

MO433 - Unsupervised Learning

Image Generation by GANs

Alexandre Xavier Falcão

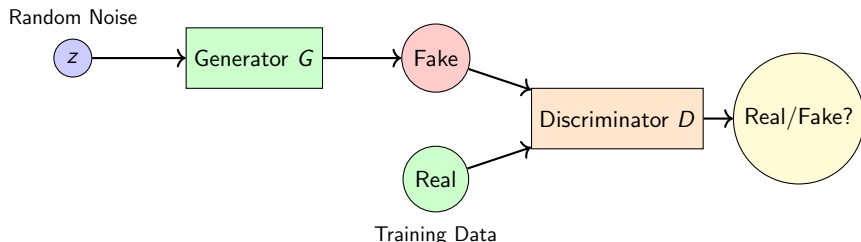
Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

What are Generative Adversarial Networks?

Two neural networks compete in a game:

- ▶ **Generator (G):** Creates fake data to fool the discriminator.
- ▶ **Discriminator (D):** Distinguishes real data from fake data.



We are interested in the generator.

Agenda: GANs from basics to advanced applications

1. Foundations of GANs.

- ▶ Generator vs. Discriminator game theory.
- ▶ Minimax optimization.

2. Deep Convolutional GAN (DCGAN).

- ▶ Architectural improvements.
- ▶ Stability through design choices.

3. Conditional GAN (cGAN).

- ▶ Controlled generation via label embeddings.
- ▶ *Applications: Class-specific generation, attribute manipulation.*

4. Advanced Applications.

- ▶ Super-Resolution GAN (SRGAN): Perceptual loss and content reconstruction.
- ▶ StyleGAN2: Latent space manipulation and style transfer.
- ▶ *Applications: Image enhancement, creative tools.*

Generator vs. Discriminator: The Adversarial Game

Generator (G)

Goal: Fool the discriminator

Input: Random noise $z \sim p_z(z)$

Output: Synthetic data $G(z)$

Objective:

$$\max_G \mathbb{E}_{z \sim p_z} [\log D(G(z))]$$

Maximize probability that D classifies fake as real

Discriminator (D)

Goal: Detect fake data

Input: Real x or fake $G(z)$

Output: Probability

$$D(x) \in [0, 1]$$

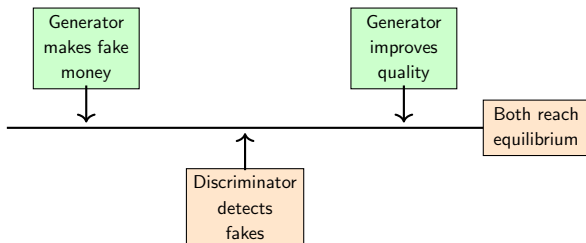
Objective:

$$\begin{aligned} \max_D \mathbb{E}_{x \sim p_{data}} [\log D(x)] \\ + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))] \end{aligned}$$

Correctly classify real as real, fake as fake

Game Theory Perspective

Think of GANs as a counterfeiter (Generator) vs. police (Discriminator):



Nash Equilibrium

Training converges when:

- ▶ Generator produces perfect fakes: $G(z) \sim p_{data}(x)$.
- ▶ Discriminator cannot distinguish: $D(x) = D(G(z)) = \frac{1}{2}$.

Minimax Optimization

GANs solve a minimax game:

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

Breaking it down:

- ▶ **First term:** $\mathbb{E}_{x \sim p_{data}} [\log D(x)]$.
Discriminator wants to maximize (assign high probability to real data).
- ▶ **Second term:** $\mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))]$
Discriminator wants to maximize (assign low probability to fake data).
Generator wants to minimize (fool discriminator).
- ▶ **Min-Max:** Discriminator maximizes V , Generator minimizes it.

Training Dynamics: Alternating Optimization

Step 1: Train Discriminator (fix G, update D)

- ▶ Sample real data $x \sim p_{data}$
- ▶ Sample noise $z \sim p_z$ and generate fake $G(z)$
- ▶ Update D to maximize:

$$\max_D [\log D(x) + \log(1 - D(G(z)))]$$

Step 2: Train Generator (fix D, update G)

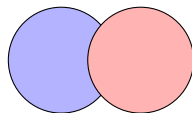
- ▶ Sample noise $z \sim p_z$
- ▶ Update G to minimize:

$$\min_G \log(1 - D(G(z))) \quad \text{or equivalently} \quad \max_G \log D(G(z))$$

Note: In practice, we maximize $\log D(G(z))$ instead of minimizing $\log(1 - D(G(z)))$ to avoid vanishing gradients early in training.

