ProGAN
StyleGAN
StyleGAN2

(NVIDIA)

slides shamelessly and mindlessly stolen from

https://towardsdatascience.com/progan-how-nvidia-generated-images-of-unprecedented-quality-51c98ec2cbd2,

https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431,

https://towardsdatascience.com/stylegan2-ace6d3da405d,

and the original papers
https://arxiv.org/abs/1710.10196
https://arxiv.org/abs/1812.04948
https://arxiv.org/abs/1912.04958

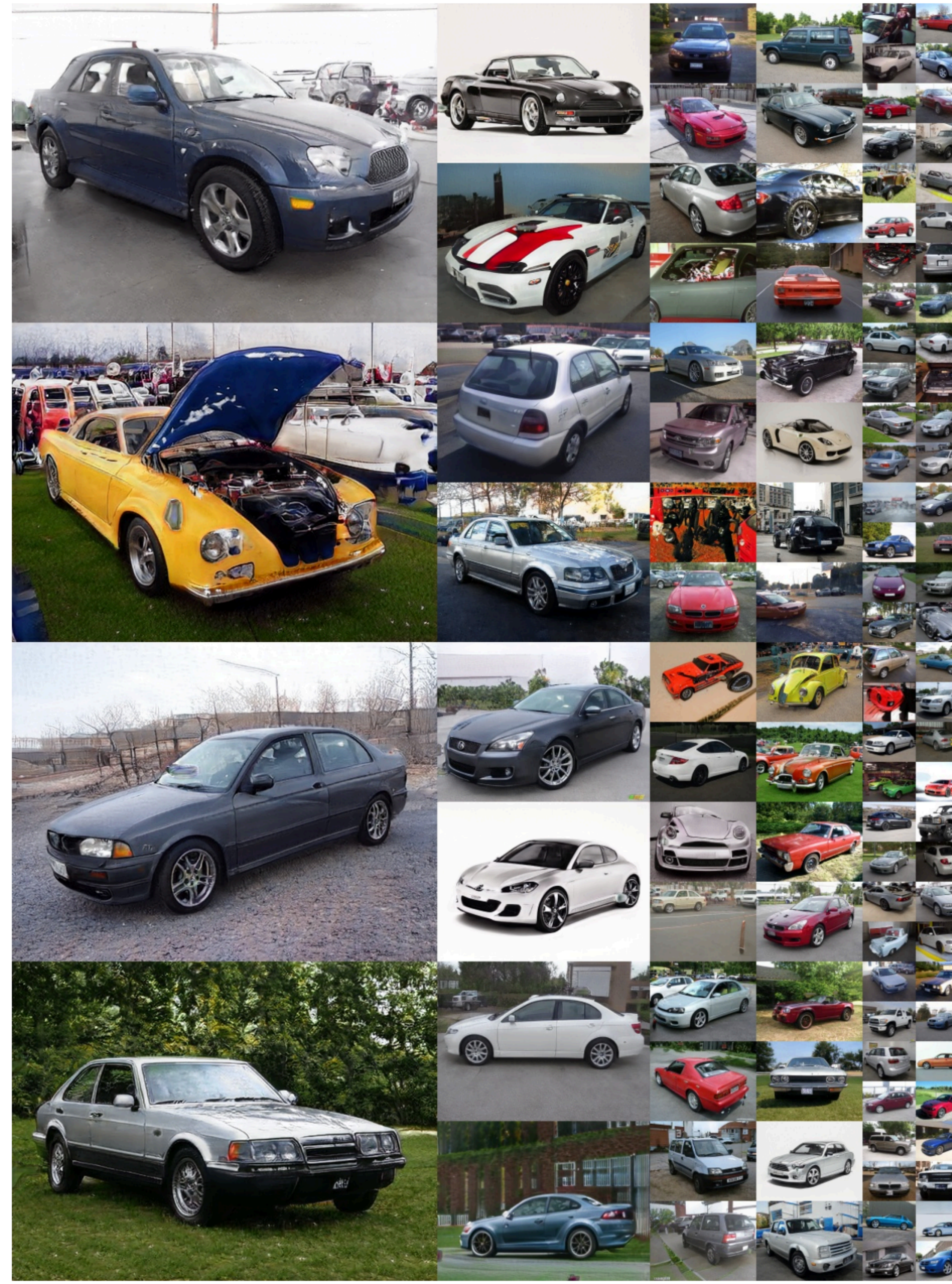
Why StyleGAN in particular?

- The talk presents a dozen or so engineering tricks employed by the creators of the very successful generative model StyleGAN (Karras et al, NVIDIA).

- 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.

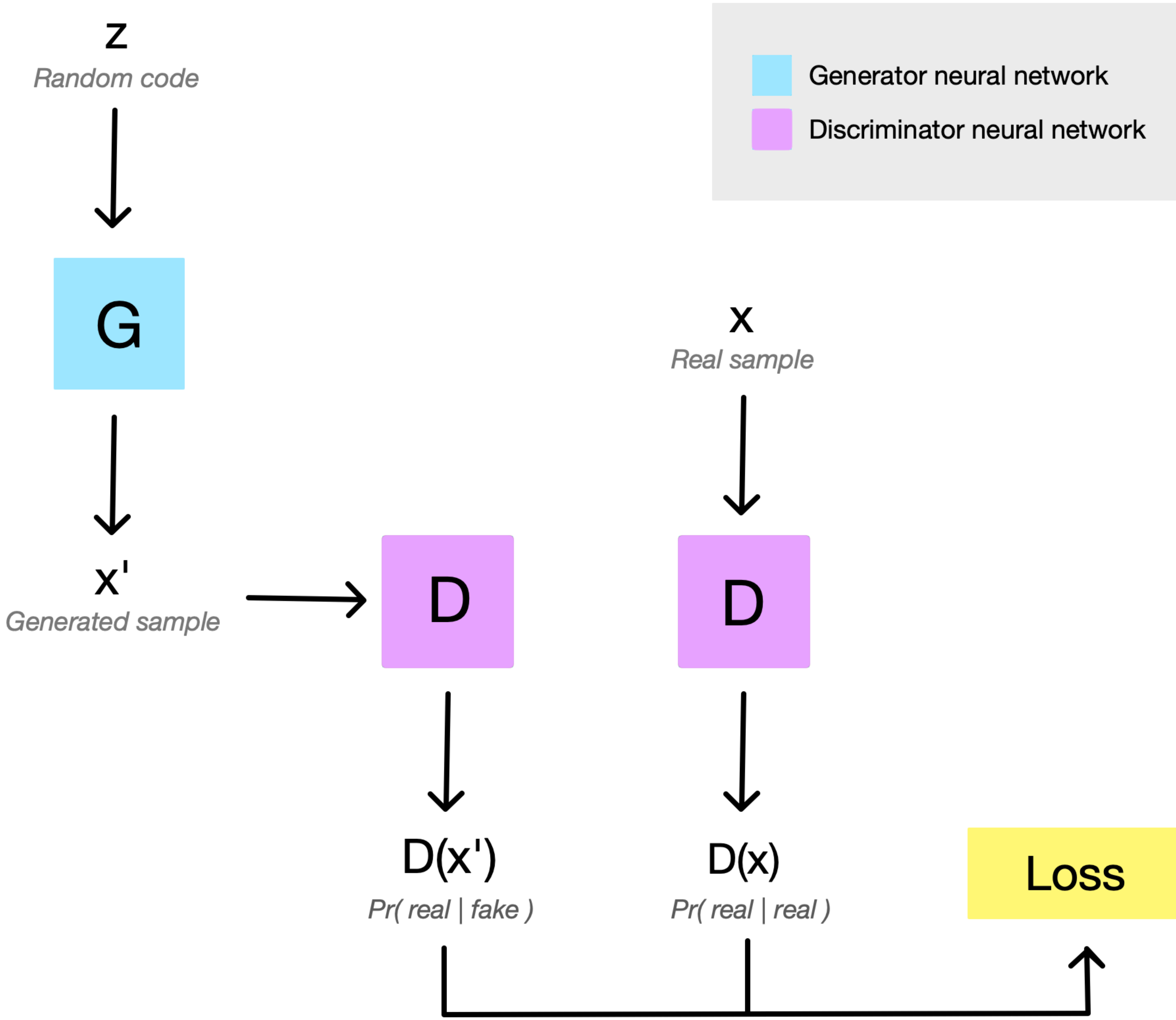
- But together they give a diverse cross-section of useful techniques that we can employ when working with image data, and especially when generating image data.

