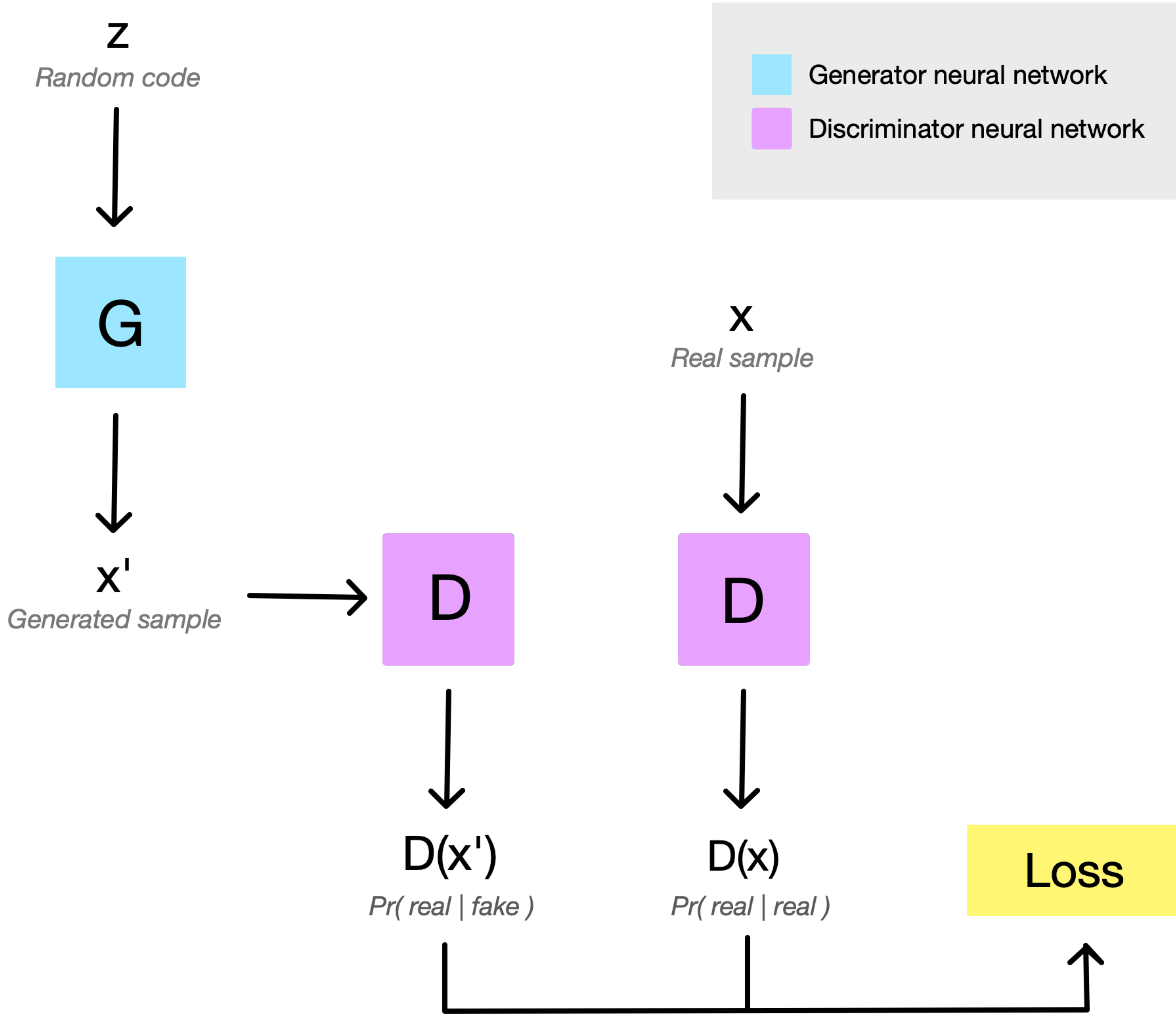
# ProGAN / StyleGAN

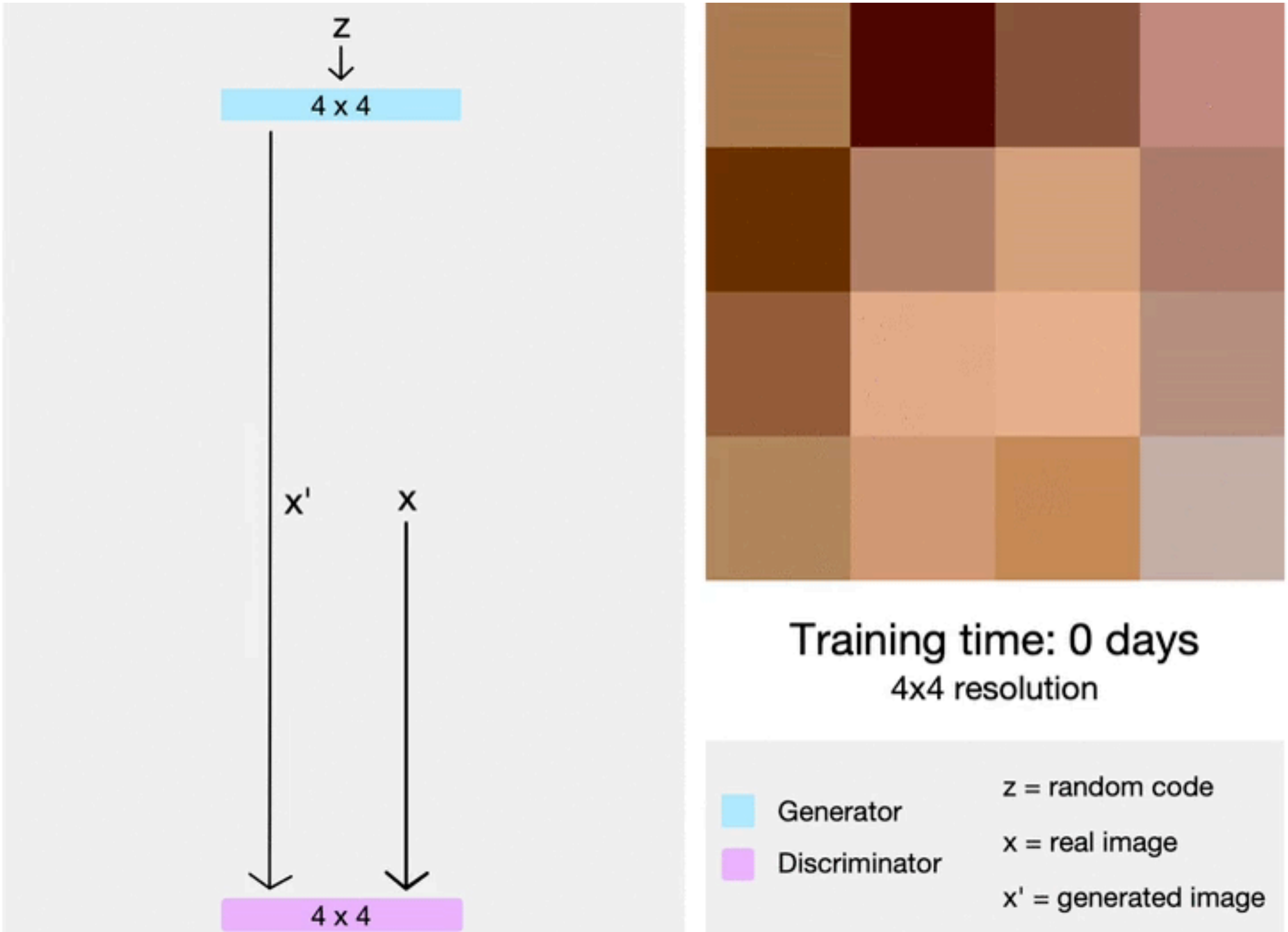
all slides shamelessly and mindlessly stolen from
https://towardsdatascience.com/progan-how-nvidia-generated-images-of-unprecedented-quality-51c98ec2cbd2
and
https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431,
and the original papers https://arxiv.org/abs/1710.10196
and https://arxiv.org/abs/1812.04948.

