UNDER REVIEW

RESCIENCE C

[Re] GANSpace: Discovering Interpretable GAN Controls

Vishnu Asutosh Dasu[1] and Midhush Manohar T.K.[2]
[1]TCS Research and Innovation, Bangalore, India – [2]Akamai Technologies, Inc.

Edited by
(Editor)

Reviewed by
(Reviewer 1)
(Reviewer 2)

Received
–

Published
–

DOI
–

A replication of GANSpace: Discovering Interpretable GAN Controls.

Reproducibility Summary

Scope of Reproducibility

The authors introduce a novel approach to analyze Generative Adversarial Networks (GANs) and create interpretable controls for image manipulation and synthesis. This is done by identifying important latent directions based on Principal Component Analysis (PCA) applied either in the latent space or the feature space. We aim to validate the claims and reproduce the results in the original paper.

Methodology

The code that was provided by the authors in Pytorch was reimplemented in Tensorflow 1.x for the pretrained StyleGAN and StyleGAN2 architectures. This was done with the help of the APIs provided by the original authors of these models.
The experiments were run on a laptop with an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz processor, 16GB RAM, NVIDIA GeForce GTX 1060 with Max-Q Design (6GB VRAM) GPU, and Ubuntu 18.04.5 LTS.

Results

We were able to reproduce the results and verify the claims made by the authors for the StyleGAN and StyleGAN2 models by recreating the modified images, given the seed and other configuration parameters. Additionally, we also perform our own experiments to identify new edits and show that edits are transferable across similar datasets using the techniques proposed by the authors.

What was easy

The paper provides detailed explanations for the different mathematical concepts that were involved in the proposed method. This, augmented with a well-structured and documented code repository, allowed us to understand the major ideas in a relatively short period of time. Running the experiments using the original codebase was straightforward and highly efficient as well, as the authors have taken additional steps to employ batch processing wherever possible.

Correspondence should be addressed to Vishnu Asutosh Dasu (vishnu98dasu@gmail.com)
The authors have declared that no competing interests exists.
Code is available at https://github.com/midsterx/ReGANSpace.
Open peer review is available at https://openreview.net/forum?id=BtZVD2f7n0F.

What was difficult

Originally we were attempting to recreate identical images with zero delta in the RGB values. However, due to differences in the random number generators between PyTorch-CPU, PyTorch-GPU and Numpy, the random values were not the same even with the same seed. This resulted in minute differences in the background artifacts of the generated images. Additionally, there is a lack of open source Tensorflow 1.x APIs to access the intermediate layers of the BigGAN model. Due to time constraints, we were unable to implement these accessors and verify the images that the authors of GANSpace created using BigGAN.

Communication with original authors

While conducting our experiments, we did not contact the original authors. The paper and codebase were organized well and aided us in effectively reproducing and validating the authors' claims.

1 Introduction

Generative Adversarial Networks (GANs) [1] are a type of machine learning framework where two neural networks, the discriminator and the generator, compete with each other in a zero-sum game. The generator tries to trick the discriminator into believing that artificially generated samples belong to real data.
GANs have proven to be powerful image synthesis tools, which are capable of producing high quality images. However, they provide little control over the features of the generated image. Existing solutions to add user control over the generated images require expensive supervised training on latent vectors.
GANSpace [2] proposes a simple technique to discover interpretable GAN controls in an unsupervised manner. This is done by identifying important latent directions based on Principal Component Analysis (PCA) applied either on the latent space or the feature space. The author's experiments on StyleGAN [3], StyleGAN2 [4] and BigGAN512-deep [5] demonstrate that layer-wise decomposition of PCA directions leads to many interpretable controls, which affects both low and high level attributes of the output image.

2 Scope of reproducibility

For our reproduction study, we aim to validate the effectiveness of the proposed technique in offering powerful interpretable controls on the output images in an unsupervised manner.
The following claims of the paper have been verified and tested successfully:

- PCA can be used to highlight important directions in the GAN's latent space.
- The GAN's output can be controlled easily in an unsupervised fashion.
- The earlier components control the higher-level aspects of an image, while the later directions primarily affect the minute details.

