COSE474 Deep Learning

Lecture 13: Generative Adversarial Networks (GANs)

Seungryong Kim

Computer Vision Lab. (CVLAB)

Department of Computer Science and Engineering

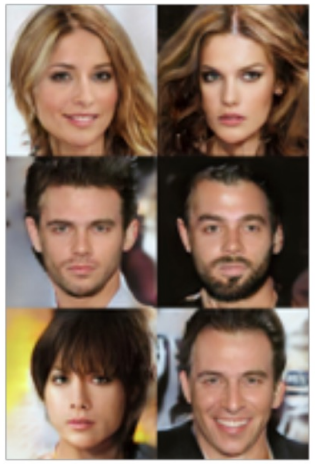
Korea University

Progress for Image Generation

2014 GAN

2015 DCGAN

2017 PG-GAN

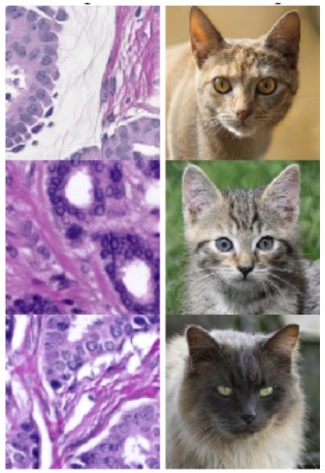


2018 StyleGAN

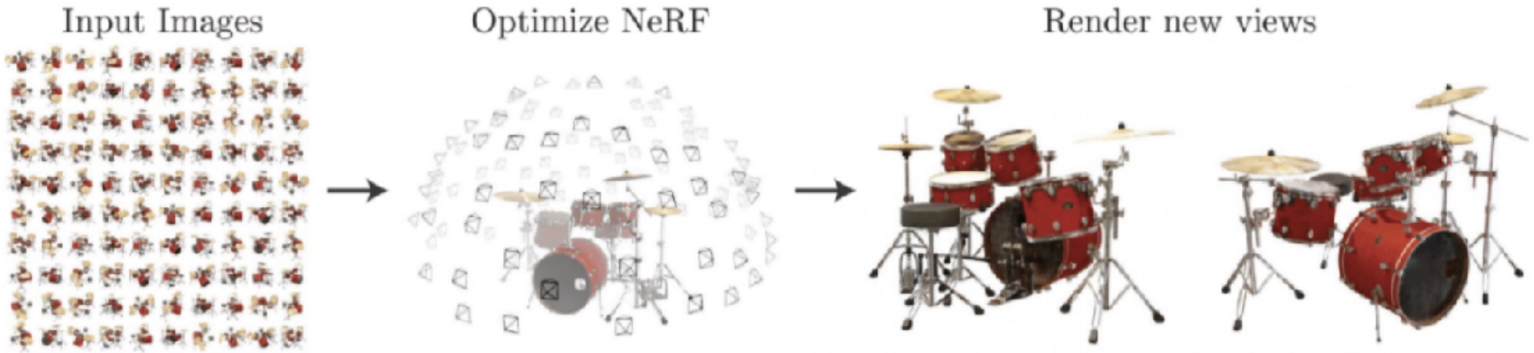
2018 BigGAN

2019 StyleGANv2

2020 StyleGAN-ADA



2020: NeRF (Neural Radiance Fields)



2021: OpenAI DALLE (VQ-VAE)

Arm chair in shape of avocado

What is Generative Model?

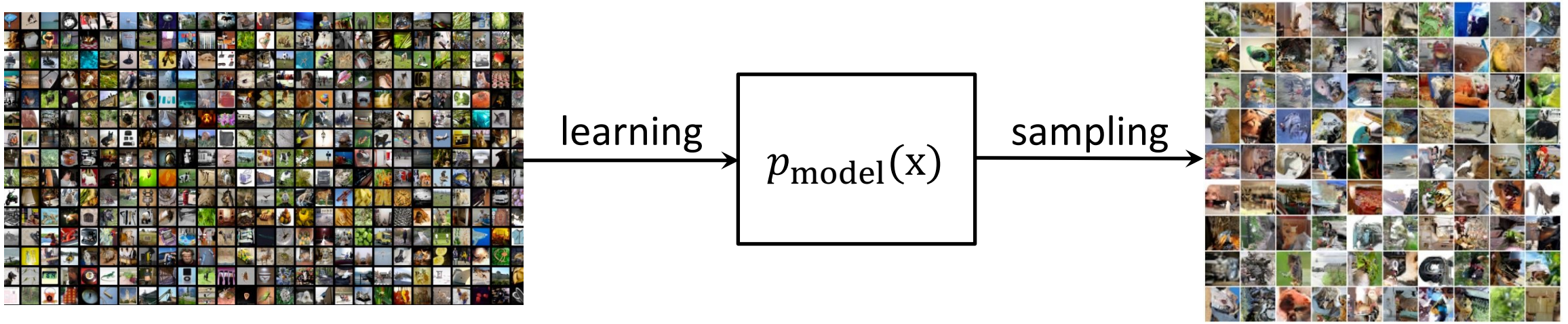
Generative Models

Discriminative model vs. Generative model

Model	Training	Test	Supervision
Discriminative model	Estimate $P(y\|\mathrm{x})$	$f:\mathrm{x}\rightarrow y$	Supervised learning
Generative model	Estimate $P(\mathrm{x})$ or $P(\mathrm{x},y)$	$f:\mathrm{seed}\rightarrow \mathrm{x}$ $f:\mathrm{seed},y\rightarrow \mathrm{x}$ $f:\mathrm{seed}\rightarrow \mathrm{x},y$	Unsupervised learning

Generative Models

Given train set, generate new samples from same distribution.



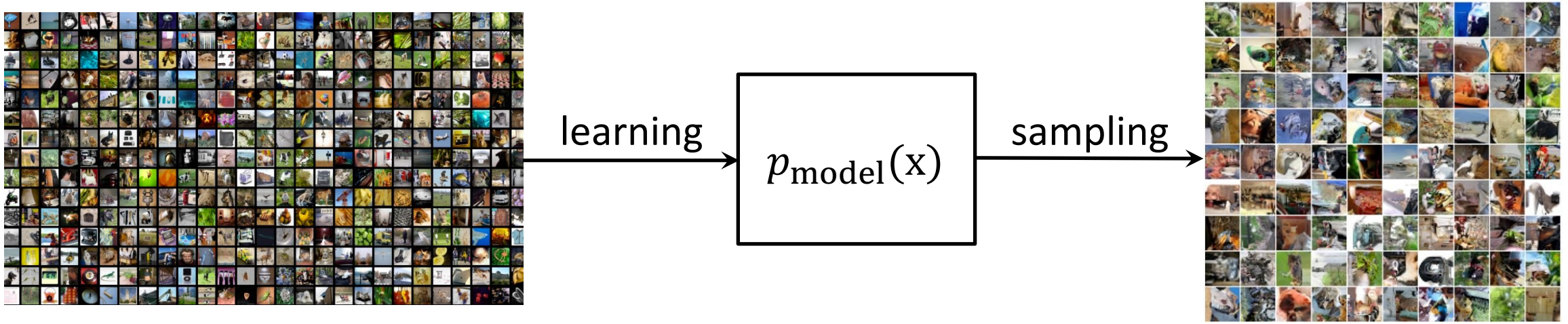
Training data $\sim p_{\text{data}}(\text{x})$

- **Objectives:**
  1. Learn $p_{\text{model}}(\text{x})$ that approximates $p_{\text{data}}(\text{x})$.
  2. Sampling new x from $p_{\text{model}}(\text{x})$.

[slide courtesy: Stanford, CS231]

Generative Models

Given train set, generate new samples from same distribution.



Training data $\sim p_{\text{data}}(\text{x})$

- **Formulate as density estimation problems:**
  - **Explicit density estimation:** explicitly define and solve for $p_{\text{model}}(\text{x})$.
  - **Implicit density estimation:** learn model that can sample from $p_{\text{model}}(\text{x})$ without explicitly defining it.

[slide courtesy: Stanford, CS231]

Taxonomy of Generative Models

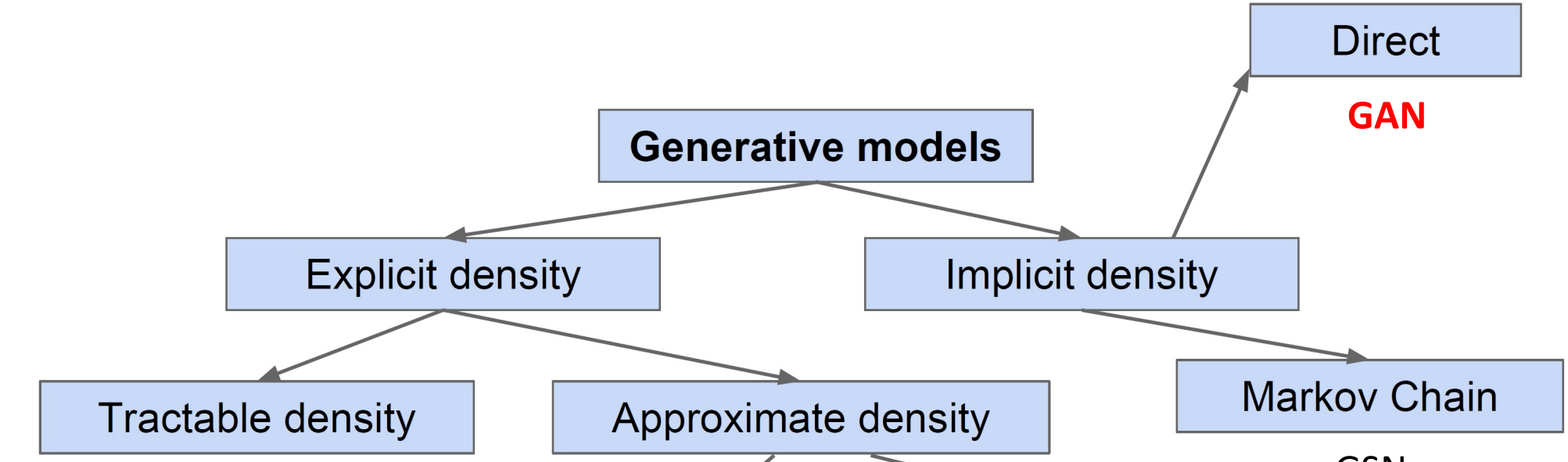


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

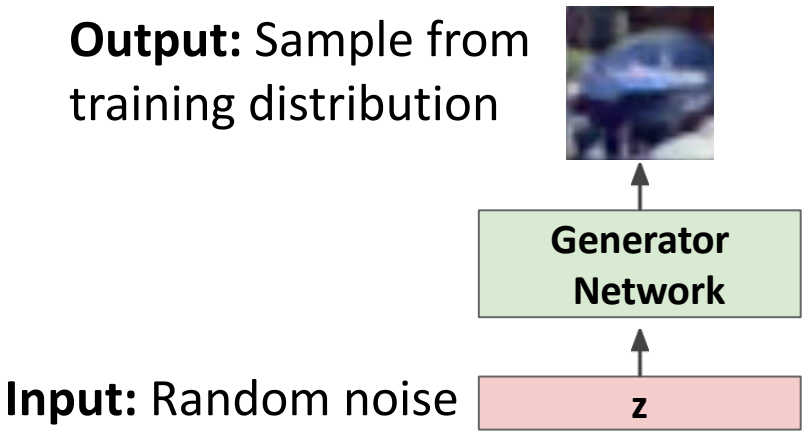
Generative Adversarial Networks (GANs)

Problem: Want to sample form complex, high-dimensional training distribution. No direct (or explicit) way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g., random noise. Learn transformation to training distribution.

Q: What can we use to present this complex transformation?

A: A neural network!



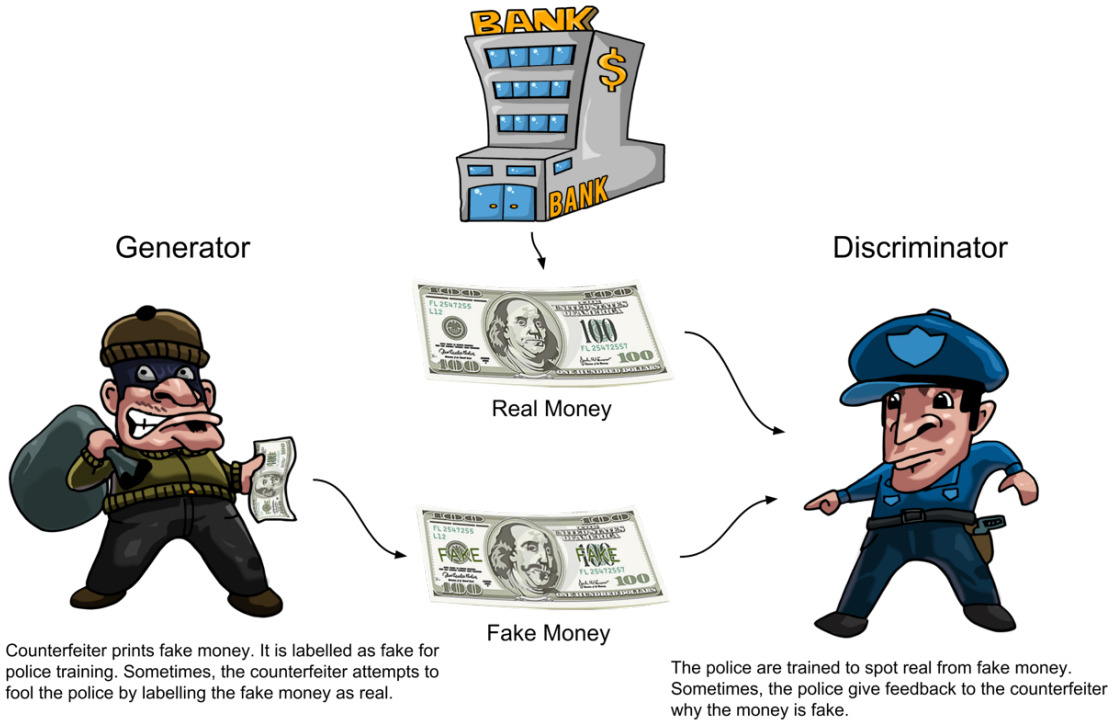
Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

[slide courtesy: Stanford, CS231]

Generative Adversarial Networks (GANs)

But we don't know which sample z maps to which training image → can't learn by reconstructing training images

Solution: Use a discriminator network to tell whether the generator image is within data distribution or not!

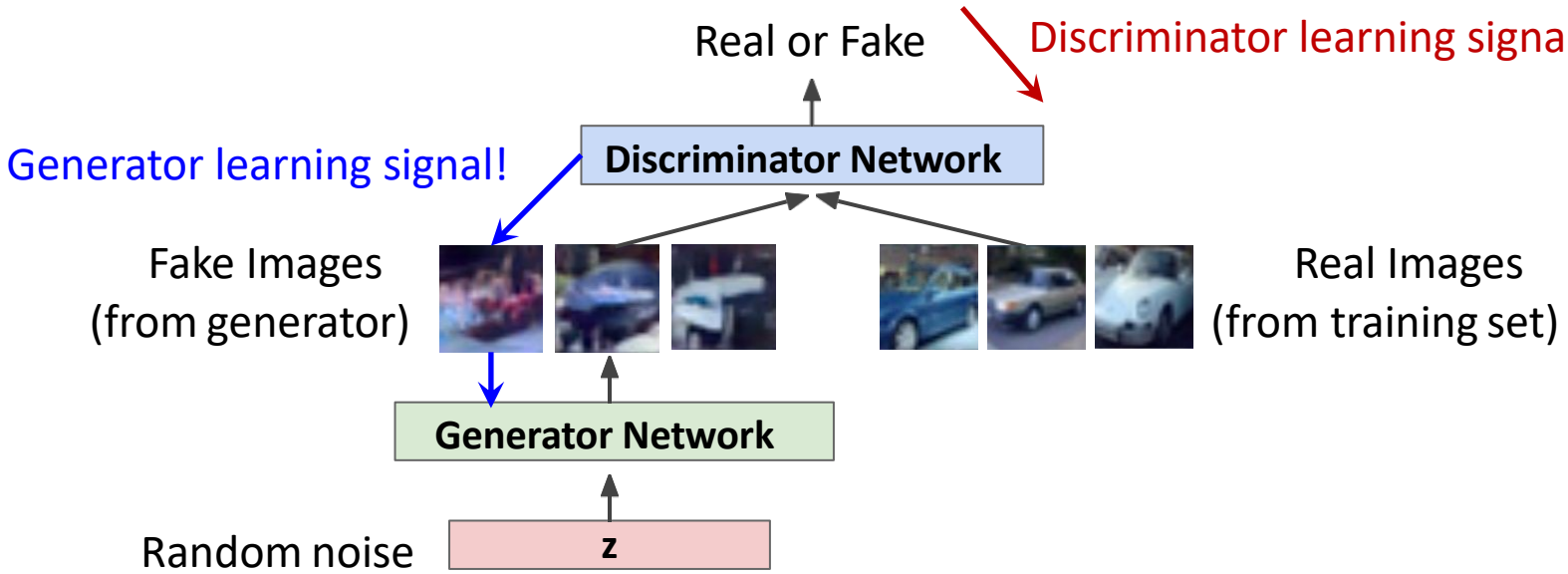


Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

[slide courtesy: Stanford, CS231]

Training GANs: Two-player Game

- **Discriminator network:** Try to distinguish between real and fake images
- **Generator network:** Try to fool the discriminator by generating real-looking images.



Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

[slide courtesy: Stanford, CS231]

Training GANs: Two-player Game

- **Discriminator network:** Try to distinguish between real and fake images
- **Generator network:** Try to fool the discriminator by generating real-looking images.
- Train jointly in minimax game!
- Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Generator Objective

Discriminative Objective

Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

[slide courtesy: Stanford, CS231]

Training GANs: Two-player Game

- **Discriminator network:** Try to distinguish between real and fake images
- **Generator network:** Try to fool the discriminator by generating real-looking images.
- Train jointly in minimax game!
- Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data $G(x)$

Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

[slide courtesy: Stanford, CS231]

Training GANs: Two-player Game

Minimax Objective Function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data $G(x)$

- Discriminator (θ_d) wants to **maximize objective** such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake).
- Generator (θ_g) wants to **minimize objective** such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real)

Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

[slide courtesy: Stanford, CS231]

Training GANs: Two-player Game

Minimax Objective Function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

- Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

1. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

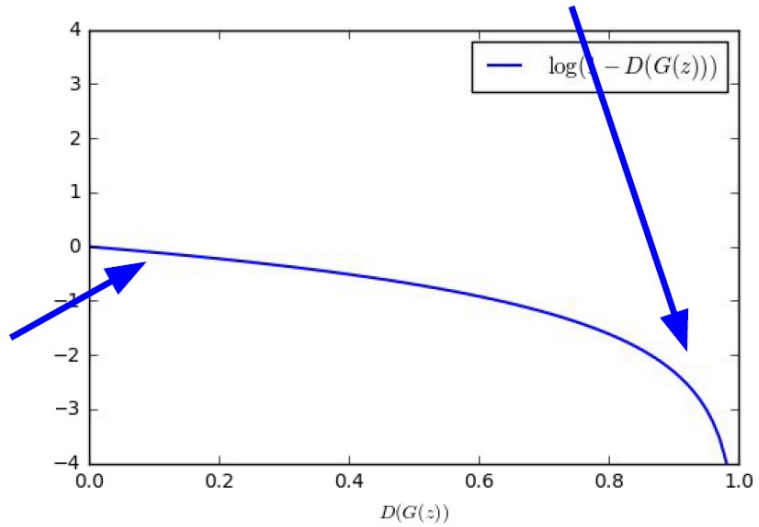
[slide courtesy: Stanford, CS231]

Training GANs: Two-player Game

Minimax Objective Function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Gradient signal dominated by region where sample is already good



In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!

Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

[slide courtesy: Stanford, CS231]

Training GANs: Two-player Game

Minimax Objective Function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

- Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

1. **Gradient ascent** on generator

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log D_{\theta_d}(G_{\theta_g}(z))$$

Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

[slide courtesy: Stanford, CS231]

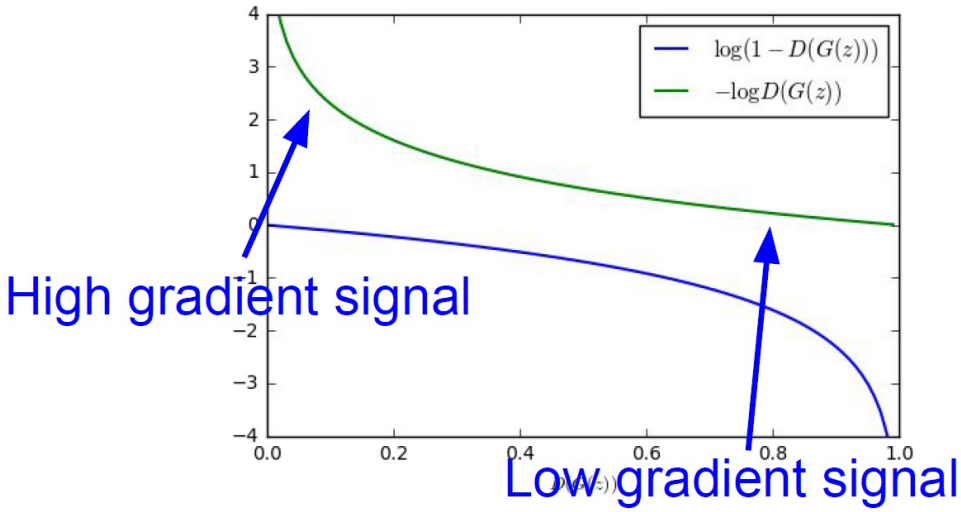
Training GANs: Two-player Game

Minimax Objective Function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.

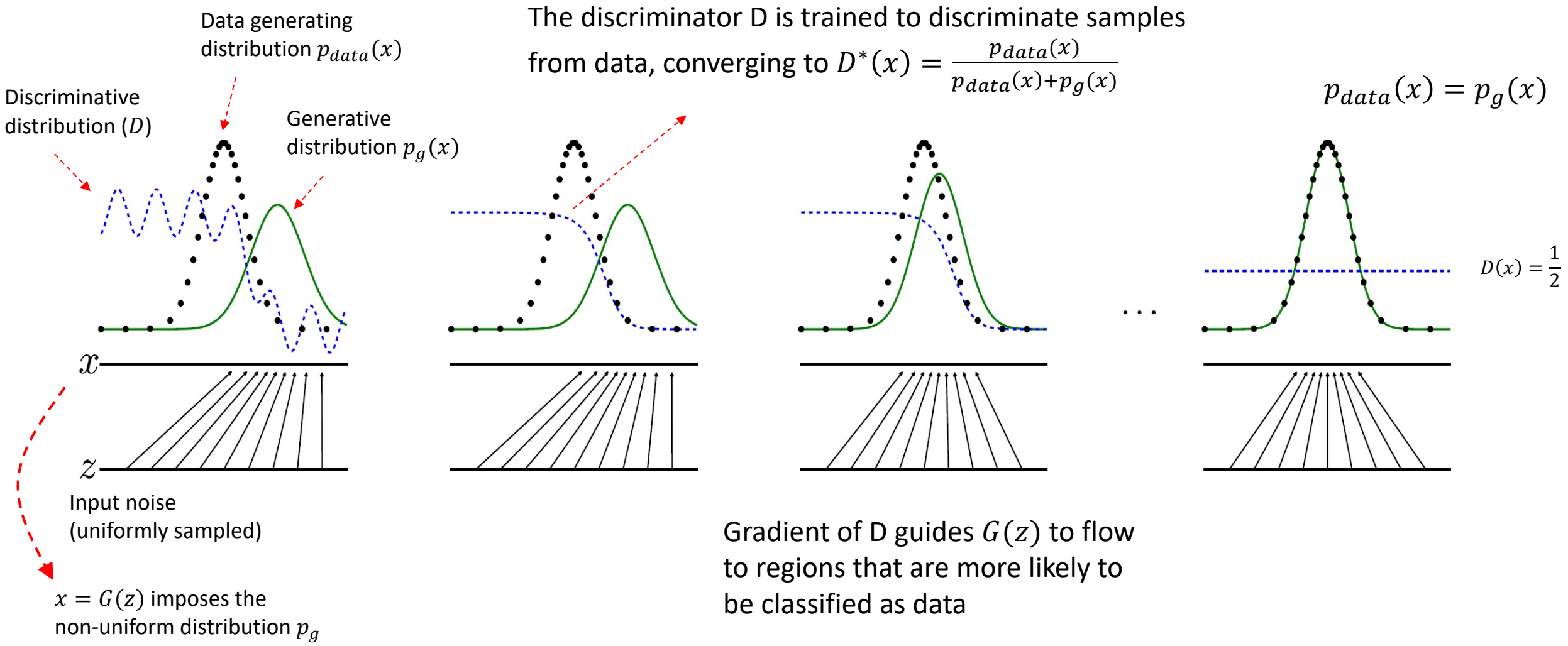
Minimizing likelihood of discriminator being correct
-> Maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples
-> Works much better! Standard in practice.



Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

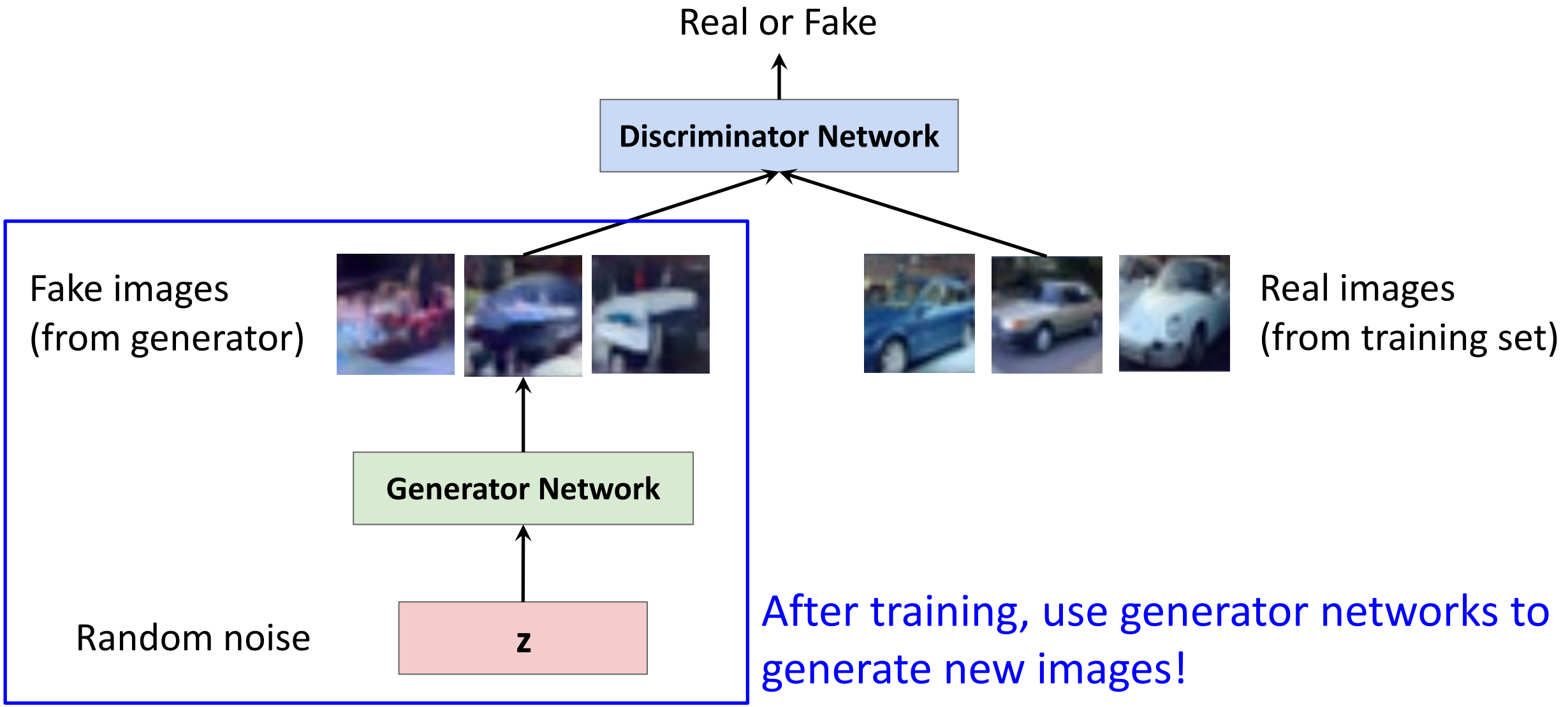
[slide courtesy: Stanford, CS231]

Training GANs: Two-player Game



Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

Training GANs: Two-player Game



After training, use generator networks to generate new images!

Goodfellow et al., "Generative Adversarial Nets," NIPS, 2014

[slide courtesy: Stanford, CS231]

Advanced GANs Model

- DCGAN, Progressive GANs, StyleGAN etc.

Deep Convolutional GANs (DCGAN)

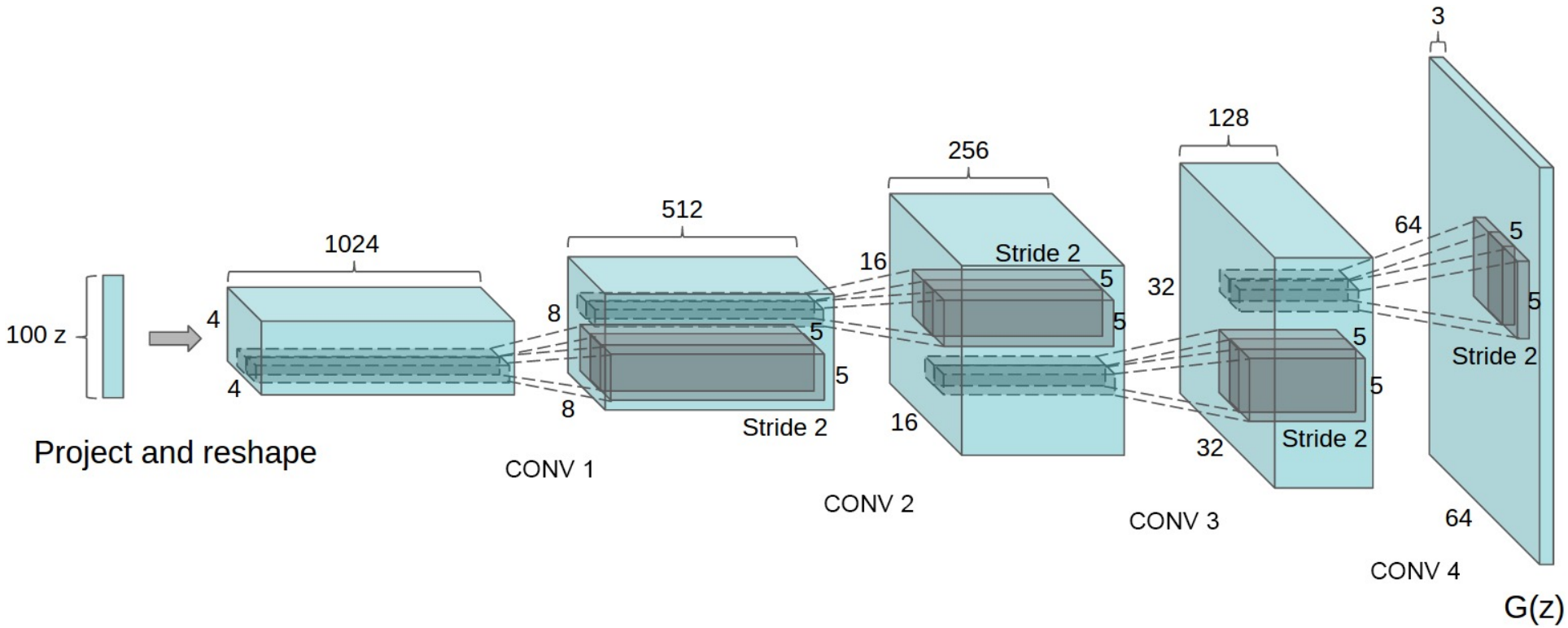
- Generator is an upsampling network with fractionally-strided convolutions.
- Discriminator is a convolutional network.

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Deep Convolutional GANs (DCGAN)



- A 100-D uniform distribution z is projected to a small spatial extent convolutional representation using four fractionally-strided convolutions.
- No fully connected or pooling layers are used.

Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

Deep Convolutional GANs (DCGAN)

Samples from the model look amazing!

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Deep Convolutional GANs (DCGAN)

Interpolating between random points in latent space

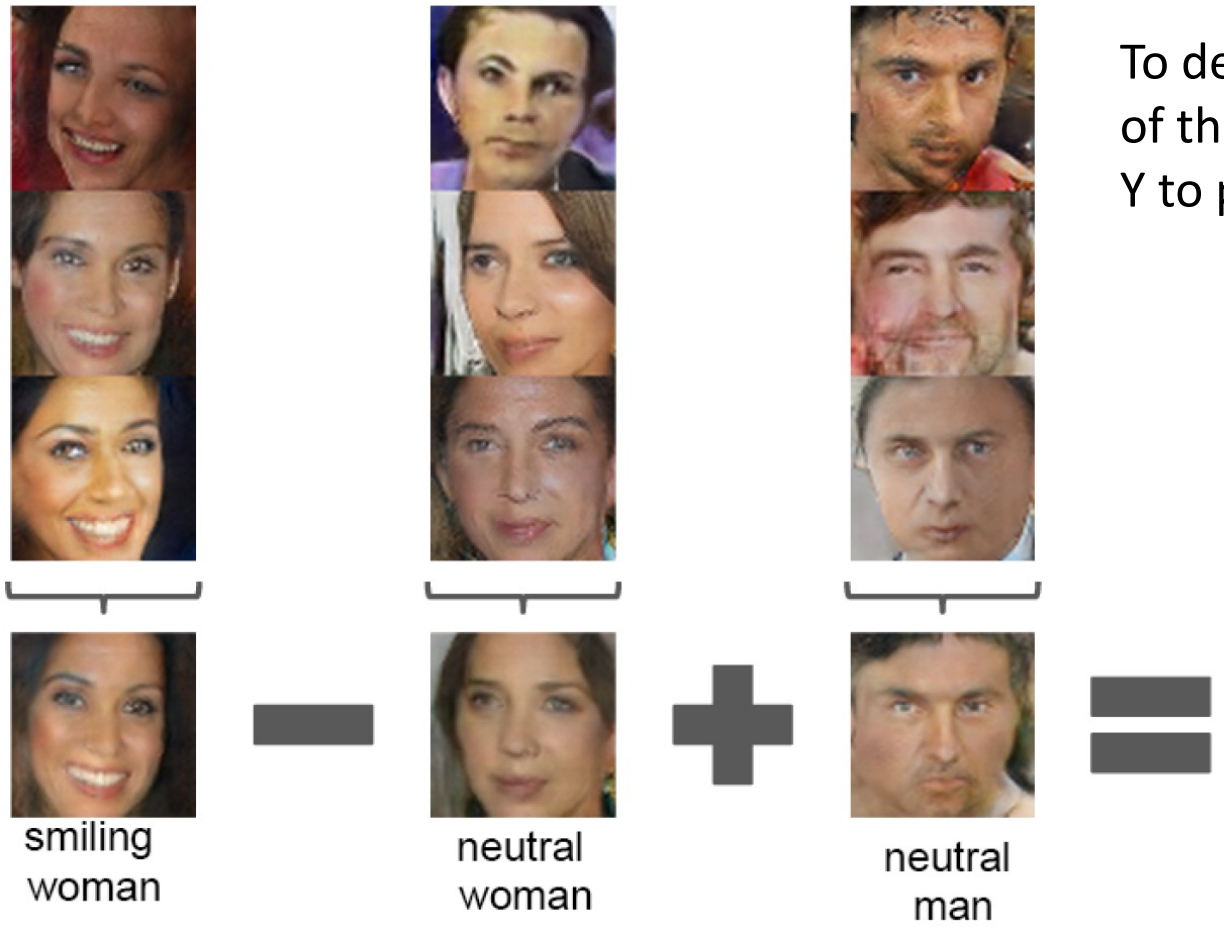
Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Deep Convolutional GANs (DCGAN)

Vector arithmetic on latent space z for visual concepts

Samples from the model

Average z vectors, and then do arithmetic to create Y



The center sample is produced by feeding Y to the generator.

To demonstrate the interpolation capabilities of the generator, uniform noise was added to Y to produce the 8 other samples.

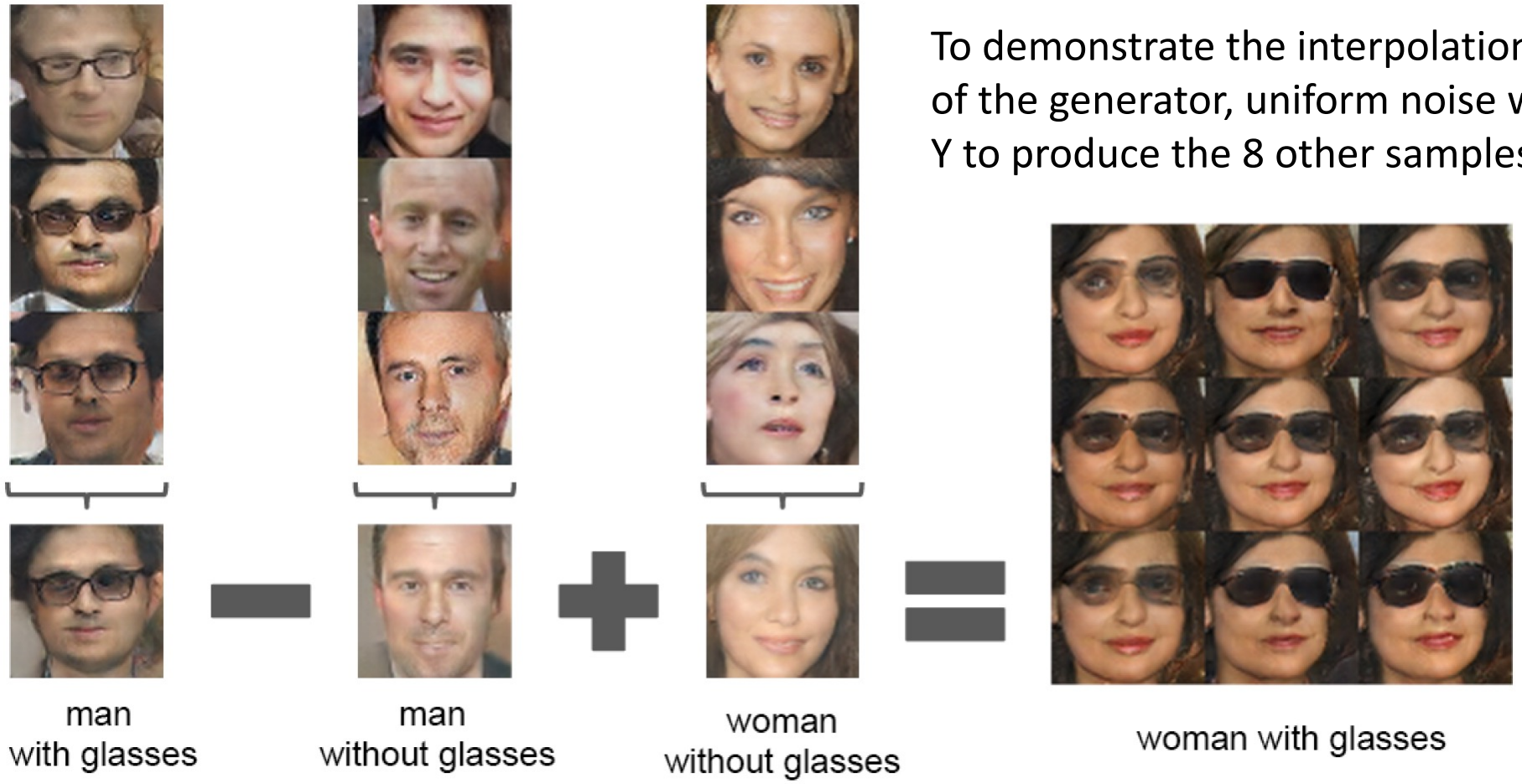
Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

Deep Convolutional GANs (DCGAN)

Vector arithmetic on latent space z for visual concepts

Samples from the model

Average z vectors, and then do arithmetic to create Y



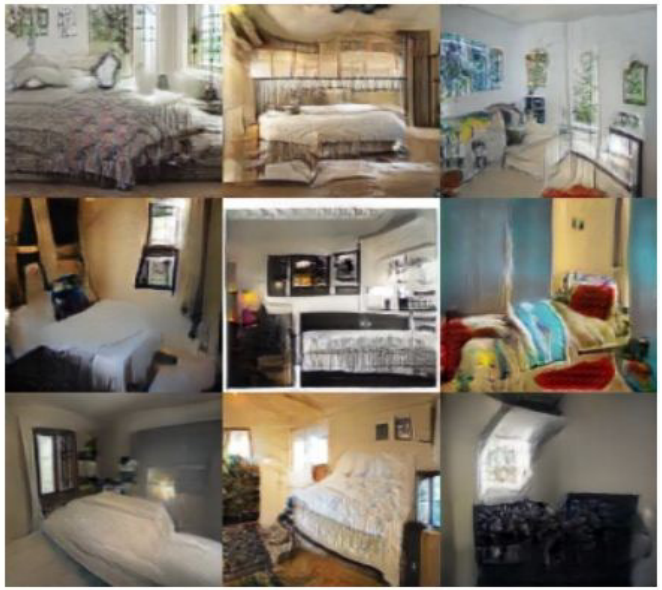
The center sample is produced by feeding Y to the generator.