# Why just StyleGAN?

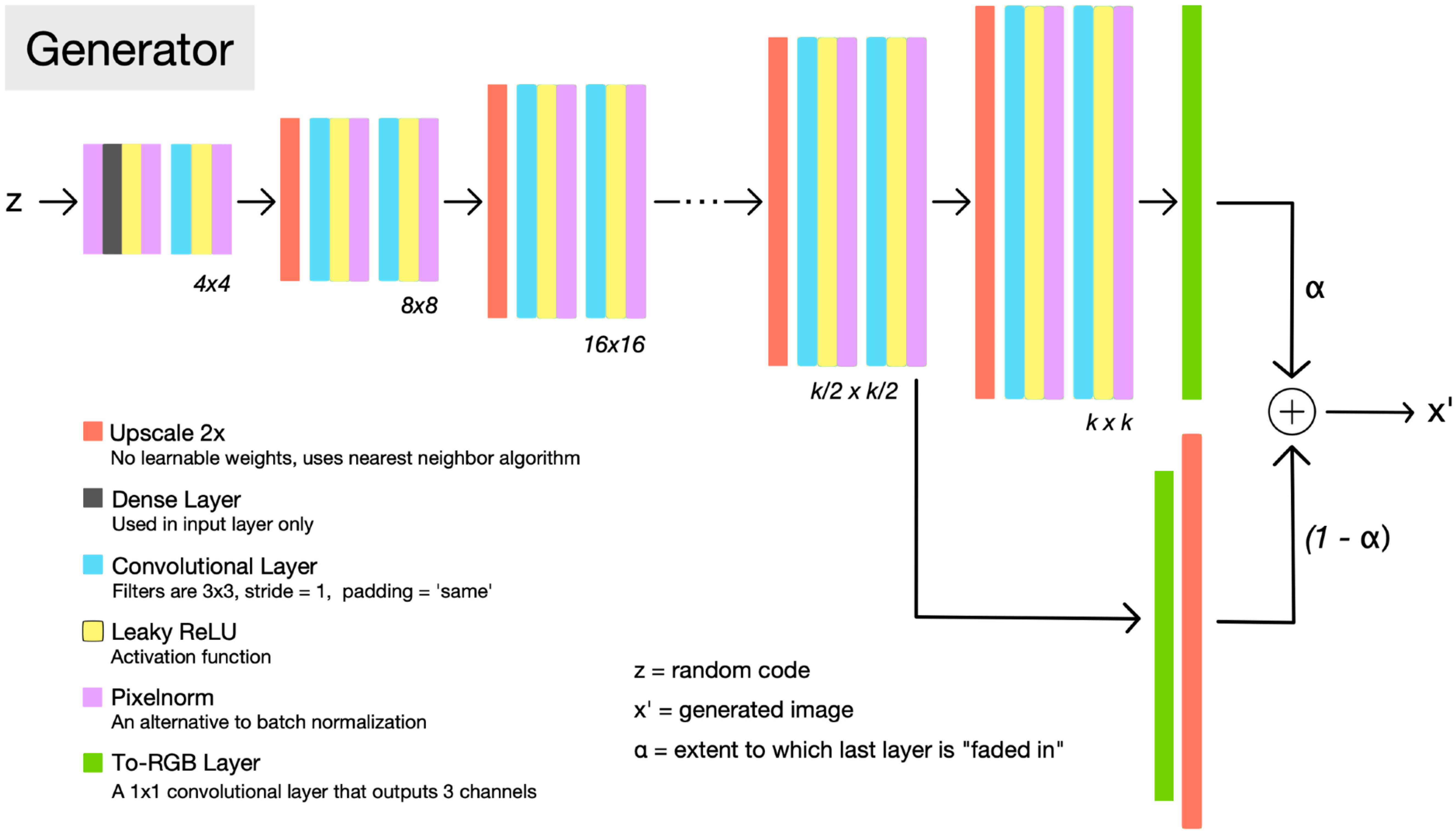
- The talk presents a dozen or so engineering tricks employed by the creators of StyleGAN (Karras et al).
- In themselves, probably not too many of these tricks would deserve extra attention from us when looking at the broader picture of deep learning methods.
- But together they give a diverse cross-section of useful techniques that we can employ when working with image data, and especially when generating image data.

