ScoreFace: Face Texture Inpainting with Score-Based Generative Models

Arda Arslan

Semester Project
September 2022

Supervisors:
Emre Aksan
Marcel Bühler
Xu Chen
Prof. Dr. Otmar Hilliges

ETH zürich

Advanced Interactive Technologies

Abstract

To train an accurate face verification network, the training dataset should consist of face images with varying poses. Reducing the pose discrepancies between the training and the test images by frontalizing the test images might also be useful in increasing the accuracy of the network. Using a 3DMM, one can extract 3D geometry information of a given face image, and a face texture. The outputs of 3DMM can be used to render the face image from different views. However, due to self-occlusions there will be missing regions in the face texture and the novel-viewed face images. In this work, we propose two pipelines which can inpaint the missing regions in the face texture. The first one makes an optimization in the image space, and the second one in the texture space. Both pipelines use gradients from a pre-trained Score-Based Generative Model, and they can be used to enrich the pose variations in the training dataset of the face verification network, and also to frontalize test images. We show that our pipelines are successful in inpainting incomplete regions in the face image (and the corresponding texture pixels) when the view is close to the original view in the image, however they are both unsuccessful in inpainting regions when the view is too different from the original view.

Contents

Introduction

Pose-invariant face verification is a challenging problem due to lack of pose variations in the training images and pose discrepancies between the training and the test images [Deng et al. 2017]. By fitting a 3D Morphable Model (3DMM) on a 2D face image, one can generate 3D face geometry information and a face texture. Then, by using a 3D renderer, one can render the face from arbitrary views. The generated face texture will be incomplete due to face self-occlusions, and this will cause the rendered faces to have missing regions as well. By completing the missing regions of a given face texture, one can enrich the pose variations in the training images for the face verification task. One can also frontalize profile face images during inference of this task, which decreases the pose discrepancy between the training and the test images [Deng et al. 2017]. In this work, we propose two pipelines for this purpose: The first pipeline uses the gradients from a Score-Based Generative Model (SGM) [Song et al. 2020] with respect to the missing regions of an incomplete face image, and makes the optimization in the face image space. The updated pixels are then manually copied to the corresponding places in the face texture. The second pipeline, which does the optimization in the texture space, also uses the gradients from this model with respect to the missing regions of an incomplete face image, and then multiplies it with the Jacobian of the 3D rendering function to find the gradients with respect to the missing regions in the face texture. Both pipelines can be used to enrich the pose variations in the training dataset of the face verification network, and also to frontalize test images.

The code for the 3DMM we use is available at https://github.com/ardarslan/TF_FLAME, and the code for our work can be found at https://github.com/ardarslan/score-face.

Related Work

2.1 Face Texture Inpainting

Previous work mostly used Generative Adversarial Networks (GAN) [Goodfellow et al. 2014] for the face texture inpainting task. [Deng et al. 2017] proposed UV-GAN in which they fit a 3DMM to a face image, and extract 3D face geometry information and an incomplete face texture. They use a GAN to fill the incomplete parts of the texture. This network consists of a generator, two discriminators, and a module that is used for preserving face identity. The generator is an autoencoder whose inputs are incomplete face textures and labels are complete face textures. The reconstruction loss is a pixel-wise L1 norm. They use two types of discriminators: a global discriminator which ensures that the whole face texture is realistic, and a local discriminator which ensures the face centre of face texture is realistic. To evaluate the reconstruction quality of generated face textures, they use Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) [Wang et al. 2004] on generated and ground truth face textures. They use UV-GAN to enrich the pose variations in the training dataset of a face verification task. They also use it to frontalize profile face images during inference to decrease the pose discrepancy between the face verification pairs. They claim that both methods increase the accuracy of the face verification network.

[Na et al. 2020] proposed a new architecture named Attention ResCUNet-GAN. In this work, the authors use 3DDFA ([cleardusk 2022], [Guo et al. 2020a], [Zhu et al. 2019]) to extract face geometry information and incomplete face textures. The generator of the proposed architecture consists of coupled U-Nets [Ronneberger et al. 2015]. The authors state that coupling connections [Tang et al. 2018] are used for enhancing information flow across consecutive U-Nets. The authors also use attention gates [Schlemper et al. 2018] for skip connections within each U-Net for preventing the decoders from using irrelevant low-level information encoded by the

encoders. As in UV-GAN, this new architecture also uses a global and a local discriminator, and an identity-preserving module. The authors report that this architecture achieves better SSIM and PSNR values on Multi-PIE dataset [Gross et al. 2008] compared to UV-GAN.

Gecer et al. [Gecer et al. 2021] brought a new perspective to solving the face texture inpainting problem. They proposed an optimization-based one-shot face texture inpainting and frontalization approach. Their proposed algorithm takes a 2D face image as input, rotates it in 3D, and fills in the missing regions by reconstructing the rotated image in a 2D face generator, based on the visible parts. This 2D reconstruction is performed by an optimization in the latent space of the generator. The generator consists of a style encoder that takes re-rendered images as input to produce parameters of a latent space, and a StyleGAN v2 [Karras et al. 2019] which takes latent parameters as input to produce 2D face images. From each 2D face image, an incomplete face texture is generated. Also from each perspective used during re-rendering, a visibility UV map is produced. As the final step, using the visibility UV maps and incomplete face textures, a complete face texture is generated. The authors report that this method also yields better SSIM and PSNR values on Multi-PIE dataset compared to UV-GAN.

2.2 Score-Based Generative Models

Generative models are trained so that the distribution of generated samples are sufficiently close to the data distribution of interest. The probability distribution of generated samples should be a normalized distribution such that sum of probabilities assigned to each sample in the domain of generated samples is 1. To calculate likelihood of a sample, we need to calculate the normalization constant, which is generally intractable. Therefore, if we would like to train a generative model with maximum likelihood training, we need to use certain types of model architectures which makes the normalization constant tractable (e.g. causal convolutions in autoregressive models or invertible networks for normalizing flows [Rezende and Mohamed 2015]), or instead of maximizing the true likelihood of the data, we need to maximize a lower bound of the data, which does not require the calculation of the normalization constant (e.g. Variational Autoencoders (VAEs) [Kingma and Welling 2013]) [Song 2022a].

One other way of dealing with the intractable normalization constant is modelling a score function of a probability distribution, which is done by SGMs. The score function is equal to the gradient of the log-probability density function of a datapoint with respect to this datapoint. In [Song 2022a], the score function is introduced as follows:

$$\mathbf{s}_\theta(\mathbf{x}) = \nabla_\mathbf{x} \log p_\theta(\mathbf{x}) = -\nabla_\mathbf{x} f_\theta(\mathbf{x}) - \underbrace{\nabla_\mathbf{x} \log Z_\theta}_{=0} = -\nabla_\mathbf{x} f_\theta(\mathbf{x}). \tag{2.1}$$

where $\mathbf{x}$ is an image whose score we are interested in, $\mathbf{s}_\theta(\mathbf{x})$ is the score function of the image, $p_\theta(\mathbf{x})$ is the normalized probability density of the image, $f_\theta(\mathbf{x})$ is the unnormalized probability density of the image, and Z_θ is the normalization constant.

As it can be observed from the equality above, the score function can be calculated without calculating the normalization constant.

To train SGMs, one needs to minimize the Fisher divergence between the model and the data distributions, defined as follows:

$$\mathbb{E}_{p(\mathbf{x})}[\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2] \tag{2.2}$$

Since the true data score $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$ is not available, a method named "score matching" ([Hyvärinen 2005], [Vincent 2011], [Song et al. 2019]) is used in order to minimize the Fisher Divergence. After a SGM is trained, Langevin Dynamics ([Parisi 1981], [Grenander and Miller 1994]) can be used to generate samples. Langevin Dynamics provides an MCMC procedure to sample from a distribution using only its score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. We simply need to initialize the chain from a prior distribution $\mathbf{x}_0 \sim \pi(\mathbf{x})$, and then apply the following iterative steps:

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon}\, \mathbf{z}_i, \quad i = 0, 1, \cdots, K,$$

where $\mathbf{z}_i \sim \mathcal{N}(0, I)$. Here, ϵ is the step size, and $\mathbf{z}_i$ is the ith noise sample. [Song 2022a]

SGMs can be used for generating high-quality samples, exact log-likelihood calculation, and controllable generation for inverse problem-solving [Song 2022a]. Some of the downstream tasks that SGMs are proven to be successful at are image generation, audio synthesis, shape generation, music generation, image inpainting, image colorization, compressive sensing, and medical image reconstruction (e.g. CT, MRI) [Song 2022a].