3 Methodology

The principal components of a collection of points in real coordinate space. The principal components are a sequence of p unit vectors, where the i^{th} vector is a direction of the line that best fits the data while beging orthogoanl to the remaining $i-1$ vectors. Principal Component Analysis (PCA) is an unsupervised algorithm used to compute the principal components and perform a change of basis of the data, using one or more of the computed components [6]. It is commonly used in exploratory data analysis and for dimensionality reduction when dealing with high-dimensional noisy data. The authors of GANSpace propose a technique for identifying interpretable controls in an unsupervised fashion on pretrained GANs using PCA. Specifically, they show that layer-wise perturbations along the principal components generated using PCA on the latent space of StyleGAN based networks can be used to generate human-interpretable transformations on the synthesized images.
Mathematically, a GAN can be expressed as a neural network $G(\mathbf{z})$ that generates an image $I : \mathbf{z} \sim p(\mathbf{z}), I = G(\mathbf{z})$. Here, $p(\mathbf{z})$ is a probability distribution from which the latent vector $\mathbf{z}$ is sampled. The network $G(\mathbf{z})$ can be further decomposed into L intermediate layers $G_1 \dots G_L$. In the StyleGAN/StyleGAN2 models, the input to the first layer is a constant y_0. The output and input to the remaining layers is computed as

$$y_i = G_i(y_{i-1}, \mathbf{w}), \text{ where } \mathbf{w} = M(\mathbf{z}) \tag{1}$$

M is a an 8-layer multilayer perceptron which is a non-linear function of z. The number of layers L depend on the resolution of the generated. At each layer, the generated image is upsampled by a factor of 2.
The images generated by StyleGAN and StyleGAN2 can be controlled by identifying principal axes of $p(\mathrm{w})$, which is the probability distribution of the output of the mapping network M. First, we sample N latent vectors $\mathrm{z}_{1:N}$ and compute the corresponding $\mathrm{w}_i = M(\mathrm{z}_i)$. The PCA of these $\mathrm{w}_{1:N}$ values gives us the basis V for $\mathcal{W}$. The output attributes of a new image given by w can then be controlled by varying the PCA coordinates of x before feeding them into the synthesis network.

$$\mathrm{w'} = \mathrm{w} + \mathrm{Vx} \tag{2}$$

Each entry x_k of x is a separate control parameter which can be modified to update the desired attributes of the output image.
We follow the same notation used by the authors to denote edit directions in this report. $E(\mathrm{v}_i, j-k)$ means moving along component v_i from layers j to k. Identifying specific edits, for example changing the color of car, is done via exploratory analysis using trial-and-error. The authors have created a GUI based application for this purpose.

3.1 Model descriptions

We use NVIDIA's official implementation of StyleGAN [1] and StyleGAN2 [2] models. The original code uses a PyTorch/NumPy implementation of StyleGAN and StyleGAN2 which creates a PyTorch model and copies the weights from NVLabs' implementations which are in Tensorflow. However, we directly use the NVLabs' APIs with NumPy and make changes to the official GANSpace codebase to support the same.

3.2 Datasets

The experiments in the paper were performed using the FFHQ, LSUN Car, CelebA-HQ, Wikiart, Horse and Cat datasets. The official Tensorflow implementation of StyleGAN contains links to download pretrained models on FFHQ, LSUN Car, Wikiart, Horse and Cat. The models trained on Wikiart were downloaded from awesome-pretrained StyleGAN[3].
In addition to the datasets using by the authors, we also perform our own experiments on the Beetles and Anime datasets which were downloaded from awesome-pretrained StyleGAN2[4].

3.3 Experimental setup

All the experiments were conducted on a laptop with an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz processor, 16GB RAM, NVIDIA GeForce GTX 1060 with Max-Q Design (6GB VRAM) GPU, and Ubuntu 18.04.5 LTS. The generated images from our experiments were evaluated visually to determine whether the edits were working as expected.

4 Results

We were able to reproduce the results and verify the claims (mentioned in Section 2) made by the authors for the StyleGAN and StyleGAN2 models by recreating the modified images, given the configuration parameters. Additionally, we also perform our own

[1] https://github.com/NVlabs/stylegan
[2] https://github.com/NVlabs/stylegan2
[3] https://github.com/justinpinkney/awesome-pretrained-stylegan
[4] https://github.com/justinpinkney/awesome-pretrained-stylegan2

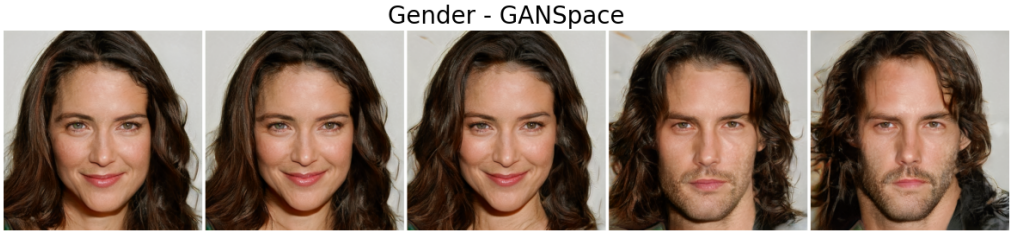
experiments to provide additional results that validate the effectiveness of the technique employed by GANSpace.

