RenderBEGAN: Adversarial Generative Domain Adaptation

Fabian Reimeier
f.reimeier@fu-berlin.de

Institute of Computer Science
Freie Universität Berlin

A thesis submitted for the degree of
Master of Science

Berlin, 2018

Supervised by
Prof. Dr. Tim Landgraf

Abstract

Supervised training methods require a lot of labeled training data which often does not exist in sufficient amounts as the data has to be annotated manually - a laborious task. This thesis describes a novel approach of unsupervised domain adaptation which turns labeled simulated data samples into labeled realistic data samples by using unlabeled real data samples for training. It is shown that the proposed model is able to generate realistic labeled images of human faces out of simulated face models generated from the Basel Face Model.

Contents

List of Figures

Chapter 1

Introduction

Solving problems with artificial intelligence (AI) is currently experiencing a hype and is more and more common in productive environments. Digital assistants like Siri and Alexa are integrated in a steadily growing set of products from smartphones, tablets and watches to glasses, speakers, remote controls, ovens and smart-home devices. They are offering an increasing amount of functionalities and are connected to more and more services. The automotive industry is pushing far into this area selling intelligent driving assistance[1] or even self-driving cars[2]. Augmented and virtual reality applications are moving into public focus. Manufacturers of mobile devices are beginning to build dedicated AI chips into their products[3]. AI has a growing number of use cases and there are a lot of models fitting well to one or another. But no matter what AI model you look at, all of them require training to become good at their tasks. Some of the models can be trained unsupervised or semi-supervised but for a lot of models it is necessary to provide a huge amount of labeled training data for supervised learning. Unfortunately, most of the times there is only unlabeled data available in huge amounts which then has to be labeled by hand in a very time consuming process (Sixt *et al.*, 2016). In some cases, there might not even be sufficient unlabeled data available to do that. Sometimes it is possible to simulate labeled training data but the simulated samples often differ much from real ones and can therefore not be used as a substitute. In this case, domain adaptation methods can help. This thesis is about a generative adversarial approach of domain adaptation which takes labeled simulated

[1] https://www.mercedes-benz.com/en/mercedes-benz/innovation/mercedes-benz-intelligent-drive/

[2] https://www.tesla.com/autopilot?redirect=no

[3] https://www.apple.com/apple-events/september-2017/

data samples as input and creates labeled realistic data samples out of them by using unlabeled real data samples for training. The considered use case for evaluating the model is the generation of realistic human faces from simulated Basel Face Model images which represents a relatively large domain shift.

Chapter 2

Related Work

At this point several approaches to solve the problem of not sufficiently available labeled data to train on are known. One way is just to increase the number of usable labeled data by augmenting existing labeled data in a way that the label information is preserved and adding the results to the training data set (Goodfellow *et al.*, 2016). Data augmentation is especially working very good with image data, e.g. by applying filters or adding noise, but requires existing labeled data and well designed augmentation functions which in some cases can be hard to find (Goodfellow *et al.*, 2016). Another approach is to pre-train *Deep Convolutional Neural Network* (DCNN) models on a large dataset and fine-tune its parameters afterwards with regard to the actual task (Girshick *et al.*, 2013; Shelhamer *et al.*, 2016). This approach can only be used if there exists a large enough dataset and labeled data for fine-tuning is available. Generative approaches which actually generate novel samples are mostly built around autoregressive models like *Pixel Recurrent Neural Networks* (PixelCNN) (Oord *et al.*, 2016), *Variational Autoencoders* (VAE) (Kingma & Welling, 2013) and *Generative Adversarial Networks* (GAN).

The GAN framework was introduced by Goodfellow *et al.*, 2014. The basic idea can be seen as a minimax game of two competing players, a generator G and a discriminator D. The generator G produces samples of a specific kind (e.g. images of faces) out of random noise from the latent space. The discriminator D tries to distinguish generated from real samples. The goal of G is to generate samples which D perceives as real and the goal of D is to not get tricked by G. By using the GAN framework, it is possible to generate natural images of such high quality that even humans find it hard to distinguish them from real ones (Denton *et al.*, 2015;

Karras *et al.*, 2017). The GAN framework alone is not capable of generating labeled data because there is no simple relationship between the latent space and dimensions the GAN is learning in and the labels of interest (Sixt *et al.*, 2016). It is therefore impossible to trace generated samples back to the high-level information used to create them. *Conditional Generative Adversarial Networks* (CGANs) (Gauthier, 2014) are capable of generating samples conditioned on class labels by feeding a conditonal data vector to both the generator and discriminator but require labeled training data. The generation of samples belonging to specified categories can be learned unsupervised by *Categorical Generative Adversarial Networks* (CatGANs) (Springenberg, 2015), but the current restriction to categorical labels makes them unusable in several domains. RenderGAN (Sixt *et al.*, 2016) combines a 3D model with the GAN framework and enables the cheap generation of high-quality labeled data without requiring labeled data for training. The generator augments a basic 3D model based on a sequence of augmentation functions with trainable parameters in a way that it looks more realistic. Even though this framework does not need labeled data for training it requires the careful definition and customization of augmentation functions and the existance of a suitable 3D model (Sixt *et al.*, 2016).

Domain adaptation methods try to learn a model from a source domain that can be applied to a different but related target domain afterwards. There has been a lot of research in this area but the latest research is focused on transferring *Deep Neural Network* (DNN) models unsupervised from a source domain with available labeled data to a target domain where labeled data is sparse or non-existent (Tzeng *et al.*, 2017). Most approaches for unsupervised domain adaptation focus on training a classifier on the source domain which is also applicable to the target domain. The main strategy is to train the classifier to extract features from samples of both domains while minimizing the difference between their distributions (Ganin & Lempitsky, 2014; Tzeng *et al.*, 2015, 2014; Long *et al.*, 2015; Liu & Tuzel, 2016; Ghifary *et al.*, 2016; Tzeng *et al.*, 2017; Bousmalis *et al.*, 2016a). Several approaches trying to find domain-invariant features are using the *Maximum Mean Discrepancy* (MMD) loss (Gretton *et al.*, 2009) to minimize the difference between feature distributions of source and target domain. Other approaches are using adversarial losses and discriminative models consisting of a source label classifier in combination with a binary domain classifier (Ganin & Lempitsky, 2014; Tzeng *et al.*, 2017, 2015). The goal is then to find representations that the label classifier can discriminate but the

domain classifier can not.

An example for generative approaches of domain adaptation are *Coupled Generative Adversarial Networks* (CoGANs) (Liu & Tuzel, 2016). They use two jointly trained GANs with tied high-level layer parameters which generate source and target images respectiveley and are capable of approximating a joint distribution of source and target domain. However, CoGANs are only applicable on simple domains as they depend on the ability of the GANs to find a mapping of shared high-level layer feature spaces to full images in both domains (Tzeng *et al.*, 2017). Another generative approach described in Bousmalis *et al.*, 2016b called *Pixel-Level Domain Adaptation* (PixelDA) uses a GAN-based model stabilized by a task specific loss and a content similarity loss that learns a transformation in pixel space between domains. It adapts images from a source domain in a way that they appear to belong to the target domain and decouples the domain adaptation process from task-specifc architecture. This approach works very good in case of small domain shifts. In Tzeng *et al.*, 2017 a general adversarial adaptation framework subsuming previous generative and discriminative approaches is described and an example instance of it is given in form of the *Adversarial Discriminative Domain Adaptation* (ADDA). This approach combines discriminative modeling, united weight sharing and a GAN loss but does not treat any sample generating tasks and is only described for categorical label classification. Unsupervised domain adaptation is still an open theoretical and practical problem.

Chapter 3

RenderBEGAN

The RenderBEGAN approach tackles the problem of establishing and explicitly controlling a relationship between latent space and generated samples without using labeled data or predefined and constraining augmentation functions. It combines a 3-D model as described in RenderGAN (Sixt *et al.*, 2016) with BEGAN (Berthelot *et al.*, 2017; Karras *et al.*, 2017) and unsupervised domain adaptation approaches (Ganin & Lempitsky, 2014; Bousmalis *et al.*, 2016b,a).

3.1 Background

3.1.1 BEGAN

The *Boundary Equilibrium Generative Adversarial Net* (BEGAN) framework (Berthelot *et al.*, 2017) is an extension of the GAN framework (Goodfellow *et al.*, 2014). It is based on the autoencoder instantiation of *Energy-Based GANs* (EBGANs) (Zhao *et al.*, 2016) and extends it by an equilibrium concept for generator and discriminator.

GAN

The generator of the original GAN framework (Goodfellow *et al.*, 2014) maps random noise vectors $z \in \mathbb{R}^{N_z} \sim \mathbb{P}_z$ to data space by a neural network $G(z;\theta_G) : \mathbb{R}^{N_z} \mapsto \mathbb{R}^{N_x}$ with parameters θ_G. The generator learns to match its distribution $\mathbb{P}_{fake}$ to the real data distribution $\mathbb{P}_{real}$. The generator is trained with the help of a discriminator. The discriminator is another neural network $D(x;\theta_D) : \mathbb{R}^{N_x} \mapsto [0,1]$ with parameters θ_D whose single scalar output represents the propability that x is a real sample from

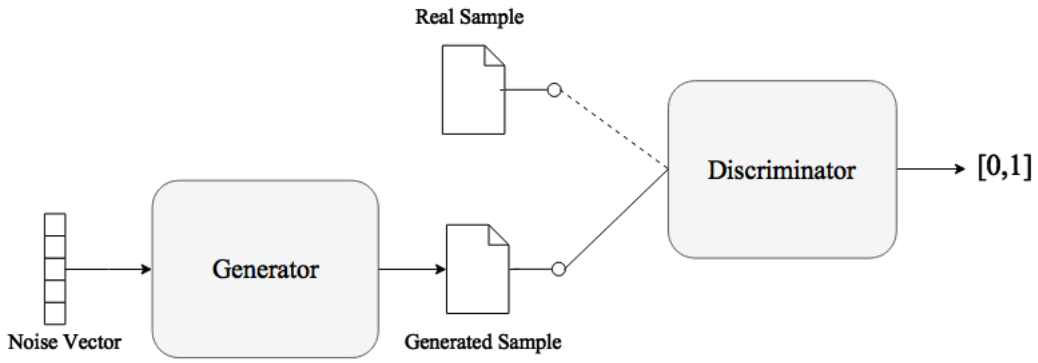

Figure 3.1: Topology of a GAN - Generator and discriminator are playing a minimax two-player game. The generator tries to generate samples that the discriminator perceives as real. The discriminator tries to not get tricked by the generator and distinguishes between generated and real samples.

data generating distribution $\mathbb{P}_{real}$ rather than model distribution $\mathbb{P}_{fake}$. The topology of a GAN can be seen in figure 3.1. The played minimax game can be described as follows, where $V(G, D)$ is the value function:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim \mathbb{P}_{real}}[logD(x; \theta_D)] + \mathbb{E}_{z \sim \mathbb{P}_z}[log(1 - D(G(z; \theta_G); \theta_D))] \quad (3.1)$$

The global optimum of this game is reached when there is a Nash Equilibrium between generator and discriminator yielding $\mathbb{P}_{fake} = \mathbb{P}_{real}$. Generator and discriminator are trained jointly through backpropagation. The loss functions for discriminator and generator for each update step can be written as:

$$\begin{cases} \mathcal{L}_D = log(D(x; \theta_D)) + log(1 - D(G(z_D; \theta_G); \theta_D)) & \text{for } \theta_D \\ \mathcal{L}_G = log(1 - D(G(z_G; \theta_G); \theta_D)) & \text{for } \theta_G \end{cases} \quad (3.2)$$

where $\mathcal{L}_D$ has to be maximized, $\mathcal{L}_G$ has to be minimized and $z_G, z_D \sim \mathbb{P}_z$.

EBGAN

The EBGAN framework (Zhao *et al.*, 2016) introduced the view of the discriminator as an energy function, which assigns high energy to generated samples and low energies to real samples. The generator is trained to generate samples with low energy. One instantiation of the framework uses an autoencoder as discriminator.

An autoencoder is a neural network that tries to approximate the identity function

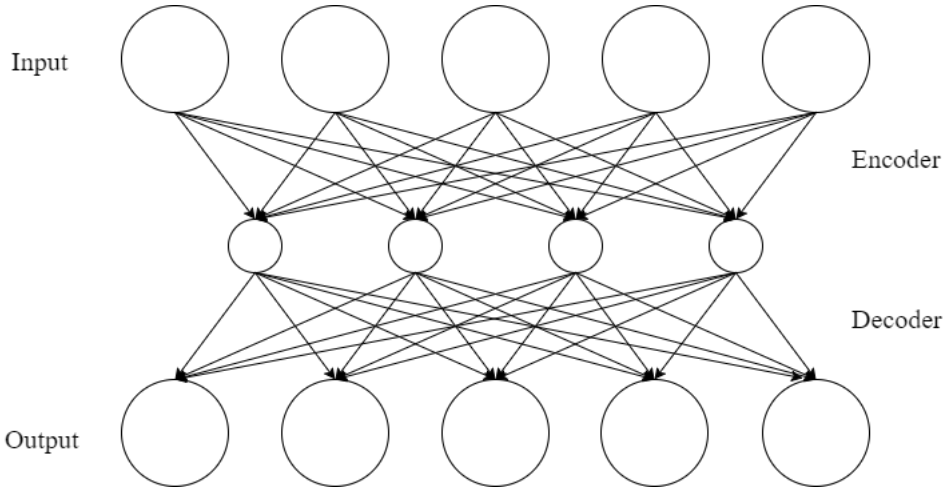

Figure 3.2: Simple Autoencoder - The autoencoder consists of one input- and one output-layer connected through one hidden layer. The encoder maps the input to its latent representation and the decoder reconstructs the original input out of it. The autoencoder tries to approximate the identity function of its input.