# Negative cherry-picking


Generator neural network
Discriminator neural network
z
Random code
G
x
Real sample
x'
Generated sample
D
D
D(x')
Pr( real | fake )
D(x)
Pr( real | real )
Loss




**The pdf is not animated, see the animation here.**



Generator
z
4x4
8x8
16x16
k/2 x k/2
k x k
α
(1 - α)
x'
Upscale 2x
No learnable weights, uses nearest neighbor algorithm
Dense Layer
Used in input layer only
Convolutional Layer
Filters are 3x3, stride = 1, padding = 'same'
Leaky ReLU
Activation function
Pixelnorm
An alternative to batch normalization
To-RGB Layer
A 1x1 convolutional layer that outputs 3 channels
z = random code
x' = generated image
α = extent to which last layer is "faded in"

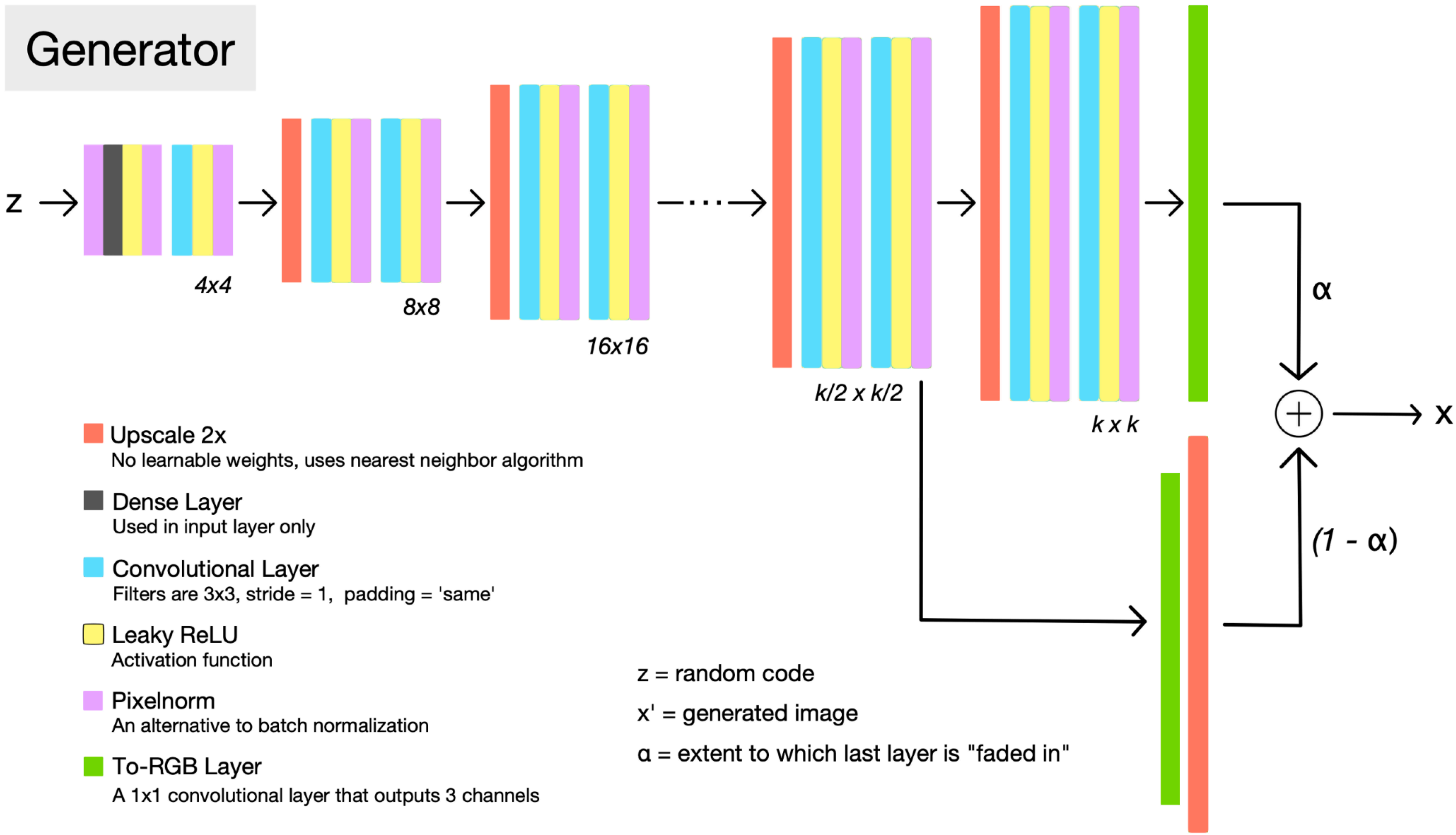
### Pixel Normalization

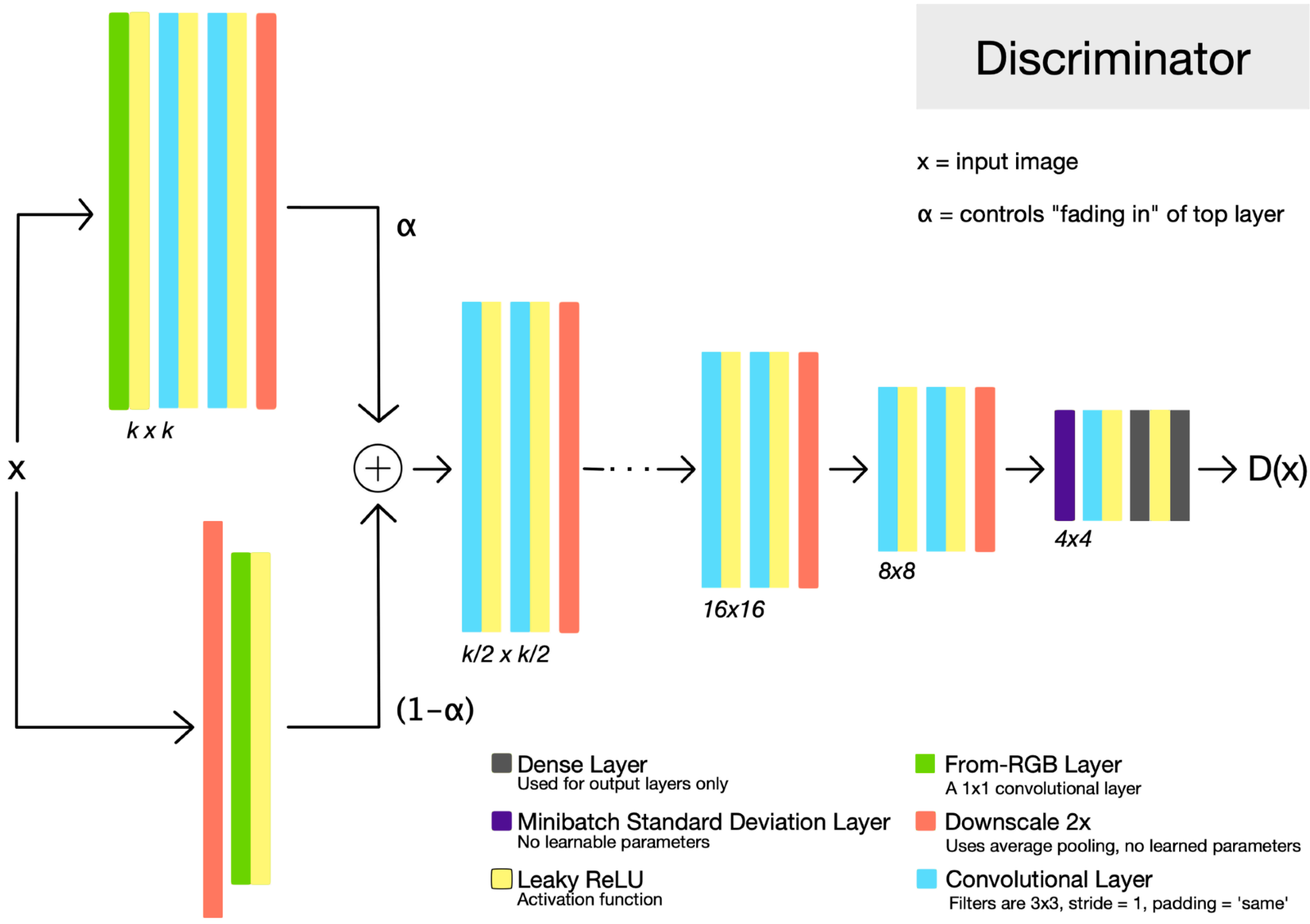
Instead of using batch normalization, as is commonly done, the authors used *pixel normalization*. This "pixelnorm" layer has no trainable weights. It normalizes the feature vector in each pixel to unit length, and is applied after the convolutional layers in the generator. This is done to prevent signal magnitudes from spiraling out of control during training.

