MASTER'S THESIS 2023

Generating abstract art using artificial neural networks

Henrik Norrman

ISSN 1650-2884
LU-CS-EX: 2023-52

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-52

Generating abstract art using artificial neural networks

Abstrakt konstgenerering med hjälp av artificiella neurala nätverk

Henrik Norrman

Generating abstract art using artificial neural networks

Henrik Norrman
he5012no-s@student.lu.se

December 8, 2023

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Daniel Byström, daniel.bystrom@elastx.se
Christin Lindholm, christin.lindholm@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

Artificial intelligence (AI) tools are slowly becoming integral to our everyday lives. Today we already rely on AI for much of our navigation and security. In the near future, much like the internet today, AI and machine learning tools are likely to assist us in almost every aspect of our lives, both with our practical and creative tasks. In recent years, there has been a surge in AI projects focused on creative pursuits, such as music production, story writing, and art creation.

This thesis aims to contribute to the discussion about what it might look like when AI is utilised to tackle the subjective task of creativity.

A two-step method of generation is proposed—consisting of a base image generator and an upscaling model. The base image generator is built using a generative adversarial network (GAN), trained on abstract art in the public domain. The generator is designed to generate base images, small images capturing the theme of an artist's work. To finalise the generation, the base image is passed to a second GAN model for upscaling. During the construction of this method, multiple problems surfaced. The empirical testing of potential solutions to these problems eventually resulted in the proposed two-step method. Due to its design, the default GAN architecture used as the foundation of the prototype proved to be a misguided choice, as directly generating any images larger than 512x512 required hardware more powerful than what was available.

Keywords: AI, ML, GAN, generator, discriminator, style transfer, machine learning, generative model

Acknowledgements

I'm grateful for the many people who have helped and supported me throughout this thesis—both the people who have been there since the start, but also those who've contributed with their thoughts and ideas along the way.

I want to thank Christin Lindholm, my supervisor at LTH, for all her effort, especially towards the end, where she played a crucial role in helping me finish the thesis.

I would also like to thank Daniel Byström, my supervisor at Posterton. His invaluable ideas and support helped continuously progress the work, even past all the demotivating dead-ends.

I'm also grateful to Mattias Skog, the CEO of Posterton, for enabling this thesis and allowing me the freedom to explore and learn everything I did.

A big thank you to André Åström, who was always eager to help and support me with whatever he could, with both technical solutions and motivation.

Lastly, I would like to thank my partner, Emelie Elvin, whose love and support was essential in finishing this thesis. I could not have done it without your help and patience.

Henrik Norrman
March, 2023

Contents

Chapter 1
Introduction

This thesis was carried out in collaboration with Posterton, a Stockholm based company selling posters and paintings online. As all their business is conducted through their website, and all purchased artwork is printed on-demand, new images can easily be added to their gallery without any investments in printed stock.

During COVID-19, as many people were forced into lockdown, general interest for home decoration increased. According to an analysis by Fortune Business Insight, the global wall art market is projected to grow from 48.50 billion USD in 2021 to 72.61 billion USD by 2028 [14].

By expanding their business with AI, Posterton can both accommodate for the growing demand for wall art, and provide more personalised and unique options for their customers.

1.1 Background

Artificial intelligence (AI) and machine learning (ML) are rapidly growing fields of computer science, making tasks previously thought of as computationally impossible now easily achievable with everyday hardware. In recent years, there has been a surge in AI projects focused on creative pursuits, such as screenwriting, music production, story writing, and art [4]. The two terms, AI and ML, are now also widely used in the world of commerce as selling points for products entirely unrelated to the world of computer science [12].

There are many concerns that the bubble may eventually burst if the new technology can't live up to the many expectations, leading to a drastic decrease in AI funding and general interest [12]. In the early 2000s, the Internet faced this same type of crash after not living up to its hype. Yet today, many of us are unable to even imagine life without the internet. Similarly, AI assisted tools and applications are likely to aid us with almost every aspect of our lives in the future—with tasks both practical and creative [4].

1.2 Purpose

To help Posterton introduce AI as part of their business model, a prototype of a basic abstract art generator will be developed and trained. This prototype can then function as a basis for a generative system, where a GUI and other possible end-user functionality can be developed on top of it.

1.3 Goals

The goal of this thesis is, through the use of machine learning and a design science methology, develop and train a prototype that is capable of generating abstract art. As the definition of what qualifies as abstract art is highly subjective, no definite qualitative goals are set regarding the visual quality of the generated images. The only exception is that the generated artwork has to be of at least full HD.

1.4 Research questions

By answering the following questions, this thesis aims to investigate how machine learning tools can be used to generate abstract art:

RQ1 What steps and technology are required to develop an abstract art generator?

RQ2 Can generative adversarial networks (GAN) be used to generate high definition artwork?

RQ3 How long does it take to generate a single image; is generating on-demand art feasible?

1.5 Scope

Any training is limited to regular personal computers; no powerful specialised hardware is available. This may put constraints on the quality of the trained models. It can also limit the number of iterations trained and the size of the training data used.

To enable Posterton to freely use the generated images, the training data will be limited to artists whose work is in the public domain, i.e. artwork not protected by copyright laws or other legal restrictions. Further development could take the form of training more and better models, enabling NFT generation, adding customer feedback weights for better results, automatic image filtering, adjusting and improving the architecture, and experimenting with other art styles. These are all potential improvements worth considering, but are outside the scope of this thesis.

1.6 Related work

This thesis is mostly inspired by the project presented in the article *GANshare: Creating and Curating Art with AI for Fun and Profit* [7], which provided a basic structure and understanding

of the required models and networks. Robert's project did however use the *VQGAN*, a more recent and advanced improvement of the basic *GAN* framework. OpenAI's CLIP is also used, a state-of-the-art text-to-image (or image-to-text) network [23]. CLIP is used in the project to control the image generation, whereas in this thesis the works of famous abstract artists are used as the basis of the generation.

Established applications like OpenAI's *DALL-E 2*, and *WOMBO dream*, have also provided a lot of inspiration, and helped with problem solving issues. Neither OpenAI nor WOMBO have disclosed how their generators work, but their set of features and generated results have provided insight into various possibilities and limitations.

Chapter 2
Theory

To better understand the following chapters, this section aims to provide the reader with the necessary theoretical information about machine learning, and about the models and methods used in the thesis. Colloquially, artificial intelligence and machine learning are sometimes used interchangeably. However, the field of AI encompasses not only the entire field of machine learning, but many other sub-fields concerned with acting and reasoning (just without the learning aspect) [4]. Figure 2.1 shows the relationship between the three fields; AI, ML, and DL.

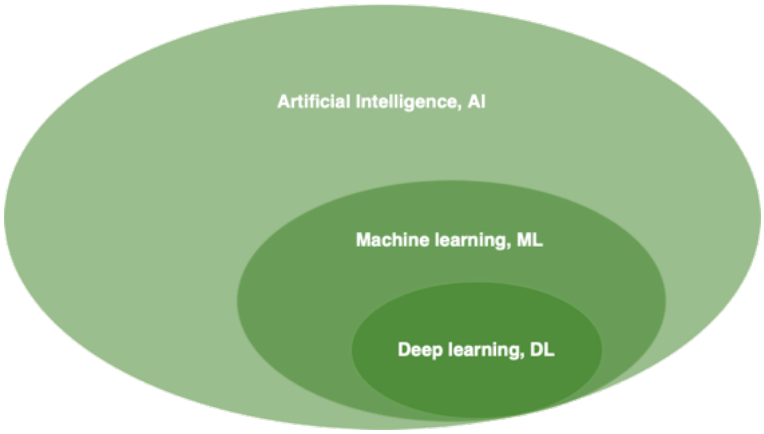


Figure 2.1: A Venn diagram of the relationship between AI, ML, and DL.

