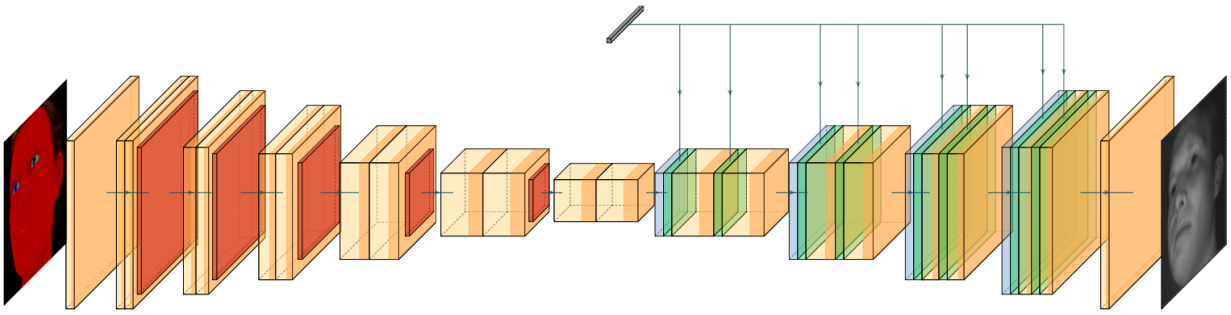
CHALMERS
UNIVERSITY OF TECHNOLOGY



Multimodal Image-to-Image Translation for Driver Monitoring System Development

Synthesizing diverse human faces in the near-infrared domain using Conditional Generative Adversarial Networks

Master's thesis in Complex Adaptive Systems

Jonathan Bergqvist

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Multimodal Image-to-Image Translation for Driver Monitoring System Development

Jonathan Bergqvist

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

Multimodal Image-to-Image Translation for Driver Monitoring System Development
Synthesizing diverse human faces in the near-infrared domain using Conditional Generative Adversarial Networks
Jonathan Bergqvist

Supervisors:
Patrik Larsson, Smart Eye AB
Huu Le, Department of Electrical Engineering

Examiner:
Christopher Zach, Department of Electrical Engineering

Master's Thesis 2020
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Encoder-decoder generator architecture

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Multimodal Image-to-Image Translation for Driver Monitoring System Development
Jonathan Bergqvist
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Driver Monitoring Systems (DMS) are vehicle safety systems that improve road safety by tracking driver behaviour and issuing warnings when signs of drowsiness or distraction are detected. Today, many DMS use machine learning to track the eye, face and head movements of the driver and depend on large amounts of data to achieve sufficient generalization and performance. However, collecting, cleaning and labeling large amounts of data can often be time-consuming and expensive. As an alternative, this thesis investigates whether Conditional Generative Adversarial Networks (CGANs) can be used to create synthetic training data by deriving a mapping between paired images of a computer-generated 3D model and real human faces in the near-infrared domain. The thesis proposes a framework for multimodal image-to-image translation capable of generating diverse and realistic images of human faces while preserving domain-invariant attributes, e.g. head pose and gaze direction. Additionally, the framework is able to extract and approximate the appearance of previously unseen subjects when generating images. The thesis concludes that image synthesis using CGANs can be a viable method for obtaining customizable training data for DMS development.

Keywords: Driver Monitoring Systems, Machine Learning, Deep Learning, Generative Adversarial Networks, Image-to-Image Translation, Synthetic-to-Real

Acknowledgements

First and foremost, I would like to thank my industrial supervisor at Smart Eye, Patrik Larsson, for his dedicated supervision over the course of working on and writing this thesis. His guidance and advice has been invaluable, and without his support, the completion of this thesis would not have been possible. Additionally, I would like to thank everyone else at Smart Eye who I have been in contact with for helping me succeed and making me feel welcome from day one.

I would also like to thank my academic supervisor, Huu Le, who has also supported me throughout the thesis. His knowledge and advice has been incredibly helpful in choosing which paths to pursue, and our discussions led to many insights and new ideas. Furthermore, I want to express my gratitude to my examiner, Christopher Zach, who also provided me with valuable advice and guidance.

Lastly, I want to thank my incredible family and friends for encouraging me to pursue my interests and work hard. I am so grateful to have you all in my life.

Jonathan Bergqvist, Gothenburg, December 2020

Contents

1

Introduction

This section is intended to serve as an introduction to the topic of the thesis and the problem it aims to solve. It begins by providing the reader with the necessary context and background to the problem statement as well as the chosen approach for solving it in Chapter 1.1. This is followed by a review of related work in Chapter 1.2. Next, Chapter 1.3 provides the main problem statement together with the research questions that are investigated. Lastly, Chapter 1.4 specifies the scope of the thesis and Chapter 1.5 presents an outline of the remaining sections.

1.1 Background

Driver monitoring systems (DMS) are vehicle safety systems that monitor the state of the driver using cameras or other sensors. Their primary purpose is to prevent accidents by issuing warnings when potentially dangerous driver behaviour is detected, such as distraction or drowsiness. DMS have become an important component technology in the ongoing effort to prevent traffic accidents and are included in a growing number of modern vehicles. In the future, they are also expected to play an important role in partially automated vehicles, where a DMS can for instance be used to confirm that the driver is attentive to the road before a vehicle-initiated handover is performed.

Smart Eye AB (from now on referred to as Smart Eye) is a Swedish software company that provides DMS as a part of its offering to the automotive market segment. Their system builds an understanding of the state of the driver by tracking attributes such as gaze direction, facial expression and head position from image data provided by near-infrared cameras. These features are inferred using machine learning models, which require large amounts of real-world data for training in order to achieve sufficient generalization and prediction accuracy. In the case of a DMS, this amounts to near-infrared image data of faces from a diverse set of people and environments. However, collecting, cleaning and labeling large amounts of high quality data remains a challenge for Smart Eye and makes up a big part of the development process.

Because of this, Smart Eye is investigating whether it is feasible to instead use computer generated 3D models as training data for their DMS, as this would allow

the company to create customizable training data to fit their needs. However, a problem with this approach is that the 3D models lack the realistic attributes needed for the systems to generalize to real data.

To solve this problem, this thesis investigates if generative models can be used to increase the realism of the 3D models by formulating it as an *image-to-image translation* problem. This refers to the problem of learning a mapping between two image domains, where each domain contains images that can be grouped as a visually distinctive category [1, 2]. In particular, the thesis focuses on using deep generative models called Generative Adversarial Networks (GANs) to derive a mapping between RGB images of a 3D model and near-infrared images of real human faces.

1.2 Related work

Generative Adversarial Networks, or GANs for short, were originally proposed by Ian Goodfellow et al. [3] and are deep generative models consisting of two neural networks; a *generator* that produces images and a *discriminator* that tries to tell generated images apart from real ones. The two networks are trained as adversaries in an unsupervised manner, competing against each other in a type of game that in theory ultimately results in the generator learning to synthesize images that are indistinguishable from real ones. GANs are currently among the state-of-the-art in generative modeling and have been shown to be capable of producing photo-realistic images of great diversity [4, 5, 6]. Furthermore, a variation called conditional GANs (CGANs) has been shown to lend itself well to image-to-image translation tasks [2, 7, 8] and to be capable of learning a variety of mappings, e.g. synthesizing photos from label maps, reconstructing objects from edge maps and colorizing images [2].

The idea of using synthetic data to train machine learning models has previously been studied for a variety of tasks. The simplest approach is arguably to directly train models on raw synthetic data, e.g. 3D models created by a game engine or CAD software, and has previously been explored for a variety of tasks, such as object detection [9, 10], gaze estimation [11], human pose estimation [12] and hand pose estimation [13, 14] to name a few. However, in most applications, the synthetic data is not realistic enough, leading to the models learning details only present in the synthetic data and failing to generalize properly to real data [15].

To address this issue, multiple studies have proposed methods for obtaining more realistic synthetic data. One approach for doing this is to use generative models to increase the realism of the synthetic data, so called *synthetic-to-real refinement* [16]. [15] proposed a method similar to the now standard CGAN method for increasing the realism of synthetic 3D model eye regions. GazeGAN [17] worked on a similar task, but instead used a framework based on CycleGAN [18] and showed that gaze estimation algorithms for mobile devices saw an increase in performance when trained with the resulting refined synthetic data. [19] used a GAN-based algorithm for pixel-level domain adaptation of synthetic datasets for both classication

and pose estimation and showed that models trained on the refined synthetic data outperformed those trained on only synthetic data as well as refined data from previous methods. [20] also used a framework similar to CycleGAN [18] to improve the realism of synthetically generated scenes used for semantic segmentation and reactive obstacle avoidance.

There is, however, limited research on improving the realism of synthetic data in the form of entire 3D model faces. StyleRig [21] used a modified version of StyleGAN [4] to take so called *3D Morphable Face Models* (3DMMs) as input and produce realistic, controllable human faces. However, the framework did not allow for controllable gaze direction and used a complicated architecture to control the output of the pretrained StyleGAN model. Furthermore, the framework was not capable of perfectly preserving changes in the synthetic faces, possibly due to bias in the data that the StyleGAN model had been trained on. The study was also limited to generating faces in RGB and not in near-infrared, which is the format commonly used in driver monitoring systems. Because of this, there is a need for further research into using GAN frameworks to improve the realism of synthetic 3D model faces while preserving domain invariant attributes, e.g. head pose and gaze direction, while simultaneously mapping the resulting images to the near-infrared domain.

1.3 Problem statement

The thesis will investigate whether GANs can be used to perform image-to-image translation between the two following image domains: the *source domain*, consisting of RGB images of a synthetic 3D model face, and the *target domain*, consisting of near-infrared images of real human faces. The generated images should ideally be realistic and preserve domain invariant attributes, e.g. head pose and gaze direction. This would allow Smart Eye to generate customizable, realistic and diverse training data that could be used to train their DMS and possibly reduce the need for collecting real data.

The research questions that the thesis attempts to answer include:

- How can a GAN framework be implemented for the purpose of image-to-image translation from the source domain to the target domain?
- How can the diversity of the generated faces be improved?
- Can domain-invariant attributes such as gaze direction and head position be preserved by the framework?
- Can the appearance of the generated faces be made controllable?

1.4 Thesis scope

The thesis focuses on using conditional GANs (explained in Chapter 2.3) to implement the mapping between the two image domains. This is because the available dataset consists of paired images from the two domains and therefore lends itself well to a conditional approach. Other studies have approached similar problems by instead using methods inspired by CycleGAN [18], where preservation of domain-invariant attributes is enforced by a so called cycle consistency loss. However, a cycle consistency loss is typically used for learning a mapping between image domains when paired images are not available. Because it requires the identity of each image to be retrievable in its corresponding mapped image, using it for this application could lead to less diversity in the generated faces and cause them to be similar in appearance to the 3D model instead of spanning a wide variety of different appearances. Thus, using a CGAN was seen as the more appropriate approach.

However, the thesis will not compare a broad range of different CGAN frameworks. Instead, the architectures that are seen as the most promising for this particular application will be chosen and used to attempt to answer the above research questions.

Furthermore, the thesis aims to have the CGAN framework implement a multimodal mapping, meaning that the 3D model can map to a variety of different faces. The 3D model is therefore analogous to a "rig" that provides control over the adjustable attributes, e.g. its head position and gaze direction. Because of this, the appearance of the 3D model is not important, so only a single appearance will be selected and used.

Lastly, the spatial resolution of all images will be set to 128×128 pixels. This is because of two reasons: 1) Smart Eye's other machine learning models that could come to use the generated images typically operate on this resolution and 2) it limits training time and VRAM usage.

1.5 Thesis outline

Next, Section 2 reviews the theoretical foundation required to understand the chosen approach and how it was implemented. It begins with a short summary of the recent progresses within the field of machine learning and how these relate to generative modeling. After this introductory part, a thorough derivation of the theory behind generative modeling and GANs is carried out.

Section 3 describes the methodological procedure, including details of the chosen approach and the conducted experiments. It begins with a presentation of the two image domains and a description of how the image data was prepared, cleaned and preprocessed before it was used to train the GAN framework. Next, the chosen approach of using a CGAN framework is explained and motivated, followed by a presentation of the architectures for the neural networks that were apart of it. The

section concludes with an presentation of the conducted experiments together with the details of how they were performed.

Section 4 presents the results of the conducted experiments and briefly comments on what is shown. Section 5 gives a more in-depth discussion of the results and suggestions for how future works could improve on the framework. The section ends with a summary of the main contributions of the thesis and a discussion of potential implications. Lastly, the conclusions of the thesis are presented in Section 6.

2

Theory

In this section, the theory necessary to understand the topic of the thesis is derived. It starts with an introduction to deep learning and generative modeling in Chapter 2.1. This is followed by a derivation of the theory behind Generative Adversarial Networks (GANs) in Chapter 2.2 as well as a modification called Conditional GANs (CGANs) in Chapter 2.3. Lastly, some common architectures and components used for the neural networks in GAN frameworks are presented in Chapter 2.4.

2.1 Introduction

This chapter provides a brief review of the recent successes in the field of deep learning and how this relates to generative modeling, which is the type of machine learning investigated in this thesis. This is followed by a more in-depth derivation of the theory behind generative modeling and closely related concepts, such as the notion of a data distribution and distance measures for probability distributions.

2.1.1 Deep learning

Machine learning is a rapidly developing field that has undergone what is often called a revolution in recent years. One of the earliest breakthroughs that paved the way for this new phase of machine learning occured in 2012, when a team led by Geoff Hinton at the University of Toronto won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) using a deep convolutional neural network. This challenge consists of developing the best algorithms for classifying objects in the ImageNet dataset, that contains images from a wide variety of real-world environments. At the time, ImageNet contained over 21 million images with objects from over a 1000 different categories [22]. The deep neural network achieved an error rate of 16 %, which was significantly better than the previously best result of 26.2 % and came as a surprise to many in the community [23]. Since then, the field of *deep learning* has grown significantly and conquered many problems that previously were seen as unbeatable by artificial systems.

While the successes of the field of deep learning during the last decade are hard to understate, there is still a lot of ground left to conquer. In fact, a lot of the problems

where deep learning algorithms first outperformed other algorithms tended to be of a *discriminative* nature. These are problems in which the model learns to correctly categorize or value the data that it is given, such as classification or regression [23]. While these problems arise in a wide range of applications and are important for solving many real-world problems, they are not the full story.

Discriminative models are tasked with assigning correct labels to observed data. However, they say little about the nature of the observed data *itself*, i.e. the distribution of the observations. Knowledge of how the content of a dataset is distributed can give new insights into solving a problem and is therefore an important objective. Furthermore, if the distribution of observed data itself is known, *it can be used to sample new data instances*. The area of machine learning dedicated to finding the distribution of the observations in a dataset is known as *generative modeling*, and is currently one of the areas at the forefront of machine learning research. While it can be argued that generative modeling is a much harder problem to solve than discriminative modeling, it is also an important one to solve due to the aforementioned reasons.

2.1.2 Generative modeling

Generative modeling can in theory be applied to datasets of any type, whether it be multivariate data, natural languages, audio, images or videos. Using images as an example, we can imagine that any given dataset has a probabilistic distribution that describes why some images are found in the dataset and other images are not. The task of a generative model is to approximate this distribution as closely as possible, and once found, sample from it to generate new unique data observations that look as if they could have been included in the original dataset [23].

The area of image generation in particular saw rapid improvements during the last decade, largely due to the invention of Generative Adversarial Networks, or GANs. As can be seen in Figure 2.1, image generation of realistic human faces has undergone significant progress even from one year to another. The rightmost image is generated by the StyleGAN [4] architecture, which is still among the state-of-the-art for image generation in terms of realism and resolution.

Figure 2.1: Progress of human face generation in the last decade [24].

2.1.3 Data distributions

Generative models learn to generate data with the same characteristics as the data they are trained on. An equivalent definition of this given in [23] is the following:

> *A generative model describes how a dataset is generated, in terms of a probabilistic model. By sampling from this model, we are able to generate new data.*

While this can be understood intuitively by most, it may not be clear what the *distribution* of a dataset actually means. In this chapter, the notion of a distribution of a dataset will be explained, as well as what it means for a generative model approximate it as closely as possible.

2.1.3.1 Distribution of a dataset

A dataset consists of a number of observations $x_1, x_2, ..., x_N$ that typically share some set of characteristic features. In the case of an image dataset, the features are in the form of pixel values and how these relate to each other. For example, in a dataset of human faces, most images contain two eyes, a nose, a mouth etc.

Mathematically, the observations x_i can be seen as samples from some unknown probability distribution $p_r(x)$ that describes how pixel values are distributed for images in the dataset. In other words, $p_r(x)$ describes a probability mass function in the high-dimensional space of all images, in which images with every possible combination of pixel values are elements.

If we for simplicity let this space of images be represented by a two-dimensional space, the distribution $p_r(x)$ can be illustrated by the blue region in Figure 2.2, denoting the set of points where the distribution is high (over some threshold value). The black dots represent the particular observations in the dataset, which as expected lie within this region.

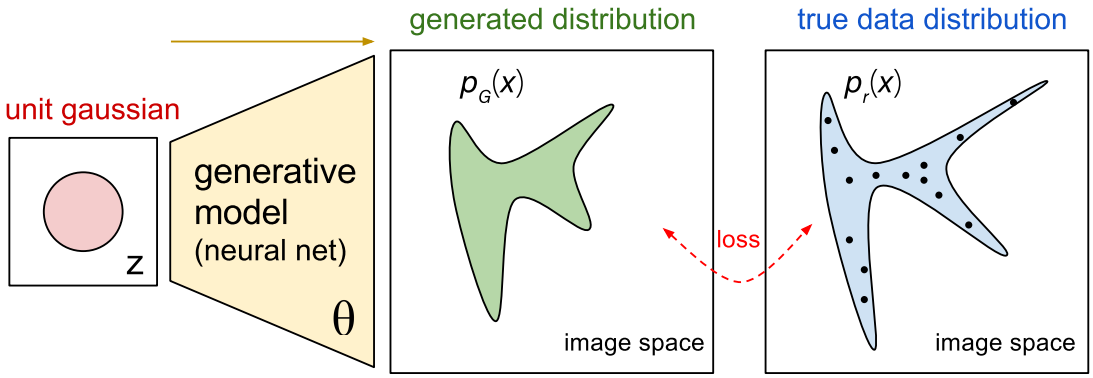


Figure 2.2: The green and blue regions illustrate the generated distribution $p_G(x)$ and real distribution $p_r(x)$, respectively. The goal of a generative model is to maximize the overlapping of the two regions by varying the parameters θ [25].

2.1.3.2 Generative models

The aim of a generative model is to approximate the data distribution $p_r(x)$ as closely as possible. This can be done by having the generative model implement a parameterized function $G_\theta(z)$, and then applying this function to elements z sampled from some fixed probabilistic distribution $p(z)$, called the *prior distribution*. In other words, the generated distribution $p_G(x)$ becomes the image of $p(z)$ under the function G_θ. By changing the parameters θ, the function G_θ, and therefore also the distribution $p_G(x)$, changes. The objective of our generative model can simply be stated as:

> *Find parameters θ of $G_\theta(z)$ such that the generated and real distributions are equal, $p_G(x) = p_r(x)$, for all x.*

Going back to the example of image generation in Figure 2.2, the distribution $p_G(x)$ can be represented by the green region in the same way as before. Changing the parameters θ of $G_\theta(z)$ will change the shape of the green region, and the goal is thus to find parameters θ such that it coincides with the blue region representing the distribution $p_r(x)$.

2.1.3.3 Distance between two distributions

To determine if two distributions are equal, such as the two distributions $p_r(x)$ and $p_G(x)$ above, a notion of the distance between the distributions is required. A quantity that can be used for this is the *Kullback-Leibler divergence* D_{KL}, or KL divergence for short. The KL divergence measures how one probability distribution $p(x)$ diverges from another probability distribution $q(x)$ and is zero when the two distributions are equal, i.e. when $p(x) = q(x)$ everywhere [26]. The KL divergence is defined as

$$D_{KL}\left(p||q\right) = \int_x p(x) \log \frac{p(x)}{q(x)} dx\,. \tag{2.1}$$

However, from Equation 2.1 it is apparent that D_{KL} is asymmetric with respect to the two distributions. In particular, the contribution to D_{KL} is small in cases where $p(x)$ is close to zero but $q(x)$ is significantly non-zero, so the KL divergence fails to capture differences between the two distributions in this case. While this is a desirable property in some applications, it can be problematic in the context of generative models where an inaccurate measure of the similarity between $p_r(x)$ and $p_G(x)$ can lead to undesirable results [26].

A more suitable quantity is the *Jensen-Shannon divergence* D_{JS}. Similar to the KL divergence, the Jensen-Shannon divergence measures the similarity between two probability distributions. It is bounded by $[0, 1]$ and is zero when the two distributions are equal, which makes it appropriate as a distance measure [26]. Most importantly, it is symmetric with respect to the two distributions and therefore avoids the above problem with the KL divergence.

For two probability density distributions $p(x)$ and $q(x)$, the Jensen-Shannon divergence is given by

$$D_{JS}\left(p||q\right) = \frac{1}{2}D_{KL}\left(p||\tfrac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q||\tfrac{p+q}{2}\right) . \tag{2.2}$$

As can be seen, the Jensen-Shannon divergence is defined in terms of the KL divergence to explicitly avoid the asymmetry. Figure 2.3 shows the integrand of the integrals in Equations 2.1 and 2.2 for the two distributions measured against each other as well as against the mean distribution $m(x) = \frac{p(x)+q(x)}{2}$. As can be seen, the KL divergence is asymmetric with respect to the two distributions while the Jensen-Shannon divergence is symmetric.

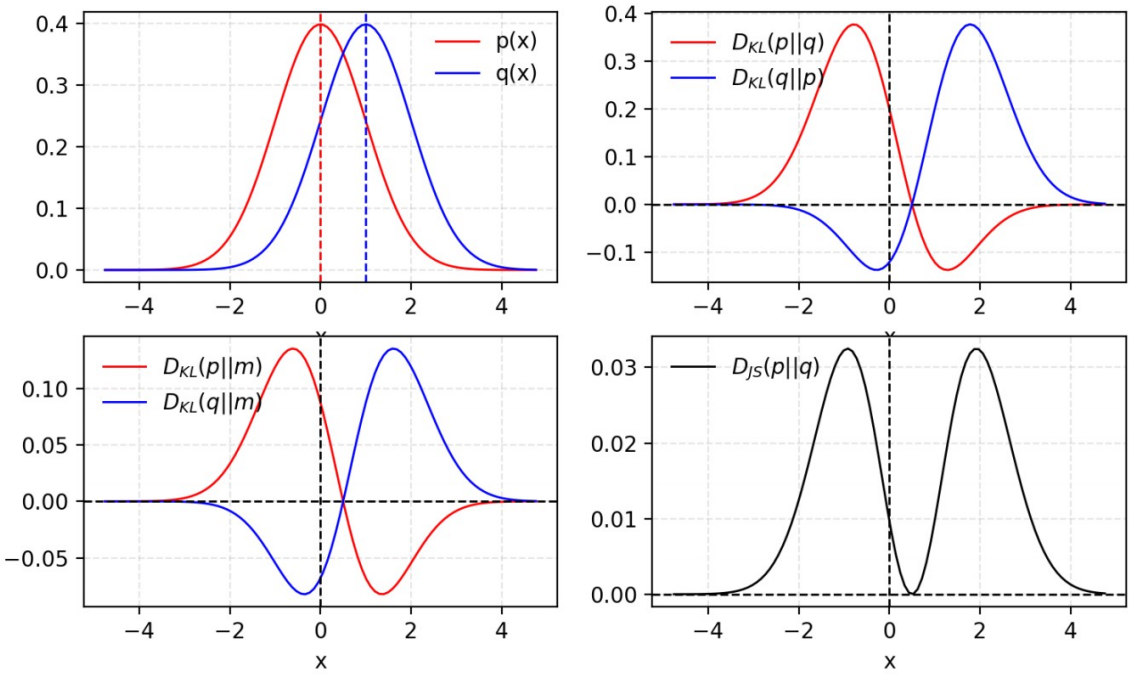


Figure 2.3: The graphs show the integrand of the integrals in Equations 2.1 and 2.2, for the two distributions $p(x)$ and $q(x)$ measured against each other as well as against the mean distribution $m(x) = \frac{p(x)+q(x)}{2}$ [26].

