ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Deep Learning

EXPLORING LATENT EMBEDDINGS IN DIFFUSION MODELS FOR FACE ORIENTATION CONDITIONING

CANDIDATE
Antonio Guerra

SUPERVISOR
Prof. Andrea Asperti

Academic year 2022-2023

Session 1st

Abstract

Facial rotation is a critical task in computer vision with numerous applications across various domains. Traditional facial rotation techniques, such as reconstruction-based and 3D geometry-based approaches, require ground truth data for training, which presents limitations. This thesis proposes two novel techniques for addressing the facial rotation problem without relying on pairs of images. These techniques leverage the latent space of a DDIM trained to reconstruct human faces and condition the generation process to produce rotated faces. A comprehensive overview of facial rotation techniques and Diffusion Models (DMs) is provided, along with the development, implementation, and evaluation of the newly proposed methods. The first method modifies specific input image pixels, while the second technique fits a linear regressor to sample from the latent space. The second method demonstrates better stability and simplicity, effectively producing rotations of up to $\pm 30°$. Future research directions include using unbiased datasets with greater face orientation variation and improving the second method's computational efficiency.

Contents

1 Introduction **1**

2 Background **4**

2.1 Diffusion Models for Image generation 4

2.1.1 Diffusion Models Overview 4

2.1.2 Forward diffusion process 5

2.1.3 Reverse diffusion process 7

2.1.4 Training and Loss function 8

2.1.5 Variance schedule 10

2.1.6 Model architecture 11

2.1.7 Denoising Diffusion Implicit Model 12

2.2 Conditioned image generation 15

2.2.1 Classifier Guidance 15

2.2.2 Classifier-free Guidance 16

2.3 CelebFaces Attributes Dataset 17

2.4 Exploring Embeddings in Denoising Diffusion Models 18

3 Related Works **20**

3.1 Face Rotation before Deep Learning 20

3.2 Face Rotation after Deep Learning 21

3.2.1 Latent Space-based Techniques 21

3.2.2 Reconstruction-Based Techniques 21

3.2.3 3D Geometry-Based Techniques 22

3.2.4 Face Rotation with Neural Radiance Fields 22

4 Proposed Method **24**

4.1 DDIM and Embedding Models 24

4.2 Dataset Preparation 25

4.2.1 Analysis of Annotations 25

4.2.2 Light Direction Analysis 26

4.2.3 Analyzing Face Orientation 28

4.3 Preprocessing 31

4.4 Postprocessing 32

4.5 Filtering CelebA images 35

4.6 Method 1: Pixel-based conditioning on Input Image 38

4.7 Method 2: Linear Regression and latent interpolation 44

4.8 Differences between the two methods 51

5 Conclusions **53**

A Slope computation analysis **55**

A.1 Slope Stability analysis 55

A.2 Slope Similarity Analysis among different Images 58

A.3 Effects of Attribute Removal on Slope Computation 60

Bibliography **62**

Acknowledgements **67**

List of Figures

1.1 Reification of the portrait of Señora Sabasa Garcia by Francisco Goya (c. 1806/1811) 2

1.2 Output of the two methods to perform facial rotation on a CelebA sample. 2

2.1 Overview of the Diffusion Process for image generation 5

2.2 The Markov chain of forward diffusion process 5

2.3 Reverse diffusion process 7

2.4 Comparison of linear (top) and cosine (bottom) variance schedulers. Source: *Improved denoising diffusion probabilistic models*[7] 11

2.5 The U-Net architecture. Source: *U-Net: Convolutional Networks for Biomedical Image Segmentation*[8] 12

2.6 Random samples from CelebA Aligned dataset. 18

2.7 Examples of embedding for the CelebA dataset. The first row shows the original images, the second row displays the synthesized latent seed, and the third row presents the reconstructed images.
Source: *Image Embedding for Denoising Generative Models* [14] 19

4.1 Distribution of CelebA attributes 26

4.2 Distribution of light direction on CelebA 27

4.3 Random sample from CelebA for each light direction 27

4.4 Yaw, Pitch and Roll angles in Head Pose Estimation 28
4.5 Figure (a) displays all facial landmarks, while Figure (b) shows a reduced set used for head pose estimation. 29
4.6 Example of results of head pose estimation on CelebA. The estimated yaw is indicated by the green color, pitch by blue, and roll by red. 30
4.7 Yaw distribution on CelebA dataset 30
4.8 In Figure (a), the original CelebaMask-hq masks are displayed, while Figure (b) shows the cropped version with unified masks used to train the segmentation model. 31
4.9 Architecture of the proposed super-resolution model 32
4.10 Output of the proposed Super-Resolution model. Using Google Colaboratory, the model generates three predictions in less than 1 second. 33
4.11 Output of the CodeFormer model. Using Google Colaboratory, the model generates three predictions in about 30 seconds. 34
4.12 The top row displays the mean images for specified yaw angles when the male attribute is -1. The bottom row shows the mean images for specified yaw angles when the male attribute is 1. 36
4.13 Figure (a) displays the mean images for different light directions, while Figure (b) shows the images for different values of the smiling attribute. 37
4.14 Pipeline of generation process using the Embedding Model and the DDIM. 38
4.15 Computation of the raw correction on the first image of CelebA. 39
4.16 Figure (a) shows the computation of the adapted mean for the starting orientations, while Figure (b) shows the computation of the adapted mean for the target orientations. 40
4.17 Computation of the final correction. 41

4.18 Final generation step 42
4.19 Method 1 applied with additional steps on 3 samples from CelebA. 42
4.20 Image 25000 from CelebA dataset: the original image (left), the pre-processed image (center), and the resized version (right) used as input. 44
4.21 This image demonstrates the process of computing the root point 0, based on a subset of attributes including 'male', 'gender', and 'smiling'. These attributes were used to get similar images from CelebA. 45
4.22 In Figure (a), the root point is computed using the embedding of the mean image, while in Figure (b), it is computed as the average of the embeddings of similar images. 46
4.23 The full set of root points that were computed for the left direction. 46
4.24 Visualization of all the components involved in the second method using the PCA with 2 principal components. 49
4.25 This image shows the result of sampling using a linear regressor with root points that extend towards the left direction. . . . 49
4.26 This image shows the result of sampling using a linear regressor with root points that extend towards the right direction. . . 50
4.27 This figure compares the outputs of the two methods on three CelebA images. The input image is in the middle, with the first method's output on the top row and the second method's output on the bottom row. 52

A.1 First attribute subset cosine similarity heatmap. 56
A.2 Second attribute subset cosine similarity heatmap. 56
A.3 Third attribute subset cosine similarity heatmap. 57
A.4 Fourth attribute subset cosine similarity heatmap. 57

A.5 Cosine similarities heatmap between different images 59
A.6 Images used for attribute utility analysis. 60

List of Tables

2.1 FID scores on CIFAR10 and CelebA datasets by diffusion models of different settings. Source: *Denoising diffusion implicit models* [9] 14

A.1 Results of left slopes cosine similarities. 61

A.2 Results of right slopes cosine similarities. 61

List of Algorithms

1 Color correction 35
2 Filtering on CelebA 37
3 Method 1 pseudocode 43
4 Method 2: Root points computation 47
5 Method 2 pseudocode 50

Chapter 1

Introduction

The generation of realistic images has long been a challenging problem in the field of machine learning that has recently become a hot topic thanks to a vast number of successful applications proposed in the literature. One approach that has shown promise is the use of diffusion models, which can generate high-quality images by iteratively refining a Gaussian noise via a Denoising process. The latent space of these models, which represents the underlying structure of the generated images, could play a crucial role in the quality and diversity as well as for the conditioning of the generated images.

The focus of the work conducted is to investigate the latent space of diffusion models for image generation. Specifically, the aim is to explore the properties of the latent space and how it can be manipulated to generate images with desired characteristics. This investigation will involve analyzing the structure of the latent space, developing techniques for manipulating it, and evaluating the impact of these manipulations on the generated images.

Specifically, two main applications of conditional generation were explored:

1. Embedding human portraits into the latent space to feed a diffusion model, trained to generate real human faces, to produce the most likely real approximation of input portraits, we call this operation ”reification”.

An output example is given in Figure 1.1

Reification Input Reification Output

Figure 1.1: Reification of the portrait of Señora Sabasa Garcia by Francisco Goya (c. 1806/1811)

2. Exploit the latent space of a diffusion model trained to reconstruct human faces and condition the generation process to produce rotated faces. Two methods were proposed: the first involves modifying specific pixels in the input image fed to the diffusion model to condition the generation, and the second fitting a linear regressor to sample from the latent space the rotated faces. The corresponding output of the two methods are shown in figure 1.2

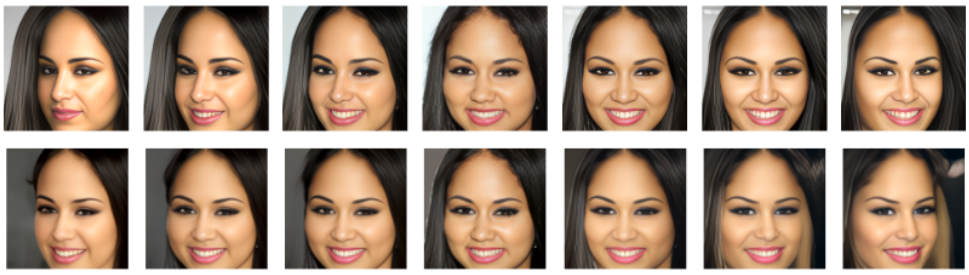


Figure 1.2: Output of the two methods to perform facial rotation on a CelebA sample.

The focus of this thesis will be to condition the generation process in order to produce rotated faces. Facial rotation is a crucial task in computer vision

with a wide range of applications in security, entertainment, and healthcare. In recent years, deep learning techniques have revolutionized the field of facial rotation, enabling the development of more accurate and efficient methods.

Traditional facial rotation techniques, including the reconstruction-based approach and the 3D geometry-based approach, have advantages and limitations. However, both approaches require ground truth data for training, which consists of pairs of images captured from various angles of a specific person, along with a frontal image of that person.

To address this limitation, this work proposes two novel techniques for solving the facial rotation problem without relying on pairs of images. Both techniques explore the latent space of a DDIM trained to reconstruct human faces and condition the generation process to produce rotated faces.

This thesis provides a theoretical background on Diffusion Models (DMs), which are a class of models that can revert the diffusion process by iteratively adding Gaussian noise to an image. The text also discusses some works, such as DDPM, that have successfully implemented DMs by simplifying their loss function. Furthermore, the evolution of DMs across recent years is explored, both from an architectural and theoretical perspective, in order to improve the sampling speed and the quality of generated samples.

In addition, this thesis provides an overview of facial rotation techniques developed over time and categorizes them into two groups: those created before the emergence of deep learning techniques and those developed afterward. Finally, this thesis covers the development, implementation, and evaluation of these novel facial rotation techniques. We also discuss future research directions in this area.

Chapter 2

Background

2.1 Diffusion Models for Image generation

Image generation is an essential task in computer vision and artificial intelligence, with applications such as generative art, content creation, image editing, and data augmentation. One of the prevailing techniques for image generation is the use of generative models, which learn the underlying data distribution and use it to produce new samples that resemble the training data.

2.1.1 Diffusion Models Overview

Diffusion Models have emerged as a promising approach to image generation, based on the principles of the Diffusion Process, a stochastic process that describes the random motion of particles in a fluid or gas.

One of the significant advantages of Diffusion Models is their ability to generate high-quality images with realistic textures and fine details. The diffusion process enables the model to capture the complex statistical dependencies between pixels in an image, a challenging task for other generative models. This makes Diffusion Models a promising approach to image generation.

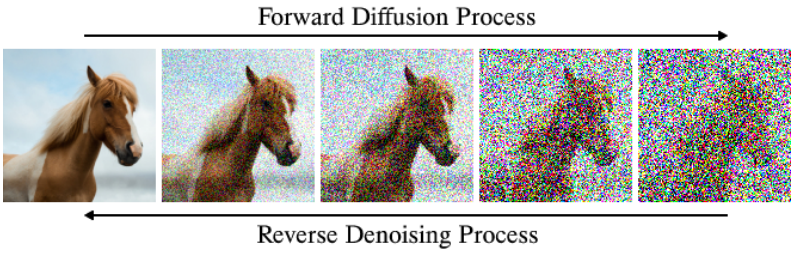


Figure 2.1: Overview of the Diffusion Process for image generation

Numerous generative models based on diffusion have been proposed, sharing similar underlying ideas. These models include: *deep unsupervised learning using Nonequilibrium Thermodynamics* [1], *noise-conditioned score network (NCSN)* [2], and *denoising diffusion probabilistic models (DDPM)* [3].

In the context of image generation, the Diffusion Model defines a Markov chain of diffusion steps that slowly adds random noise to the data, and learns to reverse the diffusion process to construct desired data samples from the noise. Unlike other generative models like VAEs or flow models, diffusion models are learned with a fixed procedure, and the latent variable has high dimensionality, the same as the original data.

2.1.2 Forward diffusion process

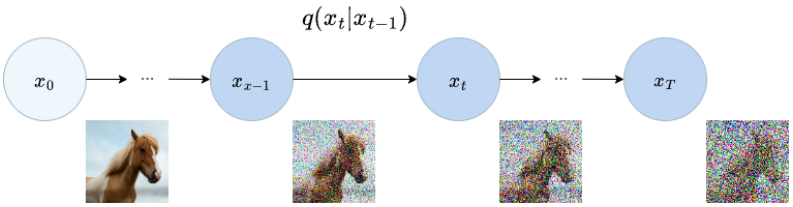


Figure 2.2: The Markov chain of forward diffusion process

As mentioned previously, the first step in generating images using diffusion models is to perform a *forward diffusion process*. This involves taking a data point sampled from a real data distribution, denoted as $x_0 \sim q(X)$, and adding

small amounts of Gaussian noise to it in a series of steps. The resulting sequence of noisy samples is denoted as $x_1, \ldots, x_T$. The step sizes are determined by a variance schedule $\beta_1, \ldots, \beta_T$, which controls how much noise is added at each step. The forward diffusion process can be mathematically described by the following equation:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad (2.1)$$

The figure 2.2 illustrates how the Diffusion Process gradually degrades the distinctive features of a data sample x_0 as the step t becomes larger. Eventually, as T approaches infinity, the resulting image x_T becomes indistinguishable from an isotropic Gaussian distribution.