[Song et al. 2020] showed that SGMs can be used for high fidelity image generation. When the time the paper was published, for the unconditional image generation task on the CIFAR-10 dataset [Krizhevsky 2009], their model yielded the highest Inception score [Salimans et al. 2016] and the lowest FID ([mseitzer 2022], [Heusel et al. 2017]) among other generative modeling architectures. One problem of the proposed model is that in order to draw samples, they need thousands of evaluation steps. According to [Jing et al. 2022], this makes inference with SGMs slower than GANs and VAEs. [Jing et al. 2022] showed that instead of doing inference on high dimensionality, one can "restrict the diffusion via projections onto subspaces as the data distribution evolves toward noise" [Jing et al. 2022]. By this way, the authors achieve lower FID for the unconditional image generation on the CIFAR-10 dataset compared to the model proposed by [Song et al. 2020]. The authors of [Jing et al. 2022] also reduce the computational cost of drawing samples for the same number of denoising steps.

ScoreFace: Face Texture Inpainting with Score-Based Generative Models

We propose two pipelines which can be used for inpainting face textures. The first pipeline does the optimization in the image space, and the second one in the texture space. Both pipelines need outputs of a 3DMM fitting as input, use view ordering, texture pixel correction, and two-round optimization.

3.1 Predictor-Corrector Sampling

In [Song 2022a], steps of Predictor-Corrector Sampling (PC Sampling) is defined as follows:

At each "predict" step, an appropriate step size Δt is selected using the predictor, and then $\mathbf{x}(t + \Delta t) \sim p_{t+\Delta t}(\mathbf{x})$ is predicted based on the current image $\mathbf{x}(t) \sim p_t(\mathbf{x})$. At each "correct" step, the image is "corrected" using the gradients from SGM, so that $\mathbf{x}(t+\Delta t)$ becomes a better-quality sample from $p_{t+\Delta t}(\mathbf{x})$. Here $\mathbf{x}(t)$ is the current image, $p_t(\mathbf{x})$ is the distribution that the current image is sampled from, $p_{t+\Delta t}(\mathbf{x})$ is a distribution whose samples are more data-like, and $\mathbf{x}(t + \Delta t)$ is a sample from this more data-like distribution.

In [Song et al. 2020], the authors report that they achieve better sample quality by using Predictor-Corrector Sampling, compared to the quality of the samples generated using Corrector-Only Sampling. They also mention that in order to get high quality samples from Corrector-Only Sampling as well, they need to use too many "correct" steps per noise level.

3.2 Inpainting with Score-Based Generative Models

At each iteration of the single-view optimization process, we use a random tensor as the background image whose noise scale is decreased manually as the optimization proceeds, and eventually becomes a zero tensor. Therefore, at the end of the optimization, we get a black image as the final background. We tried using different colors for the final background, however the other colors caused artifacts in the pixels on the mesh. For example, when the final background is green, the pixels on the boundaries of the mesh were also green.

Similarly, at each iteration of the single-view optimization process, we replace the known texture pixels with sum of these pixels and a random tensor. Again, the noise scale of this random tensor decreases as the optimization proceeds, and at the end of the optimization, the random tensor becomes a zero tensor. This means, at the end of the optimization, we will get the known pixels of the texture back.

Hence, the background pixels and the known texture pixels are not optimized. At each step of the optimization, only the unknown texture pixels (which contribute to the current 2D face image) are optimized using the gradients from SGM.

3.3 View Ordering

In both pipelines, we do consecutive single-view optimizations using the gradients from SGM. Elev and azimuth values, which determine the views that will be optimized, are taken as input from the user. Since we want to make use of the pixels which were filled by 3DMM fitting as much as possible, we find it more beneficial to start from the view which is the closest to the original view in the input image. For this purpose, we use the global rotation information which we extract during 3DMM fitting, and order the views depending on how close they are to the original view in the input image.

3.4 Texture Pixel Correction

Since the optimization process does not have any information about the boundaries of the mesh, at the end of each single-view optimization process, generally the pixels at the borders of the mesh have dark colors. To fix this, we use the following algorithm:

- Create a grayscale version of the texture so that the pixel intensities are between 0 and 1. Call a pixel in the original texture as "dark" if its gray intensity is less than 0.30.
- Find the dark and not "known" pixels on the mesh which has a neighboring background pixel in the current view. Add these pixels to a set named "pixels to be explored" and a list named "pixels to be whitened". Here "known" pixel means that we know its value from 3DMM fitting, and we do not need to inpaint it. When we say pixel A and pixel B are neighbors, this means Euclidean distance between A and B is 1.
- Do the following steps until the set "pixels to be explored" becomes empty: Pop a pixel

Figure 3.1: *An example texture when we do not use Texture Pixel Correction. The texture pixels which correspond to the boundaries of the mesh have dark colors, which is because the color of the background is black.*

from the set "pixels to be explored", and add it to a set named "explored pixels". Check each of its neighbors. If a neighbor is dark, not in the set "explored pixels", and not known then add this neighbor to the set "pixels to be explored" and the list "pixels to be whitened".

- When the set "pixels to be explored" becomes empty, replace each pixel in the list "pixels to whiten" with a white pixel.

In Figure 3.1, we share an example texture when we do not use Texture Pixel Correction.

3.5 Two-Round Optimization

After all views are optimized, there are usually still some pixels which were not filled since we are using "Texture Pixel Correction". Therefore, we run a second round of consecutive single-view optimizations, in which we do not use "Texture Pixel Correction".

3.6 Pipelines

3.6.1 Consecutive Single-View Optimizations in the Image Space

For each single-view optimization, we use the inpainting algorithm provided in [Song 2022d]. Single-view optimization algorithm takes the current texture, the final background, a UV mapping matrix and a background mask as input, it uses them to render the initial face image, which is not yet optimized for the current view. Then, it creates a zero-one mask which has ones at the locations for the known pixels in the rendered face image. After that, the algorithm creates a random tensor (whose entries are sampled from $N(0,1) * 348.0$) with the same shape as the rendered face image, and it replaces the unknown pixels in the rendered face image with the corresponding pixels in this random tensor. Then, for 1000 iterations, the algorithm runs "predict" and "correct" steps whose details are explained in section 3.1. Both steps optimize

only the unknown pixels in the face image. Here, 348.0 is the initial noise scale selected by the authors of [Song et al. 2020] for image inpainting task on FFHQ dataset.

When we are in the first-round of the Two-Round Optimization, at the end of each single-view optimization, we replace the dark pixels in the optimized face image with white pixels as explained in section 3.4. We then manually copy the optimized pixels from the face image to their corresponding places in the texture. If there are multiple pixels in the face image that are mapped to the same pixel in the texture, then we update the pixel on the texture by averaging the intensities of the pixels on the face image. By using this small texture, we can make use of already optimized pixels while doing the next single-view optimization, however if we use only the large texture, then we face the problem explained in subsection 5.1.2.

A summary of this pipeline can be found in Figure 3.2.