2.1 Machine learning

Machine learning algorithms are often divided into three categories; *supervised*, *unsupervised*, and *reinforcement* learning. **Supervised learning** is sometimes also referred to as *predictive*,

as it takes a number of input x_n, and predicts an output y. **Unsupervised learning**, or the *descriptive* algorithms, explore the data on its own, without any help from labels, and finds similarities and patterns in the data. **Reinforcement learning** acts based on a reward and punishment systems, and dynamically makes decisions based on it [21]. For this thesis, only *supervised learning* is relevant, as no unsupervised or reinforcement learning methods are used.

An example of supervised learning is the regression problem of predicting the cost of house y, given the variables x_n. Equation 2.1 describes the problem in it's most basic form, using only one variable:

$$w_0 + w_1 * x_{size} = y \tag{2.1}$$

Finding good values for the weight variables w_0, w_1 (the weight variable w_0 is sometimes called the bias), is the learning part of machine learning. The variable x_{size} describes the size of a house, and y is the selling price. The training data in this example would consist of the labels *size* and *price*.

To train the model, which means finding accurate values for the weight variables, a *loss function* is needed. This function quantifies the predicted error, and is used to direct the adjustment of the weights [24]. Function 2.2 is the *squared loss*, or L_2 function. Calculating this would result in the numerical number of the total difference between the predicted number and the ground truth—or simply, how wrong the predicted results are compared to the real values.

$$loss = \sum_{j=1}^{N} (y_j - (w_1 * x_j + w_0))^2 \tag{2.2}$$

More interesting than a numerical value is the function's derived properties. As a quick reminder in calculus, the derivative of a curve is the line tangent of that curve in any of its points. A high value of the tangent indicates a steep slope, whilst a zero indicates a plateau or minimum point.

Gradient descent, the most used algorithm in all of machine learning [10], updates the weights w_0, w_1 by using the derivative of the loss function to adjust the weights in the direction of the steepest slope. This is done iteratively until a satsifactory result to the loss function is reached, and the weights are considered optimal.

2.1.1 Deep learning

Deep learning is a sub-field of machine learning, where the word *deep* refers to it's hierarchical structure. The learning is distributed over multiple layers and neurons throughout the neural network, where every neuron solves it's own unique problem [4]. This structure forces the network to not only learn the weights, but also the features of the data (or what to weigh).

Convolutional neural networks

This thesis will mainly be concerned with convolutional neural networks (ConvNet or CNN), as a GAN is simply a combination of a convolutional neural network and a deconvolutional neural network, where a deconvolutional neural network is a ConvNet working in reverse.

The invention of ConvNets greatly improved the results in image classification tasks. The field of computer vision, including tasks such as autonomous driving, medical diagnosis, and facial recognition, heavily relies on ConvNets for their image processing [4].

Two important hyperparameters (i.e. variables controlling the process) of the ConvNets to consider are the **kernel size** and **stride**. The kernel is sometimes described as a small window through which the network looks at the image, and the kernel size defines how large this window is. Much like reading a page, the window scans from left to right, top to bottom, trying to make sense of the pixel patterns of the image. The stride determines how large the steps are. A stride of one means the kernel is only moved a single pixel at a time. If a stride of two is used, the kernel is instead centred around every other pixel. For every step, the convolution of the kernel sized matrix is calculated. Since the stride skips every other pixel, the results are saved in a new matrix of half the height and width [4].

A basic classification model implemented with a ConvNet, begins with a layer of the same size as the input image. This first layer learns the local pixel structure of the image. Gradually through the network, the width and height of the layers shrink, and the representation becomes more complex and less humanly understandable. This allows the network to learn more complex structures of the images, starting with pixels and lines, and ending with facial features and objects [4]. By using these learned complex representations associated with a classification class, the model can predict new images by looking for these representations in the image. The network results in a probability based on how well these representations match the input image, and thus how likely the image is to belong to a specific class.

2.1.2 Generative adversarial networks

Generative adversarial networks, or GANs, were introduced in 2014 by Ian Goodfellow et al. during a period in machine learning history where most advances in deep learning happened in the field of classification rather than within generative models [11].

The network consists of two competing neural networks, a *discriminator* and a *generator*, see figure 2.2. The two networks are trained together, and competing with each other in a zero-sum game. In the original report, the discriminator and generator are compared to a team of counterfeiters and the police respectively. The counterfeiter (generator) produces a set of fake banknotes. These are then passed on to the police (discriminator) who has to determine if the banknotes are real or fake. In this scenario, instead of arresting them for the crime, the police tells the counterfeiter which notes they think are real and which they think are fake. Using this information, the counterfeiter improves the process and produces slightly more realistic banknotes, then repeatedly returns to the police for new feedback [11].

The **discriminator** is a classifier network with two inputs and one output. The first input is the external training data—depending on what the GAN is trying to generate, this could be anything from images, to sound clips, to text. The second input is generated data fed directly from the generator. Since the generator is trying to fool the discriminator, this data is formatted in the same way as the real input data. The output of the discriminator is the classification results. The activation function used in the discriminator is a sigmoid function, which means the network's output is the probability of how real (i.e. from the training data) the discriminator think the data is. These results are then used to calculate the error of the system, which in turn is then finally backpropagated through both the networks [9].

The **generator** is a generative network with two inputs and one output. Unlike the dis-

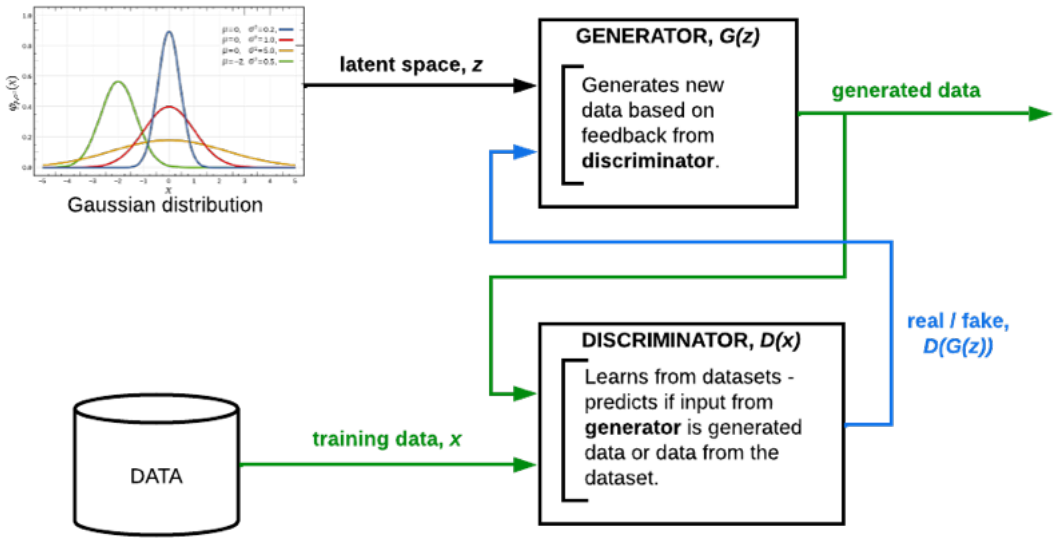


Figure 2.2: Diagram of a generative adversarial network.

criminator, the generator does not have access to the training data. The process of generation works a bit like the game of *hot or cold*, where the generator constantly has to ask the discriminator if the generated data is hot or cold (i.e. high or low probability of it being real). This feedback is one of the generator's inputs. The other input is a random noise sampled from the Gaussian distribution. This noise is often referred to as the latent space, and can be seen as injecting random creativity into the generation process. It is the processing of this latent space, and the adjustment of the internal weights and biases through training, that finally results in an output of generated data [13].

