MASTER'S THESIS | LUND UNIVERSITY 2018

Generation of Artificial Training Data for Deep Learning

Pontus Andersson, David Wessman

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2018-39

Generation of Artificial Training Data for Deep Learning

Pontus Andersson
pontus94a@gmail.com

David Wessman
david@wessman.co

August 10, 2018

Master's thesis work carried out at Axis Communications AB.

Supervisors:
Mikael Andreen, mikael.andreen@axis.com
Niclas Danielsson, niclas.danielsson@axis.com
Michael Doggett, michael.doggett@cs.lth.se
Kalle Åström, kalle@maths.lth.se

Examiner:

Niels Christian Overgaard, nco@maths.lth.se

Abstract

This thesis was written at Axis Communcations AB, Lund, together with the Department of Computer Science and the Centre of Mathematical Sciences at Lund University, Faculty of Engineering LTH.

Can artificial training data be used for deep learning applications in computer vision? We investigated this by building a framework that uses computer graphics to generate large quantities of images portraying humans. Generative Adversarial Networks (GANs) were used to bridge the gap in appearance between the generated and real images. A dataset's potential was estimated by first using it to train a person re-identification model, and then evaluating the model on real images. Results showed that datasets that had been put through a GAN had higher potential than those which had not. We were not able to replace real images with artificial ones, but our results showed promise for further work in substituting and complementing real images.

Keywords: Artificial training data, deep learning, generative adversarial networks, large scale image generation, person re-identification.

Acknowledgements

We would like to thank our university supervisors: Michael Doggett and Kalle Åström, for giving us many ideas for interesting thesis topics, helping us set up this work as a collaboration between the Department of Computer Science and the Centre of Mathematical Sciences at Lund University, Faculty of Engineering LTH, and for providing much valuable feedback throughout the duration of this master's thesis.

We would also like thank our Axis supervisors: Mikael Andreen and Niclas Danielsson, for guiding us through this project, discussing issues with us and sharing exciting ideas for improvements to our work.

Our gratitude is also extended to Axis Core Technologies Analytics (CTA) lead engineer, Markus Skans, for showing great interest in our work, discussing possible explanations of our many results, and helping us sort out many technical details. Moreover, we wish to thank the numerous CTA employees who always helped us when we needed assistance, and who lent us their computers at night time so that they could render images for us.

Lastly, we would like to thank the Axis management for providing us with an office space, computers, and an overall excellent environment for carrying out this thesis.

Contents

Chapter 1

Introduction

1.1 Background

The number of deep learning based computer vision applications used in today's society is growing rapidly. We are able to teach the computer to recognise faces [46], propose fitting captions to images [56], and even how to drive a car [5]. This teaching is based on a lot of examples: we give the computer sample problems of a specific task, and it adjusts its artificial neural networks (networks, for short) so that its performance on a given problem would have been improved if it was given the same problem again. With enough practice, the computer can then do well on new problems within the same task, as it has learnt to handle similar problems.

This approach to learning can be done well under one condition: there has to be *enough* training data. In general, the more training data we give to our network, given that the data is diverse enough in comparison to its size, the better the network's performance will become [19]. However, with only a small amount of data available for training, the network will probably fail at its task. Thus, good ways of collecting data in large amounts is essential for a deep learning based model to succeed. Along with a large *diversity* and *size* of the dataset, the networks also often require *labeled* data, meaning that we, for example, have to find ways of marking which faces in our dataset belong to the same person before using the images to train our models, as they must know what the right answer is in order to learn. Together, these properties make it cumbersome to obtain datasets that are sufficient for training functioning networks.

Our thesis aims at generating training data that can complement, or even replace, existing datasets. In particular, we will be investigating the possibility of generating full body images of humans with a graphics engine, and then refining them to be more reminiscent of real images using image-to-image translation [47]. The artificial images will then be used to train a network to perform person re-identification (re-id, for short): a computer vision task aiming at recognising a person as they move through multiple, non-overlapping camera views [18].

Training a network for re-id requires labeled data in the sense that all images of the same person are bundled together. While our ability to collect large quantities of high quality images increases with more and better cameras, the process of labeling all the collected data is a very time-consuming and tedious task. Meanwhile, more people are becoming concerned with their privacy, and, with the introduction of the General Data Protection Regulation [55], a strict process for collecting and handling images of people is required. However, neither the gathering, the labeling nor the privacy is an issue with artificial data.

Section 1.2 sheds light on related work done in the areas of artificial data generation, image-to-image translation and person re-identification. In Chapter 2, we explain and propose our solution to the problem, which is then evaluated in Chapter 3.

1.2 Related Work

1.2.1 Artificial Training Data

The use of artificial data for training machine learning models has a long history, with early works dating back over 40 years [37]. Using artificial data for deep learning saw some of its first uses in 2015, when it was proposed as a way of augmenting scarce training data for object recognition [38, 66]. After these initial works, however, this way of augmenting training data for artificial neural networks has seen an increase in attention.

With the accessibility of high-quality rendering engines, including open-source ones such as Blender [4], as well as game engines, it has become possible to generate images with realistic lighting conditions and material properties in large scale [15, 41]. This has caught the interest of machine learning researchers, and multiple works have taken advantage of these possibilities for creating images that can replace photographs in computer vision applications [29, 44, 51]. Promising results have been shown with images from very simple scene setups such as scenes enclosed by a cuboid with single textures on its respective faces [53]. A more advanced approach, which has also been successful, is capturing images from a procedurally generated world with larger scene variety [54].

So far, the works mentioned have focused on object recognition, with Virtual KITTI [15] being one of the larger artificial datasets created for training these models. To our knowledge, there are no artifical datasets of similar size and variety with full body images of humans. Closest to it is the SOMAset [1], a dataset including 50 characters and 11 cloth variations. To build the SOMAset, the creators used MakeHuman [9], an open-source 3D human modelling software in which one can model characters by hand and export them to rendering engines that, in turn, can generate images of them. This is a manual process, which could have hindered the creators of the SOMAset from increasing its number of characters. In our work, we aim at automating this process by scripting the character and image generation, similar to how it has been done for the generation of face images [30].

1.2.2 Image-to-Image Translation

Image-to-image translation is a field with the goal of translating images from one domain to another, with applications ranging from adding colour to a greyscale image [65] and from making a photograph look like The Starry Night by Van Gogh [16], all the way to increasing an image's resolution [13]. With all its applications, this field has gained a lot of attention, and several methods, including convolutional neural networks [28] and regression [7], have been employed to make image-to-image translation a reality. However, while the aforementioned solutions achieve impressive results, they do not offer what we are looking for. We wish to find a network which can be taught to translate images from one style collection to another. For example, it should be able to learn how to translate photographs to the style of Van Gogh's collection of paintings, rather than just to the style of The Starry Night. In addition to this, the network must also be taught in an unsupervised manner, i.e. it should be able to learn the translation without seeing any input-output pairs. Following the previous example, it should be able to learn without seeing photographs and their analog Van Gogh paintings, since such pairs do not exist.

In 2014, Ian Goodfellow was asked for a new way of making the computer generate photographs by itself [17]. The solution he presented was the Generative Adversarial Network (GAN, for short) [20]: a structure in which two networks, the generator and the discriminator, act as adversaries of one another, leading to a competition in which both actors improve over time. In this setup, the generator is a forger, attempting to create fake images which the critic (the discriminator) cannot tell from real ones. Thus, as the forger's work becomes more realistic, the critic's eye must be sharpened, and vice versa. As the level of both parties' results increases, the images generated can deceive even the eyes of humans. In the classic GAN, the entire image synthesis is initiated solely by noise variables. However, it is also possible to input images to the GANs, based on the idea of the *conditional* GANs [36], which opens up the avenue of image-to-image translation based on the GAN framework [27].

One large constraint with the original GAN models for image-to-image translation is that they require supervision, meaning that the training images must come in pairs of e.g. one greyscale, and its analog, colourised, image. This problem has been addressed in multiple papers, where the common denominator is the inclusion of additional loss functions, i.e. new ways of directing the generator, in the GANs. Such a loss function could, for example, enforce that the translated images have similar pixel values to the original ones [47], so that the translation would not change the images' overall structure, but still be able to affect details. In our work, where we are not necessarily interested in preserving pixel values, but rather the identity of our humans while still making large changes to the style of the images, we will make use of the CycleGAN framework [68]. The CycleGAN includes *two* generators: one takes images from the first to the second domain (e.g. greyscale to colour), and the other takes images from the second to the first (e.g. colour to greyscale). It also introduces a cycle consistency loss, enforcing that if we translate an image from the first domain to the second, and then back again, we should obtain the original image. This is the key idea that allows us to teach a model to take generated images of humans and make them look more real.

The CycleGAN is a very general framework, with the ability to do well on multiple domains, depending on its training. It can learn how to translate sketches to photographs, maps to satellite images and summer to winter images [68]. Our application is not as general, but focuses on translating humans between different image domains while preserving their identities. For exactly this end, the Person Transfer GAN (PTGAN) [57] was proposed. This framework adds one more component to the CycleGAN structure: a loss that focuses on the human in the image before and after the entire image is translated. Compared to the general CycleGAN, the single-purpose PTGAN was shown by its inventors to improve the human translation results considerably [57].

1.2.3 Person Re-identification

Video surveillance has many applications. The first time a camera was used to remotely monitor an event was in 1942, when the Germans needed it to oversee the launch of V2 rockets. In 1970, cameras were placed in banks as an additional security measure against theft, and in 1992 parents began monitoring their families using so-called "Nanny cams". However, during all this time, the camera feed had to be supervised by humans. It was not until after the attack on the World Trade Center in 2001 that machine learning algorithms found their first use in surveillance, as facial recognition software became a high priority [11].

In this work, the surveillance application we are looking at is called person re-identification (re-id). Re-id is when we try to identify which images show the same person, for example as the person moves through the views of a set of cameras. This is a challenging task, as the images' features, such as illumination, environment and character pose, could change drastically between different camera views [34]. A working re-id algorithm must achieve two things: first, one needs to detect the person in the image. There is a multitude of methods used for detecting objects such as people in images [35, 42, 43], but we will assume that this part of re-id is done for us. Instead, we will focus on the second part, which is to, given images of people, say which images show the same person. The main idea behind the solutions to this problem is representing images of the same person in a way that they are similar, and images of different people in a way that they are dissimilar, even if that is not the case for the original images [67].

One of the first approaches to recognising an object present in images taken from different camera views was based on a hand-crafted appearance model [24]. This model, used to re-identify cars, explicitly considered attributes such as colour, length and width of the object. Relying on such hand-crafted features is a common approach in machine learning, and person re-identification models based on these have seen some success over the years [3, 32, 64]. However, finding features that are invariant to differences in pose or illumination is difficult. For example, a person's garments might look different from the front and from the back, making it difficult to recognise that the person is the same based on colour alone. Due to the success of deep learning based approaches in image classification [31], re-id moved away from the more classic machine learning techniques. Instead of using hand-crafted feature extractors, re-id began leveraging artificial neural networks, where the model learns what features to look for itself [34, 63].

1.3 Contributions

Finding a way of generating full body images of humans, with random skin tones, camera angles, etc., was the first part of this work. We solved this together. During the second part, David explored the possibilities of scaling the generation, while Pontus looked into how we could improve the realism of our images. When we had made progress in those areas, we started working with the re-id part of the project together, while simultaneously making further improvements to the previous parts.

The introduction of this report was written by Pontus, while we cooperated in writing the remaining sections. The different trainings and evaluations we wanted to investigate and include in the report were carried out by David, after we had discussed what we wanted to try out.

Chapter 2

Approach

This chapter describes our approach to generating and using artificial images for person re-identification. Section 2.1 describes how we create our characters, how they are put in a scene, how images are captured and how we scale the process from generating one image to a million images. In Section 2.2, we explain the process we use to make our generated images look more like real ones using Generative Adversarial Networks [20]. The chapter is then closed by Section 2.3, in which we describe how we use our images to train models that can perform re-id, and how those models are tested to, in the end, give us a measurement of our artificial datasets' potential. Table 2.1 shows our system, with its different phases and the different input and output of each phase.

Table 2.1: Schematic table of our system, with the input and output of each phase. We begin by generating full body images of humans with randomised attributes (Sec. 2.1). After that, we use those images and real images to train a refiner model, which takes our generated images and make them look more like real ones (Sec. 2.2). The refined (or generated) images are then used to teach a model how to re-identify humans (Sec. 2.3.1). Optionally, this model can be finetuned to perform better when tested on images of real humans (Sec. 2.3.1). The final re-identification model is then tested on real images, yielding a measurement of our artificial dataset's potential (Sec. 2.3.2).

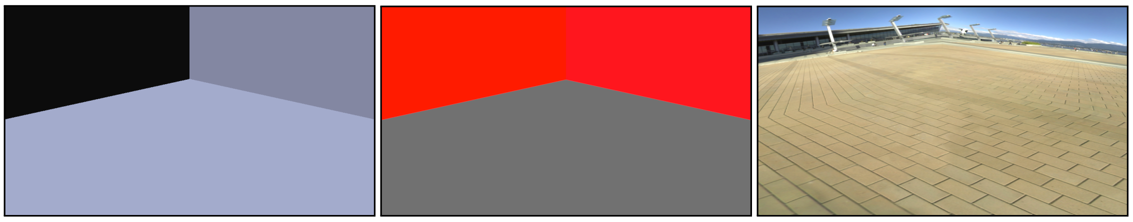
Phase	**Input**	**Output**
Generation (sec. 2.1)	Random character attributes	Generated images
Refiner training (sec. 2.2)	+	Refiner model
Refinement (sec. 2.2)	+ Refiner model	Refined images
Re-id training (sec. 2.3.1)	or	Re-id model
Finetuning (sec. 2.3.1)	Re-id model +	Finetuned re-id model
Evaluation (sec. 2.3.2)	+ (Finetuned) Re-id model	Performance measurement

2.1 Generation

In this section, we describe how we create our characters and the environment they are in (Sec. 2.1.1). We also describe how we set up our camera for each image (Sec. 2.1.2), how we crop the images to have sizes similar to the sizes of images used in our re-id application (Sec. 2.1.3), as well as how we are able to go from generating one image to a million of them (Sec. 2.1.4).

2.1.1 Creating the Scene and its Inhabitants

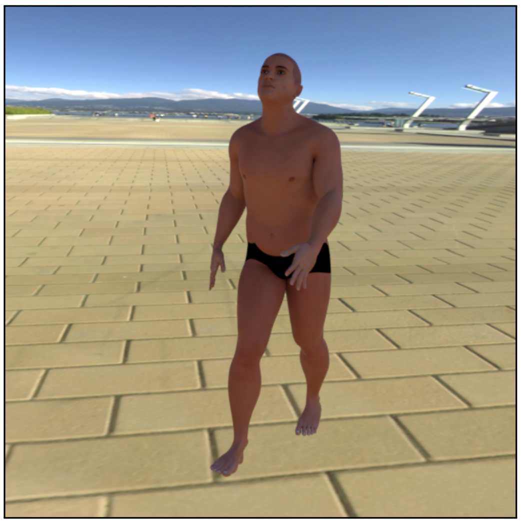
From an empty space in Blender, the scene is initialised by the insertion of a cuboidal shape, which is used as our background environment. The inside of the cuboid can be seen in Figure 2.1a. In order to increase the background's realism, we include the possibility to add different textures, so-called *cubemaps*, downloaded from Humus' collection [39], to the cuboid. This results in scenes such as the one in Figure 2.1c.



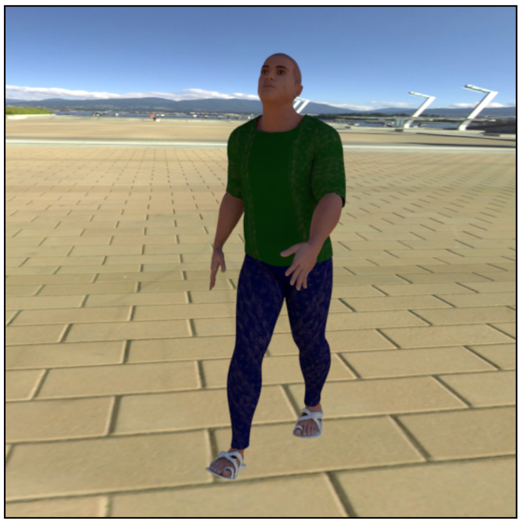
(a) Simple background. **(b)** Background with simple colours. **(c)** Background with cubemap texture.

Figure 2.1: Initialisation of our scenes in Blender. From an empty space, we add a cuboidal shape (a). We can then either add some simple colours to the cuboid's different faces (b), or a cubemap (c); the latter yielding a more realistic looking scene.

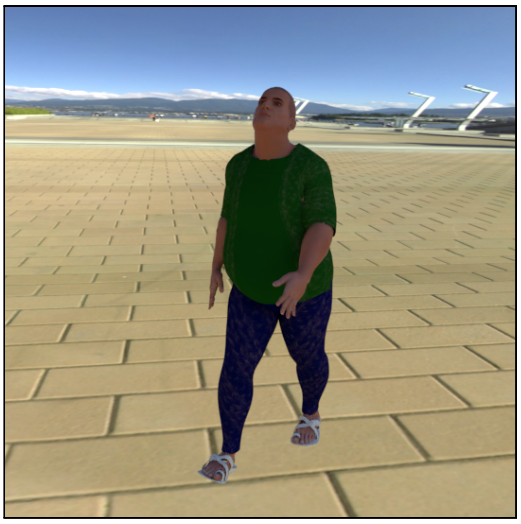
The characters we use are created with a Blender plugin called ManuelBastioniLAB [2] which adds functionality to create and modify human characters as seen in Figure 2.2. There are a lot of parameters which can be tuned in order to create a diverse dataset, but we have limited ourselves to changing the character age, mass, tone and skin. The plugin is also able to apply clothes and accessories to the characters (see Figure 2.2b). The clothes and accessories are made or adjusted by hand inside Blender and fitted to a template character supplied by the plugin. These items can then be applied to characters with different body types and sizes, as seen in Figure 2.2c, without any additional, manual adjustments. To get the characters into more natural poses, motion captures of walking people are loaded and applied to each character, allowing us to capture images of when our characters are walking around (see Figure 2.2d). The motion captures were downloaded from the CMU Graphics Lab Motion Capture Database [21].



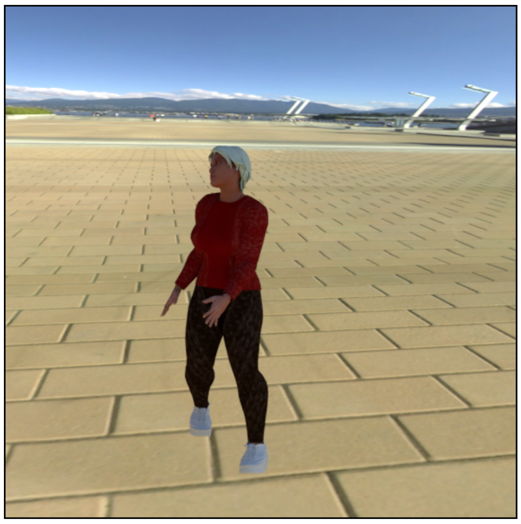
(a) Basic character.



(b) Simple clothes.



(c) Different body types.



(d) Different pose, colours, accessories and character.

Figure 2.2: The procedure for creating and setting up our human characters. We initialise characters in our scene, put clothes and accessories on them, and let them move around.

2.1.2 Positioning the Camera

The generator is able to render images from different viewpoints, which makes it useful for different use cases. For example, the generator can create images taken from above, as in a surveillance view, or images from the front, as from a web camera or a door camera. In order to achieve this, the user of the generator can define the distance from the character to the camera (the radius R) and the allowed polar angle interval (between the angles θ_1 and θ_2). This is represented as a *hemisphere segment*, on which the camera can be positioned (cf. Figure 2.3).

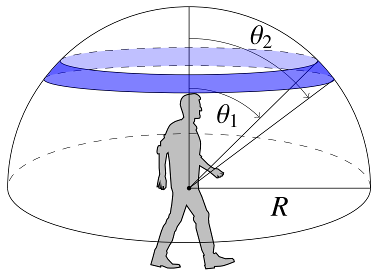


Figure 2.3: The camera position is sampled within the blue area prior to each image being captured. The blue area is a hemisphere segment defined by the user as they choose the polar angles θ_1 and θ_2, along with the radius R.

We position the camera randomly on the defined hemisphere segment. To make sure that the probability of each allowed camera position is the same, we need to sample uniformly on the segment. Choosing a polar angle $\theta \in [\theta_1, \theta_2]$ and an azimuth angle $\phi \in [0, 2\pi)$ uniformly and mapping them to a sphere does not suffice, and will result in samples that are not correctly distributed [48]. Our strategy is shown in Figure 2.4, and goes as follows: we first sample two uniform variables $(\xi_1, \xi_2) \in [0, 1)^2$, and then map that sample to an intermediate point (x_c, y_c, z_c) on the unit-height cylinder. The cylinder sample is then squeezed into a sample (x_c, y_c, z'_c) inside a height range $[h_2, h_1]$ defined by θ_1 and θ_2. Lastly, we project the cylinder sample onto the unit hemisphere, resulting in the sample (x_s, y_s, z_s). The fact that this projection is area preserving, and thus results in uniform samples on the hemisphere segment, follows from Archimede's Hat-Box Theorem [58].

The hemisphere is originally placed in the origin, and needs to be translated to the character's position. After doing so, we translate the sample outwards until the distance to the hemisphere's centre is R, resulting in a uniformly sampled camera position within the user-defined segment, as in Figure 2.3.

