UNIVERSITAT DE BARCELONA

Treball final de grau

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Comparative Analysis of State-of-the-Art Deep-Learning Based Face Editing Algorithms

Autor: María Ruiz Ávila

Directores: Dra. Petia Radeva i Dra. Maya Aghaei
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, June 10, 2024

Abstract

Facial attribute transformation, which involves manipulating specific facial features in images and videos, has become a focal point in computer vision and image processing.

This project conducts a comprehensive comparative analysis of cutting-edge methodologies, utilizing diverse models to modify latent imagery representations. We assess various state-of-the-art techniques in facial attribute editing through quantitative, qualitative, and efficiency metrics. Our study demonstrates the superior efficacy of an innovative approach using the Multi-Attribute Latent Transformer Model, which adeptly learns and modifies multiple facial attributes simultaneously. This model not only enhances operational efficiency but also maintains the authenticity and integrity of facial identities. Additionally, we investigate how the correlation of attributes in the training images introduces bias in the results.

As part of the project, we have developed a user interface that allows for the visual comparison of four models. This application enables users to observe and compare the distinctions and effectiveness of each model side-by-side.

In summary, this research advances the field of facial attribute modification by presenting an in-depth comparative study that highlights the strengths and limitations of leading methodologies in face editing, thereby laying the groundwork for future innovations in refined and scalable facial image transformation.

Keywords: Facial attribute transformation, InterFaceGAN, TediGAN, StyleCLIP, Single Latent Transformer, Latent multi-attribute transformer, Disentangled face editing, StyleGAN, Metric comparison, Attribute Correlation

Resum

La transformació d'atributs facials, que implica la manipulació de característiques facials específiques en imatges i vídeos, s'ha convertit en un punt focal en la visió per computador i el processament d'imatges.

Aquest projecte realitza una anàlisi comparativa exhaustiva de metodologies avançades, utilitzant diversos models per modificar representacions latents d'imatges. Avaluem diverses tècniques d'última generació en l'edició d'atributs facials a través de mètriques quantitatives, qualitatives i d'eficiència. El nostre estudi demostra l'alta eficàcia de l'enfocament innovador que utilitza el model Multi-Attribute Latent Transformer, que aprèn i modifica hàbilment múltiples atributs facials simultàniament. Aquest model no només millora l'eficiència operativa, sinó que també manté l'autenticitat i la integritat de les identitats facials. A més, investiguem com la correlació d'atributs en les imatges d'entrenament introdueix biaixos en els resultats.

Com a part del projecte, hem desenvolupat una interfície d'usuari que permet la comparació visual de quatre models. Aquesta aplicació permet als usuaris observar i comparar les diferències i l'efectivitat de cada model.

En resum, aquesta investigació fa avançar el camp de la modificació d'atributs facials presentant un estudi comparatiu en profunditat que destaca els punts forts i les limitacions de les metodologies líders en l'edició de rostres, establint així les bases per a futures innovacions en la transformació d'imatges facials de manera refinada i escalable.

Resumen

La transformación de atributos faciales, que implica la manipulación de características faciales específicas en imágenes y videos, se ha convertido en un punto focal en la visión artificial y el procesamiento de imágenes.

Este proyecto lleva a cabo un análisis comparativo exhaustivo de metodologías de vanguardia, utilizando diversos modelos para modificar representaciones latentes de imágenes. Evaluamos varias técnicas para editar atributos faciales a través de métricas cuantitativas, cualitativas y de eficiencia. Nuestro estudio demuestra la superior eficacia del enfoque innovador que utiliza el modelo Multi-Attribute Latent Transformer, el cual aprende y modifica hábilmente múltiples atributos faciales simultáneamente. Este modelo no solo mejora la eficiencia operativa, sino que también mantiene la autenticidad e integridad de las identidades faciales. Además, investigamos cómo la correlación de atributos en las imágenes de entrenamiento introduce sesgos en los resultados.

Como parte del proyecto, hemos desarrollado una interfaz de usuario que permite comparar visualmente cuatro modelos. Esta aplicación permite a los usuarios observar y comparar las diferencias y la efectividad de cada modelo en tiempo real.

En resumen, esta investigación contribuye al campo de la modificación de atributos faciales presentando un estudio comparativo que destaca las fortalezas y limitaciones de las metodologías líderes en la edición de rostros, sentando así las bases para futuras innovaciones en la transformación de imágenes faciales de manera refinada y escalable.

Acknowledgements

I would like to express my gratitude to all the people who contributed, directly or indirectly, to the realization of this project.

First of all, I would like to express my gratitude to Dr. Petia Radeva for introducing me to this project and for always encouraging me to take it one step further.

I would also like to extend my gratitude to my supervisor, Dr. Maya Aghaei, for all the support and guidance she has provided. Although she is not directly affiliated with my university, Maya has dedicated her personal time to assist me and share her expertise.

Finally, I want to thank Adria Carrasquilla for his collaboration and support. This project builds on his Master's Thesis, *Multi-Attribute Latent Transformer*; without it, this work would not exist. He contributed significantly with his knowledge on the topic, always offering his help and assisting in analyzing the results.

On a different note, even though they did not have a direct impact on the project, I also want to thank my family and friends who showed emotional support not only during the realization of this project but also throughout the completion of both degrees. I would like to especially thank Oscar, who has always been there during my lowest moments, and Javier, who is always even more proud than I am of my achievements.

Contents

Chapter 1

Introduction

The objective of this chapter is to outline the context and motivations behind this project, detail our methodological approach, and provide an overarching summary of the dissertation.

1.1 Motivation

The field of computer vision has seen remarkable advancements over the past few years, particularly with the introduction of generative adversarial networks (GANs). Among these, StyleGAN has stood out as a groundbreaking model that has significantly influenced face editing technologies. Since its inception, StyleGAN has demonstrated an unprecedented ability to generate highly realistic human faces, revolutionizing both academic research and practical applications in image synthesis, editing, and enhancement.

However, despite the impressive capabilities of StyleGAN, there are still some limitations in the area of facial edition that warrant further exploration. One key motivation for this project is to delve into these limitations and investigate how subsequent models have addressed or failed to address them.

Moreover, the rapid evolution of face editing models post-StyleGAN provides a rich landscape for comparison and analysis. Each new model introduces novel techniques and optimizations aimed at overcoming specific shortcomings of its predecessors. By conducting a comprehensive comparison of these models, this project aims to provide deeper insights into the current state of face editing technologies and highlight areas where further improvements are necessary.

Another motivating factor is the increasing relevance of these technologies in

real-world applications. From digital entertainment to security and forensic analysis, the ability to edit and generate realistic human faces has far-reaching implications.

In summary, this project is driven by a desire to contribute to the ongoing dialogue in the computer vision community about the capabilities and limitations of face editing models. By examining the evolution of face editing models, this research aims to shed light on the progress made thus far and identify potential directions for future development.

1.2 Context

The foundations of this project were laid in September 2023, while the author was doing an internship in Madrid. In this period, virtual discussions took place between the author and their mentor in Barcelona, Petia Ivanova Radeva. After a couple of meetings, Radeva suggested a collaboration with Maya Aghaei Gavari from the Netherlands, who had recently supervised a master's thesis on Face Editing. Authored by Adria Carrasquilla, this thesis introduced an innovative approach to facial attribute manipulation through a Latent Multi-Attribute Transformer. From this point, the author embarked on a journey to conduct a comparative analysis of this model against others in the field, evaluating their advantages and disadvantages.

1.3 Objectives of the project

In this project, we want to analyze in depth the performance of different face editing models and compare their capabilities. We also want to understand how initial attribute correlation in the training dataset affect non targeted attributes in the resulting modified images.

Therefore, the objectives of this project are:

- To achieve the necessary knowledge on which the Face Attribute Editing models are based.
- To understand and train several models to be able to evaluate them under the same basis and compare them.
- To study the introduced bias due to attribute correlation in the training dataset.

- To implement a user interface that allows for the visual comparison of the models.

1.4 Project Planning

This project was carried out during 2024. The work had already been agreed before but, as some time was necessary to acquire the foundational knowledge of neural networks while working on a full-time internship, the first official meeting between the involved parties to commence the project in earnest was held in mid-January.

From September 2023 to December 2023, we used mainly the *Dive into Deep Learning* book [1] and Andrew Ng's CS231 course [2] to acquire general knowledge in the field of computer vision and neural networks.

The month of January was committed to understanding different evaluation metrics and deciding which ones we were going to use for our study.

It was in February when we started training the different models under the same basis and getting the first results. By March, we had already got some insightful results on the performance of the desired models and this memory was started.

In April and May, a study on how the correlation of attributes introduced a bias on the results when training the models was made. Additionally, an application for visually comparing the studied models was implemented. Finally, the memory was finished.

1.5 Organization of the memory

This report is organized as follows:

1. **Introduction**. A presentation of the thesis motivation and the work organization.
2. **Scientific background**. Basic information needed in order to understand the difference between the models we have worked with.
3. **State of the Art**. Comprehensive overview of the current advancements and leading models in the field of facial editing.

4. **Methodology**. In depth explanation about the five models we have worked with and approach to attribute correlation analysis.

5. **Experiments and Results**. Conducted experiments to compare state-of-the-art facial attribute editing models.

6. **Conclusions**. Evaluation of the objectives achieved and the work done.

7. **Bibliography**. Compilation of the articles and materials used.

8. **Appendix**. Screenshots of the developed UI.

Chapter 2

Scientific background

This chapter offers an introduction to the essential ideas in machine learning. Our goal is to establish a solid understanding of the fundamental principles and foundational components that are crucial to the models explored later in this document. Beginning with the most elementary concepts, we will methodically advance towards tackling the present challenge.

2.1 Foundations of Artificial Neural Networks

Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) are interdisciplinary fields that merge concepts from computer science, mathematics, and cognitive science to develop algorithms capable of learning from data and making predictions or decisions.

These terms are often mistakenly used interchangeably, but they are distinct. AI is a broad academic discipline encompassing any technique that enables computer systems to perform tasks that typically require human intelligence. ML, a subset of AI, involves methods that allow systems to learn from training data and improve through experience, rather than being explicitly programmed. DL, a further subset of ML, utilizes artificial neural networks (ANNs) with three or more layers to mimic the human brain's learning process.

Definition 2.1. An **artificial neural network** is a computational model inspired by the neural networks in the human brain, designed to mimic how the brain processes information and learns from experiences.

Definition 2.2. At the heart of an ANN are **artificial neurons**, which are simplified, abstract versions of biological neurons. These neurons receive, process, and pass on information to other neurons, facilitating a complex network of data

transmission similar to neural activity in the brain. A neuron is a function f made of the composition of a linear function $g : \mathbb{R}^n \to \mathbb{R}$ and an activation function h (non-linear function) (see Equation (2.1)).

$$f(x) = h(g(x)) = h\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

The activation function 'decides' whether a neuron should be activated or not, i.e. it evaluates whether the information of the neuron is relevant to the network or not.

There are several types of activation functions. The first version proposed in what is considered the first ANN, created by Warren S. McCulloch and Walter Pitts [26], was a simple threshold.

$$f(x) = \begin{cases} 0, & \text{if } x < t \\ 1, & \text{if } x \geq t \end{cases} \tag{2.2}$$

Later, functions such as the Sigmoid [42] were considered:

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

This function 'potentiates' the value and projects it between 0 and 1. High values tend asymptotically to 1, but low values tend asymptotically to 0, becoming negligible. However, this function has some problems that do not allow the system to shine completely. Without going into too much detail, it is said that this function saturates and kills the gradient, causing it to converge very slowly or even not at all.

There are other famous activation functions such as $tanh(x)$, but the one that really revolutionized the world of machine learning was the Rectified Linear Unit [41], also known as ReLU.

$$\text{Relu}(x) = \max(0, x) \tag{2.4}$$

The properties of this very simple function allowed the training of much more complex models and led to a great development in the field of neural networks.

Going back to speaking about neurons, they are arranged in layers that sequentially handle information: the first layer (**input layer**) receives the initial data, while the final layer (**output layer**) produces the result of the computation. Between these, none, one or more **hidden layers** exist, contributing to the network's ability to perform complex processing through intermediate computations.

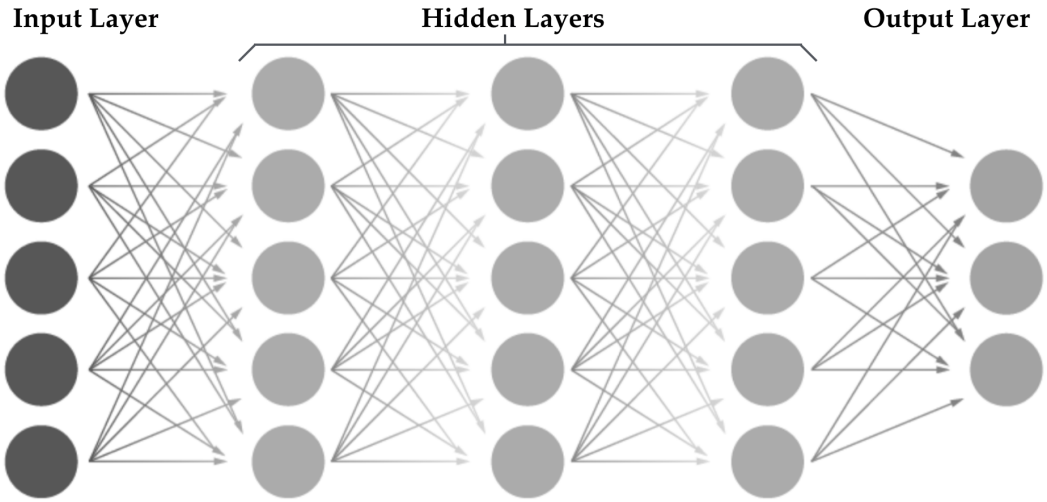

Figure 2.1: Artificial Neural Network Basic Structure

The learning process in an ANN, where the network learns to predict outputs from given inputs, involves adjusting weights.

Definition 2.3. In Equation (2.1), $w_i \in \mathbb{R}$, $i \in \{1, \ldots, n\}$, are called **weights** of the neuron while $b \in \mathbb{R}$ is called **bias**.

Definition 2.4. We call **parameter** to all those values of the neural network that can vary and can be trained, like the weights and the bias.

Initially set to random values, weights and biases are iteratively adjusted through a training process automatically, not manually. The adjustment of weights is guided by the difference between the actual output of the network and the expected output (the error), aiming to minimize this error across all training examples. This method of learning, emphasizing the critical role of weights in shaping the network's knowledge and predictive capabilities, is fundamental to the operation and effectiveness of artificial neural networks.

Definition 2.5. We call **hyperparameters** the parameters that we manually modify in a neural network. They are very important to achieve good performance, since they shape the neural network and define its training. Some examples are: the number of layers, the number of neurons per layer, the learning rate and the batch size.

2.2 Training and Convergence

There are three primary approaches to training neural networks, each distinct in how they learn from data:

1. **Supervised Learning**. This is the most straightforward method, utilizing a dataset with explicit labels to guide the learning process. In supervised learning, the model adjusts its parameters based on input-output pairs, striving to deduce the underlying rules that can generalize beyond the training data to make accurate predictions. It's typically used for regression and classification tasks.

2. **Unsupervised Learning**. Unlike supervised learning, unsupervised learning algorithms work with unlabeled data. They aim to uncover hidden patterns or structures within the data without any explicit guidance on the output. Common applications include clustering and association rule mining, where the goal is to group similar items or find relationships between different elements in the dataset.

3. **Reinforcement Learning**. This approach involves learning to make decisions by interacting with an environment. Through a process of trial and error, the model learns from the consequences of its actions, receiving rewards for beneficial actions and penalties for detrimental ones. The objective is to learn a strategy that maximizes the cumulative reward over time.

For all these training paradigms, a loss function is crucial. It quantifies the model's performance, measuring how well or poorly it is achieving its task, thereby guiding the adjustment of its parameters to optimize performance.

Definition 2.6. A **loss function** $L(y, \hat{y})$ quantifies the difference between the true values y and the predicted values $\hat{y}$ produced by the model. It is a crucial component in the training of machine learning models, guiding the optimization process. The objective during training is to minimize the loss function, which in turn improves the model's predictions. Mathematically, it can be represented as:

$$L(y, \hat{y}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.5}$$

for a simple mean squared error loss function, where n is the number of observations in the dataset.

Equation (2.5) measures the average squared difference between the estimated values and the true values. Since it is derived from the square of the Euclidean

distance, it is always positive and decreases as the error approaches zero.

An example of a fairly simple loss function would be the **Mean Squared Error (MSE)**.

Let x be the target value and y the value predicted by the system, both n-element vectors, then the MSE formula is:

$$\text{MSE}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2. \tag{2.6}$$

This function measures the mean squared error, i.e. the average squared difference between the estimated values and the true values. Since it is derived from the square of the Euclidean distance, it is always positive and decreases as the error approaches zero. It is usually applied to regression models.

The most used loss function in classification problems and the one that we will use during practically all the project is the **Cross-Entropy function**, also known as **Log Loss**:

$$L : [0,1]^n \times \{0,1\}^n \to \mathbb{R} \tag{2.7}$$

$$L(p, y) = -\sum_{c=1}^{n} y_c \log(p_c), \tag{2.8}$$

where n is the number of classes, log is the natural logarithm, y_c is the binary indicator (0 or 1) that indicates if class label c is the correct classification for this observation, and p_c is the probability predicted by the system of this observation being of class c.

This function satisfies the following properties:

1. $L(p, y) \geq 0$, $\forall p \in [0,1]^n, y \in \{0,1\}^n$ because of the fact that $\log(x) \leq 0\ \forall x \in (0,1]$
2. $\lim_{p \to y} L(p, y) = 0$.

Now that we have a function to evaluate the results, the logical question would be how to optimize this function to improve the model performance.

Definition 2.7. We call **forward propagation** to the process in which neural networks create predictions from inputs. Initially, the weight values W and the bias b are initialized to random values in each neuron. The training data goes through various layers until it reaches the output layer, where we give it its desired shape. Then, the loss function evaluates these predictions and gives a score. From this

score, we will apply the back-propagation algorithm, which will update the parameters. This cycle is repeated as many times as we indicate.

The Back-propagation algorithm is very important, because it is in charge of understanding how the parameters aspect the result and optimizing the loss function to improve the neural network's predictions. To run this algorithm we need an Optimizer.

Definition 2.8. An **optimizer** is an algorithm that modifies the parameters of the neural network so that the loss function is minimized.

One of the most famous optimizers is the **Stochastic Gradient Descend (SGD)** method. This iterative algorithm is based on finding the value of each parameter that minimizes the loss function. To find this value, we use the gradient of the loss function:

$$x^{k+1} = x^k - \eta \cdot \frac{\nabla L(x^k)}{\|\nabla L(x^k)\|} \tag{2.9}$$

where k indicates the iteration, ∇ is the gradient of the loss function, η is a parameter that dictates how much we "move" in that direction, it is usually called learning rate and can be fixed or variable during iterations, L is the loss function and $x = (x_1, \ldots, x_n)$ are the parameters.

Definition 2.9. The **gradient** of a function $f : \mathbb{R}^n \to \mathbb{R}$ is a vector formed by its partial derivatives:

$$\nabla f(x_1, x_2, \ldots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1, x_2, \ldots, x_n) \\ \frac{\partial f}{\partial x_2}(x_1, x_2, \ldots, x_n) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_1, x_2, \ldots, x_n) \end{bmatrix}. \tag{2.10}$$

Calculating the derivative can be a very difficult or even an impossible task to do computationally, however, it can be easily approximated.

The formula for an analytical derivative of f is

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{2.11}$$

But we can approximate it fairly accurately with a very small h value. Usually, $h = 1e - 6$ is used, lower values are considered noise.

$$\frac{f(x+h) - f(x)}{h}. \tag{2.12}$$

With the SGD optimizer, we can modify the parameters until we reach the combination that minimizes the loss function. However, there is a problem: during training we will probably have millions of parameters and millions of target outputs, comparing them all at once can be extremely expensive computationally.

Hence the 'stochastic' part of the name. In each iteration, instead of taking all the data points, we want to learn on, we will sort them randomly and apply the algorithm on one or a few at a time. This way we may not go straight to the minimum, we may go back and forth a bit, but we will end up getting there anyway and with a much lower computational cost.

Each time we evaluate on an example and update parameters, it is called a **step** and each time we have iterated through all the examples we call it an **epoch**.

2.2.1 Datasets for training

Before finishing this section, we have to define some basic concepts that will be discussed during the work.

Definition 2.10. The **batch size** is the number of data units that each epoch will work with.

When working with large amounts of data, as in this case, it is important to divide the information into smaller batches in order not to saturate the computation. It may be the case that the machine does not have enough memory space for so much information at once or that it is not able to process it.

Definition 2.11. Over-fitting refers to when a neural network is trained so much on the same training data that it fits them too much, causing it to give good results on this set, but losing generality and giving bad results on any other data.

Definition 2.12. Under-fitting is the opposite of over-fitting. It happens when the neural network did not learn enough on the training data and, in general, performs very poorly in all predictions.

Once we have a dataset to train our model on, we have to divide it into two or three disjoint parts:

- **Training set**: this is the data set used for the training of the neural network.
- **Validation set**: data set on which we will evaluate the training results and evaluate how the hyper-parameters performed in order to find the best hyper-parameters for the network.

- **Testing set**: this set is not used during training, it is used for the final performance evaluation once the training is finished and we have found the optimal hyper-parameters.

It is important that these sets do not overlap. We want the validation and test sets not to contain data from the training set in order to be able to assess whether over-fitting is occurring.

2.2.2 Training Properties

Many times, training a big model from scratch with random weights can be a very demanding task, as it can take a lot of time and resources. This is why this property is very interesting.

Theorem 2.13. It is more efficient to take an already trained neural network model, even if it was trained on a completely different dataset and adapt it to our problem than to train one from scratch.

There are many steps in the execution of a neural network, such as extracting features from the information or establishing relationships between them. Most of these steps are shared among all models, and, no matter how different the problems are, the structure of the neural network will be similar. Therefore, we can take a network already trained on another problem, change the input and output layer to match our situation, and even change more layers if we want, and we will save a lot of time and resources. When we adapt a neural network like this, we have to do some extra training to make it consolidate to the new problem.

Definition 2.14. We call **fine-tuning** to finish tuning an already trained neural network to a new problem and environment by further training it a bit more and adjusting the hyper-parameters.

2.3 Generative Adversarial Networks

Definition 2.15. A **Generative Artificial Neural Network** is a type of AI model that has the ability to create data such as text, images, video, or audio, that is, or aims to be, indistinguishable from real-world examples.

The field of generative artificial intelligence has experienced remarkable growth in the past decade. Image generative models have been around since the late 2000s when auto-encoders were utilized to compress and expand information, enabling the inference of new data. In 2013, Variation Auto-encoders [15] were

introduced, incorporating stochastic mechanisms into these models. A significant milestone occurred in 2014 with the introduction of Generative Adversarial Networks (GANs) [27], pioneering the generation of realistic images, style transfer, and accommodating diverse inputs for this purpose. More recently, starting from 2019, there has been an influx of research papers on diffusion models which begin with a noisy image and progressively remove noise until reaching a real looking sample. In the field of text generation, Recurrent Neural Networks (RNNs) [35] gained prominence around 2010, followed by the adoption of VAEs [15] and GANs adapted to text generation, and finally the utilization of transformers structures, that leaded to the widely-known GPT [38] models.

The focus of this project will be on the field of GANs, which represented a paradigm shift in unsupervised learning and generative models. A GAN consists of two competing neural network models:

1. **Generator**. It learns to generate plausible data. The generated instances become negative training examples for the discriminator.

2. **Discriminator**. It learns to distinguish the generator's fake data (instances created by the generator) from real data (such as real pictures of people). The discriminator penalizes the generator for producing implausible results by updating its weights through back-propagation from its loss through the discriminator network.

