Producing High Quality Face Morphs using GANs

Gerben Meijer

July 31, 2020

Contents

1 Introduction

Facial recognition systems are increasingly being used in places where the identity of a person has to be verified. One of these places is the border control post at the airport. Because of the important role of these systems, it is critical to evaluate their vulnerabilities to attacks. One way to attack these systems is by creating face morphs as described in [1]. This technique allows multiple persons to use the same travel documents for identification by creating an image that looks like all of these persons at the same time. This is done by generating a morph, an image that combines the facial features of multiple persons into one image of a face such that a facial recognition system accepts all persons involved as the morphed face. An example of morphing can be seen in Figure 1. The use of this method allows malicious actors to present themselves as a different person which could, for instance, allow a sought-after criminal to travel without getting caught at the border. Because of these security risks, it is important to improve the resilience of facial recognition systems against morphs.

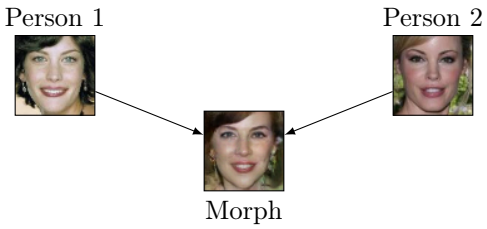


Figure 1: *An example of morphing between person 1 and person 2. In this case the morph is generated by a neural network.*

A way to improve the detection of morphs is to generate a large set of high-quality morphs that can be used to train facial recognition systems to be more robust to morphs. The manual method of creating face morphs, such as described in [1], can produce high-quality morphs but can take a lot of time per morph. This is not a problem for a malicious actor who could spend a day to manually edit the morph, but this is a problem if one wants to produce many high-quality morphs. Therefore a different method of morph generation is required. One promising method of morph generation is the use of generative neural network models such as generative adversarial networks (GAN) and variational autoencoders (VAE). In [2] a method that combines elements of these generative models is proposed to generate high-quality morphs. Similarly, in [3] another method for face generation and manipulation is proposed that combines GANs and VAEs.

In this paper, several improvements are proposed that aim to either change the morphing method or train the network to produce better morphs. The goal is to improve the morphs in such a way that they are closer to the input faces

according to a facial recognition system while remaining realistic to the human eye.

2 Background

This section introduces important background information that is in many cases necessary to understand the state of the art and the new ideas proposed in this paper.

2.1 Convolutional Neural Networks

Neural networks with convolutional layers will be used for this thesis. Since convolutions are not the main focus of this research and a well-written explanation would span multiple pages, an in-depth explanation is not provided here. In [4] an introduction is given to convolutions in general. For transposed convolutions [5] provides a good introduction.

2.2 Generative Adversarial Networks

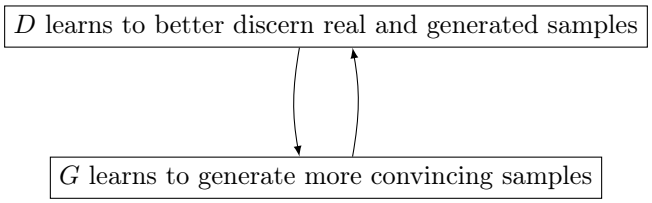


Figure 2: *The GAN training cycle*

Generative Adversarial Networks (GANs)[7] are a framework for training generative neural network models. A GAN consists of a generator network (G) and a discriminator network (D). G generates images from vectors containing random noise. The values in these so-called latent vectors are usually sampled from the standard normal distribution. D takes and image and outputs whether this image is from the dataset (real) or whether it is generated by G (fake). During training D is fed both real and fake images separately and it has to learn to discern the fake images from the real as well as possible. Figure 3 shows the flow of data through a GAN. Meanwhile G is trained to generate images such that D thinks that they are real. This creates the loop as shown in Figure 2 where both networks continuously improve. This cycle might lead one to believe that a GAN will eventually converge to near-perfect images, but unfortunately this is often not the case. GANs are often unstable and require careful tuning to get right.

In terms of network architecture, G usually consists of multiple transposed convolutional layers that slowly scale up the image from the input latent vector

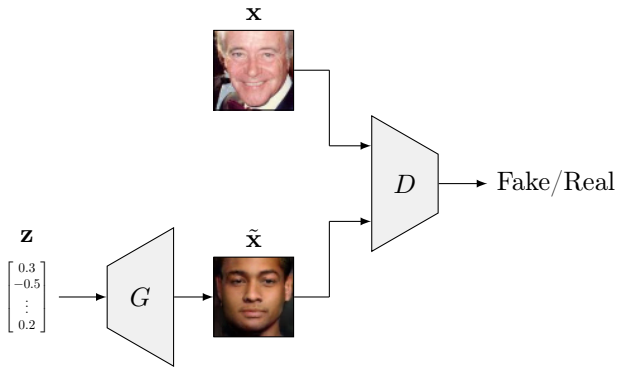


Figure 3: *A schematic of the data flow in a GAN. Note that the 2 arrows leading into D do not mean that D has 2 separate inputs but rather that both real and fake images are used as input during training. (images are from taken from the CelebA dataset[6] and from one of the trained models)*

z, which can be seen as a 1×1 pixels image with a number of channels equal to the size of the vector, up to the final output image which in this case is 64×64 pixels and 3 color channels. D is usually modelled as a convolutional neural network that takes these 64×64 pixel images and scales the image down until it finally ends up with one single output value that outputs whether the image appears real or fake.

2.3 VAE/GAN

The default GAN architecture is not able to reconstruct images, it is only able to generate new images. To generate morphs, another network is needed that "encodes" images into the latent space. This way it is possible to encode two input images, compute the mean latent vector and then generate the morph from that mean latent vector. In [3] a technique called VAE/GAN is introduced that combines variational autoencoders (VAEs) and GANs. This technique is able to encode images to a latent vector, decode images from a latent vector, as well as generate new images similar to a VAE. Due to the added GAN part it produces images of a substantially better quality than VAEs would. One of the main findings in the paper is the $\mathcal{L}_{\text{llike}}^{\text{Dis}_l}$ loss. A normal VAE uses a pixel-wise reconstruction loss to evaluate how well the model reconstructs and image when the image is first encoded to a latent vector which is then decoded back to an image. This setup often results in very soft and blurry images in places where there is much variation in the dataset (ie. the background behind faces. $\mathcal{L}_{\text{llike}}^{\text{Dis}_l}$ uses the activations of the l-th layer of the GAN discriminator instead of pixels, which greatly improves image quality. This method of measuring the difference between two images will be used in this thesis, where the activations of the l-th layer of the discriminator will be referred to as Dis_l.

2.4 Metrics for Face Recognition Systems

In order to evaluate a Face Recognition System (FRS) or a method that tries to fool an FRS, metrics are required to quantify their performance. A number of common metrics used in this field are discussed in this section.

2.4.1 FAR and FRR

The first two metrics are FAR and FRR, the **False Acceptance Rate** which measures the error of accepting a claim that is false and the **False Reject Rate** which is the proportion of truthful claims that have been denied. These metrics both apply to the authentication system.

2.4.2 MMPMR

The earlier introduced FAR and FRR are not well suited for morph evaluation. A morph is only effective if it matches all people involved in the morph. To see why FAR fails in this situation, assume that two morphing methods are being compared. The first morphing method takes two input images and returns the first one. The second morphing method produces perfect morphs 50% of the time and produces unrecognizable images the other 50% of the time. When both methods are evaluated over multiple image pairs, they will both be accepted 50% of the time and rejected 50% of the time. This is undesirable since only the second morphing method can produce useful morphs.

To improve the assessment of morphing techniques [8] introduces multiple new metrics. One of these is the **Mated Morph Presentation Match Rate** or **MMPMR**.

The MMPMR is computed over a set of M generated morphs. Each of these morphs m is a morph between N_m subjects. Between a morph and a reference photo of each of its subjects a similarity score S_m^n is calculated using some face recognition algorithm. These algorithms return the similarity score as a number representing the similarity between two faces. The threshold τ determines the minimal similarity score at which two faces are considered similar. A morphing attack only succeeds when all subjects are considered similar to the morph. Therefore only the similarity score of the subject that matches the morph the least has to be compared to the threshold to measure if a morphing attack was successful. The MMPMR counts the number of successful morphing attacks and divides this by the total number of morphs M. Formally, the MMPMR for similarity scores is defined as

$$\text{MMPMR}(\tau) = \frac{1}{M} \cdot \sum_{m=1}^{M} \left\{ \left[\min_{n=1,\ldots,N_m} S_m^n \right] > \tau \right\}$$

For this paper only $N_m = 2$ will be used, which simplifies the equation to

$$\text{MMPMR}(\tau) = \frac{1}{M} \cdot \sum_{m=1}^{M} \left\{ \left[\min(S_m^{(1)}, S_m^{(2)}) \right] > \tau \right\}$$

3 State of the Art

In MorGAN[2] the use of neural networks and more specifically GANs for morphing is introduced. This section introduces MorGAN and also briefly discusses LMA, the baseline used in MorGAN.

3.1 LandMark based Attacks (LMA)

MorGAN[2], which will be discussed later, mentions LMA as the standard method for automatically creating morphs. LMA does not use neural networks for morphing. Instead it finds facial landmarks and applies multiple operations to alirgn these features and morph them together between two images. The automatic method is described in a tutorial[9] on creating face morphs with OpenCV. LMA is not further evaluated in this thesis.

3.2 MorGAN

In MorGAN[2] an architecture for generating morphs is proposed that uses their novel MorGAN framework for training Deep Neural Networks to generate faces, which is then used to create morphs. Their technique is based on Adversarially Learned Inference (ALI)[10], which will be explained in this section as a part of MorGAN. Real images sampled from the training set are denoted as x and are sampled from $q(x)$, the distribution of real images. Latent vectors are denoted as z and are sampled from the distribution of latent vectors $p(z)$, which is in this case a standard normal distribution. In MorGAN the generators G_z and G_x are trained to generate a latent representation $\hat{z}$ from an image x and an image $\hat{x}$ from a sampled z respectively. The discriminator D learns to discern between $(x, \hat{z})$ and $(\hat{x}, z)$ samples. G_z is optimized to let D believe that the $\hat{z}$ it generated is actually a real z from $p(z)$ and G_x is optimized to let D believe that its $\hat{x}$ is a real x from $q(x)$. Additionally pixel-wise reconstruction loss is placed on the output of $G_x(G_z(x))$ to preserve identity. Figure 4 shows a diagram with the data-flow and losses in the MorGAN algorithm. Algorithm 1 shows the MorGAN training algorithm in pseudo-code.

In MorGAN the following procedure is proposed to generate a morph, assuming trained networks:

1. Take the 2 images, pre-process them and then calculate their latent representations using G_z. This gives you $\hat{z}_1$ and $\hat{z}_2$
2. Linearly interpolate them using $\hat{z} = (1 - \beta)\hat{z}_1 + \beta\hat{z}_2$ with $\beta = 0.5$.

