NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

MddGAN : Multilinear Analysis of the GAN Latent Space

Lazaros E. Avgeridis

Supervisor: **Yannis Panagakis,** Associate Professor

ATHENS

MARCH 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

MddGAN : Πολυγραμμική Ανάλυση του Λανθάνοντος Χώρου του GAN

Λάζαρος Ε. Αυγερίδης

Επιβλέπων: **Ιωάννης Παναγάκης,** Αναπληρωτής Καθηγητής

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2022

BSc THESIS

MddGAN : Multilinear Analysis of the GAN Latent Space

Lazaros E. Avgeridis
S.N.: 1115201600013

SUPERVISOR: **Yannis Panagakis,** Associate Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

MddGAN : Πολυγραμμική Ανάλυση του Λανθάνοντος Χώρου του GAN

Λάζαρος Ε. Αυγερίδης
Α.Μ.: 1115201600013

ΕΠΙΒΛΕΠΩΝ: Ιωάννης Παναγάκης, Αναπληρωτής Καθηγητής

ABSTRACT

Generative Adversarial Networks (GANs) are currently an indispensable tool for semantic image editing, being widely used in a plethora of computer vision applications. Although these models are proven to encode rich semantic knowledge in their internal representations, they still lack an intuitive way to provide direct control to users, so that they can consistently influence the output image content. Once this knowledge is extracted however, it can be converted to human-interpretable controls for altering synthesized images in a predictable way.

In this thesis, we present MddGAN, an unsupervised technique for analyzing the GAN latent space and extracting vector directions corresponding to meaningful image transformations. In contrast to existing works, we perform a multilinear decomposition on the weights of a pre-trained generator, and we argue that such an exploration scheme can be more suitable in capturing the variability factors learnt with less entanglement. Furthermore, the proposed approach can mathematically divide the discovered semantics into groups, according to their semantic content. This separation happens in a completely unsupervised way, and essentially each dimension of the produced multilinear basis represents one such group.

By conducting several experiments on GANs trained on various datasets, we show how varying the number of explanatory factors discovered in the generative representations affects the semantic manipulations discovered. Moreover, we showcase several non-trivial directions highlighting the editing potential of our method. Furthermore, we compare MddGAN to the current supervised and unsupervised baselines both qualitatively and quantitatively. The results indicate that our approach is at least on par with these methods.

SUBJECT AREA: Computer Vision

KEYWORDS: GAN, decomposition, interpretability, semantic editing, latent directions, unsupervised, deep learning

ΠΕΡΙΛΗΨΗ

Τα Παραγωγικά Αντιπαλικά Δίκτυα (ΠΑΔ) είναι επί του παρόντος ένα απαραίτητο εργαλείο για σημασιολογική επεξεργασία εικόνας, που χρησιμοποιείται ευρέως σε μια πληθώρα εφαρμογών υπολογιστικής όρασης. Αν και αυτά τα μοντέλα αποδεδειγμένα κωδικοποιούν πλούσια σημασιολογική γνώση στις εσωτερικές τους αναπαραστάσεις, εξακολουθούν να μην έχουν έναν διαισθητικό τρόπο παροχής άμεσου ελέγχου στους χρήστες, προκειμένου να μπορούν να ασκήσουν επιρροή με συνέπεια στο περιεχόμενο της εικόνας εξόδου. Μόλις εξαχθεί αυτή η γνώση ωστόσο, μπορεί να μετατραπεί σε ερμηνεύσιμα από τον άνθρωπο στοιχεία ελέγχου για την αλλαγή των συνθετικών εικόνων με προβλέψιμο τρόπο.

Σε αυτήν την πτυχιακή εργασία, παρουσιάζουμε το MddGAN, μια τεχνική χωρίς επίβλεψη για την ανάλυση του λανθάνοντος χώρου του GAN και εξαγωγή διανυσματικών κατευθύνσεων που αντιστοιχούν σε σημαντικούς μετασχηματισμούς εικόνων. Σε αντίθεση με τις υπάρχοντες επιστημονικές εργασίες, εκτελούμε πολυγραμμική αποσύνθεση στα βάρη ενός προεκπαιδευμένου μοντέλου γεννήτριας και υποστηρίζουμε ότι ένα τέτοιο σχέδιο εξερεύνησης μπορεί να είναι περισσότερο κατάλληλο στην αποτύπωση των παραγόντων μεταβλητότητας που έμαθε το μοντέλο με λιγότερο μπέρδεμα. Περαιτέρω, η προτεινόμενη προσέγγιση μπορεί να χωρίσει μαθηματικά την ανακαλυφθείσα σημασιολογία σε ομάδες, ανάλογα με το σημασιολογικό τους περιεχόμενο. Αυτός ο διαχωρισμός γίνεται με εντελώς ανεπιτήρητο τρόπο και ουσιαστικά κάθε διάσταση της παραγόμενης πολυγραμμικής βάσης αντιπροσωπεύει μια τέτοια ομάδα.

Διεξάγοντας πολλά πειράματα σε GAN που έχουν εκπαιδευτεί σε διάφορα σύνολα δεδομένων, δείχνουμε πως μεταβάλλοντας τον αριθμό των επεξηγηματικών παραγόντων που ανακαλύπτονται στις γενετικές αναπαραστάσεις επηρεάζονται οι σημασιολογικοί χειρισμοί που ανακαλύφθηκαν. Επιπλέον, παρουσιάζουμε πολλές μη τετριμμένες κατευθύνσεις που επισημαίνουν τις δυνατότητες επεξεργασίας της μεθόδου μας. Επιπλέον, συγκρίνουμε το MddGAN με τη τρέχουσα μέθοδο αναφοράς με επίβλεψη και τρέχουσα μέθοδο αναφοράς χωρίς επίβλεψη τόσο ποιοτικά όσο και ποσοτικά. Τα αποτελέσματα δείχνουν ότι η προσέγγισή μας είναι τουλάχιστον εφάμιλλη με αυτές τις μεθόδους.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Όραση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: ΠΑΔ, αποσύνθεση, ερμηνευσιμότητα, σημασιολογική επεξεργασία, λανθάνουσες κατευθύνσεις, χωρίς επίβλεψη, βαθιά μάθηση

To my family.

ACKNOWLEDGEMENTS

I want to thank my supervisor, professor Yannis Panagakis, for his guidance and advice, that contributed towards the successful completion of this thesis. I also want to thank Yannis Siglidis, firstly, for being willing to help in whatever problem I was facing and secondly, for providing free access to AWS resources.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This thesis text was undertaken as a part of the Department of Informatics and Telecommunications undergraduate curriculum of the National and Kapodistrian University of Athens.

1. INTRODUCTION

Generative Adversarial Networks (GANs) are powerful Deep Learning-based image synthesis models, which have greatly impacted the computer vision community since their initial appearance back in 2014 [4]. Since then, many breakthroughs in the field have led to unprecedented high-resolution image generation. In fact, the current state-of-the-art GAN architectures [1, 13, 14] are able to produce images that look stunningly real and most of the time are indistinguishable from real ones. Today, due to their tremendous success, GANs are used widely in practical applications, e.g data augmentation, image super-resolution, image completion, image-to-image translation, video generation and many others.

Unfortunately, there is still limited understanding when it comes to investigating the generation process of such models. A rational question to ask would be whether we can somehow control the image synthesis to a point where we can completely determine how its image output will look like. In other words, we would like to use existing general-purpose image representations learnt from the model and discover techniques for controlling them. Interpreting and extending the capabilities of existing GANs is therefore an important open problem and as a result, it has justifiably received a lot of attention.

In particular, after [22] demonstrated that GANs can successfully support semantic arithmetic in the latent space, a line of research works attempts to exploits GANs by manipulating their internal representations for visual editing purposes. In essence, these methods attempt to discover meaningful directions in the GAN latent space and there is already enough evidence that while learning to synthesize images, GANs can spontaneously represent multiple human-interpretable concepts present in the image dataset used to train it [22, 25, 21, 27, 8, 26]. After identifying these directions, simple vector arithmetic in the latent space is capable of adjusting synthesized image attributes in surprising ways. Moreover, these directions also provide insight into how the GAN model operates. Ultimately, this makes GANs the dominant paradigm for controllable generation.

In the context of this thesis, we will not emphasize on existing supervised methods, since they usually require manual human labeling or additional pre-trained models. Instead, we will focus on approaches that discover interpretable latent space directions in a purely unsupervised fashion. In addition, we take inspiration from one such approach, [26], and propose our own variant, MddGAN, which performs a multilinear analysis on the parameters of a pre-trained GAN generator with the results looking rather promising. Our method doesn't simply uncover non-trivial vector directions, but it is also capable of grouping them according to the semantic concepts they encode.

2. RELATED WORK

2.1 Notation

Throughout this thesis, we will denote matrices with uppercase and vectors with lowercase boldface letters, e.g, $\boldsymbol{X}$ denotes a matrix and $\boldsymbol{x}$ denotes a vector. We also denote the i-th column of matrix $\boldsymbol{X}$ as $\boldsymbol{x}_i$. Tensors, which are the multidimensional versions of matrices are denoted by boldface calligraphic letters, e.g, $\boldsymbol{\mathcal{X}}$.

2.2 Generative Adversarial Networks

Generative adversarial networks (GANs) is an unsupervised machine learning framework simulating a minimax two-player game. In this framework, two players compete against each other until no player can further improve their respective objectives [4]. In the fundamental case, both players are multilayer perceptrons: a generative model G, commonly called the *generator*, that captures the data distribution, and a discriminative model D, commonly called the *discriminator* or *critic*, that estimates the probability that a sample came from the training data rather than G. Typically, the generator is of main interest - the discriminator gets discarded once the generator has been trained.

Consequently, the objective of G is to fool D by transforming some simple input distribution to a complex high-dimensional distribution (e.g over natural images), and the objective of D is to get better in distinguishing between real and generated data. Competition in this game drives both sides to improve their methods until the generated samples are practically indistinguishable from the genuine data samples and therefore D is unable to differentiate between the two. In game theory, this state is called Nash-equilibrium.

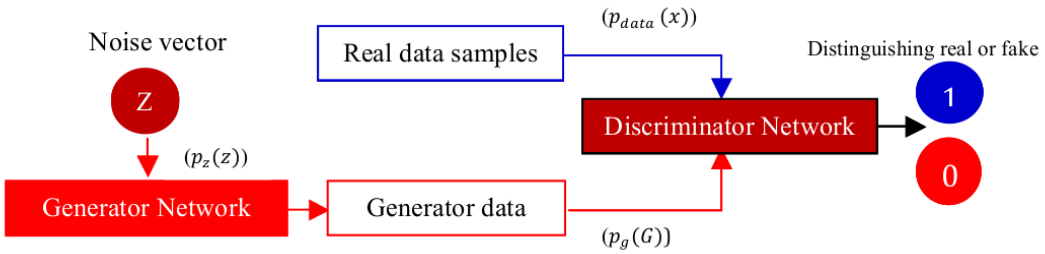


Figure 2.1: Design of the adversarial framework (source : [9]).

More formally, to learn the generator's distribution p_g over data $\boldsymbol{x}$, we define a random variable Z (e.g Gaussian) with a fixed distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$ from which we sample our input noise variables, then represent a mapping to data space as $G(\boldsymbol{z};\theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. We also define a second multilayer perceptron $D(\boldsymbol{x};\theta_d)$ that outputs a single scalar. $D(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ came from the data rather than p_g. Both G and D are trained together usually with stochastic gradient-based optimization methods, such as *stochastic gradient descent* (SGD). In practice, each training step involves alternating between performing one or more updates to D, while keeping G constant and performing one update to G, while keeping D constant. After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$, mean-

ing that the discriminator can no longer differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}, \forall \boldsymbol{x}$.

Accordingly, we train D to maximize the probability of assigning the correct label to examples from $p_{\text{data}}(\boldsymbol{x})$, which is the actual data distribution, hence:

$$\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]$$

where $\mathbb{E}$ denotes the expectation. Maximizing this term corresponds to D being able to accurately predict $D(\boldsymbol{x}) = 1$ when $\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$. The next term expresses the probability of the generator G tricking the discriminator D:

$$\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Intuitively, D wants to maximize this term, since $\log(x)$ with $0 < x < 1$ is negative, which ideally means that $D(G(\boldsymbol{z})) \approx 0$ and so G is not fooling D. The discriminator's goal becomes to ***maximize*** these two terms, thus we can define the following value function for D:

$$\max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2.1}$$

given G, which means that D properly classifies real and fake data samples.

On the contrary, we simultaneously train G to ***minimize*** $\log(1 - D(G(\boldsymbol{z})))$. In other words, G ideally wants to achieve $D(G(\boldsymbol{z})) \approx 1$ and thus trick D into believing the generated data are real. This intuition leads us to the following definition for the value function of G:

$$\min_G V(D, G) = \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2.2}$$

We can combine Eq. (2.1) and Eq. (2.2) into a single formula to obtain the loss function of this adversarial framework:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2.3}$$

For a more thorough explanation of why this framework works coupled with formal mathematical proofs, the reader should refer to the original work by Goodfellow et al. [4].

As an example, consider the common scenario where the synthesized data are images. In this case, the generator G learns a mapping from the d-dimensional latent space $\mathcal{Z} \subseteq \mathbb{R}^d$ to a higher dimensional image space $\mathcal{I} \subseteq \mathbb{R}^{H \times W \times C}$, as:

$$\begin{aligned} \boldsymbol{z} &\sim p(\boldsymbol{z}) \\ \boldsymbol{\mathcal{I}} &= G(\boldsymbol{z}) \end{aligned} \tag{2.4}$$

where $\boldsymbol{z} \in \mathcal{Z}$ and $\boldsymbol{\mathcal{I}} \in \mathcal{I}$ denote the input latent vector sampled from $p(\boldsymbol{z})$ and the produced output image respectively. On the other hand, the discriminator D receives as input an image $\boldsymbol{\mathcal{I}}'$, which can either be from the training dataset or generated by G, and outputs a scalar value in $[0, 1]$ which represents the probability of whether the input image is real or fake.

2.3 GAN Architectures

This part of Chapter 2 discusses some of the most important milestones in the GANs history.

2.3.1 Fully-Connected Architecture

GANs were initially introduced to the research community by Goodfellow et al. [4] (2014). In their work, they proposed the novel framework for training a generative and a discriminative model simultaneously via an adversarial process. When testing their method on MNIST [16] and the Toronto Face Database (TFD) datasets, both G and D were comprised of fully-connected layers, while for CIFAR-10 [15] they used a convolutional architecture for D and a "deconvolutional" architecture for G. Across all their experiments, the generator network used a mixture of rectified linear [2, 10] and sigmoid activations and the discriminator network used maxout [3] activations.

A crucial observation is that during training, Eq. (2.3) may not provide sufficient gradient for G to learn well, since in the early training stages G is poor and D rejects the generated samples with high confidence because they are clearly different from the training data. So, rather than training G to minimize $\log(1 - D(G(\boldsymbol{z})))$, they train G to maximize $\log D(G(\boldsymbol{z}))$. This slightly modified objective function indeed provides much stronger gradients early in learning.

Regarding its results, the original GAN framework produced images of poor quality, the majority of them being noisy and incomprehensible and also lacked decent generalization performance when the training process involved more complex image datasets.

2.3.2 Deep Convolutional Architecture

Radford et al. [22] (2016) proposed a family of Convolutional Neural Network (CNN) architectures that significantly stabilized GAN training across a range of datasets and allowed for training higher resolution and deeper generative models. After extensive model exploration, they ended up in a family of architecture guidelines called Deep Convolutional GANs (DCGANs).

Before proceeding to the significant contributions of this architecture-variant, we deem necessary to first give a brief introduction of Convolutional Neural Networks (CNNs).

2.3.2.1 Convolutional Neural Networks

Convolutional neural networks are designed with the explicit assumption that their inputs are 3-dimensional tensors, such as images. As a result, the layers of a CNN have neurons arranged in 3 dimensions : Height, Width and Depth. Each layer in the stack transforms an input 3D volume to an output 3D volume with some differentiable function and the final (output) layer of the network produces a single vector of class scores (arranged along the depth dimension).

CNN architectures are mainly built using three types of layers: *convolutional* layers, *pooling* layers and *fully-connected* layers. The weights of convolutional layers are in essence a set of learnable filters, also represented by 3-dimensional tensors $H \times W \times D$, where depth corresponds to the number of filters or *feature maps* used. The filters basically apply a convolution operation over the input volume. Pooling layers are periodically inserted in-between successive convolutional layers and their function is to progressively reduce the spatial size of the representation aiming at reducing the amount of weight parameters and computation in the network, and hence to also control overfitting. Lastly, fully-connected layers are connected to all activations in the preceding layer and hence their activations

can be computed with regular matrix multiplication. It is also common to apply an element-wise activation function on the output of each convolutional layer, which does not modify the dimensions of the produced volume. An example of such a function is $max(0, x)$ or Rectified Linear Unit (ReLU).

Most state-of-the-art CNN architectures stack a few convolutional-ReLU layers, followed by pooling layers, and repeat this pattern until the image has been merged spatially to a small size. At some point, it is common to transition to fully-connected layers. The last fully-connected layer holds the output, such as the class scores.

2.3.2.2 DCGAN

The authors chose a ”deconvolutional” architecture for G, which consists of fractional-strided convolutional layers, batch normalization layers and ReLU activations for all layers except the output, which uses the Tanh activation. A regular CNN architecture was used for D which is made up of strided convolutional layers, batch normalization layers and LeakyReLU activations. DCGAN was trained and evaluated on more complex datasets compared to the original GAN, such as Large-scale Scene Understanding (LSUN) [30], Imagenet-1k and Faces. The DCGAN generator used for LSUN can be viewed in Fig. 2.2

The ”deconvolutional” design of G can be viewed as the exact opposite of a standard CNN. Because the generator learns to map a low-dimensional latent vector z to a much larger image space, it tries to progressively upsample the input, done by the fractional-strided convolutions, and ultimately convert it to an RGB image. This is in contrast to what a regular CNN learns to produce, which is a vector of scalars given an image as input.

