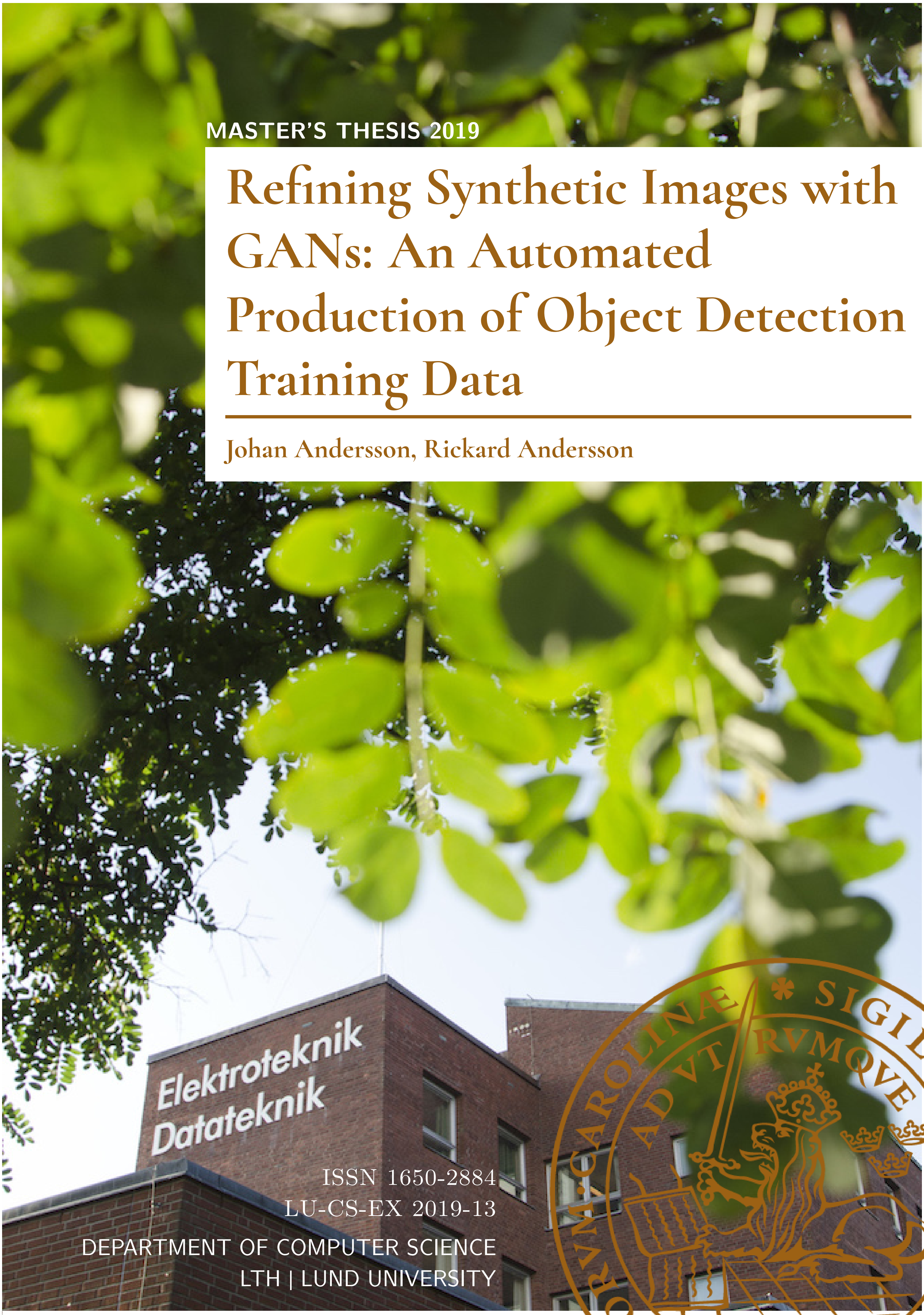
MASTER'S THESIS 2019

Refining Synthetic Images with GANs: An Automated Production of Object Detection Training Data

Johan Andersson, Rickard Andersson



ISSN 1650-2884
LU-CS-EX 2019-13

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2019-13

Refining Synthetic Images with GANs: An Automated Production of Object Detection Training Data

Johan Andersson, Rickard Andersson

Refining Synthetic Images with GANs: An Automated Production of Object Detection Training Data

Johan Andersson

johan-andersson01@protonmail.com

Rickard Andersson

rickardandersson1993@gmail.com

June 27, 2019

Master's thesis work carried out at Modcam AB.

Supervisor: Volker Krueger, volker.krueger@cs.lth.se

Examiner: Elin Anna Topp, elin_anna.topp@cs.lth.se

Abstract

With the rise of machine learning, there is an increasing demand for training data but a lack of supply. Synthetic training data has become a common tool to narrow this gap, but it often lacks complexity in order to fully approximate real data.

In this thesis, we explore methods of refining synthetic images of humans to look more realistic. Specifically, we perform the refinement with Generative Adversarial Networks and then use the refined images as training data for an object detection network.

While we do not succeed in creating realistic images, we do gain quantitative improvements from using refined synthetic data as opposed to only real data, when the size of the real training dataset is small.

With our best refined synthetic dataset, we find an improvement of 0.0165 in F_1-score compared to real training data. Compared to unrefined synthetic training data we find an improvement of 0.0289. Combined with real training data, we find an improvement of 0.0225 compared with only real training data.

Keywords: generative adversarial networks, synthetic training data, object detection, image-to-image translation, CycleGAN

Acknowledgements

We wish to express our deepest gratitude to our supervisor Volker Krueger whose encouragement and insightful suggestions have been invaluable during the course of this thesis.

Special thanks to Marcus Leyman, Karl-Anders Johansson, and the team at Modcam for their continuous enthusiasm, guidance, and support of our work.

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this thesis is to explore methods, centered around Generative Adversarial Networks (GANs), of generating photorealistic images. Specifically we generate synthetic images that roughly approximate real fisheye images of humans (see Fig. 1.1) and then refine the synthetic images with GANs to further approximate the real dataset.

Finally, we use our refined synthetic datasets as training data for an object detection network and evaluate whether the network is able to generalize from our refined datasets to real data.

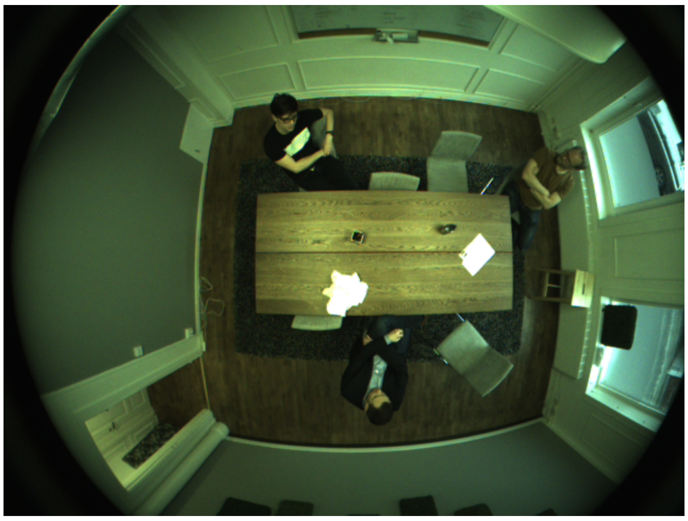


Figure 1.1: A real image sample.

1.2 Background

Since their revitalization during the last decade, deep learning techniques have always required large sets of training data in order to perform well. Collecting and annotating quality training data is however a time-consuming task, typically performed by humans. Would it not be great if we could let computers do it for us?

Synthetic data, i.e. computer-generated data, can be automatically generated with annotations in arbitrary amounts and with high variance. With large training set size and high variance, we are more likely to have an accurate representation of the real world compared to a real but smaller (and thus less varied) dataset - assuming that the synthetic data approximates the real data well enough. Using synthetic data to train machine learning models is thus enticing, but generating synthetic data that is good enough, i.e. data that looks real, is hard.

In this thesis, we therefore aim at refining annotated synthetic images to look real. We call these resulting images refined synthetic images. Specifically, the real dataset we wish to approximate is comprised of fisheye images of humans taken from above (see Fig. 1.1).

To refine the synthetic images, we use Generative Adversarial Networks (GANs). GANs are well-known by now for their ability to generate new realistic images from relatively small amounts of training data. Our approach follows two steps: First, we generate synthetic images using a graphics program. Since we generate those images, we also know the annotations. Second, we refine these synthetic images by employing a GAN-based approach.

This thesis operates in an area of data approximation where the goal is to generate synthetic images and then refine them for use as training data with Generative Adversarial Networks. Previous works in this area include [1] and [2]. Both of these works share the end goal of creating training data of small realistic images of one object without context.

In contrast to these works, our thesis aims to develop an approach for creating large realistic images where multiple humans are part of an environment, where each position of the humans is annotated. These images are then used as training data for an object detection network, where the end goal is to identify the locations of all humans in an image. We hope to approximate a real dataset to such an extent that our refined synthetic dataset can be used as training data for object detection on real images.

1.2.1 Company

This thesis was written at Modcam, a company based in Malmö, Sweden, that sells ceiling-mounted sensor solutions for analysis of people flow (see Fig. 1.2). After installation at a new location, the customer allows the company to take images during a setup period. These images are then annotated and used as training data for human detection networks. The networks are then deployed on the device and the setup period is completed.

After the setup period, neither customer nor Modcam is able to collect images from the sensor. Inferences are run on the device and images are never stored.

This is beneficial in terms of protecting people's privacy and ensuring that there cannot be any surveillance activity. However, it also limits the company's ability to acquire training data. This is the problem that our thesis tries to solve: to automate and scale the production of training data, while still safeguarding people's privacy.

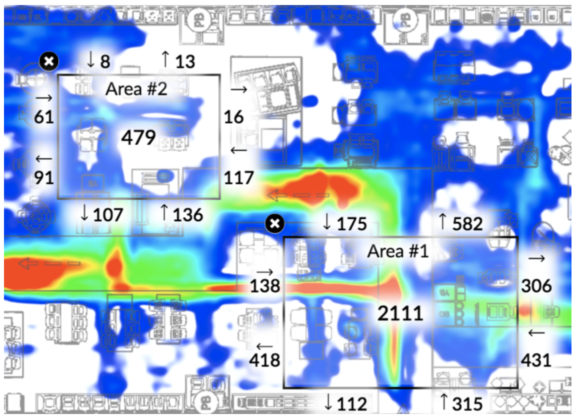


Figure 1.2: Generated heatmap of people flow, using human detection inferences.

1.2.2 Problem Formulation and Goals

The overall goal of this thesis is to refine synthetic images of humans with GANs, in order to generate a training dataset of refined synthetic images that is able to replace a real training dataset for use in a human detection task using SqueezeDet [3]. In order to replace real training data, a detection network trained on refined synthetic data must, during testing on real data, have equal or greater performance than a corresponding detection network trained on real data.

Our evaluation method is to train human detection networks on synthetic, refined synthetic, and real data respectively and then test these networks on real data. What we wish to see is that the performance of a network trained on our refined synthetic data can measure up to the performance of a network trained on real data. We also compare the performance of networks trained on refined synthetic data with a network trained on synthetic data, since negligible differences would make our approach superfluous.

We limit our human detection performance evaluation to using SqueezeDet [3], due to time constraints. However, the refinement of the synthetic images is done independently from the employed network. This way, our results should ideally generalize to other object detection networks.

1.2.3 Previous Work

Augmenting training data with the help of Generative Adversarial Networks (GANs) is by now an established and viable approach [4] [5] [6]. Another established approach of augmenting training data is to generate synthetic, annotated data with graphics software [7] [8] [9].

These approaches, while succesfully employed in some scenarios, have their limitations when it comes to generating object detection training data. GANs can produce realistic data, but as of writing they can not (to our knowledge) produce corresponding ground truth annotations of object locations. Computer graphics software can produce data with ground truth annotated objects, but has difficulties in achieving realism. It is then an intuitive idea to combine the two approaches. By first generating a synthetic dataset, we have a ground truth annotated set of training data. Then, by using GANs to make the synthetic dataset look more realistic, we ideally have both ground truth annotations and realism.

Previous work in refining synthetic images include [1] and [2]. Below, we summarize these works.

Learning from Simulated and Unsupervised Images through Adversarial Training

To our knowledge, Shrivastava *et al.*'s paper [2] was the first paper that used GANs to refine synthetic images for the purpose of training data. With the help of unannotated real images they refine synthetic images using a GAN in order to approximate the real training dataset. Specifically, this is done for a hand pose estimation dataset. Using real, synthetic and refined synthetic datasets for training a hand pose estimation network they achieve an accuracy of 74.5%, 69.7% and 72.4% respectively. Using synthetic and refined synthetic datasets three times larger in size they attain accuracies of 77.7% and 83.3% respectively, surpassing the accuracy of the real dataset.

In their dataset domain, Shrivastava *et al.* thus manage to generate datasets that, when used as training data, outperform a real training dataset [2].

Generation of Artificial Training Data for Deep Learning

In Wessman and Andersson's thesis [1], the goal was to render synthetic images of humans and refine them with a GAN to increase realism and replace the need for real re-identification training data. Their results show subjectively increased realism and quantitative improvements compared to synthetic images, but not a quantitative performance increase, when used as training data, compared to real data [1].

Their thesis led us to begin with CycleGAN [10]. However, we found its refinements to be inadequate and moved on to defining our own loss functions and trying out another GAN architecture, AGGAN.

1.3 Outline

In Chapter 2, we describe theory necessary for understanding our work. We begin with explaining Generative Adversarial Networks (GANs) in Section 2.1 and relevant papers pertaining to GANs in Sections 2.2-2.3. These sections serve as a foundation for understanding the GANs used in this thesis. In Sections 2.4-2.5, we then explain CycleGAN and AGGAN, the two GAN architectures used in this thesis. Section 2.6 then describes alternative GAN losses that we utilize and Section 2.7 describes GAN implementation details. At last in Section 2.9 and 2.10, we describe the object detection network SqueezeDet and our evaluation techniques.

In Chapter 3 we describe our method and approach. We begin in Section 3.1 by describing the overall strategy of our approach. Sections 3.2-3.4 then describe the details of the strategy, and at last the implementation details are described in Section 3.5.

In Chapter 4 we present qualitative and quantitative results. Chapter 5 then provides an expansive discussion of the results.

Final remarks are presented in Chapter 6 where we present our conclusion in Section 6.1. Section 6.2 then describes future work and Section 6.3 discusses the ethics of refining synthetic images of humans to look real.

1.4 Contributions

In this work we show that a refined synthetic dataset can give quantitative increases in human detection performance when used as training data, compared to a small real dataset. However, we are only able to show human detection performance increases when the real dataset is sufficiently small, meaning that it does not capture as much variance as it would have if it were as big as our refined synthetic dataset.

Further, we provide insights into different GAN losses and try to provide answers to problems we have encountered. We also include detailed algorithms and descriptions of our method to assure reproducibility of our results.

1.5 Workload distribution

All our work is the result of our combined efforts. There have been no dedicated responsibilities as such and work has been evenly distributed.

Chapter 2
Theory

2.1 Generative Adversarial Network

A Generative Adversarial Network (GAN) is a generative model that aims to approximate the distribution of some dataset [11]. There are other such approaches for modeling distributions, as well, e.g. Generative Stochastic Networks [12] or Variational Autoencoders [13], but what makes GANs unique is their adversarial nature. In this section we explain how GANs work, based on [11] and [14]. As in [15], we first try to give an intuitive understanding of the technique and then delve into the mathematical details that make it work.

Consider the common analogy of an art forger and an art critic: two natural adversaries. The forger's incentive is to produce realistic forgeries and the critic aims to distinguish authentic works from forgeries. Whenever the critic discovers a new forgery, the forger is forced to improve his technique such that the critic no longer can distinguish forgeries from authentics. The critic is thus again forced to find new ways to distinguish authentics from forgeries, and on it goes. This is the principal idea behind GANs: two adversarial players competing against each other, where one's advantage becomes a disadvantage to the other.

In order to be more exact, a GAN consists of a function G (the forger) and a function D (the art critic), both of which can be implemented as artificial neural networks [11]. G, also known as the generator, will try to model a distribution p_g which will approximate the true distribution p_{data} of which the dataset x_{real} is sampled from. To learn p_g, a prior input noise variable z is sampled from a noise distribution p_z and the mapping $G(z; \theta_g)$ maps z into the data space, where θ_g are the parameters of the neural network for the generator. $D(x, \theta_d)$, also known as the discriminator, will have data x and a parameter vector θ_d as input and it will output a scalar in the range $[0, 1]$, which represents the probability that x came from p_{data} rather than p_g [11].

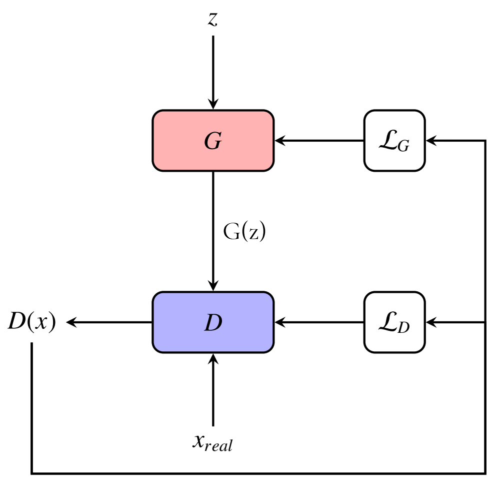


Figure 2.1: Schematic overview of a GAN. G produces generated samples $G(z)$ from the noise input z. D takes batches of generated samples $G(z)$ and real samples x_{real} and predicts whether the samples come from p_{data} or not. D's predictions are then used in the adversarial loss functions $\mathcal{L}_G$ and $\mathcal{L}_D$ to determine the weight updates of G and D.

D wants to maximize the probability of assigning correct labels to the training samples: 1 to x_{real} and 0 to generated samples from G. For D to learn this behavior, it has to optimize a loss function that reflects this: classifying generated samples as real or real samples as generated should give a high loss, while correct classifications should give a low loss. This is a standard task of binary classification where we want to distinguish two distributions and as such we can use cross entropy as our loss function. With this in mind, the discriminator loss can be derived as follows:

Given an arbitrary classification task with C classes, the cross entropy of the predicted distribution and the ground truth distribution can be expressed as

$$H(p,q) = -\sum_{i=1}^{C} p_i log(q_i)$$

where p is the distribution of ground truth labels and q is the distribution of predicted labels.

The discriminator is tasked with a binary classification, i.e. $C = 2$.

$$H(p,q) = -p_1 log(q_1) - p_2 log(q_2)$$

Since $p_1 + p_2 = q_1 + q_2 = 1$, we can write

$$H(p,q) = -p_1 log(q_1) - (1 - p_1) log(1 - q_1)$$

Denote $y : \mathbb{R}^{n \times n} \mapsto \{0, 1\}$ as a map from samples of size $n \times n$ to true binary labels, and $\hat{y}^{(i)} = q_1^{(i)}$ as the $i : th$ predicted label. Given samples N from the ground truth distribution p and samples M from the predicted distribution q, where $|N| = |M|$, the average binary cross entropy of the samples in N and M is computed as follows.

$$H(p,q) = -\frac{1}{|N|}\sum_{i}^{|N|} y(N^{(i)}) log(\hat{y}^{(i)}) - \frac{1}{|M|}\sum_{j}^{|M|}(1 - y(M^{(j)})) log(1 - \hat{y}^{(j)})$$

In the context of GANs with a discriminator D and a generator G, we see that p must be the distribution of real samples and q the distribution of generated samples. In the left sum, $\hat{y}^{(i)}$ is replaced by $D(x_{real}^{(i)})$, since N is a sample from the real distribution p. In the right sum, $\hat{y}^{(i)}$ is replaced by $D(G(z)^{(i)}))$, since M is a sample from the generated distribution q.

$$H(p,q) = -\frac{1}{|N|}\sum_{i}^{|N|} y(N^{(i)}) logD(x_{real}^{(i)}) - \frac{1}{|M|}\sum_{j}^{|M|}(1 - y(M^{(j)})) log(1 - D(G(z)^{(j)}))$$

Since $y(N(i)) = 1$ for all i and $y(M(j)) = 0$ for all j we can replace them with their actual values. Since $|N| = |M|$, the expression is simplified by replacing $|N|$ and $|M|$ with K.

$$H(p,q) = \mathcal{L}_D = -\frac{1}{K}\sum_{i}^{K} logD(x_{real}^{(i)}) - \frac{1}{K}\sum_{i}^{K} log(1 - D(G(z)^{(i)}))$$

The above can then be expressed in terms of expectations:

$$\mathcal{L}_D = -\left(\mathbb{E}_{x_{real}\sim p_{data}(x_{real})}[logD(x_{real})] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z))]\right) \tag{2.1}$$

This final expression describes a cross entropy between real and generated samples, which is used as the discriminator's loss function, where the range of $\mathcal{L}_D$ is $[0, \infty)$. It indeed encourages the discriminator to assign correct labels to x_{real} and $G(z)$ respectively, since the left term only approaches 0 when the discriminator is good at classifying real samples (assigning labels close to 1) and the right term only approaches 0 when the discriminator is good at classifying generated samples (assigning labels close to 0). Thus, in order to minimize the loss function, the discriminator must perform well in classifying samples from both distributions.

$D(x_{real})$	$D(G(z))$	Left term	Right term	L_D
0	0	$-\infty$	0	∞
1	0	0	0	0
0	1	$-\infty$	$-\infty$	∞
1	1	0	$-\infty$	∞

Table 2.1: Asymptotic values of $\mathcal{L}_D$ and the left and right terms in Eq. 2.1, given different extreme behaviors of the discriminator. As seen, the loss is minimized if and only if the discriminator is good at classifying x_{real} as real and $G(z)$ as generated.

Since the generator and discriminator are two adversaries, it makes sense to express the loss of G as

$$\mathcal{L}_G = -\mathcal{L}_D = \mathbb{E}_{x_{real}\sim p_{data}(x_{real})}[logD(x_{real})] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z))] \tag{2.2}$$

This means that the network will maximize $\mathcal{L}_D$ when trying to minimize $\mathcal{L}_G$ (and vice versa), which characterizes the zero sum game $\mathcal{L}_D + \mathcal{L}_G = 0$.

Defining the objective function to be $V(D, G) = \mathcal{L}_G$, the whole problem can be stated as a min-max game

$$\min_G \max_D V(D,G) = \min_G \max_D (\mathbb{E}_{x_{real}\sim p_{data}(x_{real})}[logD(x_{real})] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z))]) \tag{2.3}$$

According to the above, the discriminator minimizes a cross entropy whilst the generator maximizes the same cross entropy. Looking into this, we see that the generator has no control over the first term in Eq. 2.2 since it is only dependent on D discriminating on real samples. Thus we conclude that the first term is redundant. For the second term in Eq. 2.2 there is another problem regarding training: the generator's gradient will vanish when the discriminator rejects generated samples with high confidence, i.e. $D(G(z))$ is close to zero (see the lower blue curve in Fig. 2.2) [14]. Instead the generator's loss is defined as

$$\mathcal{L}_G^{new} = -\mathbb{E}_{z \sim p_z(z)}[log D(G(z))] \tag{2.4}$$

With this loss, the generator tries to maximize the log probability of the discriminator making a mistake rather than minimizing the probability of the discriminator being correct. The motivation for this is to assert that each player will have strong gradients when that player is losing the game. Consider the case where the discriminator is winning, then $D(G(z))$ will be close to zero, resulting in $\mathcal{L}_G^{new}$ to grow big (see the upper red curve in Fig. 2.2), meaning that it can not suffer from vanishing gradients. The same is not true for $\mathcal{L}_G$ when the discriminator is winning, i.e. $D(G(z))$ approaches zero. The loss will then be very small. Note that the losses no longer define a zero sum game, meaning that the optimization problem can not be described by one objective function [14].

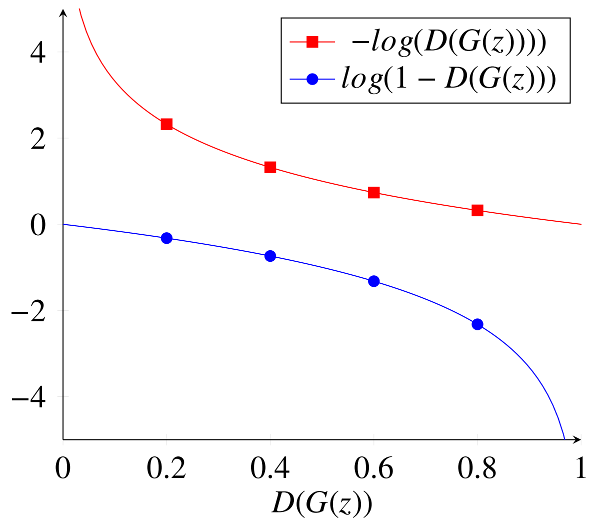


Figure 2.2: Graph demonstrating the differences in gradient for $\mathcal{L}_G^{new}$ (upper red curve) and the right term of $\mathcal{L}_G$ (lower blue curve). Note that the gradient of $\mathcal{L}_G^{new}$ is much steeper than the gradient of the right term of $\mathcal{L}_G$ when $D(G(z)) \to 0$.

With the new generator loss (Eq. 2.4), the schematic overview of the GAN in Fig. 2.1 is changed to that of Fig. 2.3.

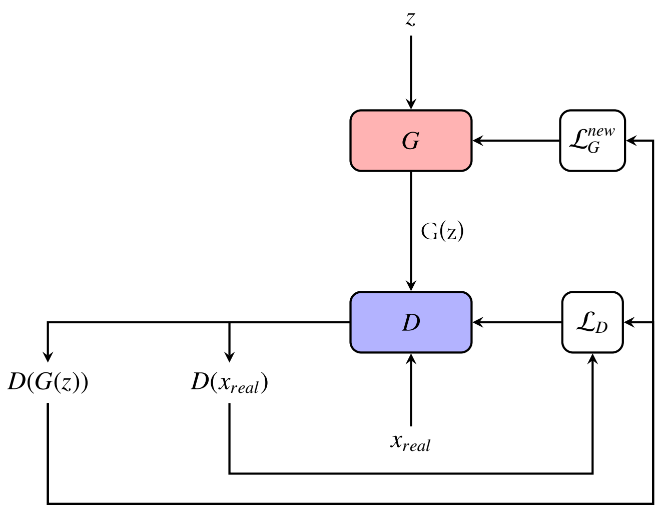


Figure 2.3: Schematic overview of a GAN with the improved generator loss $\mathcal{L}_G^{new}$. G produces generated samples $G(z)$ from the noise input z. D takes batches of generated samples $G(z)$ and real samples x_{real} and predicts whether the samples come from p_{data} or not. D's predictions are then used in the adversarial loss functions $\mathcal{L}_G^{new}$ and $\mathcal{L}_D$ to determine the weight updates of G and D.

Algorithm 1 Training of a GAN using stochastic gradient descent

```
1:  for number of train iterations do
2:      for number of times to train the discriminator do
3:          Sample $K$ noise samples $\{z^{(1)}, \ldots, z^{(K)}\}$ from $p_z$
4:          Generate $K$ samples $\{G(z^{(1)}), \ldots, G(z^{(K)})\}$ from $p_g$
5:          Sample $K$ real samples $\{x^{(1)}, \ldots, x^{(K)}\}$ from $p_{data}$
6:          Minimize the discriminator loss with respect to the weights $\theta_D$ using stochastic
    gradient descent:
                $\min_{\theta_D}\{-\frac{1}{K}\sum_i^K logD(x_{real}^{(i)}) - \frac{1}{K}\sum_i^K log(1 - D(G(z)^{(i)}))\}f$
7:      end for
8:      Sample $K$ noise samples $\{z^{(1)}, \ldots, z^{(K)}\}$ from $p_z$
9:      Generate $K$ samples $\{G(z^{(1)}), \ldots, G(z^{(K)})\}$ from $p_g$
10:     Minimize the generator loss with respect to the weights $\theta_G$ using stochastic gradient
    descent:
                $\min_{\theta_G}\{-\frac{1}{K}\sum_i^K log(D(G(z)^{(i)}))\}$
11: end for
```

2.2 Conditional Generative Adversarial Network

Like a GAN, a conditional GAN is trained on samples x, but with additional information y (e.g. classes). This additional information narrows the output space and therefore helps both adversaries in their efforts to overcome the other: the generator is told what information y its generated sample should have, and the discriminator is told what information y the generator claims its generated sample has [16].