Not just faces

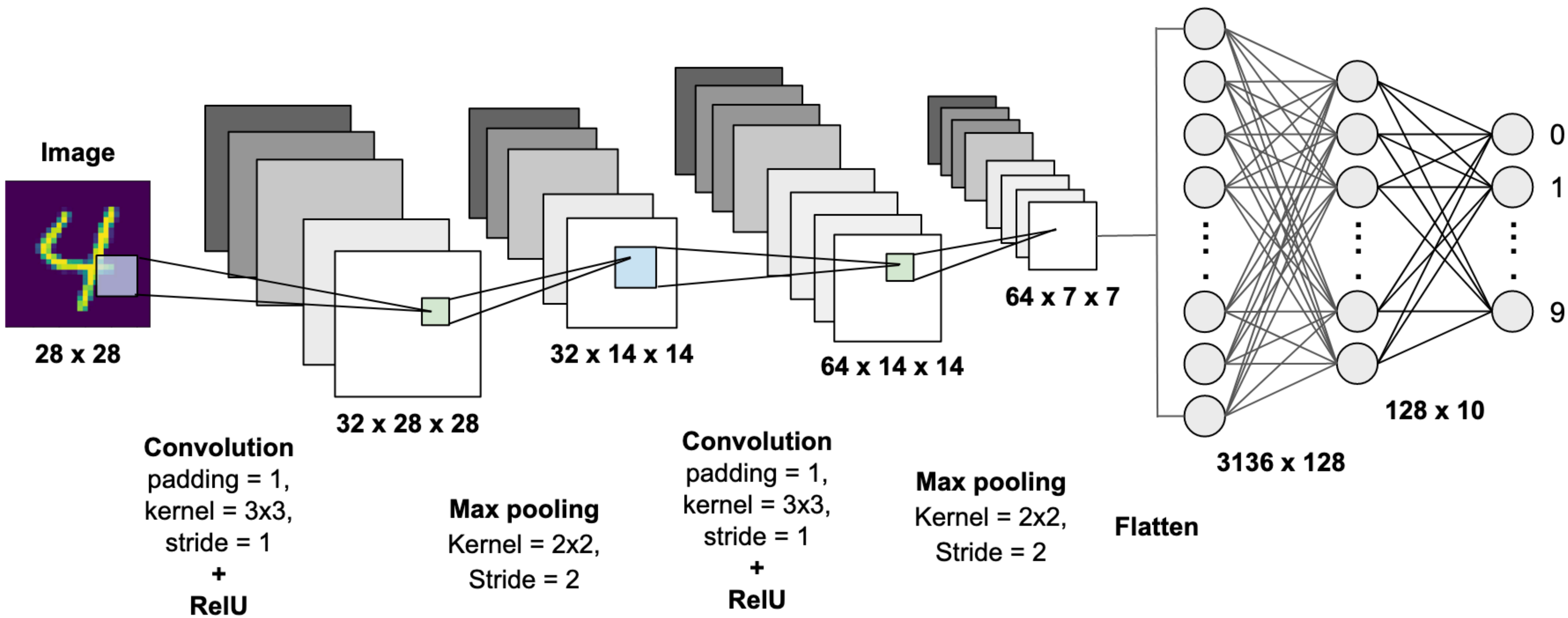


Negative cherry-picking (StyleGAN1)

