: Structure of the Generator.

4.2.1 Generator Structure

Figure 4.5 represents the complete structure of the Generator. All deconvolutional layers use kernel size of 3 and stride of 1, except for the layer following the IMR input, which uses a kernel size of 3 and dilation rate of 2, and the final convolutional layer, which uses a filter size of 5.

There are a few details worth pointing out about the architecture of our Generator: first, we start by increasing the number of channels on the IMR branch *before* it is concatenated with the noise branch. The justification for this is that when concatenating these two branches the weight given to the noise branch is greatly reduced, as it is now responsible for only 1 of the 129 channels. While this is not critical, and the network is able to learn to give less importance to the noise, we found that this simple change positively affected the final results. Another decision that may appear counter-intuitive is the number of channels across the network, starting at 129 on the main branch, then decreasing to 64 only to expand gradually to 256 before again decreasing gradually to 1. While we tested different configurations, including removing the intermediate layers such that the model goes directly from 129 channels to 256, we found this configuration to yield optimal results.

4.2.2 Conversion Critic Structure

Figure 4.6 represents the complete structure of the Conversion Critic. All convolutional layers use a filter size of 4, except for the layer following the IMR input, which uses a kernel size of 3 and dilation rate of 2, and the final convolutional layer, which uses a filter size of 5. When image width and height decreases this is done using a convolutional layer of stride 2.

The Conversion Critic has a much simpler task than that of the Realism Critic, therefore requiring less parameters and less total memory to train. This network has two separate inputs that then concatenate

[1] https://www.hexographer.com

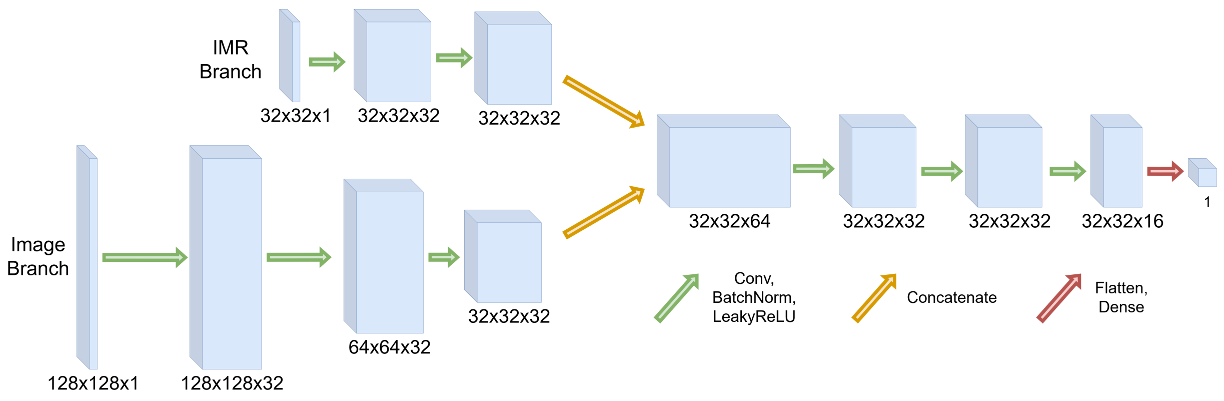


Figure 4.6: Structure of the Conversion Critic.

channel-wise. It is worth noting that the Image Branch of this network downsamples very rapidly, this is because, due to the way in which we defined the IMR format it is much easier to confirm that the two branches have similar geographical content at a lower resolution.

4.2.3 Realism Critic Structure

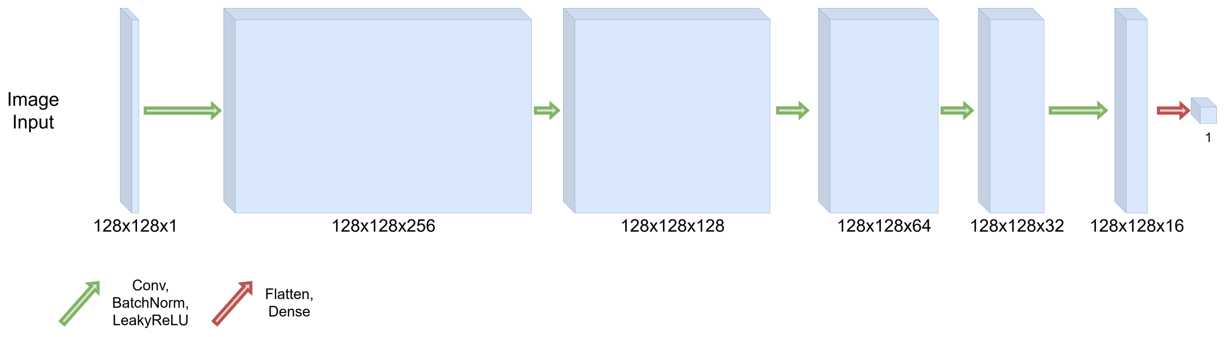


Figure 4.7: Structure of the Realism Critic.

Figure 4.7 represents the complete structure of the Realism Critic. All convolutional layers in this critic use a stride value of 1 and filter size of 4. While on a surface level this network appears smaller, as it contains less layers, it contains over 7 times more parameters. This is because while the Conversion Critic only guides the broader strokes of the image, the Realism Critic is responsible for the finer detail, which is a more difficult task that will be held to a higher standard by the end user.

A different approach to the architecture of this network would be to have more channels and more groups of Convolutional, Batch Normalization and LeakyReLU activation layers, but intertwined with some downsampling layers, so as to reduce the required memory. We tested variations of this architecture and found the results to be inferior.

5 Results

Contents

In this chapter we will evaluate the results obtained, both in their realism and in how well they translate the original low-level sketch from which the maps were generated.

All tests were done on a server using a Intel(R) Xeon(R) W-2223 CPU with 96 GB's of RAM and two GeForce RTX 3090 GPU's, however, only one was ever used at a time.

All the experiments described are compiled with the RMSProp optimizer, with a learning rate of 0.00025 for the Conversion GAN and 0.0005 for the Realism GAN. Following the results of this thesis' predecessor [Nunes et al., 2022] we trained each model for 5000 epochs.

5.1 Conversion Evaluation

In this section we will evaluate our model based on how well the content of the final height-map matches the content of the input from which it was generated. After acquiring results we determined that a simple empirical observation was enough to ascertain how closely these contents match. The reasoning behind this decision is that we believe that tools such as ours, that cater to human perception, are best evaluate by it.

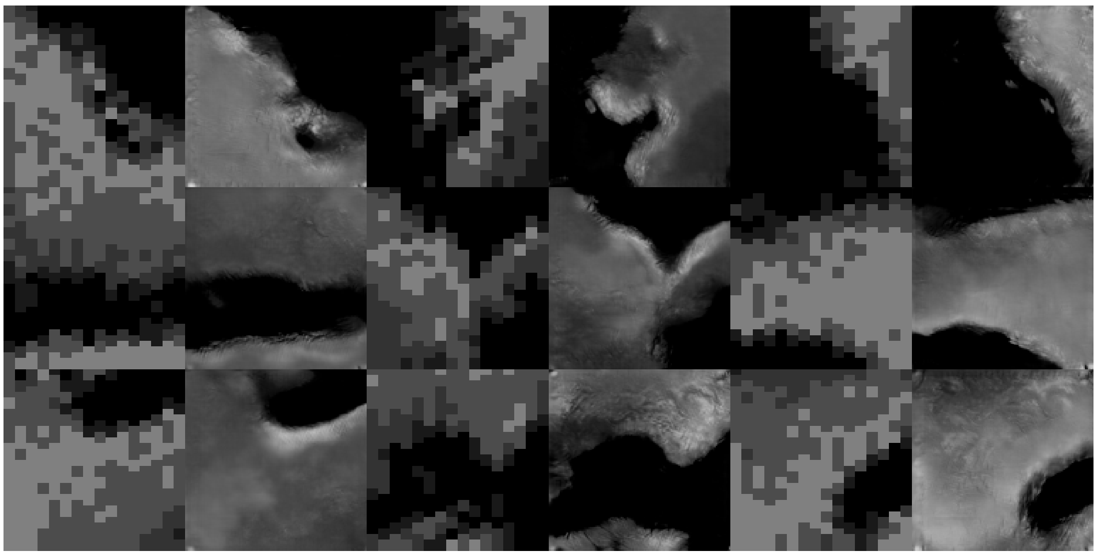


Figure 5.1: Examples of images generated from our final model.