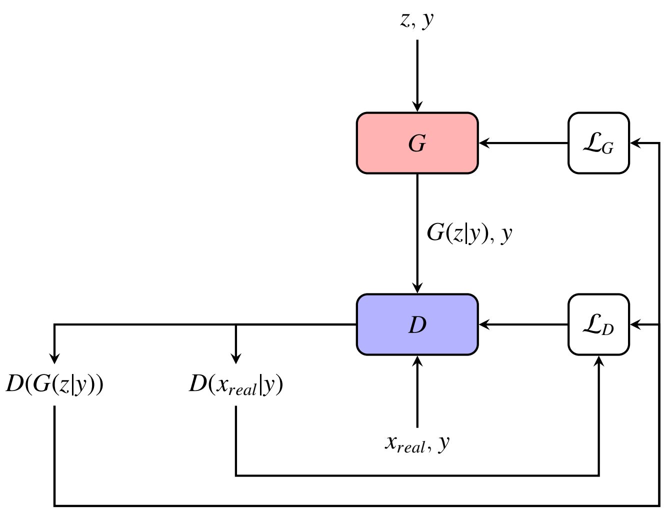


Figure 2.4: Schematic overview of a conditional GAN. G produces generated samples $G(z|y)$ from the noise input z and the conditional label y. D takes batches of generated samples $G(z|y)$ with the corresponding labels y and real samples x_{real} with corresponding labels y and predicts whether the samples come from p_{data} or not. D's predictions are then used in the adversarial loss functions $\mathcal{L}_G$ and $\mathcal{L}_D$ to determine the weight updates of G and D.

The conditional adversarial losses of the discriminator and the generator are almost identical to the adversarial losses of the original GAN. The only difference is that they are conditioned on y [16].

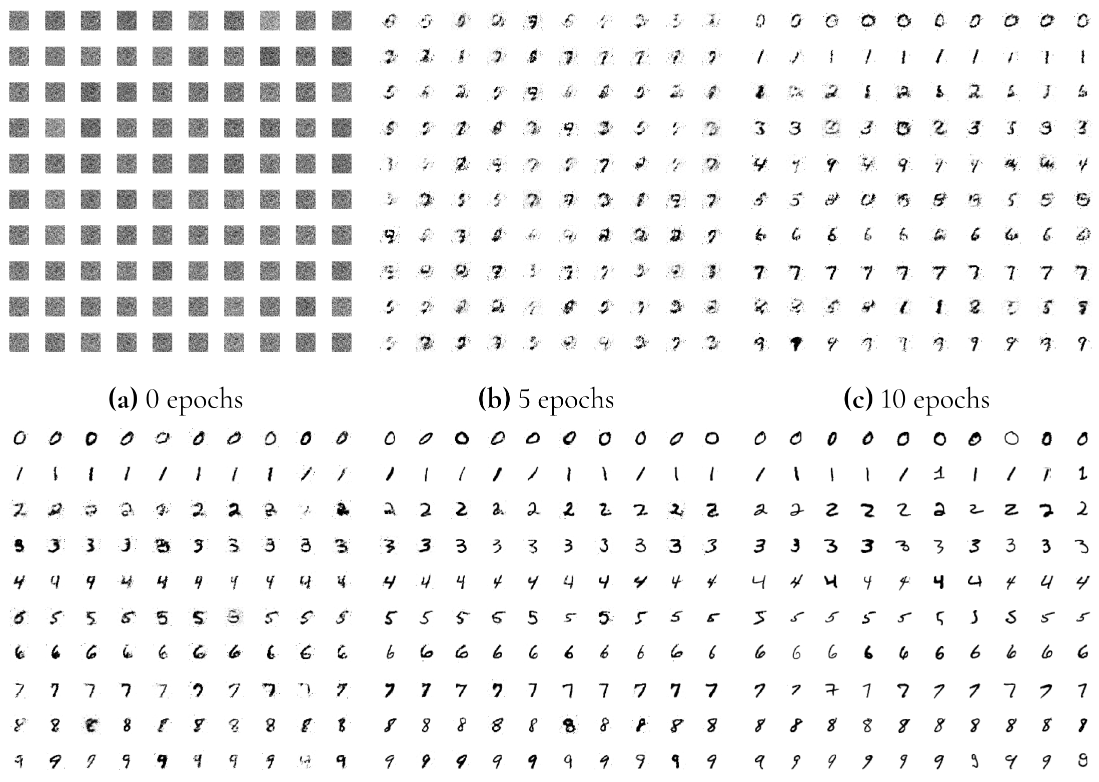
$$\mathcal{L}_D = -\left(\mathbb{E}_{x_{real}\sim p_{data}(x_{real})}[logD(x_{real}|y)] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z|y))]\right) \tag{2.5}$$

$$\mathcal{L}_G = -\mathbb{E}_{z\sim p_z(z)}[logD(G(z|y))] \tag{2.6}$$

It is widely considered a good approach to use conditional information, if available, when training GANs since it by empirical evidence greatly improves the subjective quality of generated samples. It is however not concluded if the generator's approximation of the true distribution actually becomes more accurate, or simply whether it becomes more biased towards properties that the human perception focuses on [14, p. 30].

Fig. 2.5 shows samples generated by a conditional GAN trained on the MNIST dataset [17]. Before any training, the generated samples are only noise, since the generator has not adapted its weights at all. However, after a few epochs of training, results begin to show.

When comparing Fig. 2.5e and Fig. 2.5f, one can conclude that the network can successfully generate at least some samples that look real.

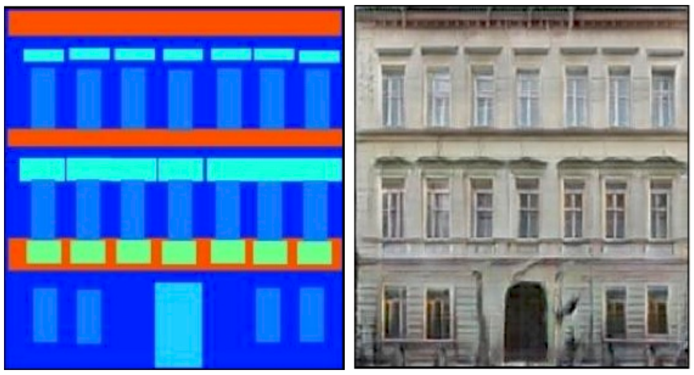


(a) 0 epochs **(b)** 5 epochs **(c)** 10 epochs

(d) 15 epochs **(e)** 75 epochs **(f)** real samples

Figure 2.5: (2.5a-2.5e): Generated samples from a conditional GAN trained on the MNIST dataset [17], where each row is conditioned on one class. (2.5f): real samples from the MNIST dataset.

2.3 Supervised Image-to-Image Translation: Pix2Pix



(a) Input **(b)** Output

Figure 2.6: Image translation by Pix2pix trained on translating semantic segmentation maps of facades to images of facades, as presented in original paper [18].

Presented by Isola *et al.*, Pix2Pix is a variation of a conditional GAN that can perform supervised image-to-image translation [18]. By providing paired input and output data, e.g. pictures of facades and corresponding semantic segmentation maps of the facade details, the network is able to translate images from one domain X to the other domain Y (See Fig. 2.6).

For each conditional image $x \in X$ there exists a paired output image $y \in Y$. In addition to the adversarial loss, a new loss is defined which computes the $L1$ norm of the difference between the transformed input $G(x, z)$ and the output y

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[||y - G(x, z)||_1] \tag{2.7}$$

where z is the input noise vector.

Another novel feature in Pix2Pix is the generator architecture. In image-to-image translation it is a common desire to translate colors and textures, whilst preserving structure. Therefore, an encoder-decoder generator network with skip connections is encouraged, allowing features that are shared between input and output domains to be shared directly through these connections. For example, in Fig. 2.6 the two images share structure but not texture, meaning that we want to draw a connection from the layers detecting structural features to our output. Pix2Pix uses the U-Net architecture [19] where each encoder layer is connected with their corresponding decoder layer (see Fig. 2.7).

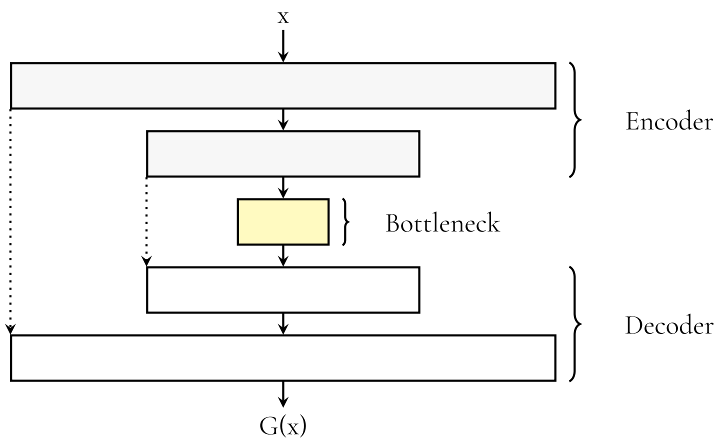


Figure 2.7: The conceptual architecture of the U-Net generator, where x is the input image and $G(x)$ image translation of x. The dotted arrows represent the skip connections between the intermediate layers. The upper blocks represent the encoder, the middle the bottleneck layer and the lower blocks the decoder.

A common problem in GANs are blurry generator samples. To alleviate this, Isola *et al.* introduced PatchGAN, a discriminator architecture [18]. PatchGAN allows the discriminator to discriminate on local patches of size $N \times N$ of the image and thus capturing high frequencies. It will only penalize structure at the scale of patches and run convolutionally across the image, averaging all of its responses to produce a final output of the discriminator.

Varying the patch size N will have a great impact on the generated samples. Setting $N = 1$ one achieves a so-called "PixelGAN" resulting in blurriness (see Fig. 2.8), but encourages greater color diversity. $N = 16$ results in sharp images, with tiling artifacts. For $N = 70$, there is a good trade-off between producing sharp images and avoiding tiling artifacts. Scaling beyond this does not seem to improve the visual quality and thus makes the discriminator easier to train due to the decreased number of connections [18].

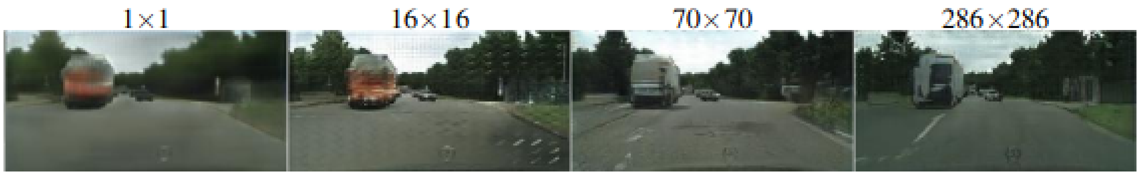
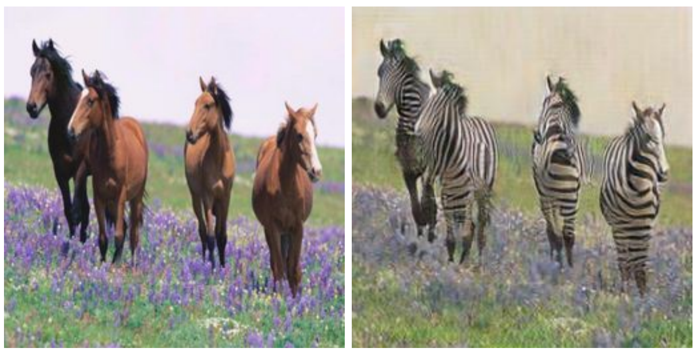


Figure 2.8: Comparison of different patch sizes for the discriminator , as presented in the original paper [18].

2.4 Unsupervised Image-to-Image Translation: CycleGAN



(a) Input **(b)** Output

Figure 2.9: Image translation by CycleGAN trained on translating horses to zebras and zebras to horses.

We have now gone through the necessary material for understanding CycleGAN, one of the GAN architectures used in this thesis. In this section we describe how it works, as explained in Zhu *et al.*'s original paper [10].

CycleGAN is a GAN that performs image-to-image translation with unpaired images. CycleGAN is fed two domains of images, e.g. horses and zebras, and is trained to translate each domain to the other.

Unlike other GANs so far mentioned, CycleGAN's architecture is cyclic: Given two domains of images, X and Y, CycleGAN trains two generators, $G : X \mapsto Y$ and $F : Y \mapsto X$. There are then two discriminators D_X and D_Y, where D_X is trained to distinguish $x \in X$ and $F(y)$ where $y \in Y$, and D_Y is trained to distinguish $y \in Y$ and $G(x)$ where $x \in X$.

The most novel feature of the network is its cycle-consistency loss. The loss is meant to "capture the intuition that if one translates from one domain to the other and back one should arrive at where one started" [10, p. 3]. In other words, after an image has gone through a cycle in the network the output should still be consistent with the input, i.e. $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ and $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. More formally, the cycle-consistency loss can be expressed as shown in Eq. 2.8 [10].

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data(x)}}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim p_{data(y)}}[||G(F(y)) - y||_1] \tag{2.8}$$

Since the cycle-consistency loss is a pixel-wise reconstruction loss, it will grow big if there were to be any large changes in position of features e.g. morphing edge shapes, preventing it from reconstructing the original input image. In order to keep a low pixel-wise reconstruction loss the generators are encouraged to only change the values of pixels i.e. color and as a result the transformations retains shape. This is also why CycleGAN is not able generalize to translation tasks where the domains have different shape, e.g. transforming dogs to cats [10].

The full objective, shown in Eq. 2.9, is then the sum of the adversarial losses for G and F and the cycle consistency loss (Eq. 2.8), regulated by λ.

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{adv}(G, D_Y, X, Y) + \mathcal{L}_{adv}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F) \tag{2.9}$$

However, the authors found that this loss can cause unstable training and blurry images. To remedy this, they replaced the cross entropy loss with a least-squares loss. Then $\mathcal{L}_{adv}$ is defined so that G minimizes $\mathbb{E}_{x \sim p_{data}(x)}[(D(G(z)) - 1)^2]$ and D minimizes $\mathbb{E}_{y \sim p_{data}(y)}[(D(y) - 1)^2] + \mathbb{E}_{x \sim p_{data}(x)}[(D(G(x))^2]$

Just like Pix2Pix, CycleGAN uses a 70×70 patch discriminator. However, the structure of the generator is different in CycleGAN. Instead of using U-Net, the authors of CycleGAN has adopted a generator architecture from [20] . See Chapter 2.7 for a more detailed description.

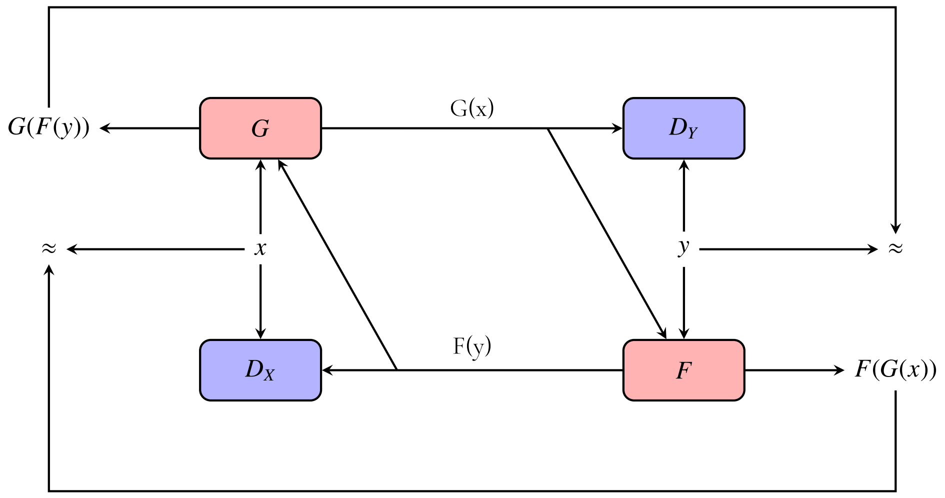
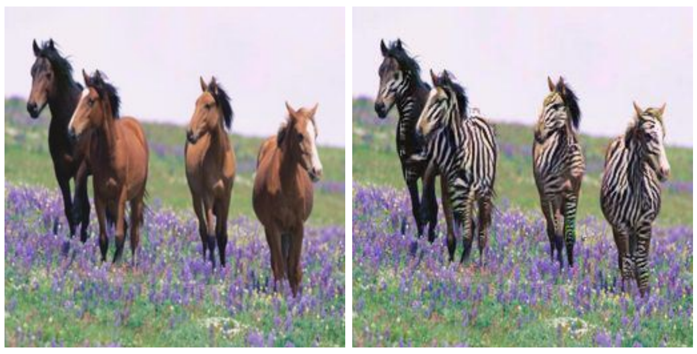


Figure 2.10: Schematic overview of CycleGAN. The generators G and F try to fool the discriminators D_Y and D_X, while still making sure that the cyclic transformations $G(F(y))$ and $F(G(x))$ are similar to y and x.

2.5 Unsupervised Attention-guided Image-to-Image Translation



(a) Input **(b)** Output

Figure 2.11: Image translation by AGGAN trained on translating horses to zebras and zebras to horses.

The second GAN architecture used in this thesis is AGGAN: an attention-guided GAN. In this section we describe how it works, as explained in Mejjati *et al.*'s original paper [21].

Comparing the input and output in Fig. 2.9, one sees that CycleGAN succeeds in translating horses into zebras. However, the background colors have also changed and even lost detail. To mitigate this problem one would like to exclusively perform the style transfer on the foreground, in this case the horses, such that original background information is retained. This can be made possible with AGGAN. The approach is based on CycleGAN but is extended by adding attention networks that select areas to style transfer by maximizing the probability that the discriminator makes a mistake, i.e. selecting the best area such that the image resembles an image from the target dataset. Denote $A_X : X \to X_a$ and $A_Y : Y \to Y_a$ as networks producing attention maps from the datasets X and Y respectively. Each attention map contains values in the range $[0, 1]$ for each pixel. These attention maps can be thought of as continuous masks filtering out relevant parts of the image. The following paragraph will explain how the attention maps operate.

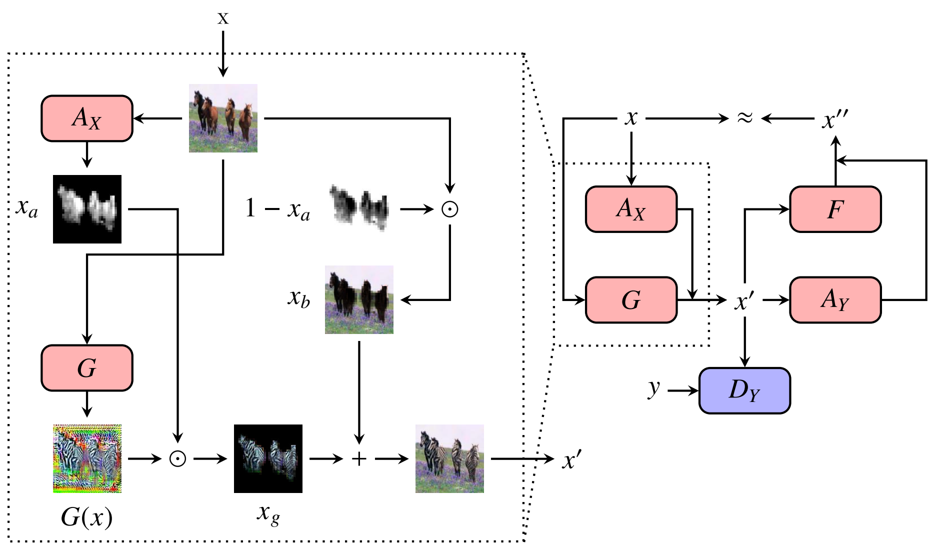


Figure 2.12: Schematic overview of the attention-guided GAN in one direction, as outlined in the original paper [21]. Aside from the attention networks and the operations detailed on the left side of the figure, the overall structure is the same as in CycleGAN. Note that F and A_Y are symmetrical to G and A_X.

Due to symmetry it is sufficient to describe the algorithm in one direction, namely $G : X \to Y$. Note that the following explanation is also illustrated in Fig. 2.12. Given an image $x \in X$, an attention map x_a and an inverse attention map $1 - x_a$ are created. The image is then fed to the generator G yielding $G(x)$. After this, x_a is applied to $G(x)$ yielding x_g (the foreground of $G(x)$) and $1 - x_a$ is applied to x yielding x_b i.e. the background of x. One could interpret x_g as the style transferred foreground of the input image and x_b as the input image where the foreground is removed. Finally x_g and x_b are added together resulting in a full image x' where only the foreground has been style transferred. Defining $\odot$ as an element-wise product, the algorithm is summarized as

$$x' = x_a \odot G(x) + (1 - x_a) \odot x \tag{2.10}$$

The adversarial loss is similar to CycleGAN's except that the discriminator will discriminate on x' rather than $G(x)$

$$\mathcal{L}_{adv}^{x}(G, A_x, D_Y) = \mathbb{E}_{y\sim p_Y(y)}[logD_Y(y)] + \mathbb{E}_{x\sim p_X(x)}[log(1 - D_Y(x')] \tag{2.11}$$

Similar to CycleGAN a cyclic-loss is defined

$$\mathcal{L}_{cyc}^{x}(x, x'') = \mathbb{E}_{x\sim p_X(x)}[||x - x''||_1] \tag{2.12}$$

where x" is obtained by applying F and A_Y on x', similarly to Eq. 2.10. Combining the adversarial and cycle-consistency losses for both the source and target domains the total loss is obtained:

$$\mathcal{L}_{tot}(G, F, A_X, A_Y, D_X, D_Y) = \mathcal{L}_{adv}^{x} + \mathcal{L}_{adv}^{y} + \lambda_{cyc}(\mathcal{L}_{cyc}^{x} + \mathcal{L}_{cyc}^{y}) \tag{2.13}$$

where λ is the weight for the cyclic loss.

A problem with this loss is that after some training the discriminator will begin to discriminate on differences in background instead of differences in foreground (since the background for horse and zebra images are not the same), leading to the attention maps converging to one (i.e the attention network attends the whole image). To prevent this, the authors stop the training of the attention network A_X after 30 epochs and one trusts that the attention maps have converged to correctly attend the foreground.

Another adjustment, made by the authors, to the algorithm after the 30:th epoch, is that D_Y will discriminate on the foreground instead of the transformed image x'. The most obvious way to do this is to let D_Y discriminate on x_g. However, this would pose a problem since then the discriminator would receive small pixel values (each pixel should be a relatively "strong" pixel from either background or foreground). Instead, the attention map x_a is thresholded to only contain pixel values greater than the transition rate τ, which is set to 0.1. The discriminator now discriminates on x'_{new} and y_{new} (see Eq. 2.14 and 2.15)

$$x'_{new} = \begin{cases} G(x), \; if A_X(x) > \tau \\ 0, \; otherwise \end{cases} \tag{2.14}$$

$$y_{new} = \begin{cases} y, \; if A_Y(y) > \tau \\ 0, \; otherwise \end{cases} \tag{2.15}$$

where x'_{new} and y_{new} are masked versions of x' and y respectively. The adversarial loss now becomes

$$\mathcal{L}_{adv}^{x}(G, A_x, D_Y) = \mathbb{E}_{y\sim p_Y(y)}[logD_Y(y_{new})] + \mathbb{E}_{x\sim p_X(x)}[log(1 - D_Y(x'_{new}))] \tag{2.16}$$

2.6 GAN losses

When training GANs one is not always satisfied with only having an adversarial loss. Sometimes one wants to impose restrictions on the GAN. In this case, defining additional losses can be of use. The choice of these additional loss function is often dependent on the application i.e. the dataset. Below we present some of these additional losses that we use in our work.

Identity loss

In CycleGAN [10], the cyclic losses force the transformations $G(x)$ and $F(y)$ to be reversible. That is, there are no losses restricting $G(x)$ (besides the adversarial loss) as long as F is able to reverse G's transformation, $F(G(x)) \approx x$, and vice versa. For some datasets this could give the generators more freedom than desired. One solution is to define a loss that penalizes changes between the input of a generator and its output - known as an identity loss. In its simplest form, it can be defined as the pixelwise L1 norm. Given a generator G it would be defined as seen below.

$$\mathcal{L}_{ID} = \mathbb{E}_{x \sim p_{data}(x)}[||G(x) - x||_1] \tag{2.17}$$

Due to the adversarial loss, G would still try to approximate its target datasets, but it would now be restrained as to not deviate too far from the source dataset. Since the penalization is pixelwise and pixels represent color, this loss would make an effort to preserve color. This is demonstrated in Fig. 2.13.

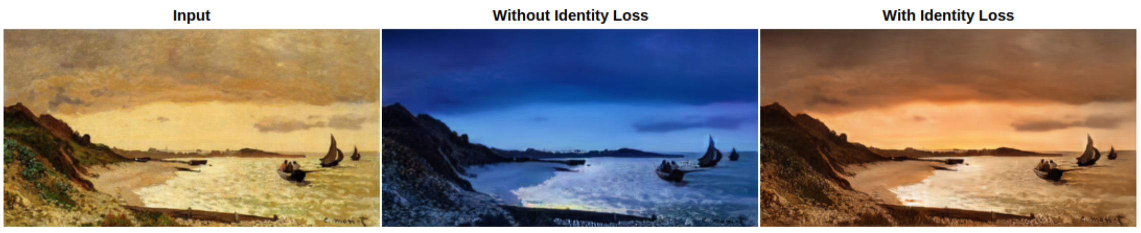


Figure 2.13: Effect of using an identity loss on a Monet to photo mapping using CycleGAN, as shown in [10].

Perceptual loss

When working with images it often makes sense to characterize an image in terms of features, such as edges, corners, contrast, or different objects. This makes it possible to describe images in a more abstract space than just representing them with pixels. These image features are also known as perceptual features, since they are supposed to capture what humans perceive in images. For example, when adjusting the value of the contrast feature of an image, a human will be able to perceive the transformation. However, when adjusting the value of a single pixel, it is not evident that a human will register this change.

Summarily, perceptual image features give us a method of abstractly expressing the defining characteristics of an image.

A perceptual loss function penalizes changes in perceptual image features between an input image and an output image. In computer vision tasks, it is common to utilize the feature space of VGG16 [22], a convolutional neural network trained on ImageNet, as first done by Johnson *et al.* [20]. The feature space is a multidimensional space of feature vectors, i.e. vectors comprising feature representations of images. The idea is that similar images produce similar feature vectors and therefore have a short euclidean distance between them. Thus, given the assumption that our features can accurately represent images, it is a good measure of image similarity.

In the context of image-to-image translation, we use this loss to preserve features of the input images. For an architecture like CycleGAN, it can be utilized as either an identity loss or as a cyclic loss.

Cyclic VGG16 loss

[23] proposed an extended CycleGAN network with a cyclic perceptual loss. This was done by adding an additional cyclic loss in Eq. 2.9 with the same structure as Eq. 2.8, but modified to measure distance in VGG16 feature space instead (See Eq. 2.18 below), where V is a map that takes an image and produces the VGG16 feature representation of its input:

$$\mathcal{L}_{CycVgg16}(G,F) = \mathbb{E}_{x\sim p_{data(x)}}[||V(F(G(x))) - V(x)||_2 + \mathbb{E}_{y\sim p_{data(y)}}[||V(G(F(y))) - V(y)||_2] \quad (2.18)$$

In Eq. 2.18, feature vectors from the second and the fifth max pooling layer of VGG16 are used. Since second max pooling layer filters detect simpler low level shapes and patterns such as circles and stripes, and the fifth max pooling layer filters detect more complex features such as faces and bodies [24], this forces the network to preserve both low and high-level features.

Identity VGG16 loss

Another application of the perceptual loss would be to use it in an identity loss between the input image x and the generated image $G(x)$

$$\mathcal{L}_{IdVgg16}(x,G) = \mathbb{E}_{x\sim p_{data(x)}}[||V(G(x)) - V(x)||_2 \quad (2.19)$$

This forces the generated image $G(x)$ to be similar to the input image x in VGG16 feature space. One might think that this restriction is too aggressive, resulting in no transformation at all. This is the case if all the image features from each max pooling layer of the VGG16 network are included, but one can decide whether to use deep or shallow layers depending on whether one wants to penalize changes of high level features or changes of low level features.

Aside from which layers are used, the aggressiveness of this loss naturally depends on its relative weight compared to other losses.

