Generative Adverserial Networks

Vinay Setty
vinay.j.setty@uis.no

University of Stavanger

Information Access & Artificial Intelligence

Department of Electrical Engineering and Computer Science
University of Stavanger

January 16, 2026

Overview

ProGAN

StyleGAN

StyleGAN2

Self-Attention GAN

BigGAN

VQ-GAN

ViT VQ-GAN

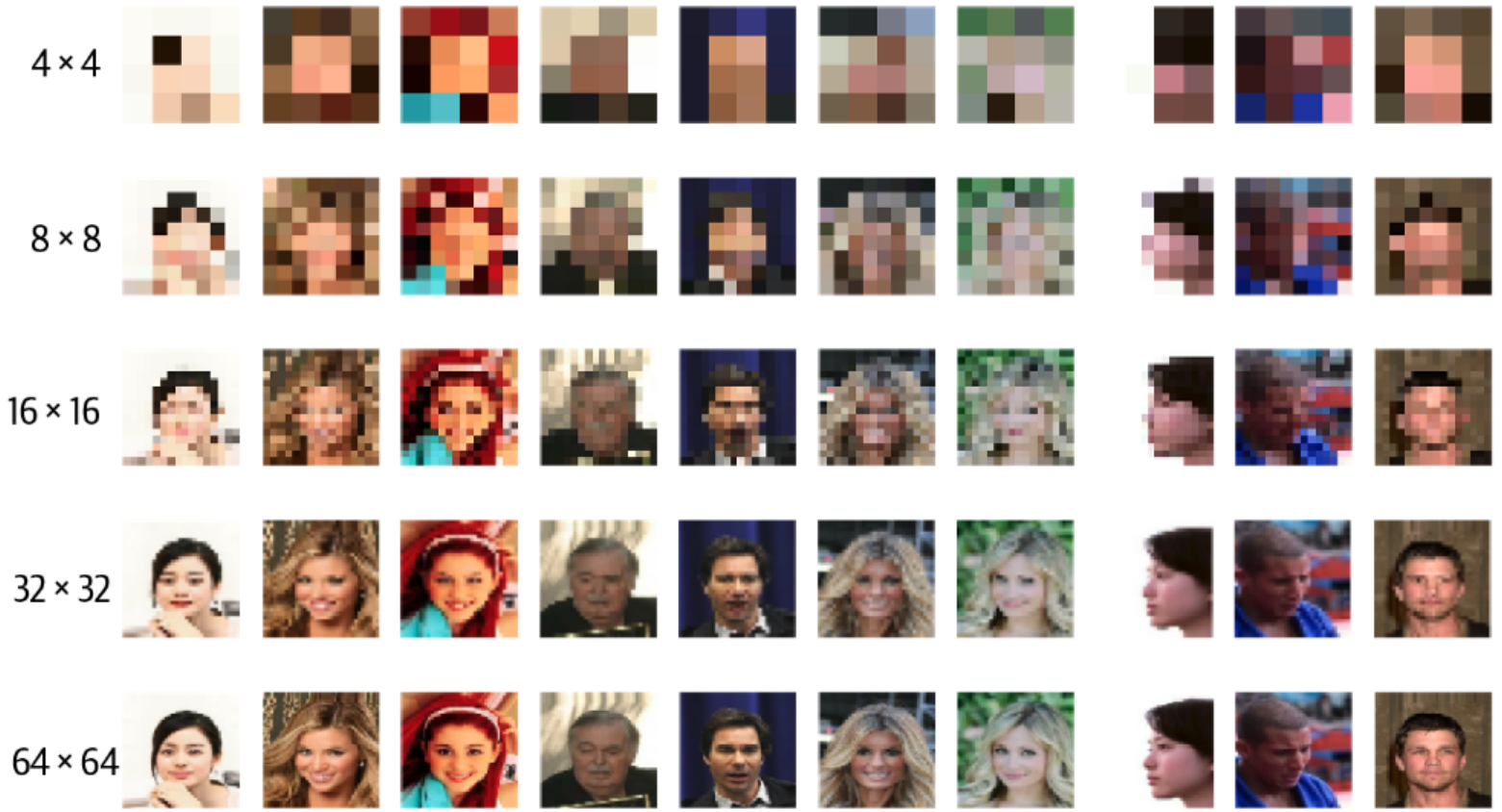
Progressive Training

- Generative Adversarial Networks consist of two competing networks: a generator and a discriminator, trained via adversarial optimization.
- In standard GAN training, the generator produces full-resolution images from the beginning of training.
- This forces the model to learn both high-level structure and fine-grained details simultaneously.
- Early in training, this can slow learning, as the generator must operate in a complex, high-dimensional image space.
- A more efficient strategy is to first learn low-resolution structure and then gradually increase image resolution.

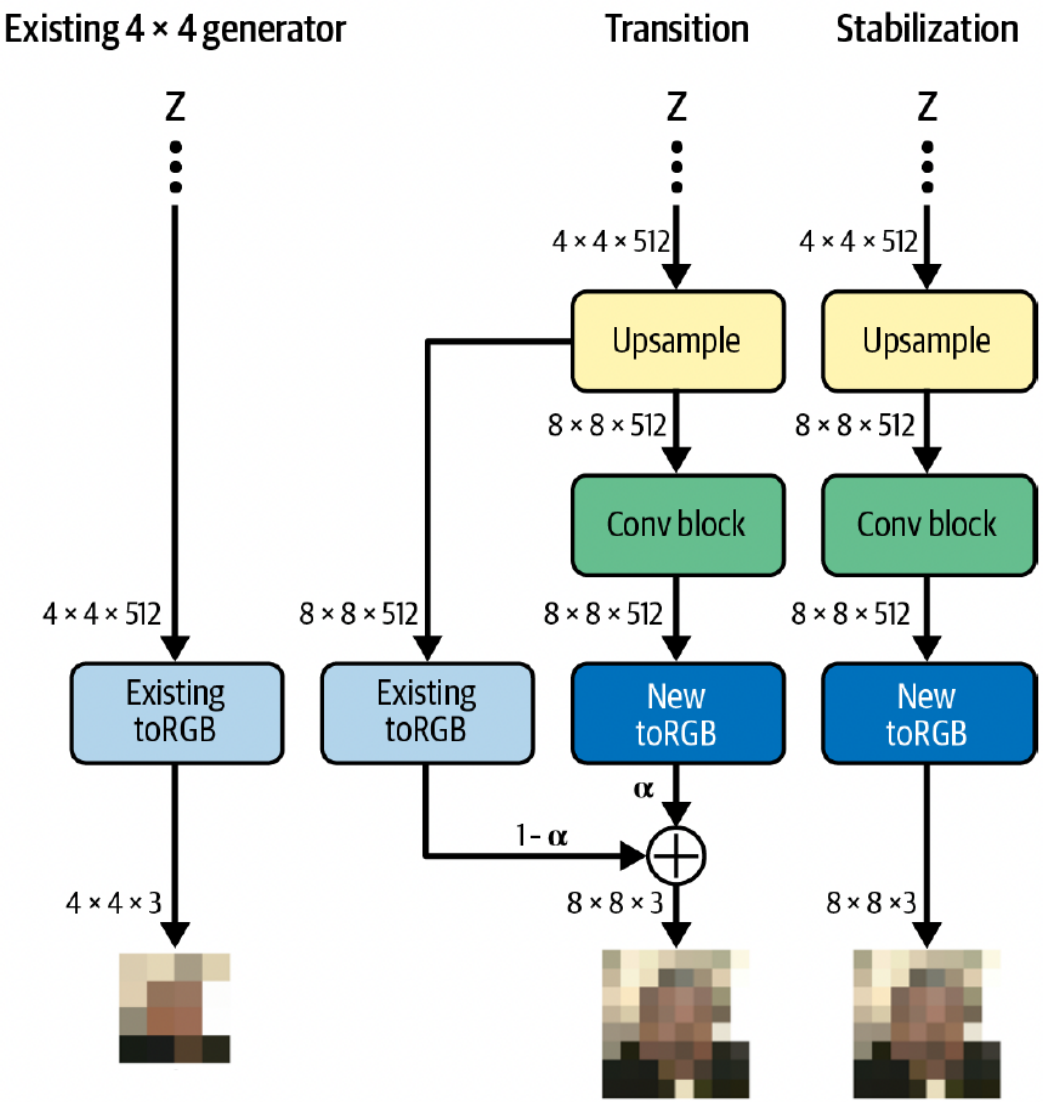
Progressive Training (cont.)