Training Process Visualization

Early Training

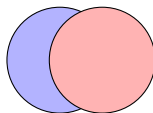


Real data Fake data

Easy to distinguish

Training
→

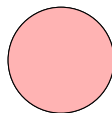
Mid Training



Getting closer

Training
→

Converged



Indistinguishable

$D(x) = 0.5$

- ▶ **Blue:** Real data distribution $p_{data}(x)$
- ▶ **Red:** Generated data distribution $p_g(x|z)$
- ▶ **Goal:** $p_g(x|z) \rightarrow p_{data}(x)$

Loss Functions in Practice

Discriminator Loss

$$\mathcal{L}_D = -[\mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]]$$

Binary Cross-Entropy:

- ▶ Label real data as 1: $\mathcal{L}_D^{real} = -\log D(x)$
- ▶ Label fake data as 0: $\mathcal{L}_D^{fake} = -\log(1 - D(G(z)))$

Generator Loss

Original formulation:

$$\mathcal{L}_G = \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

Non-saturating alternative (used in practice):

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p_z}[\log D(G(z))]$$

Provides stronger gradients when discriminator is confident

Training Algorithm: Step 1 - Train Discriminator

Discriminator Update

Goal: Maximize ability to distinguish real from fake.

Algorithm:

1. Sample minibatch of m real examples: $\{x^{(1)}, \dots, x^{(m)}\}$.
2. Sample minibatch of m noise vectors: $\{z^{(1)}, \dots, z^{(m)}\}$.
3. Generate fake data: $\{\tilde{x}^{(i)} = G(z^{(i)})\}$.
4. Compute discriminator loss:

$$\mathcal{L}_D = -\frac{1}{m} \sum_{i=1}^m \left[\log D(x^{(i)}) + \log(1 - D(\tilde{x}^{(i)})) \right]$$

5. Update D by gradient descent: $\theta_D \leftarrow \theta_D - \eta \nabla_{\theta_D} \mathcal{L}_D$.

Training Algorithm: Step 2 - Train Generator

Generator Update

Goal: Maximize probability of fooling discriminator.

Algorithm:

1. Sample minibatch of m noise vectors: $\{z^{(1)}, \dots, z^{(m)}\}$.
2. Generate fake data: $\{\tilde{x}^{(i)} = G(z^{(i)})\}$.
3. Compute generator loss:

$$\mathcal{L}_G = -\frac{1}{m} \sum_{i=1}^m \log D(\tilde{x}^{(i)})$$

4. Update G by gradient descent: $\theta_G \leftarrow \theta_G - \eta \nabla_{\theta_G} \mathcal{L}_G$.

Complete Training Loop

Alternate between Step 1 (train D) and Step 2 (train G) for each minibatch. *Often train D multiple times per G update for stability.*

Theoretical Convergence

Theorem (Goodfellow et al., 2014)

The global optimum of the minimax game is achieved when:

$$p_g(x|z) = p_{data}(x)$$

At this point: $D^(x) = \frac{1}{2}$ everywhere.*

Intuition

- ▶ When $p_g = p_{data}$, discriminator cannot distinguish.
- ▶ Best strategy for D is to guess randomly: $D(x) = 0.5$.
- ▶ Generator has no incentive to change.
- ▶ This is the Nash equilibrium.

In practice, convergence is not guaranteed due to non-convexity and training dynamics.

Example 1: Vanilla GAN on MNIST

Objective

Generate handwritten digits (0-9) from random noise to demonstrate basic GAN concepts on a simple, controlled dataset.

Why MNIST?

- ▶ Simple grayscale images (28×28 pixels).
- ▶ Low dimensionality (784 features when flattened).
- ▶ Perfect for understanding GAN fundamentals.
- ▶ Fast training (minutes on GPU).

Challenge: Can fully-connected networks learn to generate realistic images?