2.1.3 Cropping the Images

Since we are working with the second part of person re-identification, i.e. being able to tell if two images of people portray the same person, we want to generate images and crop them. For real images, we would use object detection algorithms [42] to detect which part

$$\begin{cases} \phi = 2\pi\xi_1 \\ x_c = \cos\phi \\ y_c = \sin\phi \\ z_c = \xi_2 \end{cases}$$

$$\begin{cases} h_1 = \cos\theta_1 \\ h_2 = \cos\theta_2 \\ z'_c = (h_1 - h_2)z_c + h_2 \end{cases}$$

$$\begin{cases} z_s = z'_c \\ d = \sqrt{1 - z_s^2} \\ x_s = dx_c \\ y_s = dy_c \end{cases}$$

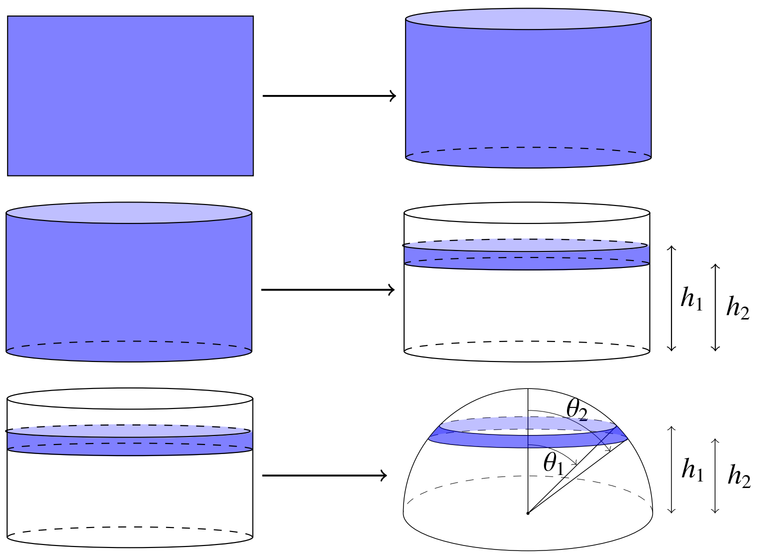


Figure 2.4: The strategy used for drawing uniform samples on a hemisphere segment defined by the two polar angles θ_1 and θ_2. First, we draw a uniform sample $[\xi_1, \xi_2] \in [0, 1)^2$ and map it to the unit cylinder (first row). The cylinder sample is then squeezed so that its height is in the range $[h_2, h_1]$ (second row), which is computed using the user defined angles θ_1 and θ_2. Lastly, the squeezed cylinder sample is projected onto the unit hemisphere (third row), resulting in a uniformly distributed sample on the hemisphere segment.

of the image contains the person, and crop the image based on that. For generated images, we instead leverage our graphics engine.

Graphics engines such as Blender describe their scenes as collections of 3D points and how they are connected to form objects [12]. To reduce the computational complexity of rendering an image, the engines often implement *bounding volumes*, which enclose the 3D objects [40]. Conveniently, the bounding volumes in Blender can be accessed by the user. Along with those, Blender also provides all the information about the cameras that is used. What that means, is that we can use the camera information to project the bounding volume enclosing our character onto our 2D image [10]. When projected onto 2D, this bounding volume, which is a cuboid in our case, forms a 2D polygon with eight corners. We find the largest and smallest x and y values of this polygon, respectively, and use those four values to form a rectangle. This rectangle then covers the entire character, and can be used for cropping. In addition to this, Blender allows us to define a *rendering border*, which determines what part of the camera's view that should be rendered. Setting the rendering border along our rectangle then results in an image that is cropped as it is rendered, and we do not need to crop them as a post-processing step. Figure 2.5 shows a comparison of when we render an image without explicitly setting the rendering border, compared to when we set it based on the character's projected bounding volume.

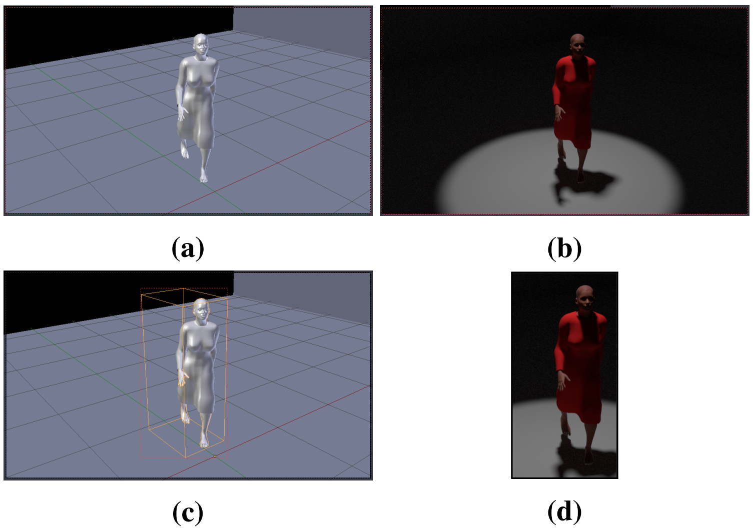


Figure 2.5: Comparison of when we render an image without explicitly setting the rendering border (red, dashed line) in Blender, and when we set it based on the character's projected bounding volume (yellow line). Image (a) shows the default rendering border set by Blender, which results in image (b) when we finish rendering. In image (c) we show the projected bounding volume of our character, and how we have fit the rendering border to that projection in such a way that we ensure that no part of the character is outside of it. This operation results in the cropped image, (d).

2.1.4 Scaling the Generation

In order to generate images on a large scale, and be able to reproduce the results, we need a system that is stable, scalable and distributable. Blender and its plugins are written in Python, and are therefore easy to integrate and automate. Using the ManuelBastioniLAB [2] plugin, we were able to automate each of the steps we took in sections 2.1.1-2.1.3. MakeHuman [9], an application with similar features as ManuelBastioniLAB, would be an alternative for creating characters. However, it is a standalone application, and would therefore require first exporting from MakeHuman and then importing into Blender. Such operations could make it hard to automate the character creation process, and would have limited the possibility to vary the character attributes. In order to setup Blender with Python in a repeatable way, we used Docker [26]. This proved to be a crucial step in creating an easily distributed, stable and scalable environment.

When we generate images of a character, we use a lot of information to describe the scene, the character, the clothes and accessories, as well as the camera positions and character poses. Varying this information allows us to generate unique images, but we need to store the information to be able to reproduce them. The information is stored in a database, which is running inside a Docker container. The database also helps in keeping track of which images and characters are completed.

The image generation is done by *workers*. The workers connect to the database, where they access the required information for each image (character mass, hair type, camera position sample, etc.) and then execute the Python scripts in order to set up the scene, create the character, position the camera, render each image and upload them to a central location. The workers can be stopped at any time and will only mark images as completed if they were rendered and stored successfully. Each worker fetches a job from a *job queue*, where the job instructs the worker which character to render. No two jobs hold information of the same character, meaning that no two workers will work on the same character simultaneously. The infrastructure can be seen in Figure 2.6, where the top box is the centralised part and the bottom box are the workers which can be distributed to multiple computers.

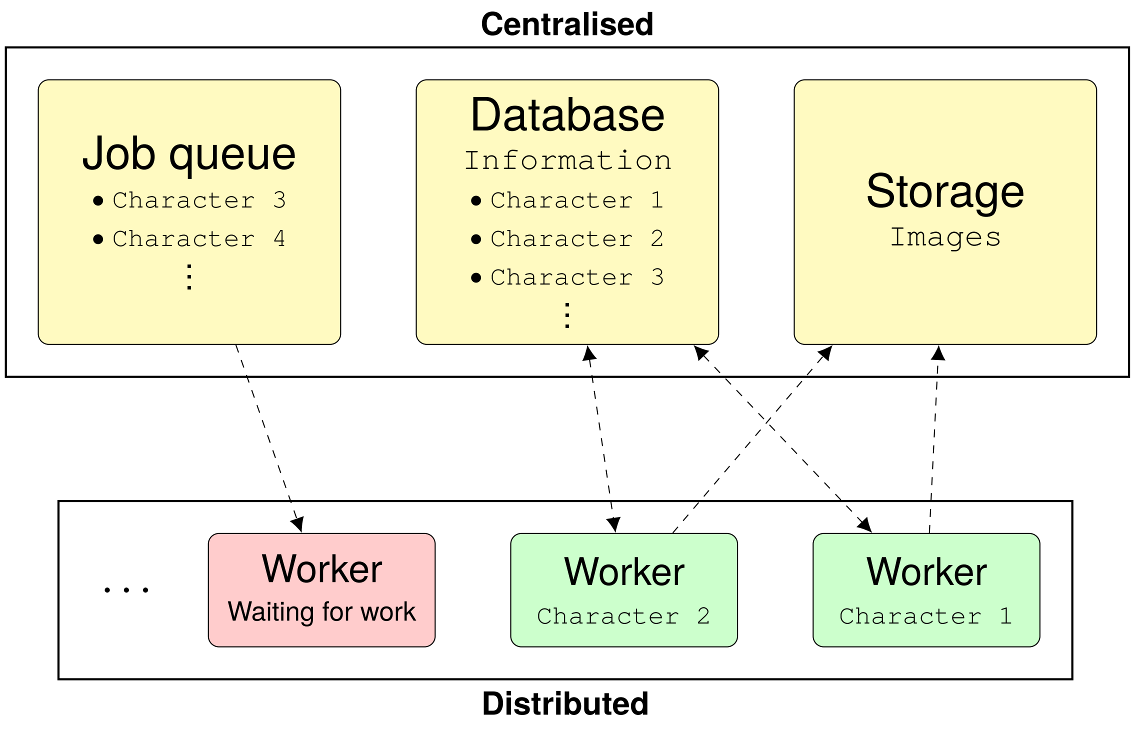


Figure 2.6: Infrastructure for the distribution of our image generation. The centralised part is handled by one computer which prepares the characters we wish to render, creates the job queue and stores the finished images. Other computers make up the distributed part. Each computer can maintain a number of workers, who fetch jobs from the job queue. During each job, the worker retrieves the required character attributes from the database, renders all images of a character, and uploads them to the storage. The worker then collects a new job from the queue. This process is repeated until the job queue is empty and all images are generated.

2.2 Refinement

This section describes how we attempt to close the gap in appearance between our generated images and images taken with real cameras, using image-to-image translation. We call this process *refinement*, and the idea is that the refined images will work better than the generated images for training re-id models that are to be used on real images.

2.2.1 Generative Adversarial Networks

The aim of a Generative Adversarial Network (GAN) [20] is to have a generator, G, learn to take samples z from a distribution p_z and turn them into samples $G(z)$ that are indistinguishable from samples coming from a data distribution p_{data}. (Henceforth, we will use the notation $x \sim p_x$ to denote that the sample x is drawn from the distribution p_x. For example, $x \sim p_{\text{data}}$ is a sample drawn from the distribution p_{data}.) In the common case, the distribution p_z is a noise distribution, with samples $z \sim p_z$ being noise vectors, while p_{data} is a distribution over photographs. The generation of real images from noise relies on the inclusion of a second component to the setup: a discriminator, D, which acts as an adversary to the generator. The discriminator's task is to take a sample (either a real sample, x, from p_{data}, or a fake sample, $G(z)$, created by the generator), and compute the probability that the sample came from the distribution p_{data}. It aims at setting a low probability, $D(G(z))$, for samples coming from the generator, and a high probability, $D(x)$, for samples coming from the data distribution, p_{data}. At the same time, the generator tries to maximise the probability of the discriminator making a mistake, i.e. it tries to maximise $D(G(z))$. The GAN framework is depicted in Figure 2.7.

While the idea of a GAN is general, and could be implemented in multiple ways, our GAN has both the generator and the discriminator implemented as feed-forward neural networks [19]. What this allows for is teaching the generator and the discriminator to perform well using backpropagation algorithms [45], meaning that we can define loss functions that tell the adversaries how well they performed on a given example, and have them both learn by minimising their respective losses. The discriminator, which should be able to distinguish well between fake samples $G(z)$, where $z \sim p_z$, and real samples $x \sim p_{\text{data}}$, has the loss

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

where $\mathbb{E}_{x \sim p_x} [f(x)]$ denotes the expected value of the function f over the distribution p_x. In practice, we generally cannot find the exact value of $\mathbb{E}_{x \sim p_x} [f(x)]$, and we instead compute an approximation by averaging the function values, $f(x)$, over all available samples, $x \sim p_x$. The discriminator's loss (2.1) has a larger value when the probability $D(x)$ is low, and a smaller value when the probability $D(G(z))$ is high. Thus, to minimise this loss, the discriminator must minimise the probability that generated images, $G(z)$, are real, while simultaneously maximising the probability that samples $x \sim p_{\text{data}}$ are real. The generator, on the other hand, whose goal is to fool the discriminator, wants the samples $G(z)$ to result in a large probability $D(G(z))$. The loss it tries to minimise is therefore

$$\mathcal{L}_G(G) = \mathbb{E}_{z \sim p_z} \left[-\log(D(G(z))) \right], \tag{2.2}$$

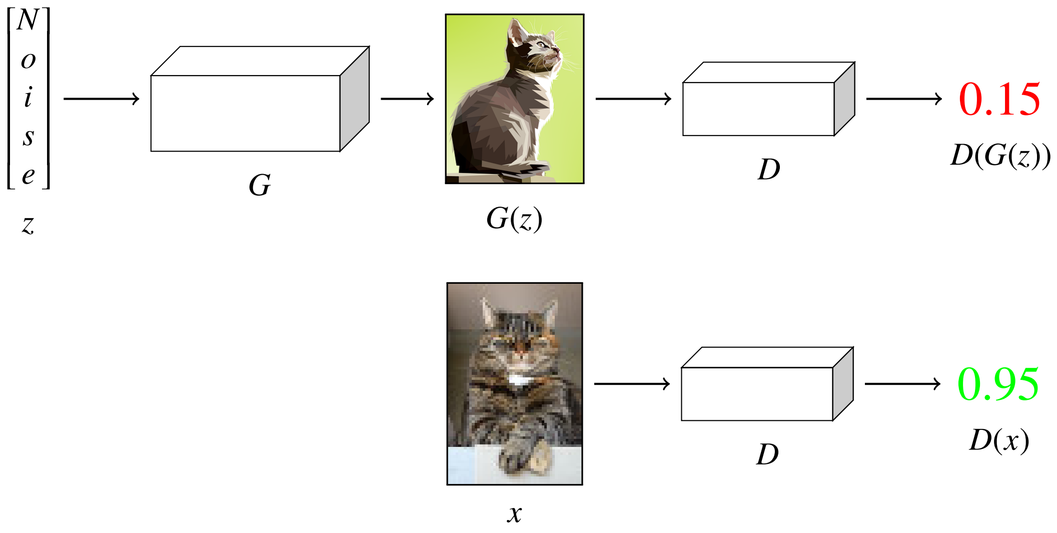


Figure 2.7: Description of the GAN framework. Given is a noise vector, z, and a photograph, x. The noise vector is put through the generator, G, resulting in a new image $G(z)$. The discriminator, D, takes this image and computes the probability, $D(G(z))$, that it came from the data distribution, p_{data}. It tries to minimise this probability, while it also tries to maximise the probability, $D(x)$, that the photograph, x, is drawn from p_{data}. (Note that the probabilities in the figure are for illustrative purposes only, and that there is no direct connection between the values of $D(G(z))$ and $D(x)$.) Meanwhile, the generator attempts to maximise $D(G(z))$, leading to it and the discriminator becoming adversaries.

whose value decreases as the probability $D(G(z))$ increases. We can combine the adversaries' losses and obtain the final objective function of the GAN:

$$\mathcal{L}_{\text{GAN}}(G, D) = \mathbb{E}_{x \sim p_{\text{data}}} [\log(D(x))] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))] , \tag{2.3}$$

where we find the optimal generator, G^*, by solving

$$G^* = \arg \min_G \max_D \mathcal{L}_{\text{GAN}}(G, D). \tag{2.4}$$

Theoretically, the game between the generator and the discriminator has a global optimum where the generative distribution, i.e. the distribution that our fake images $G(z)$ belong to, is equal to the data distribution p_{data} [20], at which point the generated samples are indistinguishable from real ones. As they propose the GAN, the authors also propose a training algorithm, based on the aforementioned losses (2.1) and (2.2), which they prove to converge at this optimum. While the feed-forward neural networks have a hard time reaching this theoretical optimum in practice, they are still able to provide us with a generator that can generate images which, in many cases, are hard to distinguish from photographs.

2.2.2 Conditional Generative Adversarial Networks

Shortly after the introduction of GANs, researchers started looking at conditioning the setup on some data y [36], posing the question whether or not we could steer the generation to a conditioned data distribution $p_{\text{data}}(x|y)$. For instance, we could let the data distribution p_{data} be over the MNIST handwritten digits database [33], and condition on that set's different labels (i.e. the digits $0-9$). This would allow the generator to learn how to generate digits of our choice, steered in that direction by the label y that we condition on.

It turns out that, in practice, what this conditioning comes down to is that the generator takes the data y as input alongside the original noise sample, $z \sim p_z$, while the discriminator takes either the pair $(G(z), y)$, or the pair (x, y), where the sample x was drawn from $p_{\text{data}}(x|y)$. For example, to generate digits similar to those of the MNIST dataset, the data y was a vector with a 1 at the index corresponding to the digit we wanted to generate, and 0s filling the other vector slots. The vector y's pair was then an image, x, that showed the digit corresponding to the index of the 1 in y [36]. Now, the idea of conditional GANs (cGANs) can be extended to use entire images as conditional data, allowing for generation of e.g. satellite images based on maps, photos based on sketches and coloured images based on greyscale ones. To achieve this extension, the creators of the pix2pix model [27] (see Figure 2.8) changed the original cGAN structure in two important ways. Firstly, they discarded the noise input z, and only input an image y to the generator. This was a change made due to the authors discovering that the noise vector they gave the system was largely ignored. However, maintaining a model's stochasicity during training is crucial as it reduces overfitting [49], i.e. when the model performs well during its training, but makes large errors when presented with new data. Therefore, instead of the noise input, the inventors of pix2pix added dropout [50], which is a way of injecting stochasicity into the network by randomly omitting some of its neurons for each training case. The second change was the addition of another loss function to the generator, namely

$$\mathcal{L}_{\text{sim}}(G) = \mathbb{E}_{(x,y)\sim p_{\text{data}}}\left[||x - G(y)||_1\right], \tag{2.5}$$

where p_{data} now is a distribution from which we draw the image pairs, (x, y). This loss imposes that the generated image, $G(y)$, should be similar to a ground truth image, x. There are two things to note here. The first one is that, with this loss, we have added the requirement of a ground truth image. Thus, the pix2pix model is *supervised*, and we need input-output pairs in order to train our model. We also note that the addition of another loss, which helps train our model towards a certain goal, e.g. similarity to the ground truth in the case of the similarity loss (2.5), is easily done to the GAN framework due to its feed-forward neural network and backpropagation foundation. Making the model take a new loss into consideration while training simply requires us to add it to the GAN's objective function.

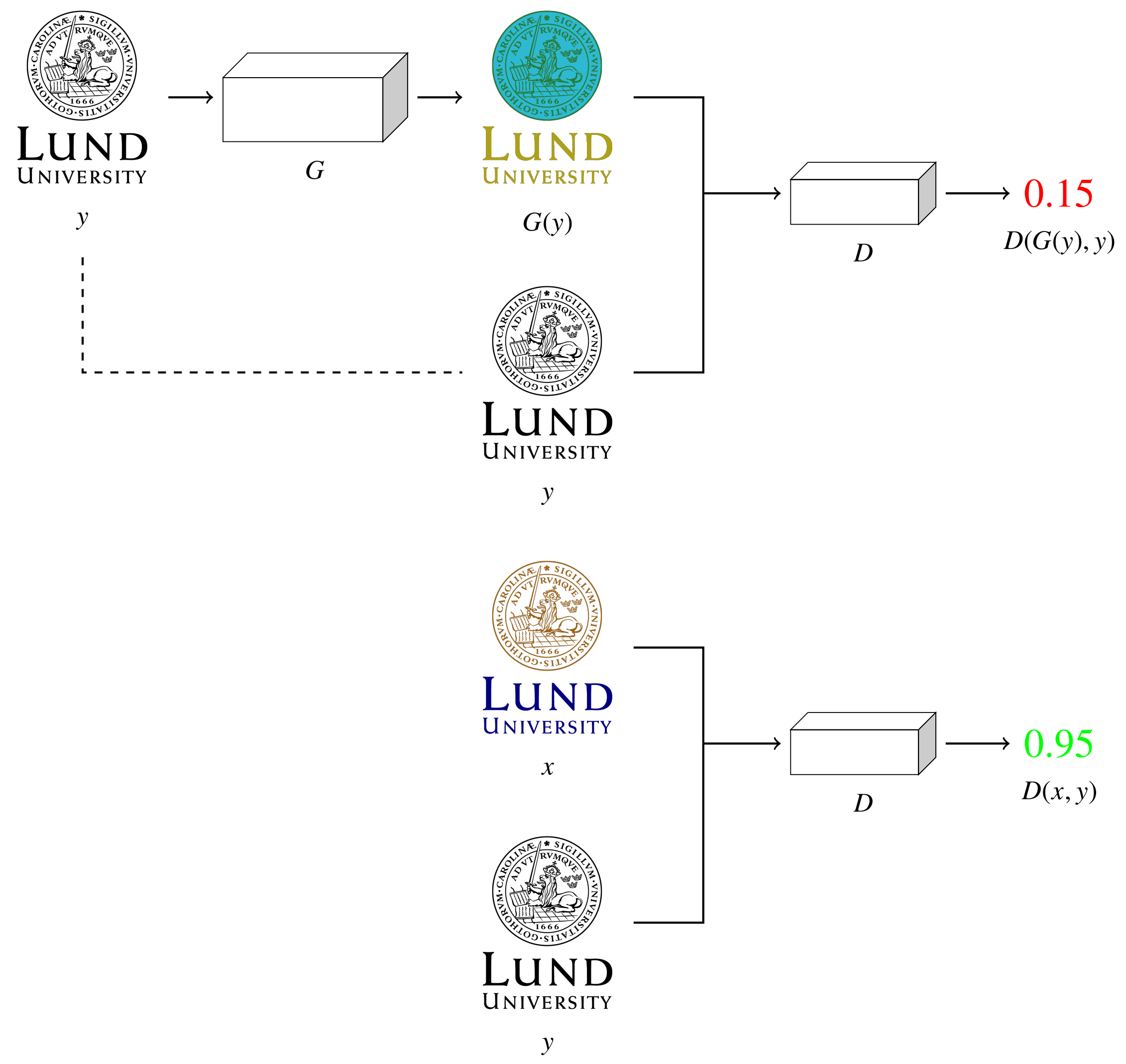


Figure 2.8: Description of the pix2pix framework [27]. Given is a pair of images, x and y. The generator, G, translates y to a new image, $G(y)$. The discriminator, D, takes both the input image, y, and the generator's output, $G(y)$, and tries to tell whether or not the two images make a pair. Its prediction is presented as the probability $D(G(y), y)$, which it aims at minimising, since the two images do not make a correct pair. At the same time, it tries to maximise the probability $D(x, y)$, since x and y do make a correct pair. In addition to attempting to maximise the probability $D(G(y), y)$, the generator also aims at minimising the pixel-wise difference between its output, $G(y)$, and y's correct pair, x. Notice how this framework requires the pair x and y (in this case a greyscale image, y, and the corresponding colour image, x). Such pairs cannot be provided in all applications; one example being when we wish to translate images of fake humans to images of real ones. The probabilities in the figure are for illustrative purposes only.