- Training starts with heavily downsampled images at 4×4 resolution.
- Network depth and image resolution are increased incrementally during training.
- This enables stable learning of global structure before introducing fine details.

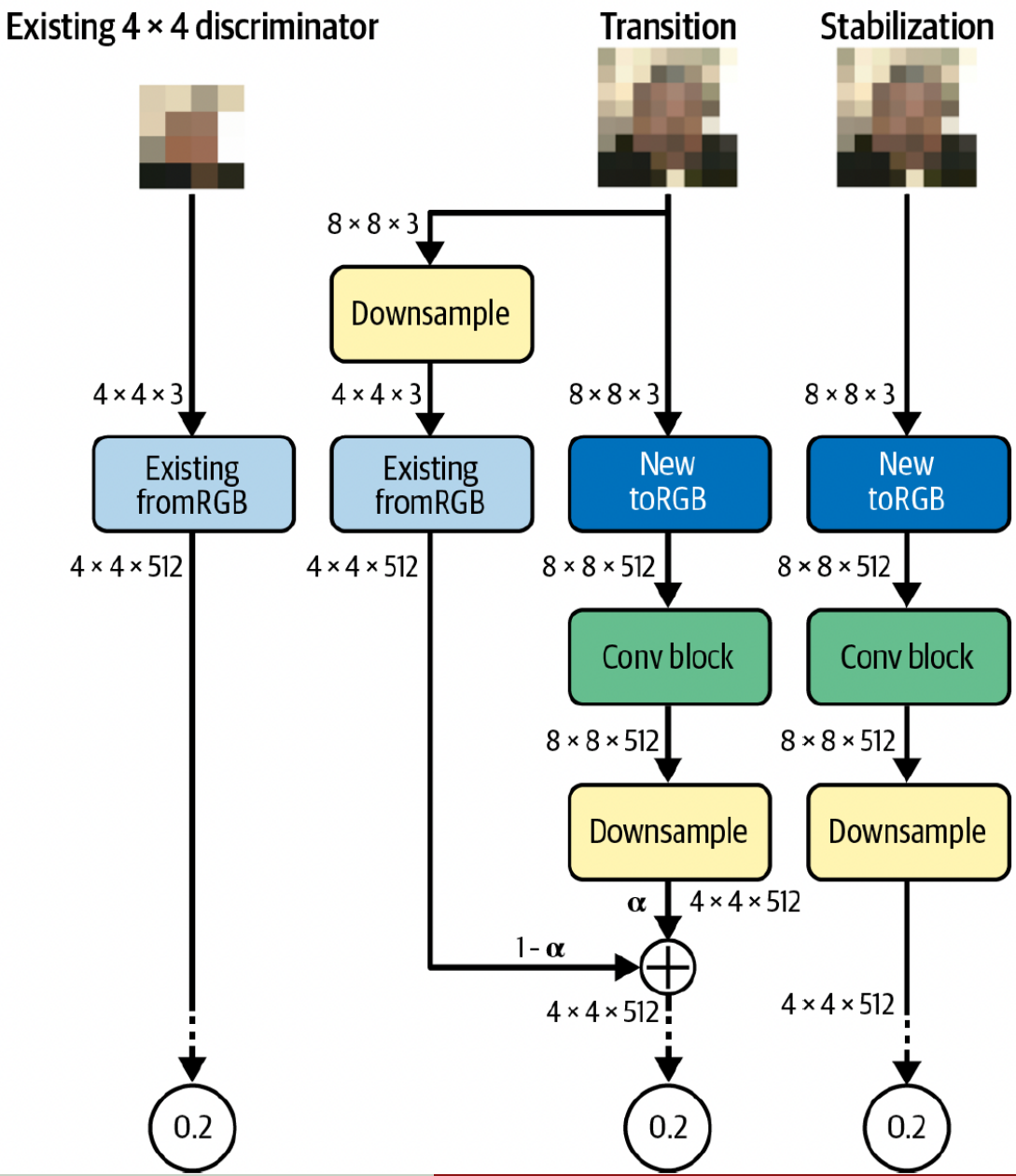
Progressive Training (cont.)



ProGAN Architecture (Generator)



ProGAN Architecture (Discriminator)



Progressive GAN Training: Transition and Stabilization

- **Progressive Growing** Training starts at low spatial resolution (e.g. 4×4) and incrementally adds layers to reach higher resolutions. Both generator and discriminator grow symmetrically.
- **Transition Phase (Fade-in)** A new convolutional block is introduced at higher resolution. Outputs from the old pathway and the new pathway are linearly blended:

$$x = (1 - \alpha)\, x_{\text{old}} + \alpha\, x_{\text{new}}$$

  where $\alpha \in [0, 1]$ is gradually increased during training.
- **Stabilization Phase** After $\alpha = 1$, the old pathway is removed. The network trains only through the new layers, allowing weights to stabilize at the current resolution.
- **Skip-like Structure** The fade-in acts as a residual interpolation across resolutions rather than layers. This smooths the optimization landscape and prevents sudden distribution shifts.
- **Effect** Reduces training instability, improves convergence, and enables high-resolution synthesis without mode collapse.

Stabilization Techniques in ProGAN

- **Minibatch Standard Deviation** The discriminator receives a feature measuring variation across the minibatch. Low variation indicates mode collapse. This forces the generator to increase output diversity.
- **Equalized Learning Rates** All weights are initialized from the same Gaussian distribution. Scaling is applied dynamically during the forward pass. This keeps learning speeds balanced across layers.
- **Pixelwise Normalization** Each pixel feature vector is normalized to unit length. This prevents activation magnitudes from growing uncontrollably. No trainable parameters are introduced.

ProGAN Generated images

StyleGAN: Main Idea

- StyleGAN builds on ProGAN. The discriminator stays the same.The generator is redesigned.
- Latent factors in GANs are often entangled. Changing one attribute can change others. Example: freckles change and the background shifts too.
- StyleGAN targets disentanglement. It injects style vectors at multiple layers. Early injections control coarse attributes (e.g., pose). Later injections control fine details (e.g., hair strands).
- The generator begins with a mapping network. The mapping network produces the style representation used across layers.