Another significant contribution of this paper was that it demonstrated that purely unsupervised models, such as GANs, could support simple vector arithmetic operations in order to produce meaningful image manipulations. In particular, the latent space $\mathcal{Z}$ of the generator can semantically evaluate expressions such as (”smiling woman” - ”neutral woman” + ”neutral man”) and will surprisingly produce images of a smiling man. Their experiments showed that such generations were in fact possible and could semantically obey the arithmetic.

DCGAN was proved to be a very important breakthrough for GAN literature managing to establish the ”deconvolution” as the main architecture used in G. Nevertheless, it was only successful on low-resolution (64x64 pixels) and less diverse images.

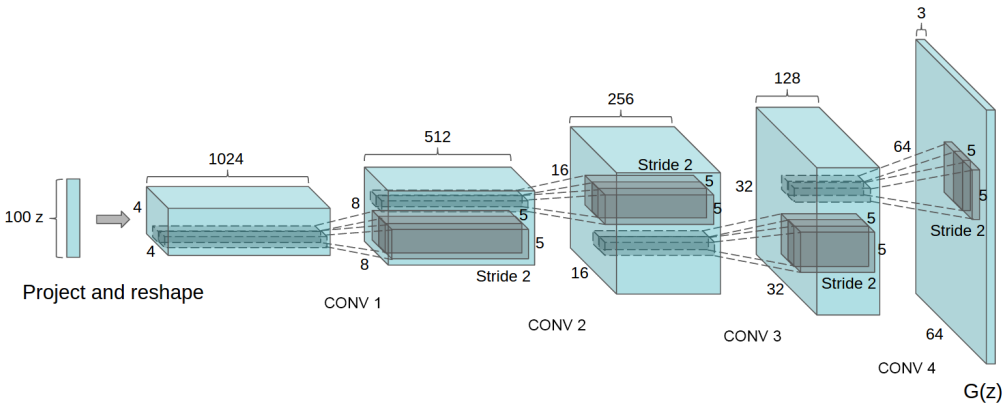


Figure 2.2: DCGAN generator used for LSUN scene modeling (source : [22]).

2.3.3 Progressive Growing Architecture

A huge improvement in GAN performance regarding image quality, image variety and stable training came from Karras et al. [12] (2017) by building upon DCGAN and incorporating the idea of *progressive networks* [24] into their approach. They key observation of their work was that the complex mapping from latent vectors to high-resolution images is easier to learn in steps.

In particular, in their training methodology both G and D start their training with low-resolution (4x4 pixels) images and then gradually increase the resolution by adding new layers to both networks, where each new layer basically doubles the resolution. The generator and the discriminator used are mirror images of each other and always grow in parallel. More specifically, both networks consist mainly of replicated 3-layer blocks, where each block operates on a specific spatial resolution $N \times N$, and each block uses two convolutional 3×3 layers together with an upsampling or downsampling layer. The authors also experiment with both Least-Squares GAN (LSGAN) [18] and the improved Wassertstein GAN (WGAN-GP) [5] loss functions instead of the original GAN loss of Eq. (2.3), and they find that both produce high-quality results, even though LSGAN is generally less stable. This process yields a novel progressive growing GAN (ProGAN) architecture that can reliably synthesize megapixel-scale sharp images (1024x1024 pixels).

Their contributions also include increased variation of the generated samples via an added minibatch standard deviation layer towards the end of the discriminator network, runtime scaling for the weights of each layer in both networks in order to maintain the same learning speed for all weights and pixel-wise normalization of the activation tensors after each convolutional layer in the generator.

ProGAN was trained on CELEBA-HQ, LSUN and CIFAR-10, achieving impressive image quality across all three datasets. To achieve these groundbreaking results, the authors trained ProGAN for 4 days in a compute cluster with 8 GPUs. Their method showcased that GANs can in fact approach convincing realism in image synthesis.

2.3.4 Style-Based Architecture

Although ProGAN generates high-quality images, its ability to control specific features of the synthesized output is minimal. To resolve this issue, Karras et al. [13] (2018) redesigned the architecture of the generator network to allow for scale-specific control of the image synthesis without compromising quality but instead substantially increasing it. This style-based generator borrowed ideas from style transfer literature and that's why it was called StyleGAN.

In reality, StyleGAN is an upgraded version of ProGAN, meaning that it still leverages the progressive growing training schedule, but the crucial modifications are targeted only for the generator, while the discriminator is left unchanged and the same exact architecture as ProGAN is used. The new architecture for G leads to automatic unsupervised separation of high-level attributes from stochastic variation in the generated images and it enables scale-specific control of the synthesis.

2.3.4.1 Generator Design

StyleGAN's generator architecture consists of 2 components: a non-linear mapping network f and a synthesis network g. The mapping network f is an 8-layer multilayer perceptron (MLP) and the synthesis network g is a model whose structure closely resembles that of the ProGAN generator, but with some important modifications.

Given a a latent code $\boldsymbol{z}$ in the input latent space $\mathcal{Z}$, the mapping network f embeds this latent code in an intermediate latent space $\mathcal{W}$ yielding a new latent code $\boldsymbol{w}$ with the same dimensionality. The authors prove that $\mathcal{W}$ is less entangled, since it is not restricted to following the probability density of the training data, such as $\mathcal{Z}$. Learned affine transformations then specialize $\boldsymbol{w}$ to produce *styles* $\boldsymbol{\mathcal{Y}} = (\boldsymbol{\mathcal{Y}}_s, \boldsymbol{\mathcal{Y}}_b)$ that control adaptive instance normalization (AdaIN) [7] operations after each convolution layer of the synthesis network g. The AdaIN operation is defined as

$$\text{AdaIN}\left(\boldsymbol{X}_i, \boldsymbol{\mathcal{Y}}\right) = \boldsymbol{Y}_{s,i} \frac{\boldsymbol{X}_i - \mu\left(\boldsymbol{X}_i\right)}{\sigma\left(\boldsymbol{X}_i\right)} + \boldsymbol{Y}_{b,i}$$

where if $\boldsymbol{\mathcal{X}}$ is a $H \times W \times fmaps$ activation tensor, $\boldsymbol{X}_i, \boldsymbol{Y}_{s,i}$ and $\boldsymbol{Y}_{b,i}$ all denote $H \times W$ matrices corresponding to the i-th feature map of $\boldsymbol{X}$, its scale and its bias respectively.

After the addition of the mapping network and AdaIN operations, the authors surprisingly observed that the synthesis network no longer benefits from feeding the latent code $\boldsymbol{z}$ into the first convolutional layer, and that it can instead produce meaningful results with the style vectors acting as an input only to intermediate layers. As a result, the traditional input layer of the generator is removed and instead the image synthesis is started from a learned $4 \times 4 \times 512$ constant input tensor $\boldsymbol{\mathcal{C}}$. So, the synthesis process in Eq. (2.4) is modified into:

$$\begin{aligned} \boldsymbol{z} &\sim p(\boldsymbol{z}) \\ \boldsymbol{w} &= f(\boldsymbol{z}) \\ \boldsymbol{\mathcal{I}} &= g\left(\boldsymbol{\mathcal{C}}, \boldsymbol{w}\right) \end{aligned} \tag{2.5}$$

A crucial detail is that during the generation process, $\boldsymbol{w}$ is broadcasted to all AdaIN layers of the synthesis network g, meaning that the input latent features are preserved across the entire network, but they control the strength of different image features at different scales. This broadcast operation is visually depicted in Fig. 2.3.

Other important modifications to the generator include random noise inputs after each convolution layer of the synthesis network, which helps the network to create stochastic variation, and a style mixing regularizer that makes the spatial layers of the generator more independent during training.

They evaluated their methods using CELEBA-HQ, FFHQ and LSUN. For CELEBA-HQ, the authors rely on WGAN-GP loss while FFHQ uses non-saturating loss with R_1 regularization [19, 23] for almost all model configurations.The authors trained StyleGAN for approximately 1 week on a compute cluster with 8 GPUs.

2.3.4.2 StyleGAN2

StyleGAN2 [14] comes with various improvements to image quality, efficiency, diversity, and disentanglement, and the results are incredibly improved. StyleGAN2 simply redesigns the normalization used in the generator of StyleGAN, which removes the artifacts such as blob-shaped artifacts that resemble water droplets. It also departs from the

progressive growing training schedule and instead uses a skip generator and a residual descriminator.

StyleGAN2 achieves even greater results in face image synthesis and quality compared to StyleGAN, and makes distinguishing between real and generated images an impossible task. One such example is shown in Fig. 2.4.

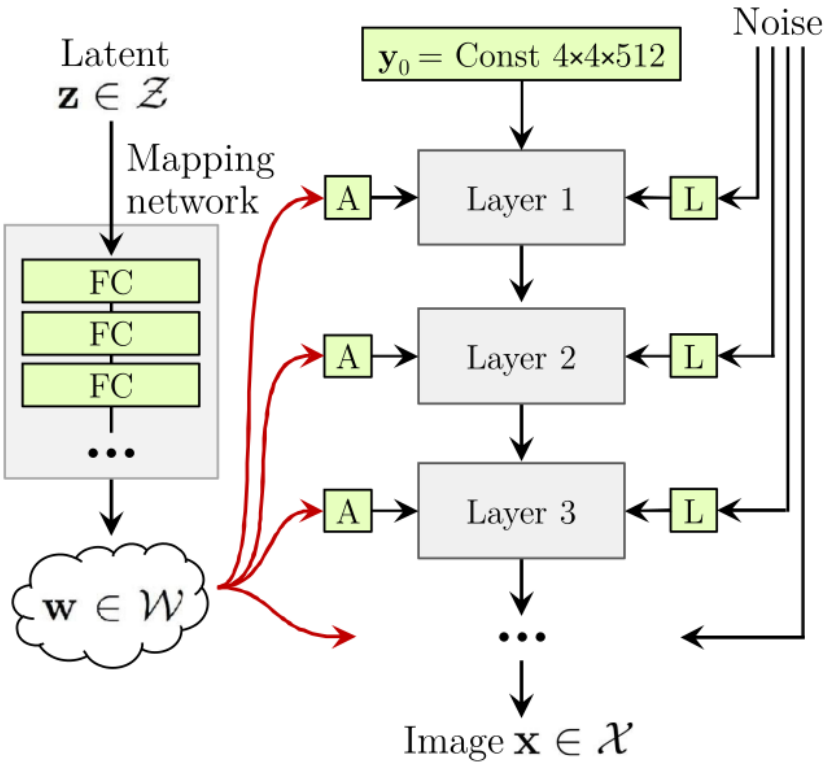


Figure 2.3: An overview of the style-based generator. The mapping network (left) embeds the starting latent code into a more disentangled intermediate space. Then, the disentangled latent code is broadcasted to all layers of the synthesis network (right) , which initiates the synthesis from a learnt constant (source : [8]).

Table 2.1: Comparison of GAN architectures presented in Section 2.3.
***Only the CIFAR-10 model**
****9 days for the FFHQ model and 13 days for the LSUN CAR model.**

		Architectures				
		GAN	DCGAN	ProGAN	StyleGAN	StyleGAN2
Features	Latent Space Dim.	100	100	512	512	512
	Max. Resolution	32^2	64^2	1024^2	1024^2	1024^2
	Deconv. G, Conv. D	Yes*	Yes	Yes	Yes	Yes
	Progressive Growing	No	No	Yes	Yes	No
	Loss	Eq. (2.3)	Eq. (2.3)	WGAN-GP	Eq. (2.3)	Eq. (2.3)
	Regularizer	No	No	No	R_1 [19]	PPL [14]
	# GPUS	1	1	8	8	8
	Training Time	$<$ 1day	$<$ 1 day	4 days	7 days	9/13 days**

(a) Synthesized Image (b) Real Image from FFHQ dataset

Figure 2.4: Hand-picked examples that demonstrate the quality of generated images of StyleGAN2.

2.4 Exploring the GAN Latent Space

GAN models typically rely on a single relatively low-dimensional noise vector, also called a *latent variable* (z), to parameterize their output. The *latent space* of these models is essentially a compact input space, whose dimensionality is smaller than that of the output and thus GANs are forced to be as efficient as possible with their internal representations and the compression of all the required information leads them to learn dataset-specific factors [27, 8, 21, 26, 20]. However, these implicit interpretable factors rarely lie on the canonical basis vectors of the input latent space, such as z_i, so additional training losses or control discovery steps are required.

Disentangling the internal representations learnt by a GAN model is a research problem that has gained considerable attention. A lot of scientific methods developed recently aim to extract directions of meaningful variation from the latent space of a pre-trained GAN generator and exploit them in a semantic editing procedure. The ultimate goal of these methods is to consistently control the generation process after identifying all the different factors of variation present on a specific training dataset that was used for training the GAN model of interest.

Concisely, a factor of variation corresponds to an image attribute that can be discerned consistently across a set of images, such as the pose or colour of objects. Ideally, we seek disentangled representations that encode independent factors of variation, which means that different factors correspond to distinct effects in the output. For instance, assuming human faces image data, one will be able to adjust gender, age, skin tone or hairstyle and even add accessories such as glasses or hats to the generated face images. The above can be achieved with simple vector arithmetic in the latent space, for example by adding or subtracting the semantic direction corresponding to glasses.

Given $\boldsymbol{z}, \boldsymbol{n} \in \mathbb{R}^d$, where $\boldsymbol{z}$ is a random latent vector and $\boldsymbol{n}$ is a direction encoding a specific semantic concept, the *editing* process used to produce the manipulated image can be

formulated as:

$$\begin{aligned} \boldsymbol{z}' &= \boldsymbol{z} + \varepsilon \boldsymbol{n} \\ I' &= G\left(\boldsymbol{z}'\right) \end{aligned} \tag{2.6}$$

where $\varepsilon \in [-k, k]$ is a scalar and $G : \mathbb{R}^d \rightarrow \mathbb{R}^{H \times W \times C}$. What the above process does is linearly shift the latent code $\boldsymbol{z}$ towards direction $\boldsymbol{n}$ with shift magnitude ε.

Broadly speaking, there are two types of approaches used to extract human-interpretable directions from the latent space of a GAN model and hence control the generation process:

1. **Supervised Methods** when used during model training, rely on labeled training data, and are used to enforce a structure in the model. In other words, the GAN model is told which aspects are considered important, often by providing explicit conditioning information for each training example. Since these techniques are applied during training, they are very time and resource intensive.

   When applied on a pre-trained model, they use supervision, such as conditioning **c** produced manually or with existing attribute detectors, to try to find factors of interest within the latent space $\mathcal{Z}$. In other words, these techniques aim to verify the hypothesis that the generator has learnt to model certain aspects on its own, thus relying on implicit interpretability.

2. **Unsupervised Methods** when used during model training, typically introduce special *loss functions* to enforce interpretable basis vectors $\boldsymbol{z_i}$ in the latent space $\mathcal{Z}$. However, the specific features that are learnt cannot be predetermined.

   When applied on a pre-trained model, these techniques directly analyze the most important factors of variation that have been learnt, and try to extract meaningful controls from these intrinsic factors. This is especially useful when working with highly abstract or novel datasets, where it might be impossible to predetermine which, or how many modes of variation are desirable or expected.

2.4.1 Supervised Methods

Even though supervised methods are not our primary focus in this thesis, we are going to briefly mention two that operate on pre-trained generator models.

Many supervised techniques rely on existing attribute detectors for extracting meaningful semantics from a an existing generative model. For instance, InterFaceGAN [25] employs attribute prediction models for five key face attributes, as well as 5-point facial landmarks used to accurately infer face pose. Their framework is based on the assumption that for any attribute that can be described with binary values, e.g male or female, there exists a hyperplane in the GAN latent space, which can accurately place all samples representing the same attribute on the same side.

Yang et al. [29] examine scene synthesis generative models and they also use several off-the-shelf classifiers to identify the emergent variation factors in the generative representations. In particular, these classifiers are treated as scoring functions, which basically assign to a synthesized scene image semantic scores corresponding to each candidate variation factor. Then, for a particular semantic concept, a decision boundary is learnt in the latent space by considering it as a binary classification task. Finally, they use a re-scoring technique to quantitatively verify the emergence of the semantic concepts of interest.

2.4.2 Unsupervised Methods

The methods presented in this Section are mainly applied to an existing pre-trained model and aim to identify the most important factors of variation that have been learnt. Our method MddGAN can be integrated in this family of techniques and in particular in those of Section 2.4.2.2.

2.4.2.1 Additional Training

Both of the techniques presented next require additional training either of the extra components (1) or of the generator network (2).

2.4.2.1.1 Extra Trainable Components Voynov and Babenko [27] were the first to propose an unsupervised approach for the discovery of semantically meaningful directions in the GAN latent space. Their method has two trainable components, the first being a matrix $\boldsymbol{A} \in \mathbb{R}^{d \times K}$, where d equals to the dimensionality of the latent space of G and K is the number of columns that also determines the number of directions the method will discover. Basically, the columns of $\boldsymbol{A}$ correspond to the discovered directions. The second trainable component is a *reconstructor* model R, which receives as input an image pair $(G(\boldsymbol{z}), G(\boldsymbol{z} + \boldsymbol{A}(\varepsilon \boldsymbol{e}_k)))$, where the first image is generated from a latent code $\boldsymbol{z} \sim \mathcal{N}(0, I)$, while the second one is generated from a *shifted* code $\boldsymbol{z} + \boldsymbol{A}(\varepsilon \boldsymbol{e}_k)$. Here $\boldsymbol{e}_k$ denotes the standard basis of $\mathbb{R}^K$ and acts as a column selector for the matrix $\boldsymbol{A}$ and ε is a scalar representing the magnitude of the shift towards direction k.