3. Decode $\hat{z}$ using G_x to get the output image.

MorGAN utilises multiple loss functions that together combine into one loss function for the generator. $\mathcal{L}_{\text{GAN-G}}$ is the GAN loss for both G_z and G_x in MorGAN, it becomes higher as the $\tilde{x}$ generated by G_x or the $\hat{z}$ generated by G_z looks more fake to D than the real z or x accompanying them in the (x, z) tuple. This loss achieves 2 things. It assures that the generated $\tilde{x}$ and $\hat{z}$ match real samples as close as possible, and that the mappings of G_x and G_z become each other's inverse. This means that $G_x(G_z(x))$ should result in an $\tilde{x}$ that looks like $\tilde{x}$ and the same for z and $\hat{z}$. The GAN losses are defined such that this goal is achieved. For a batch of M samples they are defined as follows:

$$\mathcal{L}_{\text{GAN-D}} = -\frac{1}{M}\sum_{i=1}^{M}\log(D(x^{(i)}, \hat{z}^{(i)})) - \frac{1}{M}\sum_{j=1}^{M}\log(1 - D(\tilde{x}^{(i)}, z^{(i)}))$$

$$\mathcal{L}_{\text{GAN-G}} = -\frac{1}{M}\sum_{i=1}^{M}\log(1 - D(x^{(i)}, \hat{z}^{(i)})) - \frac{1}{M}\sum_{j=1}^{M}\log(D(\tilde{x}^{(i)}, z^{(i)}))$$

This part of MorGAN is the same as the ALI algorithm. It differs from the theoretical definition of the GAN loss, where $\mathcal{L}_{\text{GAN-G}}$ would simply be equal to $-\mathcal{L}_{\text{GAN-D}}$. Both in GAN [7] and ALI [10] this choice for the alternative $\mathcal{L}_{\text{GAN-G}}$ over $-\mathcal{L}_{\text{GAN-D}}$ is made because it provides better gradients when D outperforms G by a large margin.

Unfortunately this GAN loss alone is not enough to ensure that $G_x(G_z(x))$ preserves the identity of the person in the input image. Therefore $\mathcal{L}_{\text{pixel}}$ is added by MorGAN on top of ALI. $\mathcal{L}_{\text{pixel}}$ is an additional loss that measures the pixel-wise difference between the input image and the reconstruction $G_x(G_z(x))$. These losses are then combined in $\mathcal{L}_{\text{syn}}$, which is equal to $\mathcal{L}_{\text{GAN-G}} + \alpha\mathcal{L}_{\text{pixel}}$. α is a parameter used to influence the strength of $\mathcal{L}_{\text{pixel}}$ relative to $\mathcal{L}_{\text{GAN-G}}$. A larger α will cause $\mathcal{L}_{\text{pixel}}$ to have more influence and thus will theoretically lead to better reconstructions but possibly less realistic images.

ALI and MorGAN model G_z such that it outputs a probability distribution over the latent space rather than a single latent vector. The output of G_z is a mean vector and standard deviation or variance vector depending on implementation. Values are then sampled from normal distributions with these means and standard deviations or variances. By making the output of G_z stochastic, G_z can theoretically encode stochastic features in the dataset (like the exact positions of hair or the way clouds in the background are shaped). It also allows G_z to go from a distribution with a finite number of samples ($q(x)$) to a distribution with infinite samples ($p(z)$) while still being able to cover every possible value in the latent space. Whether this stochasticity of G_z is actually necessary is up to debate, as BiGAN [11] proposes a similar architecture as ALI without the stochasticity and does not appear to have problems because of this omission.

In order to allow for back-propagation of the gradients from the GAN and reconstruction losses through this sampling operation, the so-called reparameterization trick is used. The idea of the reparameterization trick is to sample from a standard normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and then re-scale it to $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I})$ using the outputs of the encoder. The equation for sampling then becomes

$$\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \quad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

Where $\odot$ denotes element-wise multiplication between two vectors. $\boldsymbol{\epsilon}$ is in this case a vector with the same number of dimensions as the latent vector. Now it is possible to back-propagate the error through $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

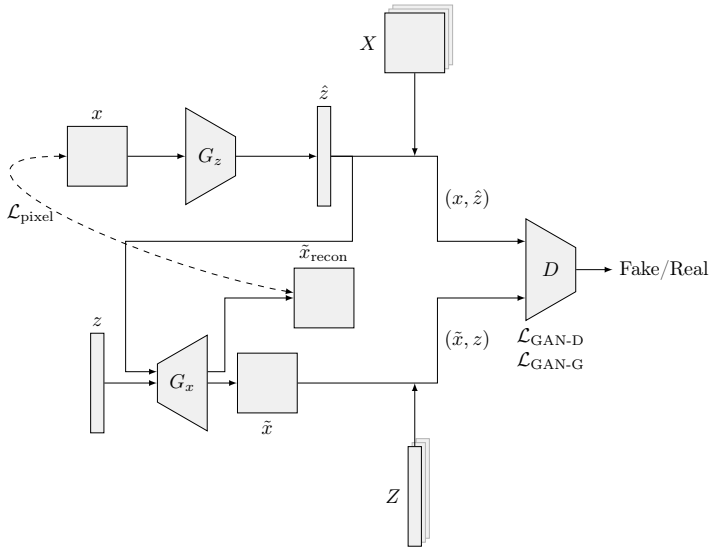


Figure 4: *A schematic of the MorGAN structure during training. This diagram shows the flow of data through the different networks. Multiple arrows into one network do not indicate that a network takes 2 simultaneous inputs, but rather that the model is fed both inputs separately during training. The tuples that are fed into D do denote that both elements in the tuple are given to D at the same time. This diagram is remade after the same diagram in [2].*

Algorithm 1: MorGAN algorithmic training procedure

$\theta_G, \theta_D \leftarrow$ randomly initialize network parameters

repeat

$x^{(1)}, \ldots, x^{(M)} \sim q(x)$ `// Sample M images from the data distribution`

$z^{(1)}, \ldots, z^{(M)} \sim p(z)$ `// Sample M vectors from a Normal distribution`

$\hat{z}^{(i)} \leftarrow G_z(x^{(i)}), i = 1, \ldots, M$ `// Compute` G_z `generation`

$\tilde{x}^{(j)} \leftarrow G_x(z^{(j)}), j = 1, \ldots, M$ `// Compute` G_x `generation`

$\tilde{x}^{(i)}_{\text{recon}} \leftarrow G_x(G_z(x^{(i)})), i = 1, \ldots, M$ `// Compute reconstructed` x

$\rho_q^{(i)} \leftarrow D(x^{(i)}, \hat{z}^{(i)}), i = 1, \ldots, M$ `// Compute Discriminator predictions`

$\rho_p^{(j)} \leftarrow D(\tilde{x}^{(j)}, z^{(j)}), j = 1, \ldots, M$

`// Compute discriminator loss`

$\mathcal{L}_{\text{GAN-D}} \leftarrow -\frac{1}{M}\sum_{i=1}^{M}\log(\rho_q^{(i)}) - \frac{1}{M}\sum_{j=1}^{M}\log(1-\rho_p^{(j)})$

`// Compute generator loss`

$\mathcal{L}_{\text{GAN-G}} \leftarrow -\frac{1}{M}\sum_{i=1}^{M}\log(1-\rho_q^{(i)}) - \frac{1}{M}\sum_{j=1}^{M}\log(\rho_p^{(j)})$

$\mathcal{L}_{\text{pixel}} \leftarrow \frac{1}{W \times H}\sum_{k=1}^{W}\sum_{l=1}^{H}\left|x^{(k,l)} - \tilde{x}^{(k,l)}_{\text{recon}}\right|$

$\mathcal{L}_{\text{syn}} \leftarrow \mathcal{L}_{\text{GAN-G}} + \alpha\mathcal{L}_{\text{pixel}}$

$\theta_D \leftarrow \theta_D - \nabla_{\theta_D}\mathcal{L}_{\text{GAN-D}}$

$\theta_G \leftarrow \theta_G - \nabla_{\theta_G}\mathcal{L}_{\text{syn}}$

until *convergence*

4 Research Questions

To what extent can the quality of morphs generated by MorGAN, as measured by a facial recognition system, be improved by

1. using a feature-based reconstruction loss similar to the one used in VAE/GAN instead of the pixel-wise reconstruction loss used by MorGAN?
2. adding a morph loss to the existing losses?
3. further optimizing the morphs using gradient descent?
4. using a neural network to morph two latent vectors instead of using the mean latent vector?

5 Global Approach

This section introduces the possible improvements introduced in the research questions in more detail while staying high-level. The exact implementation used will be discussed in the methodology section.

5.1 Reconstruction Loss Similar to VAE/GAN

In MorGAN the reconstruction loss is computed using an L1 loss between the input image and the reconstruction of that image. Pixel-wise losses like this one often make the image softer. This also happens in MorGAN if the reconstruction loss scaling factor α is increased. In order to improve the face reconstruction performance of MorGAN it would be beneficial to be able to increase α without making the image softer. A possible way to achieve this is to apply a loss similar to the $\mathcal{L}_{\text{llike}}^{\text{Dis}_l}$ loss in VAE/GAN. This loss would be used as an alternative way of computing the reconstruction loss in MorGAN. In this loss the output of the l-th layer in the discriminator is computed and the difference between those activations is used as the error. Since the discriminator learns to discern real and fake images, these activation maps might contain useful features. By comparing the images this way the hope is that facial features are mostly forced to be similar while less important information like the background receives less attention.

5.2 Morph Loss

The ultimate goal of face morph generation is to generate an image of a face that looks like the same person as all input persons, while also not looking like a generated image. GANs are optimized to do the latter, as they aim to generate realistic images. ALI introduces an encoder that is also trained using the GAN loss, but it often fails to reconstruct the image in such a way that the identity of a person is retained. MorGAN improves this by adding a reconstruction loss on top of ALI. This greatly improves the reconstruction performance, but might not necessarily improve morphing performance since it is not guaranteed that a taking the mean between two latent vectors will result in a face with the "mean identity" according to an FRS.

In order to potentially improve morphing performance, a *morph loss* is added to the default MorGAN algorithm which aims to directly optimize morphing performance of the model. Each training step the model will generate a morph in the same way as it is done when generating a normal MorGAN morph. This morph is generated between the images in the batch and a batch of random images from the train set. Then the morph loss will evaluate how close this morph is to the input images. In a perfect world this morph loss would be the same as the distance metric used for computing the MMPMR. This way the MMPMR can be optimized directly. However, using a full FRS each training step does come with a large computational cost. Using a pixel-wise distance measure is computationally efficient, but also forces the network to morph the images on a pixel level instead of only morphing the facial features. In preliminary experiments this had a large adverse effect on image quality. As a trade-off between computational cost and similarity to the MMPMR, a distance measure based on the $\mathcal{L}_{\text{llike}}^{\text{Dis}_l}$ from the VAE/GAN paper is chosen for this thesis. The exact implementation of this morph loss will be further discussed in the methodology

section.

5.3 Alternatives to Taking the Mean of Two Latent Vectors

Up until now the latent vector of the morph is always computed using the mean of two latent vectors. There is no guarantee that this method of morphing is optimal for the latent encoding learned by the model. It may, therefore, be wise to explore different methods for selecting the morph latent vector. These two approaches both aim to explore whether the morphing method itself can be improved.

5.3.1 Using Gradient Descent

In order to further improve the morphing performance of the trained models, this approach aims to further optimize the morphs and reconstructions made by fully trained models. This is done by optimizing the latent vectors with respect to the reconstruction and morph losses for a single pair of input images. This greatly increases the duration of generating a single morph, but might also yield an increase in morphing performance.

5.3.2 Using a Neural Network

Another possible method of improving the morphing process is training an extra neural network that is given two latent vectors and produces a morphed latent vector. This morphing network is trained using the morph loss and can either be trained as part of the MorGAN with morph loss training loop or it can be trained for a fully converged model. A morph network like can introduce instability into the training process and can also lead to a biased morphing process where one latent vector is favored over the other. In the methodology section the challenges and design choices will be addressed in more detail.

6 Methodology

This section describes how the possible improvements presented in the previous section will be trained, evaluated and what design decisions were made for the implementation.

6.1 Dataset

A large dataset of faces is required in order to be able to train the networks. Similar to MorGAN the CelebA dataset[6] was chosen. The CelebA dataset consists of pictures with a large variety of poses and backgrounds. It also contains information about the identity of all images in the dataset. The number of unique identities in the dataset is a lot smaller than the number of images in the dataset and thus many images are of the same identity. This is useful

since at least 2 images are required for proper evaluation, which will be further explained in the evaluation section.