Vanilla GAN: Architecture

Generator:

- ▶ Input: $z \in \mathbb{R}^{100}$ (random noise)
- ▶ Linear: $100 \rightarrow 256$
- ▶ LeakyReLU(0.2)
- ▶ Linear: $256 \rightarrow 512$
- ▶ LeakyReLU(0.2)
- ▶ Linear: $512 \rightarrow 1024$
- ▶ LeakyReLU(0.2)
- ▶ Linear: $1024 \rightarrow 784$
- ▶ Tanh (output in $[-1, 1]$)

Output: 28×28 image

Discriminator:

- ▶ Input: $x \in \mathbb{R}^{784}$ (flattened image)
- ▶ Linear: $784 \rightarrow 1024$
- ▶ LeakyReLU(0.2), Dropout(0.3)
- ▶ Linear: $1024 \rightarrow 512$
- ▶ LeakyReLU(0.2), Dropout(0.3)
- ▶ Linear: $512 \rightarrow 256$
- ▶ LeakyReLU(0.2), Dropout(0.3)
- ▶ Linear: $256 \rightarrow 1$
- ▶ Sigmoid (probability)

Architecture Type: Fully-connected (MLP) - no convolutions!

Vanilla GAN: Loss Functions

Standard GAN loss (Binary Cross-Entropy):

Discriminator Loss

$$\mathcal{L}_D = -\mathbb{E}_{x \sim p_{data}} [\log D(x)] - \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))]$$

Implemented as:

- ▶ Real images labeled as 1: `BCELoss(D(real), 1)`.
- ▶ Fake images labeled as 0: `BCELoss(D(fake), 0)`.

Generator Loss

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p_z} [\log D(G(z))]$$

Implemented as:

- ▶ Fake images labeled as 1: `BCELoss(D(fake), 1)`.

Vanilla GAN: Results



64 randomly generated digits after 200 epochs.

Observations:

- ▶ Recognizable digits generated.
- ▶ Diverse outputs (different styles).
- ▶ Some blurry/unclear samples.
- ▶ Limited detail due to MLP architecture.

Conclusion: Vanilla GANs work for simple images but struggle with complex structures.

Example 2: DCGAN for Face Generation

Objective

Generate realistic 64×64 RGB face images using convolutional architecture to capture spatial hierarchies and fine details.

Why DCGAN?

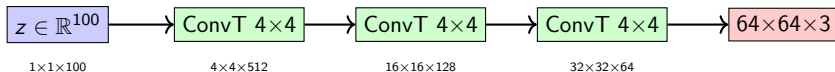
- ▶ Vanilla GAN fails on complex images (faces, objects).
- ▶ Convolutional layers preserve spatial structure.
- ▶ Proven architecture for stable training.
- ▶ High-quality outputs on structured data.

Improvement: Replace fully-connected layers with convolutional/transposed-convolutional layers.

DCGAN: Architecture Principles

Design Guidelines (Radford et al., 2015):

1. Replace pooling with strided convolutions (D) and transposed convolutions (G).
2. Use BatchNorm in both G and D.
3. Remove fully-connected hidden layers.
4. Use ReLU in G (except output: Tanh).
5. Use LeakyReLU in D.



Generator progressively upsamples from 1×1 to 64×64 .

DCGAN: Detailed Architecture

Generator:

- ▶ Input $z \in \mathbb{R}^{100 \times 1 \times 1}$.
- ▶ ConvT 100 \rightarrow 512 (4×4 , $s=1$) + BN + ReLU.
- ▶ ConvT 512 \rightarrow 256 (4×4 , $s=2$) + BN + ReLU.
- ▶ ConvT 256 \rightarrow 128 (4×4 , $s=2$) + BN + ReLU.
- ▶ ConvT 128 \rightarrow 64 (4×4 , $s=2$) + BN + ReLU.
- ▶ ConvT 64 \rightarrow 3 (4×4 , $s=2$) + Tanh.

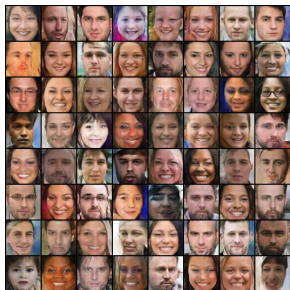
Output: $64 \times 64 \times 3$.