Mean feature VGG16 loss

A third way to use perceptual differences is to compute the mean feature vector of the target dataset in VGG16 space and try to force each generated image to have similar features to the mean feature vector. Using an identity loss we can express this idea as follows.

$$\mathcal{L}_{mean}(G,Y) = \mathbb{E}_{x\sim p_{data}(x)}[||V(G(x)) - V(\mathbb{E}_{y\sim p_Y(y)}[Y])||_2] \quad (2.20)$$

Discriminator feature cyclic loss

Since a perceptual loss is not tied to any specific feature space, we could also use the feature space of the discriminator. We define a cyclic loss for the activations coming from the $j:th$

layer in the discriminator as follows.

$$\ell^j_{CycDisc}(G,F) = \mathbb{E}_{x\sim p_{data(x)}}[||d_j(x) - d_j(F(G(x)))||_2 + \mathbb{E}_{y\sim p_{data(y)}}[||d_j(y) - d_j(G(F(x)))||_2] \quad (2.21)$$

From this the total loss is computed by summing over the different activations from the layers

$$\mathcal{L}_{CycDisc}(G,F) = \sum_{j=1}^{N} \ell^j_{CycDisc}(G,F) \quad (2.22)$$

Penalizing the network based on the discriminator's perceptual differences is free of charge since no additional nodes or weights need to be added, but unlike VGG16 the discriminator is not pretrained.

While a perceptual loss in the feature space of VGG16 helps retain all features, a perceptual loss in the feature space of the discriminator helps to retain the features that are characteristic of the dataset, since that is what the discriminator's activations are trained to detect. One drawback of course is that the feature space of the discriminator is changing throughout the training [25].

2.7 Architecture of Image-to-Image Translation networks

Minibatch discrimination

Since the discriminator lacks memory, it has no way of recalling how previously seen generated samples look. This means that the discriminator can forget different features that characterize the generated distribution, which the generator then can take advantage of.

To combat this, Shrivastava *et al.* introduced minibatch discrimination [2]. Instead of only discriminating on the latest generated samples, the discriminator discriminates on a random sample of recent generated samples. This provides the discriminator with a limited memory of how recent generated images look and limits the generator's ability of re-introducing artifacts that previously fooled the discriminator. Note that real samples are not handled this way, since that distribution is constant. There is therefore no need for the discriminator to have any memory of previously seen real samples.

This method is not to be confused with simply having a batch size larger than one. In that case, the batch of generated samples would come from the same generator and would not provide historical information. In the case of minibatch discrimination, the sampled images come from different generators.

This method was also implemented in CycleGAN, with a minibatch size of 50 [10] (and by extension in AGGAN [21]).

Fractionally strided convolutions

In image processing there is often a desire to reverse the process of an image convolution. Since a convolution reduces the dimension of its input it can not be an injective function and therefore no unique inverse can exist. However, one can create a one-to-many mapping by creating a pseudo-inverse. One approach is to use fractionally strided convolutions. It is quite similar to a normal convolution in terms of operations, but instead of down-sampling the input, it is instead up-sampled. A fractionally strided convolution will still perform a convolution on the image, but it will do so on a zero padded one.

Consider a trivial example with an input image of dimension 2×2. To perform a fractionally strided convolution one pads the input with a 2×2 zero padding, convolves this with a $3x3$ kernel with stride 1 and one ends up with an up-sampled image with dimension $4x4$ (see Fig. 2.14).

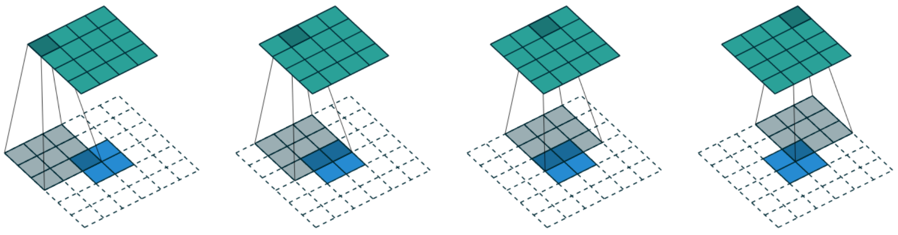


Figure 2.14: Demonstration of a fractionally strided convolution, as illustrated in [26].

In the context of GANs, this is desired in generators with an autoencoder-like architecture, where the input image is down-sampled into a feature vector with repeated convolutions and then upsampled with repeated fractional convolutions to produce an image. This was introduced by [20], whose autoencoder structure is used in GANs such as CycleGAN [10]. In their paper [20], they reason that instead of using a static upsampling function, e.g. bicubic interpolation, a fractionally strided convolution is preferable since its transformation is learnable.

Generator skip connections

In many image-to-image translation tasks one wants to preserve the structure of the input image. As mentioned in Section 2.6, this can be achieved using an identity loss. Another way to force the generator to preserve input image features is to use skip connections, as done with U-Net in [18] where encoder layer outputs are fed forward and concatenated with decoder layers. However, simpler solutions exist as well such as the one in [27], where an additive skip connection is added between the input and output layer of the generator.

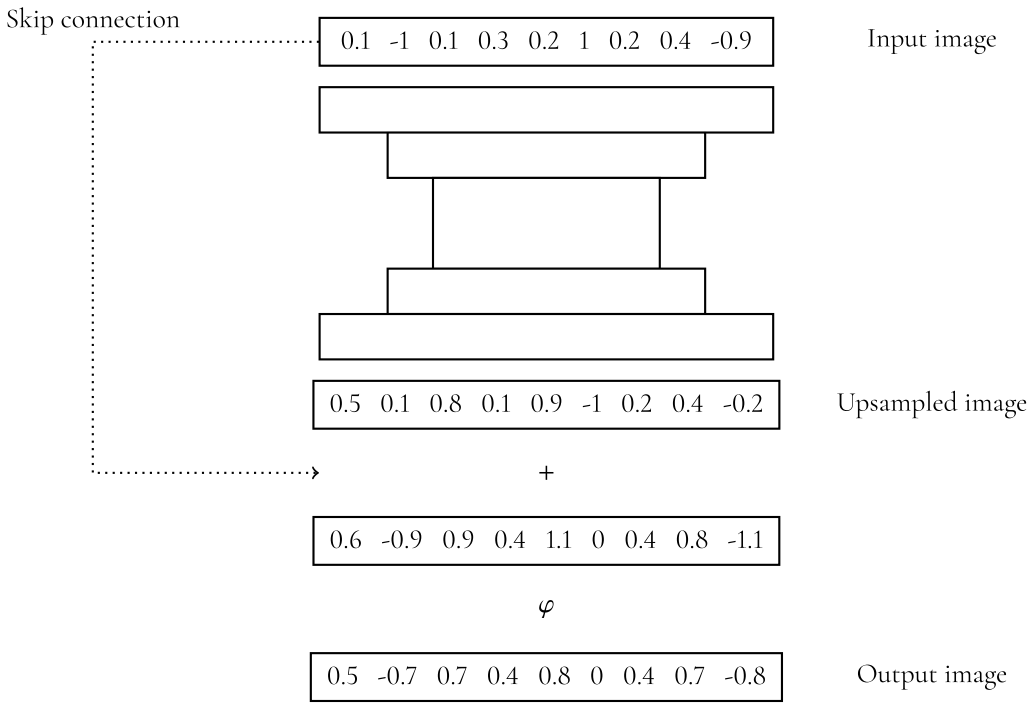


Figure 2.15: Conceptual architecture of a generator with a skip connection between the input and output layer.

Residual network

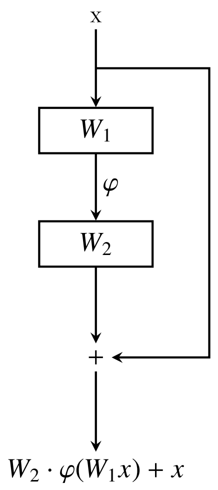


Figure 2.16: A ResNet building block. W_i are the weights of each respective layer and φ is the activation function.

Proposed by He *et al.*, ResNet [28] is a neural network structure built of components (ResNet blocks) as seen in Fig. 2.16. These ResNet blocks allow deeper networks without introducing degradation in training accuracy. Consequently they are able to increase learning, since they allow gradients to take shortcuts, making them less likely to decay when being backpropagated.

Generator

The generator architecture in CycleGAN is borrowed from Johnson *et al.*'s generator [20] and comprises three parts: an encoder, a transformer, and a decoder. The purpose of the encoder is to represent the input image in a condense space with high level feature maps. This is done using a reflection padding and three convolutional layers. The encoded result is then sent to the transformer. This is where the actual transformation of the image takes place, but it is done in the high level encoded space. The transformer is composed of a series of nine ResNet blocks each having 3×3 filters. Then this transformation is fed into the decoder which task is to revert the feature maps back into image space. This is done using two fractionally strided convolutions, a reflection padding and a convolution. A detailed description of each layer can be found in Table 2.2 and an overview of the architecture in Fig. 2.17.

Layer type	Filter size	#Filters	Stride
Reflection padding	-	-	-
Convolution-InstanceNormReLU	7X7	64	1
Convolution-InstanceNormReL	3X3	128	2
Convolution-InstanceNormReLU	3X3	256	2
9 ResNet blocks	-	-	-
Fractional-strided-Convolution-InstanceNorm-ReLU	3X3	128	$\frac{1}{2}$
Fractional-strided-Convolution-InstanceNorm-ReLU	3X3	64	$\frac{1}{2}$
Reflection mpadding	-	-	-
Convolution-InstanceNormReLU	7X7	3	1

Table 2.2: Generator architecture of CycleGAN [10].

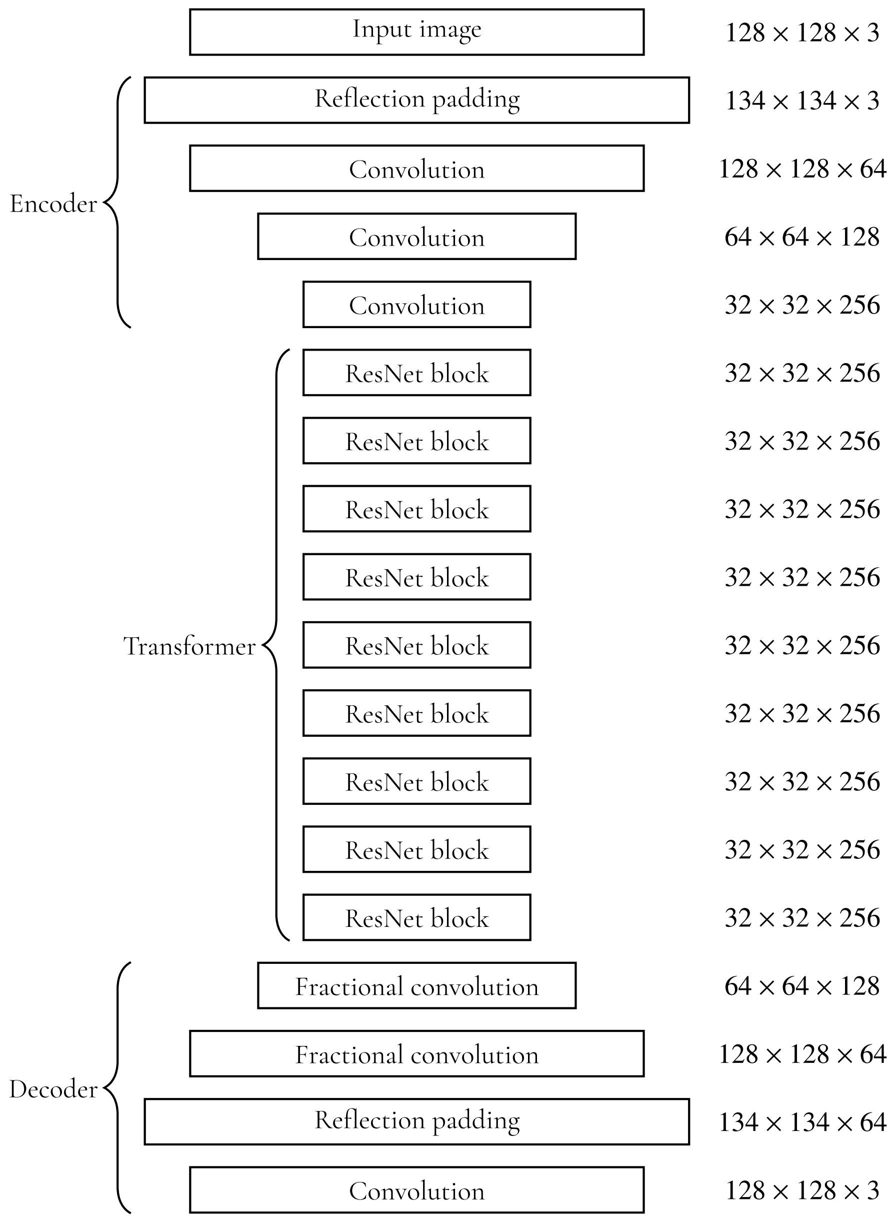


Figure 2.17: The generator architecture of CycleGAN [10]. On the right hand side of the figure, we see the dimension of each layer output.

Discriminator

Table 2.3 lists the different layers of the discriminator described in Section 2.3. A more intuitive image is also shown in Fig. 2.18.

Layer type	Filter size	#Filters	Stride
Convolution-LeakyReLU	4X4	64	2
Convolution-InstanceNorm-LeakyReLU	4X4	128	2
Convolution-InstanceNorm-LeakyReLU	4X4	256	2
Convolution-InstanceNorm-LeakyReLU	4X4	512	2
Convolutional layer that maps to a 1-dimensional output	-	-	-

Table 2.3: Discriminator architecture of CycleGAN [10].

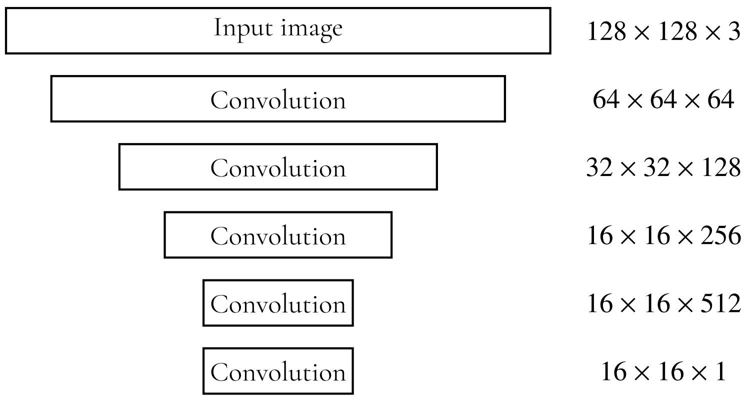


Figure 2.18: The discriminator architecture of CycleGAN [10]. On the right hand side of the figure, we see the dimension of each layer output.

2.8 How we utilize GANs

In our thesis, we use CycleGAN [10] and AGGAN [21] to refine our synthetic images.

CycleGAN was chosen due to its unsupervised approach and success with image-to-image translation where the domain shapes are the same. Since our synthetic humans have motion capture poses captured from real humans (explained in Section 3.2), we assume that the synthetic and real domains have similar enough shape in order for this to work. In addition, CycleGAN was also used by Wessman and Andersson in [1] with moderate success.

However, with CycleGAN we found two severe limitations for our application: 1) background was transformed heavily; and 2) synthetic humans were distorted. To solve the first problem of background transformation, we tried using AGGAN since that is the exact purpose for which it was introduced. To solve the second problem of distortion, we introduced additional losses (described in Section 2.6) in order for the synthetic humans to retain structure and not distort.

Theory described in Sections 2.1-2.3 and Section 2.7 is provided to improve the reader's understanding of CycleGAN and AGGAN.

2.9 Object detection: SqueezeDet

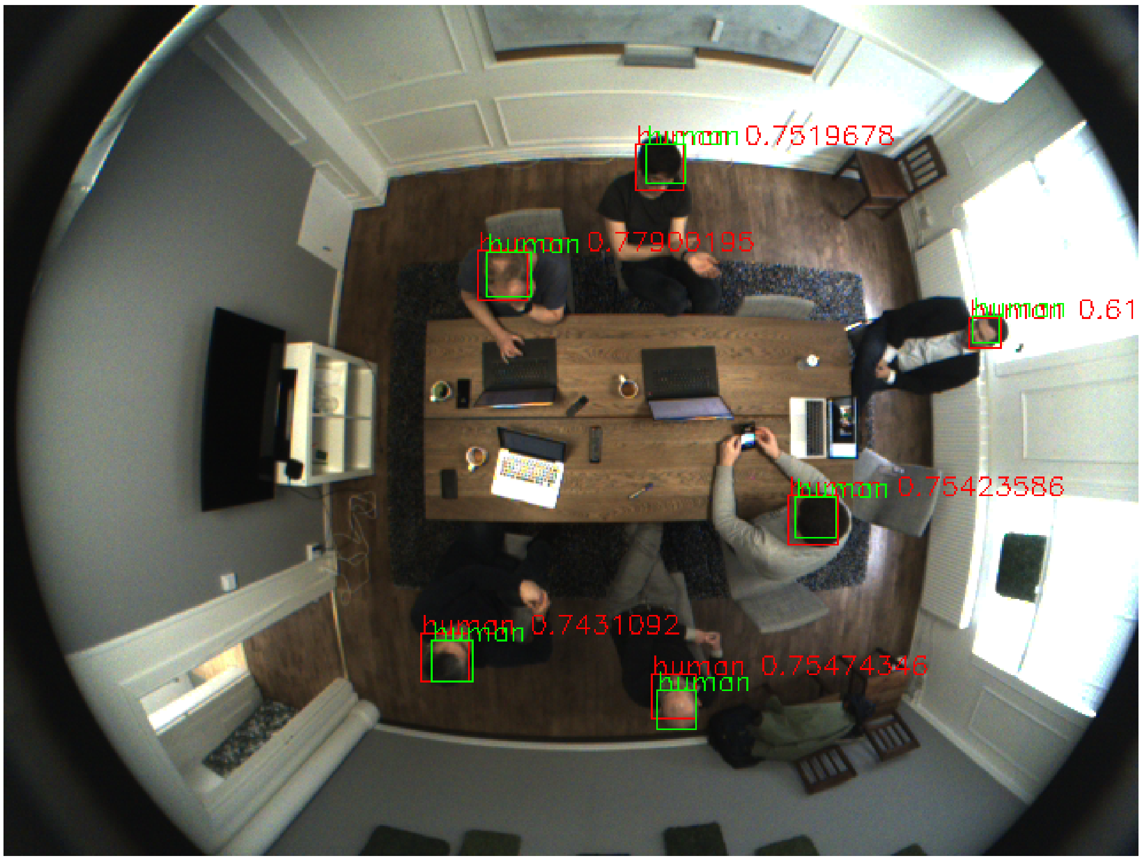


Figure 2.19: SqueezeDet detections. Boxes in green are ground truth labels. Boxes in red are the network's predictions, along with the respective predicted confidence scores.

To measure whether our refined synthetic data can replace real human head detection training data, we need an object detection network. Chosen for this task is SqueezeDet [3].

SqueezeDet is a neural network architecture that takes images as input and produces labeled bounding boxes as output. It comprises three main components: a convolutional neural network that extracts a feature map; a detection layer which takes the feature map and produces bounding boxes and label predictions; and finally a filter that sifts through the redundant labeled bounding boxes and outputs the final detections.

While there are many architectures that can be used for object detection, SqueezeDet is particularly well-suited for embedded applications. The final network size is less than 5 MB, and the inference speed is fast enough for real time inferences. When published, it also performed on par with other state-of-the-art architectures many times larger in network size and slower in inference speed.

The loss function of the network is comprised of three parts: 1) a bounding box regression which maximizes the spatial accuracy of predicted bounding boxes; 2) a confidence score regression that maximizes the probability of a predicted bounding box containing an annotated object; and 3) a class cross entropy which minimizes misclassifications.

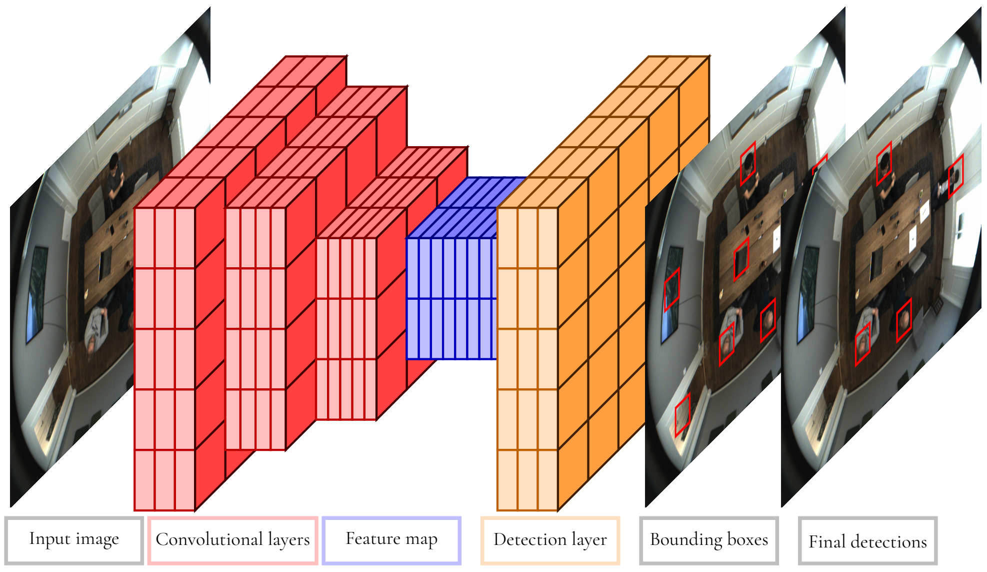


Figure 2.20: Schematic overview of SqueezeDet. The image is fed through a convolutional neural network which produces a feature map of the image. The detection layer then predicts anchors for each spatial position of the feature map. These anchors are then translated back to the coordinate system of the original image and filtered to produce the final detections.

In our case, we have one class since we wish to identify human heads only. In practice, our loss is thus comprised solely of the bounding box and confidence score regressions.

In order to predict bounding boxes, predefined bounding box dimensions are defined manually and given to the network. These are called anchors. The detection layer convolves the feature map and predicts for each position on the feature map if there any anchors that are likely to contain a human head. The feature map predictions are then translated to the original dimensions of the input image. Overlapping predictions are then filtered with Non-Maximum-Suppression and predictions with a low confidence score are removed. What remains are the final predictions of the network.

2.10 Evaluation metrics

Binary classification metrics

To evaluate the quality of a given binary classification model, one can measure its precision and recall. Precision measures the accuracy of all positive predictions. This is done by normalizing the number of true positives (TP) with the sum of the number of true positives and false positives (FP), as seen in Eq. 2.23. Recall measures how many of the true positives were positively predicted. This is done by normalizing the number of true positives with the sum of the number of true positives and false negatives (FN), as seen in Eq. 2.24.

$$p = \frac{TP}{TP + FP} \tag{2.23}$$

$$r = \frac{TP}{TP + FN} \tag{2.24}$$

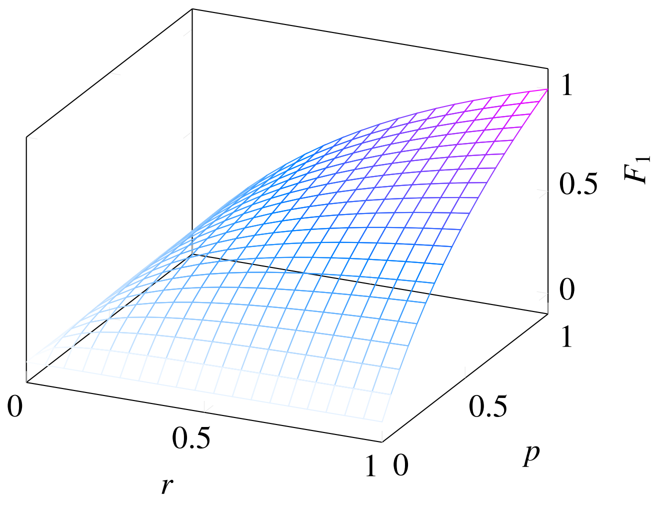


Figure 2.21: Harmonic mean of r and p (the F_1-score), as seen in Eq. 2.25.

There exists an inherent trade-off between these metrics, where it is hard to gain an increase in one metric without a decrease in the other [29]. In order to find a model with relatively good performance in each metric, one can use the harmonic mean between p and r as a metric instead. This is known as the F_1-score (Eq. 2.25).

$$F_1 = \frac{2 \cdot r \cdot p}{r + p} \tag{2.25}$$

As seen in Fig. 2.21, the F_1-score approaches 1 if and only if both p and r approach 1. Thus, when evaluating a model, the F_1-score is a suitable metric to use in order to strike a good balance between precision and recall.

While it is straight forward to calculate precision and recall given a classification task, the calculations need to be adjusted when used for object detection. To account for the spatial accuracy, a threshold value τ is introduced such that a predicted bounding box p is only counted as correct if the intersection-over-union $IOU(p,t) > \tau$, for any ground-truth bounding box $t \in TP$ (see Fig. 2.22 for an intuitive definition of $IOU(p,t)$).

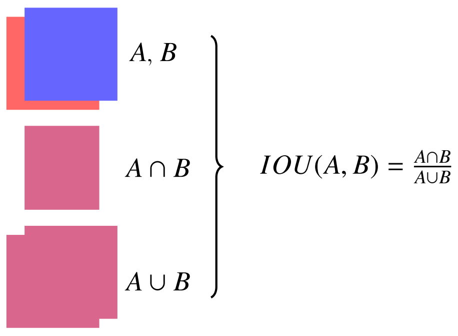


Figure 2.22: Visualization of the intersection-over-union $IOU(A,B)$ of two arbitrary bounding boxes A and B. The intersection of the bounding box areas is normalized by the union of the bounding box areas. The range is $[0, 1]$, since $0 \le A \cap B \le A \cup B$.

Fréchet Inception Distance

The Fréchet Inception Distance [30] (preceded by the Inception Score [31]) is a measurement used for comparing the similarity of two multivariate Gaussians $X_r \sim \mathcal{N}(\mu_r, \Sigma_r)$ and $X_g \sim \mathcal{N}(\mu_g, \Sigma_g)$:

$$FID(X_g, X_r) = ||\mu_r - \mu_g||^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{\frac{1}{2}}) \tag{2.26}$$

where X_r and X_g are the 2048-dimensional activations from the pool3 layer in the Inception network [32], μ_r, μ_g the mean vectors and Σ_r, Σ_g the covariance matrices.

In the context of GANs, X_r represents the activations of real image samples and X_g the activations of generated image samples. To compute the Fréchet Inception Distance between these two sets of image activations one assumes that an activation is generated from a multivariate gaussian distribution. This allows us to to estimate a real distribution p_r and a generated distribution p_g from X_r and X_g respectively i.e. computing mean vectors and covariance matrices. Since a multivariate gaussian is fully determined by its mean vector and covariance matrix, we simply compare these two entities for the two distributions p_r and p_g. This comparison is done by Eq. 2.26. If the mean vectors and the covariance matrices for the two distributions were to align, we see that Eq. 2.26 would evaluate to zero, which makes sense since it measures distance.

In our thesis we use this metric to compare visual similarities between the synthetic, refined synthetic, and real datasets. Since the metric is measuring distances between estimated distributions, we work to obtain results where the distance from the refined synthetic dataset to the real dataset is low.

Principal Component Analysis

As mentioned in Section 2.6, images can be represented with perceptual features. When using VGG16, one often extracts thousands of these features from various max pooling layers to achieve a good representation of the image. To get a better understanding of a dataset, visualizing these features is often helpful, but plotting them is impossible due to the high dimensionality. By using Principal Component Analysis (PCA) one can find a lower dimensional representation of these features and at the same time preserve as much variance as possible from the original features. These lower dimensional representations can then be plotted in order to better understand the dataset. One normally reduces the number of dimensions to two or three, to alleviate plotting.

In our thesis we use this metric to visualize how our datasets are distributed. The three datasets (synthetic, refined synthetic, real) are assumed to be one dataset with different "classes". We then perform PCA on this combined dataset and visualize the images on a scatter plot to see how the cluster of refined synthetic images relate to the synthetic and real clusters. If we are successful, the refined synthetic cluster should be very similar to the real cluster and not the synthetic cluster.

Chapter 3
Method

All computational work presented in this thesis was performed at a workstation with the following specifications: an NVIDIA Geforce RTX 2070 graphics card, an Intel Core i7-4790K CPU, and 16 GB RAM.

3.1 Strategy

We create a pipeline that produces refined synthetic datasets and evaluates the quality of these. It comprises three steps as presented below. Input are background images of the environment(s) we want to produce synthetic images of and GAN parameter settings that are used for the refinement. Output is a set of refined datasets and a comprehensive evaluation of how useful these datasets can be as training data.

Thus, after a set of GAN parameter configurations have run through the pipeline we can immediately see which of the configurations yielded the best detection performance. With this, we can swiftly test new ideas and configurations and get automatic feedback on whether the results are good or not. The process pipeline consists of three key building blocks (see Fig. 3.1):

1. **Renderer**: we render a synthetic dataset S that roughly approximates the real dataset R_{train} using the image rendering software Blender [33]. This is further explained in Section 3.2.