$H(x;\theta_H) \approx x$ for samples of the data distribution $\mathbb{P}_{real}$ through unsupervised learning (D'Avino *et al.*, 2017). The simplest version is shown in figure 3.2 and consists out of one input layer connected to one output layer through a single hidden layer. An encoder function $Enc(x;\theta_{Enc})$ transforms the input into its latent representation h and a decoder function $Dec(h;\theta_{Dec})$ reconstructs the original data out of it so that $Dec(Enc(x;\theta_{Enc});\theta_{Dec}) = x$. At first glance this problem seems trivial, easy to solve and not to provide any meaningful information, but by adding constraints like a limit on the amount of hidden units interesting structures of the data can be discovered. By limiting the hidden units of an autoencoder to a value lower than its input dimension, the searched for encoder function $Enc(x;\theta_{Enc})$ equals a compression of the input. The input data will be reduced to an embedding h of its most important features which are necessary for $Dec(h;\theta_{Dec})$ to reconstruct the original input (D'Avino *et al.*, 2017). This task is very complex for completely random inputs but if the input is structured (e.g. by correlated input features), the autoencoder will be able to discover it.

As it can be expected that samples from $\mathbb{P}_{real}$ are structured and contain correlated features, an autoencoder can function as discriminator by using the reconstruction error as the energy value for samples from $\mathbb{P}_{fake}$ and $\mathbb{P}_{real}$ (Zhao *et al.*, 2016). The autoencoder is trained on real data which will lead to a small reconstruction error for samples $x_{real} \sim \mathbb{P}_{real}$ and a greater reconstruction error for samples $x_{fake} \sim \mathbb{P}_{fake}$. The better $\mathbb{P}_{fake}$ approximates $\mathbb{P}_{real}$, the less difference will remain between x_{fake}

and its reconstruction $\hat{x}_{fake}$. BEGANs are aiming to optimize a lower bound of the Wasserstein distance between autoencoder loss distributions of samples created by generator G and real samples by minimizing the two losses (Berthelot *et al.*, 2017):

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - \mathcal{L}(G(z_D;\theta_G)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(\mathrm{G}(z_G;\theta_G)) & \text{for } \theta_G \end{cases} \tag{3.3}$$

$\mathcal{L} : \mathbb{R}^{N_x} \mapsto \mathbb{R}^+$ is the autoencoder loss of a pixel-wise autoencoder which is defined as

$$\mathcal{L}(v) = \|v - D(v;\theta_D)\|_\eta \text{ with } \begin{cases} D : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_x} & \text{autoencoder function} \\ \eta \in \{1,2\} & \text{target norm} \\ v \in \mathbb{R}^{N_x} & \text{sample of dimension } N_x \end{cases} \tag{3.4}$$

Equilibrium Concept

Additionally, the BEGAN framework introduced an equilibrium concept to maintain a balance between $\mathcal{L}_D$ and $\mathcal{L}_G$, which is not given most of the times because its easier for the discriminator to win the game. Berthelot *et al.*, 2017 define the equlibrium as

$$\mathbb{E}\left[\mathcal{L}(x)\right] = \mathbb{E}\left[\mathcal{L}(G(z;\theta_G))\right] \tag{3.5}$$

and introduced a hyper-parameter $\gamma \in [0,1] = \frac{\mathbb{E}[\mathcal{L}(G(z;\theta_G))]}{\mathbb{E}[\mathcal{L}(x)]}$ to relax this equilibrium. Lower values for γ are focusing the discriminator on autoencoding real samples rather than discriminating real from generated ones, resulting in lower sample diversity. The final BEGAN objective using this equilibrium concept in connection with proportional control theory is defined as

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - k_t \cdot \mathcal{L}(G(z_D;\theta_G)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(G(z_G;\theta_G)) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k \cdot (\gamma \cdot \mathcal{L}(x) - \mathcal{L}(G(z_G;\theta_G))) & \text{for each training step } t \end{cases} \tag{3.6}$$

where $k_t \in [0,1]$ controls the emphasis on $\mathcal{L}(G(z_D;\theta_G);\theta_D)$ during training and is updated in every training step with learning rate (or proportional gain) λ_k to maintain the equilibrium from equation 3.5 relaxed by γ. The equilibrium concept is also used to define a global measure of convergence M_{global} for BEGANs, which can be used to

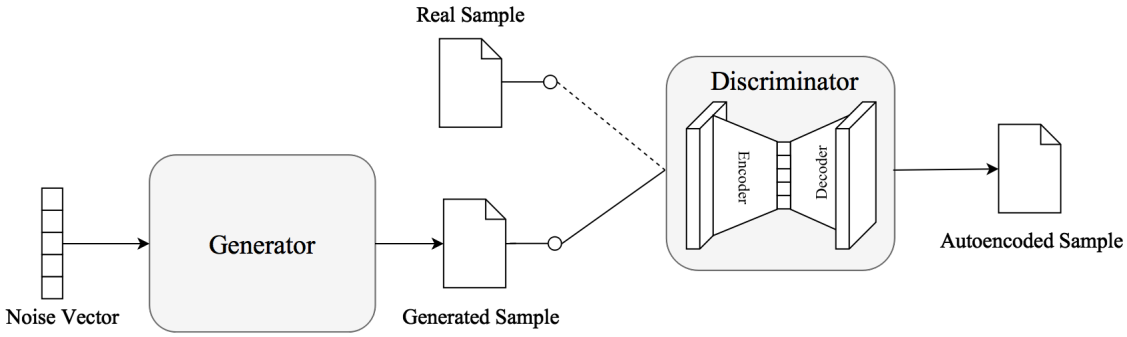

Figure 3.3: Topology of a BEGAN - The discriminator implemented as autoencoder distinguishes between real and generated samples by the reconstruction loss. The discriminator tries to keep the loss high for generated ones and low for real ones. The generator aims to generate samples that lead to small reconstruction losses of the autoencoder.

determine when the training is finished or the model has collapsed.

$$M_{global} = \mathcal{L}(x) + |\gamma \cdot \mathcal{L}(x) - \mathcal{L}(G(z_G; \theta_G))| \tag{3.7}$$

The topology of the described BEGAN model is shown in figure 3.3.

3.1.2 Unsupervised Domain Adaptation

An example for unsupervised domain adaptation is given in Ganin & Lempitsky, 2014. The proposed model is trained on a large amount of labeled simulated data samples from source domain $\mathcal{S}$ and unlabeled realistic data samples from target domain $\mathcal{T}$. It learns features invariant over both domains $\mathcal{S}$ and $\mathcal{T}$ which can still be used for classification. The architecture is shown in figure 3.4. A simulator $\mathcal{S}$ creates simulated samples out of label vectors. Real samples from $\mathcal{T}$ and simulated samples from $\mathcal{S}$ are mapped by a feature extractor $F(x; \theta_F)$ to a feature vector f which is then used by a label classifier $P(f; \theta_P)$ to predict the class label l and by a domain classifier $D(f; \theta_D)$ to determine the domain d. The aim of the training process is to find parameters θ_F and θ_P that maximize the domain classification loss $\mathcal{L}_{domain}$ while minimizing the label prediction loss $\mathcal{L}_{label}$ and parameters θ_D that minimize the domain classification loss $\mathcal{L}_{domain}$. F will then ideally create domain invariant features f which P will be able to discriminate while D will not be able to. This leads to the following losses to

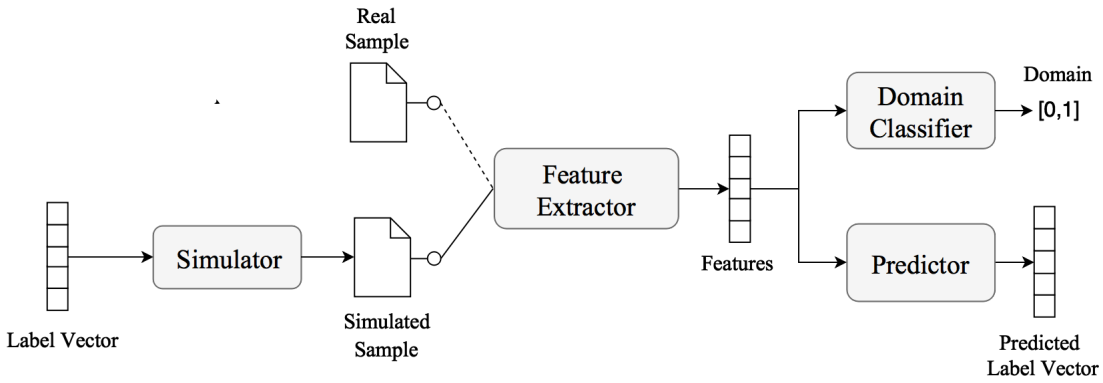

Figure 3.4: Unsupervised Domain Adaptation - Architecture proposed in (Ganin & Lempitsky, 2014). Simulated and real samples are mapped to a feature vector which is used by the predictor to reproduce the label vector and by the domain classifier to deteremine the domain of the input sample.

be minimized in each update step:

$$\begin{cases} \mathcal{L}_{FP} = \mathcal{L}_{label}(P(F(S(l);\theta_F);\theta_P), l) - \lambda \cdot \mathcal{L}_{domain}(D(F(x;\theta_F);\theta_D), d) & \text{for } \theta_F, \theta_P \\ \mathcal{L}_D = \mathcal{L}_{domain}(D(F(x;\theta_F);\theta_D), d) & \text{for } \theta_D \end{cases} \tag{3.8}$$

Ganin & Lempitsky, 2014 showed that standard stochastic gradient descent solvers (SGD) are able to find parameters θ_F, θ_P and θ_D to achieve that. A trained model will be able to classify samples from $\mathcal{T}$ related to classes of $\mathcal{S}$.

3.2 Model

The basic consideration of the RenderBEGAN approach is that label information contained in an image sample is of such importance that the autoencoder of a BEGAN architecture has to encode it into its embedding to be able to successfully reconstruct the original. Therefore it should be possible to reproduce the label information contained in an image given the belonging embedding generated by the autoencoder. If the generator is forced by the learning process to include the label information in its generated images, it should have to encode the label information in a realistic way leading to realistic generated images containing all the label information. The generated images could then also serve as labeled training dataset for a predictor model which reproduces label information out of embeddings created by the discriminator.

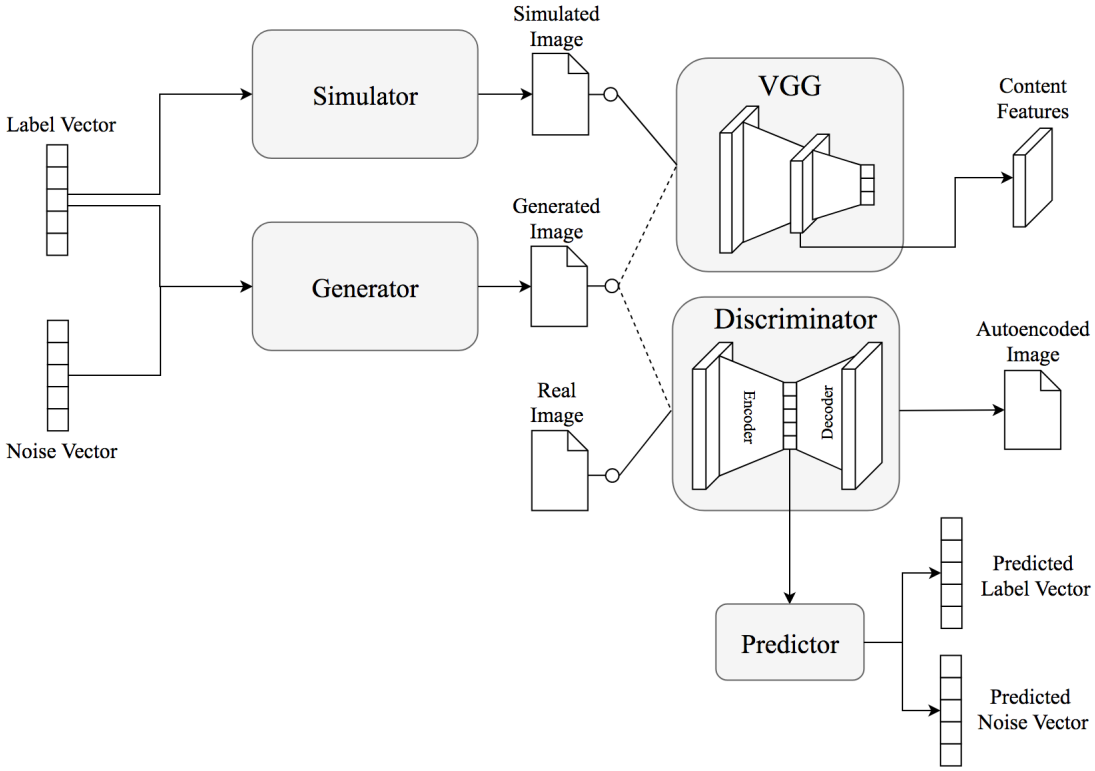

Figure 3.5: Topology of a RenderBEGAN - The generator recieves a label and a noise vector as input and generates an image out of it. The simulator creates a simulated image out of the same label vector. Generated and simulated image are processed by a pretrained VGG model to compute a perceptual loss between them indicating the similarity of their content. The discriminator works as an autoencoder. The predictor reproduces label and noise vector out of the autoencoder embedding.