GAN reminder

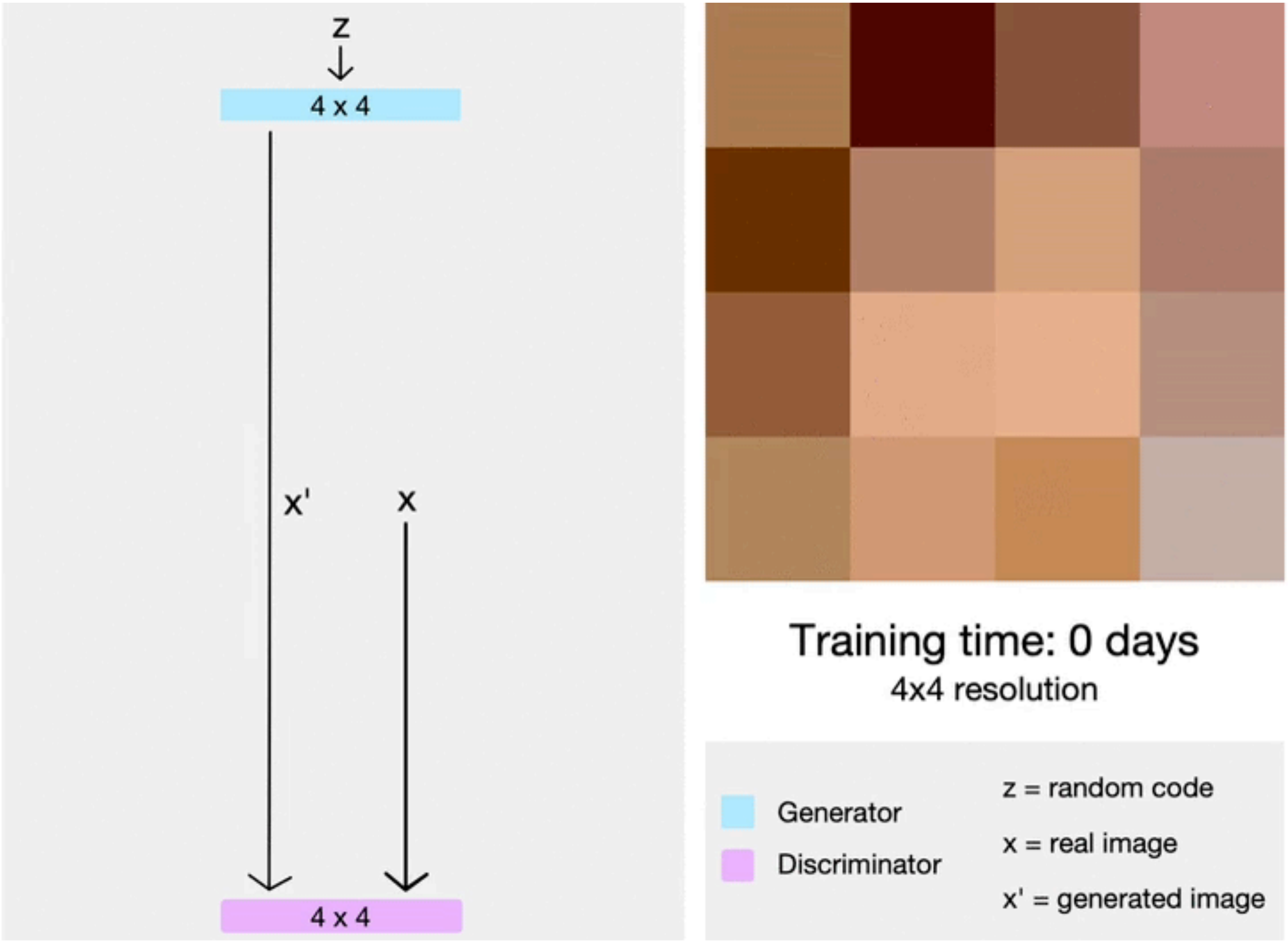


Feature map reminder

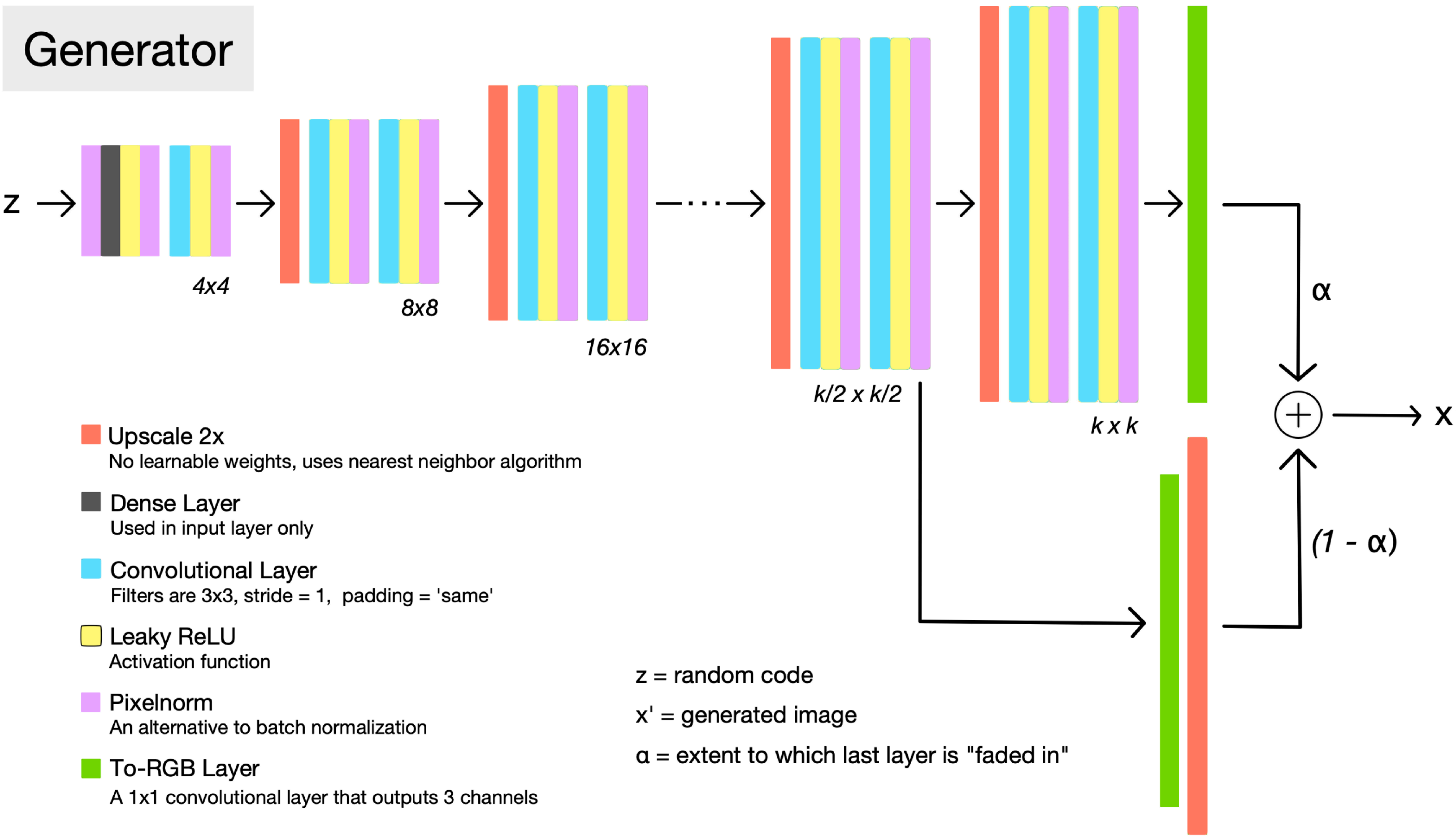


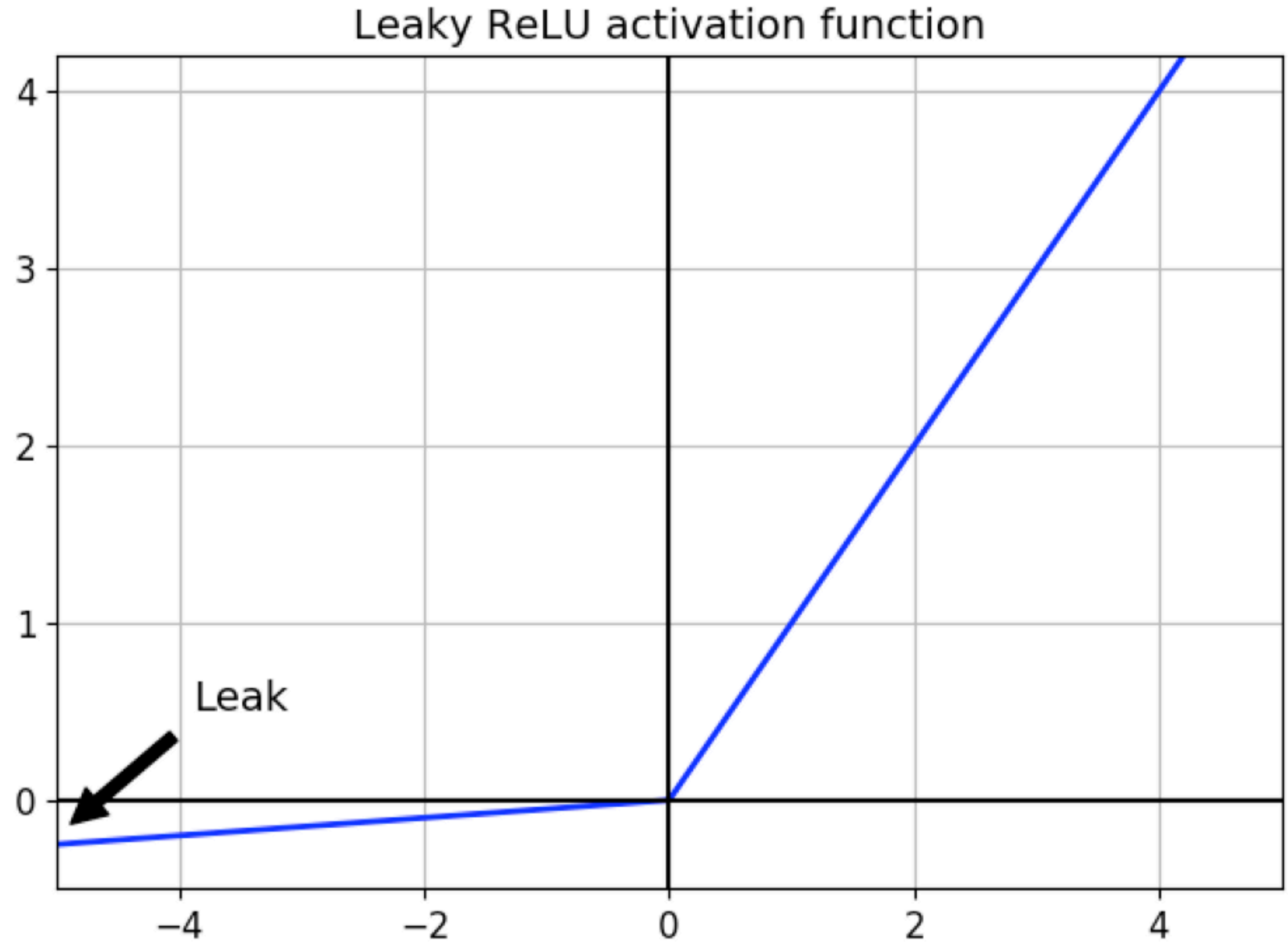
This image explains recognition, but feature maps for generators are fundamentally the same: (height x width x channels) sized 3D arrays.

ProGAN
(progressive
growing of GANs)
2017 November



The pdf is not animated, see the animation <u>here</u>.