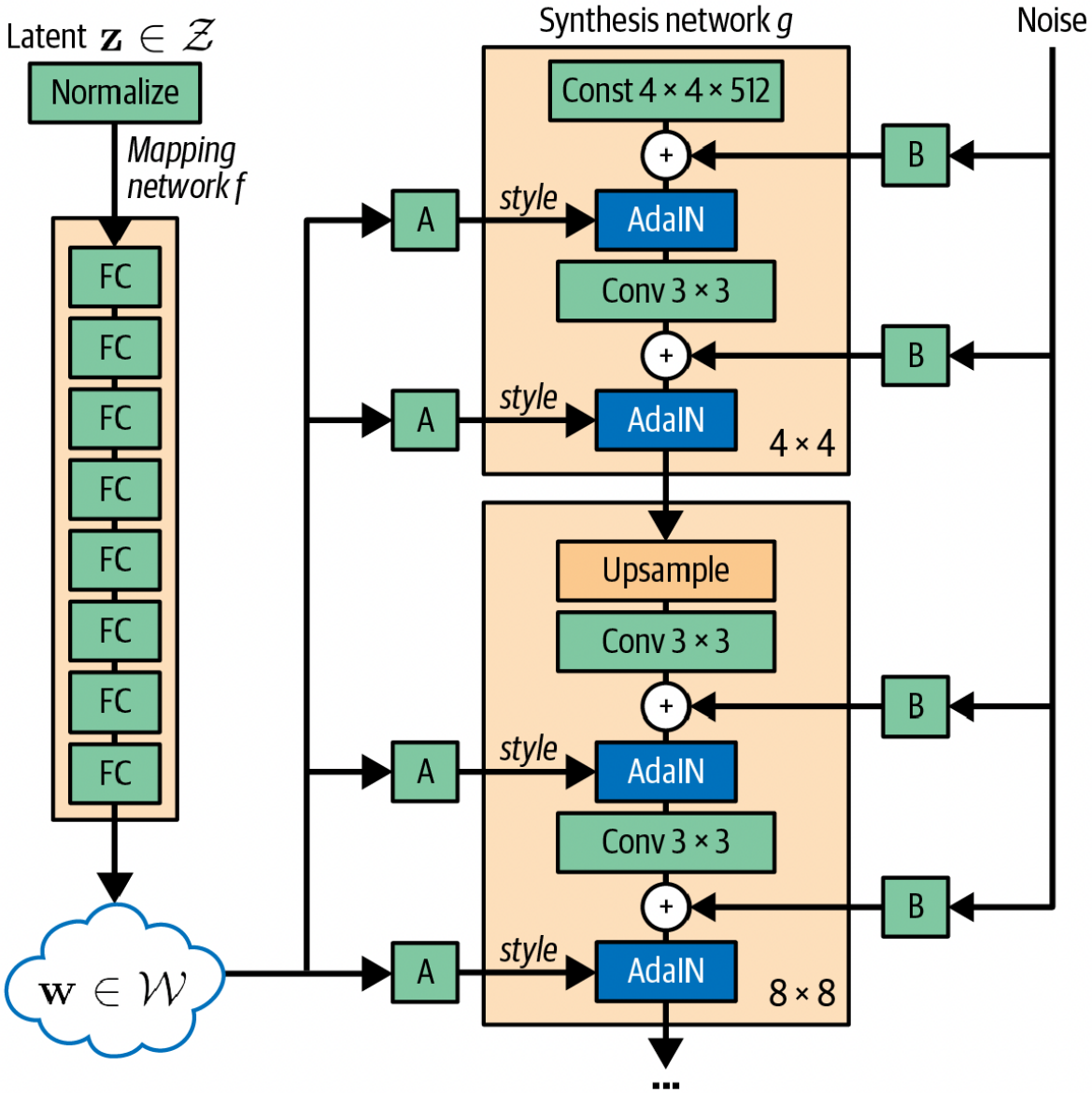
StyleGAN: Why a Mapping Network?

- Input noise: $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.
- Mapping network: $\boldsymbol{w} = f(\boldsymbol{z})$, where f is an MLP.
- Goal: disentangle factors before image synthesis.
- Interpretation: choose *style* in $\boldsymbol{w}$–space, then render in synthesis network.

StyleGAN: Synthesis Network

- The synthesis network is the generator of the actual image with a given style
- It receives a style code $\boldsymbol{w}$ from the mapping network.
- Style is injected at multiple layers.
- Each layer i has its own affine transform A_i.
- A_i produces two vectors.
- A scale vector $\boldsymbol{y}_{s,i}$.
- A bias vector $\boldsymbol{y}_{b,i}$.

StyleGAN: Mapping Network



Style Injection with AdaIN (StyleGAN)

- At layer i, the style vector $\boldsymbol{w}$ is turned into scaling and bias:

$$\boldsymbol{y}_{s,i}, \boldsymbol{y}_{b,i} = A_i(\boldsymbol{w})$$

  where A_i is a learned affine transform.

- Adaptive instance normalization (AdaIN):

$$\mathrm{AdaIN}(\boldsymbol{x}_i; \boldsymbol{y}_{s,i}, \boldsymbol{y}_{b,i}) = \boldsymbol{y}_{s,i} \odot \frac{\boldsymbol{x}_i - \mu(\boldsymbol{x}_i)}{\sigma(\boldsymbol{x}_i) + \epsilon} + \boldsymbol{y}_{b,i}.$$

- Early layers $\Rightarrow$ coarse style (pose, shape).
- Late layers $\Rightarrow$ fine style (texture, hair, skin details).

Mini example (one channel):

$x = [2, 4],\ \mu = 3,\ \sigma = 1,\ y_s = 1.5,\ y_b = 0.2 \Rightarrow \mathrm{AdaIN}(x) = 1.5 \cdot [-1, 1] + 0.2 =$

,1.7

.

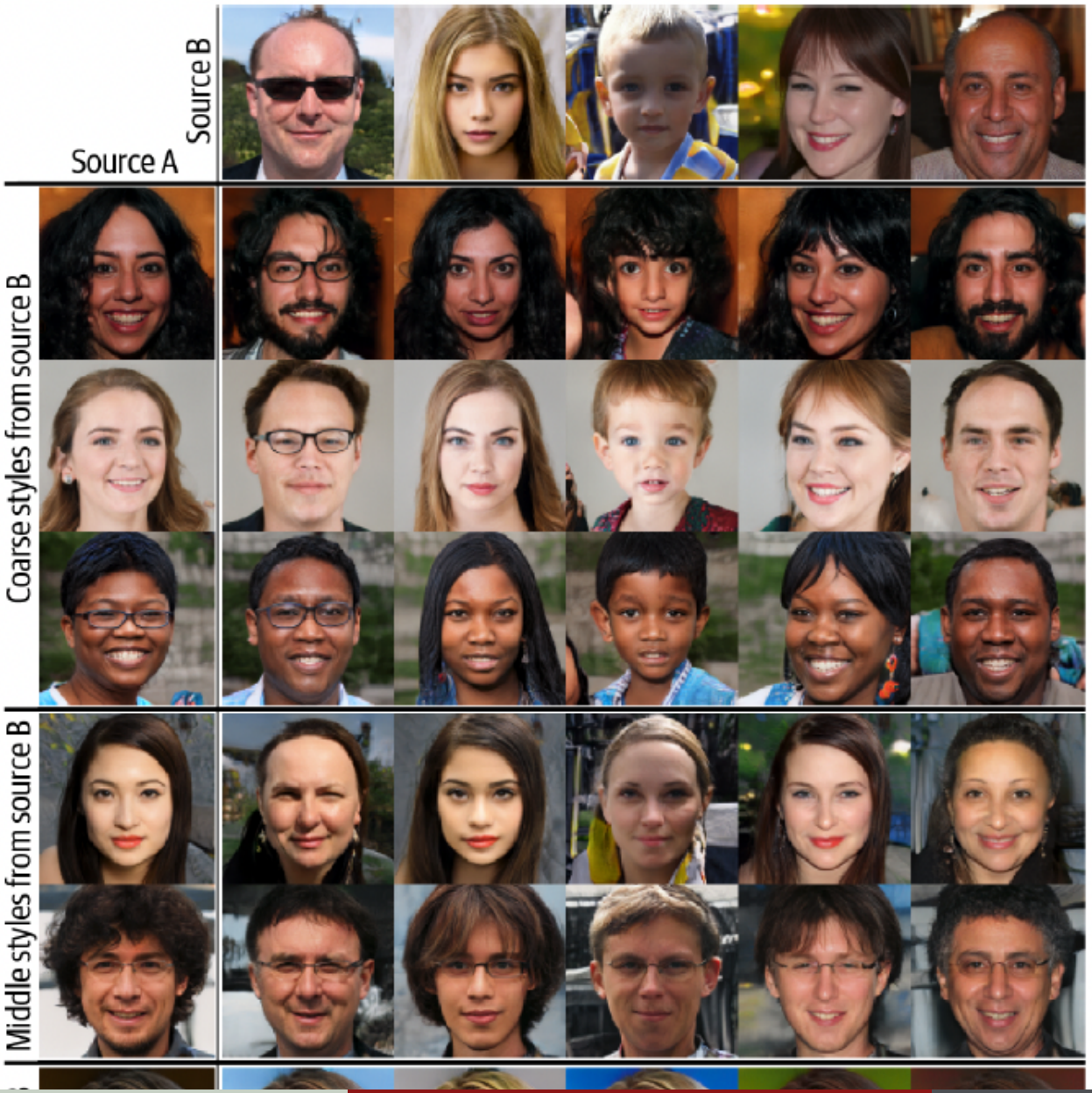
Style Mixing (Illustration + Intuition)

- Sample two latents: $\boldsymbol{z}_A, \boldsymbol{z}_B$.
- Map to styles: $\boldsymbol{w}_A = f(\boldsymbol{z}_A)$, $\boldsymbol{w}_B = f(\boldsymbol{z}_B)$.
- During training, randomly switch styles at some layer index k:

$$\boldsymbol{w}_i = \begin{cases} \boldsymbol{w}_A & i \\ & k \\ \\ \boldsymbol{w}_B & i > k \end{cases}$$

- Effect: forces layers to represent styles independently.