2.2 Generative Adversarial Networks

Generative Adversarial Networks are unsupervised generative models based on deep learning. Their invention is attributed to Ian Goodfellow, who along with colleagues published a paper outlining the idea and coining the term in 2014 [3]. At the time, GANs represented a considerable improvement of the state-of-the-art in generative modeling and the idea is now regarded as a pivotal moment for the field [23]. GANs have since then been applied to a wide variety of problems and achieved remarkable results, especially in the area of image synthesis where they have been shown to be capable of producing images of striking realism and high resolution [4].

Like any generative model, GANs can be used to generate data of many different types. However, since this thesis will be using GANs for image generation, the theory behind them will from hereon be explained in the context of image data.

2.2.1 How does a GAN work?

The basic architecture of a GAN consists of two separate models: the *generator* G that produces images and the *discriminator* D that distinguishes between generated and real images. During training, the two models are opponents, or *adversaries*, in a type of game. The goal of the generator in this game is to generate images that the discriminator cannot tell apart from real images, and the goal of the discriminator is to correctly classify images it is given as either generated and real. The goals of the two models are therefore opposite, such that any gain for one of them comes at the expense of the other.

This scenario is an example of a *minimax game* (a name will be made clearer in Chapter 2.2.3.1). It can be shown that this game theoretically results in the generator ultimately becoming so good at generating images that they are indistinguishable from real images [3].

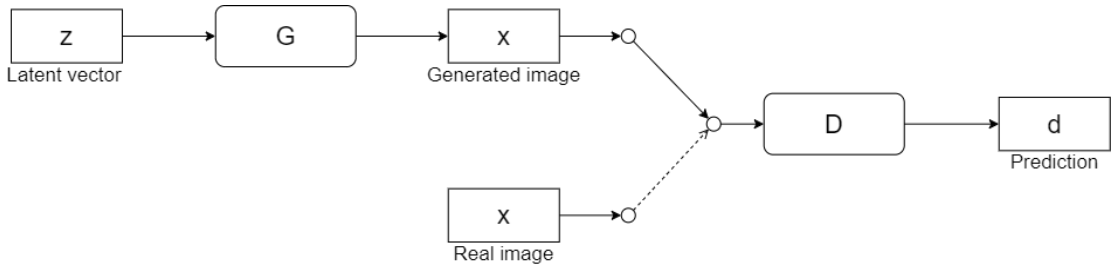


Figure 2.4: Schematic diagram of the GAN framework.

2.2.2 The Goodfellow GAN

The original GAN paper [3] suggests to implement the two models as multilayer perceptrons, or in other words, as fully-connected feedforward neural networks. The generator is trained to map vectors z from the input domain $\mathbb{Z}$ to images $\hat{x}$ in the image domain $\mathbb{X}$. The vectors z are randomly sampled from some probability distribution $p(z)$, called the *prior distribution*, and fed to the generator which produces images $\hat{x} = G(z)$ with distribution $p_G(\hat{x})$. The input domain $\mathbb{Z}$ generally has a smaller dimension than the output domain $\mathbb{X}$, and over the course of training, the generator learns to interpret the lower-dimensional elements $z \in \mathbb{Z}$ as representations of features in the generated images. In this case, $\mathbb{Z}$ is called a *latent space*.

Being a neural network, the generator implements a parameterized function G_θ, where θ are the weights in the network layers. The goal of the generator is to learn weights θ such that the discriminator cannot distinguish the generated images

$G_\theta(z)$ from real images x. This is equivalent to learning weights θ such that the distribution of the generated images $p_G(\hat{x})$ equals the distribution of the images in the training data set $p_r(x)$, i.e. $p_G = p_r$, as was shown in Chapter 2.1.3.2.

Similarly, the discriminator is a neural network that maps images, either generated or real, to a scalar value d that is the estimated probability that the input is real. It obviously follows that the probability that the image is generated, i.e. not real, is given by $1 - d$. The discriminator is therefore simply a classifier network that also implements a parameterized function D_ϕ that takes images x as input and that ideally outputs $D_\phi(x) = 1$ for real images and $D_\phi(\hat{x}) = 0$ for generated images when the network weights ϕ are optimal.

2.2.3 Training a GAN

While the goals of each network have now been determined, it remains to see how the two networks should be trained in order to achieve them. The answer is to express each goal in terms of minimizing or maximizing *objective functions*. The objective function that is used to train a GAN was in fact one of the main innovations of the original 2014 study [3] and is derived below.

2.2.3.1 Adversarial loss function

Beginning with the generator, its goal is to produce images $\hat{x} = G_\theta(z)$ that the discriminator classifies as real, that is

$$D_\phi(\hat{x}) = 1.$$

Now, let the correct prediction for a given image be denoted by q, i.e. $q(x) := 1$ for real images x and $q(\hat{x}) := 0$ for generated images $\hat{x}$. To quantify the prediction error of the discriminator, the *cross entropy* $-q(x) \log D_\phi(x)$ can be used. To fool the discriminator, the generator's objective is to maximize the expected value of the cross entropy, or equivalently, minimize its inverse. Using that $\hat{x} = G_\theta(z)$ for generated images, the generator's objective function becomes

$$\min_\theta \mathcal{L}_G = \min_\theta \mathbb{E}_{\hat{x}\sim p_G(\hat{x})}\, q(\hat{x}) \log D_\phi(\hat{x}) = \min_\theta \mathbb{E}_{z\sim p_z(z)}\, q(G_\theta(z)) \log D_\phi(G_\theta(z)) \quad (2.3)$$

where θ are the weights of the generator network. The expression $\mathbb{E}_{x\sim p(x)}$ should be read as "the expected value when x is distributed according to $p(x)$".

However, since $q(G_\theta(z)) = q(\hat{x}) := 0$ for generated images, the two rightmost expressions in Equation 2.3 will be constantly equal to 0, making them meaningless as objective functions. This can be avoided without changing the objective by replacing the discriminator's prediction in Equation 2.3 by $1 - D_\phi(x)$ and setting the correct label for generated images equal to $1 - q(\hat{x}) = 1$. Using this, the new equivalent objective function becomes

$$\min_\theta \mathcal{L}_G = \min_\theta \mathbb{E}_{z\sim p_z(z)} \log \Big[1 - D_\phi(G_\theta(z))\Big] \quad (2.4)$$

which is the final objective function for the generator.

Next, the objective function for the discriminator is derived. The discriminator has two independent goals: 1) classify real images x as real, $D_\phi(x) = 1$, and 2) classify generated images $G_\theta(z)$ as "not real", $D_\phi(G_\theta(z)) = 0$. This can be expressed in terms of the cross entropy as well, but now the objective is to maximize it. This gives

$$\max_\phi \mathcal{L}_D = \max_\phi \mathbb{E}_{x\sim p_r(x)} q(x) \log D_\phi(x) + \mathbb{E}_{z\sim p_z(z)}\, q(G_\theta(z)) \log D_\phi(G_\theta(z))$$

Again, using that $q(x) := 1$ and employing the trick of reversing the image labels and discriminator prediction in the rightmost term, we get that the discriminator's final objective function is

$$\max_\phi \mathcal{L}_D = \max_\phi \mathbb{E}_{x\sim p_r(x)} \log D_\phi(x) + \mathbb{E}_{z\sim p_z(z)} \log \Big[1 - D_\phi(G_\theta(z))\Big] \tag{2.5}$$

where ϕ are the weights of the discriminator network.

Notice that the rightmost term in both Equation 2.4 and Equation 2.5 are equivalent, but the goal is to minimize it in Equation 2.4 and to maximize it in Equation 2.5. Since the parameters θ and ϕ of the generator and discriminator can be changed independently of one another, we can combine the two objective functions into a single *minimax objective function*

$$\min_\theta \max_\phi \mathcal{L}_{GAN} = \min_\theta \max_\phi \mathbb{E}_{x\sim p_r(x)} \log D_\phi(x) + \mathbb{E}_{z\sim p_z(z)} \log \Big[1 - D_\phi(G_\theta(z))\Big] \tag{2.6}$$

which is the full objective function of the Goodfellow GAN.

2.2.3.2 Global optimum

In the original GAN paper (see Proposition 1 in [3]), it is shown that the optimal discriminator D^* for a given generator G is given by the expression

$$D^*_G(x) = \frac{p_r(x)}{p_r(x) + p_G(x)} \tag{2.7}$$

Using this, [3] shows that the global optimum of the adversarial objective function, Equation 2.6, results in minimization of the Jensen-Shannon divergence (presented in Chapter 2.1.3.3) between the real and generated data distributions p_r and p_G. This is equivalent to the two distributions being equal, $p_r = p_G$, which shows that the objective function results in the desired outcome for the game between the two adversarial networks. Thus, even though the derivation in Chapter 2.2.3.1 motivated the terms in the objective function using cross entropy, there is in fact a rigorous theoretical motivation for why the objective function is defined exactly in the form that it is.

2.2.3.3 GAN training algorithm

In addition to proposing the objective function in Equation 2.6 and showing that it is equivalent to minimization of the Jensen-Shannon divergence between p_r and p_G, [3] also provides an algorithm for training the two networks such that Equation 2.6 is optimized. In short, the proof (see Proposition 2 in [3]) shows that if the discriminator is assumed to be optimal at all times, and thus is given by Equation 2.7, Equation 2.6 has a unique optimum that is reachable by performing gradient descent updates of the generator's parameters θ.

In practice, however, it is enough that the discriminator remains approximately optimal as the generator is updated. To stay near optimality, i.e. to accurately distinguish between generated and real images, the discriminator therefore also has to be updated using gradient ascent on Equation 2.6 during training. Thus, the algorithm that [3] proposes suggests that the generator and discriminator networks are trained in an alternating fashion. For every gradient descent update step of the generator, the discriminator is also updated using one or multiple gradient ascent steps. A simplified version of the algorithm proposed by [3] is shown below in Algorithm 1, where the discriminator is updated once for every update of the generator. In addition to the already declared parameters, ∇ denotes the gradient and h the learning rate.

Algorithm 1: GAN training algorithm.

for *each training iteration* **do**

Sample real images $\{x_1, ..., x_N\}$ from the dataset;

Sample latent vectors $\{z_1, ..., z_N\}$ from prior distribution $p_z(z)$;
Generate fake images $\{G_\theta(z_1), ..., G_\theta(z_N)\}$;
Update discriminator weights ϕ by stochastic gradient ascent on $\mathcal{L}_{\text{GAN}}$

$$\phi \leftarrow \phi + h\nabla_\phi \frac{1}{N} \sum_i^N \Big[\log D_\phi(x_i) + \log \Big[1 - D_\phi(G_\theta(z_i)) \Big] \Big];$$

Sample latent vectors $\{z_1, ..., z_N\}$ from prior distribution $p_z(z)$;
Generate fake images $\{G_\theta(z_1), ..., G_\theta(z_N)\}$;
Update generator weights θ by stochastic gradient descent on $\mathcal{L}_{\text{GAN}}$

$$\theta \leftarrow \theta - h\nabla_\theta \frac{1}{N} \sum_i^N \log \Big[1 - D_\phi(G_\theta(z_i)) \Big];$$

end

Note that any other standard gradient-based learning rule can be used instead of stochastic gradient descent (ascent) to update the networks as well.

2.2.3.4 Alternative adversarial loss functions

Although the original adversarial loss function 2.6 captures the goal for each of the two networks, it has been shown to suffer from shortcomings that can lead to difficulties during training. One of the biggest disadvantages is that it tends to result in vanishing gradients when the discriminator is near optimality, which can cause learning to slow down significantly [27]. Because of this, several modifications

to Equation 2.6 that are designed to avoid the problem of vanishing gradients have been proposed as alternative adversarial loss functions to train GANs.

In the original adversarial loss of Equation 2.6, the discriminator outputs a probability $D_\phi(x) \in [0, 1]$ that x is real. In practice, this is done by passing the output of the discriminator's last layer y through a function that normalizes it to the range $[0, 1]$, most commonly the sigmoid function $\sigma(y) = \frac{1}{e^{-y}+1}$. However, several studies trace the problem of vanishing gradients to the use of functions like the sigmoid, that have saturating derivatives for large y [27, 28, 29]. Therefore, many alternative adversarial losses avoid the use of such normalization functions and are instead expressed directly in the output y from the discriminator's last layer.

To see examples of alternative adversarial losses, let Equations 2.4 and 2.5 be expressed in terms of y on the form

$$\begin{aligned} \max_D \mathcal{L}_D &= \max_D f(y) + g(y) \\ \min_G \mathcal{L}_G &= \min_G h(y)\,. \end{aligned}$$

Using these expressions, a couple of common alternative adversarial loss functions are presented in Table 2.1[1].

Loss function	$f(y)$	$g(y)$	$h(y)$
Original [3]	$-\log(1+e^{-y})$	$-y-\log(1+e^{-y})$	$-y-\log(1+e^{-y})$
Least squares [29]	$-\frac{1}{2}(y-1)^2$	$-\frac{1}{2}y^2$	$\frac{1}{2}(y-1)^2$
Hinge [30, 31]	$\min(0, y-1)$	$\min(0, -y-1)$	$-y$
Wasserstein [32]	y	$-y$	$-y$

Table 2.1: Alternative adversarial loss functions [33].

An important remark, however, is that a large-scale study by Google Brain [28] tested a large number of alternative loss functions and demonstrated that none was superior on all types of problems. Instead, there are advantages and disadvantages of each, and which one performs the best can vary depending on the problem.

2.2.4 Mode collapse

In Chapter 2.1.3.2, it was shown that the generator ideally learns to accurately model the data distribution $p_r(x)$. When this succeeds, the generator is able to produce a diversity of images with a broad range of appearances. However, a common problem during GAN training is that the generator stops producing diverse

[1]Note that with the sigmoid function applied to y, the terms in the original loss of Equation 2.6 function become $\log D_\phi(x) = \log \frac{1}{e^{-y}+1} = -\log(1+e^{-y})$ and $\log(1-D_\phi(x)) = \log(1-\frac{1}{e^{-y}+1}) = \log(\frac{e^{-y}}{e^{-y}+1}) = -y-\log(1+e^{-y})$.

images and instead only outputs a small number of samples. This is called *mode collapse* and occurs when the generator finds one or a few samples, called *modes*, that fool the discriminator particularly well, or in other words, lead to a low loss for the generator. If the benefit is high enough, the generator can learn to *only* produce images belonging to this limited set of modes. When this happens, the discriminator's best strategy becomes to reject images based solely on whether they belong to this small set of modes or not, instead of learning to discriminate based on the high-level features of the images. However, when the discriminator does this, the generator can then simply shift to another small set of modes to again start fooling the discriminator. This phenomenon can becomes a vicious cycle that continues endlessly and that stops the two networks from making progress.

2.3 Conditional GANs

Conditional Generative Adversarial Networks, CGANs, are modifications to the GAN framework described above. As the name suggests, both the generator and discriminator now produce their outputs *conditioned* on some additional information, such as a class label or an image [34].

2.3.1 Generator

In the case of the generator, it no longer produces fake images only using the input elements as before, where $\hat{x} = G(z)$. Instead, it takes both the latent vectors z and an additional piece of information y as input and produces an image conditioned on y

$$G(y, z) = \hat{x}|y\,.$$

The expression on the right-hand side should be interpreted as "$\hat{x}$ conditioned on y" [34].

In the case of image generation, y can represent some feature of the image that the generator should learn to include. For example, if the generator is trained to produce pictures of dogs and cats, y can encode the two cases *cat* and *dog* and tell the generator which of the two cases it should produce an image for.

However, only having the generator be conditioned on y would not be sufficient to make it learn the correct mapping. The generator could simply learn to produce random fake images without actually making use of y. As long as the generated images were equally realistic to the real images that are also given to the discriminator, they would fool it and the behaviour would be encouraged by the loss function.

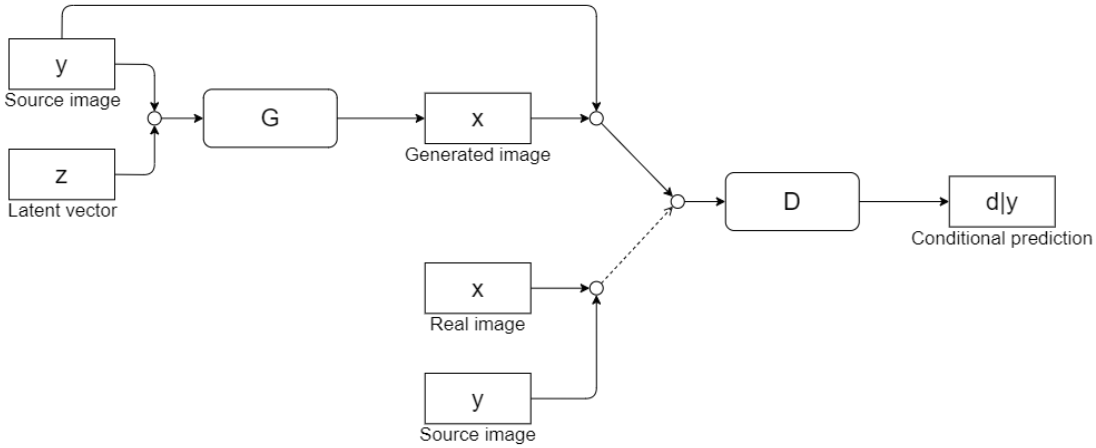


Figure 2.5: Schematic diagram of the conditional GAN framework.

2.3.2 Discriminator

To force the generator to learn the correct mapping $G(y, z) = \hat{x}|y$, the discriminator is also trained to be conditioned on y. More specifically, instead of outputting the probability that an image is real $D(x) = d$, it is also given the label y and outputs a conditional probability based on the *pair* (x, y), that is

$$D(x, y) = d|y\,.$$

Thus, the discriminator learns to both recognize if the image x is real *and* if it matches the label y. For real observations, the pair is the real image and its label (x, y), and for fake observations, it is the generated image and the label used to generate it $(\hat{x}|y, y)$ [34].

2.3.2.1 Image-to-image translation

In the case of y being an image, the generator learns to produce an output image $\hat{x}$ based on the image y and the randomly sampled input z. This type of problem, where the goal is to learn a mapping from images in a source domain $\mathbb{Y}$ to images in a target domain $\mathbb{X}$, is called *image-to-image translation*. The input elements z in the latent space $\mathbb{Z}$ then serve as stochastic variations in the generated images that provide variation.

2.4 Architectures

Up to this point, the generator and discriminator networks have only been discussed in abstract, without any details regarding their actual implementations as neural networks. In this chapter, a number of components commonly used as a part of their architectures are presented.

This chapter will not, however, present a full summary of the basics of deep learning. The reader is assumed to be familiar with foundational concepts such as artificial

neurons, convolutional neural networks (CNNs), common activation functions etc. See for [35] a brief summary of these concepts. For a more comprehensive review of deep learning, the author recommends [36] as reading material.

2.4.1 Residual networks

A reoccurring problem when training deep neural networks is the phenomenon of *vanishing gradients*, where the gradients of the early layers with respect to the loss function become very small and give rise to slow learning [23]. In general, this happens due to the saturating activation functions of subsequent layers occuring in the expression for the gradient. However, it has been shown that *residual networks*, or *ResNets*, can be trained with hundreds or even thousands of layers without suffering from the vanishing gradient problem [37]. Deep network architectures are common in GAN applications, so ResNets are often used in the architectures of both the generator and discriminator networks.

One way to construct a residual network is to build it from smaller blocks called residual blocks, or *ResBlocks*. ResBlocks are usually connected in a sequence, possibly with upsampling or downsampling layers in between to transform the size of the feature maps depending on the application. The key feature that prevents vanishing gradients is the *skip connection* included in each ResBlock. The skip connection bypasses the layers inside the ResBlock and allows information from previous layers to directly influence subsequent layers. This can be shown to remedy the vanishing problem problem in the expression for the gradient [37]. The output y of the ResBlock is the sum of the output from the internal layers $F(x)$ and the skip connection x, or in other words

$$y = F(x) + x\,.$$

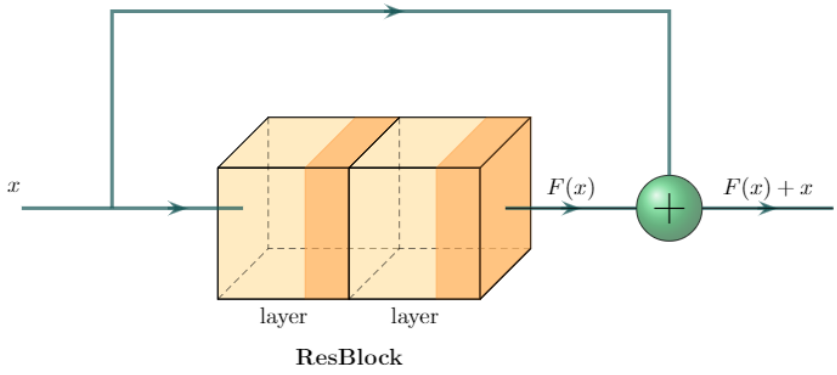


Figure 2.6: A schematic diagram of a ResBlock used to construct residual neural networks. In this example, the ResBlock has two internal weight layers.

2.4.2 Normalization

A neural network can be seen as a composition of many functions that are applied on top of each other. During training of the network, the parameters of these functions are updated simultaneously according to some optimization algorithm, typically some type of gradient descent. However, each component in the gradient is calculated with the implicit assumption that all other parameters are constant. When the weights of all layers are changed simultaneously under this assumption, it can lead to unexpected and explosive changes in the internal activations and even cause an overall collapse of the network [36]. This is known as the *exploding gradient problem* and is especially prominent for deep neural networks [23]. One way to avoid these explosive changes in the activations is to apply normalization to keep their statistical properties constant during training[2]. Normalization can also be used to control network activations dynamically at runtime, called *adaptive normalization*. A couple of normalization methods of these types are presented below.

2.4.2.1 Batch normalization

One way to avoid the problem of exploding gradients due to simultaneous weight updates is to apply *batch normalization* [40] before each layer in the network. Batch normalization is performed by simply calculating the mean and standard deviation across all data instances in a minibatch and normalizing each instance accordingly. Since this ensures that the statistical properties of the inputs to all layers remain unchanged over time, it alleviates the problem of exploding gradients caused by unexpected changes in previous layers [23].

Let $\boldsymbol{x} \in \mathcal{R}^{N \times C \times W \times H}$ denote a minibatch of data instances as a tensor. Each element is then described by four indices: w and h for the spatial dimensions, c for the channel number and n for the index in the batch. Using this, the channel-wise mean μ_c and standard deviation σ_c for a minibatch of length N are given by

$$\mu_c = \frac{1}{NHW} \sum_{n=1}^{N} \sum_{w=1}^{W} \sum_{h=1}^{H} x_{nchw} \qquad \sigma_c^2 = \frac{1}{NHW} \sum_{n=1}^{N} \sum_{w=1}^{W} \sum_{h=1}^{H} (x_{nchw} - \mu_c)^2 \,.$$

Using these, batch normalization transforms each element x_{nchw} according to

$$x'_{nchw} = \frac{x_{nchw} - \mu_c}{\sigma_c} \,.$$

2.4.2.2 Instance normalization

Instance normalization is another normalization method similar to batch normalization, but instead of normalizing across all instances in a batch, the channel in every

[2]This is the most common explanations for why normalization improves the stability and performance of neural networks, but it should be remarked that this has not been rigorously proven [38]. In fact, recent studies [39] suggest completely different reasons for why methods such as batch normalization improves the performance of neural networks.

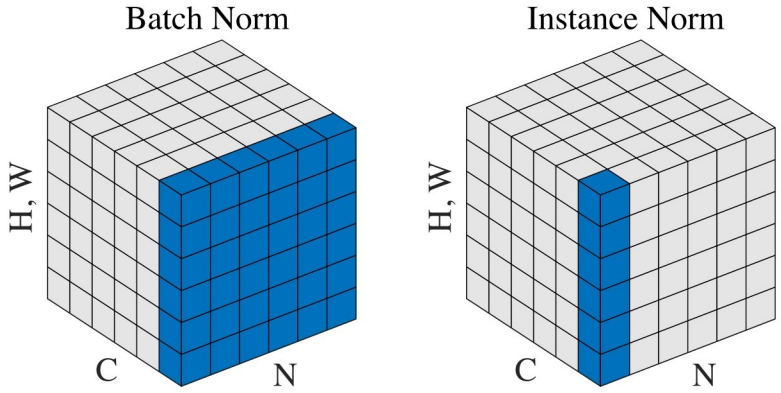


Figure 2.7: Visual comparison between batch normalization and instance normalization [42].

data *instance* is normalized separately. It was first introduced in a 2016 study [41] and was shown to be particularly beneficial for style transfer applications and has since then been used in many other GAN studies [4, 1].