In Figure 5.1 we can observe some maps generated from our final model. We believe these results reproduce the given content successfully, with little artifacts or added noise. We can observe that the model maintains the general profile of the terrain while adding some texture. It should be noted that the contour of the coastline is kept identical to the IMR used, while the elevation within land differs slightly, fluctuating depending on the noise vector given, Section 5.1.2 shows in further detail the effect of this noise.

5.1.1 Validation data-set

In order to test how well the model creates any type of terrain, and to ensure it wasn't overfitting, we created a separate data-set, comprised of 32 hexagonal maps, that were subsequently translated into the IMR format.

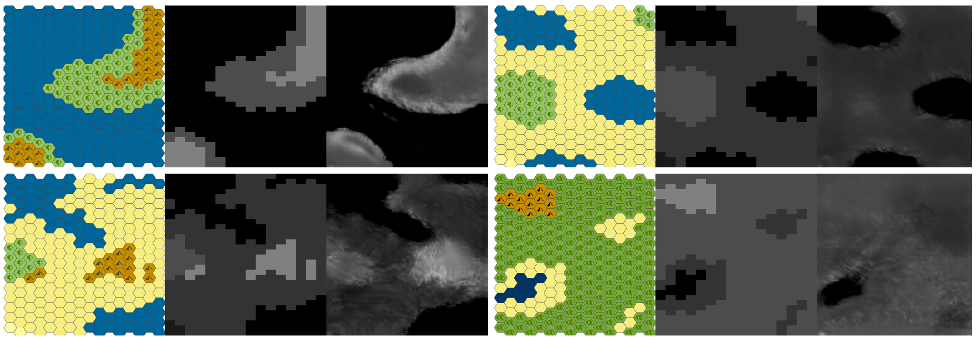


Figure 5.2: Examples of maps generated from the custom IMR data-set. For each set: **Left:** Original hex map. **Center:** IMR. **Right:** Resulting height-map.

Figure 5.2 represents some of the results obtained from the model when given IMR representations not present in the training data. This data is problematic to the model, not only because it was never observed, but also because it can express terrain that would be completely unrealistic. The quality of these results is lower than that of the observed data, as would be expected, however, when generating multiple samples of the same input some have quality on par with the previously observed results, demonstrating the importance of the noise vector.

5.1.2 Effect of the Noise Vector

As described in Section 3.1.1.A the Generator allows a random input vector, this vector requires a careful balancing of the model as, if the vector is not given enough importance, then there will be little randomness in the images, in the extreme case each IMR input can only generate a single output map. On the other hand, if the noise vector is given too much importance then the IMR input will not be respected and the output map will not represent the given input.

From Figure 5.3 it can be seen that the noise vector has a significant impact in the topology of the maps created, while at the same time maintaining a the general content present in the supplied IMR file. This is a crucial feature of the system as some of the maps may have undesired features, in spite of their semblance to the given input. The existence, and correct functioning of the noise vector allows the user to maintain most features but receive a different result, with the noise vector acting as a style guide.

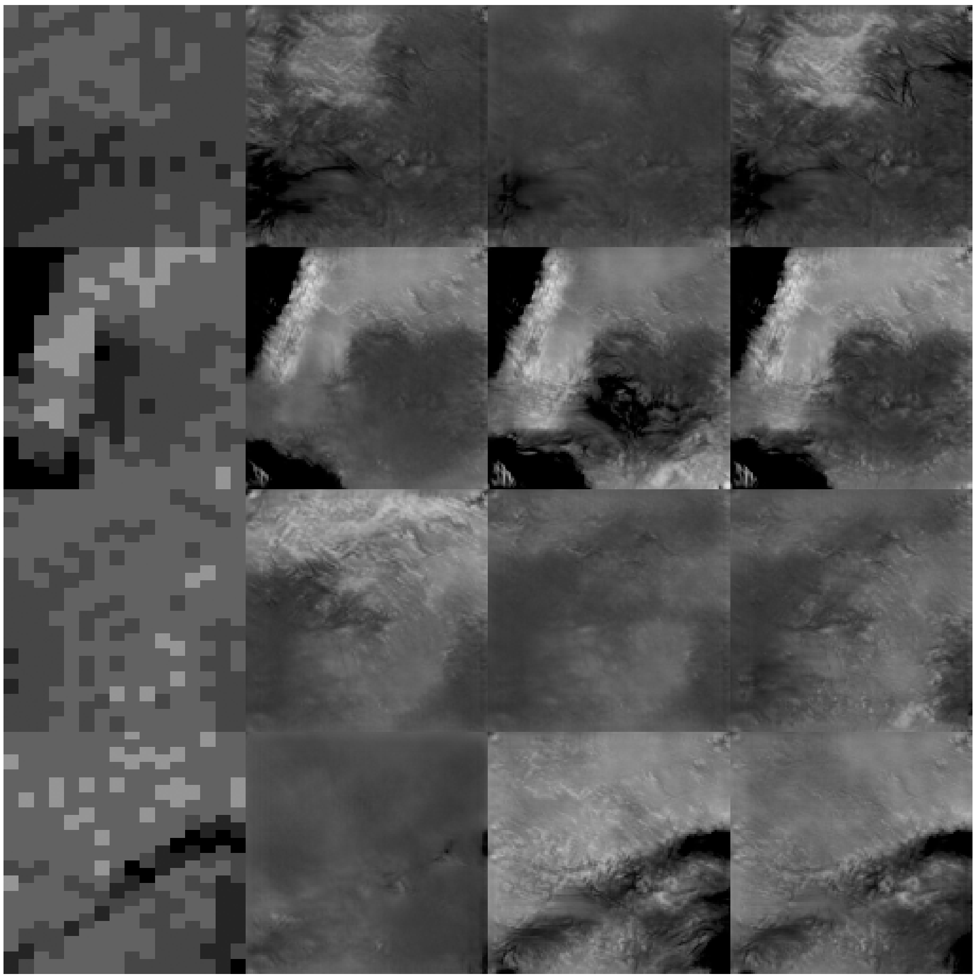


Figure 5.3: Influence of the noise vector on map generation. Left-most image is the IMR input from which the other images in the same row are generated.

5.1.3 Artifacts and Shortcomings of the Generated Maps

Most maps generated do not contain significant artifacts or other structures that may giveaway the fact that these images are generated, however, some images do contain such structures, with the most common being a small islet, as present in Figure 5.4. While this type of terrain exists on Earth (and in the SRTM data-set, where the model learned this structure) it is not as common as it is in images generated from our model. The other main limitation of our model's results comes from the types of structures it is not able to reproduce, with the most flagrant omission being rivers. We believe that rivers are notoriously difficult for networks to reproduce faithfully, as they exist not only as local information (i.e. their shape), but they also present a more global logic - they flow from higher elevation positions to the ocean - and this logic may be harder for the model to learn. While some rivers are present in the results generated, they tend to remain in the same location, rather than from the source to the ocean, however, the more common result is that the network simply neglects the creation of rivers.