Our goal is to translate images of computer generated humans to images of humans taken with real cameras. One obvious obstacle here is that we do not know how our generated humans would look if they were real and we had taken photographs of them. In other words, we have no input-output pairs. This fact rules out the possibility to use losses such as the similarity loss (2.5) for guiding the generation. Instead, we must find a model that, in an *unsupervised* manner, allows us to translate images from one domain (generated humans) to another (real humans) and preserve the identities of the characters so that the images portraying them can still be used to train re-id models.

Unsupervised learning for collection style transfer (e.g. Van Gogh paintings ↔ photographs) can be achieved with the CycleGAN framework [68]. The CycleGAN has two generators, G_A and G_B: the first takes images from domain A and translates them to domain B, and vice versa in the second generator's case. The idea is then to include a loss function that incentivises that the input image, a, from domain A is well reconstructed after passing it through G_A and then through G_B, i.e. $a \to G_A(a) \to G_B(G_A(a)) \approx a$. Having such a loss should enforce that the generator G_A preserves the overall structure of the input image, as there must be enough information remaining in the output, $G_A(a)$, for the second generator to have the possibility of generating an output, $G_B(G_A(a))$, satisfying that $G_B(G_A(a)) \approx a$. Similarly, we want an image, b, that comes from domain B, and is passed through generators G_B and G_A, to be well reconstructed, i.e. $b \to G_B(b) \to G_A(G_B(b)) \approx b$. The cycle consistency loss becomes

$$\mathcal{L}_{\text{cyc}}(G_A, G_B) = \mathbb{E}_{a\sim p_{\text{data}}(A)}\left[\|a - G_B(G_A(a))\|_1\right] + \mathbb{E}_{b\sim p_{\text{data}}(B)}\left[\|b - G_A(G_B(b))\|_1\right], \tag{2.6}$$

where $p_{\text{data}}(A)$ denotes the data distribution of images from domain A, while $p_{\text{data}}(B)$ denotes the data distribution of images from domain B. Furthermore, with the addition of G_B, we also add a second discriminator, D_B, that tries to distinguish between images coming from G_B and images, a, that originally are from domain A. The full objective function becomes

$$\mathcal{L}(G_A, G_B, D_A, D_B) = \mathcal{L}_{\text{GAN}}(G_A, D_A) + \mathcal{L}_{\text{GAN}}(G_B, D_B) + \lambda\mathcal{L}_{\text{cyc}}(G_A, G_B), \tag{2.7}$$

where λ is a hyperparameter allowing the user to set the relative impact of the adversarial and cycle consistency losses, and we now find the optimal generators by solving

$$G_A^*, G_B^* = \arg\min_{G_A, G_B}\max_{D_A, D_B}\mathcal{L}(G_A, G_B, D_A, D_B). \tag{2.8}$$

In summary, our setup now includes two cGANs in opposite directions, which work in parallel towards, in the end, producing a generator G_A that can take an image from domain A and make it indistinguishable from images from domain B while preserving the image's overall structure. The feed-forward neural networks that represent the generators and discriminators are, as mentioned in Section 2.2.1, trained with backpropagation algorithms to solve (2.8). As opposed to the pix2pix framework, the CycleGAN does not require input-output pairs. A schematic view of the CycleGAN can be seen in Figure 2.9.

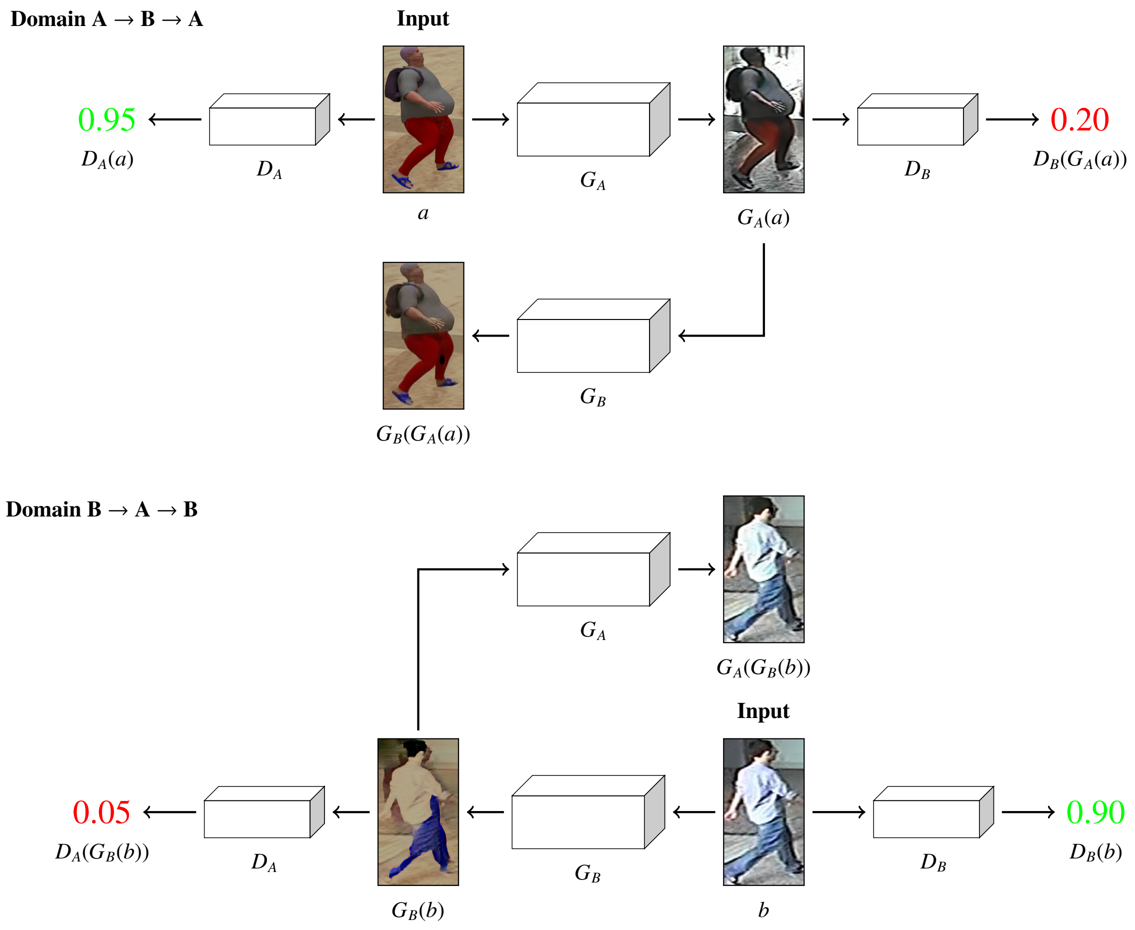


Figure 2.9: Schematic view of the CycleGAN [68], which achieves collection style transfer after unsupervised training. We input two images: a from domain A (generated human images) and b from domain B (real human images). The model contains two generators, G_A and G_B, which translate images from domain A to domain B, and from domain B to domain A, respectively. There are also two discriminators, D_A and D_B: the first one tries to distinguish between genuine domain A images (a) and fake domain A images ($G_B(b)$), while the second one aims at distinguishing between genuine domain B images (b) and fake domain B images ($G_A(a)$). The generators are trained to keep image similarities when they are cycled through both generators, i.e. $a \rightarrow G_A(a) \rightarrow G_B(G_A(a)) \approx a$ and vice versa for domain B image b. As opposed to the pix2pix framework, the given images a and b do not need to form pairs. The probabilities in the figure are for illustrative purposes only.

The CycleGAN allows us to translate images between two domains in an unsupervised manner and still preserve the overall image structure. However, it does not necessarily preserve the identity of humans in the images, which is an attribute essential for our application. A framework that teaches its generator exactly this trait is the Person Transfer GAN (PTGAN) [57]. The authors achieve this by extending the CycleGAN objective (2.7) with another loss, which enforces that the human in the image should be similar before and after the entire image is translated. One way to do this is by removing the background in the original and translated images, and then compare them. Removing the background can be done by multiplying the images element-wise with a mask, $M(\cdot)$ (cf. Figure 2.10).

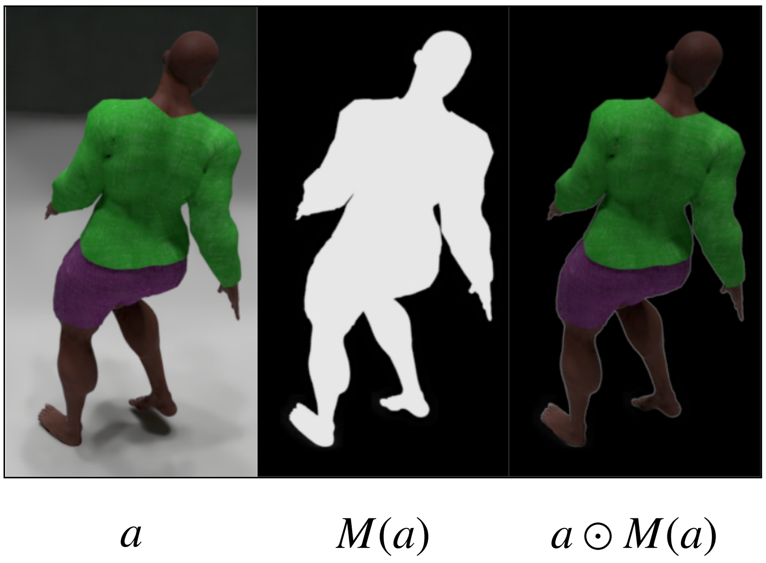


Figure 2.10: Figure showing the masking operation. The original image a is multiplied, element-wise, with its mask, $M(a)$, to yield an image $a \odot M(a)$ where the character in the image is singled out.

Attaining masks is not a trivial task for the computer, and we do not want to do it by hand. One solution to the problem is deep learning based models that can take an image as input, and generate a corresponding mask. We use the Mask R-CNN [22] model to generate masks for our real human images. In the case of our artificial images, however, we do not need a neural network to extract masks, since we can do it ourselves during the image generation. With the masks, we now have a way of comparing the character in an image before and after it has been translated, and can encourage our generators to keep the character from being heavily altered through the translation, incentivising identity preservation. The loss leading to this, called the *identity loss*, $\mathcal{L}_{\text{ID}}$, is defined as

$$\mathcal{L}_{\text{ID}} = \mathbb{E}_{a\sim p_{\text{data}}(A)}\left[\|(G_A(a) - a) \odot M(a)\|_2\right] + \mathbb{E}_{b\sim p_{\text{data}}(B)}\left[\|(G_B(b) - b) \odot M(b)\|_2\right], \qquad (2.9)$$

with $\odot$ denoting element-wise multiplication. The l_2 norm ($\|\cdot\|_2$) is shown to produce blurrier results than the l_1 norm ($\|\cdot\|_1$) [27] and after some preliminary tests we therefore change the norms in the identity loss (2.9) from l_2 to l_1.

2.3 Benchmark

To evaluate the potential of our datasets, we use them to train re-identification models. In this section, we describe how our models are trained, how we evaluate their accuracy, and the tools that we use in the process. All datasets are divided into three parts: a training set, a validation set and a test set. The test set and validation set consists of 100 separate characters each, and the rest are placed in the training set. There is no overlap between the three sets. The training set makes up the examples our model sees during its training, while the validation set is used to validate our model as it is training, allowing us to keep track of potential overfitting [49], and stop training before overfitting occurs. When a model has been trained using the training set, and selected using the validation set, it is evaluated on the previously unseen images in the test set, yielding a performance measurement.

2.3.1 Training

The idea of the training is that our model should learn how to embed images in a feature space, in which images of the same person are close to each other, and images of different people are far apart [67]. Here, embedding an image x in a feature space means applying a function $f : \mathbb{R}^{H\times W\times C} \to \mathbb{R}^N$ to it, which takes it from $\mathbb{R}^{H\times W\times C}$, where H, W and C is the height, width and number of channels in our images, respectively, to a single vector $f(x) \in \mathbb{R}^N$, with N being the length of the vector. This function is represented by an artificial neural network, meaning that applying the function to an image is done by passing the image through the network. Our code framework (see Section 2.3.4) allows us to choose the architecture of this network. How close the characters in two images are is then measured by computing the Euclidean distance between the feature embeddings of the two images. This setup is outlined in Figure 2.11.

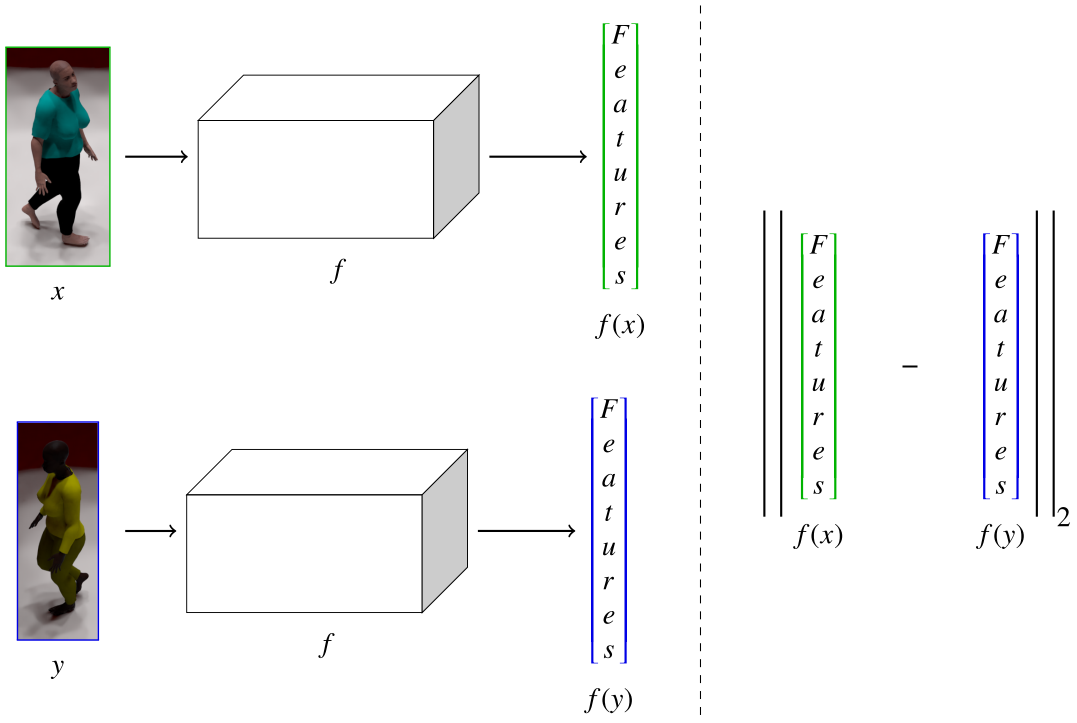


Figure 2.11: The evaluation setup. Images x and y are run through the feature embedding network, f, resulting in feature vectors $f(x)$ and $f(y)$. The architecture of the feature embedding network is determined by an option passed to our code framework (see Section 2.3.4). In this case, where the characters in the two images do not share identity, the two feature vectors should have a large (Euclidean) distance, $\|f(x) - f(y)\|_2$, between each other. When the characters do share identity, the distance should instead be small. The aim of the training is to have the network, f, adjust itself so that the features it produces have those properties.

Now the question is how the network is taught to embed our images in a good way. Similar to the networks used in our refiner (cf. Section 2.2), the feature embedding network, f, is a feed-forward neural network, and we teach it using backpropagation algorithms. This means that we have to add a loss function to the network, so that we can tell it how well it did for a given example. As was the case with the architecture of our feature embedding network, f, our code framework allows us to choose the loss function that we want to use. Our choice of loss function was the *softmax* loss [61], which is intended to enforce our model to classify each image as one of M classes, where M is the number of identities we allow the network to train on. In other words, it should teach the network to say which person is in an image, when it has M people to choose between. If the network classifies an image as portraying the correct individual, the loss is small, and vice versa. In our setup, the classification part is made up of a smaller network, which applies a function $g : \mathbb{R}^N \to \mathbb{R}^M$ to the feature vector $f(x) \in \mathbb{R}^N$, resulting in a new vector, $g(f(x)) \in \mathbb{R}^M$. The values in $g(f(x))$ are in the range $(0, 1)$ and can be seen as probabilities: a high value $g(f(x))_i$, $i \in \{1, 2, \ldots, M\}$, implies that the network predicts that character i is portrayed in image x, while a low value implies the opposite. Moreover, as the values in $g(f(x))$ are probabilities, they satisfy $\sum_{i=1}^{M} g(f(x))_i = 1$. Now, for each example, we obtain a loss based on the vector $g(f(x))$, which is used to train the *entire* model, i.e. it is used for training both the feature embedding network, f, and the classifier, g, and both of these are therefore trained as one large model. Figure 2.12 shows this setup.

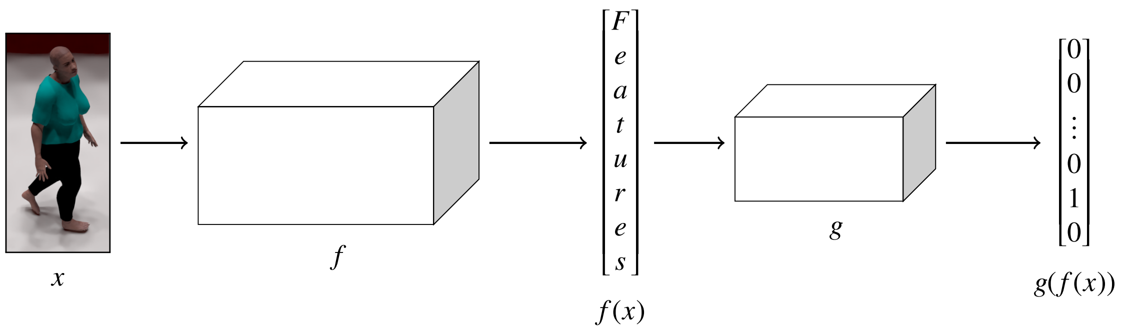

x
f
$\begin{bmatrix}F\\e\\a\\t\\u\\r\\e\\s\end{bmatrix}$
$f(x)$
g
$\begin{bmatrix}0\\0\\\vdots\\0\\1\\0\end{bmatrix}$
$g(f(x))$

Figure 2.12: The training setup is based on the softmax loss. An image is run through the feature embedding network, f, resulting in a feature vector $f(x)$. The feature vector is then passed through another, smaller network, g, which outputs a vector of probabilities, $g(f(x))$. This vector contains one element per character identity in our training set (with M identities), and each element represents the probability that the input image shows the corresponding character. Ideally, and this is what we teach the model to aim for, this vector should contain only 0s except for the element corresponding to the character in the input image, which should be a 1. The architecture of the feature embedding network, f, is selected by passing it as an option to our code framework (see Section 2.3.4). The addition of the classifier, g, is done when one passes the softmax loss as the choice of loss to the aforementioned framework. With other choices of the loss, the training setup is different.

When the model is trained to correctly classify the images we give it, the idea is that the features we obtain after passing images through the feature embedding network, f, should contain enough information to separate characters from each other. Put differently, given two images x and y, the trained network should yield features $f(x)$ and $f(y)$ which are far apart, i.e. $||f(x) - f(y)||_2$ is large, if the images contain different individuals, and close together, i.e. $||f(x) - f(y)||_2$ is small, if the images contain the same character.

Due to the appearance gap between artificial and real human images, a model trained on artificial images might have difficulties performing re-identification on real human images. However, the model is still trained to embed human images of some sort, which could make it a good starting point for training a new model that works better on real human images. Using a model trained on artificial images as the starting point for training a model with real human images is a type of *finetuning* [1, 30]. Finetuning is done by first training a model with an initial dataset, and then re-adjusting parts of the model and continue training with another dataset. In our case, we initially train a model with a dataset of artificial images, and then we continue training the model with real human images. Before we start using the real human images, we lower the rate at which the feature embedding network, f, is learning, and we reset the whole classification network, g. If there is a difference in the number of identities (M) between the datasets, we change the classification network, g, to output a vector of the correct length.

2.3.2 Evaluation Method

A trained model's accuracy is measured using the different feature vectors we obtain after passing images through its feature embedding network, f (cf. Figure 2.11). The evaluation is set up as follows:

1. Randomly choose 100 identities from an evaluation set (the validation set during training, and the test set during testing).
2. For each of the 100 identities, randomly pick two images. Put the first image into the *query*, and the second one into the *gallery*.
3. Compute and store the feature vector $f(x)$ for each image x in the query and the gallery
4. For each image x_i, $i = 1, 2, \ldots, 100$, in the query:
   (a) Compute the distance between its corresponding feature vector, $f(x_i)$, and all the feature vectors of the gallery.
   (b) Sort the distances in ascending order
   (c) See if the shortest distance is to the image in the gallery which portrays the same identity as the query image, x_i. If so, we have a rank 1 hit. If not, do the same thing for the second shortest distance. If the identities match, we have a rank 2 hit. Continue in the same fashion until the match is found.
   (d) For a rank k hit, store a vector $v_i = [0, 0, \ldots, 0, \underset{k}{1}, 1, \ldots, 1]$.
5. Sum all the vectors v_i, $i = 1, 2, \ldots, 100$, and divide each element in the resulting vector by 100.