Mathematically, when training the discriminator, the goal is to maximise the function 2.3 [11]. The first part of this equation, $\log(D(x))$ where x is the training data, denotes the discriminator, D, correctly classifying x as training data. The second part, $\log(1 - D(G(z)))$ where $G(z)$ is generated data from the generator, G, is maximised when the generator is unable to fool the discriminator, and as a result $D(G(z)) \to 0$.

$$\max_D \log\left(D(x)\right) + \log\left(1 - D(G(z))\right) \tag{2.3}$$

In this zero-sum game, the discriminator doing well means the generator isn't, and vice versa. For that reason, training the generator means minimising the second part of function 2.3, $\log(1 - D(G(z)))$, the same function that the discriminator is trying to maximise [11]. However, the results of minimising this function are the same as instead maximising the inverted function 2.4.

$$\max_G \log(D(G(z))) \tag{2.4}$$

The loss function for binary cross entropy is defined in function 2.5. By setting the variable $y = 0$, the first half of the function is cancelled out, and what remains is the second half of the discriminator in function 2.3. In the same way, setting $y = 1$ gives us the the first half of the discriminator in function 2.3, as well as the generator in function 2.4 [13].

$$\ell(x, y) = L = \{l_1, ..., l_N\}^T, l_n = -\omega_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)] \tag{2.5}$$

By controlling the y-variable, the standard binary cross entropy function can be used as a loss function in the training of the full GAN.

2.1.3 ESRGAN

Enhanced Super-Resolution Generative Adversarial Networks, ESRGAN, is an improvement of the SRGAN, a GAN based image upscaling network [27, 18].

Both upscaling networks (SRGAN and ESRGAN) are designed to solve the task of SISR (Single-image super-resolution), a task in generating an image $\boldsymbol{I}^{SR}$, from a low-resolution image $\boldsymbol{I}^{LR}$, to closely resemble it's original high-resolution image $\boldsymbol{I}^{HR}$ [18]. The low-resolution image $\boldsymbol{I}^{LR}$ is a downscaled version of the $\boldsymbol{I}^{HR}$, that has also been blurred using a Gaussian filter. The generator of the SRGAN is designed to upscale the image $\boldsymbol{I}^{LR}$, without any knowledge of the image $\boldsymbol{I}^{HR}$, and fool the discriminator that the generated image $\boldsymbol{I}^{SR}$ is an image $\boldsymbol{I}^{HR}$.

Being a direct improvement of the SRGAN, the ESRGAN works fundamentally as described above. To improve the results, the architecture of the generator was adjusted. Batch Normalisation (BN), a popular method for normalising values before they are passed to the activation function [15], normalises the values to a zero mean, reducing the size of the scalars, and by doing so, simplifies the computations in the network. The method is used to reduce the training time and improve the stability during training [15]. However, though empirical testing by the authors, using BN in the ESRGAN network lead to unwanted artifacts and increased training time, and was thus removed. A Residual-in-Residual Dense Block (RDDB), a multi-level residual network with dense connections is also introduced by the authors and added to the network [27].

The discriminator was changed to determine the quality of the upscaled image $\boldsymbol{I}^{SR}$, and output a value of how closely it resembled the image $\boldsymbol{I}^{HR}$, rather than outputting a probability. Adjustments were also done in the calculation of the perceptual loss [27], which quantifies the similarities between two images.

2.1.4 Style transfer

Style transfer, or more correctly, neural style transfer (NST) was introduced in 2015 by Leon Gatsy et al. [4]. The algorithm takes a content image, and a style image, and produces a combined image using the two. Figure 2.3 shows an example of this.

The way style transfer work is by exploiting the way deep convolutional neural networks (ConvNet or CNN) function. As explained in section 2.1.1, a ConvNet starts by learning local information about an image in the early layers of the network, gradually the representation becomes more complex, and bigger puzzle pieces of the image are learned. In the later layers, the network learns the the full structure of the image, which also is referred to as global information. In the case of style transfer, this relates to the actual content of the image. By combining the trained weights of the later layers (i.e., the data making up the content of the image, representing it's lines, shapes and structure), with the trained weights of the early layers (i.e., the data making up the style, colours, and details), a network capable of combining this into a single image can be constructed [6].

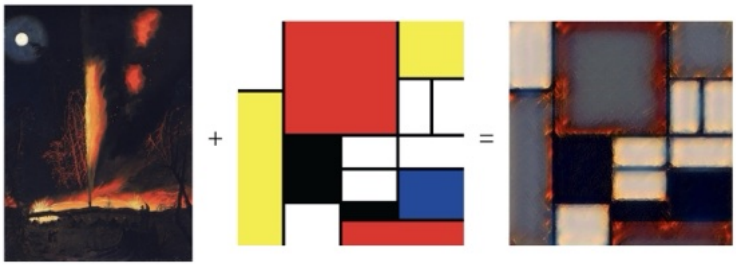


Figure 2.3: A style image (Burning Oil Well at Night, by James Hamilton) to the left, combined with the content image (by Piet Mondrian) in the middle, resulted in new image to the right.

2.2 Data

The data used to train the discriminator in the GAN all came from the dataset provided with the report *DELAUNAY: a dataset of abstract art for psychophysical and machine learning research* [8]. Primarily the art by *Piet Mondrian*, *Ellsworth Kelly*, *San Francis*, *Franz Kline*, *Richard Paul Lohse*, and *Robert Delunay* were used. The dataset is created to be a middle ground between real objects and artificial patterns, making it ideal for the task of creating abstract art.

Chapter 3

Discussion

The experimental process of making the prototype is described in section 3.1. Section 3.2 is an analysis of the finished prototype, and of what improvements could be made.

3.1 Experimental progression

The goals for this thesis has been of an experimental nature. It was only after the initial literature study that generative models, and GANs, were considered. Generative adversarial networks seemed promising as they could be trained once, then used repeatedly to generate unique images. However, these models are notoriously sensitive to badly tuned hyperparameters. Many hundred of hours were spent training and experimenting with models that then turned out to be completely unusable for generation.

3.1.1 Single model approach

The first approach was to directly from the GAN, later referred to as the base image generator, generate the final artwork. Upscaling the model to generate higher resolution images proved to be very difficult. The training time per batch of images increased exponentially with every incremental size step. Already at 256x256 pixels, the system could no longer finish an epoch of training as it ran out of RAM. As an experiment, a powerful cloud computer with 256 GB of RAM was rented to run the same code. Even this computer could not train the 512x512 model before running out of memory. The average training times mentioned are presented in table 3.1.

Size (px)	Time (laptop)	Time (cloud)
32x32	10s	1s
64x64	1m 30s	15s
128x128	14m	2m
256x256	OVERFLOW	16m
512x512	OVERFLOW	OVERFLOW

Table 3.1: Training time for early GAN model. *(average processing time per image batch.)*

3.1.2 Style transfer upscale approach

Inspired by projects described in chapter 1.6—instead of directly generating high resolution images from the generator, a smaller image was created to then instead be upscaled to a high resolution.

Downsizing an image is a much easier task than upscaling. The higher resolution image contains more data than it's lower resolution counterpart, as more pixel values are saved to represent the image. By shrinking an image, data is removed from the original image as the number of pixels decrease.

Upscaling an image works like shrinking and image, but in reverse. As the number of pixels increase, more data has to be added to create the new representation of the image.

To inject this missing data, *style transfer* [6] was used to upscale the base images. Figure 3.1 shows an early 128x128 pixel base image upscaled to a 256x256 pixel image using style transfer. This method was successful for the first couple of iterations but unable to process any larger images.

