Behind the Face: Unveiling the Effects of Background Subtraction on VAE and GAN Model Efficacy

Thomas Kientz
ENSAE Paris
thomas.kientz@ensae.fr

Elena Loumagne
ENSAE Paris
elena.loumagne@ensae.fr

Adèle Moreau
ENSAE Paris
adele.moreau@ensae.fr

Abstract

This study focuses on removing the background from datasets of faces to gauge the effect on the training and performance of facial generative models. We are also interested in the effect on the interpretability of the latent spaces of the models.

The code for this project can be found on our repository FaceAI-BGImpact.
The demo video of the web-application we made can be watched here.

1 Introduction

The realm of image generation has seen some major breakthroughs in recent years. An area of specific interest to us was that of the latent representations of complex images in generative models. Observing them gives us an insight into what best characterizes an image, in the eyes of the trained algorithm. When applied to a dataset of human faces, they can give us objective[1] insight into our most defining traits. Most image generation techniques will create an image from a scalar vector as input. During training, the algorithm will learn to map the dimensions of input vectors to different aspects of the image. This vector is what we are calling the latent vector, and its shape is the latent dimension. It can very well be interpreted as a compressed version of the image - indeed, a 128x128x3 RGB image is 49k+ scalars, and latent dimension is typically chosen to have 64-512 scalars.

The key observation that we made was that in many trained face generation algorithms, the background of the image occupied a large portion[2] of the captured variance - i.e. of the latent space. This means that some precious space from the latent vector is wasted on encoding what is behind the face, instead of the face itself.

There are a few neutral-background datasets, but none of them have nearly enough samples to train generative algorithms, and are mostly used to train or fine tune classifiers. An index of such datasets can be found here. For image generation, we should aim for 10k-100k+ unique images, but none here exceed 1k with a neutral background.

2 Related work

For the face generation task, some authors decided to circumvent the effect of the background by using a cropped version of their dataset to prevent various noisy backgrounds from interacting with the model [1].

[1] Depending on dataset bias

[2] Metrics discussed later - PCA eigenvalues

Advanced Machine Learning, Fall 2023, ENSAE Paris

The impact of the background removal on Convolutional Neural Networks has been proved for classification tasks [4]. Achieving a 12% higher accuracy on the task of Plant Disease Classification In-Situ for a dataset with images taken with a homogeneous background compared to that of a dataset with a cluttered background.

All these tricks suggest an important effect of the background on image generation. However, we have not found any studies on the importance of this effect of the background on image generation, and particularly face generation.

3 Methodology

Our idea was to create our own neutral-background dataset, from a normal face dataset, the Flicker-Faces-HQ dataset, often used for facial generation tasks.

To do this, we first decomposed the image by isolating the face and background using the `rembg` library. Then we created two datasets, one with a large Gaussian blur background and the other with a grey background. With over 70,000 images, our neutral background datasets are effective for training generative algorithms.

We focused on the 128x128 resolution version of the FFHQ dataset. The datasets we generated were uploaded to Kaggle on this link.

Our main hypothesis is that the removal of the background will have a positive impact on the performance and the training of the generative models. Furthermore, we expect the latent space to be more interpretable.

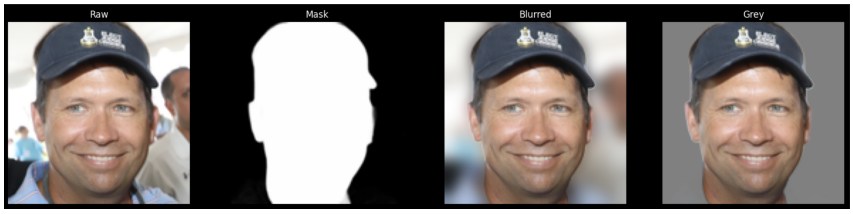


Figure 1: From left to right, original image, mask, blurred, greyed-out

To test our hypothesis, we trained models on both the original and the greyed-out versions of the same dataset. We picked versions of the two main categories of generative models: Auto-encoders (AE) and Generative Adversarial Networks (GAN).

3.1 Auto-encoders

Principle - Auto-encoders

Auto-encoders are unsupervised artificial neural networks whose main objective is to learn efficient representations of input data. An auto-encoder consists of two sub-networks: an encoder and a decoder. Firstly, the encoder compresses the input data (image) into a lower-dimensional representation. It reduces the dimensionality of the input data by mapping it to a set of hidden variables, often referred to as 'latent space'. The decoder then takes the encoded representation and reconstructs the original input data from it. The aim is to minimize the difference between the input and the reconstructed output, encouraging the model to learn a meaningful and compact representation.

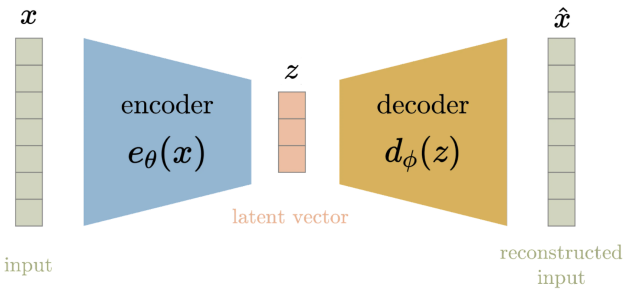


Figure 2: Auto-encoder architecture

Let $e_\theta : \mathcal{X} \to \mathcal{F}$ be the encoder function and $d_\phi : \mathcal{F} \to \mathcal{X}$ the decoder function. During training, input image x is fed to the encoder function and passed through a series of layers that reduce their dimensions to give a compressed latent vector z. The number, type and size of the layers, as well as the size of the latent space, are parameters that can be controlled. Next, the decoder function maps the $\mathcal{F}$ latent space at the bottleneck to the output, attempting to recreate the original image after some generalized non-linear compression.