$$b_{x,y}^{j} = \frac{a_{x,y}^{j}}{\sqrt{\frac{1}{C}\sum_{j=0}^{C} {a_{x,y}^{j}}^{2} + \epsilon}}$$

The values of each pixel (x, y) across C channels are normalized to a fixed length. Here, **a** is the input tensor, **b** is the output tensor, and **ε** is a small value to prevent dividing by zero.

# Fade in


Generator
z
4x4
8x8
16x16
k/2 x k/2
k x k
α
(1 - α)
x'
Upscale 2x
No learnable weights, uses nearest neighbor algorithm
Dense Layer
Used in input layer only
Convolutional Layer
Filters are 3x3, stride = 1, padding = 'same'
Leaky ReLU
Activation function
Pixelnorm
An alternative to batch normalization
To-RGB Layer
A 1x1 convolutional layer that outputs 3 channels
z = random code
x' = generated image
α = extent to which last layer is "faded in"


Discriminator
x = input image
α = controls "fading in" of top layer
α
k x k
x
(1−α)
k/2 x k/2
16x16
8x8
4x4
D(x)
Dense Layer
Used for output layers only
Minibatch Standard Deviation Layer
No learnable parameters
Leaky ReLU
Activation function
From-RGB Layer
A 1x1 convolutional layer
Downscale 2x
Uses average pooling, no learned parameters
Convolutional Layer
Filters are 3x3, stride = 1, padding = 'same'

# Minibatch standard deviation

# WGAN-GP Loss

$$Loss_G = -D(x')$$

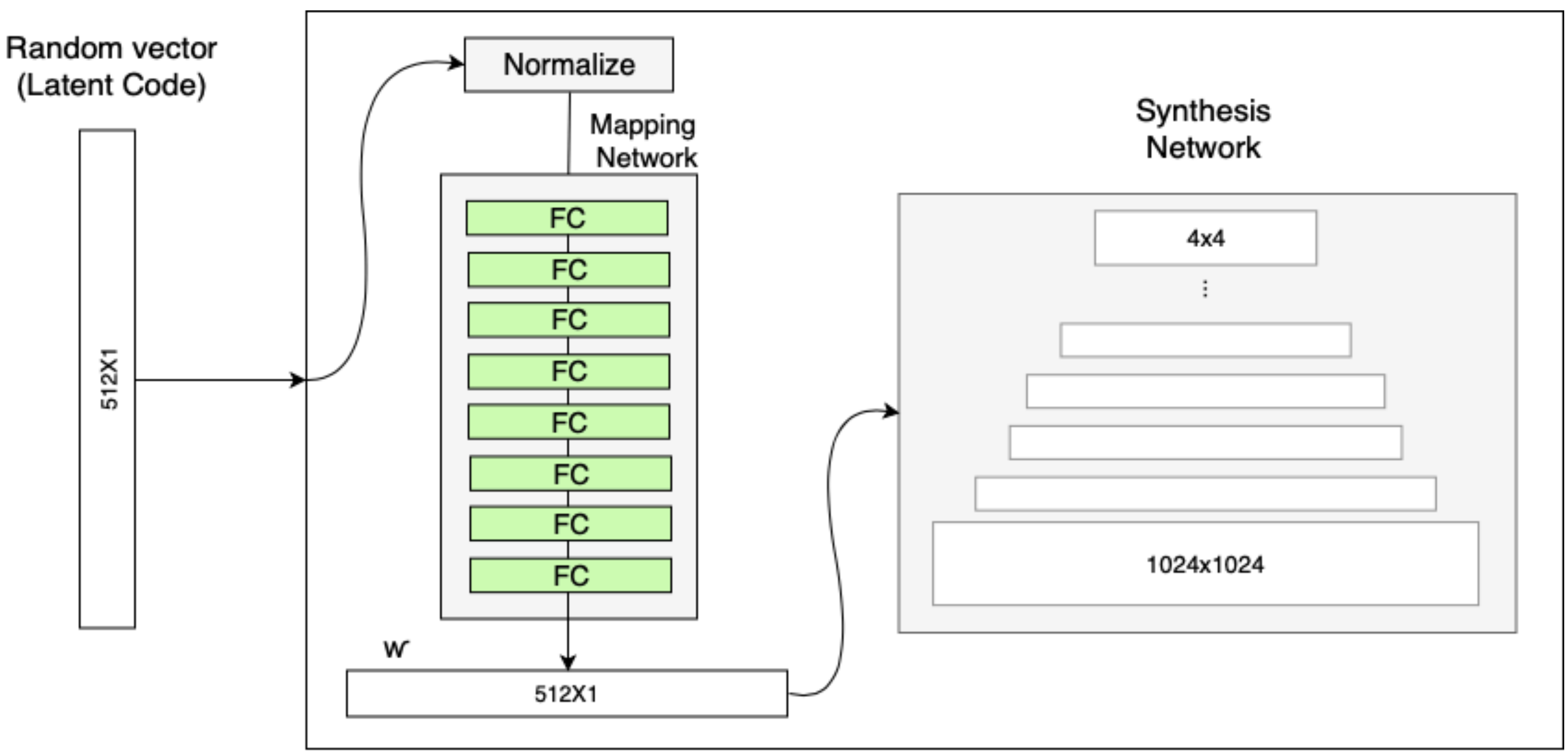
$$GP = (\|\nabla D(ax' + (1-a)x))\|_2 - 1)^2$$