The result of step 5 is a vector called the cumulative matching characteristic (CMC). The CMC shows us the percentage of our identities that resulted in rank k, $k = 1, 2, \ldots, 100$, hits, and it is usually presented as a curve (see e.g. Figure 3.1e). Ideally, we would only have rank 1 hits, as that would imply that all our images were matched correctly. This would then result in a horizontal curve at the value 1.

2.3.3 Validation

During training, the model is continuously validated to see how well the training is progressing. This is done by evaluating the model on the images in the validation set, following the procedure in Section 2.3.2. We compute a *validation score* at the end of each *epoch*, i.e. after a fixed number of training iterations, by summing the elements of the CMC (i.e. computing the area under the curve). This score should be a good measurement of how good our model is, as it is less prone to noise than the rank 1 accuracy, but is still larger the better our model's ranking accuracy is. By validating our model after each epoch, we can analyse how the score is developing, and see when the model is not getting any better, or starts overfitting to the training data [49]. Our final model is then chosen as the version of our model that achieved the highest validation score during training. We allow the training to run for many iterations, which should imply that we reach a peak, or a plateau, in validation score before the training finishes. Once there, we deem the model to have extracted as much information as it could from its training examples, and it should make a good representation of the dataset's potential. Therefore, the model should also be comparable to other models trained using the same procedure.

2.3.4 Code Framework

Open-ReID [60] is an open-source framework containing implementations of different re-id algorithms [23, 61, 62], and multiple commonly used metrics for evaluation. The framework also includes a data handler, allowing a new type of dataset (one containing artificial images, in our case) to be handled by the framework without us having to make major adjustments to neither the framework nor the dataset. The different algorithms and evaluation metrics can easily be set via the framework's execution options. The options include possibilities to use different network architectures for the feature embedding network, f, and the loss on which the training is based. Open-ReID includes the features we need to efficiently and repeatably compare the potential of our generated datasets, and was therefore our choice of evaluation framework. Note that, while we aim for optimal accuracy in each training, we focus on comparing our results to each other and not to state-of-the-art results.

We used the Inception [52] architecture for the feature embedding network and the softmax loss function for training. These options are implemented and included in the Open-ReID framework. The choice of the softmax loss function is what introduces the classifier, g, to our model (cf. Figure 2.12). The framework's data handler expects a set number of cameras for each dataset, but our artificial images did not have any fixed number of cameras or camera positions. Therefore, we randomly assigned one of two cameras to each image. All tests were done according to the CUHK-CMC calculations implemented in the Open-ReID framework.

2.3.5 Test Data

Our main tests have been done on the CUHK03 [34] dataset, which includes 1467 characters and 28193 images. It is not thoroughly documented, but the dataset includes both manually cropped and automatically cropped images of the same frames. The manual cropping means that the interesting part of the image (the character) has been cut out by hand, while the automatically cropped images have instead been cut out by a detector [34]. We used 100 test characters, defined and included in the dataset, for all our tests. During training we used the remainder of the images. The handling of the dataset is implemented in the Open-ReID framework [60].

Chapter 3

Evaluation

With the following evaluation, we aim at answering the following questions:

- How does changing the attributes of our generated images, i.e. backgrounds, number of accessories, types of garments, etc., affect the re-id model's accuracy?
- Does training on generated images passed through our refiners improve our model's accuracy?
- How does a model trained on artificial data compare to a model trained on only real data?
- Can we use real and artificial data jointly to create a model that outperforms the one trained on only real data?

To this end, we use four datasets with different attributes, created using the methods described in Section 2.1. Each of the datasets is given a number I - IV, and we will refer to their images as *generated* images. The images in dataset I are simple, with characters wearing plain coloured clothes and that do not have hair, shoes or accessories. Furthermore, the room's colours are the same for each character: the walls are red and the floor is grey. Dataset II adds some simple textures to the clothes, and this dataset has varied wall and floor colours for each character. Datasets III and IV both include more accessories and clothes, while dataset IV is the only dataset where we use a cubemap. Each dataset includes 1500 unique characters and 10 images of each character. Table 3.1 contains a summary of information about each of the four datasets. Samples from the datasets can be found in Appendix B, on pages 64-67.

The datasets are used, along with real images from the CUHK03 dataset, to train one CycleGAN refiner model each. When dataset *D* has been translated to the domain of CUHK03 images, using its corresponding refiner, we denote it R-*D*, and the images in

R-D are then referred to as *refined*. For D = IV, we also trained a PTGAN refiner. We distinguish between images from that and images from the CycleGAN refiner by denoting the corresponding datasets R-IV (CycleGAN) and R-IV (PTGAN). Samples of the refined images can be seen in Appendix B, on pages 68-72. After that, all our artificial datasets, i.e. datasets I - IV and datasets R-I – R-IV, are used to train one re-id model each, according to the training schemes presented in Section 2.3.1. We will refer to a model trained on dataset D, $D \in$ [I, II, III, IV, R-I, R-II, R-III, R-IV (CycleGAN), R-IV (PTGAN)], as model D. These models are then all tested on a set of 100, previously unseen, images from the CUHK03 dataset, following the evaluation protocol outlined in Section 2.3.2. In Section 3.1.1 we compare models I - IV, while Section 3.1.2 contains a comparison between models R-I – R-IV.

During all trainings we let the validation score converge to assure that each model's potential was reached (cf. Section 2.3.3). The validation scores of the models that produced the results included in Section 3.1 can be found in Appendix C, pages 73-79.

Table 3.1: Datasets used for evaluation and analysis, where the size of a dataset is the number of characters times the number of images per character.

Dataset	**Size**	**Description**
I	1500×10	Plain background, only shirts and pants, bright colours on clothes.
II	1500×10	Plain, varied backgrounds, only shirts and pants, textures on clothes.
III	1500×10	Plain, varied backgrounds, shirts, pants, hair, more textures on clothes.
IV	1500×10	Cubemap background, all available clothes and accessories, mixed textures on clothes.

3.1 Results

3.1.1 Training on Generated Images

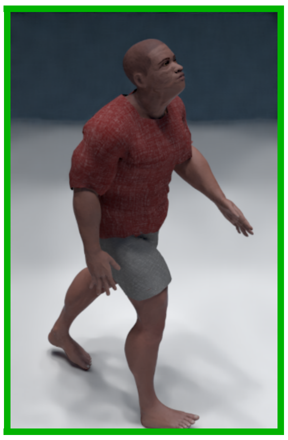
With this section, we investigate the impact of different attributes in our dataset using datasets I - IV. Sample images from these datasets can be seen in Figure 3.1. The results of testing models I - IV on the CUHK03 dataset can be seen in Figure 3.1e, where the *chance* curve represents randomly guessing a match for each character in the query.

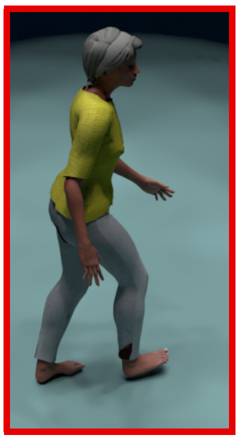
Figure 3.1e and table 3.2 show that all models perform better than chance, but have low accuracy for higher ranks (e.g. rank 1). The improvement from model I to II is significant, but it is hard to tell which change between the datasets had the most effect on the accuracy. Datasets III and IV both include more accessories and clothes, but the model based on dataset II still performs better than model III. Dataset IV is the first to add cubemaps as backgrounds, making the images look more real to the human eye. While it is the model that performs best, the difference is not as significant as one might believe when considering the difference in realism of dataset IV compared to the other three sets. The results for model IV are comparable to the results of the pre-deep learning approaches trained and tested on CUHK03 [59].

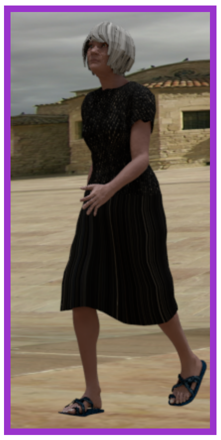
Table 3.2: Cumulative matching characteristics from our evaluations of models I - IV on the CUHK03 test set. Each row shows the corresponding model's accuracy on different ranks (in percentage). Chance is included for reference.

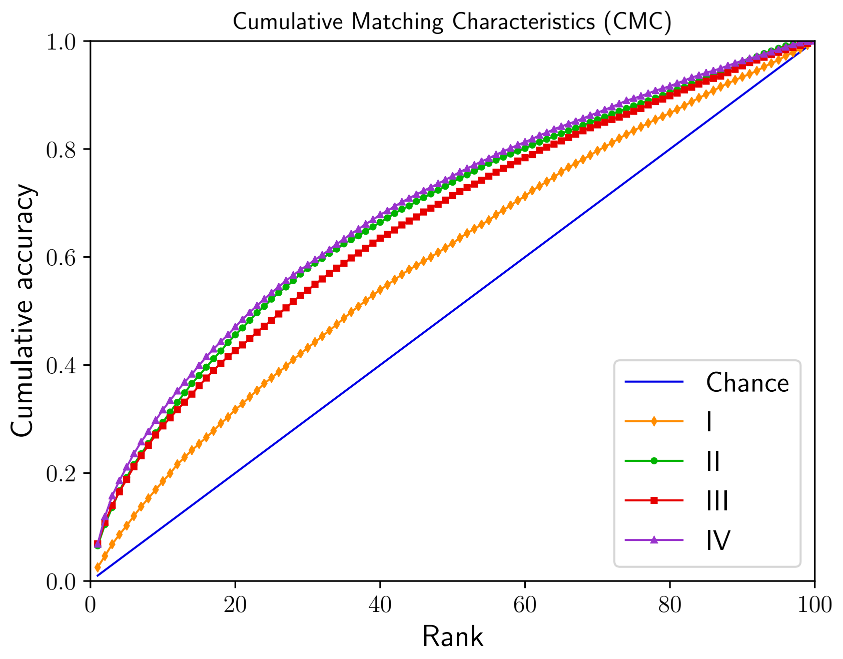
M	Rank 1	Rank 5	Rank 10	Rank 20
Chance	1.0	5.0	10.0	20.0
I	2.5	10.2	18.5	31.7
II	6.5	19.2	29.4	45.6
III	7.0	18.8	28.7	42.7
IV	7.0	21.2	31.7	47.0

(a) I.


(b) II.


(c) III.


(d) IV.



(e) CMC.

Figure 3.1: Figures (a)-(d) hold samples from datasets I - IV. Figure (e) shows the results of our evaluations of models I – IV on the CUHK03 test set, where the colours of the frames in figures (a)-(d) correspond to the colours in plot (e). For instance, model I's result is shown in orange. Figure (e) shows that models II - IV perform similarly, while model I performs slightly worse.

3.1.2 Training on Refined Images

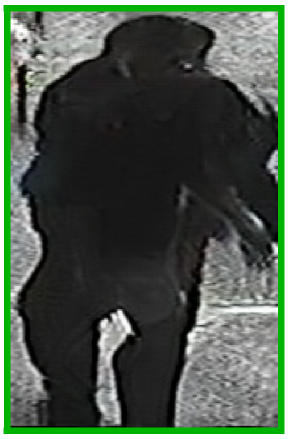
In this section, we will look at the refined images in datasets R-I – R-IV, from which sample images can be found in Figure 3.2. Comparing the images in Figure 3.1a-3.1d and Figure 3.2a-3.2d, the effects of the refiner is noticeable. Most of the colours are more dull, the backgrounds are heavily modified, and there are some artefacts around the characters, making some shapes hard to tell apart. Figure 3.2e shows a large overall accuracy improvement for all models, compared to those trained on datasets I - IV, answering the question if training on refined data improved the resulting model's accuracy. The results of model R-IV (see table 3.3) is comparable to the results of the first deep learning approach used on CUHK03 [34, 59]. Comparing the results of models I - IV to those of models R-I – R-IV, we see that there are larger differences between the models' accuracy when they are trained on refined data compared to when generated data is used. Furthermore, dataset R-II does not result in a model that performs similar to the ones yielded with datasets R-III and R-IV, which was the case for dataset II compared to datasets III and IV. Model R-III and R-IV still achieve similar accuracy, however. We also see that, in each case, a model trained on refined data achieves a higher accuracy than a model trained on generated data (cf. tables 3.2 and 3.3).

Figure 3.3 shows samples and a CMC plot yielded by models that were trained on datasets IV, R-IV (CycleGAN), R-IV (PTGAN), and also the CUHK03 dataset itself (but on a set of images separate from the test set). In Figure 3.3e we see the accuracy improvements that our refiner lead to, but we also see how the model that is trained on the CUHK03 dataset outperforms even model R-IV by a large margin.

Table 3.3: Cumulative matching characteristics from our evaluations of models R-I – R-IV on the CUHK03 test set. Each row shows the corresponding model's accuracy on different ranks (in percentage). The table also includes the evaluation result of the model that we trained on images from the CUHK03 set.

M	Rank 1	Rank 5	Rank 10	Rank 20
R-I	8.1	23.9	35.8	51.7
R-II	13.1	32.9	46.0	62.5
R-III	19.2	43.9	57.6	73.6
R-IV	22.2	48.2	62.0	76.8
CUHK03	74.1	94.7	97.7	99.0

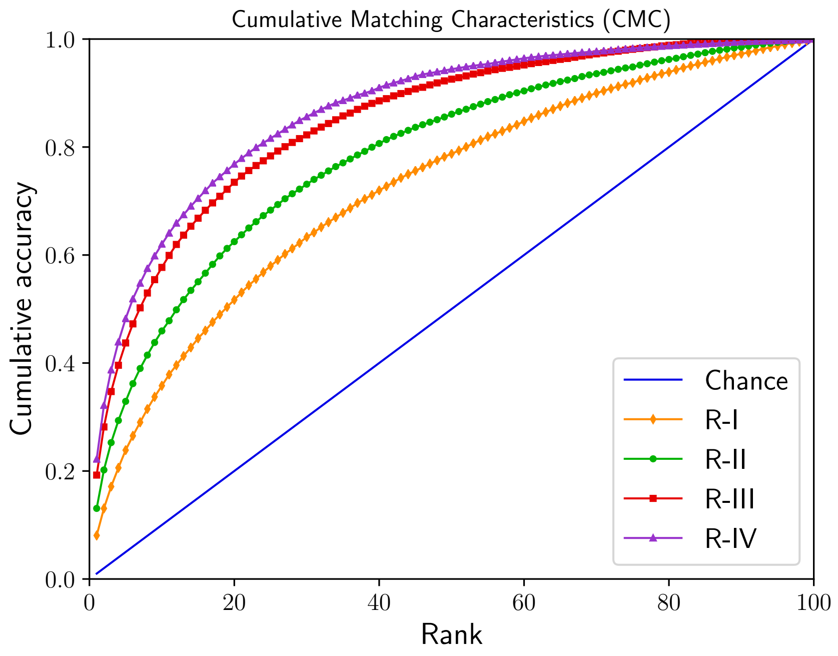
(a) R-I.



(b) R-II.

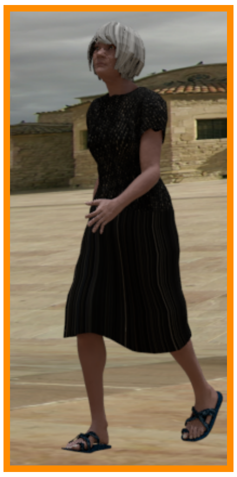
(c) R-III.

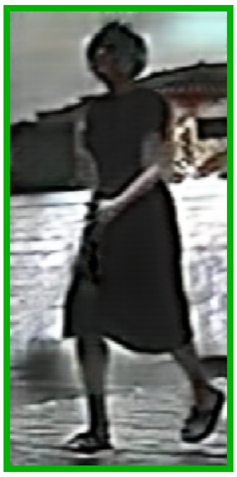
(d) R-IV.

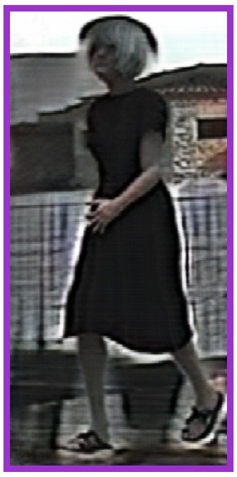


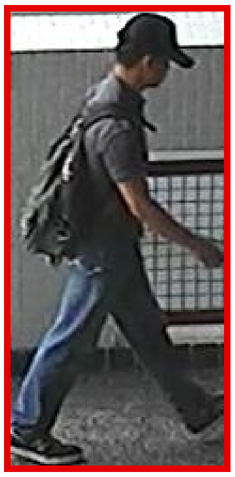
(e) CMC.

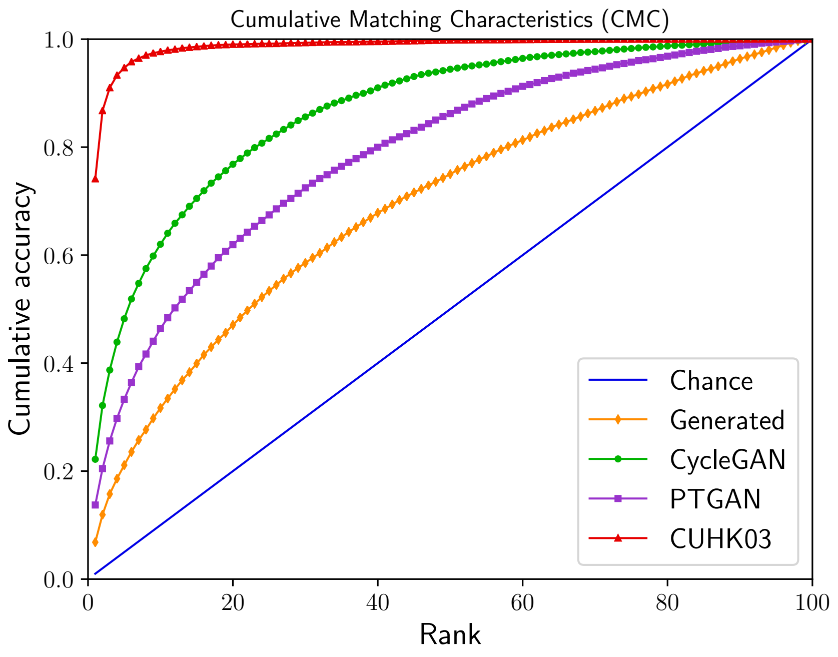
Figure 3.2: Figures (a)-(d) hold samples from datasets R-I – R-IV. Figure (e) shows the results of our evaluations of models R-I – R-IV on the CUHK03 test set, where the colours of the frames in figures (a)-(d) correspond to the colours in plot (e). We see that models R-III and R-IV shows the best accuracy, while model R-II is slightly worse, and model R-I yields the worst accuracy.


(a) Generated (IV).


(b) CycleGAN (R-IV).


(c) PTGAN (R-IV).


(d) CUHK03.



(e) CMC.

Figure 3.3: Figures (a)-(d) hold samples from datasets IV, R-IV (CycleGAN), R-IV (PTGAN) and CUHK03. Figure (e) shows the results of our evaluations of models IV, R-IV (CycleGAN), R-IV (PTGAN) and a model that we trained on CUHK03 images, where the colours of the frames in figures (a)-(d) correspond to the colours in plot (e). The evaluation was done on the CUHK03 test set. We can see that the model trained on CUHK03 images outperforms the other three, and that model R-IV (CycleGAN) performs better than models IV and R-IV (PTGAN).

3.1.3 Training on Artificial and Real Images

To answer the question if artificial and real data can be combined to train a model, we used initial trainings as in the previous section and finetuned them with real data (cf. Section 2.3.1). Figure 3.4 and table 3.4 show the accuracy of models that were initially trained on datasets IV, R-IV (CycleGAN) and R-IV (PTGAN), respectively, before being finetuned on the CUHK03 set. Using generated data for the initial training reduces the accuracy on CUHK03, and so does using the data that was refined with the PTGAN. The model that was initially trained on dataset R-IV (CycleGAN) produced a CMC that was similar to the one produced by the model trained on images from the CUHK03 set.

Table 3.4: Cumulative matching characteristics from evaluation of finetuned models, along with the model we trained on only CUHK03 images. The models were tested on the CUHK03 test set. Each row shows the corresponding model's accuracy on different ranks (in percentage).

M	Rank 1	Rank 5	Rank 10	Rank 20
IV + CUHK03	67.6	90.5	95.7	98.3
R-IV (CycleGAN) + CUHK03	72.9	93.0	96.4	98.3
R-IV (PTGAN) + CUHK03	66.5	88.6	94.4	97.5
CUHK03	74.1	94.7	97.7	99.0

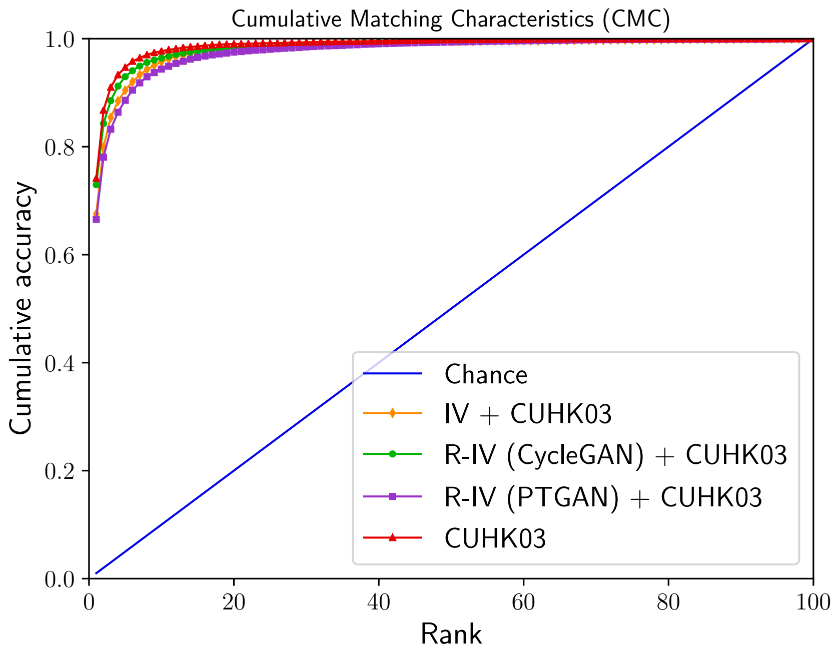


Figure 3.4: CMC yielded by models initially trained on artificial images and finetuned on CUHK03 images. The models were tested on the CUHK03 test set. Including artificial images in the training reduces the models' final results in each of the cases, compared to only using CUHK03 images.