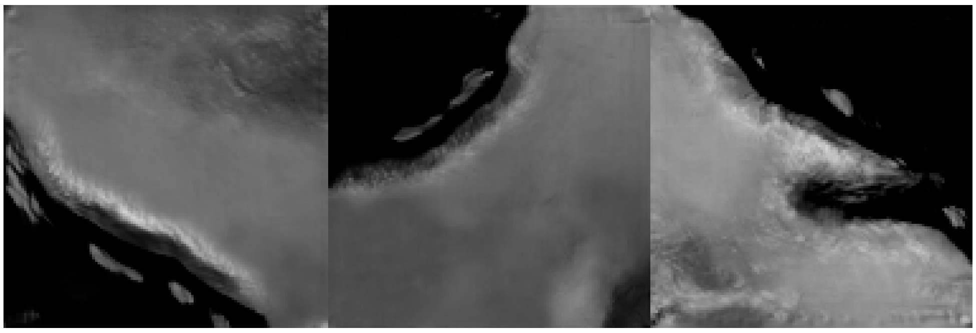


Figure 5.4: Formations generated along the coastline.

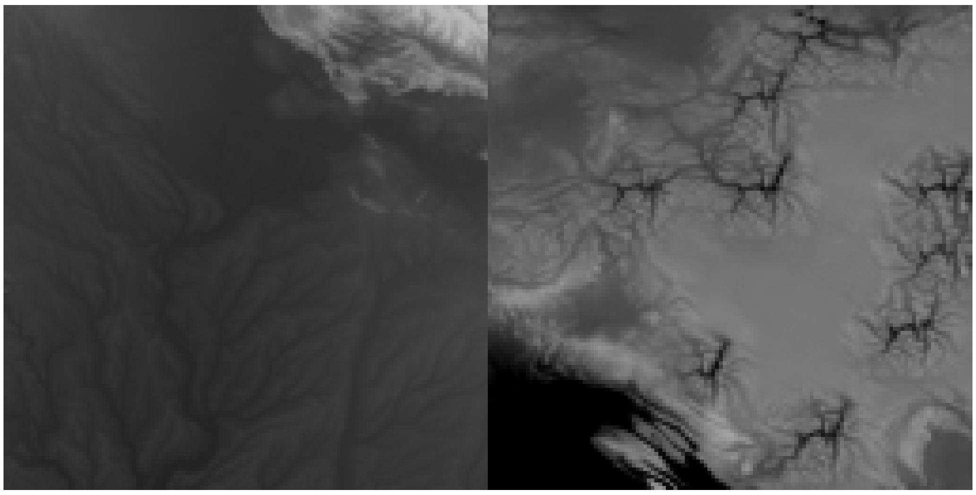


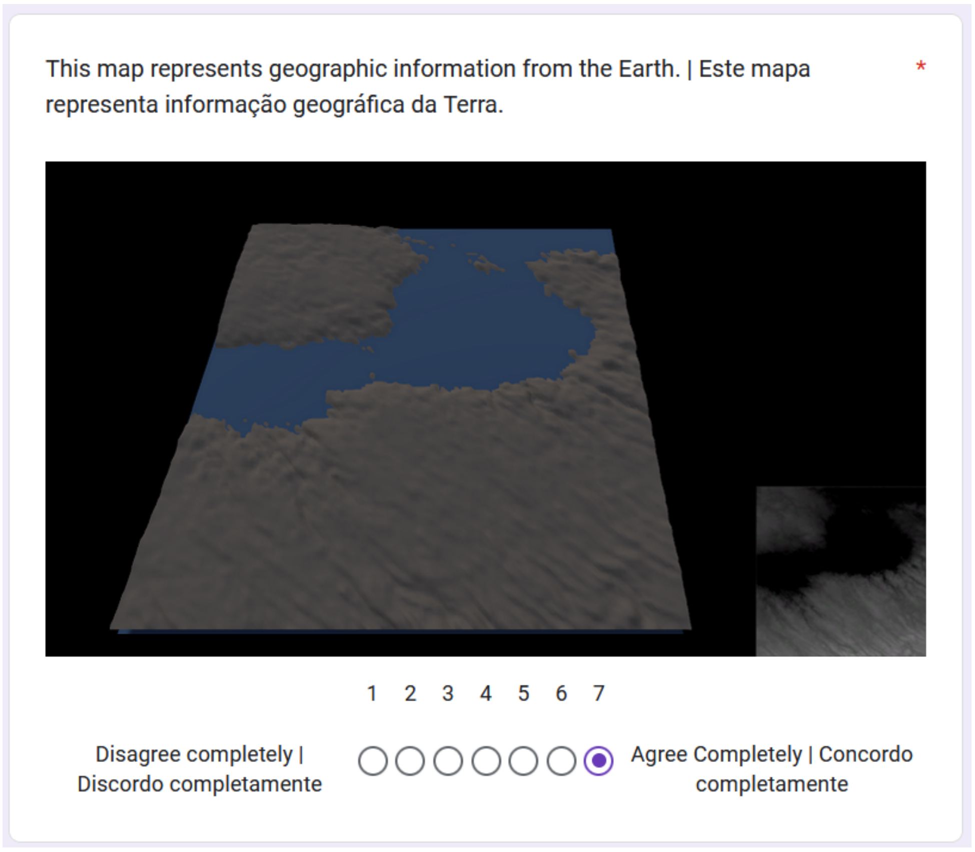
Figure 5.5: Left: Rivers as present in the SRTM data-set. Right: Rivers generated by our model.

5.2 Realism Evaluation

In this section we will explain our methodology for evaluating the results obtained on a visual basis, which we chose to evaluate through a short questionnaire, with the goal of confirming whether or not users can distinguish maps generated by our system from those present on Earth.

5.2.1 Test Overview

In order to evaluate our results on the goal of generating realistic images we conducted a user test where participants were shown 20 height-maps and corresponding 3D-renders, 10 of which from the NASA SRTM data-set, and 10 generated by our system. Participants were then asked to evaluate the origin of each map, using the sentence “This map represents geographic information from the Earth” and asking participants how much they agree with the sentence, in a Likert scale of 1 to 7; an example question can be seen in Figure 5.6. We considered the definition of “realistic” to be “representing things in a way that is accurate and true to life”, therefore, our experiment is more successful the closer the user evaluations are, across both sets of data (ground-truth and system generated).


This map represents geographic information from the Earth. | Este mapa representa informação geográfica da Terra. *

Disagree completely | Discordo completamente 1 2 3 4 5 6 [✓ 7] Agree Completely | Concordo completamente

Figure 5.6: Example of a question from the user test. The remaining questions follow this format, changing only the image.

5.2.2 Result Analysis

We obtained a total of 79 participants in the study, which we consider an acceptable value. We started by analysing the Chronbach alpha of the evaluations of all ground-truth maps and all maps generated by our system, obtaining a value of 0.889 and 0.898, respectively. These values of Chronbach alpha fall within the category of "good" internal consistency, nearing the threshold of "excellent", for which a value of 0.9 is required. The consequence of this result is that we can aggregate both categories of maps, Earth's and ours, using their mean value, which is graphed in Figure 5.7.

We then performed the Shapiro-Wilk normality test on the means obtained, resulting in p-values of 0.032 and 0.013, for ground-truth and system generated means respectively, indicating that our data does not follow a normal distribution.