The Diffusion Process, despite causing degradation of the image features over time, has a valuable property that enables the generation of a noisy image at any given time step t with a single step. This is possible due to the Gaussian distribution of the added noise, which allows for the pre-composition of various noise functions while maintaining a Gaussian distribution. To achieve this, we require the following premises:

$$\text{let} \quad \alpha_t = 1 - \beta_t \quad \text{and} \quad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i \quad (2.2)$$

thanks to 2.2 we can derive x_t as:

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} && \text{;where } \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \cdots \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} && \text{;where } \bar{\boldsymbol{\epsilon}}_{t-2} \text{ merges two Gaussians (*).} \\
&= \ldots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}
\end{aligned} \quad (2.3)$$

finally, we can obtain the formula for sampling x_t at an arbitrary time step t as:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2.4}$$

2.1.3 Reverse diffusion process

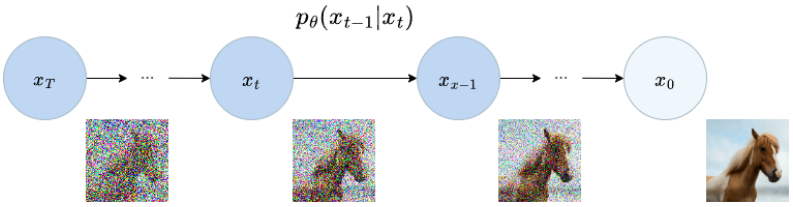


Figure 2.3: Reverse diffusion process

In contrast to the forward diffusion process, the reverse diffusion process aims to remove noise from a noisy image and generate a clear output. However, accurately estimating $q(x_{t-1}|x_t)$ is not a simple task, as it requires using the entire dataset. Therefore, we must rely on deep learning models to approximate these conditional probabilities, allowing us to execute the reverse diffusion process effectively. We will define the deep learning model p_θ as:

$$\begin{aligned} p_\theta(x_{0:T}) &= p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \\ p_\theta(x_{t-1}|x_t) &= \mathcal{N}(x_{t-1}; \boldsymbol{\mu}_\theta(x_t,t), \boldsymbol{\Sigma}_\theta(x_t,t)) \end{aligned} \tag{2.5}$$

One notable feature is that the reverse conditional probability is tractable when conditioned on x_0:

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\boldsymbol{\mu}}(x_t, x_0), \tilde{\Sigma}_t\mathbf{I}) \tag{2.6}$$

Finally, as derived in *Understanding Diffusion Models: A Unified Perspective* [4], we can conclude that:

$$\begin{aligned} \tilde{\boldsymbol{\mu}}(x_t, x_0) &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 \\ \tilde{\Sigma}_t = \tilde{\beta}_t &= \tilde{\sigma}_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\cdot\beta_t \end{aligned} \tag{2.7}$$

given that we have access to both the original image X_0 and the noisy images at X_t and X_{t-1} during the training phase, we are able to train a model on this specific distribution.

Furthermore, utilizing Equation 2.3, we can express x_0 as $\frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon})$. Subsequently, by applying Equation 2.7, we arrive at:

$$\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}\Big) = \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}\Big) \tag{2.8}$$

as our target $\tilde{\boldsymbol{\mu}}_t$ now only depends on x_t, we can employ a neural network to approximate ϵ and, in turn, approximate $\tilde{\boldsymbol{\mu}}_t$ as well:

$$\tilde{\boldsymbol{\mu}}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big) \tag{2.9}$$

2.1.4 Training and Loss function

By analyzing the situation from a broader perspective, it becomes apparent that the pairing of q and p resemblance to that of a Variational Autoencoder (VAE) [5]. As a result, we can employ the variational lower bound technique to enhance the optimization of the negative log-likelihood.

According to [6], it can be demonstrated after some mathematical manipulation that:

$$\begin{aligned} L_{\text{VLB}} =& \mathbb{E}_q[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T} \\ &+ \sum_{t=2}^{T} \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \\ &- \underbrace{\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}] \end{aligned} \tag{2.10}$$

analyzing these terms, we can observe that:

- The $\mathbf{L_0}$ term can be interpreted as a reconstruction term, similar to the one found in the evidence lower bound (ELBO) of a variational autoencoder. In [3], this term is learned using a separate decoder;
- $\mathbf{L_T}$ is a constant term that can be safely ignored during training. This is because q has no learnable parameters and x_T is assumed to be Gaussian noise, which means that it does not contribute to the gradient of the loss function. Therefore, we can neglect $\mathbf{L_T}$ without affecting the optimization process;
- $\sum_{t=2}^{T} \mathbf{L_{t-1}}$, also referred to as $\mathbf{L_t}$, is particularly interesting as it captures the difference between the desired denoising steps $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and their approximations $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$.

From equation 2.8 and equation 2.9, as shown in [6], the $\mathbf{L_t}$ of the loss function can be expressed as:

$$\begin{aligned} L_t &= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{1}{2\|\boldsymbol{\Sigma}_\theta(\mathbf{x}_t,t)\|_2^2}\|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t,\mathbf{x}_0)-\boldsymbol{\mu}_\theta(\mathbf{x}_t,t)\|^2\right] \\ &= \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t,t)\|^2\right] \end{aligned} \tag{2.11}$$

this clearly indicates that the model is not predicting the mean of the distribution, but rather the noise ϵ for each time step t.

Additionally, research presented in [3] indicates that the diffusion model achieves superior results through the use of a simplified objective that omits the weighting term. As a result, the L_t term can be expressed as:

$$\begin{aligned} L_t^{\text{simple}} &= \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\|^2\right] \\ &= \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t,t)\|^2\right] \end{aligned} \tag{2.12}$$

Moreover, in DDPMs, as described in equation 2.7 and [3], the parameters β_t

are fixed as constants, rather than being learned. In particular, the diagonal covariance matrix $\mathbf{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2\mathbf{I}$, where σ_t is set to either β_t or a modified value $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$. This approach was found to be more stable and produce higher-quality samples than attempting to learn the diagonal variance directly, which can lead to unstable training.

In the paper titled *Improved Denoising Diffusion Probabilistic Models* [7], the authors propose a method to learn $\mathbf{\Sigma}_\theta(\mathbf{x}_t, t)$ by interpolating between β_t and $\tilde{\beta}_t$. This is achieved through the use of a mixing vector $\mathbf{v}$:

$$\mathbf{\Sigma}_\theta(\mathbf{x}_t, t) = \exp(\mathbf{v} \log \beta_t + (1 - \mathbf{v}) \log \tilde{\beta}_t) \tag{2.13}$$

Since the loss L_{simple} does not depend on $\mathbf{\Sigma}_\theta$. To add the dependency, they constructed a hybrid objective:

$$L_{\text{hybrid}} = L_{\text{simple}} + \lambda L_{\text{VLB}} \tag{2.14}$$

Where to prevent the dominance of L_{VLB} over L_{simple}, λ is set to 0.001. Additionally, to ensure that L_{VLB} only guides the learning of $\mathbf{\Sigma}_\theta$, the gradient is stopped on $\boldsymbol{\mu}_\theta(x_t, t)$ for the L_{VLB} term.

2.1.5 Variance schedule

The variance parameter β_t can either be fixed to a constant or chosen as a schedule over the T time steps. For instance, in the paper *Denoising diffusion probabilistic models* [3], the forward variances are set to a sequence of linearly increasing constants, starting from $\beta_1 = 10^{-4}$ and ending at $\beta_T = 0.02$. However, in the paper *Improved denoising diffusion probabilistic models* [7], it is demonstrated that using a cosine schedule leads to better results. Specifically, they proposed the variance scheduling shown below:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)} \quad \text{where } f(t) = \cos\left(\frac{t/T + s}{1 + s} \cdot \frac{\pi}{2}\right)^2 \tag{2.15}$$

Moreover, to avoid singularities at the end of the diffusion process, they constrained β_t to be no greater than 0.999:

$$\beta_t = \begin{cases} 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, & \text{if } 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}} < 0.999 \\ 0.999, & \text{otherwise} \end{cases} \tag{2.16}$$

Figure 2.4: Comparison of linear (top) and cosine (bottom) variance schedulers. Source: *Improved denoising diffusion probabilistic models*[7]

The figure 2.4 shows latent samples generated from linear and cosine schedules, respectively, at evenly spaced values of t from 0 to T. In the linear schedule, the samples in the last quarter of the interval are mostly noise. In contrast, the cosine schedule adds noise at a slower rate, leading to smoother and more coherent samples throughout the entire interval.

2.1.6 Model architecture

The architecture of the diffusion model for image generation is primarily based on the U-net [8], a type of convolutional neural network (CNN) designed for image segmentation tasks. The U-net was chosen for its ability to process and generate high-resolution images while maintaining a relatively low number of trainable parameters, which is crucial for efficient training and reducing overfitting.

The U-net architecture consists of an encoder-decoder structure with skip connections between corresponding layers in the encoder and decoder. This design allows the network to learn both global and local structures within the image, making it suitable for the diffusion model's requirements.

In the original implementation of DDPMs [3], the U-Net consists of Wide ResNet blocks, group normalization, and self-attention blocks.

In addition to the U-Net architecture, the diffusion model for image generation also employs *sinusoidal position embeddings* to encode the noise level of each image in a batch. Then, this embedding is added into each residual block of the network. This technique is inspired by the Transformer architecture and enables the neural network to share its parameters across all the T denoising steps and to keep track of the time step it is currently processing for each image.

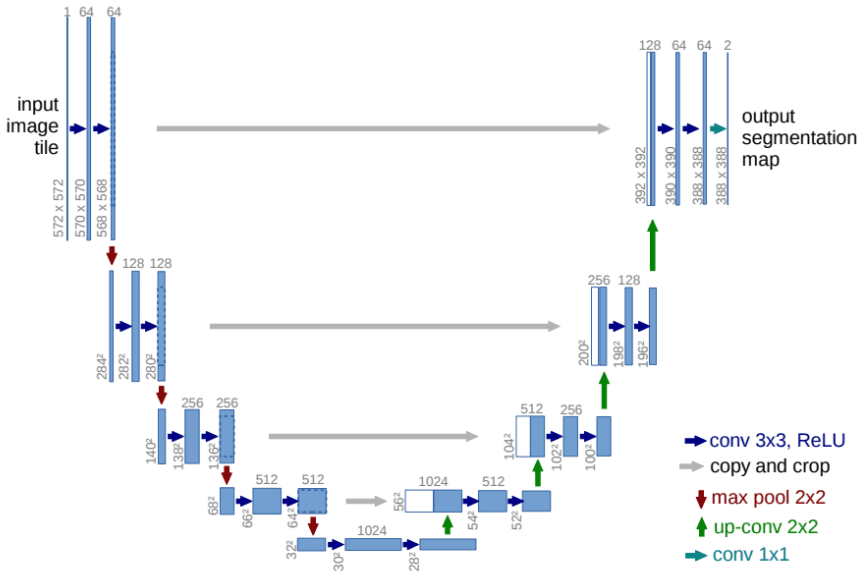


Figure 2.5: The U-Net architecture. Source: *U-Net: Convolutional Networks for Biomedical Image Segmentation*[8]

2.1.7 Denoising Diffusion Implicit Model

DDPM models have a critical drawback: they require many iterations to produce high-quality samples. This is because the generative process approximates the reverse of the forward diffusion process, which could have thousands of steps. To produce a single sample, DDPMs must iterate over all these steps, making them much slower than GANs.

In [9], Denoising Diffusion Implicit Models (DDIMs) are introduced as an alternative diffusion model to improve the efficiency of generating high-quality samples, reducing the gap between DDPMs and GANs.

One of the main differences between the DDPMs and the DDIMs is that the latter proposes a non-Markovian method that enables skipping steps in the denoising process. This means that it's no longer necessary to visit all past states before the current state. Specifically, the DDIM paper introduces a more general forward process that depends on both the initial state x_0 and the current state x_t, given the previous state x_{t-1}

To improve the denoising process, the previous state x_{t-1} is redefined to be parameterized by a desired standard deviation σ_t. The new equation is as follows:

$$\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}} + \sigma_t\boldsymbol{\epsilon} \tag{2.17}$$

This modification allows us to also parameterize the reverse denoising process $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ by a desired σ_t:

$$q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}, \sigma_t^2\mathbf{I}) \tag{2.18}$$

recalling from equation 2.6 and equation 2.7 that:

$$\begin{gathered} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I}) \\ \text{where} \quad \tilde{\beta}_t = \sigma_t^2 = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \cdot \beta_t \end{gathered} \tag{2.19}$$

to improve the denoising process in equation 2.18, we can choose σ_t^2 to be $\eta \cdot \tilde{\beta}_t$. Setting η to 0 makes the sampling process deterministic, while still maintaining the same marginal noise distribution. In this case, DDIM maps noise back to the original data samples deterministically. Another special case is when $\eta = 1$, which is the definition of the original DDPMs. Any η between 0 and 1 is an interpolation between a DDIM and DDPM.

During the generation process, we only sample a subset of S diffusion steps, denoted by $\tau_1, \ldots, \tau_S$. As a result, the inference process is given by the following equation:

$$q_{\sigma,\tau}(\mathbf{x}_{\tau_{i-1}}|\mathbf{x}_{\tau_t}, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{\tau_{i-1}}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2}\frac{\mathbf{x}_{\tau_i} - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}, \sigma_t^2\mathbf{I}) \tag{2.20}$$

In the experiments, all models are trained with $T = 1000$ diffusion steps. However, it was observed that DDIM produces the best quality samples when S is small, while DDPM performs much worse on small S. DDPM performs better when we can afford to run the full reverse Markov diffusion steps (i.e., $S = T = 1000$).

With DDIM, it is possible to train the diffusion model up to any arbitrary number of forward steps, but only sample from a subset of steps during generation.

Moreover, DDIM exhibits a "consistency" property due to its deterministic generative process. This property implies that when multiple samples are conditioned on the same latent variable, they should display similar high-level features. Thanks to this consistency, DDIM can perform semantically meaningful interpolation in the latent variable.