As we have mentioned before, usually, to train a neural net, we alter the net's weights to reduce the error or loss of its output. In our GAN, however, the generator is not directly connected to the loss that we're trying to affect. The generator feeds into the discriminator net, and the discriminator produces the output we are trying to affect. The generator loss penalizes the generator for producing a sample that the discriminator network classifies as fake.

This extra chunk of network must be included in back-propagation. Back-propagation adjusts each weight in the right direction by calculating the weight's impact on the output. But the impact of a generator weight depends on the impact of the discriminator weights it feeds into. So back-propagation starts at the output and flows back through the discriminator into the generator. But, at the same time, we don't want the discriminator to change during generator training. For avoiding that, we train the generator as follows:

1. Sample random noise.

2. Produce generator output from sampled random noise.

3. Get discriminator "Real" or "Fake" classification for generator output.
4. Calculate loss from discriminator classification.
5. Back-propagate through both the discriminator and generator to obtain gradients.
6. Use gradients to change only the generator weights.

But, since we have two different networks, the discriminator and the generator, to train, how do we train the GAN as a whole? The training proceeds in alternating periods:

1. The discriminator trains for one or more epochs.
2. The generator trains for one or more epochs.
3. Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

We keep the generator constant during the discriminator training phase. As discriminator training tries to figure out how to distinguish real data from fake, it has to learn how to recognize the generator's flaws. Similarly, we keep the discriminator constant during the generator training phase. Otherwise the generator would be trying to hit a moving target and might never converge.

It is this back and forth that allows GANs to tackle otherwise intractable generative problems. We get a toehold in the difficult generative problem by starting with a much simpler classification problem.

As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy.

2.3.1 StyleGAN

Many variants of GANs have been proposed to improve the quality of generated images or allow conditional synthesis. However, they had yet to offer intuitive, scale-specific control of the synthesis procedure until StyleGAN [4].

The first version of StyleGAN might be the biggest game changer in GAN works. As prior works just focused on improving stability, convergence, image diversification and resolution, StyleGAN emphasized on how to make attributes

of generated images could be further edited independently. It was thus mainly designed to disentangle the latent codes and enable the separate controls over style generation. The introduction of StyleGAN sparked the findings of techniques to utilize the ability to perform realistic manipulation on real images, which later became GAN inversion.

Latent Spaces of StyleGAN

Let us begin by thoroughly discussing the StyleGAN dataflow pipeline and its latent spaces. In the original GAN architectures, the latent code was found to be highly entangled and difficult to use for controlling the output image features.

One of the key ideas of StyleGAN architectures is to introduces more than one innate latent space ($\mathbb{Z}$, $\mathbb{W}$, $\mathbb{S}$), thereby allowing to learn intermediate latent representations with properties better tailored to the semantic structure of the image space. Moreover, to increase the expressive power of StyleGAN, it is common to work with extensions of these spaces ($\mathbb{Z}+$, $\mathbb{W}+$).

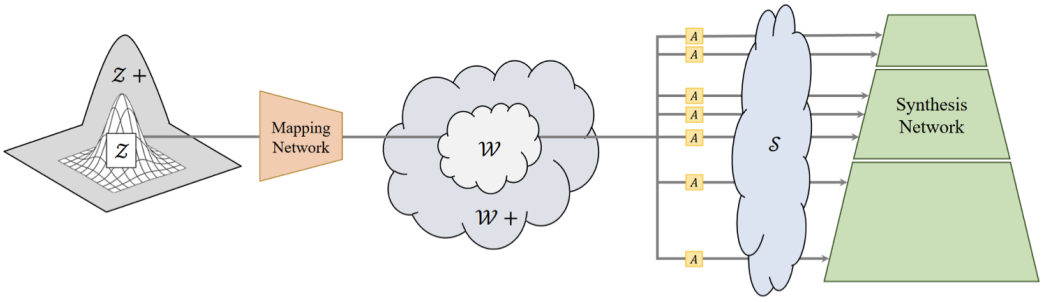

Figure 2.2: Latent Spaces of StyleGAN, extracted from [29].

In the StyleGAN architectures, 512-dimensional samples from an isotropic normal distribution with unit variance and zero mean provide the random inputs $z \in \mathbb{Z}$ at the root of the entire generation pipeline. $\mathbb{Z}+$ space implies sequential mapping of 18 $z \in \mathbb{Z}$ vectors into 18 corresponding $w \in \mathbb{W}$ vectors.

Latent codes from $\mathbb{Z}$ are transformed to latent codes in the 512-dimensional $\mathbb{W}$ space of StyleGAN through the mapping network (see Figure 2.5). This transformation, which must be learned during training, allows to distort the simple $\mathbb{Z}$-distribution into a distribution $\mathbb{W}$ of the same dimension of style parameters. In this $\mathbb{W}$ space meaningful editing operations on images can become realizable by simple axis-parallel movements of a point [4]. The eight fully connected map-

ping layers of the StyleGAN's mapping network can provide the adaptability to unfold the disc-shaped $\mathbb{Z}$ space into a space $\mathbb{W}$ whose shape is much closer to the required feature distribution (Figure 2.6C). Figure 2.6 illustrates a situation where the feature distribution of to-be-generated images excludes a combination of two features, leading to a distribution where one quadrant of all possible combinations is absent (Figure 2.6A). To create such a distribution from the disc-shaped input distribution $\mathbb{Z}$ (Figure 2.6B) requires a very non-linear mapping.

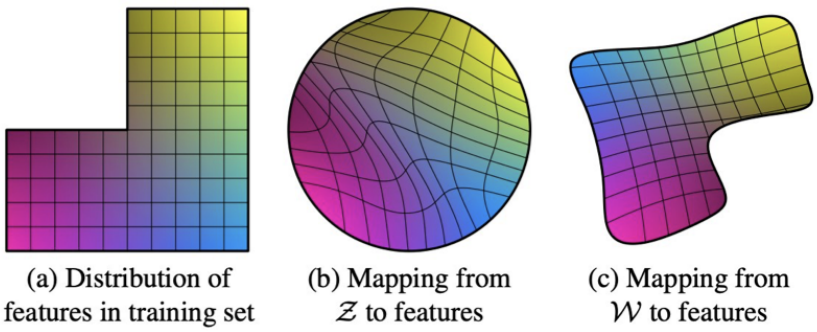

Figure 2.3: Mapping from $\mathbb{Z}$ and $\mathbb{W}$ latent spaces to features, extracted from [4].

Usually, a 512-dimensional vector $w \in \mathbb{W}$ is used 18 times as the style input to 18 layers of the StyleGAN2 [5] generator. This suggests that each of these 18 can be individually modified for fine-tuning of a generated image. This extends latent space into 18 copies of $\mathbb{W}$ ($d = 18 \times 512$) and is denoted by $\mathbb{W}+$. This larger space is able to provide a different latent code for each layer of the StyleGAN generator (e.g., 18 for a StyleGAN2 generator with a 1024×1024 output resolution).

Since the StyleGAN architecture is trained using $\mathbb{W}$ space, images sampled from $\mathbb{W}+$ do not necessarily have realistic perceptual quality. This can allow to generate entirely novel patterns that are still face-like. However, it may also lead to patterns with useless structure or level of quality. As the distribution of $\mathbb{W}$ cannot be explicitly modeled, keeping the latent code in within a range that corresponds to semantically and quality-wise useful patterns is a challenging task.

In a further step of StyleGAN processing, the latent code $w \in \mathbb{W}$ of an image is further transformed to $s \in \mathbb{S}$ vectors for each layer of the StyleGAN generators. The details of these transformations differ slightly between the versions, but a shared commonality is a mapping to a parameter vector of style parameters $\mathbb{S}$ that parametrizes a set of affine transformations (one for each layer) that either normalize the activity pattern in a layer (in the case of StyleGAN), or that directly

define a two-step scaling of the weights of a layer (mod/demod operations of StyleGAN2 [5] and StyleGAN3 [6]). While the activity normalization in StyleGAN requires a specification of two parameters (bias and scaling) for each feature map, StyleGAN2/3 get by with a mere scaling (single parameter) for the feature map scalings in the mapping layers. For the sake of brevity, we focus on the case of StyleGAN2 in the following discussion, which is very representative of the major ideas behind the style mapping.

At the style input of every layer of the StyleGAN2 generator is an independent single-layer perceptron denoted by A (affine transformation). This network maps $w \in \mathbb{W}$ vector into a new vector $s \in \mathbb{S}$ that provides for each of the layer's weight kernel a separate scalar scaling parameter. The size of the vector $s \in \mathbb{S}$ equals the number of channels in all layers of the StyleGAN2 generator. For example, in StyleGAN2 $\dim(\mathbb{W}) = 512$ and $\dim(\mathbb{S}) = 9088$. This modulated kernel is then applied to the layer $L-1$ of the StyleGAN2 generator to produce the activation of the channel in the layer L of StyleGAN2. In [7] Wu et al. proposed to name this latent space of coefficients s StyleSpace $\mathbb{S}$. Analysis from Wu et al. indicates, that the $\mathbb{S}$ space is more semantically disentangled than previous latent spaces of StyleGAN2.

Evolution

While vanilla GANs are able to generate images of reasonable quality, they suffer from limited controllability and unstable training. To overcome these problems, Karras et al. introduced a training strategy in which the neural network progressively grows more layers during training (see Figure 2.7), PGGAN[3]. Because StyleGAN is built on progressive growing GANs, we will first have a quick look through PGGAN architecture.

Like a general GAN, PGGAN [3] consists of a generator and a discriminator. However, unlike a general GAN, the PGGAN discriminator classifies images at different scales. During training, the discriminator receives inputs at varying resolutions and combines them to determine if an image is real or fake. Real data images undergo a progressive downsampling process to produce lower resolution images. Conversely, the generator produces images at these resolutions and feeds them into the discriminator.

As layers are added at increasing depths, the image resolution increases as well. Starting with low resolution images of 4×4 pixels and progressing up to a resolution of 1024×1024 pixels, the model first learns coarse structures and

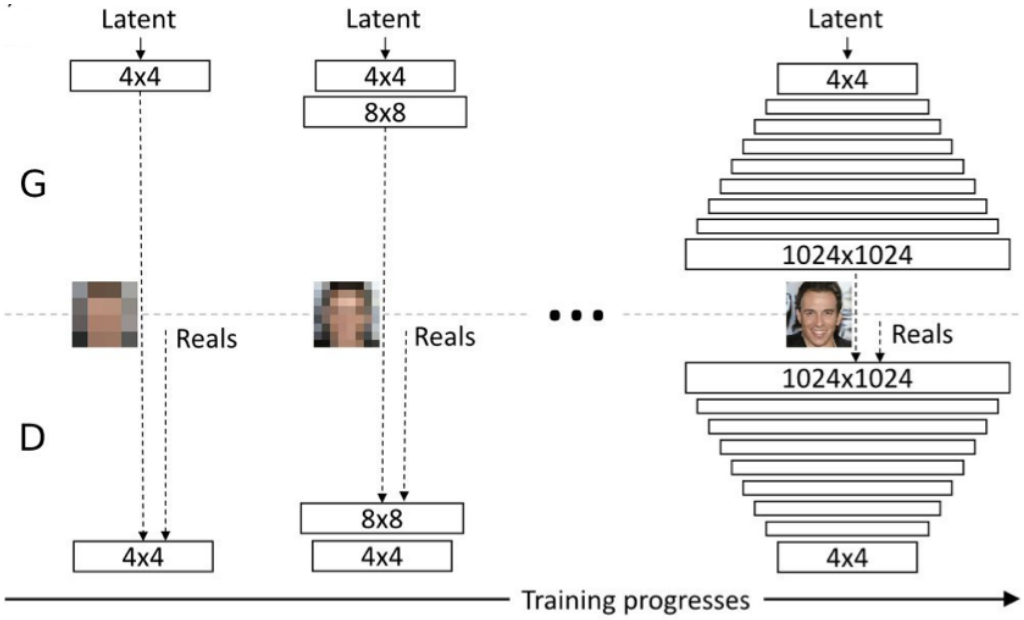

Figure 2.4: Progressive growing GANs, extracted from [3].

later fine details. This approach stabilizes training by splitting the task into simpler sub-tasks. Additionally, training time benefits from this approach, as most iterations are performed at lower resolutions within a smaller network, reducing computational demands.

StyleGAN adapts the progressive training strategy from the PGGAN. As a result, it offers a high degree of flexibility to mix image styles at different levels of its generator architecture.

StyleGAN no longer passes the random sample z directly into the generator's input layer. Instead, StyleGAN starts generating images from a learned constant ($4 \times 4 \times 512$), and the latent code $z \in \mathbb{Z}$ is fed into the network along a different route. First, $z \in \mathbb{Z}$ is mapped through a deep network of fully connected layers into an intermediate latent space $w \in \mathbb{W}$. The benefit of this $\mathbb{Z}$ to $\mathbb{W}$ transformation is that the intermediate space $\mathbb{W}$ does not need to follow the Gaussian distribution of the training data (however, $\mathbb{Z}$ does). Thus, latent space $\mathbb{W}$ can be disentangled which is a desirable property because it means that features in the generated images can be controlled independently of each other, and this is one of the most important features of StyleGAN, as it opens up great scope for working with images in latent space.

Subsequently, for each generator layer separately, $w \in \mathbb{W}$ is converted using affine transformation (fully connected layer without activation function) into a vector of style parameters that are used to shift and scale the activity pattern in the feature maps of the respective convolutional layer. This affine transformation of feature maps is called an adaptive instance normalization (AdaIN) [8].

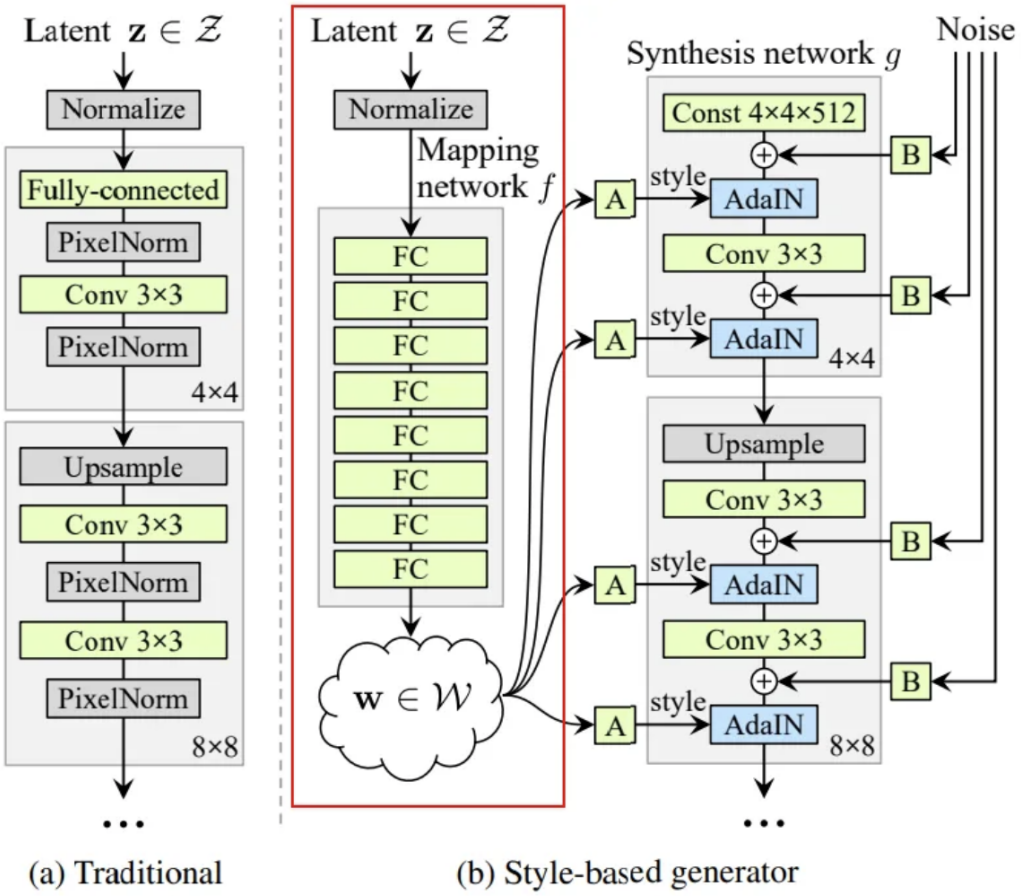

Figure 2.5: Style Learning Network, extracted from [4].

The affine transformation is implemented using two linear layers to create a style with scale $= y_s$ and bias $= y_b$. The AdaIN operation formula:

$$\text{AdaIN}(x_i, y) = y_{s,i} \left(\frac{x_i - \mu(x_i)}{\sigma(x_i)} \right) + y_{b,i},$$

Where x_i is the output feature map of the previous layer. The AdaIN first normalizes each channel x_i to zero mean and unit variance and then applies the scale y_s and y_b. This means the style y will control the statistic of the feature map for the

next convolutional layer. Where y_s is the standard deviation, and y_b is mean. The style decides which channels will have more contribution in the next convolution.

This layer-wise feeding of style parameters $w \in \mathbb{W}$ allows style-mixing by feeding code parameters w_A and w_B of two sources A and B respectively into different layer subsets of the StyleGAN generator. If a code is injected into early layers it affects rough features (e.g., shape of a face) while injection into later layers correspond to finer details (e.g., skin color), so latent codes enable modifications at different granularities.

Finally, to provide stochastic detail that would have to be learned otherwise, pixel-wise noise is injected after each convolution. The noise added to the feature map has zero mean and a small scale of variance (compared to the feature map). Therefore, the overall context of the image is preserved as the statistics of the feature map stay the same. This allows the network locally stochastic placing of fine structures, such as pores, hairs, or freckles. All these architectural innovations allow it to outperform the previous PGGAN.

Although the StyleGAN reaches state-of-the-art performance in generative tasks, it introduces a problem with artifacts in the generated images, such as blobs or droplets. In the StyleGAN2 paper, they spotted two causes for the artifacts introduced in StyleGAN1 and describe changes in architecture and training methods that eliminate them.

StyleGAN2's key change was a simplification and reorganization of the layer normalization, which in the original StyleGAN was recognized as destroying information in the relative activation strengths of the different feature maps within a layer. This was avoided by replacing the former AdaIN operations by a direct rescaling of the convolutional weights, again based on the style parameter output associated with $w \in \mathbb{W}$, followed by a normalization by the standard deviation over the scaled weights. Additionally, the noise and bias now became added outside the style block and removed from the initially learned constant input.

Another problem was that in the images created by StyleGAN some details like eye or teeth orientation seemed to be either stuck in place or jumping between positions instead of moving smoothly. This was attributed to the generator needing to produce output images at each resolution, which forces it to generate maximal frequency details. To overcome this problem, StyleGAN2 no longer trains models using progressive growing, but sums the output from different res-

olutions together and utilizes skip connections. The size of the model itself was also increased through the number of feature maps in the layers responsible for the highest resolutions.

2.3.2 GAN inversion

In response to the growing demand for interpretability and controllability in GANs, the need for GAN inversion has emerged as a pivotal technique. By mapping a given image back into the latent space of a pre-trained GAN model, GAN inversion facilitates a deeper understanding of the underlying features and structures in the latent space, enabling researchers to manipulate and interpret generated images with greater precision and insight. Since all the models we have worked with for this research project use a pre-trained StyleGAN, we will focus particularly on studying the inversion to latent space of StyleGAN.

There are three main approaches to accomplish GAN inversion:

1. **Optimization based methods**. Gradient based optimization methods directly optimize the latent vector using gradients from the loss between the real image and the generated one. Such methods can find a latent representation of the original image with a reasonable similarity. However, there still exist three main drawbacks:

    - Optimizing the procedure is a time consuming task that usually takes several minutes on a modern GPU.
    - The random initialization choice can significantly impact the final reconstruction image.
    - The found latent point in $\mathbb{W}$ or $\mathbb{W}^+$ spaces through inversion optimization steps is less stable while editing of the generated image than latent points obtained by sampling from $\mathbb{Z}$ space and generating latent points in $\mathbb{W}$ or $\mathbb{W}^+$ spaces through the mapping network (from $\mathbb{Z}$ to $\mathbb{W}$ space). The mapping network generates points in the distribution of the training dataset, while inversion optimization steps can move the latent point away from the distribution of the training dataset.

2. **Encoder based mapping methods**. Encoder based methods for finding the latent code train an encoder network over a large number of samples to directly map from the RGB image space into a latent space of StyleGAN. Once trained, the encoding can be done in the fraction of a second needed to

process through the CNN encoder. Latent points obtained from the encoder network are more suitable for editing by moving the point in the latent space, as the encoder network is trained to generate points inside the distribution of the training dataset of StyleGAN. However, training such an encoder is not trivial and the image generated from the obtained latent code may lose the identity of the original face.

Conventional image- and feature-level losses between the input image and the reconstructed image, may not be enough to guide the training of the encoder network. Wei et al. proposed the method of training the encoder in cooperation with an optimization based iterator. One more possibility for getting a better inversion quality is to compute the loss based on SSIM [33], Identity loss and LPIPS [28] to train an encoder that maps **RGB** into $\mathbb{W}$ space. An additional option is encoding into the $\mathbb{W}^+$ latent space that provides finer control over the generator, utilizing regularization methods while training of such **RGB** to $\mathbb{W}^+$ encoder.

3. **Fine tuning methods**. When an image of a face includes something outside the training distribution e.g., a tattoo, it is difficult to find a good inversion, as there is no such a point in the latent space of StyleGAN that allows for reconstruction of such details. In this case, a possible solution is to fine-tune the StyleGAN generator weights themselves, using the target image or a set of target images. This motivates a set of methods that operate directly on the generator weights to improve the inversion quality of a given image.

   Before starting such fine-tuning of the StyleGAN generator, the target image must be first inverted into StyleGAN's latent space to the best possible reconstruction match using the previously discussed approaches. Then the StyleGAN generator model can be fine-tuned using loss-functions applied to the reconstructed and target images.

   Fine-tuning the StyleGAN generator by gradient-based optimization for each new image requires a couple of minutes of computation. This makes such methods difficult to apply in practice. By analogy with encoder-based methods for predicting the inversion latent vector, we can, here too, seek an encoder whose output is used to modulate the weights of the StyleGAN generator. An example of such an approach is Hypernetworks [32]. The created Hypernetwork is trained to scale channel-weights of kernels of selected layers of the StyleGAN generator to match the target image and image generated by StyleGAN from the latent code.

In addition to preserving similarity to the original image, the central motiva-

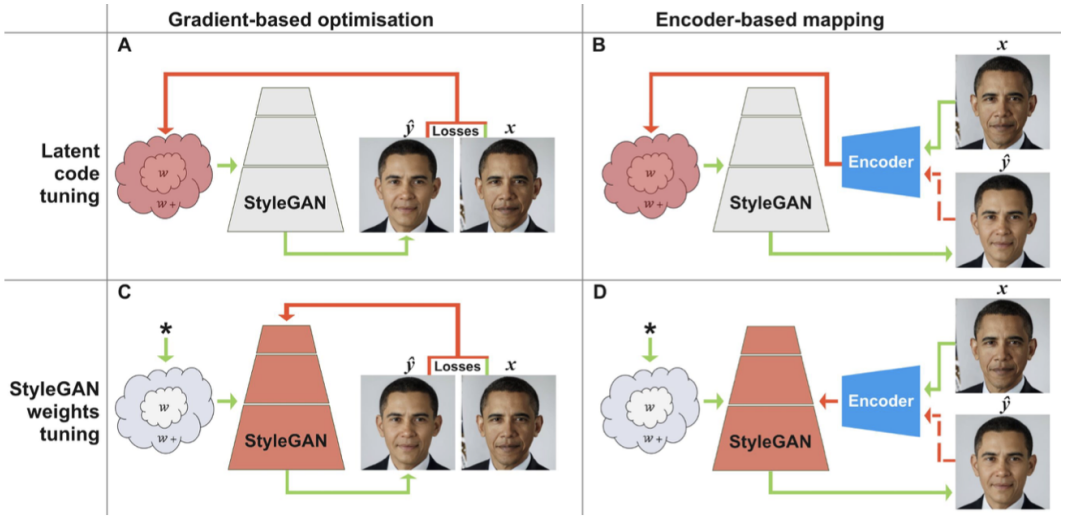

Figure 2.6: Inversion methods, extracted from [30].

tion of the inversion step is to facilitate further latent editing operations. There exist a variety of points in the latent space that result in similar images to the original one, some of these points are more suitable for latent editing than others. A successful encoding of a real image into a latent space should enable decent editability via the latent code.