The loss function used to train the neural network through the backpropagation procedure is as follows:

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|_2 = \|x - d_\phi(e_\theta(x))\|_2$$

There is a trade-off between quality and latent dimensionality. Below a certain point, the latent space is too small to capture the important features of the input data, and all the images look the same. This means that latent space of an auto-encoder also subject to irregularity due to overfitting.

Principle - Variational Auto-encoders

To tackle this problem, King et al. in 2013 [5] proposed the Variational Auto-encoder (VAE) architecture. It addresses the problem of irregular latent space in a conventional auto-encoder by mapping inputs to parameters of a probability distribution instead of a fixed vector. The VAE then imposes a constraint on this latent distribution, forcing it to be a Gaussian distribution, ensuring that the latent space is regularized.

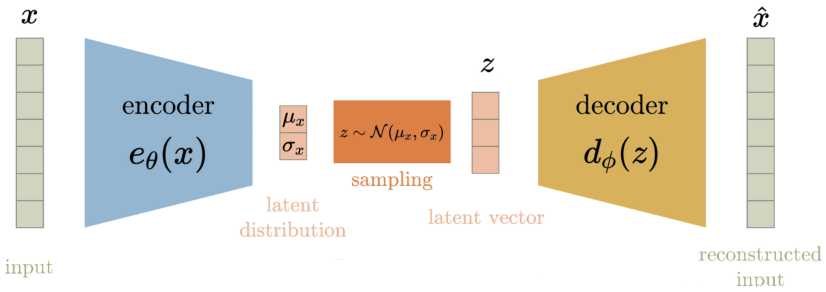


Figure 3: Variational Auto-encoder architecture

The VAE's encoder outputs the mean and standard deviation for each latent variable. The latent for the decoder is sampled from that distribution.

Rationale for choosing VAE

VAEs generally perform better than regular AEs in image generation tasks. First, by learning a probabilistic latent space representation, it allows for smoother transitions and better interpolation in the latent space, which is crucial for generating diverse and realistic images. Then, VAEs inherently include a regularization term in their objective function, which encourages the learned latent space to be well-structured. This regularization helps prevent overfitting and aids in disentangling meaningful features in the latent space.

This is especially important for us, since we want to be able to interpret the latent space of our models.

3.2 Generative Adversarial Networks

Principle - Generative Adversarial Networks

Introduced by Ian Goodfellow and al. in 2014 [2], the Generative Adversarial Network consists of a Generator and a Discriminator, two neural networks that are trained simultaneously by adversarial training.
Firstly, the Generator model attempts to model the distribution of data in order to generate realistic samples. It takes as input a random noise sample of fixed size from a latent space and generates new false images. The main objective is to fool the Discriminator by transforming the random noise into images indistinguishable from the real ones, making it harder to classify images as true or false. Then, acting like a binary classifier, the Discriminator attempts to classify the input data as real or generated one. Training stops when the generated images look like real images.

In other words, the Generator and the Discriminator play a two-player minimax game : $\min_G \max_D V(D, G)$, where $\mathbf{z}$ is a random noise vector drawn from the latent space distribution, $D(\mathbf{x})$ the output of the discriminator and $G(\mathbf{z})$ is the output of the generator.

Principle - ProGANs

One of the key features of the ProGAN architecture is the progressive nature of the layers, which enables the different visual characteristics of an image to be taken into account effectively. The concept is simple: the lower the layer and resolution, the greater the impact on coarser features. This granularity means that the inputs at each level can be modified independently, allowing precise control of the visual elements, from general characteristics to complex details, without affecting the other levels. The result is the generation of high-resolution images with a higher level of authenticity than previous models.

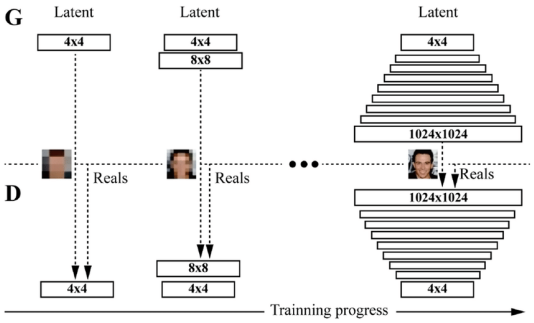


Figure 4: ProGAN training

In practice, the generator and discriminator are initialized with the full number of levels, but not all are used at first. Let's see the generation of an image at level 0 (i.e. at resolution 4x4):

$$\text{image}_{4x4}(z) = \text{ToRGB}_{4x4}(G_{4x4}(z))$$
$$\text{critic}_{4x4}(\text{image}) = D_{4x4}(\text{FromRGB}_{4x4}(\text{image}))$$

Now, at level 1 (resolution 8x8):

$$\text{image}_{8x8}(z) = \text{ToRGB}_{8x8}(G_{8x8}(G_{4x4}(z)))$$
$$\text{critic}_{8x8}(\text{image}) = D_{4x4}(D_{8x8}(\text{FromRGB}_{8x8}(\text{image})))$$

To prevent an explosion of gradients during the transition, we introduce $\alpha \in [0, 1]$, a parameter that controls the degree of blending between the two levels.

$$\text{image}_{8x8}(z, \alpha) = \alpha \times \text{image}_{8x8}(z) + (1 - \alpha) \times Upscale(\text{image}_{4x4}(z))$$

$$\text{critic}_{8x8}(\text{image}, \alpha) = \alpha \times \text{critic}_{8x8}(\text{image}) + (1 - \alpha) \times \text{critic}_{4x4}(DownScale(\text{image}))$$

The α parameter is increased linearly from 0 to 1 during the transition between levels 0 and 1. This process is repeated until the highest level is reached.