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"

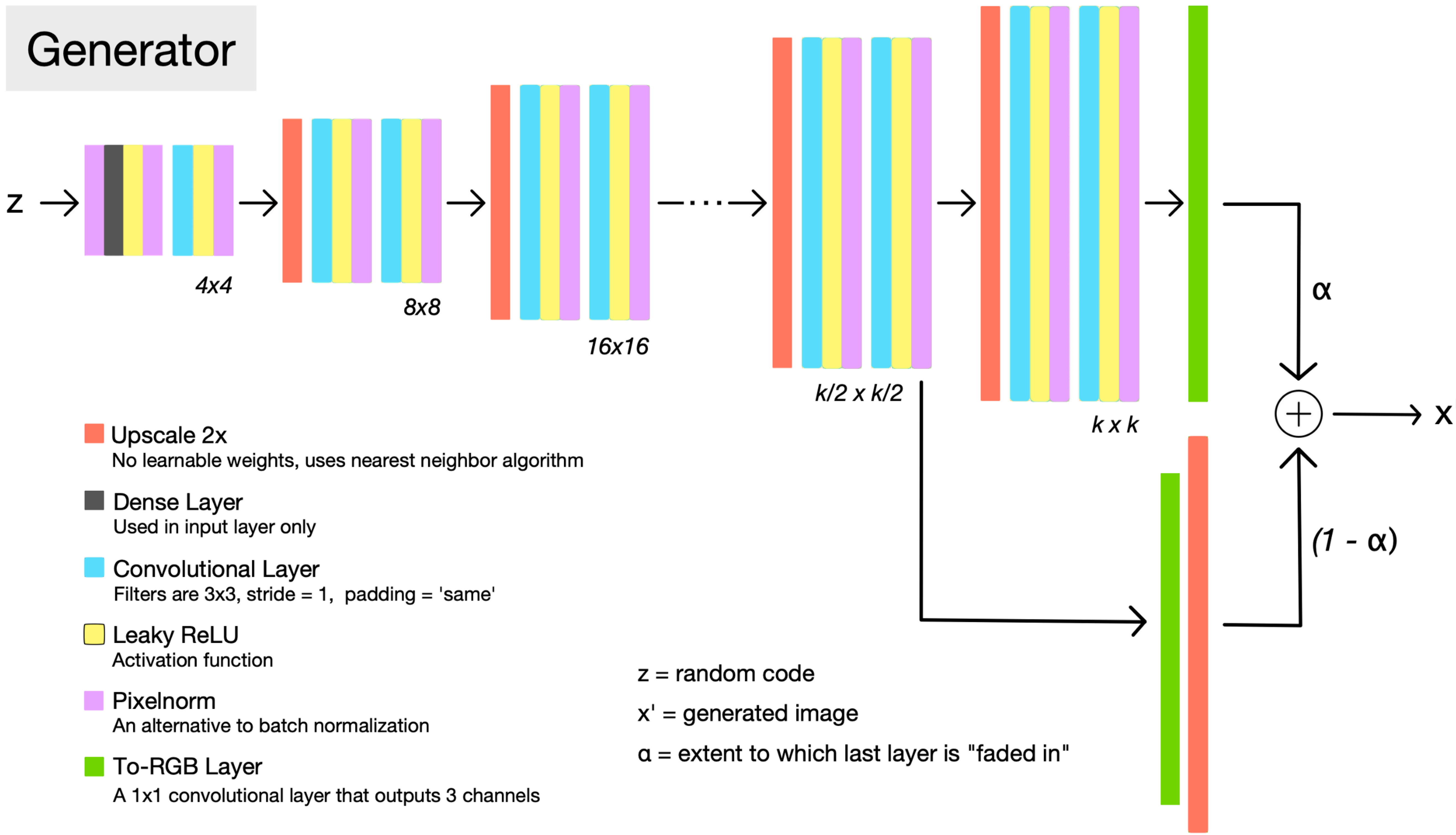

Leaky ReLU activation function
4
3
2
1
0
Leak
−4
−2
0
2
4

Pixel Normalization

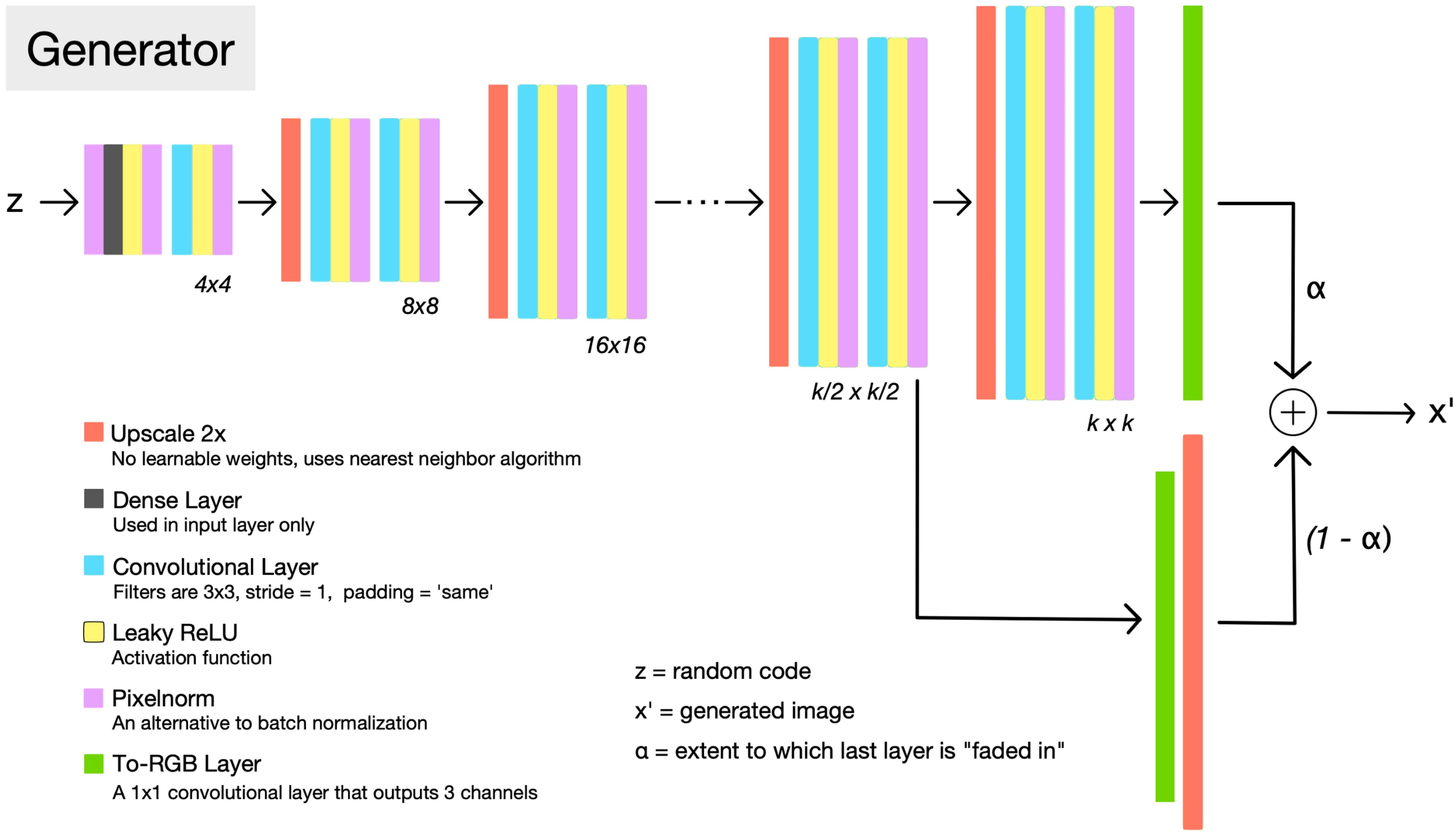
Instead of using batch normalization, as is commonly done, the authors used *pixel normalization*. This “pixelnorm” layer has no trainable weights. It normalizes the feature vector in each pixel to unit length, and is applied after the convolutional layers in the generator. This is done to prevent signal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