		CIFAR10 (32 × 32)					CelebA (64 × 64)				
	S	10	20	50	100	1000	10	20	50	100	1000
η	0.0	**13.36**	**6.84**	**4.67**	**4.16**	4.04	**17.33**	**13.73**	**9.17**	**6.53**	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
	1.0	41.07	18.36	8.01	5.78	4.73	33.12	26.03	18.48	13.93	5.98
	$\hat{\sigma}$	367.43	133.37	32.72	9.99	**3.17**	299.71	183.83	71.71	45.20	**3.26**

Table 2.1: FID scores on CIFAR10 and CelebA datasets by diffusion models of different settings. Source: *Denoising diffusion implicit models* [9]

2.2 Conditioned image generation

Image generation often requires a way to control how samples are created to manipulate the final output. This process is commonly referred to as *guided diffusion*. Several techniques have been developed to incorporate image and/or text embeddings into the diffusion process to guide the generation. In mathematical terms, "guidance" refers to the process of conditioning a prior data distribution, denoted by $p(\mathbf{x})$, with a specific condition, such as a class label or an image/text embedding. This conditioning results in a conditional distribution, denoted by $p(\mathbf{x}|y)$.

To convert a diffusion model p_θ into a conditional diffusion model, we can introduce conditioning information y at each diffusion step as follows:

$$p_\theta(\mathbf{x}_{0:T}|y) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, y) \tag{2.21}$$

In general, guided diffusion models aim to learn the gradient of the logarithm of the conditional density $p_\theta(\mathbf{x}_t|y)$. By applying Bayes' rule, we can express this gradient as:

$$\nabla_{x_t} log p_\theta(\mathbf{x}_t|y) = \nabla_{x_t} log \left(\frac{p_\theta(y|\mathbf{x}_t) \cdot p_\theta(\mathbf{x}_t)}{p_\theta(y)} \right) \tag{2.22}$$

Since the gradient operator only applies to $\mathbf{x}_t$, we can eliminate the term $p_\theta(y)$ and simplify the expression using the logarithmic product rule:

$$\nabla_{x_t} log p_\theta(\mathbf{x}_t|y) = \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) + s \cdot \nabla_{\mathbf{x}_t} \log p_\theta(y|\mathbf{x}_t) \tag{2.23}$$

Where s is a scalar term used to modulate the strength of the guidance term.

2.2.1 Classifier Guidance

The use of a second model, a classifier $f_\phi(y|\mathbf{x}_t, t))$, to guide the diffusion during training has been demonstrated in [10]. This technique involves training

a classifier $f_\phi(y|\mathbf{x}_t, t)$ on a noisy image $\mathbf{x}_t$ to predict its class y. The gradient $\nabla_{\mathbf{x}} \log f_\phi(y|\mathbf{x}_t)$ can then be utilized to guide the diffusion sampling process towards the conditioning information y by modifying the noise prediction. In particular, in [10] is showed that:

$$\begin{aligned}\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t, y) &= \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p_\theta(y|\mathbf{x}_t) \\ &= -\frac{1}{\sqrt{1-\bar{\alpha}_t}}(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t}\nabla_{\mathbf{x}_t} \log f_\phi(y|\mathbf{x}_t))\end{aligned} \tag{2.24}$$

Therefore, a new classifier-guided predictor $\hat{\boldsymbol{\epsilon}}_\theta$ can be expressed as:

$$\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t) = \boldsymbol{\epsilon}_\theta(x_t, t) - \sqrt{1-\bar{\alpha}_t}\ \cdot s \cdot \nabla_{\mathbf{x}_t} \log f_\phi(y|\mathbf{x}_t) \tag{2.25}$$

As before, s is a scalar term that modulates the strength of the guidance term.

2.2.2 Classifier-free Guidance

As demonstrated in [11], performing conditional diffusion steps can be achieved without relying on an independent classifier $f_\phi(y|\mathbf{x}_t, t)$. Specifically, by starting from the formulation in Equation 2.23, a *classifier-free guided diffusion model* can be defined as follows:

$$\nabla_{x_t} log p_\theta(\mathbf{x}_t|y) = (1-s) \cdot \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) + s \cdot \nabla_{\mathbf{x}_t} \log p_\theta(y|\mathbf{x}_t) \tag{2.26}$$

Instead of training a separate classifier, the authors trained a conditional diffusion model $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t|t, y)$ along with an unconditional model $\boldsymbol{\epsilon}\theta(\mathbf{x}_t|t, 0)$. Notably, the same neural network is used for both models. During training, the class y is randomly set to 0, exposing the model to both conditional and unconditional setups. Finally, as shown in [11], we obtain:

$$\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t|t, y) = (s+1) \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - s \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \tag{2.27}$$

This process offers two significant advantages. Firstly, it uses only a single model to guide the diffusion process. Secondly, it simplifies the conditioning process when working with data that is difficult to predict using a classifier, such as text embeddings.

2.3 CelebFaces Attributes Dataset

The CelebFaces Attributes dataset (CelebA) [12] is a large-scale dataset containing over 200,000 annotated celebrity images, widely used in computer vision and machine learning for facial recognition and attribute prediction. Each image has 40 binary attribute annotations, such as gender, age, and facial hair, as well as bounding box annotations for the faces.

CelebA's diverse collection of images vary in quality, resolution, and come from different sources, with a wide representation of ethnicities, ages, and genders. An aligned version of the dataset ensures consistent size and orientation by centering faces in a common coordinate system, making it more suitable for deep learning models.

Additionally, CelebAMask-HQ [13] is a variant of CelebA consisting of 30,000 high-resolution images with manually-annotated segmentation masks for 19 facial components and accessories. This dataset is useful for training and evaluating face parsing, recognition, and generative adversarial networks (GANs) in face generation and editing.

In conclusion, the CelebFaces Attributes dataset (CelebA) and its variant CelebAMask-HQ are valuable resources for researchers and practitioners in the fields of computer vision and machine learning.

Figure 2.6: Random samples from CelebA Aligned dataset.

2.4 Exploring Embeddings in Denoising Diffusion Models

Generative models, like the Denoising Diffusion Probabilistic Model (DDPM) and the Generative Adversarial Network (GAN), aim to capture a compressed version, or "embedding", of high-dimensional data like images. This embedding is used to either model the input's probability distribution or generate new samples that follow the learned distribution.

However, the challenge with embeddings is that there are many possible ways to represent the data, and it's unclear which one is optimal. Furthermore, embeddings must be learned through unsupervised learning, so there's no labeled data available to guide the process.

The focus of the study by [14] is to investigate the problem of embedding an image into the latent space of Denoising Diffusion Models. Specifically, the study aims to identify a suitable "noisy" image that, when denoised, results in the original image. This is a challenging task because the generator of a diffusion model is non-injective, meaning that for each sample, there exists a cloud of elements that can generate that sample. This set of elements is called the embedding of the sample, denoted as $emb(x)$.

The authors experimented with several approaches for the embedding task and found that the most effective ones were direct synthesis of the embedding representation of an image through a neural network trained to compute a "canonical" embedding for each image. The network takes an image x as input and produces a seed $z_x \in emb(x)$. The loss function used to train the network is the distance between x and the result of the denoising process starting from z_x. The best results were obtained with a U-Net architecture, which is practically identical to the denoising network.

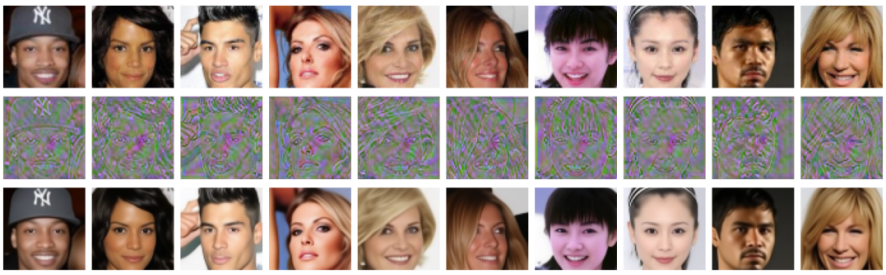


Figure 2.7: Examples of embedding for the CelebA dataset. The first row shows the original images, the second row displays the synthesized latent seed, and the third row presents the reconstructed images.
Source: *Image Embedding for Denoising Generative Models* [14]

It's important to note that embedding is not an iterative process and a single forward pass through the embedding network is enough to compute the latent representation. The reconstruction quality is high, with just slight blurriness and an MSE of around 0.0012 for the CelebA dataset.

Chapter 3

Related Works

Face rotation is a crucial task in computer vision with widespread applications in areas such as security, entertainment, and healthcare. In recent years, the field of facial rotation has been transformed by deep learning techniques, which have facilitated the development of more accurate and efficient methods for rotating faces. This section provides an overview of facial rotation techniques developed over time and categorizes them into two groups: those created before the emergence of deep learning techniques and those developed afterward.

3.1 Face Rotation before Deep Learning

Prior to the advent of deep learning, facial rotation techniques typically involved applying the characteristics of the input face image to a 3D face model and rotating it to generate a rotated version of the input image. Some examples of this approach include [15] and [16]. However, [17] proposed a different method that involved constructing a 3D transformation matrix to map each point in a 2D face image to a corresponding point on a 3D face model. Despite their ability to produce rotated face images, these techniques were limited by the distortion and blurring effects that arose from the conversion of 2D images into 3D models.

3.2 Face Rotation after Deep Learning

The advancement of deep learning has greatly accelerated facial rotation techniques, particularly those based on generative adversarial networks (GANs). GAN-based facial rotation techniques have been extensively researched and developed due to their effectiveness in producing high-quality results.

3.2.1 Latent Space-based Techniques

Recent research has developed methods to manipulate and control the attributes of generated faces through a latent space-based approach. This includes controlling attributes such as age, eyeglasses, gender, expression, and rotation angles. Several methods have been developed, including PCA analysis to extract important latent directions [18], semantic analysis to control various attributes [19], and composing a new latent vector to control multiple attributes [20]. However, these methods have not prioritized face rotation or pose control, potentially limiting their ability to produce faces with various poses and angles.

3.2.2 Reconstruction-Based Techniques

Reconstruction-based techniques use Generative Adversarial Networks (GANs) to generate a composite face image from a specific angle, with face frontalization being the most common. Face frontalization aims to enhance the accuracy of face recognition by synthesizing a frontal face image from a side view of facial images.

DR-GAN [21], TP-GAN [22], CAPG-GAN [23], and FNM [24] are popular techniques in this category. DR-GAN separates the input image's features and angle to create a frontal image, while TP-GAN learns the overall outline features and detailed features separately to synthesize the frontal face image. CAPG-GAN uses a heat map to frontalize an input face, and FNM combines labeled and unlabeled data to improve learning efficiency.

However, these methods struggle to generate convincing results for input images with angles close to the side and for angles other than the front.

3.2.3 3D Geometry-Based Techniques

Various 3D geometry-based approaches have been developed to tackle face rotation challenges by combining conventional techniques with Generative Adversarial Networks (GANs). Notable methods include FF-GAN [25], UV-GAN [26], HF-PIM [27], and Rotate-and-Render [28].

FF-GAN creates rotated face images by combining 3D deformable model coefficients with an existing 3D model, while UV-GAN rotates images using a UV map. HF-PIM synthesizes characteristic information of an input image and applies it to the result, and Rotate-and-Render uses texture feature information of an input image to construct a dataset for synthesizing rotated images from multiple angles.

Compared to reconstruction-based methods, 3D geometry-based approaches produce more realistic results for side-facing images but require additional processing for the 3D model, which consumes more computational resources.

3.2.4 Face Rotation with Neural Radiance Fields

Neural Radiance Fields (NeRF) [29] is an advanced approach for representing complex 3D scenes using neural networks. It has shown great potential in various computer vision tasks due to its ability to generate photorealistic renderings of scenes from novel viewpoints. NeRF models the radiance and volume density of a scene as a continuous function, which is parameterized by a neural network that takes a 3D coordinate and a viewing direction as inputs. The scene's appearance is then rendered by integrating the radiance along each camera ray.

Despite its impressive results, NeRF has some limitations. It is computationally expensive and requires large memory resources, making it challenging

to apply to large-scale environments. Additionally, NeRF is sensitive to input data and has limited ability to capture fine geometric details.

One area where NeRF has received increasing attention is in face rotation. For example, a recent paper called *FENeRF: Face Editing in Neural Radiance Fields* [30] proposes a two-stage approach for editing facial attributes in images using NeRFs. The first stage involves predicting a 3D representation of the face using a NeRF, and the second stage involves manipulating the 3D representation to edit the face attributes. The method has demonstrated effectiveness in changing the pose and expression of faces, among other editing tasks.

Chapter 4

Proposed Method

Facial rotation techniques that use either the reconstruction-based approach or the 3D geometry-based approach have their respective advantages and limitations. However, a common drawback of both approaches is that they rely on ground truth data for training, specifically pairs of images captured from different angles of a specific person, along with a frontal image of that person.

To overcome this limitation, this work proposes two novel techniques for solving the facial rotation problem without relying on pairs of images. Both techniques aim to explore the latent space of a DDIM trained to reconstruct human faces and condition the generation process to produce rotated faces.

4.1 DDIM and Embedding Models

The proposed techniques are built upon two key models: the Diffusion Denoising Implicit Model (DDIM) and the Embedding Model, which were introduced in [14] and analyzed in the background chapter of this work. The DDIM is trained to reconstruct realistic human faces using the CelebA dataset, while the Embedding Model is trained to produce embeddings in the latent space that enable the generation of faces. The use of a diffusion model is particularly interesting as it allows for a more visible analysis of the latent space. This is because the dimensionality of the latent space is the same as that of the

visible space.

4.2 Dataset Preparation

The dataset utilized for this experiment is a modified version of the CelebFaces Attributes dataset (CelebA-aligned). This extensive dataset includes over 200,000 images of celebrity faces, which have been annotated and centered. For this experiment, a central crop of dimension 128×128 was employed, which is a common crop size used in previous research [31, 32]. The 128×128 crop size was chosen to facilitate down-sizing to a final input-output dimension of 64×64, which is the required size for the reverse diffusion process.

4.2.1 Analysis of Annotations

Out of the 40 binary attribute annotations available in the dataset, this study has focused on a subset that includes gender, smiling, youthfulness, and mouth opening, which are considered important factors in conditioning the generation of the DDIM using the proposed techniques.

In this section, an analysis will be conducted on the distribution of these attributes across the images in CelebA to determine if there are any biases that need to be taken into account.

In CelebA, each attribute is annotated with either -1 or 1. For example, for gender, ”male = 1” indicates images of males, and ”male = -1” indicates images of females.