ProGAN yields much better results than conventional GANs, but the latent space is still very disorganized and hard to interpret.

Principle - StyleGAN

The main contribution of StyleGAN was to introduce a style-vector w in the generator.

In each forward pass, a Mapping network $\mathcal{M}$ is used to map the latent vector z to the intermediate latent space of styles, w. Then, the generator G uses w, instead of z, to generate the image. It also uses a learned constant z_0 to control the base style of the image, and some noise ϵ.

$$w = \mathcal{M}(z)$$

$$\text{image}_{4x4}(w) = \text{ToRGB}_{4x4}(G_{4x4}(w, z_0, \epsilon))$$

The key difference is that in the level 1,

$$\text{image}_{8x8}(w) = \text{ToRGB}_{8x8}(G_{8x8}(w, G_{4x4}(w, z_0, \epsilon), \epsilon))$$

The style vector w is passed to the generator at each level.

Rationale for choosing StyleGAN

The key features of StyleGAN over the other variants is that w is supposedly more interpretable than z.

Training metric - FID

To evaluate the performance of our models, we used the Fréchet Inception Distance (FID) metric. Unlike other models, the losses of GANs are not very informative. The FID metric is a measure of the similarity between two datasets of images.

The FID score is lower when the generated images closely match the real ones, indicating higher quality. The calculation involves two steps: feature extraction using the Inception model, a pre-trained deep learning model that extracts features and generates vector representations for each image. Then, the Fréchet Distance Calculation computes the mean and covariance of the feature vectors for both real and generated images, measuring the similarity between their distributions.

At resolution l and blending factor α, we're comparing blended real images $x(l, \alpha)$ and generated images $G_{l,\alpha}(w, z_0, \epsilon)$, with

$$x(l, \alpha) = \alpha \times x_l + (1 - \alpha) \times UpScale(x_{l-1})$$

One thing to keep in mind is that grey-background images weight 16% less on average (raw 1.87Mb, grey 1.58Mb), because data was removed from them. We will be comparing what we define as an adjusted FID for the original dataset as follows:

$$\text{FID}_{\text{adjusted}} = \text{FID}_{\text{raw}} \times (1 - 0.16)$$

Otherwise, the generation task is easier for the grey-background dataset, and the FID scores would be biased.

4 Training

Hardware setup

These models require a lot of computing power. Luckily, in the end, a family member was able to provide us with SSH access into a Windows-Subsystems-for-Linux (WSL) environment within their computer. This computer had a NVIDIA GeForce RTX 3080 GPU.

	M1 CPU	Colab Free	Colab Pro	SSP Cloud[3]	Family GPU
GPU	-	T4	V100	T4	RTX 3080
Time per epoch[4](minutes)	240	120	12	120	10
Cost per hour[5](cents€)	0.8	0	45.8[6]	0	4
Cost per epoch (cents€)	3.2	0	9.16	0	0.7
Cost for 400 epochs (€)	12.8	**0**	36.6	**0**	**2.8**
Time for 400 epochs (hours)	1600	800	80	800	**66.7**

Table 1: Time and Cost Analysis for the different GPU options - StyleGAN benchmark

The remote setup via SSH explains the large number of commits on the repository - we had to push our code to GitHub to be able to access it from the remote machine at each change we made.

VAE

VAEs converge very quickly, and epochs are short (40s on our hardware). Below is a graph of the evolution of the FID during training.

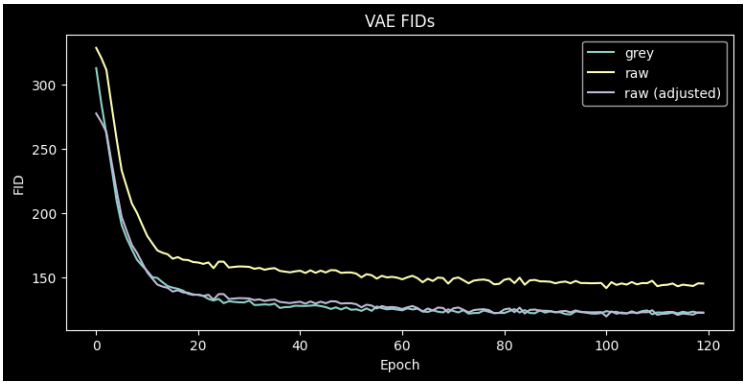


Figure 5: Training of grey and original VAEs. "adjusted" refers to the weight adjustment.

We can conclude that both models converge to a comparable FID (in their own scale), at fairly high FID values. This is expected, as VAEs are not very good at generating realistic images.

StyleGAN

StyleGAN is a very ressource-intensive model. We trained each levels (i.e. resolution, starting from 4x4) until convergence of the FIDs.

It is capital to ensure fairness in the training of the different datasets, for the results to be comparable. This means that they must train for the same number of epochs per level. To do so, we implemented a learning rate scheduler which slowly decreases the learning rate following an inverse sigmoid function during each level, and is reset at the beginning of the next level. This way, each model could train at appropriate learning rates during training.