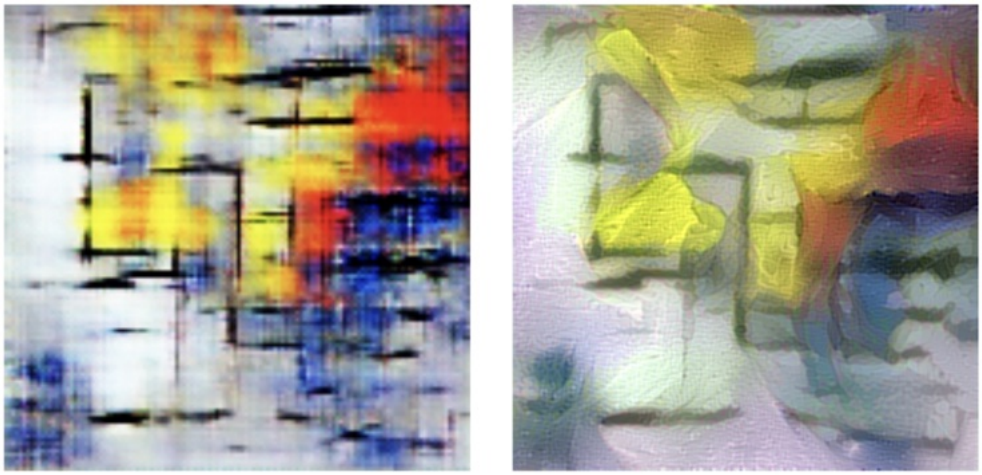


Figure 3.1: Style transfer used to upscale a base image. The left image is a 128x128 base image, and the right image is the upscaled 256x256 image.

To still utilise the successful 128 to 256 upscaling method, a new upscaling method was tested. When an image was upscaled to 256x256 pixels, it was then split up into four 128x128 sub-images and upscaled again. This was repeated until it a desirable size was reached. The

sub-images were then put back together to create a big upscaled image. This method is presented in figure 3.2. A grid of where the images were put back together were visible in all the upscaled images. Different methods to mitigate this were all unsuccessful, such as generating overlapping borders that are then blended together for a smoother transition.

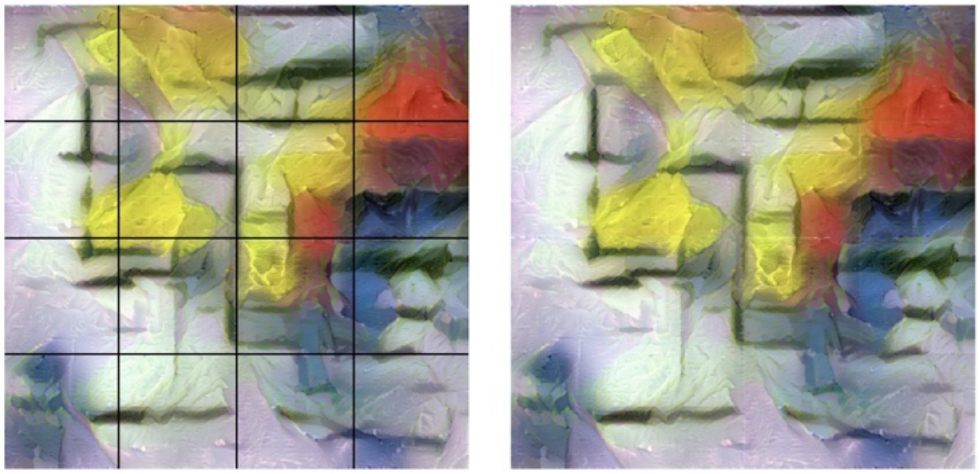


Figure 3.2: A grid of style transfer used to upscale the same image as in figure 3.1. The left image shows the grid of upscaled sub-images, and the right image shows a finished 1028x1028 image.

3.1.3 ESRGAN upscale approach

As upscaling proved to be a highly challenging task, with multiple reports focusing on this task alone [28], ESRGAN was built into the pipeline. To still take advantage of the upscaling process discussed in section 3.1.2, it was added as an intermediate step between the base image generation and the ESRGAN upscaling. Figure 3.3 shows a base image generated by the Mondrian model to the left, three style images in the middle, and the style transferred and upscaled finished 2048x2048 images to the right. The style images are by, from top to bottom, *Georges Valmier*, *Piet Mondrian*, and *James Hamilton*, all of whose artwork is in the public domain.

The style transfer step made the variation suffer, as every finished generated image resembled the style image too much. Because of this, and to keep the focus on the GAN, and the base image generation, it was removed from the final prototype.

3.2 Analysis of prototype improvements

Since its release in 2014, the original GAN architecture has inspired many researches to attempt improving the stability of the network and the generated results. The two most prominent improvements are the StyleGAN by Nvidia [16], and the VQGAN [5].

StyleGAN, introduced in 2018, replaces the regular GAN generator's latent space input z with a learnable constant, and removes the input layer all together. Inspired by style transfer, it then samples the "style" from a latent vector (rather than an image) an injects it into every layer of the discriminator and generator [16].

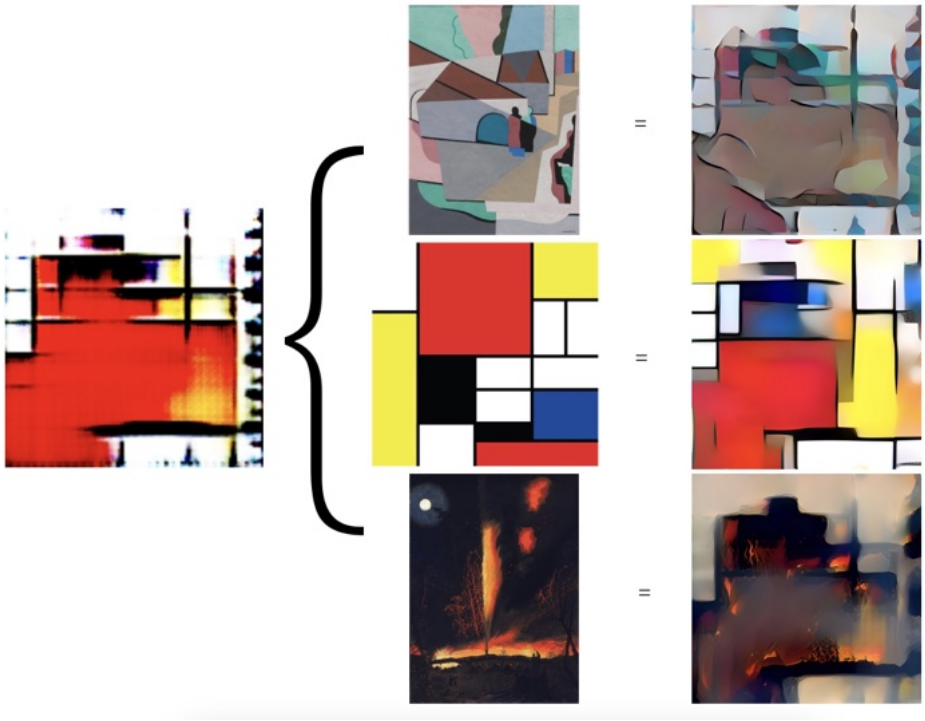


Figure 3.3: An example of a base image with style transfer and upscaling by ESRGAN applied.

The VQGAN, introduced in 2020, learns what the authors refer to as a **codebook** of image features. These features are the details of the image, together representing the full images. By having the discriminator predict these blocks of the generated image rather than the entire image, enables a less rigid design with realistic images built from realistic image details [5].

Using either one of these architectures over the default GAN would have reduced the heavy dependency on a separate upscaling model.

The datasets used to train the models were also far too small. Most of the training and fine-tuning of the hyperparameters were done on the Piet Mondrian dataset. This dataset only contained 176 images, which in the field of machine learning is very few.

Some attempts on training a model on the full DELAUNAY dataset were made. The large variety in the images combined with the small set of samples led to poor results, an unexplained *waffle* shape appeared in most of the base images, see figure 3.4. Using the VQGAN architecture would likely instead have benefited from this as many details, such as similar brush strokes, would have been learned as a codebook encoding.

