A Style-Based Generator Architecture for Generative Adversarial Networks

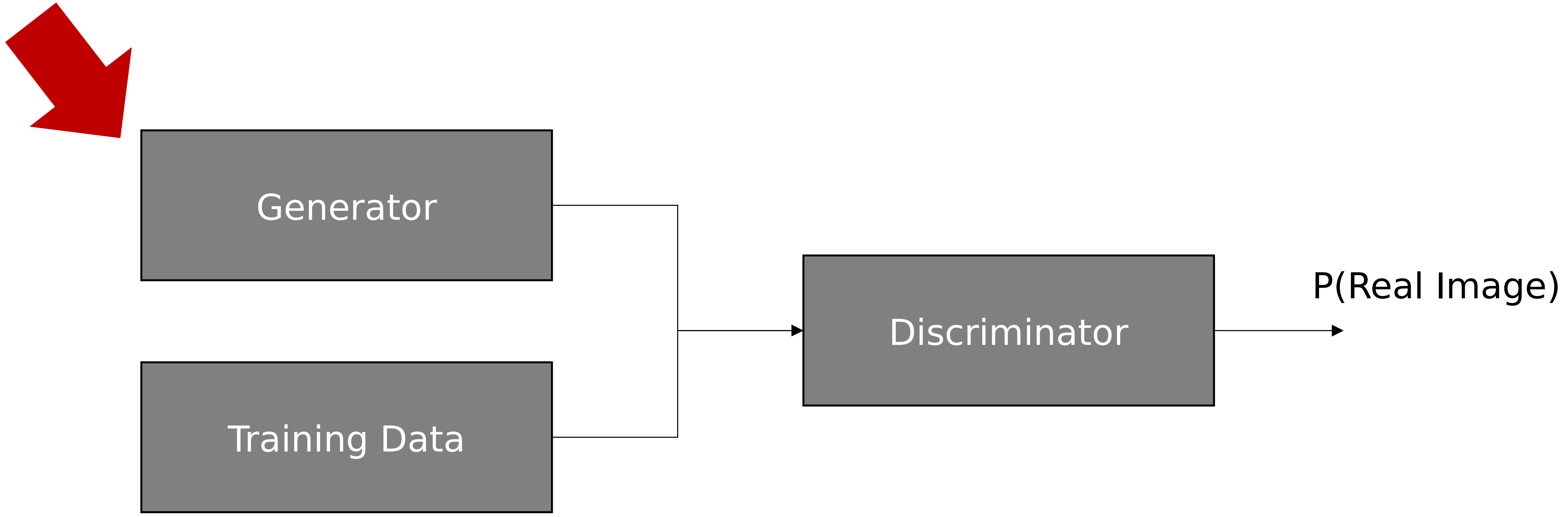
Tero Karras, Samuli Laine, Timo Aila

NVIDIA

Presenter: Diego Cantor, PhD

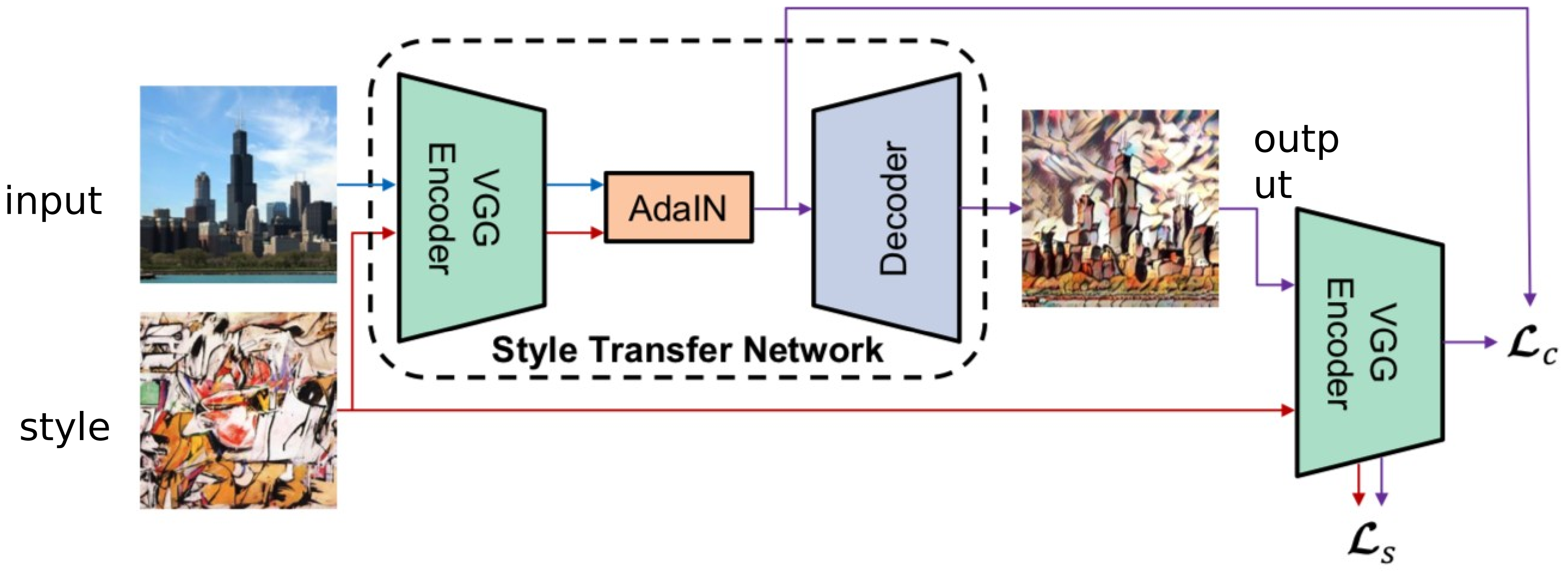
Facilitators: Michael Vertolli and David McDonald

Despite improvement in image quality synthesis, GAN generators operate as black boxes



Understanding of image synthesis is poor

This work proposes a model for the generator that is inspired by **style transfer networks**