As the generated images should get more realistic while training, the predictor will also get better and better in classifying actual real images. The proposed architecture of the RenderBEGAN model using these considerations is shown in figure 3.5. The generator $G(z, l; \theta_G) : \mathbb{R}^{N_z+N_l} \mapsto \mathbb{R}^{N_x}$ gets a noise vector $z \sim \mathbb{P}_z$ and a label $l \sim \mathbb{P}_l$ as input and generates an image $x_G \sim \mathbb{P}_{fake}$ out of it. A pretrained network $S(l; \theta_S) : \mathbb{R}^{N_l} \mapsto \mathbb{R}^{N_x}$ called simulator creates a simulated labeled image $x_S \sim \mathbb{P}_s$ out of l and thus defines the semantic of the label. To ensure that the generator interprets and encodes the label vector l using the same semantic, x_S and x_G are both processed by a pretrained VGG model (Simonyan & Zisserman, 2014) $V(x, \theta_V) : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_v}$. Thereby computed high-level features f_{x_G} and f_{x_S} are extracted from V to compute a perceptual loss (Johnson *et al.*, 2016) between x_G and x_S which is, inspired by Bousmalis *et al.*, 2016b, used as content-similarity loss for G to ensure that the label

information is contained and encoded correctly in generated images while giving the freedom to change other details. As the high-level features are taken from a relatively deep layer of the VGG, the content-similarity loss will hopefully reflect matchings of important characteristics of the images used for classification rather than their exact appearance. The last, discriminative part of the architecture is a modified version of the model described in section 3.1.2. Real samples $x \sim \mathbb{P}_{real}$ and generated samples x_G are encoded into an embedding h by the encoder function $Enc(x; \theta_{Enc}) : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_h}$ with $\theta_{Enc} \subset \theta_D$. The predictor $P(h; \theta_P) : \mathbb{R}^{N_h} \mapsto \mathbb{R}^{N_z+N_l}$ predicts given embedding h the original noise vector z to reduce mode collapse as described in Srivastava *et al.*, 2017 and the original label vector l. The decoder function $Dec(x; \theta_{Dec}) : \mathbb{R}^{N_h} \mapsto \mathbb{R}^{N_x}$ with $\theta_{Dec} \subset \theta_D$ reconstructs the original sample x or x_G out of h. Encoder and decoder function are building the discriminator $D(x; \theta_D) : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_x}$ which determines the domain of an input image through reconstruction loss. The training objective of the RenderBEGAN model is the minimization of the following three losses $\mathcal{L}_D$, $\mathcal{L}_P$ and $\mathcal{L}_G$:

$$\begin{cases} \mathcal{L}_D = \mathcal{L}_d(x) - k_t \cdot \mathcal{L}_d(G(z_D, l_D; \theta_G)) & \text{for } \theta_D \\ \mathcal{L}_P = \mathcal{L}_z(z_P, l_P) + \mathcal{L}_l(z_P, l_P) & \text{for } \theta_P \\ \mathcal{L}_G = \mathcal{L}_d(G(z_G, l_G; \theta_G)) + w_c \cdot \mathcal{L}_c(z_G, l_G) + w_z \cdot \mathcal{L}_z(z_G, l_G) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k \cdot (\gamma \cdot \mathcal{L}_d(x) - \mathcal{L}_d(G(z_G, l_G; \theta_G))) & \text{for each training step } t \end{cases} \tag{3.9}$$

The discriminator loss $\mathcal{L}_D$ is the same as the original discriminator loss shown in equation 3.6. $\mathcal{L}_d$ is the autoencoder loss defined in equation 3.4. The newly added predictor loss $\mathcal{L}_P$ used to update the parameters θ_P is a combination out of noise loss $\mathcal{L}_z(z, l) = \|z - \tilde{z}\|_2$ between the original noise vector z and its predicted counterpart $\tilde{z}$ and label loss $\mathcal{L}_l = \left\|l - \tilde{l}\right\|_2$ between the original and predicted label vectors l and $\tilde{l}$. Both predicted vectors $\tilde{z}$ and $\tilde{l}$ are results of $P(Enc(G(z, l; \theta_G); \theta_D); \theta_P)$. Obviously, the predictor will be trained to minimize the difference between an original vector and its prediction. The generator loss $\mathcal{L}_G$ is the original generator loss from equation 3.6 extended by content-similarity loss $\mathcal{L}_c(z, l) = \left\|f_{G(z,l;\theta_G)} - f_{S(l;\theta_S)}\right\|_2$, with $f_{G(z,l;\theta_G)}$ and $f_{S(l;\theta_S)}$ being the high level features extracted from V when processing $G(z, l; \theta_G)$ respectively $S(l; \theta_S)$ and the noise loss $\mathcal{L}_z$. The content-similarity loss will be minimized to ensure that label information contained in a label vector is correctly encoded into the generated image. The noise loss $\mathcal{L}_z$ will be minimized to ensure

that the noise vector z is also encoded into the image and can be reproduced by the predictor. As it is only important that z is encoded in a realistic way and not exactly how, a content-similarity loss for z is not necessary. The weights $w_c, w_z \in [0,1]$ are used to balance content-similarity loss $\mathcal{L}_c$ and noise loss $\mathcal{L}_z$ with regards to the autoencoder loss $\mathcal{L}_d$. Higher values of w_c will probably lead to better matchings of simulated face models and generated faces. Higher values of w_z should lead to higher image diversity.

3.3 Implementation

The use case for the RenderBEGAN model considered in this thesis is the generation of realistic labeled 32 x 32 images of human faces. The model was implemented in Python using Pytorch[1].

3.3.1 Simulator

The task of simulator S is to create a simulated labeled image of a face out of a label vector l to help the generator G correctly encode l in his generated sample. In this case, S creates a face model image from a 400-dimensional label vector $l \sim \mathcal{N}(0,1)$ containing the coefficients for a linear combination of principal components of the Basel Face Model (Paysan *et al.*, 2009). The vector contains 199 coefficients influencing the shape of the face, 199 coefficients influencing its texture and 2 coefficients describing the horizontal and vertical angle it is pointing at. It is important to keep in mind that the elements of l are not all having the same influence on the resulting face model image. To compute a distance between two label vectors l_1 and l_2 the texture, shape and angle part of them should be compared seperately after normalizing them related to the eigenvalues of the according principal components of the Basel Face Model. The implemented simulator is a convolutional network shown in figure 3.6 which is trained supervised on images generated using the original Basel Face Model. The reason why a neural network is trained on the Basel Face Model instead of just using it directly is that the described approach should be as general as possible and it would be conceivable to additionally learn helpful characteristics of the unlabeled training dataset to let the simulator approximate them. Figure 3.7 shows samples

[1] http://pytorch.org

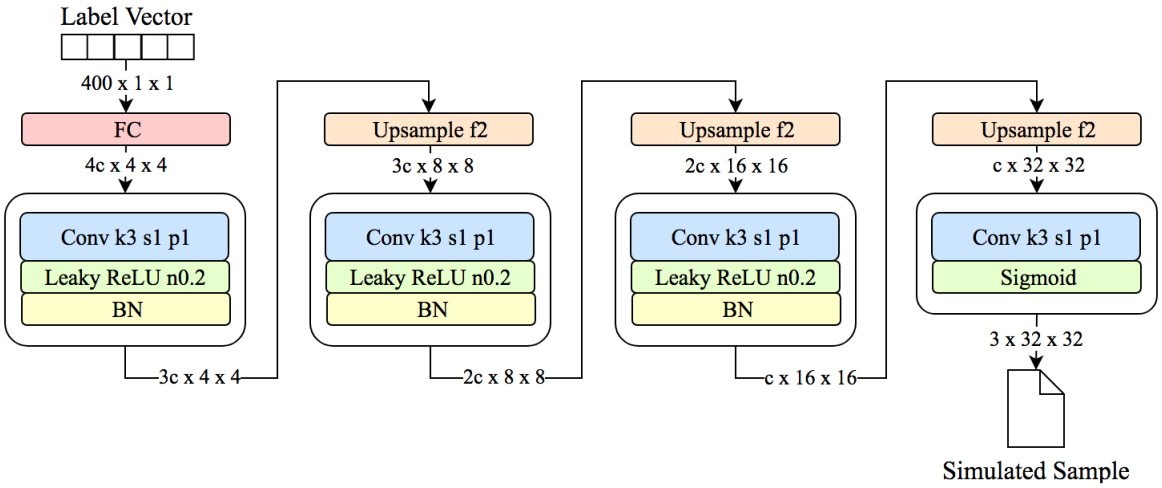

Figure 3.6: Simulator Architecture - The 400-dimensional label vector is processed by a fully connected layer (FC) and reshaped to 4c x 4 x 4 (Channels x Height x Width) with c being the basic amount of channels used. Afterwards, four convolutions with 3 x 3 kernel, stride 1 and padding 1 (Conv k3 s1 p1) are applied which are linearly decreasing the amount of channels. After each of the first three convolutions batch normalization (BN) and Leaky ReLU non-linearity with negative slope 0.2 are applied and the result is upsampled by factor 2 (Upsample f2). The last convolution is only followed by sigmoid activation and recuces the amount of channels to 3. The final result is a 32 x 32 rgb image of a simulated human face.

Figure 3.7: Basel Face Model - The upper row is showing examples of images generated from the original Basel Face Model and rescaled to size 32 x 32. The lower row is showing results of the simulator described in section 3.3.1.

generated from the original Basel Face Model and results of the trained simulator next to each other.

3.3.2 VGG

The used VGG model (Simonyan & Zisserman, 2014) is the pretrained VGG-16 model contained in Pytorch's `torchvision.models` package[2]. The network needs input images of atleast size 224 x 224. Because the implemented generator and simulator networks are just producing 32 x 32 images, they are upsampled by factor 7 before feeding them to the VGG. As the perceptual loss should only react to the similarity of the generated and simulated faces on the images and not to the similarity of the backgrounds, the generated images are masked so that the backgrounds of simulated and generated images is the same. Results of this preprocessing step can be seen in figure 3.8. The high level features used for computing the content similarity loss are the results of the ReLU layer before the penultimate max pooling layer of the network.

3.3.3 Generator

The generator G takes a noise label $z \in \mathcal{N}(0,1)$ and a label vector $l \in \mathcal{N}(0,1)$ as input and generates an image of a human face out of them. First of all, the the N_z-dimensional noise vector z and the 400-dimensional label vector l are concatenated.

[2]http://pytorch.org/docs/master/torchvision/models.html

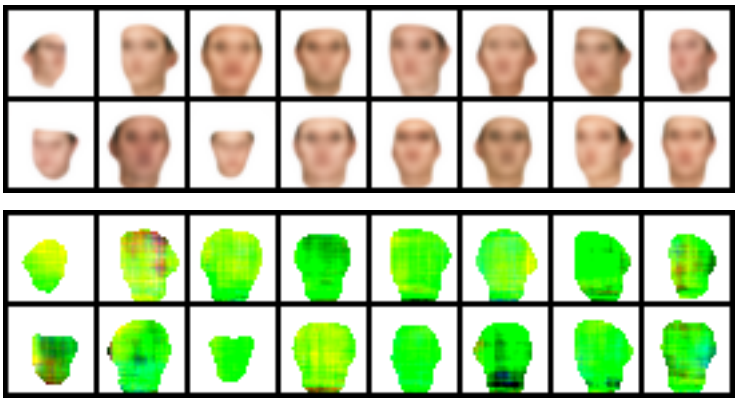

Figure 3.8: Masked Generated Images - The content similarity loss should only react to the similarity of the generated and simulated faces. The white, empty background of a simulated image in the upper row is transfered to the related generated image in the lower row before feeding them into the VGG.

The resulting $(N_z + 400)$-dimensional vector is stacked to get a tensor of shape $(N_z + 400)$ x 4 x 4 with each spatial location having all information about noise and label available. Afterwards, the tensor is processed by four residual blocks of two 3 x 3 convolutions with stride and padding 1 each followed by leaky ReLU non-linearity and pixelwise feature vector normalization to reduce the escalation of signal magnitudes especially when generator and discriminator are competing against each other. The skip connections of the residual blocks should fascilitate the training process, lead to sharper generated images and make it easier to propagate the label and noise information to the deeper layers. Pixelwise feature vector normalization is a form of local response normalization (LRN) (Krizhevsky *et al.*, 2012) and is defined in Karras *et al.*, 2017 as

$$b_{x,y} = a_{x,y} / \sqrt{\frac{1}{C} \sum_{j=0}^{C-1} (a_{x,y}^{j})^2 + \epsilon} \tag{3.10}$$

with $b_{x,y}$ and $a_{x,y}$ being the normalized respectively original feature vector in pixel (x, y), C being the number of feature maps (or channels) and $\epsilon = 10^{-8}$. Each block is reducing the amount of channels linearly. After the first three blocks the result is upsampled by factor 2 by nearest neighbor interpolation, finally leading to a c x 32 x 32 tensor. After the last block, a 3 x 3 convolution with stride and padding 1 is used to reduce the amount of channels to 3 to get a rgb image of size 32 x 32. The described generator architecture is shown in figure 3.9. Algorithm A.1 outlines how an update step of the generator is implemented.