6.1.1 Cropping and Alignment

The task of face generation can be made significantly easier by aligning faces. The faces in the CelebA are already aligned, but it proved to be hard to reproduce the exact face alignment method used by the CelebA dataset. In order to be flexible in terms of dataset and allow for the use of images from other sources as well, the images were aligned and cropped using Dlib [12]. Specifically `shape_predictor_5_face_landmarks.dat`[13] was used to align the images using the `dlib.get_face_chip` method. This method also cropped and resized the images to 64×64 pixels. The cropping cuts away a large amount of the background of the image which is irrelevant for the task of morphing. Some images before and after these steps can be found in Figure 5.

Figure 5: *Comparison of normal CelebA images (top) and the same images after the Dlib crop and align procedure (bottom)*

6.1.2 Filtering

The CelebA dataset is a dataset consisting of a diverse set of poses. Many faces are sideways, which is not the type of image usually used for passports. Therefore all images that are not in a frontal pose have been filtered out of the dataset. For this purpose the alignment data provided by the CelebA dataset itself is used. For the pose detection, the positions of the nose and eyes are used. First they are normalized to be between 0 and 1 using the width and height of the image. Then the distances from the left and right eyes to the nose are computed. The absolute difference between these distances is used as a measure for how sideways the face is. In order to stay close to the methodology of MorGAN, the threshold was chosen such that the resulting dataset size was close to the $103,480$ reported in MorGAN. This resulted in a threshold of 0.0288924 and a final dataset size of $102,025$. Any image with an absolute difference below or equal to this threshold is included.

6.1.3 Partitioning

A test and validation set of unseen images are required to be able to evaluate the networks. CelebA does come with a predefined partitioning between train,

validation and test sets. This partitioning, however, does put images of the same identity into different splits. In order to guarantee that the test and validation set only contain unseen identities, a new partitioning is made. Some identities have a large number of images attached to them, sometimes even more than 30. During evaluation every identity is used only once, so that would waste a large number of images that could have been used for training. To alleviate this problem, only identities with 3, 4, 5, or 6 images attached to them will be included in the validation and test sets. Initially the test set was chosen to contain 1000 unique identities and the validation set was chosen to contain 500 unique identities. In the evaluation section an additional filtering step will be introduced that cuts down these numbers to 632 unique identities in test and 346 unique identities in the validation set.

6.2 Training

The train set is used to train all the models. This training is done in many runs over the entire train set, such a complete run over the train set is often referred to as an *epoch*. Since the entire dataset usually does not fit into GPU memory, training is done in (mini)batches. The gradients computed for a mini-batch are an approximation of the gradients for the entire dataset, but much cheaper to compute. A different batch size appears to have many effects on training and the optimal configuration is still up for discussion. In this case, the training algorithm samples batches of size 65 from the shuffled train set, loads the respective images from disk and transforms the entire batch to a tensor with 32-bit floating point pixel values between 0 and 1. This batch of real images becomes a tensor with 65 images, consisting of 3 color channels and a resolution of 64×64. In many examples in the thesis, examples are given without the batch dimension. This is done to ensure readability, everything that has to do with training a network or latent vector is done batch-wise.

What is done with these batches of real images is depends on the algorithm. The default MorGAN algorithm serves as the basis for all other implementations except for the latent vector optimization, where the models are already trained beforehand. All models are built using the same structure, which can be found in Table 6. This layout was chosen to stay consistent with ALI and MorGAN. Better possible layouts have since been discovered, as will be alluded to in the Future Work section, but these are not relevant since the main goal is to improve the algorithm regardless of model layout.

Models are trained mainly on a NVIDIA 1080 TI graphics card, where training takes anywhere between one and three days depending on the chosen algorithm and settings. For rapid testing of improvements, most ideas were not immediately tested on 64×64 images, but rather on a smaller resolution training set of 28×28 images. This dataset is simply acquired by resizing the images in the normal dataset to 28×28 when loading them from disk. This is obviously not representative of the final data-set and can also not be used to estimate val-

ues like the MMPMR. It does allow for quick stability and bug testing and for estimating some hyperparameters. As noted before, all final models are trained and evaluated on the 64×64 data-set.

6.3 Evaluation

As stated in the introduction, the goal of this research is to improve the similarity between the morph and the input identities as measured by a facial recognition system. The performance will be measured using the $\mathbf{MMPMR}(\tau)$ over a selection of pairs in the test-set. As specified in [8], the images of a subject used to create a morph should not be used for the evaluation of the same morph. The `face_recognition`[14] library is used as the facial recognition system for evaluation. The euclidean distance between the FRS embeddings of 2 images is used as a dissimilarity metric. Two images with a euclidean distance lower than 0.6 are considered to be the same person, $\tau = 0.6$. All face encodings are computed using the option `num_jitters` = 1.0. According to the `face_recognition` documentation[14] increasing this value makes the facial recognition system more accurate, but the jittering process also introduces randomness in the results which is undesirable.

6.3.1 Input and Reference Image Selection

For each identity in both the validation and test set, a pair of input and reference image are picked. The input image is used as input for the morphing process and is denoted as x_n. The reference image is used for comparison when computing the **MMPMR** as well as all the other results. It is denoted x_n^{ref}. An identity in the test or validation set may have up to 6 images of that identity linked to it. For each image the FRS embedding is computed using the `face_recognition` library. All possible pairs of images for this identity are tried and the first pair with a euclidean distance less than 0.4 is picked. This is done to ensure that the reference image actually looks like the input person. This measure is aimed to ensure that the chosen pair does actually look like the same person. Since there are duplicate images in CelebA, a pair with a distance of 0 is not considered in this process. Identities that do not have any pair that fits the requirements are filtered out.

6.3.2 Pair Selection

Every identity in the validation and test now has an x_n and x_n^{ref}. In this section the selection of pairs of identities will be discussed. The pairs of identities used to compute the **MMPMR** are chosen such that they are close together. This is similar to what was done in the MorGAN paper. If one were to use a morphing attack, the idea is that they would be able to select someone else who already looks like them in order to ease the morphing process.

Generation of these pairs is done as follows. As long as there are enough identities that are not yet used in a pair, a random identity is picked from the set of unused identities. For all other unused identities the distance to this identity is computed. The distance is defined as the euclidean distance between the input images. The closest pair is then added to the set of pairs and removed from the set of unused pairs. The picking algorithm continues like this until there are no more pairs left to form. Any pair with a euclidean distance that is not smaller than one will be discarded to ensure that the difference between 2 identities does not become too large. The choice for this value is admittedly arbitrary and was chosen because it works. There might be more optimal choices for this threshold.

6.3.3 Reported Metrics

In this section all metrics reported in the results section will be explained. During testing $\hat{z}$ is taken to be the mean of the output distribution of G_z, as opposed to being sampled. This is done to achieve consistent results.

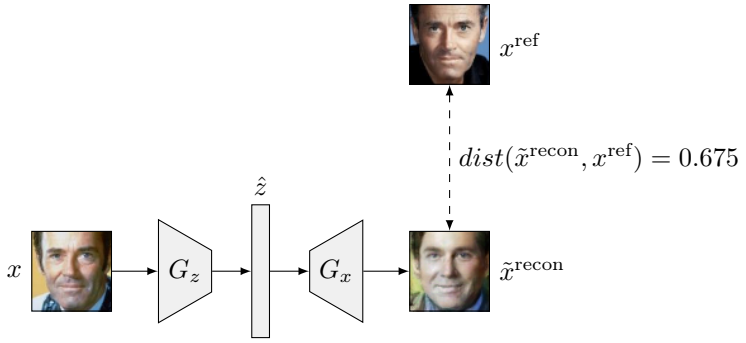


Figure 6: *A schematic of the reconstruction distance calculation for one person. Subscript n has been kept out for readability. G_x and G_z are the generator networks from a trained model. The reconstruction distance is computed using an FRS and is used for the reconstruction rate and the mean reconstruction distance over the validation and test sets.*

To measure the reconstruction performance of the models, two metrics are used. For both metrics the reconstruction distance $dist(\tilde{x}^{recon}, x^{\text{ref}})$ is used. *dist* denotes the euclidean distance between the FRS encoding of both input images. In Figure 6 the reconstruction process is shown. The first metric used for model evaluation is the reconstruction rate (RR):

$$\text{Reconstruction rate} = \frac{1}{N} \cdot \sum_{n}^{N} \left\{ dist(\tilde{x}_n^{recon}, x_n^{\text{ref}}) < 0.6 \right\}$$

Where N is the amount of images in the validation or test set. The subscript n denotes the n-th pair of (x, x^{ref}) from the validation or test set. The reconstruc-

tion rate measures the fraction of reconstructed images that looks enough like a reference image according to the facial recognition system. This metric can be used to evaluate whether the model manages to correctly reproduce the identity of the given person. For further evaluation of the reconstruction performance the mean reconstruction distance (MRD) is also computed.

$$\text{Reconstruction distance} = \frac{1}{N} \cdot \sum_{n}^{N} \left\{ dist(\tilde{x}_n^{recon}, x_n^{\text{ref}}) \right\}$$

For morphing there are also two metrics that are used to evaluate the performance of the model. The first metric used is the MMPMR. In order to stay consistent with the notation used above, the notation used here is different than the notation used in the MMPMR equation in the background section. $x_{m,1}^{\text{ref}}$ indicates the reference image of the 1st subject of morph m and $\tilde{x}_m^{\text{morph}}$ represents the generated morph image for morph m. The equation for MMPMR has already been discussed, but for clarity it is expressed once more in this notation:

$$\text{MMPMR} = \frac{1}{M} \cdot \sum_{m=1}^{M} \left\{ \max \left(dist(\tilde{x}_m^{\text{morph}}, x_{m,1}^{\text{ref}}), dist(\tilde{x}_m^{\text{morph}}, x_{m,2}^{\text{ref}}) \right) < 0.6 \right\}$$

In Figure 7 a diagram can be found that shows what is being measured. Additionally the mean morphing distance (MMD) is also denoted in the results. This is simply the mean distance between all morphs and all of their respective

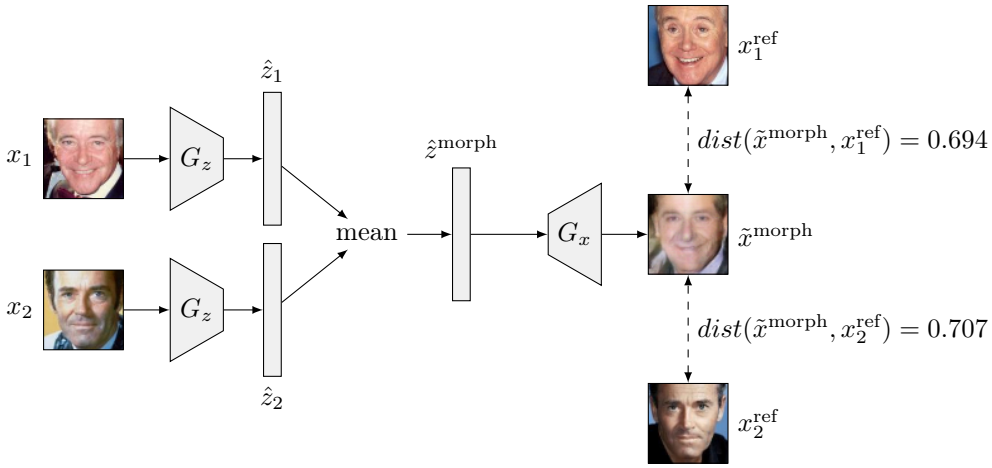


Figure 7: *A schematic of the morph distance calculation for a single morph. The subscript m has been left out for readability as all images in this diagram belong to the same m. G_x and G_z are the generator networks from a trained model. The reconstruction distance is computed using an FRS and is used for the MMPMR and the mean morph distance over the validation and test sets.*

reference images:

$$\text{Mean morph distance} = \frac{1}{M} \cdot \sum_{m=1}^{M} \left\{ 0.5 \cdot \left(dist(\tilde{x}_m^{\text{morph}}, x_{m,1}^{\text{ref}}) + dist(\tilde{x}_m^{\text{morph}}, x_{m,2}^{\text{ref}}) \right) \right\}$$

6.4 Implementation

This section contains the most important details concerning the actual implementation and design choices of the baseline and the proposed ideas. Everything is implemented in the Python programming language using the PyTorch[15] deep learning library. The code is available at `https://github.com/Gerryflap/master_thesis`.