Because our data is non-parametric, we chose to evaluate the populations using the Wilcoxon's paired rank test, from which we determined that there was no statistical difference between the two

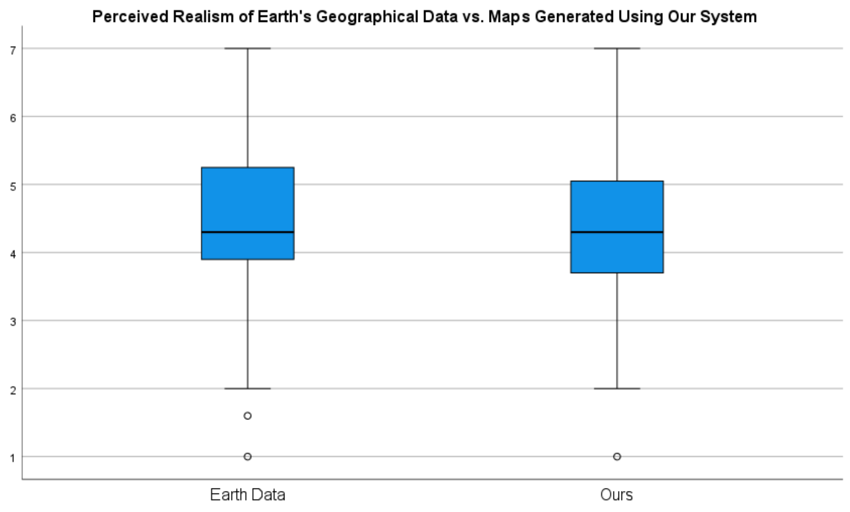


Figure 5.7: User perception of realism of maps generated with Earth's geographical data, vs. generated by our system.

populations, in other words, participants were unable to distinguish between ground-truth maps and maps generated using our system $(Z = -0.399, p = 0.69)$.

5.3 Results of Architecture Variations

In this section we will discuss the results obtained from the three architecture variations explained in Section 3.4, both in terms of their visual quality, and how well they translate the content of their input. Please note that each of these variations was tested separately, resulting in a total of three different experiments, not counting the final results, which serves as a baseline.

5.3.1 Bilinear Upscaling of IMR

After analysing the results of this experiment, from which Figure 5.8 is a subset, we concluded that the use of bilinear interpolation in the upsampling layers contributes negatively to the visual quality of the images generated. We believe that this decrease in quality is caused by an increase in the amount of values expressed by the IMR as a result of the smoother interpolation, which in turn may difficult learning.

Another observation of note is that this alteration creates patterns, which are especially evident in large, high altitude areas. One possible explanation for this fact is that this form of interpolation reduces the amount of times this type of areas are present, as sometimes they are interpolated to a lower altitude as a result of the neighbouring terrain, in turn, this may cause the model to learn the representation of this terrain from areas exhibiting a similar pattern.

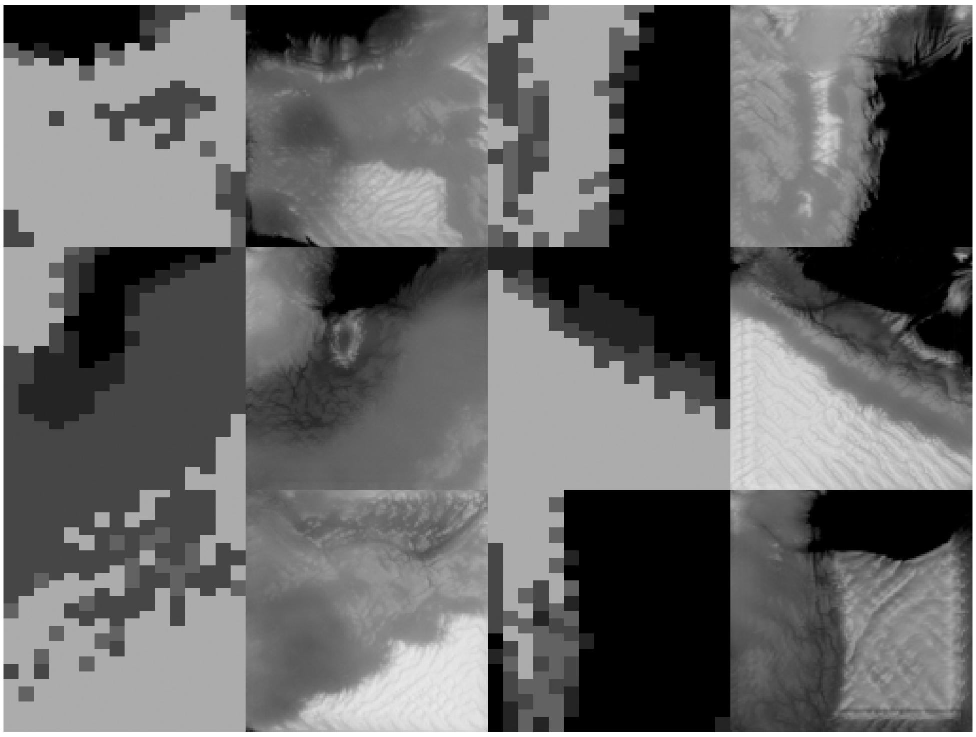


Figure 5.8: Results from bilinear interpolation experiment.

5.3.2 Forced Conversion Critic training

In this experiment we tested a forced schedule that didn't allow either GAN to train for more than 10 epochs consecutively, which consequently means that each GAN is trained at least once every 10 epochs. We verified that the results declined in visual quality, which we believe to come from the fact that the Conversion GAN is not apt to maintain visual quality, and while the results may be more aligned to the content of the IMR, this is not particularly noticeable after a certain point, and serves only to limit the freedom of the Generator to create consistent and visually appealing images.

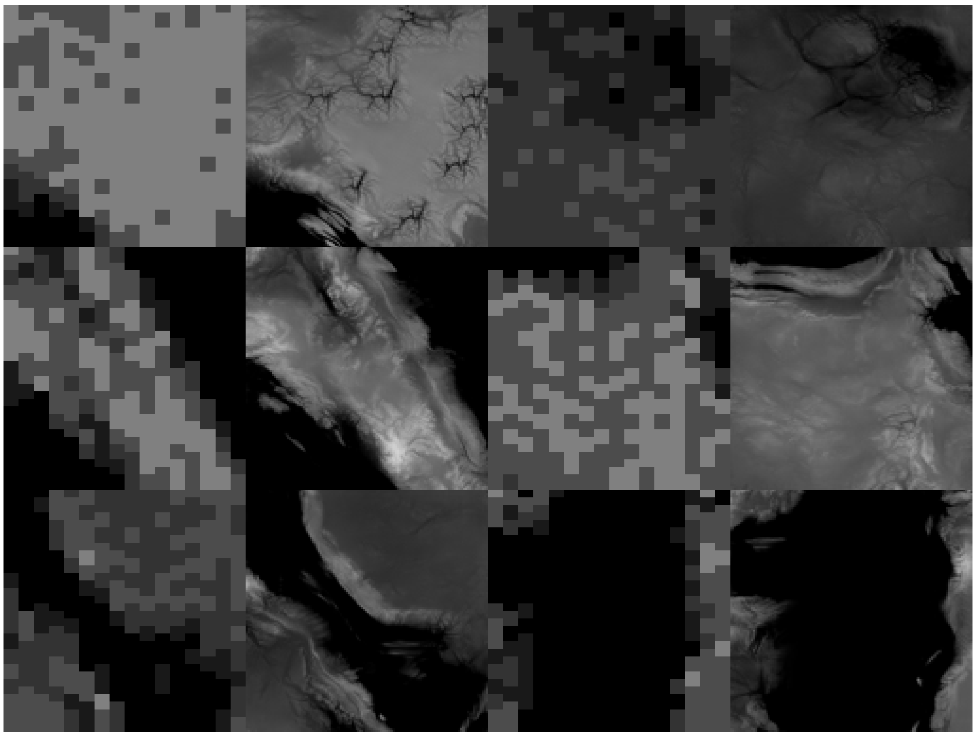


Figure 5.9: Results from forced conversion training experiment.