Again, let $\boldsymbol{x} \in \mathcal{R}^{N \times C \times W \times H}$ denote a minibatch of data instances indexed by h and w for the spatial dimensions, c for the channel number and n for the index in the batch. For instance normalization, each data instance is normalized according to its channel-wise mean μ_{nc} and standard deviation σ_{nc}. For a minibatch of length N, these are given by

$$\mu_{nc} = \frac{1}{HW} \sum_{w=1}^{W} \sum_{h=1}^{H} x_{nchw} \qquad \sigma_{nc}^2 = \frac{1}{HW} \sum_{w=1}^{W} \sum_{h=1}^{H} (x_{nchw} - \mu_{nc})^2 \,.$$

Instance normalization uses these to transform each element x_{nchw} according to

$$x'_{nchw} = \frac{x_{nchw} - \mu_{nc}}{\sigma_{nc}} \,.$$

2.4.2.3 AdaIn - Adaptive Instance Normalization

Adaptive instance normalization [43, 44, 45, 46], AdaIn, is a technique for normalizing the activations in a network and rescaling them according to learned parameters. More specifically, the input is first normalized using instance normalization and then denormalized by the learned adaptive parameters γ_{nc} and β_{nc} that apply channel-wise scaling and biasing.

Using the same notation as before, AdaIn transforms each element according to

$$x'_{nchw} = \gamma_{nc} \frac{x_{nchw} - \mu_{nc}}{\sigma_{nc}} + \beta_{nc} \,. \tag{2.8}$$

Again, the parameters γ_{nc} and β_{nc} are *learned* and *adaptive*, meaning that they are optimized according to some desired objective and can change dynamically at runtime. Typically, they are outputted by a parameterized function that takes some external information as input. In particular, this external information can be in the form of randomly sampled latent vectors z, and is therefore an alternative way to inject these into a network instead of providing them as input.

Since γ_{nc} and β_{nc} are applied channel-wise to each data instance, AdaIn is a technique of dynamically changing which feature maps are used to generate images at runtime based on external output, and in this way, enables the style of the output to be controlled. For example, AdaIn layers were used in the architecture of the StyleGAN [4] generator to provide control over the appearance of the generated photo-realistic faces.

2.4.2.4 SPADE - Spatially Adapative Normalization

SPADE [8] is short for spatially-adaptive normalization and is a technique used to introduce the conditional labels y in image-to-image translation applications, where y are images. The method works by inferring learned, adaptive and spatially-dependent parameters from the image y that are used to denormalize the feature maps inside the network. The learned parameters are spatially-dependent, meaning that they vary across the spatial coordinates of the feature maps, and adaptive, meaning that they are calculated dynamically from y. They are also learned, meaning that they are outputted by a parameterized function that is optimized according to some objective. In short, y is used to calculate a type of learned "filter" that is applied to the feature maps inside the network at runtime.

In the original study [8], SPADE is implemented as a layer that first normalizes the input using batch normalization and then denormalizes it according to spatially-dependent scale and bias factors that are calculated by applying multiple convolutional layers to y, as shown in Figure 2.8. The formula for this is given by

$$x_{nchw} = \gamma_{nchw} \frac{x_{nchw} - \mu_c}{\sigma_c} + \beta_{nchw} \,. \tag{2.9}$$

Comparing with the formula for AdaIn, Equation 2.8, it can be seen that SPADE differs in two ways: 1) batch normalization is used as the initial normalization and 2) the parameters γ_{nchw} and β_{nchw} have spatial dependence. Another difference is how these parameters are calculated. In AdaIn, they are typically calculated from randomly sampled latent vectors z, while in SPADE, they are calculated using convolutional layers applied to the source image y.

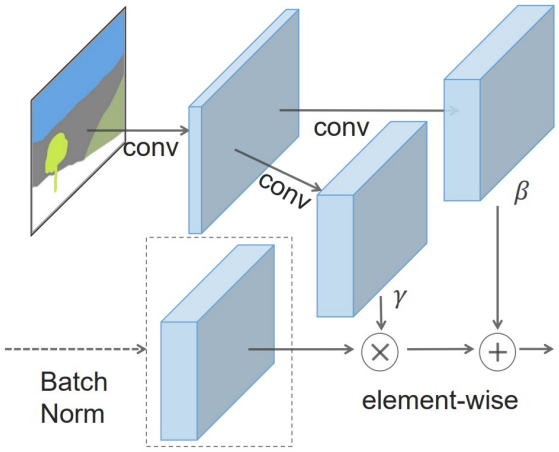


Figure 2.8: SPADE implemented using batch normalization and convolutional layers that are applied to the input image to calculate learned spatially-dependent parameters [8].

2.4.3 Network architectures

The following chapter presents network architectures that are commonly used for the generator and discriminator networks in GAN frameworks.

2.4.3.1 Decoder

A *decoder* is a common generator architecture for implementing the mapping from the latent space $\mathbb{Z}$ to the image domain $\mathbb{X}$. A decoder typically consists of a series of weight layers, e.g. convolutional or fully-connected layers, with upsampling layers in between. Thus, the inputs $z \in \mathbb{Z}$ are repeatedly transformed and upscaled until their dimensions match that of the output image domain X. To allow a deep architecture, the encoder can also be implemented as a ResNet, i.e. by a series of ResBlocks with upsampling layers in between.

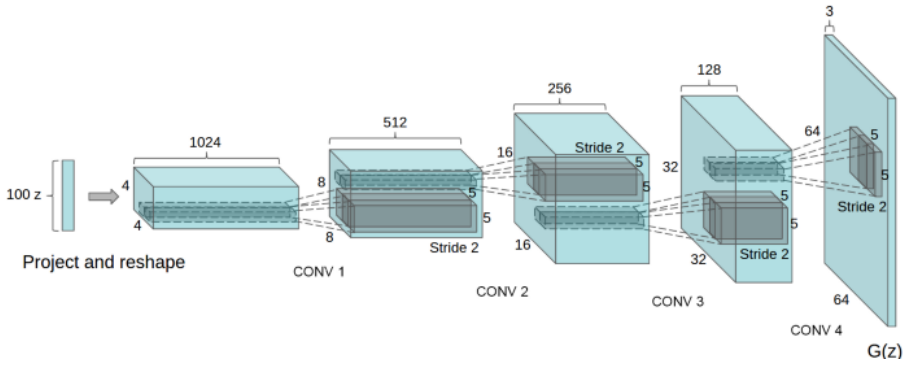


Figure 2.9: Decoder generator architecture proposed by the DCGAN [47] paper.

The decoder architecture shown in Figure 2.9 using convolutional layers was first

proposed by the DCGAN [47] paper. This study proposed several modifications that significantly improved the performance of GANs at the time (2016) and is today considered one of the major breakthroughs in the field.

2.4.3.2 Encoder

An *encoder* is an architecture that takes images as input and maps them to a lower-dimensional latent space. The architecture is similar to a reversed decoder, i.e. an encoder typically contains a series of weight layers with downsampling layers in between. Alternatively, like with the decoder, it can also be constructed using a series of ResBlocks, each followed by downsampling layers, to allow for deeper architectures.

In the context of GANs, encoders can be used as the architecture for the discriminator. However, as will shown in the next Chapter 2.4.3.3, it can also be a part of the generator during image-to-image translation.

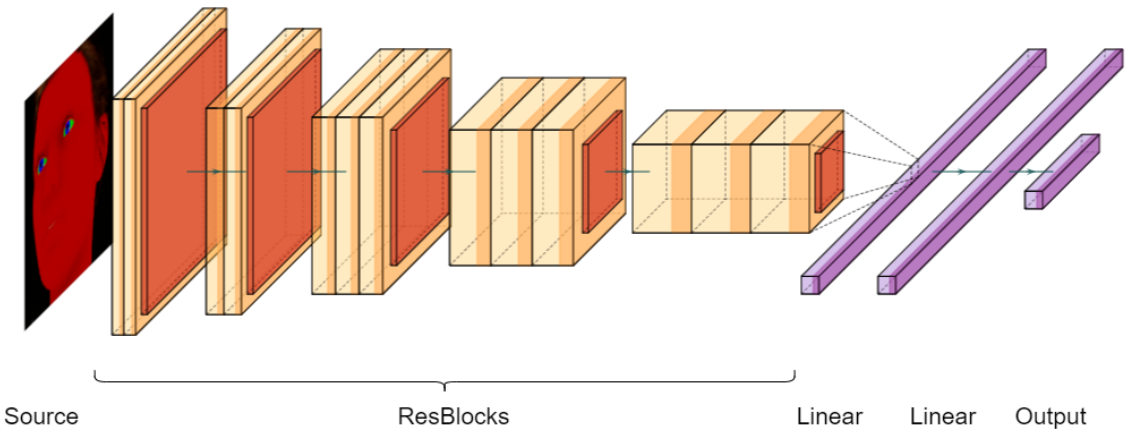


Figure 2.10: An example of an encoder implemented as a ResNet. The input image is encoded, or compressed, down to a vector representing its high-level features.

2.4.3.3 Encoder-decoder

This architecture is common for the generator during image-to-image translation and can be thought of as an extension to the decoder architecture explained in Chapter 2.4.3.1. To allow the generator to take images y as input, an encoder is appended before the decoder, such that they together form the full generator – an *encoder-decoder*. The encoder maps the images y to the intermediate latent space in the bottleneck, which are in turn passed to the decoder that uses them to produce the output images.

An encoder-decoder can also be implemented as a ResNet, i.e. a series of ResBlocks with downsampling and upsampling layers in between, as is shown in Figure 2.12. This architecture has been shown to work well for image-to-image translation and has been used in studies such as [2, 1].

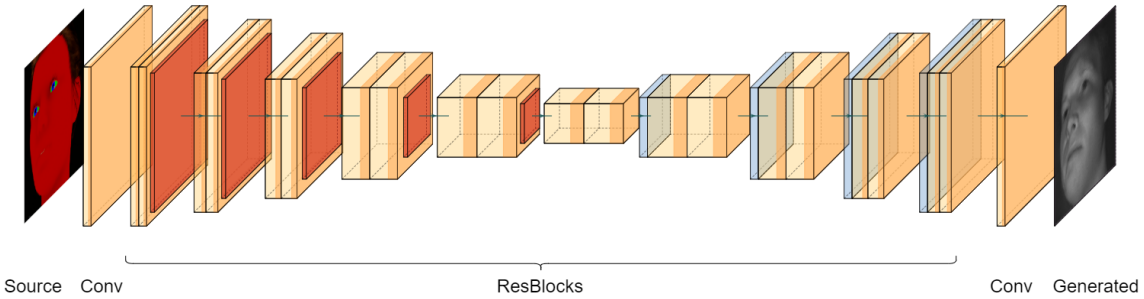


Figure 2.11: An example of an encoder-decoder implemented as a ResNet, where the source image is simply given as input to the network.

2.4.3.4 SPADE decoder

An alternative to using an encoder to introduce the images y into the decoder is to instead use SPADE normalization layers, introduced in Chapter 2.4.2.4. By removing the need for the encoder, this architecture is more light-weight and has less parameters. Additionally, the SPADE [8] study found that architectures that introduce the label images y as direct inputs to the network, as with the encoder-decoder, tend to "wash away" much of the information when convolutions and other transformations are applied. By instead using SPADE layers throughout the entirety of the decoder to introduce y, the full information can be retained even in the deeper layers.

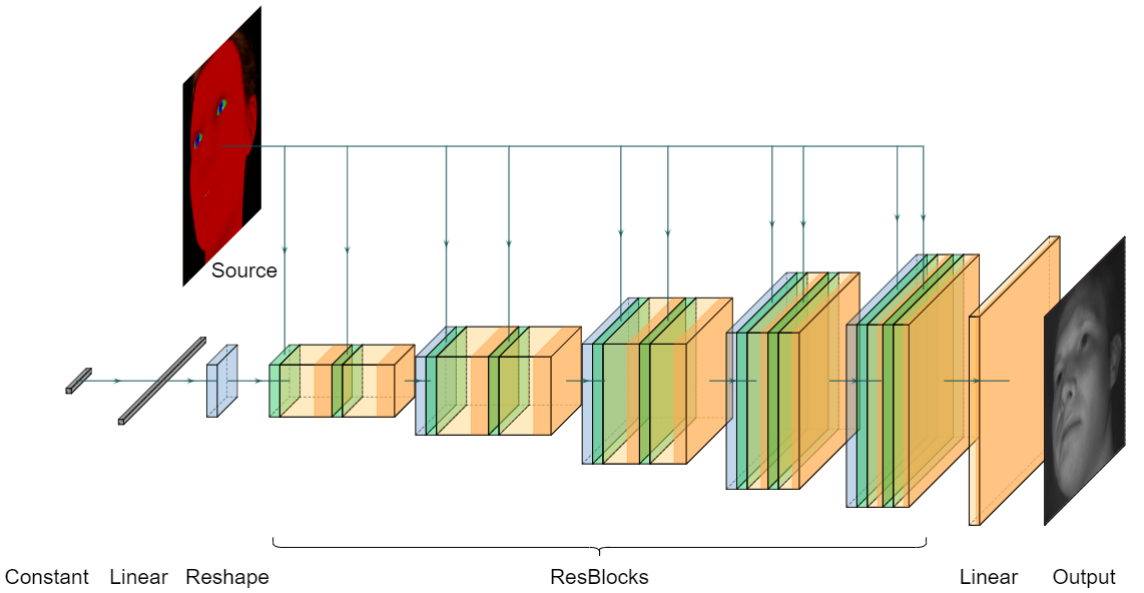


Figure 2.12: An example of a SPADE decoder implemented as a ResNet, where the source image is injected using SPADE normalization layers.

3

Method

This section describes the methodological procedure of the thesis. The section begins by describing the process of preparing and preprocessing the data that is used for training the framework in Chapter 3.1. Next, the details of the chosen approach of using a CGAN framework for performing the image-to-image translation are described and motivated in Chapter 3.2. Chapter 3.3 explains how the framework was evaluated and Chapter 3.4 presents the architectures of the framework's neural networks. Lastly, the chapter concludes by specifying the experiments that were performed as well as the motivation behind them in Chapter 3.5.

3.1 Dataset

The CGAN framework was trained to perform image-to-image translation between two image domains: the *source domain*, consisting of images of a computer generated 3D model, and the *target domain*, consisting of images of real human faces in near-infrared. This subchapter will present the two image domains in more detail and explain how they were acquired. Furthermore, it will describe how the image data was cleaned and preprocessed before it was given to the CGAN framework for training.

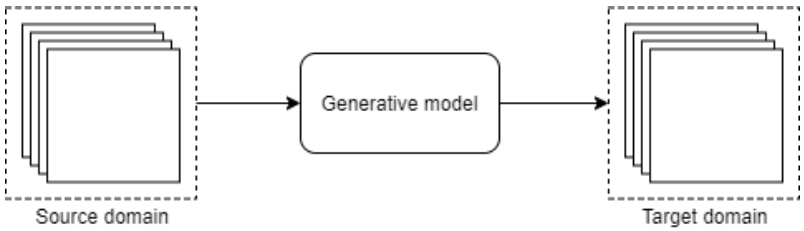


Figure 3.1: Diagram showing image-to-image translation using a generative model.

3.1.1 Source domain

The source domain consists of images of a computer generated 3D model rendered by a software tool based on the Unity [48] game engine. The 3D model is controllable such that its head position and gaze direction can be set to any desired value.

The software tool supports multiple 3D models with different genders, ethnicities and appearances, and each 3D model can be outfitted with accessories such as sunglasses, face masks and various clothing. However, since the aim is to have the CGAN framework produce a wide variety of facial attributes, the appearance of the 3D model was not seen as important, so a single appearance was chosen and used during training of the CGAN framework. The chosen 3D model appearance is shown on the left-hand Figure 3.2

Furthermore, to allow the CGAN framework to more easily infer the geometry of 3D model's face and distinguish the eye region from the rest of the face, the choice was made to use a color coded mask as texture. The mask uses red, green and blue to separate the skin, eye whites and irises respectively. The 3D model with color coded mask is shown on the right-hand side in Figure 3.2, and the three RGB color channels are shown in Figure 3.3.

Without mask **With mask**

Figure 3.2: 3D model with and without color coded mask texture.

Red **Green** **Blue**

Figure 3.3: RGB color channels of the 3D model with mask.

3.1.2 Target domain

The target domain consists of images of real faces in the near-infrared domain. The images are extracted from video recordings made with infrared cameras positioned in a variety of perspectives. Altogether, there are an estimated 500 different unique subjects in the dataset that span a wide variety of appearances. A sample from the dataset is shown in Figure 3.4.

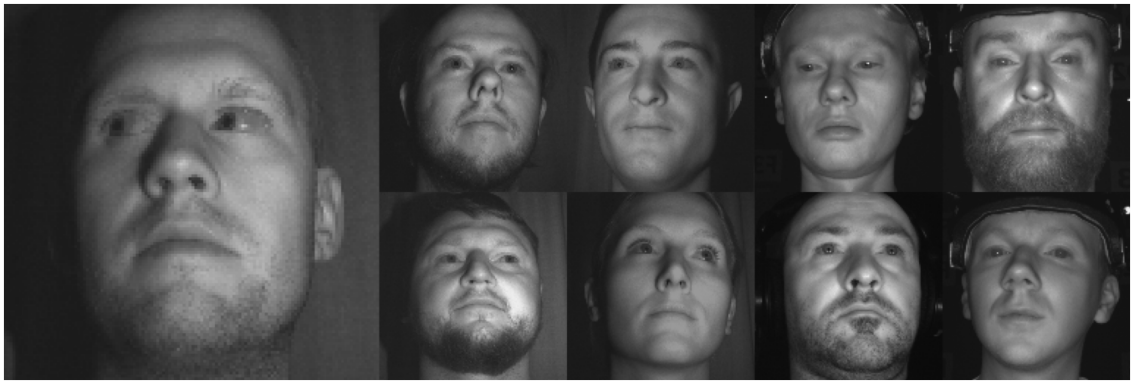


Figure 3.4: Sample images from the target domain. As can be seen, these consist of faces in the near-infrared domain.

3.1.3 Dataset creation

The dataset that is used to train the CGAN framework consists of *paired images* from the two domains. More specifically, the images of the 3D model in the source domain are synthesized from the images in the target domain. This is done by iterating through the images in the target dataset and extracting the following features for each image: the camera's spatial coordinates, the camera angle, the subject's head coordinates, the subject's head angle and the subject's gaze direction. These features are then used to synthesize a source image that is aligned with the target image, as shown in Figure 3.5. Additionally, for target images with a subject wearing glasses, the 3D model in the source image is also rendered with a standardized pair of glasses.

Figure 3.5: Example of an image pair used for training the CGAN framework.

3.1.4 Data cleaning

Since it is the training data that instructs the CGAN framework of what mapping to learn, it is important to provide it with clean data during training. While most

images in the two image domains were of high quality, a small portion contained disturbances such as partially obstructed faces, blur and other disturbances. The dataset creation process described in 3.1.3 also did not work perfectly and in some cases resulted in source images that were not properly aligned with the corresponding real images.

To find images containing visual disturbances or obstructed faces and remove them, a simple method for data cleaning was employed. The target dataset is organized into subdirectories for each unique subject and camera they were recorded with. Since the images in each such subset are similar, they can be statistically compared against each other to find the few examples with significant disturbances.

Thus, to find and remove image outliers, the mean of each image belonging to the same subdirectory was calculated and used to compute the interquartile range $\text{IQR} = Q_3 - Q_1$ from the first and third quartiles Q_1 and Q_3. Images with means that were more than $1.5 \cdot \text{IQR}$ below the first or above the third quartile were classified as outliers and removed from the dataset together with their corresponding paired images in the other domain. This was done for each subdirectory in both the target dataset and source dataset separately.

While this was a reasonably effective method for removing images with distinct visual disturbances, it failed to perfectly capture cases where the source and target images were misaligned. Luckily, image pairs like these constituted a small portion of the full dataset and could be removed manually.

3.1.5 Data preprocessing

Before training, the images were downsized to a resolution of 128×128 pixels. This was partly done for practical reasons to make the training time shorter and limit VRAM usage, but also because Smart Eye's other machine learning models that could come to use the generated images require this resolution.

Next, the target images were each scaled by a factor μ_0/μ in order to normalize their means to the same value μ_0. This was done because of significant lighting differences in the images recorded from different cameras which was found to make it difficult for the CGAN framework to learn the data distribution. Since all pixels in each image were scaled by the same factor, the relationships between them remained unchanged.

After this, the random image augmentations presented below in Chapter 3.1.5.1 were applied to each image pair. This can be seen as a way of artificially expanding the dataset using the existing images. With random augmentations, each image (pair) is alterered slightly every time before it is given to the CGAN framework and is therefore never seen twice in exactly the same way. Applying random augmentations have been shown to reduce overfitting and increase the performance of machine learning models on computer vision tasks [49]. Since the discriminator is a type of image classifier, it is less likely to overfit on the dataset when random augmentations

are used. In a similar way, it prevents the generator from memorizing the correct output for every source image in the dataset.

Lastly, after the previous alterations had been applied, the images from both domains were scaled to have values in the range $[-1, 1]$ through min-max scaling. Bounded activation functions such as tanh have been found to result in shorter training times when applied to the output of the generator, so the image scaling was done in order to make the pixel values lie within the range of the tanh function's output [47].

3.1.5.1 Random augmentation

To reduce the tendency of overfitting and improve model's ability to generalize, random augmentations were applied to each pair of images before they were given to the model. The augmentations were applied in the same way to both images in each pair in order to preserve their alignment. Table 3.1 shows the random augmentations used and the probability of applying them to each image pair.

Augmentation	Probability	Min.	Max.
Scale	0.7	1	1.2
Shift	0.7	-0.0625	0.0625
Rotation	0.7	$-\pi/4$	$\pi/4$
Gamma alteration	0.5	98	102
Horizontal flip	0.5		

Table 3.1: Random augmentations applied to image pairs in the dataset.

3.1.6 Training and test sets

After removal of poor images using the data cleaning described in Chapter 3.1.4, the dataset used to train the CGAN framework consisted of about 180 000 images in total. These were then split into a training set and a test set by randomly extracting 90% of the image pairs for the training set and using the remaining 10% for the test set. It should be noted that the full dataset contained multiple images of each subject, so the above method did allow for the same subject to appear in both the training and test set.

3.2 Framework

This chapter describes the CGAN framework used to perform the image-to-image translation from the source domain to the target domain.

3.2.1 Multimodal image-to-image translation

The CGAN framework was trained to perform image-to-image translation from the source domain to the target domain. More specifically, it took images of the 3D model as input and mapped them to images of real faces in the target domain while preserving domain-invariant attributes, such as head pose and gaze direction. However, while there is only one appearance for the 3D model, there are a wide variety of unique subjects in the target domain. When mapping images of the 3D model, the generator should therefore learn to produce a wide variety of faces with different appearances. This type of problem is known as *multimodal image-to-image translation*.

3.2.2 CGAN framework

The chosen approach for performing the multimodal image-to-image translation is to use a Conditional Generative Adversarial Network (CGAN), described in Chapter 2.3, trained on the image pairs described in Chapter 3.1.3. Both the generator and discriminator will therefore produce their respective outputs conditioned on the source domain images in these pairs.

The generator is thus a neural network that implements a function G that takes the source images y as well as randomly sampled latent vectors z as input and produces a generated image $G(y, z) = \hat{x}|y$. The latent vectors z are sampled from a probability distribution $p(z)$ that can be set as desired before training.

The purpose of the latent vectors z is to provide the generator with stochastic variations that enable it to map the source images to a variety of different faces. Ideally, the generator learns to interpret the latent space $\mathbb{Z}$ as a representation of features of the faces in the target domain. Each latent vector z will then map to a unique appearance in the target domain and allow the generator to produce a diverse set of faces. Without the latent vectors z, the generator would simply map each source image y to a unique target image $\hat{x}$. Figure 3.7 shows a diagram of the framework where the effect of the latent vectors z is illustrated.

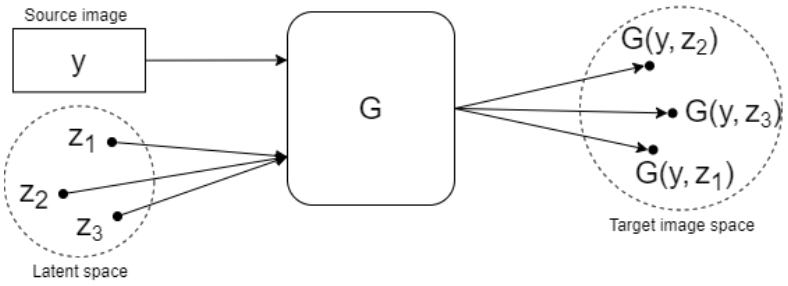


Figure 3.6: The proposed generator architecture for multimodal image-to-image translation.