6.4.1 Implementation differences wrt. the MorGAN and ALI paper

Implementing MorGAN using the details presented in the MorGAN[2] and ALI[10] papers resulted in an implementation that would diverge before 123 epochs were reached. An example can be found in Figure 8. Additionally, some parts of the ALI and MorGAN paper did not seem to align with the provided code. In this section, the known differences between both papers and the implementation used for this thesis will be highlighted. Most of these improvements are inspired by an existing ALI implementation in PyTorch[16].

Figure 8: *An example of divergence in earlier versions of the MorGAN implementation. On the left is the G_x output for a random z after the 29th epoch, on the right after the 30th epoch for the same z.*

Limiting the Number Of Discriminator Updates The divergence problem seems to be caused by D either becoming too strong or by D overfitting on the real images, which yields a useless training signal to G_x and G_z. ALI and MorGAN utilize a non-saturating GAN loss which should still provide useful gradients when D becomes too strong, but in this case it appears that this does not solve the problem. A simple way to stop this from happening is to stop updating D when it gets too far ahead. This trick proves to be very effective in this instance and is therefore used in all algorithms described in this paper. To be more precise, the step $\theta_D \leftarrow \theta_D - \nabla_{\theta_D} \mathcal{L}_{\text{GAN-D}}$ in Algorithm 1 is only performed when $\mathcal{L}_{\text{GAN-G}} < 3.5$. G_x and G_z are always updated.

Dropout Another difference with the ALI paper is the application of dropout. In the official code for ALI, dropout is applied to the $D(x)$ part of the discriminator as well ([17] `experiments/ali_celeba.py` line 127). In the paper this is

not the case. For all thesis experiments, dropout is applied as it is done in the ALI code instead of the way it is done in the paper. See Table 6 for network layout including dropout.

Biases Although not noted in the ALI or MorGAN paper, ALI and probably also MorGAN use a so-called untied bias before the output activation function of G_x. This effectively means a bias of dimensions $(3, 64, 64)$ is added to the output of the last layer. This bias is initialized such that the sigmoid of that bias outputs a "mean face". In the thesis implementation this is implemented as the mean image of the first batch. Applying this trick seems to speed up initial training.

MorGAN Reconstruction Loss The reconstruction loss was changed in scale compared to the equation in the MorGAN paper. In MorGAN, the reconstruction loss is the sum of absolute differences between pixel values, which is then divided by the width and height of the image. In the paper it is not described what is done with the batch dimension and the channel dimension. For all implementations in this thesis, the reconstruction loss is also divided by the batch dimension and the channel dimension. This makes it equal to the mean absolute error. Given the batch size (65) and the number of channels (3) this could be a scale change of $65 \cdot 3 = 195$ times smaller.

Improved Hyperparameters Since the implementation for this thesis most likely differs from the implementation used for MorGAN, tuning the hyperparameters might yield better performance. A number of preliminary experiments were conducted and some hyperparameters were changed. The dropout rate was decreased from 0.2 to 0.03 and the latent space dimension was increased from 256 to 512. When not listed, α is set to 0.3 as in MorGAN. The results for the improved hyperparameters are listed in the results section.

6.4.2 Dis_l reconstruction loss implementation

The Dis_lloss implementation in this paper deviates from the one used in VAE/GAN[3]. This difference exists to better match the default loss structure of MorGAN.

As stated in the global approach section, $\text{Dis}_l(x)$ is computed by computing the output activations of layer l in the discriminator for image x. The output tensor $\text{Dis}_l(x)$ is of the shape (N, C, H, W) with N being the batch size, C being the number of output channels of the l-th layer of D, H being the height of the output in pixels and W being the width. During training, $\mathcal{L}_{\text{recon}}$ is computed in the following way when the Dis_lversion of the reconstruction loss is used:

$$\mathcal{L}_{\text{recon}} = \frac{\sum_{n=1}^{N}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W}\left(\text{Dis}_l(x)_{n,c,h,w} - \text{Dis}_l(\tilde{x}_{\text{recon}})_{n,c,h,w}\right)^2}{N \times C \times H \times W}$$

or shorter

$$\mathcal{L}_{\text{recon}} = \text{MSE}(\text{Dis}_l(x), \text{Dis}_l(\tilde{x}_{\text{recon}}))$$

where MSE stands for Mean Squared Error. In the hyperparameter overview (Table 6), the choice of l for all experiments conducted in this paper can be found. This l is chosen to be after the same amount of downsamples as the choice of l in VAE/GAN. Further experiments regarding the choice of l could be interesting, but have not been performed here.

6.4.3 MorGAN with Morph loss

The morph loss was introduced in the global approach section as a possible addition to MorGAN that would improve morphing performance. Unless specified otherwise, this morph loss uses the Dis_l-based feature-wise distance which is discussed in detail in the section above. Based on the description of the morph loss up until now there are many possible ways of implementing it. The implementation below was chosen based on what seemed the most sensible, as it was impossible to evaluate many different possible implementations on top of the number of experiments already performed for this thesis.

As stated before, the morph loss is based on the distance between $\text{Dis}_l(x_1)$ and $\text{Dis}_l(\tilde{x}_{\text{morph}})$ and between $\text{Dis}_l(x_2)$ and $\text{Dis}_l(x_{\text{morph}})$. Similarly to the Dis_lreconstruction loss the distance metric chosen for this is the mean squared error (MSE). Now there are two losses that are to be minimized, the distance between $\text{Dis}_l(x_1)$ and $\text{Dis}_l(\tilde{x}_{\text{morph}})$ and the distance between $\text{Dis}_l(x_2)$ and $\text{Dis}_l(x_{\text{morph}})$. The morph loss is defined as the mean of these 2 distances. The mean is chosen over the max of the two distances here because the max only provides information about the largest distance, which results in throwing away the other distance. Since the max might continuously switch during training, it could lead to a more unstable training process. These are all hypotheses that are not further evaluated, it is possible that further research proves these assumptions wrong. The equation below shows the full morph loss as described above.

$$\begin{aligned}&\text{morph\_loss}(x_1, x_2, \tilde{x}_{\text{morph}}) = \\ &\quad 0.5 \cdot \text{MSE}\big(\text{Dis}_l(\tilde{x}_{\text{morph}}), \text{Dis}_l(x_1)\big) + 0.5 \cdot \text{MSE}\big(\text{Dis}_l(\tilde{x}_{\text{morph}}), \text{Dis}_l(x_2)\big)\end{aligned}$$

In order to compute the morph loss, the images $\tilde{x}_{\text{morph}}$, x_1, and x_2 are required. The x used in the normal MorGAN algorithm is used as x_1. x_2 is then picked to be a random image from the dataset belonging to a different identity. Next $\hat{z}_1 \sim G_z(x_1)$ and $\hat{z}_2 \sim G_z(x_2)$ are computed and the mean of these two latent vectors is taken to get $\hat{z}_{\text{morph}}$. This finally results in $\tilde{x}_{\text{morph}} = G_x(\hat{z}_{\text{morph}})$.

In Algorithm 2 a more formal description of the extended algorithm can be found.

The morph loss is weighted by α_{morph}, similar to α for the MorGAN reconstruction loss. The evaluated models with morph loss are trained with both α and α_{morph} equal to 3. By default the weights of both G_x and G_z are updated using the morph loss gradients. Additionally configurations with the morph loss only applied to G_x and G_z are also trained and listed in the results section.

6.4.4 Gradient Descent on z_{morph}

As specified in the global approach section, this idea attempts to increase morphing and reconstruction performance of already trained models by optimizing the morphs and reconstructions instead of the model weights. This is done by optimizing the latent vectors to minimize the reconstruction loss (for $\hat{z}_1$ and $\hat{z}_2$) or morph loss (for $\hat{z}_{\text{morph}}$). In this section the details of this experiment are discussed.

Given are the input images x_1 and x_2, as well as trained G_z, G_x and D networks. For clarity, x_1 and x_2 are the input images for the two persons of whom a morph is to be made. Similar to the normal morphing method, $\hat{z}_1 \sim G_z(x_1)$, $\hat{z}_2 \sim G_z(x_2)$ and $\hat{z}_{\text{morph}} = \frac{\hat{z}_1 + \hat{z}_2}{2}$ are computed. These vectors are used as a starting point for further optimization. $\hat{z}_{\text{morph}}$ is trained to minimize $\text{morph\_loss}\big(x_1, x_2, G_x(\hat{z}_{\text{morph}})\big)$. $\hat{z}_1$ and $\hat{z}_2$ are trained to minimize $\text{recon\_loss}\big(x_1, G_x(\hat{z}_1)\big)$ and $\text{recon\_loss}\big(x_2, G_x(\hat{z}_2)\big)$ respectively.

Training of these latent vectors is done in batches of 32 morphs at a time. Each batch is optimized for 500 training steps using the Adam optimizer with a learning rate of 0.01 and the default beta values for PyTorch (0.9 and 0.999). For both reconstruction loss and morph loss the Dis_l variant of the loss is used. The weights of the models are frozen and only the latent vectors are optimized. The evaluation of this method is done over the entire test set, which takes nearly an hour on an NVIDIA GTX 1060 6GB under these parameters. Hyperparameters were chosen by running experiments on the validation set.

6.4.5 Morph network

The morph network is introduced in the global approach section as a possible method of increasing morphing performance. In Figure 9 a schematic is shown of the morphing process when using this morphing network. There are many possibilities to consider for this morph network. It can be trained during or after the training of the other networks, it can have many different architectures, normalization can be used but might not work well, multiple regularization losses might be necessary to control it, and the list goes on. Exploration of all these possible options can be a whole thesis on its own, so for this experiment a number of choices have been made to reduce the search space.

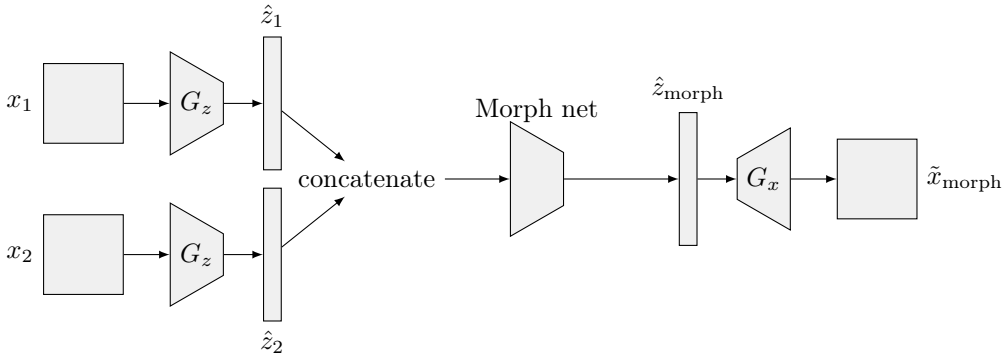


Figure 9: *A schematic of the morph generation process when using a morphing network. "Morph net" denotes the morphing network. The concatenate operation concatenates the two n-dimensional latent vectors together to form one $2n$-dimensional vector.*

The chosen network architecture for the morph network is a simple feed-forward neural network with two hidden layers. The first and second layer have 1024 outputs. The last layer has 512 output neurons to match the latent size used for this experiment. A full description of the architecture can be found in Table 7 in the Appendix.