Discriminator:

- ▶ Input $64 \times 64 \times 3$.
- ▶ Conv 3 \rightarrow 64 (4×4 , $s=2$) + LeakyReLU(0.2).
- ▶ Conv 64 \rightarrow 128 (4×4 , $s=2$) + BN + LeakyReLU(0.2).
- ▶ Conv 128 \rightarrow 256 (4×4 , $s=2$) + BN + LeakyReLU(0.2).
- ▶ Conv 256 \rightarrow 512 (4×4 , $s=2$) + BN + LeakyReLU(0.2).
- ▶ Conv 512 \rightarrow 1 (4×4 , $s=1$) + Sigmoid.

Output: probability.

DCGAN: Results



64 randomly generated 64×64 faces after 25 epochs.

Observations:

- ▶ Realistic facial structures (eyes, nose, mouth).
- ▶ Smooth textures and proper color.
- ▶ Diverse outputs (different poses, expressions).
- ▶ Occasional artifacts (weird backgrounds, distortions).

Conclusion: Convolutional architecture dramatically improves quality over vanilla GAN!

Example 3: Conditional GAN (cGAN)

Objective

Generate faces with controllable attributes (e.g., gender) by conditioning both Generator and Discriminator on class labels.

Why Conditional GAN?

- ▶ Standard GANs generate random outputs
- ▶ We want control over what is generated
- ▶ Applications: targeted generation, data augmentation, attribute manipulation

Innovation: Condition both networks on additional information c (e.g., class label).

$$G : (z, c) \rightarrow x$$

$$D : (x, c) \rightarrow [0, 1]$$

Conditional GAN: Architecture Modification

How to add conditioning:

Generator Conditioning

1. Embed label: $c \rightarrow \mathbb{R}^{32}$ (learnable embedding).
2. Concatenate with noise: $[z, \text{emb}(c)] \in \mathbb{R}^{132}$.
3. Feed to convolutional layers.

Discriminator Conditioning

1. Process image through conv layers: $x \rightarrow h \in \mathbb{R}^{256}$.
2. Embed label: $c \rightarrow \mathbb{R}^{32}$.
3. Concatenate: $[h, \text{emb}(c)] \in \mathbb{R}^{288}$.
4. Feed to fully-connected layers for classification.

Label Embedding: Learnable lookup table (like word embeddings)

$$\text{Embedding}(c) \in \mathbb{R}^{2 \times 32} \quad \text{for 2 classes (male/female)}$$

Conditional GAN: Detailed Architecture

Conditional Generator:

- ▶ Noise: $z \in \mathbb{R}^{100}$.
- ▶ Embed label: $c \rightarrow \mathbb{R}^{32}$.
- ▶ $[z, \text{emb}(c)] \in \mathbb{R}^{132}$.
- ▶ ConvT layers as in DCGAN.
- ▶ Output: $64 \times 64 \times 3$.

Example:

- ▶ $z \sim \mathcal{N}(0, I)$.
- ▶ $c = 0$ (female).
- ▶ \rightarrow Generate female face.

Conditional Discriminator:

- ▶ Image: $64 \times 64 \times 3$.
- ▶ Conv layers $x \rightarrow h \in \mathbb{R}^{256}$.
- ▶ Embed label: $c \rightarrow \mathbb{R}^{32}$.
- ▶ $[h, \text{emb}(c)] \in \mathbb{R}^{288}$.
- ▶ FC: $288 \rightarrow 100 \rightarrow 1$.
- ▶ Output: probability.

Learns:

- ▶ Is this image real?
- ▶ Does it match the label?

Training: Same as DCGAN, but with paired (x, c) data.

Conditional GAN: Loss Functions

Modified GAN objective with conditioning:

Discriminator Loss

$$\begin{aligned}\mathcal{L}_D = & -\mathbb{E}_{(x,c) \sim p_{data}} [\log D(x, c)] \\ & - \mathbb{E}_{z,c} [\log(1 - D(G(z, c), c))]\end{aligned}$$

D learns to classify real/fake AND verify label matches image.

Generator Loss

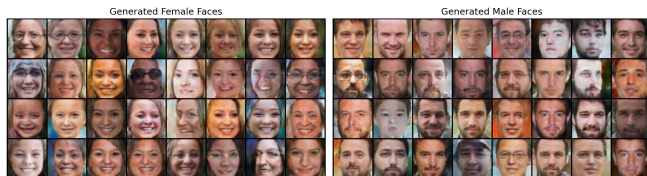
$$\mathcal{L}_G = -\mathbb{E}_{z,c} [\log D(G(z, c), c)]$$

G learns to generate realistic images that match the condition c .