2.4 Transformers

Another type of model we have to address to understand this work is Transformers, which have also revolutionized the field of ML. Their story begins with the quest to build models that could effectively handle sequential data. Before their introduction, recurrent neural networks (RNNs) [35] and their variant, long short-term memory (LSTM) [36] networks, were the standard for dealing with sequences in tasks like language translation, text summarization, and speech recognition.

RNNs and LSTMs process data sequentially, which makes them inherently slow due to their inability to parallelize the operations. They also struggle with long-range dependencies because the gradient signals have to backpropagate through all the previous steps in the sequence, leading to problems like gradient vanishing and exploding.

The Transformer model, introduced by Vaswani et al. in their landmark 2017

paper 'Attention Is All You Need' [34], addressed these issues with a novel architecture that was based entirely on attention mechanisms, dispensing with recurrence entirely. The key innovation of the Transformer is the self-attention mechanism, which allows the model to weigh the influence of different parts of the input data independently of their position in the sequence. This architecture enables the parallel processing of sequences and effectively captures long-range dependencies, making it significantly more efficient and powerful than its predecessors.

Transformers quickly became the go-to architecture for a wide range of NLP tasks, giving rise to models like BERT [37], GPT [38], and RoBERTa [39], which have set new benchmarks in the field. BERT (Bidirectional Encoder Representations from Transformers) leverages the Transformer's ability to process words in relation to all the other words in a sentence, rather than one-by-one in order. GPT (Generative Pretrained Transformer), on the other hand, uses Transformers to generate human-like text, enabling the creation of AI that can compose everything from poetry to prose.

Transformers are divided into two main parts: the Encoder and the Decoder. Both the Encoder and the Decoder are composed of blocks that are repeated a specific number of times, usually the same number for both. They both start with the same initial step: embedding the input and adding positional encoding, which provides information about the position of each token in the sequence to retain the order of the sequence. From this point on, their operations differ.

- **Encoder**. The goal of the encoder is to process the input sequence and identify relationships that enhance our understanding of it.

  The first layer of each encoder block is a self-attention layer, where the data is analyzed to find relationships or dependencies between the tokens of the sequence. The output of this layer is then added to the original input of the layer through a residual connection and normalized using layer normalization.

  The next step is a Feed-Forward Network (FFN) [40], which consists of two linear transformations with a ReLU activation in between. The result is then added to the input of the FFN layer through another residual connection and normalized again.

- **Decoder**. The goal of the decoder is to generate the output sequence, adding meaningful tokens one by one using the information gathered by the Encoder. Each pass of the Decoder adds one more token to the sequence. Thus, its input is the output of the previous pass, which is the sequence with one

additional token. It operates by checking what has been generated so far and predicting the next token.

The first layer of the Decoder is a self-attention layer, but with the restriction that it can only attend to previous positions in the output sequence. This is achieved by masking future positions, setting their attention values to zero.

The next layer is called the 'Encoder-Decoder Attention' layer. It computes attention using the outputs of the previous layer as queries and the encoder outputs as keys and values. Finally, we have an FFN block as before.

The output of the entire Decoder is a list with a score for each word in the vocabulary. These scores are passed through a softmax function to convert them into probabilities, and the most probable word is selected as the next token in the sequence.

The success in NLP led researchers to explore the application of Transformers in computer vision. In fact, it was the recently mentioned BERT model the one that inspired the application of the Transformer to this field.

The obvious question now is: an image is a single piece of information, how do we extract the different data tokens to analyze the attention? The answer is patch embedding. The image is split into different patches. These patches can be obtained using a convolution with a kernel and stride equal to the patch size. Then, they are flattened to a vector. This way the image patches have the same structure as tokens of a sequence.

2.4.1 Latent Transformers

Building upon the transformer architecture, researchers sought ways to enhance its efficiency and scalability, which gave rise to the concept of latent transformers.

Latent transformers are a variation of the original transformer models that incorporate latent variables to capture deeper and more abstract representations of the data. These latent variables act as an intermediary compressed representation of the input data, allowing the transformer to operate on a simplified version of the data, which can significantly reduce computational complexity. This is particularly useful in handling long sequences where the computation of self-attention can become prohibitively expensive.

The key innovation in latent transformers is the use of a latent space, which is a lower-dimensional space that the model's attention mechanism can efficiently

work with. Instead of calculating the attention over the entire input sequence, the model learns to map the input data to a latent space and then computes attention within this smaller, more manageable space. This not only speeds up the processing but also can lead to more generalizable representations, as the model is forced to capture the most salient features in the data.

Chapter 3

State of the Art

Facial attribute editing is a rapidly evolving field within computer vision and artificial intelligence, primarily driven by advancements in deep learning and generative models. Several methodologies have been developed to alter facial attributes while preserving the individual's identity and other essential features. This chapter provides a comprehensive overview of the latest advancements in facial attribute transformation techniques.

The journey of facial attribute editing began with the introduction of Generative Adversarial Networks in 2014 [27]. This adversarial process resulted in the generation of highly realistic images. Early GAN models laid the groundwork for subsequent advancements in facial attribute editing by demonstrating the potential of deep learning in generating synthetic images.

The first major leap in this field was marked by the development of ProGAN in 2018 [3]. ProGAN introduced a progressive training approach where the generator starts with low-resolution images and gradually increases the resolution by adding more layers. This method improved the stability and quality of the generated images, making it possible to create high-resolution faces with remarkable detail. ProGAN's success set the stage for more sophisticated GAN architectures focused on facial attribute manipulation.

Building on the foundation of ProGAN, NVIDIA introduced StyleGAN in 2019 [4], which brought significant improvements in the control over generated images. StyleGAN's innovative architecture enabled the unsupervised separation of high-level attributes, allowing for fine-grained edits of specific features like age, hair color, and expression. This was achieved by manipulating the latent space, where different attributes are disentangled. The introduction of StyleGAN and its subse-

quent versions (StyleGAN2 [5] and StyleGAN3 [6]) marked a significant milestone, offering unprecedented levels of realism and control in facial attribute editing.

Following the success of StyleGAN, researchers began to explore the latent space more systematically. In 2020, InterFaceGAN [9] was introduced, which focused on identifying linear directions in the latent space corresponding to specific facial attributes. This approach allowed for precise and continuous modifications of attributes by moving along these directions. InterFaceGAN's method of disentangling attributes provided a clearer understanding of how different features are represented in the latent space, leading to more controlled and accurate edit.

In 2021, the combination of GANs with language models brought a new dimension to facial attribute editing. StyleCLIP [11] integrated StyleGAN with CLIP (Contrastive Language–Image Pretraining) [12], enabling users to control facial edits through textual descriptions. This method allowed for intuitive and user-friendly interactions, where users could simply describe the desired changes, and the model would adjust the image accordingly. StyleCLIP demonstrated the potential of combining visual and textual data, opening up new possibilities for personalized and detailed facial edits.

The challenge of precisely manipulating attributes without unintended changes to other features led to the development of Semantic Disentangled GAN (SDGAN) in 2023 [24]. SDGAN introduced a semantic disentanglement generator that assigned facial representations to distinct attribute-specific modules. This approach ensured that edits were confined to relevant regions, avoiding undesired modifications in other parts of the image. SDGAN's ability to decouple different attributes and achieve high-quality style manipulation represented a significant advancement in the field.

Late 2023 saw the introduction of ChatEdit [19], which enabled interactive facial editing through multi-turn dialogues. This model featured a dialogue module for tracking user requests and generating responses, along with an image editing module for making the modifications. ChatEdit highlighted the potential for dynamic, user-centric facial editing applications, allowing real-time feedback and adjustments.

A Latent Transformer for Disentangled Face Editing in Images and Videos [13] built on the strengths of previous models. This method leverages transformer networks to achieve disentangled face editing in both images and videos. It integrates

seamlessly with prior advancements, enhancing the capability to manipulate attributes precisely and consistently across different media types. This approach signifies a leap towards more versatile and robust facial editing techniques. However, due to its sequential nature, this technique introduced potential issues related to the order of operations, thereby limiting both flexibility and efficiency.

Overall, most recent works have focused on incorporating the improvements achieved into different, more specific tasks such as video editing techniques or text-to-image techniques and few focused on solving the latent multi-attribute manipulation task. One of the latest innovations, a Multi-Attribute Latent Transformer [14], overcomes this issue by forcing the original image to go through different models, each one transforming one attribute at a time. In this process, the input of a given model is the output of the previous one.

Chapter 4

Methodology

With the introduction of StyleGANs, great progress has been made in face attribute edition, but how can we edit the semantical attributes of images using GANs? For example, change the age or gender of a person while preserving the general face shape and other attributes?

The main idea about editing an image using StyleGAN is that editing is achieved through some manipulation of its latent code, thereby moving the point that represents this latent code within one of the latent spaces of StyleGAN. At first we do not know how movement in the latent space will affect the generated image but there are methods to learn how to navigate the latent space to edit an image in a more controlled and semantically meaningful manner. Movement of the point in a wrong or a random direction will in the worst case lead away from the face distribution (thereby destroying the 'faceness' of the image), or lead to an undesired, simultaneous change of different attributes, most likely accompanied by a loss of identity of a person on the image.

In this chapter, we fist explain in more depth the four methods we have chosen to compare: InterFaceGAN [9], TediGAN [10], StyleCLIP [11] and Multi-Attribute Latent Transformer [14]. We also explain our proposal to determine whether the correlation between attributes in initial dataset can lead to changes of non targeted attributes in the final results.

4.1 State-of-the-Art Facial Attribute Editing Approaches

4.1.1 InterFaceGAN

Although GANs have made significant progress in face synthesis, until the publication of this paper [9], there lacked enough understanding of what GANs have learned in the latent representation to map a random code to a photo-realistic image. In 2020, Yujun Shen, Ceyuan Yang, Xiaoou Tang and Bolei Zhou proposed a framework called InterFaceGAN [9] to interpret the disentangled face representation learned by the state-of-the-art GAN models and study the properties of the facial semantics encoded in the latent space.

InterFaceGAN deeply explores the knowledge GANs learn in the latent representation and how we can reuse such knowledge to control the generation process. For example, given a latent code, how does GAN determine the attributes of the output face, e.g., an elder man or a young woman? How are different attributes organized in the latent space? Can we manipulate the attributes of the synthesized face as we want? How does the attribute manipulation affect the face identity? Can we apply a well-trained GAN model for real image editing?

A typical warping between the latent vector of two images demonstrates a smooth transition in facial features, but multiple features are entangled together, and precise control over a specific attribute is almost impossible. Despite the amazing quality of images GANs can generate, not much had been done on understanding and manipulating the latent space until this work. Previous work on semantic image editing using GANs involved retraining using carefully designed loss functions, additional attribute labels, or special architectures. Can't we use existing high-quality image generators to edit given images? This project suggests we must understand how individual facial features are encoded in the latent space both theoretically and empirically.

In order to understand how the model InterFaceGAN works, we must first consider two properties.

Definition 4.1. An **hyperplane** of a n-dimensional space is an (n-1)-dimensional subspace that can separate the original space. That is, a 2D plane can separate a 3D space and a 1D line can separate a 2D plane.

Property 1 Given $\mathbf{n} \in \mathbb{R}^d$ with $\mathbf{n} \neq \mathbf{0}$, the set $\{\mathbf{z} \in \mathbb{R}^d : \mathbf{n}^\mathsf{T}\mathbf{z} = 0\}$ defines a hyperplane in $\mathbb{R}^d$, and $\mathbf{n}$ is called the normal vector. All vectors $\mathbf{z} \in \mathbb{R}^d$ satisfying $\mathbf{n}^\mathsf{T}\mathbf{z} > 0$ locate from the same side of the hyperplane.

This property suggests that when we define the hyperplane with normal vector $\mathbf{n}^\mathsf{T}\mathbf{z} = 0$, vectors points with $\mathbf{n}^\mathsf{T}\mathbf{z} > 0$ are on a specific side of the hyperplane.

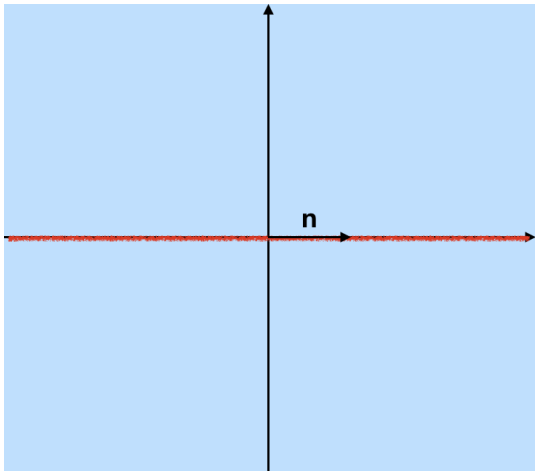

Figure 4.1: Visual representation of *Property 1* in 2-dimension space

Property 2 Given $\mathbf{n} \in \mathbb{R}^d$ with $\mathbf{n}^\mathsf{T} = 1$, which defines a hyperplane, and a multivariate random variable $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, we have

$$P(|\mathbf{n}^\mathsf{T}\mathbf{z}| \geq 2\alpha\sqrt{\frac{d}{d-2}}) \geq (1 - 3e^{-cd})(1 - 2e^{-\alpha^2/2})$$

for any $\alpha > 1$ and $d \geq 4$. Here, $P(\cdot)$ stands for probability and c is a fixed positive constant.

According to this second property, any latent $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ is likely to be close to a given hyperplane. Therefore, we can model a semantic into the linear subspace n.

With this in mind, we will try to understand the GAN latent space. As we mentioned in a previous chapter, the generator in GAN can be viewed as a function $g : Z \to X$ where Z is typically a Gaussian distribution and X is the image space. Consider a semantic space $S \subseteq \mathbb{R}^m$ with m semantics (attributes) and a semantic scoring function $f_S : X \to S$. Intuitively, the semantic score of a latent is measured as $f_S(g(z))$.

The InterFaceGAN paper suggests that we observe linear changes in the semantics contained in images when we linearly interpolate between two latent codes. Suppose there is only one semantic ($m = 1$) and consider a hyperplane with a normal vector $\mathbf{n}$ that separates the semantic.

Definition 4.2. We call the **distance** to a sample z as $d(n, z) = n^\mathsf{T} z$.

We expect the distance to be proportional to the semantic score, $f(g(\mathbf{z})) = \lambda d(\mathbf{n}, \mathbf{z})$.

We now consider the general case with $m \geq 1$ attributes. Consider $s = [s_1, \ldots, s_m]$ as the true semantic score of a generated image, $s \approx f_S(g(\mathbf{z})) = \Lambda N^\mathsf{T} \mathbf{z}$ where Λ is a constant vector and N is a matrix containing m separation boundaries. Using basic statistical rules, we can compute the mean and covariance statistics as

$$\mu_s = \mathbb{E}(AN^\mathsf{T}\mathbf{z}) = AN^\mathsf{T}\mathbb{E}(\mathbf{z}) = 0,$$
$$\Sigma_s = \mathbb{E}(AN^\mathsf{T}\mathbf{z}\mathbf{z}^\mathsf{T}NA) = AN^\mathsf{T}\mathbb{E}(\mathbf{z}\mathbf{z}^\mathsf{T})NA = AN^\mathsf{T}NA.$$

We can then conclude that s is actually sampled from a normal distribution $s \sim \mathcal{N}(0, \Sigma_s)$. Intuitively, for each vector in s to be completely disentangled, Σ_s must be a diagonal matrix. $(n_i)^\mathsf{T}(n_j)$ can also be used to measure the entanglement between the i-th and j-th attribute.

Suppose we found the decision boundary n of a certain attribute. We edit the original latent code z with $z_{\text{edit}} = z + \alpha n$. When multiple semantics are entangled, editing one attribute can affect other attributes. For example, moving a point in the n_1 direction will not only affect attribute 1 but will also change the distance of attribute 2. To counteract this, the paper applies projection to make $N^\mathsf{T}N$ a diagonal matrix where semantics are independent of each other.

With more than two attributes, we subtract the projection from the primal direction n_1 onto the plane constructed by all conditioned directions. This way, we can avoid modifying unwanted attributes.

But, how do we now apply the learned semantics to a given image for editing applications? The InterFaceGAN work describes two different approaches: GAN inversion and further training.

The first approach, GAN inversion, can be challenging because GANs do not capture the complete image distribution and there is often a loss of information.

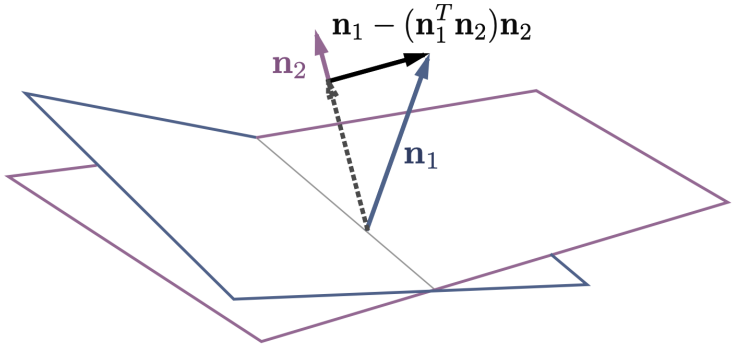

Figure 4.2: Visual representation of the conditional manipulation via subspace projection, extracted from [9].

Among the two approaches to GAN inversion discussed in the previous chapter, this proposal chooses LIA as a baseline for encoder-based inversion and searches the W+ space for optimization-based inversion, as suggested by, for instance, Image2StyleGAN paper. Optimization-based methods perform better but are much slower compared to encoder-based approaches.

Another approach is to train additional models on a synthetically generated paired dataset using a learned InterFaceGAN. The InterFaceGAN model can generate unlimited high-quality paired data. The idea is to train an image-to-image translation model such as pix2pixHD on the generated data. To implement the continuous manipulation, the translation model first learns an identical mapping network and a network fine-tuned to attribute translation. At inference, we interpolate between the model weights from the identical model to the fine-tuned model.

This last approach has a much faster inference speed, and the capability to fully preserve additional information since it removes the need for reconstruction. However, due to the inherent limitations of pix2pixHD, the model cannot learn large movements such as pose and smile. Thus, the applications are limited.

To sum up, InterFaceGAN understands the GAN latent space by assuming linear changes in the semantics. The observations conclude that each semantic is represented as a normal distribution based on the normal vector of the hyperplane. By considering a hyperplane that can linearly separate the latent space according to semantic attributes, we can model this hyperplane using a linear classifier. A linear SVM from latent to semantic labels can define the direction of semantic

attributes in the latent space. This is disentangled using subspace projection.

4.1.2 TediGAN

Concurrent to the growing interest in utilizing semantic editing directions in the latent space for editing real images, we are also witnessing exciting breakthroughs in multimodal learning. Given an image and a target description in natural language, the aim of text-guided image manipulation is to generate images that reflect the desired semantic changes while also preserving the details or attributes not mentioned in the text. Text-guided image editing is one of the most natural and intuitive ways of manipulating images, and, hence, it comes with no surprise that several recent works have focused on mapping target textual descriptions to editing directions in the latent space of StyleGAN [4].

TediGAN [10] enforces the text and image matching by mapping the images and the text to the same latent space and performs further optimization to preserve the identity of the subjects in the original image. For doing this, this paper proposes a GAN inversion technique that can map multi-modal information into a common latent space of a pretrained StyleGAN where the instance-level image-text alignment can be learned.

The inversion module aims at training an image encoder that can map a real face image to the latent space of a fixed pretrained StyleGAN model. The reason behind taking a trained StyleGAN instead of training one from scratch is that, in this way, TediGAN goes beyond the limitations of a paired text-image dataset. The StyleGAN is trained in an unsupervised setting and covers much higher quality and wider diversity, producing in this way satisfactory edited results with images in the wild. In order to facilitate subsequent alignment with text attributes, the goal for inversion is not only to reconstruct the input image by pixel values but also to acquire the inverted code that is semantically meaningful and interpretable.

In order to train an encoder for GAN inversion, TediGAN acts different from conventional methods in two main things: (a) the encoder is trained with real images rather than with synthesized images, making it more applicable to real applications; (b) the reconstruction is at the image space instead of latent space, which provides semantic knowledge and accurate supervision and allows integration of powerful image generation losses such as perceptual loss and LPIPS [28]. Hence, the training process can be formulated as

$$\min_{\Theta_v} \mathcal{L}_{E_v} = \|x - G(E_v(x))\|^2 + \lambda_1 \|F(x) - F(G(E_v(x)))\|^2 - \lambda_2 \mathbb{E}\left[D_v(G(E_v(x)))\right],$$

$$\min_{\Theta_v} \mathcal{L}_{D_v} = \mathbb{E}\left[D_v(G(E_v(x)))\right] - \mathbb{E}\left[D_v(x)\right] + \frac{\lambda_3}{2}\mathbb{E}\left[\|\nabla D_v(x)\|^2\right],$$

where Θ_v and Θ_v are learnable parameters, λ_1, λ_2, and λ_3 are the hyper-parameters, and $F(\cdot)$ denotes the VGG feature extraction model.

Once the inversion module is trained, given a real image, TediGAN maps it into the W space of StyleGAN. This paper proposes a visual-linguistic similarity module to project the image and text into a common embedding space. Given a real image and its descriptions, we encode them into the W space by using the previously trained image encoder and a text encoder. The obtained latent code is the concatenation of L different C-dimensional w vectors, one for each input layer of StyleGAN. The multimodal alignment can be trained with

$$\min_{\Theta_l} \mathcal{L}_l = \left\| \sum_{i=1}^{L} p_i (w_i^v - w_i^l) \right\|_2^2,$$

where Θ_l represents the parameters of the text encoder, subscript l means linguistic; $w^v, w^l \in \mathbb{W}^{L\times C}$ are the obtained image embedding and text embedding; $w^v = f(E_v(x))$ is the projected code of the image embedding z in the input latent space $\mathbb{Z}$ using a non-linear mapping network $f : \mathbb{Z} \to \mathbb{W}$; w^l shares a similar definition; w^v and w^l are with the same shape $L \times C$, meaning to have L layers and each with a C-dimensional latent code; and p_i is the weight of i-th layer in the latent code.

To face the challenge of identity preservation, this method implements an instance level optimization module to precisely manipulate the desired attributes consistent with the descriptions while faithfully reconstructing the unconcerned ones. It uses the inverted latent code z as the initialization, and the image encoder is included as a regularization to preserve the latent code within the semantic domain of the generator. To summarize, the objective function for optimization is

$$z^* = \arg\min_z \|x - G(z)\|_2^2 + \lambda_1 \|F(x) - F(G(z))\|_2^2 + \lambda_2 \|z - E_v(G(z))\|_2^2,$$

where x is the original image to manipulate, λ_1 and λ_2 are the loss weights corresponding to the perceptual loss and the encoder regularization term, respectively.

4.1.3 Style CLIP

Recently, more advanced techniques that integrate the power of natural language models with image generation have emerged. Contrastive Language-Image

Pretraining (CLIP) [12], created in 2020, provides an effective common embedding for images and text captions.