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

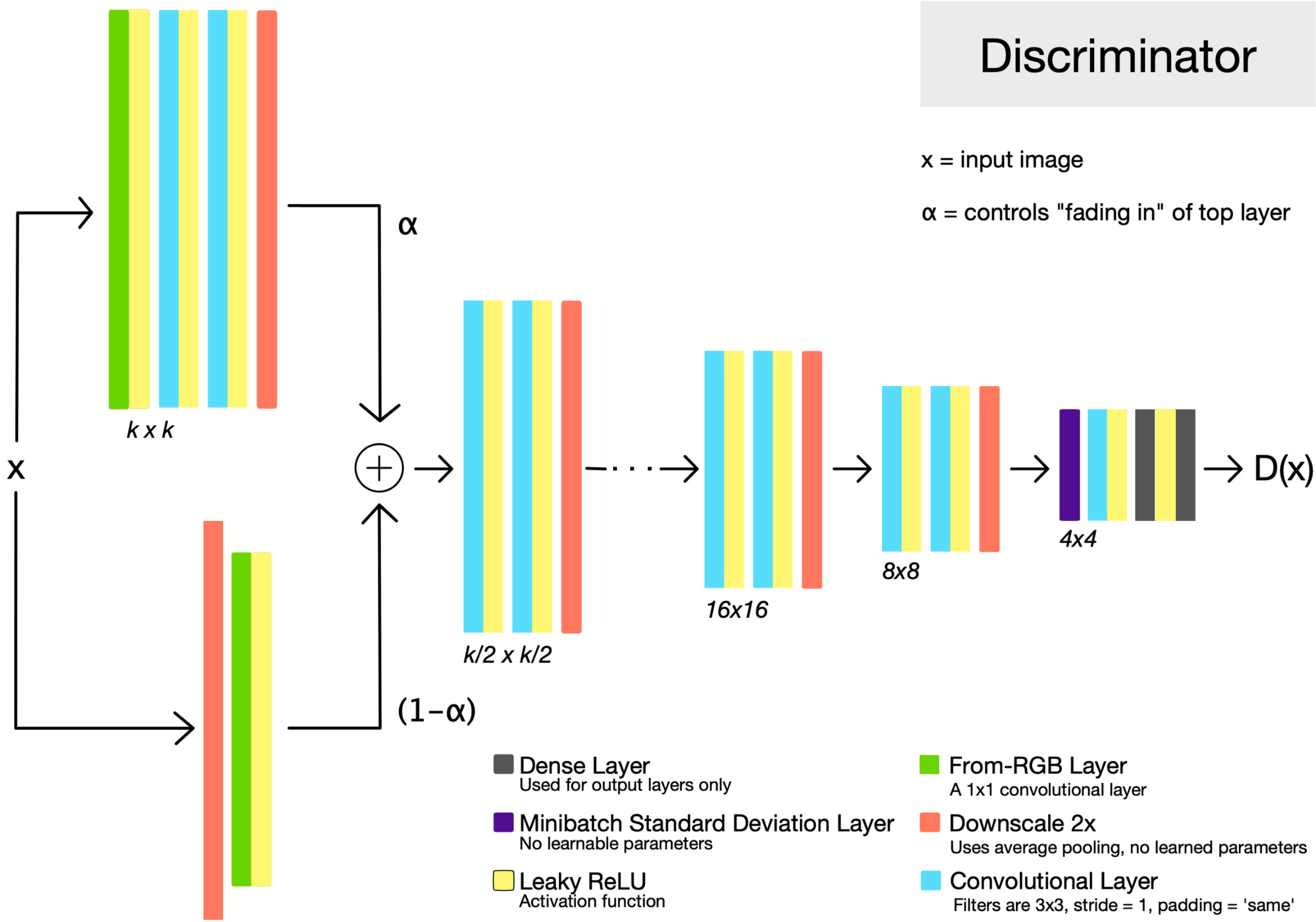
The values of each pixel (x, y) across C channels are normalized to a fixed length. Here, **a** is the input tensor, **b** is the output tensor, and **ε** is a small value to prevent dividing by zero.


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"

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
x
α
k x k
(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

- The details don't matter much, but the core idea is this:
- We'd like to avoid the generator creating beautiful but identical pictures. (See *mode collapse*.)
- So we score how diverse our generated minibatch is, based on a higher layer activation map of the discriminator.
- We add this score as another loss term.

Gradient regularization

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

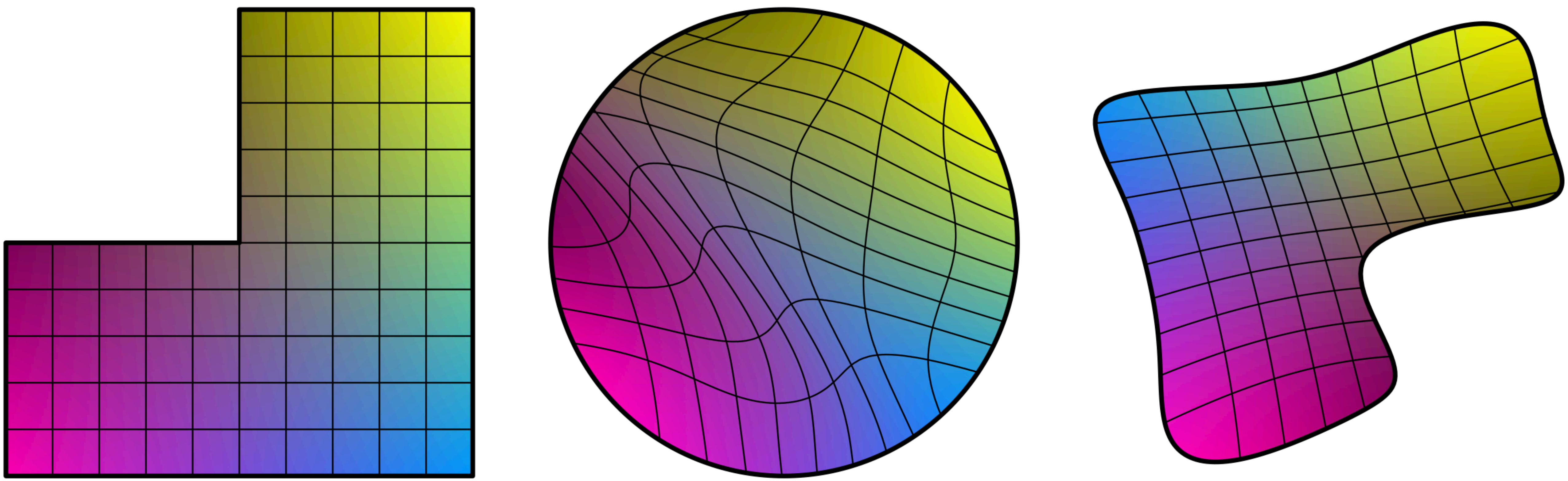
- Don't mind the particular formula (WGAN-GP, Gulrajani et al), here is the intuition:
- Usually our optimizer calculates gradients of the loss with respect to the neural weights, so we don't have to deal with gradients explicitly.
- This time we calculate gradients of the loss with respect to the ***input***.
- We use this to quantify how smooth the input-output mapping of our network is.
- The optimizer then calculates the gradients of this smoothness metric with respect to neural weights, as usual.