5.3.3 Epoch-based Scheduling

In this experiment we schedule which GAN to train at the start of each epoch, and train that GAN for the whole epoch. While this seems like the most obvious approach it has both upsides and downsides when compared to scheduling each batch. After conducting the experiment and observing the results acquired (Figure 5.10) we observe that often throughout training the content of the IMR and the resulting output stop matching, more specifically, the content matches with the opposite type of terrain: oceans become mountains, and vice-versa. We believe this is caused by an excessively high learning-rate of the Conversion GAN, which, over the course of an epoch flips the sign of some neurons, thereby causing the observed mismatch. For the final results we opted to schedule training on a per-batch basis, as opposed to lowering the learning rate, as we observed this former approach did not exhibit any of the problems theorised in Section 3.4.3.

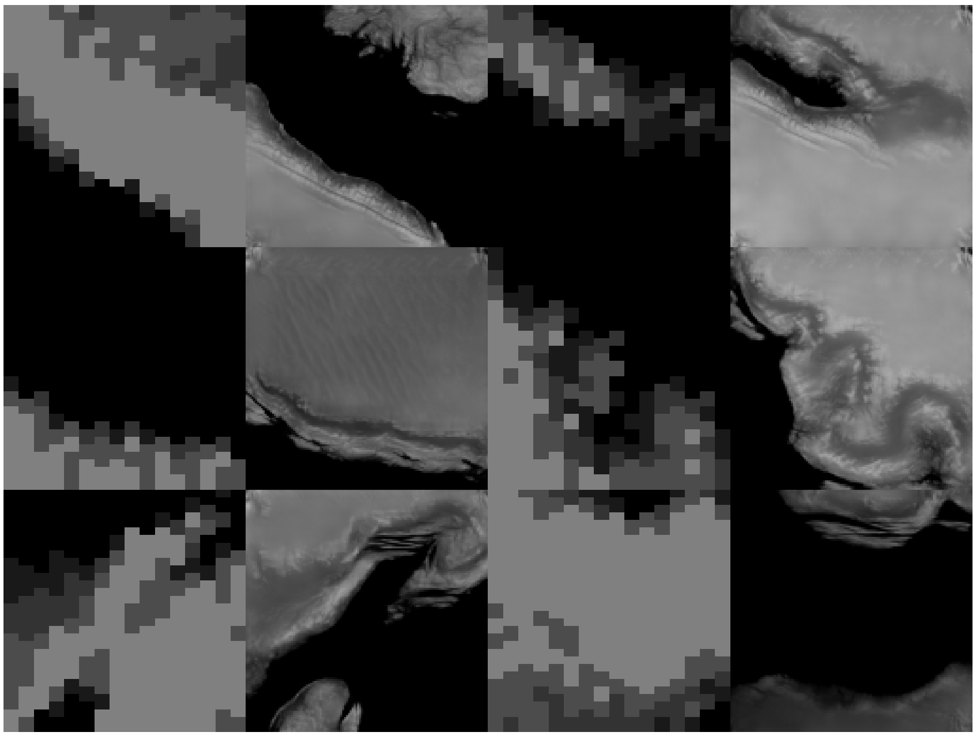


Figure 5.10: Results from the Epoch-based Scheduling experiment.

6 Conclusion

Contents

6.1 Conclusion

6.1.1 Final Remarks

Our goal for this thesis was to provide an alternative to current ways of generating height-maps for video-games. We needed a system that would create realistic and visually appealing results, while not requiring too much work or knowledge from the part of user, but at the same time allowing the user to specify exactly which geographical features should be present in the end result, and where. In order to accomplish this goal we designed and implemented a system that would take a very generic format, able to be converted to from a wide variety of existing tools, and generates a realistic height-map that contains the same features present in the supplied sketch. This generation is done using our proposed model, the Dual Critic Conditional Wasserstein Generative Adversarial Network (DCCWGAN), a new type of conditional WGAN using two critics: The first Critic to evaluate the content of the input matches that of the generated map, while the second Critic guides the results to be more and more realistic. We then evaluated our system, determining through empirical observation that the content of the outputted maps closely match those of the supplied input. We also conducted user tests, from which we were then able to prove that users are unable to distinguish between maps we generated and maps created from geographical information of the Earth, which was the standard we set for realism. Overall, we consider that, while there is room for improvement, we achieved the goals we set out for, and contributed to existing knowledge by implementing a system that performs a form of image-to-image translation using multiple critics.

6.1.2 System Limitations and Future Work

While we consider the results achieved suitable, these are not without their limitations. The first great limitation comes from the resolution of the maps generated, which is merely a 128×128 image. While the system should theoretically work on larger images, this would require more memory, which in turn would increase the time required to train.

Another large limitation is the amount of control given to the user, who currently can only change the height the map. While some features such as vegetation would be simple to implement through a post-processing phase using PCG techniques, other features such as rivers should be included in the IMR, to be used by the networks themselves, this would, however, require a more complex algorithm to translate the ground-truth data-set into the IMR format as our approach is unable to detect rivers due to their relatively small effect on the elevation of that area.

Bibliography

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan.

[Brock et al., 2019] Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale GAN training for high fidelity natural image synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[Dertat, 2017] Dertat, A. (2017). Applied deep learning - part 4: Convolutional neural networks. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2, last accessed on 25/10/2022.

[Feng et al., 2020] Feng, J., Feng, X., Chen, J., Cao, X., Zhang, X., Jiao, L., and Yu, T. (2020). Generative adversarial networks based on collaborative learning and attention mechanism for hyperspectral image classification. *Remote Sensing*, 12(7).

[Goodfellow et al., 2017] Goodfellow, I., Bengio, Y., and Courville, A. (2017). *Deep learning*. MIT Press.

[Isola et al., 2017] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134.

[Johanson, 2004] Johanson, C. (2004). Real-time water rendering. Master's thesis, Lund University.

[Karras et al., 2018] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018). Progressive growing of gans for improved quality, stability, and variation.

[Karras et al., 2020] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Lyu et al., 2017] Lyu, P., Bai, X., Yao, C., Zhu, Z., Huang, T., and Liu, W. (2017). Auto-encoder guided gan for chinese calligraphy synthesis. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 1095–1100. IEEE.

[Mahapattanakul, 2019] Mahapattanakul, P. (2019). From human vision to computer vision-convolutional neural network(part3/4). https://becominghuman.ai/from-human-vision-to-computer-vision-convolutional-neural-network-part3-4-24b55ffa7045, last accessed on 25/10/2022.

[Millington, 2019] Millington, I. (2019). *Artificial intelligence for games*. CRC Press.

[Mirza and Osindero, 2014] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *CoRR*, abs/1411.1784.

[Miyato et al., 2018] Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[Nunes et al., 2022] Nunes, V., Dias, J., and Santos, P. (2022). Gan-based content generation of maps for strategy games. In *GAME-ON 2022*.

[Park et al., 2019] Park, T., Liu, M.-Y., Wang, T.-C., and Zhu, J.-Y. (2019). Gaugan: Semantic image synthesis with spatially adaptive normalization. In *ACM SIGGRAPH 2019 Real-Time Live!*, SIGGRAPH '19, New York, NY, USA. Association for Computing Machinery.

[Pedrycz and Chen, 2020] Pedrycz, W. and Chen, S.-M. (2020). *Deep Learning: Algorithms and Applications*. Springer International Publishing.

[Richardson et al., 2021] Richardson, E., Alaluf, Y., Patashnik, O., Nitzan, Y., Azar, Y., Shapiro, S., and Cohen-Or, D. (2021). Encoding in style: A stylegan encoder for image-to-image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2287–2296.