Conditional GAN: Results

Gender-Conditioned Face Generation:

Conditional GAN: Gender-Controlled Face Generation



Left: Generated female faces — Right: Generated male faces

Observations:

- ▶ Clear gender-specific features (hair length, facial structure)
- ▶ High control over generated attributes
- ▶ Diverse outputs within each class
- ▶ Successful conditioning mechanism

Applications: Data augmentation, attribute manipulation, targeted generation

Example 4: Super-Resolution GAN (SRGAN)

Objective

Upscale low-resolution images to high-resolution ($4\times$ larger) while recovering realistic textures and fine details.

Why SRGAN?

- ▶ Traditional methods (bicubic, bilinear) produce blurry results.
- ▶ MSE-based methods optimize for PSNR but lack perceptual quality.
- ▶ SRGAN uses adversarial + perceptual loss for photo-realistic outputs.

Innovation:

- ▶ Perceptual loss (VGG features) instead of pixel-wise MSE.
- ▶ Adversarial training for realistic texture synthesis.
- ▶ Residual blocks for deep architecture.

SRGAN: Architecture

Generator (SRResNet):

- ▶ Input: Low-res image $(H/4) \times (W/4) \times 3$.
- ▶ Conv: $3 \rightarrow 64$ channels.
- ▶ $16 \times$ Residual Blocks (Conv-BN-PReLU-Conv-BN + skip).
- ▶ $2 \times$ Upsampling blocks (PixelShuffle).
- ▶ Conv: $64 \rightarrow 3$ channels.
- ▶ Output: High-res image $H \times W \times 3$.

Discriminator:

- ▶ Standard CNN classifier.
- ▶ 8 Conv layers with increasing filters.
- ▶ LeakyReLU, BatchNorm.
- ▶ Fully-connected: Real vs. Fake.

SRGAN: Novel Loss Function

Generator Loss (combined):

$$\mathcal{L}_G = \underbrace{\mathcal{L}_{\text{perceptual}}}_{\text{VGG features}} + \underbrace{10^{-3} \mathcal{L}_{\text{adversarial}}}_{\text{GAN loss}}$$

Perceptual Loss

$$\mathcal{L}_{\text{perceptual}} = \frac{1}{W_i H_i} \sum_{x,y} \left(\phi_i(I^{HR})_{x,y} - \phi_i(G(I^{LR}))_{x,y} \right)^2$$

where ϕ_i = VGG19 features at layer i (e.g., conv5_4).
Measures similarity in feature space, not pixel space.

Adversarial Loss

$$\mathcal{L}_{\text{adversarial}} = -\log D(G(I^{LR}))$$

Encourages realistic texture generation.

Discriminator Loss: Standard BCE (real vs. fake).

SRGAN: Why Perceptual Loss?

Comparison of Loss Functions:

Loss	Optimizes for	Result
MSE	PSNR	Blurry, over-smoothed
MAE	PSNR	Blurry, over-smoothed
Perceptual	Feature similarity	Sharp, realistic
Adversarial	Realism	Realistic textures
Combined	Both	Best quality

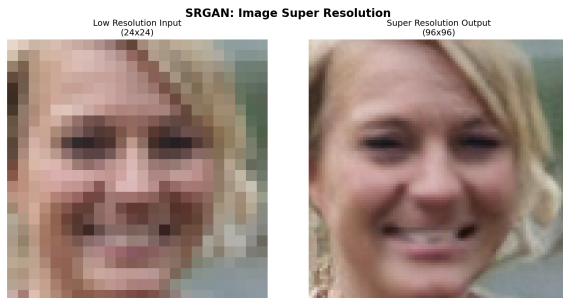
Insight:

- ▶ Pixel-wise loss \rightarrow average of all possible HR images \rightarrow blur.
- ▶ Perceptual loss \rightarrow semantically similar features \rightarrow sharp.
- ▶ Adversarial loss \rightarrow realistic distribution \rightarrow natural textures.