3.2.2.1 Mode collapse

While the latent vectors z should in theory allow the CGAN generator to produce a variety of faces with unique appearances as shown in Figure 3.7, this is generally hard to achieve in practice. In particular, when the conditional information y is structured and high-dimensional, such as in the case of images, the conditional generator has a tendency to ignore the lower-dimensional latent vectors z and only make use of y when generating images [7]. When this happens, the variety of the appearances in the generated images becomes limited, since it is the latent vectors z that account for the diversity. This can be seen as a type of mode collapse, which was explained in Chapter 2.2.4.

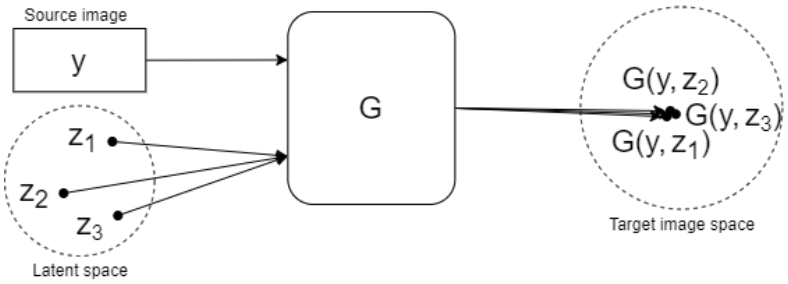


Figure 3.7: Mode collapse occurring, where the generated images $G(y, z)$ are of low diversity.

3.2.2.2 Diversity regulization

There have been numerous proposed methods for solving the problem of mode collapse in CGANs. StarGAN v2 [1] showed that mode collapse can be avoided by introducing *diversity regularization terms* in the loss function of the framework. The study proposes two regularization terms in particular:

3.2.2.2.1 Latent reconstruction loss $\mathcal{L}_{lr}$ StarGAN v2 [1], in turn inspired by [50, 51], showed that the generator can be enforced to make use of the latent vectors z when generating images $G(y, z)$ by introducing a *latent reconstruction loss*

$$\mathcal{L}_{lr} = \min_{G,E} || \, z - E(G(y, z)) \, ||_1 \tag{3.1}$$

where E is an **encoder network** that takes a generated image $G(y, z)$ as input and outputs an estimation $\hat{z} = E(G(y, z))$ for the latent vector z that was used to produce $G(y, z)$.

The latent reconstruction loss $\mathcal{L}_{lr}$ is minimized when the encoder E can make accurate estimations $\hat{z}$. This requires that the latent vectors z can be uniquely retrieved from each image $G(y, z)$, which prevents the generator from ignoring z when generating images.

3.2.2.2.2 Style diversity loss $\mathcal{L}_{sd}$ While the latent reconstruction loss enforces the generator to use the latent vectors z, it is still possible for z to have a minimal effect on the style of the generated images. For example, the generator could in theory minimize $\mathcal{L}_{lr}$ by encoding z in the outskirts of the images $G(y, z)$, without z affecting the actual appearance of the faces they contain.

StarGAN v2 [1], inspired by [7, 52], therefore proposes an additional regularization term for the loss function, called a *style diversity loss* $\mathcal{L}_{sd}$. In order to adopt $\mathcal{L}_{sd}$, two latent noise vectors z_1 and z_2 must be sampled independently at each iteration during training, which are used by the generator to produce the two images $G(y, z_1)$ and $G(y, z_2)$ from the same source image y. $\mathcal{L}_{sd}$ is then given by

$$\mathcal{L}_{sd} = \max_{G} \Big[\frac{d_I(G(y, z_1), G(y, z_2))}{d_z(z_1, z_2)} \Big] \tag{3.2}$$

where $d_z(\cdot, \cdot)$ and $d_I(\cdot, \cdot)$ are the measures used to compare the distance between elements in $\mathbb{Z}$ and $\mathbb{X}$ respectively.

As can be seen, the expression on the right-hand side of Equation 3.2 is proportional to the distance $d_I(G(y, z_1), G(y, z_2))$ between the images $G(y, z_1)$ and $G(y, z_2)$. Thus, as long as $d_I(\cdot, \cdot)$ reflects differences in the appearance of the generated images, $\mathcal{L}_{sd}$ will reward the generator for producing more diverse images.

Furthermore, as can be seen, the expression is also inversely proportional to the distance $d_z(z_1, z_2)$. At first, this might seem counter-intuitive, as it reduces the loss for larger distances $d_z(z_1, z_2)$, hence providing less incentive for z_1 and z_2 to map to different appearances in this case.

However, the main motivation for Equation 3.2 being on this form is that it reduces the tendency for mode collapse. When this occurs, the generator stops producing images with diverse appearances and instead only outputs a few limited variations, so called *modes* (see Chapter 2.2.4 for an explanation of mode collapse). According to [7, 52], latent vectors that are close in latent space are more likely to collapse to the same mode. Thus, by having $d_z(z_1, z_2)$ in the denominator, latent vectors z_1 and z_2 that are close in the latent domain are more encouraged to map to different appearances, hence avoiding the problem of mode collapse.

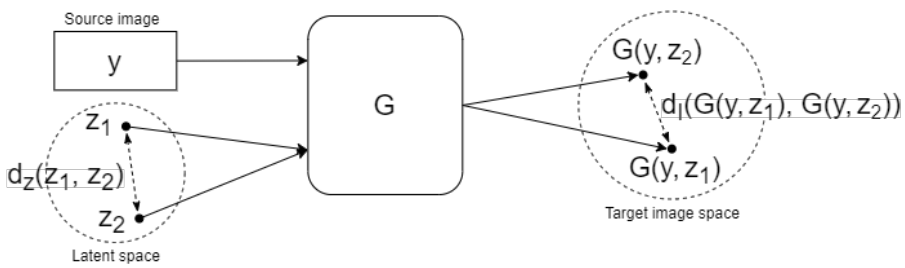
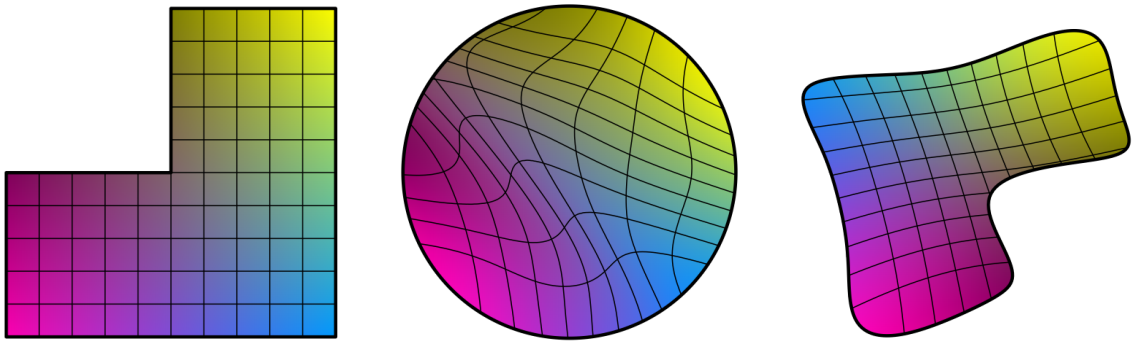


Figure 3.8: Distance measures in the latent space and images space used for calculating the style diversity loss $\mathcal{L}_{sd}$.



a) Example of distribution of features in a dataset.

b) Distribution of latent vectors $z \in \mathbb{Z}$ and mapping to dataset features (black lines).

c) Learned distribution of style vectors $s \in \mathbb{S}$ and mapping to dataset features (black lines).

Figure 3.9: Example of a curved mapping from the latent space $\mathbb{Z}$ to features in image space. a) shows an imaginary distribution of two types of features in a dataset, e.g. masculinity and degree of facial hair, with the empty region illustrating that bearded women are absent from the dataset. If latent vectors z are sampled randomly according to a symmetric distribution, as shown in b), the mapping to features in image space (black lines) must become curved in order to reproduce the real distribution. By mapping $\mathbb{Z}$ to an intermediate latent space $\mathbb{S}$, as shown in c), the curvature of the mapping (black lines) can be avoided [4].

3.2.2.3 Entangled latent space

Sampling the latent vectors z from a symmteric probability distribution $p(z)$, for example a multivariate Gaussian distribution, is a straightforward way to have the framework perform the multimodal image-to-image translation. However, a problem with using a symmetric distribution is that it can be difficult for the generator to learn a mapping from the latent space $\mathbb{Z}$ to features in the generated images if some combination of features are absent in the dataset it is trained on [4].

To illustrate this, imagine that the dataset that the framework is trained on only contains two types of features: masculinity and degree of facial hair. Assume further that some combination of the two features, say women with facial hair, is absent from the observations in the dataset. In this case, the distribution of features in the dataset will look similar to Figure 3.9 a).

Now, if the latent vectors z are sampled according to a multivariate Gaussian distribution, the distribution will be symmetric along the origin. Thus, in order for the generator to reproduce the same distribution of features as in the training dataset, the mapping $G(x, z)$ is forced to become curved, such that no part of the latent space $\mathbb{Z}$ maps to images of women with facial hair. This is illustrated in Figure 3.9 b).

However, a curved mapping from the latent space $\mathbb{Z}$ to features in the image domain is undesirable, as it will make linear separation of features unfeasible. When this

occurs, combinations of features will vary collectively and it will not be possible to identify each feature with a unique direction in the latent space $\mathbb{Z}$. The latent space $\mathbb{Z}$ is then said to be *entangled*.

3.2.2.4 Disentangling the latent space

To solve the problem of an entangled latent space, various methods can be used to disentangle it. One such method is to introduce a non-linear function L that maps the latent space $\mathbb{Z}$ to another intermediate latent space $\mathbb{S}$, as shown in Figure 3.9 [4, 1]. The latent vectors z can then be randomly sampled according to some simple symmetric distribution, say a multivariate Gaussian distribution, and be mapped to elements $s \in \mathbb{S}$. Since L is non-linear, it can map the latent space $\mathbb{Z}$ such that the intermediate latent space $\mathbb{S}$ accurately matches the distribution of features found in the dataset. This removes the need for the generator's mapping G to become curved and allows features to be linearly independent.

3.2.2.4.1 Style mapping network L The non-linear function L implemented as a regular fully connected, feed-forward neural network that takes latent vectors z as input and maps them to vectors s in the new latent space $\mathbb{S}$. The new network is called a *style mapping network*, and the vectors s in the new latent space $\mathbb{S}$ are called *style vectors*. The losses derived prior to this, such as the style diversity loss 3.2 and style reconstruction loss 3.1, can be used in the same way after simply replacing the latent vectors z by the style vectors s.

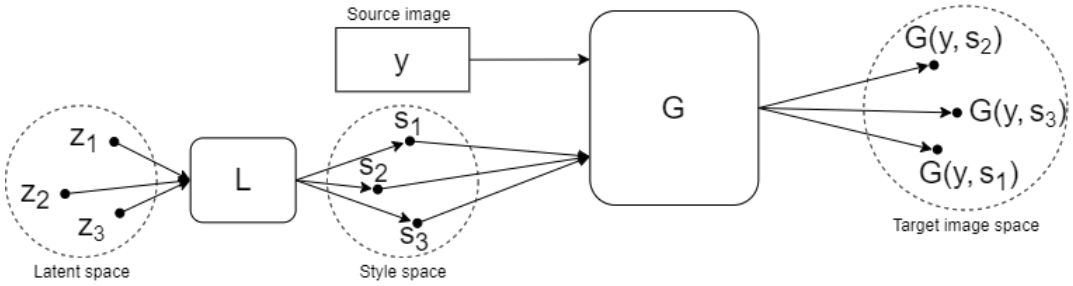


Figure 3.10: Latent vectors z replaced by the style vectors s outputted by the style mapping network L.

3.2.3 Reconstructing identities

The encoder E used for the latent reconstruction loss 3.1 enables the estimation of the latent vectors z used for generating images $G(y, z)$ and to encourage the generator to use the latent vectors z when producing images. However, while it is possible to also use E to estimate latent vectors of real images x, it is not explicitly trained to do this. It would however be desirable for the model to have the ability to extract the appearances of subjects in real images and use it to generate new images with the same appearance. Because of this, an additional supervised loss is proposed below in order to encourage the model to estimate accurate latent vectors

for real images as well.

Identity reconstruction loss $\mathcal{L}_{ir}$ The identity reconstruction loss was introduced to make the latent vector estimation and subsequent identity reconstruction by the generator work better for real images x. To do this, an additional step is added to each iteration during training. In addition to generating an image from a randomly sampled latent vector, the generator is also given a reconstructed latent vector $\hat{z} = E(x)$ and uses it to generate a reconstructed image $G(y, \hat{z})$. Now, the identity reconstruction is used to both encourage the encoder E to make accurate estimations $\hat{z}$ as well as the generator to produce a reconstructed image $G(y, \hat{z})$ similar to the real image x. It is given by

$$\mathcal{L}_{ir} = \min_{G,\,E} d_I(x, G(y, \hat{z})) \tag{3.3}$$

where $d_I(\cdot, \cdot)$ is some distance metric for the similarity in the image domain and $\hat{z} = E(x)$ is the estimated latent vector for the real image x.

3.2.4 Perceptual similarity metrics

Both the style diversity loss 3.2 and identity reconstruction loss 3.3 depend on having a distance metric $d_I(\cdot, \cdot)$ between images in the target image domain $\mathbb{X}$ that quantifies differences in their appearance. In particular, the metric should quantify differences in the appearances of the faces, independent of the head pose and perspective.

A simple way to implement such metric is to use a pixel-wise distance such as the L1 distance or Euclidian distance, but a problem is that these do not accurately capture high-level differences between the images. For example, two identical images offset from each other by one pixel would be measured as different by a pixel-wise distance, but would likely be seen as equivalent by a human. Instead, the distance metric $d_I(\cdot, \cdot)$ relevant for these purposes should quantify *perceptual* differences between the images – differences as perceived by a human.

Measuring human-perceived differences in images has proved to be a challenging computer vision problem, partly because they depend on high-order image structures but also because they are highly context-dependent [53]. For example, whether a red circle more similar to a red square or a blue circle is ambiguous. However, a recent success for measuring perceptual differences is to use the feature space learned by deep neural networks trained on image classification tasks. When such a network is trained on diverse image data, it typically learns to identify image features of abstraction levels; low-level features such as objects or categories in the earlier layers and high-level features such as edges and colors in the later layers [54]. By passing a pair of images through such a network and averaging the differences in the internal activations that they give rise to, a measure of their similarity that correlates well with human perception can be retrieved. Metrics of this type are called *perceptual similarity metrics*.

3.2.4.1 Perceptual loss

Since perceptual similarity metrics capture high-level differences between images, they can also be used to compare the similarity of the generated and real images during GAN training. This is in fact one of the original motivations behind the development of perceptual distance measures and has been demonstrated to lead to increased image quality when used in combination with the adversarial loss functions [55, 56, 57]. When implemented like a loss function, it is common to call this a *perceptual loss*.

Perceptual losses should not, however, be used as the sole loss function for training CGAN frameworks. This is because the internal feature representations in a classifier network typically are contractive in nature, meaning that many images get mapped to the same feature vector. Thus, for every realistic looking image, there are multiple non-realistic images that map to the same feature vector. Optimizing for similarity in the feature space therefore leads to high frequency artifacts. By combining a perceptual loss with an adversarial loss that encourages realistic images, the best quality is therefore achieved [55].

If f_l are the activations of layer l when the images x_1 and x_2 are passed through the network, the perceptual loss is given by

$$\mathcal{L}_p = \sum_{i}^{\text{layers}} ||f_l(x_1) - f_l(x_2)||_2^2 \tag{3.4}$$

3.2.4.2 Perceptual networks

It is common to use classification networks pretrained on large-scale image classification tasks for perceptual similarity metrics (and perceptual losses). Additionally, the feature space of the discriminator can also be used for this purpose which has been shown to be beneficial for stability and convergence during training when used as a perceptual loss [58]. Both variants were tried for the framework of this thesis. The discriminator is presented in Chapter 3.4.1 and the tried classification networks are presented below.

3.2.4.2.1 VGG-19 classification network A commonly used perceptual network is the VGG-19 [59], a 19 layer deep convolutional neural network, trained on the ImageNet dataset [22]. Due to its depth, the network's internal feature maps represent image features in a wide range of abstraction levels. When using VGG-19 for the perceptual loss, it is common to use a selection of 5 internal convolutional layers normalized by their channel sizes [60, 54]. Assuming the layers are indexed by 0-18, this implementation will use the L1 distance between the layers $1, 6, 11, 20, 29$, as has been done previously by [61].

3.2.4.2.2 ResNet-50 facial identification network Another network, a ResNet-50 [37] trained for facial identification, will also be tried as a perceptual network for the framework. This is a 50 layer deep residual neural network trained to identify

the faces of the 9131 unique subjects in the 3.31 million images large VGGFace2 dataset [62]. In this implementation, the L1 distance between the 5 feature vectors outputted from each residual block in the network are used (see architecture in [37]).

3.2.5 Full loss function

The loss functions that have been presented and that were tried for the framework are summarized in Table 3.2. These were combined to form the full loss function $\mathcal{L}_{\text{full}}$ accordingly

$$\mathcal{L}_{\text{full}} = \mathcal{L}_{\text{adv}} + \lambda_p \mathcal{L}_p + \lambda_{lr} \mathcal{L}_{lr} + \lambda_{ir} \mathcal{L}_{ir} + \lambda_{sd} \mathcal{L}_{sd} \,. \tag{3.5}$$

Name	Symbol	Equation
Adversarial loss	$\mathcal{L}_{\text{adv}}$	2.6
Perceptual loss	$\mathcal{L}_p$	3.4
Latent reconstruction loss	$\mathcal{L}_{lr}$	3.1
Style diversity loss	$\mathcal{L}_{sd}$	3.2
Identity reconstruction loss	$\mathcal{L}_{ir}$	3.3

Table 3.2: Summary of the loss functions tried for the framework.

3.2.6 Training the framework

The generator and discriminator are trained using alternating gradient descent, i.e. one network is held constant while the other is updated according to gradient descent on the full loss function 3.5.

3.2.6.1 Network optimization

During training, weight updates of the networks were performed according to the Adam optimization scheme [63], as done by many other similar studies [4, 1, 47, 29]. Each of the four networks had separate Adam optimizers and unique learning rates. Out of these, the learning rate of the style mapping network L was generally set to a significantly lower value compared to those of the other networks. This was shown by [4, 1] to be beneficial for convergence as it makes the changes to the style mapping slow which allows the other networks to stay adjusted to it.

At the beginning of training, the learning rates were set to an initial value and were held constant for a set number of epochs K_{const}. As done by [61], the learning rates then decay linearly down to zero over K_{decay} epochs. Thus, the CGAN is trained for a total of $K_{\text{total}} = K_{\text{fixed}} + K_{\text{decay}}$ epochs, whose specific values are hyperparameters.

3.2.6.2 Training algorithm

The full algorithm for training the framework is shown in Algorithm 2.

Algorithm 2: Algorithm for training the CGAN framework.

for *each training iteration* **do**
  Sample pairs of source and target images $\{(y_1, x_1), ..., (y_N, x_N)\}$;
  Sample latent vectors $\{z_1, ..., z_N\}$;
  Generate fake images $\{G(z_1, y_1), ..., G(z_N, y_N)\}$;
  Compute discriminator losses $\{\mathcal{L}_D^{(\text{target})}, \mathcal{L}_D^{(\text{fake})}\}$;
  Backpropagate discriminator losses;
  } Discriminator
  Sample latent vectors $\{z_1, ..., z_N\}$;
  Generate fake images $\{G(z_1, y_1), ..., G(z_N, y_N)\}$;
  if *using latent reconstruction loss* **then**
    Estimate latent vectors for generated images $\{E(G(z_1, y_1)), ..., E(G(z_N, y_N))\}$;
  end
  if *using style diversity loss* **then**
    Sample additional latent vectors $\{z_1^*, ..., z_N^*\}$;
    Generate fake images $\{G(z_1^*, y_1), ..., G(z_N^*, y_N)\}$;
  end
  if *using identity reconstruction loss* **then**
    Estimate latent vectors for real images $\{E(x_1), ..., E(x_N)\}$;
    Generate fake images $\{G(E(x_1), y_1), ..., G(E(x_N), y_N)\}$;
  end
  Compute generator losses $\{\mathcal{L}_G, \mathcal{L}_p, \mathcal{L}_{lr}, \mathcal{L}_{sd}, \mathcal{L}_{ir}\}$;
  Backpropagate generator losses;
  } Generator
end

Note that if the style mapping network L described in Chapter 3.2.2.4.1 is used, the latent vectors z are replaced by the style vectors $s = L(z)$.

3.3 Evaluating the framework

Since GANs use an unsupervised learning approach, there is no direct way of comparing the performance of different frameworks. One approach is to simply use the perceived image quality from human annotators, but this is not a very robust metric since human judgement can vary significantly depending on the setup of the task.

Because of this, there has been an effort to develop standardized metrics that capture the quality of the images produced by GANs and other generative models. Today, the most common method is to use metrics based on perceptual similarity, as was

described in Chapter 3.2.4, since these have been found to best correlate with human-perceived quality. Thus, two such metrics were used to evaluate the framework of this thesis and are described below.

3.3.0.1 Fréchet Inception Distance

The Fréchet Inception Distance, or FID for short, is a metric for the realism of the generated images. In particular, it compares how similar a set of generated images is to a set of real images. FID has been shown to correlate well with human-perceived image quality and to capture many different types of disturbances [64], as shown in Figure 3.11. It has therefore become on of the de-facto ways of comparing the performance of generative models and was chosen as the metric for evaluating the realism of the images generated by the framework described above.

The FID is a type of perceptual similarity metric and is typically evaluated by using the feature space learned by the Inception-V3 [65] network trained on the ImageNet dataset [22]. However, instead of directly comparing the activations per layer for each image pair, the FID is based on comparing the activation statistics of the two image sets as a whole. More specifically, it is calculated by collecting the desired activations from Inception-V3 in an activation vector and forming the mean μ and covariance matrix C for the two image sets that are compared.

With μ_g, C_g and μ_r, C_r being the means and covariance matrices of the activations inside the classification network for the generated and real images respectively, the FID is given by [64]

$$\text{FID} = ||\mu_r - \mu_g||_2^2 + \text{Trace}\left(C_r + C_g - 2(C_r C_g)^{1/2}\right) . \tag{3.6}$$

Thus, a low FID score corresponds to a high similarity between the generated and real images and is therefore an indication that the CGAN framework has learned to accurately model the real data distribution.

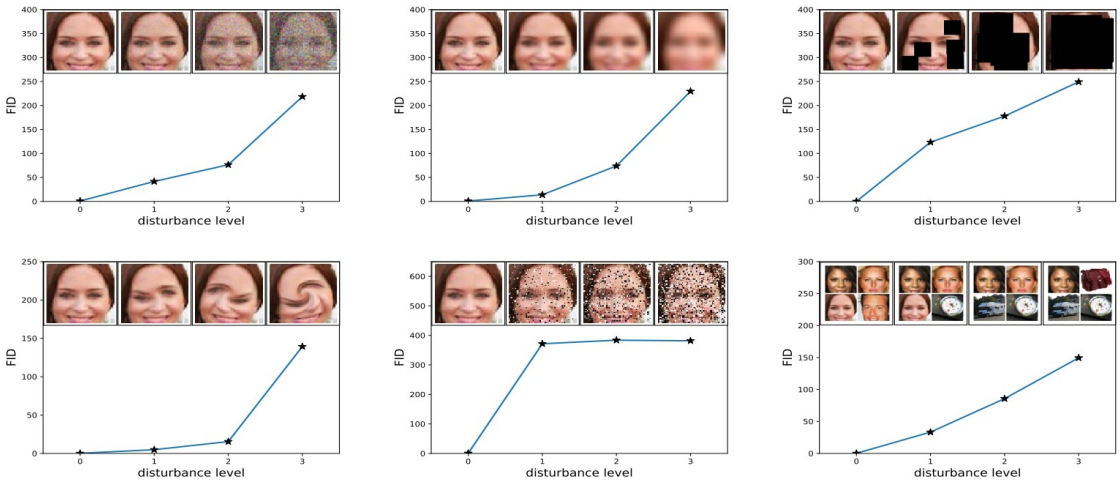


Figure 3.11: FID scores for images with varying amounts of different visual disturbances [64].