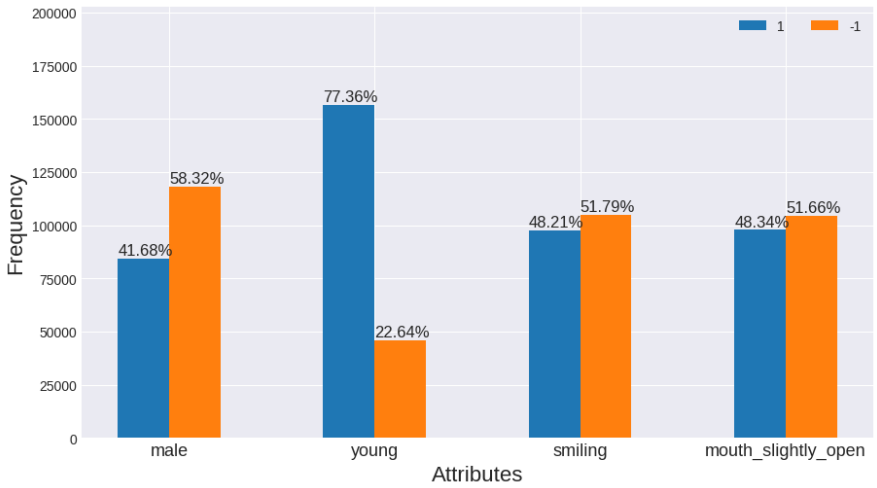


Figure 4.1: Distribution of CelebA attributes

As shown in Figure 4.1, the dataset is unbalanced. Specifically, approximately 58% of the images in the dataset are of females, and about 77% of the images are of young people. This imbalance may negatively impact the performance of the proposed method, particularly for faces of males and older individuals compared to females and younger individuals. Instead, there is no significant imbalance in the distribution of the ”smiling” and ”mouth_slightly_open” attributes.

4.2.2 Light Direction Analysis

In order to enhance the results, the CelebA-aligned dataset has been expanded to include information about the direction of light that falls on each face. This additional data enables us to differentiate between three types of light direction: ”Center”, ”Right”, and ”Left”.

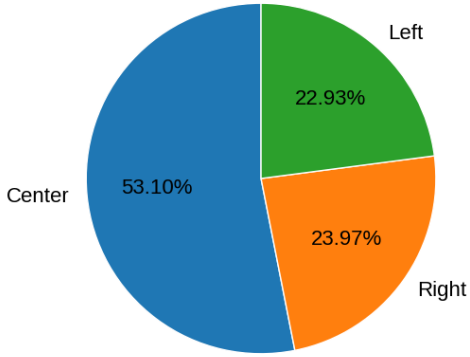


Figure 4.2: Distribution of light direction on CelebA

As illustrated in Figure 4.2, approximately half of the images in the dataset exhibit a center light direction, while the remaining images are almost equally distributed between left and right directions.

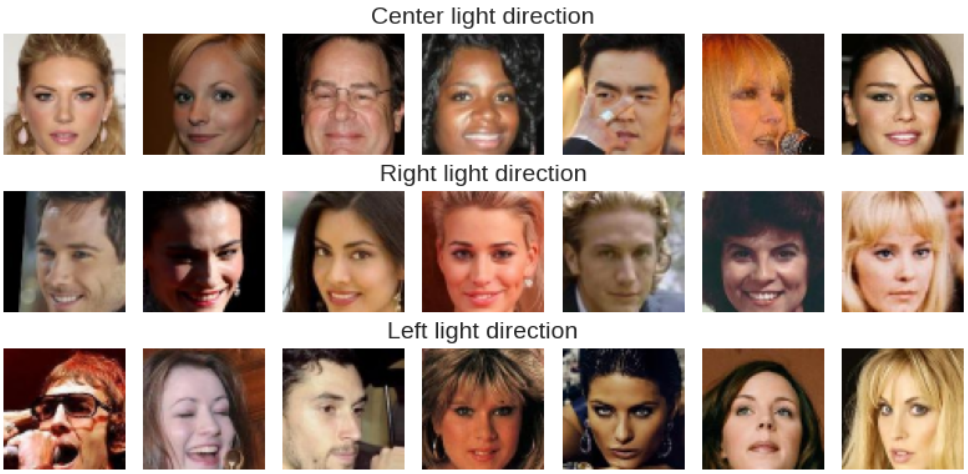


Figure 4.3: Random sample from CelebA for each light direction

4.2.3 Analyzing Face Orientation

To condition the generation of DDIM for producing faces with varying orientations, information about head orientation in the CelebA dataset is required. Unfortunately, this information is not provided in the standard CelebA-aligned dataset's annotations. As a result, the head orientation for each image in the dataset must be estimated through head pose estimation.

Head pose estimation is a computer vision task that determines a person's head orientation in three-dimensional space by calculating three rotation angles: yaw, pitch, and roll. These angles comprehensively represent the head's orientation. Specifically, yaw denotes the rotation around the vertical axis, pitch refers to the rotation around the horizontal axis, and roll signifies the rotation around an axis perpendicular to the other two.

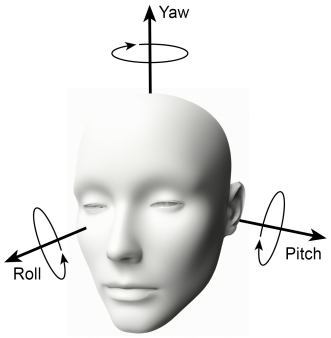
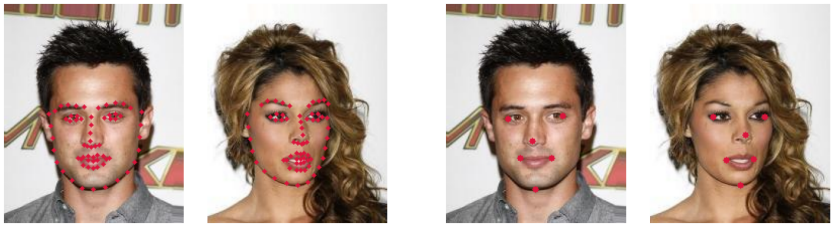


Figure 4.4: Yaw, Pitch and Roll angles in Head Pose Estimation

To calculate these angles, the facial landmarks for each image in CelebA need to be computed. For this purpose, the Face Recognition library [33] was employed, which is a robust Python library built on top of Dlib and deep learning models. The face detection algorithm used in this library is based on the Histogram of Oriented Gradients (HOG) method. This technique computes the gradient orientation and magnitude at every pixel in an image, then groups these values into cells and blocks. The resulting histogram is used to identify regions of the image that may contain a face. Once faces have been detected,

the library employs a deep learning model to extract facial landmarks, achieving an accuracy of 99.38% on the Labeled Faces in the Wild benchmark.



(a) (b)

Figure 4.5: Figure (a) displays all facial landmarks, while Figure (b) shows a reduced set used for head pose estimation.

Next, a subset of facial landmarks is selected, including the nose tip, chin, eye corners, and mouth corners (as shown in figure 4.5 (b)). By combining these points with a generic 3D face model, the face's rotation and translation vectors can be estimated. The cv2.solvePnP() function from the OpenCV library [34] is used to accomplish this, which solves the Perspective-n-Point (PnP) problem. Specifically, the iterative method (cv2.SOLVEPNP_ITERATIVE) is applied to refine the estimates. If the estimation is successful, the Euler angles (yaw, pitch, and roll) are computed from the decomposed projection matrix using the cv2.decomposeProjectionMatrix() function, based on the following equations:

$$\begin{aligned} pitch &= atan2\left(-R_{2,0}, \sqrt{R_{2,1}^2 + R_{2,2}^2}\right) \\ yaw &= atan2(R_{1,0}, R_{0,0}) \\ roll &= atan2(R_{2,1}, R_{2,2}) \end{aligned} \tag{4.1}$$

Here, $R_{i,j}$ represents the entries of the rotation matrix obtained through the cv2.solvePnP() function.

Finally, the pitch, yaw, and roll angles are converted from radians to degrees and corrected to ensure that these angles lie within the appropriate range, using the following formulas:

$$
\begin{aligned}
pitch &= \arcsin\left(\sin(pitch)\right) \\
roll &= -\arcsin\left(\sin(roll)\right) \\
yaw &= \arcsin\left(\sin(yaw)\right)
\end{aligned} \tag{4.2}
$$

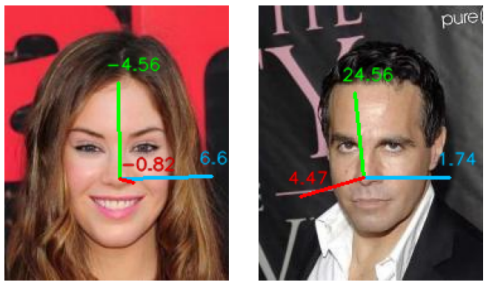


Figure 4.6: Example of results of head pose estimation on CelebA. The estimated yaw is indicated by the green color, pitch by blue, and roll by red.

After completing the head pose estimation, an analysis was performed on the distribution, especially of yaw, as it represents the most significant rotation for this task. Figure 4.7 shows that, as expected for an aligned dataset, over 40% of the images have yaw in the $[-10, +10]$ degree range. Additionally, only 4.48% of the images have yaw outside the $[-40, +40]$ degree range. This is an important factor to keep in mind as it will certainly reduce the performance of the proposed method in cases of more extreme rotations.

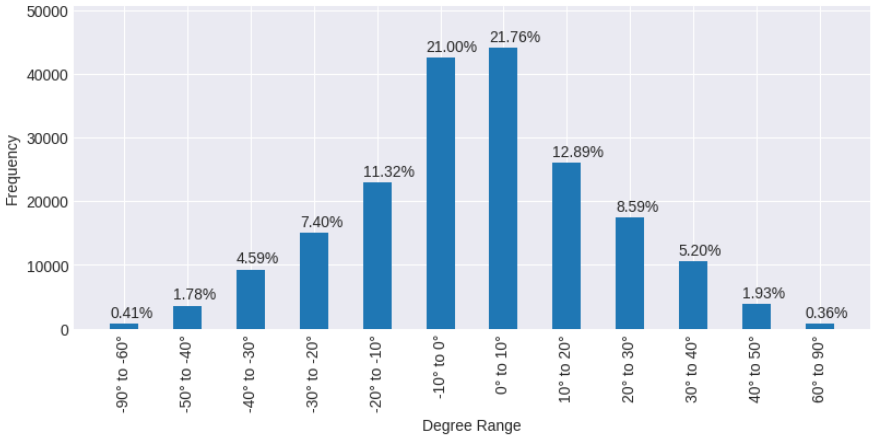
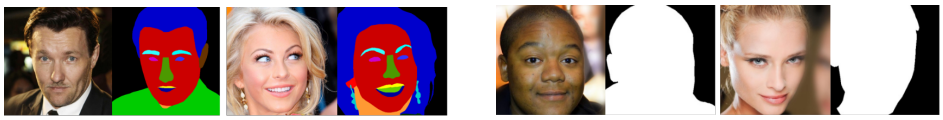


Figure 4.7: Yaw distribution on CelebA dataset

4.3 Preprocessing

To improve the obtained results, CelebA images were subjected to a preprocessing phase. Specifically, as mentioned previously, a 128×128 crop of the images was utilized, followed by resizing to 64×64. This allowed the images to be compatible with the DDIM's input requirements. Subsequently, to reduce unnecessary information and facilitate conditioning, a segmentation process was employed to eliminate the background from the images.

To execute this segmentation, a U-Net model was trained on the CelebaMask-HQ dataset, which contains high-quality face masks that were manually annotated. To ensure consistency throughout the pipeline, the dataset was cropped in the same manner as the original CelebA images used to train the DDIM. All masks were merged and treated as a binary segmentation mask problem, addressing background/foreground segmentation.



(a) (b)

Figure 4.8: In Figure (a), the original CelebaMask-hq masks are displayed, while Figure (b) shows the cropped version with unified masks used to train the segmentation model.

The U-Net model is widely employed for image segmentation tasks. In this implementation, the model accepts an input size of 256×256 and consists of four levels of convolutional and deconvolutional layers, starting with 32 filters. The model adopts a U-shaped architecture, utilizing skip connections between corresponding encoder and decoder blocks to retain spatial information and enhance segmentation performance.

This approach allowed for precise segmentation of the facial region, achieving a precision of 96.78% and a recall of 97.60%.

4.4 Postprocessing

In order to enhance the final results, a post-processing pipeline has been developed, which involves two key steps: super-resolution and color correction. The original output generated at 64×64 resolution is upscaled to 256×256 using one of two available architectures: a custom-built model developed and trained specifically for this project, or the CodeFormer [35]. While the former is faster, it produces lower quality results compared to the latter, which is slower but yields significantly better output.

This section will provide a brief analysis and discussion of both the super-resolution architecture and the color correction technique used.

Proposed Super-Resolution Model

The proposed super-resolution architecture is based on the generator architecture from [36], with the addition of a Self-Attention Mechanism as described in [37]. The architecture includes self-attention layers and residual blocks to capture long-range dependencies and avoid the vanishing gradient problem. It consists of a skip connection, 16 residual blocks with self-attention modules, and two upsampling blocks.

The overall architecture is described in figure 4.9.

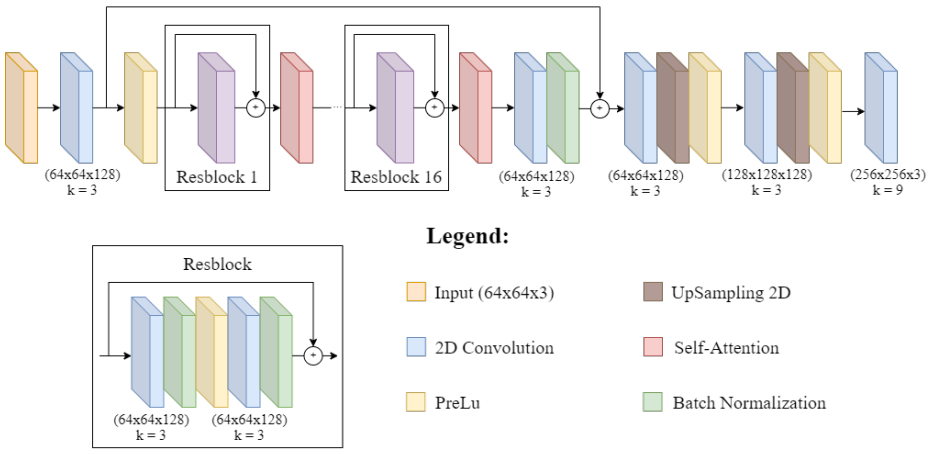


Figure 4.9: Architecture of the proposed super-resolution model

The model was trained on a cropped version of CelebAMask-HQ using a custom loss that combines mean squared error and perceptual loss based on VGG-19 [38].