Training: Pretrain with MSE, then fine-tune with perceptual + adversarial loss.

SRGAN: Results

4× Super-Resolution:



Left: Low-res input (128×128) — Right: Super-res output (512×512)

Observations:

- ▶ Sharp edges and fine details recovered
- ▶ Realistic textures (hair, skin, fabric)
- ▶ 4× upscaling with minimal artifacts
- ▶ Perceptually convincing results

Applications: Photo enhancement, medical imaging, satellite imagery.

Example 5: StyleGAN2 - Latent Space Manipulation

Objective

Encode real images into latent space, manipulate semantic attributes (smile, age, pose), and decode back to modified images.

Why StyleGAN2?

- ▶ State-of-the-art GAN for high-quality face generation.
- ▶ Disentangled latent space enables fine-grained control.
- ▶ Hierarchical style injection (coarse \rightarrow fine features).
- ▶ Enables creative applications and face editing.

Innovation:

- ▶ Style-based generator with adaptive instance normalization.
- ▶ Mapping network: z (noise) $\rightarrow w$ (disentangled latent).
- ▶ Progressive synthesis with style injection at each layer.

StyleGAN2: Generator Architecture Overview

- ▶ **Input:** A random noise vector $z \in \mathbb{R}^{512}$ is sampled from a normal distribution.
- ▶ **Mapping Network:** The vector z is passed through 8 fully connected layers to produce an intermediate latent vector $w \in \mathbb{R}^{512}$. This step helps disentangle latent factors.
- ▶ **Synthesis Network:** The vector w controls a progressive generator that builds the image from low to high resolution. Each layer receives a different style derived from w .
- ▶ **Style Injection:** Styles are injected at each layer via Adaptive Instance Normalization (AdaIN), allowing fine-grained control over features such as texture, shape, and color.
- ▶ **Output:** The final output is a high-resolution image, typically 1024×1024 pixels.

StyleGAN2 separates the latent space from the image synthesis process, enabling better control and improved image quality. The **discriminator** is a deep CNN with residual blocks.

StyleGAN2: Hierarchical Style Control

Different layers control different features:

Layers	Controls	Examples
0-3	Coarse features	Pose, face shape, glasses
4-7	Medium features	Facial features, hairstyle
8-15	Fine features	Color scheme, micro-structure
16-18	Very fine	Hair texture, skin pores

Style Mixing:

- ▶ Take w_1 from image A (coarse).
- ▶ Take w_2 from image B (fine).
- ▶ Combine: A's pose + B's details.

Semantic Editing:

- ▶ Learn direction in w -space: d_{smile} .
- ▶ Edit: $w' = w + \alpha \cdot d_{\text{smile}}$.
- ▶ Generate: $G(w') = \text{image with smile}$.

StyleGAN2: Image Encoding and Editing

Workflow:

1. Encode real image to latent:

- ▶ Use optimization or encoder network.
- ▶ Find w such that $G(w) \approx I_{\text{real}}$.

2. Manipulate latent:

- ▶ Add semantic direction: $w' = w + \alpha \cdot d$.
- ▶ d = smile, age, pose, etc.
- ▶ α = manipulation strength.

3. Generate edited image:

- ▶ $I_{\text{edited}} = G(w')$.
- ▶ Smooth interpolation possible.

Loss: Standard GAN loss during training (pretrained model used here).

StyleGAN2: Results of style mixing

Feature Transfer at Different Layers:



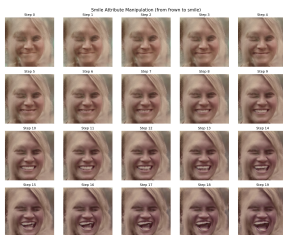
Swapping coarse features (layers 0-3): pose and face shape transfer.

Observations:

- ▶ Different layer ranges control different semantic attributes.
- ▶ Smooth interpolation between styles.
- ▶ High-quality 1024×1024 outputs.

StyleGAN2: Results of attribute manipulation

Smile Manipulation (20 steps):



Smooth interpolation from frown to smile by moving in latent space.

Observations:

- ▶ Smooth, continuous attribute changes.
- ▶ Identity preserved during manipulation.
- ▶ No mode collapse or artifacts.
- ▶ Precise control over facial expressions.

Applications: Face editing, aging, makeup, expression transfer.