3.3.0.2 Learned Perceptual Image Patch Similarity

While FID is used to evaluate how similar a set of generated images as a whole is to a set of real images by comparing the activation statistics they give rise to in the Inception V3 network, it is not typically used as a metric for comparing the similarity of separate image pairs. Another metric that can be used to do this is the LPIPS metric, or *Learned Perceptual Image Patch Similarity* [53]. It is another perceptual similarity metric, thus also based on the feature space of a classification network, and has been shown to accurately capture the perceptual similarity of image pairs [53]. For LPIPS, it is common to use the feature space of the AlexNet [22] network trained on the ImageNet dataset [22].

The LPIPS score for an image pair is calculated by passing each image through the classification network and extracting the activation tensors $f^l(x) \in \mathbb{R}^{H_l \times W_l \times C_l}$ for all L layers. The activations of each layer are then unit-normalized and scaled by a weight vector in the channel dimension. Lastly, the L2 distance is calculated between the resulting tensors, and the resulting difference tensor is spatially averaged and summed over all layers.

Let $f^l_{hw}(x) \in \mathbb{R}^{H_l \times W_l \times C_l}$ denote the activations in layer l for input x that have been unit-normalized in the channel dimension and w_l the weight vector for layer l. $\odot$ denotes element-wise multiplication. The LPIPS score for two images x_1 and x_2 is given by

$$\text{LPIPS}(x_1, x_2) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} ||w_l \odot \left(f^l_{hw}(x_1) - f^l_{hw}(x_2)\right)||_2^2 \tag{3.7}$$

3.3.1 Evaluation procedures

The FID and LPIPS metrics were used in the procedures presented below to measure the realism and diversity of the generated images.

3.3.1.1 Realism evaluation

In order to evaluate the realism of the generated images, the FID was calculated between the real images of the test dataset and the generated images produced using each corresponding test source image. More specifically, for each source image y_i in the test set, a latent vector z_i was randomly sampled and used by the generator to produce the generated image $G_i = G(z_i, y_i)$. The realism evaluation metric was then formed by calculating the FID between the two sets of generated and real images.

3.3.1.2 Diversity evaluation

In addition to producing realistic images, the CGAN should be capable of generating a wide range of appearances for each source image y by sampling different

latent vectors z. However, since the FID evaluation scheme described above only requires the CGAN to produce a single generated image per source image, it does not accurately capture the diversity of the generated images. Therefore, an additional scheme previously employed by StarGAN v2 [1] was used to evaluate the diversity of the images generated by the CGAN.

First, a set of N source images $\{y^{(1)}, y^{(2)}, ..., y^{(N)}\}$ was randomly sampled from the test dataset. Furthermore, for each source image $y^{(i)}$, a set of M latent vectors $\{z_1^{(i)}, z_2^{(i)}, ..., z_M^{(i)}\}$ was also randomly sampled according to the distribution $p(z)$. Each $y^{(i)}$ and set $\{z_m\}_{m=1}^{M}{}^{(i)}$ were then used by the generator to produce a set of M generated images $\{G(z_1^{(i)}, y^{(i)}), G(z_2^{(i)}, y^{(i)}), ..., G(z_M^{(i)}, y^{(i)})\}$.

The diversity of each such set was then evaluated by calculating the LPIPS score between all possible pairs of generated images. In other words, if $z_s^{(i)}, z_t^{(i)}$ denotes two different latent vectors sampled for the source image $y^{(i)}$, the quantity

$$L_i = \frac{1}{\binom{M}{2}} \sum_{s=1}^{M-1} \sum_{t=s+1}^{M} \text{LPIPS}(G(z_s^{(i)}, y^{(i)}), G(z_t^{(i)}, y^{(i)})) \tag{3.8}$$

was used to evaluate their diversity.

The full diversity evaluation metric was then given by averaging the N diversity scores L_i

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i \,.$$

Since diverse generated images result in high pairwise LPIPS scores in Equation 3.8, a high L score for the CGAN framework indicates a more diverse output. For simplicity, the rest of the thesis will refer to the quantity L simply as "LPIPS".

3.4 Network architectures

This chapter describes the architectures used for the different networks that make up the full CGAN framework.

3.4.1 Discriminator

The discriminator used in the framework is called a *PatchGAN* [2]. It is a type of classifier that takes images as input and makes predictions by passing them through a series of convolutional layers. However, instead of making a single prediction for the image as a whole as being real or fake, like a standard discriminator, a PatchGAN splits the input image into $P \times P$ *patches* and makes a prediction for each. The output is then given by the average of these predictions.

The image patch belonging to each prediction is called its *receptive field* [2]. The

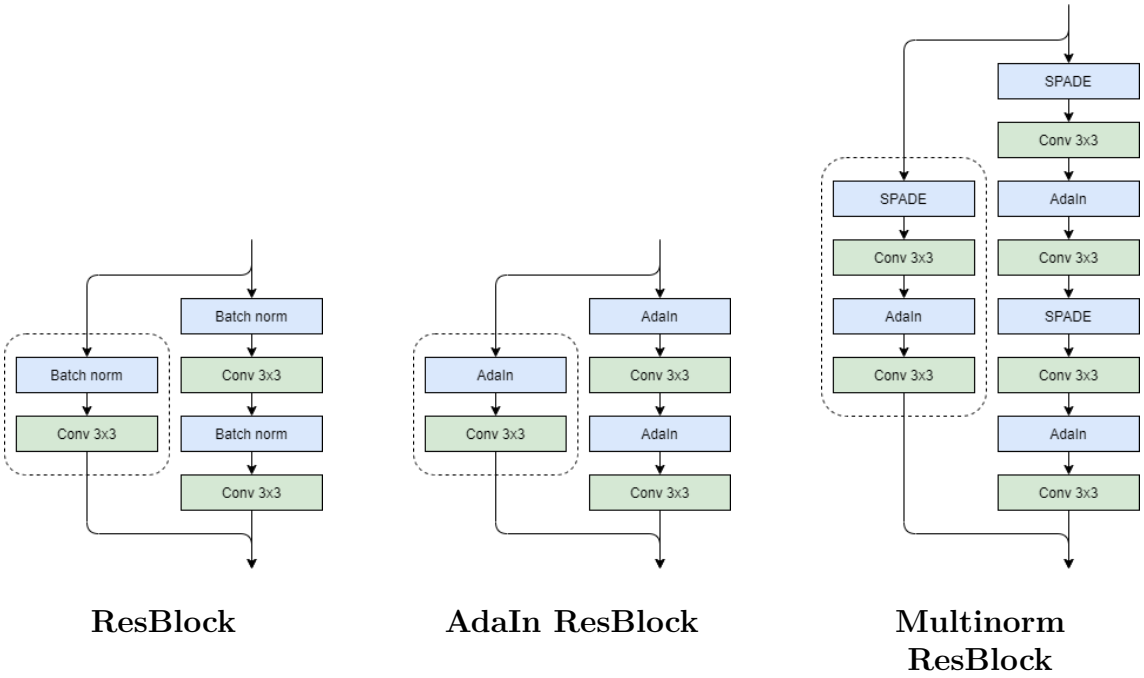


Figure 3.12: Residual blocks (ResBlocks) used in the network architectures, inspired by [1, 8, 66]. The dashed boxes denote learned skip connections that are used when the in- and output channel sizes are different.

receptive field size can be determined by recursively applying the following relationship between the input and output size of each convolutional layer

$$\text{receptive field size} = (\text{output size} - 1) \cdot S + K\,.$$

where S is the convolutional stride and K is the kernel size (filter size) [2].

The original study [2] showed that a PatchGAN can be trained effectively even with very small patch sizes. Since smaller patch sizes correspond to a shallower network, this means that performance remains high even with very few parameters. This is beneficial as it makes overfitting less likely, especially for small datasets where larger networks easily overfit. This is one reason it was adopted for the CGAN framework of this thesis.

Furthermore, by analyzing image patches instead of the image as a whole, PatchGANs can be understood as modelling images as Markov random fields, assuming that pixels separated by more than a patch diameter are independent [2]. This is also a common assumption in style and texture models, so the output of a PatchGAN can be seen as a type of texture or style loss. This makes it an appropriate discriminator for image-to-image translation, which is another reason for adopting it for the framework.

The details of the PatchGAN architecture used in the framework was inspired by [61] and is shown in Table 3.3. As can be seen, it makes a prediction for a grid

Table 3.3: PatchGAN discriminator network architecture.

Layer	Resample	Normalization	Activation	Output shape
Source & target/generated	-	-	-	128 ×128 × 4
Conv	-	-	LeakyReLU	65 ×65 × 32
Conv	-	Instance	LeakyReLU	33 ×33 × 64
Conv	-	Instance	LeakyReLU	17 ×17 × 128
Conv	-	Instance	LeakyReLU	9 ×9 × 256
Conv	-	Instance	LeakyReLU	10 ×10 × 512
Conv	-	-	-	11 ×11 × 1

of 11×11 image patches with a size of 12×12 pixels each. The predictions are computed by passing the input through a series of instance normalization layers, each followed by convolutional layers with so called LeakyReLU [67] activation functions. Since it should make predictions conditioned on source images, each pair of source image and generated/real image are stacked channel-wise before being given to the discriminator. This gives a total channel size of $3+1=4$ for the RGB source image and grayscale target/generated image.

3.4.2 Generator

Two different architectures were tried for the generator and are presented below.

Encoder-decoder This is an encoder-decoder architecture (see Chapter 2.4.3.3) inspired by the StarGAN v2 [1] generator. The encoder is built using regular ResBlocks followed by average pooling layers (AvgPool) for downsampling and the decoder is built using AdaIn Resblocks followed by bilinear interpolation layers for upscaling. The adaptive instance normalization (AdaIn) layers inside the AdaIn ResBlocks are used to introduce the latent vectors z (or style vectors s) into the network. Schematic diagrams of both ResBlocks are shown in Figure 3.12. Furthermore, an illustration of the generator is shown in Figure 3.13 and the details of the architecture are presented in Table 3.4.

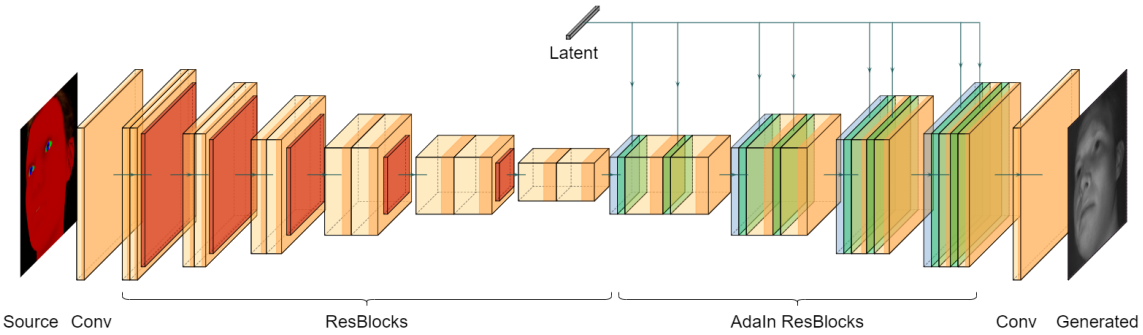


Figure 3.13: Schematic diagram of Encoder-decoder generator architecture.

Table 3.4: Encoder-decoder generator network architecture.

Layer	Resample	Normalization	Activation	Output shape
Input image	-	-	-	128 ×128 × 3
Conv	-	-	ReLU	128 ×128 × 16
ResBlock	AvgPool	Batch norm	-	64 ×64 × 32
ResBlock	AvgPool	Batch norm	-	32 ×32 × 64
ResBlock	AvgPool	Batch norm	-	16 ×16 × 128
ResBlock	AvgPool	Batch norm	-	8 ×8 × 256
ResBlock	AvgPool	Batch norm	-	4 ×4 × 256
ResBlock	Interpolate up	AdaIn	-	8 ×8 × 256
ResBlock	Interpolate up	AdaIn	-	16 ×16 × 128
ResBlock	Interpolate up	AdaIn	-	32 ×32 × 64
ResBlock	Interpolate up	AdaIn	-	64 ×64 × 32
ResBlock	Interpolate up	AdaIn	LeakyReLU	128 ×128 × 16
Conv	-	-	Tanh	128 ×128 × 1

Multinorm decoder This architecture is a modified version of the SPADE [8] decoder (see Chapter 2.4.3.4). Instead of taking random noise as input like the original architecture, it takes a learned constant input tensor, as is done in the generator used in StyleGAN [4]. In addition to this, the generator uses a modified type of ResBlock compared to those described in Chapter 2.4.3.4, called *Multinorm* ResBlocks. Figure 3.12 shows a schematic diagram of these. As can be seen, the Multinorm ResBlocks uses both SPADE layers to inject the source images y and AdaIn layers to inject the latent vectors z (or style vectors s), with convolutional and bilinear interpolation layers in between. Figure 3.14 shows an illustration of the generator and Table 3.5 presents the architecture's details.

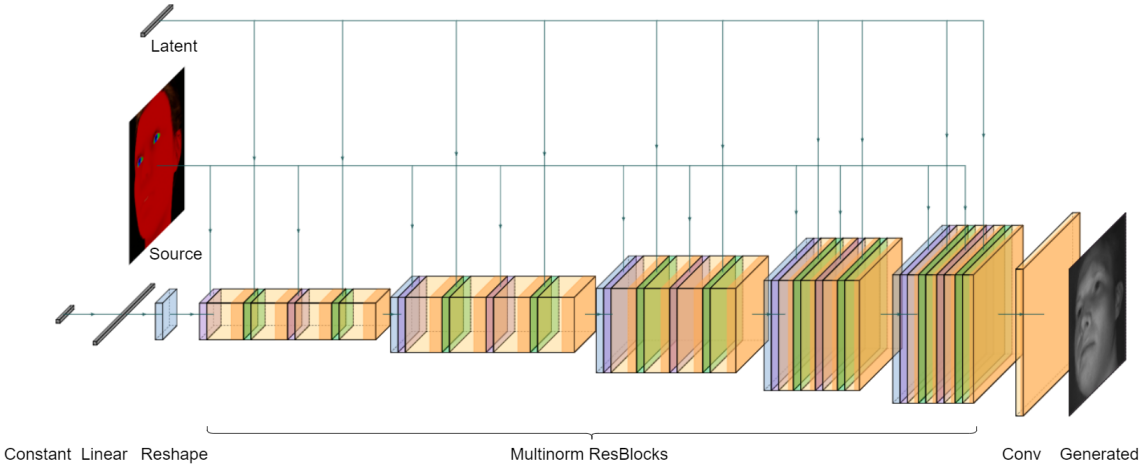


Figure 3.14: Schematic diagram of Multinorm decoder generator architecture.

Table 3.5: Multinorm decoder generator network architecture.

Layer	Resample	Normalization	Activation	Output shape
Latent noise	-	-	-	1 ×1 × 512
Linear	-	-	ReLU	1 ×1 × 4096
Reshape	-	-	ReLU	4 ×4 × 256
Multinorm Resblock	Interpolate up	SPADE & AdaIn	-	8 ×8 × 256
Multinorm Resblock	Interpolate up	SPADE & AdaIn	-	16 ×16 × 128
Multinorm Resblock	Interpolate up	SPADE & AdaIn	-	32 ×32 × 64
Multinorm Resblock	Interpolate up	SPADE & AdaIn	-	64 ×64 × 32
Multinorm Resblock	Interpolate up	SPADE & AdaIn	LeakyReLU	128 ×128 × 16
Conv	-	-	Tanh	128 ×128 × 1

3.4.3 Latent encoder network E

Also inspired by the StarGAN v2 [1] study, the latent encoder E is built using ResBlocks followed by average pooling (AvgPool) layers for downsampling, as well as convolutional layer in the beginning and end of the network. The details of the architecture are shown in Table 3.6.

Table 3.6: Latent encoder network architecture.

Layer	Resample	Normalization	Activation	Output shape
Latent noise	-	-	-	1 ×1 × 512
Conv	-	-	-	128 ×128 × 32
ResBlock	AvgPool	-	-	64 ×64 × 64
ResBlock	AvgPool	-	-	32 ×32 × 128
ResBlock	AvgPool	-	-	16 ×16 × 256
ResBlock	AvgPool	-	-	8 ×8 × 512
ResBlock	AvgPool	-	-	4 ×4 × 512
ResBlock	AvgPool	-	LeakyReLU	2 ×2 × 512
Conv	-	-	LeakyReLU	1 ×1 × 512

3.4.4 Style mapping network

The style mapping network L was implemented as a fully connected feed-forward neural network, as this has been shown to be sufficient by previous studies [4, 1]. Specifically, the implementation used for L had 8 hidden fully connected (linear) layers with ReLU activation functions for all layers except the last. The details of the network are shown in Table 3.7.

Layer	Activation	Output shape
Latent noise	-	$1 \times 1 \times 512$
Linear	ReLU	$1 \times 1 \times 512$
Linear	ReLU	$1 \times 1 \times 512$
Linear	ReLU	$1 \times 1 \times 512$
Linear	ReLU	$1 \times 1 \times 512$
Linear	ReLU	$1 \times 1 \times 512$
Linear	ReLU	$1 \times 1 \times 512$
Linear	ReLU	$1 \times 1 \times 512$
Linear	-	$1 \times 1 \times 512$

Table 3.7: Style mapping network architecture.

3.5 Experiments

In this chapter, the experiments that were performed to evaluate the performance of the framework are presented. These were divided into two sections, one for evaluating different configurations and one for analyzing a selection of these in more detail.

Parameter name	Symbol	Value
Epochs total	K_{total}	20
Epochs fixed	K_{fixed}	10
Epochs decay	K_{decay}	10
Batch size	N	12
Weight initialization		Xavier [68]
Optimizer		Adam, $\beta_1 = 0.5$, $\beta_2 = 0.999$ [63]
G learning rate	l_G	$1.5 \cdot 10^{-4}$
D learning rate	l_D	$4.5 \cdot 10^{-4}$
E learning rate	l_E	$1.5 \cdot 10^{-4}$
L learning rate	l_L	$4.5 \cdot 10^{-6}$
Latent space dimension	$\text{Dim}(\mathbb{Z})$	512
Latent distribution	$p(z)$	$\mathcal{N}(0, 1)$
Style space dimension	$\text{Dim}(\mathbb{S})$	512

Table 3.8: Default hyperparameter settings for the CGAN framework during all experiments.

3.5.1 Default model settings

A number of experiments were performed in order to find the best performing model configuration of the above framework. While each experiment describes the specific component that is tested in more detail, all of the models share a number of default parameter settings. These are presented in Table 3.8.

3.5.2 Framework configuration

The experiments described in this part aim to evaluate the performance of different configurations of the framework and find the best performing configuration in terms of realism and diversity. This will be done by starting with a simple version of the framework and successively adding more components and evaluating variations of each. However, to avoid having to test all possible configurations, the best performing configuration in each experiment will in general be used as the default for the following experiments.

The experiment was divided into the following parts, where the realism and diversity were evaluated by the FID and LPIPS scores as well as perceived visual quality:

1. **Generator architecture** The two generator architectures presented in Chapter 3.4.2 are compared after training for 20 epochs with least square adversarial loss.

2. **Adversarial loss** Adversarial loss of the types 1) original, 2) least square and 3) hinge are evaluated for the framework using the encoder-decoder generator after 20 epochs of training.

3. **Perceptual loss** Perceptual loss (Chapter 3.2.4.1) based on the feature spaces of the 1) Discriminator, 2) VGG-19 pretrained on ImageNet and 3) ResNet-50 pretrained on VGGFace2 are combined with adversarial loss and evaluated for the framework using the encoder-decoder generator after 20 epochs of training.

4. **Diversity regularization** The two diversity regularization losses of Chapter 3.2.2.2 are evaluated for the framework using the encoder-decoder generator after 20 epochs of training.

   (a) **Latent reconstruction loss and encoder E** The latent reconstruction loss and encoder E is added to the framework and evaluated in combination with adversarial loss.

   (b) **Style diversity loss** The style diversity loss is added to the above configuration and evaluated.

5. **Identity reconstruction loss** The identity reconstruction loss is evaluated in combination with the latent reconstruction loss and adversarial loss using the encoder-decoder generator after 20 epochs of training.

6. **Style mapping network L** The style mapping network L is added to the framework and evaluated using a selection of the above configurations trained for 20 epochs.

3.5.3 Framework analysis

Once the various model configurations described above had been evaluated, this part selected the best performing configuration and investigated these in more detail. First, the framework's ability to preserve domain-invariant attributes such as head pose and gaze direction were evaluated. Next, the latent space of the framework was examined for signs of entanglement or mode collapse.

3.5.3.1 Attribute preservation

An important property of the framework is its ability to preserve domain-invariant attributes, in particular head position and gaze direction. To evaluate how well it learned to do this, sequences of source images where a single attribute of the 3D model was varied smoothly were given to the framework. If the framework had successfully learned to preserve each attribute, the faces in the resulting generated images should follow the movement of the 3D model in the source images perfectly. Sequences varying the following attributes were tried:

1. Head pose, horizontally
2. Head pose, vertically
3. Gaze direction, horizontally

3.5.3.2 Latent space exploration

This part investigated the learned latent space $\mathbb{Z}$ in more detail. This was done both to demonstrate the capabilities that a latent space manipulation provides but also to troubleshoot it for common problems such as entanglement or overfitting. Two types of latent space manipulation techniques were performed and are presented below.

3.5.3.2.1 Latent space interpolation Given that the model is trained successfully, each latent vector z will map to a unique facial appearance in the image domain. Ideally, the model has also learned to separate the facial features into distinct directions in the latent space, such that each feature can be varied independently of others – a disentangled latent space. If this is not the case and the latent space is entangled, multiple features of the generated images will be dependent on each other and vary collectively.

In the case of a disentangled latent space, it should be possible to smoothly transition between any two faces by interpolating between their corresponding latent vectors and generating images for the intermediate latent vectors. To explore this

concept, latent space interpolation was done for a selection of generated as well as reconstructed faces. As long as the latent space was disentangled, the transition from every intermediate latent vector to every other should have been smooth. If features did not vary smoothly or if noticeably distinctive-looking faces appeared during the interpolation, it was interpreted as a sign of an entangled latent space.

3.5.3.2.2 Latent space arithmetic Another way to investigate model's latent space is through so called latent space arithmetic. This is when latent vectors z (or style vectors s) are manipulated using basic arithmetic operations in order to alter and control the features of the corresponding generated images. Analyzing how such operations affect the features in the generated images can help build an understanding of how well the framework has learned to encode high-level image features into the lower-dimensional latent space. Furthermore, if specific features can be manipulated independently of others through latent space arithmetic, this further indicates that the latent space is disentangled.

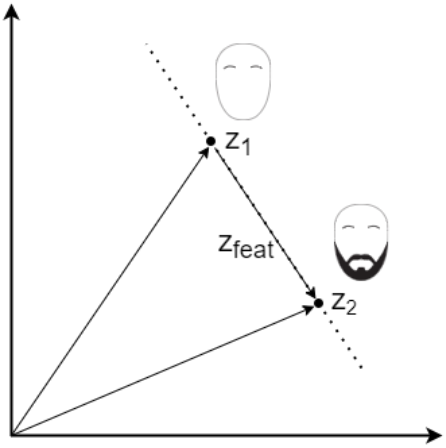


Figure 3.15: Example of two latent vectors z_1 and z_2 representing a face without and with a beard respectively. In this case, the difference between the two vectors z_{feat} corresponds to the beard feature.

Figure 3.15 shows an example of two latent vectors z_1 and z_2 representing a face without a beard and a face with a beard respectively. Suppose z_1 and z_2 are found through visual inspection of their resulting faces in the image domain. Now, if features in the latent space are not entangled, the latent vector representing a beard should correspond to z_{feat}, also shown in Figure 3.15, which is retrievable through simple subtraction, i.e. by

$$z_{\text{feat}} = z_2 - z_1 \,.$$

By isolating the z_{feat} of this example in this way, it can be used to attach a beard to other generated faces by simply adding it to their corresponding latent vectors. Since z_{feat} only represents the single facial feature, in this case a beard, the resulting generated faces should only change with respect to this feature while remaining otherwise the same. Figure 3.15 shows an illustration of of this.

This experiment consisted of identifying generated images that differed on such single facial features, but were otherwise similar. The images were found through simple visual inspection of a collection of randomly generated samples. Once suitable image pairs had been found for a given facial feature, their latent vectors were retrieved and used to derive z_{feat} corresponding to the difference in the specific feature. To investigate how well specific features could be extracted in this way, z_{feat} was then added to the latent vectors of other generated faces, and the resulting latent vectors were then used to generate new faces. This was done for a number of different facial features, again that were all identified through visual inspection.