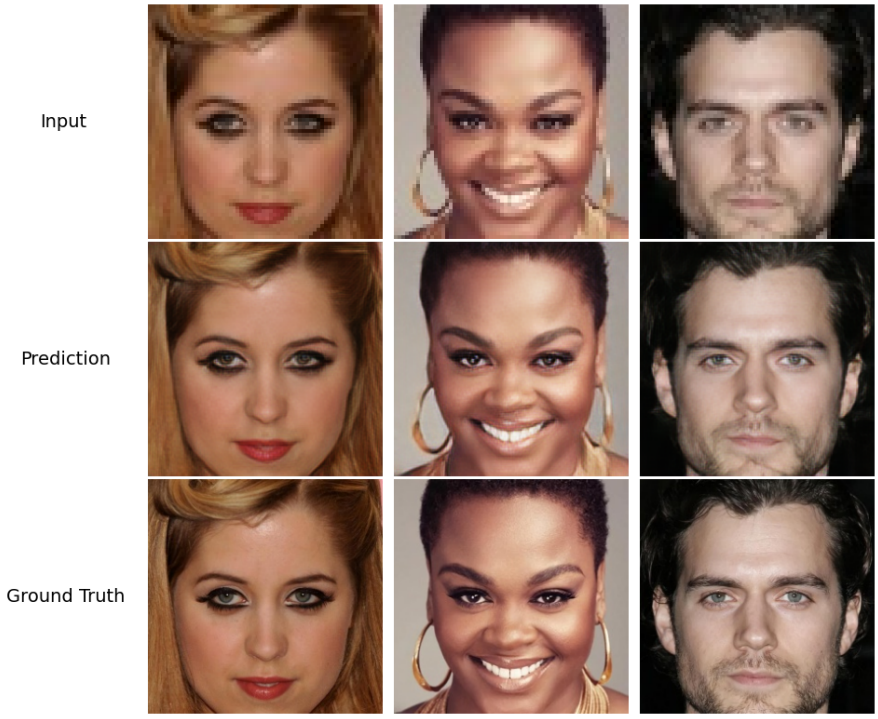


Figure 4.10: Output of the proposed Super-Resolution model. Using Google Colaboratory, the model generates three predictions in less than 1 second.

CodeFormer for Super-Resolution

The CodeFormer [35] is a recently proposed model for Blind Face Restoration and Super-Resolution. It effectively combines the powerful characteristics of transformers and codebooks to achieve high-quality results. Transformers have become a widely popular class of models, extensively employed in natural language processing and computer vision tasks. Codebooks, conversely, serve as a means to quantize and represent data in a more compact and efficient manner.

The primary advantage of employing Codebook Lookup Transformers for

face restoration tasks lies in their capacity to capture and leverage the structure and semantics of face images. By utilizing a pre-defined codebook encompassing facial features or characteristics, the model can effectively restore high-quality face images from low-quality or degraded inputs, even in presence of various types of noise, artifacts, and occlusions. Owing to the expressive codebook prior and global modeling capabilities, the CodeFormer surpasses state-of-the-art methods in terms of both quality and fidelity.

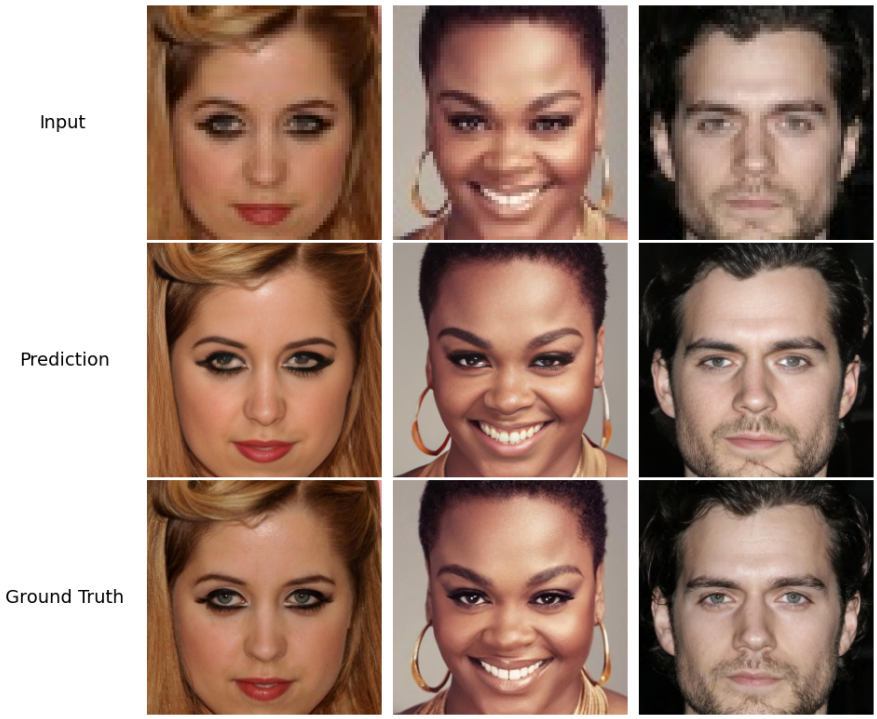


Figure 4.11: Output of the CodeFormer model. Using Google Colaboratory, the model generates three predictions in about 30 seconds.

Color Correction

As a final step in the post-processing phase, a color correction technique is applied to minimize any color discrepancies between the generated faces and their corresponding source images. This technique helps to enhance the overall visual coherence of the final result.

The color correction process utilizes the Lab color space to align the color

statistics of two images. This process involves several steps, beginning with the conversion of the images to Lab color space. The Lab channels of the target image are then normalized using the mean and standard deviation of the source image, followed by the conversion of the target image back to the RGB color space. These steps are outlined in detail in Algorithm 1.

Algorithm 1: Color correction

Input: target_image, source_image
Output: Color corrected target image
lab_target = convert_color_space(target_image, ”RGB”, ”LAB”)
lab_source = convert_color_space(source_image, ”RGB”, ”LAB”)

mean_target = compute_mean(lab_target)
mean_source = compute_mean(lab_source)
std_target = compute_standard_deviation(lab_target)
std_source = compute_standard_deviation(lab_source)

$\text{lab\_target} = (\frac{(\text{lab\_target} - \text{mean\_target})}{\text{std\_target}}) \times \text{std\_source} + \text{mean\_source}$
target_image = convert_color_space(lab_target, ”LAB”, ”RGB”)
return target_image

The LAB color space is a widely used color model in color correction algorithms due to its ability to separate the luminance (brightness) component from the chrominance (color) information. This separation allows for the manipulation of color and brightness information independently, resulting in a more precise and reliable color correction process.

4.5 Filtering CelebA images

To properly condition the generation of the diffusion model, it’s crucial to understand how filtering on CelebA is performed. This step involves defining a function that retrieves from the CelebA dataset all the images that match a specific subset of attributes and fall within a specified range of orientations.

The filtering algorithm plays a critical role, the more images that meet the specified conditions, the better the results will be.

The filtering process involves several augmentation steps, such as including flipped images to change the light direction. For example, if the search is for images with a yaw of -45 degrees and a light direction of 'LEFT', images with a yaw of +45 degrees and a light direction of 'RIGHT' can also be used by flipping them. This greatly increases the subset of possible images. Additionally, the accepted range of orientations also grows with each augmentation step. After the filtering process, the mean of all retrieved images is computed, which in this work will be referred to as "mean image". Typically, using a number of images between 500 and 1000 produces good results. Increasing the number of images can lead to noisier mean images due to the increased number of augmentation steps performed on the orientation range.

The algorithm is presented in pseudocode in Algorithm 2, while Figure 4.12 and Figure 4.13 display different mean images obtained using different subsets of attributes and orientations.

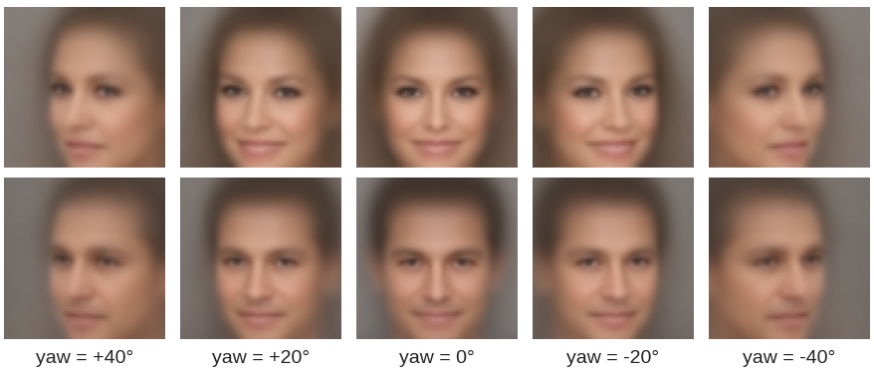
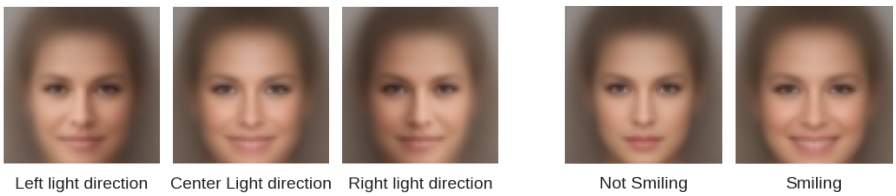


Figure 4.12: The top row displays the mean images for specified yaw angles when the male attribute is -1. The bottom row shows the mean images for specified yaw angles when the male attribute is 1.



(a) (b)

Figure 4.13: Figure (a) displays the mean images for different light directions, while Figure (b) shows the images for different values of the smiling attribute.

Algorithm 2: Filtering on CelebA

```
Input: orientations, attributes, max_augmentation_attempt
Output: Images matching specified attributes and orientations.
yaw_offset, pitch_offset, roll_offset := 2
images, flipped_images := []
attempt_count := 0
flip_orientations, flip_attributes := flipData(orientations, attributes)

while (length(images) + length(flipped_images) < 500)
 and (attempt_count < max_augmentation_attempt) do
    images.append(getImages(orientations, attributes,
                            yaw_offset, pitch_offset,
                            roll_offset))
    flipped_images.append(getImages(flipped_orientations,
                                    flipped_attributes, yaw_offset,
                                    pitch_offset, roll_offset))
    attempt_count := attempt_count + 1
    yaw_offset = yaw_offset + 2
    pitch_offset = pitch_offset + 2
    pitch_offset = pitch_offset * 2
end
merged_list := images + flipped_images
return merged_list[:min(length(merged_list, 1000)]
```

4.6 Method 1: Pixel-based conditioning on Input Image

To explain the first proposed conditioning method, it's necessary to understand how an image is generated using the Embedding Model and the DDIM.

Suppose an image, X_1, is given as input to the Embedding Model, which approximates its representation in the latent space, denoted as l_1. Finally, the reverse diffusion process is applied through the DDIM on the embedding l_1 to generate the output image y_1. Figure 4.14 illustrates this process.

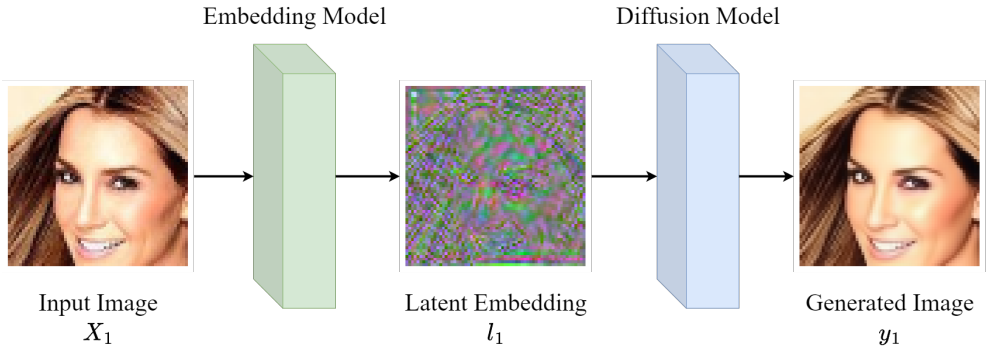


Figure 4.14: Pipeline of generation process using the Embedding Model and the DDIM.

The first proposed method involves conditioning the generation of the diffusion model by modifying specific pixels in the input image X_1 passed as input to the Embedding Model. This ensures that the resulting embedding l_1 reflects the conditioning, allowing the DDIM to produce a conditioned image.

This process consists of several steps. In this section, the process is described in detail using the first image of CelebA aligned as an example. We refer to this image as X_1. As seen in Figure 4.15, the face is turned to the left, so in this case, $starting\_yaw \approx 30°$. Suppose the goal is to condition DDIM to generate the same face but more centered, so the target yaw is set to $target\_yaw = starting\_yaw - 15° \approx 15°$. This operation is called a *single*

conditioning step.

The process is as follows:

- Use the function described in section 4.5 to obtain the two mean images associated with the starting yaw and the target yaw, respectively called $original\_mean$ and $target\_mean$. Then, subtract the $original\_mean$ from the $target\_mean$ to obtain a first correction called $correction_{raw}$. This process is shown in Figure 4.15 and described in equation 4.3:

$$correction_{raw} = target\_mean - original\_mean \tag{4.3}$$

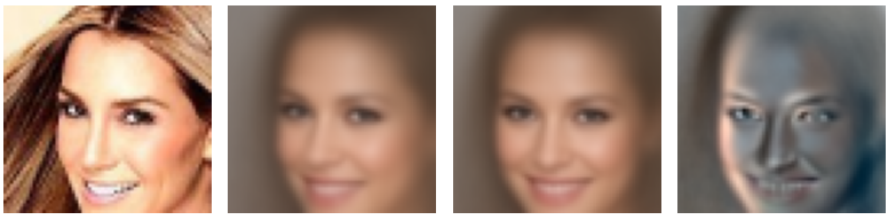


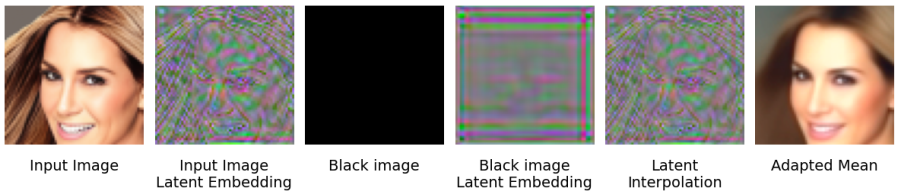
Figure 4.15: Computation of the raw correction on the first image of CelebA.