4.1 Effectiveness of PCA

Figure 1. Sequences of image edits performed using control discovered with StyleGAN2 cars: “Initial Image" → “Change Color" → “Add Grass" → “Rotate" → "Change Type"

Figure 1 highlights the effectiveness of PCA on changing the low and high level attributes of the image. We are able to control object shape, colour and pose as well as nuanced landscape attributes.
The edit directions corresponding to each of the edits are: $E(\mathrm{v}_{22}, 9-10)$ ("Change Color"), $E(\mathrm{v}_{11}, 9-10)$ ("Add Grass"), $E(\mathrm{v}_{0}, 0-4)$ ("Rotate") and $E(\mathrm{v}_{16}, 3-5)$ ("Change type").

4.2 Unsupervised vs Supervised methods



(a) Edit directions identified by PCA ($E(\mathrm{v}_1, 0-1)$)

(b) Edit directions identified by supervised methods [7]

Figure 2. Comparison of edits using unsupervised and supervised methods

The original authors point out that previous methods for finding interpretable directions in GAN latent spaces require outside supervision, such as labeled training images or pre-trained classifiers, whereas GANSpace aims to automatically identify variations intrinsic to the model without supervision. This has been validated using the CelebA HQ Faces dataset by comparing the edit directions found through PCA to those found in previous works using supervised methods.

4.3 Effect of different components

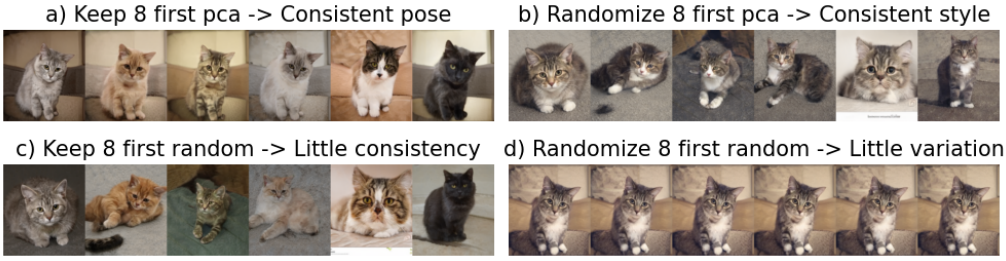


Figure 3. Illustration of the significance of the principal components as compared to random directions in the intermediate latent space of StyleGAN2.

The original authors claim that the earlier components primarily control the geometry and other high-level aspects, while the lower components capture minute details. This has been illustrated in Figure 3. Fixing and randomizing the early principal components shows a separation between pose and style. In contrast, fixing and randomizing randomly-chosen directions does not yield a similar meaningful decomposition.

4.4 Additional results not present in the original paper

New edits – We identify new edits on the Stylegan2 Beetles dataset. Edit $E(\mathrm{v}_2, 0-17)$, referred to as "Patterns", adds a pattern on the shell of the beetle. The generated pattern varies depending on the seed used to sample w.

(a) Beetle generated with seed 1819967864

(b) Beetle generated with seed 1

Figure 4. "Patterns" edit applied on the output images of StyleGAN2 Beetles

Truncation Trick on StyleGAN – The "truncation trick" is a procedure applied to the latent vectors to improve the quality of the generated images at the expense of diversity in the images. It does this by sampling the latent vectors from a truncated distribution that is closer to the average of the latent vectors sampled during training, thereby reducing the variance of the latent vectors used during inference. The authors of [5] show that using the truncation trick improves the FID score.
In the StyleGAN/StyleGAN2 models, the truncation trick is applied on the output of the mapping network $w = M(z)$. During the training process, a running average w_{avg}

of the latents is comptued. Later, the latents sampled during inference are truncated to lie close to w_{avg}. Equation 3 shows the truncation process on StyleGAN/StyleGAN2 models.

$$w' = w_{avg} + \psi(w - w_{avg}) \tag{3}$$

During our experiments, we notices that the original authors use the truncation trick on images generated using StyleGAN2 to improve their quality. However, this is not enabled for StyleGAN images. We found that enabling truncation while applying edits on StyleGAN images improved their quality as well. We demonstrate this using the Wikiart dataset using the "Head Rotation" ($E(\mathrm{v}_7, 0-1)$) and "Simple Strokes" ($E(\mathrm{v}_9, 8-14)$ edits. In Figure 5, we can see that the generated faces contain less noise and artifacts when the truncation trick is used. For example, the lower half of the person's face in the "Head Rotation" image does not contain as much noise as their counterpart which does not employ the truncation trick. Here, we can also observe the change in the generated images as truncation psi is decreased a lower value of 0.25. This truncates the sampled latents to lie very close to the average and results in images that look very similar to each other. If truncation psi is set to 0, then according to Equation 3, we can see that the truncated latent w' is always equal to w_{avg}.