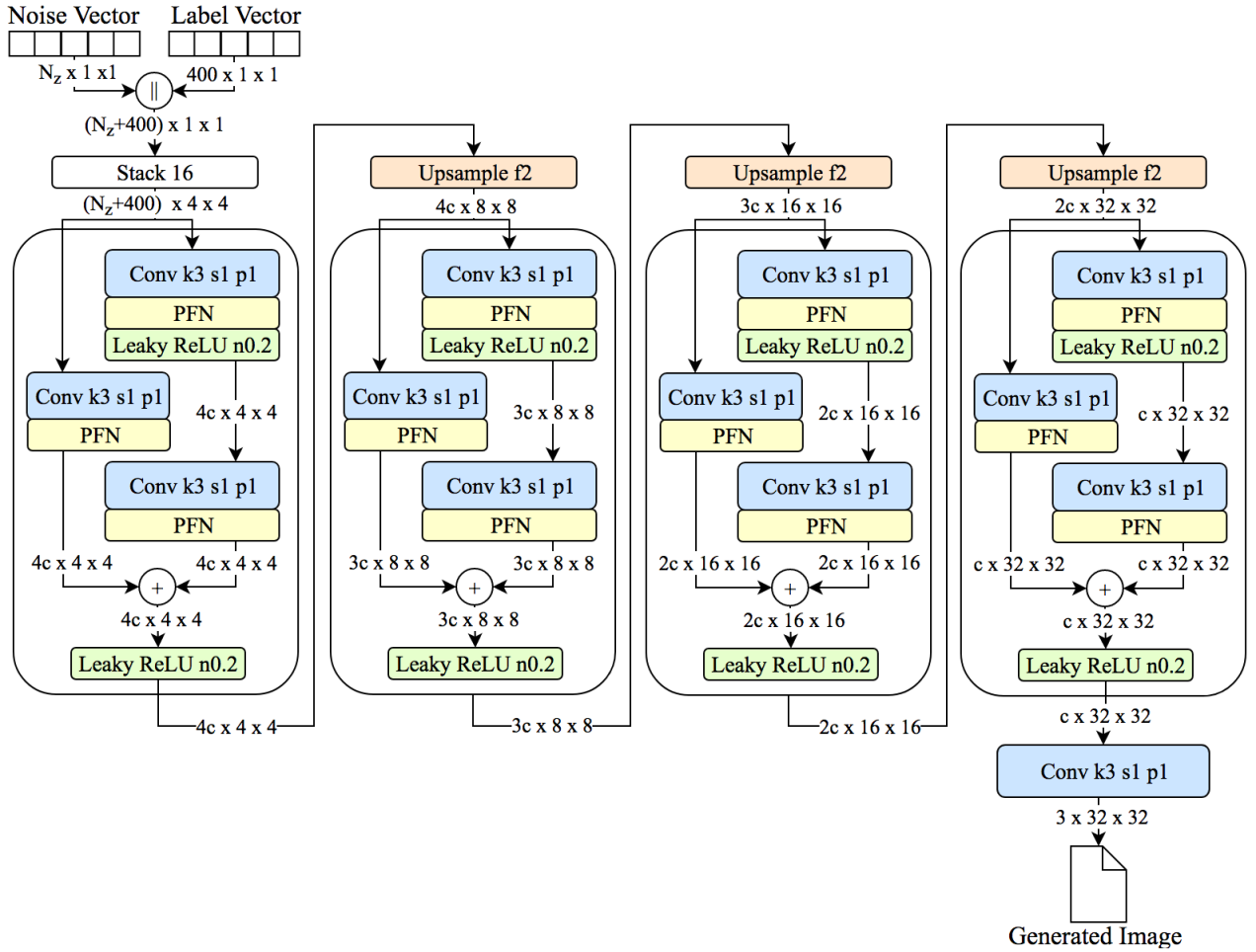

Figure 3.9: Generator Architecture - Noise and label vector are concatenated and stacked to get a $(N_z + 400)$ x 4 x 4 tensor which is then processed by four residual blocks of two 3 x 3 convolutions with stride and padding 1 followed by leaky ReLU non-linearity with negative slope 0.2 and pixelwise feature vector normalization (PFN). After the first three blocks, the result is upsampled by factor 2. The amount of channels is linearly decreased with each block. After the last block, the amount of channels is reduced to 3 by a single 3 x 3 convolution with stride and padding 1.

3.3.4 Discriminator

The discriminator is built out of two parts. The first part is an encoder that reduces a generated image of the generator or a real image from the cropped Large-scale CelebFaces Attributes (CelebA) (Liu *et al.*, 2015) and Labeled Faces in the Wild (LFW) (Huang *et al.*, 2007) dataset to an embedding h. The sample is processed by a 3 x 3 convolution with stride and padding 1 to increase the amount of channels. Then, 4 residual blocks with two convolutions similar to the ones used by the generator described in section 3.3.3 are following. Instead of upsampling the result by factor 2 after each block as done by the generator, the result is downsampled by average pooling with a 2 x 2 kernel and stride 2. Before the last block of convolutions, minibatch normalization (MBN) as described in Karras *et al.*, 2017 is applied to increase variation and therefore reduce mode collapse. After the last block, the result is mapped by two fully connected layers to an N_h-dimensional embedding h. A dropout layer is placed between the two fully connected layers to increase the autoencoders generalization capabilites. This embedding is afterwards again processed by two fully connected layers with a dropout layer inbetween and reshaped to a tensor of shape 5n x 4 x 4. This tensor is used by the second part of the discriminator, the decoder, to reconstruct the original input sample. The decoder architecture of the discriminator is, apart from the eventually different amount of channels of the input, identical to the generator architecture described in section 3.3.3 and shown in figure 3.9. The whole architecture of the discriminator is outlined in figure 3.10. An update step of the discriminator network is outlined in algorithm A.2.

3.3.5 Predictor

The Predictor maps an embedding h created by the encoder part of the discriminator to a (N_z+400)-dimensional vector with the first N_z elements being the predicted label vector $\tilde{z}$ and the last 400 elements being the predicted label vector $\tilde{l}$. The predictor consists out of three fully connected layers and a dropout layer as shown in figure 3.11. Algorithm A.3 outlines an update step of the predictor network.

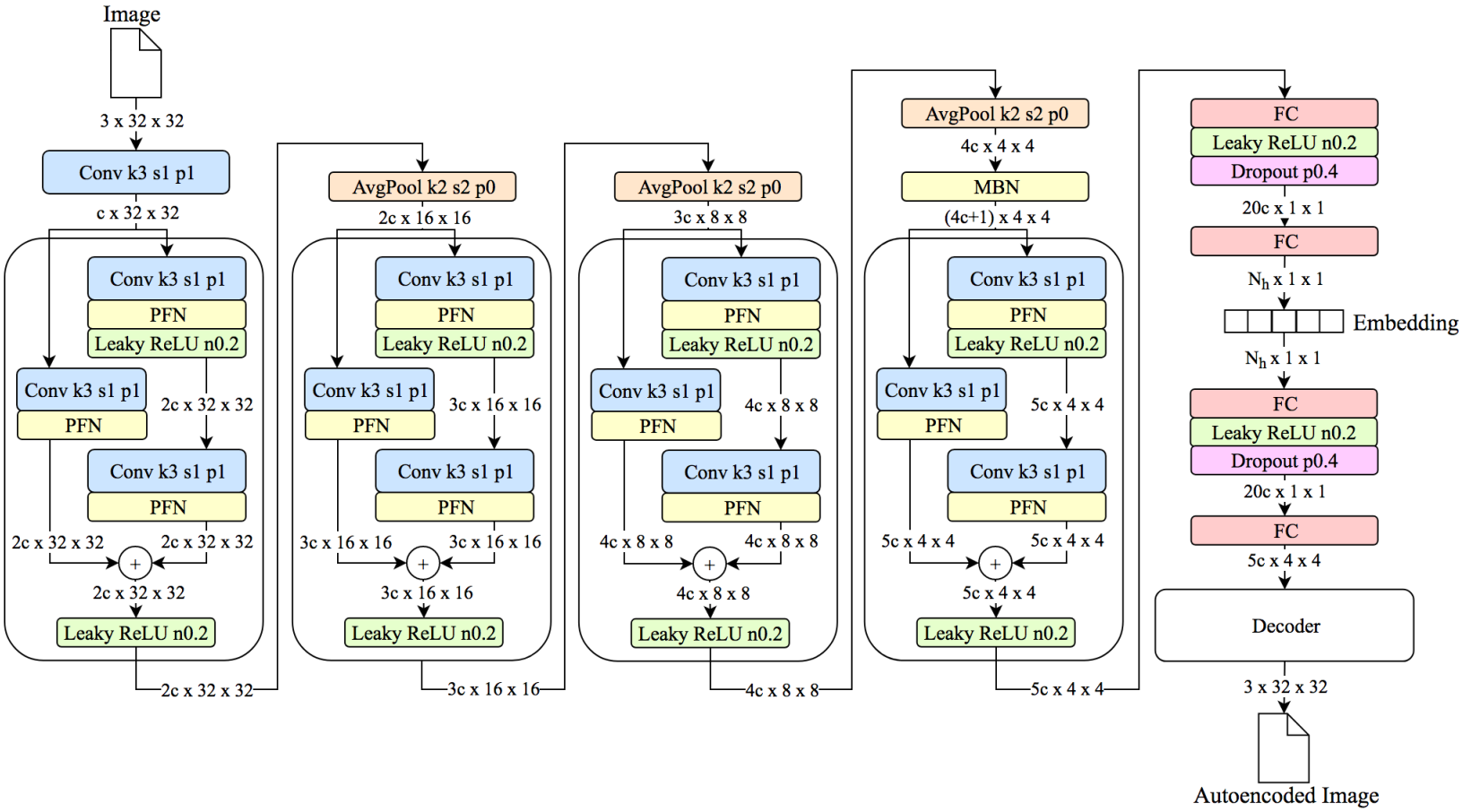

Figure 3.10: Discriminator Architecture - The sample is processed by one 3 x 3 convolution and four residual blocks similar to the ones of the generator. After the first three blocks average pooling with 2 x 2 kernel, stride 2 and padding 0 (AvgPool k2 s2 p0) is applied. Minibatch normalization (MBN) after the last downsampling layer is used to increase variation. After the last block the result is mapped by two fully connected layers to an N_h-dimensional embedding. This embedding is then again mapped by two fully connected layers to a 5c x 4 x 4 tensor which is used by the decoder to reconstruct the input image.

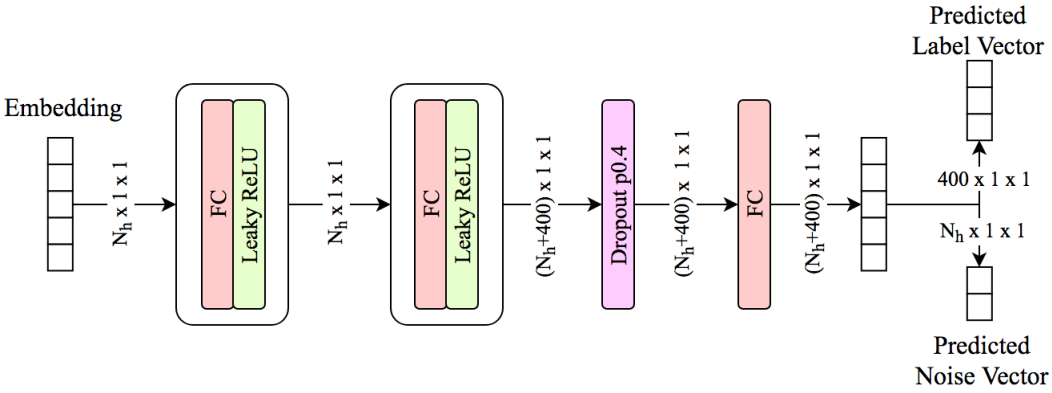

Figure 3.11: Predictor Architecture - The embedding is processed by three fully connected layers. The first two are followed by leaky ReLU non-linearity with negative slope 0.2. The penultimate fully connected layer is followed by a dropout layer with dropout probability 0.4 (Dropout p0.4). The result is a (N_z+400)-dimensional vector. The first N_z elements are the predicted noise vector $\tilde{z}$, the last 400 elements the predicted label vector $\tilde{l}$.

Chapter 4

Evaluation

In this chapter the model described in section 3.2 implemented as outlined in section 3.3 is evaluated. It was trained in a Docker[1] container instance running Debian[2]. The used graphic processor was a Nvidia GTX 1060[3].

4.1 Training

One training iteration consists out of three discriminator update steps and one predictor and generator update step executed seperately after each other. The learning signal the discriminator provides will therefore get better faster and all update steps will take place in the most stable environment possible. Instead of standard SGD, Adam optimization (Kingma & Ba, 2014) was used for parameter updates. The used default hyperparameters can be seen in figure 4.1. The first thing to point out is that the BEGAN equilibirum is not touched by the additional losses. As shown in figure 4.2, neither the generator nor the discriminator is getting a big advantage over the other after the initial balancing. The global measure of convergence M_{global} shown in figure 4.3 approves this as well and indicates that generator and discriminator are still able to learn from each other. The adjustments of k_t shown in figure 4.4 to control the equilibrium are telling that the generator has an easier time to trick the discriminator in the beginning of training and that it is getting harder with increasing amount of iterations. The generator loss $\mathcal{L}_G$ is higher than the discriminator loss $\mathcal{L}_D$ because he has to deal with the additional losses $\mathcal{L}_c$ and $\mathcal{L}_z$ shown in figure 4.5. The

[1] https://www.docker.com
[2] https://www.debian.org
[3] http://www.nvidia.de/graphics-cards/geforce/pascal/gtx-1060/

Parameter	Value	Description
B	24	The used batch size. Relatively small because of hardware limitations.
I_{max}	$50,000$	Maximum amount of iterations after which the training stops. Small due to time limitations.
N_z	32	Dimension of noise vectors z.
N_h	64	Dimension of the discriminators autoencoder embedding h.
c	32	Basic number of channels used in simulator, discriminator and generator.
w_c	0.1	Weight of content-similarity loss $\mathcal{L}_c$ on generator loss.
w_z	0.05	Weight of noise loss $\mathcal{L}_z$ on generator loss.
γ	0.7	BEGAN parameter to relax equilibrium and influence sample diversity.
λ_k	$1 \cdot 10^{-3}$	BEGAN learning rate for k_t updates controlling the equilibrium.
λ_D, λ_G, λ_P	$5 \cdot 10^{-5}$	Learning rates for discriminator, generator and predictor update steps.