- After obtaining the raw correction, the next step involves computing the $mean\_original_{adapted}$ and $mean\_target_{adapted}$ images. These mean images are computed using linear interpolation in the latent space of the DDIM to obtain images that more closely resemble the input image.To compute the $mean\_target_{adapted}$ image, the raw correction is used to perform the interpolation. For the $mean\_original_{adapted}$ image, a fully black image is used instead. Equation 4.4 shows the computation of $mean\_original_{adapted}$, while equation 4.5 describes the computation of $mean\_target_{adapted}$. The results are shown in Figure 4.16.

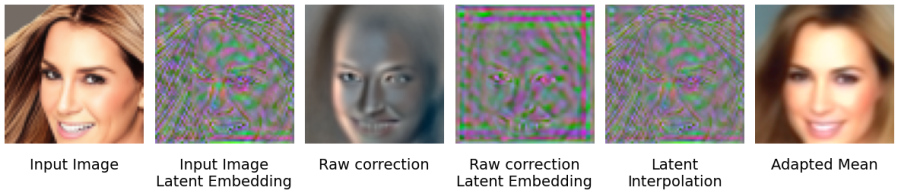
$$
\begin{aligned}
l_1 &= \texttt{EmbeddingModel.predict}(X_1) \\
l_{black\_image} &= \texttt{EmbeddingModel.predict}(black\_image) \\
l_{\text{interpolated}} &= (w) \cdot l_1 + (1-w) \cdot l_{black\_image} \\
mean\_original_{adapted} &= \texttt{DDIM.reverseDiffusion}(l_{\text{interpolated}})
\end{aligned}
\tag{4.4}
$$

$$
\begin{aligned}
l_1 &= \texttt{EmbeddingModel.predict}(X_1) \\
l_{raw\_correction} &= \texttt{EmbeddingModel.predict}(raw\_correction) \\
l_{\text{interpolated}} &= (w) \cdot l_1 + (1-w) \cdot l_{raw\_correction} \\
mean\_target_{adapted} &= \texttt{DDIM.reverseDiffusion}(l_{\text{interpolated}})
\end{aligned}
\tag{4.5}
$$

Where w represents the weight assigned to the latent embedding of the input image. Increasing w leads to greater similarity between the adapted mean and the original image, but it also results in less rotation.



(a)



(b)

Figure 4.16: Figure (a) shows the computation of the adapted mean for the starting orientations, while Figure (b) shows the computation of the adapted mean for the target orientations.

- After calculating the $mean\_target_{adapted}$ and $mean\_original_{adapted}$, the next step is to compute the $correction_{final}$. This correction is simply the difference between the adapted means and is calculated using the equation shown in 4.6. A visual representation of this process can be seen in figure 4.17.

$$correction_{final} = mean\_target_{adapted} - mean\_original_{adapted} \tag{4.6}$$

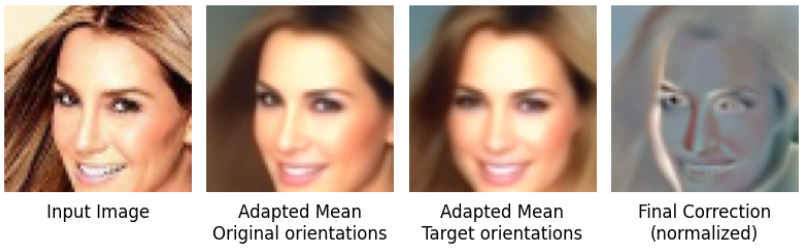


Figure 4.17: Computation of the final correction.

- Finally, the next step is to add the $correction_{final}$ to the input image X_1 using a weighted sum. This modified image X_{cond} is then passed as input to the Embedding Model, which produces the conditioned latent embedding l_{cond}. To generate the final output, the DDIM model is used to perform reverse diffusion on l_{cond}, resulting in the image y_{cond} with a changed head orientation. This entire process is illustrated in Figure 4.18 and described by the equation shown in 4.7

$$\begin{aligned} X_{\text{cond}} &= X_1 + w \cdot correction_{final} \\ l_{\text{cond}} &= \texttt{EmbeddingModel.predict}(X_{\text{cond}}) \\ y_{\text{cond}} &= \texttt{DDIM.reverseDiffusion}(l_{\text{cond}}) \end{aligned} \tag{4.7}$$

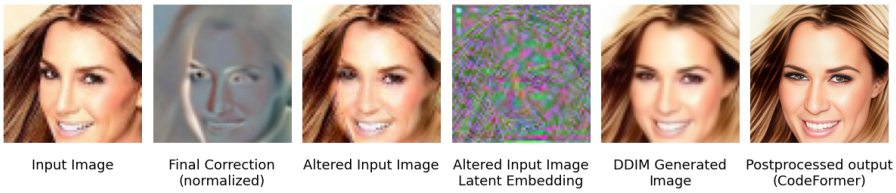


Figure 4.18: Final generation step

Once the output image y_{cond} is obtained, the process can be repeated by using y_{cond} as the new input X_1 and generating a new output image y'_{cond}. This can be done iteratively to obtain multiple output images with different head orientations. The pseudocode for the entire process is provided in Algorithm 3.

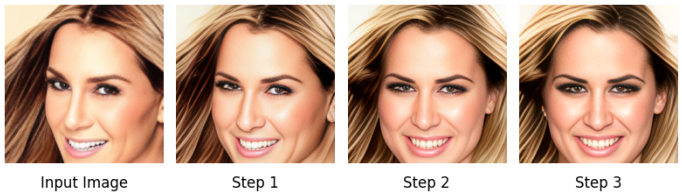


Image 1

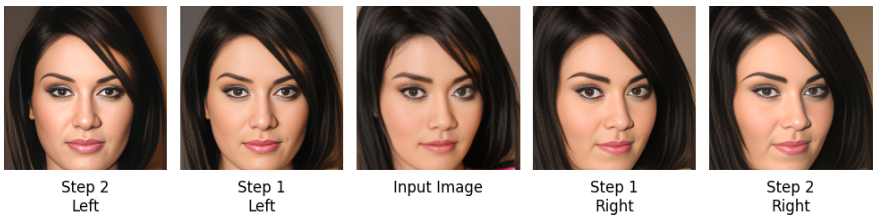


Image 7865

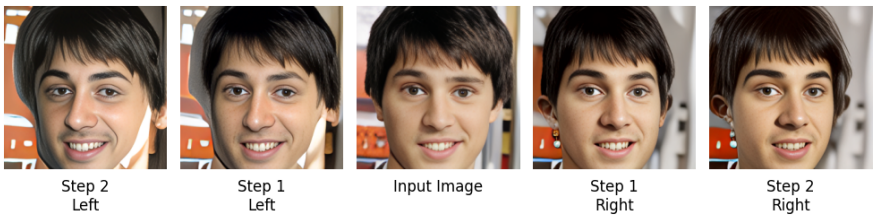


Image 114

Figure 4.19: Method 1 applied with additional steps on 3 samples from CelebA.

Algorithm 3: Method 1 pseudocode

Input: image, steps, correction_degree, correction_weight

Output: Images with different head orientation

output := []

$image_{focus}$ = image

orientations = getOrientation($image_{focus}$)

attributes = getAttributes($image_{focus}$)

for $i \leftarrow 0$ **to** *steps* **do**

original_images = filterCelebA(orientations, attributes)

target_images = filterCelebA(orientations + correction_degree, attributes)

original_mean = average(original_images)

target_mean = average(original_images)

$correction_{raw}$ = target_mean - original_mean

$mean\_original_{adapted}$ = getAdaptedMean($image_{focus}$, $image_{black}$)

$mean\_target_{adapted}$ = getAdaptedMean($image_{focus}$, $correction_{raw}$)

$correction_{final}$ = $mean\_target_{adapted}$ - $mean\_original_{adapted}$

X_{cond} = $image_{focus}$ + correction_weight · $correction_{final}$

l_{cond} = EmbeddingModel.predict(X_{cond})

y_{cond} = DDIM.reverseDiffusion(l_{cond})

output.append(y_{cond})

$image_{focus}$ = y_{cond}

orientations = orientations + correction_degree

end

return output

Figure 4.19 displays some results obtained using the first proposed method. However, this technique has several drawbacks that need to be addressed. Firstly, it requires extensive parameter tuning, including adjusting the input image weight for both the adapted mean and the final generation, as well as the yaw correction degree, for each image individually. Secondly, as the number of steps increases, this method tends to produce artifacts that can have a cascading effect on subsequent steps, compromising the overall quality of the output. Despite these limitations, this technique still enables a reasonably accurate rotation of the face for at least one step, and as mentioned in the chapter introduction, it does not necessitate training a dedicated network using a specially curated dataset with ground truth images.

4.7 Method 2: Linear Regression and latent interpolation

The second proposed method involves using two linear regressors, one for sampling images to the left and the other for sampling images to the right. These regressors are used to perform sampling in the latent space of the diffusion model. The resulting samples are denoted as lr_i for $i \in 0, \ldots, n\_sample_{right}$ for the samples rotated to the right, and ll_i for $i \in 0, \ldots, n\_sample_{left}$ for the samples rotated to the left. To illustrate this method more clearly, let's consider a concrete example using image X_1, which is the 25000th image of CelebA.

Figure 4.20: Image 25000 from CelebA dataset: the original image (left), the pre-processed image (center), and the resized version (right) used as input.

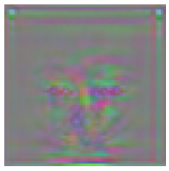
As shown in Figure 4.20, the face in this image is slightly rotated to the right, with a starting yaw of approximately -5 degrees. This example aims to sample in the latent space of the DDIM to rotate the face to the left. Note that the procedure for rotating to the right is identical and will not be discussed further.

The process is divided into several steps:

- The first step is to create the data on which the Linear Regressor will be fitted, these data are called $root\_points_i$. To obtain these points, the function described in section 4.5 is used to select a subset of CelebA images that match the attributes and orientations of the input image.

(a) 5 Random sample of similar images

(b) Embedding of the similar images

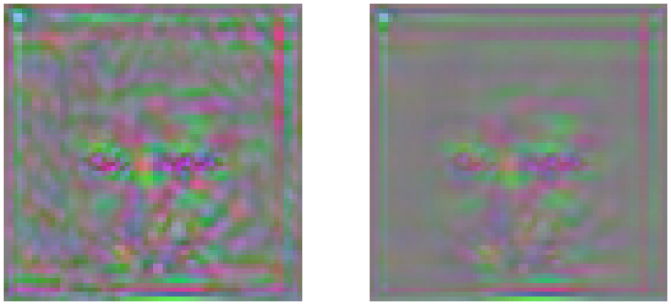


(c) Root point 0

Figure 4.21: This image demonstrates the process of computing the root point 0, based on a subset of attributes including 'male', 'gender', and 'smiling'. These attributes were used to get similar images from CelebA.

Next, the embedding for each image in this subset is calculated, and the average of all the embeddings is taken to obtain the first $root\_point_0$. To obtain the remaining $root\_point_i$, the same process is repeated while gradually increasing the yaw angle of the selected images. For example, to obtain the second $root\_point_1$, images with a yaw angle of approximately $starting\_yaw - 8° \approx 3°$ are selected, based on the current example. This process is continued until the desired number of root points

is obtained, which in this case is 5. So, the last $root\_point_5$ will be derived from the CelebA image subset with a yaw angle of approximately $35°$.



(a) (b)

Figure 4.22: In Figure (a), the root point is computed using the embedding of the mean image, while in Figure (b), it is computed as the average of the embeddings of similar images.

In Figure 4.22, it can be observed that the average of all embeddings was preferred over computing the embedding directly on the ”mean images”. This decision was made due to the fact that using the average of all embeddings results in less noisy root points.

Algorithm 4 outlines the steps involved in this operation, and the resulting root points for this example are illustrated in Figure 4.23.

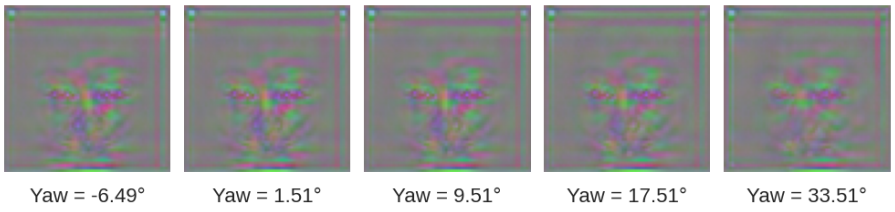


Figure 4.23: The full set of root points that were computed for the left direction.

Algorithm 4: Method 2: Root points computation

```
Input: image, n_points, yaw_step
Output: List containing the root points
root_points := []
orientations := getOrientation(image)
attributes := getAttributes(image)

for i ← 0 to n_points do
    target_orientations = orientations + i · yaw_step
    similar_images = filterCelebA(target_orientations, attributes )
    embeddings := []
    for j ← 0 to length(similar_images) do
        embeddings.append(EmbeddingModel.predict(similar_images[j]))
    end
    root_point_i = average(embeddings)
    root_points.append(root_point_i)
end
return root_points
```

- After obtaining the root points, the data must be prepared for linear regression. The independent variable, denoted as 'X', is just a sequence of integers from 0 to the number of root images. Meanwhile, the dependent variable, denoted as 'y', is obtained by flattening the root points computed in the previous step. This operation transforms the shape from $(n\_root\_points, 64, 64, 3)$ to $(n\_root\_points, 64 \cdot 64 \cdot 3)$, which is $(5, 12288)$ in this case.

  Once the data is prepared, a linear regression line can be fitted to the data, and the slope, denoted as m, of the regression line can be computed using the Ordinary Least Squares method. This process is described by

the following equation 4.8:

$$\begin{aligned} y &= mX + b + \epsilon \\ m &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \end{aligned} \tag{4.8}$$

In this equation, b represents the intercept term, and ϵ represents the error term.