StyleGAN output



Artifact Problem

StyleGAN2: Remove AdaIN, Modulate Weights

- AdaIN caused characteristic artifacts ("droplet" patterns).
- StyleGAN2 injects style by **modulating convolution weights**.
- For layer i, affine gives per-input-channel scale $\boldsymbol{s}_i$:

$$\boldsymbol{s}_i = A_i(\boldsymbol{w}).$$

- Modulation (per input channel):

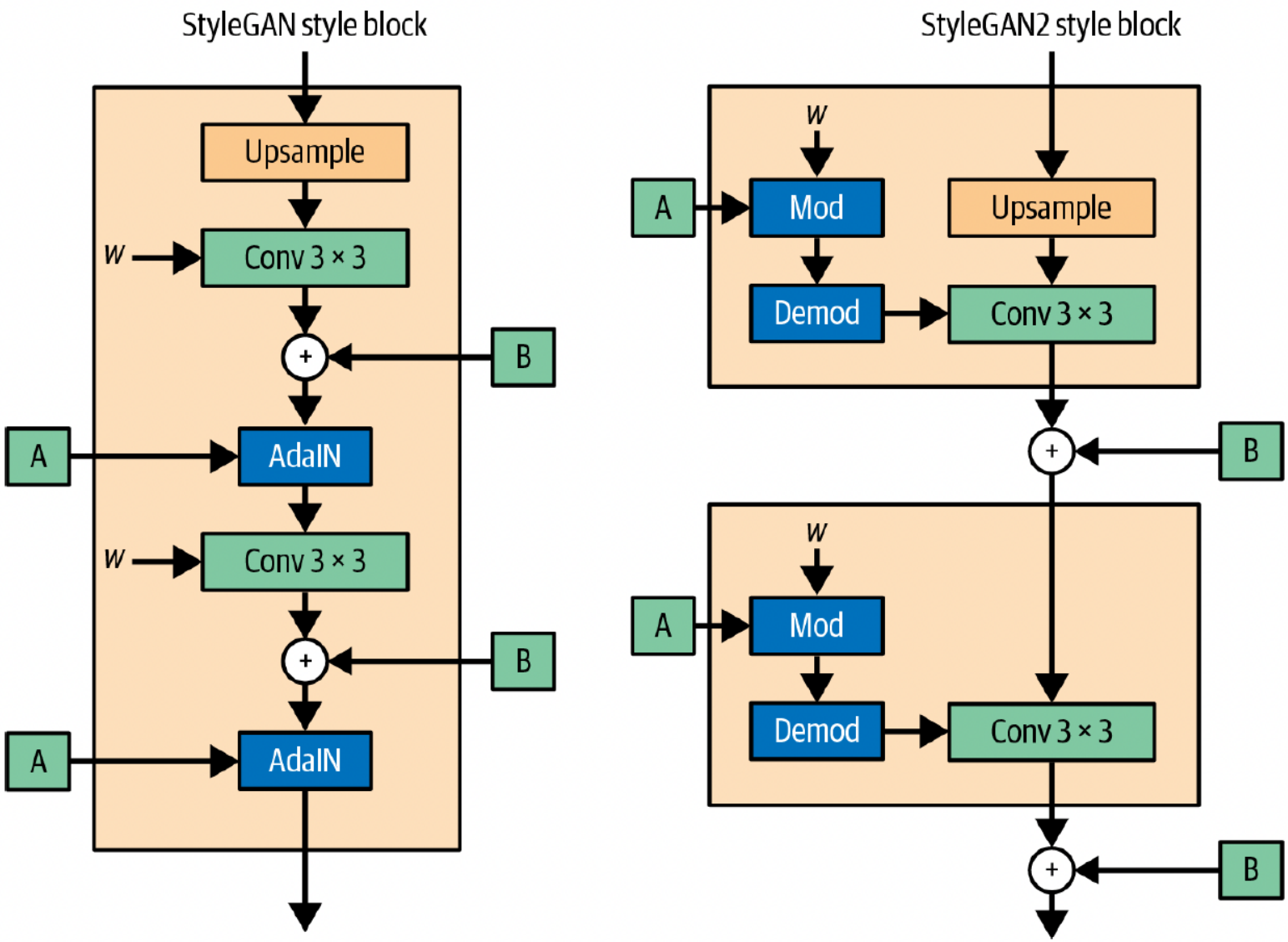
$$\boldsymbol{w}'_{i,j,k} = s_i \cdot \boldsymbol{w}_{i,j,k}.$$

- Demodulation (normalize per output channel j):

$$\boldsymbol{w}''_{i,j,k} = \frac{\boldsymbol{w}'_{i,j,k}}{\sqrt{\sum_{i,k} (\boldsymbol{w}'_{i,j,k})^2 + \epsilon}}.$$

Takeaway: style controls *filters*, not feature statistics.

Weight Modulations



Mod/Demod: Tiny Numeric Example

One output channel, two weights:

$$\boldsymbol{w} = [2, 1], \quad \boldsymbol{s} = [0.5, 2.0].$$

Modulation:

$$\boldsymbol{w}' = \boldsymbol{s} \odot \boldsymbol{w} = [1, 2].$$

Demodulation factor:

$$d = \sqrt{1^2 + 2^2 + \epsilon} \approx \sqrt{5} = 2.236.$$

Demodulated weights:

$$\boldsymbol{w}'' = \boldsymbol{w}'/d \approx [0.447, 0.894].$$

- Modulation changes relative influence of inputs.
- Demodulation prevents exploding channel magnitudes.

No Progressive Growing (StyleGAN2 Training)

- ProGAN trained with resolution phases and fade-in:

$$x = (1 - \alpha)x_{\text{old}} + \alpha x_{\text{new}}, \quad \alpha \uparrow .$$

- StyleGAN2 trains the full model end-to-end.
- Uses skip connections in generator and residual connections in discriminator.

Path Length Regularization (Smooth Latent Control)

- We want a fixed step in $\boldsymbol{w}$ to cause a consistent change in image.
- Let image be $\boldsymbol{y} = G(\boldsymbol{w})$ (optionally with noise).
- Jacobian w.r.t. style: $\boldsymbol{J}_{\boldsymbol{w}} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}}$.
- Path length (one form):