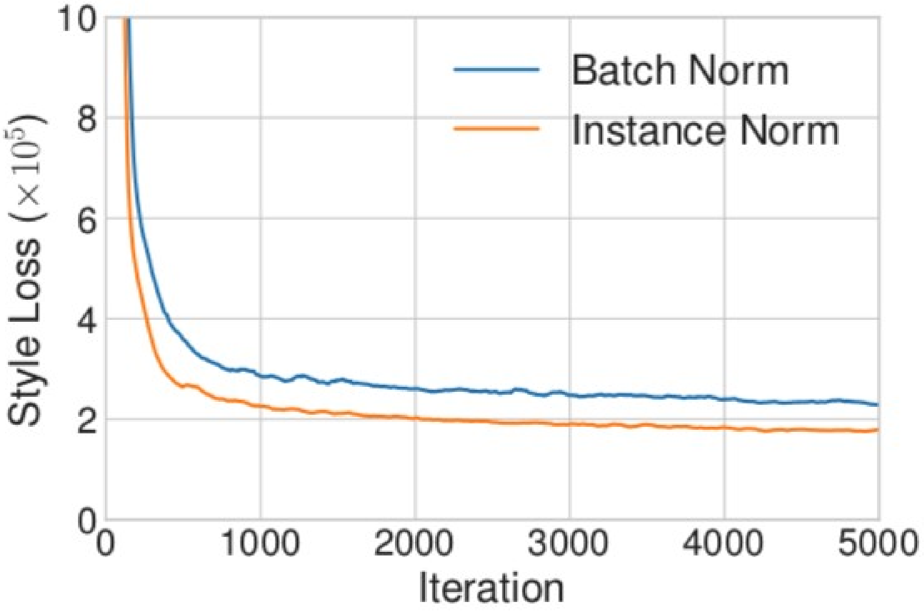


Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, Huang and Belongie, 2017

Everything started with the usage of batch normalization to improve training

$$\mathrm{BN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

$$\mathrm{IN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$



Gamma and Beta are learned from data

Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, Huang and Belongi

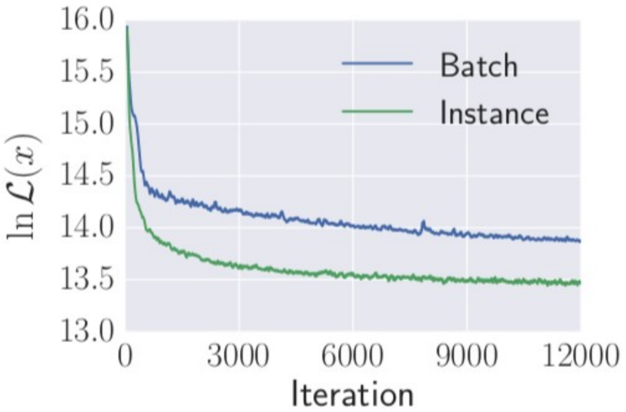
Instance normalization improves style-transfer loss when compared to other approaches

Content

Style

StyleNet BN

StyleNet IN (ours)



Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, Ulyanov and Vedaldi, CVPR, 2017

Adaptive Instance Normalization simply scales the normalized input with style spatial statistics. This has profound implications.

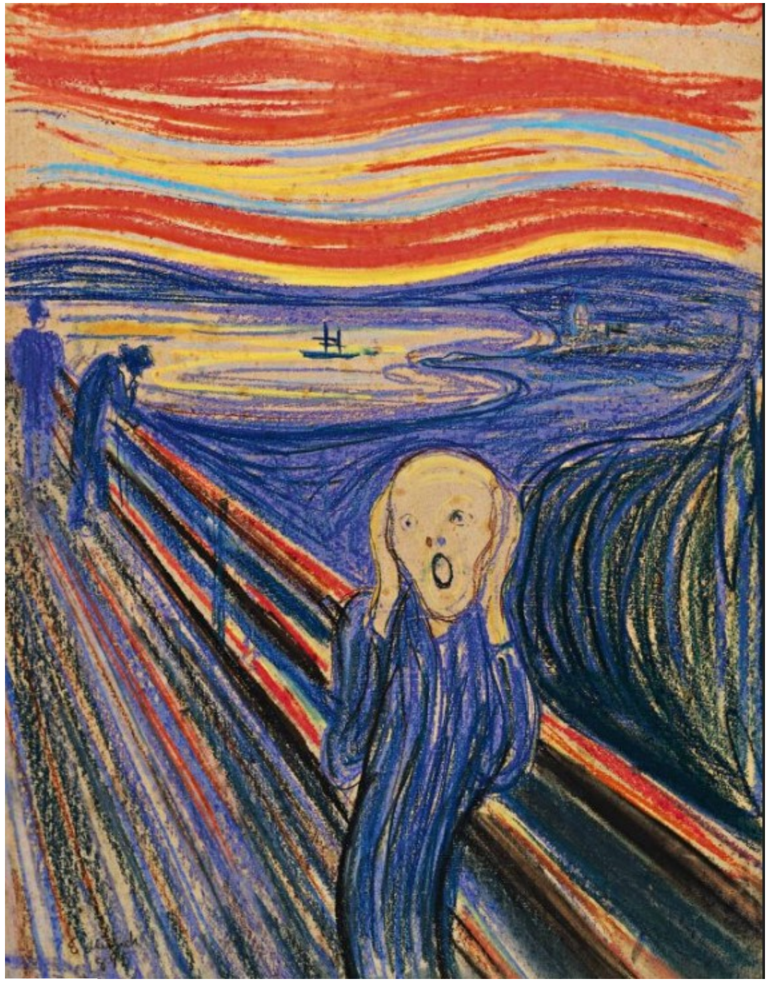
input style

$$\mathrm{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$

Style statistics are not learnable. So AdaIN has no learnable parameters

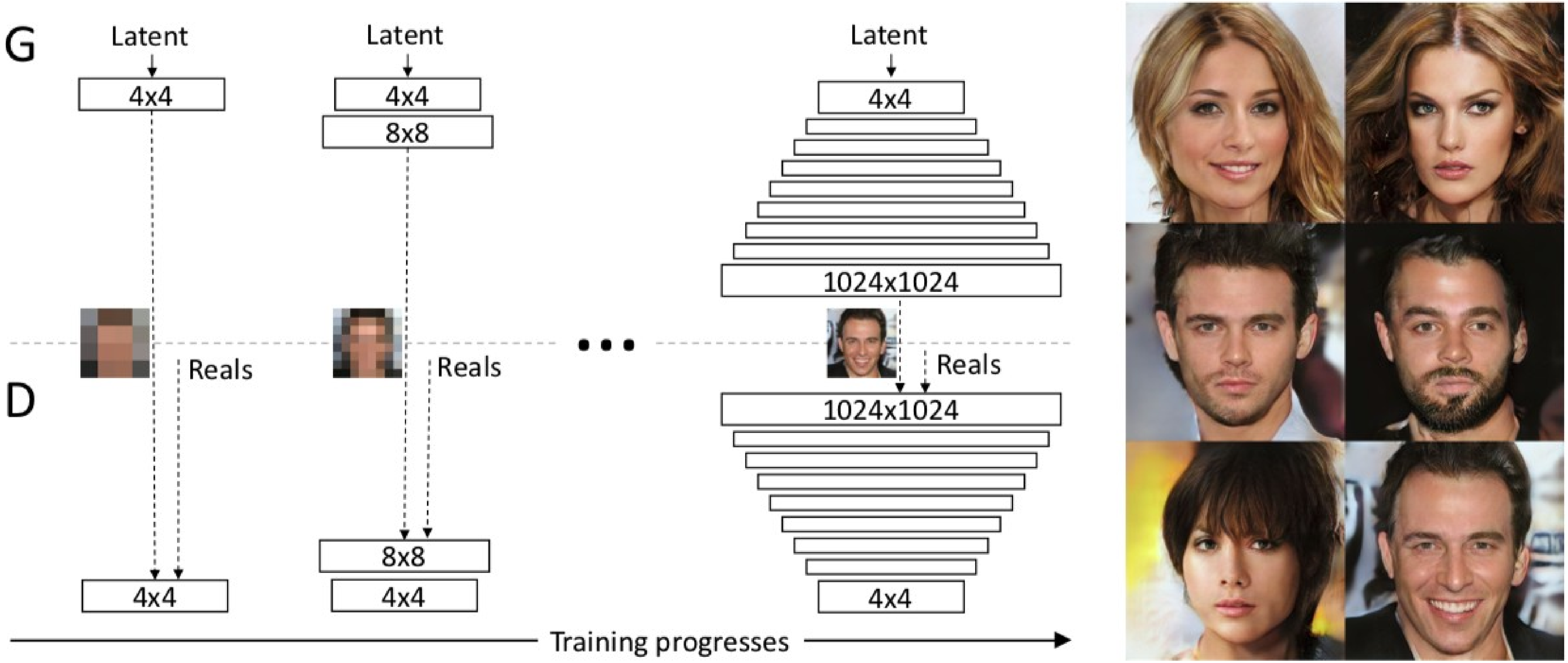
$$\mathrm{BN}(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$$