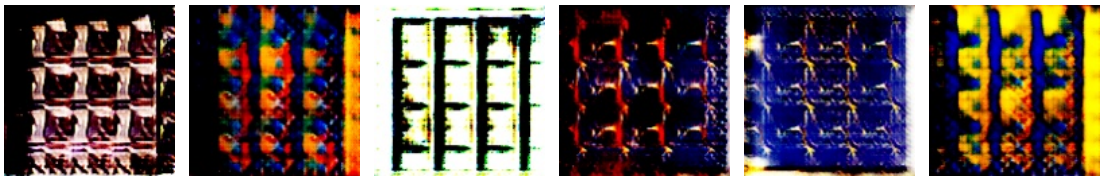


Figure 3.4: The recurring waffle shape in generated DELAUNAY dataset images.

3.3 Ethical and legal aspects

Many state-of-the-art image generators have recently fallen under scrutiny for using copyrighted material in their training data without consent from the original artist. In January 2023, the first lawsuit against AI generated art was filed [3]. This lawsuit mainly focuses on *Stable Diffusion* by Stable AI, *DreamUp* by DeviantArt, and *Midjourney* by Midjourney inc, but the purpose of it is rather to define the rules regarding the legality of computer generated art. This may in turn require a clear definition of what constitutes as *'inspiration'* for a computer system. As a result of the legal implications of building an art generator is actively discussed, the prototype of this thesis was trained using copyright-free material.

Copyrighted material is not the only ethical issue to be considered. With the general rule-of-thumb in machine learning that more data is better, the perpetual hunt for more data has led to datasets being created by simply 'scraping the internet of everything available' [19]. These publicly available datasets do not only contain copyrighted and personal material, but also discriminatory and stereotypical material with a strong negative racial and gender bias [2]. In art generation, this may seem like an insignificant problem, barely worthy of any attention, but as generative systems continuously improve, so does their ability to visualise the bias on which they're trained, and in turn, further propagate this unwanted inequity.

The topic of ethics related to machine learning is a big and important one. As Birhane and Cummins [2] state in their report, when data is used that contains social and historical stereotypes without a deep casual understanding, machine learning as a tool cannot be seen as factual and void from any moral responsibility.

Chapter 4

Design

The architecture presented in this chapter is the result of the process further discussed in chapter 3.1. This chapter focuses on describing and motivating the details of the final art generator prototype. As the GAN is the core of the prototype, and thesis, it'll be described in the most detail.

4.1 Overall architecture

The generative adversarial network (GAN) is the foundation of the system. After being trained, the generator of the GAN generates a 128x128 pixel **base image** which is an image derived from the training data fed to the GAN. The image is represented as a (3, 128, 128) tensor, with every pixel being stored in three colour channels as a continuous variable in the range [-1, 1]. This image is then upscaled using a pretrained ESRGAN model. The described architecture is visualised in figure 4.1.

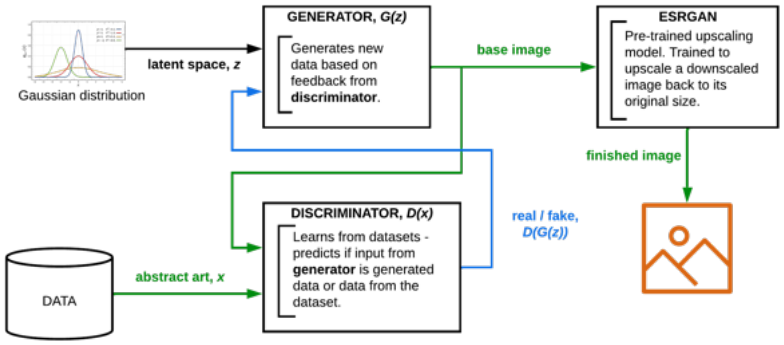


Figure 4.1: Architectural diagram of the prototype.

4.2 Architecture design process

To tune the parameters of the base image generator, an experimental setup was built. The goal was to find a numerical value representing the quality of a generated image. This value could then be used to automatically optimise the hyper-parameters by iteratively adjusting and comparing them to the quality score.

The generated base images were saved during training, along with their respective calculated loss from both the generator and the discriminator. However, neither of the loss functions could be used for the purpose of evaluation. As an example to prove this, two models were trained on the same Mondrian dataset (sample images are displayed in figure 4.2), but using different hyper-parameters and architectures.

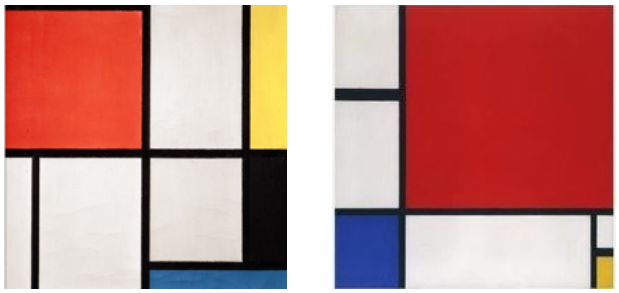


Figure 4.2: A small sample of Mondrian training data.

Both networks were trained for 400 epochs, and generated similar results from the loss functions. The discriminator loss for both the generated data and the training data oscillated around 0.5, and the generator loss in the range 1 and 2. Despite the numerical similarities in loss values, the first model generated the base images displayed in figure 4.3, and the improved model generated the images in figure 4.4.

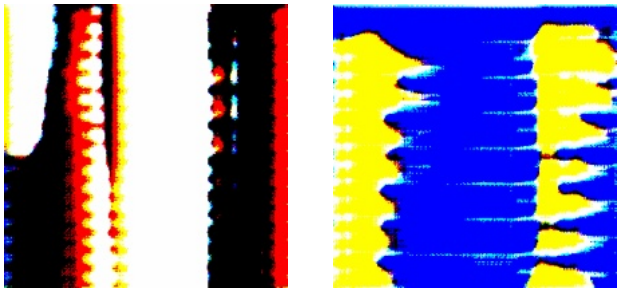


Figure 4.3: A small sample of generated base images produced by badly tuned model.

The colours and grid structure of the generated results displayed in figure 4.4 more closely resembles the Mondrian training data than the results in figure 4.3. Yet, based on the numerical analysis of the loss functions, they're equally as good.

Even within the same model, the loss functions proved to be insufficient in the valuation of the results. During training, the model that generated the good base images displayed in figure 4.4, resulted in the loss function results visualised in diagram 4.5. As shown in the

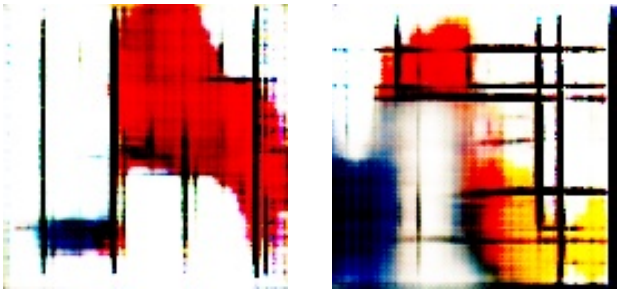


Figure 4.4: A small sample of good generated base images produced by well tuned model.

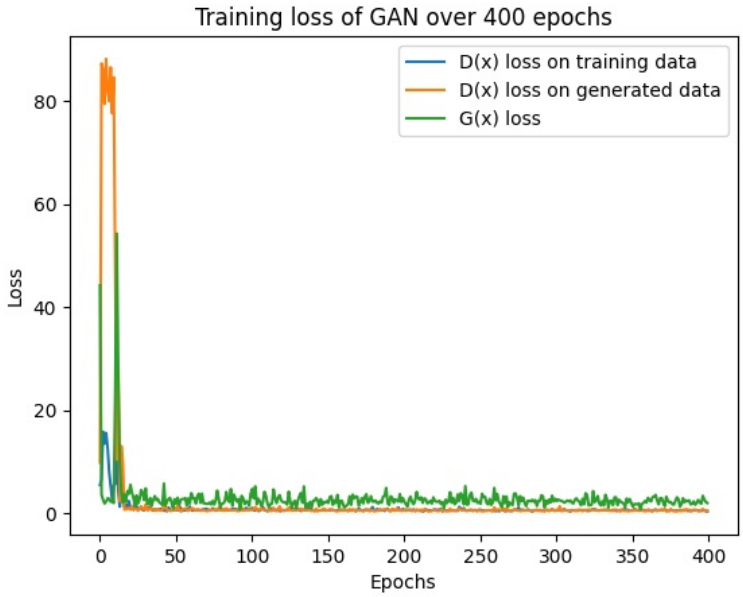


Figure 4.5: A line diagram of the calculated losses during training.

diagram, after only 25 epochs the loss functions stabilises and produces similar values for the rest of the training.