StyleGAN
2018 December


source
destination
Coarse styles copied


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

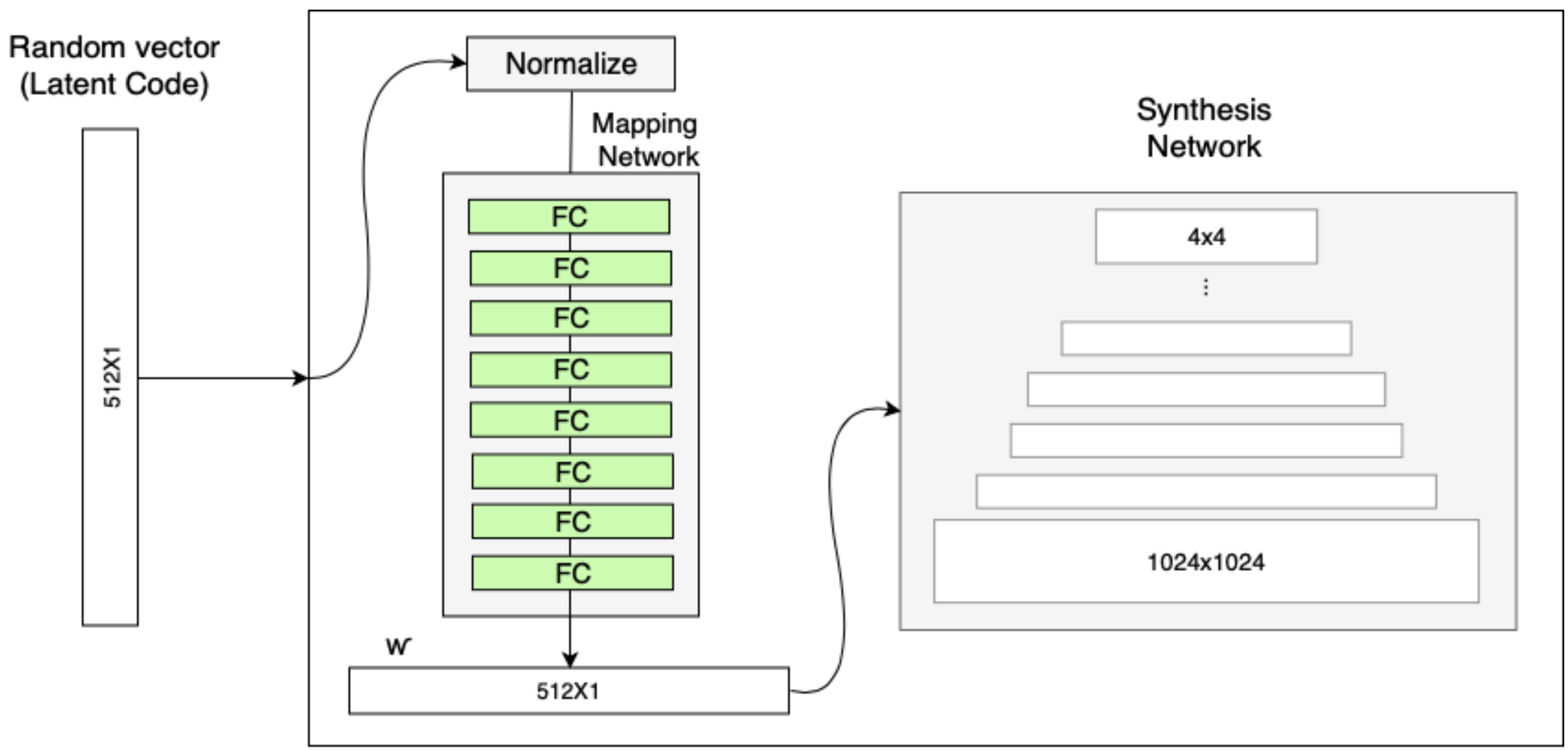
Intermediate latent space W



(a) Distribution of features in training set

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

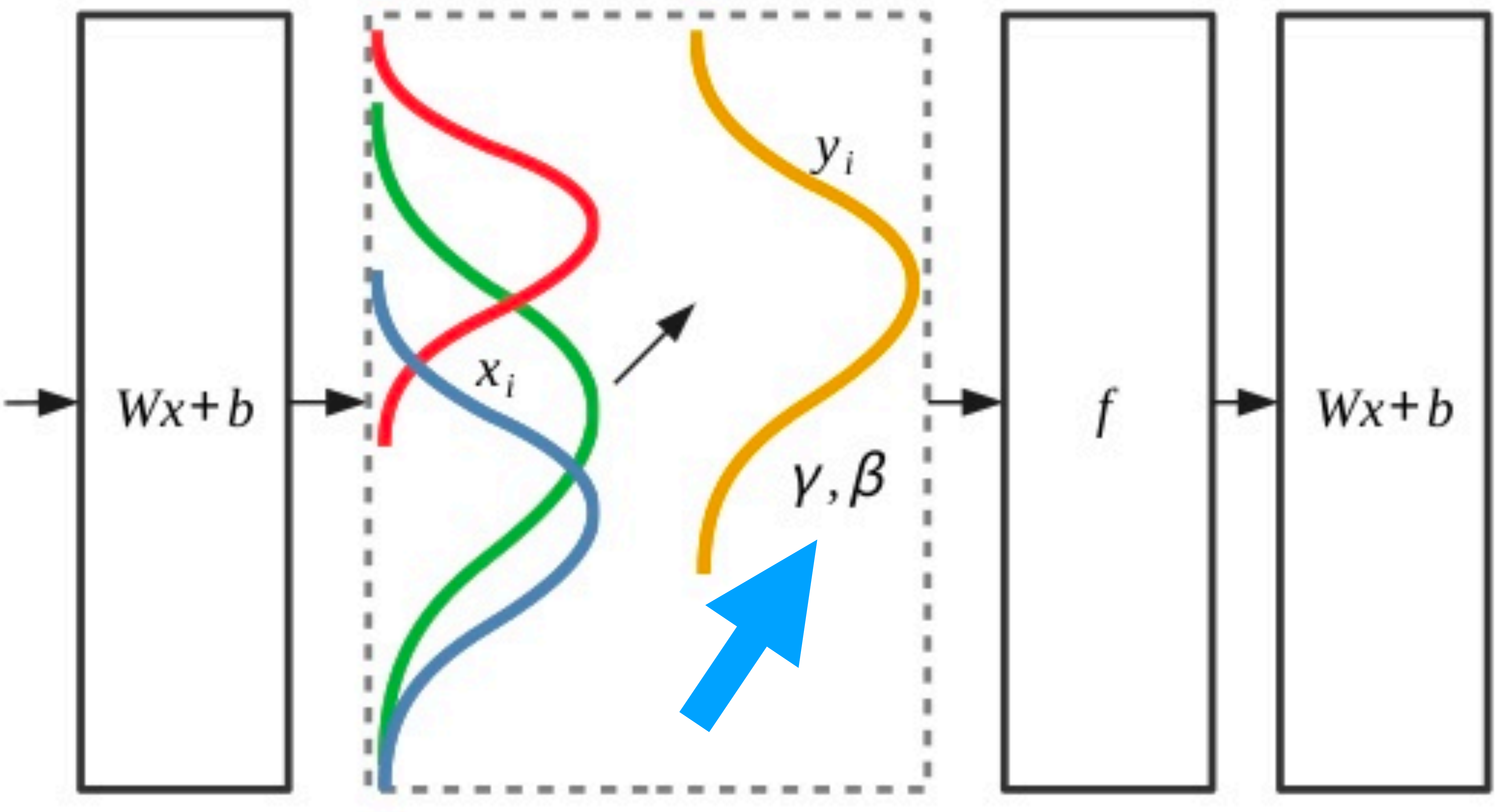
(c) Mapping from $\mathcal{W}$ to features