Figure 4.1: The default hyperparameters used for evaluation.

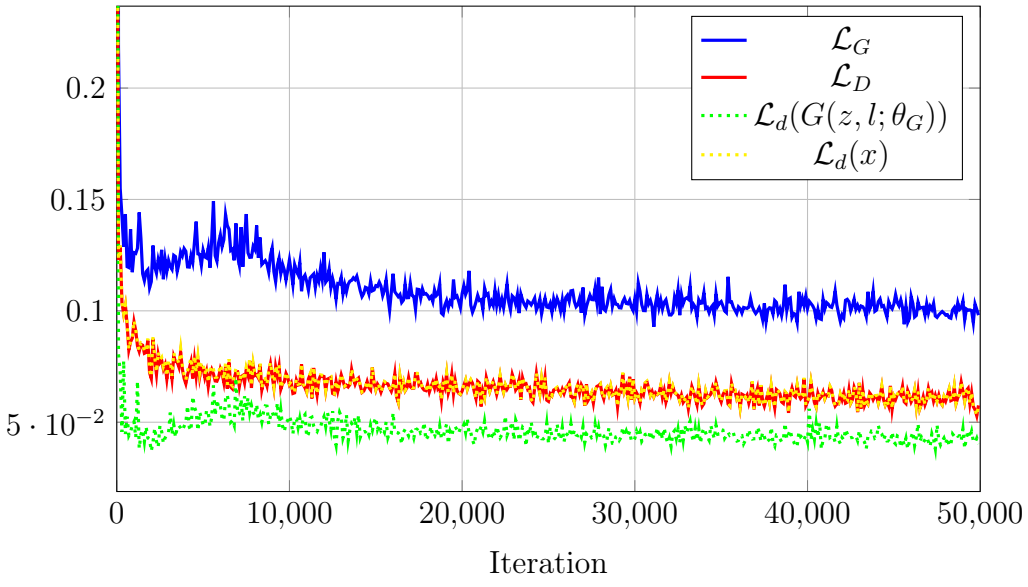

Figure 4.2: Plot of Discriminator and Generator Loss - After the initial balancing neither discriminator nor generator is getting an advantage over the other. Even though the autoencoder loss for generated samples is kept smaller than the one for real images by the equilibrium, the generator loss is higher than the discriminator loss because of the additional losses $\mathcal{L}_c$ and $\mathcal{L}_z$.

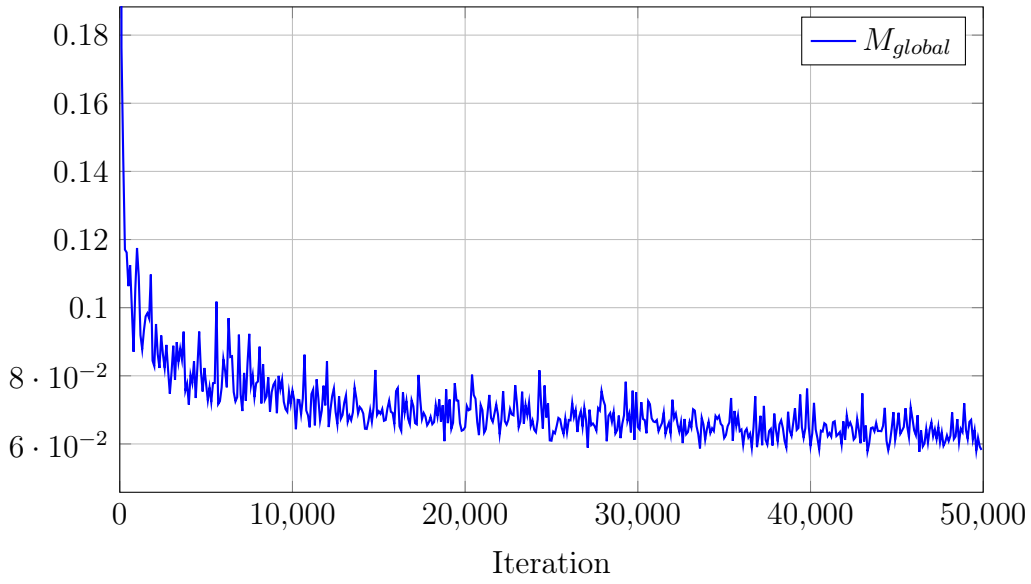

Figure 4.3: Plot of Global Convergence Measure M_{global} - The model is converging. Discriminator and generator are able to learn from each other.

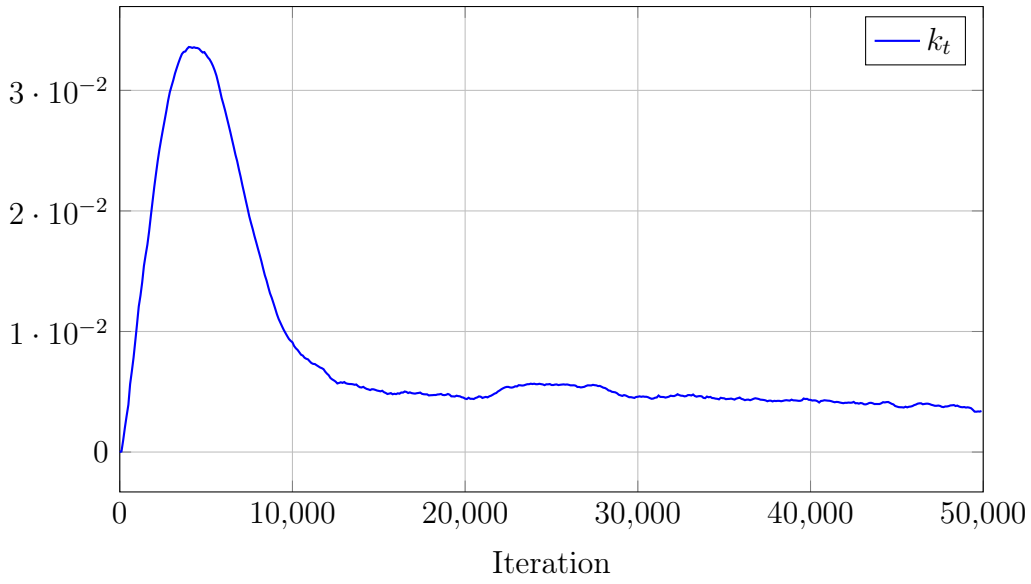

Figure 4.4: Plot of k_t - It is easier for the generator to trick the discriminator in the beginning of the training. It gets harder with increasing amount of iterations.

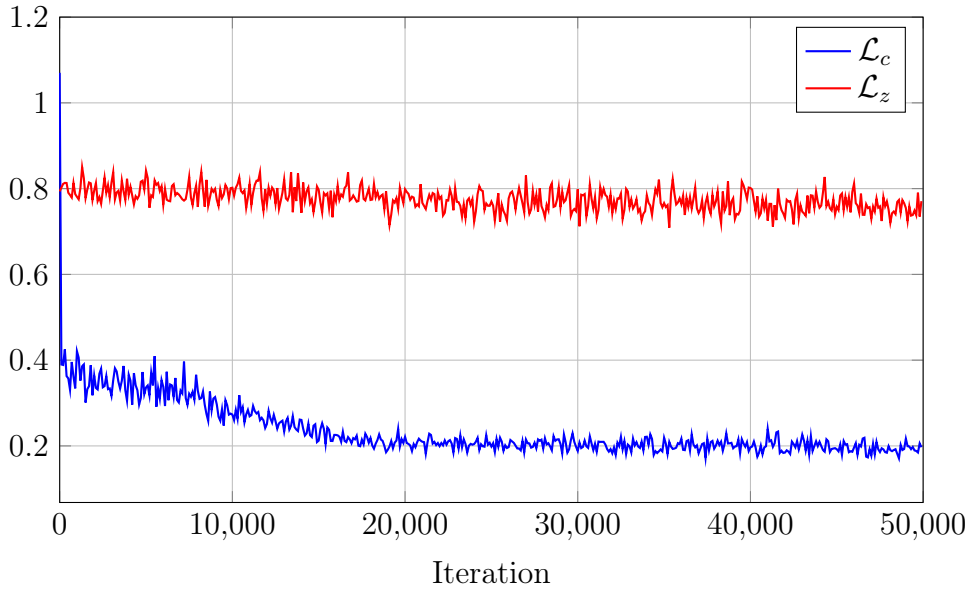

Figure 4.5: Plot of $\mathcal{L}_c$ and $\mathcal{L}_z$ - The generator is learning to encode label information correctly into generated images and to use the noise vector to realistically customize them. It is harder for the generator to find a way to encode the noise vector realistically so that the predictor can reproduce it than to minimize the difference between the VGG features of simulated face model images and masked generated images.

decrease of both losses proves that the generator is learning to include label information correctly into his generations and is trying to customize the images realistically by using the noise information from z. Finding a way to encode z realistically so that the predictor can reproduce them is harder than minimizing the perceptual loss between masked generated images and simulated face model images. The decreasing label loss $\mathcal{L}_l$ shown in figure 4.6 shows that the training of the predictor is working as well. Its interesting to see that the label loss is basically solely depending on the difference between original and predicted horizontal and vertical angles the faces are pointing at. The shape and texture seem to be easy to predict and it is impossible for the predictor to get better at it. The training was interrupted after 50.000 iterations even though some parts of the model were still making progress. Better results might be possible by increasing the maximum amount of iterations.

4.2 Results

The previous section suggests that the training process works at least to a certain degree. Figure 4.7 shows examples of images created by the generator after training

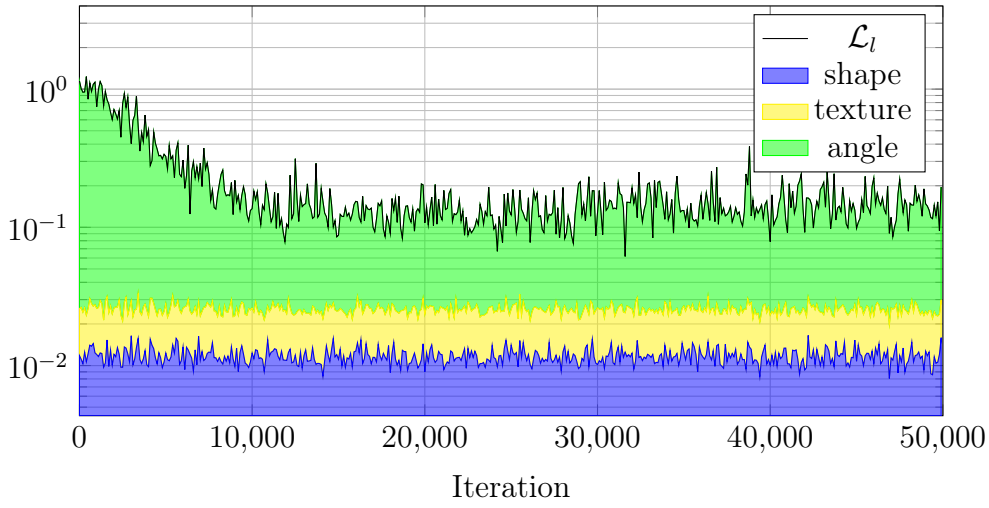

Figure 4.6: Plot of $\mathcal{L}_l$ and its Components - The loss related to the angles the faces are looking at is by far the most influencing component. It seems to be impossible for the predictor to get better in predicting the label vector parts related to texture and shape.

next to samples from the training dataset of real images. The generator does create realistic looking images. The most obvious difference between real and generated images are the white borders around the faces. These borders correspond to the simulated model faces used to enforce the correct encoding of the label information which can be seen in figure 4.8. The generated images are also blurrier than the real ones which is probably the result of the L1 image loss used to compute the autoencoder reconstruction error and the blurry simulated face model images used for computing the content-similarity loss. The white border and other artifacts are

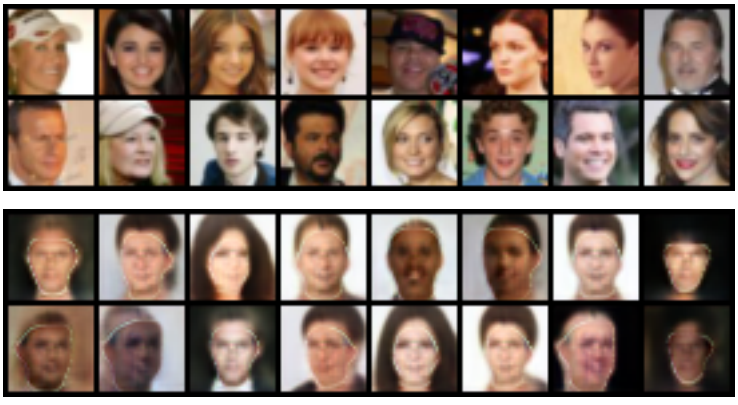

Figure 4.7: Real and generated Images - The upper row shows real images from the training dataset. The lower row shows results of the generator.

Figure 4.8: Generated and simulated Images - The upper row shows results of the generator $G(z, l; \theta_G)$ after training. The lower row shows the related results of the simulator $S(l; \theta_S)$.