Since the second image is a transformation of the first one, the reconstructor's goal is to reproduce the shift in the latent space. More specifically, R produces 2 outputs $R(I_1, I_2) = (\widehat{k}, \widehat{\varepsilon})$, where $\hat{k}$ is a prediction of a direction index $k \in \{1, \ldots, K\}$, and $\hat{\varepsilon}$ is a prediction of a shift magnitude ε. The above can be expressed more formally as a mapping

$$R : (I_1, I_2) \longrightarrow (\{1, \ldots, K\}, \mathbb{R}).$$

Learning is performed via minimizing the following loss function:

$$\min_{A,R} \mathop{\mathbb{E}}_{\boldsymbol{z},k,\varepsilon} L(A, R) = \min_{A,R} \mathop{\mathbb{E}}_{\boldsymbol{z},k,\varepsilon} \left[L_{cl}(k, \widehat{k}) + \lambda L_r(\varepsilon, \widehat{\varepsilon}) \right]$$

where the authors used the cross-entropy loss function for the classification term $L_{cl}(\cdot, \cdot)$ and the mean absolute error loss function for the regression term $L_r(\cdot, \cdot)$. In their experiments the weight coefficient λ was always set to 0.25. We should note that G is a non-trainable component of their method, and its parameters do not change during learning.

In brief, this joint optimization process seeks to obtain such columns of $\boldsymbol{A}$ that the corresponding image transformations are easier to distinguish from each other, to make the classification problem for the reconstructor simpler. Indeed, the discovered directions do not interfere, meaning that each one affects only a single factor of variation and are easy-to-interpret. The authors experiment with different datasets and generator architectures and showcase several human-interpretable and practically important directions, such as background removal, skin tone, presence of eyeglasses, luminance and so on.

2.4.2.1.2 Derivative-Based Loss Function Latent space disentanglement can also be enforced by introducing derivative-based regularizers in generative models as demonstrated by Peebles et al. [21]. In their work they propose a regularization term that encourages the Hessian of a generative model with respect to its input to be diagonal.

To provide intuition for their method, they use the example of a scalar-valued function $G : \mathbb{R}^d \to \mathbb{R}$, where d denotes the dimensionality of the $\mathcal{Z}$ latent space. Mathematically, to describe how each z_i component changes the output image we can use the derivative $\frac{\partial G}{\partial z_i}$. To disentangle G with respect to $\boldsymbol{z}$ would mean that by varying z_i , the produced change in the output of G should be mostly independent of the other components $z_{j\neq i}$. To measure this independency, we use another derivative, this time with respect to z_j: $\frac{\partial}{\partial z_j}\left(\frac{\partial G}{\partial z_i}\right)$. If this value was large, the effect z_i has on the output is strongly dictated by z_j and if not, then z_j has no effect on how perturbing z_i will change G 's output. Thus, we would like this value to be small. Now, all the pair-wise interdependencies between the $\boldsymbol{z}$ components are contained in G 's Hessian matrix of second derivatives, $\boldsymbol{H} \in \mathbb{R}^{d\times d}$, and assuming $d = 3$, the Hessian matrix is:

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 G}{\partial z_0 \partial z_0} & \frac{\partial^2 G}{\partial z_0 \partial z_1} & \frac{\partial^2 G}{\partial z_0 \partial z_2} \\ \frac{\partial^2 G}{\partial z_1 \partial z_0} & \frac{\partial^2 G}{\partial z_1 \partial z_1} & \frac{\partial^2 G}{\partial z_1 \partial z_2} \\ \frac{\partial^2 G}{\partial z_2 \partial z_0} & \frac{\partial^2 G}{\partial z_2 \partial z_1} & \frac{\partial^2 G}{\partial z_2 \partial z_2} \end{bmatrix}$$

By minimizing these pair-wise interdependencies, equivalently regularizing the Hessian matrix to be diagonal we can achieve exactly what we wanted

$$\underset{j\neq i}{H_{ij}} = \frac{\partial^2 G}{\partial z_i \partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{\partial G}{\partial z_i}\right) = 0$$

In practice, this is done by simply minimizing the sum of squared off-diagonal terms

$$\mathcal{L}_H(G) = \sum_{i=1}^{d}\sum_{j\neq i}^{d} H_{ij}^2 \tag{2.7}$$

Because generative models are clearly not scalar-valued functions, Equation 2.7 with a slight modification can be extended to vector-valued functions such as GANs.

Computing Hessian matrices during training is slow when d is large but luckily, equation 2.7 can be rewritten to be the variance of the second directional derivatives using Hutchinson's estimator:

$$\mathcal{L}_H(G) = \mathsf{Var}_v\left(v^T H v\right)$$

where the v vectors are random Rademacher vectors

$$\mathsf{Pr}\left(v_i = 1\right) = \mathsf{Pr}\left(v_i = -1\right) = \frac{1}{2}$$

meaning that each entry has equal probability of being -1 or $+1$, and $v^T H v$ is the second directional derivative of G in the direction v times $|v|$. The second directional derivatives can be quickly approximated with finite differences. Thus, we land on the following equality:

$$\mathcal{L}_H(G) = \sum_{i=1}^{d}\sum_{j\neq i}^{d} H_{ij}^2 \approx \frac{1}{2}\mathsf{Var}_v\left[\frac{G(z+\epsilon v) - 2G(z) + G(z-\epsilon v)}{\epsilon^2}\right]$$

where $\varepsilon > 0$ is a hyperparameter that controls the granularity of the second directional derivative estimate.

During training, the discriminator's loss function remained unchanged (see (2.1)) but the generator's loss, instead of (2.2), was modified to be:

$$\mathcal{L}_{\mathsf{G}} = \underbrace{\mathop{\mathbb{E}}_{z\sim p_z(z)}[f(1 - D(G(z)))]}_{\text{Standard Adversarial Loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{z\sim p_z(z)}[\mathcal{L}_{\mathsf{H}}(G)]}_{\text{The Hessian Penalty}}$$

where f denotes the GAN loss function used and weight λ balances the two terms. Interestingly, the authors point out that fine-tuning a pre-trained GAN with the above loss in many cases tends to work as well as or better than training from scratch with the Hessian Penalty.

In their experiments, they show that training with the Hessian Penalty causes axis-aligned disentanglement to emerge in latent space when applied to ProGAN on several datasets. In addition, they observe that the Hessian Penalty "turns-off" unneeded z components when a generator's latent space is overparameterized. Moreover, similar to [27], the Hessian Penalty can be extended to identify interpretable directions in a pre-trained generator's latent space i.e BigGAN, but this time G 's weights are kept fixed throughout training.

2.4.2.2 Direct Analysis

The next two approaches do not involve additional neural network (model) training, yet they exhibit results directly comparable to the methods described above. More specifically, they directly perform statistical analysis (e.g through Principal Component Analysis - PCA) of the input latent or feature space.

2.4.2.2.1 Data Sampling Analysis Harkonen et al. [8] hypothesize that the internal activations within the generator form abstract spaces with informative shapes and the principal components of these activation tensors on the early internal layers of GANs represent important factors of variation. They verify their hypothesis on two models: StyleGAN and BigGAN.

For StyleGAN, they analyze the intermediate latent space $\mathcal{W}$ directly by identifying the principal axes of $p(\boldsymbol{w})$. To do so, they sample N random latent vectors $\boldsymbol{z}_{1:N}$, and compute the corresponding $\boldsymbol{w}_i = M(\boldsymbol{z}_i)$ values in the intermediate latent space. They then compute PCA of these $\boldsymbol{w}_{1:N}$ values, which gives a low-rank basis $\boldsymbol{V}$ for $\mathcal{W}$. Since this $\mathcal{W}$ space is proven to admit nicer disentanglement properties, the PCA basis can be used to directly modify the output image by modifying its representation in $\mathcal{W}$. Concretely, given an image defined by $\boldsymbol{w}$, we can edit this image by varying PCA coordinates $\boldsymbol{x}$ directly, as in (2.5), (2.6):

$$\boldsymbol{w}' = \boldsymbol{w} + \boldsymbol{V}\boldsymbol{x}$$
$$\boldsymbol{\mathcal{I}}' = G(\boldsymbol{\mathcal{C}}, \boldsymbol{w}')$$

where each entry x_k of $\boldsymbol{x}$ is a separate control parameter. The entries x_k are initially zero until modified by a user. In this way, they are able to cause interesting modifications to the output image, even though some entanglement can be observed in the produced effects.

In BigGAN, a learned intermediate latent space similar to StyleGAN's $\mathcal{W}$ does not exist, so PCA is now applied at an intermediate network layer i and then the discovered directions are transferred back to $\mathcal{Z}$ latent space , as follows. Similar to before, N random latent vectors $\boldsymbol{z}_{1:N}$ are sampled, processed through the model and they produce N activation

tensors $\boldsymbol{y}_{1:N}$ at the i-th layer, where $\boldsymbol{y}_j = \hat{G}_i(\boldsymbol{z}_j)$. Next, PCA is applied on these N activation tensors, which produces a low-rank basis matrix $\boldsymbol{V}$, and the data mean μ. The PCA coordinates $\boldsymbol{x}_j$ of each activation are then computed by projection: $\boldsymbol{x}_j = \boldsymbol{V}^T(\boldsymbol{y}_j - \mu)$. This basis is then transferred to latent space by linear regression, as follows. They start with an individual basis vector $\boldsymbol{v}_k$ (i.e., a column of $\boldsymbol{V}$), and the corresponding PCA coordinates $x^k_{1:N}$, where x^k_j is the scalar k-th coordinate of $\boldsymbol{x}_j$. They solve for the corresponding latent basis vector $\boldsymbol{u}_k$ as:

$$\boldsymbol{u}_k = \arg\min \sum_j \left\| \boldsymbol{u}_k x^k_j - \boldsymbol{z}_j \right\|^2$$

to identify a latent direction corresponding to this principal component. Equivalently, the whole basis is computed simultaneously with:

$$\boldsymbol{U} = \arg\min \sum_j \left\| \boldsymbol{U}\boldsymbol{x}_j - \boldsymbol{z}_j \right\|^2$$

using a standard least-squares solver, without any additional orthogonality constraints. Each column of $\boldsymbol{U}$ then aligns to the variation along the corresponding column of $\boldsymbol{V}$. A new set of N latent vectors is used for the regression task. The individual dimensions x_k each correspond to different edits, many of which are easily interpretable. Given a new image with latent coordinates $\boldsymbol{z}$, edits may then be made by varying the coordinates of $\boldsymbol{x}$, as in (2.6):

$$\boldsymbol{z}' = \boldsymbol{z} + \boldsymbol{U}\boldsymbol{x}$$
$$\boldsymbol{\mathcal{I}}' = G(\boldsymbol{z}')$$

where $\boldsymbol{x}$ is initially a zero vector.

Analyzing their findings, the authors offer some surprising insights into GAN and PCA properties. For example, it is evident that dataset properties are reflected in the principal directions, emphasizing, not suprisingly, the importance of the training data. They also show that certain edits have an effect that is dependent on the starting image, for instance a ”beard” edit only modifies male faces, whereas the ”lipstick” edit only modifies female faces, indicating that the model is imitating biases learned from the training data. Furthermore, across all trained models they tested, large-scale changes to geometric shape and viewpoint of an image are limited to the first 20 principal components; successive components leave layout unchanged, and instead control object appearance or background and details.

2.4.2.2.2 Trained Weights Analysis Perhaps, the most straight forward approach is the one from Shen and Zhou [26]. In their work they propose a closed form factorization algorithm for discovering latent semantics by directly decomposing the weights of a pre-trained GAN generator, not requiring data sampling or model training. Given a latent vector $\boldsymbol{z} \in \mathcal{Z} \subseteq \mathbb{R}^d$, the first step of the generation process of GANs typically involves passing $\boldsymbol{z}$ through a fully-connected layer, and yielding a projected vector of increased dimensionality, which is then separately reshaped into a $H \times W \times fmaps$ tensor. More specifically, the fully-connected layer applies the following (affine) transformation on $\boldsymbol{z}$:

$$G_1(\boldsymbol{z}) = \boldsymbol{y} = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{b} \tag{2.8}$$

where $\boldsymbol{y} \in \mathbb{R}^m$ is the m-dimensional projected code and $d \leq m$. $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{b} \in \mathbb{R}^m$ denote the weight and bias used in the first transformation step $G_1(\cdot)$ respectively.

After identifying a semantically meaningful direction $\boldsymbol{n} \in \mathbb{R}^d$, and under the formulation of the affine transformation in (2.8), the manipulation model in Equation 2.6 can be simplified as

$$\begin{aligned} \boldsymbol{y}' = G_1(\boldsymbol{z}') &= G_1(\boldsymbol{z} + \alpha \boldsymbol{n}) \\ &= \boldsymbol{A}\boldsymbol{z} + \boldsymbol{b} + \alpha \boldsymbol{A}\boldsymbol{n} = \boldsymbol{y} + \alpha \boldsymbol{A}\boldsymbol{n} \end{aligned} \tag{2.9}$$

From Equation 2.9 we observe that given any latent code $\boldsymbol{z}$ together with a certain latent direction $\boldsymbol{n}$, the editing can be always achieved by adding the term $\alpha \boldsymbol{A}\boldsymbol{n}$ onto the projected code after the first step. Assuming the weight parameter $\boldsymbol{A}$ contains all the essential knowledge of the image variation, important latent directions can be extracted by decomposing $\boldsymbol{A}$. By solving the following optimization problem

$$\boldsymbol{n}^* = \underset{\left\{\boldsymbol{n} \in \mathbb{R}^d : \boldsymbol{n}^T \boldsymbol{n} = 1\right\}}{\arg\max} \|\boldsymbol{A}\boldsymbol{n}\|_2^2 \tag{2.10}$$

we can uncover directions that are capable of causing large variations after the projection of $\boldsymbol{A}$. In a scenario where we seek the k most important directions $\{\boldsymbol{n}_1, \boldsymbol{n}_2, \ldots, \boldsymbol{n}_k\}$ instead of just one, Equation 2.10 can be expanded into

$$\boldsymbol{N}^* = \underset{\left\{\boldsymbol{N} \in \mathbb{R}^{d \times k} : \boldsymbol{n}_i^T \boldsymbol{n}_i = 1 \forall i = 1, \cdots, k\right\}}{\arg\max} \sum_{i=1}^{k} \|\boldsymbol{A}\boldsymbol{n}_i\|_2^2 \tag{2.11}$$

where $\boldsymbol{N} = [\boldsymbol{n}_1, \boldsymbol{n}_2, \ldots, \boldsymbol{n}_k]$ correspond to the top-k semantics. Introducing the Lagrange multipliers $\{\lambda\}_{i=1}^k$ into Equation 2.11 we get

$$\begin{aligned} \boldsymbol{N}^* &= \underset{\boldsymbol{N} \in \mathbb{R}^{d \times k}}{\arg\max} \sum_{i=1}^{k} \|\boldsymbol{A}\boldsymbol{n}_i\|_2^2 - \sum_{i=1}^{k} \lambda_i \left(\boldsymbol{n}_i^T \boldsymbol{n}_i - 1\right) \\ &= \underset{\boldsymbol{N} \in \mathbb{R}^{d \times k}}{\arg\max} \sum_{i=1}^{k} \left(\boldsymbol{n}_i^T \boldsymbol{A}^T \boldsymbol{A}\boldsymbol{n}_i - \lambda_i \boldsymbol{n}_i^T \boldsymbol{n}_i + \lambda_i\right) \end{aligned} \tag{2.12}$$

and by taking the partial derivative on each $\boldsymbol{n}_i$, we have

$$2\boldsymbol{A}^T \boldsymbol{A}\boldsymbol{n}_i - 2\lambda_i \boldsymbol{n}_i = 0 \tag{2.13}$$

All possible solutions to Eq. 2.13 should be eigenvectors of the matrix $\boldsymbol{A}^T \boldsymbol{A}$. To get the maximum objective value and make $\{\mathrm{n}_i\}_{i=1}^k$ distinguishable from each other, we choose columns of $\boldsymbol{N}$ as the eigenvectors of $\boldsymbol{A}^T \boldsymbol{A}$ associated with the k largest eigenvalues.

They apply their method to the state-of-the-art GAN models trained on a variety of datasets and their technique manages to discover versatile semantics in each scenario. They also compare it with existing supervised and unsupervised alternatives and the results clearly highlight its potential on semantic image editing.

3. METHOD

This section describes MddGAN, our new technique for analyzing the internal representations of an existing pre-trained GAN through an unsupervised edit discovery scheme. The proposed method is a variant of [26] and thus does not require costly generator retraining nor data sampling. Where our methods diverge is that we perform a *multilinear* decomposition on the weights of the pre-trained generator in an attempt to uncover semantically meaningful vector directions that can be further organized in categories. Such a separation is currently not attainable by SeFa or any other unsupervised approach. For the following sections, we will exclusively focus on the task of image synthesis. We will also use the terms *direction* and *semantic* interchangeably when we want to refer to $\boldsymbol{n}$ from Eq. (2.6).

3.1 Unsupervised Multilinear Matrix Decomposition

Principal Component Analysis (PCA) and the closely related Singular Value Decomposition (SVD) are probably the most popular statistical methods to find a single mode of variation that explains the data. However, visual data tend to have many different and possibly independent modes of variation and hence dimensionality reduction techniques such as PCA are not able to fully disentangle these factors. To tackle this problem, Wang et al. [28] proposed a novel unsupervised multilinear decomposition of matrices which uncovers the potential multilinear structure of incomplete sets of data and the corresponding low-dimensional latent variables (coefficients) explaining different types of variation.