[Veen, 2016] Veen, F. V. (2016). The neural network zoo. https://www.asimovinstitute.org/neural-network-zoo/, last accessed on 25/10/2022.

[Wang et al., 2018] Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Tao, A., Kautz, J., and Catanzaro, B. (2018). High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

A

Network Structures

In this appendix we present the full structure of all the networks used in to obtain the final results.

Table A.1: Generator structure of the GAN model used

Layer name	Act.	Stride	Kernel size	Input shape	Output shape
IMR Input Branch	-	-	-	-	$32 \times 32 \times 1$
Deconv[1]	-	1	3	$32 \times 32 \times 1$	$32 \times 32 \times 128$
LeakyReLU	LeakyReLU	-	-	$32 \times 32 \times 128$	$32 \times 32 \times 128$
Noise Input Branch	-	-	-	-	$128 \times 1 \times 1$
Dense	-	-	-	$128 \times 1 \times 1$	$1024 \times 1 \times 1$
Reshape	-	-	-	$1024 \times 1 \times 1$	$32 \times 32 \times 1$
Concatenate	-	-	-	$32 \times 32 \times (128 + 1)$	$32 \times 32 \times 129$
Upsampling	-	-	-	$32 \times 32 \times 129$	$64 \times 64 \times 129$
Upsampling	-	-	-	$64 \times 64 \times 129$	$128 \times 128 \times 129$
Deconv	-	1	4	$128 \times 128 \times 129$	$128 \times 128 \times 64$
BatchNorm	-	-	-	$128 \times 128 \times 64$	$128 \times 128 \times 64$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 64$	$128 \times 128 \times 64$
Deconv	-	1	4	$128 \times 128 \times 64$	$128 \times 128 \times 128$
BatchNorm	-	-	-	$128 \times 128 \times 128$	$128 \times 128 \times 128$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 128$	$128 \times 128 \times 128$
Deconv	-	1	4	$128 \times 128 \times 128$	$128 \times 128 \times 256$

[1] Dilation rate = 2

BatchNorm	-	-	-	$128 \times 128 \times 256$	$128 \times 128 \times 256$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 256$	$128 \times 128 \times 256$
Deconv	-	1	4	$128 \times 128 \times 256$	$128 \times 128 \times 128$
BatchNorm	-	-	-	$128 \times 128 \times 128$	$128 \times 128 \times 128$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 128$	$128 \times 128 \times 128$
Deconv	-	1	4	$128 \times 128 \times 128$	$128 \times 128 \times 64$
BatchNorm	-	-	-	$128 \times 128 \times 64$	$128 \times 128 \times 64$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 64$	$128 \times 128 \times 64$
Deconv	-	1	4	$128 \times 128 \times 64$	$128 \times 128 \times 32$
BatchNorm	-	-	-	$128 \times 128 \times 32$	$128 \times 128 \times 32$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 32$	$128 \times 128 \times 32$
Deconv	-	1	4	$128 \times 128 \times 64$	$128 \times 128 \times 16$
BatchNorm	-	-	-	$128 \times 128 \times 16$	$128 \times 128 \times 16$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 16$	$128 \times 128 \times 16$
Conv	tanh	1	5	$128 \times 128 \times 16$	$128 \times 128 \times 1$

Table A.2: Conversion Critic structure of the GAN model used

Layer name	Act.	Stride	Kernel size	Input shape	Output shape
IMR Input Branch	-	-	-	-	$32 \times 32 \times 1$
Conv[2]	-	1	3	$32 \times 32 \times 1$	$32 \times 32 \times 32$
BatchNorm	-	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
LeakyReLU	LeakyReLU	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
Conv	-	1	4	$32 \times 32 \times 32$	$32 \times 32 \times 32$
BatchNorm	-	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
LeakyReLU	LeakyReLU	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
Image Input Branch	-	-	-	-	$128 \times 128 \times 1$
Conv	-	1	4	$128 \times 128 \times 1$	$128 \times 128 \times 32$
BatchNorm	-	-	-	$128 \times 128 \times 32$	$128 \times 128 \times 32$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 32$	$128 \times 128 \times 32$
Conv	-	2	4	$128 \times 128 \times 32$	$64 \times 64 \times 32$
BatchNorm	-	-	-	$64 \times 64 \times 32$	$64 \times 64 \times 32$
LeakyReLU	LeakyReLU	-	-	$64 \times 64 \times 32$	$64 \times 64 \times 32$
Conv	-	2	4	$64 \times 64 \times 1$	$32 \times 32 \times 32$
BatchNorm	-	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
LeakyReLU	LeakyReLU	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
Concatenate	-	-	-	$32 \times 32 \times (32 + 32)$	$32 \times 32 \times 64$
Conv	-	1	4	$32 \times 32 \times 64$	$32 \times 32 \times 32$
BatchNorm	-	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
LeakyReLU	LeakyReLU	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
Conv	-	1	4	$32 \times 32 \times 32$	$32 \times 32 \times 32$
BatchNorm	-	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
LeakyReLU	LeakyReLU	-	-	$32 \times 32 \times 32$	$32 \times 32 \times 32$
Conv	-	1	4	$32 \times 32 \times 32$	$32 \times 32 \times 16$
BatchNorm	-	-	-	$32 \times 32 \times 16$	$32 \times 32 \times 16$
LeakyReLU	LeakyReLU	-	-	$32 \times 32 \times 16$	$32 \times 32 \times 16$
Flatten	-	-	-	$128 \times 128 \times 256$	$16384 \times 1 \times 1$
Dense	linear	-	-	$16384 \times 1 \times 1$	$1 \times 1 \times 1$

[2]Dilation rate = 2

Table A.3: Realism Critic structure of the GAN model used

Layer name	Act.	Stride	Kernel size	Input shape	Output shape
Image Input Branch	-	-	-	-	$128 \times 128 \times 1$
Conv	-	1	4	$128 \times 128 \times 1$	$128 \times 128 \times 256$
BatchNorm	-	-	-	$128 \times 128 \times 256$	$128 \times 128 \times 256$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 256$	$128 \times 128 \times 256$
Conv	-	1	4	$128 \times 128 \times 256$	$128 \times 128 \times 128$
BatchNorm	-	-	-	$128 \times 128 \times 128$	$128 \times 128 \times 128$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 128$	$128 \times 128 \times 128$
Conv	-	1	4	$128 \times 128 \times 128$	$128 \times 128 \times 64$
BatchNorm	-	-	-	$128 \times 128 \times 64$	$128 \times 128 \times 64$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 64$	$128 \times 128 \times 64$
Conv	-	1	4	$128 \times 128 \times 64$	$128 \times 128 \times 32$
BatchNorm	-	-	-	$128 \times 128 \times 32$	$128 \times 128 \times 32$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 32$	$128 \times 128 \times 32$
Conv	-	1	4	$128 \times 128 \times 32$	$128 \times 128 \times 16$
BatchNorm	-	-	-	$128 \times 128 \times 16$	$128 \times 128 \times 16$
LeakyReLU	LeakyReLU	-	-	$128 \times 128 \times 16$	$128 \times 128 \times 16$
Flatten	-	-	-	$128 \times 128 \times 16$	$262144 \times 1 \times 1$
Dense	linear	-	-	$262144 \times 1 \times 1$	$1 \times 1 \times 1$

B

Generated Images

In this appendix we present a sample of the images generated by our final model, as well as the architecture variations.