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

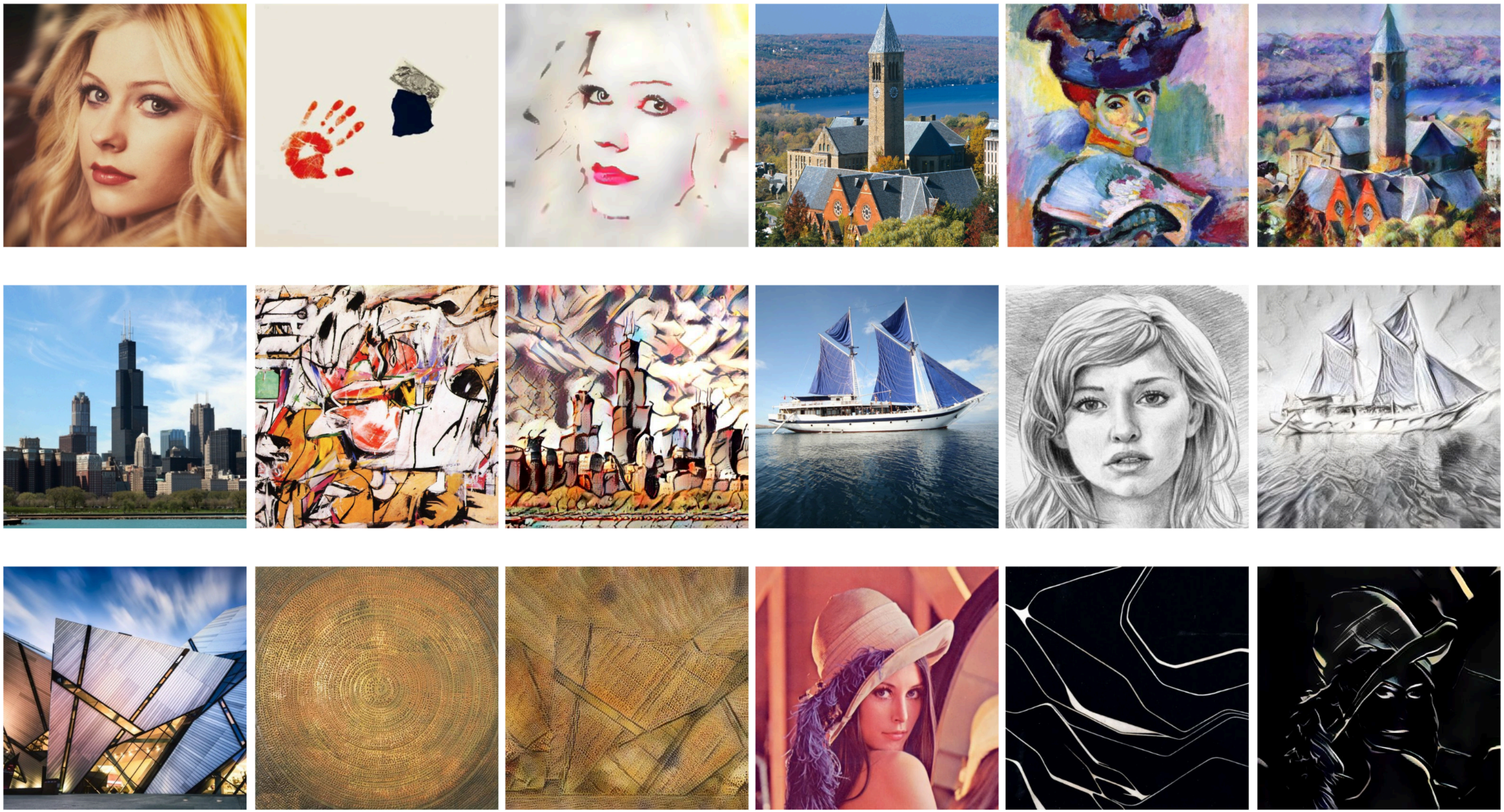
Adaptive Instance Normalization

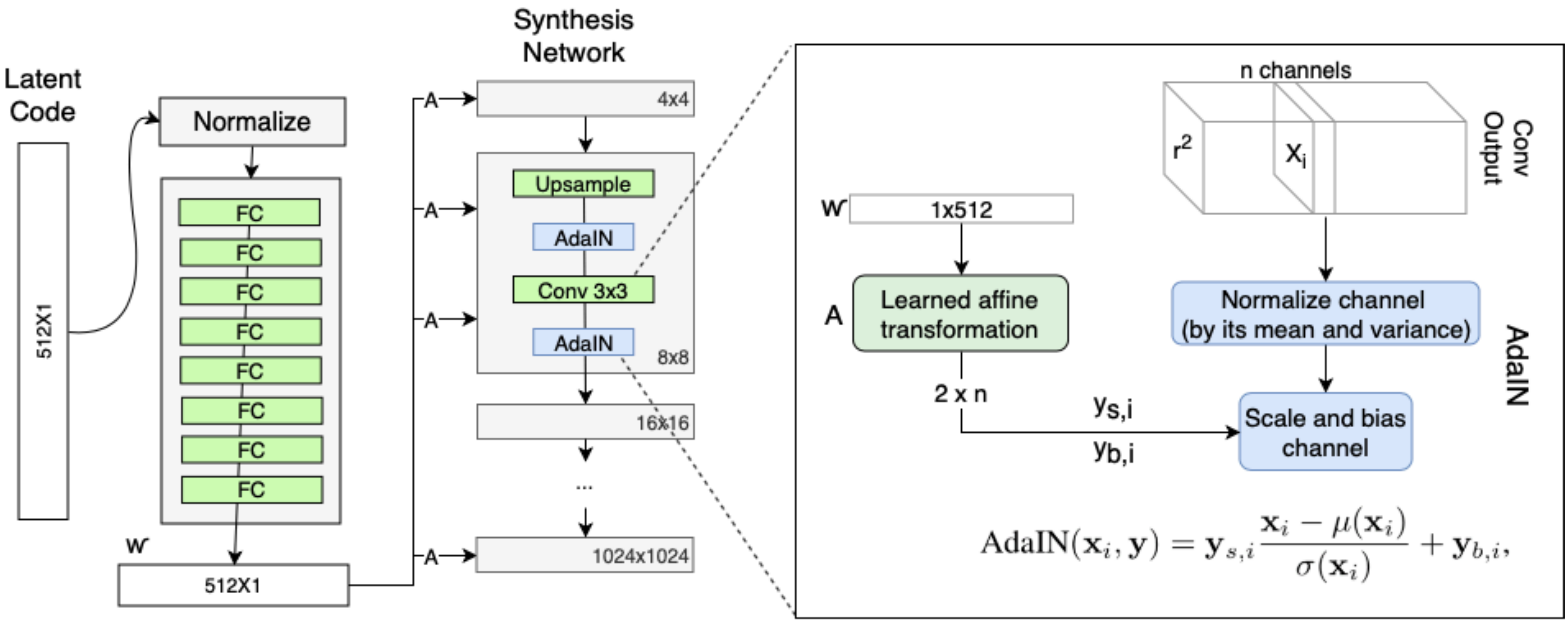
Batch normalization reminder

Ensure the output statistics of a layer are fixed.