Sampled at a later epoch, as well as producing slightly lower loss values for both the generator and discriminator, the images displayed in figure 4.6 are subjectively worse representations of the Mondrian dataset than those displayed in figure 4.4, but numerically better.

As a result of this observation, and due to no better metric being found, the tuning of the hyper-parameters and design of the architecture had to be done though manual evaluation of the results. Different models using the same training data, but with different hyper-parameters were trained sequentially. To notice the results of the training, at least 400 epochs had to be trained. Using the available hardware, this process took 6-8 hours per model, making the feedback loops very slow.

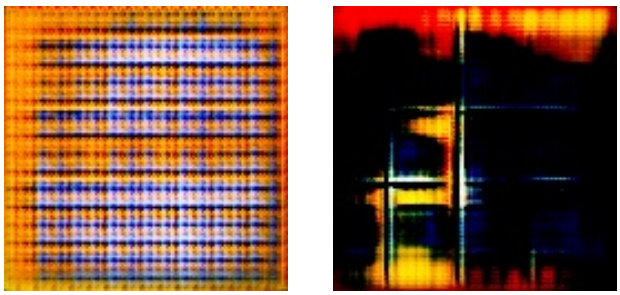


Figure 4.6: A small sample of bad generated base images produced by a well tuned model.

4.3 Base image generator

The generation of the base image is the most important step of the generative process. It's at this step the artwork is created. The subsequent steps are used to enhance the base image.

As described in chapter 2.1.2. The binary cross-entropy is used as a loss function to train both internal networks. This function compares the true or false labels of the input images with the predicted results, and returns a scalar corresponding to the accuracy of the predictions. As a higher value indicates more uncertainty, the training of both networks aims to minimise this loss function.

A single training step, which occurs once for every image batch in every epoch, consists of three sub-steps; training of the discriminator on real images, $D(x) \rightarrow 1$, training of the discriminator on generated images, $D(G(z)) \rightarrow 0$, and training of the generator $D(G(x)) \rightarrow 1$. The sub-steps all have the same structure:

1. **Load or generate a batch of images.** Images are loaded from the training data when the discriminator is trained on real images, otherwise a batch of images are generated from the generator network, $G(z)$.

2. **Create a vector of image labels.** For the networks to successfully differentiate between the generated and training images, labels are required. As suggested by Salimans et al.[25], rather than using hard labels (i.e. 0 and 1), smooth labels (i.e. 0.1 and 0.9) should be used to stabilise the system by injecting some uncertainty. For that reason, the false labels are randomised in the range of [0.0, 0.3] and true labels are of the range [0.7, 1.2]. As described in chapter 2.1.2, rather than trying to maximise the loss function when training the generator, the labels can instead be flipped for this step to enable the usage of the default minimising features.

3. **Classify the image batch using the discriminator.** The batch of images are passed through the discriminator to predict their class (i.e. generated or from the training data). The discriminator returns the probability that the classified image is real, where 1.0 means that the discriminator is 100% sure that the image comes from the training data, and 0.0 means it's 100% sure it's a generated image.

4. **Calculate the loss using binary cross entropy.** The generated labels, together with their respective predictions are then used in the calculation of the binary cross entropy to quantify how well the networks are doing.

5. **Calculate the gradient and update the weights.** The gradients are calculated using the **Adam** [17] optimiser, a slightly adjusted version of the gradient descent described in chapter 2.1. The weights are then updated throughout the network.

A main disadvantage of training generative adversarial networks is it's lack of an objective functions able to track it's progress [25, 4]. As the two inner networks are competing with each other, and the progress of one benefits both networks, there is no numerical threshold indicating that the system has finished training. Thus, training a GAN is mainly about building a stable network that doesn't crash.

A common error when generating images using GANs are *checkerboard artifacts*, these occur when some pixels are processed more than others from overlapping kernels, and as a result a noticeable grid is visible in the image. To avoid this, the kernel size has to be divisible by the stride for both the generator and discriminator [4].

The input to the generator network is a tensor of shape (100, 1, 1) sampled from a Gaussian distribution. This input data is referred to as latent space, z. The dimension of 100 hold no mathematical importance, but is by many considered the default value [20]. Since the generator should generate an image of size 128x128, the output layer needs to output a tensor of shape (3, 128, 128).

The discriminator takes an image as input. A (3, 128, 128) tensor represents the input image to be classified. As suggested by Radford et al. [22], in every layer of the discriminator, batchnorm should be used to stabilise the model, and LeakyReLU as the activation function. The structure needs to end with a sigmoid function. This allows for the model to transfer the result to a probability in the range [0.0, 1.0].

4.4 Single-image super-resolution

Due to the high demands on hardware when training an upscaling model, a pretrained ESRGAN model was imported. According to the ESRGAN report, the pretrained model was trained using four ML specialised GPUs, NVIDIA Tesla V100. The model was also trained for almost 1.5 million iterations [26].

Copyright-free images from datasets like DIV2K [1] were used to synthesise the low-resolution images. The network was then trained to learn how to transfer the downscaled details back to it's upscaled version.

By feeding the model with the generated low resolution base images, it interpreted the details of the generated image as something previously seen. With the assumption that the base image is to be upscaled back to it's original size, a new high resolution version of the base image is generated.

4.5 Training data

In order to more easily compare the results, to improve the hyperparameters and the architecture, the *Piet Mondrian* subset of the DELAUNAY was mainly used during training. A sample from this Mondrian subset is displayed in figure 4.7. When using regular supervised learning algorithms, the dataset is split into subsets for training, testing, and validation. Despite

internally using a classification algorithm, the GAN does not benefit form having a separate testing or validation set. For this reason, the entire dataset can be used to train the model.

Figure 4.7: Samples from the Piet Mondrian training data.

Chapter 5

Results

In this chapter, the finished architecture of the prototype will be presented, as well as generated images sampled from each step along the prototype pipeline.

5.1 Overall architecture

The final overall architecture of the prototype resulted in a two-step process. First a base image is generated using the base image generator trained on a single artist (for this thesis, all examples are from the Piet Mondrian model). The output of the trained generator model is then passed to the pretrained upscaling model to produce the final images. A figure of the pipeline is displayed in figure 5.1.

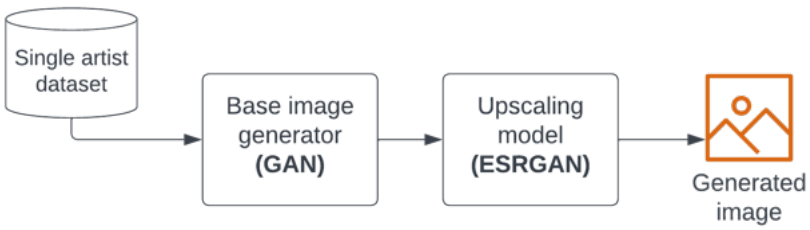


Figure 5.1: The two-step generation pipeline, from dataset to generated image.

5.2 Base image generator

The final architecture of the generator is displayed in table 5.1, and the discriminator in table 5.2. Using these models, and the dataset sampled in figure 4.7, a GAN was trained. A small sample of generated 128x128 pixel base images are presented in image 5.2.