The morphing network evaluated in the results section is trained together with the other networks in a similar way as the MorGAN with morph loss algorithm. Instead to generating a morph by taking the mean latent vector, the morphing network is used. In order to improve stability a number of additions and changes are made to the training algorithm, losses and hyperparameters.

The first addition is a pre-training phase where the morphing network is trained to output the mean of the two latent vectors. This is done by sampling pairs of latent vectors z_1 and z_2 from a standard normal distribution. The model is then trained to output $\frac{z_1+z_2}{2}$ using a MSE loss function. This is done for $50,000$ batches of 64 pairs of latent vectors. Parameters are optimized using the Adam optimizer and a learning rate of 0.0001.

When the normal morph loss is used, the morphing network learns to output either $\hat{z}_1$ or $\hat{z}_2$ all the time. In order to solve this problem, a redefined morph loss is used. The redefined morph loss is defined as

$$\begin{aligned}&\text{morph\_loss}_{\max}(x_1, x_2, \tilde{x}_{\text{morph}}) = \\ &\qquad \max\Big(\text{MSE}\big(\text{Dis}_l(\tilde{x}_{\text{morph}}), \text{Dis}_l(x_1)\big), \text{MSE}\big(\text{Dis}_l(\tilde{x}_{\text{morph}}), \text{Dis}_l(x_2)\big)\Big)\end{aligned}$$

This way, the morph loss is always equal to the highest of the two reconstruction losses. Whenever the morph is biased towards one input it will always be pulled towards the other input again. During the morphing network experiment an α and α_{morph} of 3.0 are used. The morph loss is not applied to G_x in this experiment to improve stability.

A further step to improve stability is the addition of the *morph consistency loss*. This loss is added to ensure that morphing any $z \sim p(z)$ with itself will also yield itself as output. This loss is computed by sampling this $z \sim p(z)$, computing a morph between this z and itself using the morphing network and then computing the MSE between this output and the z used as an input. This loss is added to the other losses without any further scaling (a scaling factor of 1).

During preliminary experiments the morphing network would output latent vectors with way too large values in it for a small percentage of the morphs. The inputs for which this happened were not constant and often changed from epoch to epoch. The large values cause the output morph images to consist solely of noise. In order to fix this, an extra loss was added to constrain the output latent vector. The goal of this loss is to ensure that the L2 norm of the output latent vector is smaller or equal to the maximal L2 norm of the two input vectors, $||\hat{z}_{\text{morph}}||_2 \leq \max\big(||\hat{z}_1||_2, ||\hat{z}_2||_2\big)$. Due to a programming error in the code used for the results, this was implemented as $||\hat{z}_{\text{morph}}||_2 \leq \max\big(||\hat{z}_1||_2, ||\hat{z}_1||_2\big)$ which becomes $||\hat{z}_{\text{morph}}||_2 \leq ||\hat{z}_1||_2$. Due to time constraints this error was not resolved. The loss is implemented as $\text{ReLU}(||\hat{z}_{\text{morph}}||_2 - ||\hat{z}_1||_2)^2$. The Rectified Linear Unit (ReLU) activation function is defined as $\text{ReLU}(x) = \max(0, x)$ and is thus 0 when the input is smaller than 0 and otherwise equal to the input. By using the ReLU activation function, this loss only has a nonzero gradient when $||\hat{z}_{\text{morph}}||_2$ is larger than $||\hat{z}_1||_2$. This ensures that $||\hat{z}_{\text{morph}}||_2$ is not minimized when the constraint is not violated. In order to achieve its purpose, the loss had to be scaled with a scaling factor of 10. Therefore the final loss term becomes $10 \cdot \text{ReLU}(||\hat{z}_{\text{morph}}||_2 - ||\hat{z}_1||_2)^2$. After applying this loss, the extreme values in the output latent vectors disappeared.

7 Results

7.1 Improved Hyperparameters

This section shows the results for MorGAN and MorGAN with the improved hyperparameters for this implementation (called MorGAN+) that were introduced in section 6.4.1. In MorGAN+ the latent dimension is increased from 256 to 512 and dropout is decreased from 0.2 to 0.03. Table 1 lists the results for these to models.

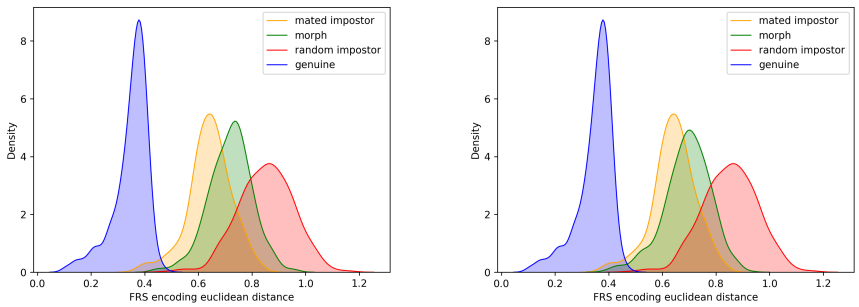
In Figure 10 the distributions of FRS distances can be found for these models. This figure shows the distribution of FRS encoding distances between multiple

sets of image pairs. In blue the distribution of distances between genuine pairs x with x^{ref} are shown. The green plot shows the distribution of distances between the generated morphs $\tilde{x}_{\text{morph}}$ and their respective reference images x_1^{ref} and x_2^{ref}. In orange the mated impostor distance distribution is shown. This denotes the distance between two reference images from the same morphing pair in the test set (x_1^{ref} and x_2^{ref}) Finally, the red plot show this distribution between pairs of x^{ref} images that are not specifically chosen to look like each other (random impostor). There are two impostor graphs because of the evaluation method used. Pairs of persons are made such that they already look like each other and then these pairs are used for morphing. Therefore it is useful to know how the distances between these chosen pairs are distributed.

In many cases the morph distances in these density plots are on average larger than the mated morph distances. This indicates that the reference images are actually closer to each other than they are to the morph. In the ideal case this distance would actually be half the distance between mated impostors, since that would indicate that the morph sits right in-between the two reference images.

Model	**MMPMR**	**RR**	**MMD**	**MRD**
MorGAN	0.032	0.082	0.719	0.720
MorGAN+	0.071	0.145	0.693	0.695

Table 1: *Results for MorGAN and MorGAN+. The table lists Mated Morph Presentation Match Rate (MMPMR), Reconstruction Rate (RR), Mean Morph Distance (MMD), and Mean Reconstruction Distance (MRD)*



(a) *MorGAN* ***(b)*** *MorGAN+*

Figure 10: *Distribution of FRS encoding distances for morphs compared to genuine, mated impostor and random impostor for MorGAN with normal hyperparameters and MorGAN with improved hyperparameters.*

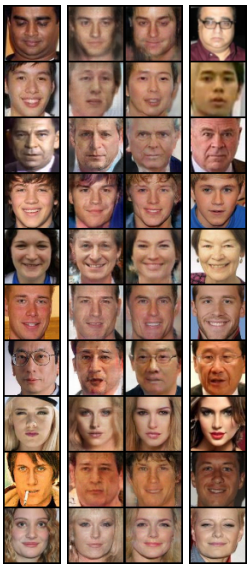


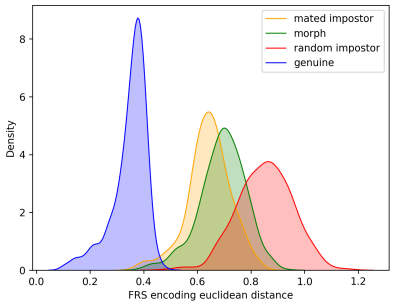
Figure 11: *Morph inputs and generated morphs by the MorGAN (1) and MorGAN+ (2) models respectively. Morph inputs x_1 and x_2 are displayed left and right of the morphs. The morphs are displayed in-order in the middle with every column being a different model.*

7.2 Results for Dis_l Loss and Increasing α

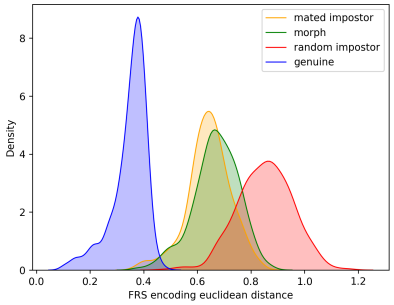
Table 2 the results are shown for models with pixel-wise or Dis_l reconstruction loss for various values of α. In Figure 12 the distribution of FRS distances for all configurations can be found.

Model	**Dis_l**	α	**MMPMR**	**RR**	**MMD**	**MRD**
MorGAN+	×	0.3	0.071	0.145	0.693	0.695
MorGAN+	✓	0.3	0.080	0.215	0.669	0.664
MorGAN+	×	3.0	0.141	0.264	0.654	0.651
MorGAN+	✓	3.0	0.151	0.349	0.646	0.634
MorGAN+	×	30.0	0.151	0.465	0.631	0.607
MorGAN+	✓	30.0	0.315	0.677	0.599	0.564

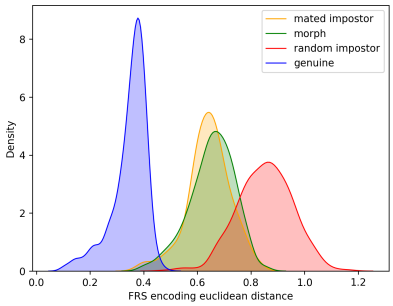
Table 2: *Results for various models with either pixel-wise or Dis_l reconstruction loss and varying values for reconstruction loss factor α*



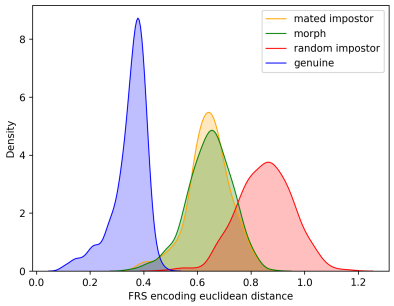
(a) *MorGAN+,* $\alpha = 0.3$



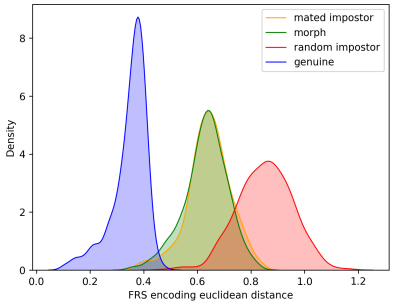
(b) *MorGAN+ with* Dis_l *reconstruction loss,* $\alpha = 0.3$



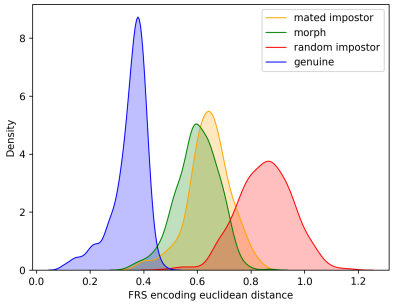
(c) *MorGAN+,* $\alpha = 3$



(d) *MorGAN+,* Dis_l *,* $\alpha = 3$



(e) *MorGAN+,* $\alpha = 30$



(f) *MorGAN+,* Dis_l *,* $\alpha = 30$

Figure 12: *FRS encoding distance comparison for the different reconstruction loss configurations*