CLIP is a model trained to pair images with text. To do so, there are two independent encoders: one for the images and one for the text, that are trained to produce a vector in the same dimensional space. For every batch of images with their captions, a list of image vectors $I_1, \ldots, I_n$ and a list of text vectors $T_1, \ldots, T_n$ are produced. Then, a scalar product is made among all the image and text vectors in a batch, obtaining a matrix of size $n \times n$. This matrix contains on the diagonal the product corresponding to correct match of image and text vectors and outside - the wrong ones. Consequently, the objective of the model, a contrastive loss, is to maximize the values in the diagonal and minimize the ones outside.

StyleCLIP [11] uses the recent CLIP model in a loss function to train a mapping network that takes text descriptions of image edits and an image encoded in the latent space of a pre-trained StyleGAN generator and predicts an offset vector that transforms the input image according to the text description of the edit.

There are three StyleCLIP approaches. The first approach for leveraging CLIP is a simple latent optimization technique. This method regresses the input latent code in the StyleGAN's $W+$ space to the desired latent code by minimizing a loss computed in the CLIP space. This approach performs a dedicated optimization for each input pair, making it versatile, but also slowing it down as each manipulation requires several minutes.

The second approach uses an auxiliary mapping network to manipulate the image's desired attributes as described by the text prompt. The mapper consists of three fully connected networks that correspond to three different levels of details: coarse, medium, and fine. These networks have the same architecture as the StyleGAN mapper but with fewer layers, 4 instead of 8. Although the mapper infers manipulation steps based on the input image for a given text prompt, these steps have high cosine similarities over vastly different input images. This means that the direction of manipulation steps in the latent space for a text prompt is generally the same irrespective of the input latent code.

The third approach described in the StyleCLIP paper maps the text prompts into a global direction in StyleGAN's style space S. This approach enables fine-grained disentangled manipulations because style space is more disentangled than other latent spaces.

4.1.4 Single Latent Transformer

Another advanced model for face editing is the proposed in [13]. This method, similar to InterfaceGAN, consists of using a StyleGAN encoder-decoder mechanism to project the images into the StyleGAN latent space, transform such latent representation of the image, and decode it back into an actual image.

In this work, they train 40 different Latent Transformers (one for each target attribute) that modify the latent representation w of the image given a target value k and an intensity coefficient α. These Latent Transformers consist of 18 independent fully connected layers of 512 neurons. This corresponds to the projection of the latent space of StyleGAN. As we explained in the previous chapter, the regular latent space $\mathcal{W}$ consists of 512 encoding values, while its projection $\mathcal{W}^+$ is formed by 18 layers of 512 elements each. Each Latent Transformer independent layer maps to one of the projected latent space layers and they do not share any connection with the other independent layers. Given this, each layer has a 512 encoding vector chunk from the latent representation and its output is another 512 vector.

All individual 18 vectors of 512 elements each are concatenated to form the transformed latent representation $f(w)$. This output provides the direction of the attribute learned by the Latent Transformer. In order to control its intensity and direction (positive or negative transformation), it is necessary to multiply it by a coefficient value α. The result of such multiplication is added to the original latent space representation to achieve the final transformed latent encoding of the image $T_k(w)$.

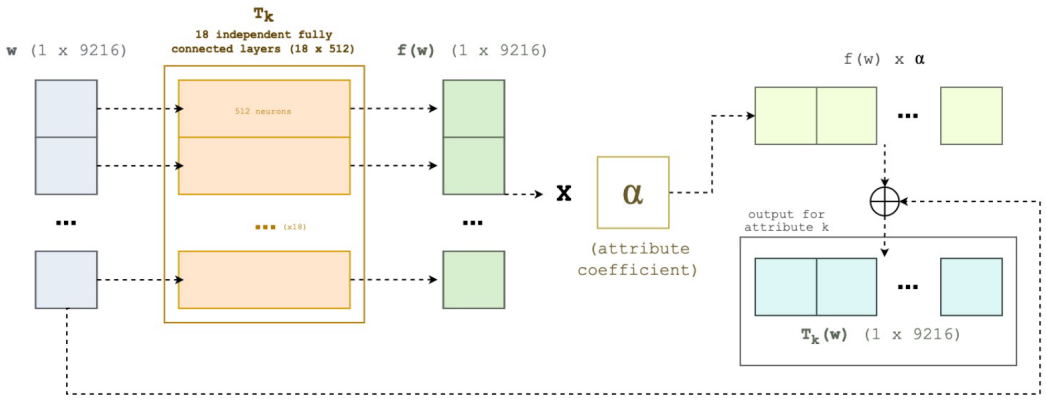

Figure 4.3: Architecture of an individual Latent Transformer in charge of learning attribute k, extracted from [14].

The question now is how to calculate the coefficient value α. During training, each sample's latent representation w is inputted into a Latent Classifier, which predicts the probabilities for all 40 attributes. The architecture of the Latent Classifier consists of 3 fully connected layers that use the ReLU as the activation function. The classifier C is fixed during the training of the Latent Transformer and is a key component of the architecture since it is in charge of predicting the presence of an attribute, and this has a direct impact on the training losses and evaluation metrics.

During training, only the attribute currently being learned is of interest. The objective is to invert the presence of the attribute in the image. If the predicted probability for the attribute exceeds 0.5, the target value, which is considered as the attribute coefficient α, is set to -1; otherwise, it is set to 1. By doing so, it is easier for the model to better learn which elements from the latent representation of the image are directly related to the attribute to be transformed. This procedure is repeated for all training samples and each individual attribute.

4.1.5 Latent Multi-Attribute Transformer

One of the latest contributions to facial attribute manipulation is an unified transformer capable of learning multi-attribute transformations. This approach builds on the previous method we have explained, the Single Latent Transformer. This proposal consists of unifying all the individual transformers into a single one capable of learning multi-attribute transformations.

The Latent Multi-Attribute Transformer [14] uses the same latent encoding and decoding approach by using Image2StyleGAN [31] and the same Latent Classifier to predict the probability of each attribute in a given image.

In this proposal, however, StyleGAN projects the image in an extended latent space $W+$ that consists of 18 layers of 512 encoding values each, compared to the single layer of 512 values from the original StyleGAN in the latent space W.

As all the attributes are being learning with just one transformer, the attribute coefficients encoding vector α is a $1 \times n$ sized vector of floating values, where n is the total number of attributes the transformer has learned. The encoding value of a given attribute can be any real number:

- A value of 0 would mean no change at all; the attribute remains as it originally is in the base image.

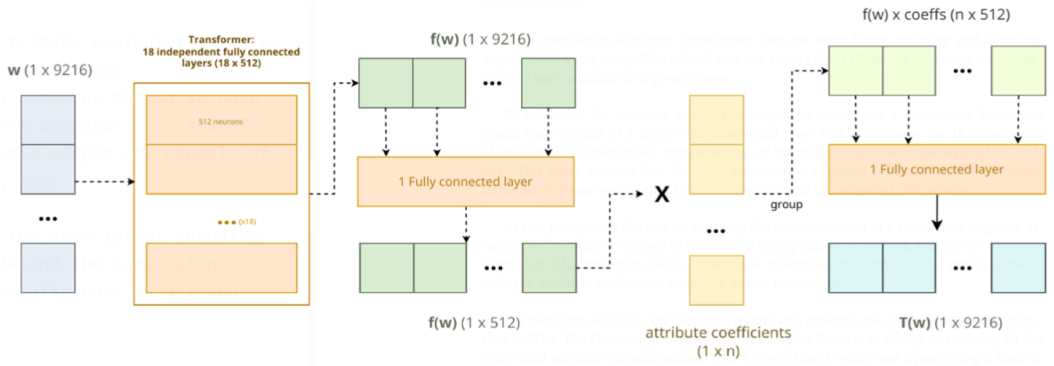

Figure 4.4: Architecture of the Multi-Attribute Latent Transformer, extracted from [14].

- When the value is > 0 it means positive transformation and it aims to increase the presence of the given attribute in the image. The higher the value, the higher the intensity of its presence.
- If the value is < 0, the transformation is negative and it aims to reduce or invert the presence of the attribute. Smaller, more negative values mean higher intensity of the inversion.

The Latent Transformer network consists of 18 independent fully connected layers, each of them is in charge of one of the 18 blocks from the projected latent representation in $W+$, w, that can be depicted as either a 18×512 or a 1×9216 vector. The intermediate result of the Transformer, $f(w)$ is the concatenation of the individual layers from the transformer back into a single vector of 1×9216 latent values.

To minimize the memory usage, this approach introduces a Dimension Reduction Block, that consists of a single fully connected layer that compresses the 18 dimensions of the extended latent space representation of the image ($f(w)_{1\times9216}$), into one single layer of 512 values ($f(w)_{1\times512}$). This block ensures that the main information required for transforming the facial attributes is preserved, without introducing artifacts or undesired side effects.

At this point, with the aim of applying the transformation in a positive or negative direction with different degrees of intensity, a multiplication is needed, that is, the Attribute Coefficient Multiplication Block. This block multiplies the output of the previous block, $f(w)_{1\times512}$, with the attribute coefficients encoding vector α transposed.

The next step is to take the resulting matrix and generate the final output with dimension 1×9216. The Dimension Upscaling and Merging Block is in charge of unifying all the individual attribute transformations into a single latent vector and upsampling it back to match the original projected latent space representation. This block learns to differentiate between attributes and how to combine all the transformations into a single output.

Then, this approach combines the original latent representation of the image w with the final transformation output $T(w)$ using a simple addition:

$$T'(w) = w + T(w)$$

Finally, just like in the previous method, the results can be visualized by decoding $T'(w)$ using the Image2StyleGAN decoder. It generates the resulting image with all the attribute transformations applied.

4.2 Attribute Correlation

To evaluate the attribute correlation in face editing models, we selected two models for comparison: the best-performing model based on various performance metrics (detailed in the next chapter), and its simplified version, which serves as the baseline. Both models were trained using the same dataset to ensure consistency in comparison. Specific training parameters and configurations were kept identical across both models to ensure a fair comparison.

After training, the focus shifts to analyzing the relationships between facial attributes manipulated by the models. This analysis aims to understand how the presence or alteration of one attribute affects the probability of other attributes in the edited images.

For each attribute (e.g., smiling, male, eyeglasses), we first calculate the baseline probability of a given image having this attribute. This probability represents how likely the attribute is present in the images generated or edited by the model without any intentional attribute manipulation.

To analyze the correlation, we systematically modify each non-target attribute one by one while keeping the target attribute constant. For example, if evaluating the "smiling" attribute, we would alter other attributes such as "male" and "eyeglasses" individually, observing the changes in the probability of the image

exhibiting a smile.

After manipulating each non-target attribute, we recalculate the probability of the target attribute in the modified images. This process allows us to measure any changes in the probability of the target attribute resulting from modifications to other attributes. For instance, if changing the "eyeglasses" attribute significantly affects the probability of the image having a smile, this correlation is noted and quantified.

The observed correlations between attributes in the modified images are then compared with the initial correlations present in the training dataset. This comparison helps to determine if the models learned or altered the attribute relationships in a way that deviates from the original dataset patterns.

Finally, we compare the results of the two evaluated models to identify differences in how attribute correlations are handled. This analysis provides insight into the best performing models' behavior and effectiveness in isolating or unintentionally linking facial features.

Chapter 5

Experiments and Results

In this chapter, we present the results of the experiments that we conducted to compare different state-of-the-art approaches. For doing so, we first analyze the used datasets. We then describe the statistical and performance metrics that are employed to quantitatively evaluate the quality of the models. We also describe the evaluation pipeline and general settings. The chapter concludes with the results of all the performed experiments.

5.1 Datasets

First of all, we describe the datasets we use for training and testing the models explained in the previous chapter.

5.1.1 Training set

The training dataset plays a crucial role in the development and learning process of the models. It provides the necessary samples to train the model on a diverse range of facial attributes. For this work, we use the CelebA-HQ [3] dataset.

CelebA-HQ is a high-quality version of CelebA that consists of 30.000 images at 1024^2 resolution, each annotated by 40 facial attributes. A notable property of the dataset is the distribution of binary attributes. Attributes have a value of either 1 or -1, denoting the presence or absence of said attribute. As we can see in Figure 4.1, this distribution in the CelebA-HQ dataset is not always equal. Most of the time, the absence of an attribute is more common than its presence. In fact, only 'No Beard', 'Young', 'Attractive' and 'Wearing Lipstick' have positive majority classes, that is, at least 50% of the samples are positive. This imbalance

is most likely intrinsic, thus a result of the nature of the data space, as the 40 attributes are most likely not distributed evenly in the human population.

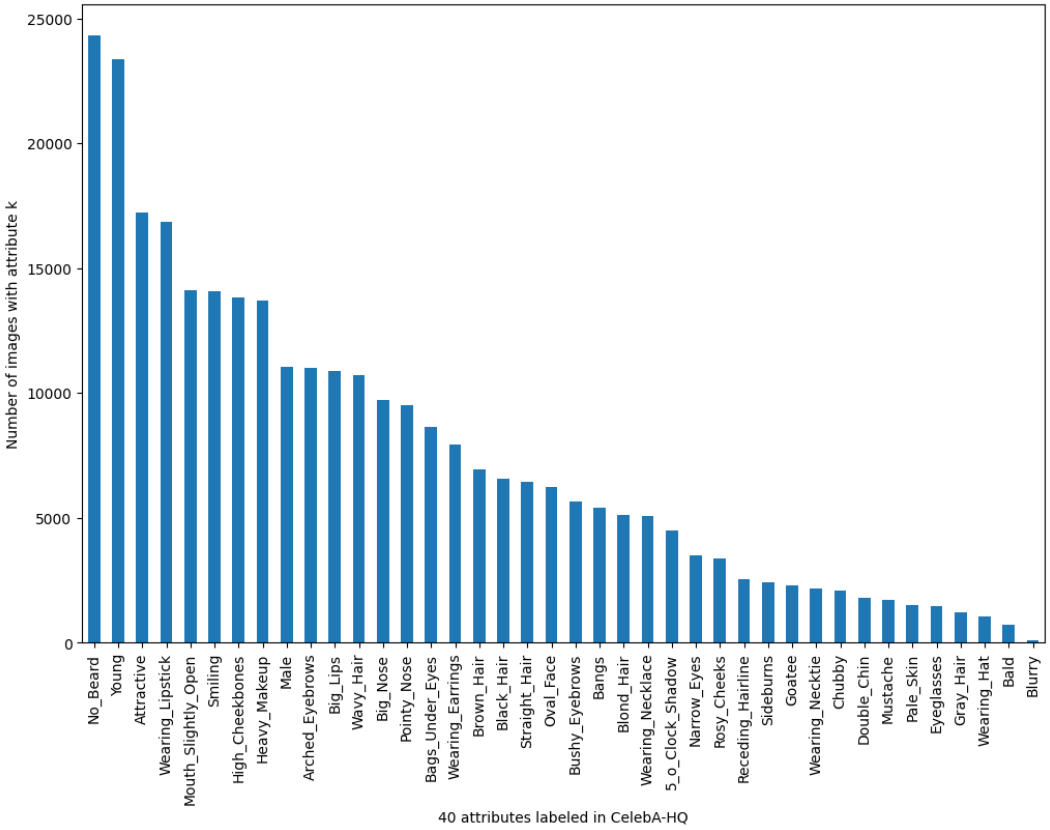

Figure 5.1: Distribution of each attribute on the CelebA-HQ dataset.

However, some of the imbalance may stem from the sampling of the data set, also known as extrinsic imbalance [17]. A possible example is the attribute 'Attractive'. There may be a sample bias for this attribute, as the subjects in the CelebA-HQ dataset are celebrities, and, therefore, represent a sample of individuals that are commonly more attractive than the general population.

For our research, we have followed [14] and decided to work only with 20 of the 40 attributes due to GPU constraints, Table 5.1. As we can see in Figure 5.2, the most extreme case of imbalance of an attribute in the CelebA-HQ dataset out of the 20 attributes we have selected is the attribute 'Bald', where only 2.37% of all samples are positive.

Bald	Male	Smiling
Big_Lips	Young	No_Beard
Big_Nose	Pale_Skin	Narrow_Eyes
Bangs	Attractive	Wearing_Lipstick
Eyeglasses	High_Cheekbones	Gray_Hair
Blond_Hair	Bushy_Eyebrows	Chubby
Wavy_Hair	Arched_Eyebrows	

Table 5.1: Selected 20 attributes

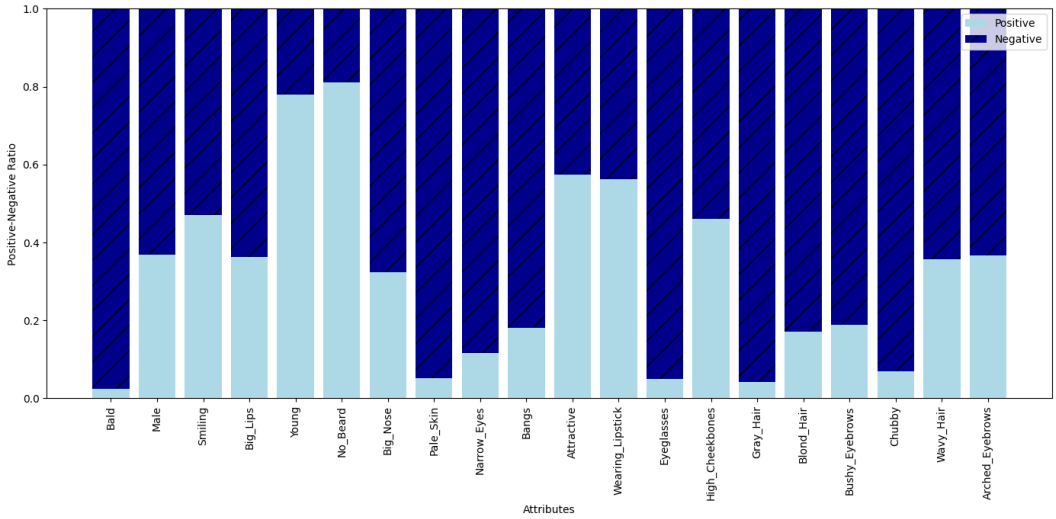

Figure 5.2: Presence and absence of the 20 selected attributes in the CelebA-HQ dataset.

Attribute Correlation

The amount of annotated attributes and the total number of samples makes CelebA-HQ an ideal dataset to analyze how the correlation between attributes in the training dataset can introduce a bias in face editing models.

Previous works, such as [21] and [20], analyze the correlation between attributes by dividing attributes into location-based groups. For this work, we want to take it a step further and study statistically the CelebA-HQ dataset.

In Figure 5.3, we can see that, for example, male faces do not correlate well with the 'Wearing Lipstick' attribute. This feature correlates almost exclusively with female faces. Moreover, a large fraction of the male faces have a beard, while this is not the case for women. Faces labelled as 'Attractive' mostly belong to young

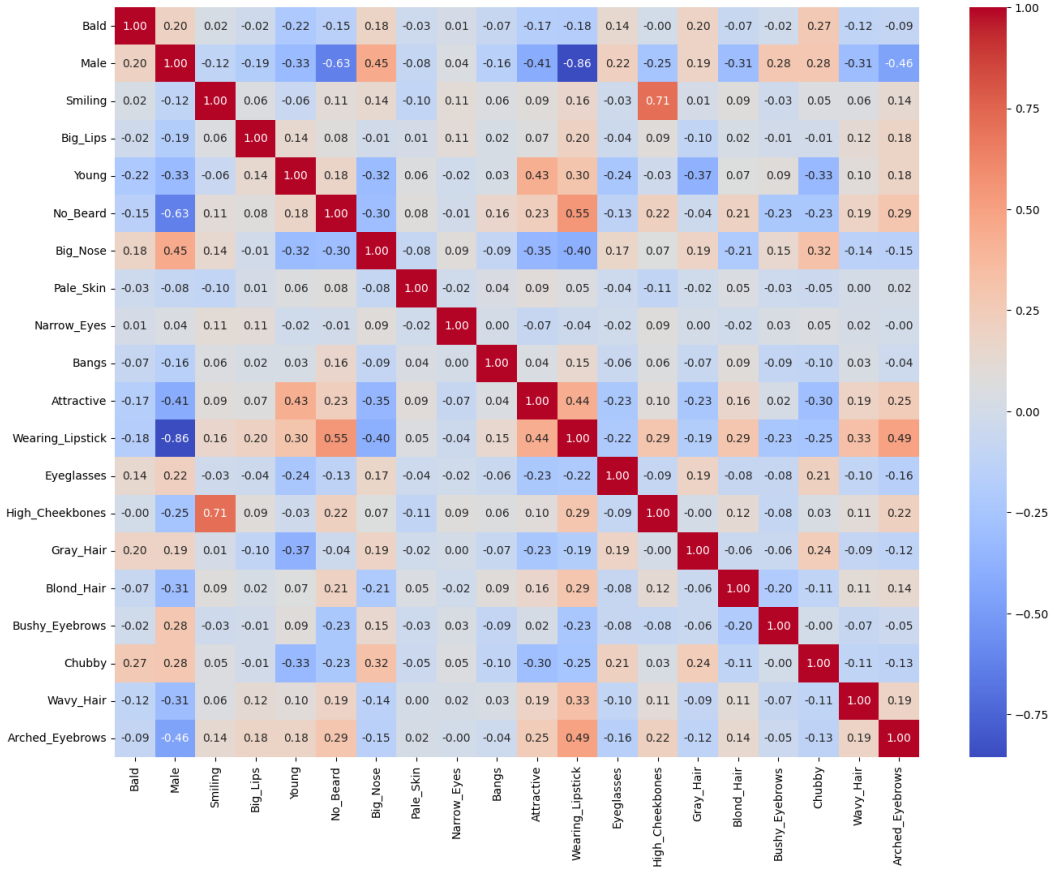

Figure 5.3: Correlation between the selected 20 attributes of the CelebA-HQ dataset.

and female faces wearing accessories and makeup. Let us delve more deeply into these correlations.

A key challenge for analyzing the CelebA-HQ dataset is the multidimensional nature of the data; the large number of samples combined with the fact that each sample has multiple attributes, makes visualizing and consequently analyzing how attributes correlate challenging.

To explore this correlation, we employ principal component analysis (PCA), which helps us reduce the dimensionality of the data while preserving its essential characteristics. PCA is a dimension reduction technique that utilizes the eigenvectors of the correlation matrix to calculate the principal components [22].

To accurately comprehend the utility of PCA in this scenario, it is essential to

first elucidate the mathematical principles underlying this statistical concept. First of all, it is important to standardize the data, which involves centering the data by subtracting the mean and scaling it by dividing by the standard deviation. Standardization ensures that all features have equal importance in the analysis.

After the standardization of the data, we have to compute the covariance matrix. The covariance between two variables measures how they change together. The covariance matrix for a dataset with n features is an $n \times n$ matrix that summarizes the relationships between all pairs of features. In our context, we have a 30.000×20 matrix.

The next step is to compute the eigenvalues and eigenvectors of the covariance matrix. These eigenvalues represent the amount of variance explained by each eigenvector (principal component). Eigenvalues and eigenvectors are mathematical concepts related to linear transformations and matrices. In the context of PCA, they play a central role in identifying the principal components.

- **Eigenvalue**.An eigenvalue represents a scalar that indicates how much variance is explained by the corresponding eigenvector. In PCA, eigenvalues quantify the importance of each principal component. They are always non-negative, and the eigenvalue corresponding to a principal component measures the proportion of the total variance in the data explained by that component.

- **Eigenvector**. An eigenvector is a vector associated with an eigenvalue. In PCA, eigenvectors represent the directions along which the data varies the most. Each eigenvector points in a specific direction in the feature space and corresponds to a principal component. Eigenvectors are typically normalized, meaning their length is 1.