3.2 Discussion

3.2.1 Analysis

Looking at Figure 3.1, we see that all four models have rather low accuracy. Models II - IV show similar accuracy, while model I performs worse. In order to see if the models did not train properly or if they could not handle the appearance difference between training images and real human images, we tested models I - IV on their corresponding test sets. The results can be seen in Figure A.1 in Appendix A, page 60. They show high accuracy for all models, implying that the problem lies in the appearance differences compared to real human images. We believe that, with the vibrant colours of our generated datasets, the re-id models trained on them become closer to colour re-identifiers than person re-identifiers, as the colour variations in the datasets make the characters identifiable based on their cloth colours alone. The real human images pose a larger difficulty, as the people within them cannot be re-identified based only on their cloth colours, which might be why our models do not achieve high accuracy when tested on such. The reason, then, that models II - IV perform better than model I could be that the cloth colours in the corresponding datasets are less saturated, making them look more like the colours in the CUHK03 set. Moreover, the fact that the backgrounds are more varied in the latter three sets could suggest that the re-id models have to learn how to reduce the background's impact on its predictions, helping them to perform well despite being faced with new backgrounds. In the case of the first dataset, the model might instead learn to look for, and neglect, a certain background. When that background is then changed in the test case, the model cannot recognise it, and could begin using the background to identify the character in the image, making the identification more difficult.

The results in Figure 3.1 and 3.2 give us some information about how the attributes of a dataset impact the model's accuracy. We see that with a dataset containing more cloth types, cloth textures, hair styles, and accessories, the resulting model's accuracy increases. The improvement is negligibly small between datasets II - IV, but, when the datasets are put through the refiners, we see that the added attributes between dataset II and datasets III – IV, seem to have had a greater impact, as model R-II performs significantly worse than models R-III and R-IV. We believe that this is caused by the refiners: when a refiner is trained on a dataset that covers a larger space, i.e. has more attributes, it will have more information to work with when it tries to translate images to fool the discriminator. This could then mean that it is able keep more of the information in the original image, thus preserving the character's identity better, while still achieving a high probability of being real. In turn, this could imply that the re-id model we train with these refined images becomes better, as the appearance of a character is more consistent between images.

Our model's accuracy is improved greatly when we train on refined images, compared to when we train on generated ones (cf. Figure 3.3e). This could follow from the discussion in the first paragraph of this section, where we saw that the similarity in colours between the training images and test images seemed to have a large impact on the trained model's accuracy. Thus, as the refiner translates our images to be more similar to the ones in the dataset from which the test set is taken, it seems reasonable to believe that the refined images would lead to a better model than the generated ones. Note that we write *more similar to the test set*, rather than *more realistic*. The refined images of dataset R-IV (CycleGAN) do not necessarily look more realistic than the generated images of dataset IV, but they are more similar to images from the CUHK03 set, which is probably why they lead to a better model. Therefore, creating artificial images similar to the images that the model will be faced with during its test phase should be prioritised over achieving realistic looking ones. Comparing the results of model R-IV (CycleGAN) with those yielded by the model trained on CUHK03 images, however, we see that there seems to still be a large difference between the artificial and real images, as model R-IV (CycleGAN) has significantly worse accuracy. In order to examine the importance of our dataset's similarity to the test set versus its realism, we tested model IV, R-IV (CycleGAN), R-IV (PTGAN) and the model trained on images from the CUHK03 set on a new, real test set, not originating from CUHK03 (see results in Figure A.2, Appendix A, page 61). This test showed us that the realism of the CUHK03 images did lead to a better model than the others, but the improvement was not as significant as when we tested the models on CUHK03 images.

Images from the R-IV (PTGAN) dataset portray characters that seem to have been preserved better during translation than the characters in the R-IV (CycleGAN) images, implying that the PTGAN did fulfil its purpose. The reason its refined images did not lead to a better model than the ones refined by the CycleGAN could be that the improved identity preservation made the refined images keep the colour of the humans from the original ones. This could have lead to the re-id model becoming a colour re-identifier again, resulting in a worse model.

Combining artificial and real data is an interesting idea, as it could be a way of producing better results than what could be achieved using only real data. However, the accuracy of our finetuned models, seen in Figure 3.4 and table 3.4, do not imply this. Prior works have used similar finetuning procedures from which they obtain models that were improved over the model trained with only real data [1, 30]. We believe that our artificial data, re-id infrastructure, and training procedure could be adapted to optimise for this use case, and, with that, result in models that do perform better than the one trained on real data alone.

3.2.2 Future Work

There are multiple interesting continuations to our work. For the generation, it would be interesting to add more cloth types, accessories, face and body variations, as well as using more realistic colour choices for the character's garments and accessories. We have not examined different sampling schemes for these different character attributes, but doing so could be beneficial when one wants to use a large set of artificial images to train a re-id model, since a good data distribution can have significant impact when training and using computer vision models [30]. We also see exciting possibilities with more advanced camera models that make the generated images look more like they were taken by real cameras [6]. Furthermore, it would be interesting to examine the effect of multiple characters in the same scene, along with other distractors such as objects occluding parts of our characters [53].

Our refiner models are trained with a loss function that is based on the pixel-wise difference between different images (see equations (2.6) and (2.9) on page 25). However, there have been improvements in the quality of image-to-image translation results when the models instead consider differences between images after they have been passed through parts of another network, specialised in object classification [14, 28]. It might also be worthwhile to investigate the refinement results when other datasets than CUHK03 are used during the refiner training, as this could lead to a refiner which produces images that later result in a more general re-id model. On that same topic, another intriguing continuation would be to have a single refiner that can be told to translate our generated images to look like images that could have come from one of multiple different, real datasets, and not just from one, which was the case for our refiner models [25].

Another possibility would be to do site-specific generation and refinement. That is, if one aspires to set up a re-id system at a specific location, e.g. a store, they could capture a cubemap at the location and use that as background during image generation. This would lead to images that look like they were captured at the site where the re-id system was to be set up. Additionally, one could capture a small set of real images at the site and use those to train a refiner that in turn can make the generated images be even more similar to the images that the re-id model will face when it is in use. If this process was streamlined, a site-specific model could be deployed at a site without having to do any manual annotation, and it could possibly perform well as it was trained for exactly that site.

The training setup we used for our re-id models was not adapted to artificial data, but was the same as the training setup used for training re-id models with real data. Adapting the setup for artificial data would presumably improve our models' accuracy. For example, different architectures of the feature embedding network, f (see Figure 2.12), could be explored to make it more apt to artificial images and finetuning [1]. The feature embedding network could also be a network that was already thoroughly trained to perform another, related task (e.g. object detection), before we used it as our feature embedder. Such a network could, with its prior training in image processing, make a good starting point as a feature embedding network for re-id. Furthermore, the loss function could be replaced with a function which directly separates the feature vectors from the feature embedding network [23], instead of separating them indirectly via the classifier, g (see Figure 2.12).

In addition to this, it would be interesting to see if the training setup could be adapted to use both real and artificial images simultaneously. One way to do this would be to alternate which data is used between training iterations and change the hyperparameters based on which data is in use and how the training is progressing. The difference compared to the finetuning we performed (see Section 2.3.1) is that we would alternate datasets during training, as opposed to first using only artificial images and then only real images.

Our framework for generating images could be extended for other tasks. It could, for example, be interesting to replace our characters with other objects and use the images to train models for object detection [53] and tracking [15]. A more direct use of our framework would be to use our full body human images to train models that can perform e.g. pose estimation [8], as it is easy to extract the character's pose during image creation, and the user can decide what poses should be possible for a character.

3.2.3 Ethical Aspects

Using artificial data allows for training deep learning models without a lot of real images, and therefore rely less on personal data. However, using artificial data adds another crafted step where bias from the developers can be introduced, possibly resulting in worse model accuracy. Person re-identification does not require recognising a person as a specific individual, but rather just comparing if two images hold the same person, which does not necessitate knowing the person's identity. This might not be intrusive on a person's privacy in itself, but if it is combined with a step where the person's identity is known, this privacy is lost. Even if the use of artificial training data helps with personal integrity in some ways, the trained computer vision model could be used for a lot of purposes that work in the opposite direction.

Chapter 4

Conclusions

In this project we have generated images of humans with different backgrounds, clothes, hair styles and accessories. Our main contribution is a framework that allows for generating varied characters and images in large scale. We show that the image datasets can be used as training data for deep learning applications by training re-identification models with them. In order to bridge the gap in appearance between generated and real images, we have used refiners based on Generative Adversarial Networks, which were shown to improve our datasets' potential.

Our models trained on artificial data did not achieve accuracy as high as models trained on only real data. The generation, as well as the training setup for our re-identification models, could be further investigated and improved upon to increase this accuracy. Combining artificial and real data should help in reaching higher accuracy than when real data is used alone. We did not achieve this in our tests, but we believe it could be done with further investigations of our procedures.

Bibliography

[1] Igor Barros Barbosa, Marco Cristani, Barbara Caputo, Aleksander Rognhaugen, and Theoharis Theoharis. Looking beyond appearances: Synthetic training data for deep cnns in re-identification. *Computer Vision and Image Understanding*, 2017.

[2] Manuel Bastioni. Manuelbastionilab. `http://www.manuelbastioni.com/`, accessed 13-06-2018, 2018. Version 1.6.1a.

[3] Loris Bazzani, Marco Cristani, and Vittorio Murino. Symmetry-driven accumulation of local features for human characterization and re-identification. *Computer Vision and Image Understanding*, 117:130–144, 2013.

[4] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2018.

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[6] Alexandra Carlson, Katherine A Skinner, and Matthew Johnson-Roberson. Modeling camera effects to improve deep vision for real and synthetic data. *arXiv preprint arXiv:1803.07721*, 2018.

[7] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 1, 2017.

[8] Wenzheng Chen, Huan Wang, Yangyan Li, Hao Su, Zhenhua Wang, Changhe Tu, Dani Lischinski, Daniel Cohen-Or, and Baoquan Chen. Synthesizing training images for boosting human 3d pose estimation. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 479–488. IEEE, 2016.

[9] The MakeHuman Community. Makehuman. http://www.makehumancommunity.org/, accessed 18-06-2018.

[10] Joey de Vries. Learn opengl, 2017. Third printing.

[11] Rick Delgado. From edison to internet: A history of video surveillance. https://www.business2community.com/tech-gadgets/from-edison-to-internet-a-history-of-video-surveillance-0578308, accessed 12-06-2018.

[12] Michael Doggett and Jacob Munkberg. Computer graphics introduction to 3d - edaf80. http://fileadmin.cs.lth.se/cs/Education/EDA221/lectures/Lecture1_web.pdf, accessed 24-06-2018.

[13] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38:295–307, 2016.

[14] Deniz Engin, Anıl Genç, and Hazım Kemal Ekenel. Cycle-dehaze: Enhanced cyclegan for single image dehazing. *arXiv preprint arXiv:1805.05308*, 2018.

[15] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. *arXiv preprint arXiv:1605.06457*, 2016.

[16] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[17] Martin Giles. The ganfather: The man who's given machines the gift of imagination. *MIT Technology Review*, 2018.

[18] Shaogang Gong, Marco Cristani, Shuicheng Yan, and Chen Change Loy. *Person re-identification*. Springer, 2014.

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

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[21] Cmu graphics lab. Cmu graphics lab motion capture database. http://mocap.cs.cmu.edu/, accessed 13-06-2018.

[22] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[23] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[24] Timothy Huang and Stuart Russell. Object identification in a bayesian context. In *IJCAI*, volume 97, pages 1276–1282, 1997.

[25] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. *arXiv preprint arXiv:1804.04732*, 2018.

[26] Solomon Hykes. *Docker*. Docker, Inc., Docker, Inc., San Francisco, 2018.

[27] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, 2017.

[28] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[29] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 746–753. IEEE, 2017.

[30] Adam Kortylewski, Andreas Schneider, Thomas Gerig, Bernhard Egger, Andreas Morel-Forster, and Thomas Vetter. Training deep face recognition systems with synthetic data. *arXiv preprint arXiv:1802.05891*, 2018.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[32] Igor Kviatkovsky, Amit Adam, and Ehud Rivlin. Color invariants for person reidentification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:1622–1634, 2013.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

[34] Wei Li, Rui Zhao, Tong Xiao, and Xiaogang Wang. Deepreid: Deep filter pairing neural network for person re-identification. In *CVPR*, 2014.

[35] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[36] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[37] Ramakant Nevatia and Thomas O Binford. Description and recognition of curved objects. *Artificial intelligence*, 8:77–98, 1977.

[38] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 1278–1286. IEEE, 2015.

[39] Emil Persson. Humus. http://www.humus.name/, accessed 13-06-2018.

[40] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[41] Weichao Qiu, Fangwei Zhong, Yi Zhang, Siyuan Qiao, Zihao Xiao, Tae Soo Kim, and Yizhou Wang. Unrealcv: Virtual worlds for computer vision. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 1221–1224. ACM, 2017.

[42] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[44] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, pages 102–118. Springer, 2016.

[45] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[46] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[47] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from simulated and unsupervised images through adversarial training. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, page 6, 2017.

[48] Cory Simon. Generating uniformly distributed numbers on a sphere. `http://corysimon.github.io/articles/uniformdistn-on-sphere/`, accessed 12-06-2018.

[49] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[50] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15:1929–1958, 2014.

[51] Baochen Sun and Kate Saenko. From virtual to reality: Fast adaptation of virtual object detectors to real domains. In *BMVC*, volume 1, page 3, 2014.

[52] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[53] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Train-

ing deep networks with synthetic data: Bridging the reality gap by domain randomization. *arXiv preprint arXiv:1804.06516*, 2018.

[54] Apostolia Tsirikoglolu, Joel Kronander, Magnus Wrenninge, and Jonas Unger. Procedural modeling and physically based rendering for synthetic data generation in automotive applications. *arXiv preprint arXiv:1710.06270*, 2017.

[55] The European Union. The general data privacy regulation. `https://www.eugdpr.org/`, accessed 06-04-2018.

[56] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3156–3164. IEEE, 2015.

[57] Longhui Wei, Shiliang Zhang, Wen Gao, and Qi Tian. Person transfer gan to bridge domain gap for person re-identification. *arXiv preprint arXiv:1711.08565*, 2017.

[58] Erik W. Weisstein. Archimede's hat-box theorem. `http://mathworld.wolfram.com/ArchimedesHat-BoxTheorem.html`.

[59] Lin Wu, Chunhua Shen, and Anton van den Hengel. Personnet: Person re-identification with deep convolutional neural networks. *CoRR*, abs/1601.07255, 2016.

[60] Tong Xiao. Open-reid. `https://github.com/Cysu/open-reid/tree/3293ca79a07ebee7f995ce647aafa7df755207b8`, accessed 14-06-2018, 2017.

[61] Tong Xiao, Hongsheng Li, Wanli Ouyang, and Xiaogang Wang. Learning deep feature representations with domain guided dropout for person re-identification. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 1249–1258. IEEE, 2016.

[62] Tong Xiao, Shuang Li, Bochao Wang, Liang Lin, and Xiaogang Wang. Joint detection and identification feature learning for person search. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3376–3385. IEEE, 2017.

[63] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Deep metric learning for person re-identification. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 34–39. IEEE, 2014.

[64] Wojciech Zajdel, Zoran Zivkovic, and BJA Krose. Keeping track of humans: Have i seen this person before? In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2081–2086. IEEE, 2005.

[65] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.

[66] Xi Zhang, Yanwei Fu, Andi Zang, Leonid Sigal, and Gady Agam. Learning classifiers from synthetic data using a multichannel autoencoder. *arXiv preprint arXiv:1503.03163*, 2015.

[67] Liang Zheng, Yi Yang, and Alexander G Hauptmann. Person re-identification: Past, present and future. *arXiv preprint arXiv:1610.02984*, 2016.

[68] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

Appendices

Appendix A

Complementary Results

In this appendix we include results that were not presented in Section 3.1.

Figure A.1 shows how models I - IV perform when they are evaluated on the test sets of their corresponding datasets.

Figure A.2 shows how model IV, model R-IV (CycleGAN), model R-IV (PTGAN) and the model trained on CUHK03 perform on another, real dataset. We call this dataset CHICA103. CHICA103 is a dataset collected at a Swedish supermarket.

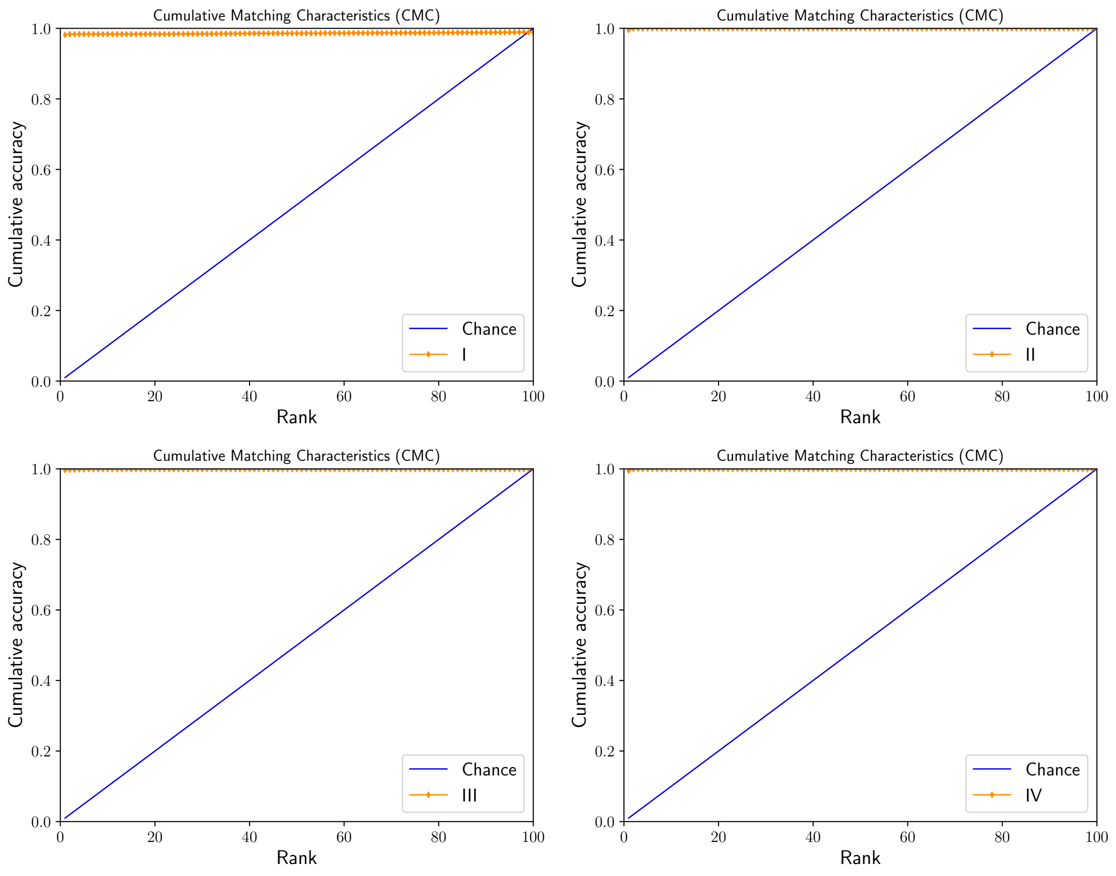
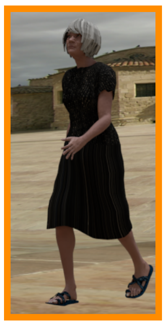
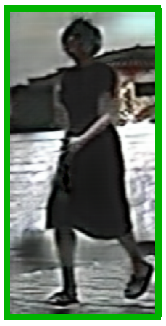
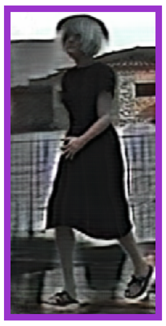


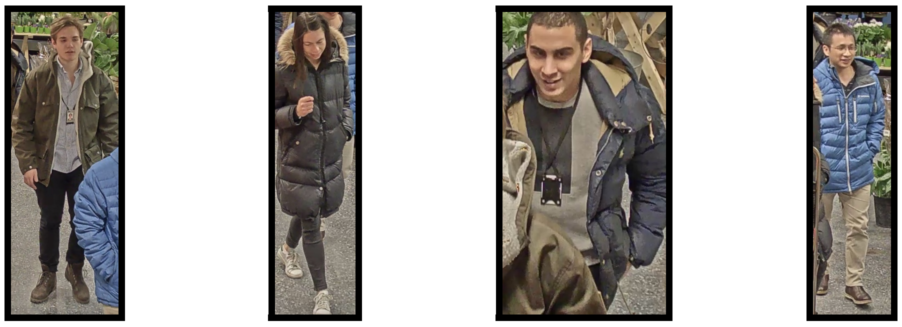
Figure A.1: Models I - IV each tested on the test set of the dataset it was trained on.