[1]ENSAE provides us with an access to Datalab - SSP Cloud, with access to GPU instances

[2]Average over the different levels of StyleGAN - Resolution 4x4 is faster to train than 128x128.

[3]Taking into account electricity (M1 at 40W and RTX 3080 at 200W, at 0.20€/kWh).

[4]A Colab Pro subscription provides you with 100 compute units per month. These ran out in less than 24 hours of V100 training.

Each level (=resolution) was trained sequentially, starting from 4x4 up to 128x128. Below are the FID scores for the 5th and final level (128x128).

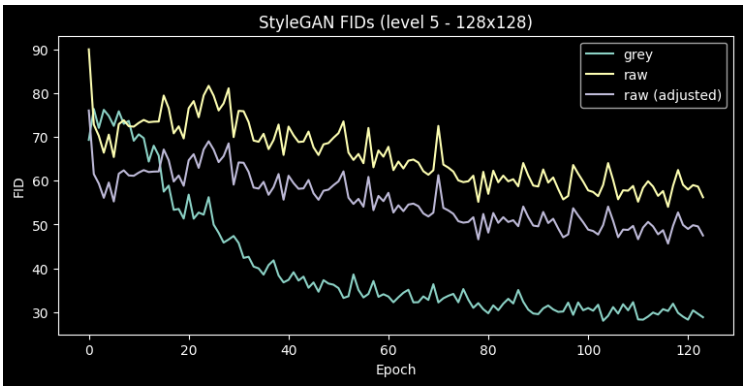


Figure 6: Training of grey and original StyleGANs. "adjusted" refers to the weight adjustment.

Observations:

1. **Even adjusted for image weight, the grey-background FIDs seems to converge faster and to a lower (better) value.**
2. The FID levels are much lower than for the VAEs, which is expected.

5 Sample and control of the latent space

Even with these models, sampling from the latent spaces is not straightforward.

VAE

For each input image x, the encoder outputs μ_x, σ_x. We can then sample z from $\mathcal{N}(\mu_x, \sigma_x)$. To generate now images, we can fit a Principal Component Analysis (PCA) on a large sample of z generated from the true images to get the inverse transform V_{VAE}. This allows us to sample and control the latent space using its eigenvectors in z space:

$$z(x) = V_{\text{VAE}} \cdot x$$

This approach was inspired by CodeParade's video "Computer Generates Human Faces", which introduced the idea of using PCA to control the latent space of a VAE.

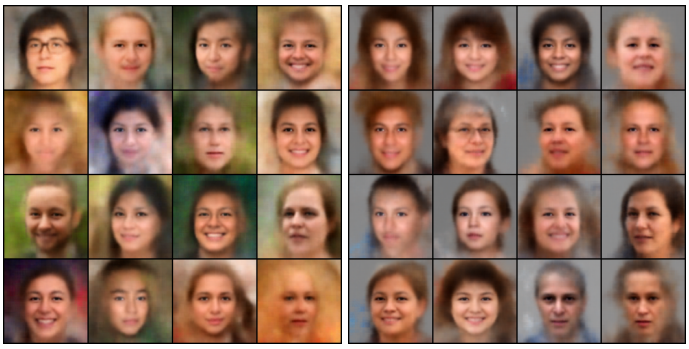


Figure 7: Samples from the raw and grey VAEs' latent spaces

It is hard to see any differences between the quality of the faces in the samples, which is expected given the equivalent adjusted FID scores.

StyleGAN

For StyleGAN, we want to sample w directly. For this, we will generate random z and use the Mapping network $\mathcal{M}$ to map them to w. We can then fit a PCA on a large sample of our synthetic w to get the inverse transform V_{StyleGAN}.

$$w(x) = V_{\text{StyleGAN}} \cdot x$$

We also implemented style-blending, which is a technique where you can give different w to different levels of the generator. The GANSpace paper [3] was very helpful in understanding how to control the latent space of StyleGAN.

Here are samples from the latent space, w:

Figure 8: Samples from the raw and grey GANs' style latent spaces

Overall, these are much more detailed and diverse than the VAE samples, but introduce a lot of visual artefacts. The grey-background does seem to produce better quality images, as the FIDs would indicate, but that remains subjective.

Let's now compare the strongest eigenvectors of the PCA of the latent spaces. Each row below is the effect of varying the corresponding eigenvector from -3 to 3, leaving the others at 0. Each row i represents the style vectors $(w_{i,t})_{t\in[0,3]} = t \cdot \delta_i \cdot V_{\text{StyleGAN}}$, where δ_i is the Kronecker delta, and V is the PCA inverse transform.

Original (raw) - StyleGAN w eigenvectors

$\lambda_{\text{raw},0} = 7.9$
BG brightness, hair

$\lambda_{\text{raw},1} = 5.8$
BG[7] red-ness, head rotation

$\lambda_{\text{raw},2} = 5.2$
BG blue-ness, hair

$\lambda_{\text{raw},3} = 4.8$
Glasses, gender, age

$\lambda_{\text{raw},4} = 4.4$
Glasses, gender, hair

The eigenvectors are very tangled with background variations, and the effects on the facial features themselves are not big.