$$Loss_D = -D(x) + D(x') + \lambda * GP$$

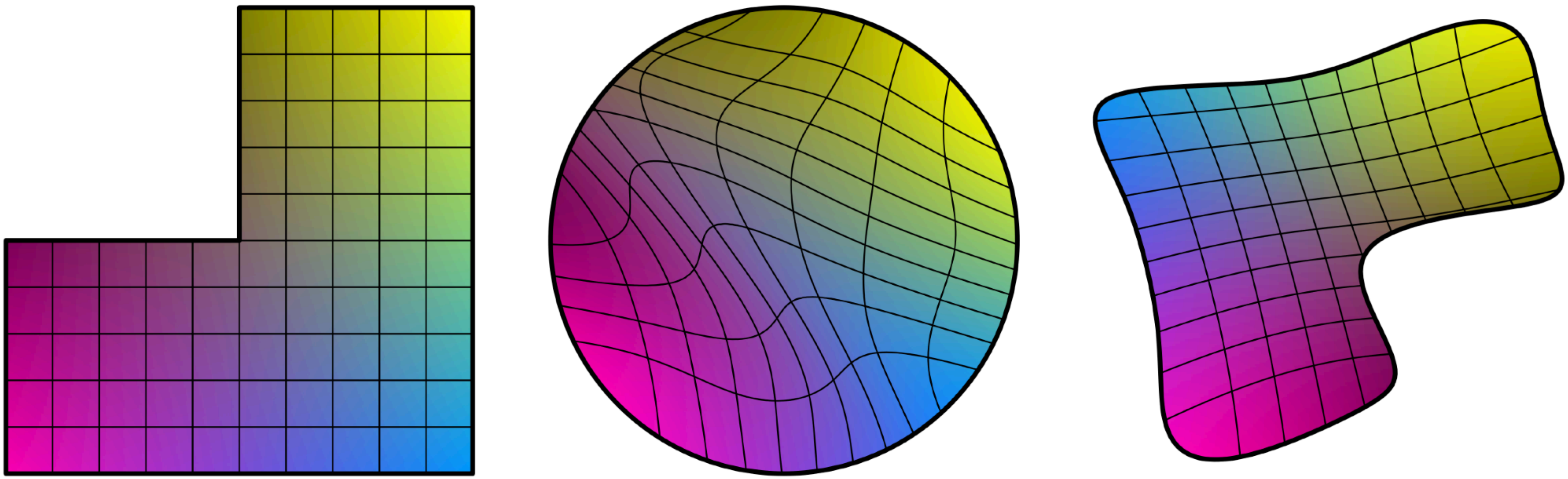
# StyleGAN


source
destination
Coarse styles copied


Source B
Source A
Coarse styles from source B
Middle styles from source B
Fine from B



Random vector
(Latent Code)
512X1
Normalize
Mapping
Network
FC
FC
FC
FC
FC
FC
FC
FC
w
512X1
Synthesis
Network
4x4
⋮
1024x1024