$$\mathrm{IN}(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$$



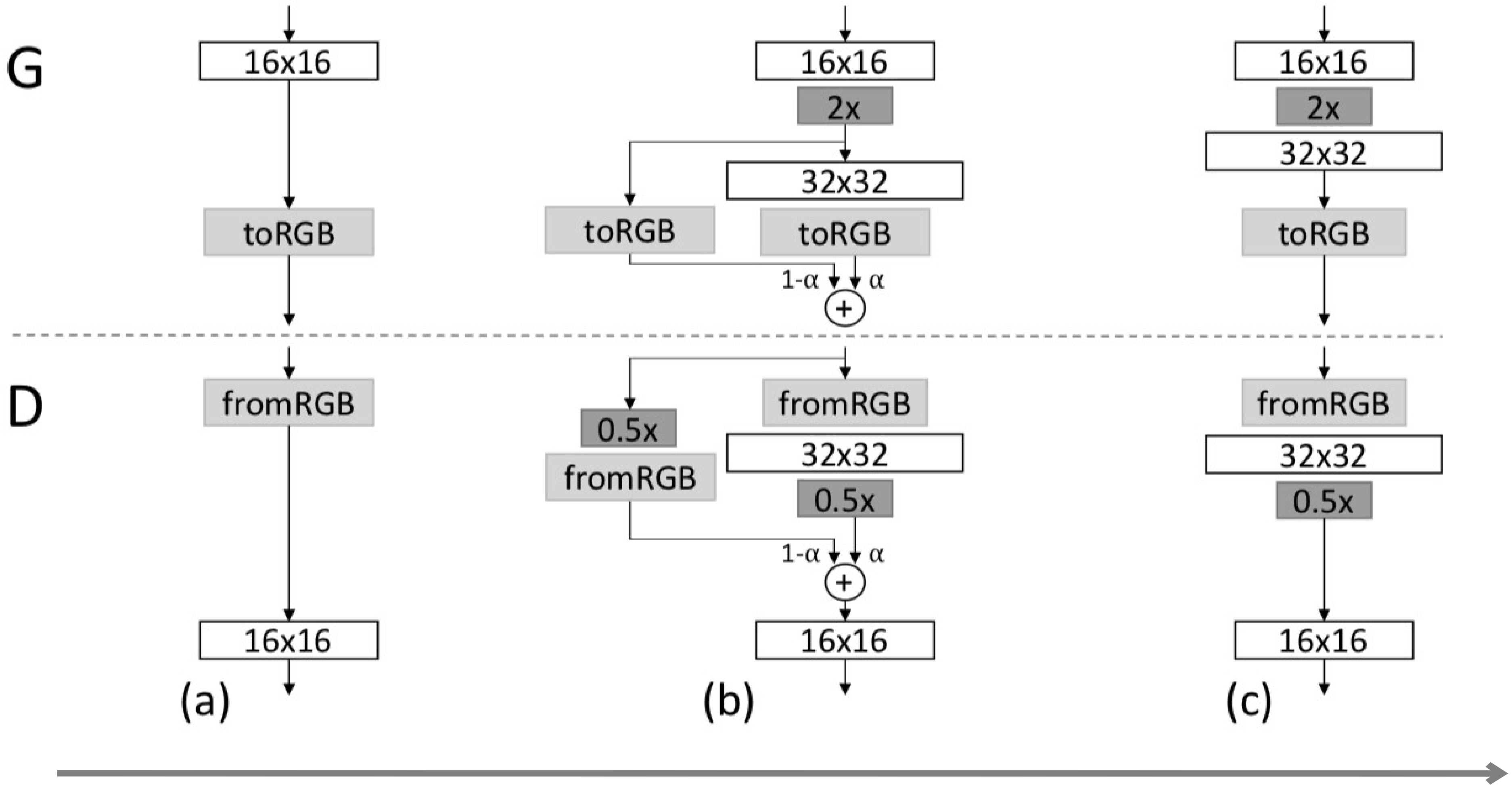
$$\mathrm{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

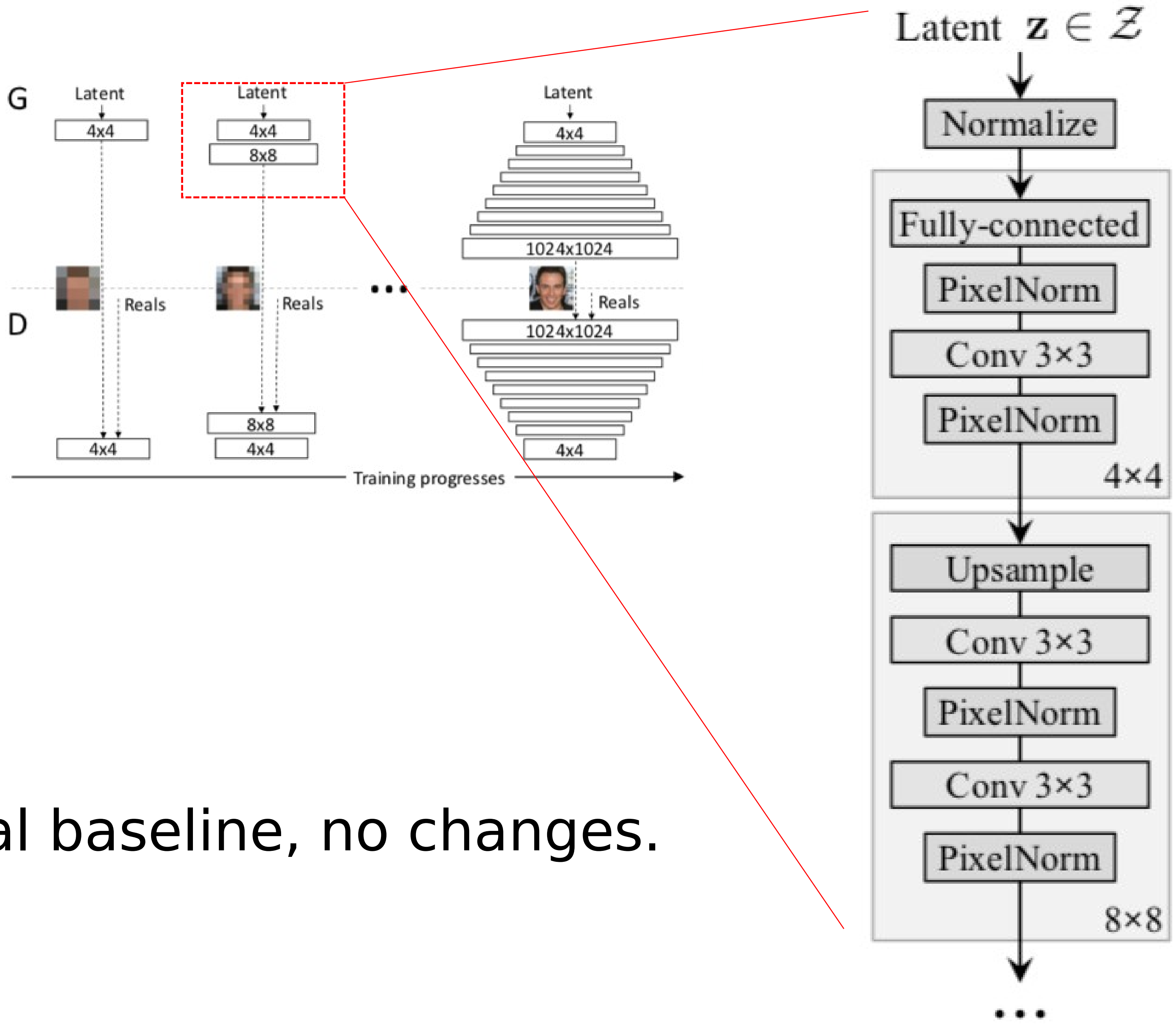
The baseline configuration is the progressive GAN setup (same research group at NVIDIA)