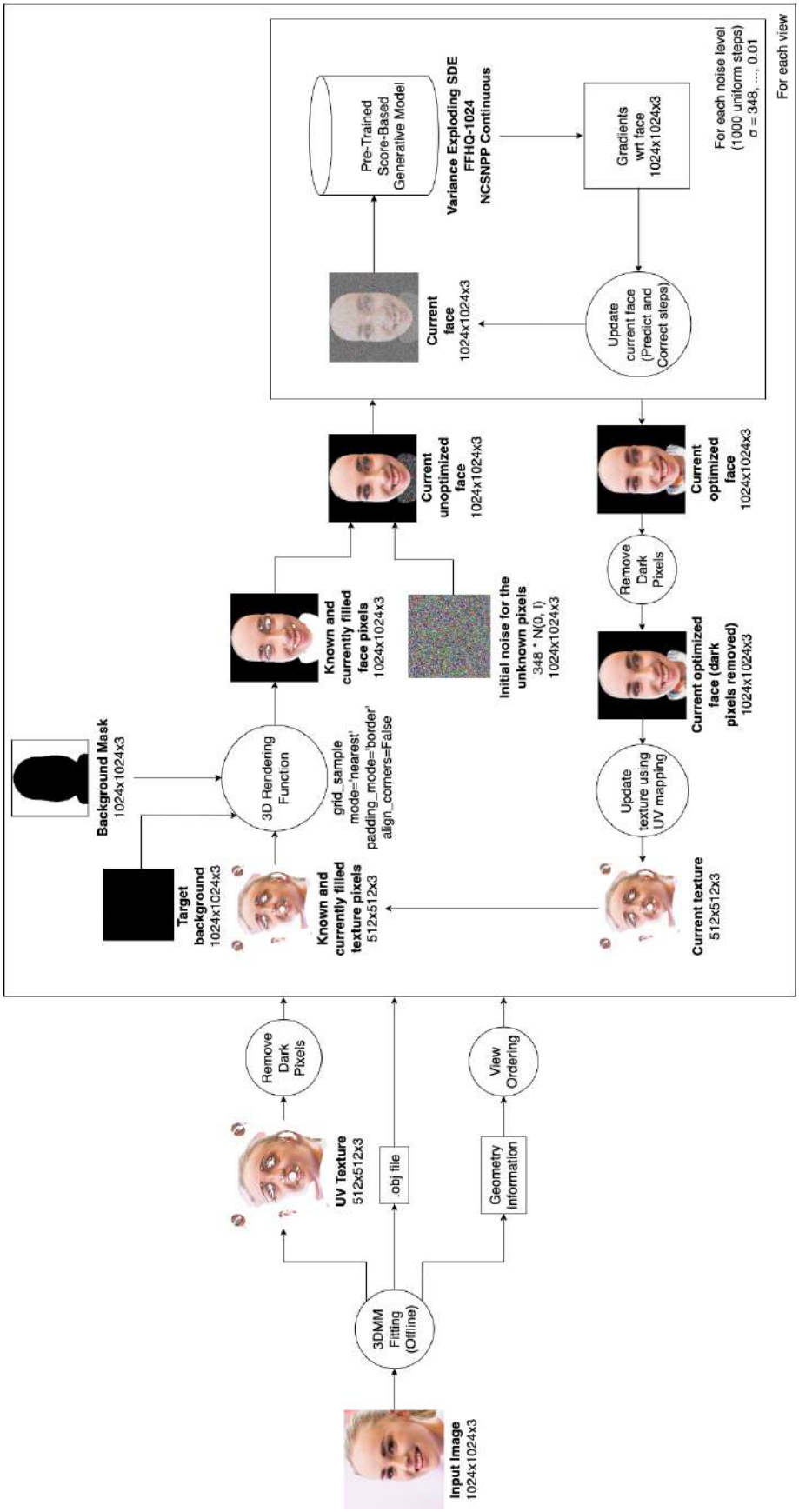

Figure 3.2: *Summary of optimization in the image space*

3.6.2 Consecutive Single-View Optimizations in the Texture Space (Two Textures)

For each single-view optimization, we use a modified version of the inpainting algorithm provided in [Song 2022d]. Our version takes the current small texture, the current large texture, the unoptimized large texture (which has only the known pixels), a mask which has ones at the locations for the known pixels in the unoptimized large texture, a UV mapping matrix, and a background mask, and the final background as input. At each iteration of the single-view optimization process, the background and the filled texture are recreated as described in section 3.2. After creating these matrices, we do 6 "correct" steps at each noise level. The "correct" step used here is different from the one used in subsection 3.6.1. The modified "correct" step does the following:

- Use SGM to get the gradients with respect to the current face image.
- Calculate Jacobian of the 3D rendering function, multiply this with the gradients from the previous step. This gives gradients with respect to the large texture.
- Create a noise tensor whose shape is the same with the shape of the large texture.
- Calculate Frobenius norm of the noise tensor and the gradient tensor. Using these norms, and the SNR value, calculate the step size.
- Update the unknown pixels of the large texture using the gradients from step 2, the noise tensor from step 3, and the step size from step 4.
- Using the updated large texture, render the new face image. This new face image will be used while calculating gradients in the next "correct" step.

At the end of each single-view optimization, we find the dark pixels in the optimized face image that should be replaced by white pixels as explained in section 3.4. Since the optimization is done in the large texture, replacement of pixels with white pixels is also done in there. Then, using the UV mapping matrix, and the updates we did on the large texture, we update the small texture. If there are multiple pixels on the large texture that are mapped to the same pixel on the small texture, then we update the pixel on the small texture by averaging the intensities of the pixels on the large texture.

A summary of this pipeline can be found in Figure 3.3.

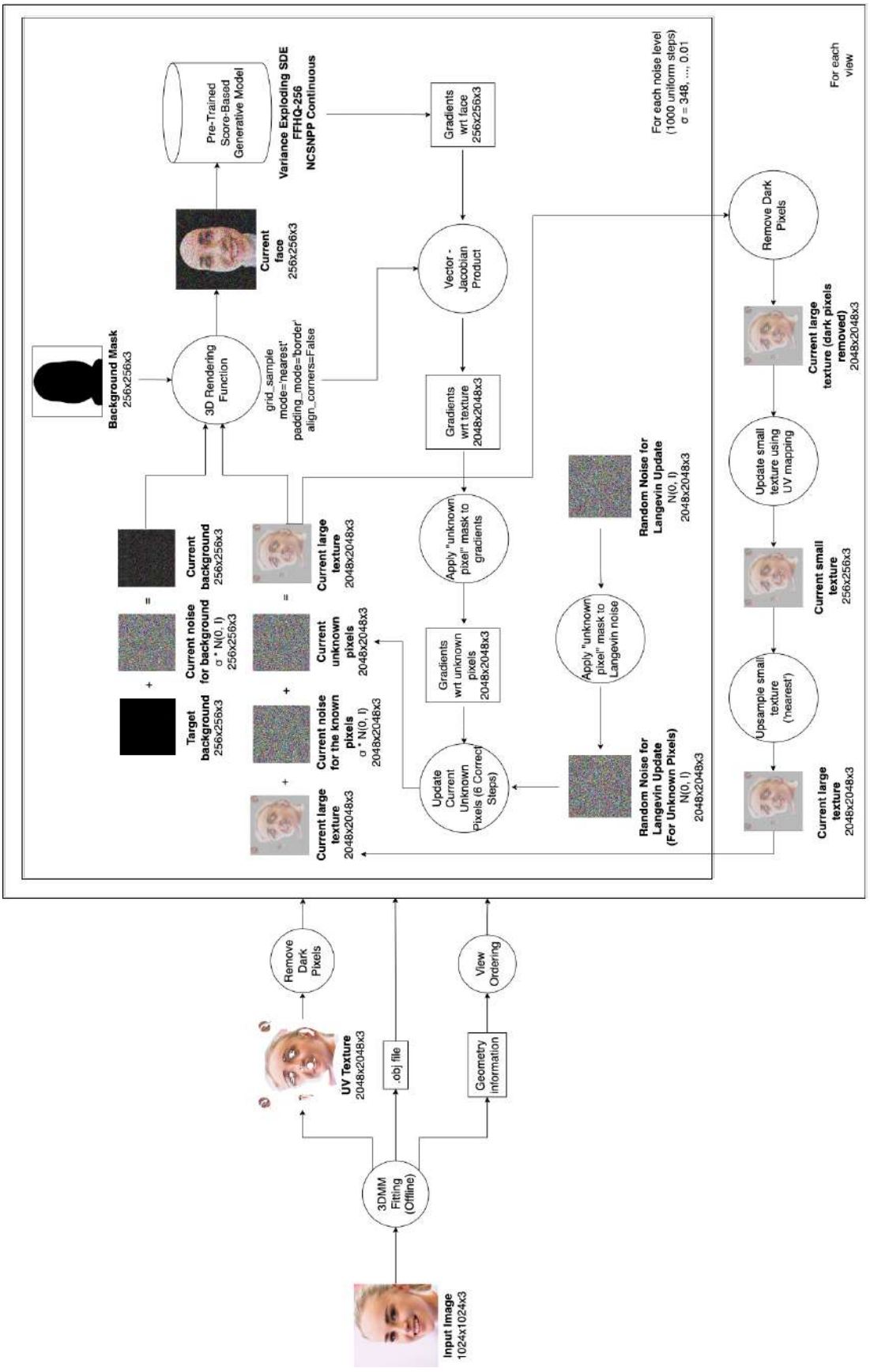

Figure 3.3: *Summary of optimization in the texture space*

Experimental Setup

We use PyTorch [Paszke et al. 2019] as the deep learning framework and PyTorch3D [Ravi et al. 2020] for neural rendering.