[1] Background

Grey-background - StyleGAN w eigenvectors

$\lambda_{\text{grey, 0}} = 4.3$
Gender, hair

$\lambda_{\text{grey, 1}} = 3.1$
Gender, lipstick, hair

$\lambda_{\text{grey, 2}} = 2.5$
Head rotation

$\lambda_{\text{grey, 3}} = 2.2$
Gender, skin, hair

$\lambda_{\text{grey, 4}} = 1.9$
Glasses, gender, hair

The grey-background eigenvectors seem to have a much more pronounced and clear effect on the facial attributes, even though the eigenvalues are smaller.
This demonstrates that a lot of the variance is uselessly captured by the background in the original model, and shows the effectiveness of our method at tackling this problem.
This is clear when comparing $\lambda_{\text{raw},1}$ and $\lambda_{\text{grey, 2}}$, two eigenvectors that capture head rotation.

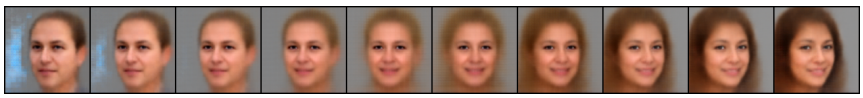
The same can be observed in the VAE eigenvectors:

Original (raw) - VAE z eigenvectors

$\lambda_{\text{raw},0} = 1.19$
BG color

Grey-background - VAE z eigenvectors

$\lambda_{\text{grey},0} = 1.18$
Hair, head rotation



A full list of the eigenvectors can be found in the Appendix.

References

[1] Jon Gauthier. Conditional generative adversarial nets for convolutional face generation. Technical report, Stanford University, 2014.

[2] Ian Goodfellow et al. Generative adversarial networks, 2014.

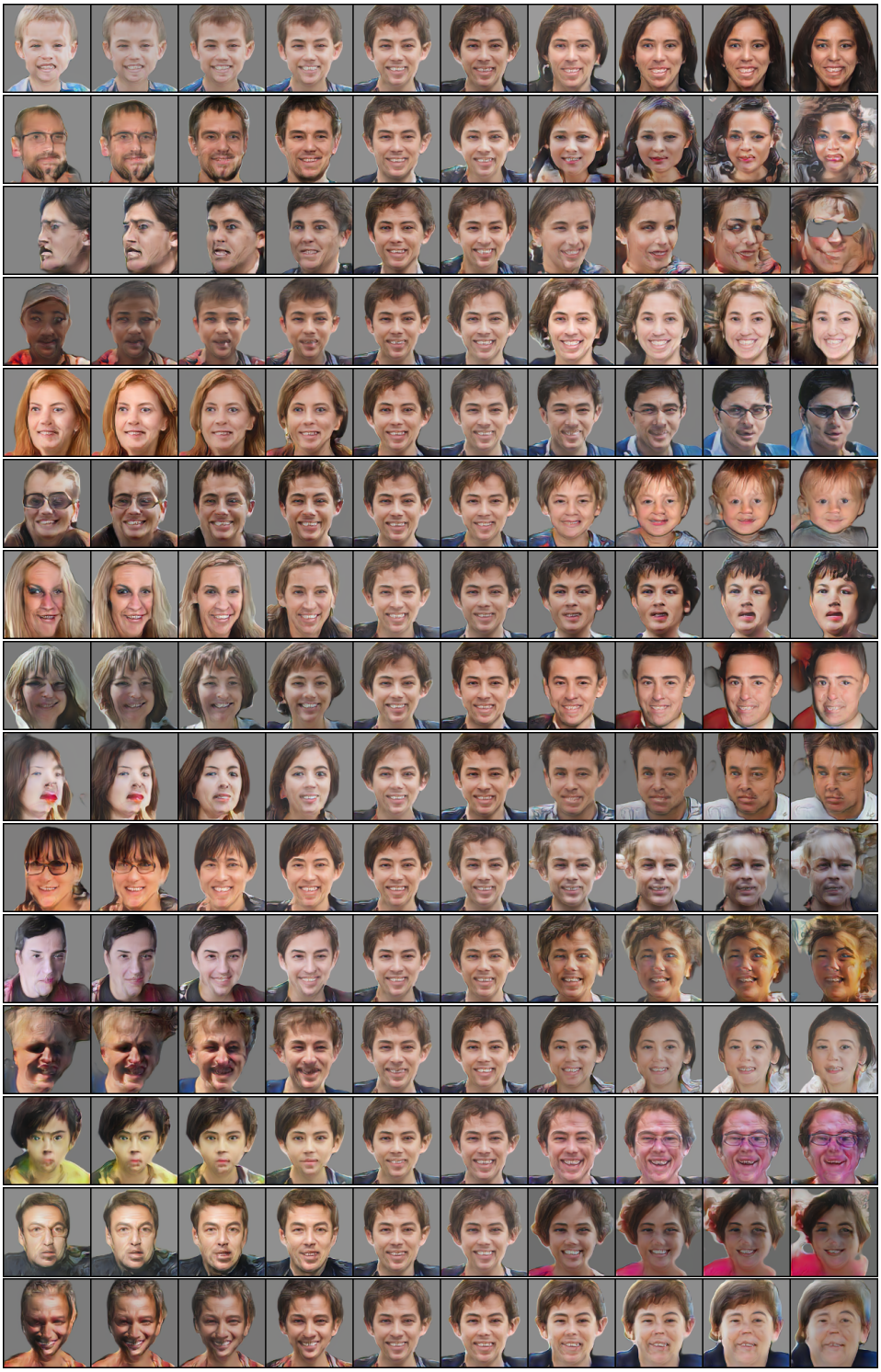
[3] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls, 2020.