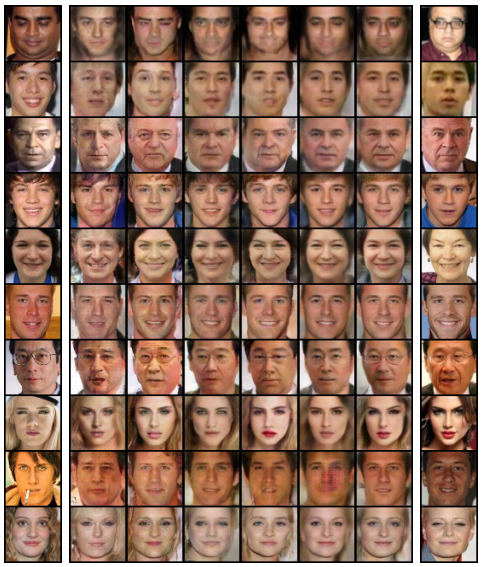


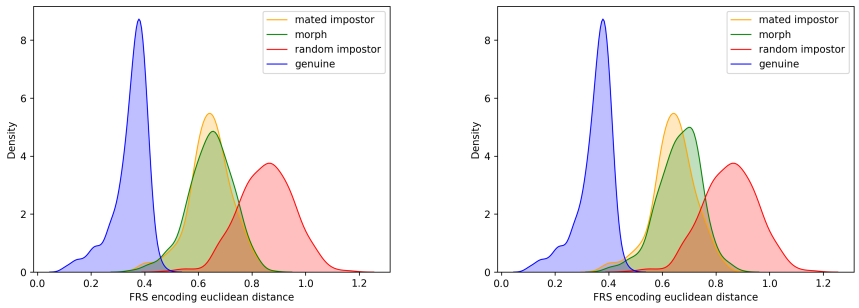
Figure 13: *Morph inputs and generated morphs by MorGAN+ (1), MorGAN+ Dis_l (2), MorGAN+ $\alpha = 3$ (3), MorGAN+ Dis_l $\alpha = 3$ (4), MorGAN+ $\alpha = 30$ (5), and MorGAN+ Dis_l $\alpha = 30$ (6) models. Morph inputs x_1 and x_2 are displayed left and right of the morphs. The morphs are displayed in-order in the middle with every column being a different model.*

7.3 Morph Loss Results

In Table 3 a comparison can be found between different morph loss configurations. All models are trained with Dis_l reconstruction loss and $\alpha = 3$ and use the MorGAN+ hyperparameters.

α_{morph}	**ML**	**MMPMR**	**RR**	**MMD**	**MRD**
0.0	Neither	0.151	0.349	0.646	0.634
3.0	Both	0.090	0.230	0.664	0.655
3.0	G_z	0.103	0.281	0.653	0.644
3.0	G_x	0.125	0.315	0.650	0.634

Table 3: *Results for the Dis_l morph loss. The ML column lists which networks were trained using the morph loss.*



(a) *No morph loss*

(b) *Morph loss applied to both networks*

(c) *Morph loss applied only to* G_z

(d) *Morph loss applied only to* G_x

Figure 14: *FRS encoding distance comparison for the different morph loss configurations*

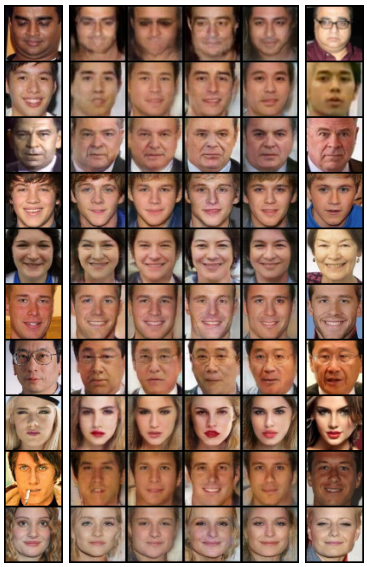


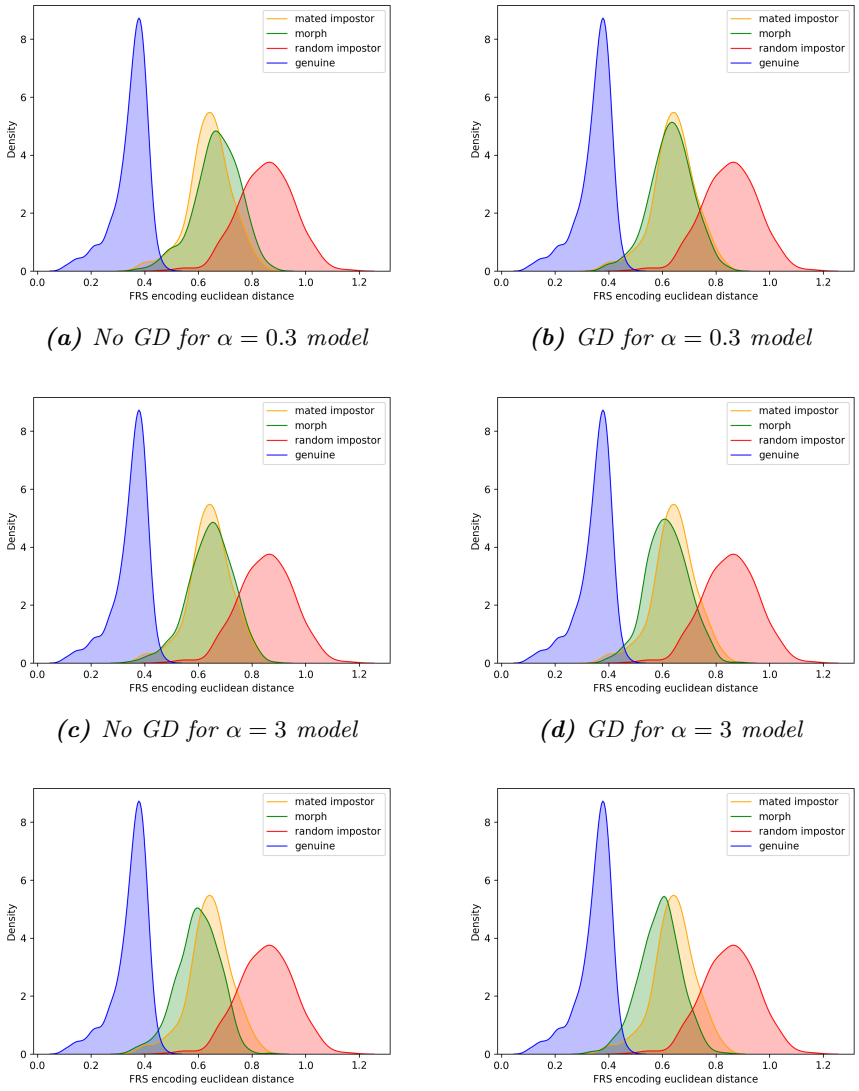
Figure 15: *Morph inputs and generated morphs by models trained without morph loss (1), with morph loss (2) with morph loss only on G_z (3), and with morph loss only on G_x (4). Morph inputs x_1 and x_2 are displayed left and right of the morphs. The morphs are displayed in-order in the middle with every column being a different model.*

7.4 Gradient Descent on $\hat{z}_{\text{morph}}$

In this section the results are listed for the gradient descent on latent vectors experiments. For these experiments $\hat{z}_{\text{morph}}$ as well as $\hat{z}_1$ and $\hat{z}_2$ are optimized for the entire train set. In Table 4 the results for these experiments and for the original models are listed. The original models are the Dis_l models evaluated in Section 7.2. The respective α values used during training of the models are also listed in the table. These α values are only used during training of these models and are not used when optimizing the latent vectors for these experiments.

Using GD	α	**MMPMR**	**RR**	**MMD**	**MRD**
×	0.3	0.080	0.215	0.669	0.664
✓	0.3	0.148	0.542	0.633	0.594
×	3.0	0.151	0.349	0.646	0.634
✓	3.0	0.232	0.674	0.619	0.566
×	30.0	0.315	0.677	0.599	0.564
✓	30.0	0.344	0.897	0.591	0.515

Table 4: *Results for the gradient descent on latent vectors experiments. The "Using GD" column denotes whether the latent vectors have been optimized using gradient descent*



(a) *No GD for* $\alpha = 0.3$ *model*

(b) *GD for* $\alpha = 0.3$ *model*

(c) *No GD for* $\alpha = 3$ *model*

(d) *GD for* $\alpha = 3$ *model*

(e) *No GD for* $\alpha = 30$ *model*

(f) *GD for* $\alpha = 30$ *model*

Figure 16: *FRS encoding distance comparison for the different models without and with gradient descent on the latent vectors.*

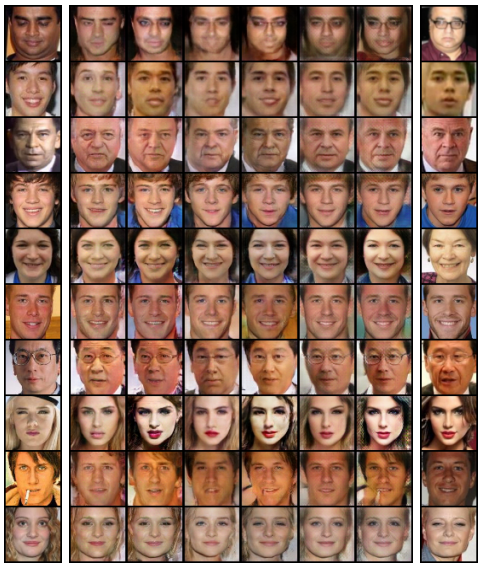


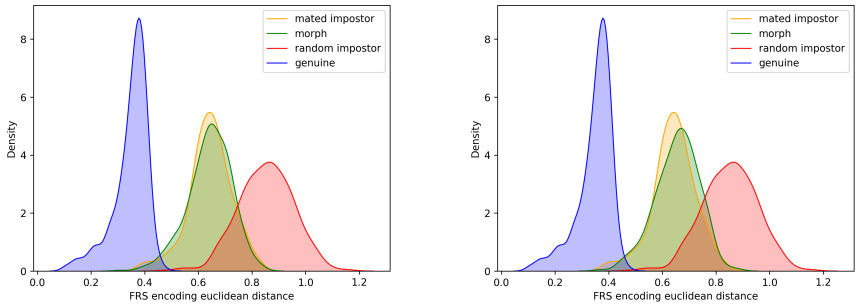
Figure 17: *Morph inputs and generated morphs by models trained with* $\alpha = 0.3$ *(1),* $\alpha = 3$ *(2) and* $\alpha = 30$ *(3)* Dis_l *reconstruction loss without (a) and with (b) gradient descent on the latent vectors after training. Morph inputs* x_1 *and* x_2 *are displayed left and right of the morphs. The morphs are displayed in-order in the middle with every column being a different model.*

7.5 Morph Network

This section contains results for the morphing network. The results are for the same trained G_z and G_x networks, the weights are exactly the same in both evaluations. The only difference is that either the morphing network or linear interpolation (the default MorGAN morphing method) is used for morphing $\hat{z}_1$ and $\hat{z}_2$. This is also reflected in the results for the reconstruction rate and mean reconstruction distance where the morphing method has no influence and therefore the results are the same.