3.1.1 Preliminaries

This section requires the introduction of additional notation. The *mode-m matricization* of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots I_M}$ maps $\boldsymbol{\mathcal{X}}$ to a matrix $\boldsymbol{X}_{(m)} \in \mathbb{R}^{I_m \times \bar{I}_m}$ with $\bar{I}_m = \prod_{\substack{k=1 \\ k \neq m}}^{M} I_k$ such that the tensor element $x_{i_1, i_2, \ldots, i_M}$ is mapped to the matrix element $x_{i_m, j}$ where $j = 1 + \sum_{\substack{k=1 \\ k \neq m}}^{M} (i_k - 1) J_k$ with $J_k = \prod_{\substack{n=1 \\ n \neq m}}^{k-1} I_n$.

The *mode-m vector product* of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_M}$ with a vector $\boldsymbol{x} \in \mathbb{R}^{I_m}$, denoted by $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{x} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$. The result is of order $M - 1$ and is defined element-wise as

$$(\boldsymbol{\mathcal{X}} \times_m \boldsymbol{x})_{i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_M} = \sum_{i_m = 1}^{I_m} x_{i_1, i_2, \ldots, i_M} x_{i_m}.$$

In order to simplify the notation, we denote $\boldsymbol{\mathcal{X}} \times_1 \boldsymbol{x}^{(1)} \times_2 \boldsymbol{x}^{(2)} \times_3 \ldots \times_M \boldsymbol{x}^{(M)} = \boldsymbol{\mathcal{X}} \prod_{m=1}^{M} \times_m \boldsymbol{x}^{(m)}$.

The *Khatri-Rao* (column-wise Kronecker product) product of matrices $\boldsymbol{A} \in \mathbb{R}^{I \times N}$ and $\boldsymbol{B} \in \mathbb{R}^{J \times N}$ is denoted by $\boldsymbol{A} \odot \boldsymbol{B}$ and yields a matrix of dimensions $(IJ) \times N$. Furthermore, the Khatri-Rao of a set of matrices $\left\{ \boldsymbol{X}^{(m)} \in \mathbb{R}^{I_m \times N} \right\}_{m=1}^{N}$ is denoted by $\boldsymbol{X}^{(1)} \odot \boldsymbol{X}^{(2)} \odot \cdots \odot \boldsymbol{X}^{(M)} \doteq \bigodot_{m=1}^{M} \boldsymbol{X}^{(m)}$.

3.1.2 Basic Model

Their method assumes a given input matrix of observations $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{d \times N}$ where each of the N columns represents a vectorized image $\boldsymbol{x}_i$ of d pixels. In order to discover $M-1$ different modes of variation, the proposed decomposition for a single sample is:

$$\boldsymbol{x}_i = \mathcal{B} \times_2 \boldsymbol{a}_i^{(2)} \times_3 \boldsymbol{a}_i^{(3)} \cdots \times_m \boldsymbol{a}_i^{(M)} = \mathcal{B} \prod_{m=2}^{M} \times_m \mathbf{a}_i^{(m)}, \tag{3.1}$$

where $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{d \times K_2 \times K_3 \times \ldots \times K_M}$ represents the common multilinear basis of $\boldsymbol{X}$ and the set of vectors $\{\boldsymbol{a}_i^{(m)} \in \mathbb{R}^{K_m}\}_{m=2}^{M}$ represents the variation coefficients in each mode specific to the vectorized image $\boldsymbol{x}_i$. The value specified for each dimension K_m, where $m \in [2, M]$, indicates the number of different occurrences of this variability factor in the input data. For example, if lighting is among the modes of variation our input data consists of and there are 5 different lighting conditions, then the value of the corresponding dimension can be set to 5, i.e $K_{lighting} = 5$. Together the values $K_2, K_3, \ldots, K_M$ specify the order of the multilinear basis $\boldsymbol{\mathcal{B}}$.

Therefore, for the observation matrix $\boldsymbol{X}$, the above decomposition can be written in matrix form as:

$$\boldsymbol{X} = \boldsymbol{B}_{(1)} \left(\boldsymbol{A}^{(2)} \odot \boldsymbol{A}^{(3)} \cdots \odot \boldsymbol{A}^{(M)} \right) = \boldsymbol{B}_{(1)} \left(\bigodot_{m=2}^{M} \mathsf{A}^{(m)} \right), \tag{3.2}$$

where $\boldsymbol{B}_{(1)} \in \mathbb{R}^{d \times K_2 \cdot K_3 \cdot \ldots \cdot K_M}$ is the mode-1 matricization of $\boldsymbol{\mathcal{B}}$ and $\{\boldsymbol{A}^{(m)} \in \mathbb{R}^{K_m \times N}\}_{m=2}^{M}$ gathers the variation coefficients for all images across $M-1$ modes of variation.

To find the unknown multilinear basis $\boldsymbol{\mathcal{B}}$ and the variation coefficients $\{\boldsymbol{A}^{(m)}\}_{m=2}^{M}$, the proposed optimization problem to solve is:

$$\underset{\boldsymbol{B}_{(1)}, \{\boldsymbol{A}^{(m)}\}_{m=2}^{M}}{\arg\min} \quad \|\boldsymbol{X} - \boldsymbol{B}_{(1)} (\bigodot_{m=2}^{M} \mathsf{A}^{(m)})\|_F^2 \text{ s.t. } \boldsymbol{B}_{(1)}^T \boldsymbol{B}_{(1)} = \mathsf{I}. \tag{3.3}$$

The procedure of solving Eq. (3.3) is summarized in Algorithm 1.

3.1.3 Properties

In line 4, the i-th column of $\boldsymbol{Q}[t]$, $\boldsymbol{q}_i[t]$, is used to construct tensor $\boldsymbol{\mathcal{Q}_i} \in \mathbb{R}^{K_M \times K_{M-1} \times \ldots \times K_2}$ and since $\boldsymbol{Q}[t]$ is a $d \times N$ matrix $\forall t$, $\boldsymbol{q}_i[t]$ will be a d-dimensional vector $\forall t$ and therefore

$$K_M \cdot K_{M-1} \cdot \ldots \cdot K_2 = d. \tag{3.4}$$

In other words, the product of the input dimensions must be equal to the dimensionality d of each input data sample $\boldsymbol{x}_i$.

Algorithm 1 terminates when the convergence condition in line 13 is satisfied, and produces a $d \times d$ matrix $\boldsymbol{B}_{(1)}$ than can be optionally tensorized into $\boldsymbol{\mathcal{B}}$ (line 16). If we seek a sparser basis of $\boldsymbol{X}$, Algorithm 1 in the general case can output a $l \times d$ matrix $\boldsymbol{B}_{(1)}$ where $l \leq d$. This reduced representation can be obtained by replacing the full SVD by its truncated version in lines 1 and 10. Truncated SVD basically calculates only the l column vectors of $\boldsymbol{U}$ and the l row vectors of $\boldsymbol{V}$ corresponding to the l largest singular values of $\boldsymbol{\Sigma}$. Of course, using the truncated SVD no longer leads to the exact decomposition of the input matrix. Instead we acquire the closest approximation to it that can be achieved by

Algorithm 1: Multilinear Data Decomposition Algorithm

Input : Data matrix $\boldsymbol{X} \in \mathbb{R}^{d \times N}$, dimensions $K_2, K_3, \ldots, K_M$
Output: $\boldsymbol{\mathcal{B}}, \boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}, \ldots, \boldsymbol{A}^{(M)}$

1 Initialization: $t \leftarrow 0$, $[\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}] \leftarrow SVD(\boldsymbol{X})$, $\boldsymbol{B}_{(1)}[0] = \boldsymbol{U}\sqrt{\boldsymbol{\Sigma}}$, $\boldsymbol{Q}[0] = \sqrt{\boldsymbol{\Sigma}}\boldsymbol{V}^T$
2 **while** *not converged* **do**
3 **forall** *image* $i = 1 \ldots N$ **do**
4 construct $\boldsymbol{\mathcal{Q}_i} \in \mathbb{R}^{K_M \times K_{M-1} \times \cdots \times K_2}$ from $\boldsymbol{q}_i[t]$ ▷ $K_M \cdot K_{M-1} \cdot \ldots \cdot K_2 = d$
5 $[\boldsymbol{S}_i, \boldsymbol{U}_i] \rightarrow HOSVD(\boldsymbol{\mathcal{Q}_i})$
6 **foreach** *mode* $m = 2, \ldots, M$ **do**
7 $\boldsymbol{a}_i^{(m)}[t+1] = (\boldsymbol{S}_i)_1^{(M-m+1)}$
8 **end**
9 **end**
10 $[\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}] \leftarrow SVD\left(\boldsymbol{X}\left(\bigodot_{m=2}^{M} \boldsymbol{A}^{(m)}[t]\right)^T\right)$
11 $\boldsymbol{B}_{(1)}[t+1] = \boldsymbol{U}\boldsymbol{V}^T$
12 $\boldsymbol{Q}[t+1] = \boldsymbol{B}_{(1)}[t+1]^T\boldsymbol{X}$ ▷ $\boldsymbol{Q}[t+1] \in \mathbb{R}^{d \times N}$
13 Check convergence condition: $\frac{\|\boldsymbol{X} - \boldsymbol{B}_{(1)}[t+1]\boldsymbol{Q}[t+1]\|_F^2}{\|\boldsymbol{X}\|_F^2} < \epsilon$
14 $t \leftarrow t + 1$
15 **end**
16 Tensorize $\boldsymbol{B}_{(1)}$ into $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{d \times K_2 \times \cdots \times K_M}$

a rank l matrix. In this case, $\boldsymbol{Q}[t]$ calculated in lines 1 and 12 is a $l \times N$ matrix $\forall t$, and as such

$$K_M \cdot K_{M-1} \cdot \ldots \cdot K_2 = l, \text{ with } l \leq d \tag{3.5}$$

Another important note is that the number of dimensions $M - 1$, or equivalently the number of different modes of variation in the data is assumed to be known in advance and by varying this number we are basically exchanging reconstruction detail of the data with the ability of the decomposition to sufficiently separate the variation factors. For instance, consider an input data matrix $\boldsymbol{X} \in \mathbb{R}^{d \times N}$ where each column $\boldsymbol{x}_i$ represents a "flattened" face image. We know in advance that the only factors of variation present in this dataset are expression and identity and more specifically the dataset consists of 10 facial expressions and 200 identities amounting to 2000 images. The decomposition in Eq. (3.2) becomes in this case:

$$\boldsymbol{X} = \boldsymbol{B}_{(1)}(\boldsymbol{A}^{(2)} \odot \boldsymbol{A}^{(3)})$$

where $\boldsymbol{B}_{(1)} \in \mathbb{R}^{d \times K_2 \cdot K_3}$ is the orthogonal mode-1 matricization of tensor $\boldsymbol{\mathcal{B}}$, $\boldsymbol{A}^{(2)} \in \mathbb{R}^{K_2 \times N}$ is the matrix of expression coefficients and $\boldsymbol{A}^{(3)} \in \mathbb{R}^{K_3 \times N}$ is the matrix of identity coefficients. In order to sufficiently separate the two factors, K_2 should be set to the approximate number of differing expressions in the data and equivalently K_3 should be set to the approximate number of differing identities in the data. By setting $K_2 = 10$ and $K_3 = 50$, we apply the decomposition to discover $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{d \times K_2 \times K_3}$ becomes a basis of expression and identity and for example, $\pm\boldsymbol{\mathcal{B}}_{:\,i:}$ are bases corresponding to expressions in the dataset.

3.2 Constructing the Weight Matrix

Current state-of-the-art GAN models typically adopt convolutional neural networks with multiple hidden layers as the generator architecture (Table 2.1). Consequently, the process of generating images by a GAN generator described in Eq. (2.4) can be equivalently viewed as a sequence of consecutive generation steps, each one performed by an intermediate layer in the hierarchy structure. Consider the case where the generator model consists of L intermediate layers $G_1 \dots G_L$. The first (input) layer takes the latent vector as input and produces a set of activations $\boldsymbol{\mathcal{Y}}_1 = G_1(\boldsymbol{z})$, known as activation tensor. The remaining layers each produce activations as a function of the previous layer's output

$$\boldsymbol{\mathcal{Y}}_i = \hat{G}_i(\boldsymbol{z}) = G_i(\boldsymbol{\mathcal{Y}}_{i-1})$$

The output of the last layer $\boldsymbol{\mathcal{I}} = G_L(\mathcal{Y}_{L-1})$ is an RGB image.

In the first step of this generation process, the generator learns to project the input latent space $\mathcal{Z}$ to an intermediate activation space, which is then used as the input to succeeding convolutional layers that start the actual synthesis. In other words, it acts as the stepping stone to shaping the uninformative latent space into a meaningful output distribution and prior work has proven that it can represent important factors of variation. In particular, this first generation step that we will focus on can be formulated as an affine transformation (Eq. (2.8)) :

$$G_1(\boldsymbol{z}) = \boldsymbol{y} = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{b}$$

As in [26], we first need to construct weight matrix $\boldsymbol{A} \in \mathbb{R}^{N \times d}$, which encodes important factors of variation. Assembling the transformation matrix $\boldsymbol{A}$ is dependent on the GAN model we wish to intepret, and below we describe this process for Progan [12] and both StyleGAN [13] and StyleGAN2 [14]. These are the models we used to evaluate our method in Chapter 4 and Chapter 5.

ProGAN. ProGAN represents the traditional generator architecture where the latent code $\boldsymbol{z} \in \mathbb{R}^d$ is fed through the input layer only. In practice, the first transformation step in ProGAN is implemented using a fully-connected layer, which first increases the dimensionality of the input vector by performing the mapping: $\mathbb{R}^d \rightarrow \mathbb{R}^{8192}$, yielding an output vector $\boldsymbol{y}$. Then, this output vector is separately reshaped into a $4 \times 4 \times 512$ activation tensor $\boldsymbol{\mathcal{Y}}$, which is used as input to the subsequent convolution layers.

For the ProGAN model, we are interested in decomposing the weights of the fully-connected layer that performs the above mapping and following notation in Eq. (2.8), we construct the weight matrix $\boldsymbol{A} \in \mathbb{R}^{8192 \times d}$.

StyleGAN. As outlined in Section 2.3.4, StyleGAN's generator architecture includes a mapping network f that transforms the input latent space $\mathcal{Z}$ into an intermediate latent space $\mathcal{W}$ that leads to better disentanglement of the latent factors of variation as described by Eq. (2.5). The output vector $\boldsymbol{w}$ produced by the mapping network f is then replicated once for each AdaIN layer present on the synthesis network g. Each such replica $\boldsymbol{w}_i$ is then specialized to a style tensor $\boldsymbol{\mathcal{Y}}_i$ through an affine transformation, which is implemented once again using a fully-connected layer and if $\boldsymbol{w}_i \in \mathbb{R}^d$ and $fmaps$ denotes the number of feature maps produced by the convolution layer before the AdaIN operation, the fully-connected layer performs the mapping: $\mathbb{R}^d \rightarrow \mathbb{R}^{fmaps \times 2}$.

We are interested in this particular transformation since it is the main way for the synthesis network g to control the strength of different image features on different scales. As in SeFa, our method can decompose all or any subset of the fully-connected layers' weights and

this is achieved by concatenating them along the first axis. Following notation in Eq. (2.8), the above procedure yields a weight matrix $\boldsymbol{A} \in \mathbb{R}^{N \times d}$, where N is the resulting dimension from the concatenation.

3.3 Applying the Multilinear Decomposition on the Weights

At this point the weight parameter $\boldsymbol{A}$ has been constructed and it should contain some informative but entangled shapes that encode important semantic attributes. The hypothesis is that by applying the decomposition in Algorithm 1 and computing the common multilinear basis $\boldsymbol{\mathcal{B}}$, we will be able to extract these attributes and exploit them in order to reliably control the image synthesis.

In practice, we simply treat the weights of the pre-trained model in question as the observations matrix $\boldsymbol{X} \in \mathbb{R}^{d \times N}$, and we apply the multilinear decomposition on this weight matrix. Naturally, when analyzing generators trained on novel or complex datasets, the number of different factors of variation present is usually uknown and hard to determine. As a result, both the number of input dimensions $K_2, \ldots, K_M$ given as input as well as the values selected for each K_m require some experimentation, since different choices here lead to different types of image manipulations. These issues are explored in more detail in Chapter 4.

After these values have been selected, the algorithm produces the multilinear basis $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{d \times K_2 \times K_3 \ldots \times K_M}$ of the weight matrix, which will be used in the semantic editing process, as it hopefully contains important semantic attributes that are sufficiently disentangled to be human-interpretable. Along with tensor $\boldsymbol{\mathcal{B}}$, the algorithm also yields variation coefficients $\boldsymbol{A}^{(2)}$, $\boldsymbol{A}^{(3)}, \ldots, \boldsymbol{A}^{(M)}$, which do not contain any meaningful information for our use case and are thus discarded.

3.4 Semantic Editing with the Multilinear Basis

As explained in Eq. (2.9), the editing process is not dependent on the latent vector $\boldsymbol{z}$ being used, and so a manipulated image can always be produced by adding the term $\alpha \boldsymbol{An}$ onto $\boldsymbol{y} = G_1(\boldsymbol{z})$ from Eq. (2.8), which is the projected code after the first transformation step.

In the general case the number of input dimensions is $M - 1$ and thus the produced multilinear basis is $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{K_2 \times \cdots \times K_M}$. Basis vectors arranged along the same dimension of the tensor express the same variability factor. Intuitively, the decomposition has divided the discovered semantics in categories and semantics within the same category are expected to yield similar image transformations. To access all basis vectors belonging to the same category, we simply slice tensor $\boldsymbol{\mathcal{B}}$ across the corresponding dimension.