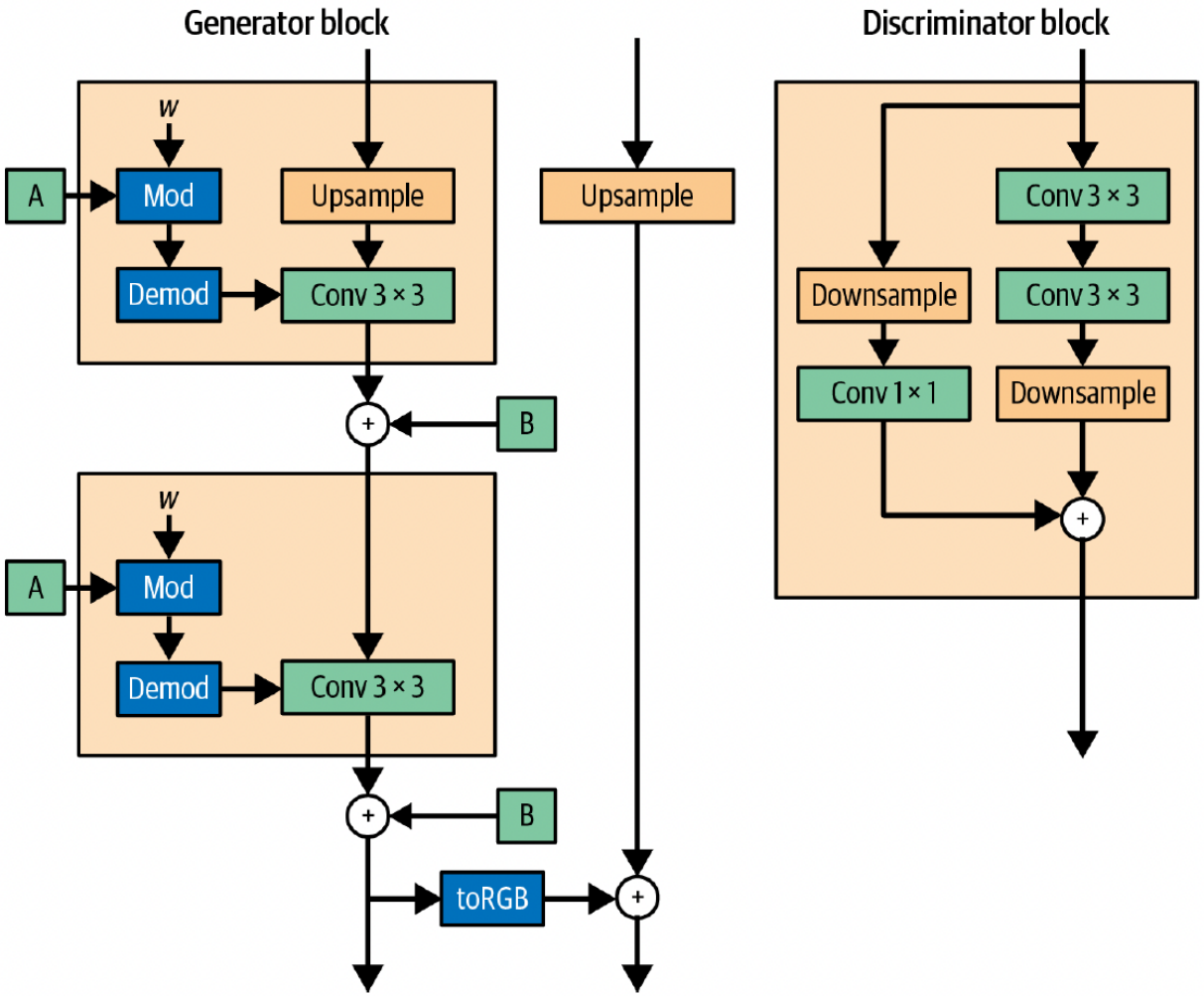
$$\ell(\boldsymbol{w}, \boldsymbol{y}) = \left\| \boldsymbol{J}_{\boldsymbol{w}}^{\mathsf{T}} \boldsymbol{y} \right\|_2 .$$

- Regularize toward a target magnitude a:

$$\mathcal{L}_{\mathrm{pl}} = \mathbb{E}_{\boldsymbol{w},\boldsymbol{y}} \left(\ell(\boldsymbol{w}, \boldsymbol{y}) - a \right)^2 .$$

Effect: smoother edits in latent space and more consistent interpolation.

StyleGAN2 Architecture



Lazy Regularization (Compute It Sometimes)

- Some regularizers are expensive (e.g., gradient penalties, path length).
- Apply them once every N steps (e.g., $N = 16$).
- Scale to keep expected contribution comparable:

$$\tilde{\lambda} = \lambda \cdot N.$$

- Practical result: almost same quality, much less compute.

Resolution Contribution Over Training (How to Read the Plot)

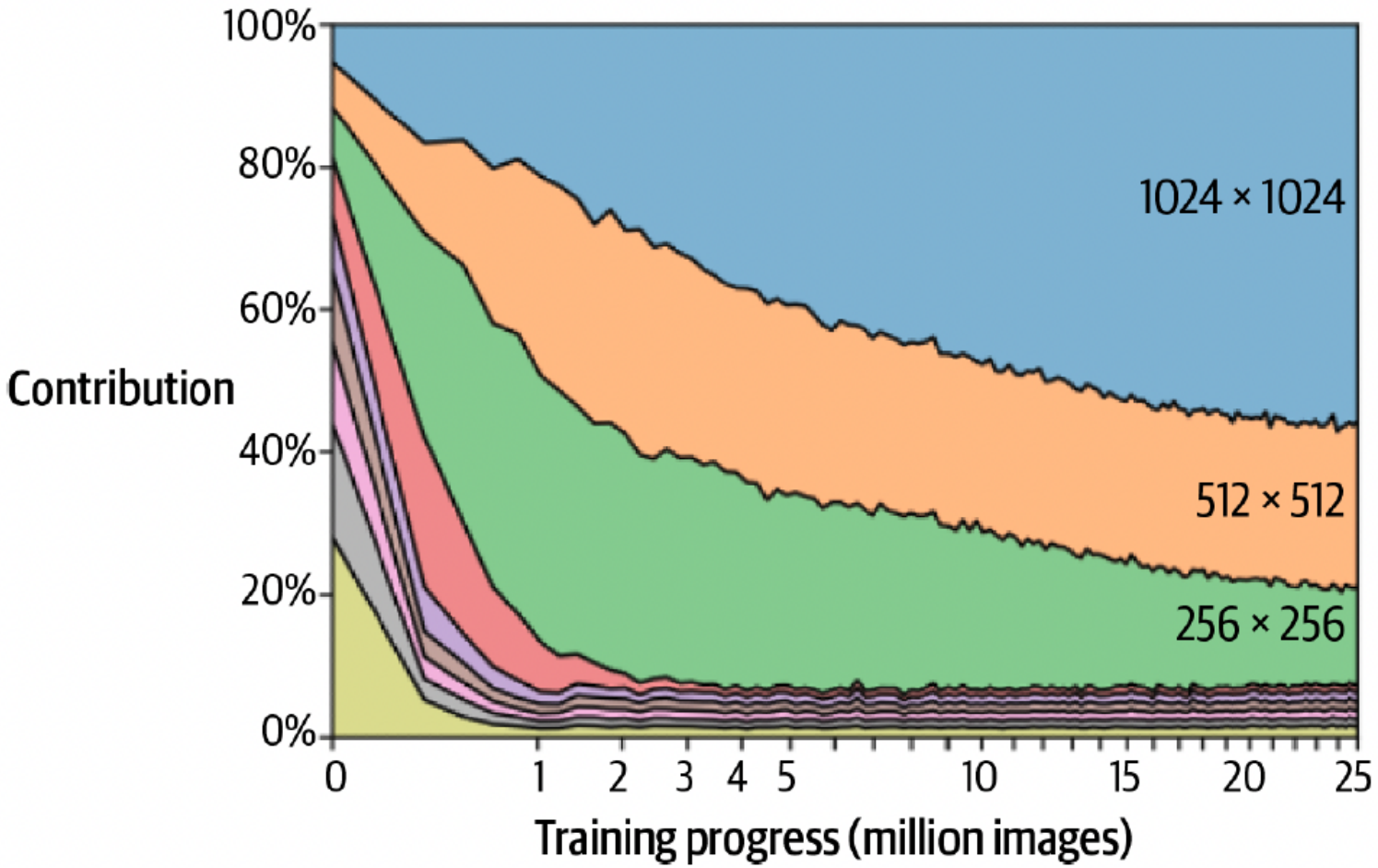
- Output image is a sum of contributions from multiple resolution paths.
- Early in training: low-res layers dominate (structure first).
- Later: high-res layers dominate (detail refinement).