- After calculating m, we can use this slope to create a new line with the same slope, but passing through the embedding of the starting image, which is represented by the point (x_1, l_1). To accomplish this, we can use the point-slope form of the equation of a line, which is given by:

$$y - l_1 = m \cdot (x - x_1)$$

Since the input image has the same orientation as the first root point, where the x coordinate is defined as 0, x_1 is set to 0. After performing some algebraic manipulations, the line shown in equation 4.9 is obtained:

$$y = m \cdot (x) + l_1 \tag{4.9}$$

- Obtaining the equation 4.9 allows for sampling points in the latent space of the diffusion model from this line by substituting values for x. The best results are typically obtained by using values of x within the range of $[0, \ldots, n\_root\_points]$. After reshaping the samples to their original dimensions of $(64, 64, 3)$, the samples can be passed as input to the diffusion model to generate conditioned images with different orientations.

Figure 4.24 provides a graphical representation of this method, where Principal Component Analysis (PCA) is utilized to visualize all the components involved in this example in two dimensions.

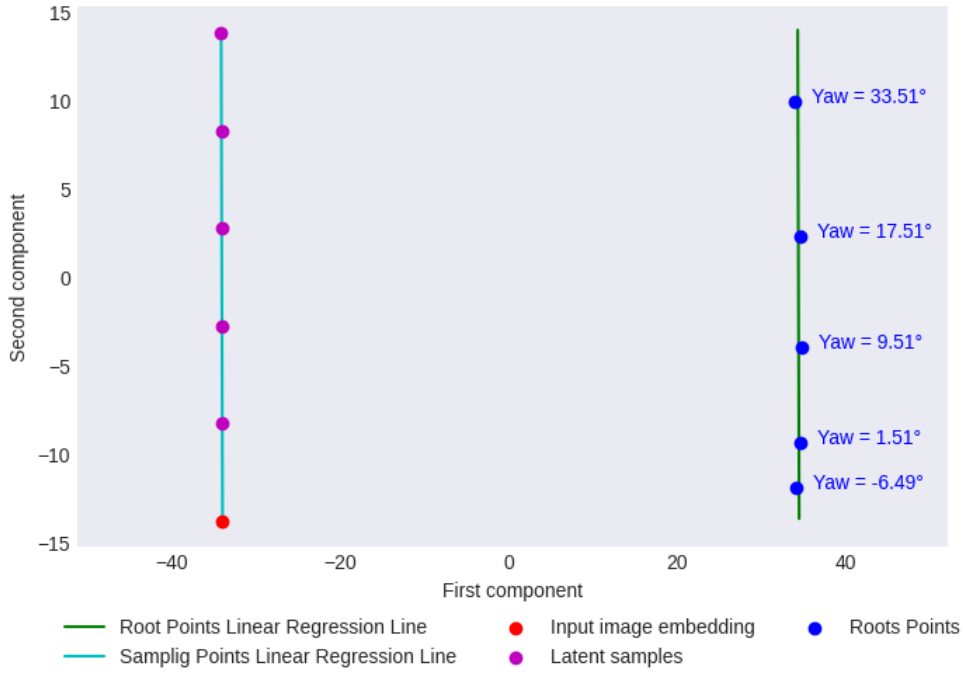
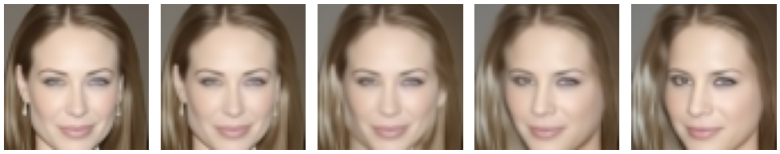


Figure 4.24: Visualization of all the components involved in the second method using the PCA with 2 principal components.

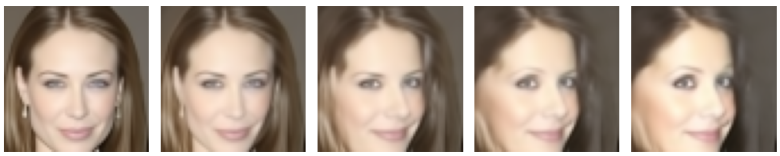
Furthermore, the complete process for this method is outlined in pseudocode in Algorithm 5. Additionally, the results for sampling images conditioned to the left direction are presented in Figure 4.25, while the results for sampling images conditioned to the right direction are shown in Figure 4.26.



(a) Not post-processed

(b) Post processed with CodeFormer

Figure 4.25: This image shows the result of sampling using a linear regressor with root points that extend towards the left direction.



(a) Not post-processed

(b) Post processed with CodeFormer

Figure 4.26: This image shows the result of sampling using a linear regressor with root points that extend towards the right direction.

Algorithm 5: Method 2 pseudocode

Input: image, n_root_points, yaw_step, x_sampling_points
Output: Images with different head orientation
output := []
root_points := getRootPoints(image, n_root_points, yaw_step)
root_points = [flattenMatrix(r) for r in root_points]

slope:=linearRegression(x= [0, . . . , n_root_points], y = root_points)
slope = reshapeToMatrix(slope, (64, 64, 3))

l_1 := EmbeddingModel.predict(image)
x_{input_image} := 0
foreach *x in x_sampling_points* **do**
    l_{cond} = slope · (x - x_{input_image}) + l_1
    y_{cond} = DDIM.reverseDiffusion(l_{cond})
    output.append(y_{cond})
end
return output

This type of conditioning in the DDIM latent space has enabled interesting manipulations in head orientation, covering a wide rotation angle of $\pm 30°$.

Moreover, it has the advantage of being rather ”intuitive” in its operation. These results are particularly intriguing, and suggest possibilities for future development, such as making the calculation of root points and slope independent of the input image. Since the subset of attributes on which the slope depends is shared by images with similar attributes, the same root points and slope could be used for different images.

However, it's important to note that there are limitations to this method. For instance, the CelebA dataset used in this study has biases towards female and young faces over male and elderly faces, as discussed in section 4.2.1. Additionally, the dataset has limited variation in face orientation, which could affect the generalizability of the method, as described in section 4.2.3.

For a more comprehensive understanding of the methodology and stability of the slope computation process used in this method, please refer to Appendix A for a detailed discussion.

4.8 Differences between the two methods

The results presented in Figure 4.27 demonstrate that the second method outperforms the first method in most cases. Specifically, the first method tends to suffer from artifacts in subsequent steps, while the second method avoids this issue by generating images independently of those produced in earlier steps. This results in a more stable and reliable image generation process overall.

In addition to its improved stability, the second method also offers several advantages over the first method. One of the main advantages is that it requires tuning fewer parameters, namely only the number of root points, the yaw step between each root point, and the x coordinates for sampling in the latent space. This makes the second method much simpler to implement and easier to use than the first method, which requires tuning a larger number of parameters.

Another advantage of the second method is that it typically produces more uniform and significant rotation in the generated images, allowing for greater

control over the rotation angle and resulting in more consistent results. In contrast, the first method is more prone to producing images with uneven or irregular rotation.

Overall, the results presented in Figure 4.27 strongly suggest that the second method is superior to the first method for generating rotated images. Its improved stability, simplicity, and consistency make it a highly attractive option for a wide range of applications in image processing and computer vision.

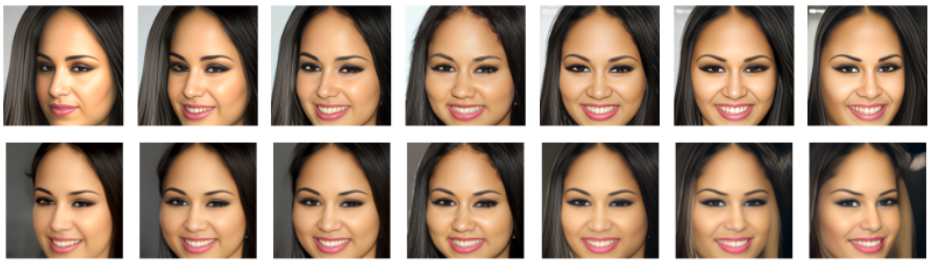


Image 132

Image 114

Image 16399

Figure 4.27: This figure compares the outputs of the two methods on three CelebA images. The input image is in the middle, with the first method's output on the top row and the second method's output on the bottom row.

Chapter 5

Conclusions

This thesis provides a comprehensive overview of facial rotation techniques and the challenges posed by traditional approaches that rely on ground truth data for training. To overcome these limitations, two novel techniques leveraging the latent space of a DDIM trained to reconstruct human faces were proposed to condition the generation process to produce rotated faces.

Both techniques can produce reasonably accurate face rotations without the need for a dedicated network or specially curated ground truth dataset. The first technique modifies specific pixels in the input image fed to the diffusion model, while the second technique fits a linear regressor to sample from the latent space. However, the first method requires extensive parameter tuning and may produce artifacts as the number of steps increases.

The second technique is superior to the first one, with improved stability and simplicity. It can effectively produce a wide rotation angle of $\pm 30^{\circ}$ while preserving important facial attributes. It is worth noting that the biases and limited variation in face orientation in the CelebA dataset used for this study may affect the generalizability and applicability of the proposed techniques to other datasets and more extreme head poses.

Future research directions may include using a more unbiased dataset with greater variation in face orientation to improve the DDIM's ability to generate

rotated images with angles larger than $\pm 30^{\circ}$. Additionally, further development of the second method by making the calculation of root points and slope independent of the input image could lead to greater computational efficiency and more versatile image generation.

Appendix A

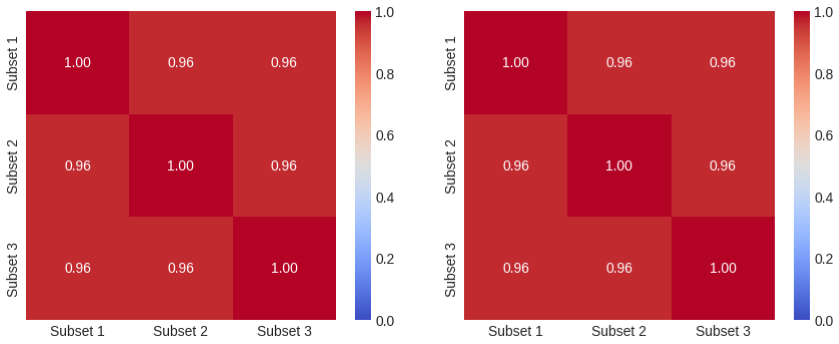
Slope computation analysis

This section analyzes the calculation of slope using various image subsets, assesses the stability of the proposed method, and evaluates the usefulness of the attributes considered. The primary objectives are to provide a comprehensive understanding of the factors that influence slope calculation and their impact on the accuracy of the results.

A.1 Slope Stability analysis

The first analysis aims to evaluate the stability of the proposed slope computation method by determining whether consistent results can be obtained when using different image subsets with the same orientations and attributes. To accomplish this, three distinct image subsets will be used to calculate the slope and conduct a cosine similarity analysis between them. This procedure will be repeated for a total of four attribute subsets, ranging from the most common attributes to less common ones.

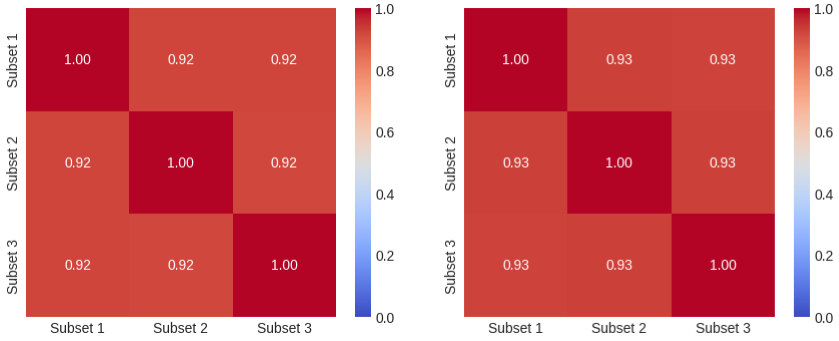
- The first attribute set includes: 'male' (-1), 'light_direction' ('CENTER'), 'young' (1), and 'smiling' (1). The left and right slopes of three non-overlapping image subsets with these attributes are computed, and their cosine similarity is visualized in a heatmap (Figure A.1).



(a) Left slopes (b) Right slopes

Figure A.1: First attribute subset cosine similarity heatmap.

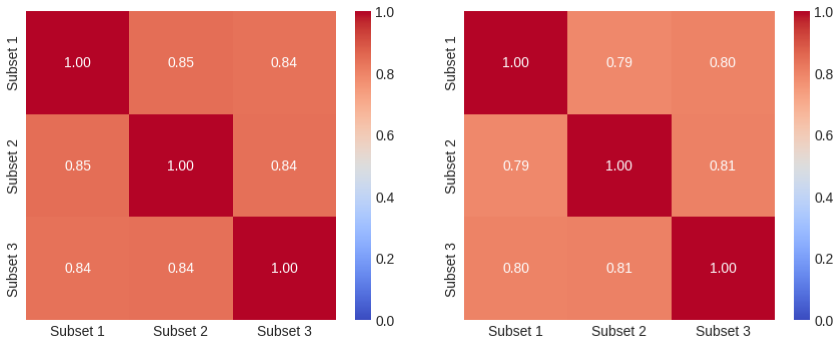
- The second attribute set comprises: 'male' (1), 'light_direction' ('CENTER'), 'young' (1), and 'smiling' (-1). Similar to the first case, the left and right slopes of three non-overlapping image subsets with these attributes are computed, and their cosine similarity is visualized in a heatmap (Figure A.2).



(a) Left slopes (b) Right slopes

Figure A.2: Second attribute subset cosine similarity heatmap.