2. **Refiner**: we use GANs to refine the synthetic dataset S in order to further approximate R_{train}, producing a set of refined synthetic datasets S^+. Note that we use R_{train} to train the GAN that is then used to refine the synthetic dataset S. This is further explained in Section 3.3.

3. **Evaluation**: we train SqueezeDet on S, R_{train}, and each member $s_i^+ \in S^+$ respectively. We then evaluate their performance on R_{test}. This is further explained in Section 3.4.

The real dataset R has been splitted into three subsets: R_{train}, R_{val}, and R_{test}. R_{train} is used as the target dataset in the refinement step. R_{val} is used in the evaluation step to select the best performing models, and R_{test} is used to evaluate the real world performance.

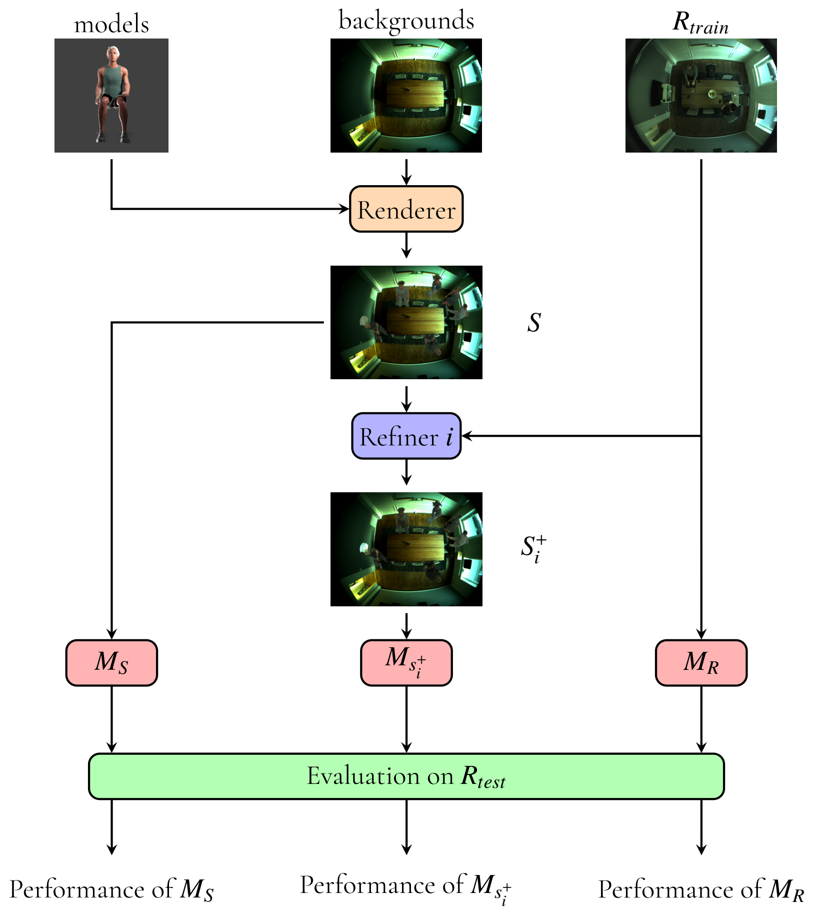


Figure 3.1: Schematic overview of our pipeline. For each refiner $i, i \in \{1 \ldots N\}$, we produce a refined synthetic dataset S_i^+. In turn, for each dataset S_i^+ we train a SqueezeDet network $M_{S_i^+}$. These results are then evaluated and compared with the performance of M_S (network trained on synthetic data) and M_R (network trained on real data).

3.2 Rendering

We use Blender 2.79 [33], a graphics software tool, to render 3D human models and superimpose them on top of the background image. To avoid unrealistic placements of our human models, we annotate the background with object locations, such as tables, and randomize the human positions until there are no collisions with other human models or objects. Not only does this let us avoid object collisions, but it also allows us to place models behind annotated objects (notice for example in Fig. 3.4 that the legs of the sitting models have been cut at the table edge, giving a sense of depth).

To realistically superimpose our models on the image, we adjust the size of the models depending on the euclidean distance to the camera. We then randomize the lighting, add some slight motion blur to our models, and project model shadows on to the background image. Further, the rendering is done with a fisheye lens, to accurately capture the deformations of the real dataset.

To render human models, we use a Blender addon called ManuelBastioniLAB [1], which allows us to vary different characteristics of our human models (such as clothes, skin, and hair) and to apply different poses to our models. See Algorithm 4 (Appendix) for more details.

For sitting poses, we use a default pose available in ManuelBastioniLAB and randomly rotate the body tilt. For standing poses, we use motion capture data from *CMU Graphics Lab Motion Capture Database* [35].

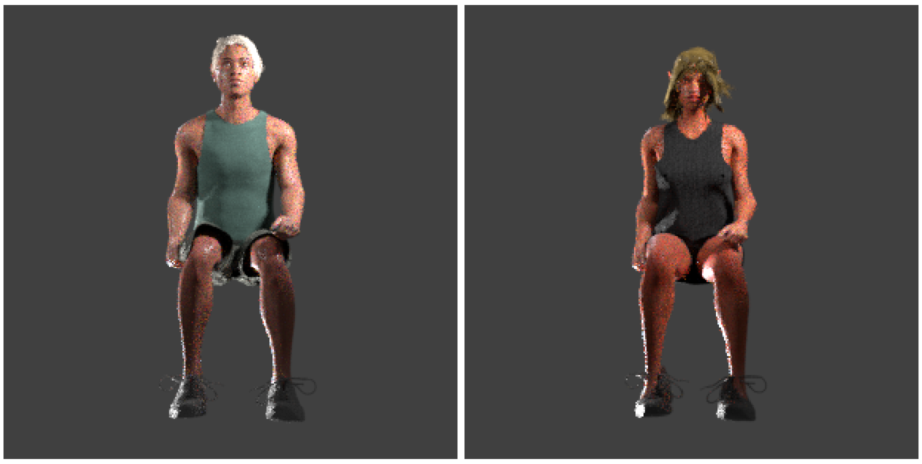

(a) (b)

Figure 3.2: Sitting models.

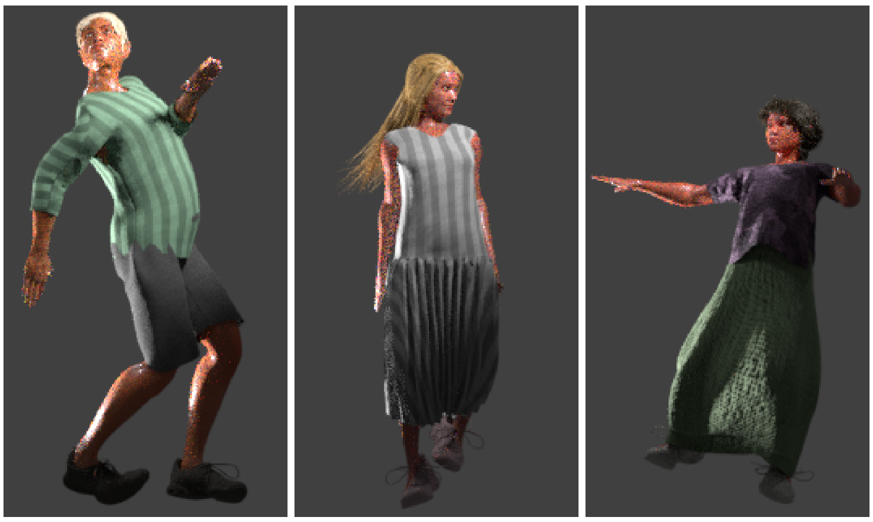

(a) (b) (c)

Figure 3.3: Standing models with randomly sampled motion capture poses.

[1]ManuelBastioniLAB is no longer maintained or publicly available at its website `http://www.manuelbastioni.com/`, but is available from the Github fork MB-Lab [34]

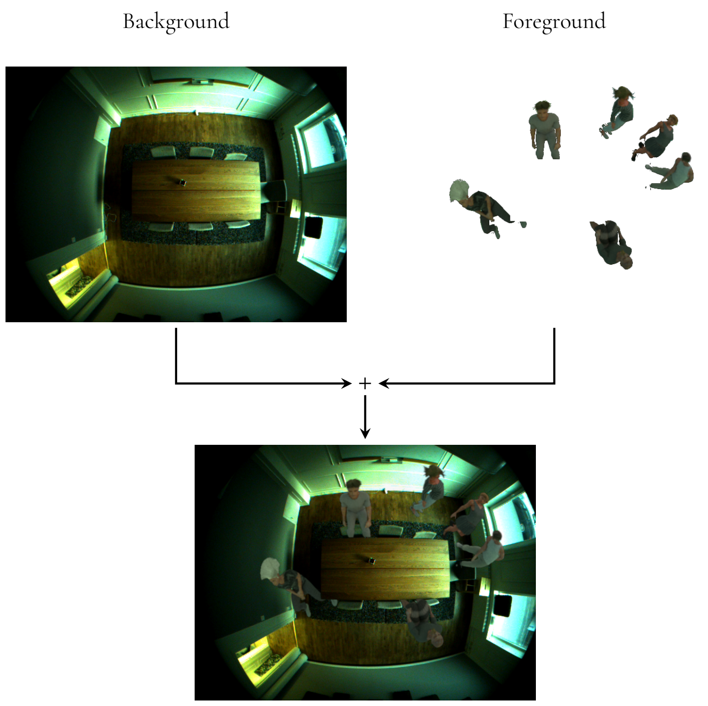
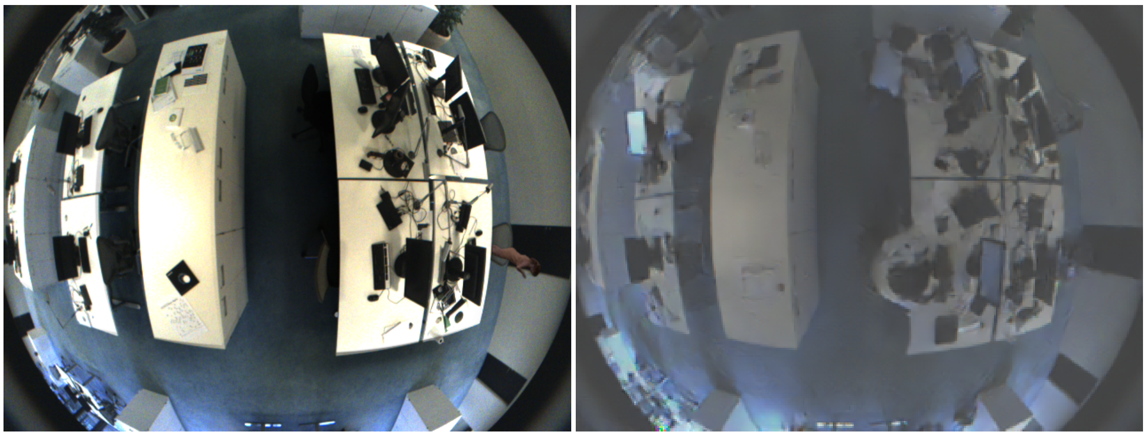


Figure 3.4: Schematic overview of how the synthetic images are generated.

3.3 Refiner

3.3.1 Full size images

We began with refining a dataset of synthetic people rendered on 40 different office locations. The real dataset comprised images from the same locations with real people in them. We resized our images to 448x336px (the maximum capability of our workstation due to memory limitations) and set λ to 10 (Eq. 2.9), in accordance with the CycleGAN paper [10].



(a) Synthetic image **(b)** Refined image

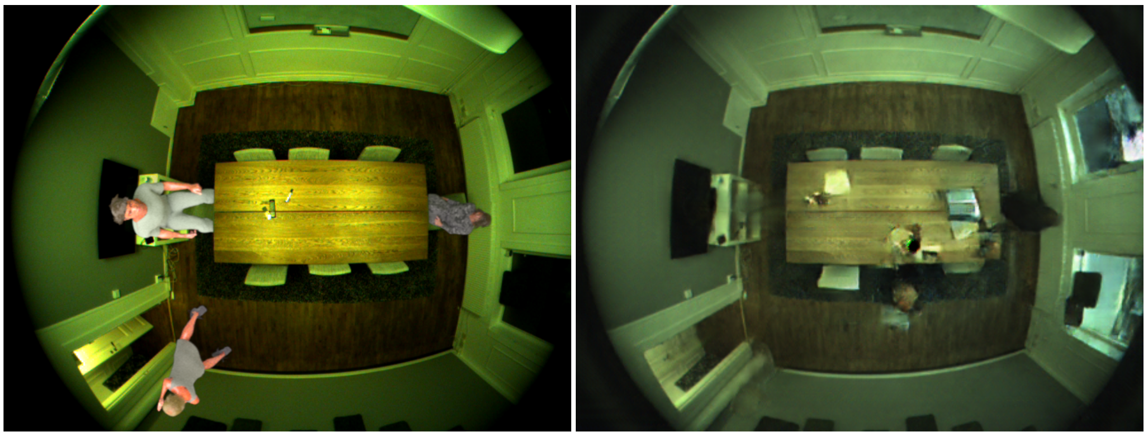
Figure 3.5: A synthetic image sample refined by CycleGAN. Annotations are not preserved and image quality is reduced.

Fig. 3.5 shows a sample of the results of this approach. Synthetic humans were categorically erased and replaced with background information and real humans were occasionally drawn in positions where there was no synthetic human in the input image. Even if we had managed to refine some synthetic humans, we would thus still have a problem of false negatives - since having humans in unannotated positions would penalize the object detection network.

In Fig. 3.5a, there is one synthetic human positioned in the right side of the image with brown clothing. As seen in Fig. 3.5b, the synthetic human has been erased and replaced with background information - while two "real" humans have been drawn in the middle of the image. The observant reader may also notice that some of the computer screens have been turned on. Aside from the annotation information not being preserved, we can also observe that the image quality is drastically reduced (especially the lighting and contrast). Background information is retained, but at much lower detail.

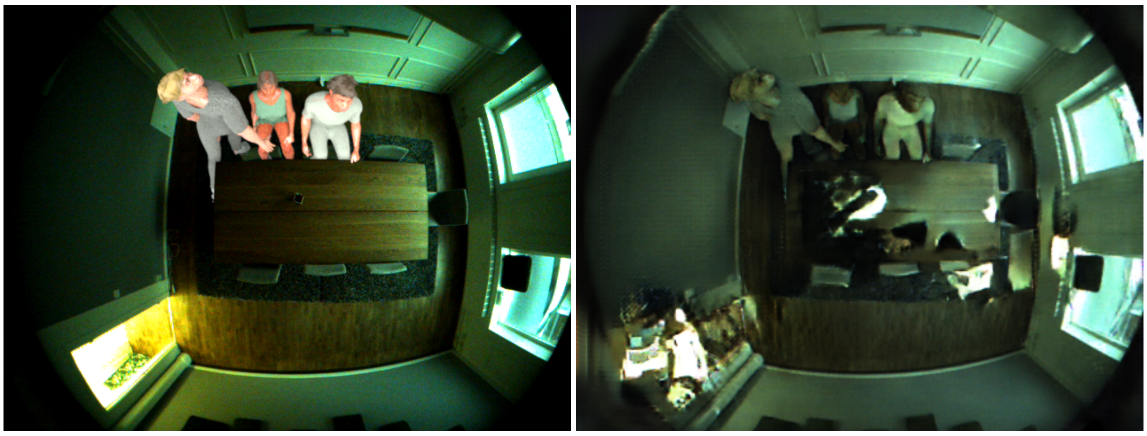
Due to the results, we did not perform a full pass through the pipeline and stopped the refinement.

We then moved on to a synthetic dataset of people rendered on only one office location ($\boldsymbol{S}$), and a real dataset of images from the same location ($\boldsymbol{R}$). This was an attempt to simplify the problem: intuitively the distribution of one location should be easier to approximate. λ was again set to 10 (Eq. 2.9).



(a) Synthetic image from S **(b)** Refined image from S

Figure 3.6: A synthetic image sampled from S refined by CycleGAN. Annotations are not preserved and realism is questionable.



(a) Synthetic image from S **(b)** Refined image from S

Figure 3.7: A synthetic image sampled from S refined by CycleGAN with a cyclic pixel ($\lambda = 10$) and feature loss ($\lambda = 10^{-5}$). Annotations are more preserved than in Fig. 3.6, but are still lost to some extent while realism is still questionable.

This approach did not end well either, with frequent erasures of synthetic humans, and unrealistic modifications of the background. In the sample in Fig. 3.6, we see that two out of three synthetic humans have been erased and a fourth human seems to have been drawn from scratch.

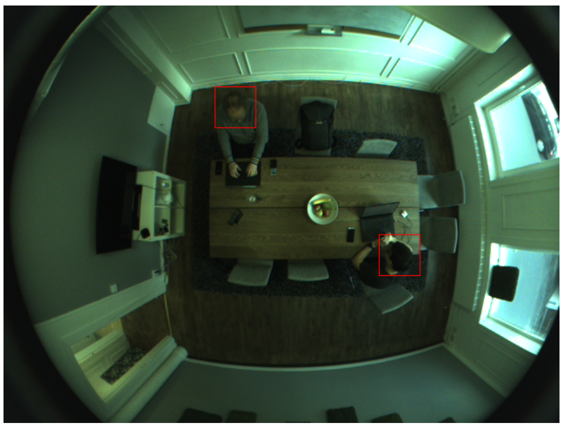
In Fig. 3.7, we see a sample using the same approach but with a cyclic feature loss and skip connections. We observe that the GAN still wishes to erase some synthetic humans and unrealistic drawings are added.

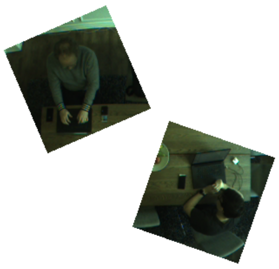
The images in $\boldsymbol{R}$ frequently contain people sitting at their computers (see e.g. Fig. 3.8). The result of this is that our images consistently contain computers and they are thus a representative feature of the dataset. It is thus not surprising that during refinement, the GANs tried to draw computers from scratch - since many real images have them! This can be seen in both Fig. 3.6 and Fig. 3.7. The realism of these computers is however questionable.

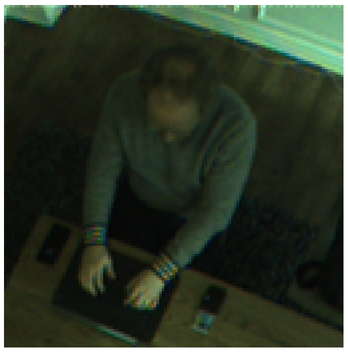
It is likely that the erasure of synthetic humans was due to the discriminator's architecture. Recall that CycleGAN uses a PatchGAN as its discriminator (explained in Section 2.3) and as such only discriminates on local patches of an image. Thus, given full size images, the majority of real image patches would be comprised of background information and the generator would therefore be incentivized to generate images comprising background information. Further, the cyclic loss of replacing synthetic humans with background information is quite low, since the synthetic humans amount to a small share of the total number of pixels.

We only later discovered that multi-instance transformation, where the desired foreground is not explicit, is still a hard problem in this field, where progress has only recently begun to take shape [36]. However, the technique presented in "InstaGAN: Instance-aware Image-to-Image Translation" [36] is supervised and requires conditional masks of both domains. While we could produce ground truth masks for the synthetic images, we naturally do not have the capability to do so for real images. Of course, we could use a pretrained masking network such as [37], but whether it would perform well on fisheye images from above is uncertain. We therefore moved on to the following approach.

3.3.2 Cropped images


(a) Image with annotated head positions


(b) Masked bounding boxes of humans


(c) Crop of the human on the left

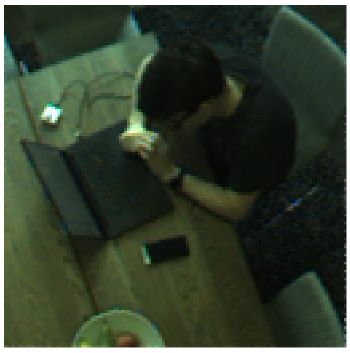

(d) Crop of the human on the right

Figure 3.8: Given an image with annotated head positions, we extract bounding boxes directed towards the center of the image and rotate them upright. We then crop these bounding boxes and receive our cropped images used for refinement.

Due to the previous failed results, we decided to crop the images to isolate the humans as the foreground.

The real dataset $\boldsymbol{R}$ comprises around 800 images annotated with human head positions. Using these labels, we automatically cropped the images such that they always featured a human head at the same location, as seen in Fig. 3.8. A more detailed description of this cropping procedure can be seen in Algorithm 5 (Appendix).

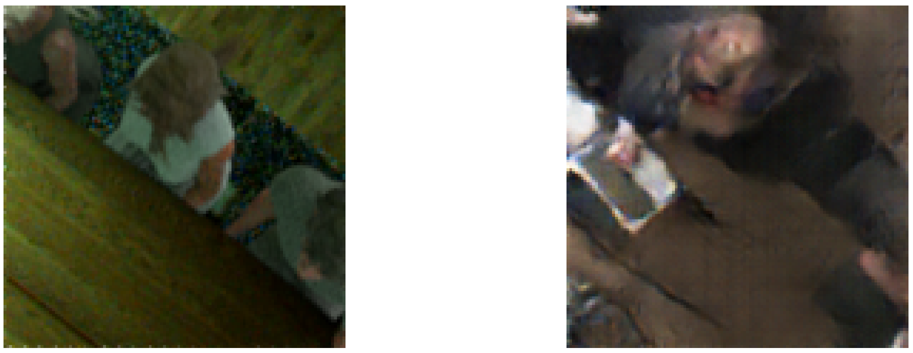
Note that we now transition from an unsupervised approach to a semi-supervised approach, since we now need positional annotations of humans in order to create our real dataset.

To create our synthetic dataset we again follow Algorithm 5 (Appendix) and perform the same procedure as in Fig. 3.8, the difference being that the head annotations are automatically generated at render-time.

We then refine the cropped synthetic dataset with a GAN. After refinement, the cropped images are reinserted onto their original images, superimposed onto their original location, and masked such that only the synthetic human is refined and not any background information (see Algorithm 6 in Appendix). We then use this as training data for the object detection network, as outlined in Fig. 3.1. Due to the added crop- and reinsert procedures, the Refiner block in Fig. 3.1 is modified as shown in Fig. 3.10. Whereas it in our previous approach was simply a GAN with $\boldsymbol{S}$ and $\boldsymbol{R}_{train}$ as input, it is now modified to encompass an image cropper, a GAN, and an image reinserter. With our new approach it takes, in addition to $\boldsymbol{S}$ and $\boldsymbol{R}_{train}$, a dataset $\boldsymbol{S}_a$ as input, where each image $s_a \in \boldsymbol{S}_a$ is a ground truth mask of an image $s \in \boldsymbol{S}$.

As seen in Fig. 3.9, our early results showed generated computers (just like with the full size images, but now a bit more realistic). To prevent this, we filtered the cropped real training set ${}^{c}\boldsymbol{R}_{train}$ to not consistently include computers. As expected, the GANs then stopped drawing them.

In Table 3.1, we present the final synthetic and real datasets (cropped and uncropped) that we have used. Since [2] only gained better performance with a synthetic dataset larger than the real, we also make sure that our synthetic dataset is much larger than our real dataset.



(a) Input **(b)** Output

Figure 3.9: Synthetic and refined synthetic sample, where a computer has been drawn from scratch.

Dataset	Size	Description
R_{train}	102	Real train dataset
R_{val}	142	Real validation dataset
R_{test}	575	Real test dataset
${}^{c}R_{train}$	272	Cropped images from R_{train}
S	1498	Synthetic dataset
${}^{c}S$	5576	Cropped images from S

Table 3.1: A table of our synthetic and real datasets.

3.3.3 GAN networks

Having produced a set of "good" crops of real images we started training different GANs through the pipeline. Among them were the original CycleGAN, CycleGAN with various combinations of feature and pixel losses, and AGGAN. For each configuration we also experiment with skip connections connecting the input and output layer in the generator.

The complete list of used GAN models is seen in Table 3.2 along with their respective refined datasets. We do not list the corresponding cropped refined datasets, but instead rely on the following notation. For any refined dataset X, let ${}^{c}X$ denote the dataset of cropped images from X. Note that the size of each refined dataset X is equal to that of the synthetic dataset S and the size of each cropped refined dataset ${}^{c}X$ is equal to that of the cropped synthetic dataset ${}^{c}S$.

Also note that the right hand side subscripts of the dataset names indicate what losses were used during training of the GAN. CP is a cyclic pixel loss (Eq. 2.8), CF is a cyclic feature loss (Eq. 2.18), IP is an identity pixel loss (Eq. 2.17), and IF is an identity feature loss (Eq. 2.19).

A right hand side superscript S indicates that skip connections were used in the generator. Thus, there exist two versions of each refined synthetic dataset: X, without generator skip connections, and X^S, with generator skip connections.

All weight variations (the values of λ) in Table 3.2 were done by trial-and-error. Most loss- and architecture variations were performed by referencing literature in the field, such as using VGG16 [23] and discriminator feature losses [25]. Datasets M and M^S are of our own invention, where we had the intuitive idea that a mean feature loss (presented in Section 2.6) would help to nudge the generator's transformations towards the direction of the target dataset.

We also include a dataset (C_{CP}) produced by a default implementation of CycleGAN for reference, to validate that our variations are better at this task than the default implementation.

Dataset	Size	Refined with …	Loss equation	VGG16 pooling layers	Discriminator convolutional layers
C_{CP}	1498	CycleGAN with a cyclic pixel loss ($\lambda = 10$)	2.8		
C_{CPCF}	1498	CycleGAN with a cyclic pixel loss ($\lambda = 10$) and a cyclic feature loss ($\lambda = 10^{-5}$)	2.8, 2.18	2	
C_{CPCFIF}	1498	CycleGAN with a cyclic pixel loss ($\lambda = 10$), a cyclic feature loss ($\lambda = 10^{-5}$), and an identity feature loss ($\lambda = 5 \cdot 10^{-6}$)	2.8, 2.18, 2.19	2	
C_{CPCFIP}	1498	CycleGAN with a cyclic pixel loss ($\lambda = 10$), a cyclic feature loss ($\lambda = 10^{-5}$), and an identity pixel loss ($\lambda = 10$)	2.8, 2.18, 2.17	2	
D	1498	CycleGAN with a cyclic feature loss in the discriminator's feature space ($\lambda = 10^{-5}$) and an identify feature loss ($\lambda = 5 \cdot 10^{-6}$)	2.21, 2.19		1, 2, 3, 4
M	1498	CycleGAN with a mean feature loss ($\lambda = 10^{-5}$) and an identity feature loss ($\lambda = 5 \cdot 10^{-6}$)	2.20, 2.19	2, 5	
A	1498	AGGAN with a cyclic pixel loss ($\lambda = 10^{-5}$)	2.12		
C^S_{CP}	1498	CycleGAN with a cyclic pixel loss ($\lambda = 10$)	2.8		
C^S_{CPCF}	1498	CycleGAN with a cyclic pixel loss ($\lambda = 10$) and a cyclic feature loss ($\lambda = 10^{-5}$)	2.8, 2.18	2	
C^S_{CPCFIF}	1498	CycleGAN with a cyclic pixel loss ($\lambda = 10$), a cyclic feature loss ($\lambda = 10^{-5}$), and an identity feature loss ($\lambda = 5 \cdot 10^{-6}$)	2.8, 2.18, 2.19	2	
C^S_{CPCFIP}	1498	CycleGAN with a cyclic pixel loss ($\lambda = 10$), a cyclic feature loss ($\lambda = 10^{-5}$), and an identity pixel loss ($\lambda = 10$)	2.8, 2.18, 2.17	2	
D^S	1498	CycleGAN with a cyclic feature loss in the discriminator's feature space ($\lambda = 10^{-5}$) and an identify feature loss ($\lambda = 5 \cdot 10^{-6}$)	2.21, 2.19		1, 2, 3, 4
M^S	1498	CycleGAN with a mean feature loss ($\lambda = 10^{-5}$) and an identity feature loss ($\lambda = 5 \cdot 10^{-6}$)	2.20, 2.19	2, 5	
A^S	1498	AGGAN with a cyclic pixel loss ($\lambda = 10^{-5}$)	2.12		

Table 3.2: A table of our refined datasets. Unless otherwise specified, feature losses are defined in VGG16's feature space. Datasets with a superscripted S indicate that the respective generator had skip connections drawn between its input and output layers. The last two columns describe which layers were used in the respective feature losses.

3.4 Evaluation

Having rendered a synthetic dataset and refined it with different GAN models (refiners), we evaluate the quality of the refined dataset using it as training data for SqueezeDet and test the performance on R_{train}. We then compare the test F_1-score of all trained SqueezeDet networks, to evaluate the quality of our datasets. Further, we measure the Fréchet Inception Distance from the refined datasets to the synthetic and real datasets to get a quantitative measure of how much we are able to approximate the real dataset. For some datasets we also use PCA, as explained in Section 2.10.