[4] K. KC, Z. Yin, D. Li, and Z. Wu. Impacts of background removal on convolutional neural networks for plant disease classification in-situ. *Agriculture*, 11(827), 2021.

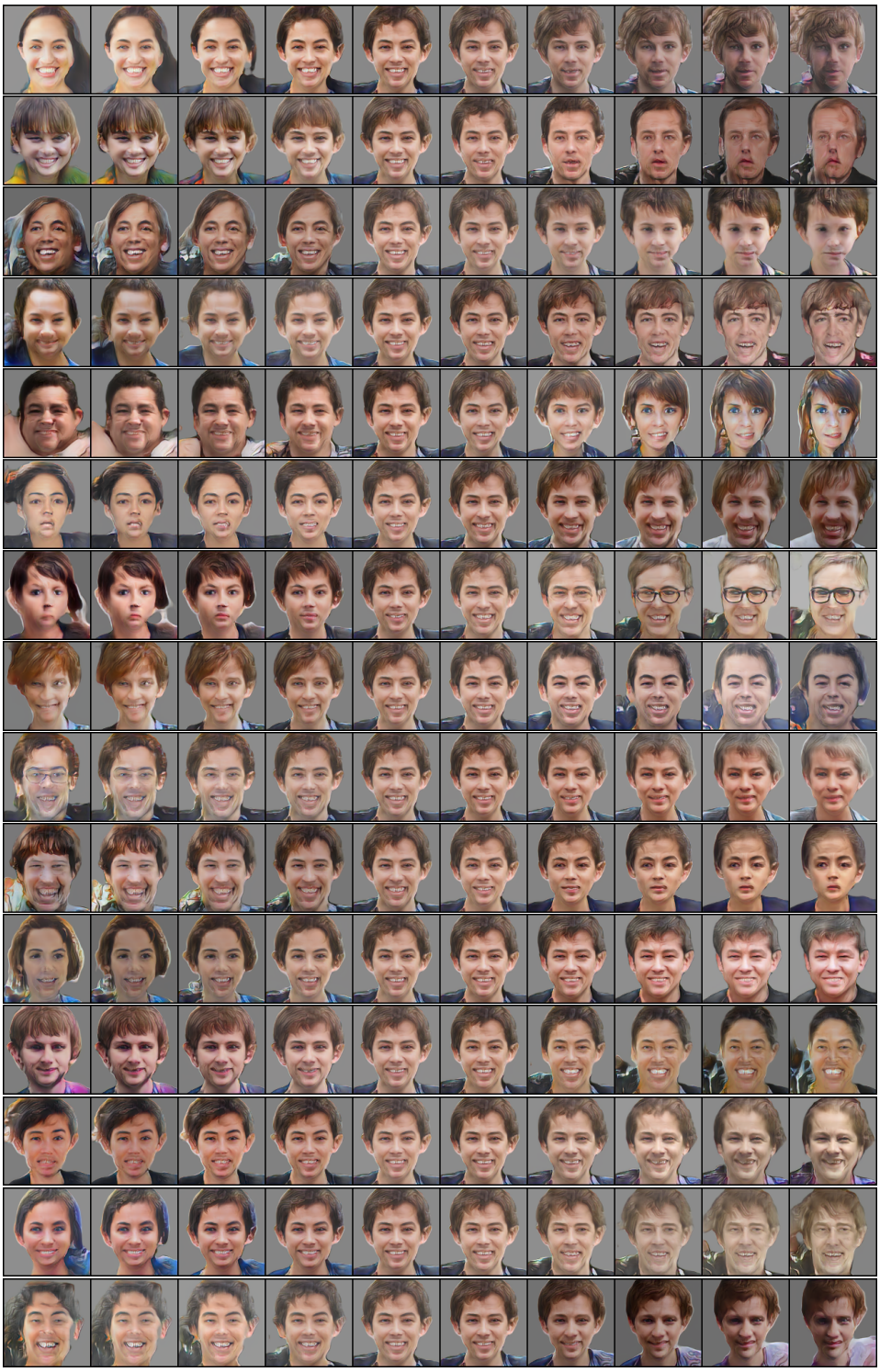
[5] Diederik Kingma and Max Welling. Auto-encoding variational bayes, 2013.