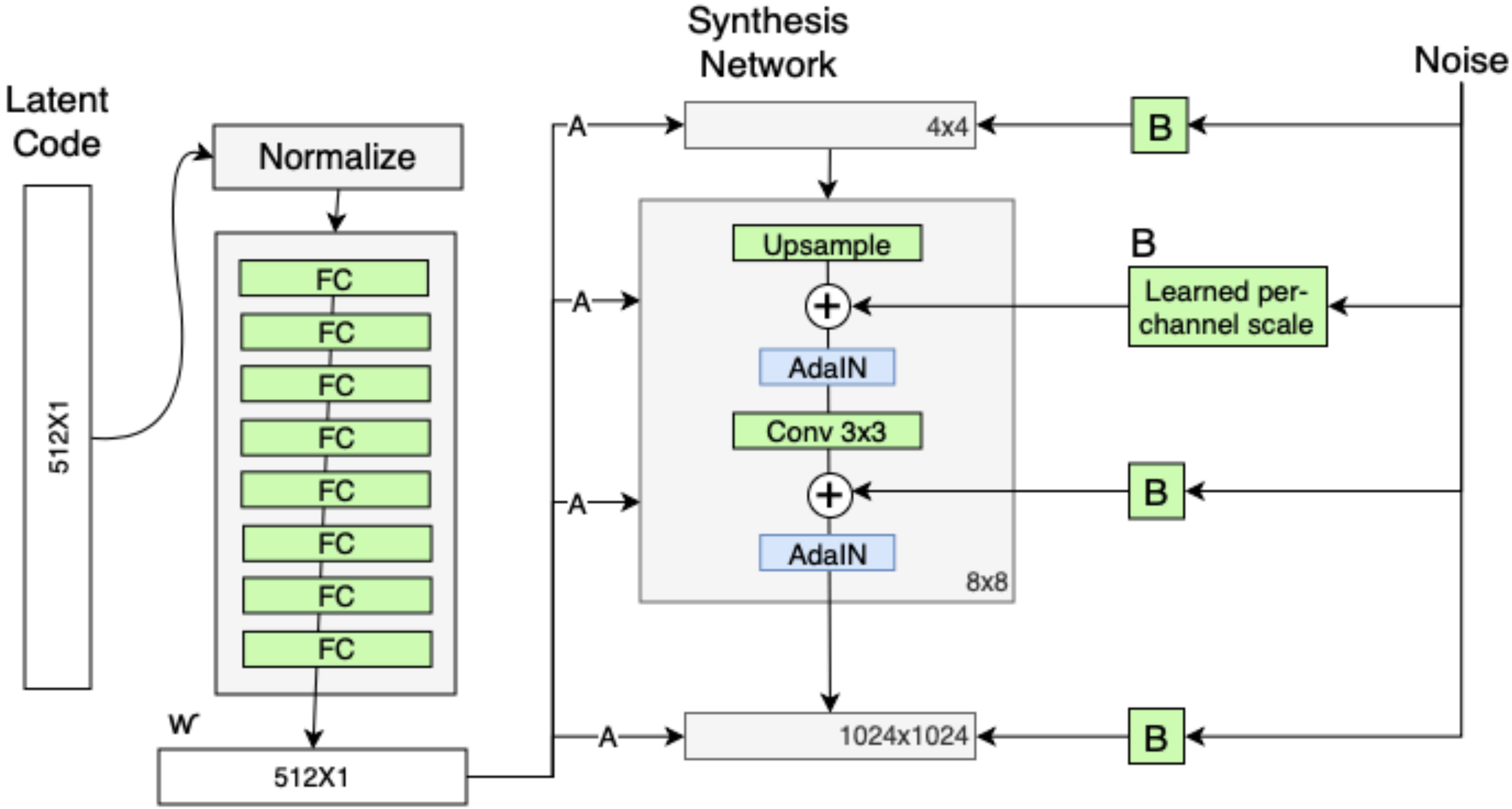


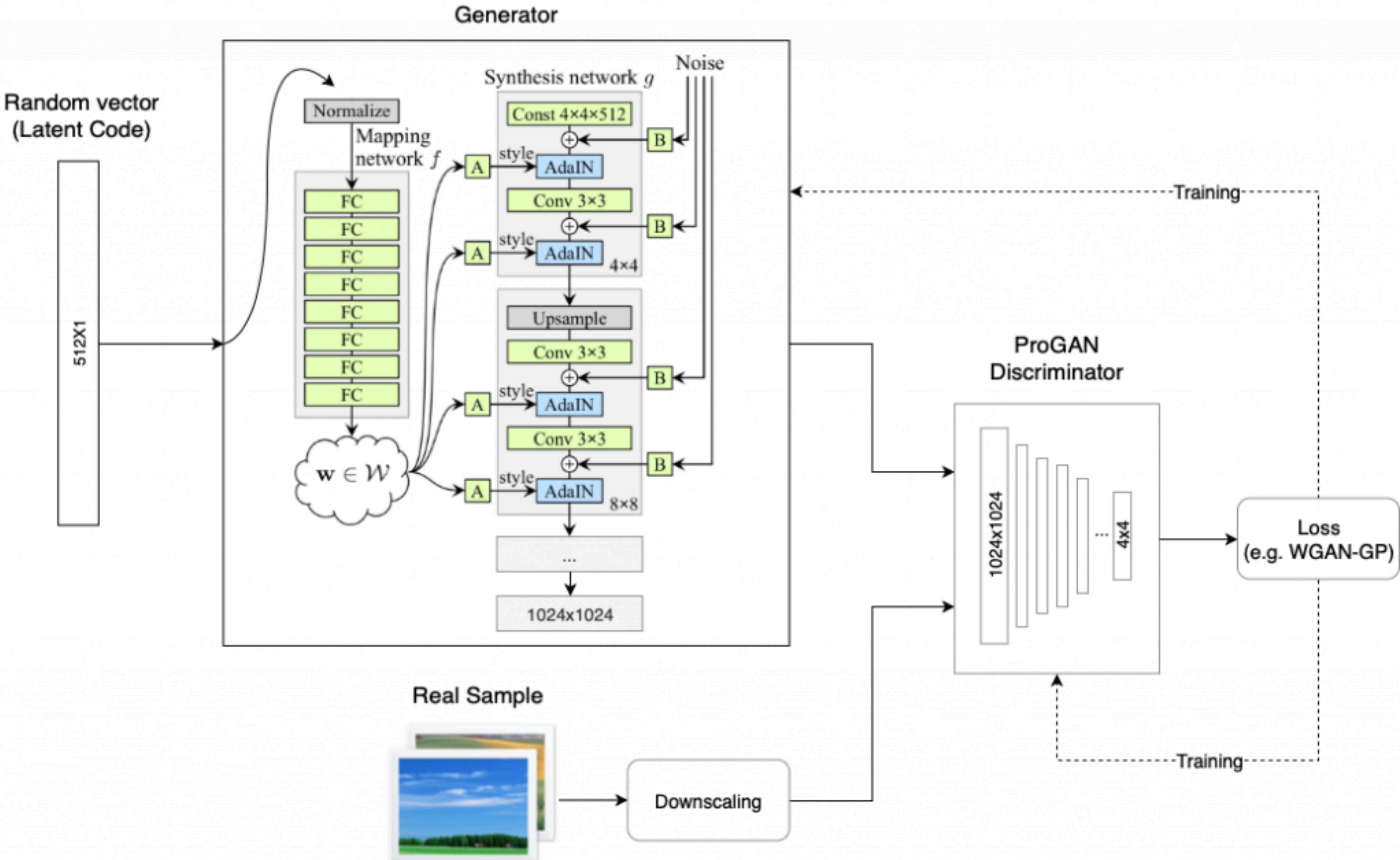
Style transfer





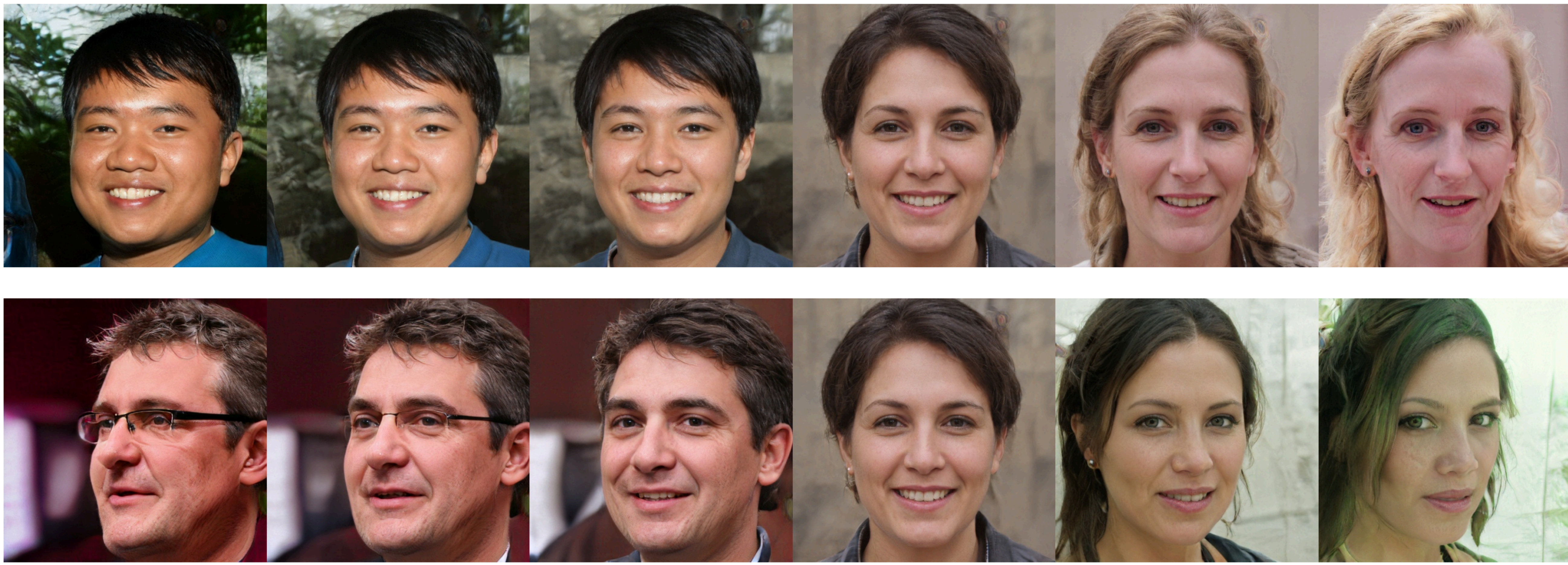
Latent Code
512X1
Normalize
FC
FC
FC
FC
FC
FC
FC
FC
w
512X1
Synthesis Network
A
4x4
A
Upsample
AdaIN
A
Conv 3x3
AdaIN
8x8
16x16
...
A
1024x1024
n channels
r2
Xi
Conv Output
w
1x512
A
Learned affine transformation
2 x n
ys,i
yb,i
Normalize channel (by its mean and variance)
Scale and bias channel
AdaIN
$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i}\frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$


Latent
Code
512X1
Normalize
FC
FC
FC
FC
FC
FC
FC
FC
w
512X1
Synthesis
Network
A
A
A
A
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
B
style
A
AdaIN
Conv 3×3
B
style
A
AdaIN
4×4
Upsample
Conv 3×3
B
style
A
AdaIN
Conv 3×3
B
style
A
AdaIN
8×8
...
1024x1024
ProGAN
Discriminator
1024x1024
...
4x4
Training
Loss
(e.g. WGAN-GP)
Training
Real Sample
Downscaling

Truncation trick in W

(it’s shrinking, really)



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

StyleGAN2
2019 December

(very briefly)

Getting rid of progressive growing: Multi-Scale Gradients