Progressive growing of GANs for improved quality, stability and variation, Karras et al., ICLR 2018

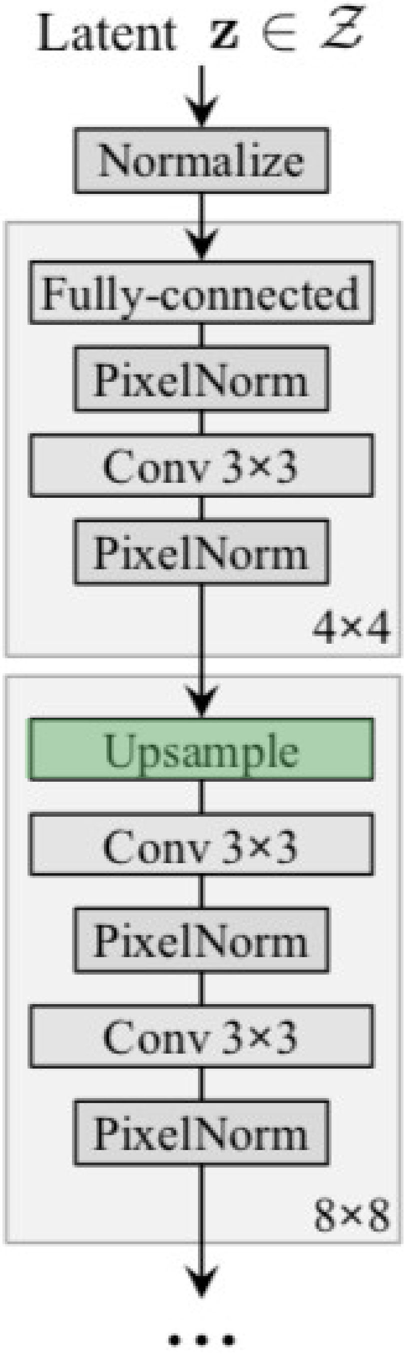
Smooth transition into higher-res layers using bilinear interpolation





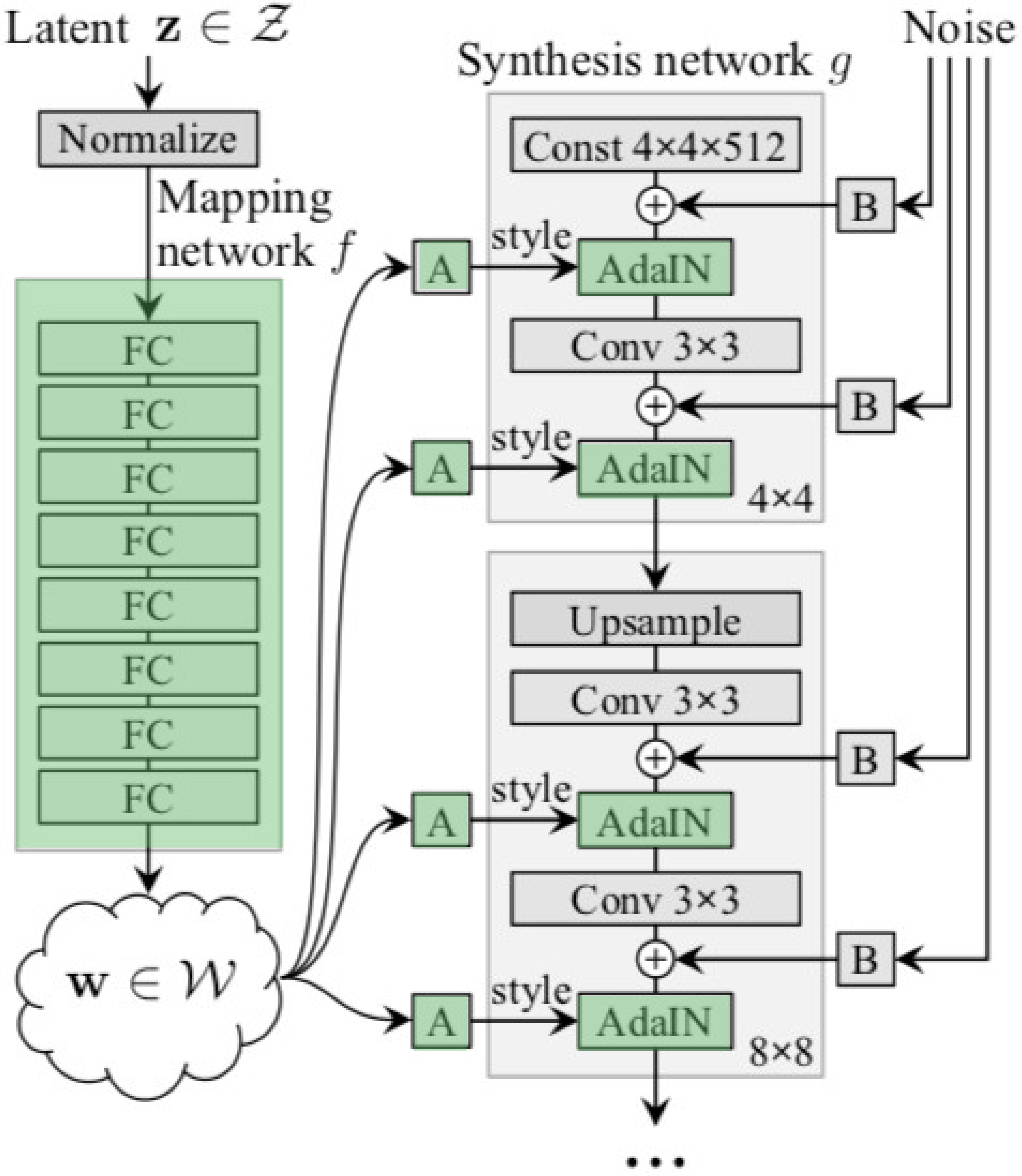
Step A

Original baseline, no changes.