To demonstrate the interpolation capabilities of the generator, uniform noise was added to Y to produce the 8 other samples.

Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

2017: Explosion of GANs

Better Training and Generation



LSGAN, Zhu, 2017

Wasserstein GAN,
Arjovsky 2017,
Improved Wasserstein GAN,
Gulrajani, 2017

Progressive GAN,
Karra, 2018

[slide courtesy: Stanford, CS231]

2017: Explosion of GANs

Source to Target Domain Transfer

CycleGAN, Zhu et al., 2017

Reed et al.,, 2017

Pix2Pix, Isola, 2017

[slide courtesy: Stanford, CS231]

2019: BigGAN

Brock et al., 2019

[slide courtesy: Stanford, CS231]

Other Loss Functions

Name	Paper Link	Value Function
GAN	Arxiv	$L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$ $L_G^{GAN} = E[\log(D(G(z)))]$
LSGAN	Arxiv	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	Arxiv	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$
WGAN_GP	Arxiv	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha G(z)))\rvert - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	Arxiv	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha x_p))\rvert - 1)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$
CGAN	Arxiv	$L_D^{CGAN} = E[\log(D(x, c))] + E[\log(1 - D(G(z), c))]$ $L_G^{CGAN} = E[\log(D(G(z), c))]$
infoGAN	Arxiv	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c, c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c, c')$
ACGAN	Arxiv	$L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c\|x)] + E[P(class = c\|G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c\|G(z))]$
EBGAN	Arxiv	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$
BEGAN	Arxiv	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$

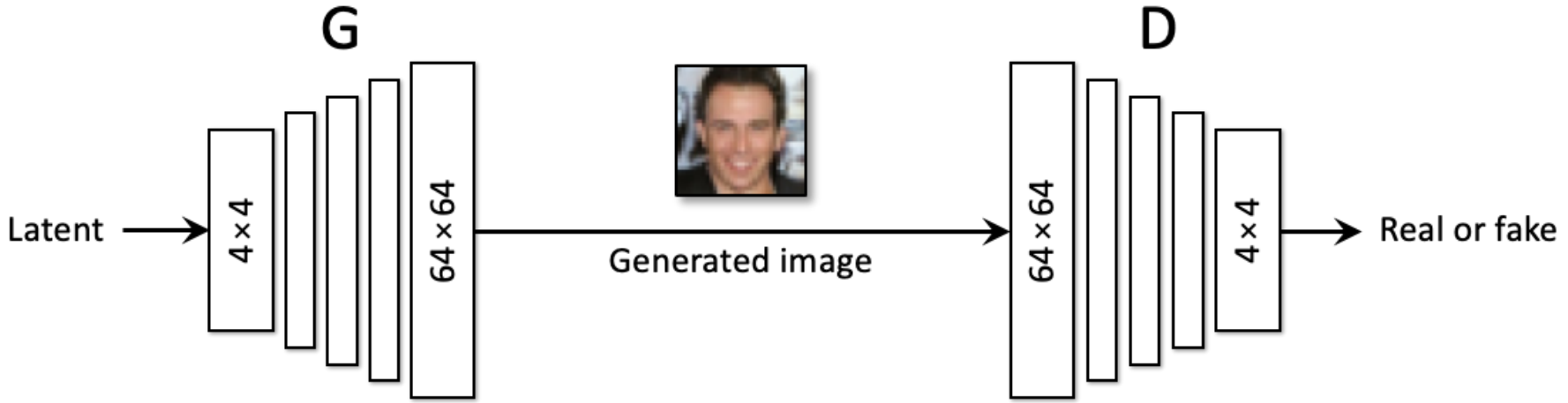
Method	Local convergence (a.c. case)	Local convergence (general case)
unregularized (Goodfellow et al., 2014)	✓	✗
WGAN (Arjovsky et al., 2017)	✗	✗
WGAN-GP (Gulrajani et al., 2017)	✗	✗
DRAGAN (Kodali et al., 2017)	✓	✗
Instance noise (Sønderby et al., 2016)	✓	✓
ConOpt (Mescheder et al., 2017)	✓	✓
Gradient penalties (Roth et al., 2017)	✓	✓
Gradient penalty on real data only	✓	✓
Gradient penalty on fake data only	✓	✓

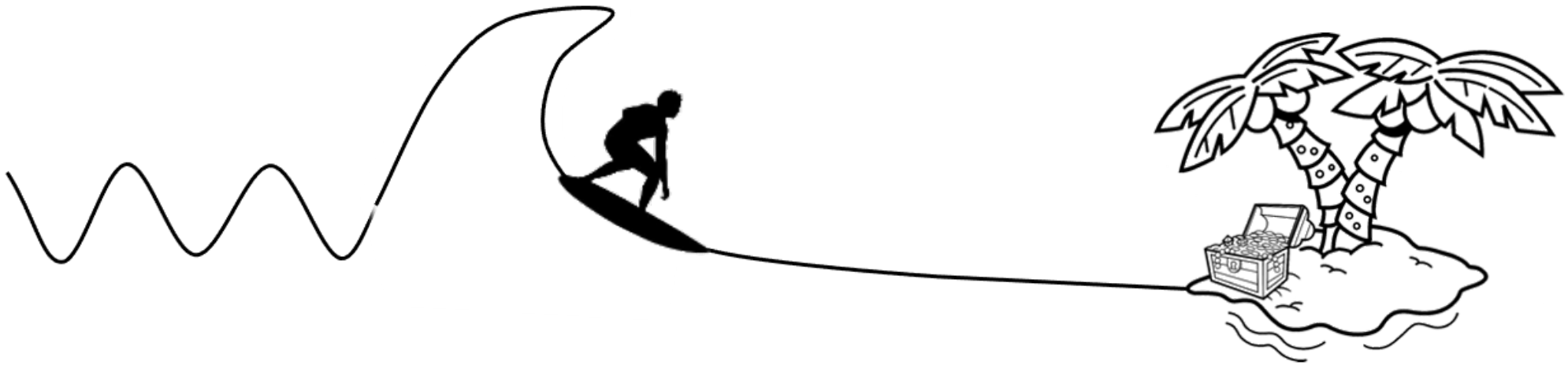
Table 1. Convergence properties of different GAN training algorithms for general GAN-architectures. Here, we distinguish between the case where both the data and generator distributions are absolutely continuous (a.c.) and the general case where they may lie on lower dimensional manifolds.

https://github.com/hwalsuklee/tensorflow-generative-model-collections
https://github.com/LMescheder/GAN_stability
Which Training Methods for GANs do actually Converge?

Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

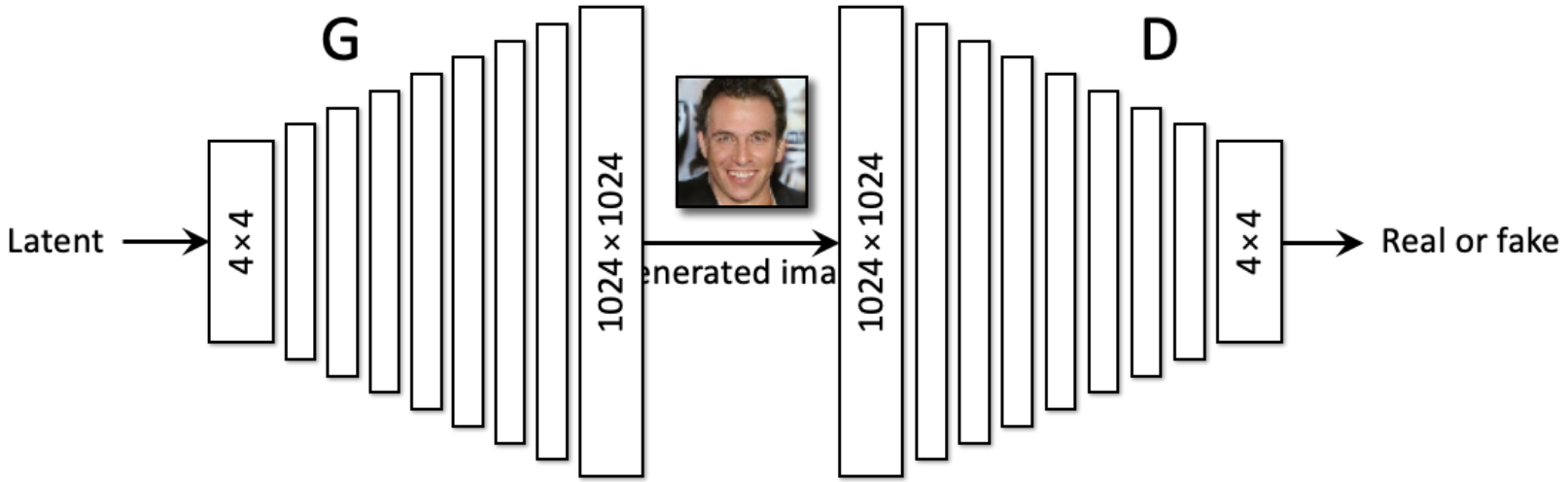
Progressive Generation

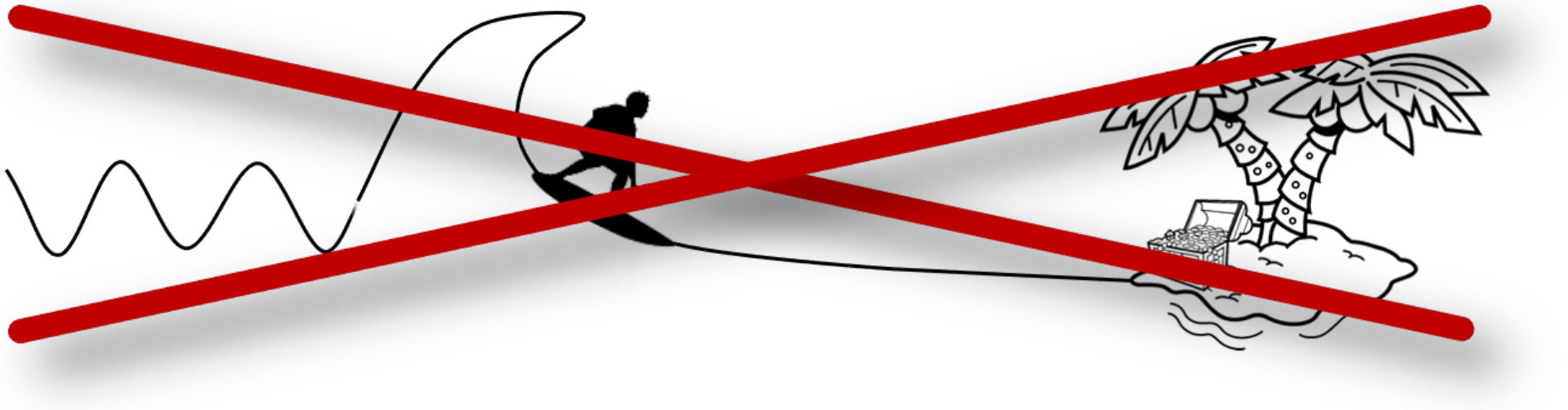




Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

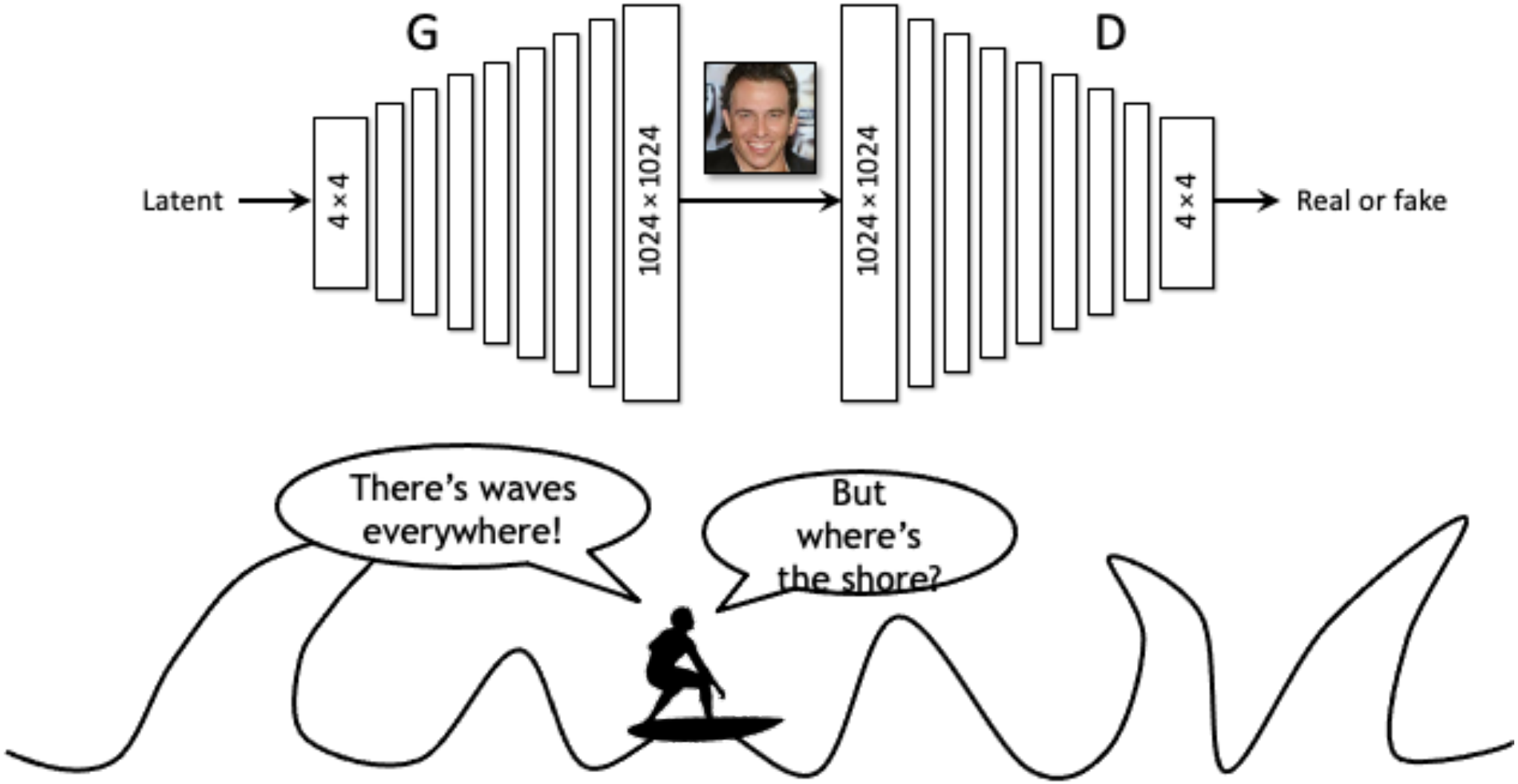
Progressive Generation





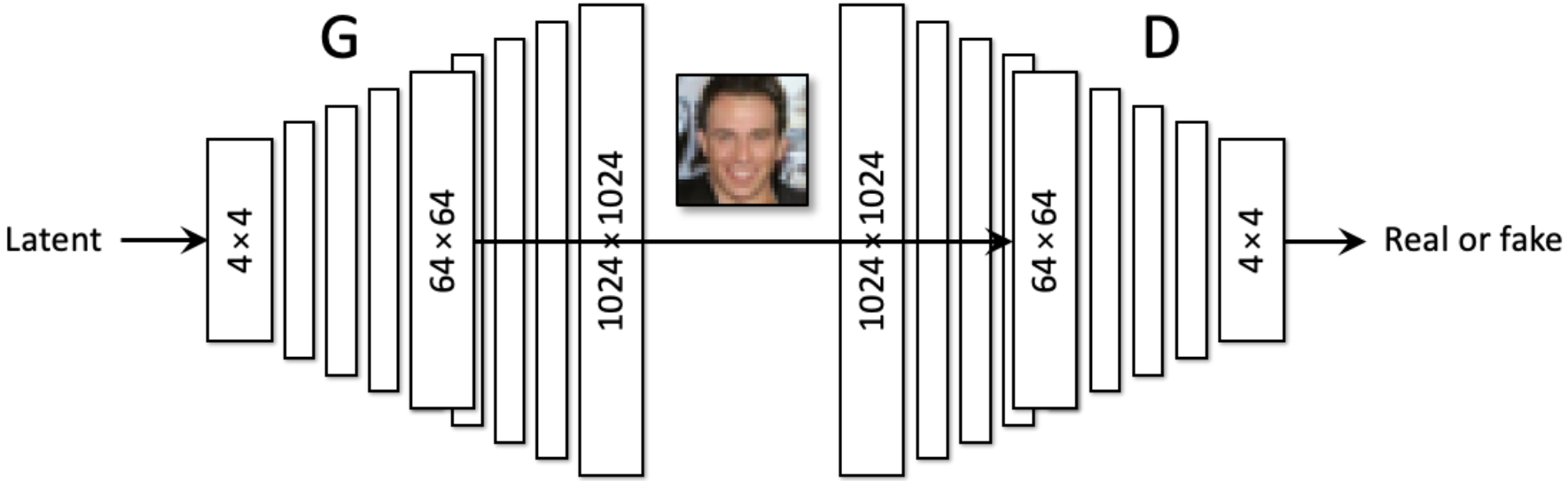
Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