4.1 3DMM Fitting

To extract the 3D geometry information and create a face texture from a given input image, we need to use a 3DMM. DECA [Feng et al. 2021], 3DDFA [cleardusk 2022, Guo et al. 2020a, Zhu et al. 2019], and TF_FLAME [Bolkart 2022] are some of the options. 3DDFA does not generate a full head mesh, this causes a domain mismatch between the rendered face image and the images that the SGM was trained on, therefore the gradients from SGM are not able to optimize the texture in this case. Although DECA generates a full head mesh, it produces a texture with a maximum resolution of 256, however this causes the rendered faces to have a very low quality. We share some examples of rendered faces using DECA and TF_FLAME in Figure 4.1. Therefore, we use a modified version of the TF_FLAME repository. It outputs three files that will be used in the optimization process:

- An .obj file which has the 3D geometry information of the 2D face image,
- A face texture image,
- A .npy file which has the global rotation information in axis-angle representation.

Our modifications in the TF_FLAME repository are as follows:

- The original TF_FLAME repository expects the path of the 2D facial landmarks of the face image as input. In our version, we use a code snippet which detects these 2D facial

Figure 4.1: *Input FFHQ image (left), a mesh generated using DECA (middle) and a mesh generated using TF_FLAME (right). The face image generated using DECA has a lower quality. (Both images were inpainted using gradients from SGM.)*

Figure 4.2: *Image at the left shows the rendered face image when the mouth of the mesh is open. Image at the right shows the optimized texture. Optimizing the interior region of the mouth causes artifacts in the texture.*

landmarks using DLIB, therefore providing the path for these landmarks is not needed. [King 2009].

- As mentioned above, we output the global rotation information in axis-angle representation. We will use this information while determining the ordering of the views that the SGM will optimize.
- One of the outputs of the original repository is the shape details of a mesh whose mouth can be open. After running 3DMM fitting, we set the pose parameters for the jaw so that the mouth of the mesh is closed. This change is needed, because otherwise we would need to optimize the interior part of the mouth as well, and this increases the number of required optimization steps by a large amount. Also, optimizing the interior part of the mouth, causes the texture to have artifacts as shown in Figure 4.2.

TF_FLAME uses TensorFlow 1 [Abadi et al. 2015] and the repository of SGM uses TensorFlow 2. Since it is not straightforward to solve the dependency issues in this case, 3DMM fitting is done offline in both of our pipelines.

Before we input these textures as input to our optimization pipelines, we replace dark pixels in the texture with white pixels. We are doing this modification because when the texture has

pixels from the background due to a bad 3DMM fitting, the optimized pixels also have artifacts. Also, by removing dark pixels, we create small incomplete regions in the rendered face image, which can be usually inpainted successfully using the SGM, and gives us an opportunity to test our pipelines. While removing dark pixels, we create a grayscale version of the texture so that the pixel intensities are between 0 and 255. Then we replace the pixels which has a gray intensity less than or equal to 50, with a white pixel.

4.2 Pipelines

In both pipelines, the batch size is 1, number of iterations in the diffusion process (the number of different noise levels) is 1000, the minimum azimuth is -40, the maximum azimuth is 40, the minimum elev is -10, and the maximum elev is 10.

4.2.1 Consecutive Single-View Optimizations in the Image Space

We use a face image resolution of 1024x1024, and therefore we use the pre-trained SGM [Song 2022b] which was trained on FFHQ [Karras et al. 2018] images with resolution 1024x1024. We use a texture resolution of 512x512. The azimuth values we use while determining the views increase with a step size of 20. The elev values we use increase with a step size of 10. The signal-to-noise ratio (SNR) we use is 0.15. When SNR is too small, then the unoptimized pixels remain as noisy pixels. When it is too large, then the optimization does not converge, and we end up with a non-realistic result. We use only one "correct" step after each "predict" step. Doing optimization in the image space for a single texture takes approximately 4 hours on an NVIDIA Titan Xp 12196MiB.

4.2.2 Consecutive Single-View Optimizations in the Texture Space (Two Textures)

We use a face image resolution of 256x256, and therefore we use the pre-trained SGM [Song 2022c] which was trained on FFHQ [Karras et al. 2018] images with resolution 256x256. We use two textures with different resolutions. The small one has a resolution of 256x256 and the large one has a resolution of 2048x2048. The azimuth values we use while determining the views increase with a step size of 10. The elev values increase with a step size of 5. Using "predict" step in this pipeline, causes the unoptimized pixels to remain as noise at the end of the optimization process. Therefore, we follow the suggestion made by the authors of [Song et al. 2020], and we use multiple "correct" steps, and we do not use the "predict" step. In our experiments, we observed that using 6 "correct" steps is a good trade-off between quality of the inpainted regions and the running time. The SNR we use is 0.015. Doing optimization in the texture space for a single texture takes approximately 18 hours on an NVIDIA Titan Xp 12196MiB.

Results and Discussion

5.1 Failed Experiments

The initial idea was to do the optimization in the texture space, use a single texture, whether do consecutive single-view optimizations or multi-view optimization, and allow two settings: inpainting a given texture and sampling a novel texture. In this section, we discuss the failure points in the experiments which did not yield the results we expected.

5.1.1 Sampling Novel Textures

We tried generating novel textures using the gradients from SGM. The main difference between the pipeline we used in "Sampling Novel Textures" experiment and the pipelines we are proposing in ScoreFace is that we do not have a known region in the texture. Therefore, we are not decreasing noise of any part of the texture, but only decreasing noise of the background as the optimization process proceeds. This experiment did not work as expected because while we are doing a single-view optimization process for a given view, it is not guaranteed that the texture and the mesh will be aligned. For example, the gradients of SGM can fill the texture so that in the current view, the rendered image is a realistic face image, however the texture is actually not aligned with the mesh. For example, the texture could have nose pixels at the region which corresponds to the mouth of the mesh, and it can have mouth pixels at the region corresponding to the chin. When rendered, we will see a mouth under the nose, however they will not correspond to the real mouth and nose regions in the mesh. In Figure 5.1, we share an example where the textures and the meshes are not aligned.

Figure 5.1: *These images were generated during the "Sampling Novel Textures" experiment (see subsection 5.1.1). Optimizations were made in the texture space, and in both experiments we used the same mesh. Although the meshes are the same, the nose shape and length are different. This shows that the textures and the meshes are not aligned.*

Figure 5.2: *All images are produced by fitting 3DMM on an FFHQ image, and using 3D rendering on the outputs of the 3DMM fitting. The image at the left uses an unoptimized texture. The image in the middle was generated using an optimized texture with a resolution of 256x256 and the image at the right was generated using an optimized texture with a resolution of 1024x1024. When the texture resolution is smaller than 1536x1536, the degrees of freedom in the texture is not sufficient so that we can get a face image after rendering the texture.*

5.1.2 Consecutive Single-View Optimizations in the Texture Space (One Texture)

In one of our experiments, we tried using only one texture, and optimizing it. In this case, when the face image resolution is 256x256, the texture resolution should be at least 1536x1536. Otherwise, the degrees of freedom in the texture is not sufficient for the gradients of the SGM to inpaint the texture so that, when we render the texture, we can get a face image. In Figure 5.2, we show some examples when the texture resolution is lower than 1536x1536 and the inpainted parts of the texture remains as noise. In our experiments, we used a larger texture resolution (2048x2048) since it is the minimum resolution supported by TF_FLAME which is also larger than 1536x1536. When we run a single-view optimization, we observed that the rendered face image was inpainted for the optimized view successfully. However, when we rotated the mesh with a very small angle, we observed that the pixels that were optimized in the previous view are not being used in the new view, since the texture is too large in size. This caused the updates to the texture to be inconsistent among different views. In Figure 5.3, we share an inpainted face image and its rotated versions in 3D by small angles.