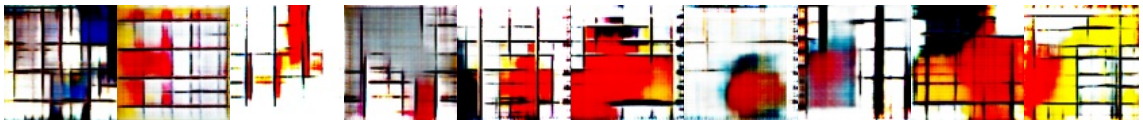


Figure 5.2: A sample of base images generated by the Mondrian model.

As shown in table 5.1 and 5.2, many of the final hyper-parameters, such as kernel size and stride produced the best results following the theoretical recommendations [22, 4].

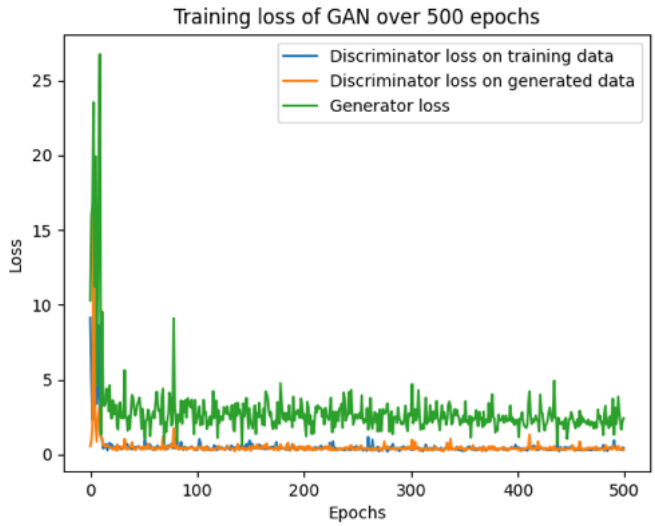


Figure 5.3: A line diagram of the calculated losses during training.

The results of the loss function during training of the Mondrian model are presented in the line diagram 5.3. The diagram shows that the discriminator loss for both the training and generated data is consistently lower than the generator loss. To compensate for the fact that the discriminator is training faster than the generator, in every GAN training step, the generator is trained twice for every one discriminator step. If the network is trained unevenly, the generator gets less feedback due to not being able to fool the discriminator, and this in turn leads to a growing gap. The diagram shows that the loss values stay consistent, and both networks continue training.

Table 5.1: The structure of the generator, G(z).

Model Structure		Input	Kernel		Output	Parameters
		Dimension (C, H, W)	Stride/Padding	Dimension (H, W)	Dimension (C, H, W)	
Input layer	ConvTranspose2d	100, 1, 1	1 / 0	4, 4	2048, 4, 4	3,276,800
	BatchNorm2d	2048, 4, 4			2048, 4, 4	4096
	LeakyReLU	2048, 4, 4			2048, 4, 4	
Hidden layer 1	ConvTranspose2d	2048, 4, 4	2 / 1	4, 4	1024, 8, 8	33,554,432
	BatchNorm2d	1024, 8, 8			1024, 8, 8	2048
	Dropout2d	1024, 8, 8			1024, 8, 8	
	LeakyReLU	1024, 8, 8			1024, 8, 8	
Hidden layer 2	ConvTranspose2d	1024, 8, 8	2 / 1	4, 4	512, 16, 16	8,388,608
	BatchNorm2d	512, 16, 16			512, 16, 16	1024
	Dropout2d	512, 16, 16			512, 16, 16	
	LeakyReLU	512, 16, 16			512, 16, 16	
Hidden layer 3	ConvTranspose2d	512, 16, 16	2 / 1	4, 4	256, 32, 32	2,097,152
	BatchNorm2d	256, 32, 32			256, 32, 32	512
	Dropout2d	256, 32, 32			256, 32, 32	
	LeakyReLU	256, 32, 32			256, 32, 32	
Hidden layer 4	ConvTranspose2d	256, 32, 32	2 / 1	4, 4	128, 64, 64	524,288
	BatchNorm2d	128, 64, 64			128, 64, 64	256
	Dropout2d	128, 64, 64			128, 64, 64	
	LeakyReLU	128, 64, 64			128, 64, 64	
Output layer	ConvTranspose2d	128, 64, 64	2 / 1	4, 4	3, 128, 128	6147
	Tanh	3, 128, 128			3, 128, 128	

5.3 Single-image super-resolution

To reach the target size, the generated base images passed through the ESRGAN model twice—once to upscale the base image from 128x128 to 512x512, then again to upscale it from 512x512 to 2048x2048. Some samples of finished 2048x2048 images, upscaled using the pretrained ESRGAN model, are presented in figure 5.4.

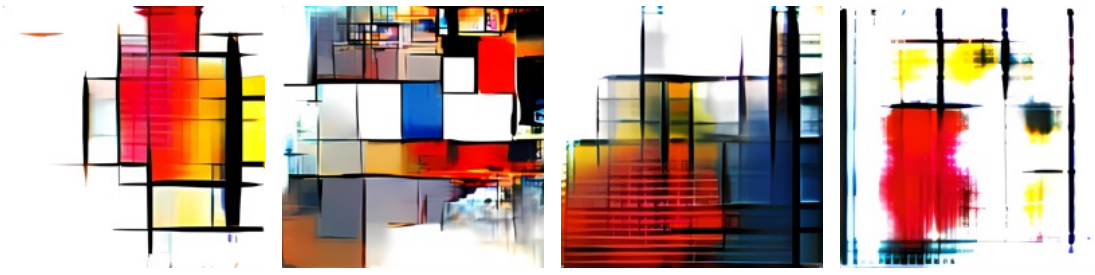


Figure 5.4: Finished images inspired by Piet Mondrian.

Table 5.2: The structure of the discriminator, D(x).

Model Structure		Input	Kernel		Output	Parameters
		Dimension (C, H, W)	Stride/Padding	Dimension (H, W)	Dimension (C, H, W)	
Input layer	Conv2d	3, 128, 128	2 / 1	4, 4	128, 64, 64	6144
	LeakyReLU	128, 64, 64			128, 64, 64	
Hidden layer 1	Conv2d	128, 64, 64	2 / 1	4, 4	256, 32, 32	524,288
	BatchNorm2d	256, 32, 32			256, 32, 32	512
	LeakyReLU	256, 32, 32			256, 32, 32	
Hidden layer 2	Conv2d	256, 32, 32	2 / 1	4, 4	512, 16, 16	2,097,152
	BatchNorm2d	512, 16, 16			512, 16, 16	1024
	LeakyReLU	512, 16, 16			512, 16, 16	
Hidden layer 3	Conv2d	512, 16, 16	2 / 1	4, 4	1024, 8, 8	8,388,608
	BatchNorm2d	1024, 8, 8			1024, 8, 8	2048
	LeakyReLU	1024, 8, 8			1024, 8, 8	
Hidden layer 4	Conv2d	1024, 8, 8	2 / 1	4, 4	2048, 4, 4	33,554,432
	BatchNorm2d	2048, 4, 4			2048, 4, 4	4096
	LeakyReLU	2048, 4, 4			2048, 4, 4	
Output layer	Conv2d	2048, 4, 4	1 / 0	4, 4	1, 1, 1	32,768
	Sigmoid	1, 1, 1			1, 1, 1	

Chapter 6
Conclusion

In this chapter, the research questions stated in chapter 1 are concisely answered. Some ideas of how to further improve the prototype are presented in the second sub-chapter.

6.1 Research questions answered

The following questions were presented in the first chapter as a foundation of the thesis, with the purpose of answering them thought the making of the generator prototype.

RQ1 What steps and technology are required to develop an abstract art generator?

In this thesis, an approach of how to build and train an art generator prototype is presented. Due to the difficulties of directly generating large images from the generative model due to the huge working storage needed (more than 256 GB RAM), separating the task of generation and upscaling proved to be a better solution. This resulted in a two-step process. The first step being a generator designed to generate base images, small images capturing the theme of an artist's work. To finalise the generation, the base image is then passed to an ESRGAN model to be upscaled.