The whole pipeline algorithm can be seen in its most distilled form in Algorithm 3. See the Appendix for the subroutines called on in Algorithm 3.

It should be noted that the sizes of our refined synthetic datasets are not equal to the size of the real training dataset R_{train}. We are however careful to make sure that the refined synthetic datasets require exactly the same real images present in R_{train} during the refinement process. Thus, even though the dataset sizes differ, the detection performance comparisons are still fair since all datasets have required the same amount of supervised data to produce (excluding the synthetic dataset S).

Algorithm 3 Our pipeline algorithm

```
1:  collect a real dataset $R$. Split into $R_{train}$, $R_{val}$, $R_{test}$
2:  collect a set of background images $B$ of the same environment as $R$
3:  let $S$ = RENDER($B$)
4:  let $refiner\_configs$ = refiner configurations from Table 3.2
5:  **for** $r \in refiner\_configs$ **do**
6:      let $S^+$ = REFINE($S, R_{train}, r$)
7:      train SqueezeDet on $S^+$
8:      pick the training checkpoint $i$ with the best $F_1$-score by validating on $R_{val}$
9:      evaluate the network using checkpoint $i$ on $R_{test}$
10: **end for**
```

3.5 Implementation

3.5.1 CycleGAN and AGGAN

For CycleGAN [10], we use a TensorFlow implementation [38] and modify it to support custom image sizes and the losses described in Section 2.6. Further we add support for skip connections between the generator input and output. The discriminators and generators used are thus identical to that of Fig. 2.17 and 2.18, except for the added skip connection in the generator.

For AGGAN [21], we re-implement the original paper by extending the same TensorFlow implementation as before [38] and validate our implementation on the horse2zebra dataset.

The following training parameters were used for all GANs in Table 3.2. The batch size was set to 1, so that for each iteration a discriminator would discriminate on one real and one generated image. The number of training epochs used was 160, so that the network would in total process $160 \cdot min(|R_{train}|, |S|)$ number of images during training. The initial learning rate was set to 0.0002 and then set to slowly decay after 80 epochs (as seen in Algorithms 8 and 9 in Appendix). In all networks based on CycleGAN, the number of filters in the first convolutional layer of the discriminator and generator is 64. For AGGAN, the corresponding numbers of filters are 64 for the discriminator and 32 for the generator. The discriminator uses a minibatch discrimination size of 50. We use an Adam optimizer with a momentum term of 0.5.

3.5.2 SqueezeDet

For SqueezeDet [3], we use a Keras implementation [39] that uses pretrained weights from ImageNet [40] and modify it to support our problem. Specifically we change the anchor sizes to reflect the sizes of human heads, add support for our annotation files, and configure it to automatically integrate with the pipeline.

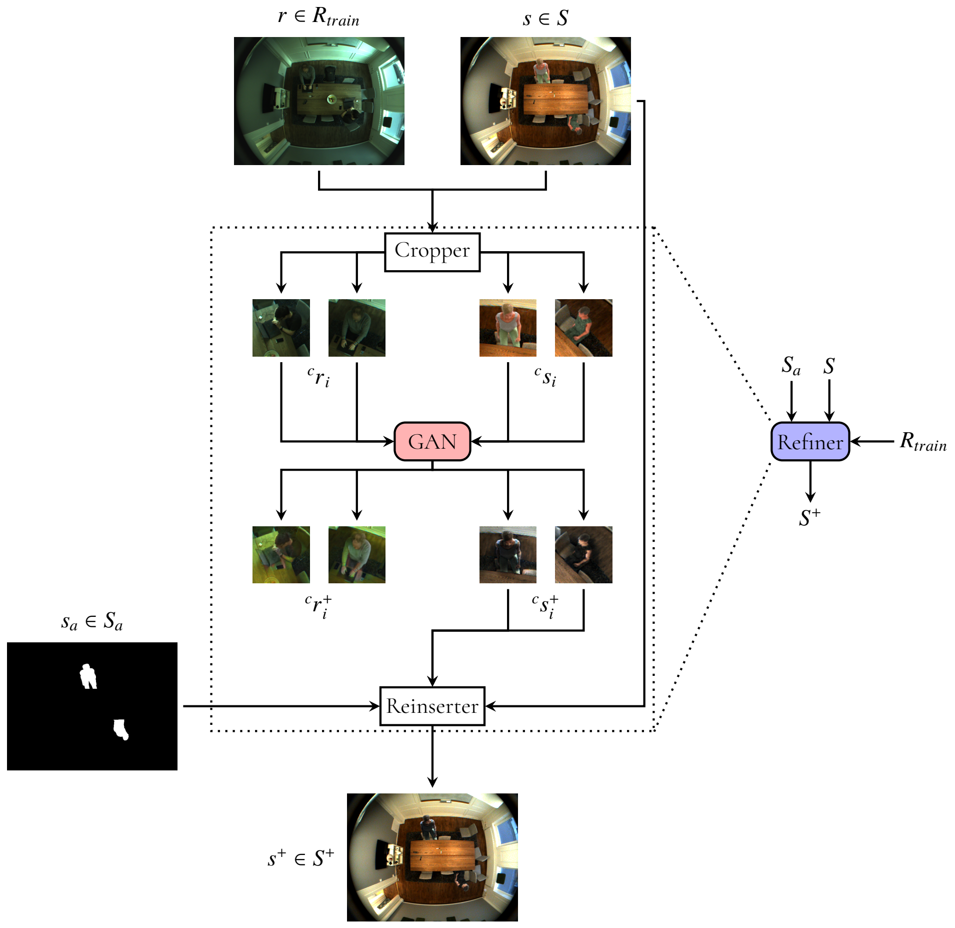


Figure 3.10: Schematic overview of the refiner block. Each real image $r \in R_{train}$ and each synthetic image $s \in S$ is cropped, producing the cropped real image ${}^c r_i \in {}^c R_{train}$ and cropped synthetic image ${}^c s_i \in {}^c S$. The GAN is then trained to translate ${}^c R_{train} \to {}^c S$ and ${}^c S \to {}^c R_{train}$. After the training is complete, each image ${}^c s_i$ is then refined by the generator G producing ${}^c s_i^+$. ${}^c s_i^+$ is then reinserted into the original image s which then is masked by s_a, producing the full size refined synthetic image s^+.

Chapter 4

Results

4.1 Overview

In this chapter we present our results. First, on pages 52-56, we present our visual results. We then go on to show our quantitative results, on pages 57-62.

For our visual results, we show random and cherrypicked samples of both cropped and full size images. In Fig. 4.1 and 4.2 we show unmasked generator outputs (unless otherwise stated) to better present the nature of each refinement. Elsewhere, cropped images are masked such that they accurately depict the end result.

In our quantitative results, we present our quantitative evaluations of the refined synthetic datasets along with comparisons of how these results compare to the results of the real and synthetic dataset. In summary, our quantitative results are the following:

- Comparison of the Fréchet Inception Distance from all refined synthetic datasets to both the synthetic dataset S and the real dataset R_{train}.
- Comparison of test F_1-score of SqueezeDet networks trained on the synthetic, real, and all refined synthetic datasets (respectively).
- PCA plot of our best performing refined synthetic dataset, visualizing how it relates to the synthetic and real dataset (S and R_{train}).
- Validation plots of SqueezeDet networks trained on our best performing dataset D^S, the real dataset R_{train}, and the synthetic dataset S (respectively).
- Analysis of correlation between the Fréchet Inception Distance and F_1-score.
- Analysis of how the size allocation of refined synthetic and real images in a combined dataset $D^S + R_{train}$ affects the detection performance.

As described in Section 3.3.3, datasets with a left hand side superscripted C indicate that they are cropped. Datasets with a right hand side superscripted S indicate that skip connections were used in the generator during training and refinement. Right hand side subscripts, such as CP, describe what losses were used for the respective datasets.

4.2 Visual results

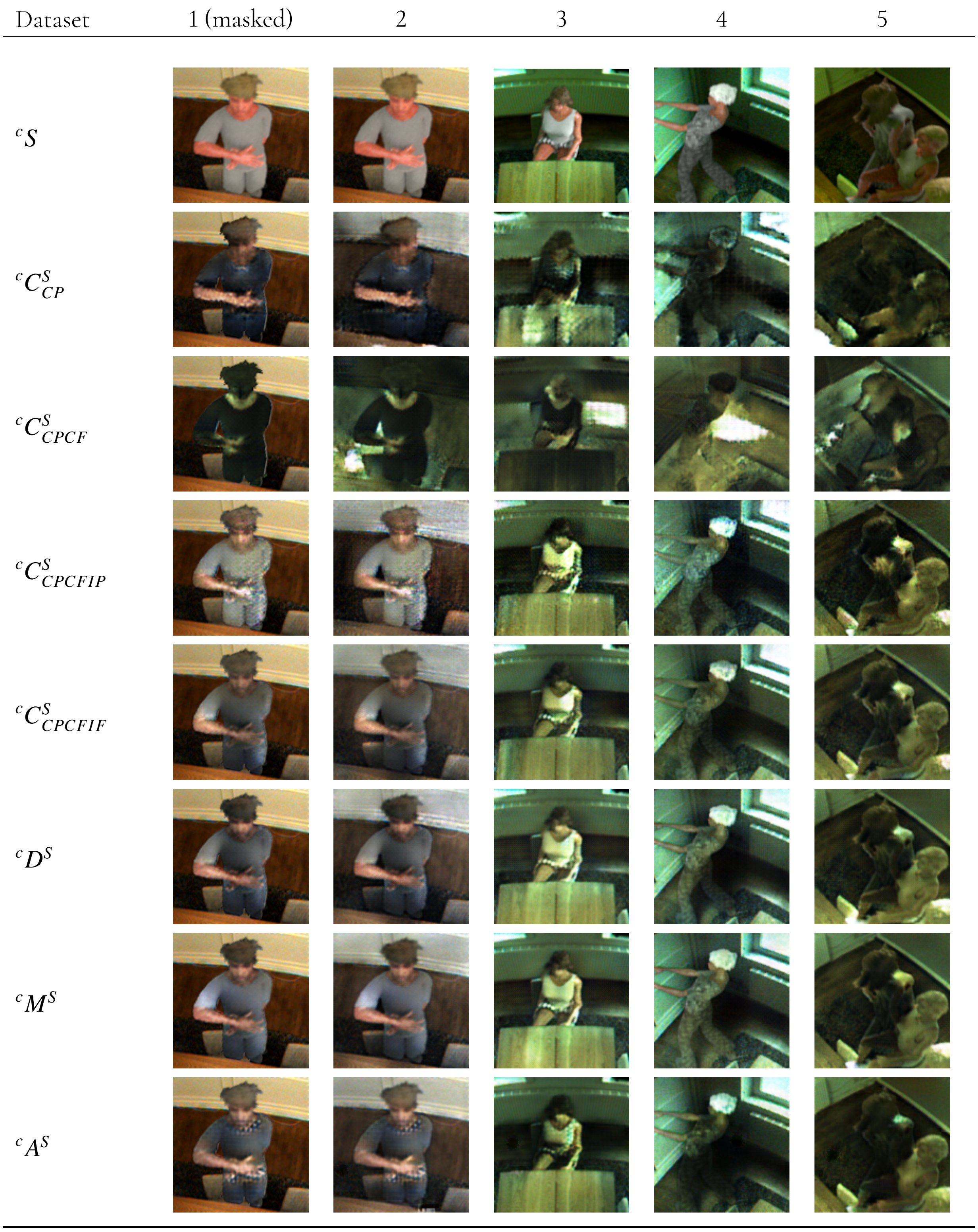


Figure 4.1: Randomly sampled refinements of the cropped synthetic dataset ${}^{c}S$ for different GANs with skip connections. The first column shows masked refinements, and the second to fifth column show unmasked refinements.

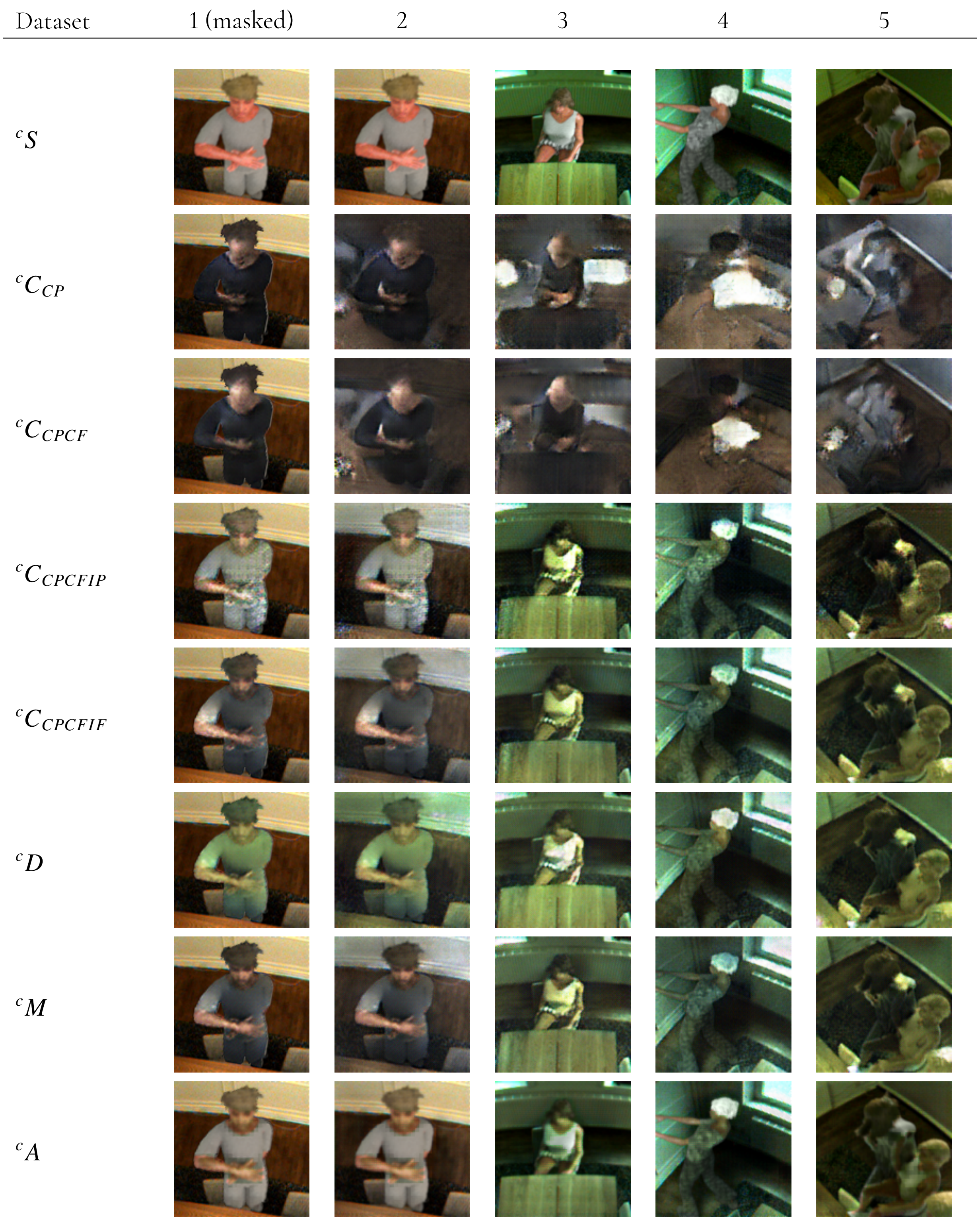


Figure 4.2: Randomly sampled refinements of the cropped synthetic dataset ^cS for different GANs without skip connections. The first column shows masked refinements, and the second to fifth column show unmasked refinements.

Fig. 4.1 and 4.2, seen on pages 52-53, show random samples from the cropped synthetic dataset ${}^{c}S$ and corresponding refinements from all cropped refined synthetic datasets. In both figures, the first column shows crops with their mask applied, whereas the second to fifth columns show crops without masks. These results are discussed in Section 5.1-5.5.

For our quantitatively best performing dataset D^S, we present more extensive material. Fig. 4.3 shows cherrypicked samples from the cropped refined synthetic dataset ${}^{c}D^S$. Fig. 4.4 shows cherrypicked samples from the fullsize refined synthetic dataset D^S. Randomly sampled fullsize images of D^S are presented in Fig. 4.5.

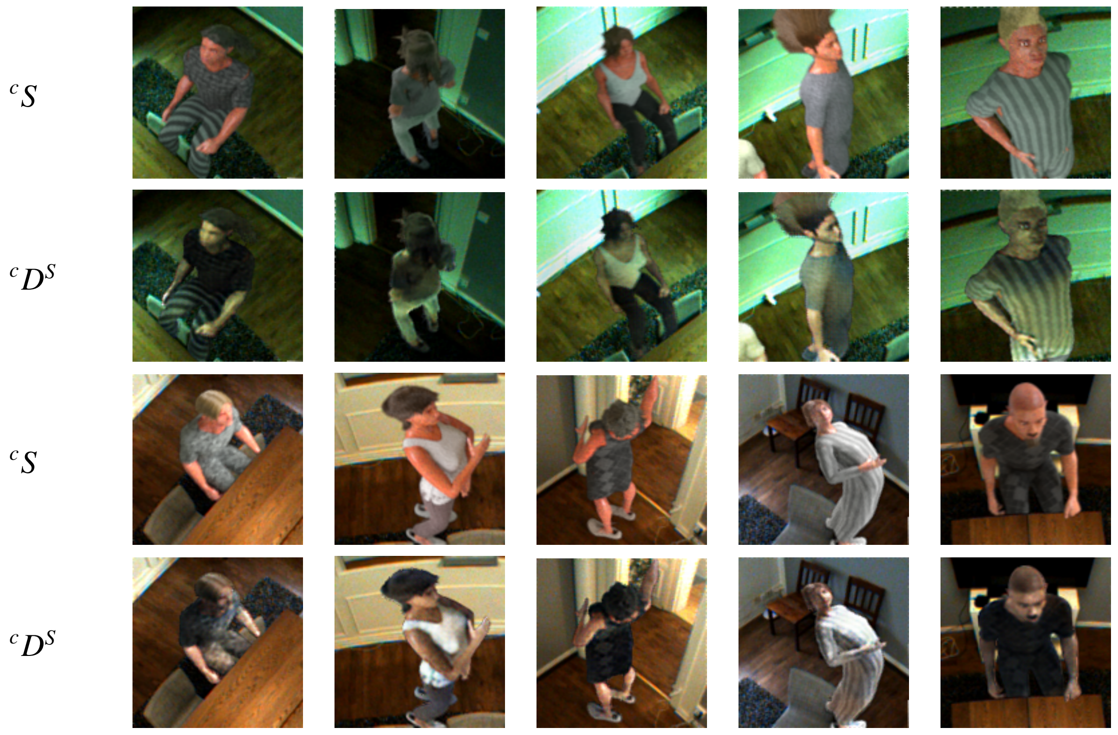


Figure 4.3: Subjectively cherrypicked samples from ${}^{c}D^S$ along with the original samples from ${}^{c}S$

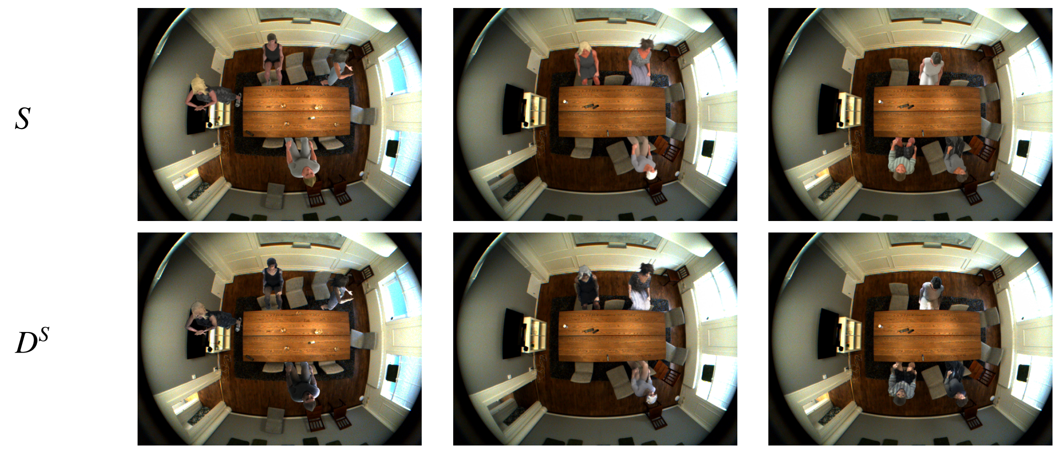


Figure 4.4: Subjectively cherrypicked samples from D^S along with the original samples from S.

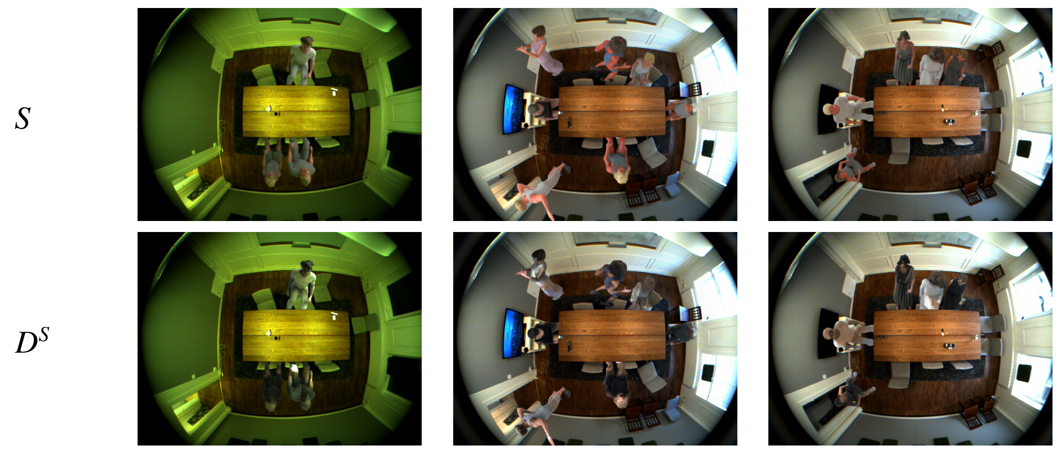


Figure 4.5: Random samples from D^S along with the original samples from S.

In Fig. 4.6, we show generated attention maps of datasets ${}^cA$ and ${}^cA^S$ that are produced by AGGAN, since the quality of these is crucial to the ability of AGGAN. The brightness of the attention maps reflects the attention of the network. High brightness reflects high confidence that the area constitutes foreground and thus should be transformed by the generator. Low brightness reflects high confidence that the area constitutes background and thus should remain unchanged. We see that neither ${}^cA$ nor ${}^cA^S$ have confident attention maps. A probable explanation is discussed in Section 5.5.1.

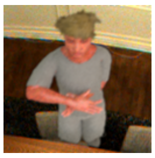
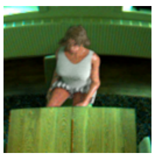
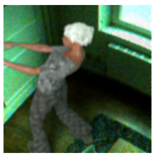
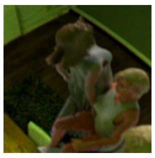
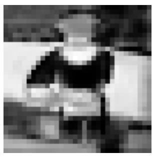
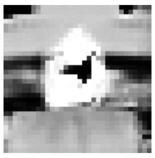
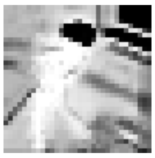
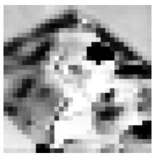
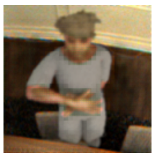
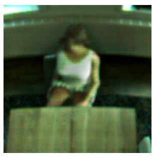
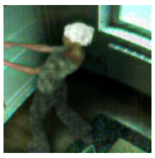
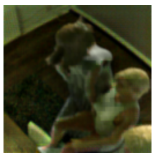
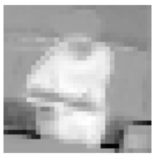
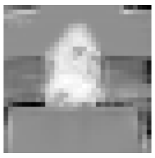
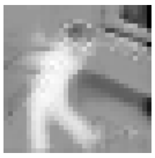
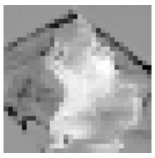
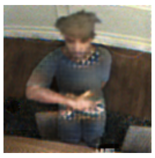
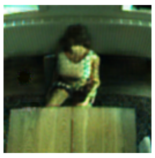
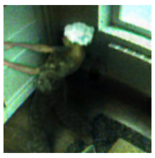
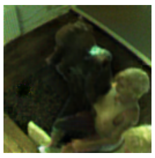
Dataset	1	2	3	4
Input samples from ${}^{c}S$				
Attention maps of ${}^{c}A$				
${}^{c}A$				
Attention maps of ${}^{c}A^{S}$				
${}^{c}A^{S}$				

Figure 4.6: Attention maps of ${}^{c}A$ and ${}^{c}A^{S}$, along with the input images from ${}^{c}S$ and the respective refinements.

4.3 Quantitative results

4.3.1 Fréchet Inception Distance

Table 4.1 lists the Fréchet Inception Distance from each cropped refined dataset to the cropped synthetic dataset ${}^{c}S$ and the cropped real dataset ${}^{c}R_{train}$. Apart from ${}^{c}S$ itself, we see that ${}^{c}A$ is closest to ${}^{c}S$. The dataset furthest from ${}^{c}S$ is ${}^{c}R_{train}$. Apart from ${}^{c}R_{train}$ and ${}^{c}D^{S} + {}^{c}R_{train}$, we see that D^{S} is closest to ${}^{c}R_{train}$. The dataset farthest from ${}^{c}R_{train}$ is ${}^{c}C_{CP}$.

Note that the distances between ${}^{c}S$ and ${}^{c}S$ and ${}^{c}R_{train}$ and ${}^{c}R_{train}$ should theoretically be 0. However, due to small sample sizes of the datasets this is not the case [41].

	Distance to ${}^cS$	Distance to ${}^cR_{train}$
${}^cR_{train}$	167.37	7.54
${}^cS$	4.6	165.67
${}^cD^S + {}^cR^C_{train}$	70.74	135.39
${}^cC^S_{CP}$	106.56	166.18
${}^cC^S_{CPCF}$	94.88	154.86
${}^cC^S_{CPCFIP}$	90.57	154.75
${}^cC^S_{CPCFIF}$	70.8	153.3
${}^cD^S$	69.8	147.82
${}^cM^S$	63.08	149.63
${}^cA^S$	74.43	149.99
${}^cC_{CP}$	105.66	170.87
${}^cC_{CPCF}$	98.19	169.3
${}^cC_{CPCFIP}$	89.54	155.54
${}^cC_{CPCFIF}$	72.87	149.57
${}^cD$	73.86	153.16
${}^cM$	69.55	151.21
${}^cA$	47.41	152.9

Table 4.1: Fréchet Inception Distances between the cropped synthetic dataset ${}^cS$, the cropped real dataset ${}^cR^C_{train}$, and the masked and cropped refined datasets.

4.3.2 SqueezeDet Performance Metrics

Tables 4.2-4.6 show SqueezeDet test results in terms of precision, recall and F_1-score for each dataset, where the first column represents the epoch from which the SqueezeDet network checkpoint was selected. Each of the datasets has been tested on the dataset R_{test}.

Specifically, Table 4.2 shows the test results of the SqueezeDet networks trained on the synthetic dataset S and the real dataset R_{train} which is used as a baseline when comparing the performance of our refined datasets. Table 4.3 shows the test results of the SqueezeDet networks trained on all the refined datasets created using a skip connection in the GAN generator. Table 4.4 is identical to Table 4.3, the only difference being that the refined datasets were created without the use of a skip connection in the generator. Table 4.5 shows test results of SqueezeDet networks trained on the combined datasets $R_{train} + S$ and $R_{train} + D^S$. Table 4.6 shows the performance of datasets where R_{train} has been combined with our best performing refined dataset D^S, where the size of D^S is varying. Fig. 4.7 shows a plot of Table 4.6.

	Epoch	Precision	Recall	F_1
S	24	0.875	0.832	0.853
R_{train}	89	0.9	0.833	0.865

Table 4.2: SqueezeDet test results on R_{test}, trained on synthetic (S) and real (R_{train}) datasets.

	Epoch	Precision	Recall	F_1
C^S_{CP}	39	0.844	0.827	0.835
C^S_{CPCF}	10	0.823	0.788	0.805
C^S_{CPCFIP}	30	0.819	0.837	0.828
C^S_{CPCFIF}	18	0.861	0.889	0.875
M^S	26	0.879	0.812	0.844
D^S	9	0.872	0.892	0.882
A^S	16	0.844	0.883	0.863

Table 4.3: SqueezeDet test results on R_{test} of refined datasets trained with skip connections.