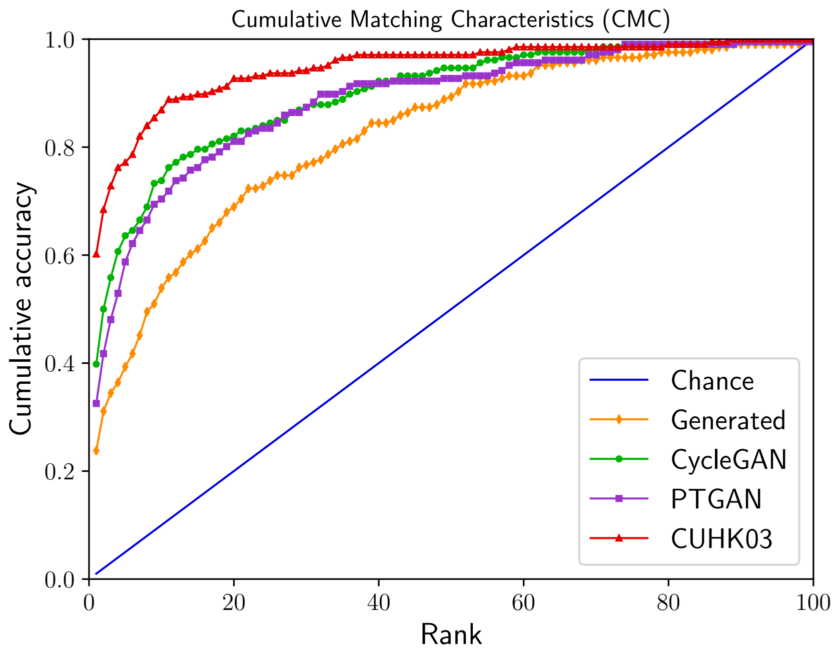

(a) Generated (IV).


(b) CycleGAN (R-IV).


(c) PTGAN (R-IV).

(d) CUHK03.


(e) CHICA103.



(f) CMC

Figure A.2: Figures (a)-(d) hold samples from datasets IV, R-IV (CycleGAN), R-IV (PTGAN) and CUHK03. Figure (e) shows samples from CHICA103. Figure (f) shows evaluation results yielded by models trained on the aforementioned datasets, where the colours of the frames in figures (a)-(d) correspond to the colours in plot (f). We can see that the model trained on CUHK03 images outperforms the other three, and that model R-IV (CycleGAN) performs better than models IV and R-IV(PTGAN).

Appendix B

Images from Our Datasets

Figures B.1-B.4 show samples of images from datasets I - IV.

Figures B.5-B.9 show samples of images from datasets R-I – R-IV (CycleGAN and PTGAN).

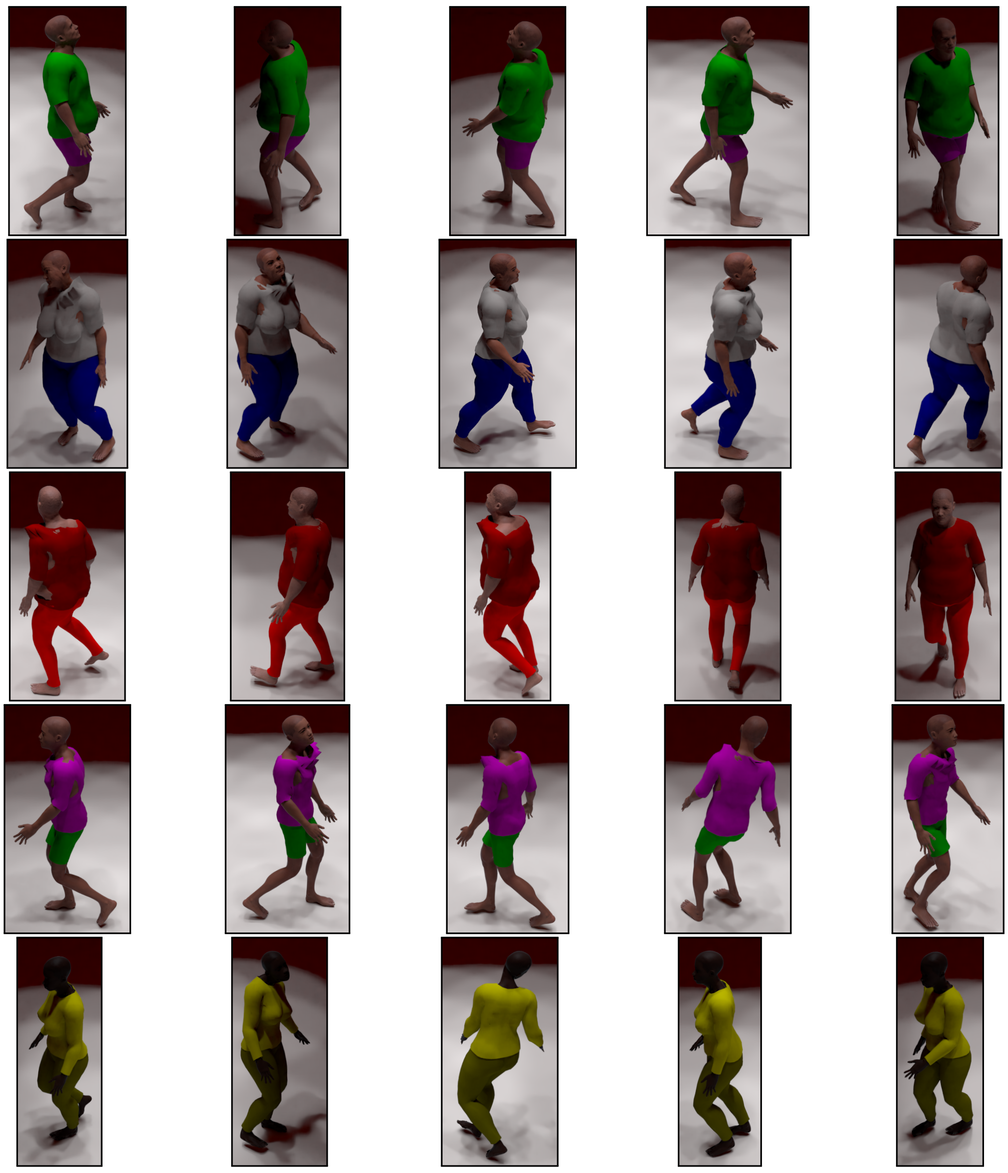


Figure B.1: Dataset I. Description: Plain background, only shirts and pants, bright colours on clothes.

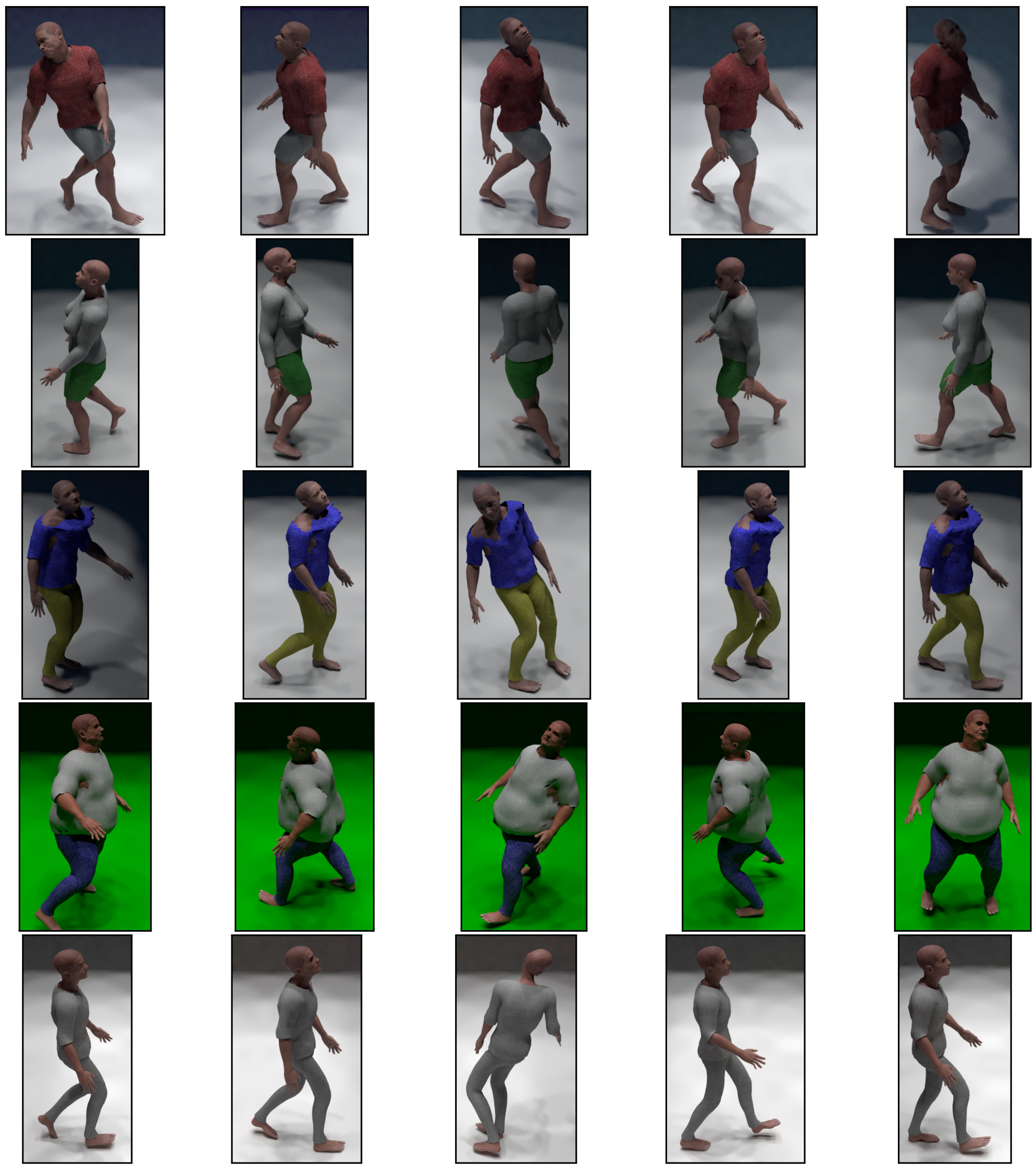


Figure B.2: Dataset II. Description: Plain, varied backgrounds, only shirts and pants, textures on clothes.

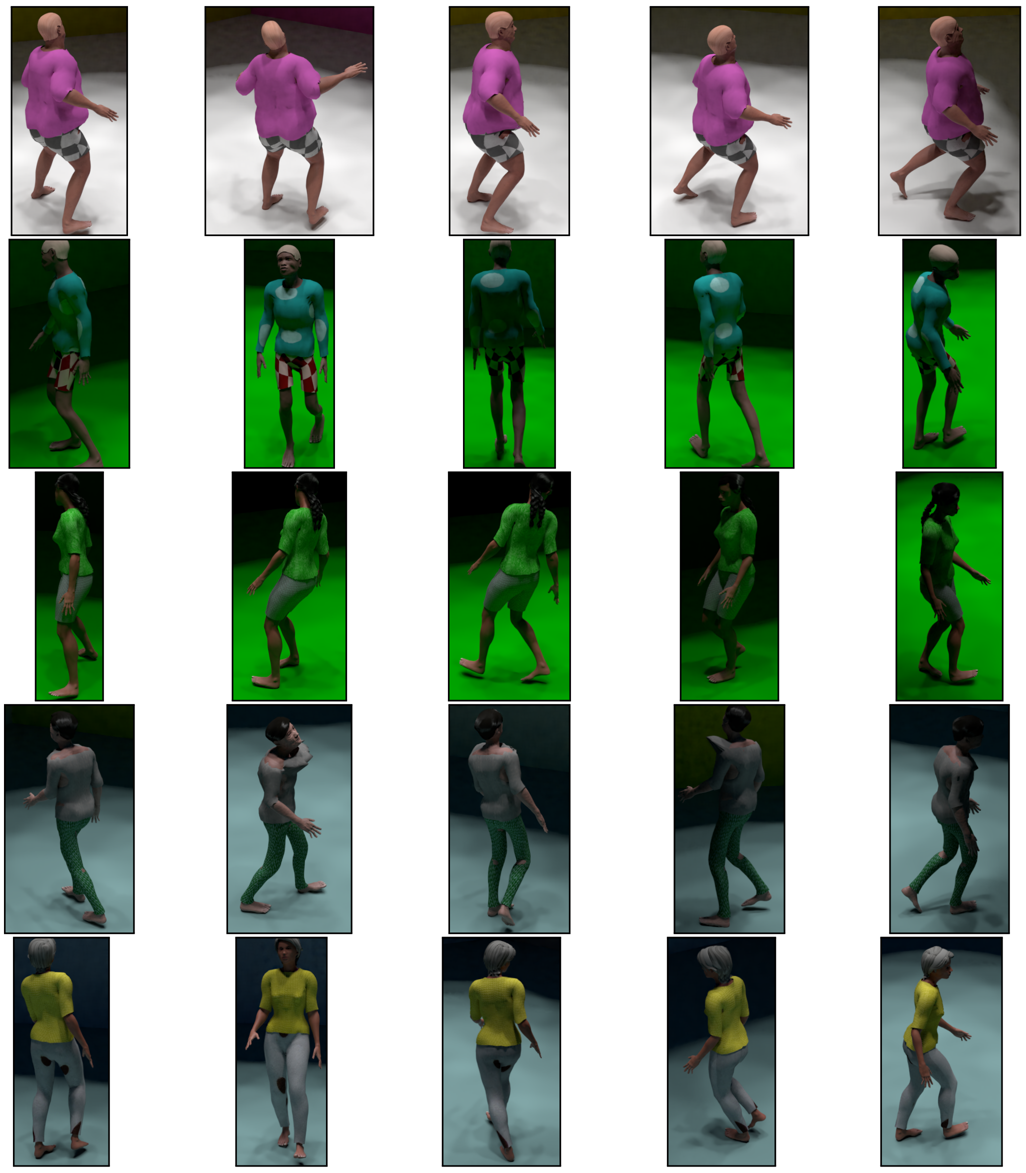


Figure B.3: Dataset III. Description: Plain, varied backgrounds, shirts, pants, hair, more textures on clothes.

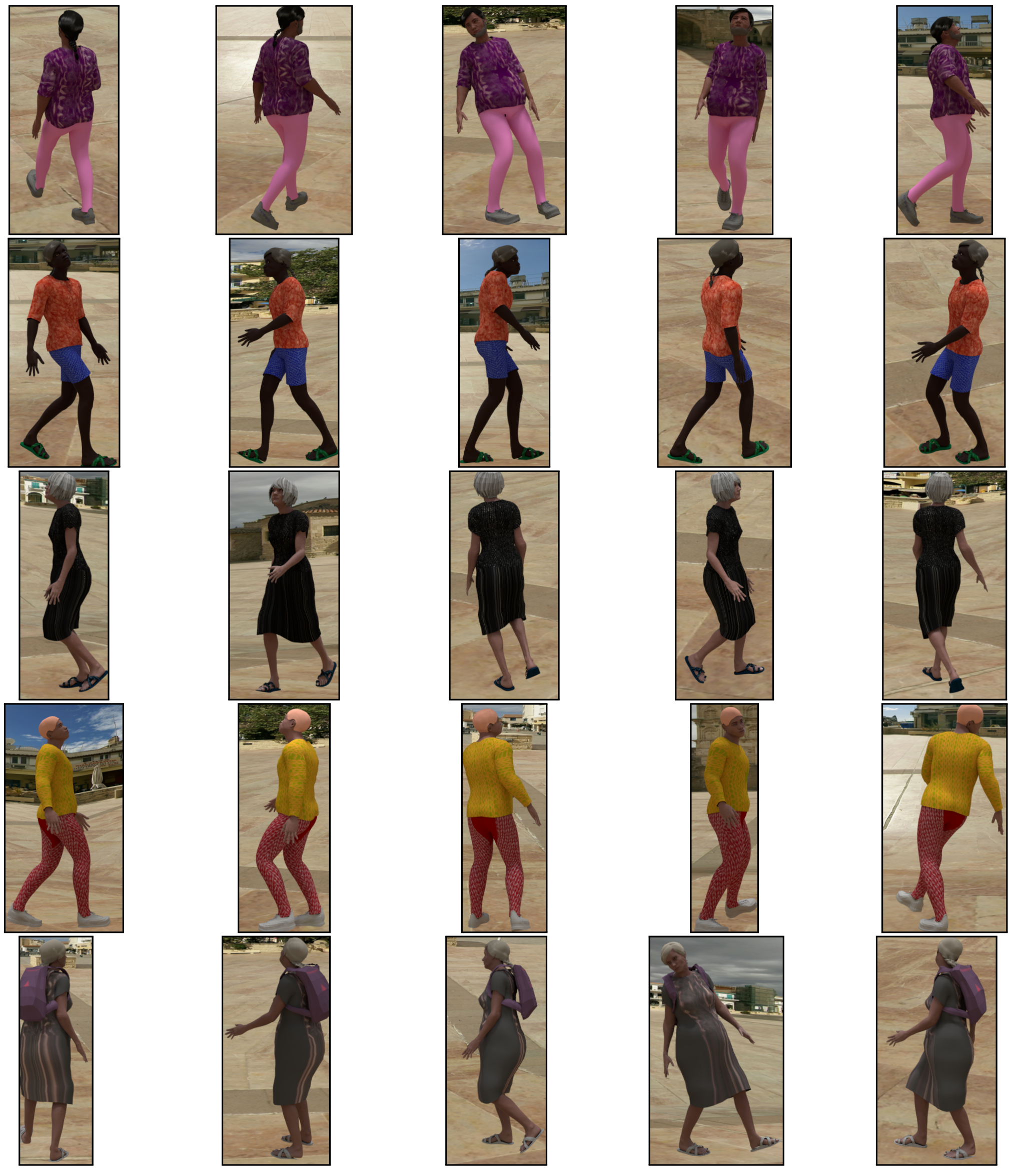


Figure B.4: Dataset IV. Description: Cubemap background, all available clothes and accessories, mixed textures on clothes.

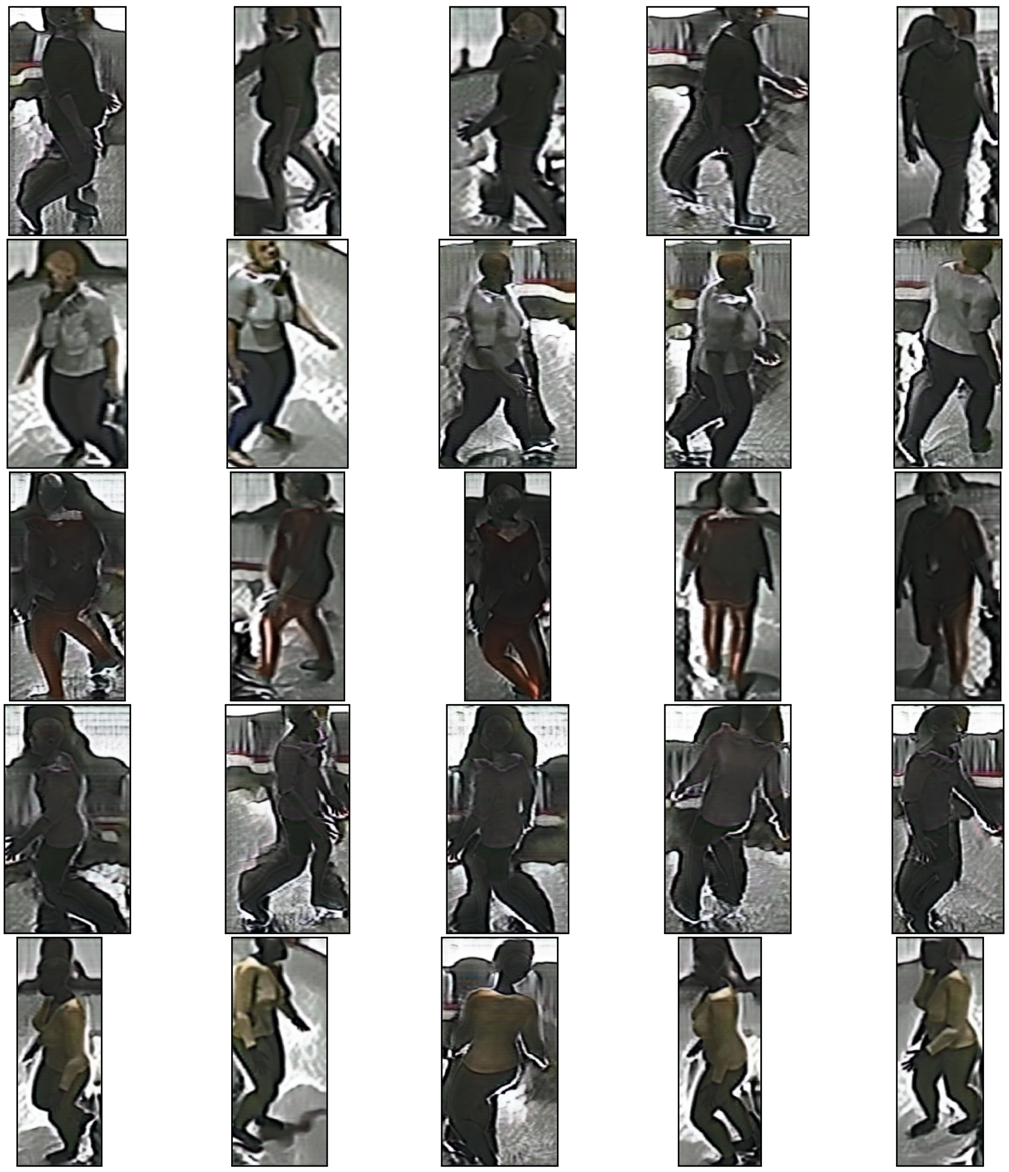


Figure B.5: Dataset R-I. Refined version of dataset I with the CycleGAN refiner.