For example, consider again the decomposition scenario in Section 3.1 where tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{d \times K_2 \times K_3}$ was a basis of 2 modes of variation, K_2 being expression and K_3 being identity. Basis vectors corresponding to expression can be accessed with $\boldsymbol{\mathcal{B}}_{:,i,:}$, while bases corresponding to identity can be accessed with $\boldsymbol{\mathcal{B}}_{:,:,i}$. All the expression bases can be collected in a matrix by keeping an identity base fixed, e.g the first one, by slicing tensor $\boldsymbol{\mathcal{B}}$ as $\boldsymbol{\mathcal{B}}_{:,:,0}$, which results in a matrix $\boldsymbol{E} = [\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_{|K_2|}] \in \mathbb{R}^{d \times K_2}$. Therefore, to semantically alter the expression of an image produced by the given generator, the editing

operation in Eq. (2.6) can be written as:

$$\boldsymbol{z}' = \boldsymbol{z} + \varepsilon \boldsymbol{e}_i$$
$$\boldsymbol{\mathcal{I}}' = G\left(\boldsymbol{z}'\right),$$

where $\boldsymbol{z}, \boldsymbol{e}_i \in \mathbb{R}^d$ denote the input latent code and a latent direction encoding a face expression respectively and $i \in [1, K_2]$.

A special case of the above decomposition is when we select only one input dimension K_2. In this case, the algorithm produces a matrix $\boldsymbol{B}_{(1)} \in \mathbb{R}^{d \times K_2}$ and the ability of grouping the discovered basis vectors according to the variability factor they control no longer exists. Consequently, the latent semantics identified are all contained in the columns of the matrix $\boldsymbol{B}_{(1)} = [\boldsymbol{n}_1, \boldsymbol{n}_2, \ldots, \boldsymbol{n}_{|K_2|}]$. An interesting observation is that the decomposition orders the semantics by the magnitude of change they cause in the internal representations of the generator model. For most datasets, these directions of largest change usually affect the geometric configuration and the overall layout of the image. This is consistent to what [8] have already demonstrated.

4. EXPERIMENTS

In this Chapter we conduct several experiments on state-of-the-art models and discuss important details of our method that can potentially affect the discovered image transformations and how these are organized. Our goal is to demonstrate that the multilinear decomposition presented in Chapter 3 is completely model-agnostic and capable of uncovering surprising vector directions that can be exploited for high-fidelity image editing.

4.1 Models and Datasets

We apply the proposed technique on 3 of the most prominent GAN architectures, ProGAN (see Section 2.3.3), StyleGAN and StyleGAN2 (see Section 2.3.4). The ProGAN model is pre-trained on the synthetic dataset CLEVR-Simple [21], which will be discussed in the next Section, the StyleGAN models are pre-trained on FFHQ [13] and CelebaHQ [17, 12], and the StyleGAN2 models are pre-trained on FFHQ and a variety of LSUN categories [30] (Car, Horse, Cat, Church).

4.2 Disentanglement Study on ProGAN

It is difficult to determine if a disentanglement algorithm "works" by only testing on real data since the ground truth factors of variation in such datasets are usually unknown and sometimes subjective. The most principled way to measure disentanglement is to train a generative model on a synthetic dataset with known factors of variation, and assessing whether or not the true factors emerge in the latent space. Peebles et al. [21] followed this exact path when testing their approach, and created some synthetic datasets based on CLEVR [11]. Among them is CLEVR-Simple, a dataset which has four factors of variation: object color, shape, and location (both horizontal and vertical) and consists of approximately 10,000 images. They trained a baseline ProGAN model on CLEVR-Simple, which is the model on which we test our method here. Synthesized images are 128×128 resolution.

The fact that the dataset consists of 4 different modes of variation should reasonably guide us to use the multilinear decomposition in Algorithm 1 with 4 input dimensions as input, i.e K_2, K_3, K_4, K_5 and expect to acquire disentangled representations corresponding to each variability factor. Nevertheless, Eq. (3.4) prohibits this scenario, since there is not a possible variable assignment such that $K_2 \cdot K_3 \cdot K_4 \cdot K_5 = d$, where d denotes the dimensionality of the pre-trained model's latent space, $\mathcal{Z} \in \mathbb{R}^d$, which is equal to 12. This forces us to use 3 modes of variation instead of 4, i.e K_2, K_3, K_4 and thus the decomposition becomes:

$$\boldsymbol{A} = \boldsymbol{B}_{(1)}(\boldsymbol{A}^{(2)} \odot \boldsymbol{A}^{(3)} \odot \boldsymbol{A}^{(4)}),$$

where $\boldsymbol{A} \in \mathbb{R}^{d \times N}$, is the weight matrix constructed following the procedure described in Section 3.2 for ProGAN, $\boldsymbol{B_{(1)}} \in \mathbb{R}^{d \times K_2 \cdot K_3 \cdot K_4}$ is a basis of 3 different factors of variation and $\{\boldsymbol{A}^{(m)}\}_{m=2}^{4} \in \mathbb{R}^{K_m \times N}$ are the variation coefficients.

Figure 4.1 shows all of the 12 vector directions discovered by Algorithm 1. More specifically, the i-th row of this figure depicts a linear interpolation of direction $\boldsymbol{n}_i, \forall i \in [1, 12]$, and additionally, across the same row, each image is produced by a semantic editing operation, $\boldsymbol{\mathcal{I}}_{edited} = G(\boldsymbol{z} + \varepsilon \boldsymbol{n}_i)$. Recall that ε is the intensity of the semantic manipulation, and so the rightmost image of a row is generated using $\varepsilon = +k$, whereas the leftmost image

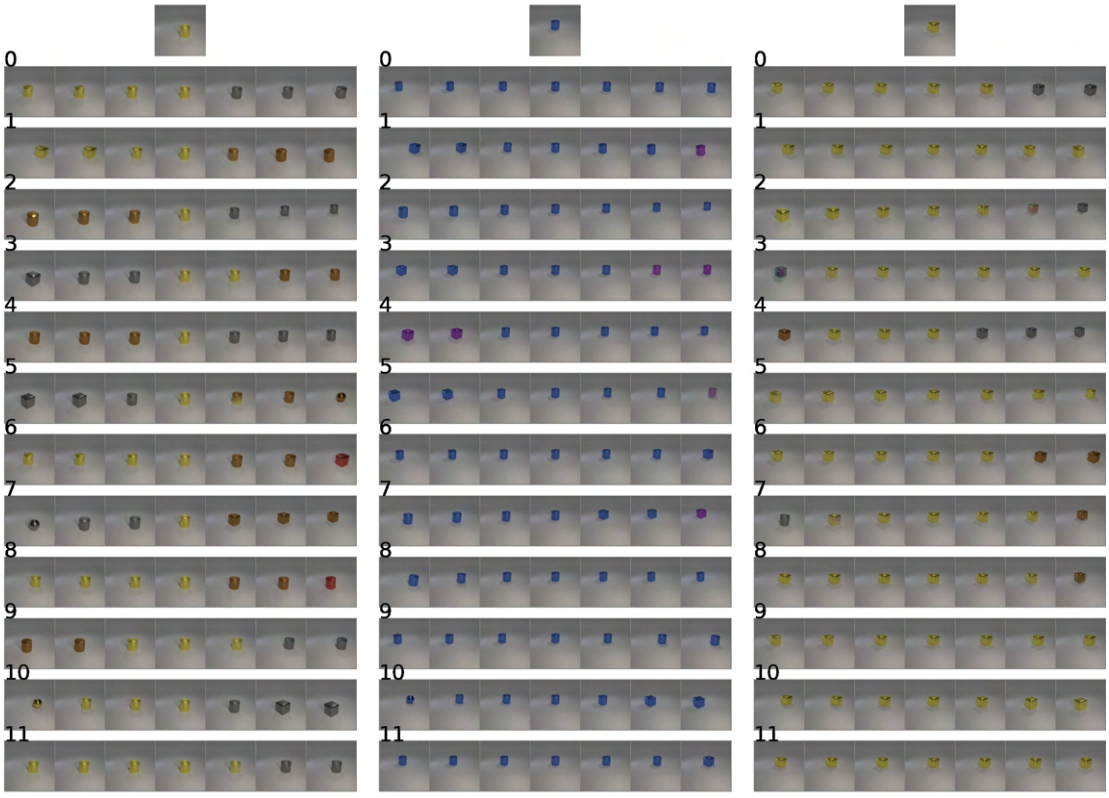


Figure 4.1: Latent semantics discovered by MddGAN applied on the pre-trained ProGAN model. The starting image is always in the middle column.

of a row is generated using $\varepsilon = -k$. Three different generated samples are provided, in order to better assess how each direction affects the original image (top row).

Ideally, we seek meaningful directions that encode independent factors of variation. This means that if one direction modifies a single image attribute, e.g color, then the remaining aspects of the image should remain largely unaffected. For instance, the row with index 1 of the third sample, successfully controls only the horizontal position of the object without influencing the other variability factors. However, when we examine the first sample and the same row (index 1) we observe that the same direction controls not only the horizontal position but the color and the shape of the object simultaneously.

4.3 Decomposing Specific Layer Ranges

Despite the fact that applying our method on all layers of a StyleGAN model discovers human-interpretable directions, they often entail some entanglement. For instance, in the first 4 rows of Fig. 4.2, we linearly interpolate along some of the directions that cause large-scale changes to the source image, and it is evident that they alter several semantic attributes at once - age, hair colour, expression, race and background to name a few. The desired goal of our method however, would be to restrict the global effects of these directions and instead to manipulate the starting image in a more controlled way.

To address this problem, we target certain layer ranges within the generator that are proven to control specific aspects of the generated output [29]. More precisely, for a given StyleGAN generator, we can divide the hierarchical content creation that takes place into three abstraction levels. In the first level, which includes the early layers of the generator,

the network learns how to construct the spatial layout of the output image, and as such controls semantic concepts like camera angle or shape (overall geometry of the image). In the second level, which includes the middle layers, the network synthesizes smaller-scale features related to the category of the generated image data, which for face images can refer to eyes, hair or mouth. The third level includes layers close to the output of the network, which tend to control colors and lighting.

In practice, after selecting the subset of layers about which we wish to extract semantic information, we apply the decomposition only on this subset of AdaIN layers of G, and then through the editing process, we modify only the $\boldsymbol{w}$ inputs to this subset of layers, leaving the other layers' inputs unchanged. For the StyleGAN2 FFHQ model (1024×1024), we label the first four AdaIN layers of the generator as *Bottom* layers, the last ten AdaIN layers as *Top* layers and the remaining ones as *Middle* layers.

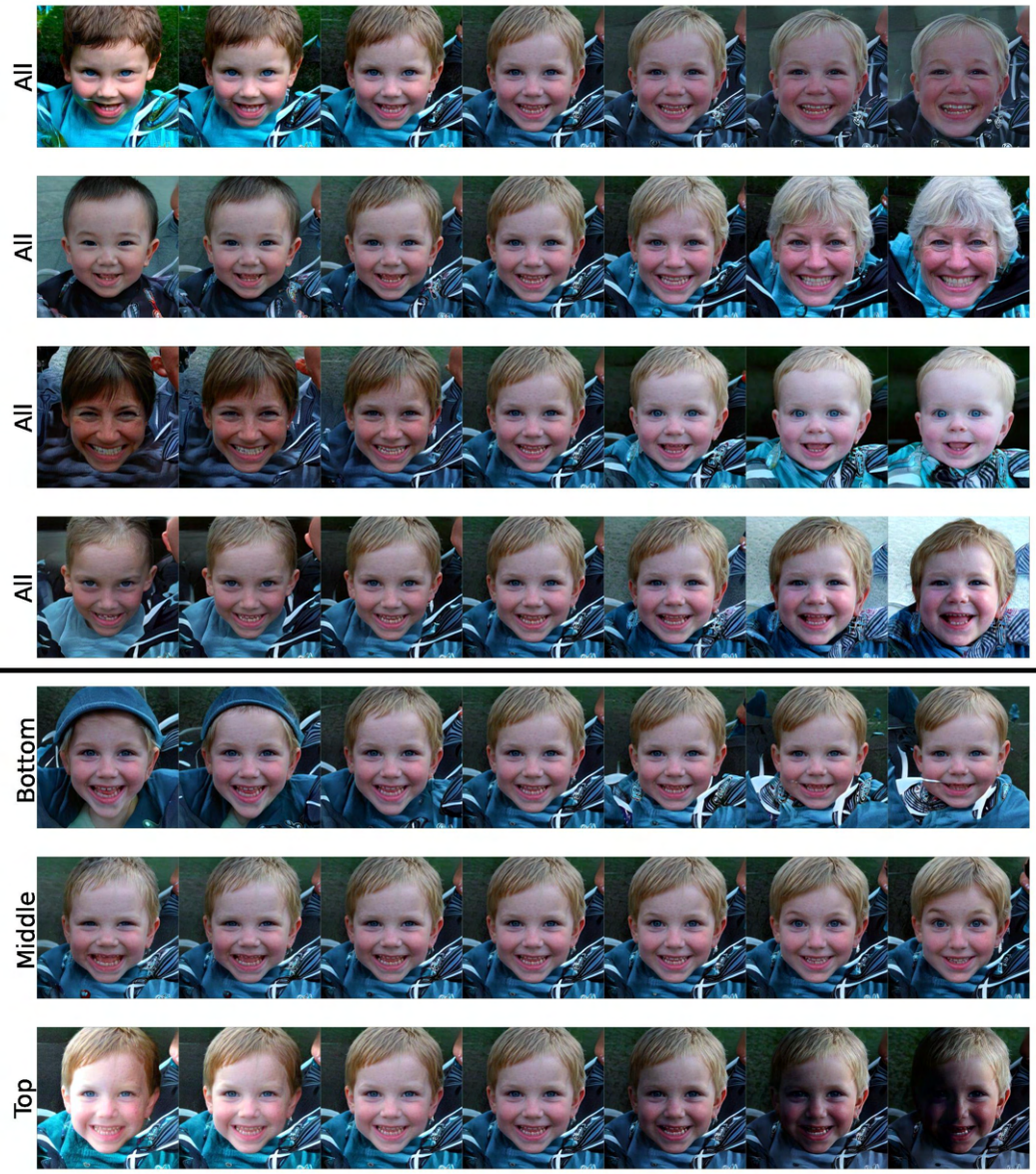


Figure 4.2: A selection of latent semantics discovered by our method applied on StyleGAN2 FFHQ. Rows 1-4 illustrate some of the effects that can be produced by decomposing the weights of all AdaIN layers. Rows 5-7 showcase the effect of layerwise edits. The starting face is always in the middle column. For the interpolation, a shift magnitude $\varepsilon \in [-5, 5]$ was used.

The last three rows of Fig. 4.2 showcase how decomposing the three subsets of layers

previously mentioned can surprisingly restrict the extracted semantics to have a more targeted effect, as opposed to the first four rows. For example, row 5 depicts a discovered semantic that progressively adds a beanie-like accessory to the starting face, leaving the remaining image content largely unchanged. Row 6 demonstrates a semantic which controls expression and hair density and row 7 shows a semantic that modifies lighting and the overall color scheme of the image.

Essentially, the last 3 rows highlight that the key for a more controlled semantic editing lies in separating the generation process into 3 stages and based on the types of image manipulations we wish to perform, analyze the corresponding range of layers.

4.4 Separating the Extracted Semantics into Categories

In the previous sections we simply exhibited a small set of interesting effects that were caused by semantically editing the given generated images using basis vectors discovered by our method. In particular, we didn't attempt to somehow group directions that their corresponding semantic edits transformed the source image in a similar way. For instance, if one of the variation factors among the synthesized data is face expression, it would be more meaningful to be able to gather all basis vectors encoding any type of facial expression and put them in the same category.

Algorithm 1 provides such capabilities, since it attempts to separate the input factors of variation into $M-1$ groups and afterwards by tensorizing matrix $\boldsymbol{B}_{(1)}$ (line 16) we can gather all basis vectors encoding the same variability factor simply by slicing tensor $\boldsymbol{\mathcal{B}}$. At this point, two matters for consideration arise regarding the decomposition. Firstly, as we have already indicated, the true number of variation factors a well trained GAN generator has learnt to model is unknown, and thus we are (mostly) inclined to follow a trial and error approach in our attempt to decide on the optimal value for $M-1$ used in the algorithm. Secondly, the decomposition is a completely unsupervised process and divides the semantic knowledge encoded in the weights of the generator purely mathematically. Thus, in order to assess whether the separation of the discovered image transformations agrees with human logic, we need to visually inspect the results.

As an example, consider a StyleGAN2 generator trained on FFHQ. We choose to analyze the *Bottom* layers of G using Algorithm 1 with two dimensions as input, K_2 and K_3, where $K_2 = 16$ and $K_3 = 32$. By selecting two dimensions, we are intuitively instructing the algorithm to decompose the input weight matrix assuming there are only two explanatory factors in the data. Furthermore, we also imply that mode of variation 1 (K_2) appears in 16 and mode of variation 2 (K_3) in 32 different interpretations. Note that $K_2 \cdot K_3 = d = 512$, where d is the dimensionality of the StyleGAN latent space, so Eq. (3.4) is satisfied. Thus, the algorithm produces the multilinear basis $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{d \times 16 \times 32}$ of order 3. If for example we wish to collect all 16 bases corresponding to mode 1, we can simply slice tensor $\boldsymbol{\mathcal{B}}$ in its second dimension while keeping mode 2 fixed, e.g $\boldsymbol{\mathcal{B}}_{:,:,0}$. The first 5 bases of each mode discovered by the decomposition are plotted in Fig. 4.3.

Examining the results, we can see that bases corresponding to variation mode 1 seem to subtly "stretch" the face either horizontally or vertically when used to semantically manipulate the starting image, while the other image attributes remain largely unaffected (only exception is row 1, with $\varepsilon = -5$ where the hairstyle changes). On the other hand, bases corresponding to variation mode 2 consistently change the hairstyle of the face. In addition, in the 4th row of mode 2 we can also detect a change in the pose.