	Epoch	Precision	Recall	F_1
C_{CP}	9	0.841	0.788	0.813
C_{CPCF}	12	0.777	0.851	0.812
C_{CPCFIF}	21	0.788	0.834	0.81
C_{CPCFIF}	21	0.845	0.886	0.865
M	30	0.889	0.843	0.865
D	7	0.781	0.88	0.827
A	3	0.777	0.89	0.83

Table 4.4: SqueezeDet test results on R_{test} of refined datasets trained without skip connections.

	Epoch	Precision	Recall	F_1
$R_{train} + S$	7	0.861	0.856	0.858
$R_{train} + D^S$	19	0.878	0.898	0.888

Table 4.5: SqueezeDet test results on R_{test}, trained on $R_{train} + D^S$ and $R_{train} + S$.

	Epoch	Precision	Recall	F_1
$R_{train} + 0x$	89	0.9	0.833	0.865
$R_{train} + 1x$	76	0.92	0.853	0.885
$R_{train} + 3x$	53	0.904	0.854	0.878
$R_{train} + 5x$	46	0.913	0.837	0.873
$R_{train} + 7x$	43	0.882	0.875	0.879
$R_{train} + 10x$	12	0.857	0.918	0.886
$R_{train} + 15x$	19	0.878	0.898	0.888

Table 4.6: SqueezeDet test results on R_{test}, trained on $R_{train} + D^S$, where the size of D^S is varied to be x times larger than R_{train}.

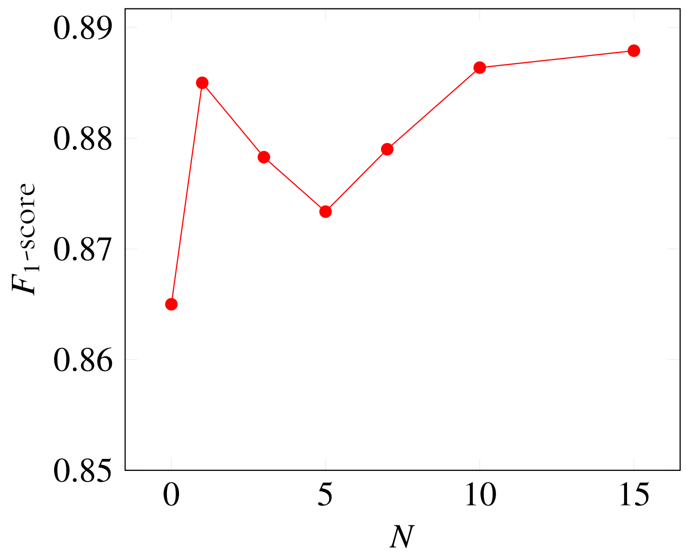


Figure 4.7: F_1-score on R_{test}, trained on $R_{train} + D^S$, where the size of D^S is varied to be N times larger than R_{train} (Plot of Table 4.6).

4.3.3 Correlation between F_1-score and Fréchet Inception Distance

In Fig. 4.8, we see that there exists a negative correlation between the Fréchet Inception Distance from cropped refined data to ${}^cR_{train}$ and the F_1-score on R_{test}.

In Fig. 4.9, we see that there also exists a negative correlation between the Fréchet Inception Distance from cropped refined data to ${}^cS$ and the F_1-score on R_{test} as well.

Table 4.7 shows the average Fréchet Inception Distance from the real dataset to all the refined datasets who used a skip connection in the generator and all who did not.

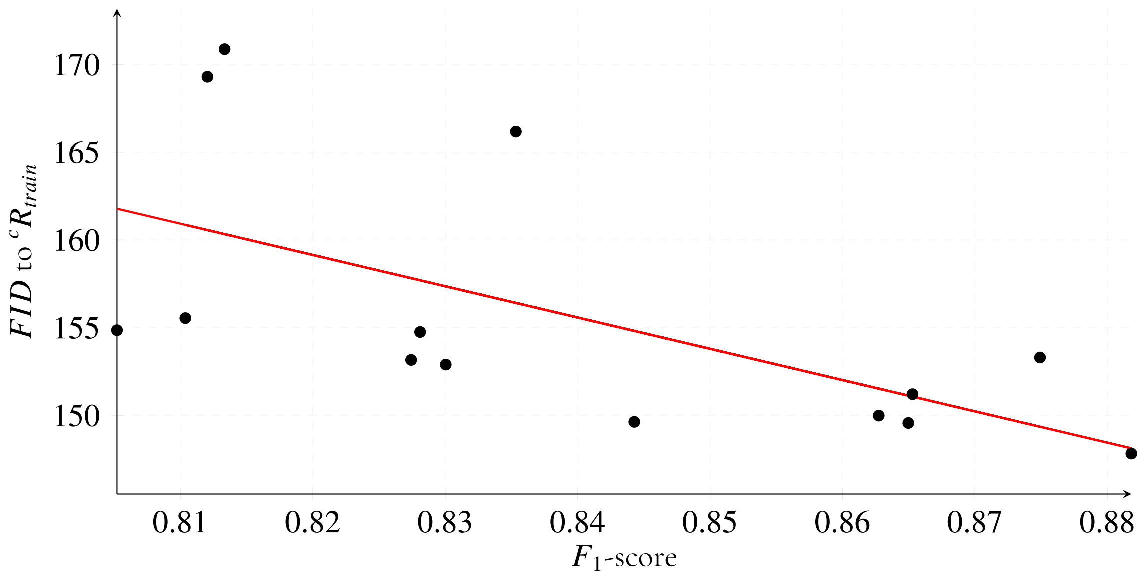


Figure 4.8: Linear regression of the F_1-score on R_{test} and the Fréchet Inception Distance (FID) to the cropped real dataset ${}^cR_{train}$ of all cropped refined datasets.

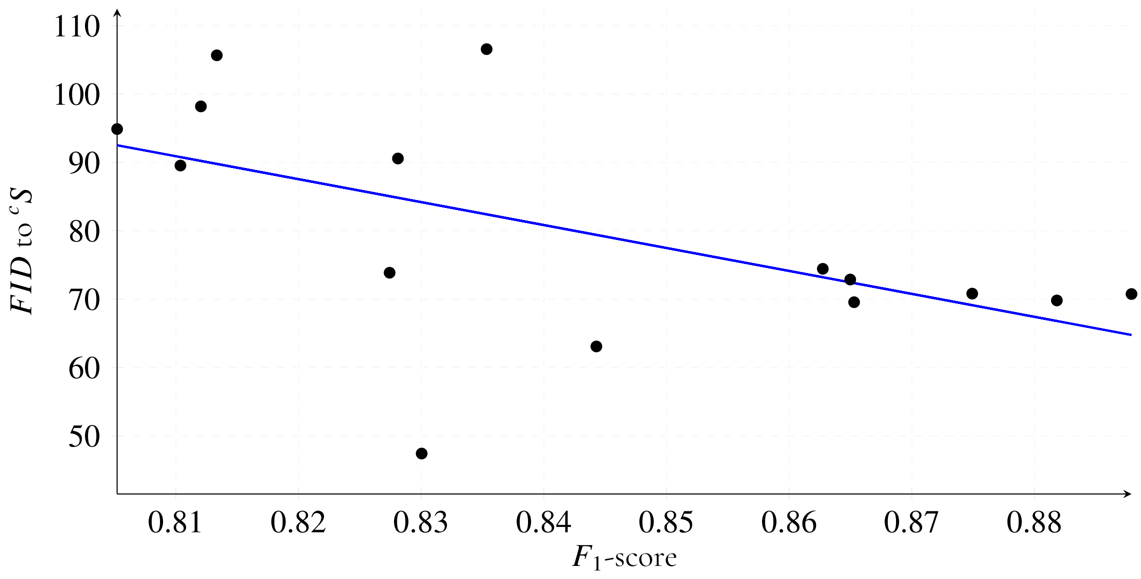


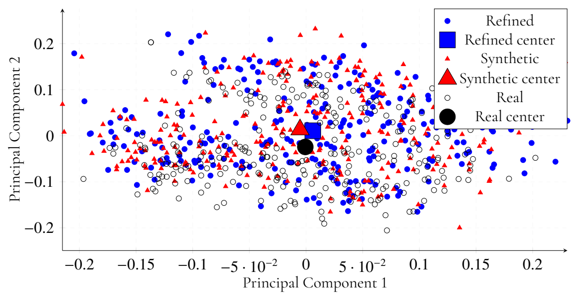
Figure 4.9: Linear regression of the F_1-score on R_{test} and the Fréchet Inception Distance (FID) to the cropped synthetic dataset ^cS of all cropped refined datasets.

Datasets	**FID**	F_1
X^S	153.79	0.847
X	157.51	0.832

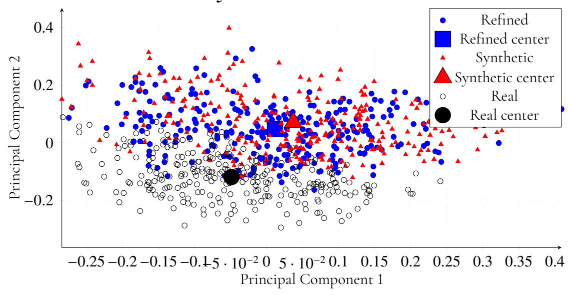
Table 4.7: Average Fréchet Inception Distance (FID) to $^cR_{train}$ and F_1-score on R_{test} of datasets trained with skip connections (X^S) and datasets trained without skip connections (X).

4.3.4 PCA Plots

Fig. 4.10 shows a PCA plot of $^cD^S$, ^cS, and $^cR_{train}$. In Fig. 4.10a we see the plot for the second max pooling features and in Fig. 4.10b we see the plot for the fifth max pooling features. For 4.10a, the $^cD^S$ center is not significantly closer to the $^cR_{train}$ center. In Fig. 4.10b, the $^cD^S$ center is closer to the $^cR_{train}$ center than the ^cS center.



(a) PCA plot of features from second max pooling layer of VGG16



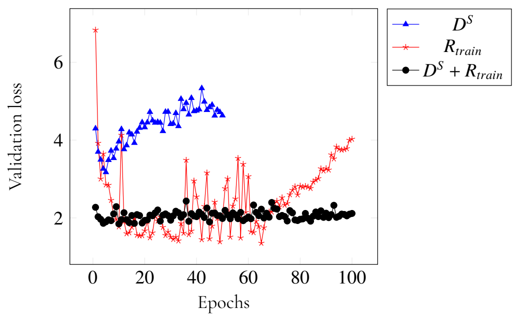
(b) PCA plot of features from fifth max pooling layer of VGG16

Figure 4.10: Plots of the two principal components of the VGG16 features for ${}^{c}D^{S}$, ${}^{c}S$, and ${}^{c}R_{train}$. ${}^{c}D^{S}$ and ${}^{c}S$ have been randomly sampled to match the size of ${}^{c}R_{train}$.

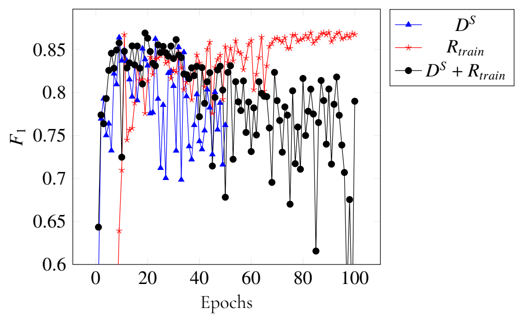
4.3.5 Training Plots

Fig. 4.11 shows validation loss, recall, precision and F_1-score for three SqueezeDet networks evaluated on the real dataset R_{val}. The first network has been trained on the dataset D^S with 50 epochs of training. The second has been trained on the dataset R_{train} with 100 epochs of training. The last network has been trained on the dataset $R_{train} + D^S$ with 100 epochs of training. We see that the SqueezeDet network trained on D^S quickly begins to overtrain since its validation loss increases, whereas the network trained on $D^S + R_{train}$ has a constant validation loss and thus does not overtrain. The network trained on R_{train} has much more stable validation plots, presumably since it is trained on the same distribution it is tested on (unlike the other networks). It is hard however to draw any conclusions from Fig. 4.11 since the dataset sizes differ, meaning that the networks are trained on different amounts of images during each epoch.

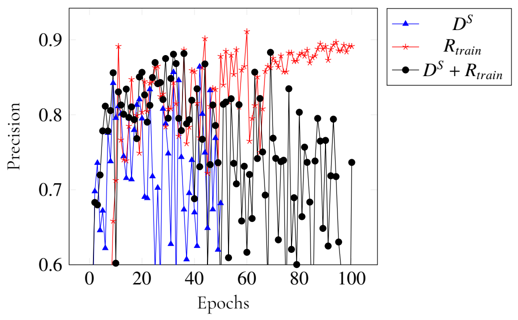
Fig. 4.12 shows validation loss, recall, precision and F_1-scores for two SqueezeDet networks evaluated on the real dataset R_{val}. The first network was trained on the dataset D^S and the second on the dataset S. Both networks are trained for 50 epochs. Inspecting the validation plots, they are very similar and roughly follow the same trajectory. In general the network trained on D^S seems to consistently have a somewhat better recall and precision (and consequently F_1-score) than the network trained on S. We do not see significant differences here, but analysing the test performances in Table 4.2 and 4.3, we see that the network trained on D^S has a significantly better F_1-score - nigh on three percentage points higher than the network trained on S.



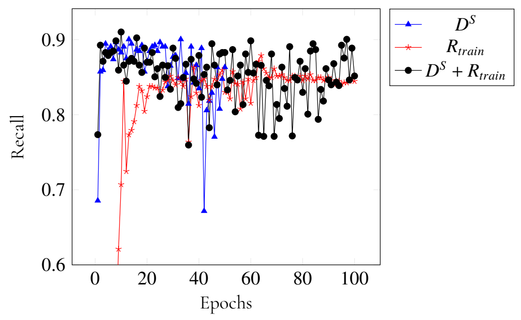
(a) Validation loss



(b) Validation F_1-score

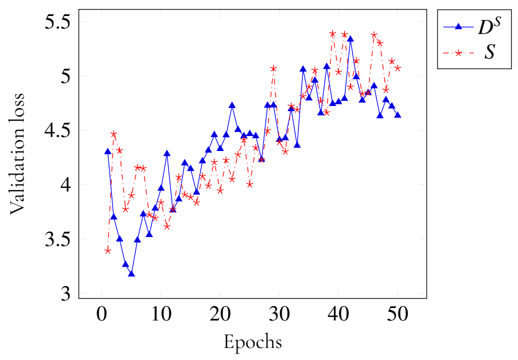


(c) Validation precision

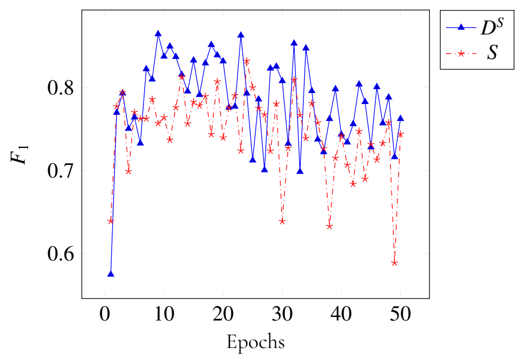


(d) Validation recall

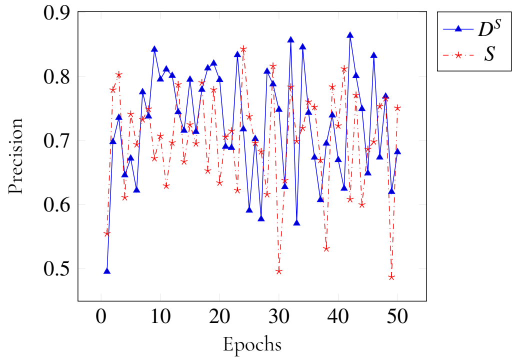
Figure 4.11: Plot of validation performance for D^S, R_{train} and $D^S + R_{train}$ for all epochs



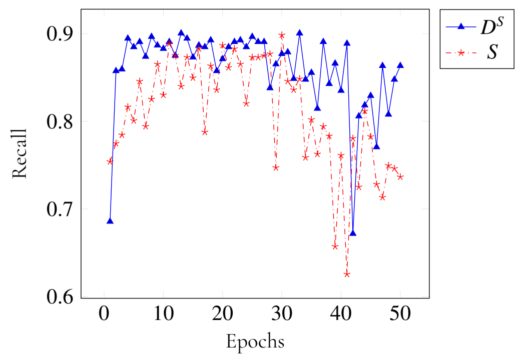
(a) Validation loss



(b) Validation F_1-score



(c) Validation precision



(d) Validation recall

Figure 4.12: Plot of validation performance for D^S and S for all epochs

Chapter 5

Discussion

5.1 CycleGAN

5.1.1 C_{CP}, C_{CP}^{S}

In Fig. 4.1 and Fig. 4.2, we see that C_{CP}^{S} retains the color scheme of the input images, whereas C_{CP} transforms the input into one color scheme. This illustrates the role of the skip connection, where C_{CP}^{S} is able to preserve more of the original structure than C_{CP}.

While none of these look truly realistic, we subjectively perceive C_{CP}^{S} to look more realistic than C_{CP}. This is also reflected by the Fréchet Inception Distance, where ${}^{c}C_{CP}^{S}$ is closer to ${}^{c}R_{train}$. In terms of SqueezeDet performance, we see that C_{CP}^{S} has a 0.02 higher F_1-score than C_{CP}, which is in agreement with the negative correlation between F_1-score and Fréchet Inception Distance to ${}^{c}R_{train}$ seen in Fig. 4.8.

Why C_{CP} has a lower F_1-score is likely due to the fact that it does not preserve the original image structure as much as C_{CP}^{S} does. Good examples of this can be seen in the fourth and fifth columns in Fig. 4.2, where the original structure of ${}^{c}S$ is hard to distinguish in ${}^{c}C_{CP}$.

Comparison with previous work

Compared to Wessman and Andersson's results in [1, p. 71], we see that our refinements made by a default implementation of CycleGAN (${}^{c}C_{CP}$) are much more deformed than their corresponding results. We suspect that this is due to Wessman and Andersson spending much more effort on rendering, making their synthetic dataset better approximate the real.

In contrast, most of our effort in this thesis has been directed towards GANs and little towards rendering settings, such as clothes and body shapes. We could therefore have observed better results if we, like Wessman and Andersson, dedicated more effort to producing a synthetic dataset that approximated the real dataset to a larger degree. Hypothetically, all our different GAN models would benefit from this since the translation task would be easier if the perceptual differences between the synthetic dataset and the real dataset were already small.

5.2 CycleGAN with VGG16 feature losses

5.2.1 C_{CPCFIP}, C_{CPCFIF}, C_{CPCF}, C^S_{CPCFIP}, C^S_{CPCFIF}, C^S_{CPCF}

Focusing on the three datasets trained with skip connections, we see that C^S_{CPCF} is different from C^S_{CPCFIP} and C^S_{CPCFIF} in terms of color. C^S_{CPCF} seems to have attained a dark green color palette characteristic of some images in R_{train}, while having lost realistic structure. This is probably due to the fact that this is the only network without an identity-loss. In other words, it is not forced to preserve the input image as much as the other two networks. For example, the white hair color of the fourth column in Fig. 4.1 has been painted black in ${}^cC^S_{CPCF}$, whereas it remains white for datasets with identity losses.

The same observations can be made of the dataset C_{CPCF}, which also has attained a color palette characteristic of R_{train}. C_{CPCFIF} and C_{CPCFIP} on the other hand manage to retain the color palette of the original synthetic images, just like the corresponding datasets trained with skip connections. We can thus observe that skip connections and identity losses in our application are equivalent in function: they both try to retain the structure of the input image. Identity losses are however a more hands-on approach, where the implementer decides the importance of structure preservation by regulating the weight in the loss. Skip connections on the other hand only nudge the network in a direction that encourages retained structure, but the network is not forced to do so.

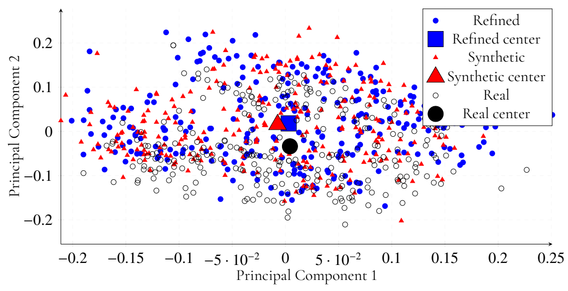
Looking at the Fréchet Inception Distance to R_{train} for the different datasets, we see that C_{CPCF} is the furthest from R_{train} with a distance of 169.3. This is likely due to the heavy transformations which can be observed in the row for C_{CPCF} in Fig. 4.2, where e.g. the image in the fourth column seems to have been replaced with a table and a stack of papers. The corresponding dataset C^S_{CPCF} is also heavily disfigured, arguably more so, but has a significantly better Fréchet Inception Distance of 154.86.

When comparing C_{CPCF} and C^S_{CPCF} with C_{CP} and C^S_{CP}, we observe that the cyclic feature loss is not enough to further nudge the GAN towards a good translation. Higher λ-values could potentially prove this wrong, but in our experience we found that strong cyclic losses swamped the adversarial losses resulting in no transformation at all.

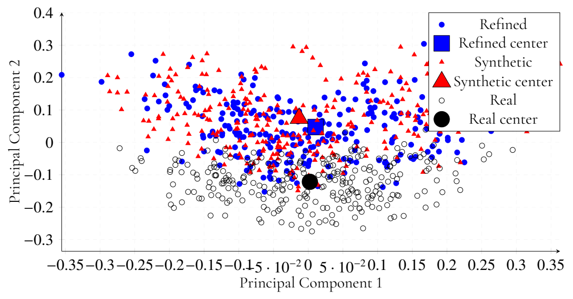
Among the six datasets we compare SqueezeDet performances and conclude that C^S_{CPCFIF} has the best F_1-score of 0.875.

Comparing C^S_{CPCFIF} and C^S_{CPCFIP} in Fig. 4.1 we see that the latter suffers from mis-coloring artifacts, which can be explained by the locality of the identity pixel loss. This issue is not present in C^S_{CPCFIF}, since its identity feature loss preserves textures on a higher level than pixels enforcing a more coherent transformation.

The dataset ${}^cC^S_{CPCFIP}$ has a VGG16 loss penalizing changes in the second pooling layer. Inspecting the PCA plot of the second max pooling layer for this dataset in Fig. 5.1 we see that the refined cluster center has barely moved from the synthetic cluster center. Looking at the PCA plot of the fifth layer, the refined cluster center has moved apart from the synthetic cluster center and started heading towards the real cluster center. This suggests that our loss penalization works.



(a) PCA plot of features from second max pooling layer of VGG16



(b) PCA plot of features from fifth max pooling layer of VGG16

Figure 5.1: Plots of the two principal components of VGG16 features for ${}^{c}C^{S}_{CPCFIP}$, ${}^{c}S$, and ${}^{c}R_{train}$. ${}^{c}C^{S}_{CPCFIP}$ and ${}^{c}S$ have been randomly sampled to match the size of ${}^{c}R_{train}$.

5.3 CycleGAN with a discriminator feature loss

5.3.1 D, D^S

In terms of both Fréchet Inception Distance and F_1-score, we see that D^S performs best among all different datasets with an F_1-score of 0.882 and a Fréchet Inception Distance of 147.82. The distinguishing feature of D^S is that a cyclic discriminator feature loss was used in the refinement.

Interesting to note is that D does not perform well at all in terms of F_1-score (even worse than S). This could imply that the skip connections between the generator's input and output layers were crucial to the success of D^S, but we are not positioned to guarantee such a strong statement. See Section 5.11 for more details on why.

Comparing ${}^{c}D$ to all other cropped datasets, we see in the second column of Fig. 4.2, ${}^{c}D$ is the only dataset where green color has been added to an input image which was not originally green. Comparing ${}^{c}S$ and ${}^{c}R_{train}$, we found that ${}^{c}S$ had more green-tinted images than ${}^{c}R_{train}$. It is thus quite surprising that any of the refined datasets have added green color to the transformation, since green color is more descriptive of the synthetic dataset than the real. One possible reason could be that the cyclic loss is too strong, swamping the adversarial loss. Meaning that instead of training the generator to refine the synthetic image to look real, it focuses on trying to restore to its original domain.

In our experiments, we used the same weight for the cyclic discriminator feature loss and the cyclic VGG16 feature loss. This could however be a naive decision, since they are two different vector spaces and thus have no guarantee of having the same range of distances. While D^S was a success, we could thus perhaps have seen increased performance with a more thorough parameter decision process.

5.4 CycleGAN with a mean feature loss of the target dataset

5.4.1 M, M^S

The mean feature loss dataset M had an F_1-score of 0.865, which is highest among all datasets without skip connections. Comparing F_1-scores of M and M^S we see that M outperforms M^S. We conclude that the skip connections are too restraining and will not allow for drastic changes.

The unique characteristic of these datasets is that they were refined using a mean feature loss, as described in Chapter 2.6. The image representation of the mean feature vector would most likely be nonsensical since it would just be the mean of the whole dataset, so enforcing the features of generator's output to strongly resemble it would be a bad idea. However, we thought it would make sense to penalize the distance to this vector slightly, such that the feature vectors of the refined images can be guided in which direction to transform.

In terms of PCA one could think of the mean feature vector as the cluster center for ${}^cR_{train}$. However, comparing M with D^S (see Fig. 4.10), we see no indication of the cluster center for M being closer to the cluster center of ${}^cR_{train}$ than D^S (see Fig. 5.3) - which could imply that the mean feature loss had no effect at all. It is however important to note that if the mean feature for M has not approached ${}^cR_{train}$ in 2D PCA space, it does not necessarily mean that it is not closer, since our two principal components only accounts for approximately 20 % of the explained variance.

Assuming that the mean feature loss had no effect, the most probable cause is too small a weight being used. To verify this we increased the weight a disproportionate amount (from 10^{-5} to 10^{-1}) and created the new refined dataset ${}^cM_{new}$ which we performed PCA on. The results are presented below.

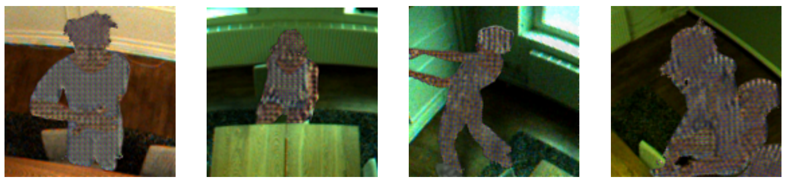
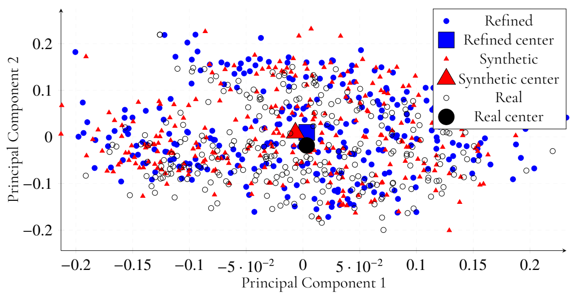


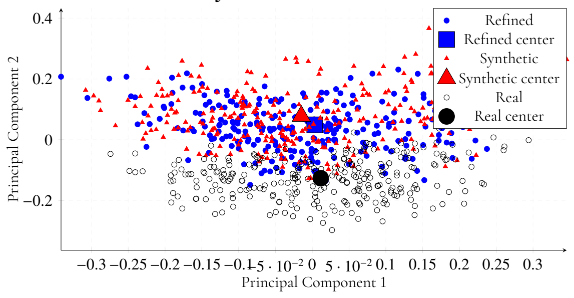
Figure 5.2: Random samples from ${}^cM_{new}$.

In Fig. 5.2 we see the results of this approach. The refined images have been washed out in color resulting in a gray palette. Further, we see a grid-like texture on the bodies. While a somewhat constant transformation is what we expected, we did not expect the results seen in Fig. 5.4. We expected the refined cluster center to gravitate towards the real cluster center, since that is the only way for the network to avoid the penalization of the mean feature loss. Instead, we observe the opposite. The refined cluster has moved away significantly from both the real and synthetic clusters, especially for the fifth max pooling layer features in 5.4.

Unable to interpret the reason for this result, we inspected the loss function of the GAN and observed that the mean feature loss increased during training! This explains the results shown in Fig. 5.4, but we do not know why the loss increased during training. Since we increased the weight dramatically, effectively swamping the other losses, we have no explanation for why it increased. Intuitively it should have at least plateaued and not increased. An immediate suspicion was that we had flipped the sign of the loss, meaning that we incentivized the network to maximize the mean feature distance. This would be a satisfactory explanation, but upon inspection we saw no evidence suggesting that we had done so.