Morphing Method	**MMPMR**	**RR**	**MMD**	**MRD**
Linear interpolation	0.141	0.294	0.642	0.637
Morph Network	0.132	0.294	0.655	0.637

Table 5: *Results for the morphing network experiments*



(a) *Linear interpolation* ***(b)*** *Using the morphing network*

Figure 18: *FRS encoding distance comparison for the same model when using the morphing network and when using the linear interpolation method used in MorGAN*

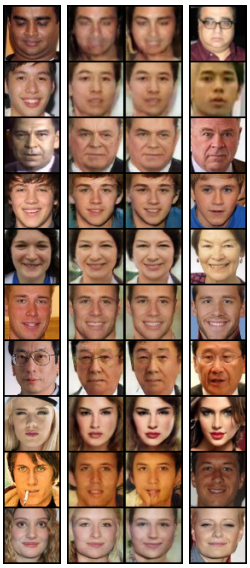


Figure 19: *Morph inputs and generated morphs by the model trained with morphing network. (1) shows morphs with the normal linear morphing method and (2) with the morphing network for the same model. Morph inputs x_1 and x_2 are displayed left and right of the morphs.*

8 Discussion

This section will discuss the results presented in the section above as well as the limitations of this research.

8.1 Improved Hyperparameters

The improved hyperparameters do yield a substantial boost in both morphing and reconstruction performance. Additionally the images generated by the improved hyperparameter model seem less noisy and seem to match better with the input images in Figure 11, although this is very subjective. One of the improvements to the hyperparameters is a greatly reduced dropout rate (0.2 to 0.03). Dropout was added to regularize D and to stop sudden training collapse, but this is not as important anymore since the implementation used for all experiments in this report uses an extra regularization that only updates D when the G GAN loss is below a certain threshold. When dropout is lowered, D will become better at detecting fake images and thus the threshold will be reached more often. Therefore D is still prevented from becoming too strong, but simply by another regularization technique.

In [18] the authors also encountered problems with instability and experi-

mented with different regularization strategies to stabilize training. After initially experimenting with gradient penalties they note that "*Repeating this experiment with various strengths of Orthogonal Regularization, DropOut, and L2, reveals similar behaviors for these regularization strategies: with high enough penalties on D, training stability can be achieved, but at a substantial cost to performance*". Note that these experiments were done with a normal GAN algorithm and not with ALI or MorGAN. Still the results for reduced dropout on MorGAN do line up with their results, a lower dropout rate improves overall quality.

Attempts to remove all regularization on ALI or MorGAN result in collapse before training has converged. Finding out why this happens is not in the scope of this thesis, but the hypothesis is that this happens because the ALI and MorGAN discriminator has many more parameters due to it receiving both x and z as inputs. Therefore it is easier for D to overfit on the dataset and the game between G and D is also less balanced.

The improved hyperparameters also included a larger latent space size. While this might also have led to the improved performance, it is likely that the reduced dropout rate is responsible for the majority of the improvements. In separate experiments done on 28×28 CelebA images during the initial hyperparameter search, reduced dropout rates had the same improving effect without changing the latent space. Still it is impossible to deduce from these results which improvement had the most effect and as such, no conclusion on the individual effects of these changes will be made.

8.2 Results for Dis_l Loss and Increasing α

Looking at Table 2 and Figure 12 a number of observations can be made. The first observation is that a higher α value, which is the factor that scales the reconstruction loss, results in better morphing and reconstruction performance. This shows that the morphing performance is at least partially tied to the reconstruction performance. This makes sense, since a model that cannot retain the identity of a person reliably will probably also not fare well on the harder task of generating a morph that retains two identities. This result also shows that the latent space is laid out in such a way that the mean between 2 latent vectors often does result in a face that has properties of both inputs. This result can also mean that many of the other ideas might not improve much since the main bottleneck to better morphing performance is the reconstruction performance. The other experiments shed further light on this.

A second observation that can be made is that models trained with a Dis_l-based reconstruction loss perform better in both MMPMR and RR for the same value of α than models trained with a pixel-based loss. This does not necessarily show that the Dis_l-based loss is better. Since the pixel-based loss is based on the absolute difference between color values of pixels (always between 0-1) and the

Dis_l-based loss on the squared error between activations of convolutional layers (output of leaky ReLU, unbounded), these two losses cannot be compared directly for the same α value as their actual influence can be different. Generally the Dis_l-based loss seems to be larger than the pixel-based loss and thus has a larger influence on the trained model. This most likely explains the difference between the models for $\alpha = 0.3$ and $\alpha = 3.0$.

The Dis_l reconstruction loss does show a major improvement over the pixel-based loss when $\alpha = 30$. The MMPMR and RR for the Dis_l model are significantly higher than the metrics for the pixel model. The Dis_l-based loss appears to be better aligned with the ultimate objective of generating reconstructions that look like the same person rather than generating exactly the same image on a pixel level. It is important to note that because G_x and G_z barely improve wrt. the GAN loss, D is not updated often.

Because the GAN loss is overpowered by the reconstruction loss when $\alpha = 30$, the flaws of the reconstruction loss are not corrected. For the pixel-based loss this can be seen in Figure 13 where in column (5) the images appear soft and less detailed when looked at from up close. Details in the hair and in the background become a blur and facial features often also lose a lot of detail. The Dis_l-based loss has its own flaws as well. In the 4th morph from the top in column (6) of the same figure one can see a repetitive pattern in the hairline which appears way less natural than all samples from models with $\alpha < 30$. In many other images produced by the $\alpha = 30$ Dis_l model, weird noise artefacts can be seen as well. Although it is not objectively measurable, these images do appear more sharp and realistic and finding these artefacts is harder than spotting the blurriness of the pixel-based model. The suspicion is that a model could easily be trained to spot these artefacts and that its use for generating face morphs might be limited.

Overall the α parameter becomes a trade-off between image realism and reconstruction performance. For many of the following experiments $\alpha = 3$ and the Dis_l-based loss are chosen since this configuration provides a balance between MMPMR and image quality.

8.3 Morph loss results

Table 3 and Figure 14 show a comparison between models trained with various configurations of the proposed Dis_l morph loss. From these results it becomes clear that the morph loss does not improve morphing performance but instead reduces both the reconstruction and the morphing performance of the models.

In order to further investigate this reduced performance, two models were trained with the morph loss only used to train either G_z or G_x. From the results it appears that the effects of the morph loss on G_z have a bigger negative effect on performance. That being said, neither of the configurations manages to

improve over the baseline of no morph loss and the differences between morph loss on G_x and G_z are small enough that they may not be significant.

As discussed in the discussion of the previous experiment, the latent space of these models already appears organized in such a way that the reconstruction and morphing performance are tied closely together. The reconstruction performance of the models appears to be the bottleneck for morphing performance and the morph loss only appears to add more "confusion" to the training process instead of improving the morphing performance. In the next sections more experiments with the morph loss in different settings will be discussed.

8.4 Gradient Descent on $\hat{z}_{morph}$

In Table 4 and Figure 16 the results for optimizing the output latent vectors of already trained models for a specific morph can be found. As mentioned before, this morphing method is significantly slower than just generating a morph using the models. This extra time investment does appear to yield a significant improvements in both morphing and reconstruction performance. For all models the reconstruction rate increases dramatically when reconstructions are optimized using the Dis_l reconstruction loss. Contrary to the results in the previous section, the morph loss does also manage to greatly improve morphing performance when used to optimize latent vectors. There does appear to be a case of diminishing returns for the morphing performance, as the improvement for the $\alpha = 30$ is significantly less than for the other two models.

If one can simply optimize a latent vector like this, do we actually need G_z? Based on preliminary research done on these models it appears that starting with a randomly initialized latent vector leads to getting stuck in local optima. G_z is therefore necessary to generate a good starting point for latent vector optimization. It also appears that G_z is far from perfect, since its outputs can be significantly improved.

Interestingly, when looking at Figure 17, one can observe that this further optimization for a Dis_l based loss does not appear to lead to visible noise artefacts in the models with a lower α, but does lead to stronger artefacts in some images for $\alpha = 30$ (for instance in the 3rd image from the bottom). A possible reason for this is that the morphs are constrained to the output space of G_x. If these artefacts are non-existent in the output space of G_x, it is simply impossible to output images that contain them. Another possible explanation could be that D, which is used to compute Dis_l for the losses, is trained better for $\alpha = 0.3$ and $\alpha = 3$ since it less limited by the GAN loss threshold. This could mean that the convolutional filters are more "refined" and are not as sensitive to these noisy artefacts.

It would be interesting to measure whether trade-off between image quality and morphing/reconstruction performance is also present here. Given the design of the discriminator in MorGAN/ALI, this cannot easily be done with the discriminator as it outputs whether x or z is fake. It is not possible to get an output for only x and simply feeding in the optimized z with the generated morph will likely result in z being identified as fake since it is not constrained in any way during optimization. Something like the Fréchet Inception Distance (FID) [19] could be used to evaluate the quality before and after optimization, but this is beyond the scope of this work.

8.5 Morph Network

Given the significant performance benefits of optimizing $\hat{z}_{\text{morph}}$ for already trained models, there might be a possible benefit in training a neural network to generate more optimal morphs. Therefore the morphing network introduced earlier has the possibility to add something to the morphing performance. Unfortunately, as can be seen in Table 5, it does not. The morphing network was trained together with G_z, G_x and D for the full 123 epochs but is outperformed by simply using that same G_z and G_x and morphing the normal way. This is after adding multiple regularization steps to improve the performance of this morphing network.

There are multiple possible reasons as to why the performance is bad. The network might not have enough capacity to learn a good operation. Additionally the network also has to keep a good performance across the entire latent space. It might also be that, as discussed in earlier sections, the latent space is already shaped such that taking the mean of two latent vectors is close to optimal. The results presented here do not cover all options and as such it cannot be concluded that the idea of a trained morphing method like the morphing network can never work.

8.6 Limitations

8.6.1 Implementation Difference with MorGAN

A limitation to the results of this thesis is that the MorGAN implementation most likely differs from the official implementation used for the results in the MorGAN paper. Using the code for ALI[17] gave insight into a lot of the implementation details, but obviously did not include the MorGAN-specific parts. A lot of effort was put into mimicking the details and behaviours described in ALI and MorGAN as close as possible, but tricks like the D limiting regularization remained necessary to stop divergence. The results for the default MorGAN implementation used in this paper are not necessarily representative of the actual MorGAN performance.

8.6.2 Significance of the Results

Since all results are acquired by training a single model and computing the results only once it is not known exactly how statistically significant the results are. The result generation script has been tweaked multiple times in order to make it consistent and to fix any small bugs that were still left in. Throughout many runs on the validation set as well as some on the test set, all results that are discussed in the discussion section and conclusion have remained consistent although the exact numbers did sometimes vary. This is, of course, no replacement for actual significance testing and the reader should be aware that this approach has been taken.

9 Conclusion

Four possible improvements to the MorGAN algorithm are introduced and evaluated in the sections above. The first possible improvement, using a Dis_l-based reconstruction loss, does not necessarily improve morphing performance by itself. However, as the value for the reconstruction loss factor α is raised to 30, both morphing performance and reconstruction performance are significantly better for models trained with the Dis_l-based reconstruction loss for this value of α. This result also shows that the morphing performance is tied to the reconstruction performance and that therefore the morphing performance can be greatly improved by simply improving the reconstruction performance. By using the Dis_l-based loss and $\alpha = 30.0$, the MMPMR increased from 0.071 for MorGAN with $\alpha = 0.3$ and 0.151 for MorGAN with $\alpha = 30.0$ to an MMPMR of 0.315.

The second possible improvement is the addition of the morph loss. This added loss is aimed at improving the morphing performance, but is shown to instead reduce both morphing and reconstruction performance. The chosen implementation of the morph loss is therefore not able to improve the quality of generated morphs.

Further optimization of the morphs and reconstructions generated by fully trained models is introduced as the next possible improvement. The results show that this method is very effective at increasing both morphing and reconstruction performance. This method does, however, slow down the morphing process considerably. This could mean that generating a dataset of morphs using an already trained model becomes a matter of hours rather than minutes. The best performing model ($\alpha = 30.0$, Dis_l) improved from an MMPMR of 0.315 to an MMPMR of 0.344 using this further optimization technique, while models trained with a lower α showed larger gains in performance.