Mental model:

$$\text{image} \approx \sum_{r \in \{4,8,\ldots,1024\}} c_r(t) \cdot \text{features}_r, \quad \sum_r c_r(t) = 1.$$

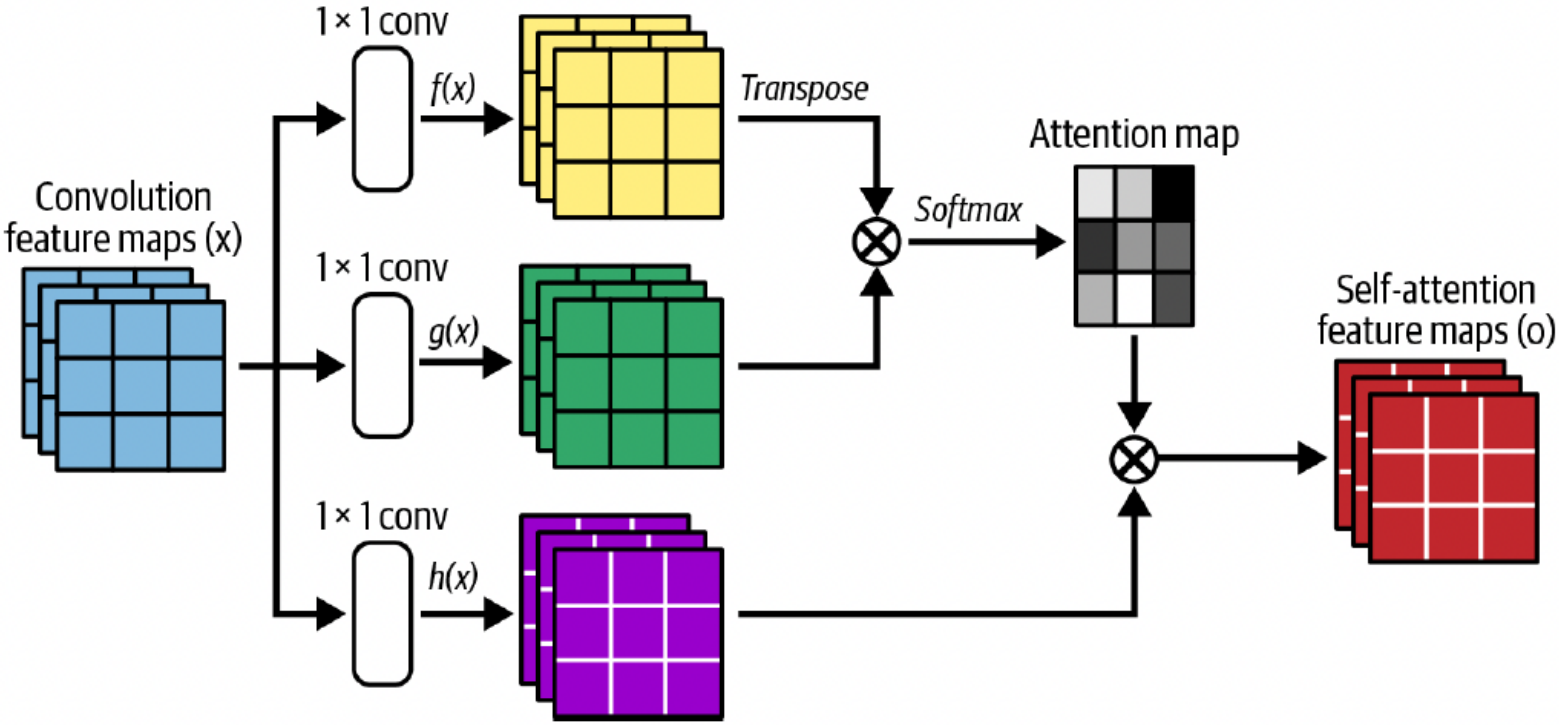
- At small t, $c_{4,8,16}$ are large.
- At large t, $c_{512,1024}$ increase.

Progressive Resolution

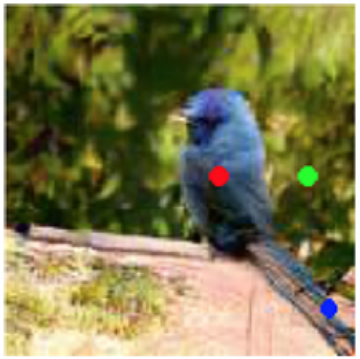


StyleGAN2 Output

Self-Attention GAN SAGAN



Self-Attention GAN SAGAN




BigGAN: Scaling GANs and the Truncation Trick

- BigGAN extends SAGAN by aggressively scaling model size and batch size.
- Trained on ImageNet with class-conditional generation.
- Large capacity enables high visual fidelity at moderate resolution.
- **Key idea: Truncation Trick**
- During training: latent vectors sampled as

$$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}).$$

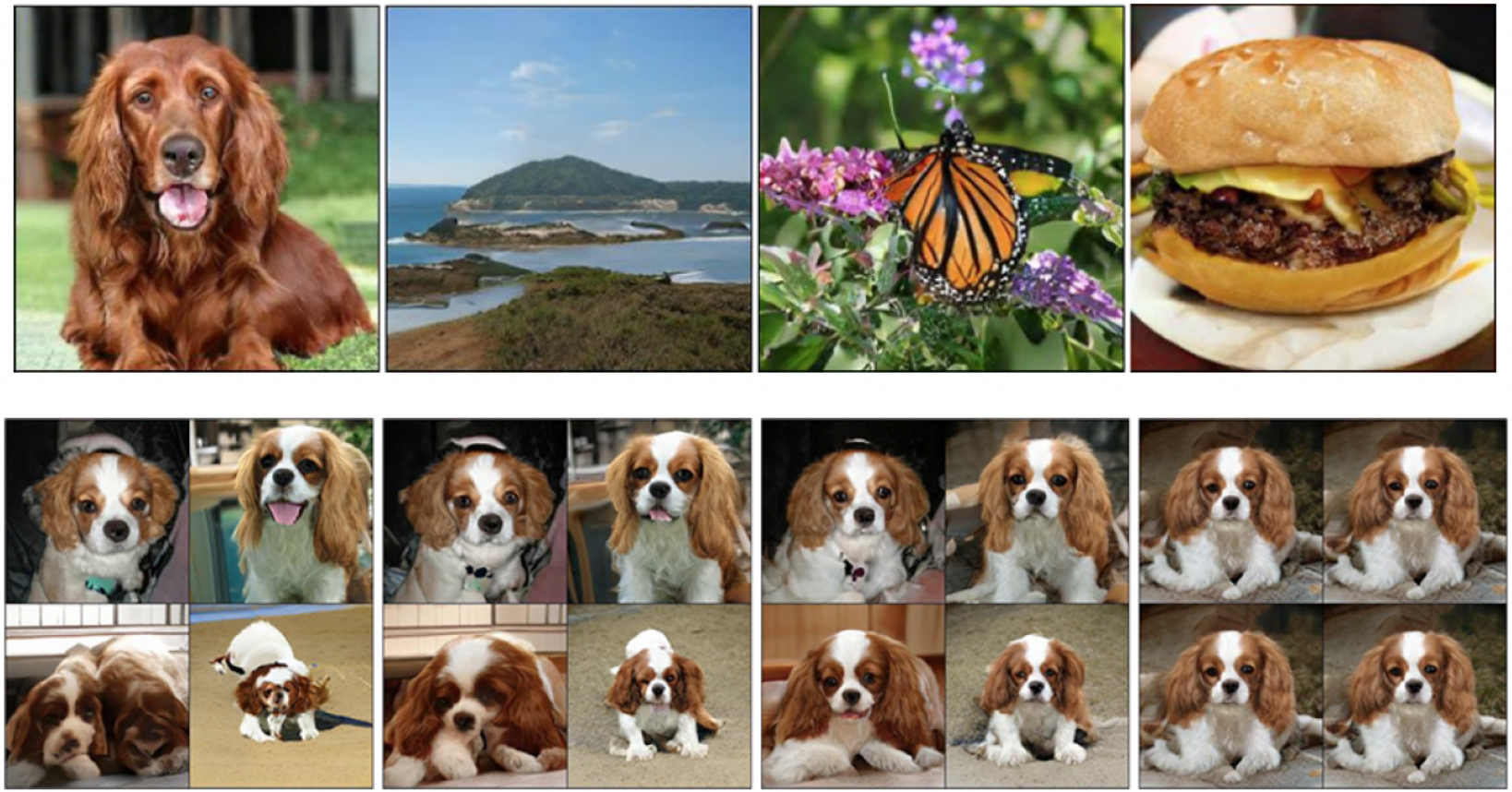
- During sampling: use a truncated normal distribution.
- Reject samples with large magnitude:

$$z_i \leftarrow z_i \quad \text{only if} \quad |z_i|$$

τ

.