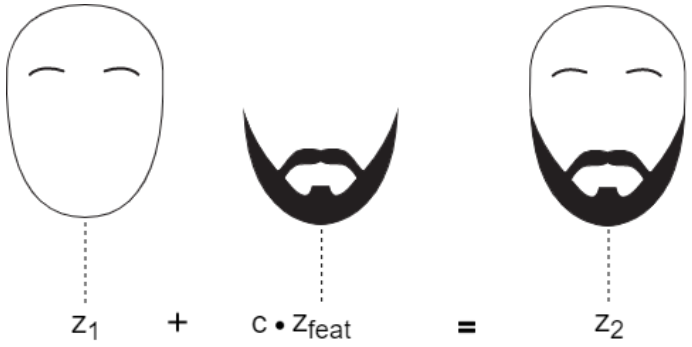


Figure 3.16: An example of latent space arithmetic. In this case, the latent vector z_{feat} corresponds to a beard feature in the image domain. By adding z_{feat} to z_1, representing a face without a beard, the resulting latent vector z_2 will represent the same face but with a beard added.

4

Results

4.1 Framework configurations

In the following chapter, the realism and diversity of various configurations of the framework were evaluated using the FID and LPIPS scores. Furthermore, to allow for visual inspection, samples were generated by each configuration using the source image shown in Figure 4.1 and randomly sampled latent vectors z (or style vectors s).

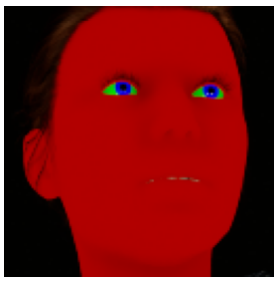


Figure 4.1: Source image used to generate the samples shown in Chapter 4.1.

To make reference easier, each configuration was given a unique name on the form `Generator-Loss1-Loss2-Loss3` etc., using the abbreviations shown in Table 4.1.

Abbreviation	Meaning
ED	Encoder-decoder generator
Mult	Multinorm generator
O	Original adversarial loss
H	Hinge adversarial loss
LS	Least square adversarial loss
LR	Latent reconstruction loss
SD	Style diversity loss
IR	Identity reconstruction loss
SM	Style mapping network

Table 4.1: Abbreviations used for the configuration names in Chapter 4.1.

4.1.1 Generator architecture

The first experiment investigated the two generator architectures described in Chapter 3.4.2. The resulting FID and LPIPS scores are shown in Table 4.2 and random samples generated using a fixed source image are shown in Figure 4.2.

As can be seen, the encoder-decoder generator had both the lowest FID score and the highest LPIPS score, indicating better realism and greater diversity compared to the multinorm generator. Inspection of the samples shown in Figure 4.2 also appears to confirm this. Because of this, the encoder-decoder is chosen as the default generator architecture to use for the remaining experiments.

Ref.	Configuration	Generator	FID	LPIPS
A	ED-LS	Encoder-decoder	**14.75**	**0.180**
B	Mult-LS	Multinorm	18.75	0.166

Table 4.2: FID and LPIPS scores for the framework using each generator architecture.

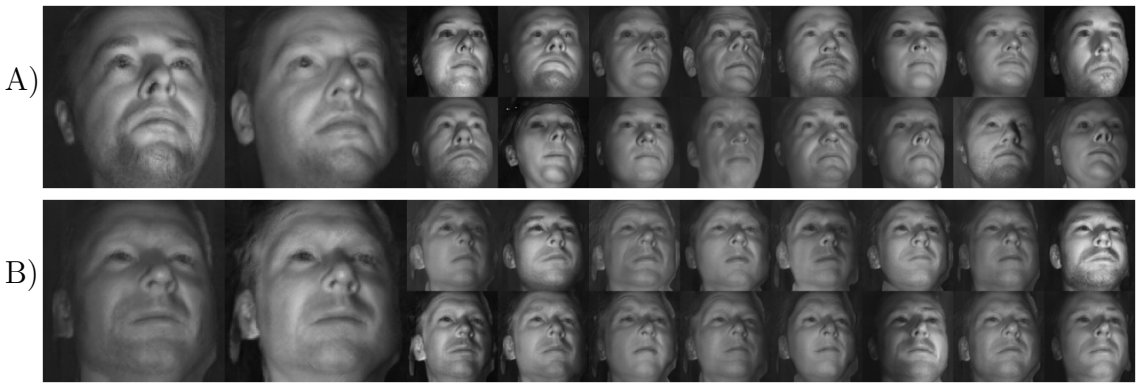


Figure 4.2: Random samples of generated images for the framework using each of the generators presented in Table 4.2.

4.1.2 Adversarial loss

The next experiment compared the quality of the generated images with different adversarial losses $\mathcal{L}_{\text{adv}}$ for the CGAN framework using the encoder-decoder generator. Table 4.3 shows the configurations that were tried and the resulting FID and LPIPS scores of each. Furthermore, Figure 4.3 shows random samples generated by each of the resulting frameworks using a fixed source image.

As can be seen from both the resulting scores and the random samples, the adversarial loss function plays an important role for the quality and diversity of the generated images. While the original and the least square losses yielded similar FID scores, the hinge loss had a significantly higher FID and lower realism.

At first glance, the hinge loss samples in Figure 4.3 do not seem to be compatible with an FID score this high. However, by using randomly sampled source images to generate images, it is revealed that the image generation fails completely for source images where the 3D model has glasses which explains the high score. Figure 4.4 shows an example of this.

Ref.	Configuration	Adv. loss	FID	LPIPS
A	ED-O	Original	16.5	0.148
B	ED-H	Hinge	78.5	0.0592
C	ED-LS	Least square	**14.7**	**0.180**

Table 4.3: FID and LPIPS scores for the framework using each of the listed adversarial losses.

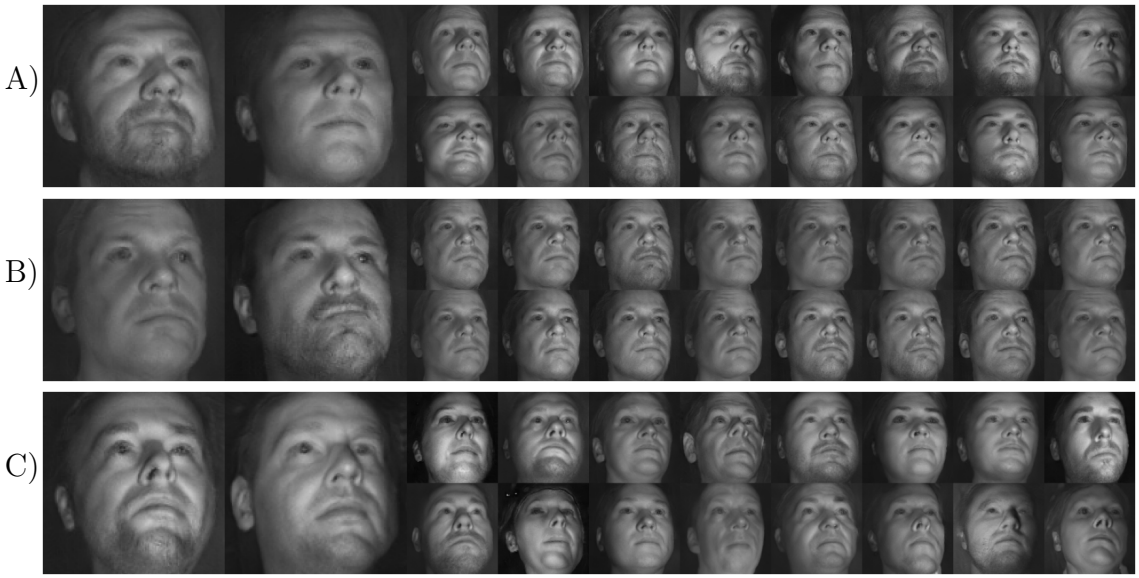


Figure 4.3: Random samples of generated images for the configurations presented in Table 4.3.

4.1.3 Perceptual loss

Next, the addition of perceptual loss $\mathcal{L}_p$ investigated. The tests primarily used least square adversarial loss since it was the best performing in the previous experiment, but hinge loss was also tried in order to investigate if the instability problem shown in Figure 4.4 could be remedied. Table 4.4 shows the investigated configurations and the resulting FID and LPIPS scores of each, and Figure 4.5 shows samples generated using a fixed source image. Additionally, perceptual loss using the ResNet-50 FaceId network was also tried but resulted in significant overfitting, so the results were not included.

Figure 4.4: Generated images for randomly sampled source images for configuration B in Table 4.3.

Ref.	Configuration	Adv. loss	Network	Layers	λ_p	FID	LPIPS
A	ED-H-P	Hinge	Discriminator	4 & 5	1	18.6	0.134
B	ED-LS-P-1	Least square	Discriminator	4 & 5	1	18.3	**0.151**
C	ED-LS-P-2	Least square	Discriminator	All	1	22.5	0.133
D	ED-LS-P-3	Least square	VGG-19 ImageNet	All	1	**15.3**	0.138

Table 4.4: FID and LPIPS scores for various configurations of the framework using perceptual loss.

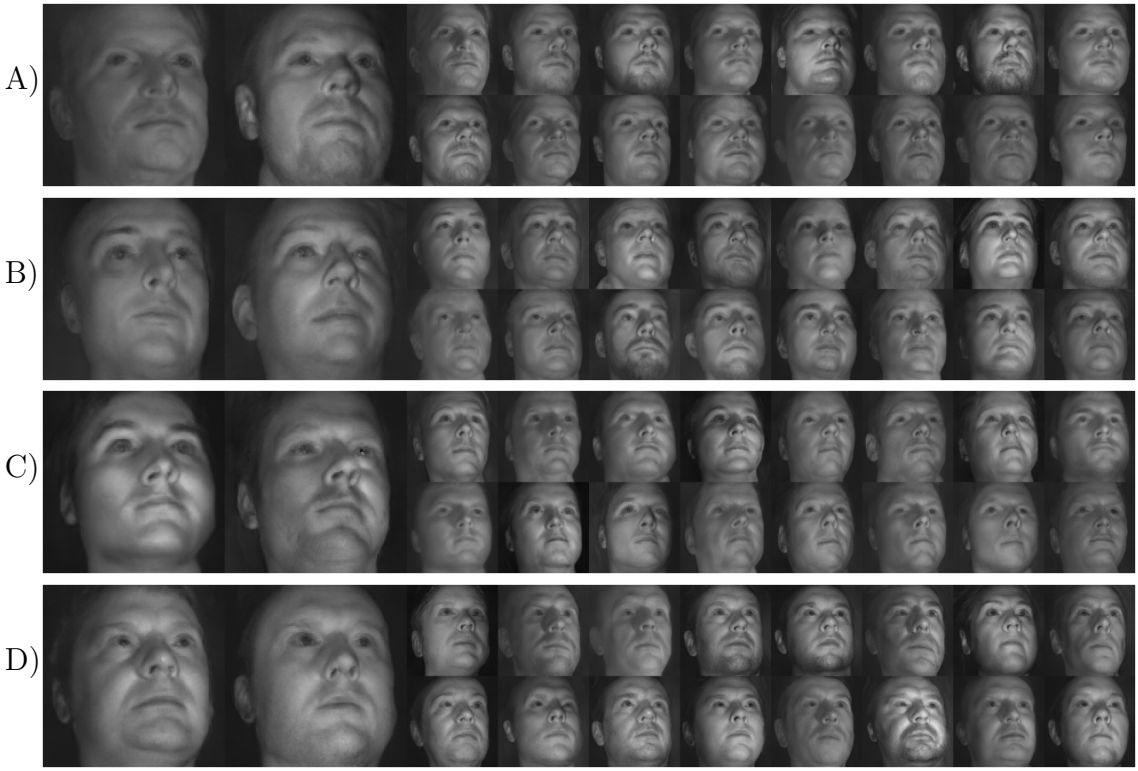


Figure 4.5: Random samples of generated images for the configurations presented in Table 4.4.

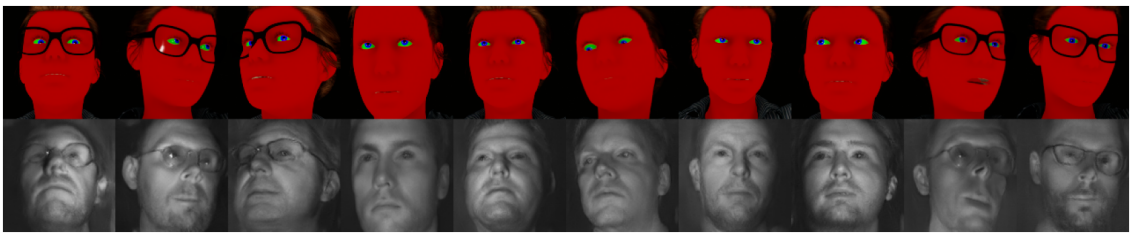


Figure 4.6: Generated images for randomly sampled source images for configuration A in Table 4.4.

Comparing with the results of Chapter 4.1.2, Table 4.4 shows that the configurations using least square loss had a higher FID and lower LPIPS score for all perceptual losses. However, for hinge loss, feature matching resulted in a significantly lower FID score which indicates that the instability problem was improved. Figure 4.6 shows generated images for the same random sample of source images as in Figure 4.4 and indeed shows that feature matching seems to have stabilized the hinge loss configuration.

4.1.4 Diversity regularization

This part investigates the diversity regularization presented in Chapter 3.2.2.2, again for the CGAN framework using the encoder-decoder generator and least square adversarial loss.

4.1.4.1 Latent reconstruction loss

To begin, the effect of adding the latent reconstruction loss $\mathcal{L}_{lr}$ (Chapter 3.2.2.2.1) to the CGAN framework was investigated. The configurations that were tried are presented in Table 4.5 together with the resulting FID and LPIPS scores. Figure 4.7 shows random samples generated using a fixed source image.

Ref.	Configuration	Adv. loss	λ_{lr}	FID	LPIPS
A	ED-LS-LR-1	Least square	1	14.7	**0.179**
B	ED-LS-LR-2	Least square	2	**12.3**	0.171
C	ED-LS-LR-3	Least square	5	15.4	0.171
D	ED-H-LR	Hinge	1	18.6	0.135

Table 4.5: FID and LPIPS scores for the framework using latent reconstruction loss.

Comparison with the results of Chapter 4.1.2 shows that the addition of the latent reconstruction loss resulted in a lower FID score for all tested configurations, as well as a slightly lower LPIPS score. Figure 4.7 also seems to indicate that the quality of the generated images is higher.

Figure 4.7: Random samples of generated images for the configurations presented in Table 4.5.

4.1.4.2 Style diversity loss

In this part, the effect of adding the style diversity loss $\mathcal{L}_{sd}$ (Chapter 3.2.2.2) in addition to the latent reconstruction loss was investigated, as well as using different distance measures $d_z(\cdot,\cdot)$ and $d_I(\cdot,\cdot)$ in latent space and image space respectively. Table 4.6 presents the configurations that were tried together with the resulting FID and LPIPS scores. Figure 4.8 shows random samples generated for each configuration using a fixed source image.

As can be seen in Figure 4.8, addition of the style diversity loss resulted in a breakdown in the generated images for configuration `ED-LS-LR-SD`, i.e. for the least square loss. It did, however, lead to a lower FID score for most configurations using hinge loss. Out of these, configuration `ED-H-LR-SD-1` performed best, both in terms of FID and LPIPS.

As a side point, it is interesting to note that the generated images for configuration A contain what seems to be distinct patterns. While this would have to be investigated in more detail to tell for certain, it is possible that each pattern encodes the latent vector z that was used to generate it, such that the encoder E can still make accurate estimations $\hat{z}$ for minimization of the latent reconstruction loss.

Ref.	Configuration	Adv. loss	$d_I(\cdot)$	$d_z(\cdot)$	λ_{sd}	FID	LPIPS
A	ED-LS-LR-SD	Least square	FaceId ResNet-50	Const 1	1	-	-
B	ED-H-LR-SD-1	Hinge	FaceId ResNet-50	Const 1	1	**17.1**	**0.164**
C	ED-H-LR-SD-2	Hinge	FaceId ResNet-50	L1	1	19.4	0.144
D	ED-H-LR-SD-3	Hinge	L1	Const 1	1	17.4	0.147
E	ED-H-LR-SD-4	Hinge	Discriminator	Const 1	1	55.8	0.0864
F	ED-H-LR-SD-5	Hinge	VGG-19 ImageNet	Const 1	1	40.5	0.0619

Table 4.6: FID and LPIPS scores for the configurations of the framework using style diversity loss, including different distance measures $d_I(\cdot)$ and $d_z(\cdot)$.

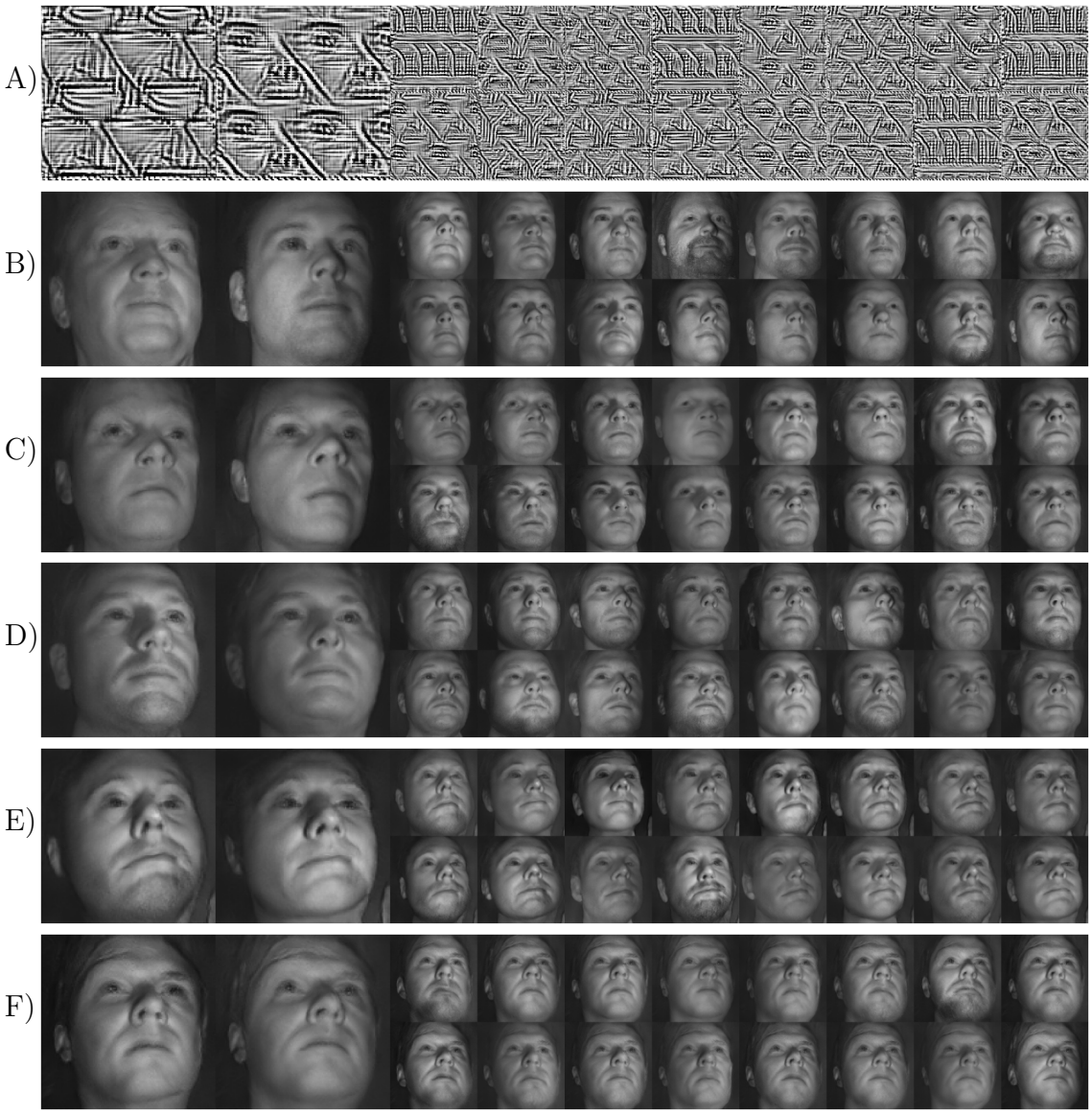


Figure 4.8: Random samples of generated images for the configurations presented in Table 4.6.

4.1.5 Style mapping

This experiment investigated the addition of the style mapping network L presented in Chapter 3.4.4. Three different configurations were tried and are presented in Table 4.7 together with the resulting FID and LPIPS scores of each. Random samples from each configuration are shown in Figure 4.9.

As can be seen, the style mapping did not result in an improvement for the framework, but instead a type of mode collapse. In the baseline configuration A, the style mapping network seem to map all latent vectors z to a fixed style vector s and produces images of poor quality. This problem is somewhat alleviated by introduction of the diversity regularization as in configuration B and VGG feature matching in

configuration C, but the generated images are still of limited diversity.

It is notable that the LPIPS score for configurations B was high, despite Figure 4.9 showing that the corresponding samples had limited diversity. This is likely explained by the breakdown in some of the generated images that is also visible, which causes a high LPIPS score when these are measured against the other generated faces.

Ref.	Configuration	Setting	FID	LPIPS
A	ED-LS-SM	Baseline encoder-decoder	25.3	$2.82 \cdot 10^{-6}$
B	ED-LS-LR-SD-SM	+ Diversity regularization	20.1	**0.193**
C	ED-LS-P-LR-SD-SM	+ Feature matching	**15.8**	0.164

Table 4.7: FID and LPIPS scores for configurations of the framework using the style mapping network.

Figure 4.9: Random samples of generated images for the configurations presented in Table 4.7.

4.1.6 Identity reconstruction loss

This part investigates how the identity reconstruction loss $\mathcal{L}_{ir}$ improves the CGAN frameworks ability to reconstruct latent vectors $\hat{z} = E(x)$ that reproduce the style of real images x. To begin, the effect of $\mathcal{L}_{ir}$ on the quality of the generated images is evaluated. Table 4.8 shows configurations tried using the identity reconstruction loss and the resulting FID and LPIPS score of each.

Comparing the results with those of Table 4.5, the identity reconstruction loss is seen to have increased the FID score slightly compared to the same configurations

Ref.	Configuration	Adv. loss	$d_I(\cdot)$	λ_{lr}	λ_{ir}	FID	LPIPS
A	ED-LS-LR-IR-1	Least square	FaceId	1	1	15.7	0.173
B	ED-LS-LR-IR-2	Least square	FaceId	2	1	15.3	0.178
C	ED-LS-LR-IR-3	Least square	L1	2	1	15.7	0.180
D	ED-LS-LR-IR-4	Least square	VGG	2	1	**12.7**	**0.183**
E	ED-H-LR-IR	Hinge	1	2	1	13.1	0.161

Table 4.8: FID and LPIPS scores for configurations of the framework using identity reconstruction loss.

from previous chapters. However, while a low FID score is desirable, it was not the core motivation for adding the latent reconstruction loss.

To show the effect of the identity reconstruction loss on the framework's ability to reproduce the style of previously unseen real images, Figure 4.10 shows examples of real images reconstructed by the configurations B, C and D in Table 4.8. To see the difference from a similar configuration without identity reconstruction loss, configuration `ED-LS-LR-2` (configuration B in Table 4.5) was also used to reconstruct the same real faces. As can be seen, the identity reconstruction loss improves the framework's ability to reproduce the appearance of the face in the real image.

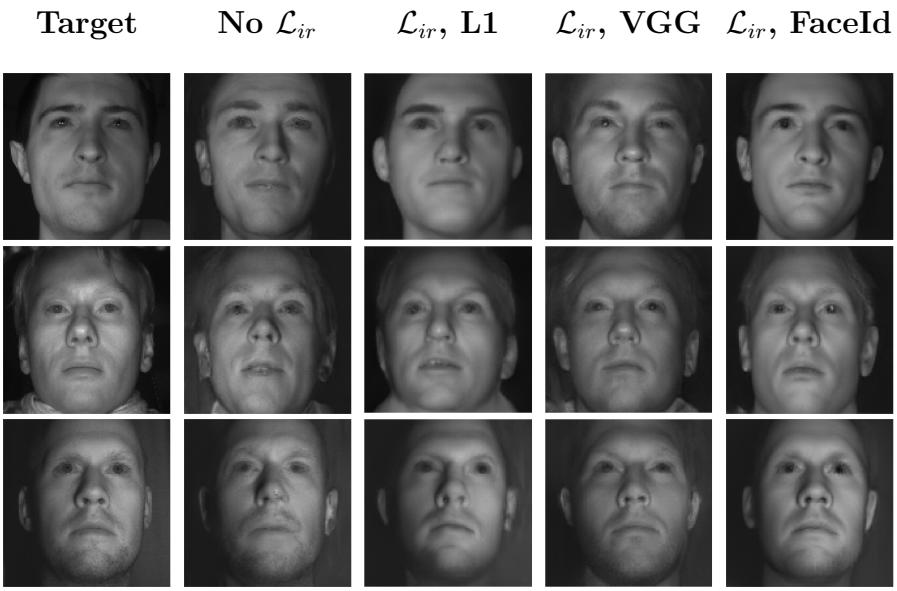


Figure 4.10: Examples of reconstructed appearances for the configurations B, C and D in Table 4.8 using identity reconstruction loss, as well as configuration B in Table 4.5 for comparison.