5 Discussion

After performing our experiments, we feel that the results justify the claims of the paper. This is further bolstered by the fact that the proposed method worked on different datasets which were not covered by the original authors.

5.1 What was easy

The paper provides detailed explanations for the different mathematical concepts that were involved in the proposed method. This, augmented with a well-structured and documented code repository, allowed us to understand and verify the major ideas in a relatively short period of time. Additionally, the paper provided a lot of examples on various datasets to demonstrate exactly how their algorithm works. The authors ensured that all the figures in the paper had accompanying code to recreate them.
NVIDIA's implementation of StyleGAN and StyleGAN2 provided access to well written API's which we could integrate easily into the author's codebase.

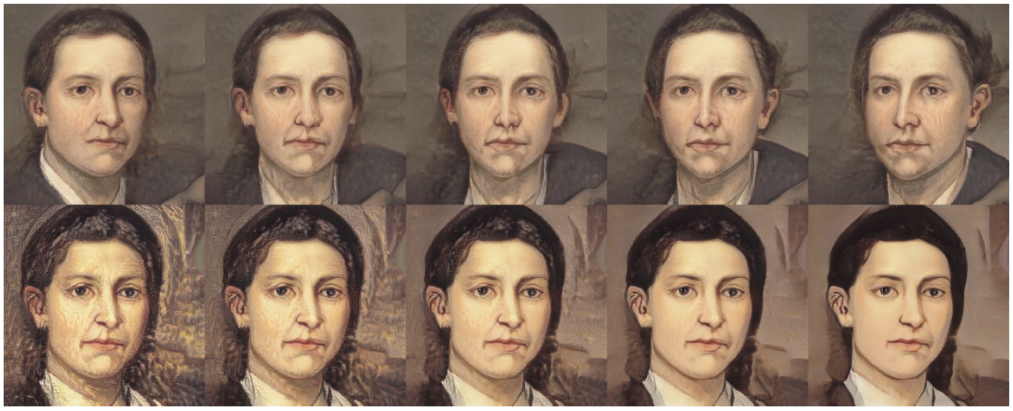
5.2 What was difficult

While running our experiments, we noticed that there was a small difference in the RGB values of the recreated images. This was due to the difference in the random values generated by PyTorch-CPU, PyTorch-GPU and Numpy random number generators even when seeded with the same seed. The noise variables in the StyleGAN networks were not identical because of this. This resulted in minute differences in background artifacts of the images.

Python Library	Random Number
PyTorch 1.3.1 (CPU)	0.3367
PyTorch 1.3.1 (GPU)	0.1940
Numpy 1.20.1	0.49671415

Table 1. Random values generated using different Python libraries seeded with 42

We were not able to replicate the author's experiments on BigGAN-512 deep due to time constraints.



(a) "Head Rotation" and "Simple Strokes" edits on StyleGAN Wikiart with truncation psi set to 0.25

(b) "Head Rotation" and "Simple Strokes" edits on StyleGAN Wikiart with truncation psi set to 0.7

(c) "Head Rotation" and "Simple Strokes" edits on StyleGAN Wikiart without truncation psi

Figure 5. Quality of images generated by StyleGAN before and after applying the "truncation trick".

5.3 Communication with original authors

While conducting our experiments, we did not contact the original authors. The paper and codebase were organized well and aided us in effectively reproducing and validating the authors' claims.

References

1. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. "Generative Adversarial Nets." In: **Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada**. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. 2014, pp. 2672–2680.
2. E. Härkönen, A. Hertzmann, J. Lehtinen, and S. Paris. "GANSpace: Discovering Interpretable GAN Controls." In: **Proc. NeurIPS**. 2020.
3. T. Karras, S. Laine, and T. Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks." In: **IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019**. Computer Vision Foundation / IEEE, 2019, pp. 4401–4410.
4. T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. "Analyzing and Improving the Image Quality of StyleGAN." In: **2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020**. IEEE, 2020, pp. 8107–8116.
5. A. Brock, J. Donahue, and K. Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis." In: **7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019**. OpenReview.net, 2019.
6. Wikipedia contributors. **Principal component analysis — Wikipedia, The Free Encyclopedia**. [Online; accessed 2-May-2022]. 2022.
7. Y. Shen, J. Gu, X. Tang, and B. Zhou. "Interpreting the Latent Space of GANs for Semantic Face Editing." In: **2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020**. IEEE, 2020, pp. 9240–9249.