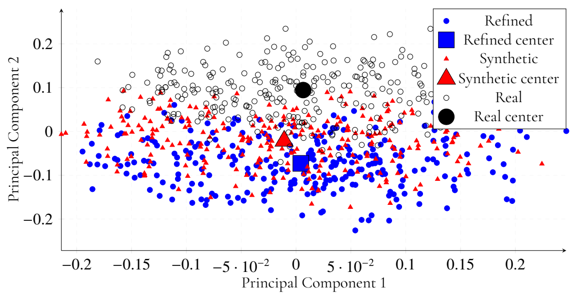


(a) PCA plot of features from second max pooling layer of VGG16

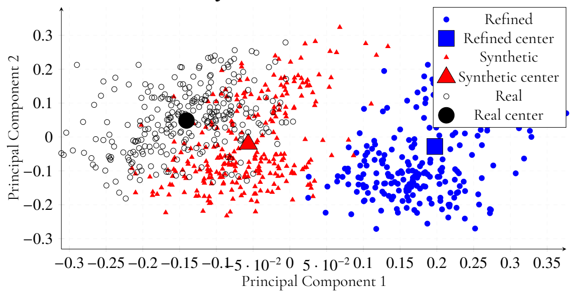


(b) PCA plot of features from fifth max pooling layer of VGG16

Figure 5.3: Plots of the two principal components of VGG16 features for ${}^{c}M$, ${}^{c}S$, and ${}^{c}R_{train}$. ${}^{c}M$ and ${}^{c}S$ have been randomly sampled to match the size of ${}^{c}R_{train}$.



(a) PCA plot of features from second max pooling layer of VGG16



(b) PCA plot of features from fifth max pooling layer of VGG16

Figure 5.4: Plots of the two principal components of VGG16 features for ${}^{c}M_{new}$, ${}^{c}S$, and ${}^{c}R_{train}$. ${}^{c}M_{new}$ and ${}^{c}S$ have been randomly sampled to match the size of ${}^{c}R_{train}$.

5.5 Attention-guided GAN

5.5.1 A, A^S

In the second column in Fig. 4.1, we see an interesting phenomenon in ${}^{c}A^{S}$, which we show in detail below in Fig. 5.5. The regularly occurring dots of pixels seen in the refined image are a symptomatic sign of fractional stride artifacts [42], where the fractionally strided convolution layers have not learned to realistically upsample the image features into an image. Why we only see this effect in ${}^{c}A^{S}$ and not in any other dataset is unclear, but since the upsampling procedure is learnable it is clearly due to the generator being inhibited in its learning. One possible reason for the inhibition of the generator's learning in ${}^{c}A^{S}$ could be the corresponding attention network. If we review the attention maps of ${}^{c}A^{S}$ in Fig. 4.6, we see that there is high uncertainty in the foreground attention. With low confidence in the attention map, the generator would receive weak gradients which would then slow down the learning of the generator, which could explain the artifacts.

As opposed to ${}^{c}A^{S}$, we see in Fig. 4.6 that the attention maps of ${}^{c}A$ are more confident (even though the foreground is not correctly identified). We suspect that this is the reason why we do not see any fractionally strided artifacts in ${}^{c}A$: its gradients have not been inhibited by an uncertainty of its attention network.

Why the attention maps of ${}^{c}A$ are more confident than those of ${}^{c}A^{S}$ is still up for discussion. The only difference between the two datasets is the additional generator skip connection in ${}^{c}A^{S}$, but we fail to see why this would inhibit the learning of the attention maps. Instead we suspect that the difference is due to different weight initializations, which according to the authors of [21] can play a major role in convergence [27].

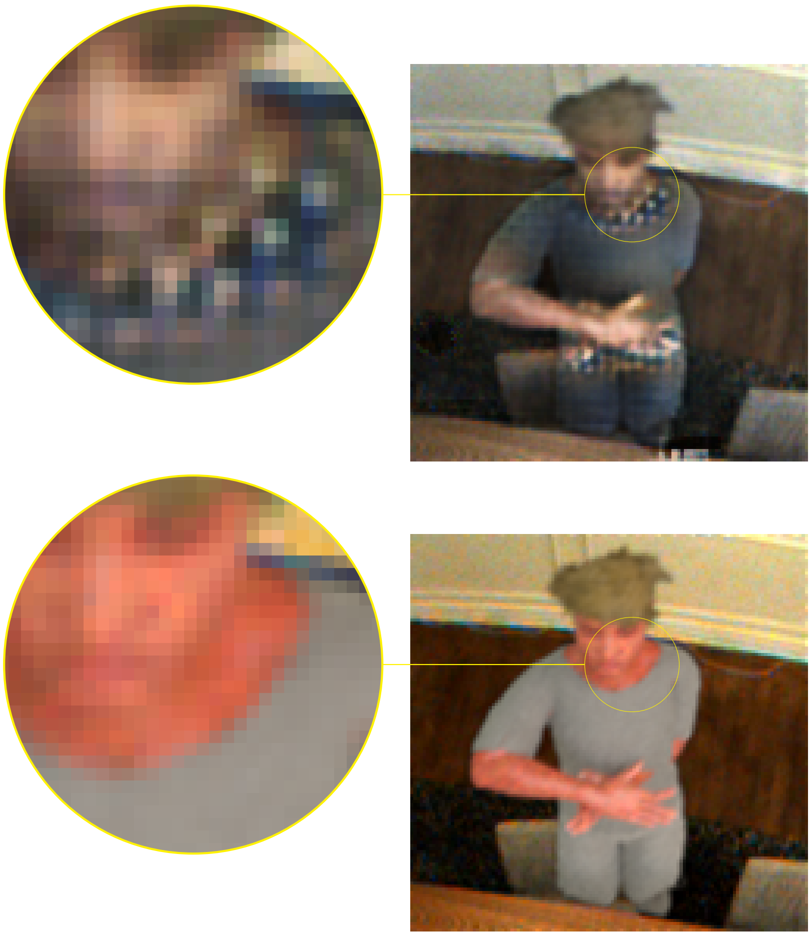
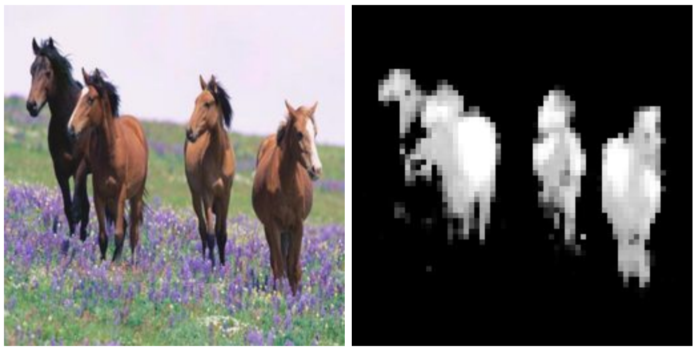


Figure 5.5: Fractional stride artifact. The upper image shows the refined synthetic human and the lower image shows the original synthetic human.

Why the attention networks failed

We see that the attention networks of A and A^S somewhat reliably identify the synthetic humans. However, compared to other datasets, such as horse2zebra, the confidence in the attention masks is very low. For instance, compare the attention masks in Fig. 4.6 with Fig. 5.6 below. We see that the network trained on horse2zebra produces much more confident attention masks (it does not attend any background) than the network trained on our dataset, which can not confidently exclude the background from its attention.



(a) Input **(b)** Attention mask

Figure 5.6: The attention network is confident. While it fails to attend the full body of one of the horses, it does not attend any background information.

When prospecting common image translation datasets, e.g. horse2zebra and apples2oranges, there is a lot of background variety. In contrast, our backgrounds are all 128x128px crops of a single location.

Our hypothesis was that given a domain with lots of different backgrounds, the network is more confident in identifying the foreground since it is the only common feature between the images. If however the foreground is not the only common feature in the domain, we hypothesized that the attention networks are unable to confidently exclude the background from their attention since it is too descriptive a feature.

To verify that a lack of background variety is the cause of the attention networks' uncertainty, we created toy datasets of circles. The circles were generated with varying size, color, and position. For each dataset, we created two domains X and Y, where we were interested to see where the attention network A_X put its attention. In all datasets, X and Y share background colors but not foreground colors. Y has always white foreground color, while X has either constant or varying foreground color.

Fig. 5.7 shows samples of both domains from all datasets.

We find that if X has constant background color and various foreground colors (X_3 in Fig. 5.8), the attention network A_X learns to mask the background. Instead, if X has various background color and constant foreground color (X_1 in Fig. 5.8) A_X correctly learns to mask the foreground.

This is similar to what can be observed in our datasets. In horse2zebra, there is high background variance (different location in each image) and low foreground variance. In our dataset, there is low background variance (128x128px crops from one location) and high foreground variance (many different human models with highly variant features).

As described in [21] in their paper, the attention networks give attention to features that describe the dataset. While humans might say that circles, regardless of color, describe our toy datasets, we find that this is not always in line with the attention network. Given constant background color and varying foreground color, it is the background that is the describing feature of the dataset. However if there is more background variety than there is foreground variety, then the foreground will be the describing feature.

With this result in mind, we created a dataset of synthetic and real humans on various backgrounds in order to increase the background variety. However, contrary to our expectations, the attention networks could still not confidently mask our foreground. We are not completely sure why, but suspect that it is because the human foreground still has too large

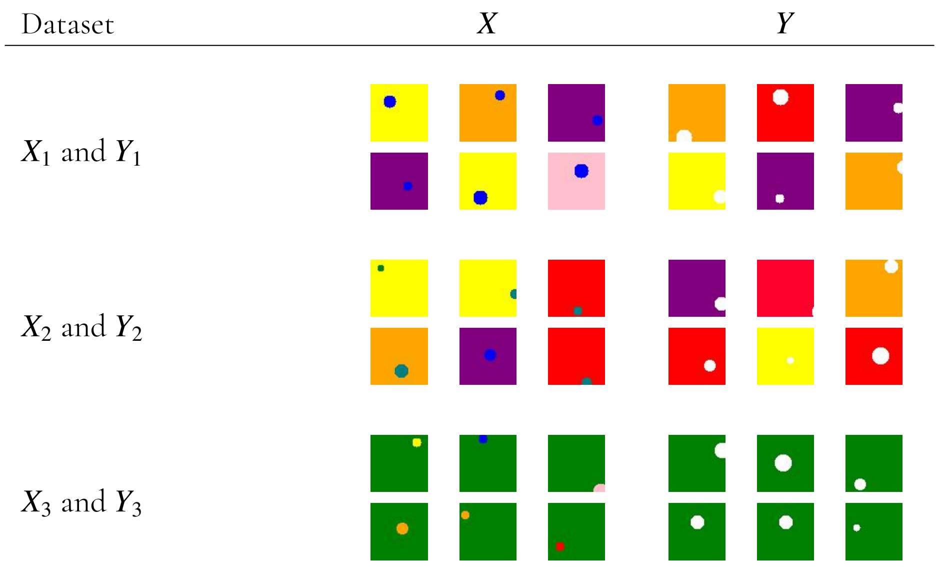


Figure 5.7: Samples from the toy datasets $(X_i, Y_i), i \in \{1, 2, 3\}$. X_1 has constant foreground and varying backround, X_2 has slighly varying foreground and varying background, and X_3 has highly varying foreground and constant background. All datasets Y_i have white foreground and the same sorts of background as their corresponding domain X_i.

variance for the attention networks to cope.

We also tried replacing the attention network A_X that tries to mask the synthetic humans with the ground truth masks produced by Blender. This did not help the network in any perceivable way, supposedly because it defeats the whole point of the architecture in [21] where the authors make a point of continuous attention maps being necessary for learning. With binary ground truth masks, the generator would receive a gradient of zero for all transformations performed outside the mask, whereas for a continuous mask it would be nonzero.

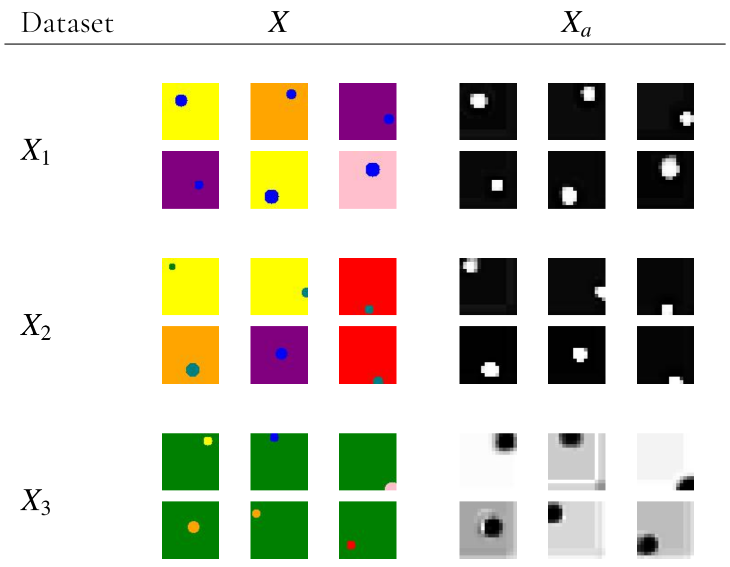


Figure 5.8: Samples from $X_i, i \in \{1, 2, 3\}$, and their respective attention maps, of different variations on the toy dataset. Trained on AGGAN for 1 epoch.

5.6 Why the refinement increases object detection performance

As shown by Geirhos *et al.* in [43], convolutional neural networks (pre-)trained on ImageNet [40] are biased to recognize textures rather than shapes. Following this insight, they style transferred ImageNet to a set of different styles for use as training data (let us call this stylized dataset ImageNet$^+$) in order to force the network to be more biased towards shapes rather than texture (see Fig. 5.9).

They then trained ResNet [28] on ImageNet + ImageNet$^+$, and fine-tuned the training on ImageNet. With this approach they were able to increase the accuracy of ResNet compared to when trained only on ImageNet. They also showed significant performance increases on the Pascal VOC object detection dataset [44], when using transfer learning from their ResNet network trained on ImageNet + ImageNet$^+$.

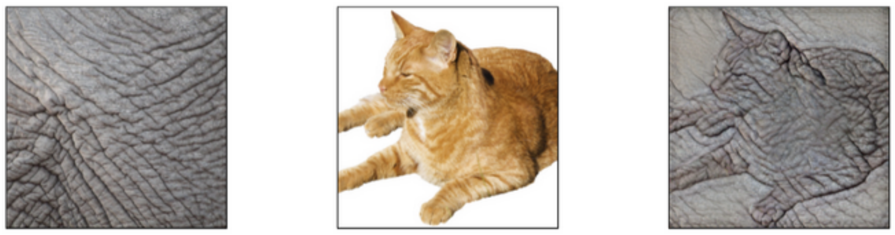


Figure 5.9: Result of style transfering an image of a cat with the texture of an elephant's skin, as presented in [43].

This could have significant ramifications for our thesis. With the results of [43] in mind, the refined dataset could possibly perform well due to increased variance in texture forcing the filters of the network to detect shapes rather than textures. If true, this could imply that our well-performing refined datasets did not get closer to R_{train}.

To investigate whether it was increased variance rather than realistic texture that caused our performance increases, we used the same set of paintings as used in [43] and blended them with our synthetic humans. This resulted in a dataset S_{style} with extremely high texture variance, but with very low texture resemblance to R_{train} (Fig. 5.10). Our hypothesis was then as follows. If the test performance on R_{test} of a network trained on S_{style} yielded good results, we probably only increased the variance of the synthetic dataset and did not manage to get the refined texture to look like that of R_{train}. Whereas if the results were bad, we probably approximated the texture of R_{train} to at least some degree.

Note however that this simple test does not answer whether it could be a combination of high variance and texture resemblance that is responsible for the performance increases, which is the case in [43].

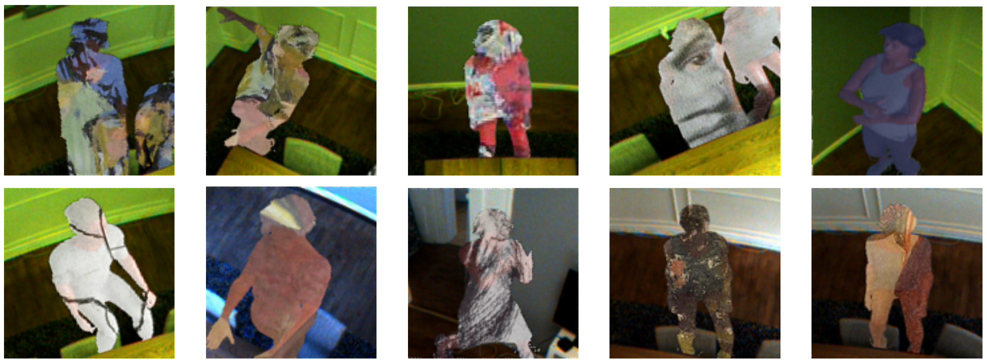


Figure 5.10: Random samples of cropped images from S_{style}.

	Epoch	Precision	Recall	F_1
S_{style}	7	0.741	0.786	0.762

Table 5.1: SqueezeDet test results on R_{test}, trained on S_{style} for 50 epochs. The checkpoint was selected by maximizing the F_1-score on R_{val}.

As seen in Table 5.1, the stylized synthetic dataset S_{style} has decreased performance: it has a lower F_1-score than the synthetic dataset S and all refined datasets. This implies that our increased performance when using refined synthetic training data is due to increased texture resemblance. This conclusion is also strengthened by the negative correlation between the F_1-score on R_{test} and the Fréchet Inception Distance to ${}^cR_{train}$, seen in Fig. 4.8, which also implies that texture resemblance yields better detection performance.

In Fig. 4.9, we see that there is also a negative correlation between Fréchet Inception Distance to the synthetic dataset ${}^cS$ and the F_1-score on R_{test}. We believe that this correlation is misleading and that this does not hold in general. It only appears this way since all of our refined datasets that are far from S have deformed drastically, leading to a low object detection performance. Had we truly approximated R_{train}, the distance to S would naturally be increased while the F_1-score on R_{test} would increase. Thus, theoretically, there should exist a positive correlation between the Fréchet Inception Distance to S and the F_1-score on R_{test}, assuming well-approximating refinements - not a negative!

5.7 Does real training set size matter?

In our experiments we used a very conservative dataset split, where $|R_{train}| = 102$, in order to increase training speed and decrease our reliance on supervised data. For such a small dataset, we showed improved performance by adding refined synthetic data. But what if R_{train}, the dataset used for refinement as well as training the SqueezeDet network, is larger? Is there still a performance increase?

To investigate this we inversed our data split, such that $|R_{val_{new}}| = 20$, $|R_{test_{new}}| = 80$, and $|R_{train_{new}}| = 719$. We then refined S with the same configurations used in D^S and tested the performance of the new refined dataset D^S_{new} with SqueezeDet. The number of training epochs for CycleGAN was adjusted, due to the increased training set size, such that the number of training steps was identical to before. We then trained SqueezeDet on $R_{train_{new}}$, D^S_{new} and $D^S_{new} + R_{train_{new}}$.

With this approach we found no improvement when comparing the performance of D^S_{new} and $D^S_{new} + R_{train_{new}}$ with $R_{train_{new}}$ (see Table 5.2). This suggests that our performance increases presented in Chapter 4 were due to the small size of R_{train} and that our approach does not generalize to larger datasets where the object detection performance of real data has already saturated.

However, it is possible that our refinement parameters are dependent on dataset size, i.e. the refinement parameters of D^S are not optimal for D^S_{new}. If true, this could imply that our approach is able to generalize to larger datasets after all. But we do not investigate this, due to time constraints.

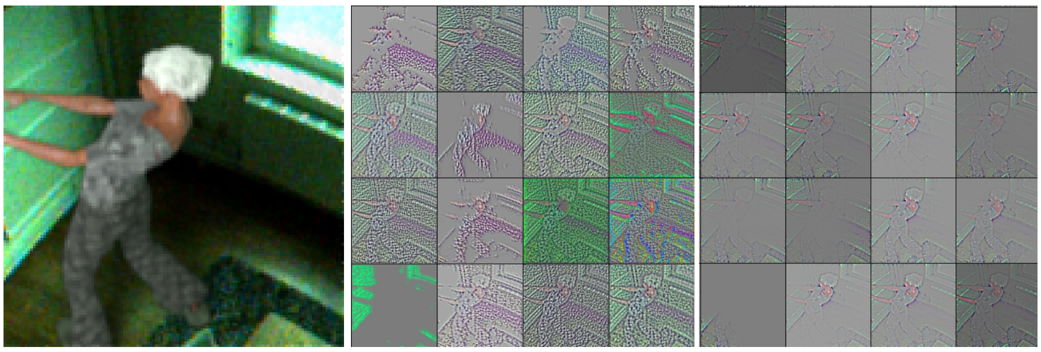
Also interesting to note is that Fig. 4.7 implies that the performance of $R_{train} + D^S$ is maximized when $N = 15, N \in \{0 \dots 15\}$ where $|D^S| = N \cdot |R_{train}|$. In this investigation, the size of D^S_{new} is roughly $2 \cdot |R_{train_{new}}|$. The performance of $D^S_{new} + R_{train_{new}}$ could thus hypothetically be increased if the size of D^S_{new} is increased, assuming that the observations seen in Fig. 4.7 are independent of dataset size. But again, we do not investigate this.

	Epoch	Precision	Recall	F_1
$R_{train_{new}}$	16	0.899	0.977	0.936
D^S_{new}	26	0.847	0.888	0.867
$D^S_{new} + R_{train_{new}}$	32	0.870	0.959	0.913

Table 5.2: SqueezeDet test results on $R_{test_{new}}$, trained on D^S_{new}, $R_{train_{new}}$, and $D^S_{new} + R_{train_{new}}$.

5.8 Which layers from VGG16 should be used

Engin *et al.* [23] and Zeiler and Fergus [24] both claim that shallow layer filters activate on low-level features and deep layer filters activate on high-level features. In Fig. 5.11 we show that this also holds true for our dataset.



(a) Synthetic input image **(b)** Second max pooling layer **(c)** Fifth max pooling layer

Figure 5.11: Visualized feature activations for different max pooling layers of VGG16 [45].

In Fig. 5.11b we see that the second max pooling layer's activations are triggered by texture, edges and color. In Fig. 5.11c we see that the body and the wall edges are the most prominent features, implying that the fifth max pooling layer's activations are triggered by more complex shapes.

In Chapter 4, all datasets with VGG16 losses used second max pooling layer activations, except M and M^S where second and fifth layer activations were used. As per [23] we originally wanted to use the second and fifth layers for all datasets, but due to an oversight in our implementation we ended up using only the second layer for most datasets.

In an attempt to remedy this, we present below the results of using the same configurations as D^S but with either only the fifth pooling layer or both the second and the fifth.

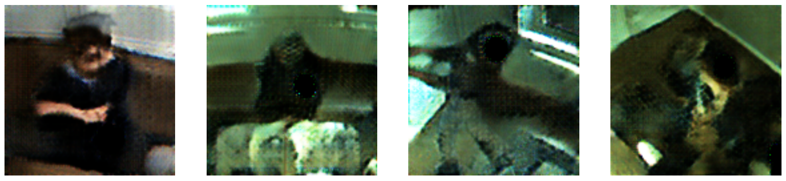


Figure 5.12: Samples from ${}^{c}D^{S}_{VGG5}$, where the VGG16 feature loss used the fifth pooling layer.

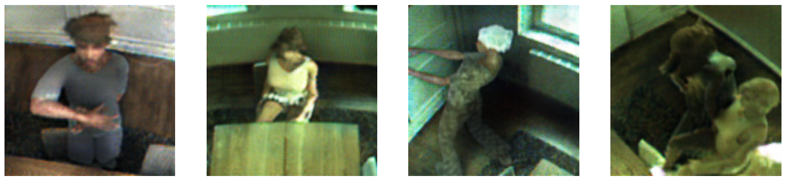


Figure 5.13: Samples from ${}^{c}D^{S}_{VGG5-2}$, where the VGG16 feature loss used the fifth and second pooling layers.

In Fig. 5.12 we see that using only the fifth layer activations was not restrictive enough. The abstract shapes in the images are retained, but the texture is transformed unrealistically. The lack of realism is also reflected by the quantitative SqueezeDet results where the F_1-score on R_{test} was 0.824.

In Fig. 5.13 we see that the low level features are retained much stronger than in Fig. 5.12. Subjectively, these look very similar to the results of D^S, but quantitatively the results are worse with an F_1-score of 0.841.

It is important to keep in mind, in spite of these results, that we have optimized our loss hyperparameters for the datasets presented in Chapter 4. This comparison is thus not entirely fair, since a different set of parameters could increase the performance of these attempts.

Had we had the chance to redo this part, we would have optimized our hyperparameters to only use the fifth max pooling layer activations in our cyclic and identity losses, since we want to retain high-level features while transforming low-level features. In our datasets with VGG16 losses, we have penalized the networks for transforming low-level features - which is in conflict with our intuition of the problem.

5.9 Skip connections

In our experiments we have used a very simple skip connection, demonstrated in Fig. 2.15, connecting only the input and output layer of the generator. We find that convergence speed is faster when the skip connection is added. This is quite intuitive, since the generator's task then becomes an additive transformation "on top" of the input instead of a ground-up transformation of the input.

Aside from convergence speed, we find that datasets trained with skip connections on average also have better quantitative results (Table 4.7). This is also in agreement with our subjective perception where, in Fig. 4.1 and 4.2, structure is retained much better in datasets trained with skip connections.

5.10 Did we achieve photorealism?

Compared to the synthetic images, we do (to our mind) increase the subjective realism for our best samples. Lighting is more varied with more prominent shadows and the skin tone looks less orange.

As for the question "did we achieve photorealism?", the answer is no. While some of our refined images could possibly fool humans (see Fig. 4.3), we do not think any of the randomly sampled images in Fig. 4.1 and 4.2 could fool a human.

The generated samples being far from real-looking is also confirmed by the quantitative results. If we look at Fig. 4.11, we see that SqueezeDet quickly begins to overtrain on D^S since the validation loss increases after only a few epochs of training.

Further, with our crop-refine-reinsert approach we introduce two kinds of artifacts that heavily reduced the perceived realism of our end results. First, when overlapping crops have conflicting refinements. This produces sharp edges where the cropped images meet, as demonstrated in Fig. 5.14. Second, when the $128x128$px crops do not encompass the whole body of the synthetic human, resulting in a sharp border between the original synthetic human and the refined synthetic human (see Fig. 5.15).

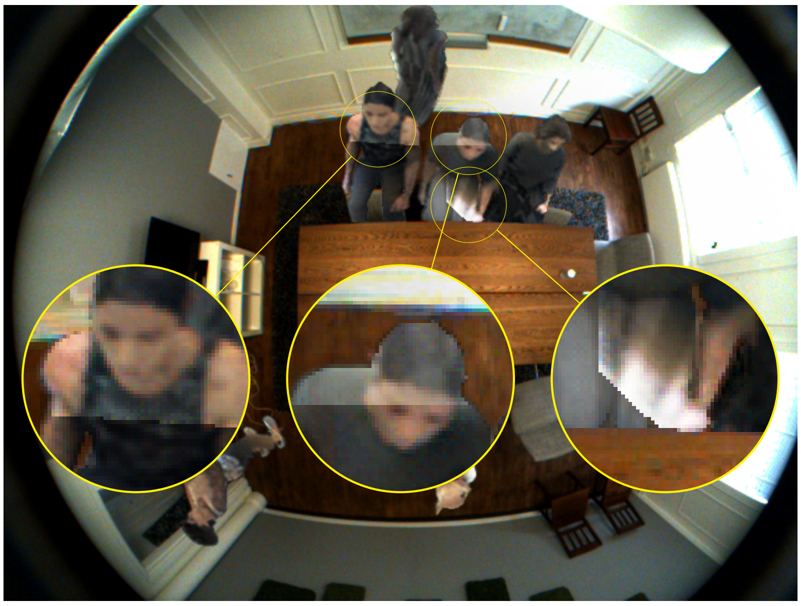


Figure 5.14: Reinsertion artifacts where overlapping image crops have conflicting refinements, resulting in sharp borders.