Inpainted DECA mesh

Rotated by an azimuth of 0.01

Rotated by an azimuth of 0.02

Rotated by an azimuth of 0.04

Rotated by an azimuth of 0.08

Rotated by an azimuth of 0.16

Figure 5.3: *Image at the top-left corner is created by first fitting DECA on an FFHQ image, then upsampling the resulting texture to 2048x2048, and inpainting it using gradients from SGM. Other images are produced by rotating the mesh in this image in 3D by small angles. While we are doing consecutive single-view optimizations with a single texture of resolution 2048x2048, whether we need to do too many single-view optimizations, or we cannot use the texture pixels that were optimized in a previous single-view optimization.*

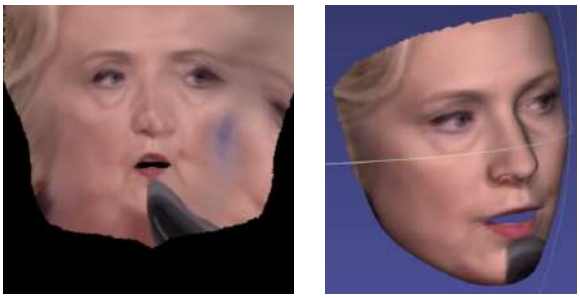

Figure 5.4: *A texture generated by BFM (left) and a mesh generated by BFM (right). Images were taken from 3DDFA_v2 Github repository ([Guo et al. 2020b], [Guo et al. 2018]).*

Figure 5.5: *Blue pixels in the image at the left are the pixels used as input to the warping function. The image in the middle shows the warping result for the texture. The image at the right shows the rendered face image when the mesh is rotated in 3D.*

To solve this problem, we first tried the following: Do a single-view optimization to optimize texture pixels, then apply warping on the texture pixels so that the pixels between the optimized pixels are also filled with an interpolation. If we were using a 3DMM which uses Basel Face Model (BFM) [Gerig et al. 2017] (such as 3DDFA [cleardusk 2022, Guo et al. 2020a, Zhu et al. 2019]), this would have worked. [1] However, as we mentioned before, SGM cannot optimize the texture while we are using BFM, since BFM does not produce a full head mesh. An example texture and mesh generated using BFM can be found in Figure 5.4. Warping did not work with TF_FLAME either, because of the following: As we mentioned before, we can optimize a single texture of resolution 2048x2048, from a single-view successfully. However, pixels that are adjacent in the face image are not always adjacent in the texture when the texture resolution is large. For example, in a given view, while a part of the right ear and a part of the right cheek of a mesh are visible, some parts of the right cheek might be invisible. This means, in the texture there will be an incomplete region due to the invisible parts of the right cheek. When we use "warping", this region is interpolated using the pixels on the ear and visible parts of the cheek. This causes the pixels which correspond to the invisible region of the right cheek to be filled with pixels that are actually far away from each other, therefore the warping causes artifacts on this filled region. We share an example where this problem occurs in Figure 5.5.

Since we could not make warping work, we used two textures instead of one. The details are explained in subsection 4.2.2.

[1] An example where "warping" worked with BFM can be found here. `https://github.com/apple2373/mediapipe-facemesh/blob/main/main.ipynb`.

5.1.3 Multi-View Optimization in the Texture Space (One Texture)

For a given noise level, calculate the gradients with respect to the unknown texture pixels that are contributing to the face image in the current view and optimize these texture pixels. Keep the noise level as the same, rotate the mesh, and optimize the texture pixels that are contributing to the face image in the new view. Once all views are optimized, decrease the noise level, and do the multi-view optimization in the new noise level.

There are at least four variations of this experiment, and we had time to do only the first one mentioned below:

- For each new view, optimize an unknown texture pixel that is contributing to the current view, even if it was already optimized before in the current noise level.
- For each new view, optimize an unknown texture pixel that is contributing to the current view, only if it was not optimized in the current noise level yet.
- For each new view, save the gradients with respect to the unknown texture pixels that are contributing to the current view, but do not optimize the texture pixels yet. After the gradients are saved for each view, optimize each unknown texture pixel using the mean of the gradients that belong to this texture pixel.
- For each new view, save the gradients with respect to the unknown texture pixels that are contributing to the current view, but do not optimize the texture pixels yet. After the gradients are saved for each view, optimize each unknown texture pixel using the gradient with respect to this pixel that was calculated in the view where the distance between the corresponding face pixel and the camera was minimum.

5.2 Results

5.2.1 Consecutive Single-View Optimizations in the Image Space

Example results can be found in Figure 5.6, Figure 5.7, Figure 5.8, and Figure 5.9.

5.2.2 Consecutive Single-View Optimizations in the Texture Space (Two Textures)

Example results can be found in Figure 5.10, Figure 5.11, Figure 5.12, and Figure 5.13.

5.3 Limitations

As it can be observed in the results shared in subsection 5.2.1 and subsection 5.2.2, both pipelines are successful in inpainting small regions such as the white regions on the eyes and eyebrows. The pipelines are also successful in inpainting the incomplete regions which can be

Figure 5.6: *Image space optimization results for FFHQ image with ID 33673.*

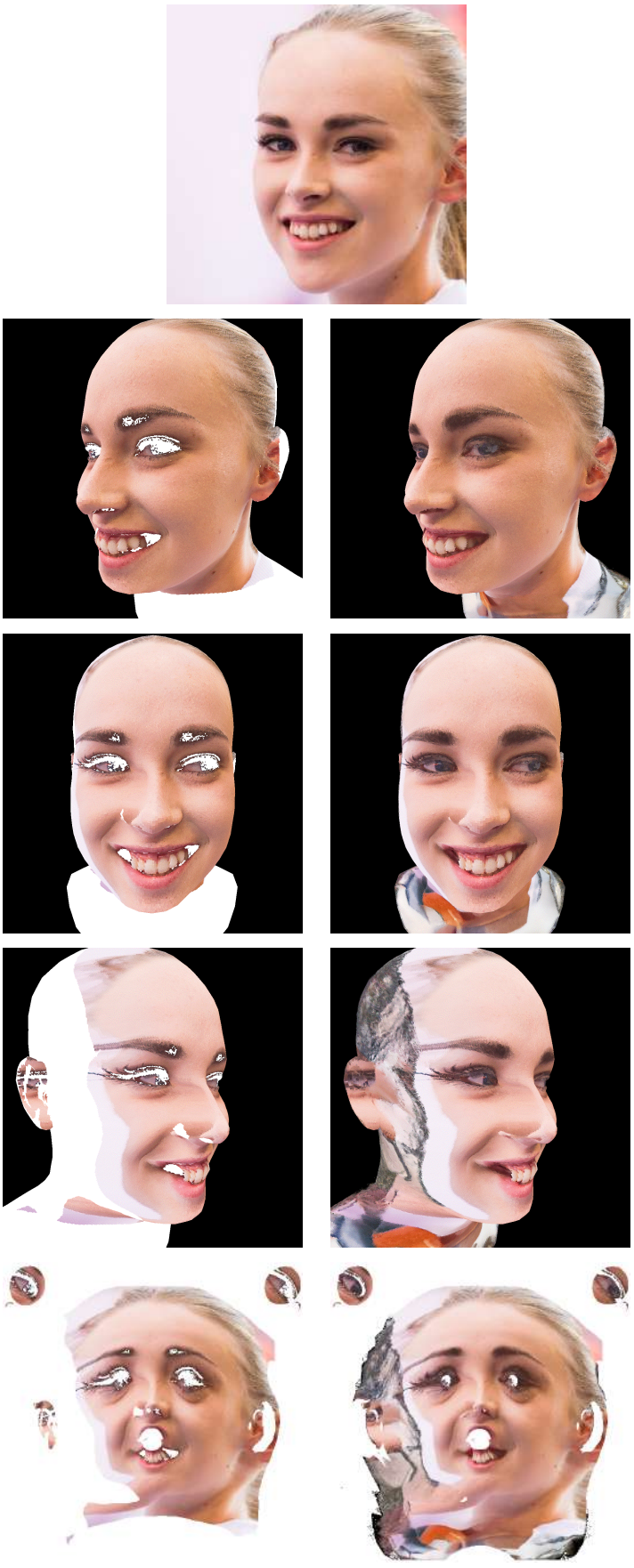

Figure 5.7: *Image space optimization results for FFHQ image with ID 58232. The white region on the left side of the face is because of bad 3DMM fitting. This causes the inpainted regions on the left side of the face to have artifacts.*