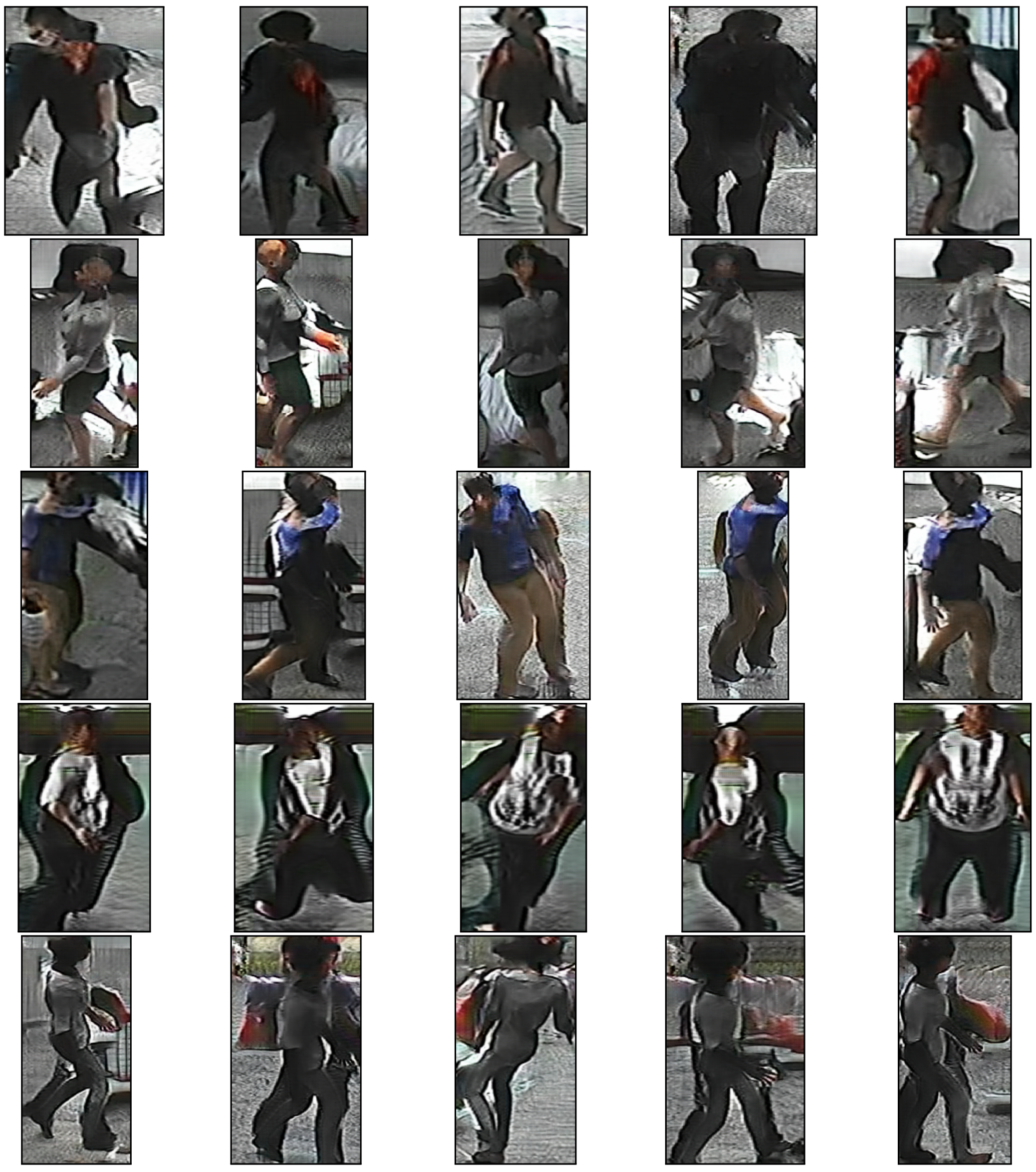


Figure B.6: Dataset R-II. Refined version of dataset II with the CycleGAN refiner.

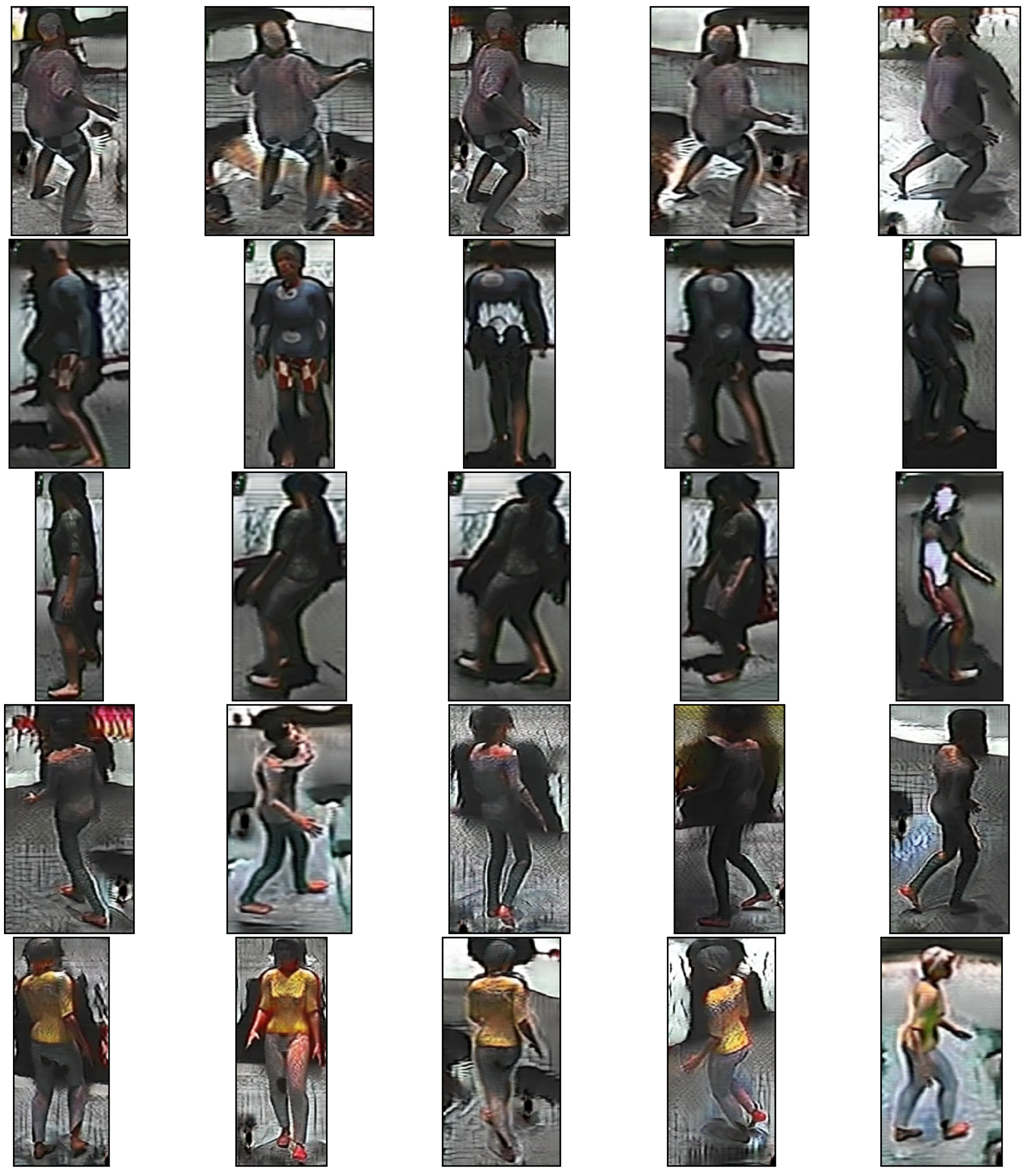


Figure B.7: Dataset R-III. Refined version of dataset III with the CycleGAN refiner.

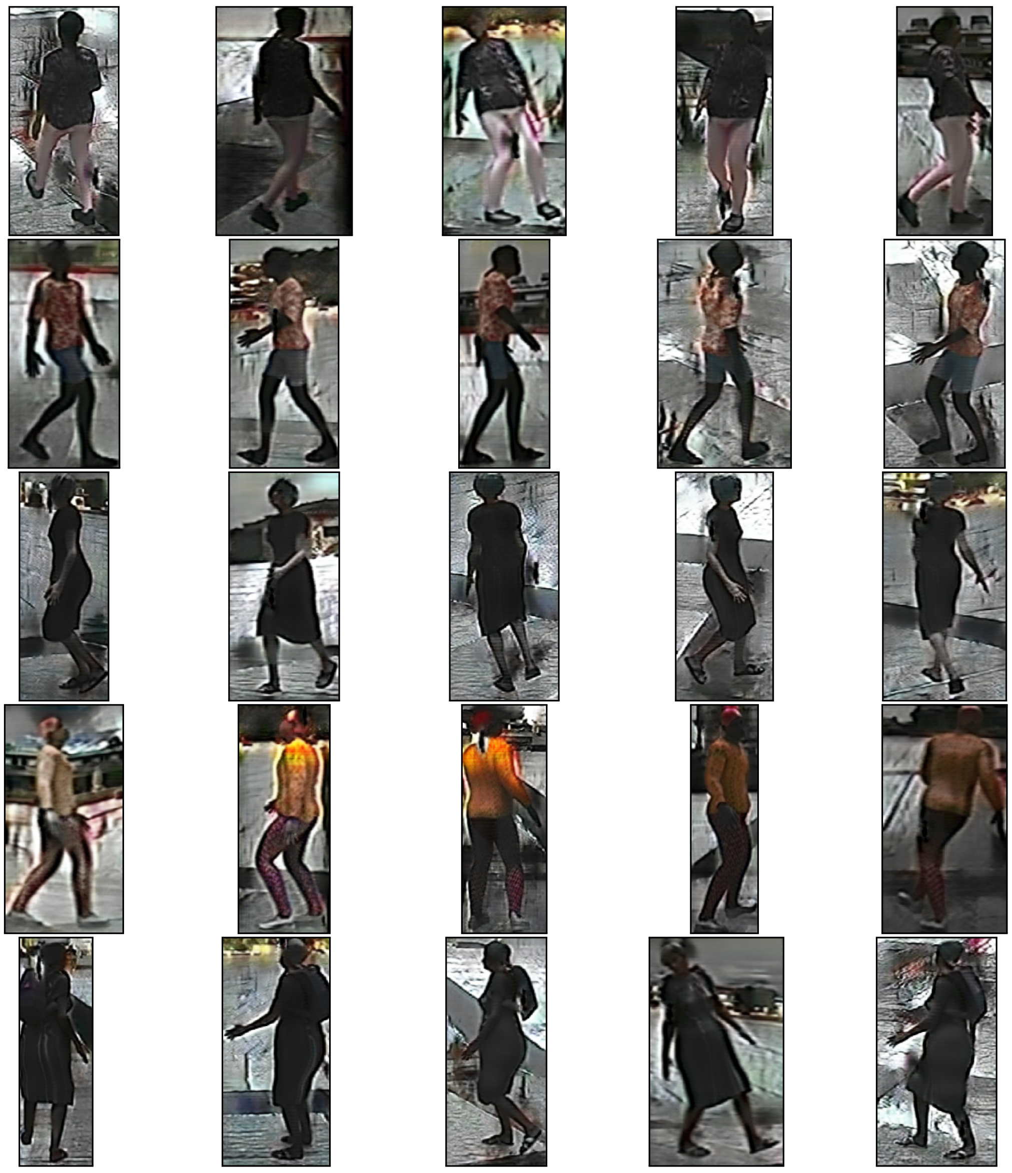


Figure B.8: Dataset R-IV (CycleGAN). Refined version of dataset IV with the CycleGAN refiner.

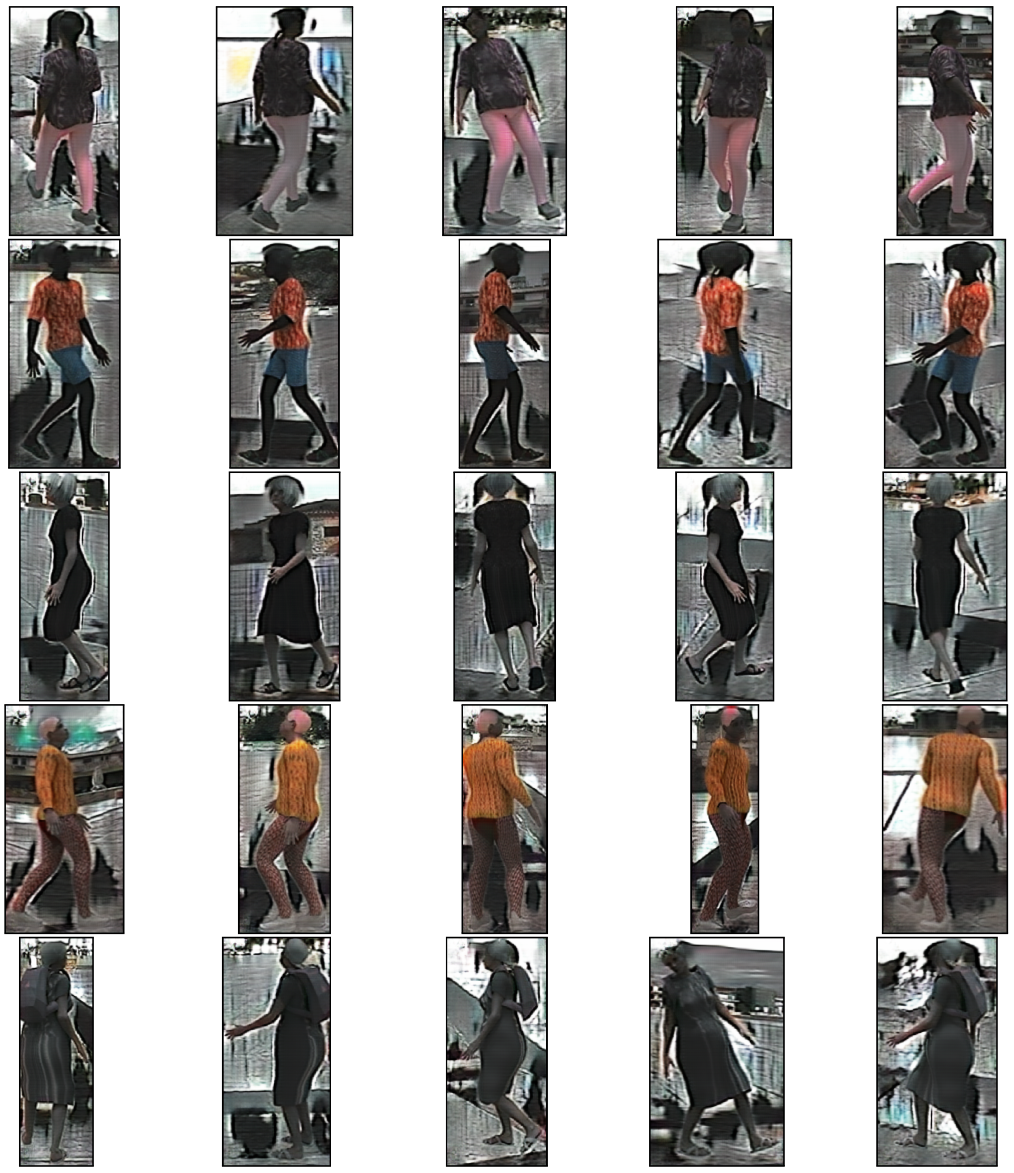


Figure B.9: Dataset R-IV (PTGAN). Refiner version of dataset IV with the PTGAN refiner.

Appendix C

Training Scores

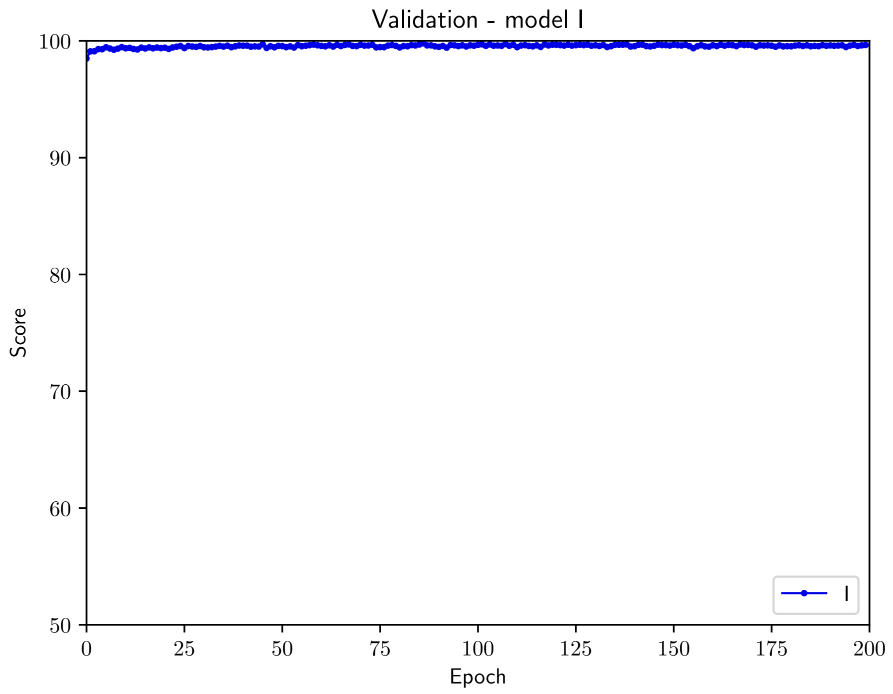


Figure C.1: Validation scores for model I.

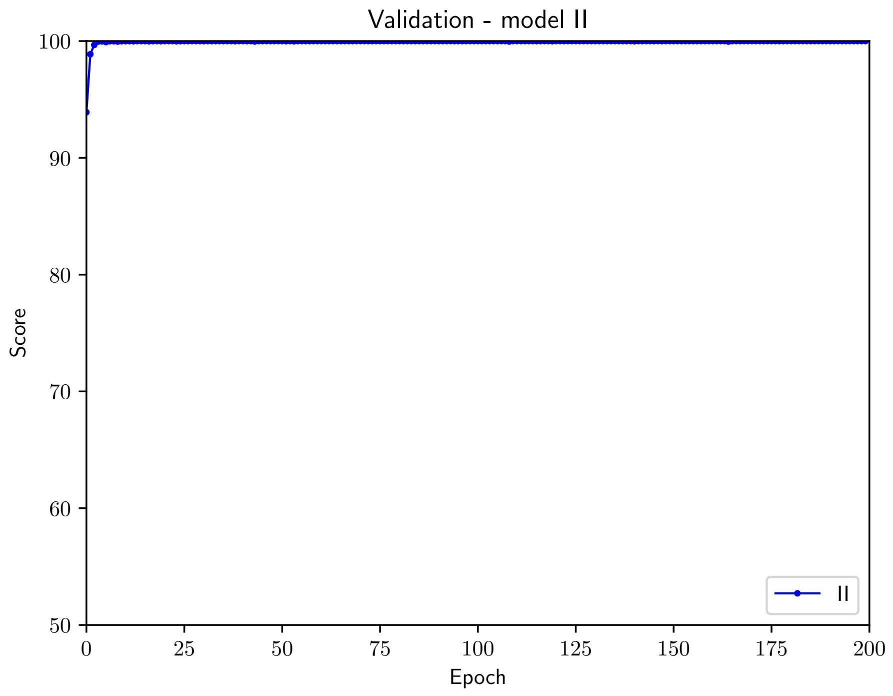


Figure C.2: Validation scores for model II.

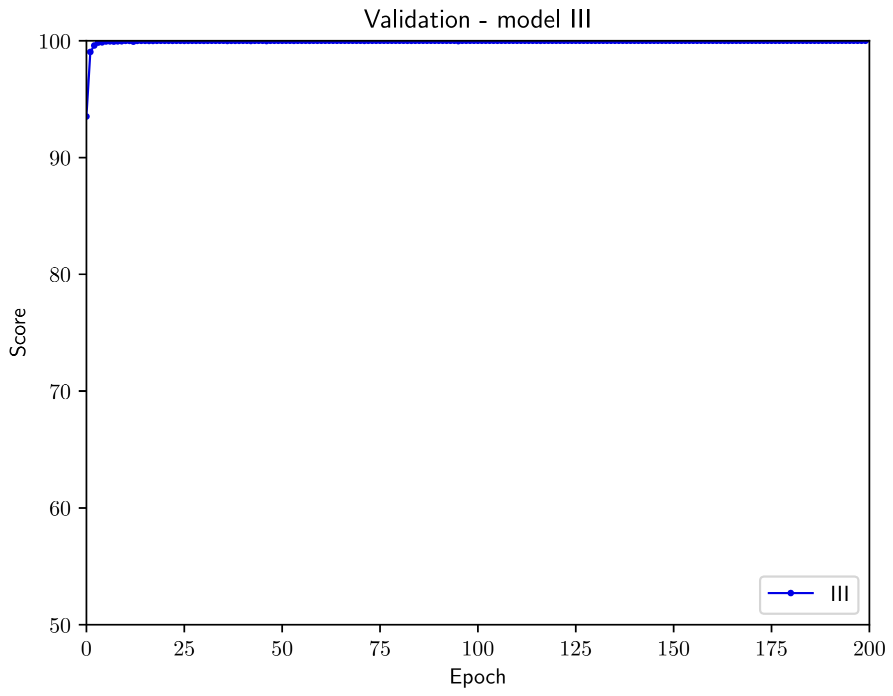


Figure C.3: Validation scores for model III.

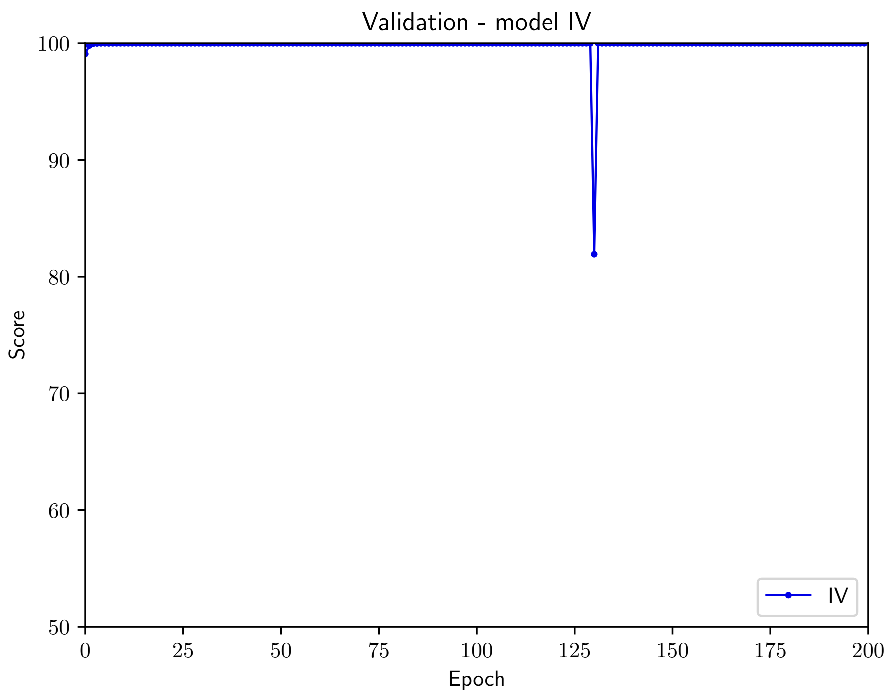


Figure C.4: Validation scores for model IV.