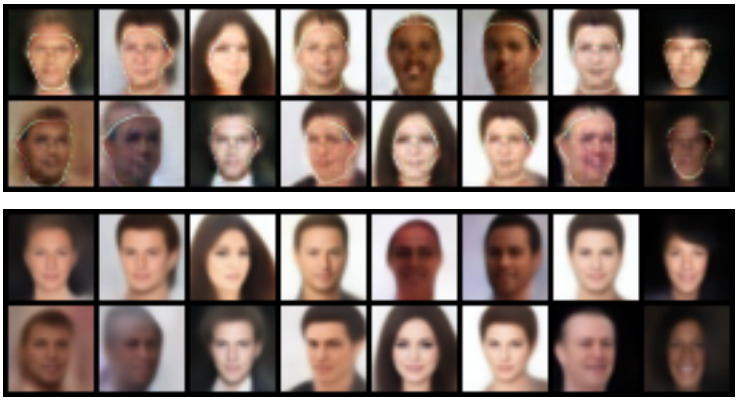

Figure 4.9: Generated Images and their autoencoded Counterparts - The autoencoder is removing the white border around the faces and other artifacts in exchange of blurriness.

removed by the autoencoder as can be seen in figure 4.9. The label information seems to be encoded by the generator. The most obvious indication for that is that the lines of sight and the shape of most generated faces correspond really good to the related face models. This can be seen even better in figure 4.10 which shows masked generated images used as input for the pretrained VGG model next to the related original and predicted face models. Exceptions are face models pointing at extreme vertical angles, like faces looking straight to the top or bottom. The reason for this should be that these type of faces are not at all or only sparsely represented in the training dataset of real images. Textures are not transfered as good as the angles and the shape, but the predictor has no problems in reproducing them. The cause of that might be that the content-similarity loss using feature layers of the pretrained VGG model is

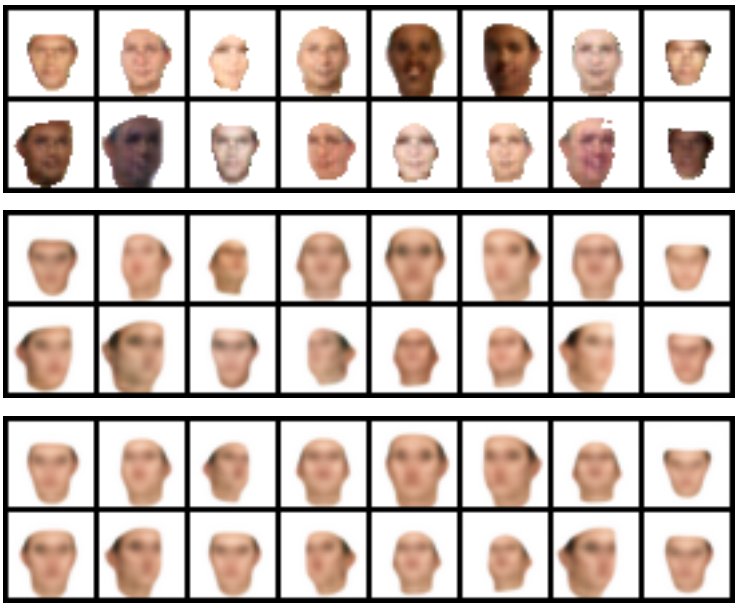

Figure 4.10: Masked Generated Images next to simulated and predicted Face Models - The top row shows samples of the masked results of the generator. The middle row shows the related simulated face models used to ensure the correct encoding of the label information by the generator. The bottom row shows the face models created by the simulator out of the predicted label vector generated by the predictor.

more focused on structures and contours than on textures so the generator is likely to change them and that the textures of the simulated face models do not vary much. The overall small difference between original and predicted face models in figure 4.10 shows that the predictor is able to reproduce the label vectors with high accuracy even though the default value of w_c is small. Higher values of w_c are leading to a smaller difference between masked generated images and simulated face model images but also make it harder for the generator to create images that trick the discriminator as shown in figure 4.11 and 4.12. Increasing w_c therefore causes a decrease of diversity of generated images. Figure 4.13 shows that the predictor is able to classify real images from the training dataset to a certain degree. The label predictions are good in most cases. Extreme angles are posing a problem again and the texture is not predicted as good as the lines of sight and the shape. Because the simulated face models do not cover all of the characteristics of faces in the training dataset of real images especially with regard to textures or face properties like beards, this was to be expected. It is also important to note that increasing w_c reduces the capability of the predictor to reproduce correct label vectors out of autoencoder embeddings from real faces, as can be seen in figure 4.14. The reason for that is that higher values of w_c prevent

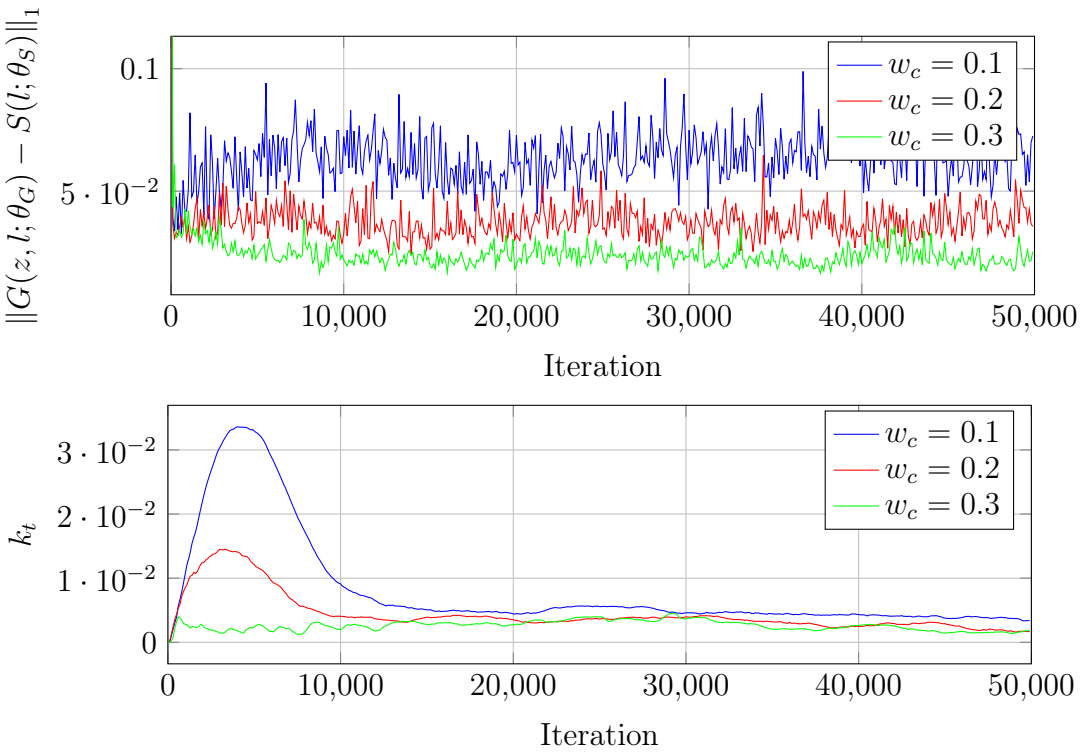

Figure 4.11: Influence of w_c on the generator - Higher values of w_c lead to better matches of generated images and simulated face models but make it harder for the generator to create realistic images tricking the discriminator which is reflected by an overall lower value of k_t.

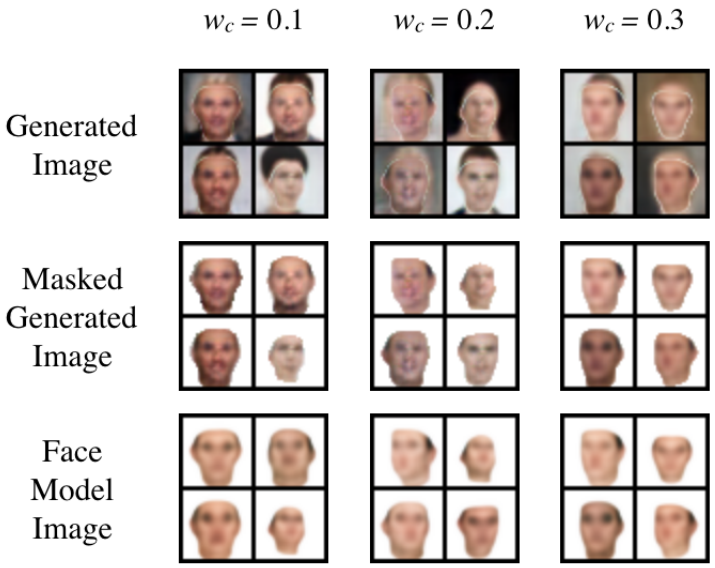

Figure 4.12: Generated Images for different values of w_c - With increasing w_c the image diversity is decreasing and the matching of masked generated images and simulated face models is increasing.

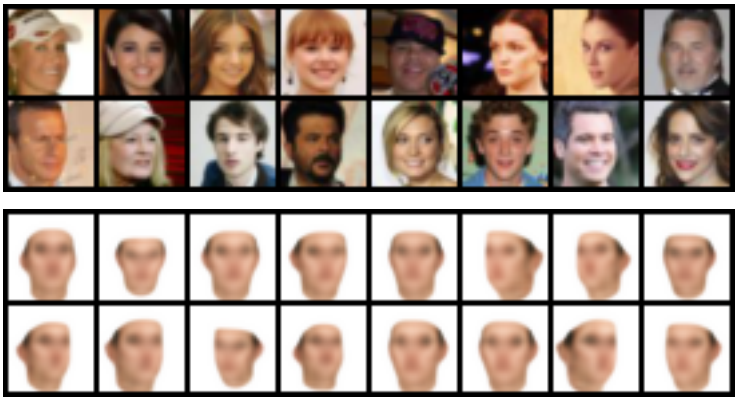

Figure 4.13: Real Images and predicted Face Models - The upper row shows samples of real images from the training dataset. The lower row shows the face model images created by the simulator from the predicted label vector the predictor generated out of the according autoencoder embeddings of the real images.

Figure 4.14: Influence of w_c on the predictor - Higher values of w_c are decreasing the capability of the predictor to predict correct label vectors for real images.

the generator from adapting the faces to appear more realistic. Because of that the domain shift between generated images and real images stays higher. The samples the predictor is trained on are therefore not very similar to real images and learned characteristics of the domain of generated images can not be transfered to the real image domain that easy.

Chapter 5

Discussion

The proposed model has two outcomes. The first one is a generator that is able to generate realistic images corresponding correctly to label information given to him in form of a label vector l and which can be customized realistically through a noise vector z. Figure 5.1 shows that the model actually learned to create those images by itself and is not reusing real images from the training dataset. Artifacts like white borders around the faces are removed by the discriminator in exchange of blurriness. Difficulties are arising from face models pointing at extreme angles because they are not represented well in the training dataset of real images. There is no mode collapse, but increasing w_c to put more emphasis on correctly encoding label information into the generated images decreases the image diversity. The reason for that is most probably the domain shift between the face models and real faces in the training dataset which makes it hard to create realistic images when being forced to include the face models one-to-one. Figure 5.2 shows a matrix of results of the generator for

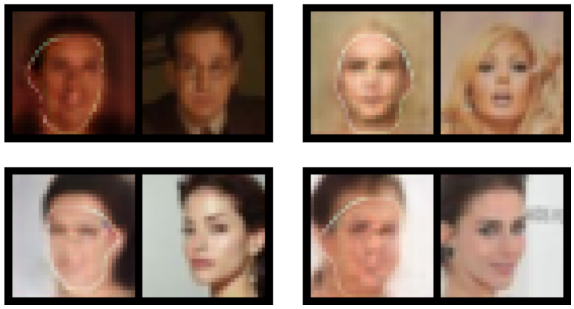

Figure 5.1: Generated Images next to their Nearest Neighbours in the Training Dataset - The difference between the generated images and their nearest neighbours shows that the model learned to actually create new images by itself.

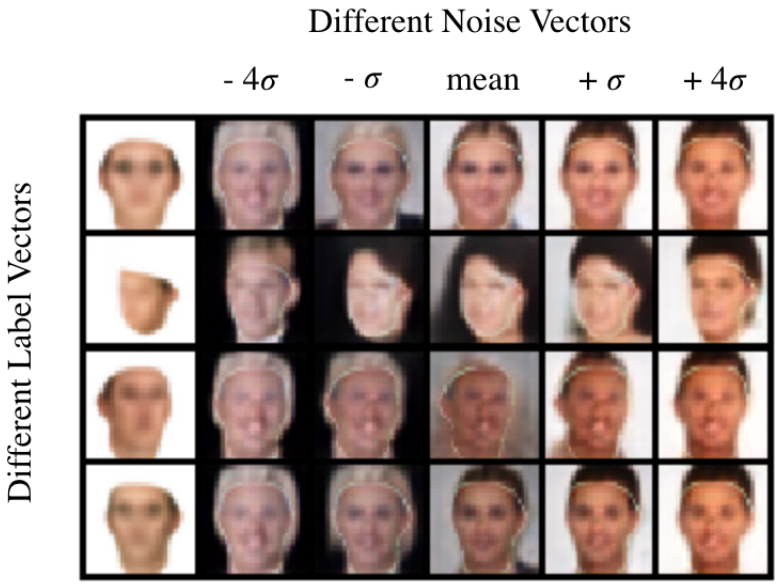

Figure 5.2: Generator Results for fixed Label or Noise Vectors - The label information is preserved when changing the noise vector. Changing the label vector while using the same noise vector does lead to different images of the same style.

fixed label respectively noise vectors. It can be seen that it is possible to generate different realistic samples containing the same or different label information which theoretically enables the generation of infinite labeled, realistic training datasets of images of human faces. The difference of images generated from a fixed label vector and varying noise vectors are not huge but definetely noticeable. Ideally, the label information given by the label vector should only be encoded into the part of the image where the human face is located. Additionally, label information solely related to for example the nose should only change the appearance of the nose and not of the eyes. Figure 5.2 shows that in some cases changing the label vector does in fact not only change the representation of the human face, but also the whole image. That means that some label information may not be encoded correctly or influencing more parts of the image than it should be. The problem is to tell the generator what the label information actually means, force him to only encode it accordingly, maybe let him omit information he can not encode realistically and give him the freedom to change other details. The pretrained VGG model does not seem to be able to provide the generator with a perfect learning signal, even though it works quiet well overall.