Step B

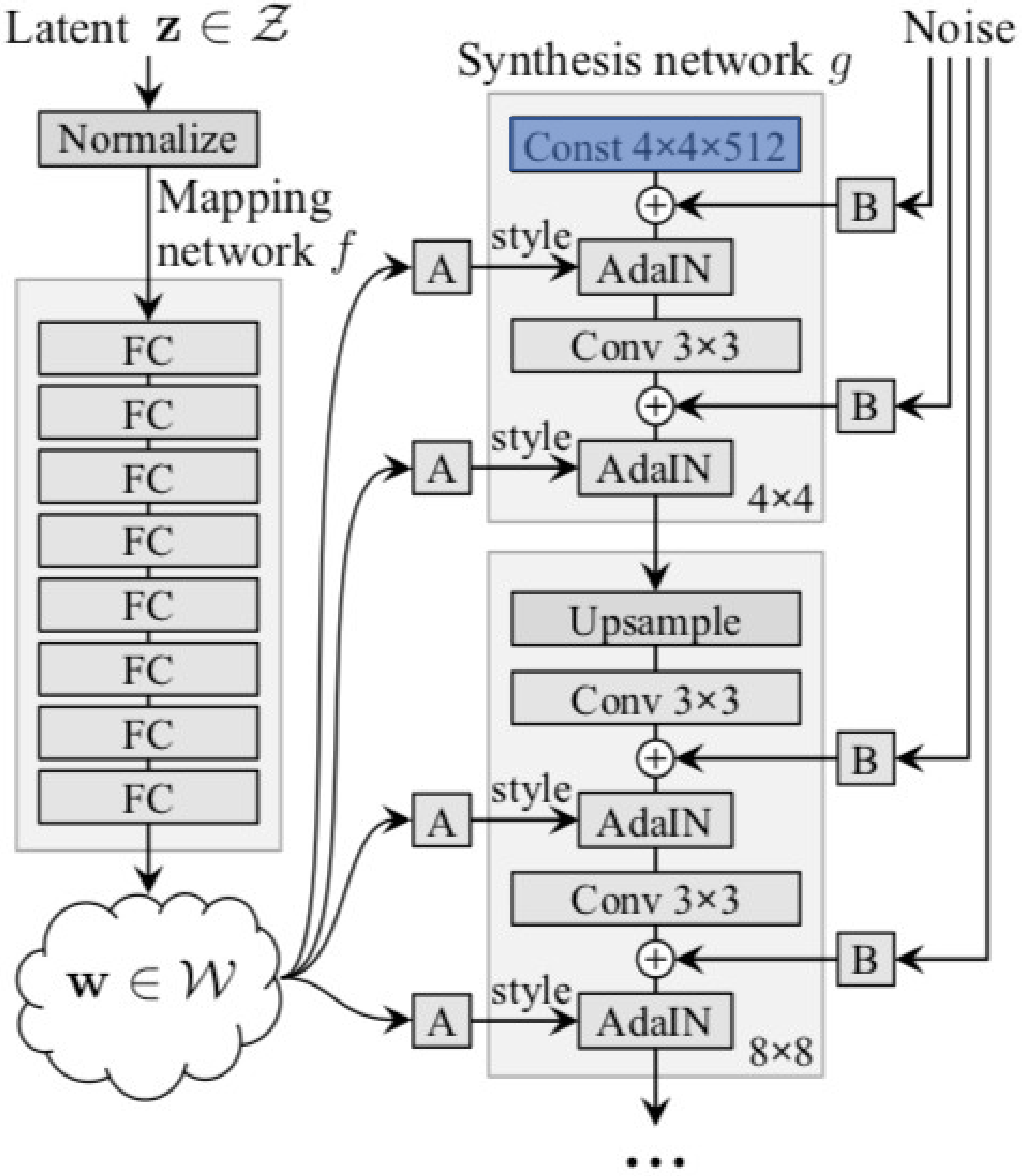
- Replace nearest neighbor with bilinear upsampling
- Replace pooling with bilinear downsampling (in the discriminator)



Step C

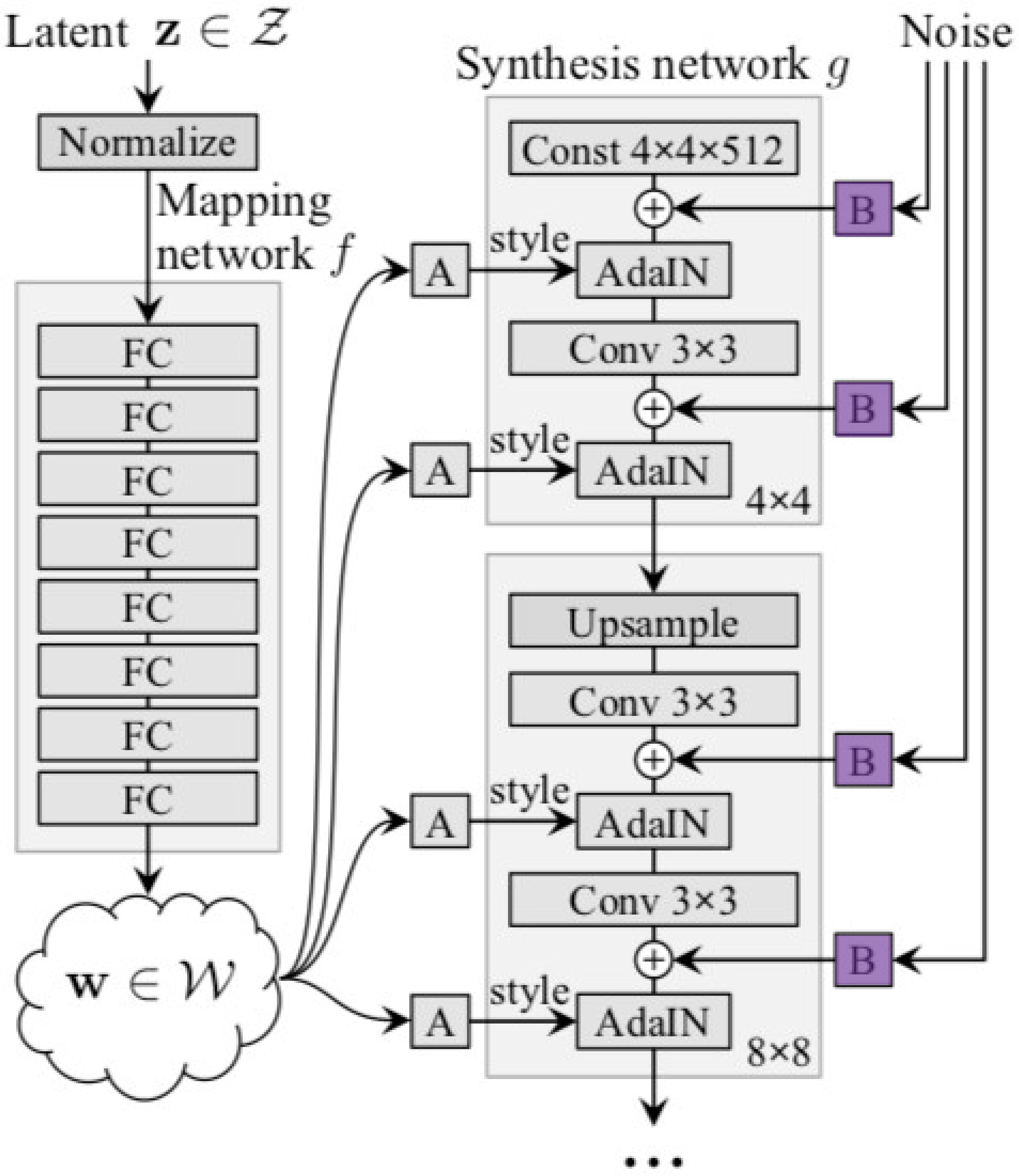
Add mapping network and styles.