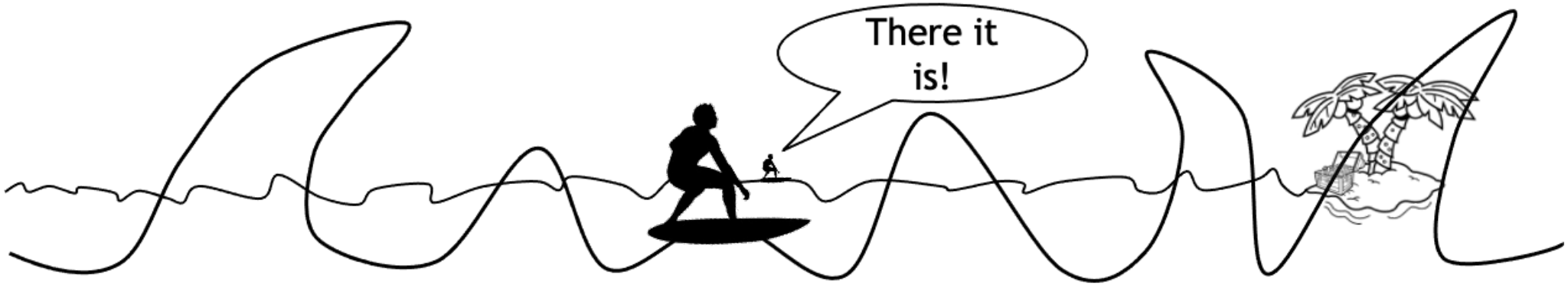
Progressive Generation



Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

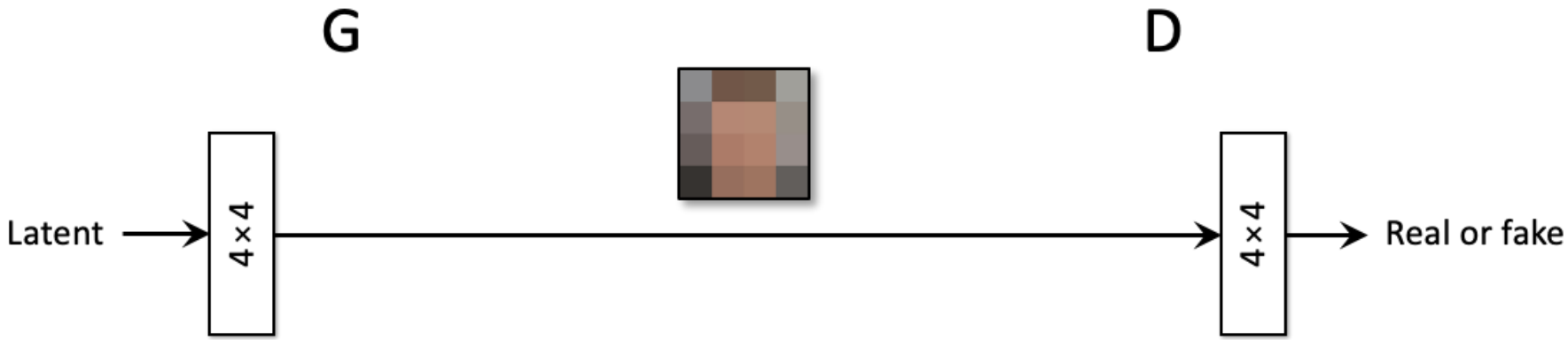
Progressive Generation

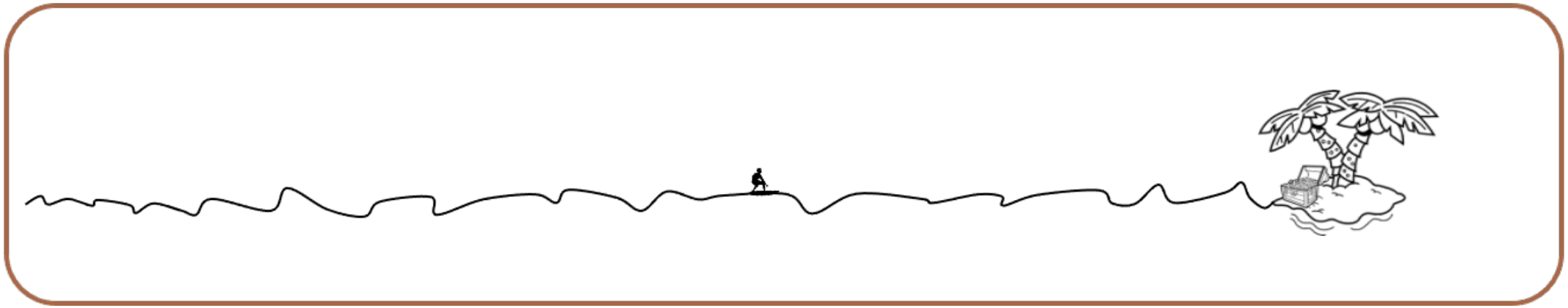




Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

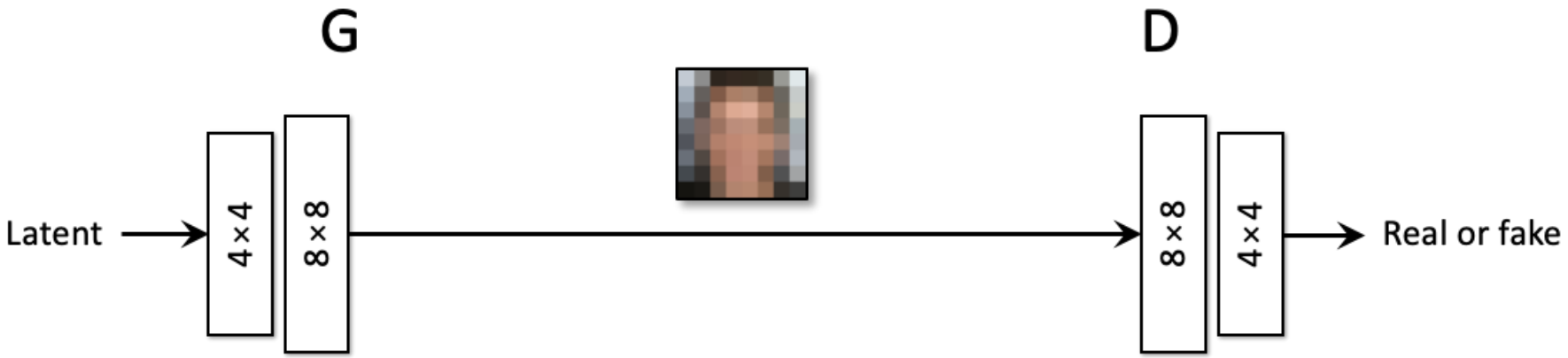
Progressive Generation

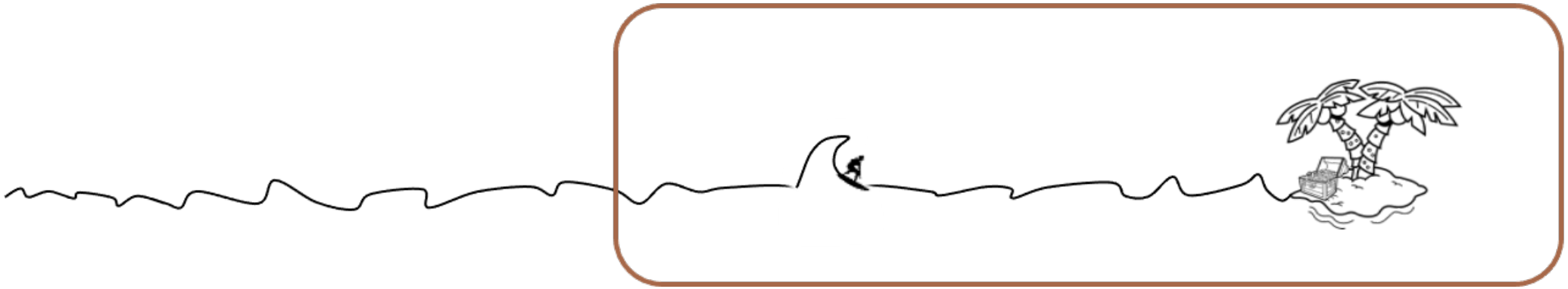




Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

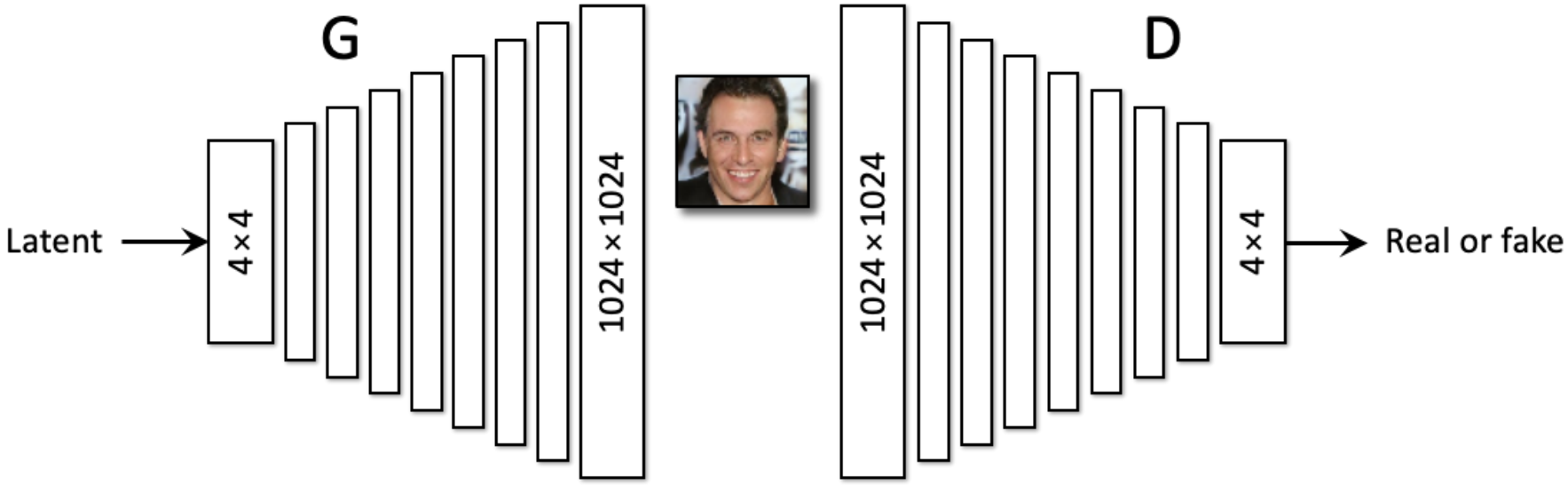
Progressive Generation

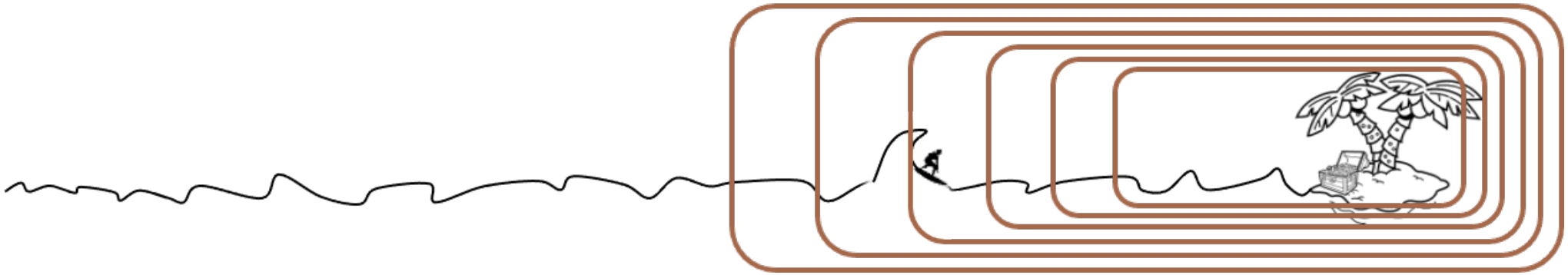




Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

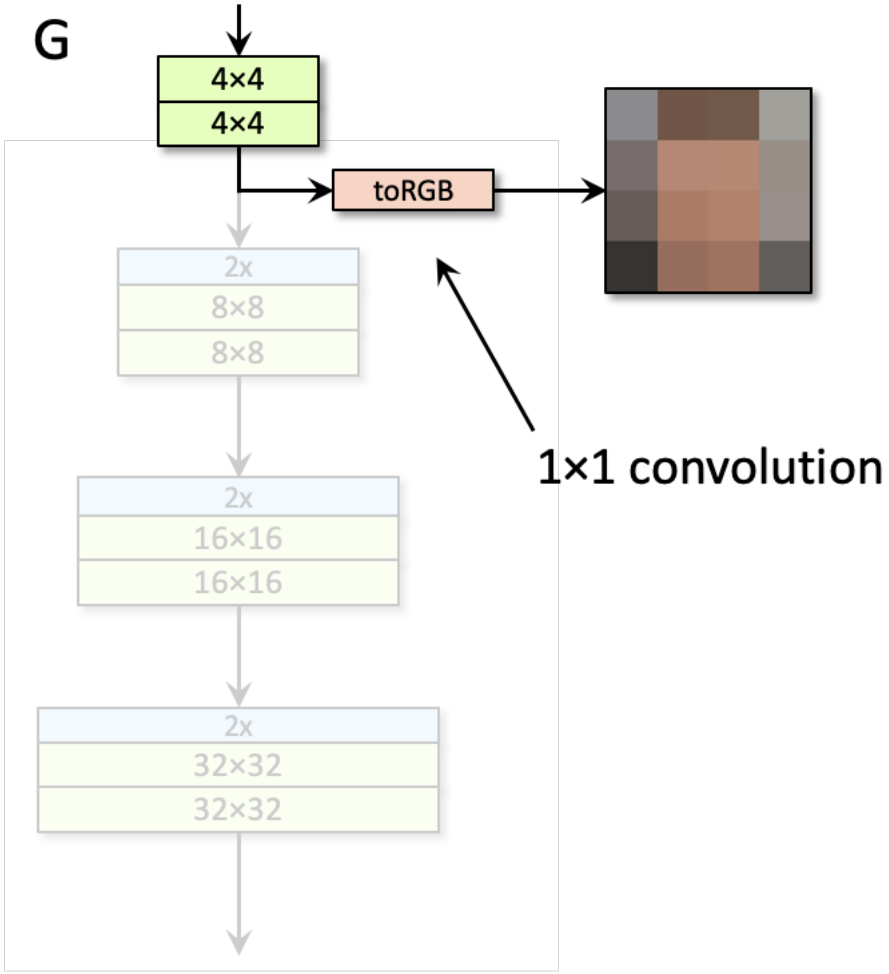
Progressive Generation





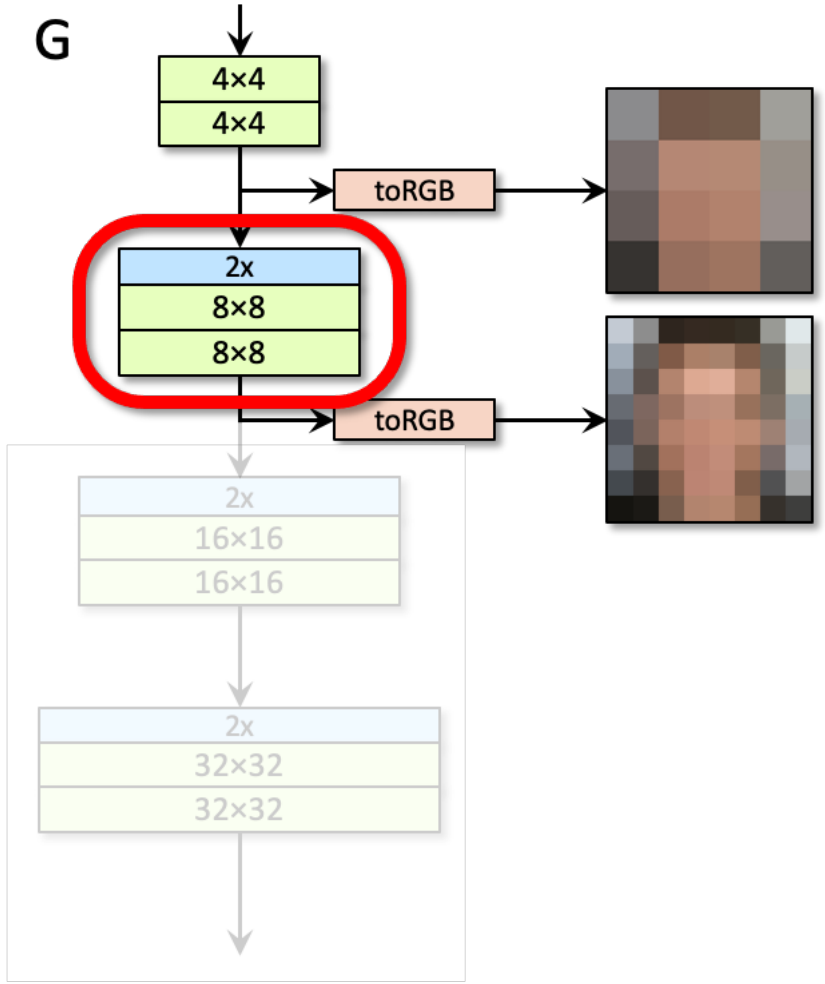
Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

Progressive Generation



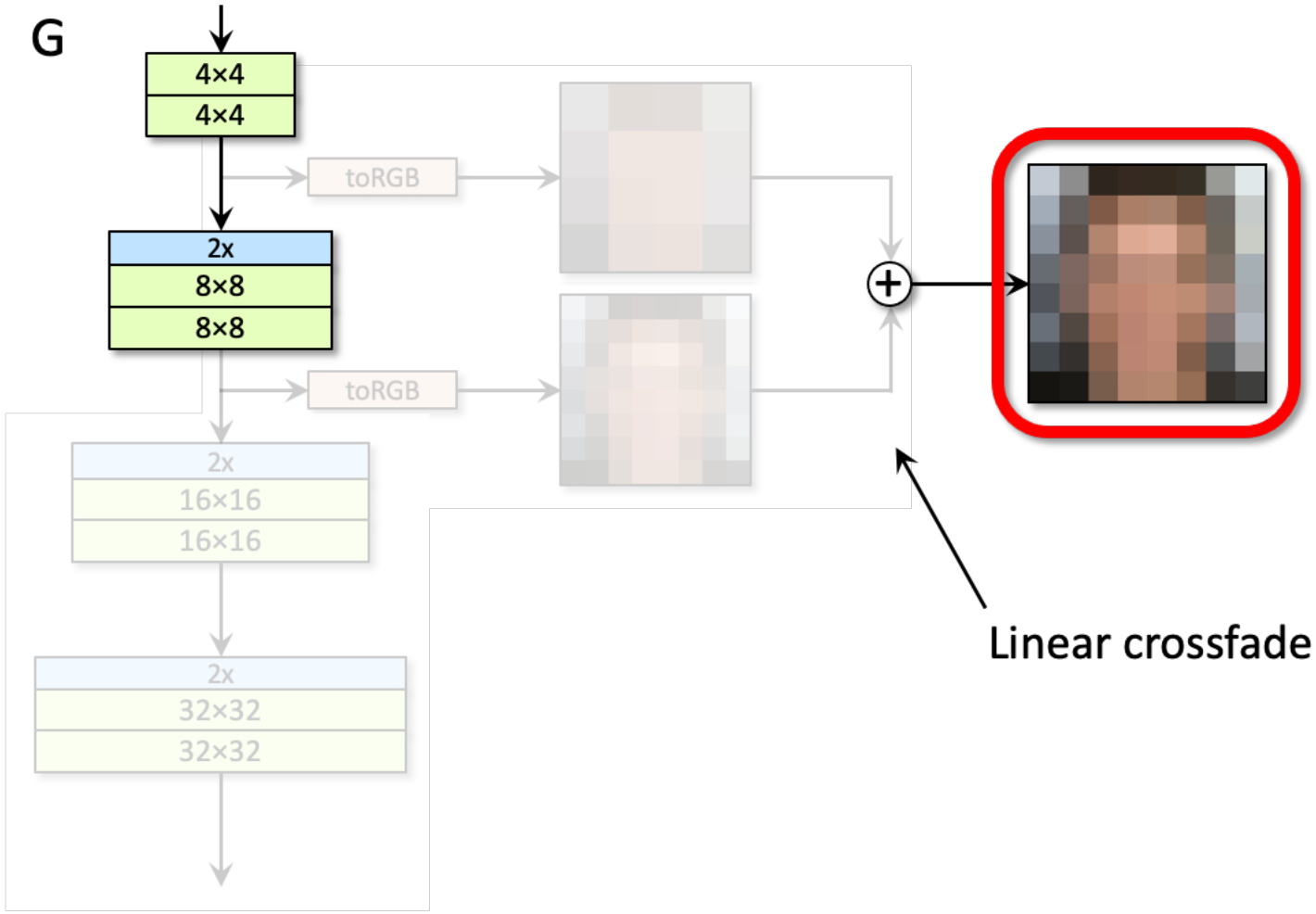
Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