Self-Attention GAN SAGAN



VQ-VAE: Discrete Latent Space

- Encoder maps image to a grid of latent vectors.
- Each latent vector is quantized to a codebook entry.
- Latent space is discrete, not continuous.

Vector quantization:

$$\boldsymbol{z}_q(x) = \arg\min_{\boldsymbol{e}_k \in \mathcal{E}} \|\boldsymbol{z}_e(x) - \boldsymbol{e}_k\|_2$$

- Reconstruction loss trains encoder–decoder.
- Commitment and alignment losses train the codebook.
- No KL divergence is used.

Why VQ-GAN?

- VQ-VAE reconstructions are often blurry.
- GANs produce sharper images.
- VQ-GAN combines both.

Core idea:

- Keep discrete latent representation.
- Add a GAN discriminator on decoder output.
- Train encoder–decoder adversarially.

Result:

- Discrete and interpretable latents.
- Sharper and more realistic images.

VQ-GAN Architecture and Loss

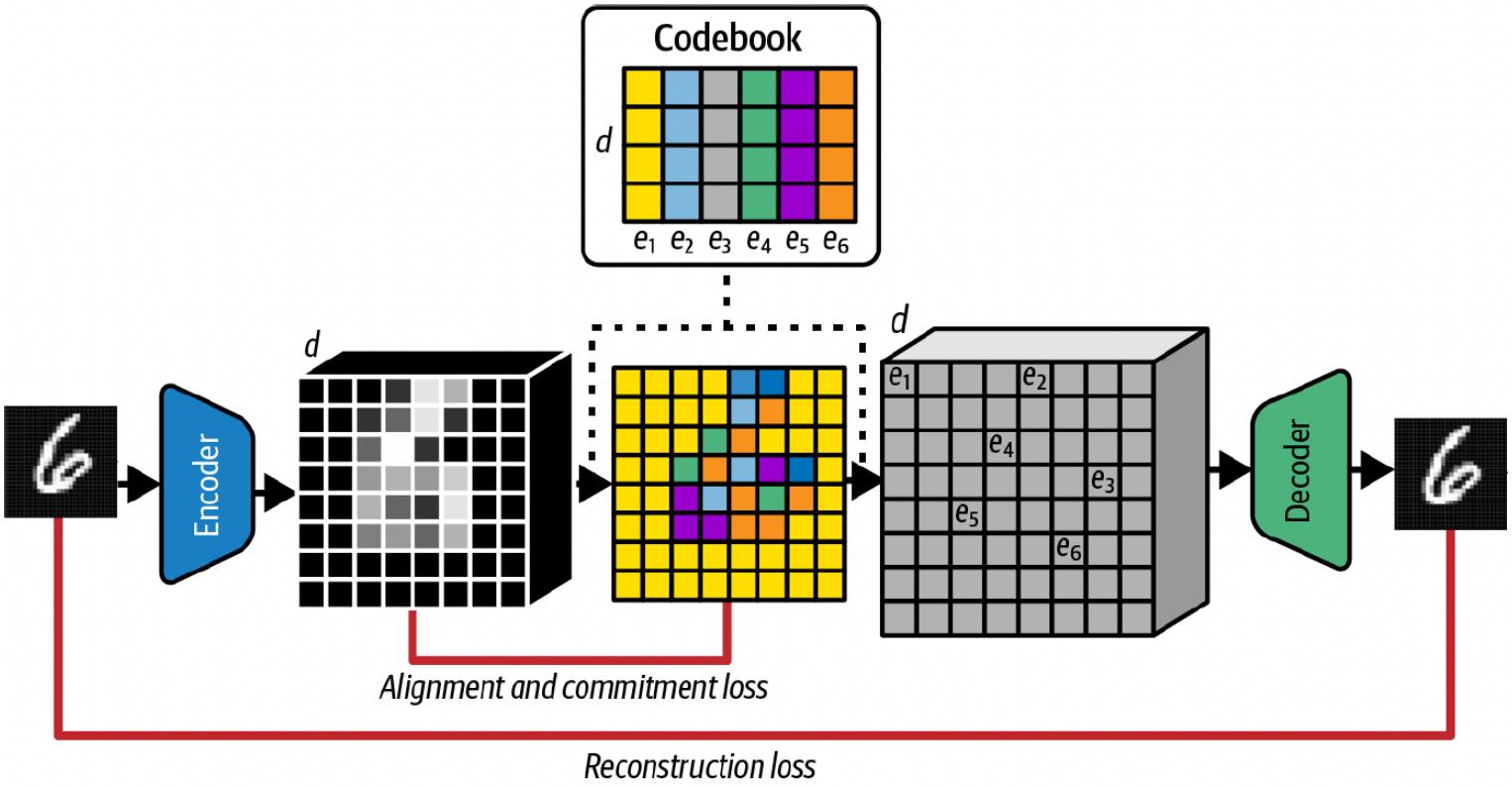
- Encoder produces latent grid.
- Codebook quantizes latent vectors.
- Decoder reconstructs image.
- Discriminator judges real vs fake patches.

Total loss:

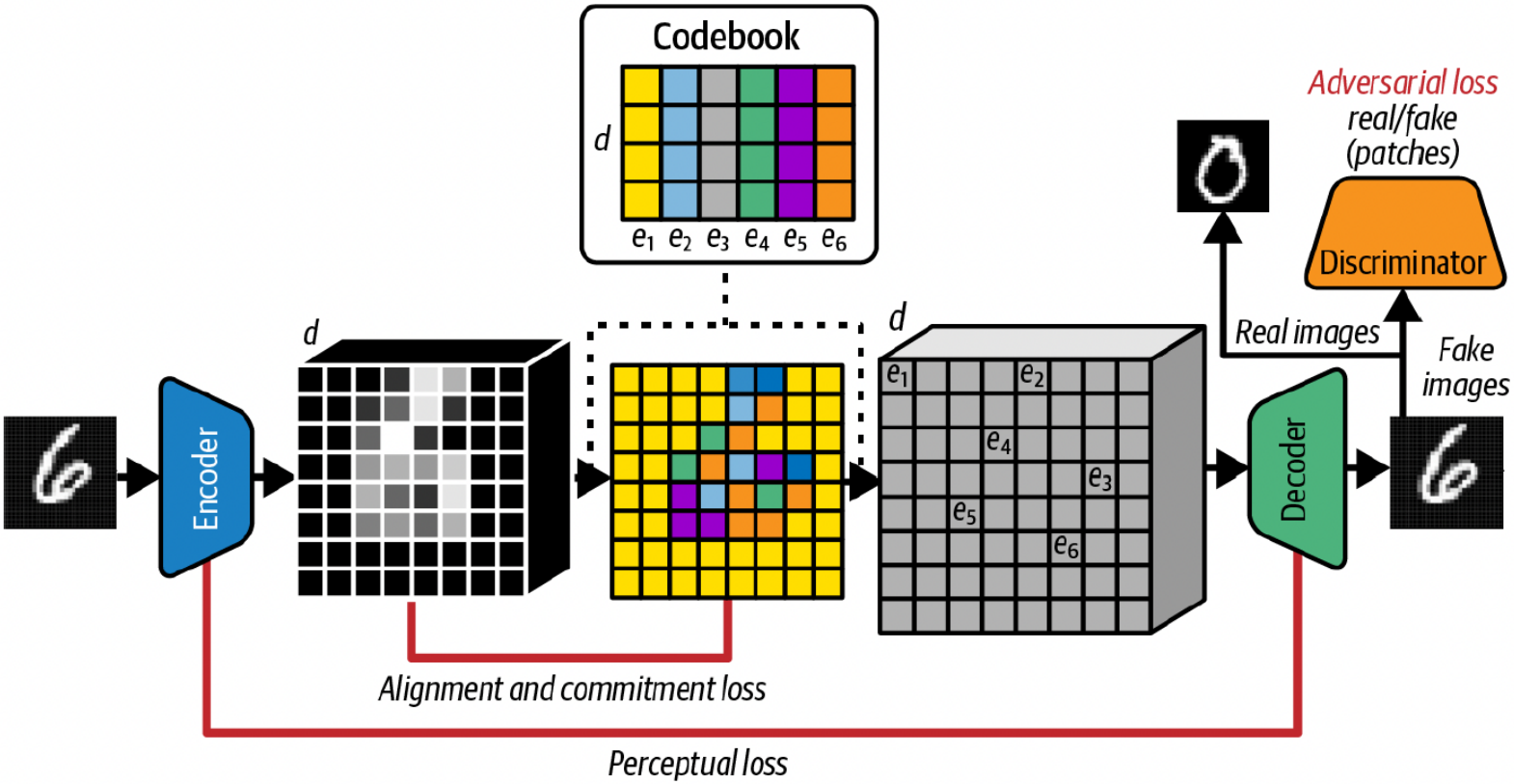
$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{VQ}} + \lambda_{\text{GAN}}\mathcal{L}_{\text{adv}} + \lambda_{\text{perc}}\mathcal{L}_{\text{perc}}$$