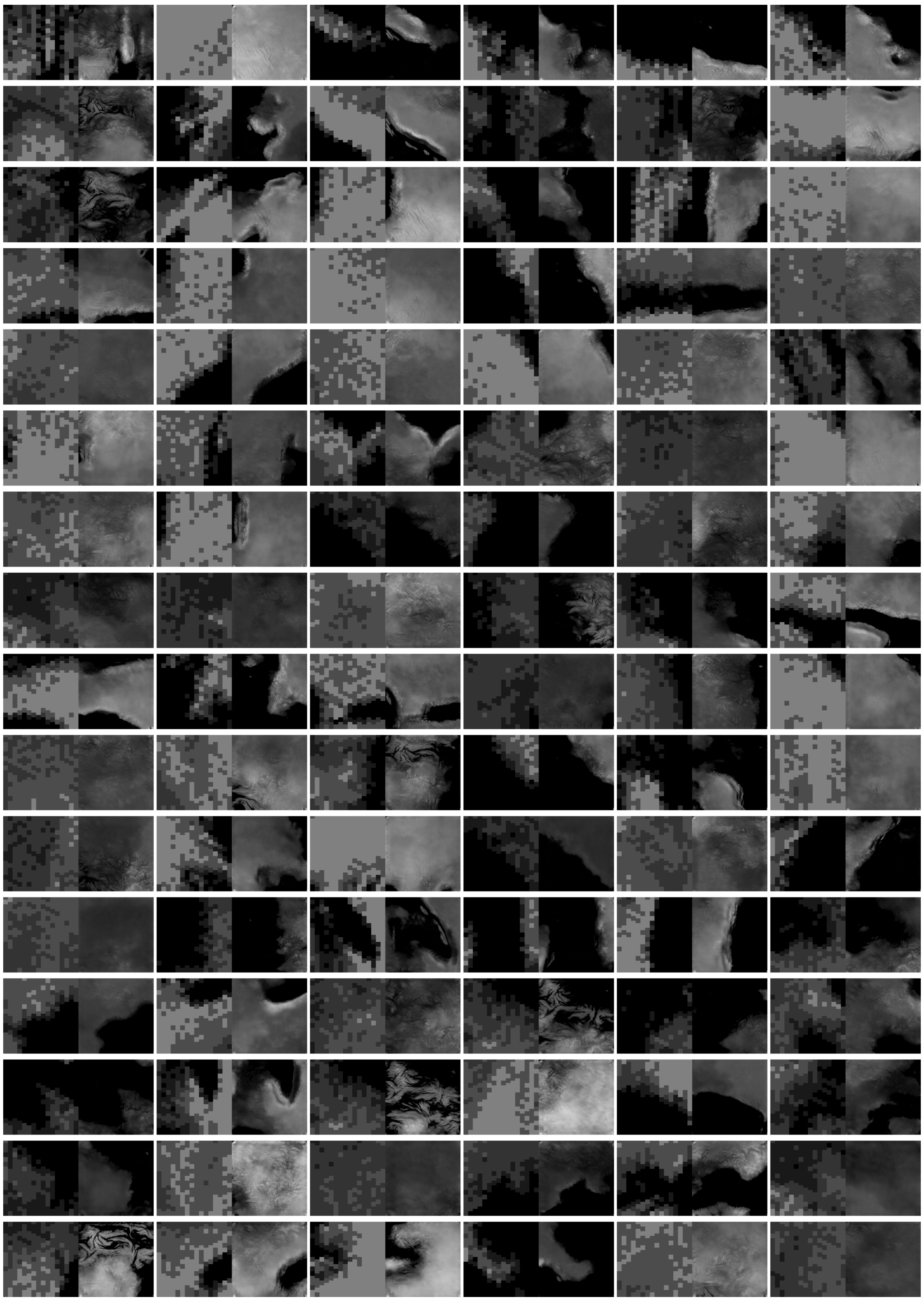


Figure B.1: Results obtained during the final epochs of training, using what we consider to be the best parameters.

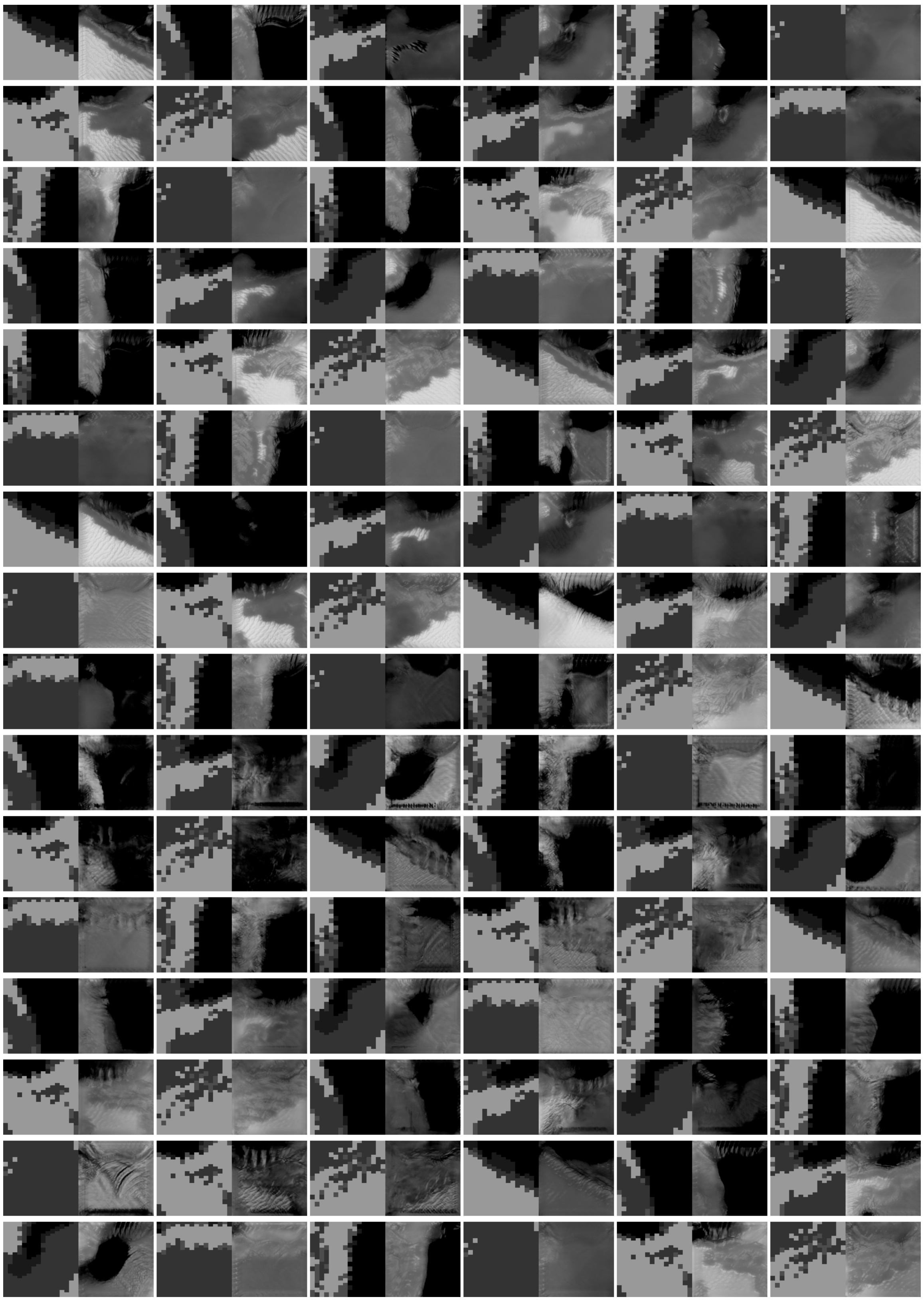


Figure B.2: Results obtained during the final epochs of training, using bilinear upsampling layers.