We now choose a subset of the top eigenvectors to form a transformation matrix. After computing the eigenvalues and eigenvectors of the covariance matrix, they are sorted in descending order based on the magnitude of their eigenvalues. The principal components are then selected from the top eigenvectors. The first principal component corresponds to the eigenvector with the largest eigenvalue (highest variance), the second principal component corresponds to the eigenvector with the second-largest eigenvalue (second highest variance), and so on. These principal components are orthogonal, meaning they are uncorrelated. This matrix is used to project the original data into a lower-dimensional space, resulting in the

reduced dataset.

For the concrete implementation of this PCA we have used the scikit-learn module.

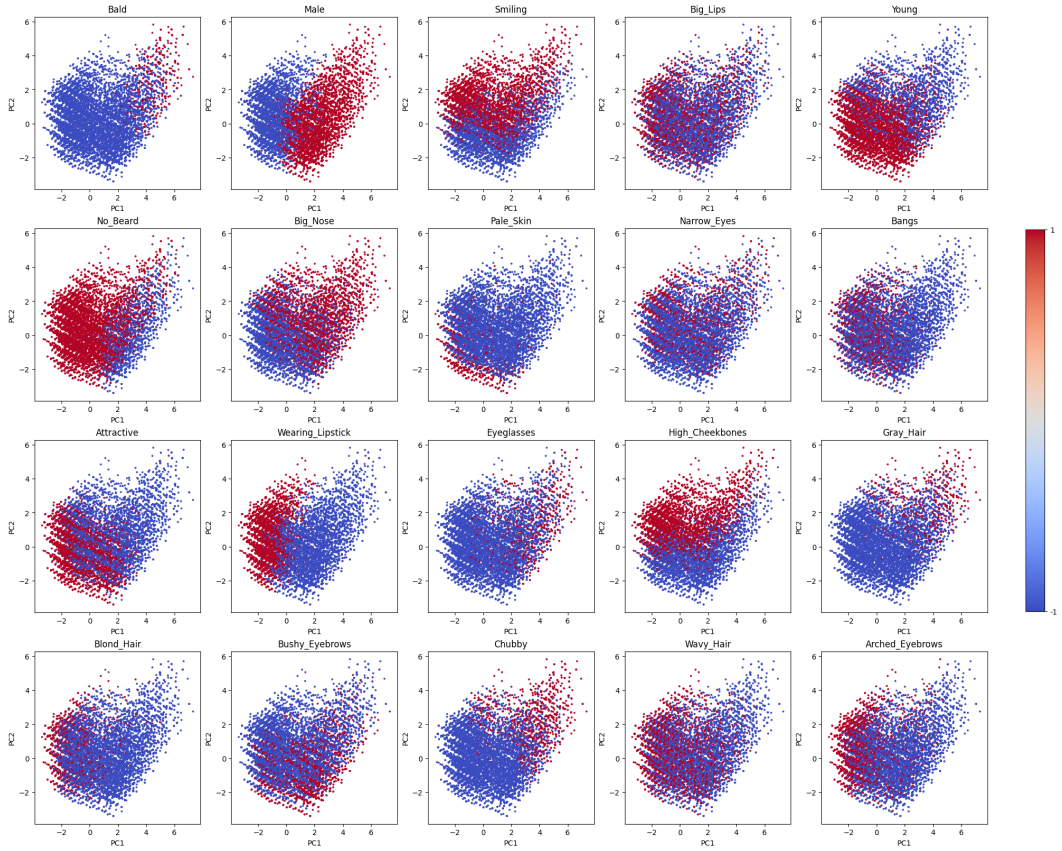

Figure 5.4: PCA of the 20 selected attributes.

The main conclusions we extract from Figure 5.4 are the following:

- In the 'Bald', 'Male', 'No Beard' and 'Wearing Lipstick' plots there is a strong horizontal separation between the two groups (red and blue), suggesting that the principal attributes used in these plots capture a significant difference between the bald and no bald, male and female, no beard and beard and wearing lipstick and not wearing lipstick groups, respectively.
- For the 'Smiling' and 'High Cheekbones' plot, although there is a tendency for the red points to be above the blue points, the overlap in the plots individually only indicate that the features used are not as effective in distinguishing

between smiling and non-smiling and high cheekbones and non-high cheekbones categories. However, the similarity between both plots indicate that both attributes are highly correlated. We can confirm this by checking their correlation in the Figure 5.3.

- We can see a similar dynamic between 'Attractive' and 'Young'. These two attributes are also highly correlated attributes in the dataset.
- An opposite effect can be observed between 'Big Nose' and 'Pale Skin'. The plots showcase a tendency in the samples of the smaller the nose the palest the skin.
- The concentration of blue points in the 'Chubby' plot may indicate a homogeneous group with similar characteristics for chubby faces, while the dispersion of red points may suggest greater variability within thin faces.

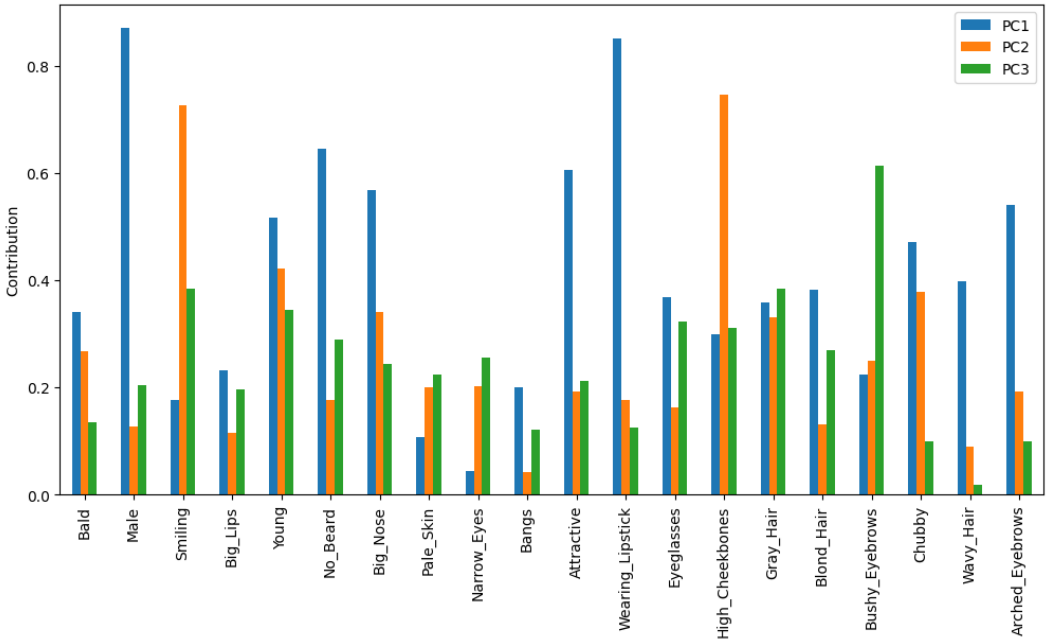

Figure 5.5: Attribute Contributions to Principal Components.

At this point, it is obvious that there is a strong correlation between some of the attributes. But, what is the root cause of this correlation? Is this correlation just a representation of society? It might be one of the reasons, but definitely not the only one.

An analysis made in [23] about the incorrectness of the labelling in the CelebA dataset estimate that 6.78% of images have at least one contradictory label. This means that, for instance, there are images labeled as 'Straight Hair' and 'Wavy Hair' at the same time. We have avoided introducing this incorrectness in our models by avoiding contradictory features in the selection of the 20 attributes.

On the other hand, most attributes are subjective and have been manually labelled. As discussed by previous work, on average attributes have a gender skew of 80.0% [25]. For example, 27.9% of images labeled with 'Male' are also labeled with 'Attractive', whereas 67.9% of images not labeled with 'Male' are not. It is difficult to tell to what extent this is a result of bias in labeling rather than bias in data selection, but there are some cases were correlation is clearly indicative of bad labeling. The clearest example is high cheekbones, which has a correlation of 0.68 with smiling. 85.6% of images labeled high cheekbones are also labeled smiling. This is likely because cheekbones appear higher while smiling, particularly when the smile is wide. Therefore, it is highly unlikely that high cheekbones provides an accurate label of cheekbone height irrespective of expression. Additionally, some gender correlations are too strong to be explained by data selection. We find that women are 3.1 times more likely to be labeled with 'Pointy nose', whereas men are 2.9 times more likely to be labeled with 'Big nose'. This is despite the fact that the probability of a random male nose being larger than a random female nose in terms of segment size is just 54.3%, indicating gender bias substantially influences labeling.

We have analyzed the correlation in human labeled attributes in the CelebA-HQ dataset. Later, we will apply the gained insights to address the effects of this attribute correlation when training the face editing models with it. Moreover, we will conclude in which cases the bias generated by this correlations is positive and in which is not.

5.1.2 Testing set

The test dataset is used to evaluate the performance and generalization capabilities of the trained models in different scenarios. It consists of a subset of 1k images without labels from the FFHQ [4] dataset.

FFHQ is a dataset of 70.000 high-quality PNG images at 1024×1024 resolution, featuring diverse human faces with variation in age, ethnicity, and accessories such as eyeglasses and hats. Originally designed for GAN benchmarking, the images were obtained from Flickr and were automatically aligned and cropped using dlib

[16], inheriting the website's biases.

5.2 Metric Evaluation

Metric evaluation plays a pivotal role in the development and enhancement of facial editing systems. These metrics serve as quantitative and qualitative measures that allow us to understand the performance of different approaches.

5.2.1 Statistical Metrics

Key points for evaluating facial editing models include:

- **Target attribute modification** is the model's ability to accurately and effectively alter the specified attribute while achieving a natural and realistic appearance. This metric is essential for assessing the functionality and practicality of the face editing model. The target modification should be noticeable but not exaggerated, and it should blend seamlessly with the rest of the face. For measuring this, we compute:

  - **Attribute Change Ratio (CR).** It calculates the proportion of target attributes that are effectively altered. An attribute is deemed modified if its classification probability exceeds 0.5 when the intention is to enhance it, or falls below 0.5 when the intention is to reduce it. The final score is the average of all successfully altered attributes when there are multiple attributes involved. A score nearer to 1 signifies better performance. The CR value is computed using the following formula:

$$CR = \frac{\sum_{i=0}^{n_t} \left(|C(w)_i - t_i| < 0.5\right)}{\#t}$$

    where:

    * $C(x)_i$: Classification probability for attribute i.
    * t_i: Target value of attribute i, either 1 or 0.
    * $\#t$: Total number of target attributes.

    For example, consider a scenario where there are 5 target attributes with the following classification probabilities and target values:

    To calculate the CR value, we first determine whether each attribute is successfully modified:

    * Big Nose: $|0.7 - 1| = 0.3$ (successfully modified, < 0.5)
    * Smiling: $|0.2 - 1| = 0.8$ (not successfully modified, > 0.5)

Attribute (i)	**Classification Probability ($C(w)_i$)**	**Target Value (t_i)**
Big Nose	0.7	1
Smiling	0.2	1
Wearing Lipstick	0.8	1
Male	0.4	0
Wavy Hair	0.9	1

Table 5.2: Example of Classification Probabilities and Target Values

  * Wearing Lipstick: $|0.8 - 1| = 0.2$ (successfully modified, < 0.5)
  * Male: $|0.4 - 0| = 0.4$ (successfully modified, < 0.5)
  * Wavy Hair: $|0.9 - 1| = 0.1$ (successfully modified, < 0.5)

  Therefore, the CR value is:

$$CR = \frac{4}{5} = 0.8$$

  In this example, the CR value is 0.8, indicating that 80% of the attributes were effectively altered according to their respective target values but that 20% weren't.

- **Non-target attribute preservation** refers to the model's ability to retain all facial features that are not intended to be altered. This is vital for maintaining the identity and overall integrity of the face. For instance, if we choose to change the hair color of a person with a model, it should ensure that other attributes such as facial structure, eye color, and skin tone remain unchanged. Crucial factors of non-target attribute preservation include:

  – **Identity Preservation (IP).** The edited image should still be recognizable as the same individual. To measure this, IP calculates the Mean Squared Error (MSE) between the original latent representation w of the image and its transformed version $T(w)$. In this context, a lower value indicates better performance.

$$IP = MSE(w, T(w))$$

  – **Attribute Preservation (AP).** Attributes not related to the intended edit should remain consistent. To measure this, AP calculates the proportion of non-target attributes that remain unchanged. Unlike the Attribute Change Ratio, which focuses on modified attributes, this metric specifically evaluates the stability of non-target attributes. It calculates the

ratio of attributes that remain unaltered, with the following formula:

$$AP = \frac{\sum_{i=0}^{n_t} (|C(w)_i - C(T(w))_i| < 0.25)}{\#nt}$$

where nt is the list of non-target attributes.

For example, consider a scenario where there are 5 non-target attributes with the following classification probabilities before and after transformation:

Attribute (*i*)	**Original Probability ($C(w)_i$)**	**Transformed Probability ($C(T(w))_i$)**
No Beard	0.7	0.65
Bald	0.2	0.3
Attractive	0.8	0.75

Table 5.3: Example of Classification Probabilities Before and After Transformation

To calculate the AP value, we first determine whether each non-target attribute remains unchanged (i.e., if the absolute difference is less than 0.25):

* No Beard: $|0.7 - 0.65| = 0.05$ (unchanged, < 0.25)
* Bald: $|0.2 - 0.3| = 0.1$ (unchanged, < 0.25)
* Attractive: $|0.8 - 0.75| = 0.05$ (unchanged, < 0.25)

All 3 attributes are considered unchanged, so the AP value is:

$$AP = \frac{5}{5} = 1$$

In this example, the AP value is 1, indicating perfect stability as all non-target attributes remained effectively unchanged.

The ideal goal is to reach the right balance between CR, IP and AP.

5.2.2 Performance Metrics

For evaluating and comparing the performance of the selected models, we take into account three different metrics:

- **Training Time.** Training time measures how long a model takes to learn from a dataset. It varies based on dataset size, model architecture, hyperparameters, and computational power. Efficient training time is crucial for quick experimentation and model refinement. Long training times can delay projects and reduce productivity.

- **Inference Time.** Speed is a critical metric that measures how quickly a model can process inputs and generate outputs. Factors affecting speed include model complexity, the efficiency of the underlying algorithms, and the hardware used for inference.

- **GPU Memory.** GPU memory usage indicates the amount of GPU memory required by the model during inference. High GPU memory usage can be a bottleneck, especially in environments where GPU resources are shared or limited.

5.3 Evaluation Pipeline

In all of our experiments, we use the public implementations provided by the authors, adapting them to be able to fairly compare them.

5.3.1 Statistical Metrics

We compare InterFaceGAN, TediGAN, StyleCLIP and the Multi-Attribute Latent Transformer. A comparison between the latter and the Single Latent Transformer has already been made in [14].

The evaluation pipeline consists of iterating over the test set and for each individual sample applying the corresponding transformation. For each sample, we randomly define a fixed transformation that consists of a subset of attributes to be modified. Each transformation can modify a different number of attributes. Finally, we use the result to compute the three main metrics: Attribute Change Ratio, Identity Preservation Score, and Attribute Preservation Score.

The evaluation metrics were implemented in a global manner so that they could be used by all the models, yet some modification had to be made in order to have an unified approach and be able to perform a fair comparison.

TediGAN and StyleCLIP need a text input in order to transform an image and give the desired output. In order to the calculate the desired metrics, we have established such text input to be 'add' + attribute. As for the InterfaceGAN and the Multi-Attribute Latent Transformer models, the intensity with which the facial transformations are made in the evaluation is +1.

5.3.2 Performance Metrics

We analyze and compare the performance of InterFaceGAN, TediGAN, StyleCLIP and the Multi-Attribute Latent Transformer. A performance comparison between the Single and Multi-Attribute Latent Transformers have already been made in [14].

To perform a fair comparison of all the approaches, we trained all the models with the same setting configuration.

Computer characteristics

We have done all the experiments with a computer with the characteristics in Table 4.4.

Component	Specification
Processor	Intel Core i9 12900KF
RAM	64 GB DDR4 3200MHz
GPU	NVIDIA RTX 4060 8 GB
Storage	2 TB SSD (NVME) + 2 TB HDD
Operating System	Windows 11 Home

Table 5.4: Specifications of the Computer Used for Model Training

Training details

For comparing the performance of all the five face editing models, we do not train the text encoders needed for TediGAN and StyleCLIP and take pre-trained encoders instead. Additionally, the used classifiers are also pre-trained prior to the training of the Latent Transformers for the Single Latent Transformer and the Latent Multi-Attribute Transformer. Finally, we take a pre-trained StyleGAN for all the models.

On the other hand, it is important to note that, when training the models, we use a batch size of 1.

5.4 Results

5.4.1 Statistical Comparison

In this section, we present and discuss the results of the experiments that we conducted to compare the different face editing approaches. We evaluated each model based on three criteria: Identity Preservation (IP), Attribute Preservation (AP), and Change Ratio (CR). Lower values for IP indicate better performance while high values are better for AP and CR. Both AP and CR values range between 0 and 1. However, IP can be any non-negative number.

Model	IP	AP	CR
InterfaceGAN	0.49	0.68	0.70
TediGAN	0.54	0.56	0.59
Multi-Attribute Latent Transformer	0.38	0.66	0.76
StyleCLIP	0.47	0.72	0.72

Table 5.5: Statistical Comparison of Facial Attribute Editing Models

The results in Table 5.5 indicate the performance of each model across the three metrics.

- **InterfaceGAN** shows a balanced performance with moderate IP, AP, and CR values, indicating a good overall ability to edit faces while maintaining identity and attribute accuracy.
- **TediGAN** has the highest IP value, indicating it does not perform well in preserving the original identity. It also has lower AP and CR scores, suggesting it might struggle with maintaining and changing attributes as effectively.
- **Multi-Attribute Latent Transformer** achieves the best CR score, indicating it is very effective at making desired changes, and has the lowest IP value, suggesting it preserves the original identity well. Its AP score is also high, showing good attribute preservation capabilities.
- **StyleCLIP** exhibits the highest AP score, making it the most reliable model for maintaining the desired attributes during editing. It also has a high CR score, making it effective in applying changes, while maintaining a moderate IP value.

These results highlight the strengths and weaknesses of each model in different aspects of face editing.

5.4.2 Performance Comparison

As we have explained before, we evaluate the performance of the face editing models based on three criteria: Training Time, Inference Time and GPU Memory usage. These metrics are crucial for understanding the computational efficiency and resource requirements of each model. We calculate the inference time by measuring the amount of time an approach takes to change one specific attribute in one specific image. For measuring and comparing the training time in a fair way, we have used a batch size of 1 to train the four models.

Model	Training Time	Inference Time	GPU
InterfaceGAN	5h 19min	0.8s	0.7GB
TediGAN	13h 36min	99s	1.2GB
Multi-Attribute Latent Transformer	58min	0.2s	0.8GB
StyleCLIP	9h 22min	98s	1.2GB

Table 5.6: Performance Comparison of Facial Attribute Editing Models

The results in Table 5.6 highlight the following observations:

- **InterfaceGAN** has a moderate training time of 5 hours and 19 minutes, with an inference time of 0.8 seconds and GPU memory usage of 0.7 GB. This suggests that while the model is relatively quick to train and efficient in GPU memory usage, its inference time is slower compared to the Multi-Attribute Latent Transformer.

- **TediGAN** has the longest training time of 13 hours and 36 minutes, and the highest inference time of 99 seconds. It also consumes the most GPU memory at 1.2 GB, along with StyleCLIP. This indicates that TediGAN is the most resource-intensive model in terms of both training and inference. Given the necessity of a text encoder, it is understandable that this model requires additional processing time.

- **Multi-Attribute Latent Transformer** stands out with the shortest training time of 58 minutes and the fastest inference time of 0.2 seconds. It uses 0.8 GB of GPU memory, making it the most efficient model in terms of both training and inference speed. This model is particularly suitable for scenarios where rapid processing and lower resource usage are critical.

- **StyleCLIP** has a relatively long training time of 9 hours and 22 minutes and an inference time of 98 seconds. It also uses 1.2 GB of GPU memory, similar to TediGAN. While StyleCLIP is resource-intensive, its high AP and

CR scores from the previous comparison suggest that its performance might justify the higher computational cost in some applications.

In summary, the performance comparison reveals significant differences in the computational demands of each model. The choice of model should therefore consider not only the accuracy and effectiveness of face editing but also the available computational resources and the required processing speed. The Multi-Attribute Latent Transformer offers the best efficiency, while StyleCLIP, although providing a stronger performance in attribute preservation, is more resource-intensive.

5.4.3 Attribute Correlation Analysis

In this section, we will analyze how altering one attribute influences the probability of another attribute being present. Previously, we observed that the Multi-Attribute Latent Transformer exhibited the best overall performance among the four models. Consequently, we have chosen to focus this analysis on this model, comparing it to its original version, the Single Latent Transformer.

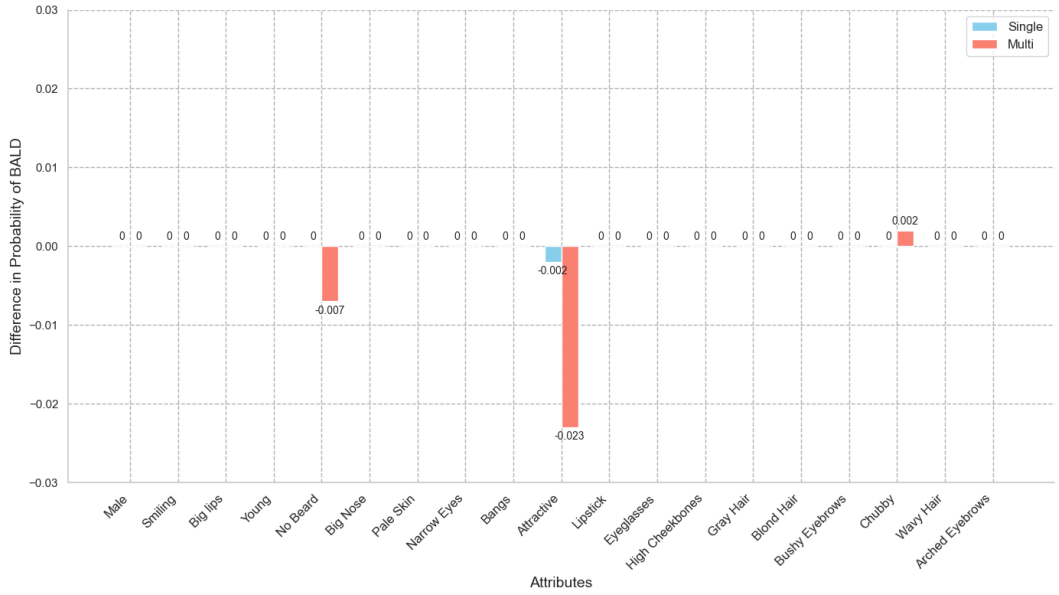

Figure 5.6: P(Bald) before and after changing another attribute.

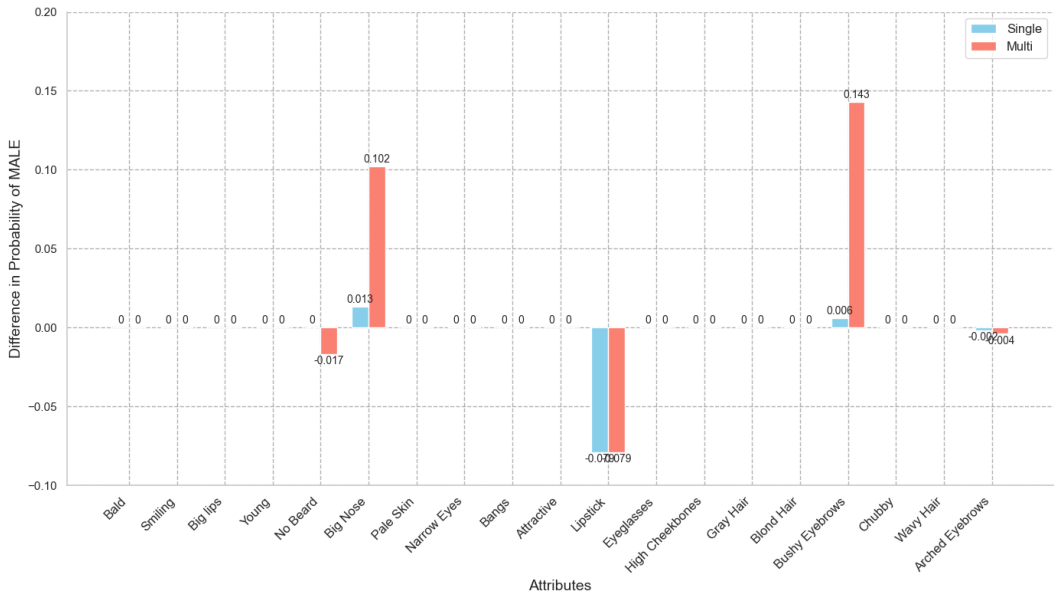

Figure 5.7: P(Male) before and after changing another attribute.