- Adversarial loss improves sharpness.
- Perceptual loss compares deep features, not pixels.
- PatchGAN discriminator focuses on local realism.

VQ-GAN


Codebook
d
e_1 e_2 e_3 e_4 e_5 e_6
d
d
Encoder
e_1
e_2
e_4
e_3
e_5
e_6
Decoder
Alignment and commitment loss
Reconstruction loss

VQ-GAN


Codebook
d
e_1 e_2 e_3 e_4 e_5 e_6
Adversarial loss
real/fake
(patches)
Discriminator
Real images
Fake
images
Encoder
d
d
e_1
e_2
e_4
e_3
e_5
e_6
Decoder
Alignment and commitment loss
Perceptual loss

Autoregressive Prior in VQ-GAN

- Decoder needs a prior over code sequences.
- Uniform sampling does not work.

Solution:

- Train an autoregressive model on code grids.
- Originally PixelCNN.
- Later replaced by Transformers.

Sampling:

$$p(\boldsymbol{z}) = \prod_i p(z_i \mid z_{<i})$$

ViT VQ-GAN: Motivation

- Convolutions limit long-range modeling.
- Transformers model global context naturally.

Key change:

- Replace CNN encoder and decoder with Transformers.
- Operate on image patches instead of pixels.

Outcome:

- Better global coherence.
- Improved scaling to larger images.

ViT VQ-GAN Architecture

- Image split into non-overlapping patches.
- Patches embedded and position-encoded.
- Transformer encoder processes patch sequence.
- Latents are vector-quantized using a codebook.

Decoding:

- Quantized tokens passed to Transformer decoder.
- Decoder reconstructs image patches.
- Patches stitched into full image.

Training Pipeline: ViT VQ-GAN

- Stage 1: train VQ-GAN autoencoder with discriminator.
- Stage 2: train autoregressive Transformer on codes.

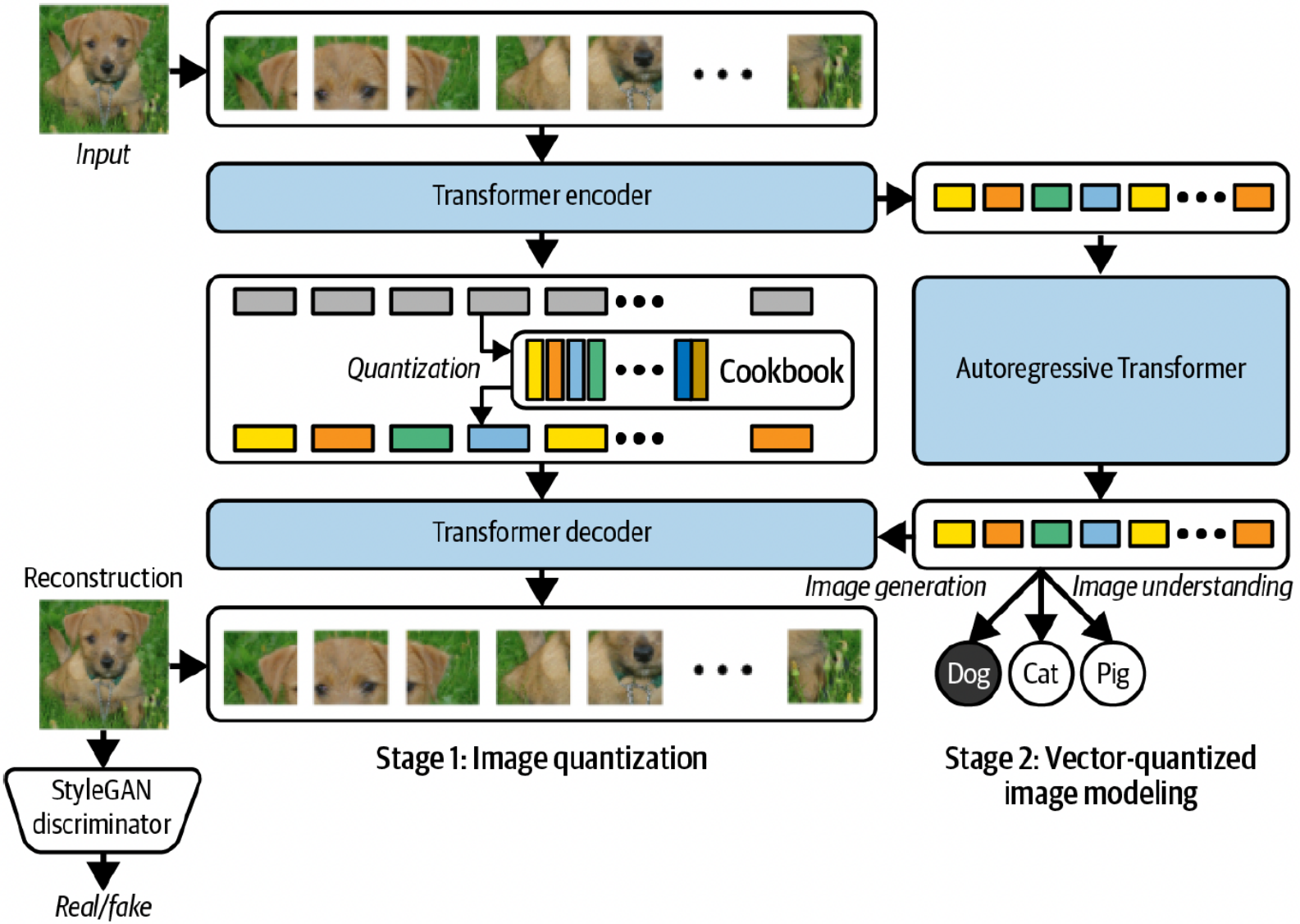
Components:

- Transformer encoder.
- Transformer decoder.
- Autoregressive Transformer prior.
- GAN discriminator.

Used in:

- DALL·E-style image generation.
- Image understanding and synthesis tasks.

ViT-VQ-GAN


Input
Transformer encoder
Quantization
Cookbook
Autoregressive Transformer
Transformer decoder
Reconstruction
Image generation
Image understanding
Dog
Cat
Pig
StyleGAN discriminator
Real/fake
Stage 1: Image quantization
Stage 2: Vector-quantized image modeling

ViT-VQ-GAN

Bibliography