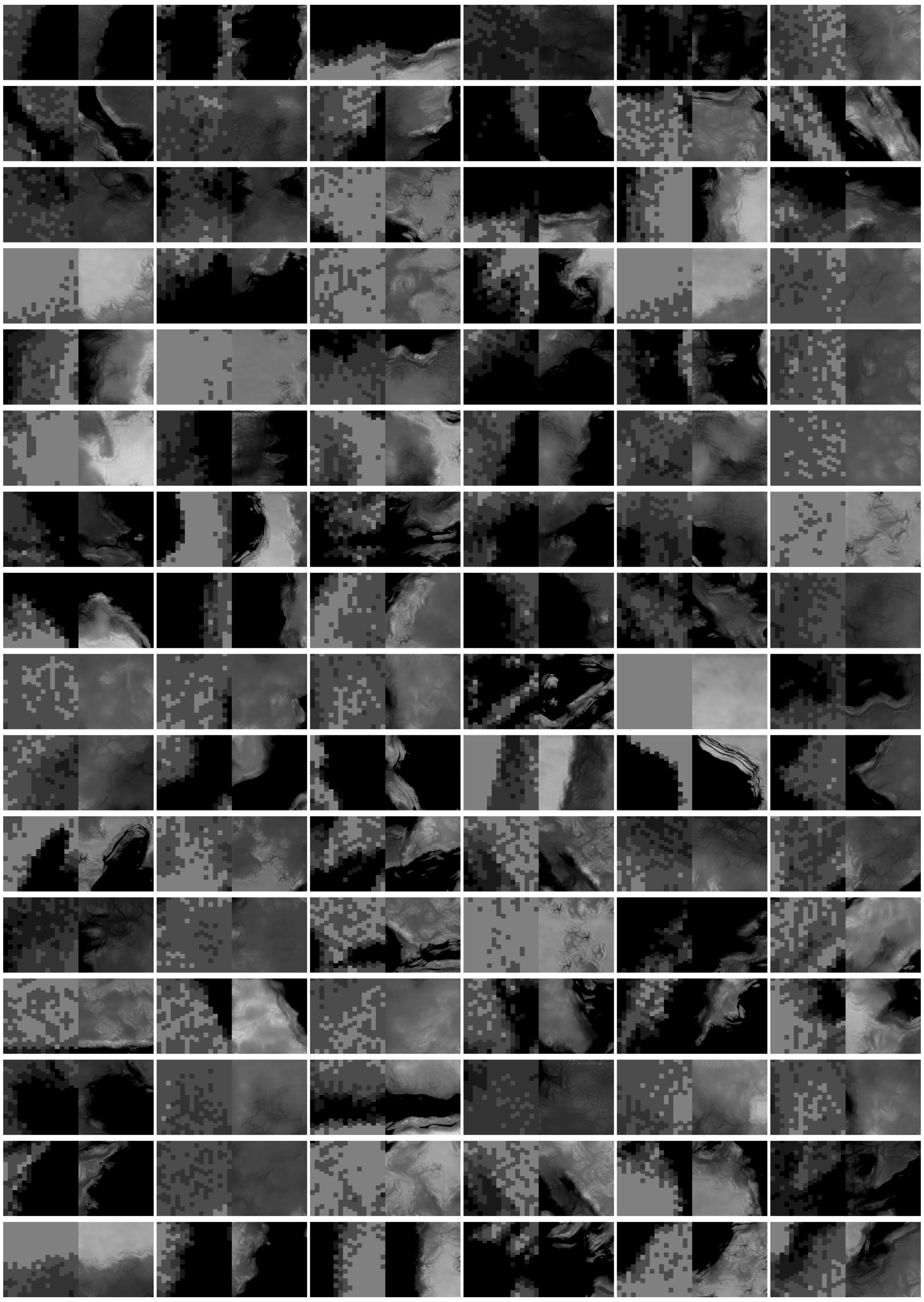


Figure B.3: Results obtained during the final epochs of training, using forced content training.

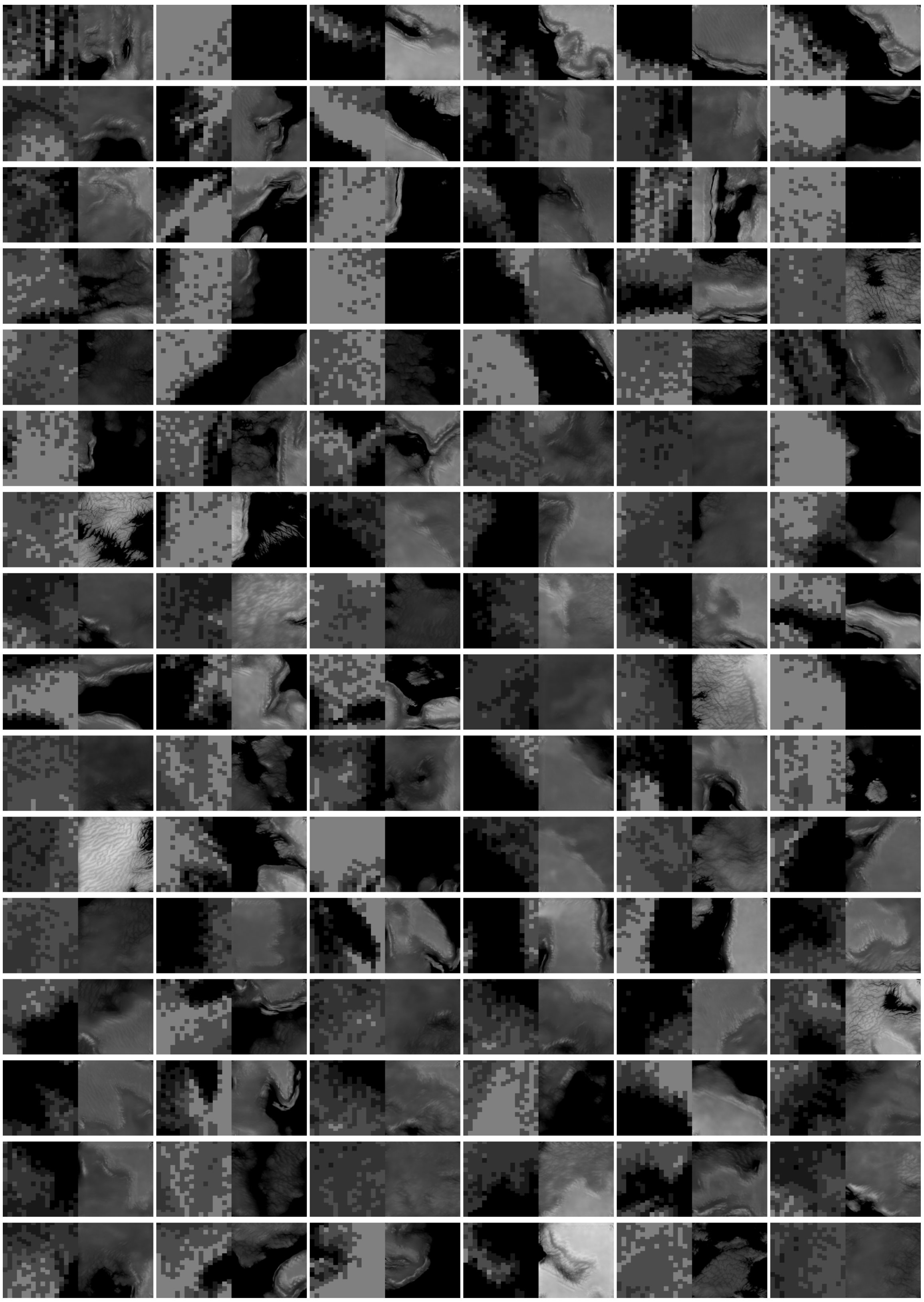


Figure B.4: Results obtained during the final epochs of training, using per-epoch scheduling.