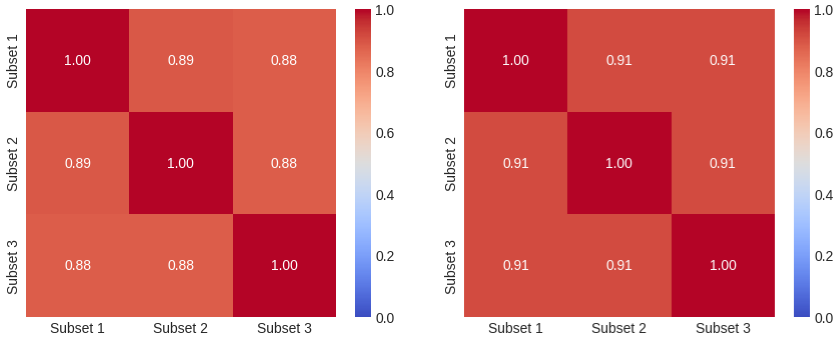
- The third attribute set consists of: 'male' (-1), 'light_direction' ('LEFT'), 'young' (-1), and 'smiling' (1). Again, the left and right slopes of three non-overlapping image subsets with these attributes are computed, and their cosine similarity is visualized in a heatmap (Figure A.3).



(a) Left slopes (b) Right slopes

Figure A.3: Third attribute subset cosine similarity heatmap.

- The final attribute set includes: 'male' (1), 'light_direction' ('RIGHT'), 'young' (1), and 'smiling' (1). Similarly, the left and right slopes of three non-overlapping image subsets with these attributes are computed, and their cosine similarity is visualized in a heatmap (Figure A.4).



(a) Left slopes (b) Right slopes

Figure A.4: Fourth attribute subset cosine similarity heatmap.

As expected, when using a common attribute subset such as the first one tested, the slope computation is highly stable, regardless of the image subset used for the computation. The cosine similarity never falls below 0.96. However, when employing a rarer attribute subset, the process becomes less stable. For instance, in the third case, both 'young=-1' and 'light_direction=LEFT' significantly reduce the available images in the dataset for slope computation. Consequently, the slope calculation becomes less stable, and the cosine

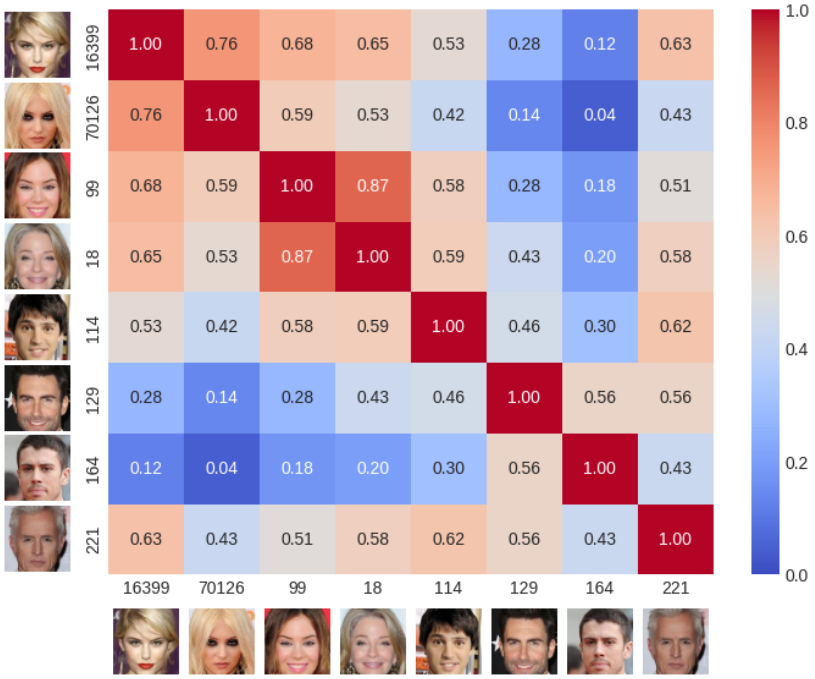
similarity between different subsets with the same attributes can decrease to approximately 0.80 in certain cases.

A.2 Slope Similarity Analysis among different Images

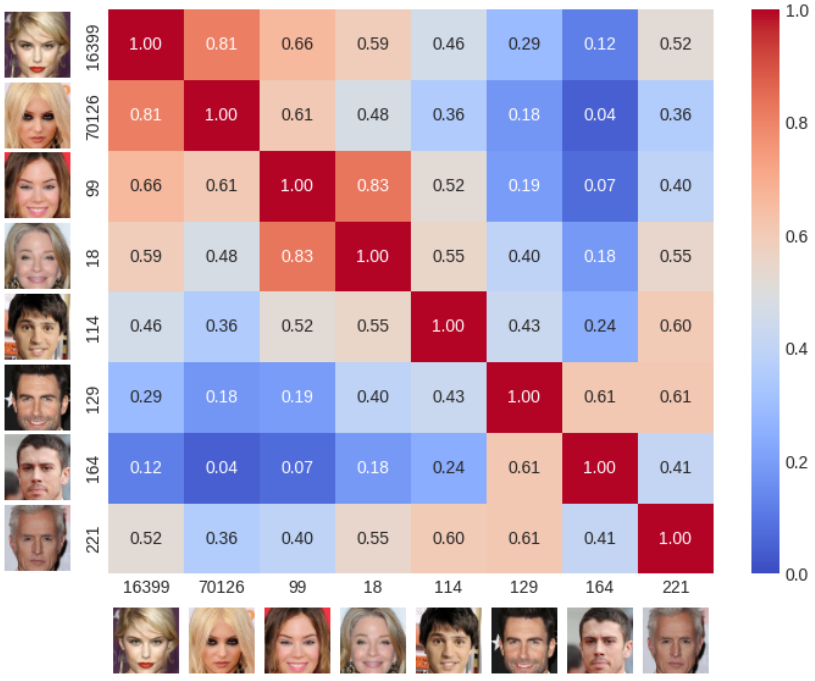
The second analysis examines the similarity of slopes among different images with varying orientations and attributes, to determine whether similar images have similar slopes and vice versa. To conduct this analysis, a sample of eight CelebA images will be used, and the corresponding cosine similarity heatmap is presented in Figure A.5.

The results show that similar images have similar slopes, with gender and light direction appearing to have the most significant effect on slope similarity. For instance, the cosine similarities between images 16399, 70126, 99, and 18, which are all women, are higher than those between these images and images 114, 129, 164, and 221, which are all men. Moreover, when both the gender and light direction are the same, the similarity is even higher, as observed for images 99 and 18, where both have ”light_direction = center”.

Interestingly, the similarities for male images appear to be generally lower, which may be due to the bias in CelebA towards female images, resulting in fewer images available for slope computation. Additionally, small changes in the yaw degree of $\pm 5^{\circ}$ seem to impact the similarity, as seen in images 114 and 164. Lastly, the ”young” attribute also appears to influence the slope, with images 18 and 221, both having ”young = -1”, exhibiting slightly similar slopes according to cosine similarity.



(a) Left slopes



(b) Right slopes

Figure A.5: Cosine similarities heatmap between different images

A.3 Effects of Attribute Removal on Slope Computation

The third analysis aims to investigate the impact of removing one attribute from the full subset of selected attributes, which includes 'male', 'light_direction', 'young', 'smiling', and 'mouth_slightly_open'. The objective is to determine whether any of these attributes can be disregarded without affecting the accuracy of the slope computation. A subset of three images, namely image 18, image 114, and image 99, have been selected to perform this analysis. These images are displayed in Figure A.6.

(a) Image 18 (b) Image 114 (c) Image 99

Figure A.6: Images used for attribute utility analysis.

The attribute values for each image are as follows:

- Image 18: 'male' = -1, 'light_direction' = 'CENTER', 'young' = -1, 'smiling' = 1, and 'mouth_slightly_open' = 1;
- Image 114: 'male' = 1, 'light_direction' = 'RIGHT', 'young' = 1,'smiling' = -1, and 'mouth_slightly_open' = 1;
- Image 99: 'male' = -1, 'light_direction' = 'CENTER', 'young' = 1, 'smiling' = 1 and 'mouth_slightly_open' = 1;

The experiment's results are presented in Tables A.1 and A.2, which display the cosine similarities for the missing attributes on the left and right slopes, respectively.

Image ID	Missing attribute				
	Mouth Slightly Open	Smiling	Young	Light Direction	Male
18	0.99	0.99	0.92	0.96	0.92
114	0.98	0.95	0.96	0.86	0.82
99	0.98	0.98	0.99	0.98	0.97

Table A.1: Results of left slopes cosine similarities.

Image ID	Missing attribute				
	Mouth Slightly Open	Smiling	Young	Light Direction	Male
18	0.96	0.99	0.93	0.96	0.92
114	0.99	0.97	0.98	0.88	0.83
99	0.98	0.98	0.98	0.97	0.97

Table A.2: Results of right slopes cosine similarities.

Based on the results, it appears that the attribute ”mouth_slightly_open” can be safely removed, as it consistently demonstrated very low impact on similarity across all tested cases. In fact, the cosine similarity without using this attribute was consistently around 0.98 relative to the slope that considers it. On the other hand, the impact of all other attributes on similarity varied depending on the characteristics of the considered image. For instance, if the input image is of an elderly woman, the attribute ”young” is more important than for a younger woman. This may be due to the bias in CelebA, which contains a much larger proportion of images of young people. Similarly, the attributes ”male” and ”smiling” also have varying impacts on similarity.

Interestingly, the ”light_direction” attribute appears to be much more important when the light direction is not central. This may be because root points computed with central light are similar to those computed without considering the light source at all. Overall, these findings suggest that careful consideration of image attributes is necessary to accurately assess similarity, particularly in cases where certain attributes may be more relevant than others depending on image characteristics.

Bibliography

[1] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015. arXiv: 1503.03585.

[2] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[3] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[4] C. Luo. Understanding diffusion models: a unified perspective. *arXiv preprint arXiv:2208.11970*, 2022.

[5] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[6] L. Weng. What are diffusion models? *lilianweng.github.io*, July 2021. url: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[7] A. Q. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[8] O. Ronneberger, P. Fischer, and T. Brox. U-net: convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[9] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[10] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

[11] J. Ho and T. Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[12] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[13] C.-H. Lee, Z. Liu, L. Wu, and P. Luo. Maskgan: towards diverse and interactive facial image manipulation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[14] A. Asperti, D. Evangelista, S. Marro, and F. Merizzi. Image embedding for denoising generative models. *arXiv preprint arXiv:2301.07485*, 2022.

[15] T. Hassner, S. Harel, E. Paz, and R. Enbar. Effective face frontalization in unconstrained images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4295–4304, 2015.

[16] X. Zhu, Z. Lei, J. Yan, D. Yi, and S. Z. Li. High-fidelity pose and expression normalization for face recognition in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 787–796, 2015.

[17] J. R. A. Moniz, C. Beckham, S. Rajotte, S. Honari, and C. Pal. Unsupervised depth estimation, 3d face rotation and replacement. *Advances in neural information processing systems*, 31, 2018.

[18] E. Härkönen, A. Hertzmann, J. Lehtinen, and S. Paris. Ganspace: discovering interpretable gan controls. *Advances in Neural Information Processing Systems*, 33:9841–9850, 2020.

[19] Y. Shen, J. Gu, X. Tang, and B. Zhou. Interpreting the latent space of gans for semantic face editing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9243–9252, 2020.

[20] R. Abdal, P. Zhu, N. J. Mitra, and P. Wonka. Styleflow: attribute-conditioned exploration of stylegan-generated images using conditional continuous normalizing flows. *ACM Transactions on Graphics (ToG)*, 40(3):1–21, 2021.

[21] L. Tran, X. Yin, and X. Liu. Disentangled representation learning gan for pose-invariant face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1415–1424, 2017.

[22] R. Huang, S. Zhang, T. Li, and R. He. Beyond face rotation: global and local perception gan for photorealistic and identity preserving frontal view synthesis. In *Proceedings of the IEEE international conference on computer vision*, pages 2439–2448, 2017.

[23] Y. Hu, X. Wu, B. Yu, R. He, and Z. Sun. Pose-guided photorealistic face rotation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8398–8406, 2018.

[24] Y. Qian, W. Deng, and J. Hu. Unsupervised face normalization with extreme pose and expression in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9851–9858, 2019.

[25] X. Yin, X. Yu, K. Sohn, X. Liu, and M. Chandraker. Towards large-pose face frontalization in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3990–3999, 2017.

[26] J. Deng, S. Cheng, N. Xue, Y. Zhou, and S. Zafeiriou. Uv-gan: adversarial facial uv map completion for pose-invariant face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7093–7102, 2018.

[27] J. Cao, Y. Hu, H. Zhang, R. He, and Z. Sun. Learning a high fidelity pose invariant model for high-resolution face frontalization. *Advances in neural information processing systems*, 31, 2018.

[28] H. Zhou, J. Liu, Z. Liu, Y. Liu, and X. Wang. Rotate-and-render: unsupervised photorealistic face rotation from single-view images. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5911–5920, 2020.

[29] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[30] J. Sun, X. Wang, Y. Zhang, X. Li, Q. Zhang, Y. Liu, and J. Wang. Fenerf: face editing in neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7672–7682, 2022.

[31] B. Dai and D. P. Wipf. Diagnosing and enhancing vae models. In *Seventh International Conference on Learning Representations (ICLR 2019), May 6-9, New Orleans*, 2019.

[32] A. Asperti, L. Bugo, and D. Filippini. Enhancing variational generation through self-decomposition. *IEEE Access*, 10:67510–67520, 2022. doi: 10.1109/ACCESS.2022.3185654. url: https://doi.org/10.1109/ACCESS.2022.3185654.

[33] P. J. Thilaga, B. A. Khan, A. Jones, and N. K. Kumar. Modern face recognition with deep learning. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 1947–1951. IEEE, 2018.

[34] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[35] S. Zhou, K. C. Chan, C. Li, and C. C. Loy. Towards robust blind face restoration with codebook lookup transformer. In *NeurIPS*, 2022.

[36] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[37] Q.-M. Liu, R.-S. Jia, C.-Y. Zhao, X.-Y. Liu, H.-M. Sun, and X.-L. Zhang. Face super-resolution reconstruction based on self-attention residual network. *IEEE Access*, 8:4110–4121, 2019.

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

Acknowledgements

As I approach the end of my university journey, I want to express my gratitude to everyone who has supported me along the way.

Firstly, I am deeply thankful to my supervisor, Prof. Andrea Asperti, for his patient guidance and giving me the opportunity to explore a fascinating topic.

My family, especially my parents Francesca and Michele, have always believed in me and provided unwavering support.

My friends, including Gabriele, Nicolas, and Gianluca, have shared both the joys and challenges of these years with me.

Finally, once again, I extend special thanks to Gabriele for his commitment to collaborating with me on this project.