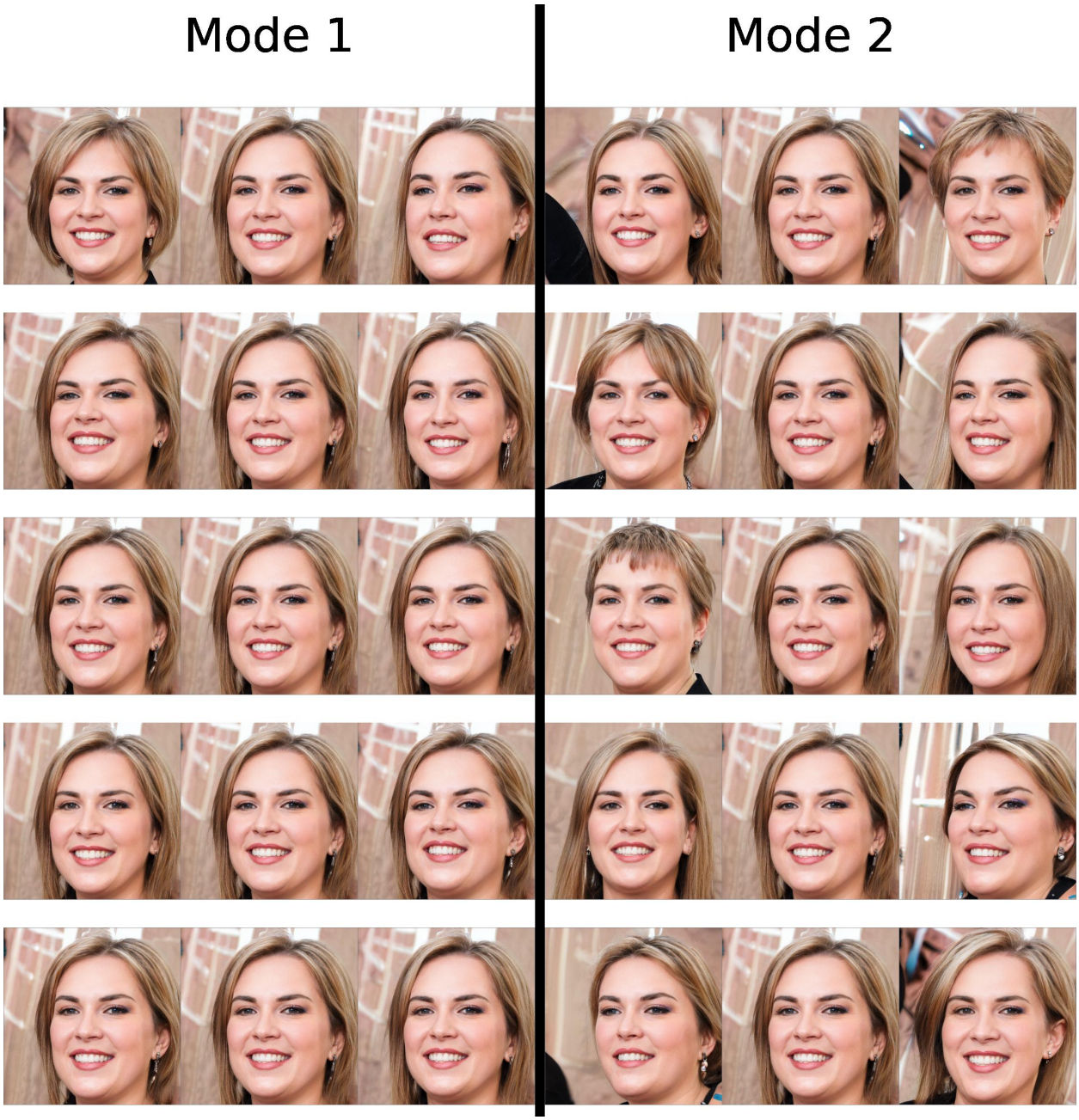


Figure 4.3: Decomposition with two modes of variation, $K_2 = 16$ and $K_3 = 32$, on the bottom layers of the StyleGAN2 FFHQ model. A shift magnitude $\varepsilon \in [-5, 5]$ was used.

The fact that the produced image transformations of mode 2 alter not only the hairstyle, but also the pose, makes us assume that two input dimensions in Algorithm 1 were probably insufficient, because clearly bottom layers have learnt more than two semantic concepts. This naturally raises the question of how to optimally choose the number of variation modes the decomposition will separate the input data into, which is discussed next.

4.4.1 How Many Modes of Variation?

A significant decision/choice to be made when decomposing the weights of the generator, is how many modes of variation $M - 1$, Algorithm 1 will discover in the input data. In other words, the number of input dimensions $K_2, K_3, \ldots, K_M$ along with the values chosen for each dimension $K_m, \forall m \in [2, M]$ can affect the types of image manipulations the decom-

position will uncover. Ideally, we would like to set $M-1$ equal to the true number of variation factors learnt by the model and consequently encoded in its weights. Unfortunately, this number is not known in advance.

Returning to the example of Fig. 4.3, recall that that 2 modes of variation or equivalently 2 input dimensions K_2 and K_3 as input to the decomposition algorithm proved to be inadequate. So, we proceed by repeating the experiment using three modes of variation this time, K_2, K_3 and K_4, with equal values $K_2 = K_3 = K_4 = 8$ and hopefully, this will be a better approximation of the true number of variation modes. Note that in this experiment, the decomposition yields a basis tensor $\mathcal{B} \in \mathbb{R}^{d \times 8 \times 8 \times 8}$ of order 4 and the equality constraint in Eq. (3.4), $K_4 \cdot K_3 \cdot K_2 = d = 512$ still holds. Figure 4.4 shows the first 5 bases corresponding to each mode of variation.

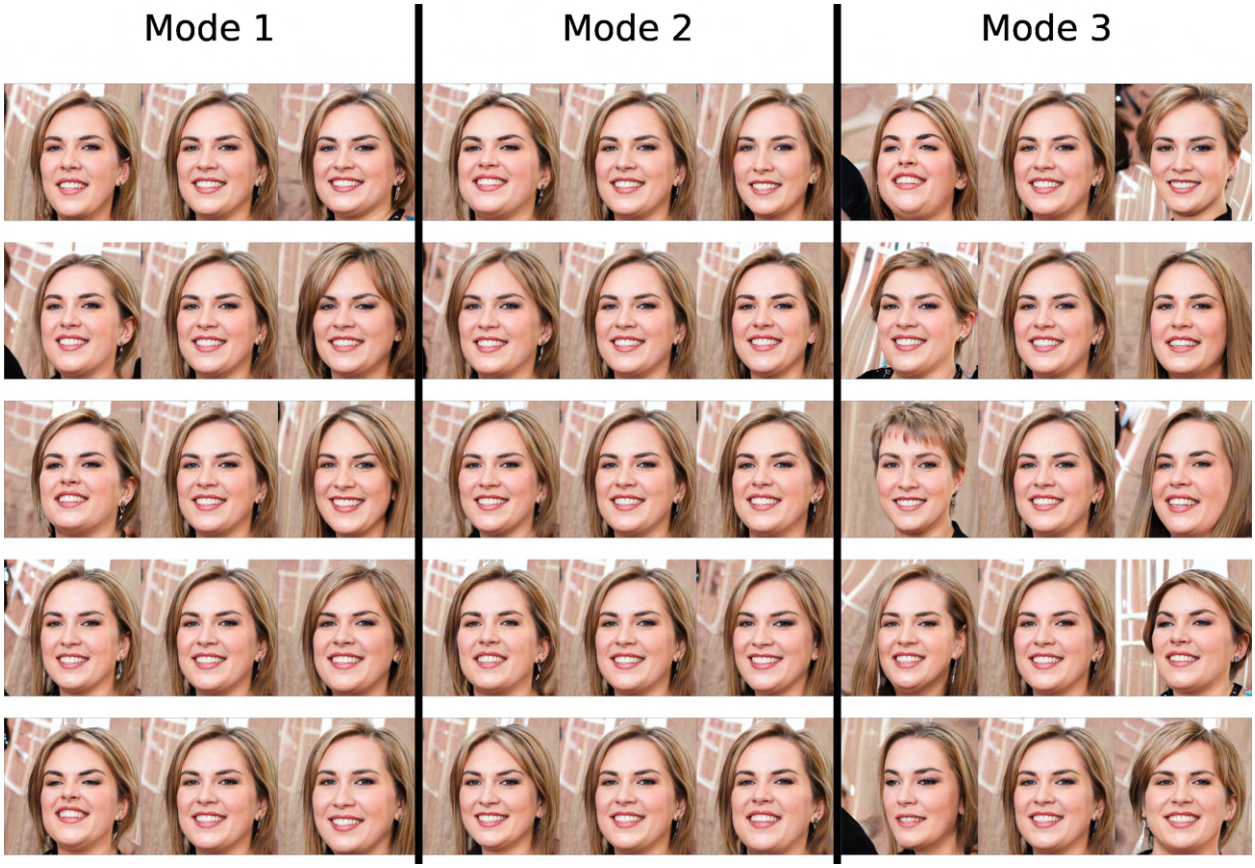


Figure 4.4: Decomposition with three modes of variation, K_2, K_3 and K_4 with $K_2 = K_3 = K_4$, on the bottom layers of the StyleGAN2 FFHQ model. A shift magnitude $\varepsilon \in [-5, 5]$ was used.

Interestingly, Mode 1 once again includes bases that appear to stretch the starting face either horizontally or vertically while mostly preserving the hairstyle of the face. On the other hand, it is not quite clear how Mode 2 bases affect the original image, since they do not produce any substantial changes after the semantic editing procedure (undefined). Lastly, bases corresponding to variation mode 3, edit the synthesized image in a way very similar to Mode 2 bases from Fig. 4.3. More specifically, they manage to consistently control the hairstyle and also change the pose of the subject (e.g Row 5).

So, increasing the number of input dimensions to 3, didn't improve the separation any further. On the contrary, we created an extra category of image transformations (Mode 2) that appears to cause incredibly subtle, almost non-existent changes to the synthesized image. Randomly trying every allowed / possible number of input dimensions in Algorithm 1 clearly isn't a viable approach, and instead we should try and define an estimate of the

optimal decomposition in terms of the number of input dimensions. Basically, we want this estimate to assist us on determining the true number of variation factors a well trained generator has learnt to model.

In fact, the convergence condition (line 13) of Algorithm 1 could act as such an estimate, since it calculates the absolute relative error of the decomposition and can therefore provide a measurement of how accurate is the reconstruction of the input data. So, by keeping the number of discovered directions fixed, we progressively increase the number of variation modes discovered by the decomposition. Although model-dependent, Fig. 4.5 on the top-left depicts that by increasing the input modes of variation, we can expect to obtain a data reconstruction of superior quality and this is observed across all layer ranges.

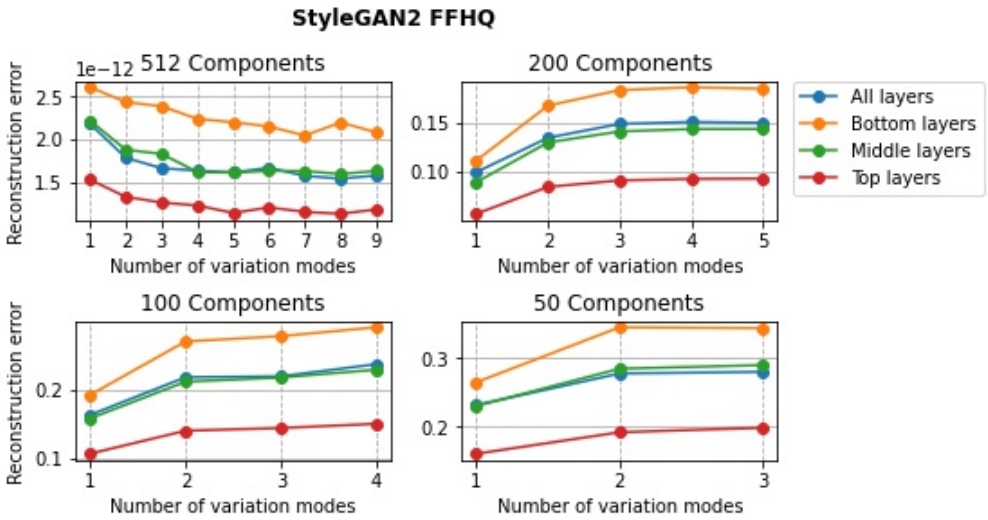


Figure 4.5: How increasing the number of variation modes impacts the decomposition in terms of reconstruction error on StyleGAN2 FFHQ

Comparing Fig. 4.3 to Fig. 4.4 we can hypothesize that a lower reconstruction error does not necessarily imply a better separation of the input modes of variation, at least visually. For example, in Fig. 4.4 the decomposition might intentionally put all basis vectors that do not produce any substantial effects in Mode 2, because it cannot identify any other common patterns in the weights of the model.

4.5 Reducing the Number of Discovered Directions

So far, all of our experiments involved applying the unsupervised multilinear matrix decomposition on the weights of a well trained GAN generator and producing a full-rank matrix $\boldsymbol{B}_{(1)}$ $(d \times d)$, that contains all the necessary information required by the model to generate images. Nonetheless, the $d \times d$ full-rank matrix captures 100% of the variance of the input data, which is something we want to avoid. In particular, the number of interpretable directions d might be too many to lead to a fully-comprehensible result.

For example, StyleGAN models use $d = 512$ as their latent space dimensionality and as a result, Algorithm 1 by default attempts to discover 512 basis bectors, each one encoding some semantic concept. However, we have found that across all trained (StyleGAN) models we have explored, when analyzing all layers, distinctive changes to generated image content make up only about 150-200 of the total basis vectors discovered, while for the three previously mentioned subsets of layers (Bottom-Middle-Top), the corresponding

number is even smaller, around 100 of the total basis vectors discovered. The rest do not impact the synthesized image in any significant way.

As mentioned in Section 3.1.3, opting for the Truncated SVD instead of the original SVD, is capable of producing a matrix $\boldsymbol{B}_{(1)} \in \mathbb{R}^{l \times d}$ or equivalently extracting a reduced number of latent semantics $l < d$. The observation of the previous paragraph provides the main intuition behind the values of l we chose to experiment with in this Section. More specifically, we tested four different values for l: 512 ($= d$), 200, 100 and 50. Of course, as we outlined also in Section 3.1.3, applying the decomposition seeking a sparser basis of the input weights leads to significantly higher reconstruction error, also confirmed by Fig. 4.5.

Figure 4.6 and Fig. 4.7 demonstrate how a reduced number of discovered semantics can affect the corresponding image transformations. In these examples we used three input dimensions, similar to Fig. 4.4, but instead of $d = 512$, we instruct the algorithm to identify $l = 100$ directions with $K_2 = 4$, $K_3 = 4$ and $K_4 = 5$ in Fig. 4.6 and $l = 50$ directions with $K_2 = 2$, $K_3 = 5$ and $K_4 = 5$ in Fig. 4.7. In each case, Eq. (3.5) is satisfied. Evidently, the image transformations produced in these 2 scenarios are different compared to what the full-rank matrix decomposition generated. For instance, in Fig. 4.7 we can discern a discovered semantic that adds eyeglasses to the starting face (Mode 3 - row 3), which was not uncovered in our previous experiments. Additionally, Figs. 4.8 to 4.11 explore how the discovered basis vectors are organized in the multilinear basis's $\boldsymbol{\mathcal{B}}$ dimensions in the case of StyleGAN2 generators pre-trained on LSUN categories.

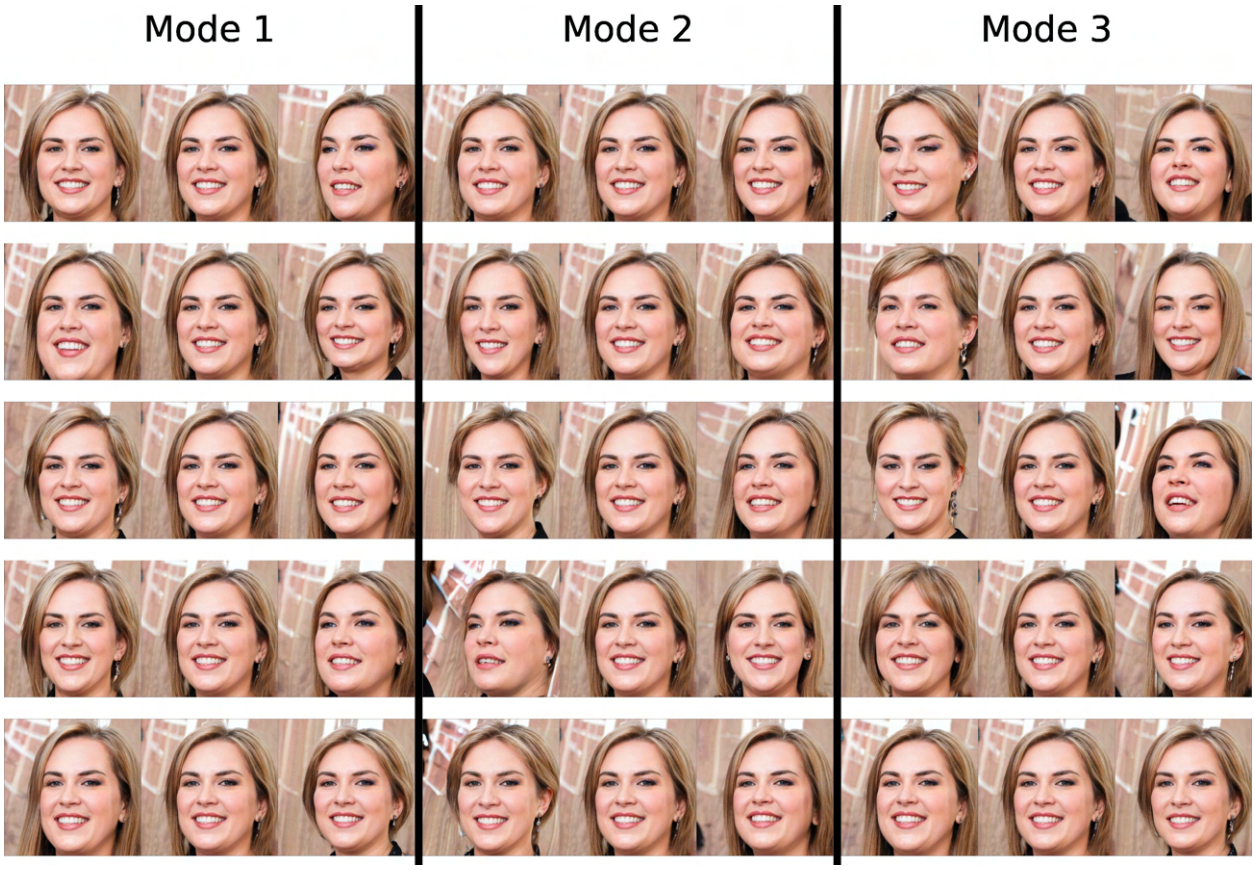


Figure 4.6: Decomposition with three modes of variation, K_2, K_3 and K_4 with $K_2 = 4, K_3 = 4$ and $K_4 = 5$, on the bottom layers of the StyleGAN2 FFHQ model. Here, the number of discovered directions is $l = 100 < d$. A shift magnitude $\varepsilon \in [-5, 5]$ was used.