Progressive Generation

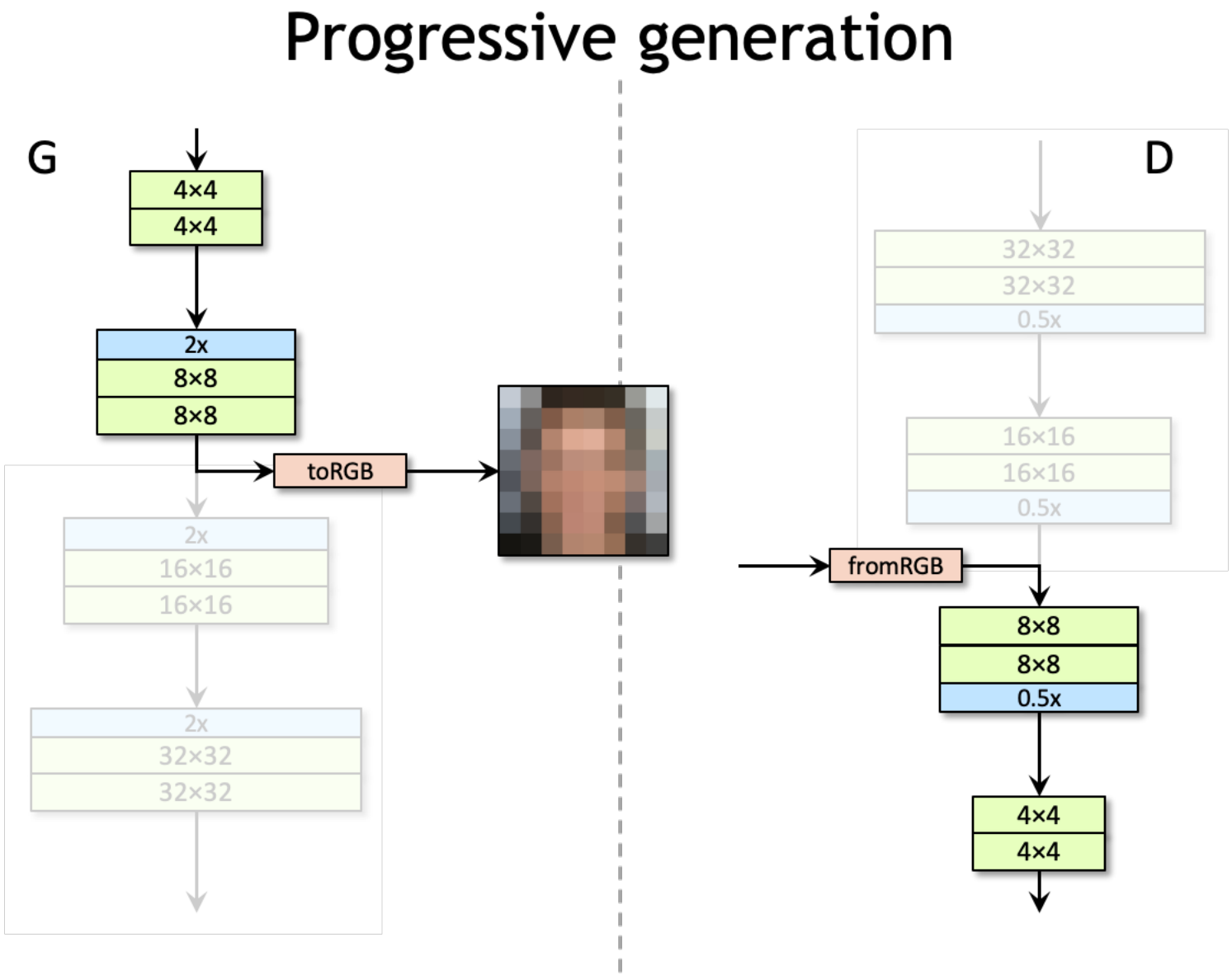


Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

Progressive Generation



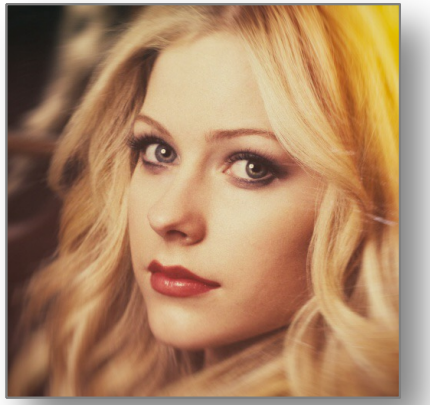
Progressive Generation



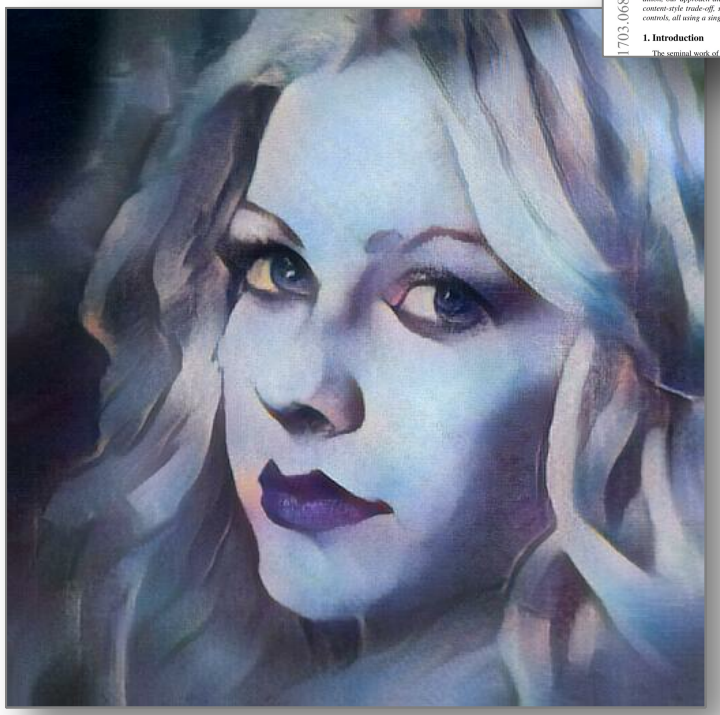
Huang et al., Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, ICCV, 2017

AdaIN in Style Transfer

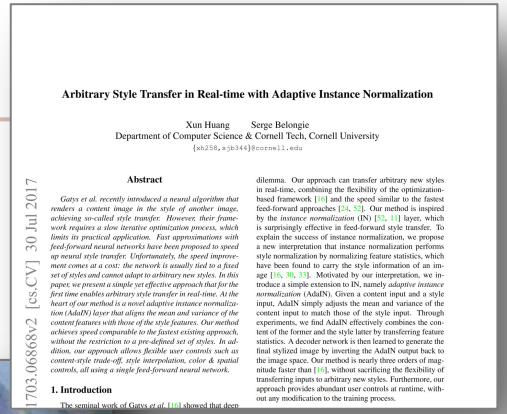
Style



Content



Result



Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

Xun Huang Serge Belongie
Department of Computer Science & Cornell Tech, Cornell University
{xh258,sjb344}@cornell.edu

Abstract

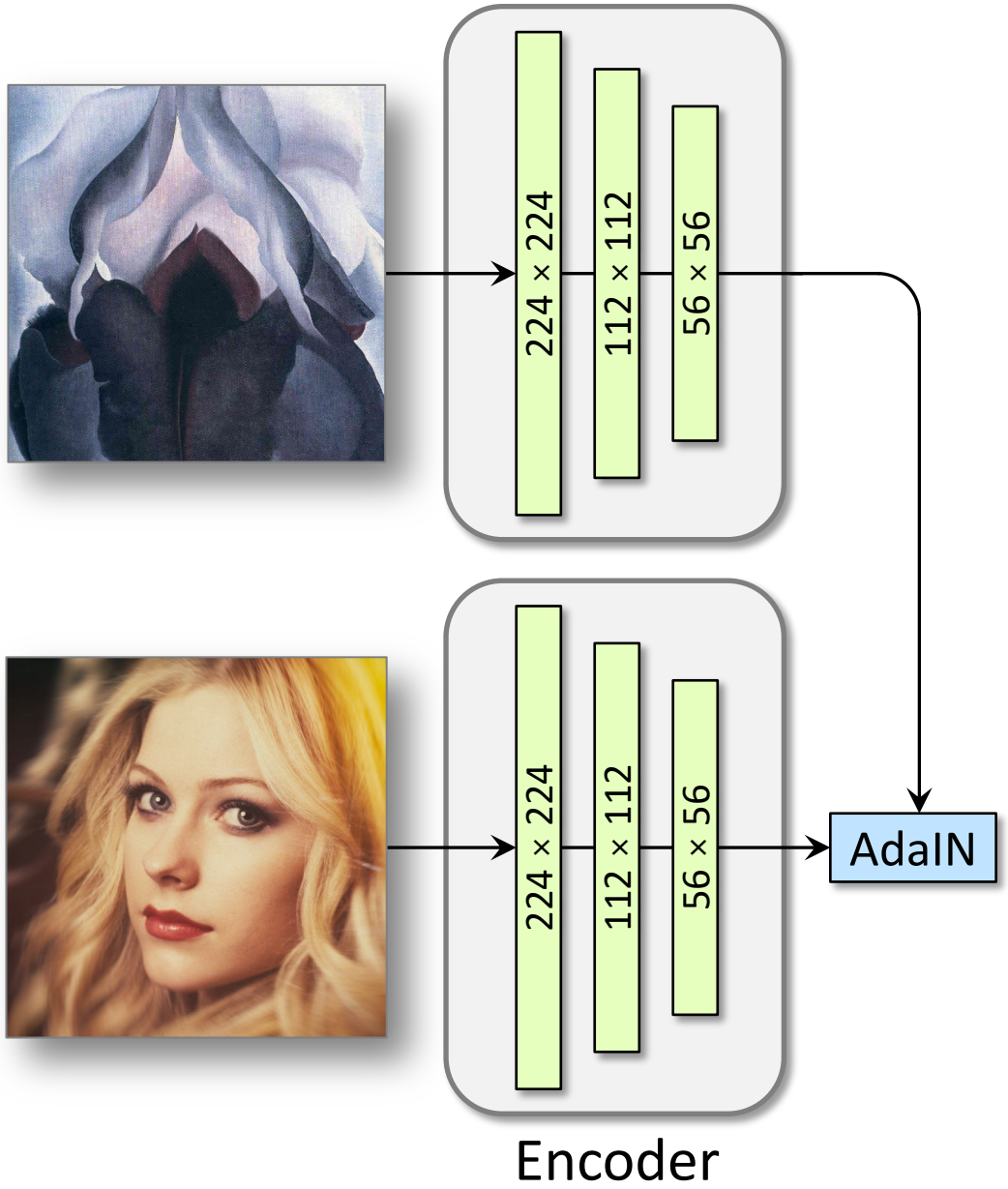
Gatys et al. recently introduced a neural algorithm that renders a content image in the style of another image, achieving so-called style transfer. However, their framework requires a slow iterative optimization process, which limits its practical application. Fast approximations with feed-forward neural networks have been proposed to speed up neural style transfer. Unfortunately, the speed improvement comes at a cost: the network is usually tied to a fixed set of styles and cannot adapt to arbitrary new styles. In this paper, we present a simple yet effective approach that for the first time enables arbitrary style transfer in real-time. At the heart of our method is a novel adaptive instance normalization (AdaIN) layer that aligns the mean and variance of the content features with those of the style features. Our method achieves speed comparable to the fastest existing approach, without the restriction to a pre-defined set of styles. In addition, our approach allows flexible user controls such as content-style trade-off, style interpolation, color & spatial controls, all using a single feed-forward neural network.

1. Introduction

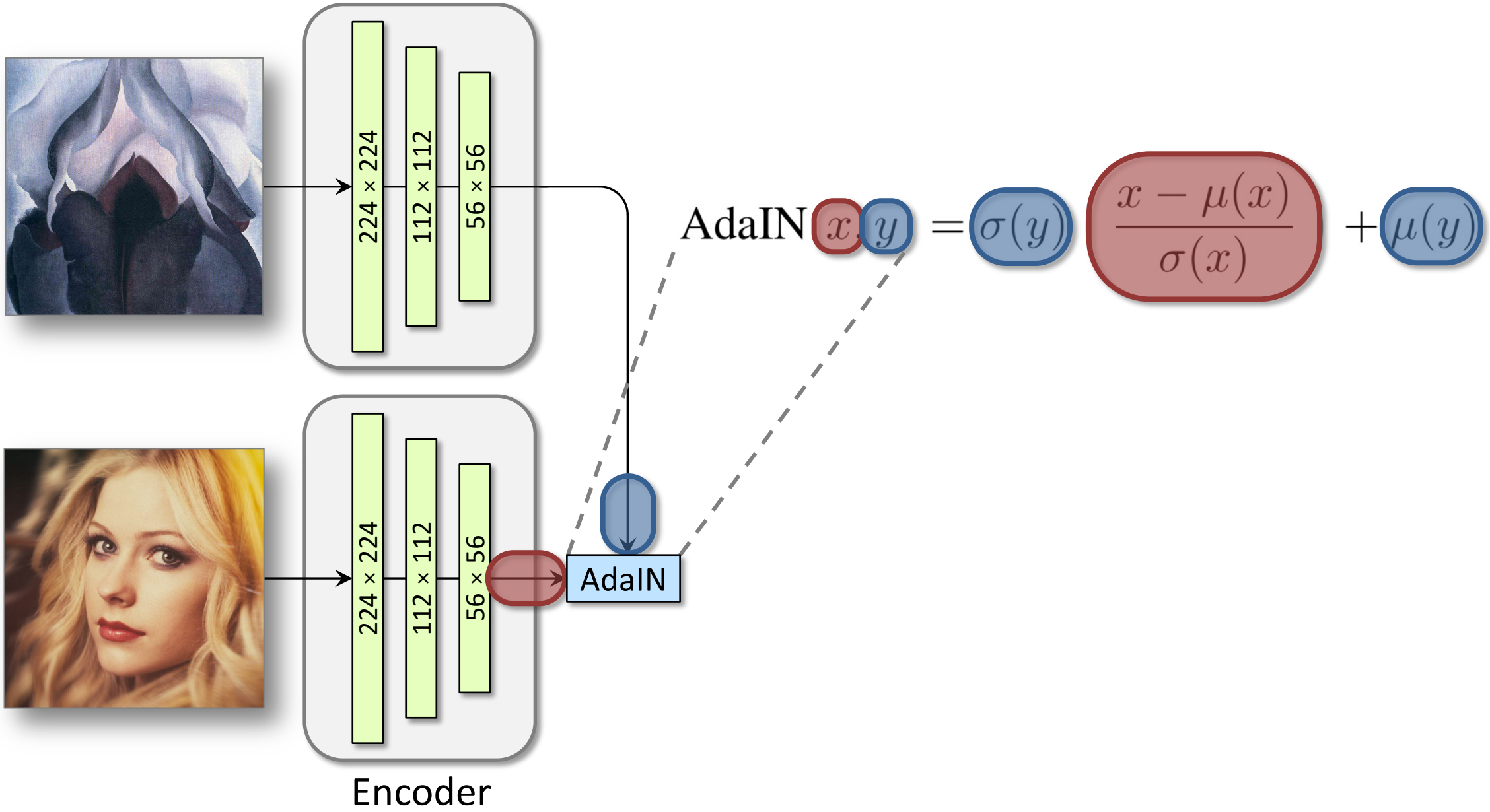
The seminal work of Gatys et al. [16] showed that deep

dilemma. Our approach can transfer arbitrary new styles in real-time, combining the flexibility of the optimization-based framework [16] and the speed similar to the fastest feed-forward approaches [24, 32]. Our method is inspired by the *instance normalization* (IN) [52, 11] layer, which is surprisingly effective in feed-forward style transfer. To explain the success of instance normalization, we propose a new interpretation that instance normalization performs style normalization by normalizing feature statistics, which have been found to carry the style information of an image [16, 30, 37]. Motivated by our interpretation, we introduce a simple extension to IN, namely *adaptive instance normalization* (AdaIN). Given a content input and a style input, AdaIN simply adjusts the mean and variance of the content input to match those of the style input. Through experiments, we find AdaIN effectively combines the content of the former and the style latter by transferring feature statistics. A decoder network is then learned to generate the final stylized image by inverting the AdaIN output back to the image space. Our method is nearly three orders of magnitude faster than [16], without sacrificing the flexibility of transferring inputs to arbitrary new styles. Furthermore, our approach provides abundant user controls at runtime, without any modification to the training process.

StyleGAN

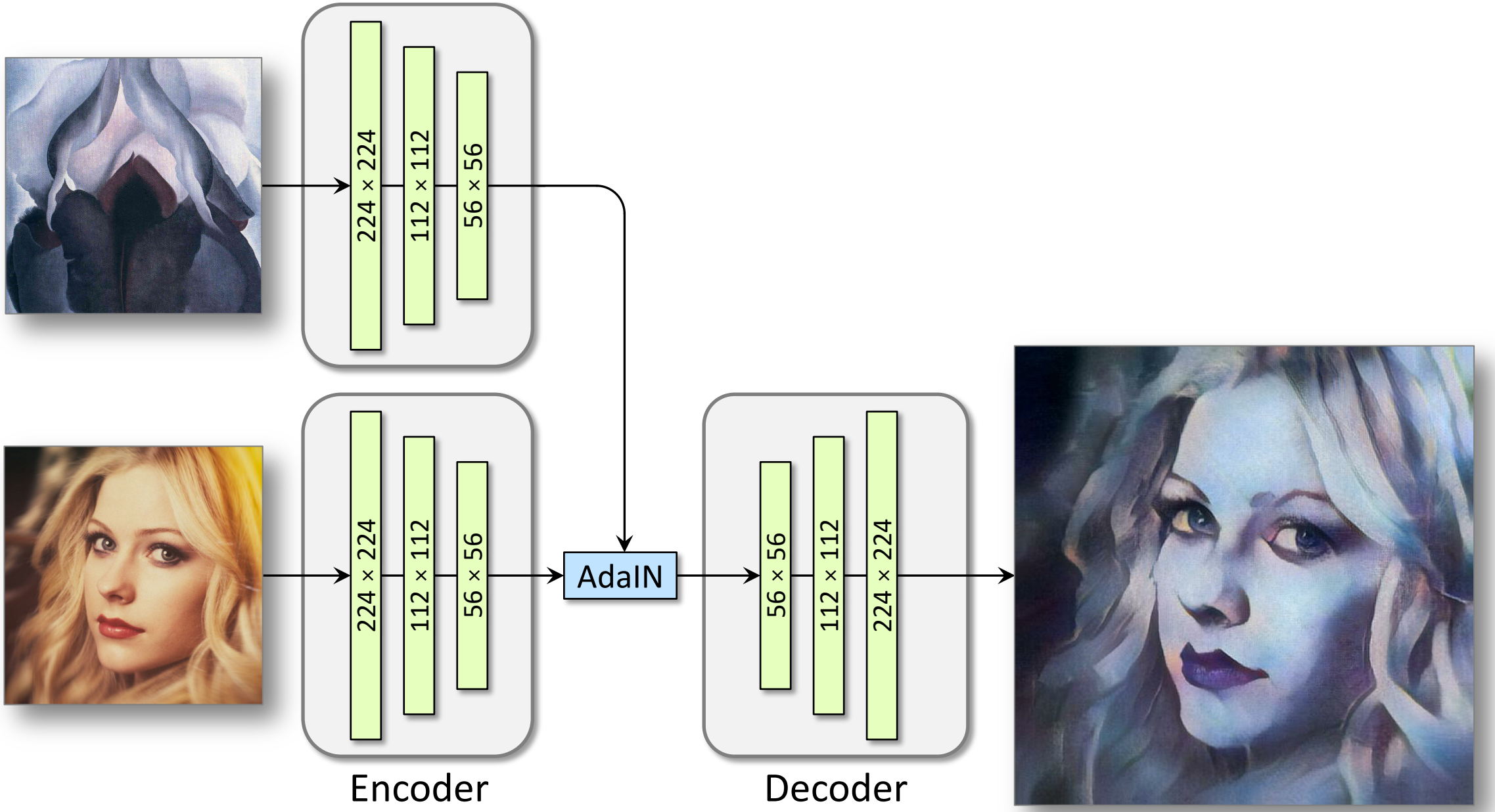


StyleGAN


224 × 224
112 × 112
56 × 56
224 × 224
112 × 112
56 × 56
AdaIN
$\text{AdaIN}(x, y) = \sigma(y)\frac{x - \mu(x)}{\sigma(x)} + \mu(y)$
Encoder