Appendix

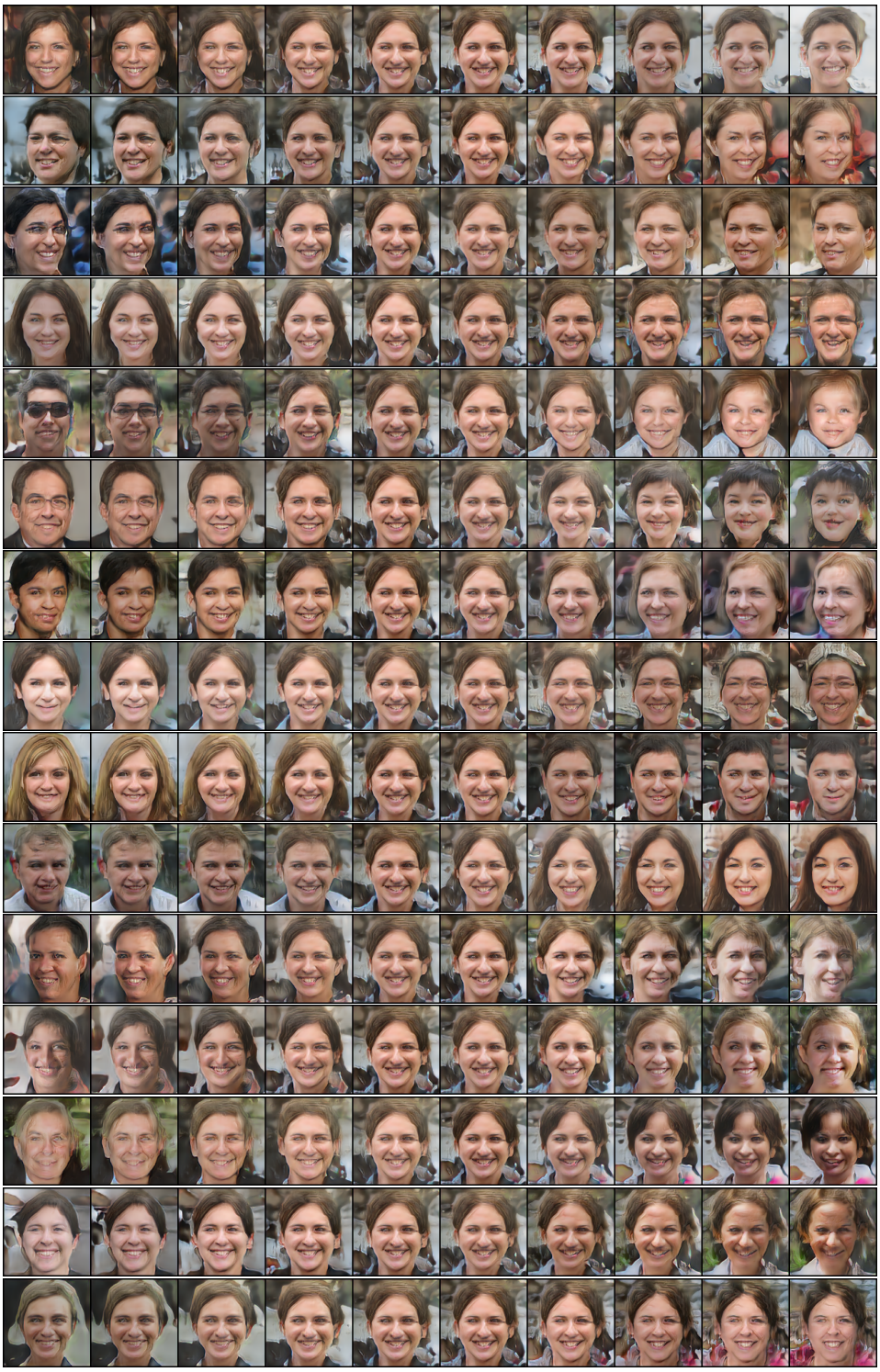
Grey-background - StyleGAN w eigenvectors 0-14



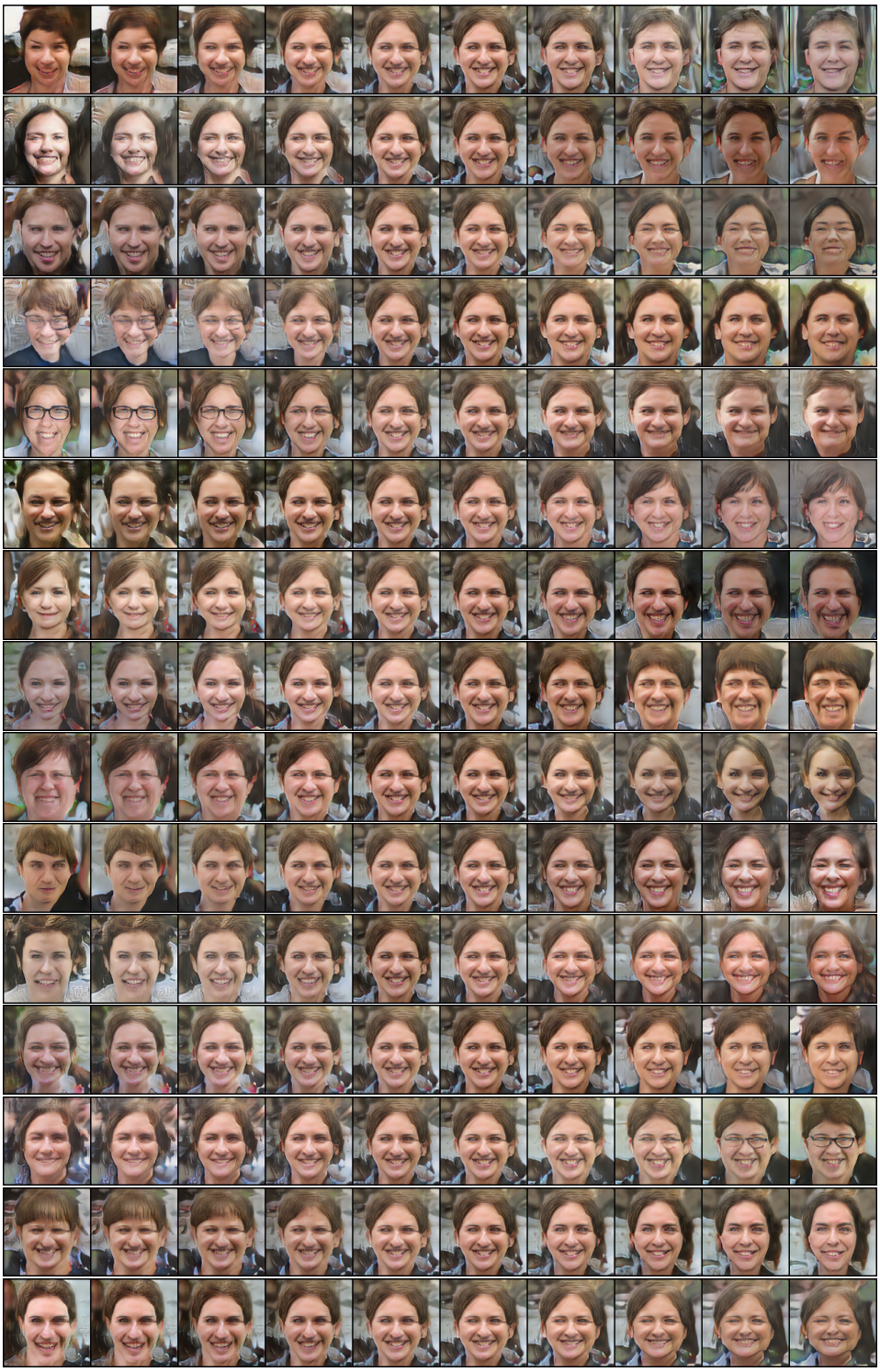
Grey-background - StyleGAN w eigenvectors 15-29