The correlations between the attributes in the CelebA-HQ dataset are manifested in the final results. While both models are affected by this correlations, it is the Multi-Attribute Latent Transformer the one that exhibits a higher sensitivity to the biases in the dataset.

The higher sensitivity observed in the Multi-Attribute Latent Transformer model can be directly linked to its superior performance in the Change Ratio metric. As previously discussed, this metric evaluates how effectively the model adjusts the desired attributes. Consequently, it stands to reason that this model exhibits stronger changes in correlated attributes due to its enhanced ability to capture and respond to attribute interactions. Conversely, for non-correlated attributes, neither the Single nor the Multi-Attribute models significantly alter undesired attributes, thereby maintaining their intended behavior and stability. In particular, the attributes 'Pale Skin,' 'Narrow Eyes,' and 'Bangs' remain unaffected by changes in any other attribute in both models, and thus, their graphs are not included. The correlation matrix in Figure 5.3 confirms that these attributes have low correlation with the others.

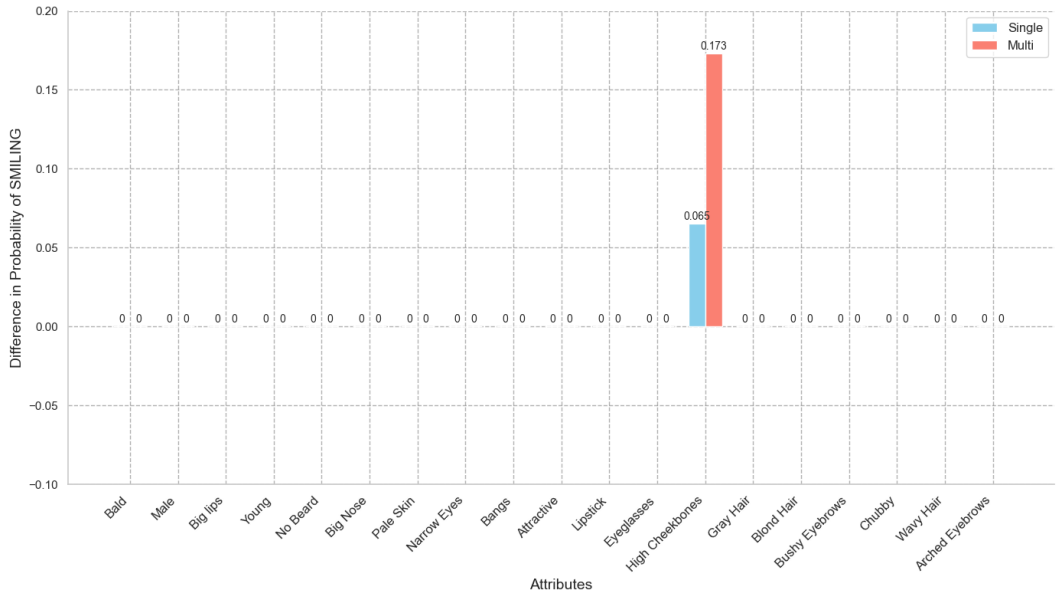

Figure 5.8: P(Smiling) before and after changing another attribute.

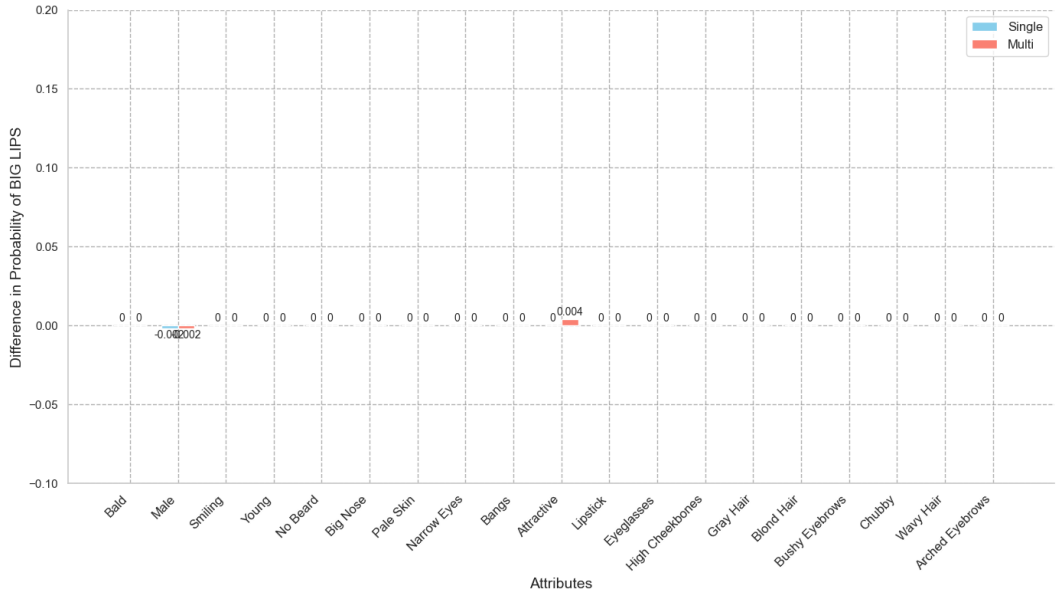

Figure 5.9: P(BigLips) before and after changing another attribute.

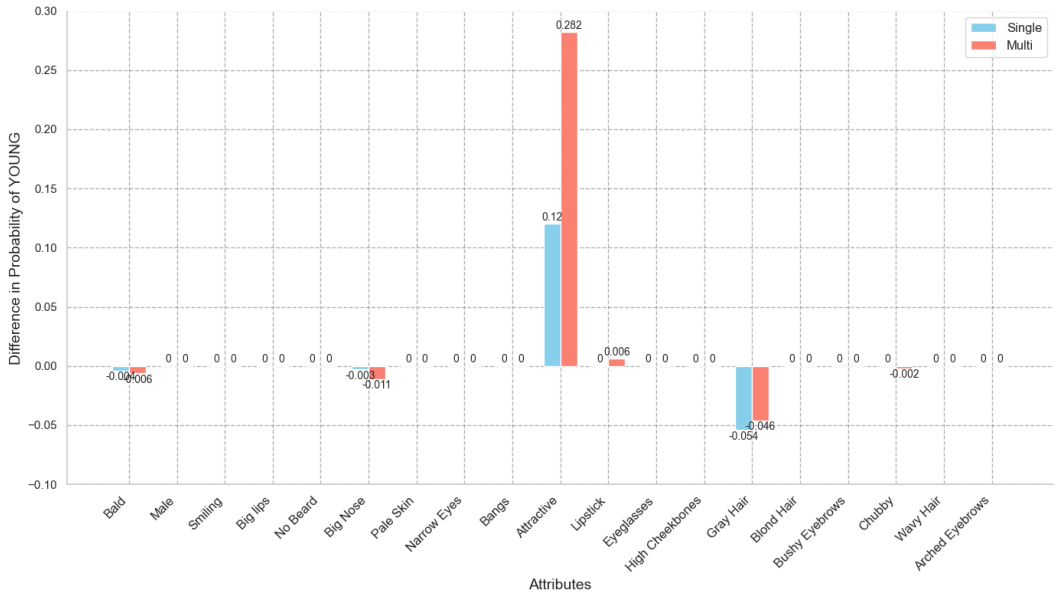

Figure 5.10: P(Young) before and after changing another attribute.

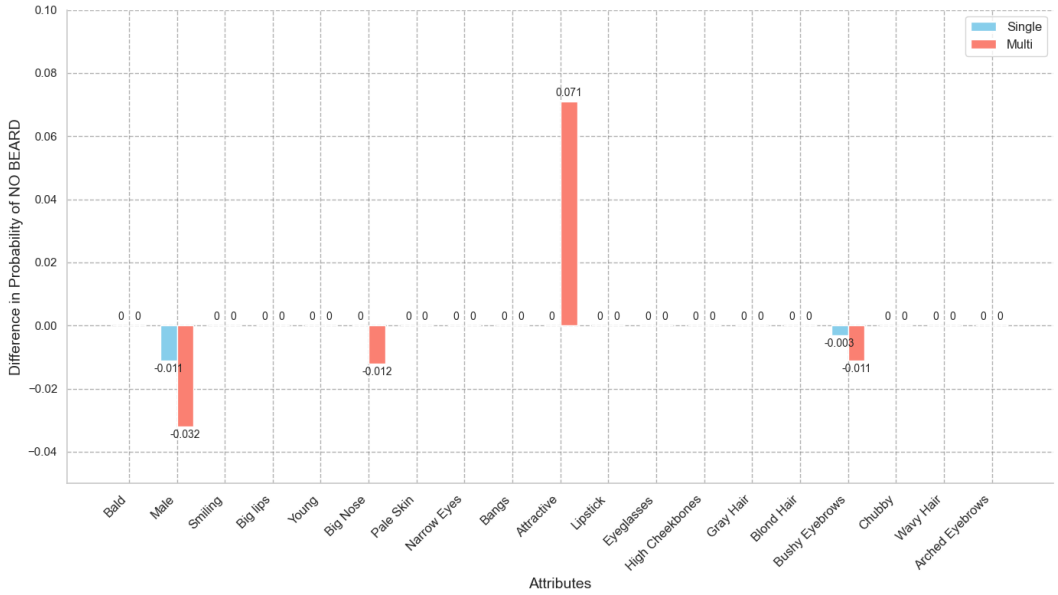

Figure 5.11: P(NoBeard) before and after changing another attribute.

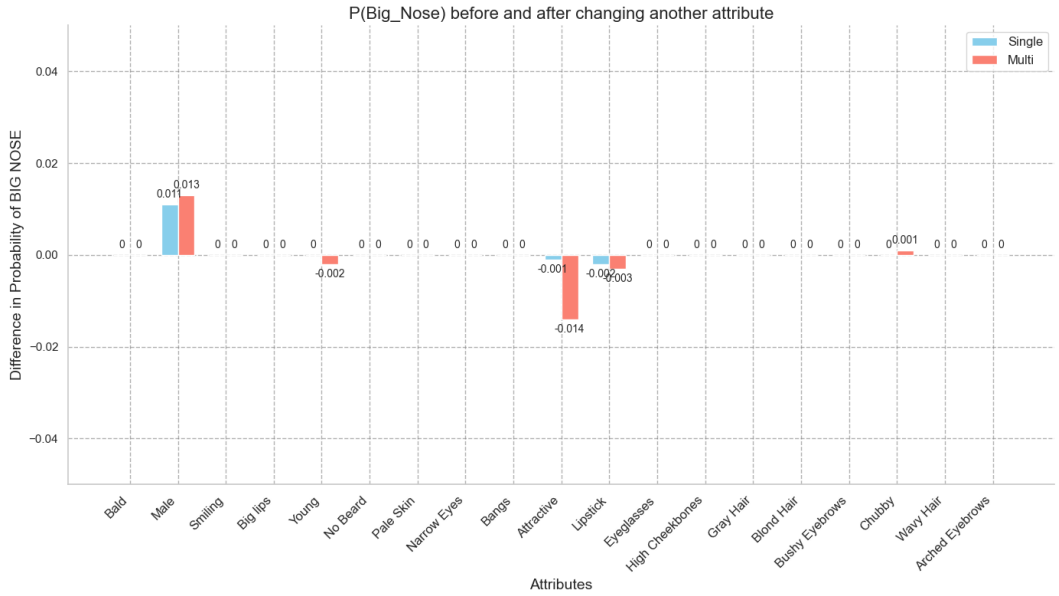

Figure 5.12: P(BigNose) before and after changing another attribute.

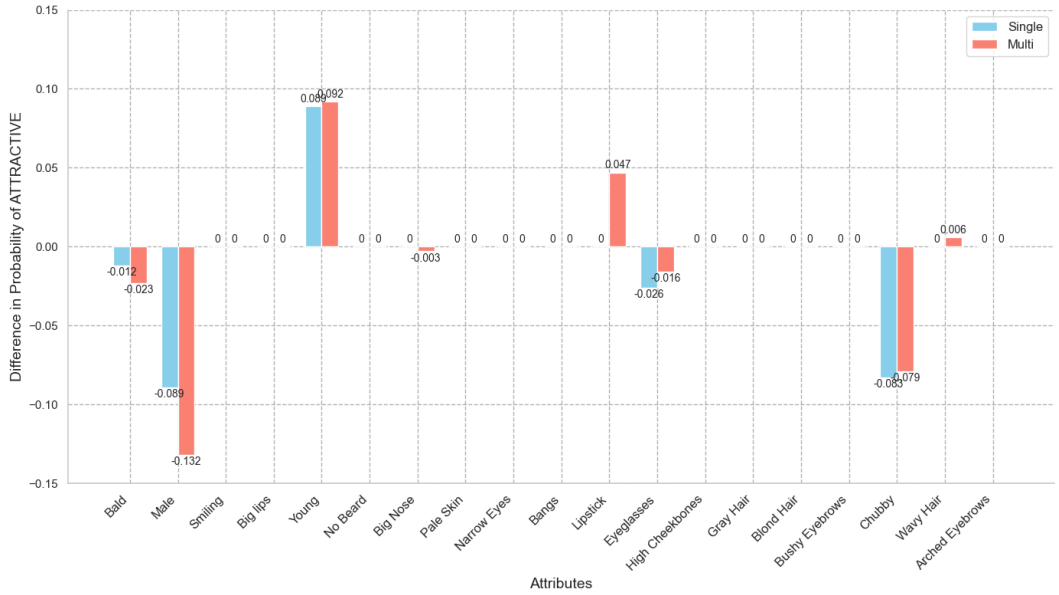

Figure 5.13: P(Attractive) before and after changing another attribute.

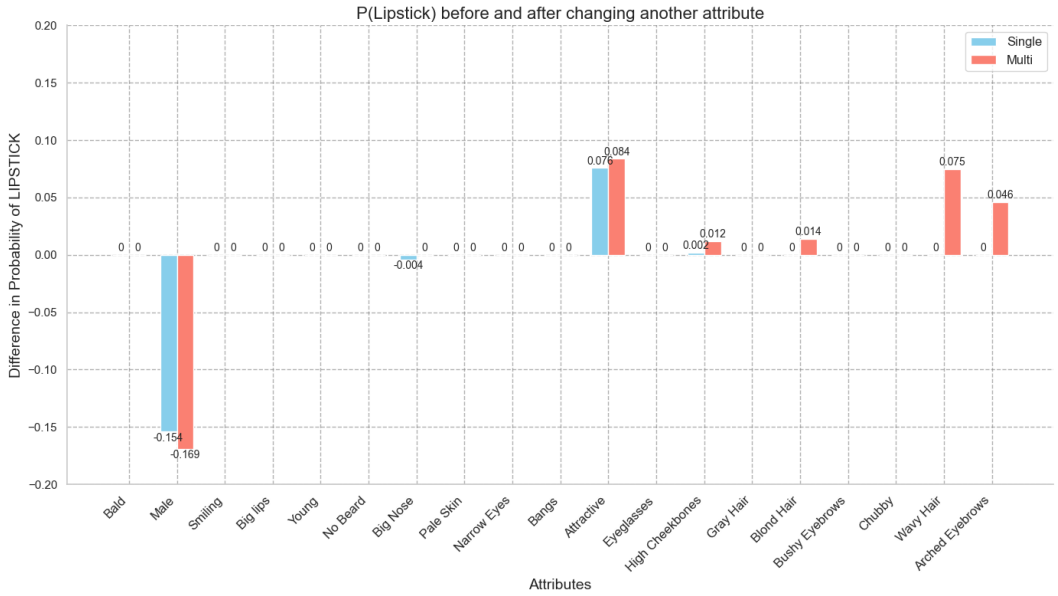

Figure 5.14: P(WearingLipstick) before and after changing another attribute.

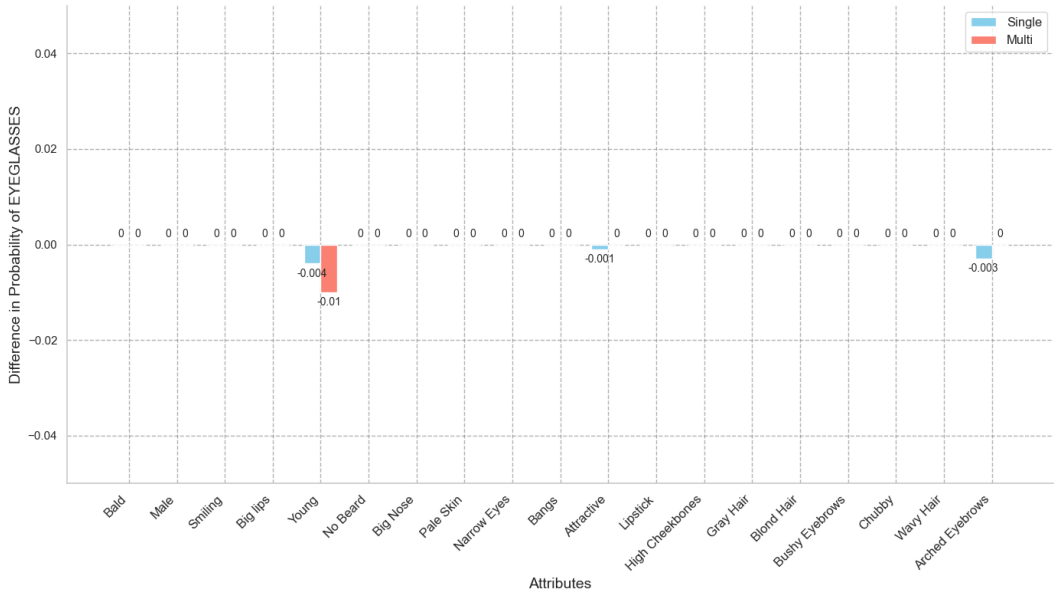

Figure 5.15: P(Eyeglasses) before and after changing another attribute.

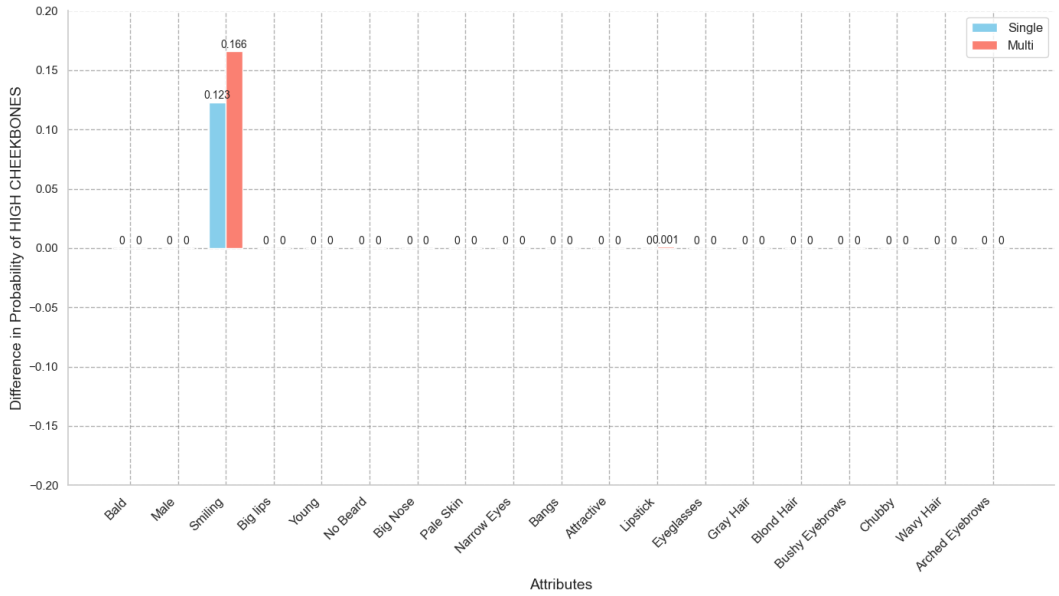

Figure 5.16: P(HighCheekbones) before and after changing another attribute.

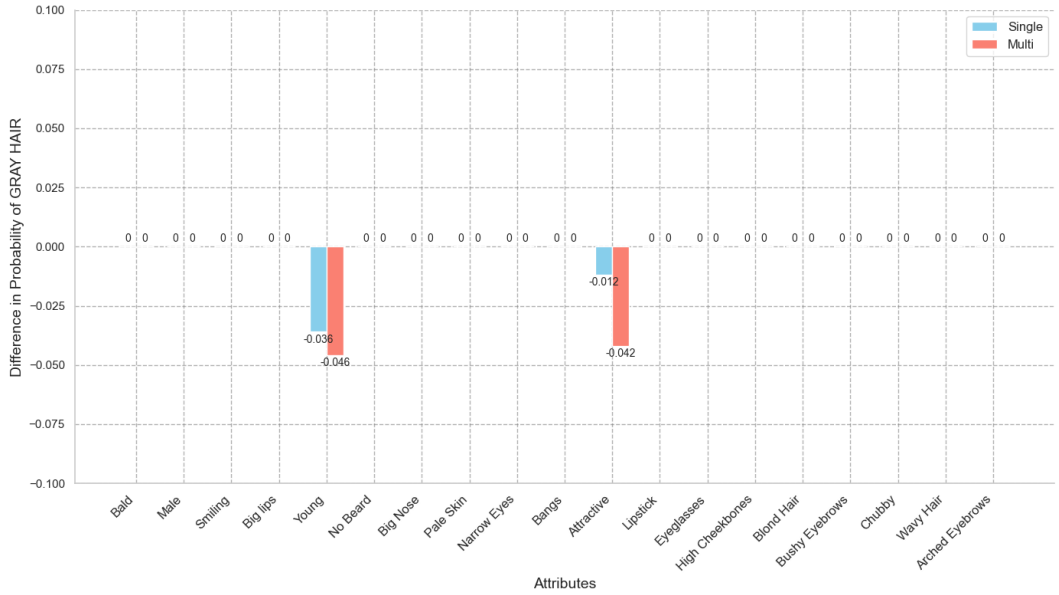

Figure 5.17: P(GrayHair) before and after changing another attribute.