The second outcome is a predictor that can predict label vectors related to the Basel Face Model from generated as well as real images of human faces. The predictor

is working better for generated images than for real ones. Original and predicted face models of generated images shown in figure 4.10 are almost equal. Predicting the horizontal and vertical angles real faces are pointing at and the shape of them is working well. The prediction of textures is not working that well because of the limited range of textures provided by the simulated face models.

Chapter 6

Conclusion

In this thesis a novel generative approach of unsupervised domain adaptation based on the BEGAN architecture was described. The implementation of the proposed model was set out in detail. It was shown that the model is able to overcome a relatively large domain shift between simulated images of face models and real images of human faces. The model created realistic, labeled images of human faces and was able to predict label vectors from images of real human faces to a certain degree.

6.1 Final Thoughts

The biggest problem was to force the generator to encode label information in the way it should be encoded while giving the freedom to change other details. The generator should for example encode the face model label information only into human faces and not at all into the background as well as label information solely related to the nose should not be changing the appearance of the eyes. This obviously gets harder with increasing semantic complexity of the labels. The usage of a pretrained VGG model to enforce content-similarity of simulated and generated image is working pretty well for the considered use case but it is also the biggest drawback of the described approach. It should work for most domains whose labels are describing objects in images but it might have to be substituted to apply it on other domains. A first step to tackle this problem could be to evaluate if feature layers of the encoder part of the discriminator can be used instead of the feature layers of a pretrained VGG to ensure the content-similarity.

6.2 Future Studies

There are a lot of things that could be done following up the approach described in this thesis. A basic one could be to apply the approach on other domains than images of human faces, like images of cars in street environments. It may also be possible to improve the image quality of generated samples by letting the generator and discriminator grow with increasing amount of iterations as described in Karras *et al.*, 2017. Using images of higher resolution than 32 x 32 might improve the results as well. Finding other ways to ensure label-aware content-similarity is probably the most promising but also hardest subject of future studies. Another interesting idea would be to let the generator directly create embeddings out of a noise and label vector for a pretrained autoencoder. The generator could then be trained to generate embeddings that are decoded to realistic looking images whose embeddings created by the pretrained encoder part of the autoencoder are still containing the label information included in the original ones. For this approach *Deep Invertible Networks* (i-RevNets) (Jacobsen *et al.*, 2018) might be useful.

References

Arjovsky, Martin, Chintala, Soumith, & Bottou, Léon. 2017. Wasserstein GAN. *arXiv:1701.07875 [cs, stat]*, Jan. arXiv: 1701.07875.

Berthelot, David, Schumm, Thomas, & Metz, Luke. 2017. BEGAN: Boundary Equilibrium Generative Adversarial Networks. *arXiv:1703.10717 [cs, stat]*, Mar. arXiv: 1703.10717.

Bousmalis, Konstantinos, Trigeorgis, George, Silberman, Nathan, Krishnan, Dilip, & Erhan, Dumitru. 2016a. Domain Separation Networks. *arXiv:1608.06019 [cs]*, Aug. arXiv: 1608.06019.

Bousmalis, Konstantinos, Silberman, Nathan, Dohan, David, Erhan, Dumitru, & Krishnan, Dilip. 2016b. Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks. *arXiv:1612.05424 [cs]*, Dec. arXiv: 1612.05424.

Bousmalis, Konstantinos, Irpan, Alex, Wohlhart, Paul, Bai, Yunfei, Kelcey, Matthew, Kalakrishnan, Mrinal, Downs, Laura, Ibarz, Julian, Pastor, Peter, Konolige, Kurt, Levine, Sergey, & Vanhoucke, Vincent. 2017. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. *arXiv:1709.07857 [cs]*, Sept. arXiv: 1709.07857.

Chen, Xi, Duan, Yan, Houthooft, Rein, Schulman, John, Sutskever, Ilya, & Abbeel, Pieter. 2016. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. *arXiv:1606.03657 [cs, stat]*, June. arXiv: 1606.03657.

D'Avino, Dario, Cozzolino, Davide, Poggi, Giovanni, & Verdoliva, Luisa. 2017. Autoencoder with recurrent neural networks for video forgery detection. *arXiv:1708.08754 [cs]*, Aug. arXiv: 1708.08754.

Denton, Emily, Chintala, Soumith, Szlam, Arthur, & Fergus, Rob. 2015. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. *arXiv:1506.05751 [cs]*, June. arXiv: 1506.05751.

Ganin, Yaroslav, & Lempitsky, Victor. 2014. Unsupervised Domain Adaptation by Backpropagation. *arXiv:1409.7495 [cs, stat]*, Sept. arXiv: 1409.7495.

Gatys, Leon A., Ecker, Alexander S., & Bethge, Matthias. 2015. A Neural Algorithm of Artistic Style. *arXiv:1508.06576 [cs, q-bio]*, Aug. arXiv: 1508.06576.

Gauthier, Jon. 2014. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester*, **2014**(5), 2.

Ghifary, Muhammad, Kleijn, W. Bastiaan, Zhang, Mengjie, Balduzzi, David, & Li, Wen. 2016. Deep Reconstruction-Classification Networks for Unsupervised Domain Adaptation. *arXiv:1607.03516 [cs, stat]*, July. arXiv: 1607.03516.

Girshick, Ross, Donahue, Jeff, Darrell, Trevor, & Malik, Jitendra. 2013. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv:1311.2524 [cs]*, Nov. arXiv: 1311.2524.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, & Bengio, Yoshua. 2014. Generative Adversarial Nets. *Pages 2672–2680 of:* Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., & Weinberger, K. Q. (eds), *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc.

Goodfellow, Ian, Bengio, Yoshua, & Courville, Aaron. 2016. *Deep Learning*. MIT Press.

Gretton, A., Smola, AJ., Huang, J., Schmittfull, M., Borgwardt, KM., & Schölkopf, B. 2009. Covariate shift and local learning by distribution matching. *Pages 131–160 of: Dataset Shift in Machine Learning*. Cambridge, MA, USA: Biologische Kybernetik.

Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent, & Courville, Aaron. 2017. Improved Training of Wasserstein GANs. *arXiv:1704.00028 [cs, stat]*, Mar. arXiv: 1704.00028.

Heusel, Martin, Ramsauer, Hubert, Unterthiner, Thomas, Nessler, Bernhard, Klambauer, Günter, & Hochreiter, Sepp. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium. *arXiv:1706.08500 [cs]*, June. arXiv: 1706.08500.

Huang, Gary B., Ramesh, Manu, Berg, Tamara, & Learned-Miller, Erik. 2007 (Oct.). *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rept. 07-49. University of Massachusetts, Amherst.

Jacobsen, Jörn-Henrik, Smeulders, Arnold, & Oyallon, Edouard. 2018. i-RevNet: Deep Invertible Networks. *arXiv:1802.07088 [cs, stat]*, Feb. arXiv: 1802.07088.

Jing, Yongcheng, Yang, Yezhou, Feng, Zunlei, Ye, Jingwen, & Song, Mingli. 2017. Neural Style Transfer: A Review. *arXiv:1705.04058 [cs]*, May. arXiv: 1705.04058.

Johnson, Justin, Alahi, Alexandre, & Fei-Fei, Li. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *arXiv:1603.08155 [cs]*, Mar. arXiv: 1603.08155.

Karras, Tero, Aila, Timo, Laine, Samuli, & Lehtinen, Jaakko. 2017. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *arXiv:1710.10196 [cs, stat]*, Oct. arXiv: 1710.10196.

Kingma, Diederik P., & Ba, Jimmy. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, Dec. arXiv: 1412.6980.

Kingma, Diederik P., & Welling, Max. 2013. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, Dec. arXiv: 1312.6114.

Krizhevsky, Alex, Sutskever, Ilya, & Hinton, Geoffrey E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Pages 1097–1105 of:* Pereira, F., Burges, C. J. C., Bottou, L., & Weinberger, K. Q. (eds), *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc.

Liu, Ming-Yu, & Tuzel, Oncel. 2016. Coupled Generative Adversarial Networks. *arXiv:1606.07536 [cs]*, June. arXiv: 1606.07536.

Liu, Ziwei, Luo, Ping, Wang, Xiaogang, & Tang, Xiaoou. 2015 (Dec.). Deep Learning Face Attributes in the Wild. *In: Proceedings of International Conference on Computer Vision (ICCV)*.

Long, Mingsheng, Cao, Yue, Wang, Jianmin, & Jordan, Michael I. 2015. Learning Transferable Features with Deep Adaptation Networks. *arXiv:1502.02791 [cs]*, Feb. arXiv: 1502.02791.

Oord, Aaron van den, Kalchbrenner, Nal, & Kavukcuoglu, Koray. 2016. Pixel Recurrent Neural Networks. *arXiv:1601.06759 [cs]*, Jan. arXiv: 1601.06759.

Paysan, P., Knothe, R., Amberg, B., Romdhani, S., & Vetter, T. 2009. *A 3D Face Model for Pose and Illumination Invariant Face Recognition*. Genova, Italy: IEEE.

Radford, Alec, Metz, Luke, & Chintala, Soumith. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434 [cs]*, Nov. arXiv: 1511.06434.

Rothe, Rasmus, Timofte, Radu, & Gool, Luc Van. 2016. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision (IJCV)*, July.

Shelhamer, Evan, Long, Jonathan, & Darrell, Trevor. 2016. Fully Convolutional Networks for Semantic Segmentation. *arXiv:1605.06211 [cs]*, May. arXiv: 1605.06211.

Simonyan, K., & Zisserman, A. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, **abs/1409.1556**.

Sixt, Leon, Wild, Benjamin, & Landgraf, Tim. 2016. RenderGAN: Generating Realistic Labeled Data. *arXiv:1611.01331 [cs]*, Nov. arXiv: 1611.01331.

Springenberg, Jost Tobias. 2015. Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks. *arXiv:1511.06390 [cs, stat]*, Nov. arXiv: 1511.06390.

Srivastava, Akash, Valkov, Lazar, Russell, Chris, Gutmann, Michael U., & Sutton, Charles. 2017. VEEGAN: Reducing Mode Collapse in GANs using Implicit Variational Learning. *arXiv:1705.07761 [stat]*, May. arXiv: 1705.07761.

Tzeng, Eric, Hoffman, Judy, Zhang, Ning, Saenko, Kate, & Darrell, Trevor. 2014. Deep Domain Confusion: Maximizing for Domain Invariance. *arXiv:1412.3474 [cs]*, Dec. arXiv: 1412.3474.

Tzeng, Eric, Hoffman, Judy, Darrell, Trevor, & Saenko, Kate. 2015. Simultaneous Deep Transfer Across Domains and Tasks. *arXiv:1510.02192 [cs]*, Oct. arXiv: 1510.02192.

Tzeng, Eric, Hoffman, Judy, Saenko, Kate, & Darrell, Trevor. 2017. Adversarial Discriminative Domain Adaptation. *arXiv:1702.05464 [cs]*, Feb. arXiv: 1702.05464.

Zhang, Chiyuan, Bengio, Samy, Hardt, Moritz, Recht, Benjamin, & Vinyals, Oriol. 2016. Understanding deep learning requires rethinking generalization. *arXiv:1611.03530 [cs]*, Nov. arXiv: 1611.03530.

Zhao, Junbo, Mathieu, Michael, & LeCun, Yann. 2016. Energy-based Generative Adversarial Network. *arXiv:1609.03126 [cs, stat]*, Sept. arXiv: 1609.03126.

Acknowledgements

I wish to express my sincere thanks to Prof. Dr. Tim Landgraf for supervising this thesis and the whole BioRobotics-Lab team for the support, ideas and feedback throughout the process of writing it. Our meetings were very motivating and helped me a lot. I also thank my family and friends for the unceasing encouragement, support and attention. Last but not least I want to express gratitude to my fellow students Luca Keidel, David Bohn and Lars Hochstetter, who had always a friendly ear for me.