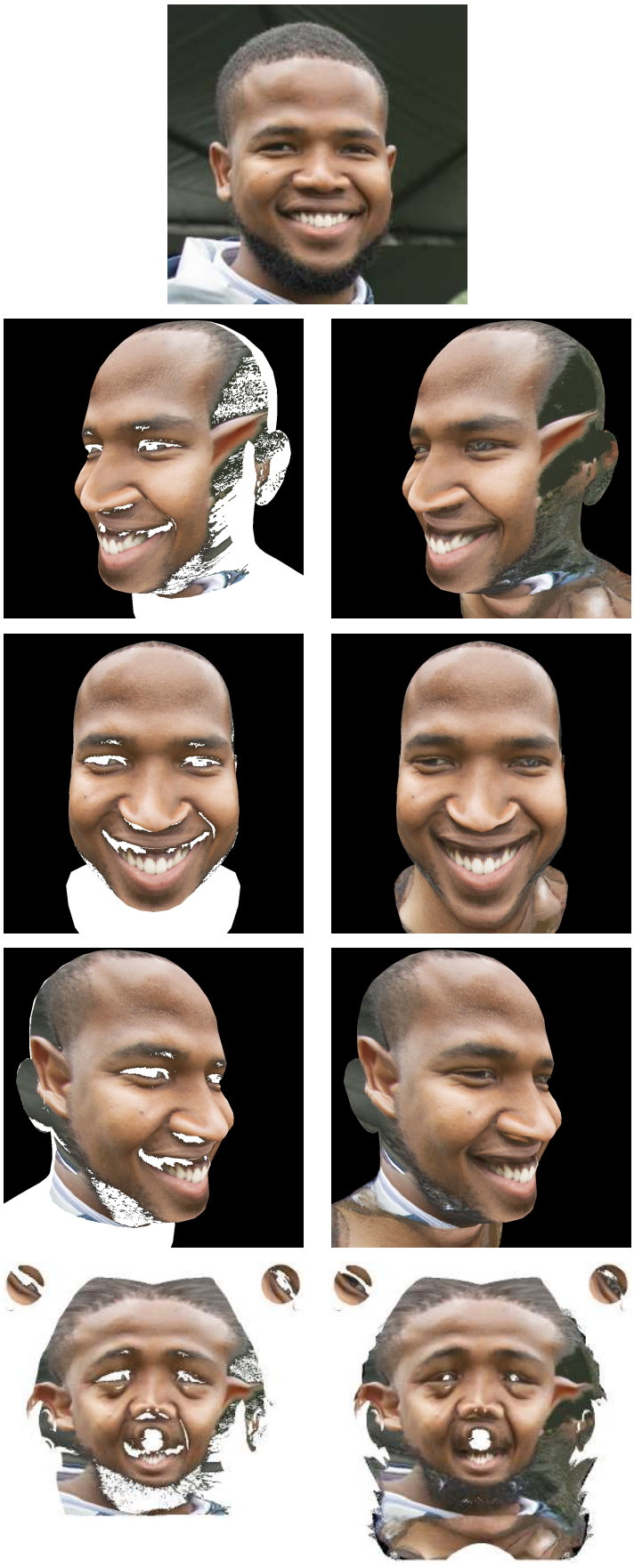

Figure 5.8: *Image space optimization results for FFHQ image with ID 60527.*

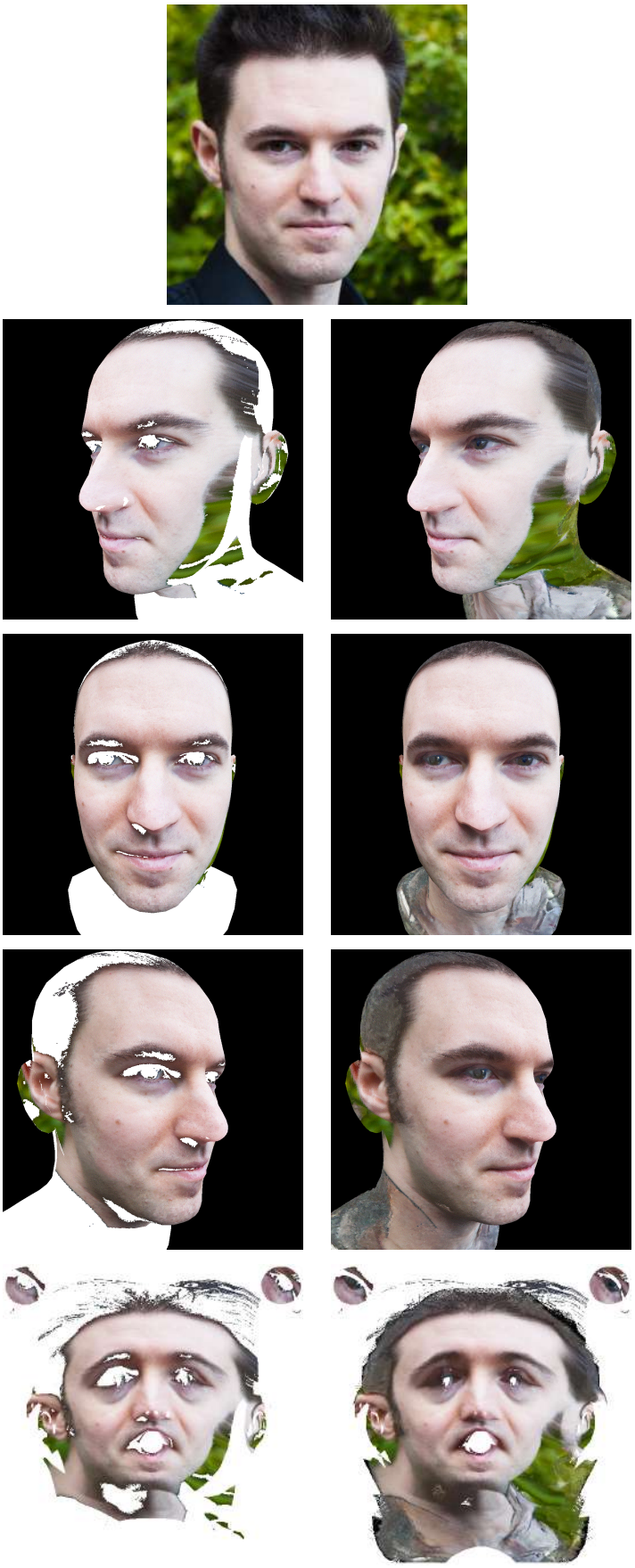

Figure 5.9: *Image space optimization results for FFHQ image with ID 63119. The green region on the right side of the face is because of bad 3DMM fitting. This causes the green area to get larger as a result of the optimization.*