RQ2 How can generative adversarial networks (GAN) be used to generate high definition artwork?

Both steps of the presented two-step process, generation model and upscaling model, are generative adversarial networks. The first GAN is trained to generate a new image from the latent space, based on external training data fed to the internal discriminator. The second GAN is trained to transfer a downscaled image back to it's original size. These two GANs working together are then used to generate the high definition artwork. Other methods, such as using VQGAN, or StyleGAN can also be used for the task.

RQ3 How long does it take to generate a single image; is generating on-demand art feasible?

Once trained, both the generation network and upscaling network require very little time to produce a new image, thus making on-demand generation easily achievable. The time-consuming training of the models only has to be done once for every dataset. The trained models are then saved and reusable for art generation.

6.2 Future work

Possible improvements to the prototype are presented below:

- Inspired by the style transfer upscaling method presented in chapter 3.1, training art style specific ESRGAN models could greatly improve the upscaling to look more realistic, as well as to avoid the usage of the controversial massive datasets.
- Rebuilding the GAN powered base image generator to instead use the VQGAN architecture could remove the constraint of only being able to generate abstract art, due to limitations in the generator.
- Proposed early as a goal for the thesis—a filtering network could be trained based on feedback from the end-users. This network could then automatically filter the subjectively good results from the bad ones, as well as feeding the generator with subjective feedback. This would however require a lot of data.

References

[1] Eirikur Agustsson and Radu Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[2] Abeba Birhane and Fred Cummins. Algorithmic injustices: Towards a relational ethics, 2019. https://arxiv.org/abs/1912.07376.

[3] Matthew Butterick. Stable diffusion litigation, January 2023. https://stablediffusionlitigation.com.

[4] François Chollet. *Deep Learning with Python*. Manning, 2021.

[5] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis, 2020. https://arxiv.org/abs/2012.09841.

[6] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015. https://arxiv.org/pdf/1508.06576.pdf.

[7] Robert A. Gonsalves. GANshare: Creating and curating art with AI for fun and profit, October 2021. https://towardsdatascience.com/ganshare-creating-and-curating-art-with-ai-for-fun-and-profit-1b3b4dcd7376.

[8] Camille Gontier, Jakob Jordan, and Mihai A. Petrovici. DELAUNAY: a dataset of abstract art for psychophysical and machine learning research, 2022. https://arxiv.org/abs/2201.12123.

[9] Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks, 2017. https://arxiv.org/pdf/1701.00160.pdf.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[12] James Hendler. Avoiding another ai winter. *IEEE Intelligent Systems, Intelligent Systems, IEEE, IEEE Intell. Syst*, 23(2):2 – 4, 2008.

[13] Yongjun Hong, Uiwon Hwang, Jaeyoon Yoo, and Sungroh Yoon. How generative adversarial networks and their variants work. *ACM Computing Surveys*, 52(1):1–43, feb 2019.

[14] Fortune Business Insghts. Wall art market size, share & covid-19 impact analysis, April 2022. https://www.fortunebusinessinsights.com/wall-art-market-105009.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. https://arxiv.org/pdf/1502.03167.pdf.

[16] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2018. https://arxiv.org/abs/1812.04948.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. https://arxiv.org/pdf/1412.6980.pdf.

[18] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2016. https://arxiv.org/abs/1609.04802.

[19] Alexandra Sasha Luccioni and Joseph D. Viviano. What's in the box? a preliminary analysis of undesirable content in the common crawl corpus, 2021. https://arxiv.org/abs/2105.02732.

[20] Ivana Marin, Sven Gotovac, Mladen Russo, and Dunja Božić-Štulić. The effect of latent space dimension on the quality of synthesized human face images. *Journal of Communications Software and Systems*, 17:124–133, 06 2021.

[21] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[22] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015. https://arxiv.org/abs/1511.06434.

[23] Alec Radford, Ilya Sutskever, Jong Wook Kim, Gretchen Krueger, and Sandhini Agarwal. Clip: Connecting text and images, January 2021. https://openai.com/blog/clip/.

[24] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[25] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs, 2016. https://arxiv.org/abs/1606.03498.

[26] Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. Real-ESRGAN: Training real-world blind super-resolution with pure synthetic data, 2021. https://arxiv.org/abs/2107.10833.

[27] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. ESRGAN: Enhanced super-resolution generative adversarial networks, 2018. https://arxiv.org/abs/1809.00219.

[28] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing-Hao Xue, and Qingmin Liao. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia*, 21(12):3106–3121, 2019.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-03-16

EXAMENSARBETE Generating abstract art using artificial neural networks
STUDENT Henrik Norrman
HANDLEDARE Daniel Byström (Posterton), Christin Lindholm (LTH)
EXAMINATOR Jacek Malec (LTH)

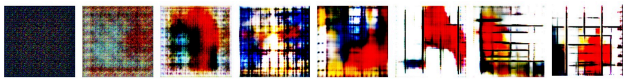
Generering av abstrakt konst med hjälp av artificiell intelligens (AI)

POPULÄRVETENSKAPLIG SAMMANFATTNING **Henrik Norrman**

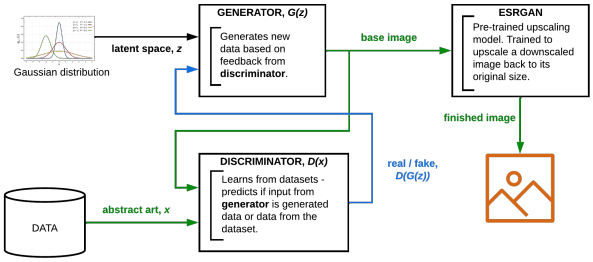
Under de senaste åren har vi sett en explosion av AI-applikationer designade för kreativa uppgifter som manusskrivande, musikskapande och konstgenerering. I detta examensarbete presenteras en tvåstegsprocess för generering av abstrakt konst.

I takt med att hård- och mjukvara fortsätter utvecklas tar AI också en allt större roll i våra liv. Likt hur vi använder internet idag, tror många forskare att vi i framtiden kommer använda AI-verktyg i nästan varje del av vår vardag, både för praktisk och kreativ hjälp. I detta examensarbete utforskas hur en konstgenerator för abstrakt konst kan byggas med hjälp av artificiella neuronnät.

För att förstå hur genereringen fungerar behöver man förstå hur ett GAN (generative adversarial network) fungerar. GAN är en maskininlärningsteknik från 2014 där två artificiella neuronnät tävlar mot varandra (ett genereringsnätverk och ett klassificeringsnätverk). Klassificeringsnätverket tränas som ett neuronnät för bildigenkänning på ett dataset med abstrakt konst. Detta nätverk har som uppgift att svara på frågan: "Vad är sannolikheten att denna bild är från mitt dataset (och inte genererad)?". Genereringsnätverket tränas för att försöka lura klassificeringsnätverket med bilderna den genererar. Utan att se bilderna i träningssetet försöker genereringsnätverket återskapa dessa med hjälp av återkopplingen från klassificeringsnätverket. Genererade bilder som lurar klassificeringsnätverket byggs vidare på och förbättras. De åtta bilderna visar hur en modell tränad på konst från Piet Mondrian iterativ förbättras med träning.



Vid generering av större bilder blev arbetsbördan för denna GAN-modell helt för stor. Både träningstiden och arbetsminnets storlek eskalerade till en ohanterbar storlek. Som lösning presenteras slutligen en tvåstegsmodell: först genereras en mindre basbild som tränas på att fånga stilen av konstnärens konstverk, sedan skalas basbilden upp till önskad storlek av en separat uppskalningsmodell. Ett diagram över konstgeneratorn visas nedan.



Konstgeneratorn skapades som en grundläggande bas för vidareutveckling till en mer komplett generator, men också som en bas för vidare utforskning av generativ AI. Tvåstegsprocessens design ger möjlighet att implementera mellansteg för förfining av den genererade basbilden innan den slutförs av uppskalningen.