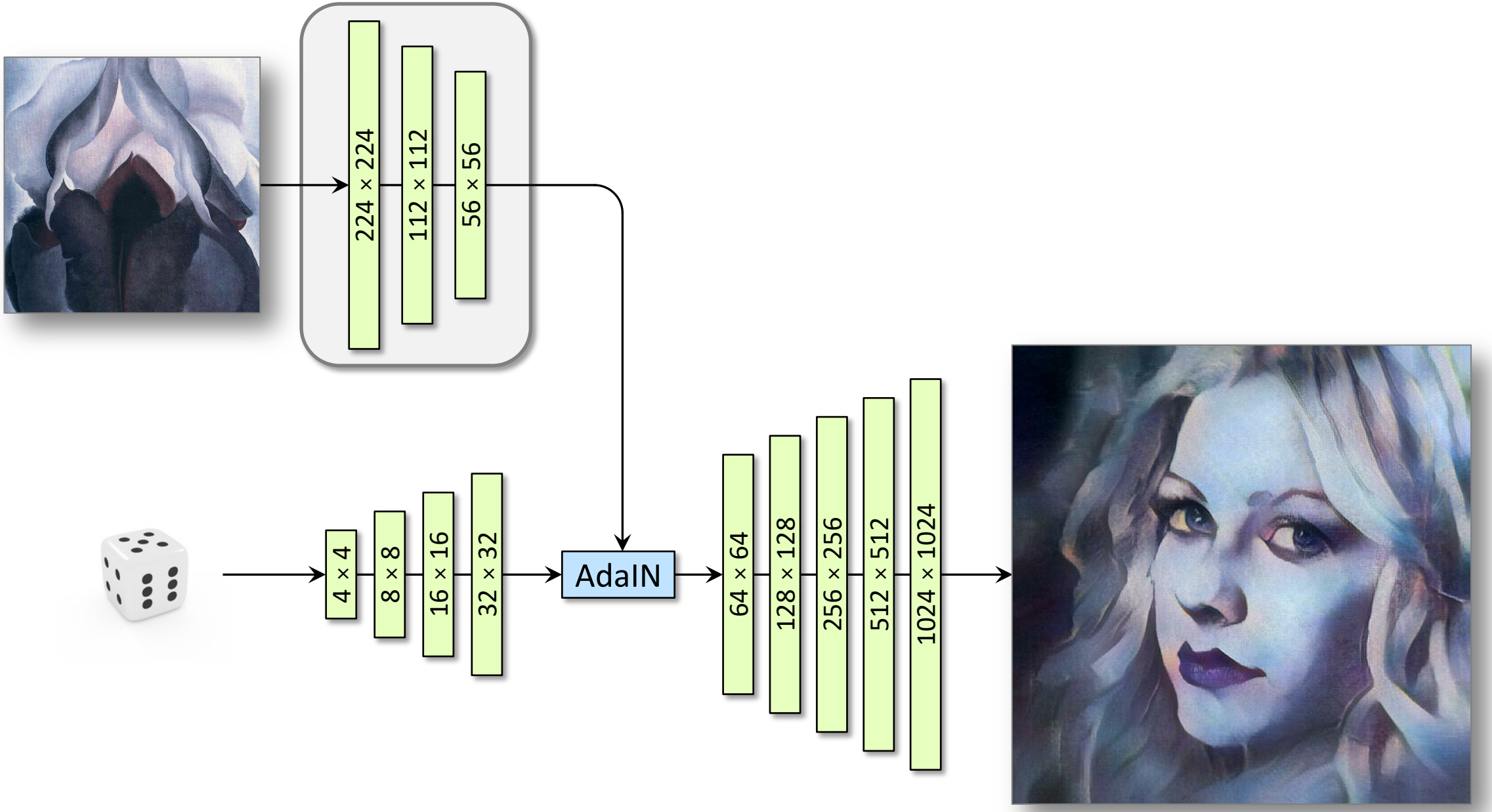
Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN



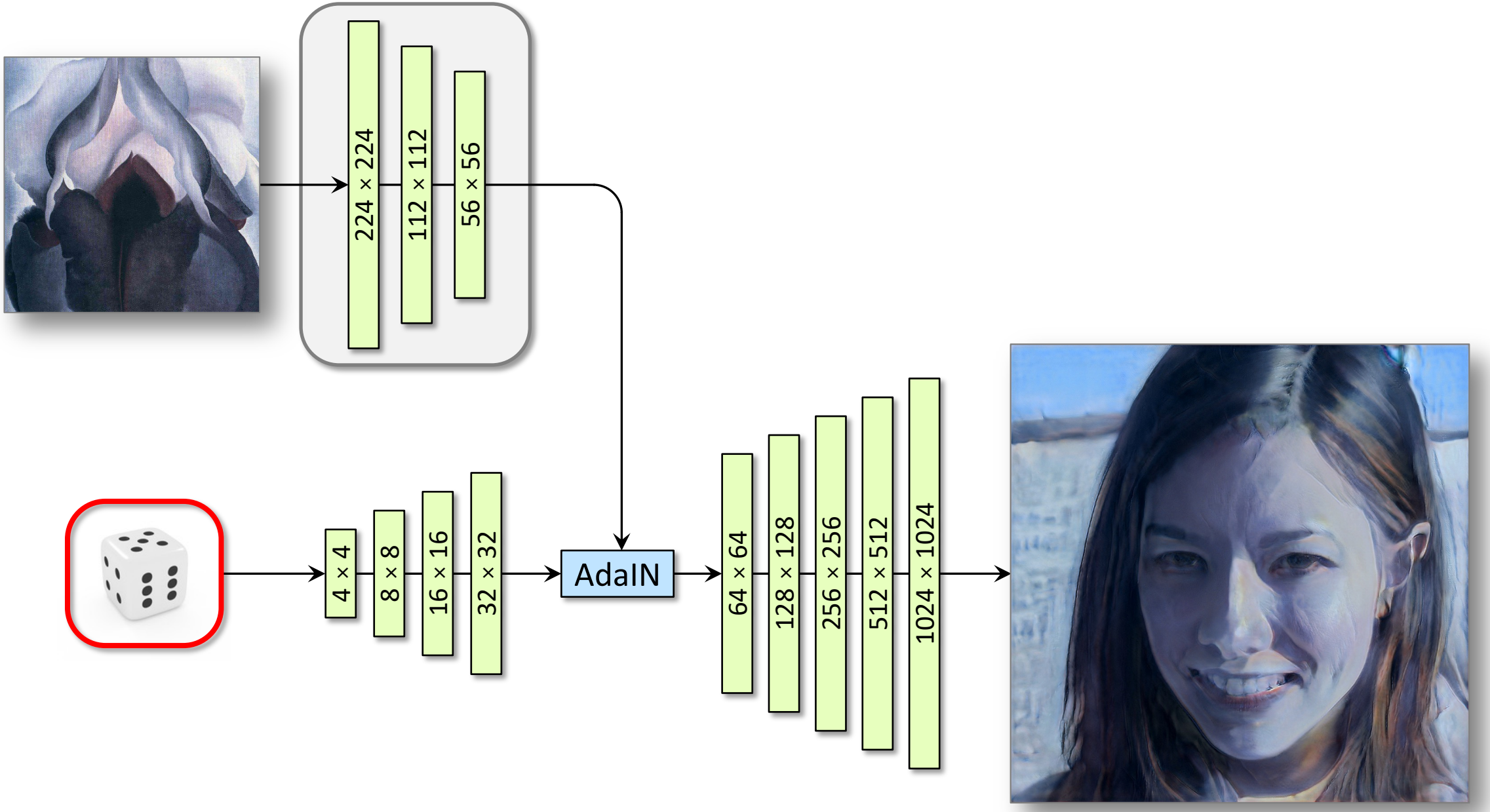
Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN



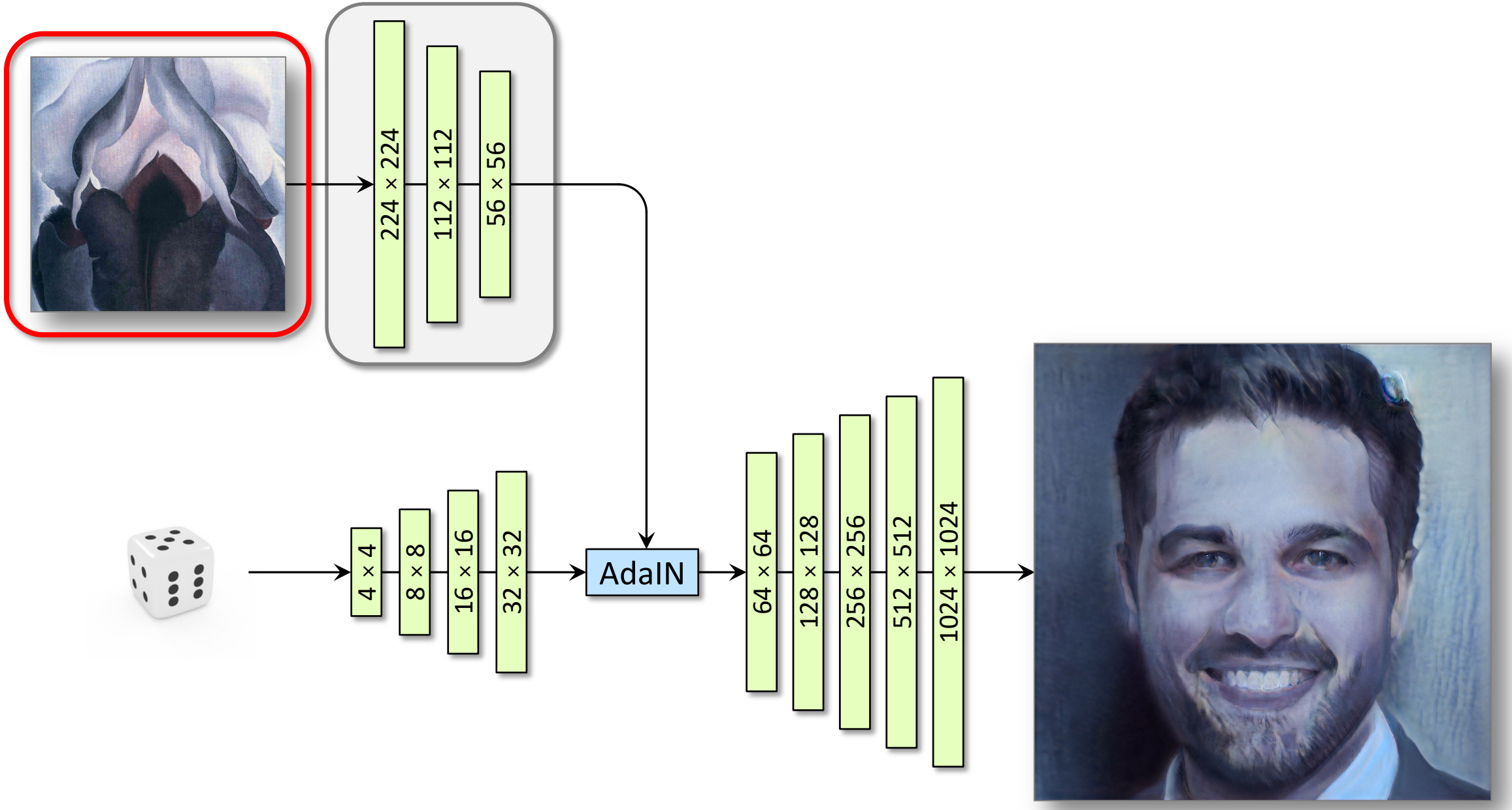
Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN

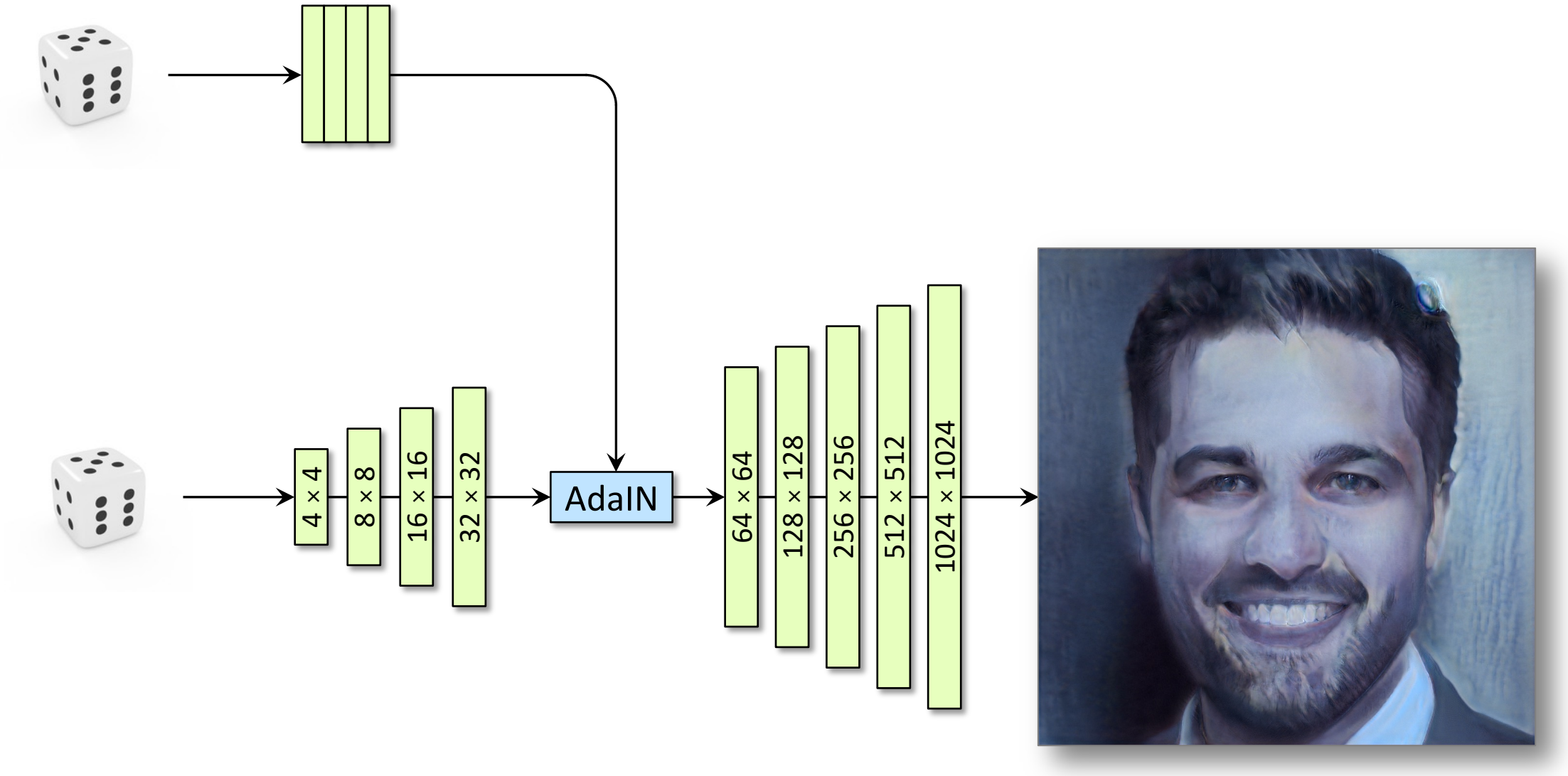


Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN

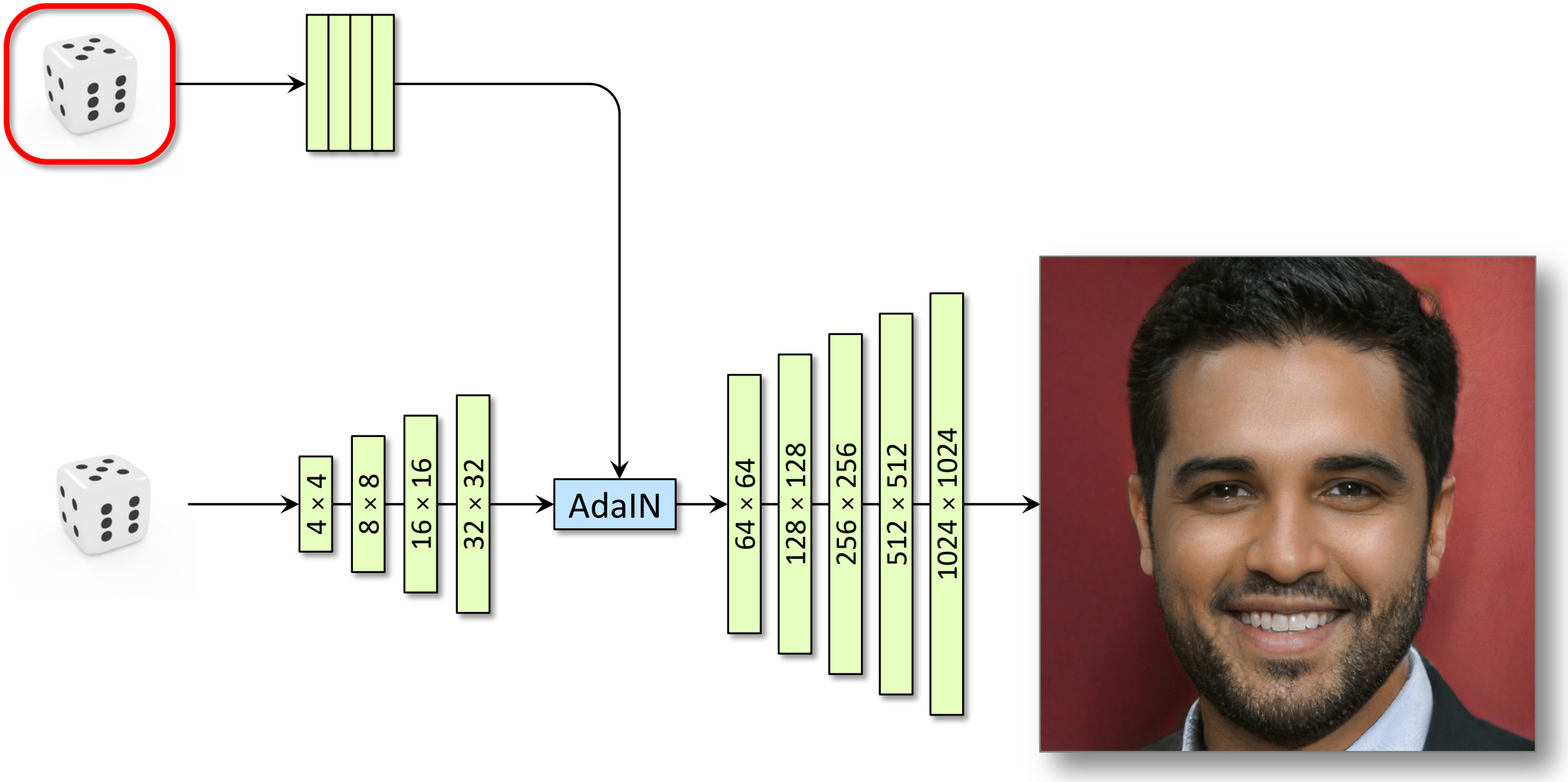


StyleGAN



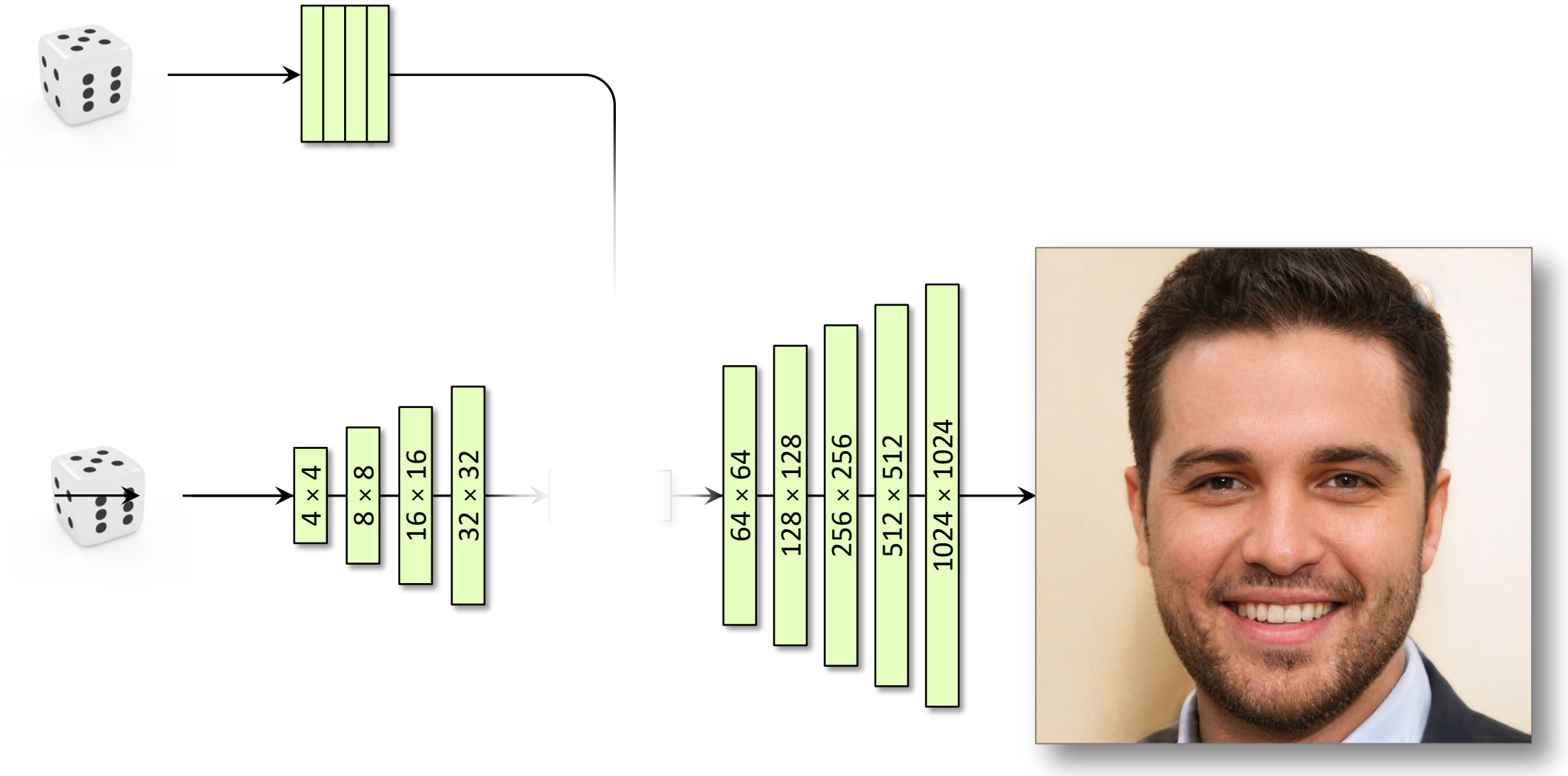
Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN



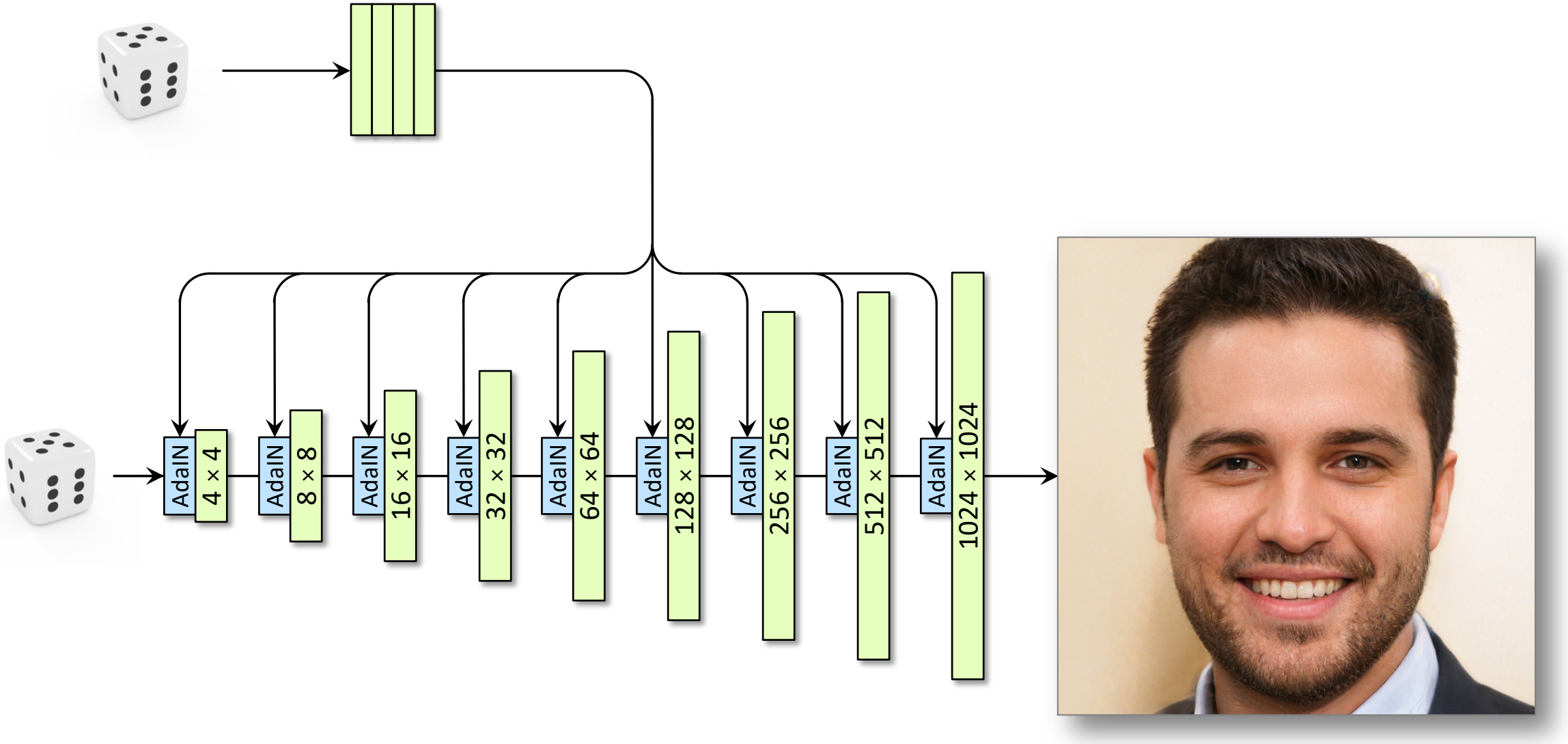
Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN



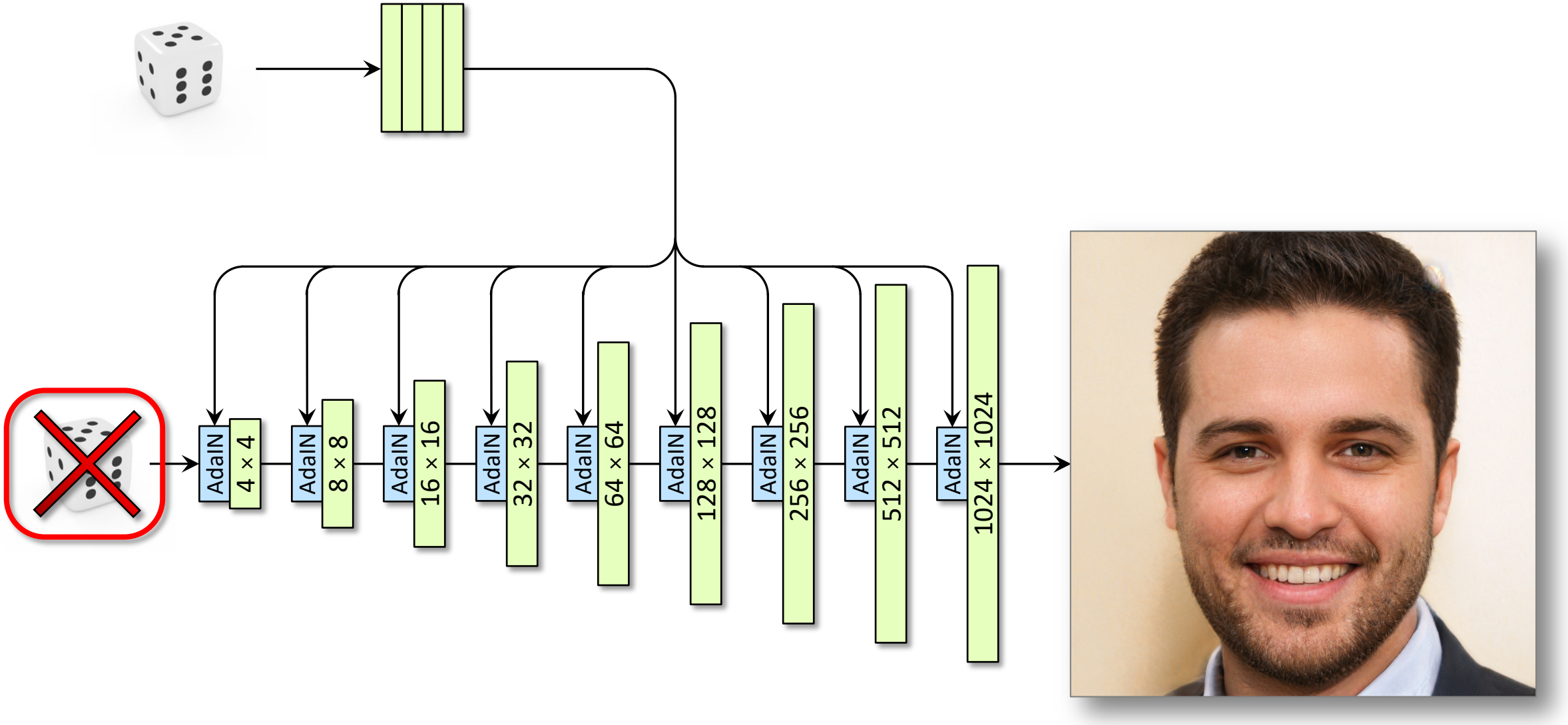
Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN

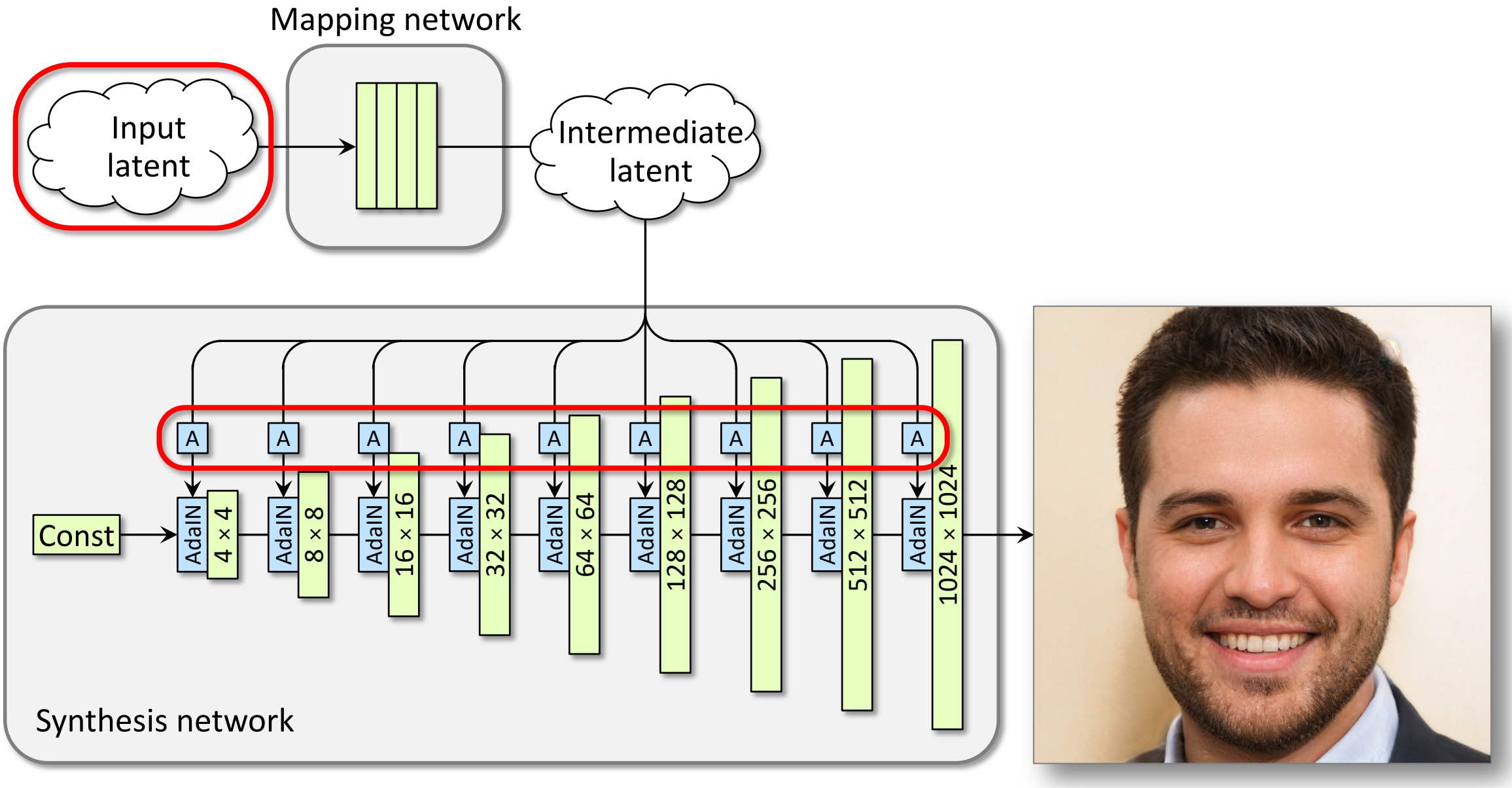


Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN

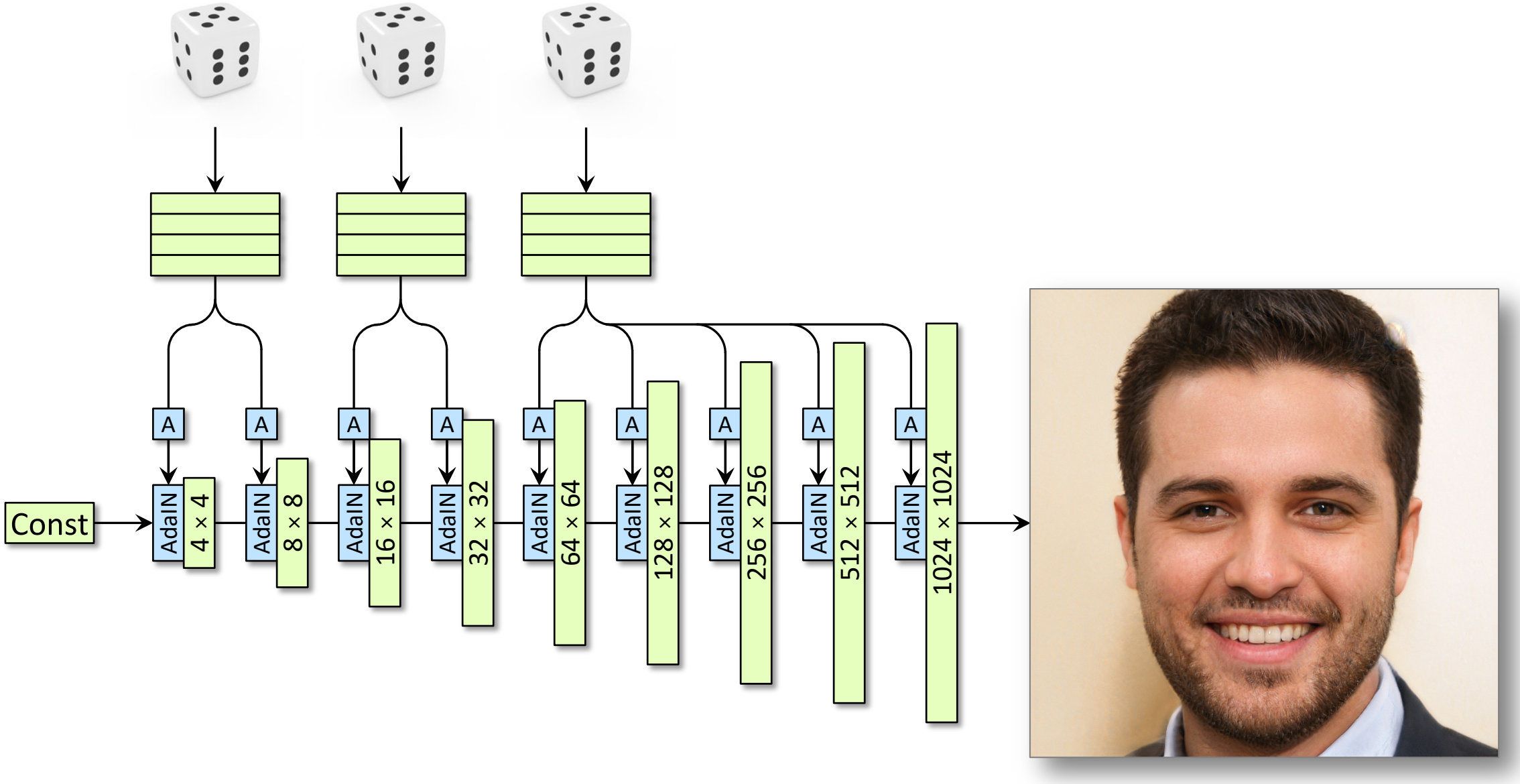


StyleGAN


Mapping network
Input latent
Intermediate latent
A
Const
AdaIN
4 × 4
8 × 8
16 × 16
32 × 32
64 × 64
128 × 128
256 × 256
512 × 512
1024 × 1024
Synthesis network

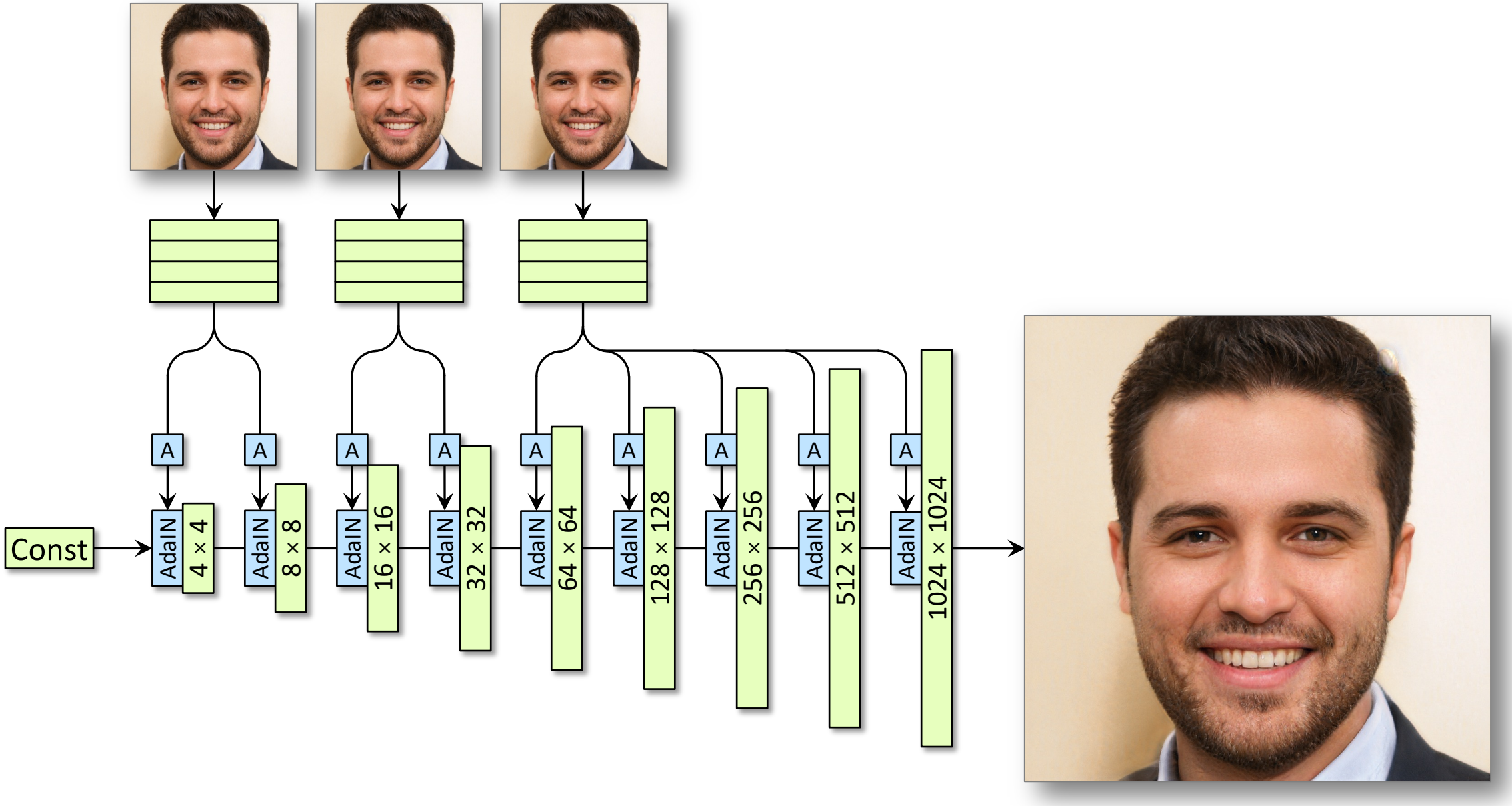
Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN

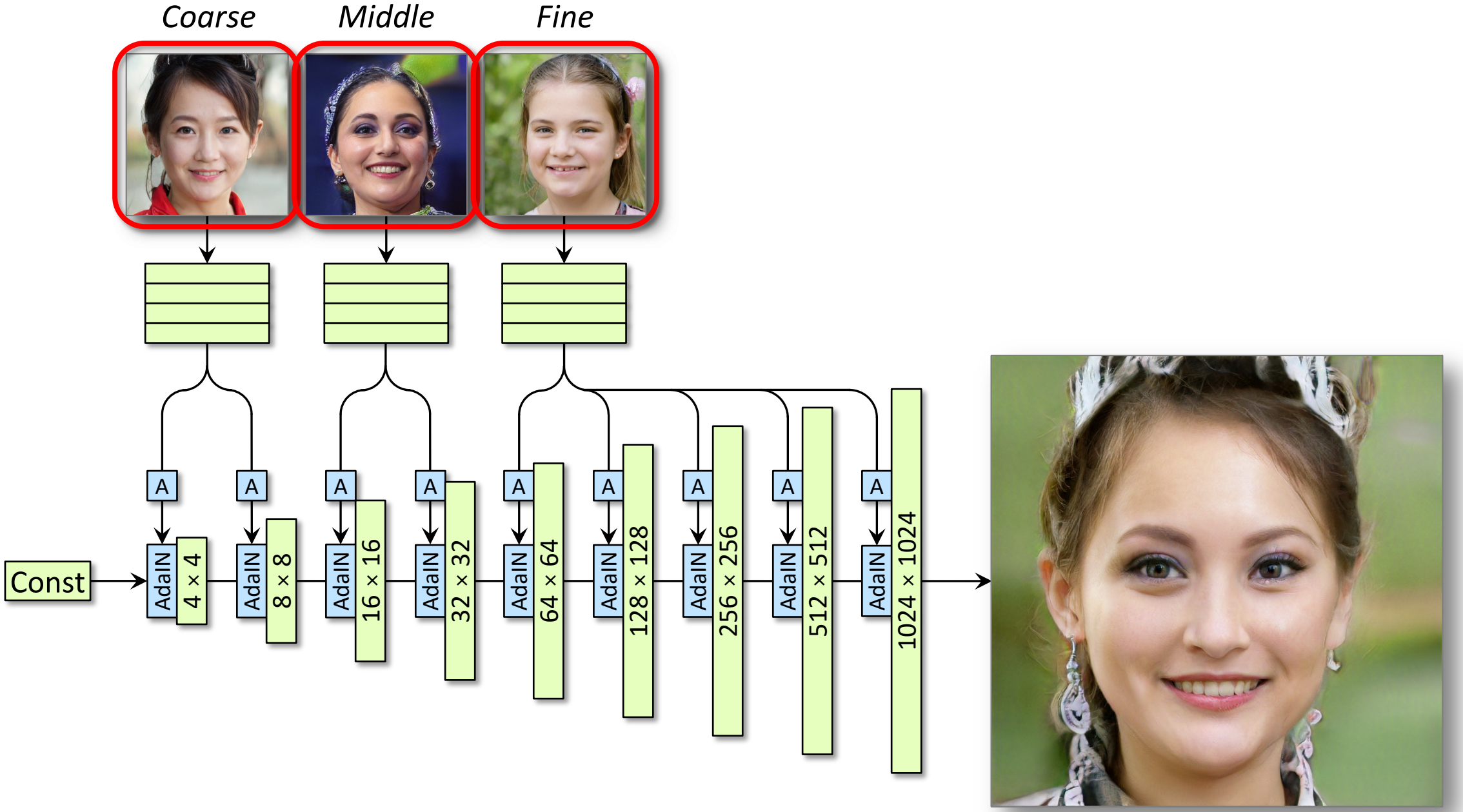


Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

StyleGAN

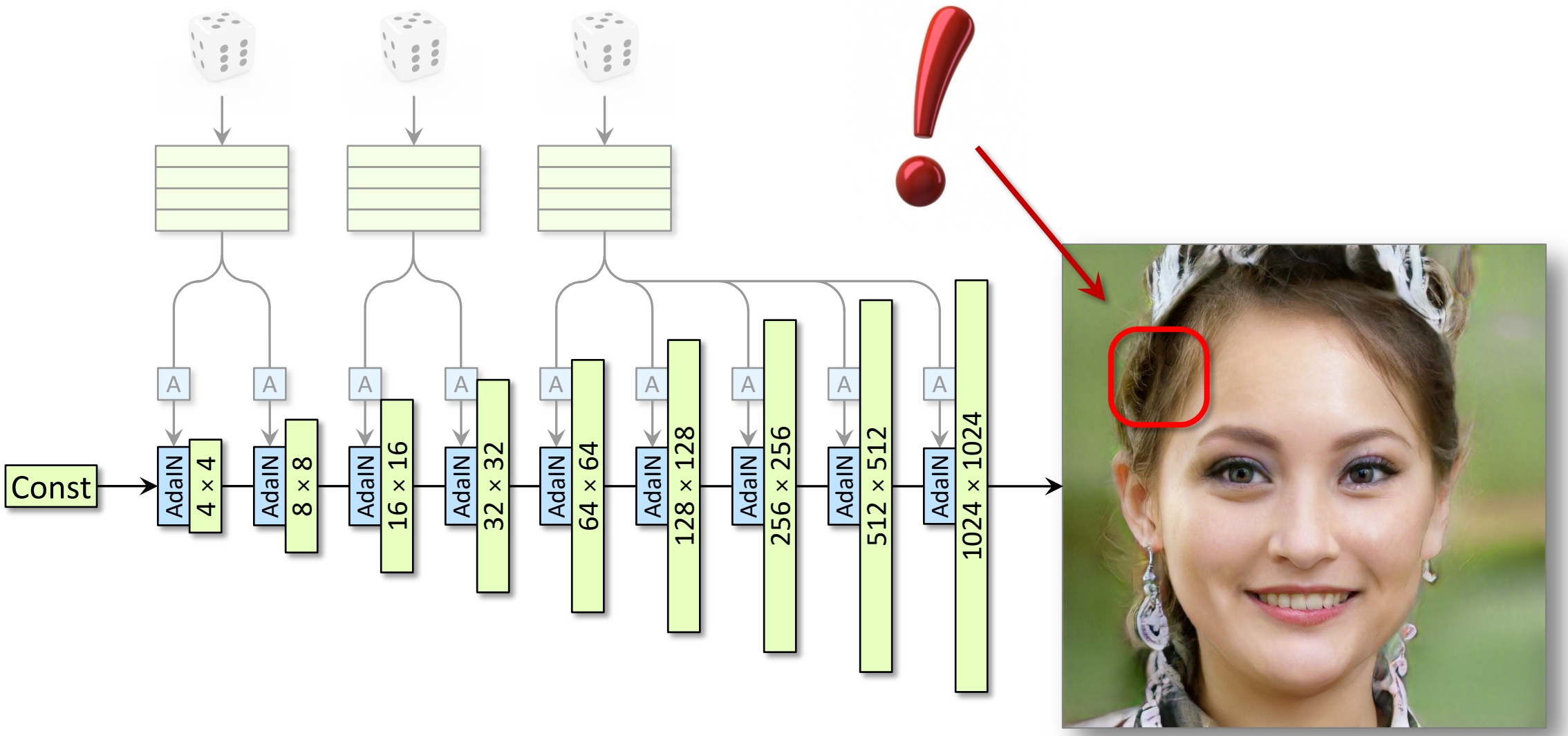


StyleGAN

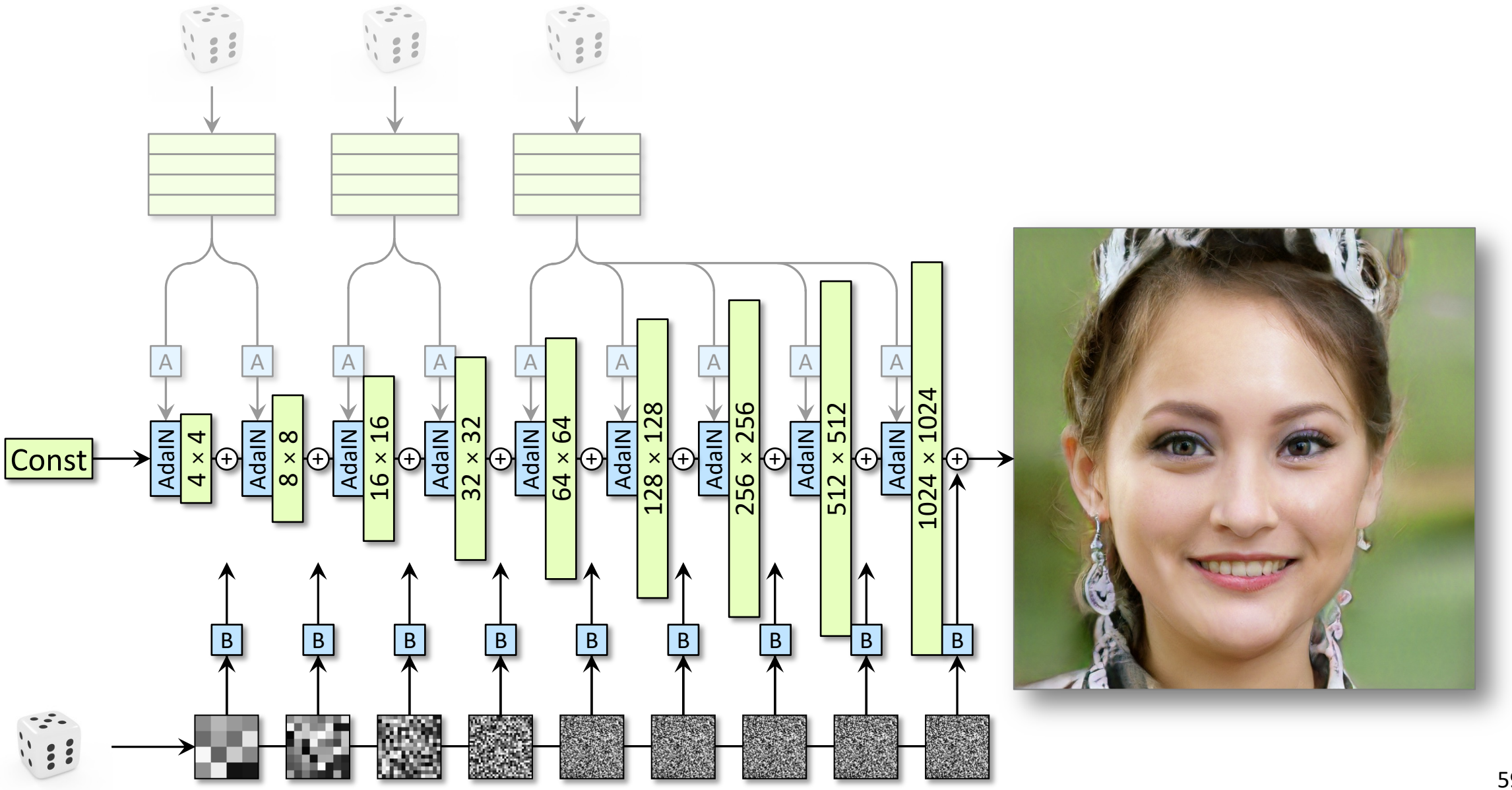

Coarse
Middle
Fine
Const
A
AdaIN
4 × 4
8 × 8
16 × 16
32 × 32
64 × 64
128 × 128
256 × 256
512 × 512
1024 × 1024

Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR, 2019

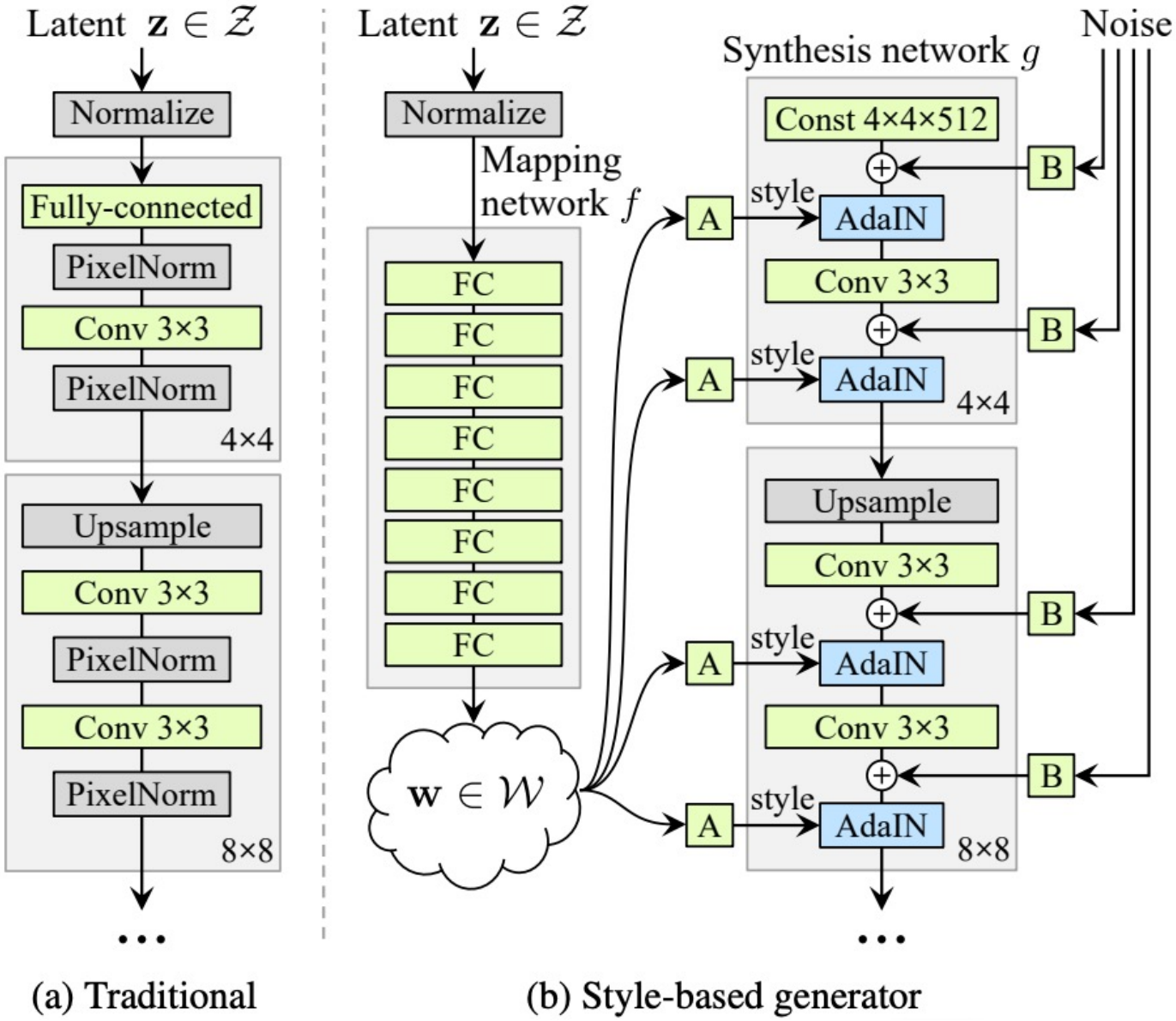
StyleGAN



StyleGAN


Const
A
AdaIN
4 × 4
8 × 8
16 × 16
32 × 32
64 × 64
128 × 128
256 × 256
512 × 512
1024 × 1024
B

StyleGAN



(a) Traditional (b) Style-based generator

StyleGAN- Experiments

CelebA-HQ

30,000 images

Limited variation

Image quality not always ideal

Flickr-Faces-HQ

70,000 images

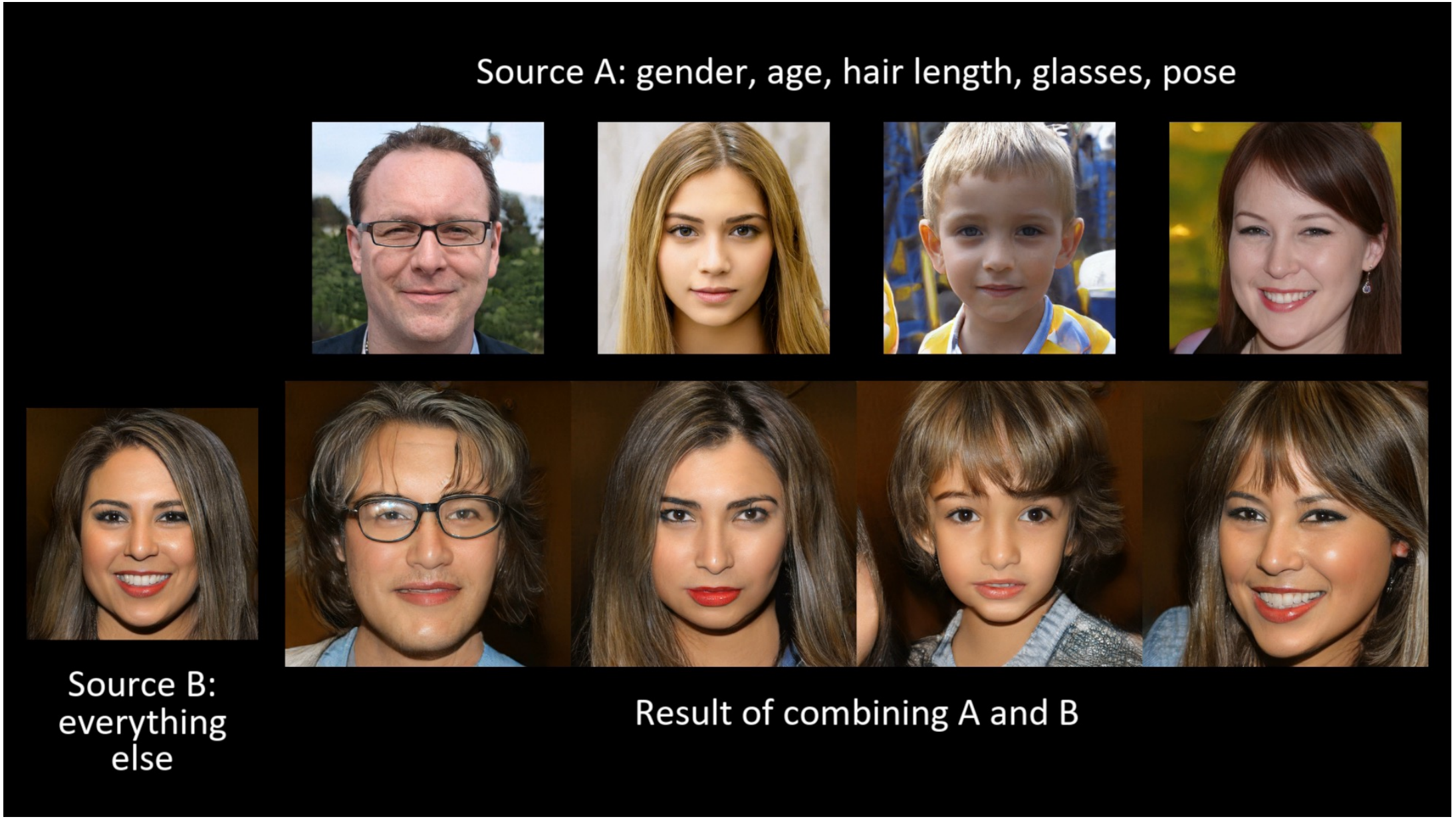
Varied age, ethnicity, background, …

Consistent high quality

StyleGAN- Experiments

Coarse styles
$(4^2 – 8^2)$

StyleGAN- Experiments



StyleGAN- Experiments

Coarse styles
$(4^2 – 8^2)$

Middle styles
$(16^2 – 32^2)$

Fine styles
$(64^2 – 512^2)$

StyleGAN- Experiments

Cars

Thank you!
Q & A