Styles are generated from W and used in AdaIN operations



Step D

remove traditional input

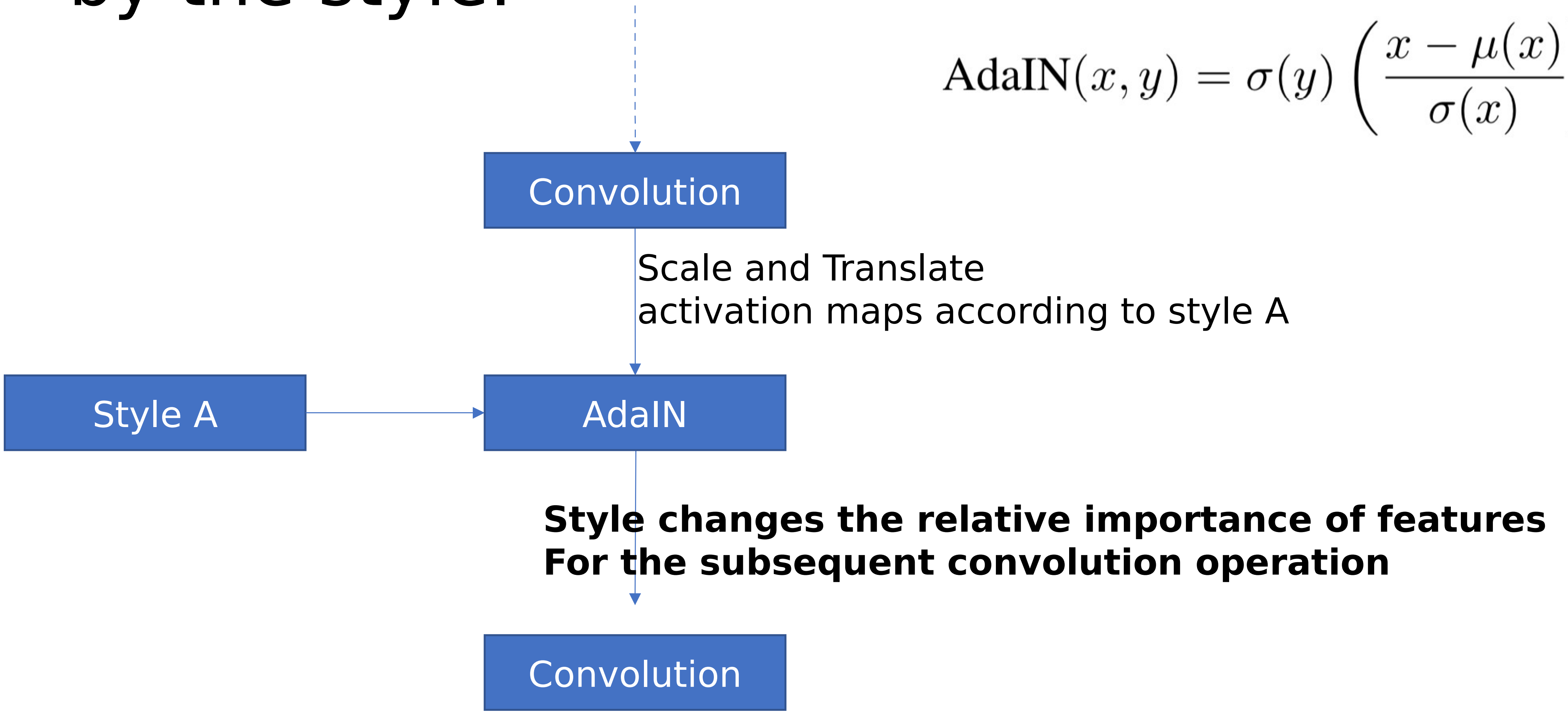


Step E

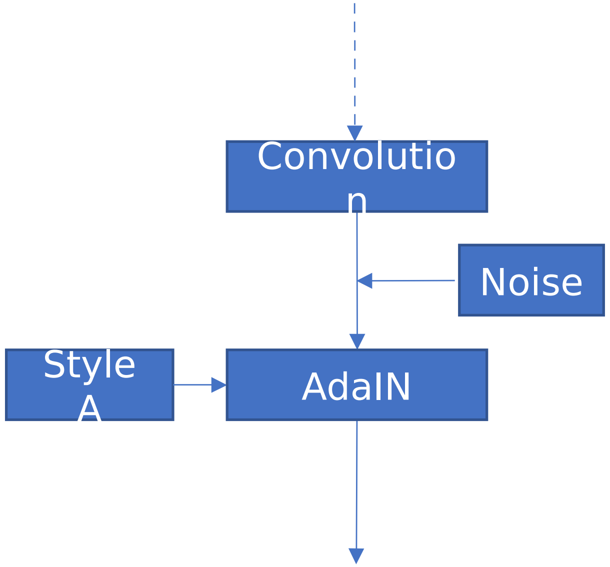
Add noise inputs (enables generating stochastic detail)

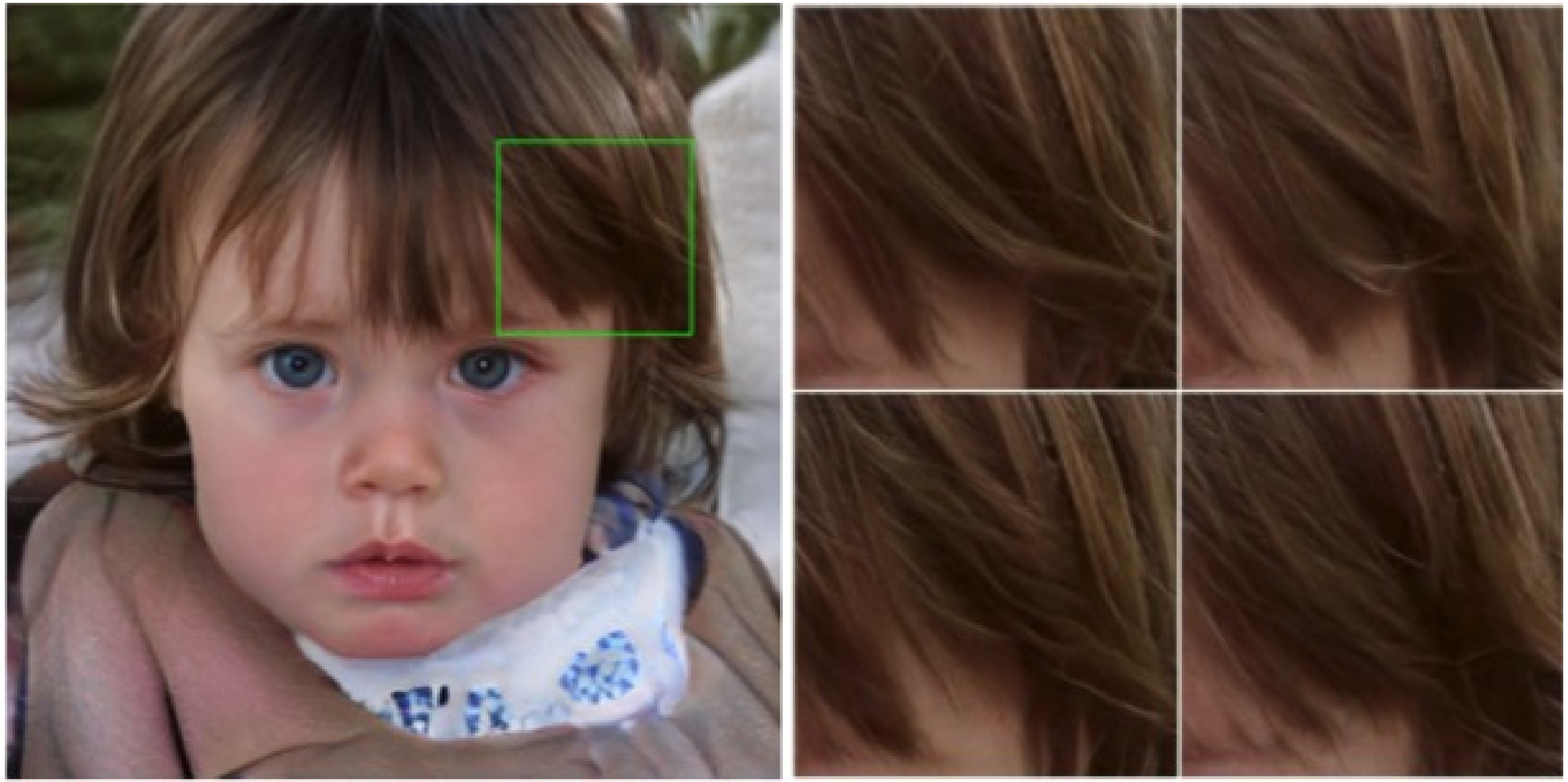
This is the key: AdaIN operation **affects the relative importance of features at every scale**. How much? This is determined by the style.