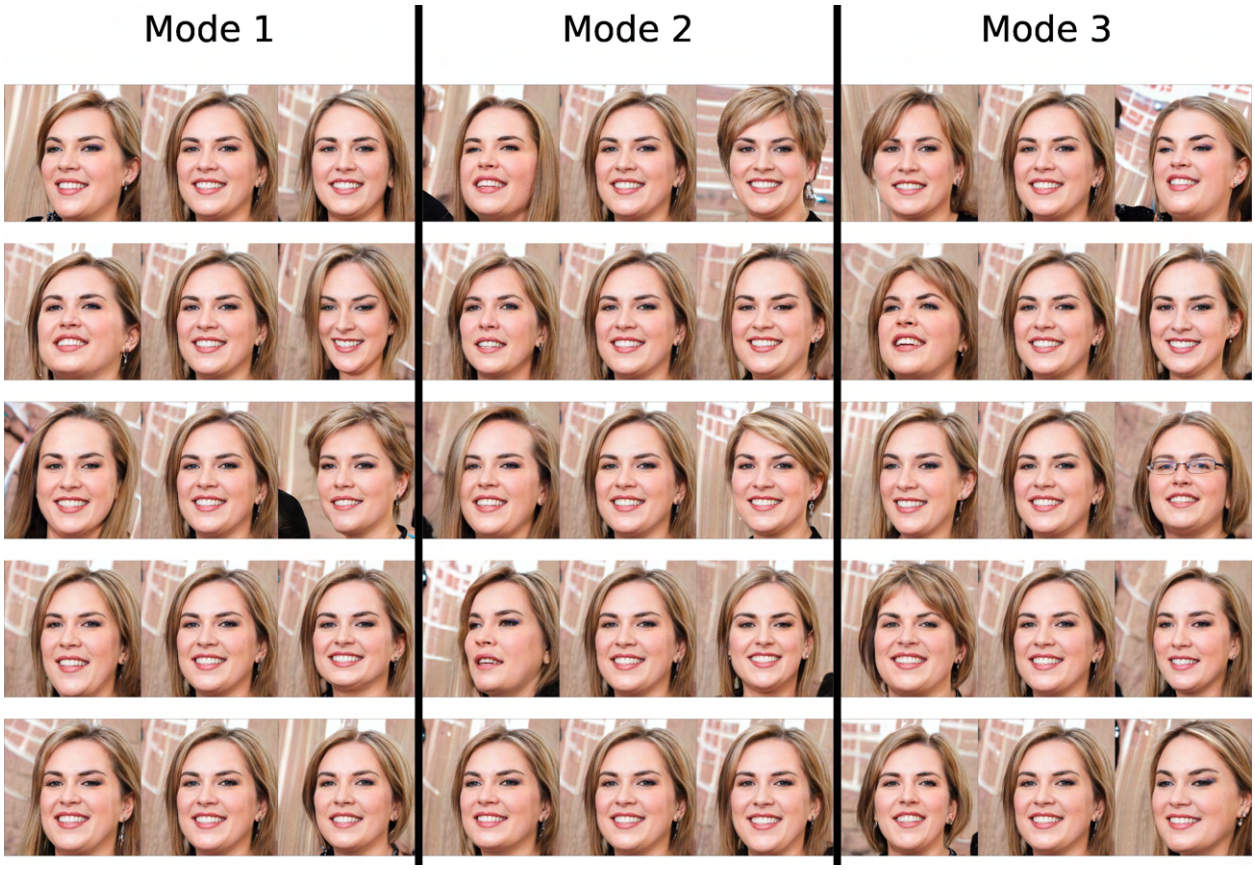


Figure 4.7: Decomposition with three modes of variation, K_2, K_3 and K_4 with $K_2 = 2, K_3 = 5$ and $K_4 = 5$, on the bottom layers of the StyleGAN2 FFHQ model. Here, the number of discovered directions is $l = 50 < d$. A shift magnitude $\varepsilon \in [-5, 5]$ was used.

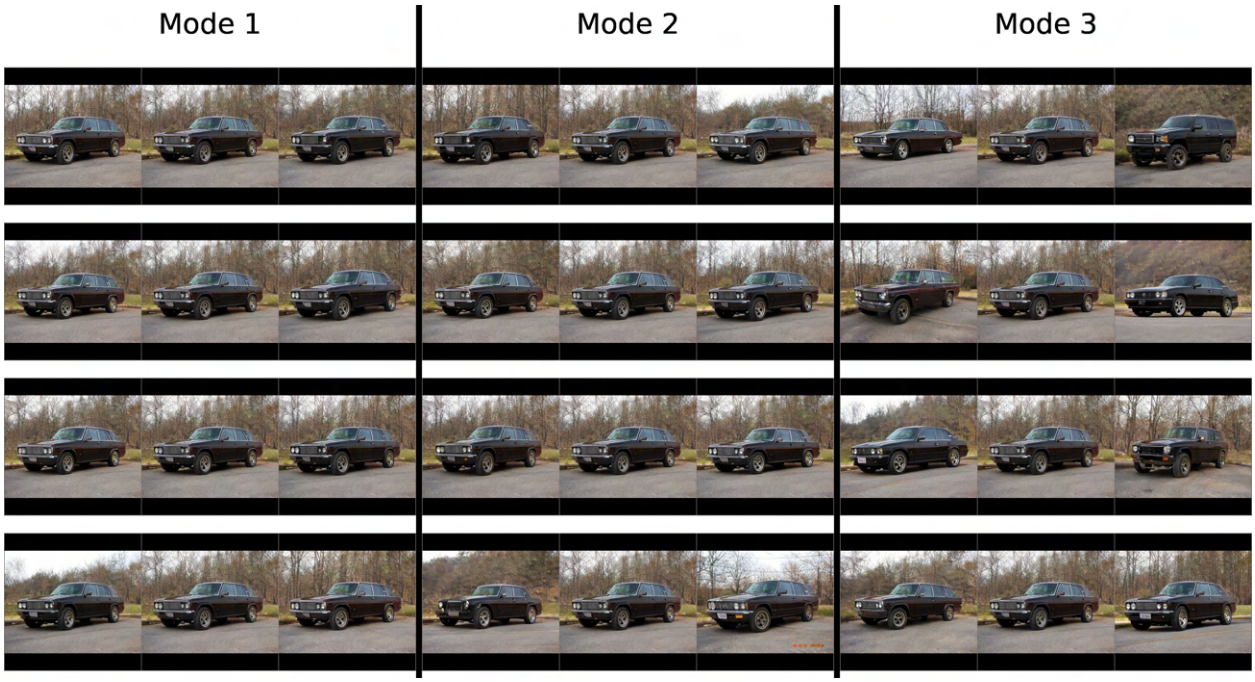


Figure 4.8: Decomposition with three modes of variation, K_2, K_3 and K_4 with $K_2 = 4, K_3 = 5$ and $K_4 = 5$, on the middle layers of the StyleGAN2 LSUN-Car model. Here, the number of discovered directions is $l = 100 < d$. A shift magnitude $\varepsilon \in [-5, 5]$ was used.

Figure 4.9: Decomposition with three modes of variation, K_2, K_3 and K_4 with $K_2 = 4, K_3 = 5$ and $K_4 = 5$, on the top layers of the StyleGAN2 LSUN-Cat model. Here, the number of discovered directions is $l = 100 < d$. A shift magnitude $\varepsilon \in [-5, 5]$ was used.

Figure 4.10: Decomposition with two modes of variation, K_2 and K_3 with $K_2 = K_3 = 10$, on the bottom layers of the StyleGAN2 LSUN-Church model. Here, the number of discovered directions is $l = 100 < d$. A shift magnitude $\varepsilon \in [-5, 5]$ was used.

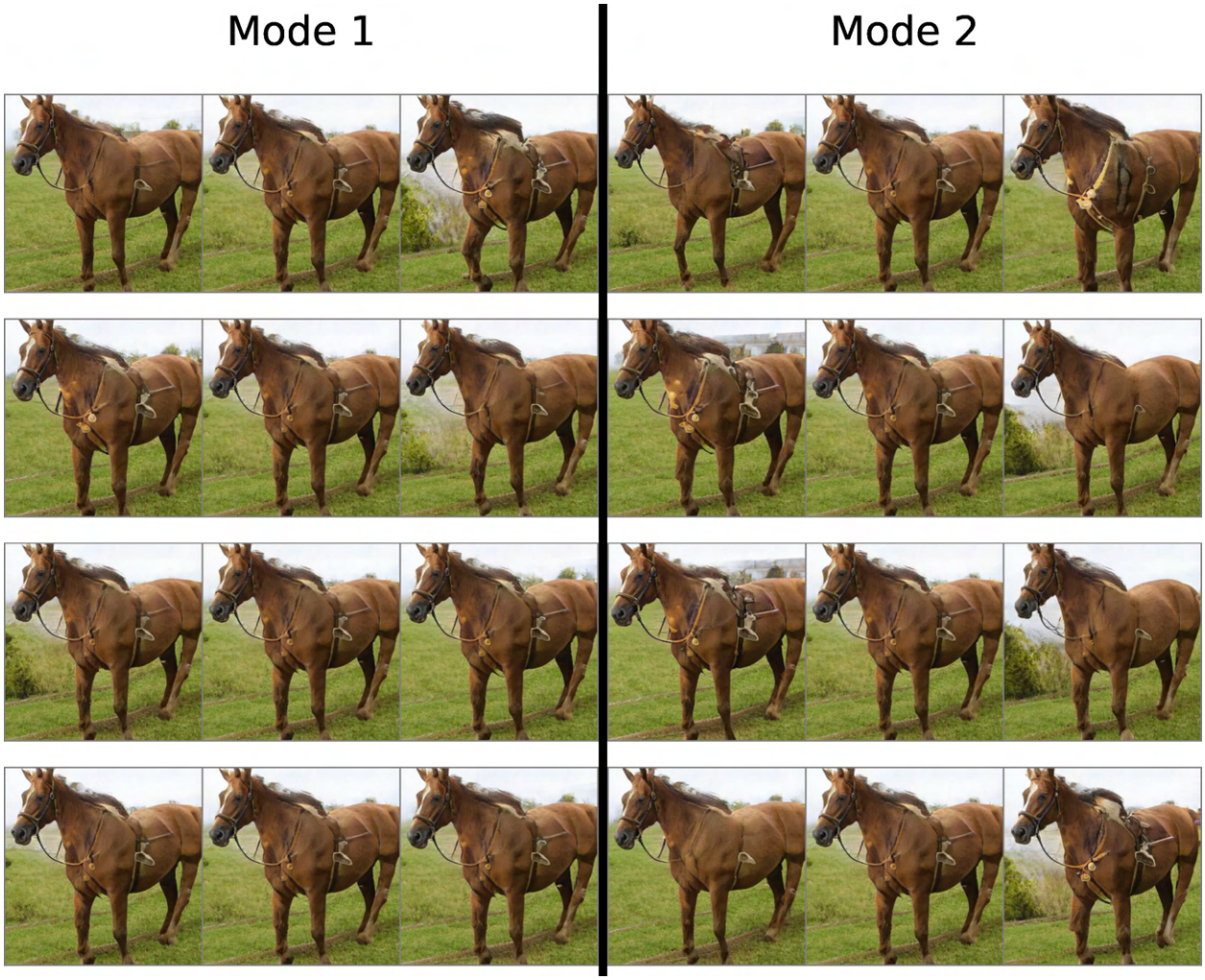


Figure 4.11: Decomposition with two modes of variation, K_2 and K_3 with $K_2 = 10$ and $K_3 = 5$, on the middle layers of the StyleGAN2 LSUN-Horse model. Here, the number of discovered directions is $l = 50 < d$. A shift magnitude $\varepsilon \in [-5, 5]$ was used.

5. EVALUATION

In this chapter we provide comparisons to the more powerful class of techniques that similarly to our method MddGAN, attempt to analyze and interpret the latent space of GANs. In particular, we compare MddGAN to the current supervised and unsupervised baselines, InterFaceGAN [25] and SeFa [26] respectively.

5.1 Supervised Method Comparison

InterFaceGAN [25] is a prominent supervised control method, which uses existing detectors for face attributes and keypoints to find directions for face editing.

5.1.1 Qualitative Results

To perform an extended visual comparison between the 2 methods, we chose a set of interpretable vector directions corresponding to the following facial attributes: pose, gender, age, smile and eyeglasses. For InterFaceGAN, we use the above vector directions found in the authors' official implementation. Note that all semantic editing operations take place in the $\mathcal{W}$ space of StyleGAN. For the comparisons presented in Fig. 5.1 and Fig. 5.2 we used the StyleGAN model trained on the CelebaHQ and FFHQ datasets.

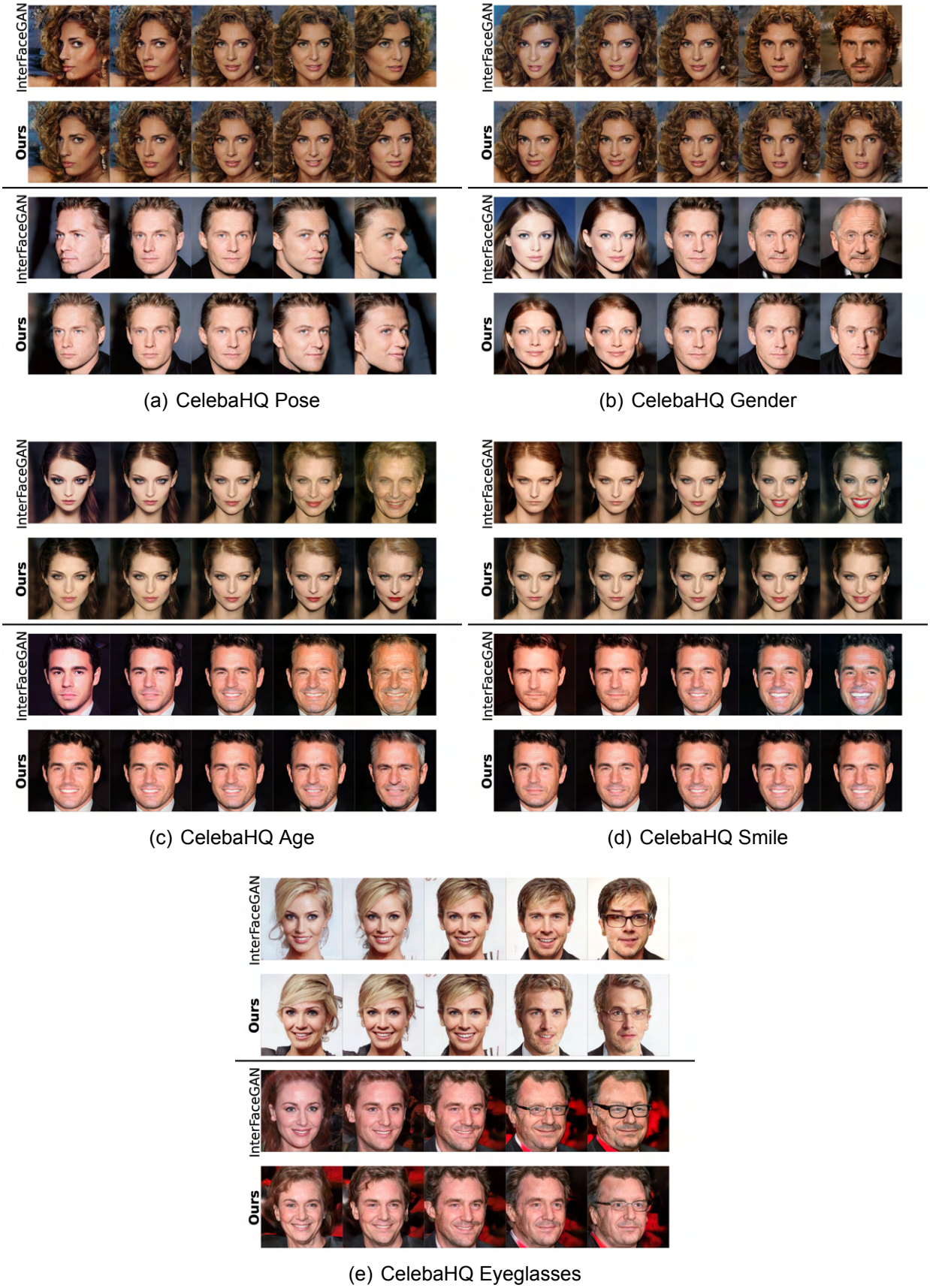
We find that all of the above edits can be very accurately recreated with our technique, despite it being unsupervised and not having access to the target transformations. In fact, MddGAN manages to even surpass InterfaceGAN by better preserving the starting face identity and the overall image content in general. For instance, in Fig. 5.1, in the case of the gender attribute comparison, the effect produced by MddGAN alters the image in a more controllable manner leaving the remaining aspects of the image such as hair and age largely unchanged, whereas [25] also affects age, hair and skin tone. Similarly, in the case of the age semantic, [25] captures the ”old” aspect of the transformation better, but it fails to preserve the starting expression (sample 2) and it also changes the skin color to a yellow tone.