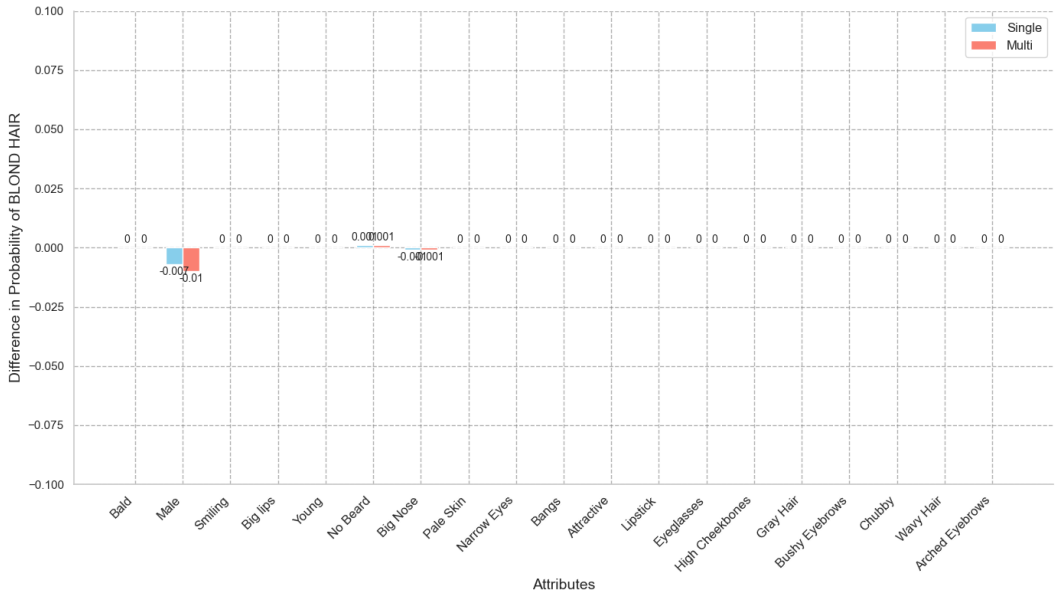

Figure 5.18: P(BlondHair) before and after changing another attribute.

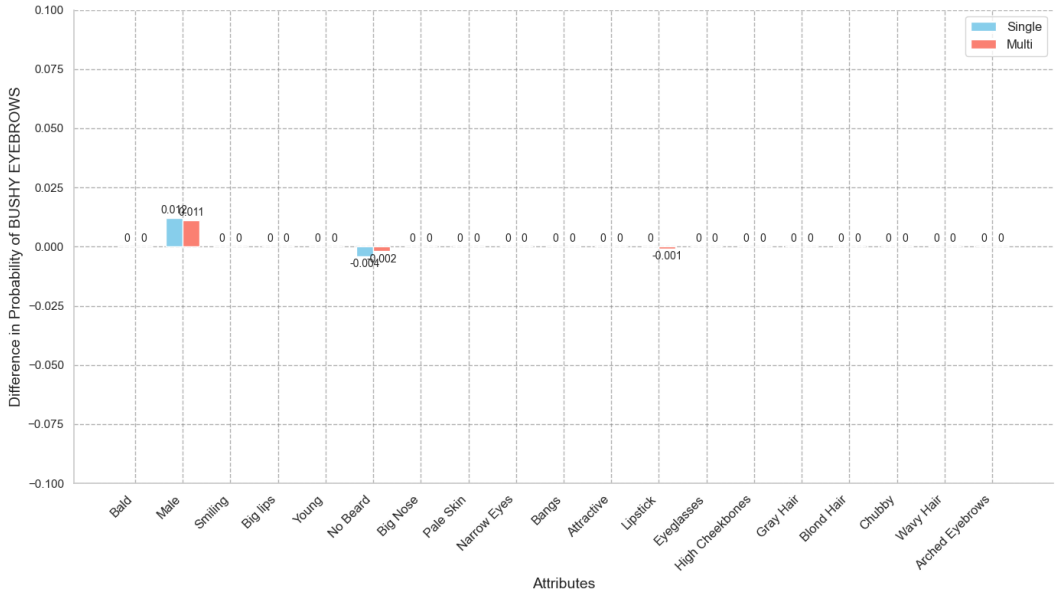

Figure 5.19: P(BushyEyebrows) before and after changing another attribute.

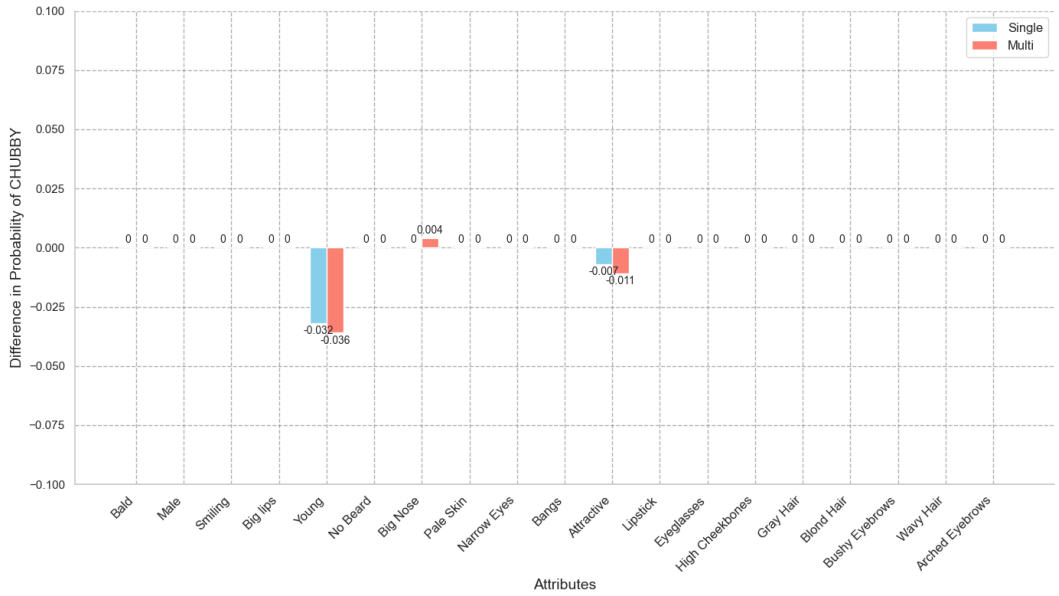

Figure 5.20: P(Chubby) before and after changing another attribute.

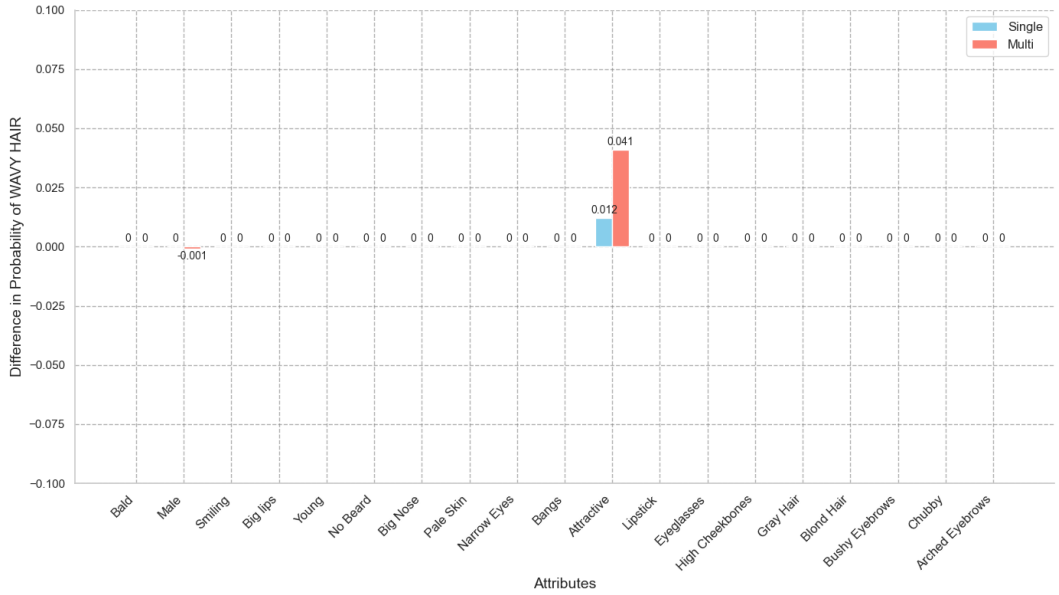

Figure 5.21: P(WavyHair) before and after changing another attribute.

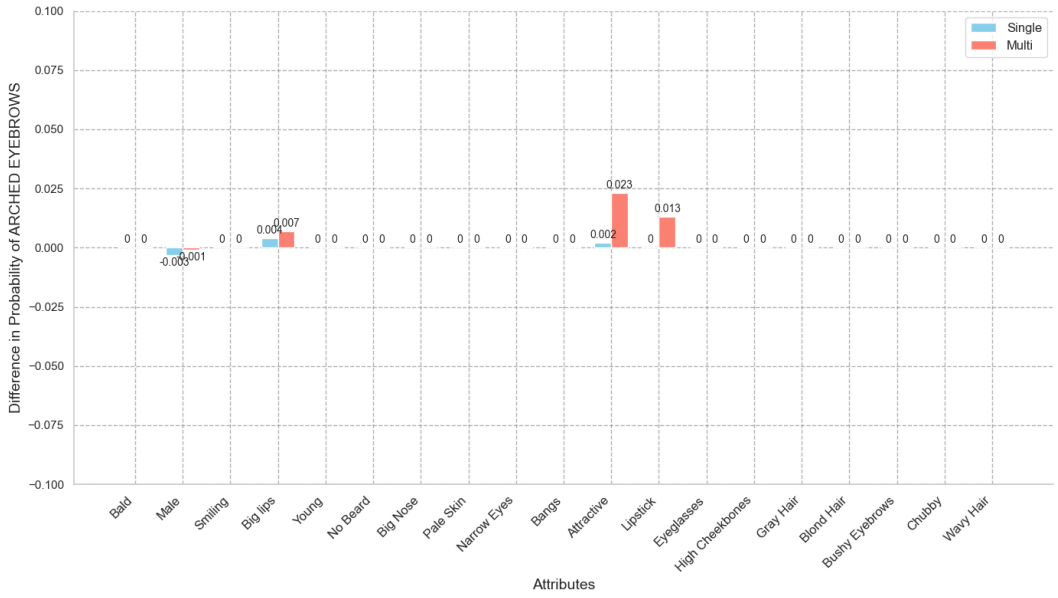

Figure 5.22: P(ArchedEyebrows) before and after changing another attribute.

Chapter 6

Conclusions

In this chapter, we will summarize the key findings and outcomes of our research, reflect on the extent to which we have met our initial objectives, and explore potential directions for future work. Additionally, personal reflections and insights gained throughout the project will be shared.

6.1 Answer to Research Objectives

Let us review the objectives of this work and assess whether they have been achieved.

Definitely, we have gained a lot of knowledge on generative adversarial networks and the latent space. Furthermore, we have studied and trained different models: InterFaceGAN, TediGAN, StyleCLIP, Single Latent Transformer and Multi-Attribute Latent Transformer.

In evaluating these models, we ensured a consistent basis for comparison. This involved using the same datasets, metrics, and evaluation criteria to objectively assess each model's performance. Through this evaluation process, we identified the strengths and weaknesses of each model. Our findings highlighted the overall superior performance of the Multi-Attribute Latent Transformer model, while also pinpointing its areas for improvement.

Additionally, we conducted a thorough investigation into the biases introduced by attribute correlations within the training datasets. This analysis provided significant insights into how these correlations can affect the models' outputs, leading to biased results. The insights gained from this investigation reinforced our previous conclusions, confirming the validity of our model evaluations.

One significant achievement of this project was the successful implementation of a user interface. This UI allows users to visually compare the performance of the different models, providing an interactive and intuitive platform for exploring the nuances of face attribute editing. Users can manipulate various attributes and see real-time results, facilitating a deeper understanding of each model's capabilities and limitations.

In conclusion, all the initial objectives of this project have been met. We have acquired the necessary knowledge on face attribute editing models, trained and evaluated multiple models, studied dataset biases, and developed a user-friendly web application for model comparison. This comprehensive approach has not only advanced our understanding of the field but also contributed practical tools and insights that can benefit future research and applications in facial attribute editing.

6.2 Future Work

The insights gained during this project have laid a solid foundation for several promising avenues of future research. One of the primary directions will be to incorporate these findings into a forthcoming paper on the Multi-Attribute Latent Transformer, authored by Adria Carrasquilla [14].

Building on our current work, future research can focus on the following areas:

- **Enhance Multi-Attribute Latent Transformer.** Despite its superior overall performance compared to other models, there is room for improvement in attribute preservation. Future efforts will involve experimenting with larger and more diverse datasets and implementing novel loss functions that more accurately capture the nuances of attribute manipulation.
- **Mitigate Undesired Biases.** Strategies need to be developed to address the unwanted biases identified during this work, such as the tendency to increase the 'Smile' attribute when changing 'High Cheekbones'. This could involve creating or sourcing more balanced datasets and implementing bias correction algorithms during the training process.
- **Deploy Face Editing Application**. While the code is available on GitHub, many users are unable to utilize it due to a lack of coding knowledge. The next steps should include deploying the application in a user-friendly format, making it accessible for everyone to use without requiring coding skills.

6.3 Personal Conclusion

The world of Artificial Intelligence has always fascinated me, but it was during my Erasmus semester in Rome, while taking a 'Computer Vision' course, that my interest in the this field truly ignited. At that time, I felt there were countless aspects of AI I wanted to explore and understand. This curiosity drove me to choose computer vision as the focus of my final project and I am happy for that. I am truly grateful for the opportunity that I have had to explore these face editing technologies and to conduct these experiments.

Initially, I planned to become a data analyst after graduating and leave further studies behind. However, working on this project has fueled my desire to gain more expertise in AI. As a result, I have decided to also pursue a Master's Degree in Data Science, Big Data, and Artificial Intelligence while continuing to work. I believe this step will allow me to keep expanding my knowledge and skills in this exciting field.

Appendix

This additional chapter serves to supplement the project by providing screenshots of the developed application for visual model comparison.

A User Interface

During the development of this work, we also developed a User Interface using the Gradio UI Framework for Machine Learning applications. The main reason behind the creation of this interface is for it to be used for anyone external to the project. For that reason, we have tried to make it as user-friendly as possible. Also, it serves as a way to show the final results interactively. Source code is available in this GitHub repository.

The UI was designed with a focus on simplicity and ease of use. Users can easily upload their images, select the desired machine learning model, and visualize the output without needing any prior technical knowledge. They can also generate the results for all models at the same time to be able to directly visually compare them. In the interface some basic information about each of the models, the links to additional information and instructions on how to use the UI are also provided. By including these features, we aim to make the interface a comprehensive tool for both demonstration and practical use. Below are some screenshots illustrating the key aspects of the interface:

Figure 6.1: UI. Initial page.

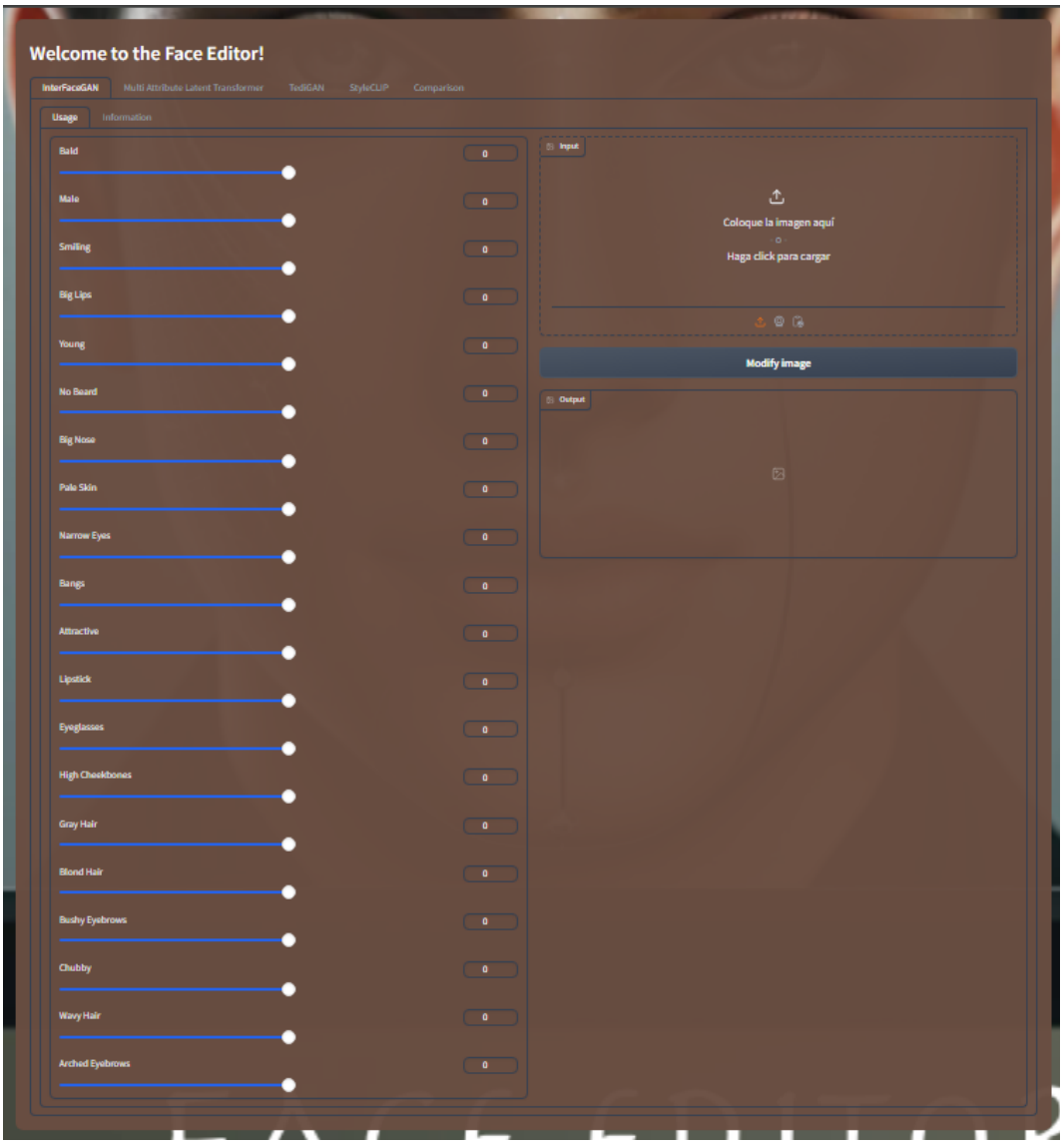

Figure 6.2: UI. InterFaceGAN.

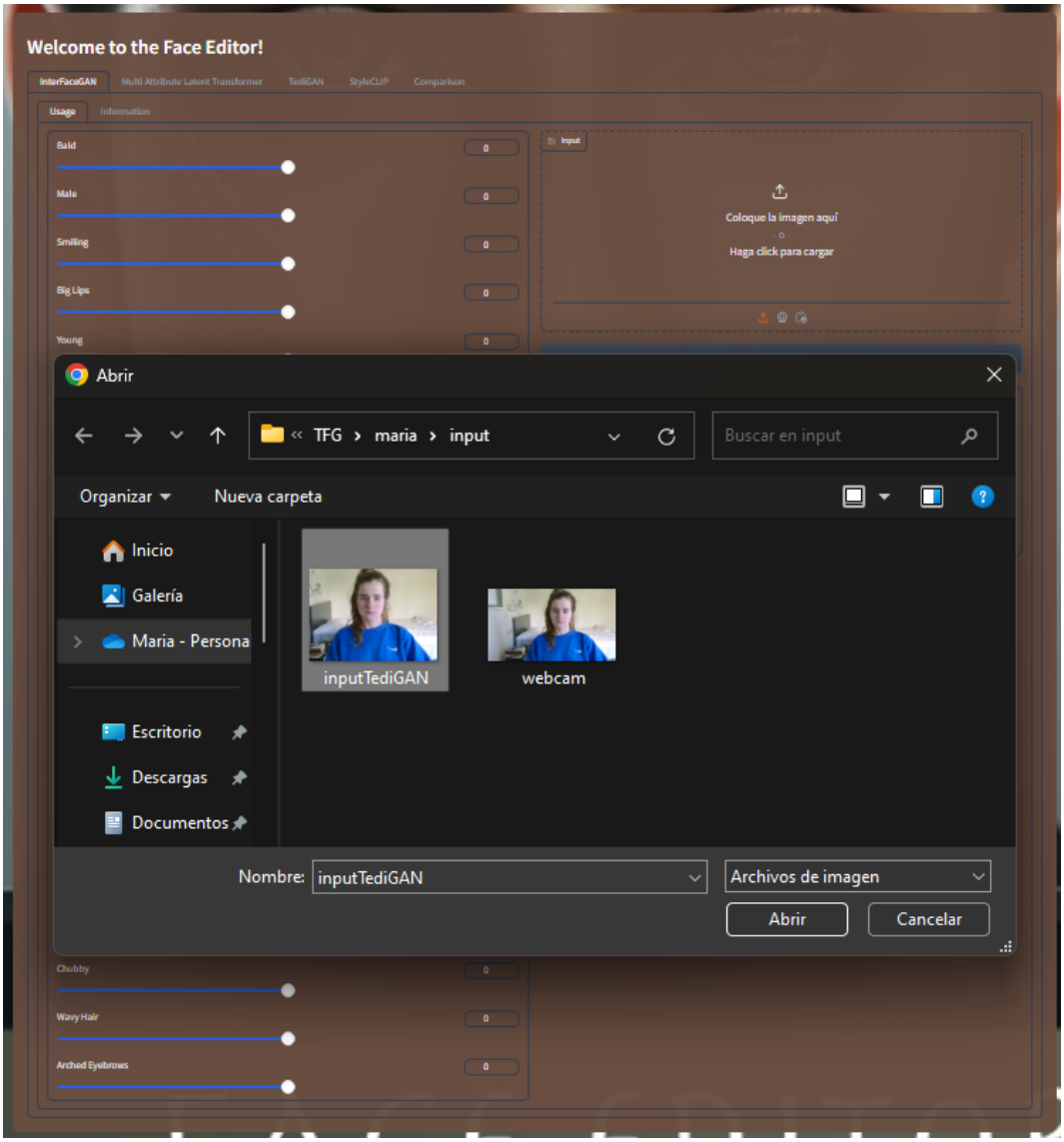

Figure 6.3: UI. Upload iamge from device.

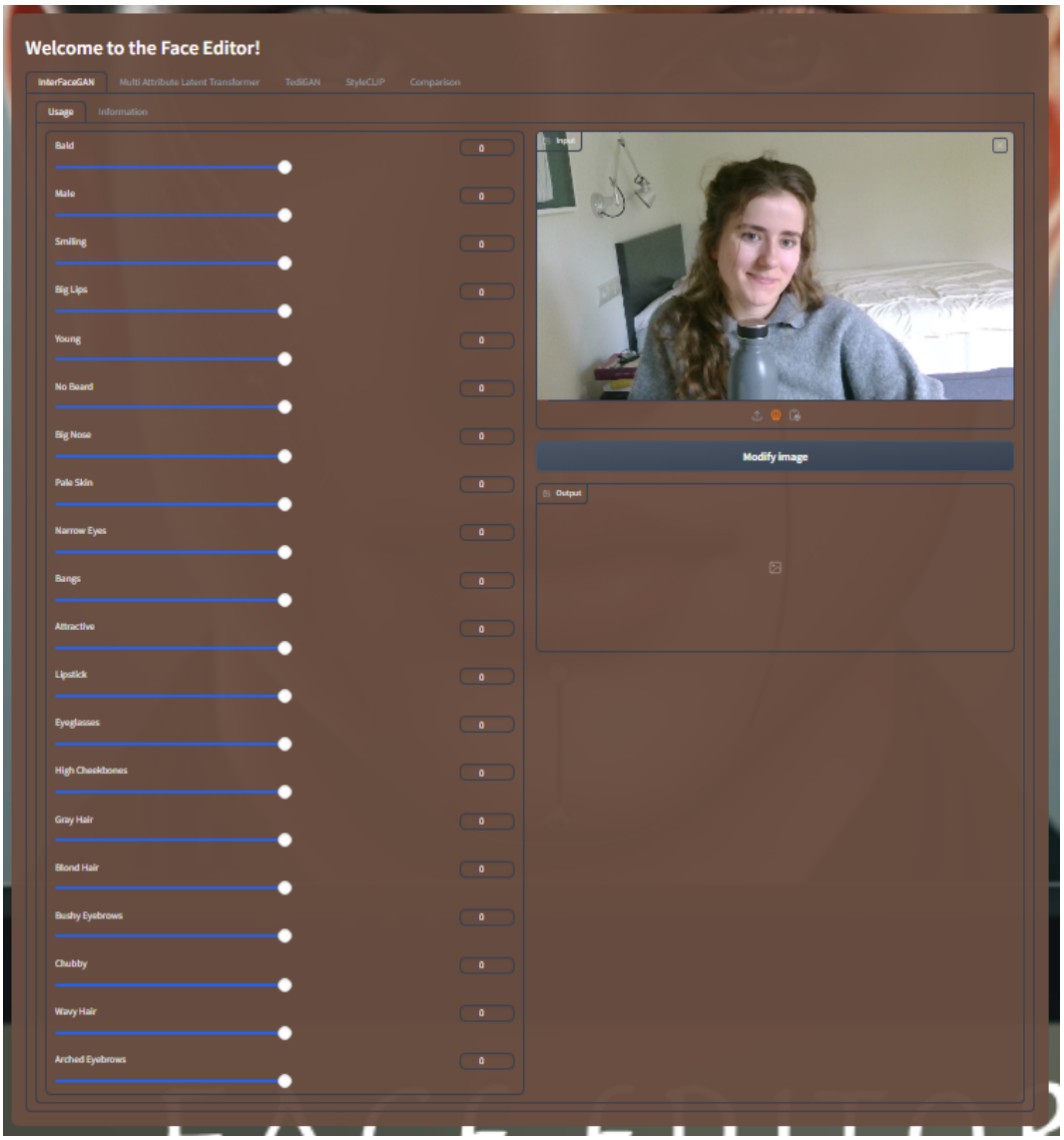

Figure 6.4: UI. Webcam.