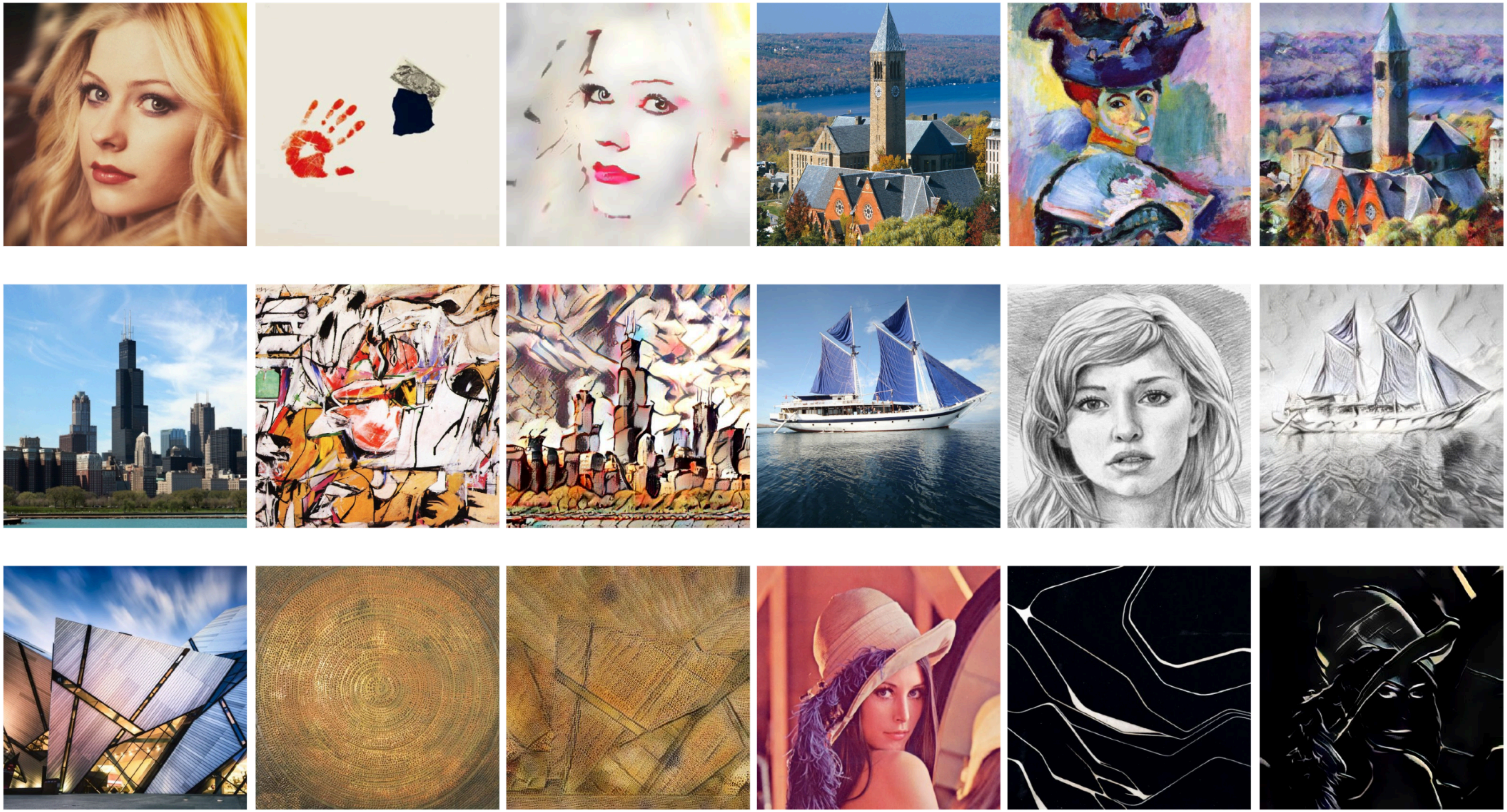


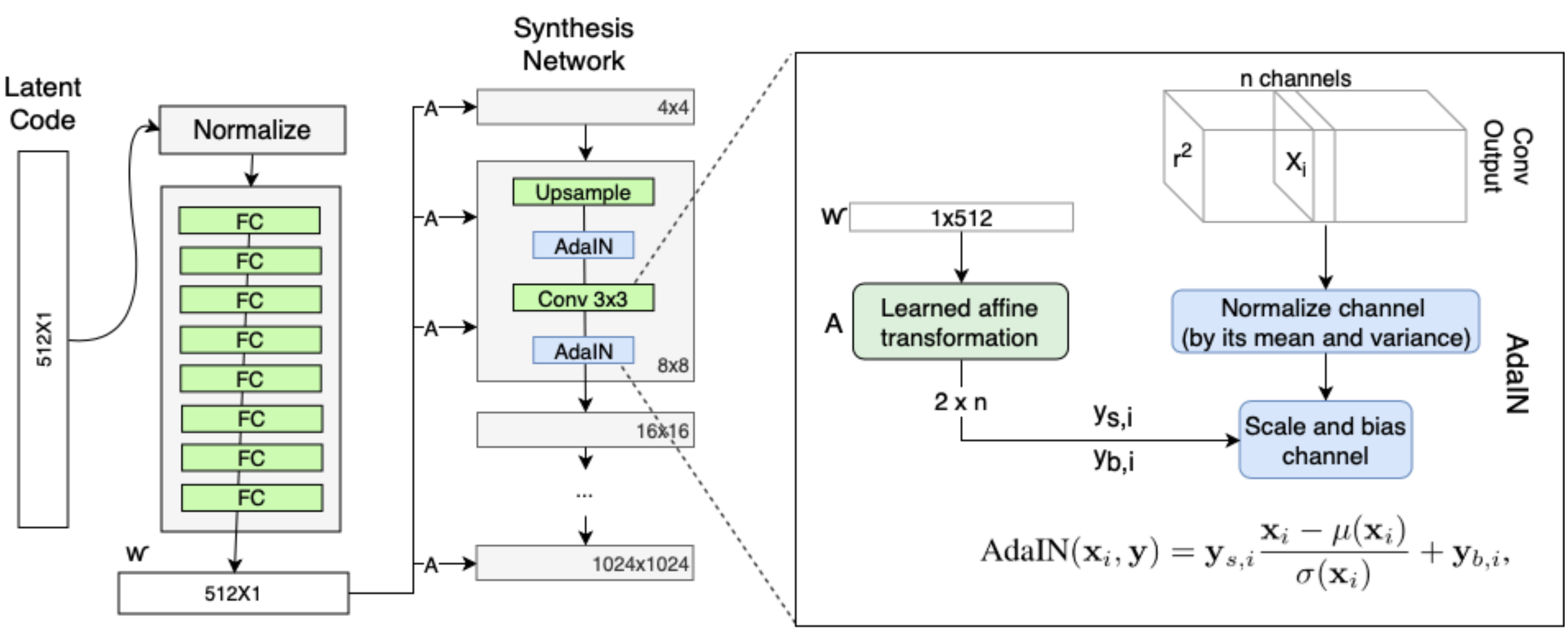
(a) Distribution of features in training set

(b) Mapping from $\mathcal{Z}$ to features

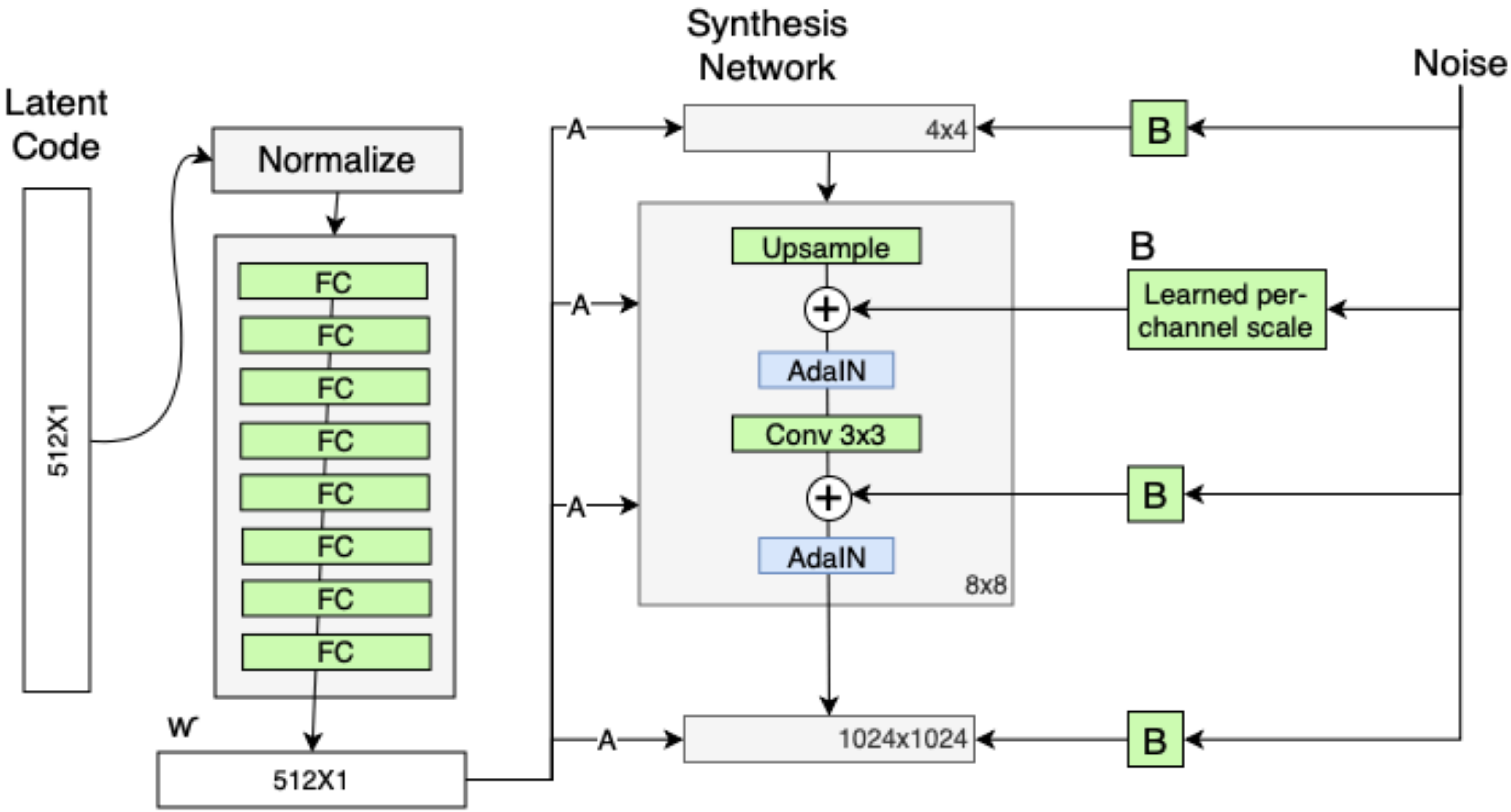
(c) Mapping from $\mathcal{W}$ to features