However, it is evident that both methods are unable to uncover a sufficiently disentangled eyeglasses vector direction, at least in the case of the Stylegan CelebaHQ model. In particular, Fig. 5.1 shows that the eyeglasses direction discovered seems to also control age and gender. As explained in [25], this may be due to the biases inherited from the training set, the CelebaHQ in this case, where people wearing eyeglasses appear to be mostly older males.

5.1.2 Quantitative Results

5.1.2.1 Correlation between Attributes

In this Section we will focus on the relationships between different hidden semantics and study how they are coupled with each other. In particular, we compute the cosine similarity between two given attribute vectors as $\cos(\boldsymbol{n}_1, \boldsymbol{n}_2) = \boldsymbol{n}_1^T \boldsymbol{n}_2$, where $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ stand for unit vectors.



(a) CelebaHQ Pose (b) CelebaHQ Gender

(c) CelebaHQ Age (d) CelebaHQ Smile

(e) CelebaHQ Eyeglasses

Figure 5.1: Comparison to InterfaceGAN [25] for the StyleGAN CelebaHQ model.

Table 5.1 and Table 5.2 report the results. We observe that in MddGAN's correlation matrix, gender, age and eyeglasses are highly correlated with each other. This is also evident

(a) FFHQ Pose (b) FFHQ Gender

(c) FFHQ Age (d) FFHQ Smile

(e) FFHQ Eyeglasses

Figure 5.2: Comparison to InterfaceGAN [25] for the StyleGAN FFHQ model.

in Fig. 5.2 (e), where linearly interpolating towards the negative side of the MddGAN attribute vector, causes a change to the gender of the face. The eyeglasses-age attribute

relationship, as mentioned earlier, is inherited from the training dataset. The same attribute pair also seems to yield the highest correlation score in the correlation matrix of InterFaceGAN.

In general though, we can see that InterFaceGAN benefits from its supervised training scheme, since the 5 attributes are mostly decoupled, which is not the case for MddGAN. The interpretable semantics were extracted from the StyleGAN FFHQ model.

Table 5.1: Correlation matrix of attribute vectors for MddGAN. The attribute vectors were extracted from the StyleGAN FFHQ model.

	Pose	Gender	Age	Eyeglasses	Smile
Pose	1.00	-0.12	-0.17	-0.05	0.04
Gender	-	1.00	0.04	0.71	-0.01
Age	-	-	1.00	0.45	0.04
Eyeglasses	-	-	-	1.00	0.18
Smile	-	-	-	-	1.00

Table 5.2: Correlation matrix of attribute vectors for InterFaceGAN [25]. The attribute vectors were extracted from the StyleGAN FFHQ model.

	Pose	Gender	Age	Eyeglasses	Smile
Pose	1.00	0.03	0.02	-0.02	0.00
Gender	-	1.00	0.07	0.00	-0.11
Age	-	-	1.00	0.10	0.03
Eyeglasses	-	-	-	1.00	0.01
Smile	-	-	-	-	1.00

5.1.2.2 Diversity Comparison

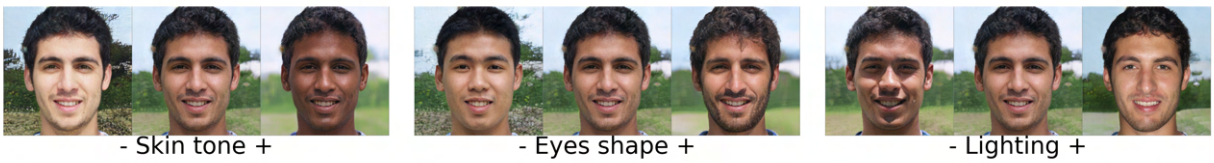
InterFaceGAN is able to discover latent space directions corresponding to interpretable semantic attributes only when there are available classifiers for these attributes. Consequently, for attributes that cannot be effectively described with only 2 values e.g race or hairstyle, there exists no straight-forward way to acquire semantic predictors. This limitation makes InterFaceGAN and similar supervised techniques impossible to rely on for general-purpose semantic image editing, since the number of meaningful directions that can be discovered is significantly reduced.

In contrast, our method MddGAN does not contain any form of supervision, yet it is vastly more flexible when it comes to discovering diverse image manipulations. For example, the weight decomposition manages to successfully extract vector directions related to geometric properties (face shape, zoom) and hairstyle for the case of StyleGAN CelebaHQ, as well as race (skin tone, eyes shape) and light exposure in the case of StyleGAN FFHQ. The above edits are demonstrated in Fig. 5.3.

5.2 Unsupervised Method Comparison

In this Section, we compare our method to the current state-of-the-art unsupervised approach, SeFa [26]. Our proposed method MddGAN can be viewed as a variant of SeFa,

(a) Semantics extracted by MddGAN from the StyleGAN CelebaHQ model



(b) Semantics extracted by MddGAN from the StyleGAN FFHQ model

Figure 5.3: Evidence of diverse semantics that cannot be explicitly modeled with binary values, and hence cannot be identified by InterFaceGAN [25].

since it similarly inspects the parameters of a pre-trained generator directly and extracts meaningful vector directions that can be later used for semantic image editing.

5.2.1 Qualitative Results

As in Section 5.1.1, we will compare the 2 methods by visualizing vector directions corresponding to the 5 semantic attributes: pose, gender, age, smile and eyeglasses. Because the authors of SeFa don't specify the individual attribute vectors used in their experiments, we manually seek the most relevant ones that can be directly compared to those identified by our method. In other words, when analyzing the generator model of interest, we apply SeFa on exactly the same subset of layers that we applied MddGAN in order to find the best matching basis vector.

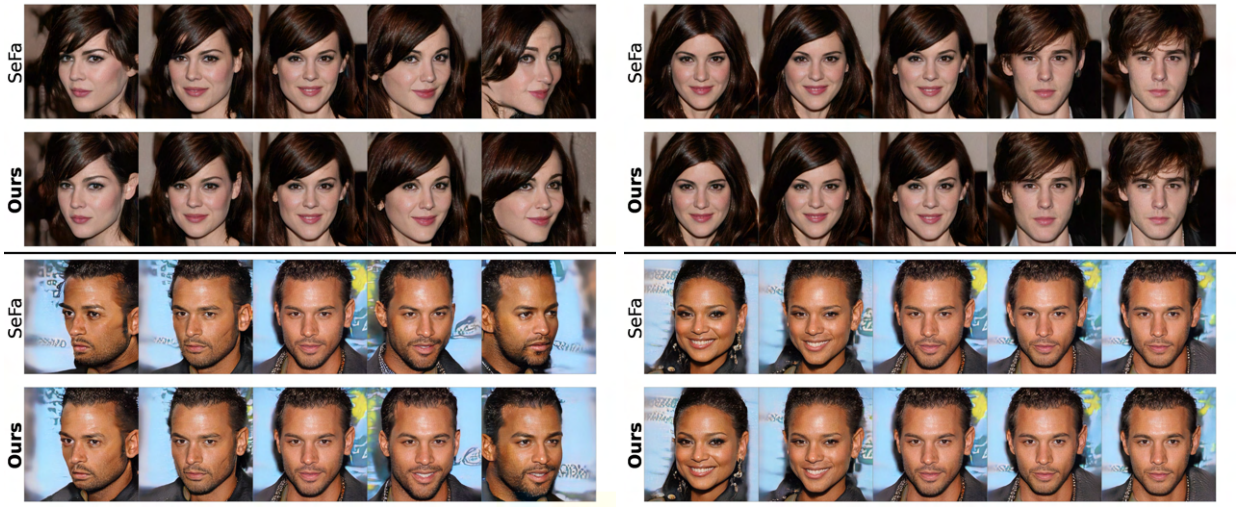
Surprisingly, the identified semantics depicted in Fig. 5.4 are almost identical in terms of produced semantic manipulations. The only differences we can detect, are in (a), where the MddGAN attribute vector alters the image in a more realistic manner, whereas the SeFa attribute seems to introduce inconsistencies in the output image. Moreover, in (d) SeFa fails to completely close the mouth of the face on the negative side of the direction.

5.2.2 Quantitative Results

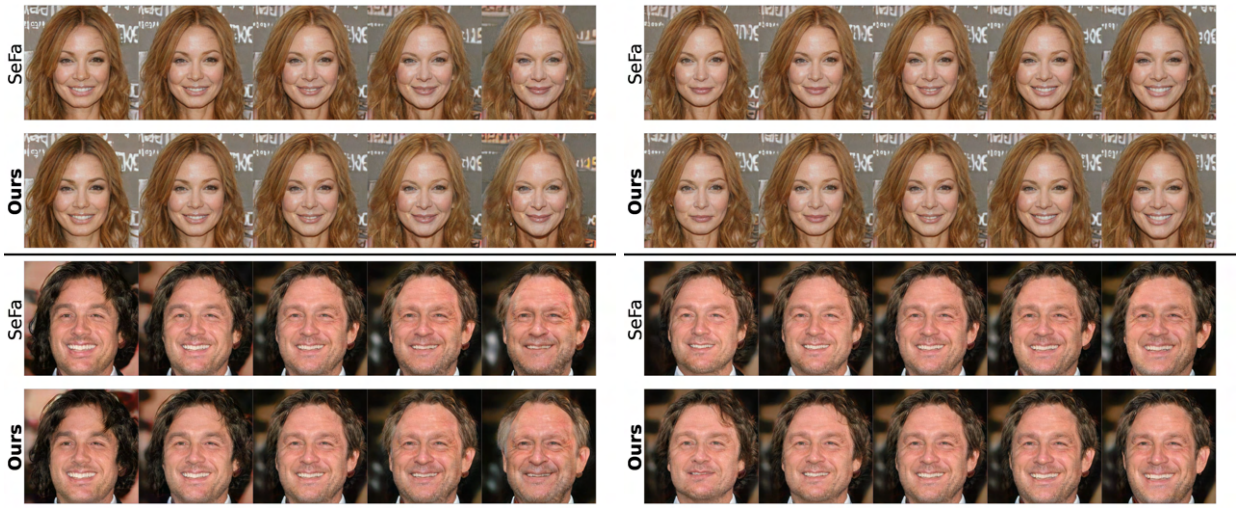
5.2.2.1 Correlation between Attributes

Similar to the InterFaceGAN comparison, we will once again investigate the relationships between the different hidden semantics found by MddGAN and SeFa.

Table 5.3 and Table 5.4 report the results. We observe that MddGAN in general produces higher correlation scores than SeFa, even though the attribute vectors visualized in Fig. 5.4 cause almost identical effects. The highest correlation values however emerge in the gender-age attribute relationship, which basically implies that they both have a high influence on one another. The interpretable semantics were extracted from the StyleGAN CelebaHQ model.



(a) CelebaHQ Pose (b) CelebaHQ Gender



(c) CelebaHQ Age (d) CelebaHQ Smile

(e) CelebaHQ Eyeglasses

Figure 5.4: Comparison to SeFa [26] for the StyleGAN CelebaHQ model.

Table 5.3: Correlation matrix of attribute vectors for MddGAN. The attribute vectors were extracted from the StyleGAN CelebaHQ model.

	Pose	Gender	Age	Eyeglasses	Smile
Pose	1.00	0.11	0.19	0.16	-0.05
Gender	-	1.00	0.53	0.00	0.11
Age	-	-	1.00	-0.08	-0.03
Eyeglasses	-	-	-	1.00	0.04
Smile	-	-	-	-	1.00

Table 5.4: Correlation matrix of attribute vectors for SeFa [26]. The attribute vectors were extracted from the StyleGAN CelebaHQ model.

	Pose	Gender	Age	Eyeglasses	Smile
Pose	1.00	-0.22	-0.09	0.12	-0.05
Gender	-	1.00	0.29	-0.01	-0.03
Age	-	-	1.00	0.08	0.08
Eyeglasses	-	-	-	1.00	-0.06
Smile	-	-	-	-	1.00

5.2.2.2 Fréchet Inception Distance Comparison

The Fréchet Inception Distance (FID) [6] has become the standard metric for evaluating GAN performance. In practice, the FID compares the generated data distribution p_g with the distribution of the real data used to train the model p_{data}. In the case of image data, lower FID scores have been shown to correlate well with higher quality images, proving that the FID is consistent with human judgement.

In order to calculate the FID of each approach, we first need a sample size of 50,000 images drawn randomly from the training set. This dataset resembles the real data distribution p_{data}. We then generate 50,000 fake images in minibatches and across a minibatch we use the same attribute vector $\boldsymbol{n}$ to edit along with the same magnitude ε. In particular, for each minibatch we randomly select one of the 5 attribute vectors discussed previously and we also pick the magnitude value to be either $-k$ or $+k$. This process yields a synthetic dataset of images that were also semantically edited, which resembles the generator distribution p_g. Finally, the FID score between the two distributions can be computed. For this comparison, we used the StyleGAN model trained on CelebaHQ. The FID scores, which are remarkably close are presented in Table 5.5.

Table 5.5: Comparison results in terms of edited image quality. Lower is better.

Methods	**FID**$\downarrow$
SeFa [26]	**25.0**
MddGAN (Ours)	25.1

6. CONCLUSIONS

GANs are powerful deep-learning based models that keep pushing the boundaries of high-fidelity image synthesis. But whereas the image quality of these models has improved and will probably continue to improve, their controllability has lagged behind. This thesis proposes MddGAN, an unsupervised method for discovering human-interpretable directions by performing multilinear analysis on the parameters of a pre-trained generator model. As a result, this method does not require repeating the training process nor sampling numerous data. Instead it can be applied directly on the model we are interested in interpreting. Furthermore, our method does not simply identify meaningful image transformations, but it is also capable of separating them into different categories and the experimental results indicate that this separation is generally aligned with human perception.

When comparing MddGAN with the supervised baseline, we observe that even though our method does not have access to the specific attribute classifiers, it can successfully identify the corresponding directions and in fact, during editing, it manages to better preserve the starting identity and the overall image content in general. The comparison results with the unsupervised baseline are remarkably close, as evidenced by both the qualitative and quantitative results. These findings highlight the effectiveness of the proposed method, which can clearly challenge the current state-of-the-art ones.

ABBREVIATIONS - ACRONYMS

GAN	Generative Adversarial Network
SGD	Stochastic Gradient Descent
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
MLP	Multilayer Perceptron
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
AdaIN	Adaptive Instance Normalization
FFHQ	Flickr-Faces-HQ
FID	Fréchet Inception Distance
MDD	Multilinear Data Decomposition

BIBLIOGRAPHY

[1] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. 2018.

[2] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323, 2011.

[3] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1319–1327, 2013.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[5] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein GANs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[6] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6629–6640, 2017.

[7] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1510–1519, 2017.

[8] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable GAN controls. In *Proc. NeurIPS*, 2020.

[9] Abdul Jabbar, Xi Li, and Bourahla Omar. A survey on generative adversarial networks: Variants, applications, and training. 2020.

[10] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, 2009.

[11] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. Zitnick, and R. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997, 2017.

[12] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. 2017.

[13] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. 2018.

[14] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8107–8116, 2020.

[15] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. volume 86, pages 2278–2324, 1998.

[17] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3730–3738, 2015.

[18] X. Mao, Q. Li, H. Xie, R. K. Lau, Z. Wang, and S. Smolley. Least squares generative adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2813–2821, 2017.

[19] Lars M. Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *ICML*, 2018.

[20] Weili Nie, Tero Karras, Animesh Garg, Shoubhik Debnath, Anjul Patney, Ankit Patel, and Animashree Anandkumar. Semi-supervised StyleGAN for disentanglement learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 7360–7369, 2020.

[21] William Peebles, John Peebles, Jun-Yan Zhu, Alexei A. Efros, and Antonio Torralba. The hessian penalty: A weak prior for unsupervised disentanglement. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2020.

[22] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016,*, 2016.

[23] Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *AAAI*, 2018.

[24] Andrei Rusu, Neil Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. 2016.

[25] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9240–9249, 2020.

[26] Yujun Shen and Bolei Zhou. Closed-form factorization of latent semantics in gans. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1532–1540, 2021.

[27] Andrey Voynov and Artem Babenko. Unsupervised discovery of interpretable directions in the GAN latent space. In *International Conference on Machine Learning*, pages 9786–9796, 2020.

[28] Mengjiao Wang, Yannis Panagakis, Patrick Snape, and Stefanos Zafeiriou. Learning the multilinear structure of visual data. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6053–6061, 2017.

[29] Ceyuan Yang, Yujun Shen, and Bolei Zhou. Semantic hierarchy emerges in deep generative representations for scene synthesis. *International Journal of Computer Vision*, 129, 2021.

[30] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: construction of a large-scale image dataset using deep learning with humans in the loop. 2015.