4.1.7 Summary

The FID and LPIPS scores of the configurations presented throughout Chapter 4.1 are summarized in the bar chart of Figure 4.11, separated into groups for each

experiment. Out of the tested configurations, `ED-LS-LR-2` had the lowest FID score while also having a relatively high LPIPS score. Because of this, it was selected for further investigation in Chapter 4.2.

For a more comprehensive view of the realism and diversity of the faces generated by configuration `ED-LS-LR-2`, the Appendix shows an additional 80 random samples generated by this configuration for the same fixed source image used throughout Chapter 4.1.

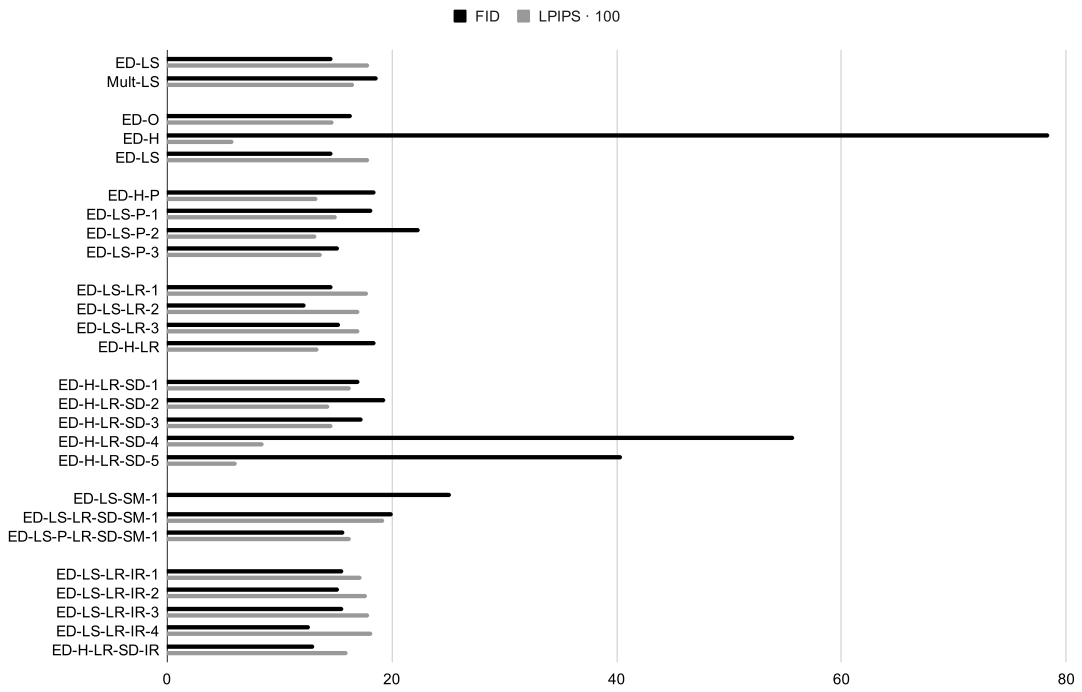


Figure 4.11: FID and LPIPS scores of the various configurations presented in Chapter 4.1, grouped by each experiment.

4.2 Framework analysis

The results presented in this chapter are primarily generated using configuration `ED-LS-LR-2`, since this was found to have the lowest FID score out of the tested configurations while simultaneously having a high LPIPS score.

However, this configuration did not use the identity reconstruction loss $\mathcal{L}_{ir}$, which was shown in Chapter 4.1.6 to result in better reconstruction of the appearances of faces in real images. Because of this, the images generated from reconstructed latent vectors in this chapter were instead generated using the configuration `ED-LS-LR-IR-2`, which is the same as the first configuration but with $\mathcal{L}_{ir}$ added.

4.2.1 Attribute preservation

In this part, the frameworks ability to preserve head pose and gaze direction is demonstrated.

4.2.1.1 Head pose

Figure 4.12 shows source image and the corresponding generated images for the chosen configurations. A) shows images generated using randomly sampled latent vectors by configuration `ED-LS-LR-2` and B) images generated from reconstructed latent vectors by configuration `ED-LS-LR-IR-2`.

As can be seen, the generated faces shown in A) seem to follow changes in head pose well, while those in B) can be seen to not align perfectly for the most extreme angles. It is also evident that the quality of all generated images declines for extreme head angles.

4.2.1.2 Gaze direction

Next, the framework's ability to preserve gaze direction was investigated. Figure 4.14 shows source image and A) images generated using randomly sampled latent vectors by configuration `ED-LS-LR-2` and B) images generated from reconstructed latent vectors by configuration `ED-LS-LR-IR-2`.

In addition to this, Figure 4.15 shows the same types of images, but where both A) and B) were generated by configuration `ED-H-LR-SD-IR` instead.

It is apparent that there is a difference in the ability to preserve gaze direction depending on which adversarial loss function the framework was trained with. In particular, the images generated by the framework trained with hinge loss can be seen to roughly follow the gaze direction of the 3D model, while those of the framework trained with least square loss does not.

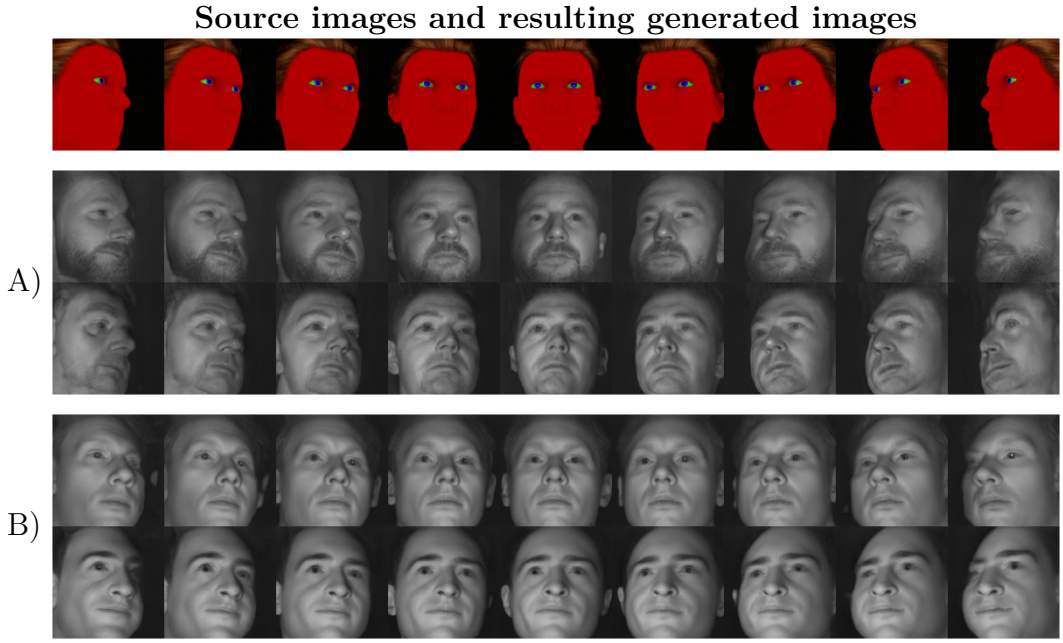


Figure 4.12: Horizontal variation in head pose for A) images generated using randomly sampled latent vectors by configuration `ED-LS-LR-2` and B) images generated from reconstructed latent vectors by configuration `ED-LS-LR-IR-2`.

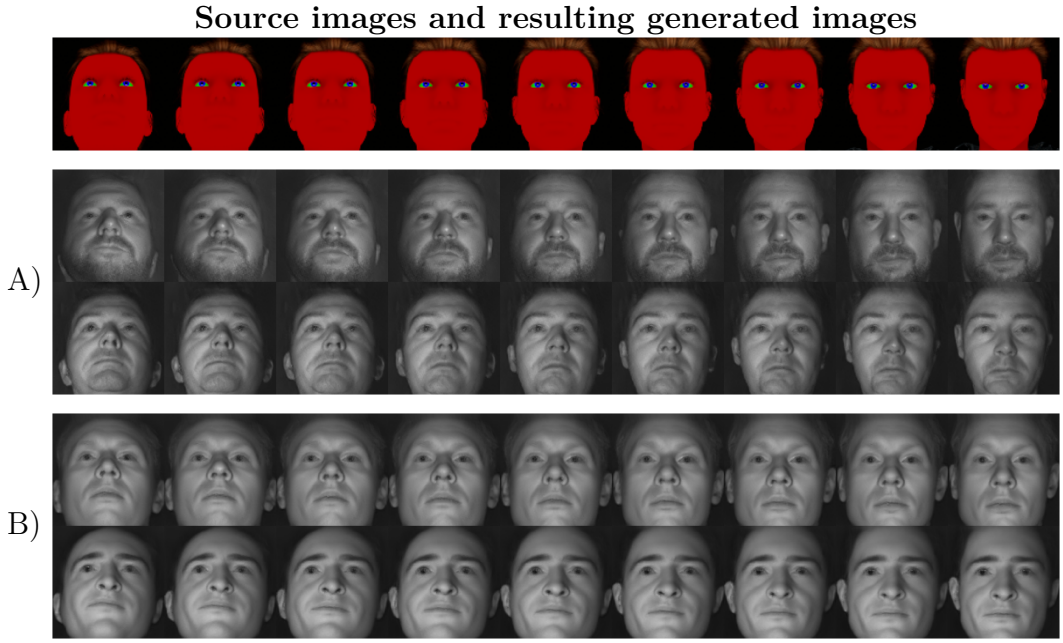


Figure 4.13: Vertical variation in head pose for A) images generated using randomly sampled latent vectors by configuration `ED-LS-LR-2` and B) images generated from reconstructed latent vectors by configuration `ED-LS-LR-IR-2`.

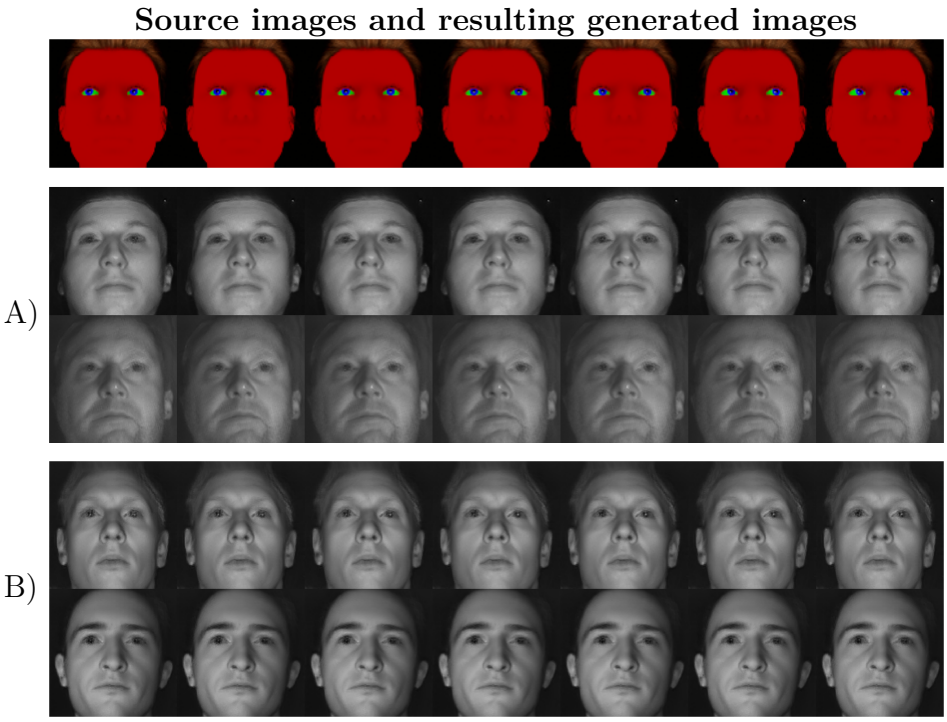


Figure 4.14: Horizontal variation in gaze direction for A) images generated using randomly sampled latent vectors by configuration `ED-LS-LR-2` and B) images generated from reconstructed latent vectors by configuration `ED-LS-LR-IR-2`.

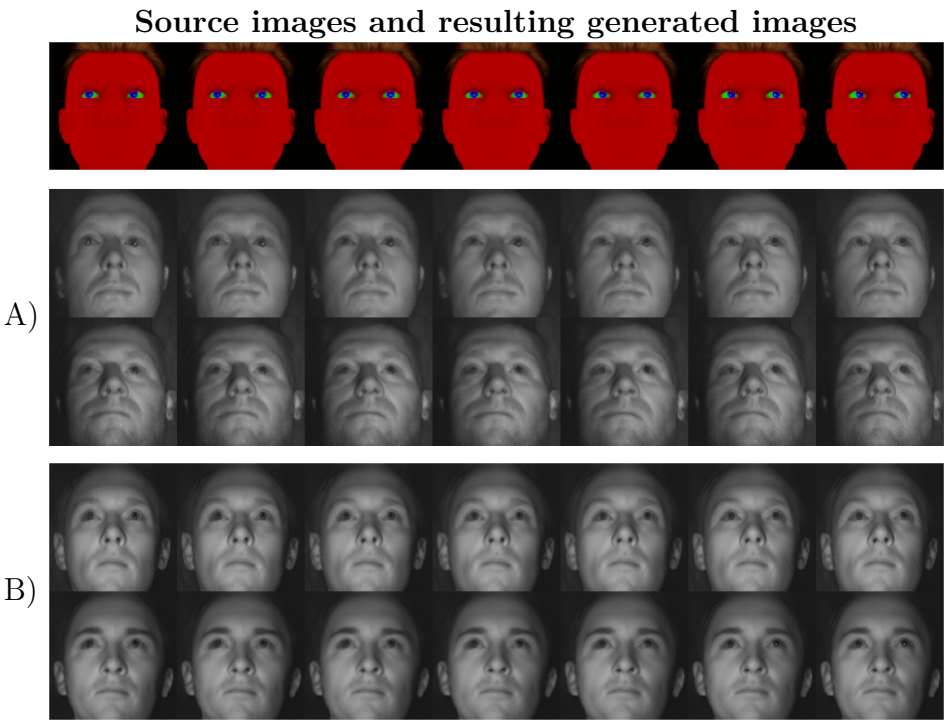


Figure 4.15: Horizontal variation in gaze direction for A) images generated using randomly sampled latent vectors and B) images generated from reconstructed latent vectors, both by configuration `ED-H-LR-SD-IR`.

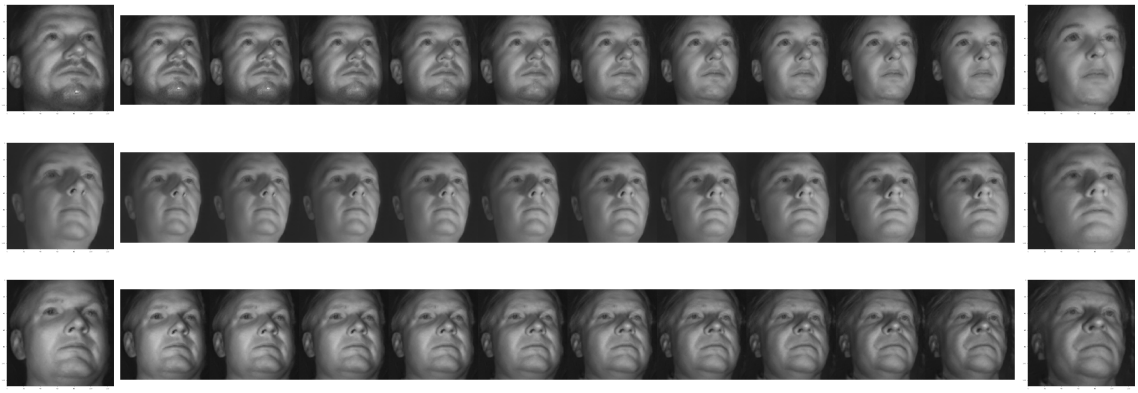


Figure 4.16: Latent space interpolation between the reconstructed latent vectors of the generated images on each side for configuration `ED-LS-LR-2`.

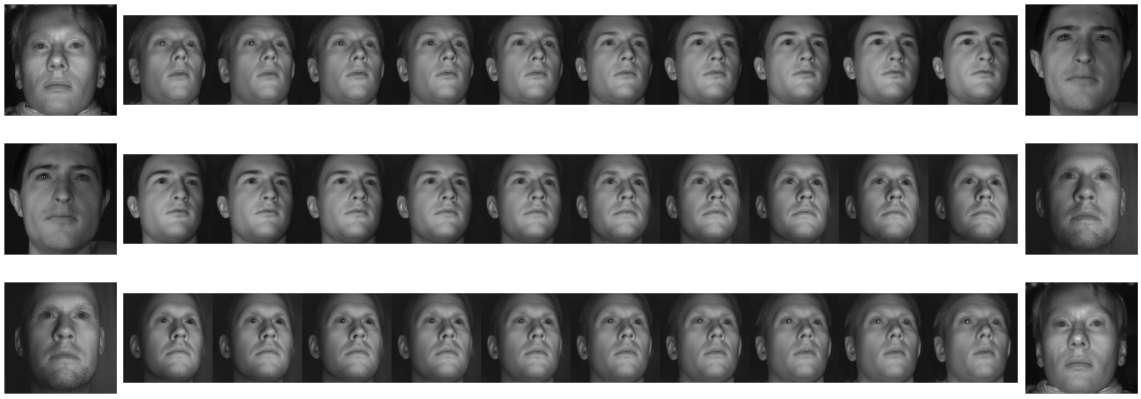


Figure 4.17: Latent space interpolation between the reconstructed latent vectors of the real images on each side for configuration `ED-LS-LR-IR-2`.

4.2.2 Latent space exploration

In this part, the latent space for configuration `ED-LS-LR-2` was investigated in more detail in order to see if there were signs of entanglement or mode collapse.

4.2.2.1 Latent space interpolation

To demonstrate that the framework has not simply memorized the appearance of a limited number of subjects, Figure 4.16 shows interpolation between the latent vectors of the generated images on each side for configuration `ED-LS-LR-2`. As can be seen, interpolating between these and generating images for the intermediate latent vectors creates a smooth transition between two faces. This suggests that the appearances of the generated faces are points in a continuous latent space representing different facial features.

In addition to this, Figure 4.17 shows interpolation using reconstructed latent vectors $\hat{z}$ for configuration `ED-LS-LR-IR-2`. Here too, it can be seen that interpolation between the latent vectors creates a smooth transition between the two reconstructed faces.

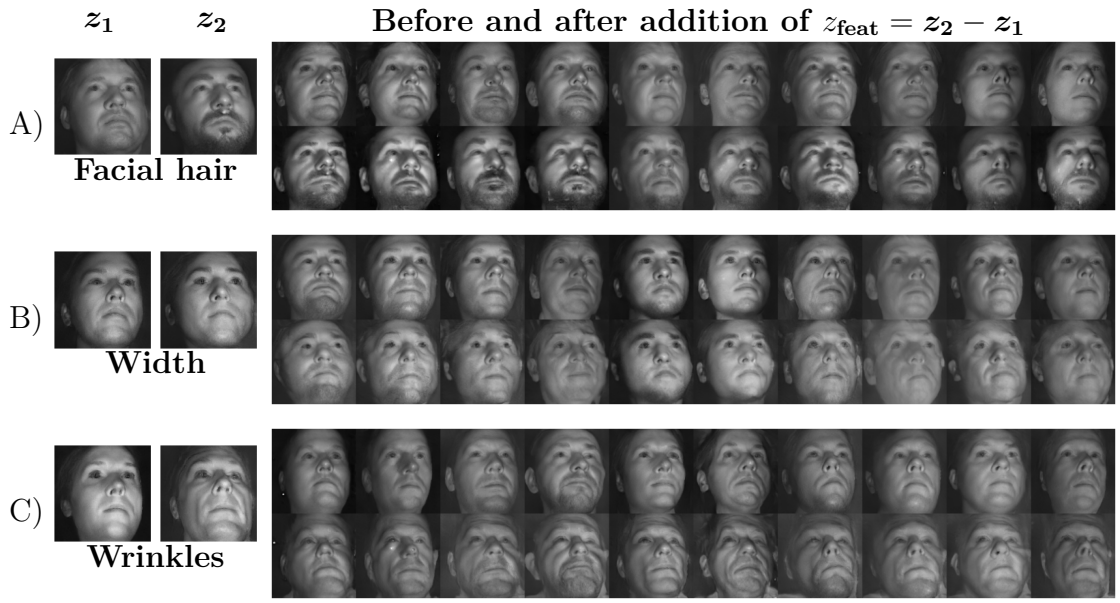


Figure 4.18: Resulting latent space arithmetic for configuration `ED-LS-LR-2`. The samples on the right-hand side are shown before and after addition of the feature vectors $z_{\text{feat}} = z_2 - z_1$ retrieved from the latent vectors z_1 and z_2 shown to the left.

4.2.2.2 Latent space arithmetic

To further verify that the latent space is disentangled, this experiment investigated if distinct features could be extracted using the latent space arithmetic technique described in Chapter 3.5.3.2.2.

Figure 4.18 shows latent arithmetic for a number of features for configuration `ED-LS-LR-2`. The left side shows the generated images corresponding to the two latent vectors z_1 and z_2 and the right side shows the generated faces before and after addition of the feature vector $z_{\text{feat}} = z_2 - z_1$ representing the extracted feature.

As can be seen, arithmetic using latent vectors allows facial attributes to be extracted and added to the corresponding faces. This suggests that different directions in the latent space correspond to distinct features, not just locally, but globally across the entire space.

5

Discussion

5.1 Discussion of results

5.1.1 Framework configuration

Generator The evaluation of the generator architectures performed in Chapter 4.1.1 showed that the encoder-decoder architecture was superior, both in terms of realism and diversity of the generated faces. From inspection of the generated images, it was apparent that the shape of the faces generated by the multinorm generator seemed to be more constrained to the shape of the 3D model. This is likely because the source images are being introduced using SPADE layers, which effectively turns them into pixel-wise filters that are applied to the internal feature maps, thus restraining the shape of the generated faces. SPADE was primarily developed for using semantic segmentation images as input, where there is a clear relationship between each pixel-value in the two domains. Because of this, SPADE might not be a suitable method for the type of multimodal mapping to a variety of faces that is desirable in this application.

Adversarial loss The results of Chapter 4.1.2 demonstrated that the type of adversarial loss used is important for both the realism and diversity of the generated images. Out of the tested adversarial losses (see Chapter 2.2.3.1), the least square adversarial loss was found to be superior both in terms of diversity and realism of the generated images. However, the experiments of Chapter 4.1.4 and 4.2.1.2 suggested that it also suffered from shortcomings, such as being more sensitive to instability issues and preserving gaze direction less well. While there are many potential explanations for these differences, they were not investigated in more detail and were instead taken as empirical results.

Perceptual loss None of the perceptual losses that were investigated resulted in an improvement in neither the FID nor LPIPS values for the least square adversarial loss, and inspection of the generated images confirms that no significant improvement was visible. However, the perceptual loss did result in a significant improvement for the hinge adversarial loss compared to the results in the previous Chapter 4.1.2. However, the previously high FID score for the hinge loss was shown to result from a breakdown in the generated images for source images where the 3D

model was wearing glasses, which the perceptual losses seem to have remedied. However, this breakdown was also corrected by later configurations, so the improvement can not be seen as directly attributable to the perceptual losses.