$$\mathrm{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$



Style affects the entire image but noise is added per pixel. The network learns to use it to control **stochastic variation.**

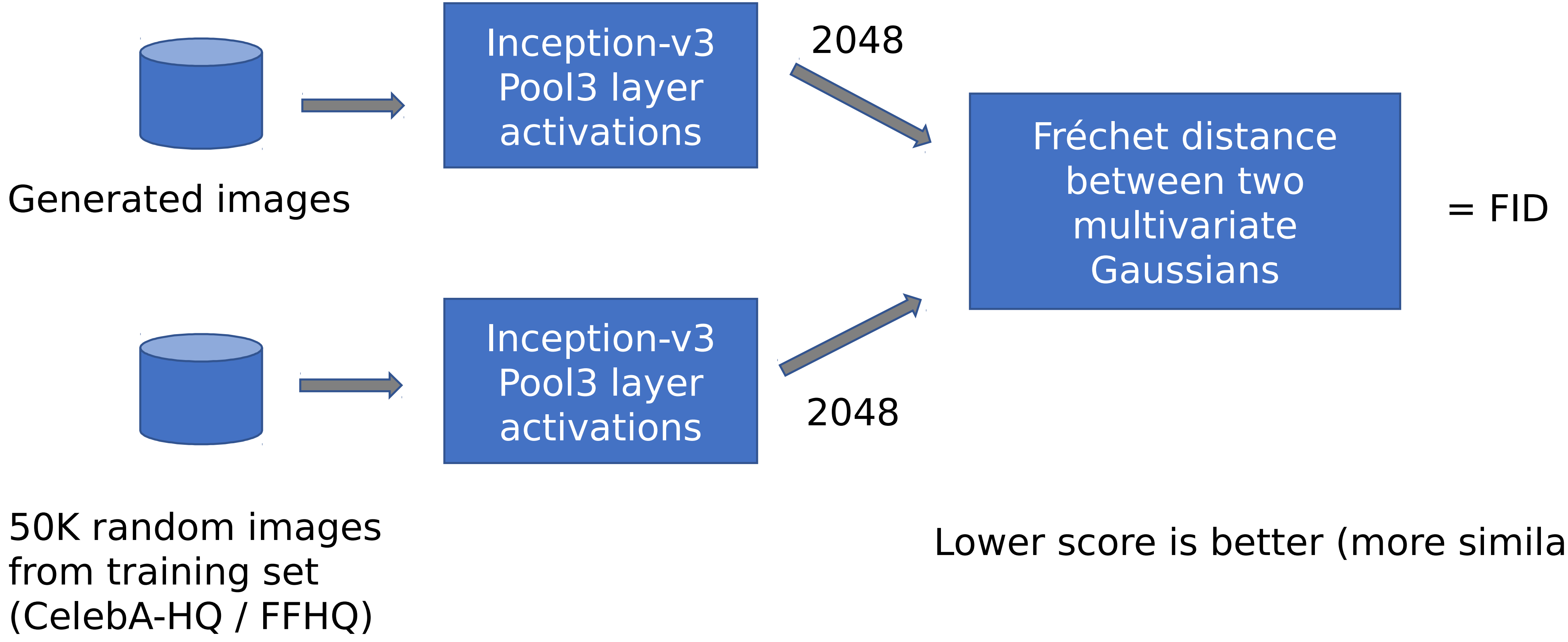




2 min break

Results

This group used the **Fréchet inception distance** (FID) to measure the quality of generated images

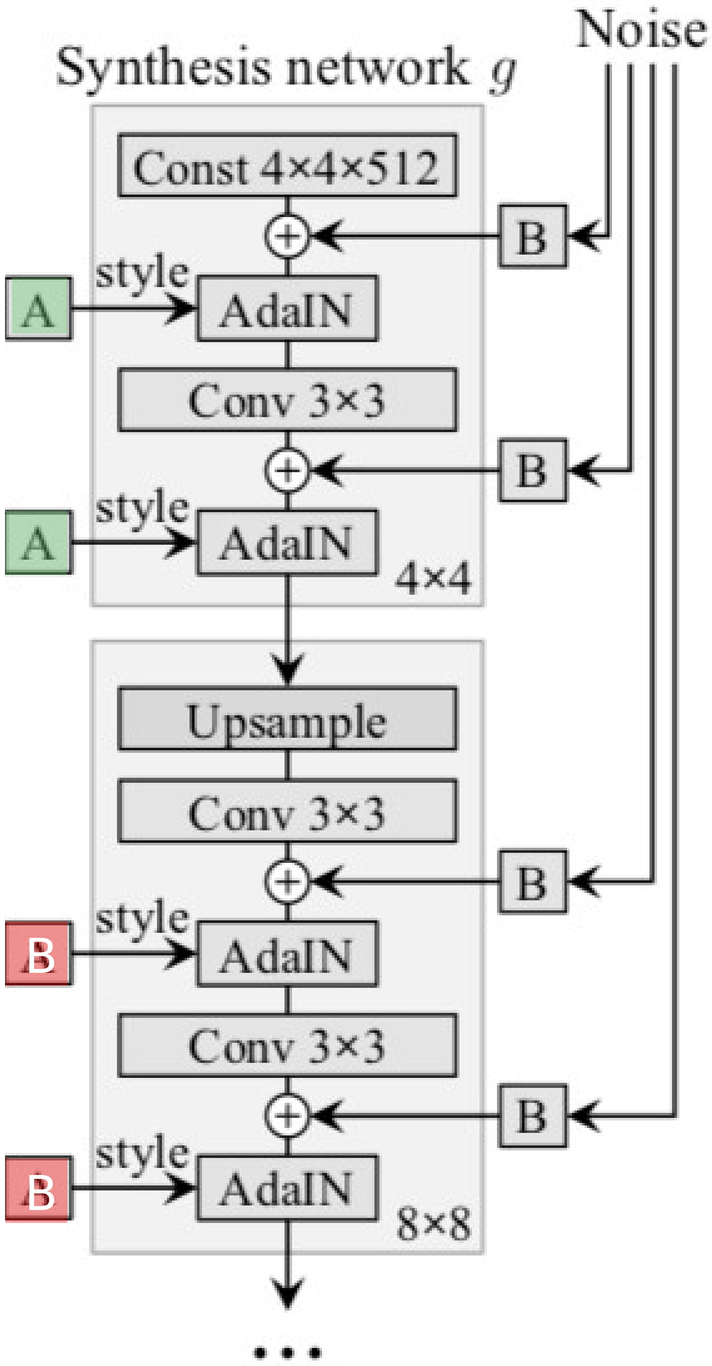


Lower score is better (more similar)