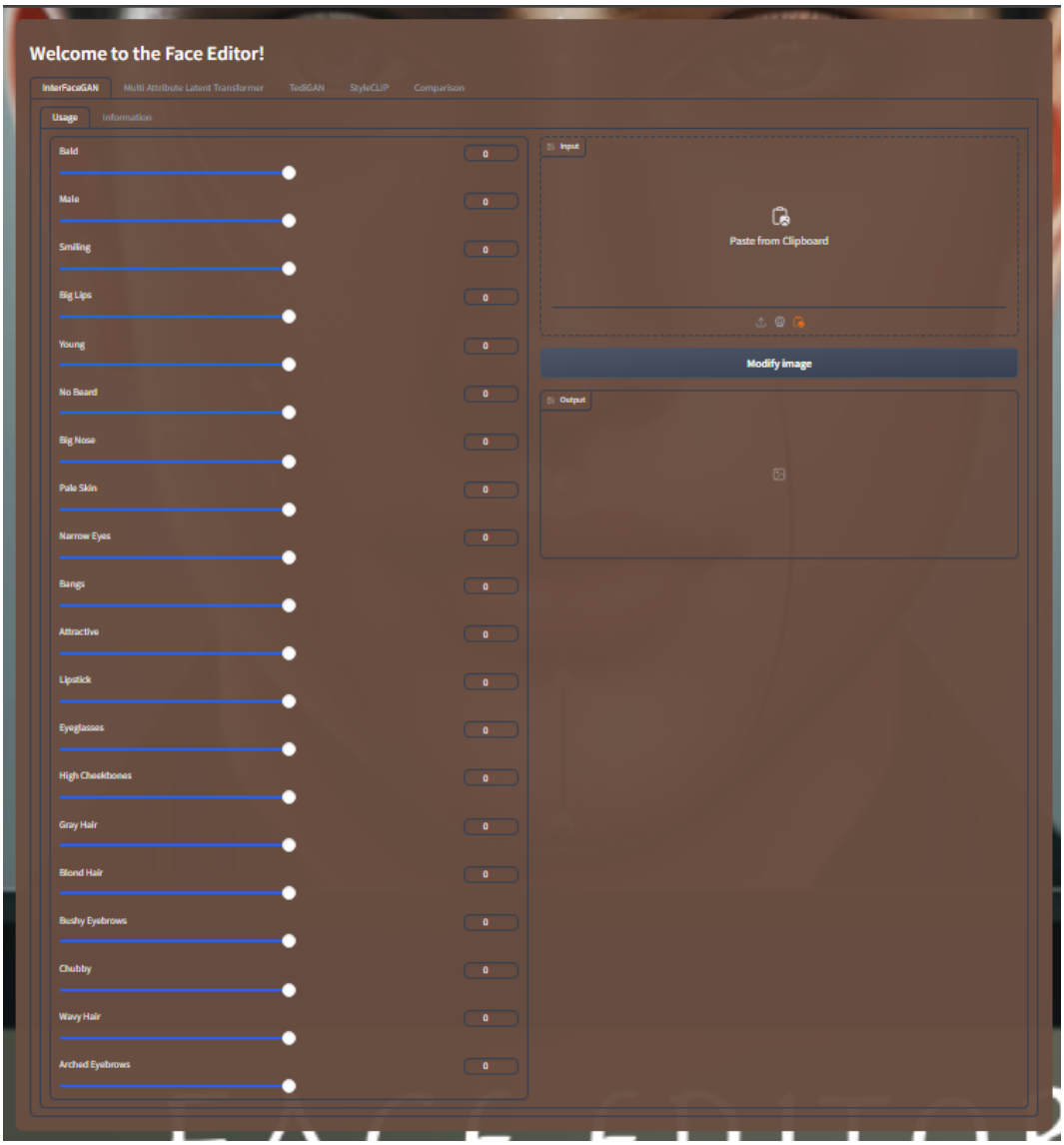

Figure 6.5: UI. Paste image.

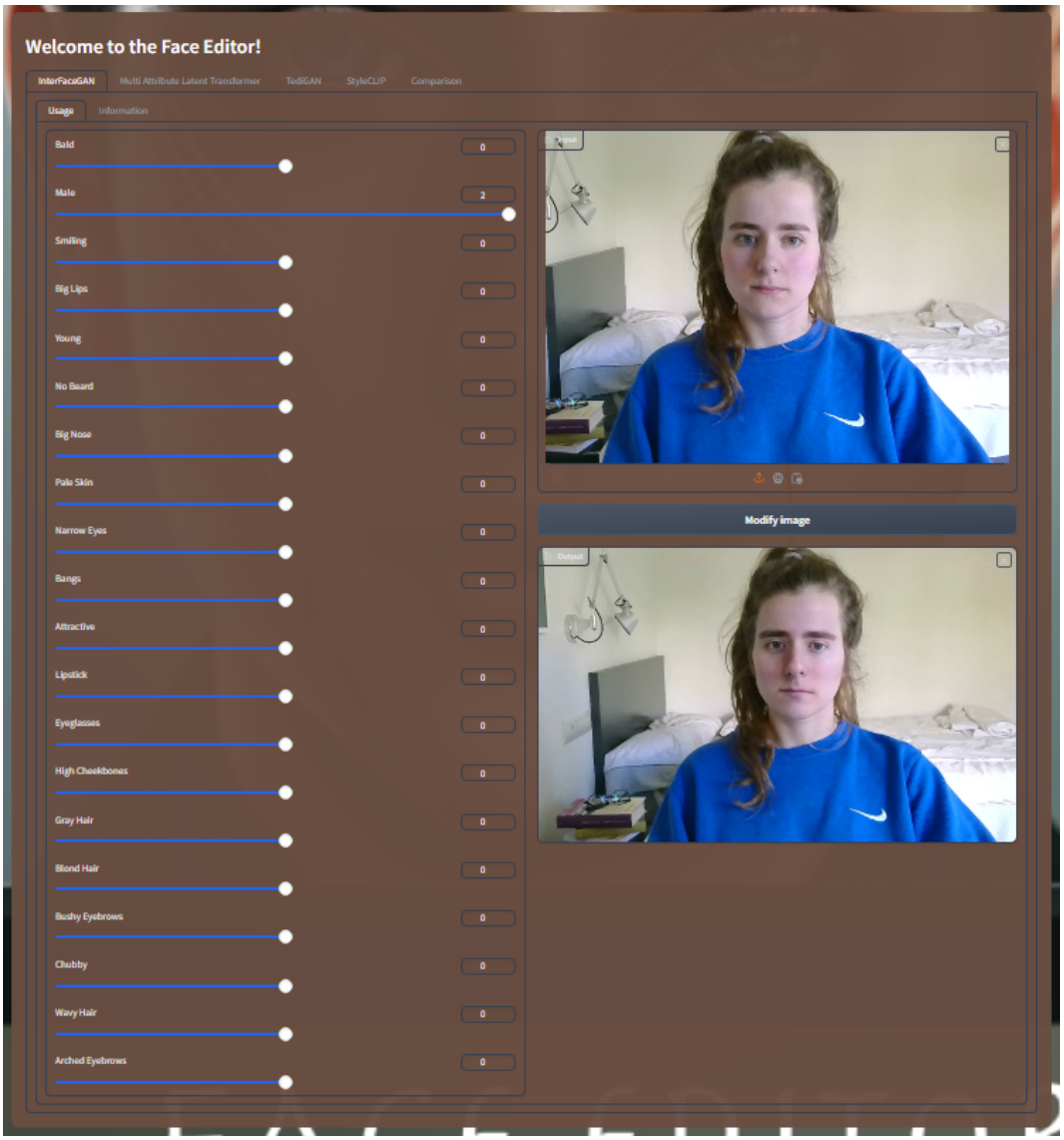

Figure 6.6: UI. Modify 'Male' attribute of image with InterfaceGAN.

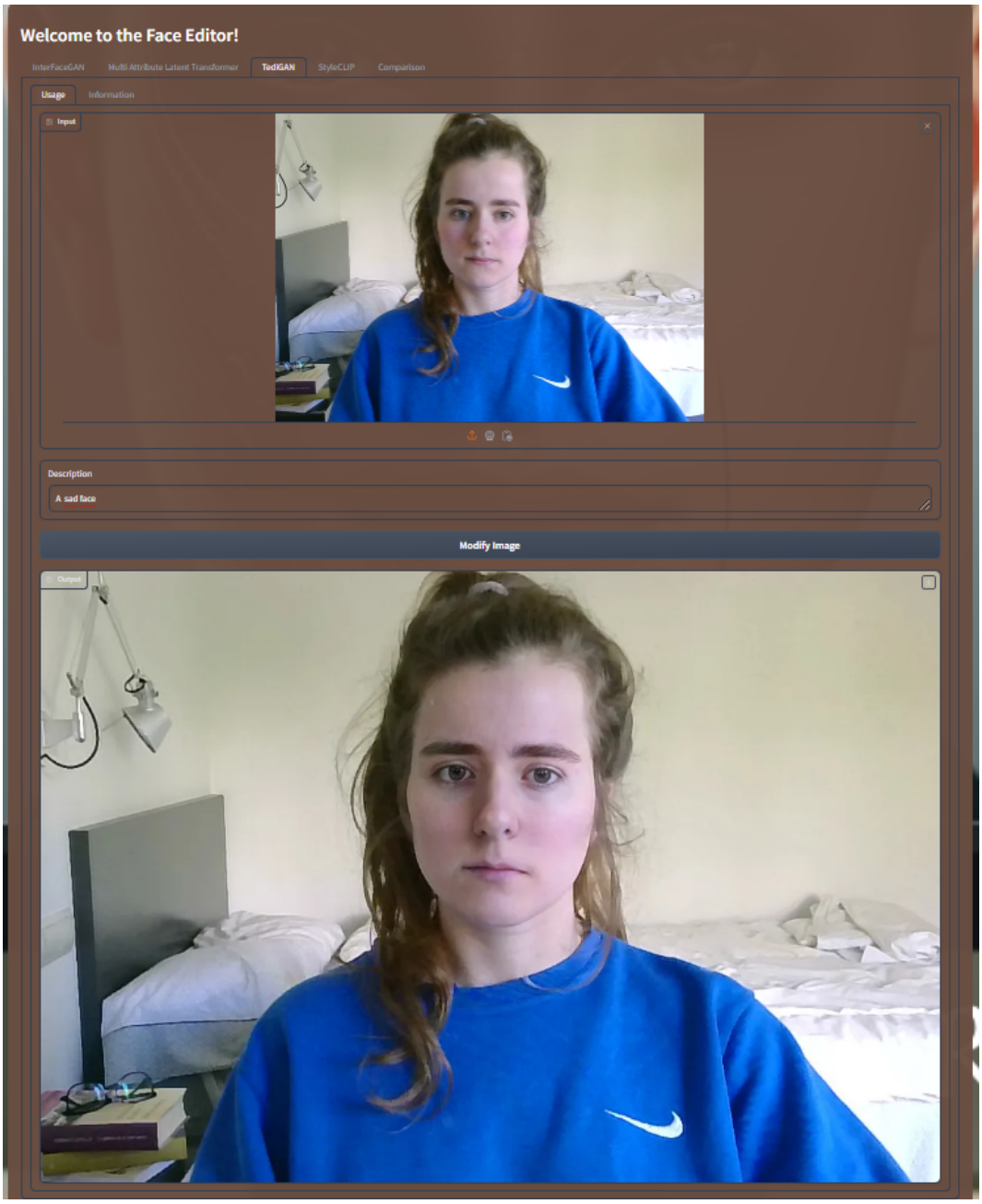

Figure 6.7: UI. Modify image with TediGAN using text input 'A sad face'.

Figure 6.8: UI. Information about InterFaceGAN.

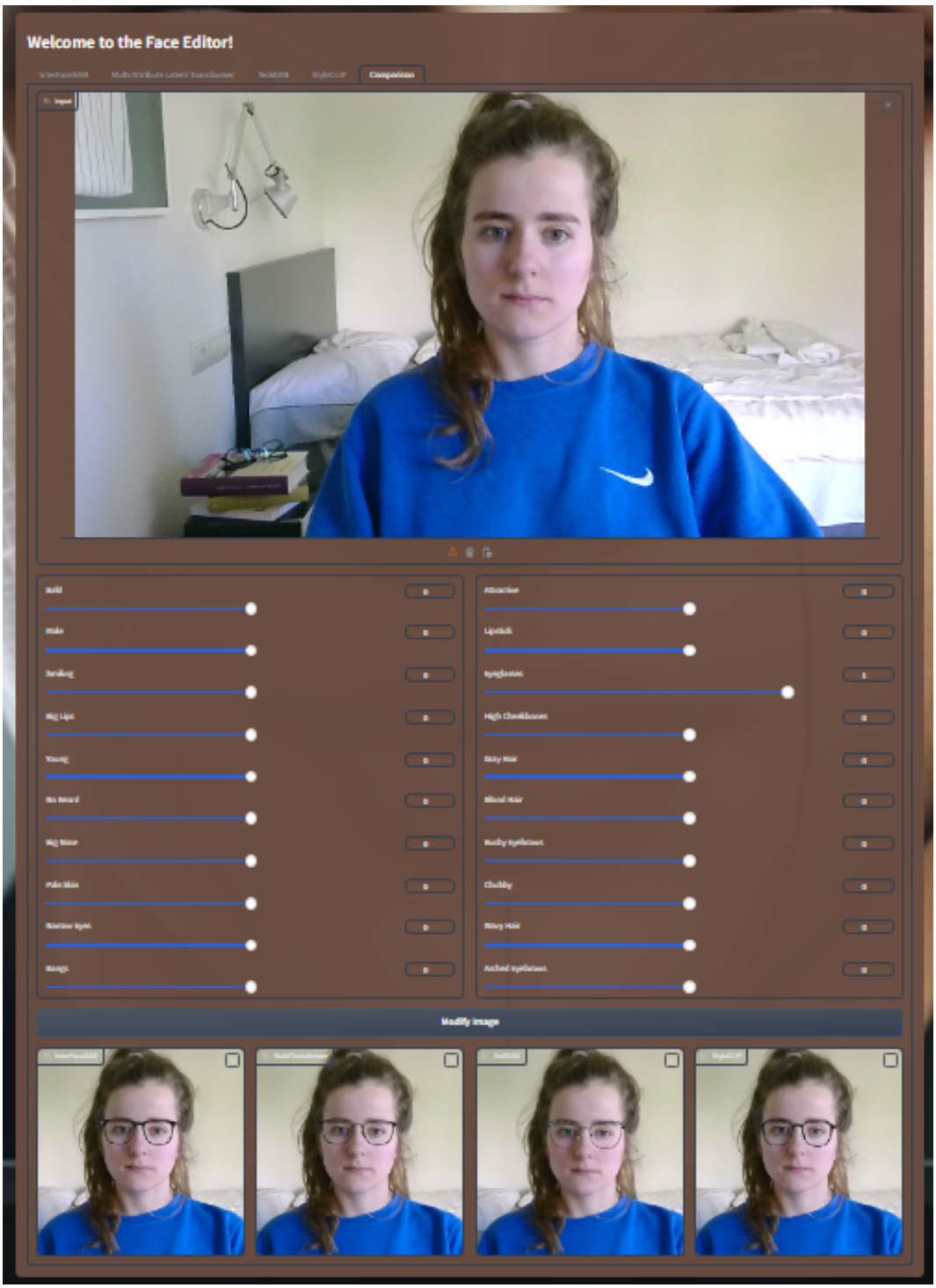

Figure 6.9: UI. Visual comparison of all the models.

Bibliography

[1] Aston Zhang, Zachary C. Lipton, Mu Li, Alexander J. Smola, *Dive into Deep Learning*, 2021.

[2] Andrew Ng's Number One Fan, *CS231n: Convolutional Neural Networks for Visual Recognition*, GitHub repository, 2021.

[3] Tero Karras, Samuli Laine, Timo Aila, Jaakko Lehtinen, *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, ICLR, 2018.

[4] Tero Karras, Samuli Laine, Timo Aila, *A Style-Based Generator Architecture for Generative Adversarial Networks*, NVIDIA, 2019.

[5] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, Timo Aila, *Analyzing and Improving the Image Quality of StyleGAN*, CVPR, 2020.

[6] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. *Alias-Free Generative Adversarial Networks*, arXiv preprint arXiv:2106.12423, 2021.

[7] Wu, Y., Zhang, Y., Huang, K., Zhang, B., *Stylespace Analysis: Disentangled Controls for Generative Image Synthesis*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[8] Xun Huang, Serge Belongie, *Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization*, Department of Computer Science, Cornell Tech, Cornell University, 2017.

[9] Yujun Shen, Jinjin Gu, Xiaoou Tang, Bolei Zhou, *InterFaceGAN: Interpreting the Disentangled Face Representation Learned by GANs*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020.

[10] Weihao Xia, Yujiu Yang, Jing-Hao Xue, Baoyuan Wu, *TediGAN: Text-Guided Diverse Face Image Generation and Manipulation*, Tsinghua Shenzhen International Graduate School, Tsinghua University, China, 2021.

[11] Or Patashnik, Zongze Wu, Eli Shechtman, Dani Lischinski, Daniel Cohen-Or, *StyleCLIP: Text-Driven Manipulation of StyleGAN Imagery*, ACM Transactions on Graphics, 2021.

[12] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever, *Learning Transferable Visual Models From Natural Language Supervision*, OpenAI, 2021.

[13] Xu Yao, Alasdair Newson, Yann Gousseau, Pierre Hellier, *A Latent Transformer for Disentangled Face Editing in Images and Videos*, LTCI, Télécom Paris, Institut Polytechnique de Paris, France, 2021.

[14] Adrià Carrasquilla Fortes, *Latent Multi-Attribute Transformer for Face Editing in Images*, Thesis report, Universitat Rovira i Virgili (URV), Universitat de Barcelona (UB), Universitat Politècnica de Catalunya (UPC) - BarcelonaTech, 2023.

[15] Kingma, D. P., Welling, M., *Auto-Encoding Variational Bayes*, Proceedings of the 2nd International Conference on Learning Representations (ICLR), 2014.

[16] D. E. King, *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[17] He, H. and Garcia, E. A., *Learning from imbalanced data*, IEEE Transactions on Knowledge and Data Engineering, 2009.

[18] Johnson, J. M. and Khoshgoftaar, Survey on deep learning with class imbalance. *Journal of Big Data*, 2019.

[19] Yi Wu, Amittai Axelrod, David S. Hipp, and Yasemin Altun. *ChatEdit: Towards Personalized Text Editing Assistance via Interactive Natural Language Feedback*, arXiv preprint arXiv:2209.12340, 2022.

[20] Cao, J., Li, Y., and Zhang, Z., *Partially shared multi-task convolutional neural network with local constraint for face attribute learning*, in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.

[21] Hand, E. and Chellappa, R., *Attributes for improved attributes: A multi-task network utilizing implicit and explicit relationships for facial attribute classification*, Proceedings of the AAAI Conference on Artificial Intelligence, 2017.

[22] Jolliffe, I., *Principal Component Analysis*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[23] Bryson Lingenfelter, Sara R. Davis, and Emily M. Hand, *A Quantitative Analysis of Labeling Issues in the CelebA Dataset*, University of Nevada, Reno.

[24] Hongwei Wang, Jure Leskovec. *SD-GAN: Structural and Denoising GAN for Learning Robust and Discriminative Representations*, arXiv preprint arXiv:1905.04215, 2019.

[25] Wang,Z.,Qinami,K.,Karakozis,I.C.,Genova,K.,Nair,P.,Hata,K.,Russakovsky, O., *Towards fairness in visual recognition: Effective strategies for bias mitigation*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020

[26] Warren S. McCulloch and Walter Pitts. *A logical calculus of the ideas immanent in nervous activity*, in The bulletin of mathematical biophysics 5.4, 1943.

[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Nets*, in Advances in Neural Information Processing Systems 27 (NIPS 2014), 2014.

[28] Zhang, R., Isola, P., Efros, A. A., Shechtman, E., Wang, O., *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[29] A. H. Bermano, R. Gal, Y. Alaluf, R. Mokady, Y. Nitzan, O. Tov, O. Patashnik, and D. Cohen-Or, *State-of-the-art in the architecture, methods and applications of stylegan*, Computer Graphics Forum, vol. 41, no. 2. Wiley Online Library, 2022.

[30] Tewari, A., Elgharib, M., Bharaj, G., Bernard, F., Seidel, H.-P., Pérez, P., Zollo, P., Theobalt, C., *Face Generation and Editing with StyleGAN: A Survey*, 2022.

[31] Abdal, R., Qin, Y., Wonka, P., *Image2StyleGAN: How to Embed Images Into the StyleGAN Latent Space*, Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2019.

[32] Ha, D., Dai, A., Le, Q. V., *HyperNetworks*, Proceedings of the International Conference on Learning Representations (ICLR), 2017.

[33] Wang, Z., Bovik, A. C., Sheikh, H. R., Simoncelli, E. P., *Image Quality Assessment: From Error Visibility to Structural Similarity*, IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, 2004.

[34] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L·L Polosukhin, I., *Attention Is All You Need*, Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2017.

[35] Elman, J. L., *Finding Structure in Time*, Cognitive Science, vol. 14, no. 2, pp. 179-211, 1990.

[36] Hochreiter, S., Schmidhuber, J., *Long Short-Term Memory*, Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[37] Devlin, J., Chang, M. W., Lee, K., Toutanova, K., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), 2019.

[38] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., *Improving Language Understanding by Generative Pre-Training*, OpenAI, 2018.

[39] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, arXiv preprint arXiv:1907.11692, 2019.

[40] Goodfellow, I., Bengio, Y., Courville, A., *Deep Learning*, MIT Press, 2016.

[41] Nair, V., Hinton, G. E., *Rectified Linear Units Improve Restricted Boltzmann Machines*, Proceedings of the 27th International Conference on Machine Learning (ICML), 2010.

[42] Rumelhart, D. E., Hinton, G. E., Williams, R. J., *Learning representations by back-propagating errors*, Nature, vol. 323, no. 6088, pp. 533-536, 1986.