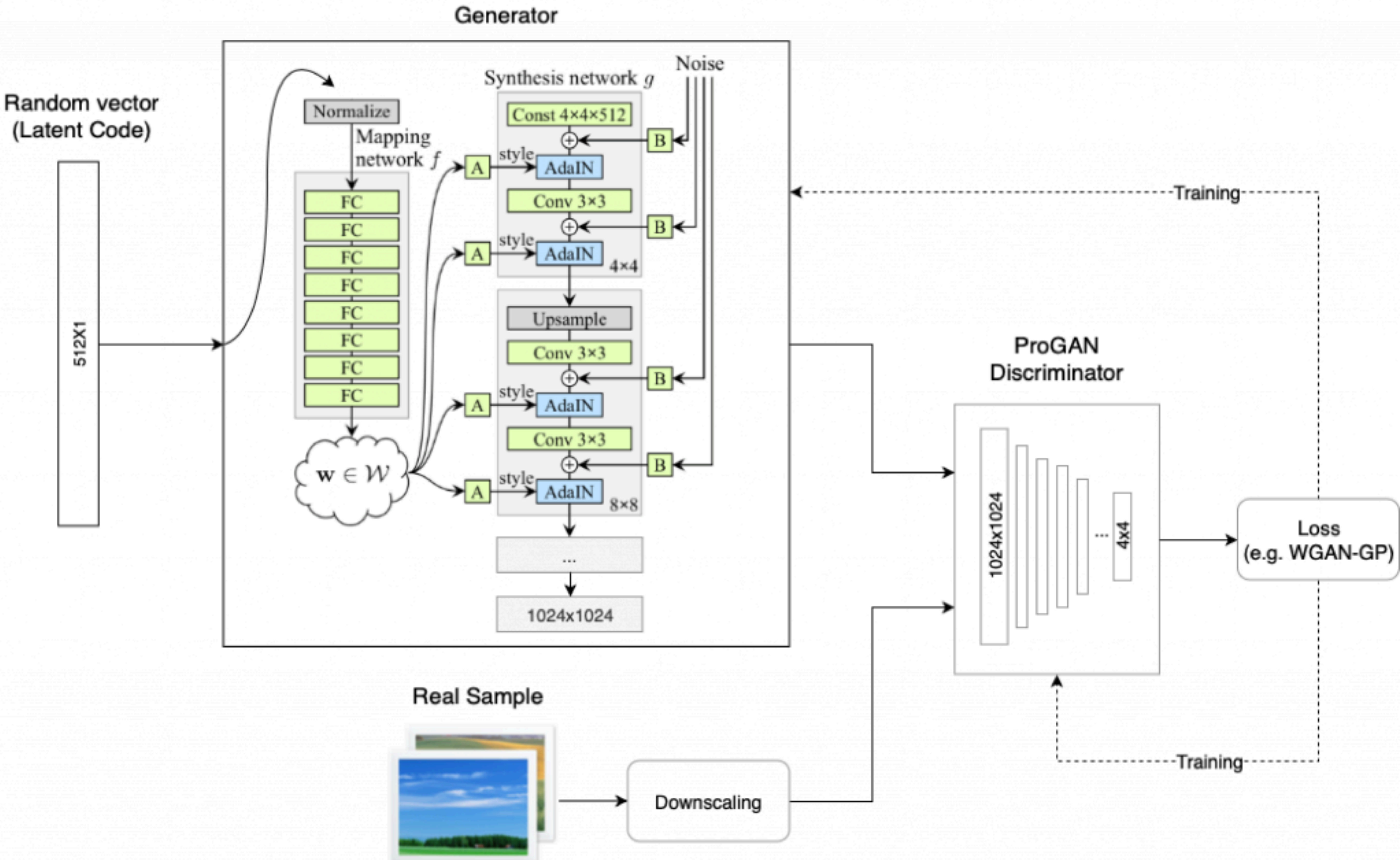
# Adaptive Instance Normalization





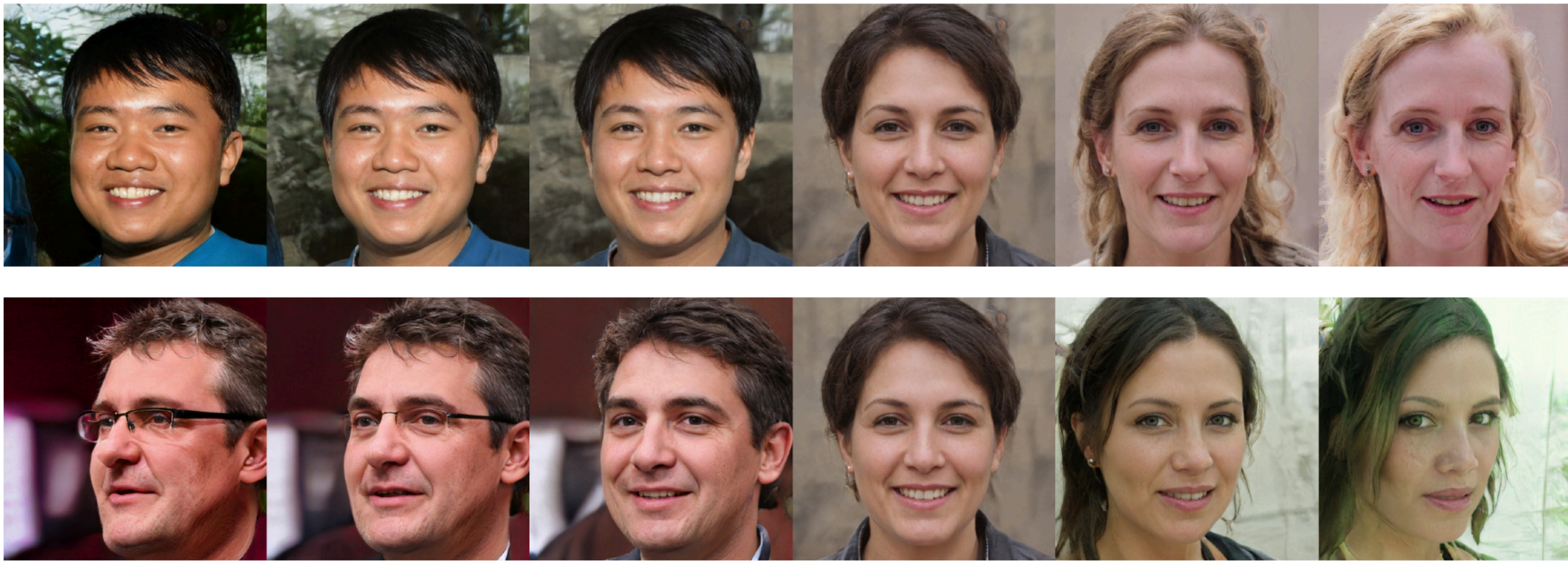
Latent Code
512X1
Normalize
FC
FC
FC
FC
FC
FC
FC
FC
w
512X1
Synthesis Network
A
4x4
A
Upsample
AdaIN
A
Conv 3x3
AdaIN
8x8
16x16
...
A
1024x1024
n channels
$r^2$
$X_i$
Conv Output
w
1x512
A
Learned affine transformation
Normalize channel (by its mean and variance)
AdaIN
2 x n
$y_{s,i}$
$y_{b,i}$
Scale and bias channel
$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$