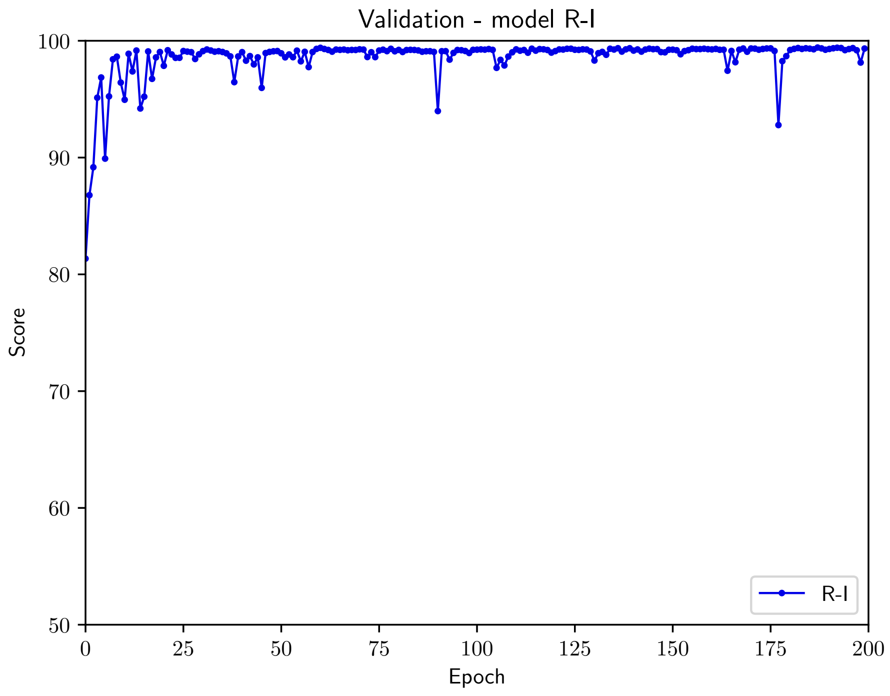


Figure C.5: Validation scores for model R-I.

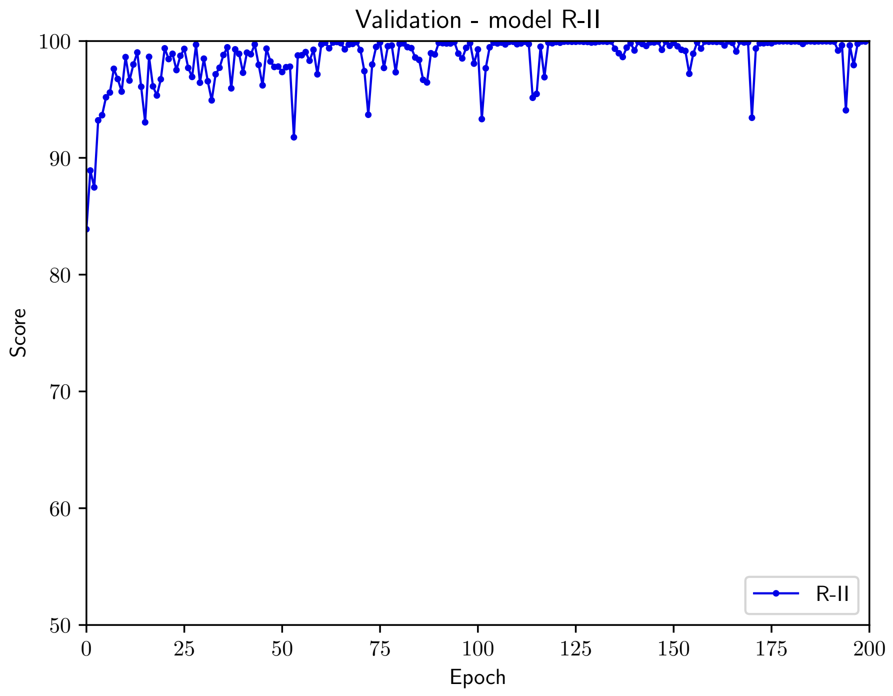


Figure C.6: Validation scores for model R-II.

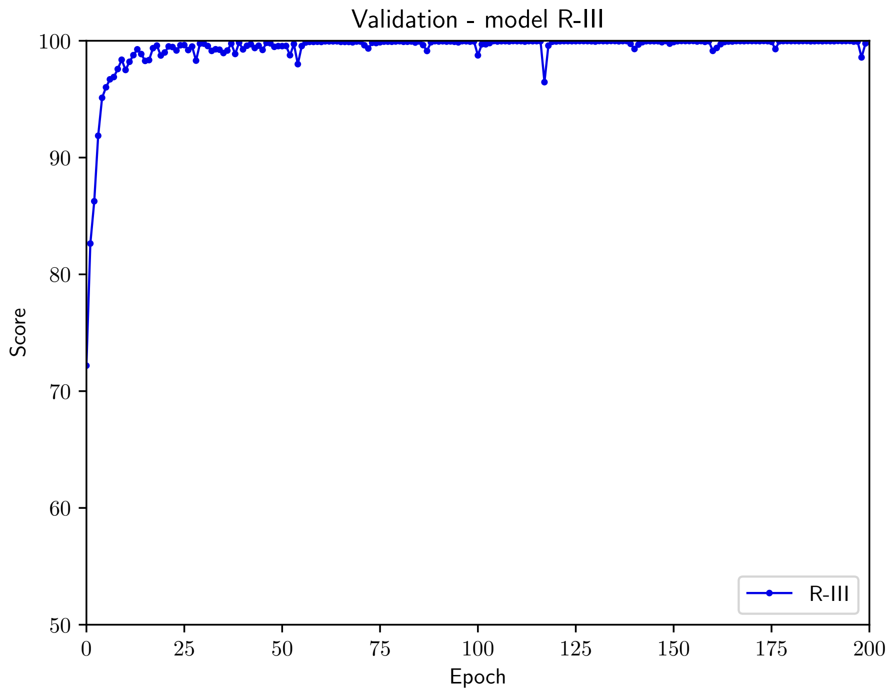


Figure C.7: Validation scores for model R-III.

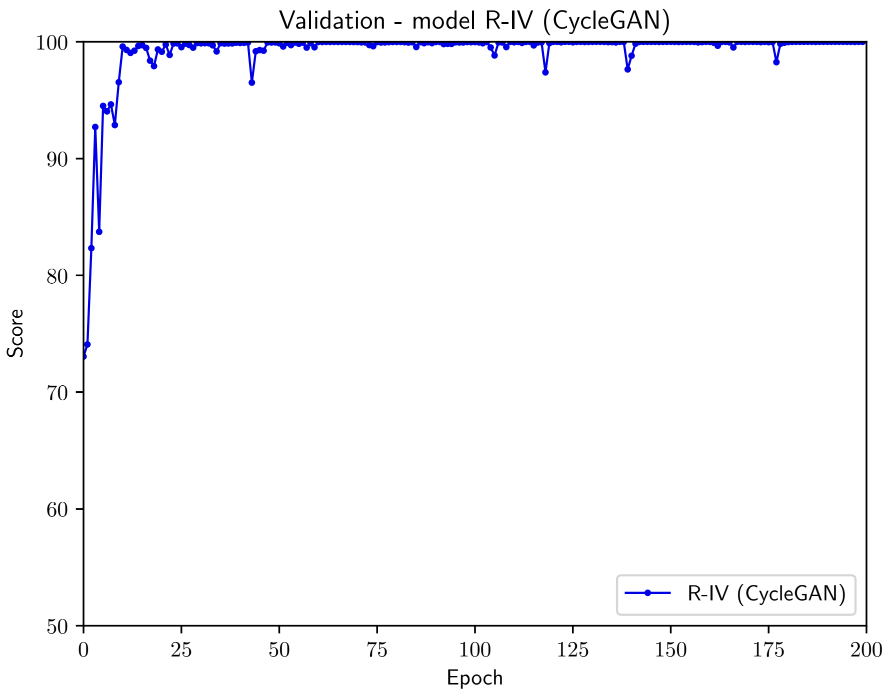


Figure C.8: Validation scores for model R-IV (CycleGAN).

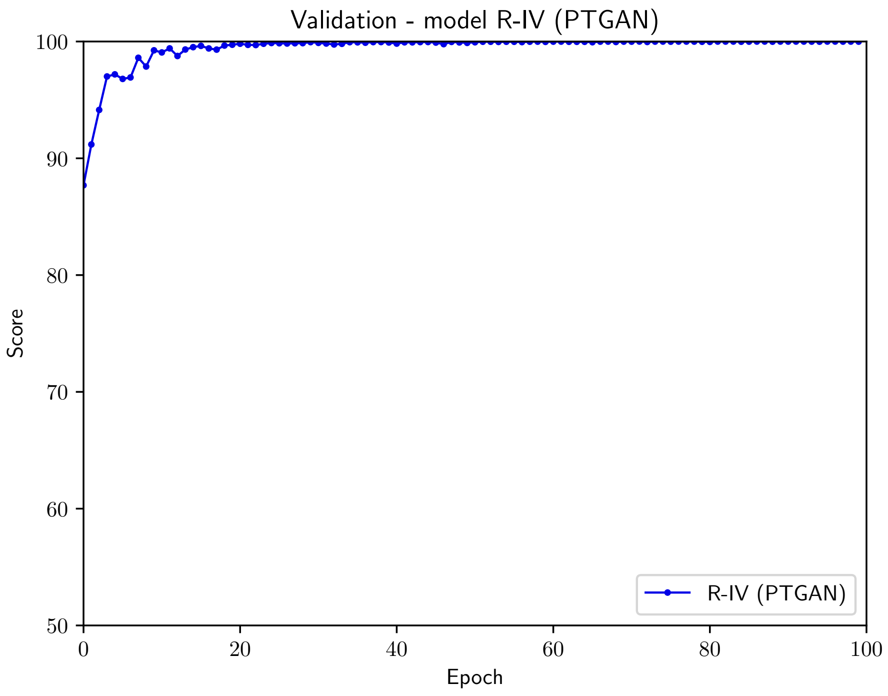


Figure C.9: Validation scores for model R-IV (PTGAN).

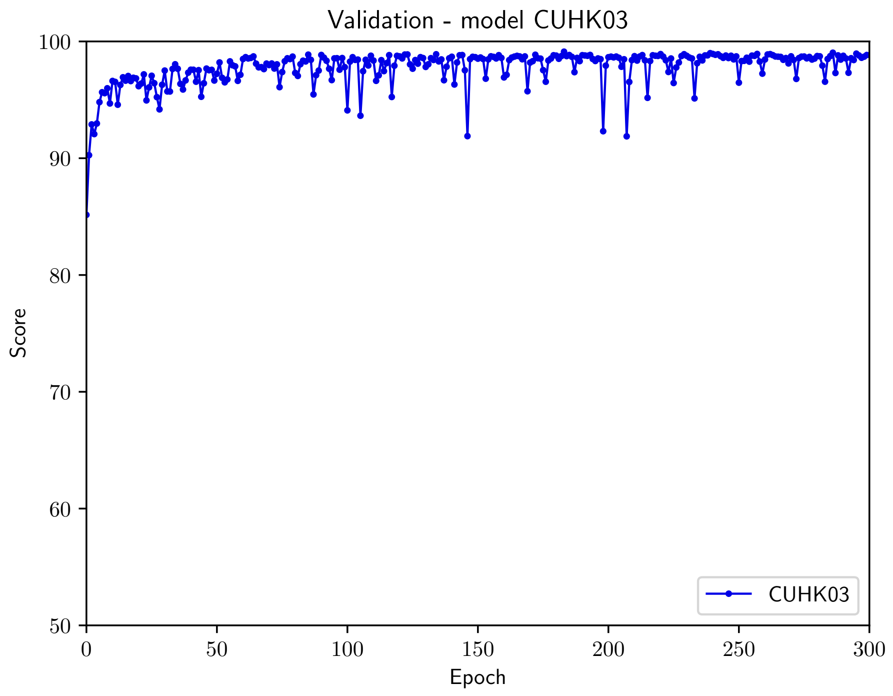


Figure C.10: Validation scores for model trained on CUHK03.

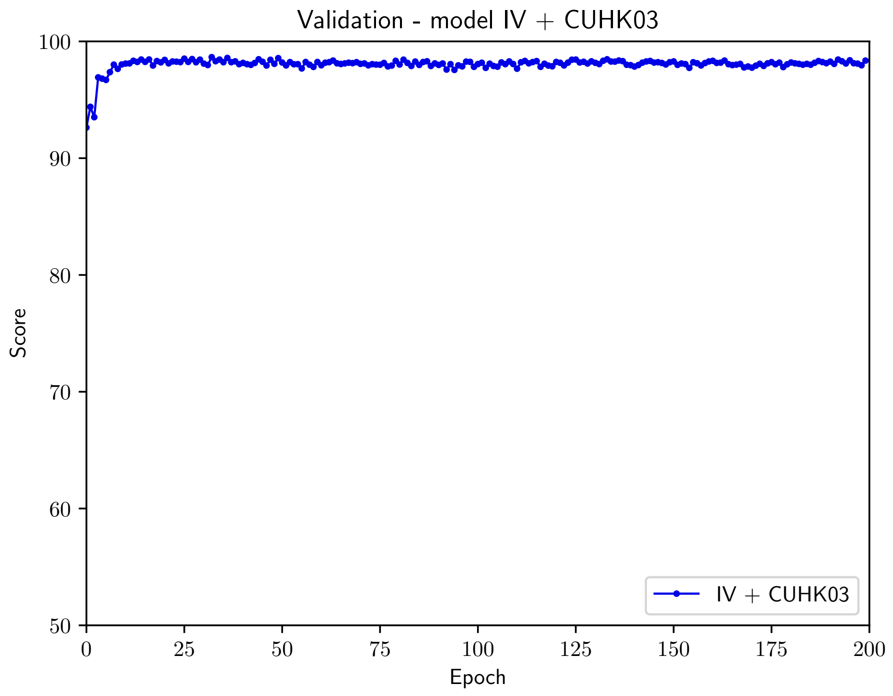


Figure C.11: Validation scores for finetuning on CUHK03, with initial training on dataset IV.

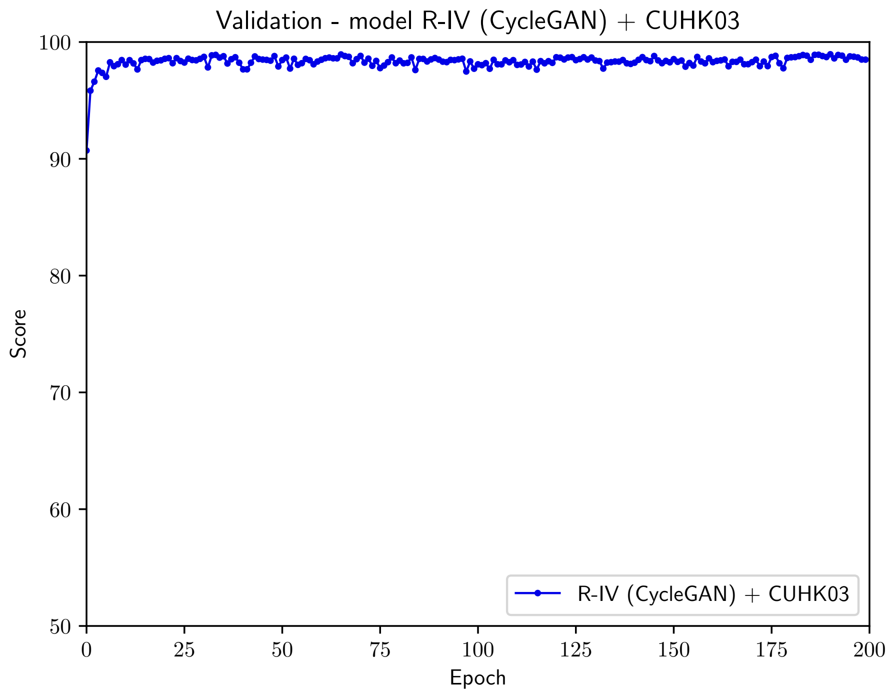


Figure C.12: Validation scores for finetuning on CUHK03, with initial training on dataset R-IV (CycleGAN).

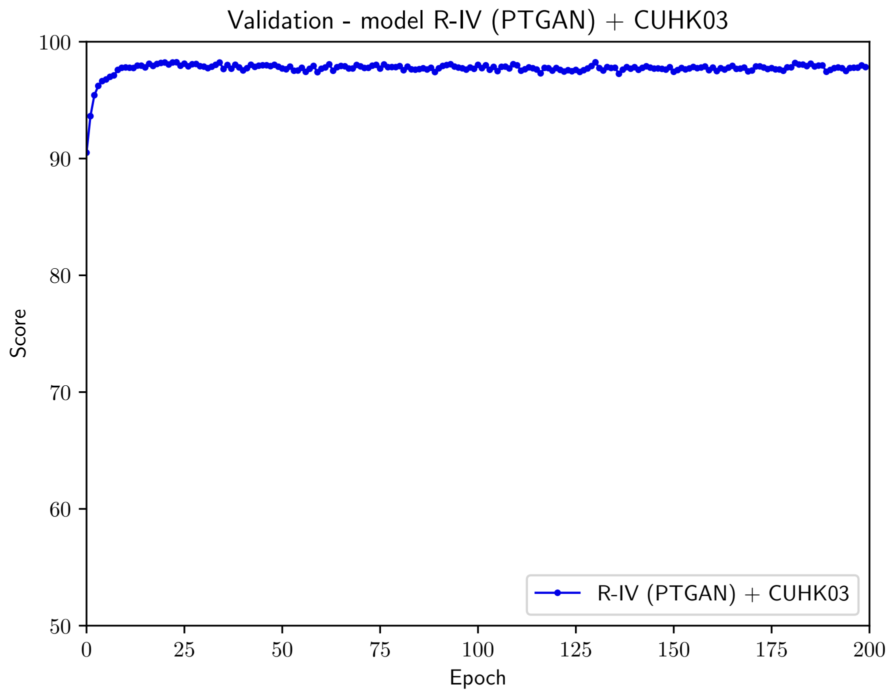


Figure C.13: Validation scores for finetuning on CUHK03, with initial training on dataset R-IV (PTGAN).

INSTITUTIONEN FÖR DATAVETENSKAP & MATEMATIKCENTRUM | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2018-08-23

EXAMENSARBETE Generation of Artificial Training Data for Deep Learning
STUDENTER Pontus Andersson, David Wessman
HANDLEDARE Michael Doggett (LTH), Kalle Åström (LTH)
EXAMINATOR Niels Christian Overgaard (LTH)

Konstgjorda träningsexempel för artificiell intelligens

POPULÄRVETENSKAPLIG SAMMANFATTNING **Pontus Andersson, David Wessman**

Kan bilder av konstgjorda människor ersätta bilder av riktiga människor som träningsexempel för artificiell intelligens? Vi undersöker denna fråga och skapar ett ramverk för att generera stora mängder konstgjorda träningsexempel.

Säg att vi vill träna upp en artificiell intelligens (AI) för att hitta katter i bilder. Det viktigaste som behövs är träningsexempel. Först behövs en bild på en katt och sen måste en manuellt markera var i bilden katten är. Helst behövs det hundratusentals bilder, några med och några utan katter. Hur löser vi detta?

Vårt förslag är att skapa konstgjorda träningsexempel! Istället för katter har vi tittat på träningsexempel för att kunna avgöra om det är samma människa som syns i två olika bilder. Bilderna genereras med datorgrafik, på samma sätt som i dator- och TV-spel.

När vi testar vår AI, efter att den tränats endast med genererade bilder, är resultatet inte lika bra som när vi tränar på riktiga bilder. Detta kan förklaras av skillnaderna som syns om man jämför bilder vi genererar (**A**) med bilder från riktiga övervakningskameror (**C**). Kan skillnaderna göras mindre? Vi försöker åstadkomma detta genom att sända alla våra egna bilder genom en processor vars uppdrag är att *förfina* våra bilder, dvs. att få dem att se ut som att de också var tagna med riktiga övervakningskameror och innehöll verkliga människor (**B**).

I våra resultat ser vi att konstgjorda bilder inte kan ersätta riktiga bilder som träningsexempel helt och hållet. Däremot upptäcker vi att förfinade bilder utgör bättre träningsexempel än de som inte förfinats.

Om konstgjorda bilder kunde ersätta riktiga, skulle det bli mycket enklare att skapa färdiga och uppmärkta träningsexempel, även för tillämpningar där det idag inte finns bilder eller träningsexempel att använda. Till skillnad från bilder av riktiga personer, är genererade bilder dessutom inte integritetskränkande.

För att kunna generera stora mängder träningsexempel lånar vi våra kollegors datorer på natten. Vi har byggt ett system där alla datorer genererar bilder som automatisk skickas till vår dator. Detta gör att vi kan generera hundratusen bilder varje natt.

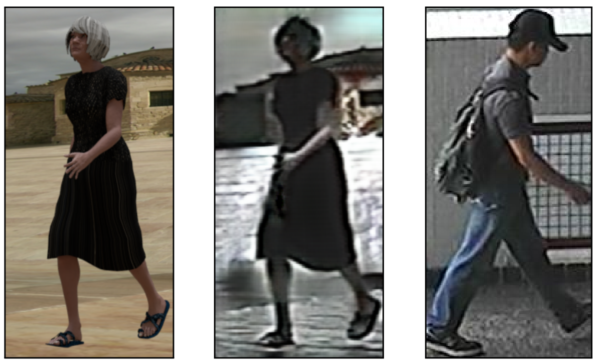


Bild (C) från bildsamlingen **CUHK03**.