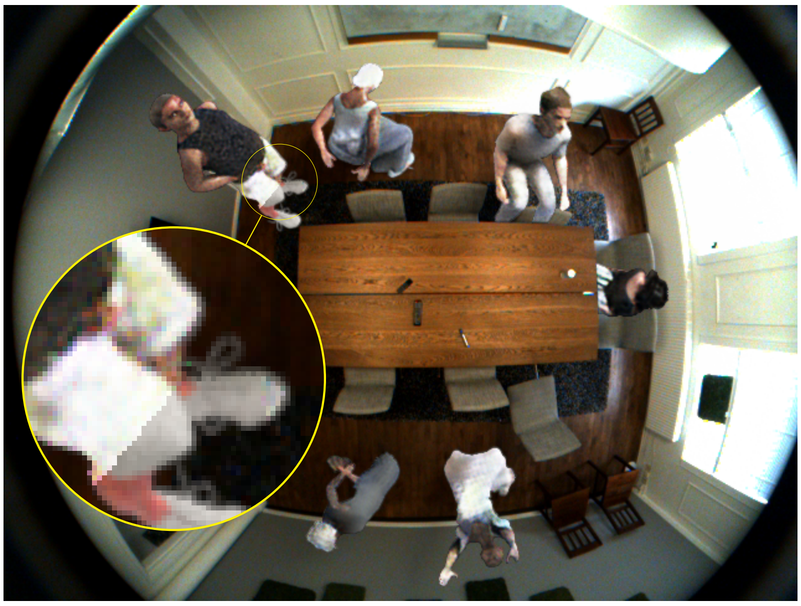


Figure 5.15: Reinsertion artifact where an incomplete portion of the synthetic human has been refined, resulting in a sharp border.

5.11 Reproducibility

Recall that training GANs is not an optimization of one function, but an attempt to optimize a two-player game. GANs are thus notoriously hard to train, compared to other machine learning models, and can have quite unstable training procedures. In our discussion we operate under the assumption that the different results we have recorded are due to different architectures and losses. It should however be noted that stochasticity can also play a major role in our results, thus decreasing the confidence of our architectures' contributions to our results. To ensure reproducible results, we could run each training multiple times, but we have not had time to do so for each configuration. Instead we only do so for D^S.

	Epoch	**Precision**	**Recall**	F_1
$D^S_{reproduction}$	7	0.841	0.878	0.859

Table 5.3: SqueezeDet test results on R_{test}, trained on $D^S_{reproduction}$.

Table 5.3 shows the test results of $D^S_{reproduction}$. The performance is significantly lower than the performance of D^S (see Table 4.3), but slightly better than S. With this result in mind, we can conclude that there is large variance in the refinement result even with fixed parameters. If we retrained all networks, we could therefore very likely observe different results.

5.12 Shortcomings

While an increase from 0.865 (R_{train}) to 0.888 (D^S + R_{train}) in F_1-score is a satisfactory result, it is important to note that this is with different training set sizes: D^S comprises 1498 images while R_{train} comprises 102 images. One might think this is an unfair comparison, since the refined dataset is roughly 15x larger than the real dataset, but recall that the only supervised data required to produce our best performing dataset D^S is R_{train} and unpopulated backgrounds of the same environment as R_{train}. Thus we think it is perfectly fair to let the refined synthetic dataset use its advantage in size to make up for its lack of realism.

In order to do fair comparisons between our different GAN models, we tried to freeze as many parameters as possible, e.g. number of training epochs (see Section 3.3.3). This helps to ensure the integrity of our comparisons, but it is also unfair since it disregards differences in convergence speed. The performance of our datasets could look very differently if we had done a subjective approach where we investigated each model and picked the number of training epochs manually. For future works, we recommend using a Wasserstein loss [46] in order to ensure equal convergence for all models. For increased training stability, an architecture such as PGGAN [47] would also be preferable when comparing the effect of different losses.

Chapter 6
Conclusions

6.1 Conclusion

In essence, the purpose of this thesis was to transfer the style of the real dataset $\boldsymbol{R}_{train}$ to the synthetic dataset $\boldsymbol{S}$, while preserving the overall structure of $\boldsymbol{S}$, in order to create new training data.

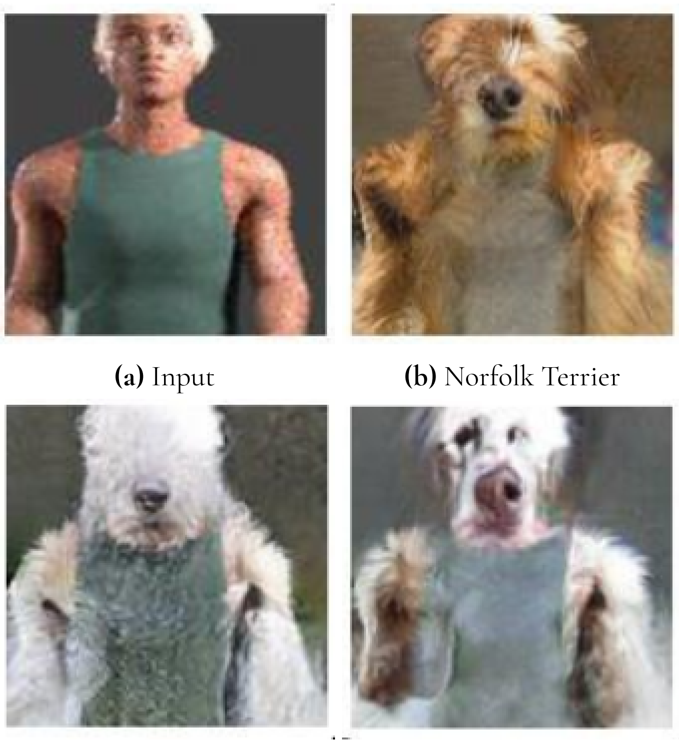
We attempted to do this with various GAN architectures, all based on CycleGAN, and achieved quantitative improvements with some of our GAN models, when the size of the real training dataset was small. We did however not achieve the original goal of synthetic photo-realism. We believe that the underlying discriminator and generator architectures of our experiments were too "weak" for our problem. For better results, we have two suggested improvements: 1) experimenting with other architectures (see Section 6.2 for some suggestions); and 2) put more effort on rendering a realistic synthetic dataset, such that the image-to-image translation task becomes easier.

While we initially wanted a completely unsupervised approach, we ended up requiring a semi-supervised approach since we need positional annotations of real humans to create ${}^{c}\boldsymbol{R}_{train}$. However, a benefit of using our approach with cropped images is decreased training time and a decrease in memory usage.

While unfit for industrial use in its current state, our approach seems agnostic in terms of which GAN is used in the Refiner block. Further research could thus be done easily by simply replacing the GAN architecture.

6.2 Future work

In this thesis we have laid much focus on different losses and variations of CycleGAN [10]. For future work, it would be interesting to focus on how different generator and discriminator architectures can help in this field. One example is FUNIT [48], a recently proposed paper where remarkable results are achieved with GAN image-to-image translation, without access to large datasets. The code of FUNIT has yet to be published, but we have great expectations for this architecture. In Fig. 6.1, we show samples generated by FUNIT with a synthetic human as input. We see that structure is preserved, such as pose and face position, while the texture is completely transformed. Note that this model has been trained on translating animals and that it has never seen similar images.



(a) Input **(b)** Norfolk Terrier

(c) Bedlington Terrier **(d)** Staffordshire Bullterrier

Figure 6.1: Image translations by FUNIT [48] with a synthetic human as input. The model is trained on close up images of animal faces and is able to generalize to the synthetic human, without ever having seen similar images.

While we abandoned the concept of translating full-size images rather quickly, it could also be interesting to continue on that path and see if it has any promise. One approach would be to decrease the memory usage by training the refiner on smaller full-sized images and then use our trained refiner on original sized images. Since the architecture of the generator is fully convolutional meaning that it can take any image size, this would be possible. However, there is no guarantee that the refinement for the larger images would work as well as the refinement for the smaller ones.

As for rendering, one interesting approach would be to utilize TensorFlow Graphics [49] in order to estimate the rendering parameters based on how real images look. This could potentially eliminate the need for GANs altogether, or at least provide synthetic data that looks more realistic from the start.

Another approach could also be conditional image generation as opposed to image translation. The input would be a noise vector, a conditional background image, and conditional bounding boxes. The output would be an image with humans generated on the background at the positions specified by the conditional bounding boxes. This approach would however also require an annotated set of real data, in order for the discriminator to learn the relation between bounding boxes and humans.

6.3 Ethics

It is common hearsay that data is the new oil [50]. Accepting that premise, it is not a big leap to draw parallels between the internet giants of today and Big Oil. The concentration of power that can be observed in companies such as Google, Facebook, and Amazon also implies a concentration of data. Users give companies data in exchange for services. The companies then use it to improve their services which in turn attracts more users, who in turn give more data. Small companies are thus hard-pressed to compete with large companies in services based on, or enhanced by, machine learning, since they can not compete in amounts of training data.

One way to reduce this concentration of power is to democratize data. This could be done by publishing existing private datasets, but such an approach seems economically and legally unfeasible. A more promising alternative is to generate data instead of collecting it. If the generated data approximates the real to the degree that it can not be distinguished from real data, it can then can be used to train machine learning models. This would give anyone, regardless of their current amount of data, the power to utilize machine learning techniques.

Data democratization could thus prove useful for competition, but it could also enhance destructive actors' ability to produce harm. A recent example is the occurrence of "deep fakes": videos of humans where generative models have been used to swap faces. As soon as the technique was released, degrading videos appeared where faces of celebrities had been inserted. While such a technique could be very useful in e.g. replacing expensive CGI animations, there is a plethora of harmful uses: faking video evidence, defamation, and misinformation to name a few. In Fig. 6.2 we show an example of this technique, where the face of one person has been replaced by another's.

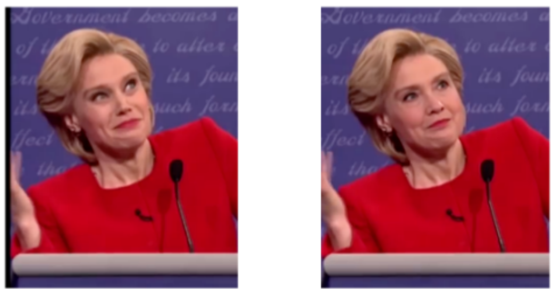


Figure 6.2: A "deep fake" translation, as seen in [51]. The left image is the original and the right is the translation result.

With synthetic data, the above could also be performed, but with a synthetic source video instead of a real video. While realism in synthetic-to-real video-to-video translation has not yet publicly been shown, we believe that it is plausible in the near future. If realistically produced, this could have significant ramifications for society, since the production of e.g. fake news could then be automated without human supervision, enabling an arbitrary production speed.

Bibliography

[1] D. Wessman and P. Andersson, *Generation of artificial training data for deep learning*, eng, Student Paper, 2018.

[2] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2242–2251. DOI: 10.1109/CVPR.2017.241.

[3] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 129–137.

[4] F. Pollastri, F. Bolelli, R. Paredes Palacios, and C. Grana, "Improving skin lesion segmentation with generative adversarial networks", in *2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS)*, 2018, pp. 442–443. DOI: 10.1109/CBMS.2018.00086.

[5] C. Bowles, L. Chen, R. Guerrero, P. Bentley, R. N. Gunn, A. Hammers, D. A. Dickie, M. del C. Valdés Hernández, J. M. Wardlaw, and D. Rueckert, "GAN augmentation: Augmenting training data using generative adversarial networks", *CoRR*, vol. abs/1810.10863, 2018. arXiv: 1810.10863. [Online]. Available: http://arxiv.org/abs/1810.10863.

[6] D. Kornish, S. Ezekiel, and M. Cornacchia, "DCNN augmentation via synthetic data from variational autoencoders and generative adversarial networks", in *2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, 2018, pp. 1–6. DOI: 10.1109/AIPR.2018.8707390.

[7] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?", in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 746–753. DOI: 10.1109/ICRA.2017.7989092.

[8] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization", in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, pp. 1082–10 828. DOI: `10.1109/CVPRW.2018.00143`.

[9] S. Hinterstoisser, V. Lepetit, P. Wohlhart, and K. Konolige, "On pre-trained image features and synthetic images for deep learning", *CoRR*, vol. abs/1710.10710, 2017. arXiv: `1710.10710`. [Online]. Available: `http://arxiv.org/abs/1710.10710`.

[10] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks", in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets", in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[12] Y. Bengio, E. Laufer, G. Alain, and J. Yosinski, "Deep generative stochastic networks trainable by backprop", in *International Conference on Machine Learning*, 2014, pp. 226–234.

[13] Y. Bengio, L. Yao, G. Alain, and P. Vincent, "Generalized denoising auto-encoders as generative models", in *Advances in Neural Information Processing Systems*, 2013, pp. 899–907.

[14] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks", *CoRR*, vol. abs/1701.00160, 2017. arXiv: `1701.00160`. [Online]. Available: `http://arxiv.org/abs/1701.00160`.

[15] E. Sjöstrand and J. Jönsson, *Cell image transformation using deep learning*, eng, Student Paper, 2018.

[16] M. Mirza and S. Osindero, "Conditional generative adversarial nets", *CoRR*, vol. abs/1411.1784, 2014. arXiv: `1411.1784`. [Online]. Available: `http://arxiv.org/abs/1411.1784`.

[17] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]", *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[18] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4.

[20] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution", in *European conference on computer vision*, Springer, 2016, pp. 694–711.

[21] Y. A. Mejjati, C. Richardt, J. Tompkin, D. Cosker, and K. I. Kim, "Unsupervised attention-guided image to image translation", *CoRR*, vol. abs/1806.02311, 2018. arXiv: `1806.02311`. [Online]. Available: `http://arxiv.org/abs/1806.02311`.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Online]. Available: `http://arxiv.org/abs/1409.1556`.

[23] D. Engin, A. Genç, and H. Kemal Ekenel, "Cycle-dehaze: Enhanced cyclegan for single image dehazing", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 825–833.

[24] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", in *European conference on computer vision*, Springer, 2014, pp. 818–833.

[25] C. Wang, C. Xu, C. Wang, and D. Tao, "Perceptual adversarial networks for image-to-image transformation", *IEEE Transactions on Image Processing*, vol. 27, no. 8, pp. 4066–4079, 2018.

[26] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning", *arXiv preprint arXiv:1603.07285*, 2016. [Online]. Available: `https://arxiv.org/abs/1603.07285`.

[27] Y. A. Mejjati, C. Richardt, J. Tompkin, D. Cosker, and K. I. Kim, *Unsupervised attention-guided image to image translation*, `https://github.com/AlamiMejjati/Unsupervised-Attention-guided-Image-to-Image-Translation`, Accessed: 2019-05-29, 2018.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: `10.1109/CVPR.2016.90`.

[29] M. Buckland and F. Gey, "The relationship between recall and precision", *Journal of the American society for information science*, vol. 45, no. 1, pp. 12–19, 1994.

[30] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium", in *Advances in Neural Information Processing Systems*, 2017, pp. 6626–6637.

[31] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans", in *Advances in neural information processing systems*, 2016, pp. 2234–2242.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[33] B. Foundation, *Blender*, Accessed: 2019-02-26. [Online]. Available: `https://www.blender.org/`.

[34] MB-Lab-Community, *Mb-lab*, Accessed: 2019-02-26. [Online]. Available: `https://github.com/animate1978/MB-Lab`.

[35] C. M. G. Lab, *Cmu graphics lab motion capture database*, Accessed: 2019-02-26. [Online]. Available: `http://mocap.cs.cmu.edu/`.

[36] S. Mo, M. Cho, and J. Shin, "Instagan: Instance-aware image-to-image translation", *CoRR*, vol. abs/1812.10889, 2018. arXiv: `1812.10889`. [Online]. Available: `http://arxiv.org/abs/1812.10889`.

[37] "Mask r-cnn.", *2017 IEEE International Conference on Computer Vision (ICCV), Computer Vision (ICCV), 2017 IEEE International Conference on, ICCV*, p. 2980, 2017, ISSN: 978-1-5386-1032-9.

[38] X. Hu, *CycleGAN-Tensorflow*, `https://github.com/xhujoy/CycleGAN-tensorflow`, Accessed: 2019-04-04, 2018.

[39] C. Ehmann, *Squeezedet on keras*, `https://github.com/omni-us/squeezedet-keras`, Accessed: 2019-05-26, 2018.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[41] TensorFlow, *Frechet_classifier_distance_from_activations*, Accessed: 2019-06-25. [Online]. Available: `http://www.tensorflow.org/api_docs/python/tf/contrib/gan/eval/frechet_classifier_distance_from_activations`.

[42] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts", *Distill*, vol. 1, no. 10, e3, 2016.

[43] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness", *CoRR*, vol. abs/1811.12231, 2018. arXiv: `1811.12231`. [Online]. Available: `http://arxiv.org/abs/1811.12231`.

[44] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge", *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[45] F. S. Bhagyesh Vikani, *Cnn visualization*, `https://github.com/InFoCusp/tf_cnnvis/`, Accessed: 2019-05-26, 2017. DOI: `10.5281/zenodo.2594491`.

[46] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan", *arXiv preprint arXiv:1701.07875*, 2017.

[47] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation", *CoRR*, vol. abs/1710.10196, 2017. arXiv: `1710.10196`. [Online]. Available: `http://arxiv.org/abs/1710.10196`.

[48] M.-Y. Liu, X. Huang, A. Mallya, T. Karras, T. Aila, J. Lehtinen, and J. Kautz, "Few-shot unsupervised image-to-image translation", in *arXiv*, 2019.

[49] TensorFlow, *Tensorflow graphics*, `https://github.com/tensorflow/graphics`, Accessed: 2019-05-26, 2019.

[50] E. author, "The world's most valuable resource is no longer oil, but data", *The economist*, 2017. [Online]. Available: `https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data`.

[51] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen, "Mesonet: A compact facial video forgery detection network", in *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, IEEE, 2018, pp. 1–7.

Chapter 7
Appendix

Parameter	Value
# Phenotypes	18
# Facial expressions	43
Skin age	350
Skin bumb	$\mathcal{U}(0, 1)$
Skin oil	$\mathcal{U}(0, 1)$
Skin hue	$\mathcal{U}(0.49, 0.51)$
Melanin	$\mathcal{U}(0.2, 1)$
Fabric saturation	$\mathcal{N}(0, 0.3)$
Fabric value	$\mathcal{N}(0.5, 0.2)$

Table 7.1: Parameters for rendering synthetic human models where $\mathcal{U}(a, b)$ is a uniform distribution with min and max values a and b respectively. $\mathcal{N}(\mu, \sigma)$ is a normal distribution with mean value μ and standard deviation σ.

Parameter	Value
#Backgrounds	300
#Scenes per background	5
#Sitting models per image	$\mathcal{U}(1, 5)$
#Standing models per image	$\mathcal{U}(0, 4)$
Render samples	350

Table 7.2: Parameters for rendering synthetic images where $\mathcal{U}(a, b)$ is a uniform distribution with min and max values a and b respectively.

Algorithm 4 Our rendering algorithm.
bgs: array of unpopulated background images.
N: number of scenes per background
Returns *S*: array of annotated synthetic images.

```
1:  procedure RENDER($bgs$)
2:      let $S = []$
3:      for background $b \in bgs$ do
4:          for $i \in 1..N$ do
5:              let $p_{models}$ = Randomized parameters for human models according to Table 7.1
6:              let $models$ = rendered human models using $p_{models}$
7:              let $p_{image}$ = Randomized parameters for image rendering according to Table 7.2
8:              let $image = models$ superimposed on $b$ using $p_{image}$
9:              let $a$ = annotations of $image$
10:             append $(image, a)$ to $S$
11:         end for
12:     end for
13:     return $S$
14: end procedure
```

Algorithm 5 Our cropping algorithm.
D: array of images with annotated head positions; *W*: width of crops; *H*: height of crops.
Returns D_C: array of cropped images with size $W \times H$.

```
1:  procedure CROP($D, W, H$)
2:      let $D_C = []$
3:      for image $d \in D$ do
4:          for annotation $a$ of $d$ do
5:              let $c$ = box of of size $W \times H$ at the position $(x, y)$ specified by $a$
6:              set center position of $d$ as origin for $c$
7:              let $\theta = atan(x, y)$
8:              rotate $c$ around the origin by $\theta$
9:              let $d_c = d$ cropped using $c$ as a mask
10:             if $x \leq 0$ then
11:                 let $\theta_{inv} = -\theta - \frac{\pi}{2}$
12:             else
13:                 let $\theta_{inv} = -\theta + \frac{\pi}{2}$
14:             end if
15:             rotate $d_c$ by $\theta_{inv}$ to regain an upright position
16:             save $d_c$ with annotations of its original image $d$, position $(x, y)$, rotation $\theta$
17:             append $d_c$ to $D_C$
18:         end for
19:     end for
20:     return $D_C$
21: end procedure
```

Algorithm 6 Our reinsertion algorithm.
D_C: array of cropped images (as obtained by Algorithm 5); S: array of the original images of D_C.
Returns S^+: array of images where each image $d_c \in D_C$ has been reinserted into S.

```
1:  procedure REINSERT($D_C$, $S$)
2:      let $S^+ = []$
3:      for image $s \in S$ do
4:          let $s^+ = s$
5:          let $m$ = mask of $s$
6:          for cropped image $d_c \in D_C$ do
7:              if $s$ not original image of $d_c$ then
8:                  continue
9:              end if
10:             let $p$ = original position of $d_c$ in $s$
11:             let $\theta$ = original rotation of $d_c$ in $s$
12:             let $d_c = d_c$ rotated by $\theta$
13:             let $s^+ = d_c$ superimposed at $p$ on $s^+$
14:         end for
15:         let $s^+ = s^+ \cdot m + (1 - m) \cdot s$ (Use foreground of $s^+$ and background of $s$)
16:         append $s^+$ to $S^+$
17:     end for
18:     return $S^+$
19: end procedure
```

Algorithm 7 Our refinement algorithm
X, Y: Image domains
r: Training configuration object
Returns X^+: array of refined annotated images

```
1:  procedure REFINE($X$, $Y$, $r$)
2:      let crop_X = CROP(X, r.crop_width, r.crop_height)
3:      let crop_Y = CROP(Y, r.crop_width, r.crop_height)
4:      if architecture of $r$ is CycleGAN then
5:          let $model = TRAIN\_C(crop\_X, crop\_Y, r)$
6:      else if architecture of $r$ is AGGAN then
7:          let $model = TRAIN\_A(crop\_X, crop\_Y, r)$
8:      end if
9:      let $crop\_X^+ = model.predict(crop\_X)$
10:     let $X^+ = REINSERT(crop\_X^+)$
11:     return $X^+$
12: end procedure
```

Algorithm 8 CycleGAN training algorithm
X, Y: Image domains
r: Training configuration object

```
 1: procedure TRAIN_C($X, Y, r$)
 2:     let $learning\_rate = r.learning\_rate$
 3:     for $e$ in $epochs$ do
 4:         if $e \geq \frac{epochs}{2}$ then
 5:             let $decay = \frac{epochs-e}{\frac{epochs}{2}}$
 6:             let $learning\_rate = r.learning\_rate \cdot decay$
 7:         end if
 8:         for $x, y$ in batches of $X, Y$ do
 9:             let $x' = G(x)$
10:             let $x'' = F(G(x))$
11:             let $y' = F(y)$
12:             let $y'' = G(F(y))$
13:             add $(x', y')$ to minibatch pool
14:             update weights of $G$ according to $r.losses$ of $x'$, $x''$, and $y''$
15:             update weights of $F$ according to $r.losses$ of $y'$, $y''$, and $x''$
16:             sample $x'_{pool}$ and $y'_{pool}$ from minibatch pool
17:             update weights of $D_X$ according to $r.losses$ of $D_X(x)$ and $D_X(y'_{pool})$
18:             update weights of $D_Y$ according to $r.losses$ of $D_Y(y)$ and $D_Y(x'_{pool})$
19:         end for
20:     end for
21:     return trained model
22: end procedure
```

Algorithm 9 AGGAN training algorithm
X, Y: Image domains
r: Training configuration object

```
1: procedure TRAIN_A($X$, $Y$, $r$)
2:     let $learning\_rate = r.learning\_rate$
3:     for $e$ in $epochs$ do
4:         if $e \geq \frac{epochs}{2}$ then
5:             let $decay = \frac{epochs - e}{\frac{epochs}{2}}$
6:             let $learning\_rate = r.learning\_rate \cdot decay$
7:         end if
8:         for $x$, $y$ in batches of $X, Y$ do
9:             let $x' = translate(G, A_X, x)$
10:            let $y' = translate(F, A_Y, y)$
11:            let $x'' = translate(F, A_Y, x')$
12:            let $y'' = translate(G, A_X, y')$
13:            add $(x', y')$ to minibatch pool
14:            update weights of $G$ according to $r.losses$ of $x'$, $x''$, and $y''$
15:            update weights of $F$ according to $r.losses$ of $y'$, $y''$, and $x''$
16:            if $e < r.switch$ then
17:                update weights of $A_X$ according to $r.losses$ of $x'$, $x''$, and $y''$
18:                update weights of $A_Y$ according to $r.losses$ of $y'$, $y''$, and $x''$
19:            end if
20:            sample $x'_{pool}$ and $y'_{pool}$ from minibatch pool
21:            if $e \geq r.switch$ then
22:                let $x = threshold(x, A_X(x))$
23:                let $y = threshold(y, A_Y(y))$
24:            end if
25:            update weights of $D_X$ according to $r.losses$ of $D_X(x)$ and $D_X(y'_{pool})$
26:            update weights of $D_Y$ according to $r.losses$ of $D_Y(y)$ and $D_Y(x'_{pool})$
27:        end for
28:    end for
29:    return trained model
30: end procedure
31: procedure TRANSLATE($generator$, $attention\_net$, $img$)
32:    let $a = attention\_net(img)$
33:    if $e \geq r.switch$ then
34:        let $img = generator(img) \cdot a$
35:        let $img' = threshold(img, a)$
36:    else
37:        let $img' = generator(img) \cdot a + (1 - a) \cdot img$
38:    end if
39:    return $img'$
40: end procedure
41: procedure THRESHOLD($img$, $a$)
42:    let $P$ = number of pixels in $img$
43:    return $[img[i]$ if $a[i] > 0.1$ else $0$ for $i \in \{0 \ldots P\}]$
44: end procedure
```

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2019-06-11

EXAMENSARBETE Refining Synthetic Images with GANs
An Automated Production of Object Detection Training Data
STUDENTER Johan Andersson, Rickard Andersson
HANDLEDARE Volker Krueger (LTH)
EXAMINATOR Elin Anna Topp (LTH)

Att få datorgenererade bilder av människor att se verkliga ut med AI

POPULÄRVETENSKAPLIG SAMMANFATTNING **Johan Andersson, Rickard Andersson**

Med maskininlärning på framfart finns det en ökande eferfrågan på träningsdata, men en brist på tillgång. Datorgenererad träningsdata har således blivit ett vanligt verktyg, men det saknar ofta komplexitet. I vårt arbete har vi utforskat hur man med Generativa Adversiella Nätverk kan få datorgenererade bilder av människor att se verkliga ut.

Sedan deras uppsving under det senaste decenniet, har många tekniker inom maskininlärning alltjämt krävt stora mängder träningsdata för att ge bra resultat. Att samla och annotera träningsdata är dock en tidskrävande uppgift, vanligtvis utförd av människor. Vore det inte därför fantastiskt om vi skulle kunna låta datorer göra det åt oss?

Generativa adversiella nätverk (GAN) är en teknik där två artificiella neurala nätverk tävlar mot varandra i ett nollsummespel. Föreställ dig en konstförfalskare och en konstkritiker. Förfalskaren har ett incitament att producera målningar som ser äkta ut, medan kritikern gärna vill kunna urskilja ifall en målning är förfalskad eller ej. Ifall kritikern upptäcker en förfalskning, måste förfalskaren förbättra sin teknik så att hens målningar inte längre upptäcks av kritikern. Därefter måste kritikern bli bättre på att upptäcka förfalskade målningar, och igen måste då förfalskaren bli ännu bättre . . . Det blir alltså ett slags evigt spel, där förfalskaren och kritikern hela tiden försöker bli bättre än den andra.

Denna slags situation är huvudidén bakom GAN, där en generator (konstförfalskaren) försöker lura en diskriminator (konstkritikern) att dess genererade data är verklig.

I vårt examensarbete har vi använt GAN till att få datorgenererade bilder av människor att efterlikna verkliga bilder. Vi utgår från renderade bilder som i grova drag efterliknar ett verkligt dataset av bilder. Vår generator tar då dessa renderade bilder som input och målar om dem så att de ska efterlikna det verkliga datasetet.

Efter att generatorn har tränats, använder vi dess förfalskade bilder som träningsdata för att detektera människor. Jämfört med små mängder verklig träningsdata lyckas vi med våra förfalskade bilder få bättre träffsäkerhet. I framtiden tror vi att en sådan här metod kommer att bli allt mer vanlig och därför revolutionera området, eftersom vem som helst då kan generera egen träningsdata.