Diversity regularization The addition of the latent encoder E and the latent reconstruction loss $\mathcal{L}_{lr}$ (Equation 3.1) resulted in an improvement in the FID score for the adversarial losses tried (and a small reduction in LPIPS) as well as making the estimation of latent vectors possible. It can therefore be seen as a clear improvement for the framework. The style diversity loss $\mathcal{L}_{sd}$ (Equation 3.2), however, showed mixed results. In the case of hinge adversarial loss, it was shown to increase the diversity of the generated images as expected. Out of the tested distance measures, the combination using the feature space of the ResNet-50 facial identity network as $d_I(\cdot, \cdot)$ and $d_z(\cdot, \cdot)$ constantly equal to 1 were found to give the best results in terms of FID and LPIPS. In contrast to this, the style diversity loss resulted in a breakdown of the output for the framework trained with least square adversarial loss. As was commented on previously, the exact reason for this was not investigated further and was therefore taken as an empirical result.

Style mapping The style mapping results presented in Chapter 4.1.5 showed that the style mapping network L did not contribute to a more disentangled latent space at all. On the contrary, the diversity of the generated images was non-existent for the first configuration tried, where the only loss used was the least square adversarial loss. The addition of the diversity regularization as well as perceptual loss resulted in a minor improvement for the diversity of the generated faces, but despite this, the diversity remained low and the latent space did not seem represent a continuous space of facial features.

While the limited diversity of the generated faces in the later configurations seem to suggest mode collapse occurring, this does not explain why the images had no diversity when only the style mapping network was used. Furthermore, investigation of the style vectors outputted by the style mapping network for the first configuration revealed that these remained constant for all inputs, which is why all the generated samples had the same appearance.

A possible explanation for this is that the style mapping network fails to find a useful mapping of the latent vectors z to style vectors s, and therefore only learns to output a constant that minimizes the mean loss for all images. It is also conceivable that the problem can be attributed to issues with network optimization. The style mapping network was trained with a learning rate two orders of magnitude lower compared to the other networks, in order to allow them to adapt to the changing mapping. However, since the Adam algorithm used for optimization of each network is momentum-based, it is possible that it throws off the desired ratios between the learning rates and causes instabilities during training of the framework. Other optimization algorithms were not tried due to time limitations, but it is possible that a different optimization algorithm that is not based on momentum, such as SGD, could alleviate the problem.

Identity reconstruction loss The addition of the identity reconstruction loss 3.3 was shown to improve the framework's ability to reconstruct the appearance of subjects in real images from estimated latent vectors. The reconstruction was found to be most accurate when the ResNet-50 facial identity network was used for measuring $d_I(\cdot, \cdot)$. Furthermore, the reconstruction was found to be accurate even when the subjects were previously unseen by the framework.

Inspection of the reconstructed faces, however, seems to suggest a slightly blurrier texture compared to faces generated from randomly sampled latent vectors. While this may seem strange at first, it can likely be explained by how the identity reconstruction was implemented during training. In particular, in the implementation that was used, the images generated from reconstructed latent vectors were only subject to the identity reconstruction loss, and not the adversarial loss unlike images generated from random latent vectors. It was hypothesized that it would be sufficient to only use adversarial loss for one type of generated images and that this would ensure the framework to learn the correct mapping for all latent vectors. However, since it is the adversarial loss that enforces the realism of the generated images, its absence for reconstructed images likely explains why their texture appears less realistic.

5.1.2 Framework analysis

Attribute preservation The results presented in Chapter 4.2.1.1 show that the model has learned to preserve variations in head pose reasonably well, which shows that the approach of using a CGAN for enforcing this is a viable solution. However, it is evident that the quality of the generated faces declines for the most extreme head angles. This is likely explained by the fact that image pairs with extreme head angles constitute a minority in the training dataset and are not seen as often by the framework during training, resulting in the generator getting less feedback on image quality in this case. It is also evident that the images generated from reconstructed latent vectors seem to follow variations in head pose less well compared to those generated from randomly sampled latent vectors. This is also likely caused by reconstructed images not being subject to the adversarial loss, as this is what also enforces the alignment between generated and source images.

For the the gaze direction, the results of Chapter 4.2.1.2 showed that the model did not learn to perfectly preserve this attribute. Somewhat surprisingly, however, this seemed to depend on the adversarial loss function used. Out of the tested configurations, the ones using least square adversarial loss seemed to almost completely ignore changes in gaze direction, while the ones using hinge loss followed it to some degree. The exact reason for this remains unclear, but one possible explanation is that the hinge loss more strictly enforces the generated/source image pairs to resemble the real/source image pairs. However, this would have to be investigated in more detail before such a conclusion could be made.

Latent interpolation and arithmetic The two experiments performed in Chapter 4.2.2 showed that the latent space of the model is disentangled to a large extent. Starting with the latent space interpolation of Chapter 4.2.2.1, the faces can be seen to smoothly transition as the intermediate latent vectors are varied. Since smooth transitions require the features of the faces to vary gradually and independently, this is a likely indication of a disentangled latent space. The results of Chapter 4.2.2.2 further showed that high-level features can be extracted and added to other generated images through latent space arithmetic. Since it was shown to work well for a variety of randomly sampled latent vectors, this indicates that high-level features like these correspond to distinct directions in the latent space, and that this structure is a global property for the entire space.

5.2 Future work

Attribute preservation losses One possible way to force the model to better preserve domain-invariant attributes, such as head pose and gaze direction, could be to add preservation losses that are derived from estimation networks. More specifically, by introducing separate networks for inferring these attributes for the generated images, the difference between their estimations $F(G(z, y))$ and the ground truth labels $w(y)$ for the source image could be used to form a loss on the form

$$\mathcal{L}_{\text{ap}} = \min F(G(z, y)) - w(y)\,.$$

The GazeGAN [17] study found that a loss similar to this was beneficial for preserving gaze direction during image-to-image translation. Although Smart Eye does have gaze estimation networks available, these are implemented in TensorFlow and were therefore not compatible with the PyTorch implementation of the CGAN framework of this thesis. It is therefore left as a possible improvement for future work on this topic.

Ground truth labels Another potential modification to improve the CGAN frameworks ability to preserve domain-invariant attributes could be to provide the ground truth labels as a part of the conditional information in addition to the source images. In the current implementation, both the generator and discriminator are only given the source images as the conditional information and are therefore required to infer the gaze direction and head pose from them. Providing the ground truth labels directly would remove the need for this and allow both networks to work with accurate labels. In fact, the effect of adding ground truth labels would be similar to the attribute preservation loss described above, but instead of enforcing it using an estimation network, it is the conditional discriminator that would force the generator to learn the correct relationship.

Facial identity network trained on near-infrared data Using the feature space of the facial identity network presented in Chapter 3.2.4.2.2 as the distance

measure $d_I(\cdot, \cdot)$ in the identity reconstruction loss was shown to result in the most accurate reconstruction of subjects in real images. However, while the reconstructed faces resembled the real subjects they were approximated from, the reconstruction did not work perfectly in all cases. The facial identity network used was a ResNet-50 network pretrained on the VGGFace2 dataset containing RGB images. Even though the faces in the RBG images likely share features with those in near-infrared images, the agreement is most likely not perfect. Thus, pretraining the facial identity network on a dataset of near-infrared images would likely result in even better reconstruction of real subjects.

Larger and more diverse dataset Another factor that would likely improve the quality as well as the diversity of the generated is to increase the size of the dataset that is used to train the model. A dataset containing more unique subjects in a variety of head poses, gaze angles and perspectives would allow both generator and discriminator to generalize better and reduce the risk for overfitting. It would also likely improve the quality of the generated images for all head poses, including extreme head angles, as well as making the identity reconstruction work better.

Additionally, it is likely that better data cleaning methods would improve the framework's performance. Even though data cleaning was employed to remove bad samples from the used dataset, it did not work perfectly and there were still image pairs with poorly aligned head poses or gaze directions. Since it is the image pairs that instruct the generator of the correct mapping, it is likely that a cleaner dataset would result in an increase in the framework's ability to preserve domain-invariant attributes.

Tuning hyperparameters and layer sizes It is also possible that further tuning of the various hyperparameters and layer sizes in the network architectures would result in better results. While some amount of tuning was done during the thesis work, no large scale investigation to find the best values was performed. As was discussed briefly in Chapter 5.1.1, it is also possible that using an optimization algorithm that is not based on momentum, such as SGD, could alleviate the instability problems encountered for some configurations.

5.3 Main contributions

The CGAN framework presented in Chapter 3.2 has been demonstrated to be capable of learning to generate controllable, realistic faces in the near-infrared domain with diverse facial attributes. Furthermore, with the addition of the identity reconstruction loss, the framework was able to extract a latent vector representing the appearance of subjects in previously unseen real images and use it to generate images with a similar appearance. The chosen approach of training a CGAN framework to perform multimodal image-to-image translation for paired images from the two image domains has therefore been shown to be a viable method for synthesizing data suitable for DMS development.

A clear benefit of this approach is the fine-grained control that the 3D model allows over the adjustable attributes, such as head pose and gaze direction, which can be varied independently of the stylistic attributes of the generated faces. This type of control is also possible to achieve using other approaches, for example by making these attributes a part of the latent space and using manipulation of the latent vectors z to control the generated images. However, methods of this type have been shown to suffer from entanglement issues similar to those described in Chapter 3.2.2.3. For example, if some facial attribute is over-represented for a given head pose in the training dataset, it could lead to this attribute varying in unison with the head pose. Remedying this requires the employment of disentanglement methods, which increases the complexity of the framework and is not guaranteed to perfectly disentangle the latent space [69].

The thesis has not, however, investigated how synthetic training data affects the performance of DMS systems, as this was out of the thesis' scope. This is therefore left for future work to investigate.

5.4 Ethical, societal and environmental implications

The identity reconstruction was demonstrated to work reasonably well for previously unseen subjects. Although the reconstructed faces are still distinguishable from real faces, it is conceivable that derivative works could improve the framework further and make it capable of producing photo-realistic images like those of StyleGAN [4]. It could therefore be argued that it falls under the category known as *deep fakes*, i.e. AI technology capable of imitating the appearance of real people. As with any technology of this type, it could therefore have potentially malicious use cases, such as defamation or spread of misinformation. It is therefore important that new and better methods for detecting fake media produced by deep learning algorithms are developed, as this provides tools for preventing technology like this from being used for harmful purposes. There is a growing amount of attention and resources dedicated to this issue and progress is continuously being made [70].

On the other hand, methods for synthesizing realistic data like the one investigated in this thesis could also alleviate some of the privacy concerns that arise from collecting data from real users. In times where data privacy and protection are a big societal issues, this can be argued to be a positive feature of this technology. For example, there is ongoing research on using GANs for anonymization of data collected from real users. Several studies [71, 72] have investigated synthesis of medical data using GANs, creating datasets of "fake patients" that allow researchers to work with realistic data without having to be concerned about violating the privacy of real patients. It is conceivable that this is a use-case that could be relevant for other applications as well, such as anonymizing the highly personal data used for training DMS and similar vehicular safety systems.

From an environmental perspective, research and developments in generative modeling can have both positive and negative impacts. As with any type of technological development, it is possible that generative models and other deep learning algorithms could help reduce global energy consumption by fueling new, more energy efficient technologies. In the case of generative modeling in particular, advancements in synthetic data generation could potentially reduce the need for collecting and storing large amounts of real-world data, and instead make more efficient use of existing datasets by learning to create synthetic data with similar properties. To what extent this would alleviate the need for real data is, however, debatable.

However, generative models are also very computationally expensive to train, especially in the case of image synthesis. The images that were used in this thesis were of a relatively low resolution (128×128), so the amount of energy consumed during the project is estimated to have been relatively modest. There are, however, examples of studies where the research project as a whole has resulted in a significant energy consumption. For example, the StyleGAN 2 [73] paper estimated that the total electricity consumed during the project amounted to approximately 131.61 megawatt hours (MWh), which is enough to satisfy the yearly energy needs of about 5 average Swedish households [74].

With demands this high, it is important that future research projects focus on minimizing energy consumption, for example by avoiding unnecessary training sessions and possibly by increasing cooperation around large-scale models. Furthermore, large and computationally expensive generative models such as StyleGAN [4, 73] can often be re-purposed for many different tasks through so called fine-tuning, which can be an alternative to training a new model from scratch and thus reduce unnecessary energy consumption.

6

Conclusion

The thesis has investigated image synthesis with Generative Adversarial Networks (GANs) as an alternative way of obtaining training data for Driver Monitoring System (DMS) development. The chosen approach consisted of training a Conditional Generative Adversarial Network (CGAN) framework to derive a mapping between paired images of a synthetic 3D model and real human faces in the near-infrared domain that preserves domain-invariant attributes, e.g. head pose and gaze direction.

The thesis proposed a framework that was shown to be capable of multimodal image-to-image translation between the two image domains, mapping each 3D model image to a multitude of diverse and realistic human faces. Furthermore, with the addition of the proposed identity reconstruction loss, the framework was able to extract latent vectors representing the appearances of previously unseen subjects in real images and use them to generate faces with a similar appearances in novel poses and perspectives. The generated faces were found to remain consistent and adequately follow variations in perspective and pose of the 3D model, but were found to respond to changes in gaze direction less well.

Altogether, the thesis concludes that image synthesis using CGANs can be a viable method for obtaining customizable training data for DMS development. Future research is needed, however, on how refined synthetic data of this type affects the performance of other algorithms when used to enrich existing datasets or as a replacement of them entirely.

Bibliography

[1] Yunjey Choi et al. "StarGAN v2: Diverse Image Synthesis for Multiple Domains". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[2] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". *CoRR* abs/1611.07004 (2016). arXiv: 1611.07004. URL: http://arxiv.org/abs/1611.07004.

[3] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[4] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". *CoRR* abs/1812.04948 (2018). arXiv: 1812.04948. URL: http://arxiv.org/abs/1812.04948.

[5] Tero Karras et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". *CoRR* abs/1710.10196 (2017). arXiv: 1710.10196. URL: http://arxiv.org/abs/1710.10196.

[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". *CoRR* abs/1809.11096 (2018). arXiv: 1809.11096. URL: http://arxiv.org/abs/1809.11096.

[7] Qi Mao et al. "Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis". *CoRR* abs/1903.05628 (2019). arXiv: 1903.05628. URL: http://arxiv.org/abs/1903.05628.

[8] Taesung Park et al. "Semantic Image Synthesis with Spatially-Adaptive Normalization". *CoRR* abs/1903.07291 (2019). arXiv: 1903.07291. URL: http://arxiv.org/abs/1903.07291.

[9] Xingchao Peng et al. "Exploring Invariances in Deep Convolutional Neural Networks Using Synthetic Images". *CoRR* abs/1412.7122 (2014). arXiv: 1412.7122. URL: http://arxiv.org/abs/1412.7122.

[10] Saurabh Gupta et al. "Learning Rich Features from RGB-D Images for Object Detection and Segmentation". *CoRR* abs/1407.5736 (2014). arXiv: 1407.5736. URL: http://arxiv.org/abs/1407.5736.

[11] Erroll Wood et al. "Learning an Appearance-Based Gaze Estimator from One Million Synthesised Images". *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research Applications*. 2016, pp. 131–138.

[12] J. Shotton et al. “Efficient Human Pose Estimation from Single Depth Images”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (2013), pp. 2821–2840. DOI: 10.1109/TPAMI.2012.241.

[13] Jonathan Tompson et al. “Real-time continuous pose recovery of human hands using convolutional networks”. English (US). *ACM Transactions on Graphics* 33.5 (Aug. 2014). ISSN: 0730-0301. DOI: 10.1145/2629500.

[14] James Steven Supancic III et al. “Depth-based hand pose estimation: methods, data, and challenges”. *CoRR* abs/1504.06378 (2015). arXiv: 1504.06378. URL: http://arxiv.org/abs/1504.06378.

[15] Ashish Shrivastava et al. “Learning from Simulated and Unsupervised Images through Adversarial Training”. *CoRR* abs/1612.07828 (2016). arXiv: 1612.07828. URL: http://arxiv.org/abs/1612.07828.

[16] Sergey I. Nikolenko. “Synthetic Data for Deep Learning”. *CoRR* abs/1909.11512 (2019). arXiv: 1909.11512. URL: http://arxiv.org/abs/1909.11512.

[17] Matan Sela et al. “GazeGAN - Unpaired Adversarial Image Generation for Gaze Estimation” (2017). URL: http://arxiv.org/abs/1711.09767.

[18] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks” (2017). URL: http://arxiv.org/abs/1703.10593.

[19] Konstantinos Bousmalis et al. “Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks”. *CoRR* abs/1612.05424 (2016). arXiv: 1612.05424. URL: http://arxiv.org/abs/1612.05424.

[20] Gregory J. Stein and Nicholas Roy. “GeneSIS-RT: Generating Synthetic Images for training Secondary Real-world Tasks”. *CoRR* abs/1710.04280 (2017). arXiv: 1710.04280. URL: http://arxiv.org/abs/1710.04280.

[21] Ayush Tewari et al. “StyleRig: Rigging StyleGAN for 3D Control over Portrait Images, CVPR 2020”. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. June 2020.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[23] D. Foster. *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. O’Reilly Media, 2019. ISBN: 9781492041894. URL: https://books.google.be/books?id=RqegDwAAQBAJ.

[24] Miles Brundage et al. “The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation”. *CoRR* abs/1802.07228 (2018). arXiv: 1802.07228. URL: http://arxiv.org/abs/1802.07228.

[25] Andrej Karpathy et al. *Generative Models*. June 2016. URL: https://openai.com/blog/generative-models/.

[26] Lilian Weng. “From GAN to WGAN”. *CoRR* abs/1904.08994 (2019). arXiv: 1904.08994. URL: http://arxiv.org/abs/1904.08994.

[27] Martin Arjovsky and Léon Bottou. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017. arXiv: 1701.04862 [stat.ML].

[28] Mario Lucic et al. *Are GANs Created Equal? A Large-Scale Study*. 2017. arXiv: 1711.10337 [stat.ML].

[29] Xudong Mao et al. *Least Squares Generative Adversarial Networks*. 2017. arXiv: 1611.04076 [cs.CV].

[30] Dustin Tran, Rajesh Ranganath, and David M. Blei. *Hierarchical Implicit Models and Likelihood-Free Variational Inference*. 2017. arXiv: 1702.08896 [stat.ML].

[31] Jae Hyun Lim and Jong Chul Ye. *Geometric GAN*. 2017. arXiv: 1705.02894 [stat.ML].

[32] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[33] Hao-Wen Dong and Yi-Hsuan Yang. "Towards a Deeper Understanding of Adversarial Losses". *CoRR* abs/1901.08753 (2019). arXiv: 1901.08753. URL: http://arxiv.org/abs/1901.08753.

[34] J. Langr and V. Bok. *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning Publications, 2019. ISBN: 9781617295560. URL: https://books.google.de/books?id=HojvugEACAAJ.

[35] J. Wu. "Introduction to Convolutional Neural Networks". 2017.

[36] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[37] Kaiming He et al. "Deep Residual Learning for Image Recognition". *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[38] Zachary C. Lipton and Jacob Steinhardt. *Troubling Trends in Machine Learning Scholarship*. 2018. arXiv: 1807.03341 [stat.ML].

[39] Shibani Santurkar et al. *How Does Batch Normalization Help Optimization?* 2019. arXiv: 1805.11604 [stat.ML].

[40] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[41] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". *CoRR* abs/1607.08022 (2016). arXiv: 1607.08022. URL: http://arxiv.org/abs/1607.08022.

[42] Yuxin Wu and Kaiming He. "Group Normalization". *CoRR* abs/1803.08494 (2018). arXiv: 1803.08494. URL: http://arxiv.org/abs/1803.08494.

[43] Xun Huang and Serge J. Belongie. "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization". *CoRR* abs/1703.06868 (2017). arXiv: 1703.06868. URL: http://arxiv.org/abs/1703.06868.

[44] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. "A Learned Representation For Artistic Style". *CoRR* abs/1610.07629 (2016). arXiv: 1610.07629. URL: http://arxiv.org/abs/1610.07629.

[45] Golnaz Ghiasi et al. *Exploring the structure of a real-time, arbitrary neural artistic stylization network*. 2017. arXiv: 1705.06830 [cs.CV].

[46] Vincent Dumoulin et al. "Feature-wise transformations". *Distill* (2018).

[47] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[48] Arthur Juliani et al. "Unity: A General Platform for Intelligent Agents". *CoRR* abs/1809.02627 (2018). arXiv: 1809.02627. URL: http://arxiv.org/abs/1809.02627.

[49] Ekin Dogus Cubuk et al. "AutoAugment: Learning Augmentation Policies from Data". *CoRR* abs/1805.09501 (2018). arXiv: 1805.09501. URL: http://arxiv.org/abs/1805.09501.

[50] Xun Huang et al. "Multimodal Unsupervised Image-to-Image Translation". *CoRR* abs/1804.04732 (2018). arXiv: 1804.04732. URL: http://arxiv.org/abs/1804.04732.

[51] Jun-Yan Zhu et al. "Toward Multimodal Image-to-Image Translation". *CoRR* abs/1711.11586 (2017). arXiv: 1711.11586. URL: http://arxiv.org/abs/1711.11586.

[52] Dingdong Yang et al. "Diversity-Sensitive Conditional Generative Adversarial Networks". *CoRR* abs/1901.09024 (2019). arXiv: 1901.09024. URL: http://arxiv.org/abs/1901.09024.

[53] Richard Zhang et al. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". *CoRR* abs/1801.03924 (2018). arXiv: 1801.03924. URL: http://arxiv.org/abs/1801.03924.

[54] Qifeng Chen and Vladlen Koltun. "Photographic Image Synthesis with Cascaded Refinement Networks". *CoRR* abs/1707.09405 (2017). arXiv: 1707.09405. URL: http://arxiv.org/abs/1707.09405.

[55] Alexey Dosovitskiy and Thomas Brox. "Generating Images with Perceptual Similarity Metrics based on Deep Networks". *CoRR* abs/1602.02644 (2016). arXiv: 1602.02644. URL: http://arxiv.org/abs/1602.02644.

[56] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". *CoRR* abs/1603.08155 (2016). arXiv: 1603.08155. URL: http://arxiv.org/abs/1603.08155.

[57] Ting-Chun Wang et al. "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs". *CoRR* abs/1711.11585 (2017). arXiv: 1711.11585. URL: http://arxiv.org/abs/1711.11585.

[58] Tim Salimans et al. "Improved Techniques for Training GANs". *CoRR* abs/1606.03498 (2016). arXiv: 1606.03498. URL: http://arxiv.org/abs/1606.03498.

[59] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. *International Conference on Learning Representations*. 2015.

[60] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. “Image Style Transfer Using Convolutional Neural Networks”. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[61] Ting-Chun Wang et al. “Few-shot Video-to-Video Synthesis”. *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[62] Q. Cao et al. “VGGFace2: A dataset for recognising faces across pose and age”. *International Conference on Automatic Face and Gesture Recognition*. 2018.

[63] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[64] Martin Heusel et al. “GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium”. *CoRR* abs/1706.08500 (2017). arXiv: 1706.08500. URL: http://arxiv.org/abs/1706.08500.

[65] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[66] Takeru Miyato and Masanori Koyama. “cGANs with Projection Discriminator”. *CoRR* abs/1802.05637 (2018). arXiv: 1802.05637. URL: http://arxiv.org/abs/1802.05637.

[67] Andrew L. Maas. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. 2013.

[68] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS’10). Society for Artificial Intelligence and Statistics*. 2010.

[69] Chaoyou Fu et al. “High Fidelity Face Manipulation with Extreme Pose and Expression”. *CoRR* abs/1903.12003 (2019). arXiv: 1903.12003. URL: http://arxiv.org/abs/1903.12003.

[70] Ruben Tolosana et al. *DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection*. 2020. arXiv: 2001.00179 [cs.CV].

[71] Pedro Costa et al. “Towards Adversarial Retinal Image Synthesis”. *CoRR* abs/1701.08974 (2017). arXiv: 1701.08974. URL: http://arxiv.org/abs/1701.08974.

[72] Edward Choi et al. “Generating Multi-label Discrete Electronic Health Records using Generative Adversarial Networks”. *CoRR* abs/1703.06490 (2017). arXiv: 1703.06490. URL: http://arxiv.org/abs/1703.06490.

[73] Tero Karras et al. “Analyzing and Improving the Image Quality of StyleGAN”. *CoRR* abs/1912.04958 (2019). arXiv: 1912.04958. URL: http://arxiv.org/abs/1912.04958.

[74] Swedish Energy Agency. Mar. 2015. URL: https://www.energimyndigheten.se/en/news/2011/new-regional-energy-statistics-for-single--or-two-dwelling-buildings/.

A

Appendix

Figure A.1: Random samples of generated images for configuration `ED-LS-LR-2` (see Chapter 4.1.4.1).

s