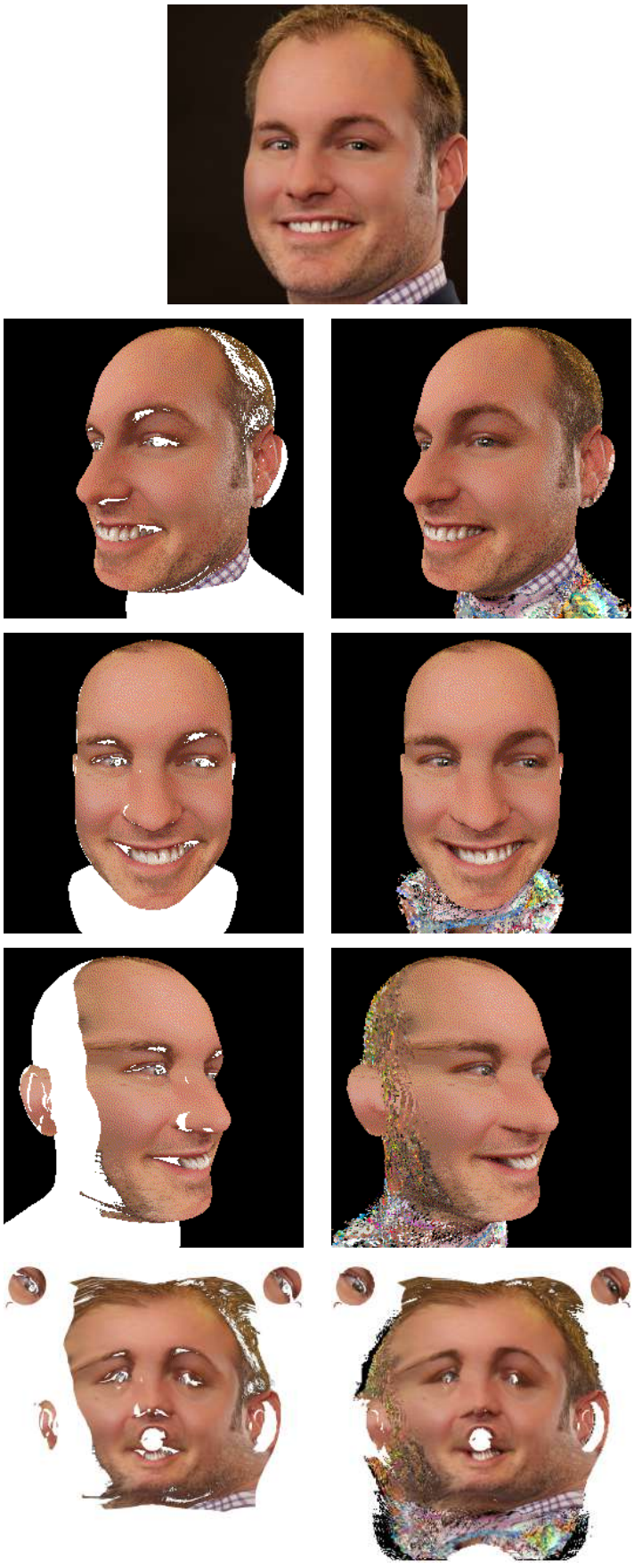

Figure 5.10: *Texture space optimization results for FFHQ image with ID 33673.*

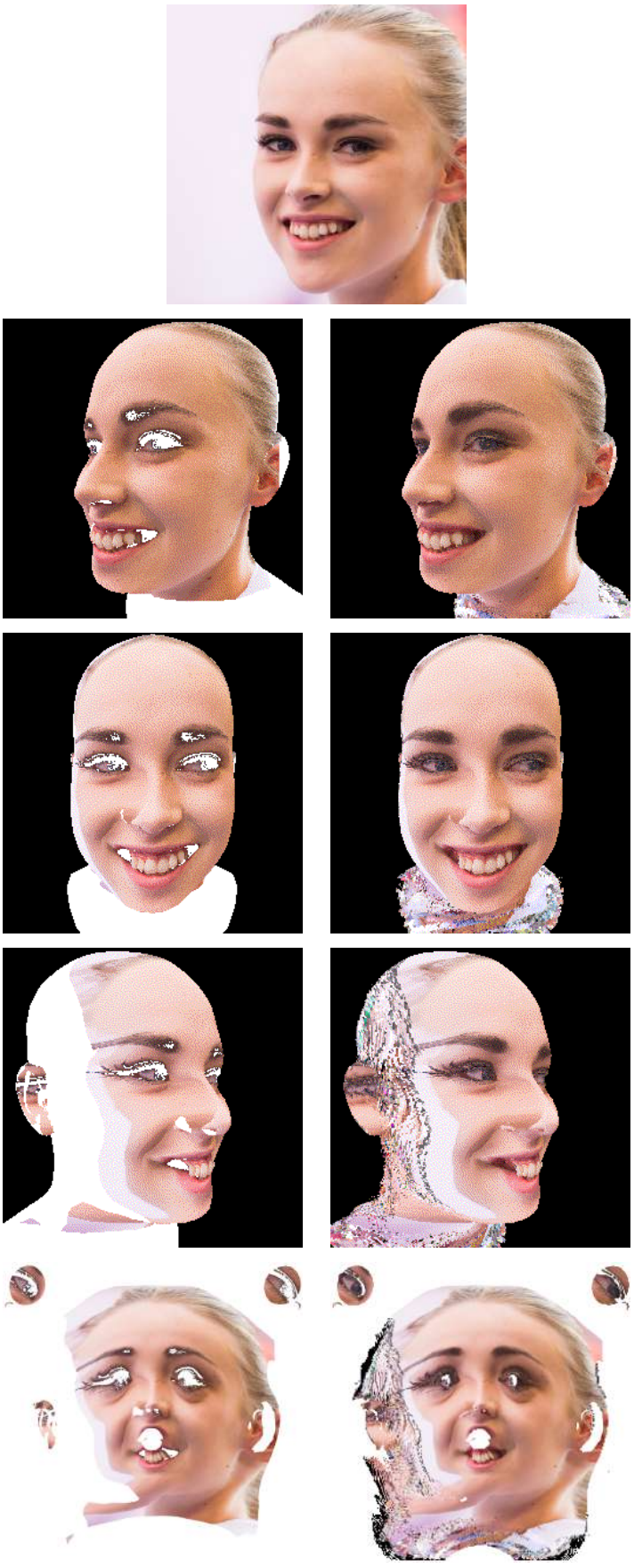

Figure 5.11: *Texture space optimization results for FFHQ image with ID 58232. The white region on the left side of the face is because of bad 3DMM fitting. This causes the inpainted regions on the left side of the face to have artifacts.*

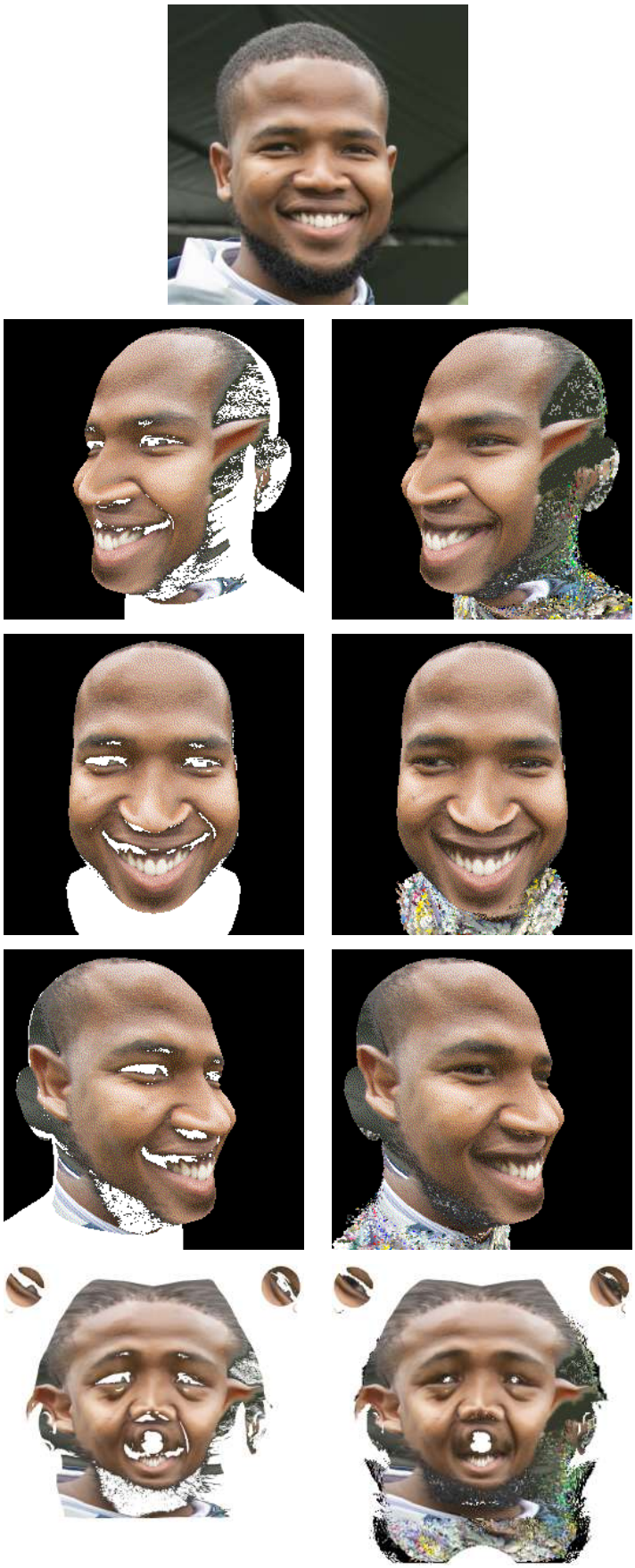

Figure 5.12: *Texture space optimization results for FFHQ image with ID 60527.*

Figure 5.13: *Texture space optimization results for FFHQ image with ID 63119. The green region on the right side of the face is because of bad 3DMM fitting. This causes the green area to get larger as a result of the optimization.*

seen at a view that is close to the original view in the input image. However, they are both unsuccessful in inpainting regions which can be seen at a view that is far from the view in the input image. This might be because of two things:

- The filled regions of the opposite side of the face might not be as realistic as the original view.
- Since 3DMM fitting is not always perfect, there might be background pixels in the unoptimized texture. These pixels might cause larger artifacts in the optimized face image.

One other limitation is that even though we use view ordering, there is no consistency among different views.

5.4 Future Work

For the texture inpainting task, future work may focus on the last three experiments mentioned in subsection 5.1.3. One can also try to find a good heuristic for the selection of input pixels for the warping function mentioned in subsection 5.1.2. The pixels that will be used while interpolating can be chosen based on a rule so that warping works as expected.

One other possible direction for future research on the texture inpainting task is to improve the pipeline so that the optimized face is less affected by the errors due to the 3DMM fitting, which are mentioned in section 5.3. This might be done by using gradients of SGM during 3DMM fitting as well.

For the novel texture sampling task, one can generate a novel face image using gradients from the SGM, fit 3DMM to it, and then use one of the pipelines we are proposing. This would be a solution to the problem mentioned in subsection 5.1.1.

Conclusion

To train an accurate face verification network, the training dataset should have sufficient pose variety. Moreover, frontalizing the test images also increase the accuracy of the network, since the discrepancy between the train and the test images will be reduced. We use a 3DMM on a 2D face image to extract 3D face geometry information and a face texture. Outputs of 3DMM are used as input to our pipelines: the first one inpaints a texture by doing an optimization in the texture space, and the second one does the optimization in the image space. We show that both pipelines are successful at inpainting incomplete regions when the optimized view is close to the original view in the input face image, and they are not successful at inpainting incomplete regions when the optimized view and the original view are not close to each other.

Bibliography

ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., AND ZHENG, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

BOLKART, T., 2022. Tf_flame.

CLEARDUSK, 2022. Face alignment in full pose range: A 3d total solution.

DENG, J., CHENG, S., XUE, N., ZHOU, Y., AND ZAFEIRIOU, S., 2017. Uv-gan: Adversarial facial uv map completion for pose-invariant face recognition.

FENG, Y., FENG, H., BLACK, M. J., AND BOLKART, T. 2021. Learning an animatable detailed 3D face model from in-the-wild images. vol. 40.

GECER, B., DENG, J., AND ZAFEIRIOU, S. 2021. OSTeC: One-shot texture completion. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE.

GERIG, T., FORSTER, A., BLUMER, C., EGGER, B., LÜTHI, M., SCHÖNBORN, S., AND VETTER, T. 2017. Morphable face models - an open framework. *CoRR abs/1709.08398*.

GOODFELLOW, I. J., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y., 2014. Generative adversarial networks.

GRENANDER, U., AND MILLER, M. I. 1994. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society. Series B (Methodological) 56*, 4, 549–603.

GROSS, R., MATTHEWS, I., COHN, J., KANADE, T., AND BAKER, S. 2008. Multi-pie. In *2008 8th IEEE International Conference on Automatic Face Gesture Recognition*, 1–8.

GUO, J., ZHU, X., AND LEI, Z., 2018. 3ddfa. `https://github.com/cleardusk/3DDFA`.

GUO, J., ZHU, X., YANG, Y., YANG, F., LEI, Z., AND LI, S. Z., 2020. Towards fast, accurate and stable 3d dense face alignment.

GUO, J., ZHU, X., YANG, Y., YANG, F., LEI, Z., AND LI, S. Z. 2020. Towards fast, accurate and stable 3d dense face alignment. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

HEUSEL, M., RAMSAUER, H., UNTERTHINER, T., NESSLER, B., AND HOCHREITER, S. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium.

HYVÄRINEN, A. 2005. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research 6*, 24, 695–709.

JING, B., CORSO, G., BERLINGHIERI, R., AND JAAKKOLA, T., 2022. Subspace diffusion

generative models.

KARRAS, T., LAINE, S., AND AILA, T., 2018. A style-based generator architecture for generative adversarial networks.

KARRAS, T., LAINE, S., AITTALA, M., HELLSTEN, J., LEHTINEN, J., AND AILA, T., 2019. Analyzing and improving the image quality of stylegan.

KING, D. E. 2009. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research 10*, 1755–1758.

KINGMA, D. P., AND WELLING, M., 2013. Auto-encoding variational bayes.

KRIZHEVSKY, A. 2009. Learning multiple layers of features from tiny images. Tech. rep.

MSEITZER, 2022. Fid score for pytorch.

NA, I., TRAN, C., NGUYEN, D., AND SANG, D. 2020. Facial uv map completion for pose-invariant face recognition: a novel adversarial approach based on coupled attention residual unets. *Human-centric Computing and Information Sciences 10* (12).

PARISI, G. 1981. Correlation functions and computer simulations. *Nuclear Physics B 180*, 3, 378–384.

PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L., DESMAISON, A., KOPF, A., YANG, E., DEVITO, Z., RAISON, M., TEJANI, A., CHILAMKURTHY, S., STEINER, B., FANG, L., BAI, J., AND CHINTALA, S. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 8024–8035.

RAVI, N., REIZENSTEIN, J., NOVOTNY, D., GORDON, T., LO, W.-Y., JOHNSON, J., AND GKIOXARI, G. 2020. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*.

REZENDE, D. J., AND MOHAMED, S., 2015. Variational inference with normalizing flows.

RONNEBERGER, O., FISCHER, P., AND BROX, T. 2015. U-net: Convolutional networks for biomedical image segmentation. vol. 9351, 234–241.

SALIMANS, T., GOODFELLOW, I. J., ZAREMBA, W., CHEUNG, V., RADFORD, A., AND CHEN, X. 2016. Improved techniques for training gans. *CoRR abs/1606.03498*.

SCHLEMPER, J., OKTAY, O., SCHAAP, M., HEINRICH, M. P., KAINZ, B., GLOCKER, B., AND RUECKERT, D. 2018. Attention gated networks: Learning to leverage salient regions in medical images. *CoRR abs/1808.08114*.

SONG, Y., GARG, S., SHI, J., AND ERMON, S., 2019. Sliced score matching: A scalable approach to density and score estimation.

SONG, Y., SOHL-DICKSTEIN, J., KINGMA, D. P., KUMAR, A., ERMON, S., AND POOLE, B., 2020. Score-based generative modeling through stochastic differential equations.

SONG, Y., 2022. Generative modeling by estimating gradients of the data distribution.

SONG, Y., 2022. Pre-trained sgm for resolution 1024.

SONG, Y., 2022. Pre-trained sgm for resolution 256.

SONG, Y., 2022. Score-based generative modeling through stochastic differential equations.

TANG, Z., PENG, X., GENG, S., ZHU, Y., AND METAXAS, D. N., 2018. Cu-net: Coupled u-nets.

VINCENT, P. 2011. A connection between score matching and denoising autoencoders. *Neural Computation 23*, 7, 1661–1674.

WANG, Z., BOVIK, A., SHEIKH, H., AND SIMONCELLI, E. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4, 600–612.

ZHU, X., LIU, X., LEI, Z., AND LI, S. Z. 2019. Face alignment in full pose range: A 3d total solution. *IEEE Transactions on Pattern Analysis and Machine Intelligence 41*, 1 (jan), 78–92.