A final possible improvement that is introduced is the so-called morphing network. This network replaces the default MorGAN morphing method of lin-

early interpolating between latent vectors. Training of the morphing network proves to be hard to stabilize and requires multiple auxiliary regularization losses. Even with these added steps, the morphing network fails to improve over the baseline morphing method. Therefore the conclusion is that this method in its current form is unable to improve the morphing performance of MorGAN.

10 Future Work

In this section some possibilities for future work are listed.

10.1 Using an FRS-based Morph Loss

A possible way to improve the results listed in this thesis is to use a trained facial recognition system to compute the "distance" between a morph and its inputs instead of Dis_l activations. Combined with the max-based morph loss used for the morphing network, this would result in almost directly optimizing the MMPMR. Some exploratory experiments were performed with this idea, but it turned out to be too hard to get right given the time-frame for the thesis. An FRS-based morph or reconstruction loss will likely have similar problems as the Dis_l-based loss with noisy artefacts when pushed to 0. Therefore it could be interesting to apply the loss with a threshold. Only when the loss is higher than a certain distance, its gradient will be non-zero. This might force the generated morphs to look like their inputs while also giving the GAN loss some "space" to work with.

10.2 Using More Stable Training Methods

ALI and MorGAN use a fairly normal GAN training objective. This has led to problems with training collapse. The current state of the art in GAN research often uses gradient penalties like R1 regularization [20] or Wasserstein GANs with a gradient penalty (WGAN-GP)[21]. These methods are more stable and allow for way deeper networks with residual layers or skip-connections like used in StyleGAN2 [22]. Some early experiments with both WGAN-GP and R1 regularization were performed, but the design of the ALI discriminator made implementing these gradient penalties a non-trivial exercise. Nevertheless these early experiments showed a lot of promising results. A possible simplification could be to split the discriminator up in a D_z and D_x and have them only be responsible for the realism of generated $\tilde{x}$ and $\hat{z}$ samples rather than also using D to train the reconstruction performance. This would then be solely handled by the Dis_l reconstruction loss.

In [23] the authors introduce a new technique that trains networks similar to G_z and G_x using a novel GAN-based model with R1 regularization. This paper uses a StyleGAN[24] generator as well as a style-based encoder network.

Their method achieves high-resolution (1024×1024 pixels) face generation and reconstruction. Unfortunately it appears (visually) that the technique does not have a high reconstruction rate. This technique could be further extended with a Dis_l or FRS based reconstruction loss to improve reconstruction performance and make it suitable for morphing.

10.3 Using an End-to-End Morphing Network

A final suggestion for possible research is something that might sound quite trivial. Instead of training a whole autoencoder-like setup with a stochastic latent vector, one could instead train an entire network to take 2 input images and generate a morph. Without the bottleneck of a latent vector, this model could be easier to train and could deliver way more optimal results. The model could utilize a design with skip-connections like U-net [25], but then modified to take 2 input images instead of one. This setup might suffer from similar problems that also plagued the morphing network, but ultimately it might result in better morphs at the cost of not being able to generate any images from the model. This all-in-one morphing network could be trained by applying a GAN loss to safeguard realism and a morph loss to assess morphing quality.

References

[1] Matteo Ferrara, Annalisa Franco, and Davide Maltoni. The magic passport. *IJCB 2014 - 2014 IEEE/IAPR International Joint Conference on Biometrics*, 12 2014.

[2] Andreas Braun Naser Damer, Alexandra Moseguí Saladié and Arjan Kuijper. Morgan: Recognition vulnerability and attack detectability of face morphing attacks created by generative adversarial network.

[3] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *CoRR*, abs/1512.09300, 2015.

[4] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`.

[5] Divyanshu Mishra. Transposed convolution demystified. `https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba`.

[6] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

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

[8] U. Scherhag, A. Nautsch, C. Rathgeb, M. Gomez-Barrero, R. N. J. Veldhuis, L. Spreeuwers, M. Schils, D. Maltoni, P. Grother, S. Marcel, R. Breithaupt, R. Ramachandra, and C. Busch. Biometric systems under morphing attacks: Assessment of morphing techniques and vulnerability reporting. In *2017 International Conference of the Biometrics Special Interest Group (BIOSIG)*, pages 1–7, Sep. 2017.

[9] Satya Mallick. Face morph using opencv — c++ / python. `https://www.learnopencv.com/face-morph-using-opencv-cpp-python/`.

[10] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference, 2016.

[11] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *CoRR*, abs/1605.09782, 2016.

[12] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[13] Dlib shape predictor 5 face landmarks download. `http://dlib.net/files/shape_predictor_5_face_landmarks.dat.bz2`.

[14] Adam Geitgey. The `face_recognition` python package. `https://face-recognition.readthedocs.io/`.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[16] 9310gaurav. Pytorch implementation of ali. `https://github.com/9310gaurav/ali-pytorch/`.

[17] Ali official implementation. `https://github.com/IshmaelBelghazi/ALI`.

[18] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018.

[19] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017.

[20] Lars M. Mescheder. On the convergence properties of GAN training. *CoRR*, abs/1801.04406, 2018.

[21] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[22] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[23] Stanislav Pidhorskyi, Donald A Adjeroh, and Gianfranco Doretto. Adversarial latent autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14104–14113, 2020.

[24] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018.

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

Appendices

A Algorithms in pseudo-code

Algorithm 2: MorGAN with morph loss algorithmic training procedure. Note that sampling of x_2 is not described in full detail.

$\theta_G, \theta_D \leftarrow$ randomly initialize network parameters
repeat
  $x_1^{(1)}, \ldots, x_1^{(M)} \sim q(x)$ `// Sample M images from the data distribution`
  $x_2^{(1)}, \ldots, x_2^{(M)} \sim q(x)$ `// Sample M images of a different identity`
  $z^{(1)}, \ldots, z^{(M)} \sim p(z)$ `// Sample M vectors from a Normal distribution`
  $\hat{z}_1^{(i)} \leftarrow G_z(x_1^{(i)}), i = 1, \ldots, M$ `// Compute` G_z `generation for` x_1
  $\hat{z}_2^{(i)} \leftarrow G_z(x_2^{(i)}), i = 1, \ldots, M$ `// Compute` G_z `generation for` x_2
  $\hat{z}_{\text{morph}}^{(i)} \leftarrow \frac{\hat{z}_1^{(i)} + \hat{z}_2^{(i)}}{2}, i = 1, \ldots, M$ `// Compute` $\hat{z}_{\text{morph}}$
  $\tilde{x}^{(j)} \leftarrow G_x(z^{(j)}), j = 1, \ldots, M$ `// Compute` G_x `generation`
  $\tilde{x}_{\text{recon}}^{(i)} \leftarrow G_x(\hat{z}_1^{(i)}), i = 1, \ldots, M$ `// Compute reconstructed` x
  $\tilde{x}_{\text{morph}}^{(i)} \leftarrow G_x(\hat{z}_{\text{morph}}^{(i)}), i = 1, \ldots, M$ `// Compute` $\tilde{x}_{\text{morph}}$
  $\rho_q^{(i)} \leftarrow D(x^{(i)}, \hat{z}^{(i)}), i = 1, \ldots, M$ `// Compute Discriminator predictions`
  $\rho_p^{(j)} \leftarrow D(\tilde{x}^{(j)}, z^{(j)}), j = 1, \ldots, M$
  `// Compute discriminator loss`
  $\mathcal{L}_{\text{GAN-D}} \leftarrow -\frac{1}{M}\sum_{i=1}^{M}\log(\rho_q^{(i)}) - \frac{1}{M}\sum_{j=1}^{M}\log(1-\rho_p^{(j)})$
  `// Compute generator loss`
  $\mathcal{L}_{\text{GAN-G}} \leftarrow -\frac{1}{M}\sum_{i=1}^{M}\log(1-\rho_q^{(i)}) - \frac{1}{M}\sum_{j=1}^{M}\log(\rho_p^{(j)})$
  $\mathcal{L}_{\text{recon}} \leftarrow \text{recon\_loss}(x_1, \tilde{x}_{\text{recon}})$
  $\mathcal{L}_{\text{morph}} \leftarrow \text{morph\_loss}(x_1, x_2, \tilde{x}_{\text{morph}})$
  $\mathcal{L}_{\text{syn}} \leftarrow \mathcal{L}_{\text{GAN-G}} + \alpha\mathcal{L}_{\text{pixel}} + \alpha_{\text{morph}}\mathcal{L}_{\text{morph}}$
  if $\mathcal{L}_{\text{GAN-G}} < 3.5$ **then**
    $\theta_D \leftarrow \theta_D - \nabla_{\theta_D}\mathcal{L}_{\text{GAN-D}}$
  end if
  $\theta_G \leftarrow \theta_G - \nabla_{\theta_G}\mathcal{L}_{\text{syn}}$
until *convergence*

B Hyperparameters

Operation	Kernel	Strides	Feature maps	BN?	Dropout	Nonlinearity
$G_z(x)$ – $3 \times 64 \times 64$ input						
Convolution	2×2	1×1	64	$\surd$	0.0	Leaky ReLU
Convolution	7×7	2×2	128	$\surd$	0.0	Leaky ReLU
Convolution	5×5	2×2	256	$\surd$	0.0	Leaky ReLU
Convolution	7×7	2×2	256	$\surd$	0.0	Leaky ReLU
Convolution	4×4	1×1	512	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	512	$\times$	0.0	Linear
$G_x(z)$ – $256 \times 1 \times 1$ input						
Transposed convolution	4×4	1×1	512	$\surd$	0.0	Leaky ReLU
Transposed convolution	7×7	2×2	256	$\surd$	0.0	Leaky ReLU
Transposed convolution	5×5	2×2	256	$\surd$	0.0	Leaky ReLU
Transposed convolution	7×7	2×2	128	$\surd$	0.0	Leaky ReLU
Transposed convolution	2×2	1×1	64	$\surd$	0.0	Leaky ReLU
Convolution	1×1	1×1	3	$\times$	0.0	Sigmoid
$D(x)$ – $3 \times 64 \times 64$ input						
Convolution	2×2	1×1	64	$\times$	0.2	Leaky ReLU
Convolution	7×7	2×2	128	$\surd$	0.2	Leaky ReLU
Convolution	5×5	2×2	256	$\surd$	0.2	Leaky ReLU
Convolution	7×7	2×2	256	$\surd$	0.2	Leaky ReLU
The ouput of the layer above is used as $Dis_l(x)$						
Convolution	4×4	1×1	512	$\surd$	0.2	Leaky ReLU
$D(z)$ – $512 \times 1 \times 1$ input						
Convolution	1×1	1×1	1024	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	1024	$\times$	0.2	Leaky ReLU
$D(x, z)$ – $1536 \times 1 \times 1$ input						
Concatenate $D(x)$ *and* $D(z)$ *along the channel axis*						
Convolution	1×1	1×1	2048	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	2048	$\times$	0.2	Leaky ReLU
Convolution	1×1	1×1	1	$\times$	0.2	Sigmoid
Optimizer	Adam ($\alpha = 10^{-4}$, $\beta_1 = 0.5$)					
Batch size	100					
Epochs	123					
Leaky ReLU slope	0.02					
Weight, bias initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$), Constant(0)					

Table 6: *CelebA model hyperparameters (unsupervised). Modified version of the table from ALI[10]*

Operation	Output neurons	BN?	Dropout	Nonlinearity
Concatenate two latent vectors of size 512 together into one vector of size 1024				
Morphing Network – 1024 input				
Fully Connected Layer	1024	×	0.0	Leaky ReLU
Fully Connected Layer	1024	×	0.0	Leaky ReLU
Fully Connected Layer	512	×	0.0	Linear

Table 7: *Morphing network architecture*