Results: quality of the generated images. Lower FID is better

Method	CelebA-HQ	FFHQ
A Baseline Progressive GAN	7.79	8.04
B + Tuning (incl. bilinear up/down)	6.11	5.25
C + Add mapping and styles	5.34	4.85
D + Remove traditional input	5.07	4.88
E + Add noise inputs	**5.06**	4.42
F + Mixing regularization	5.17	**4.40**

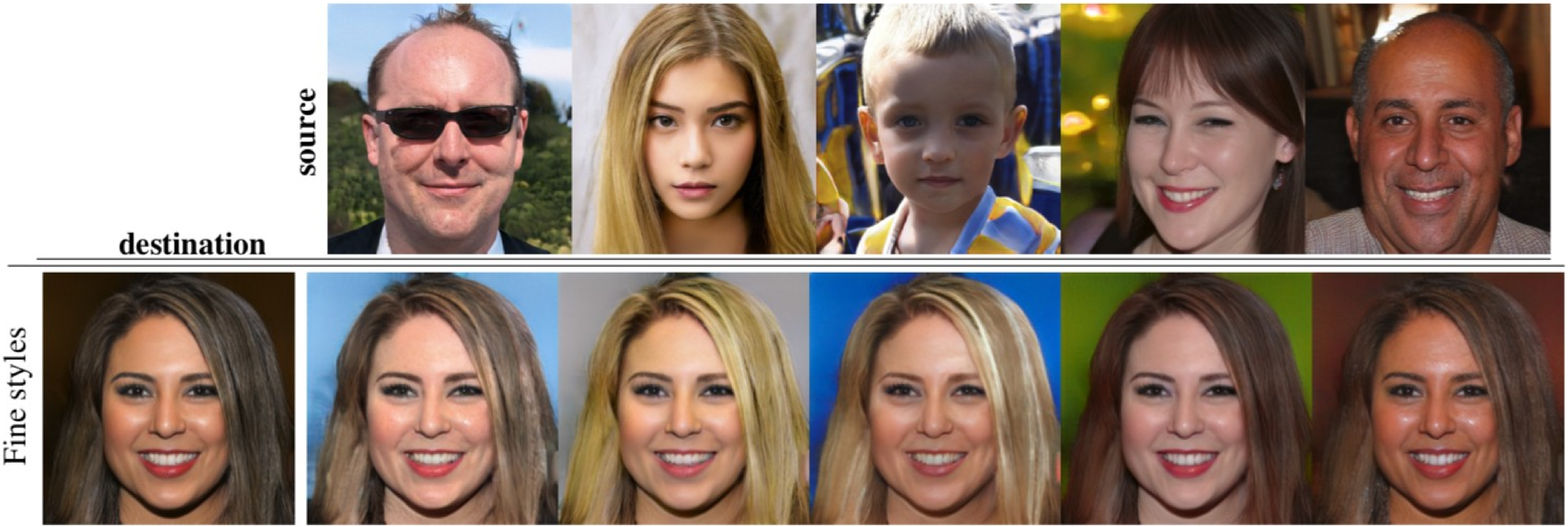
Mixing styles during image synthesis



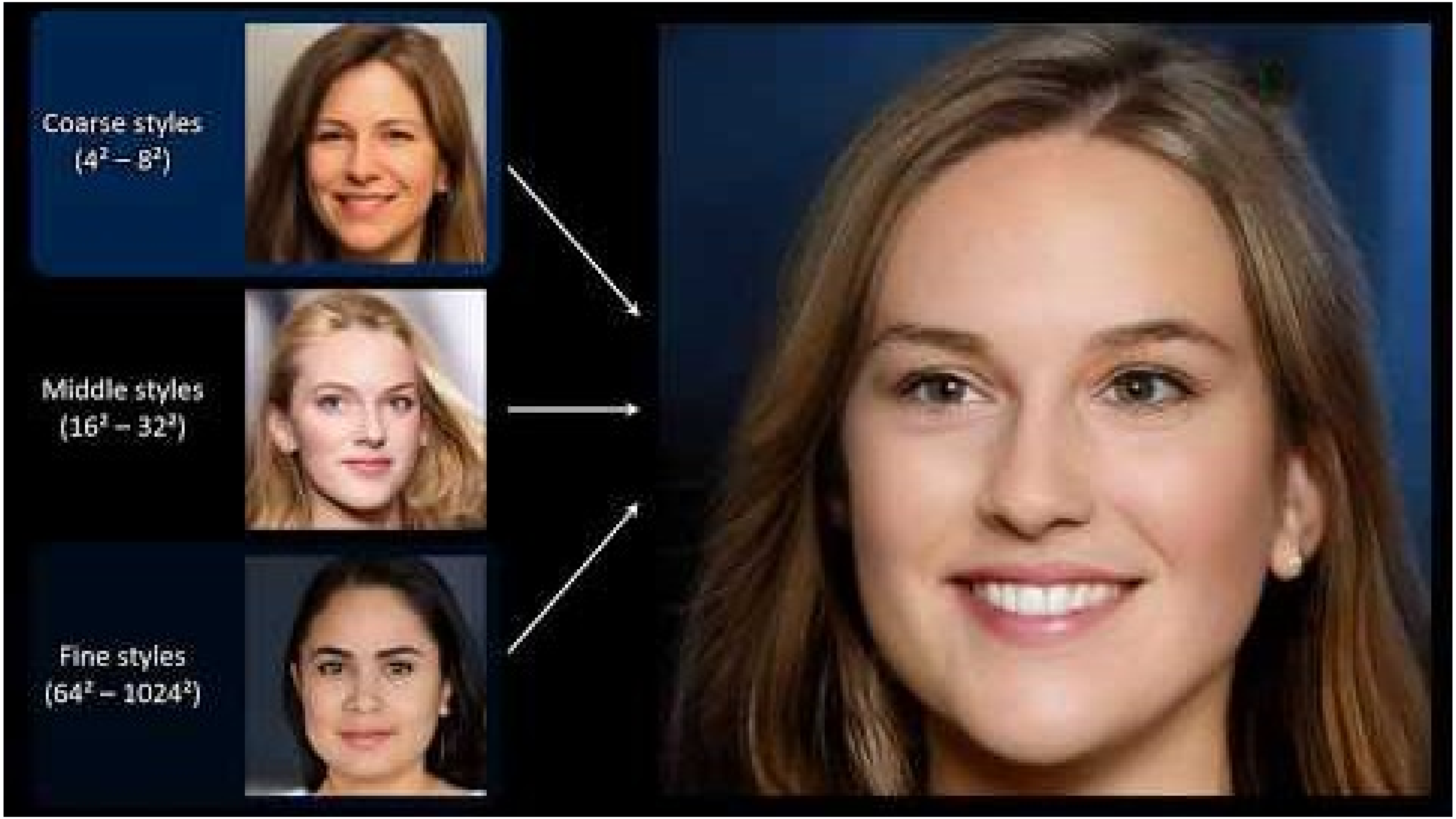
Mixing styles during image synthesis. Coarse styles such as pose, face shape and glasses are copied.

Middle styles copied: hair style, facial features but not pose or glasses

Copying only fine resolution style such as colour scheme



Style-based generator architecture



Major contributions

1. Significant improvement over traditional GAN generators architecture
2. Separation of high-level attributes from stochastic effects
3. Does not generate new images **from scratch** but rather through a smart combination of styles that are embedded in sample images (latent codes)