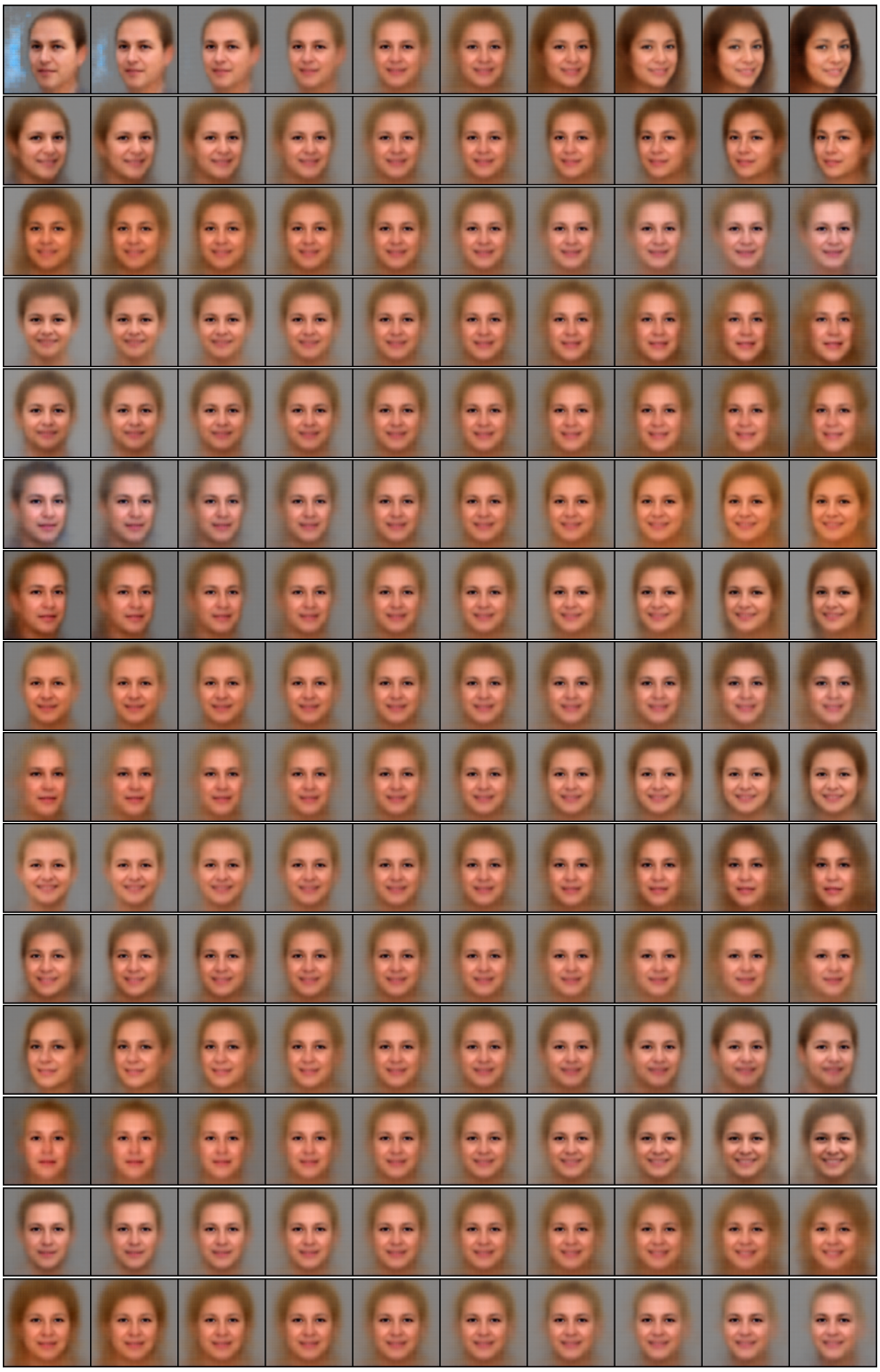
Original (raw) - StyleGAN w eigenvectors 0-14



Original (raw) - StyleGAN w eigenvectors 15-29



Grey-background - VAE z eigenvectors 0-14



Original (raw) - VAE z eigenvectors 0-14