Latent
Code
512X1
Normalize
FC
FC
FC
FC
FC
FC
FC
FC
w
512X1
A
A
A
A
Synthesis
Network
4x4
Upsample
AdaIN
Conv 3x3
AdaIN
8x8
1024x1024
Noise
B
B
Learned per-
channel scale
B
B



Generator
Random vector
(Latent Code)
512X1
Normalize
Mapping
network f
FC
FC
FC
FC
FC
FC
FC
FC
w ∈ W
Synthesis network g
Noise
Const 4×4×512
style
AdaIN
Conv 3×3
4×4
Upsample
8×8
...
1024x1024
A
B
Training
ProGAN
Discriminator
1024x1024
4x4
Loss
(e.g. WGAN-GP)
Real Sample
Downscaling
Training

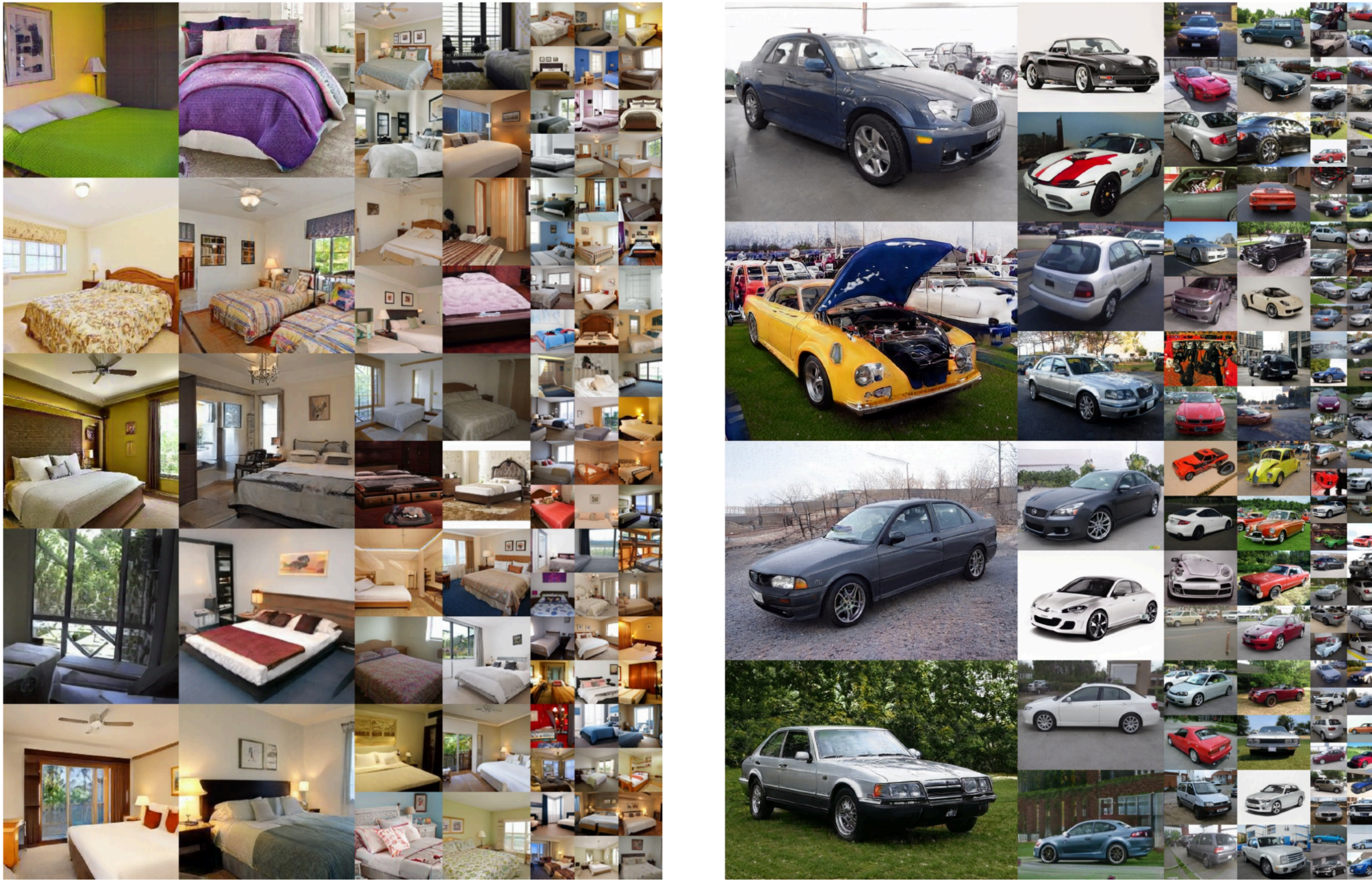
# Truncation trick in W



$\psi = 1$ $\psi = 0.7$ $\psi = 0.5$ $\psi = 0$ $\psi = -0.5$ $\psi = -1$

# Feature disentanglement

# Not just faces



https://colab.research.google.com/drive/1xR-Sj1nWwE6wZ5vIUy45c23auObFKLSE#scrollTo=KMDrI2w3GC_t