Appendix A

Algorithms

Algorithm A.1 Generator Update Step

for $i \in 1...B$:
  #Sample noise and label vector
  sample $z^i \sim \mathbb{P}_z$
  sample $l^i \sim \mathbb{P}_l$

  #Create face model from label
  $x_S^i = \mathrm{S}(l^i)$
  #Generate and discriminate fake sample
  $x_G^i = \mathrm{G}(z^i, l^i)$
  $ae_{x_G^i},\ h_{x_G^i} = \mathrm{D}(x_G^i)$
  #Predict label and noise vector from embedding
  $z_{x_G^i}, l_{x_G^i} = P(h_{x_G^i})$
  #Process generated face and face model through VGG
  #and extract high level features
  mask = getMask(x_S^i)
  $f_{x_G^i} = \mathrm{VGG16}(mask \cdot x_G^i).\mathrm{relu4\_3}$
  $f_{x_S^i} = \mathrm{VGG16}(x_S^i).\mathrm{relu4\_3}$

#Compute losses (baselLoss: normalized loss from section 3.3.1)
$\mathcal{L}_d = \frac{1}{B}\sum_i ||ae_{x_G^i} - x_G^i||_1$
$\mathcal{L}_c = \frac{1}{B}\sum_i ||f_{x_G^i} - f_{x_S^i}||_2$
$\mathcal{L}_z = \frac{1}{B}\sum_i ||z_{x_G^i} - z^i||_2$
$\mathcal{L}_G = \mathcal{L}_d + w_c \cdot \mathcal{L}_c + w_z \cdot \mathcal{L}_z$

#Compute gradients and perform SGD update
$grad_{\theta_G} \leftarrow \nabla_{\theta_G}\mathcal{L}_G$
$\theta_G \leftarrow \theta_G - \lambda_G \cdot grad_{\theta_G}$

Algorithm A.2 Discriminator Update Step

for $i \in 1...B$:
  #Sample real image, noise and label vector
  sample $z^i \sim \mathbb{P}_z$
  sample $l^i \sim \mathbb{P}_l$

  #Generate and discriminate fake sample
  $x_G^i = \mathrm{G}(z^i, l^i)$
  $ae_{x_G^i}, \ h_{x_G^i} = \mathrm{D}(x_G^i)$

#Compute losses
$\mathcal{L}_D = \frac{1}{B}\sum_i ||ae_{x^i} - x^i||_1 - k_t \cdot \frac{1}{B}\sum_i ||ae_{x_G^i} - x_G^i||_1$

#Compute gradients and perform SGD update
$grad_{\theta_D} \leftarrow \nabla_{\theta_D}\mathcal{L}_D$
$\theta_D \leftarrow \theta_D - \lambda_D \cdot grad_{\theta_D}$

Algorithm A.3 Predictor Update Step

for $i \in 1...B$:
  #Sample real image, noise and label vector
  sample $z^i \sim \mathbb{P}_z$
  sample $l^i \sim \mathbb{P}_l$

  # Discriminate real sample
  $ae_{x^i}, \ h_{x^i} = \mathrm{D}(x^i)$
  #Generate and discriminate fake sample
  $x_G^i = \mathrm{G}(z^i, l^i)$
  $ae_{x_G^i}, \ h_{x_G^i} = \mathrm{D}(x_G^i)$
  #Predict label and noise vector from embedding
  $z_{x_G^i}, \ l_{x_G^i} = \mathrm{P}(h_{x_G^i})$

#Compute losses (baselLoss: normalized loss from section 3.3.1)
$\mathcal{L}_P = \frac{1}{B}\sum_i \ \mathrm{baselLoss}(l_{x_G^i}, \ l^i) \ + \ \frac{1}{B}\sum_i ||z_{x_G^i} - z^i||_2$

#Compute gradients and perform SGD update
$grad_{\theta_P} \leftarrow \nabla_{\theta_P}\mathcal{L}_P$
$\theta_P \leftarrow \theta_P - \lambda_P \cdot grad_{\theta_P}$

Appendix B

Additional Results

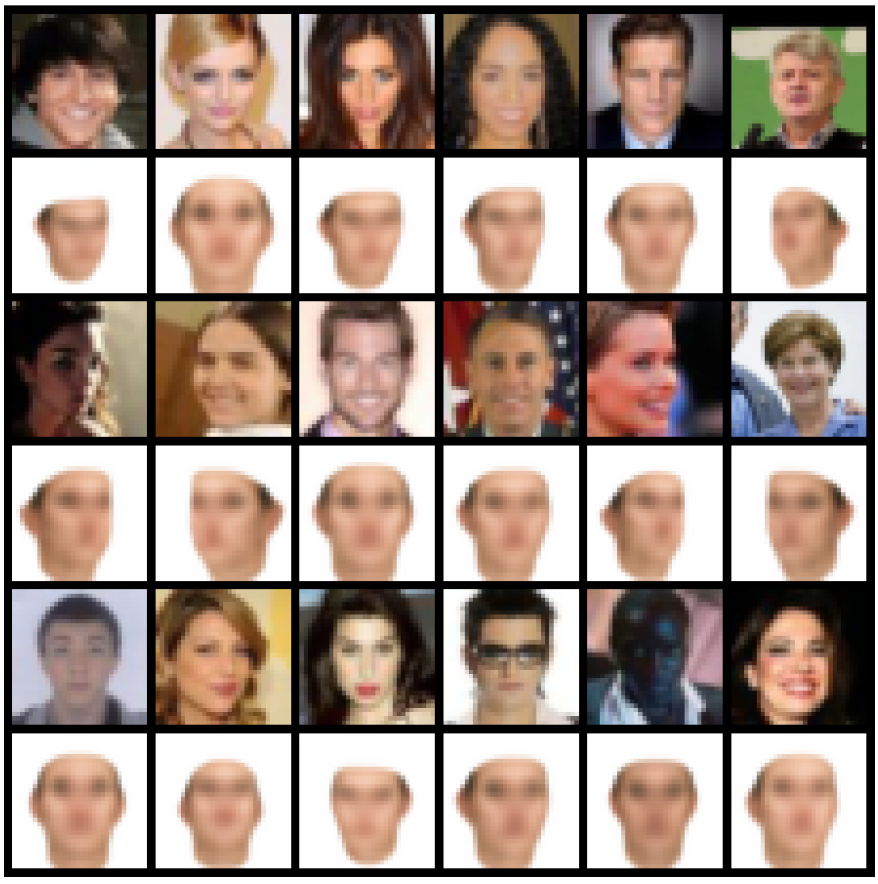

Figure B.1: Predictor Results - Real images above the related predicted face models produced by the simulator out of label vectors created by the predictor.

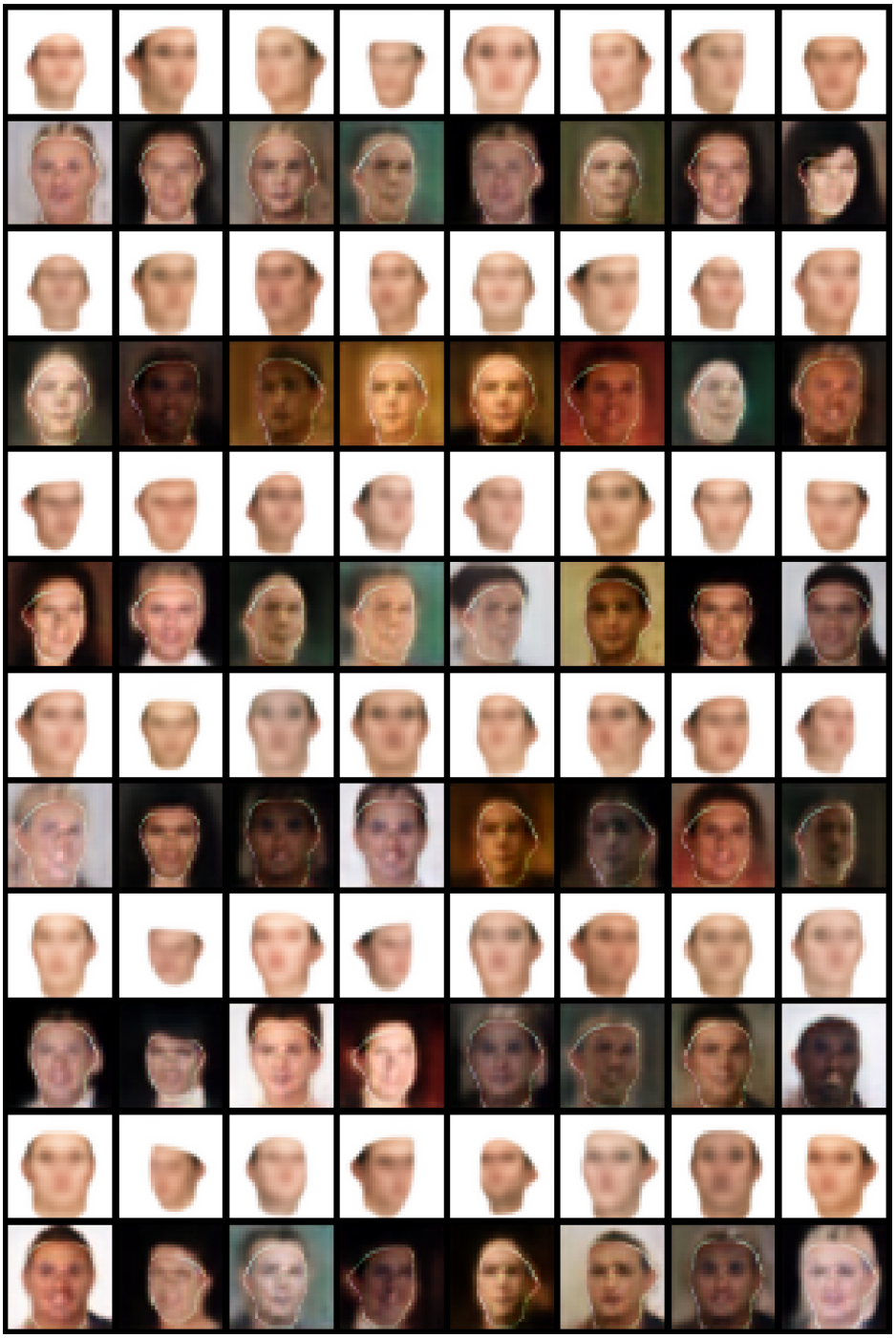

Figure B.2: Generator Results and Face Model Images - Generator results under the according simulated face model images.

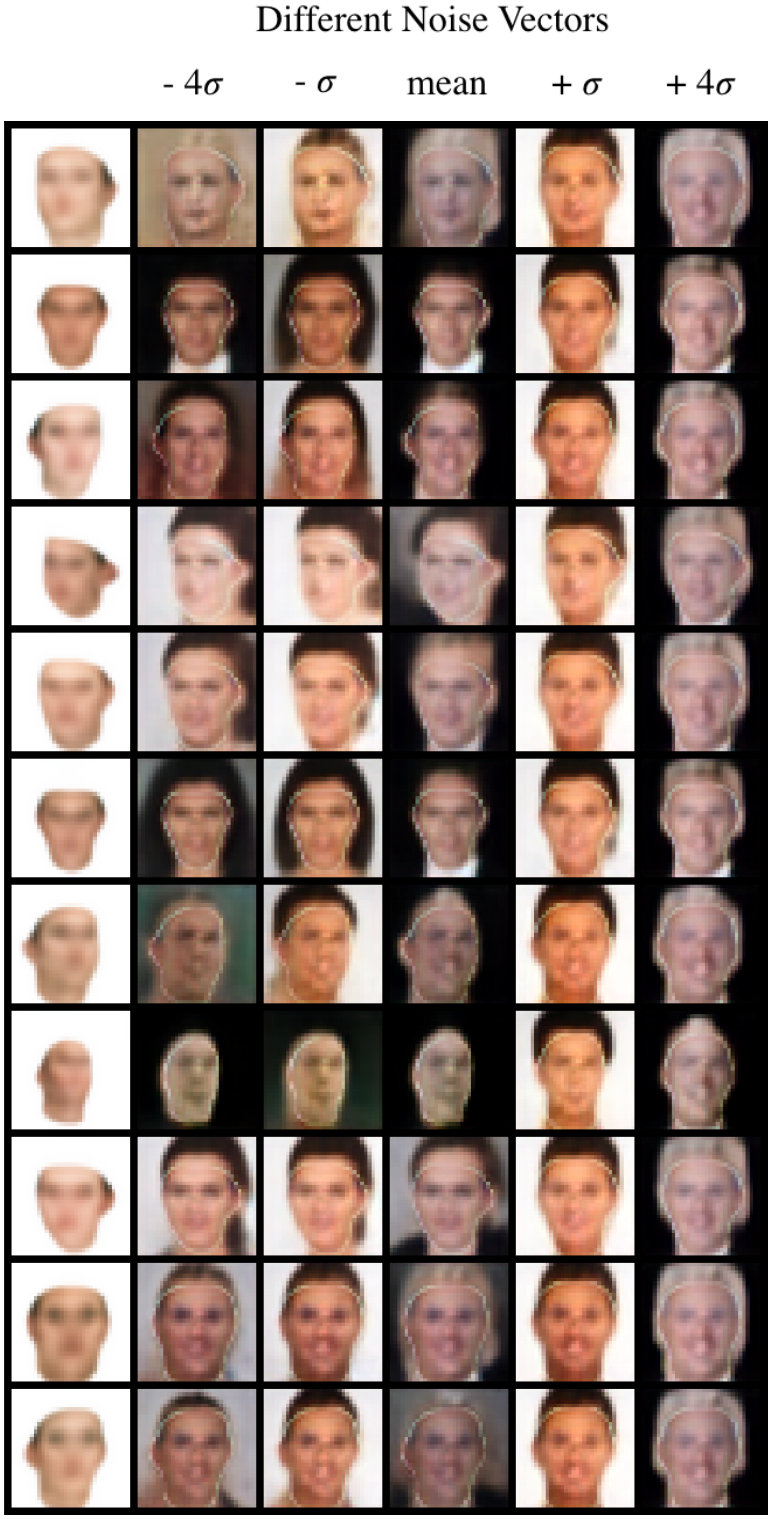

Figure B.3: Generator Results for fixed Label or Noise Vectors - Generator results for different noise vectors from left to right and for different label vectors from top to bottom.