



LG전자 Deep Learning 과정

Recent Advances in GANs

Gunhee Kim

Computer Science and Engineering



서울대학교
SEOUL NATIONAL UNIVERSITY

Image Synthesis



PGGAN [Brock et al., ICLR 2019]

StyleGAN [Karras et al., CVPR 2019]

[1] Brock et al., Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019

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

Image Synthesis with Generative Models

Generative models are trained to learn an image distribution p_{data}

- VAE maximizes lower bound of $E_{p_{data}}[\log p_\theta(x)]$
- GANs minimize an estimate of some divergence metric between $p_{data}(x)$ and $p_\theta(x)$

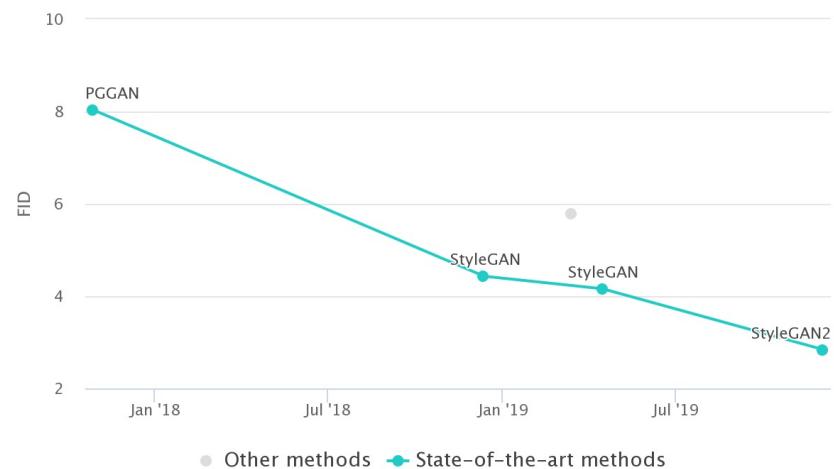
Most of the modern image synthesis pipelines are based on GANs

- Samples from GANs are quite ‘crisp’ while samples from VAEs are usually ‘blurry’
- This sample quality gap is becoming smaller (e.g. VQ-VAE v2)
- But in this talk, we will mainly focus on GAN-based image synthesis

Status Quo of Image Synthesis on FFHQ



StyleGAN v2 [Karras et al., CVPR 2019]



FID on FFHQ [2]

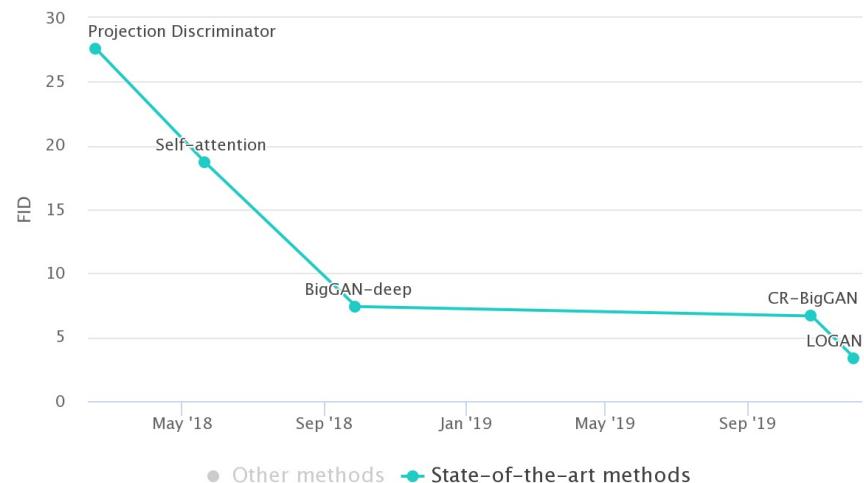
[1] Karras et al. Analyzing and Improving the Image Quality of StyleGAN, CVPR2019

[2] Flickr-Faces-HQ Dataset (FFHQ): <https://paperswithcode.com/sota/image-generation-on-ffhq>

Status Quo of Image Synthesis on ImageNet



LOGAN [Wu et al., 2019]



FID on ImageNet 128x128 [2]

[1] Wu et al. LOGAN: Latent Optimisation for Generative Adversarial Networks, ICLR 2019 (rejected)

[2] ImageNet 128x128 dataset: <https://paperswithcode.com/sota/conditional-image-generation-on-imagenet>

Challenges

GAN does a great job for producing high-fidelity samples

However, its training is notoriously difficult

- Training often collapses even on a simple toy dataset
- Empirically, this instability becomes larger with the size of model

To synthesize photorealistic images, we need

- A model that has enough capacity and right inductive bias to transform $p(z)$ to $p_{data}(x)$
- And a proper way to deal with training instability that comes with the scale

Progressive Growing GAN (PGGAN)

Goal: a new training method to produce images of unprecedented quality

Idea: propose to grow image resolution as training progresses

First photo-realistic samples (1024x1024) on CelebA-HQ

- vs. CIFAR-10 (32x32) and ImageNet (128x128)

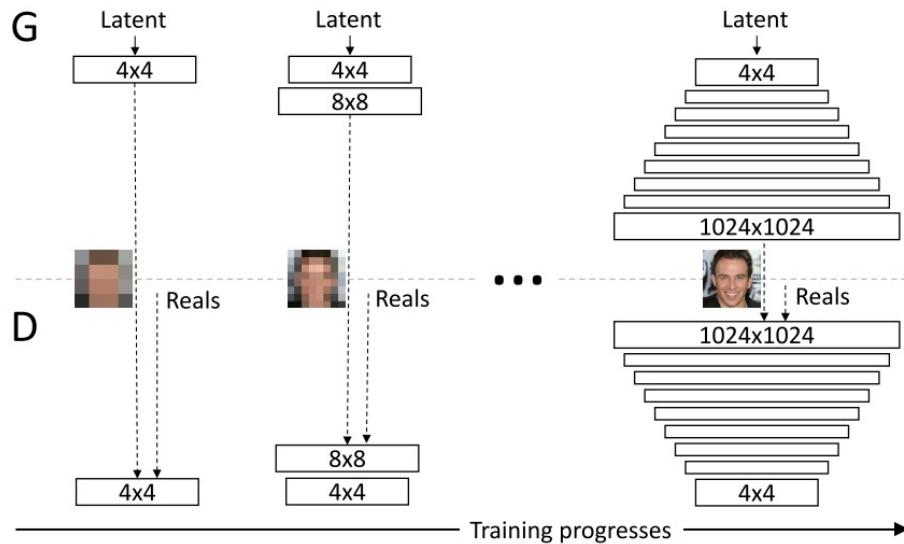


Progressive Growing GAN (PGGAN)

Issues of generation of high-resolution images

- It is much easier for D to capture unrealistic artifacts in high-resolution
- Necessitate using small minibatches due to memory constraint

Key idea: **Grow resolution in a coarse-to-fine manner** as training progresses



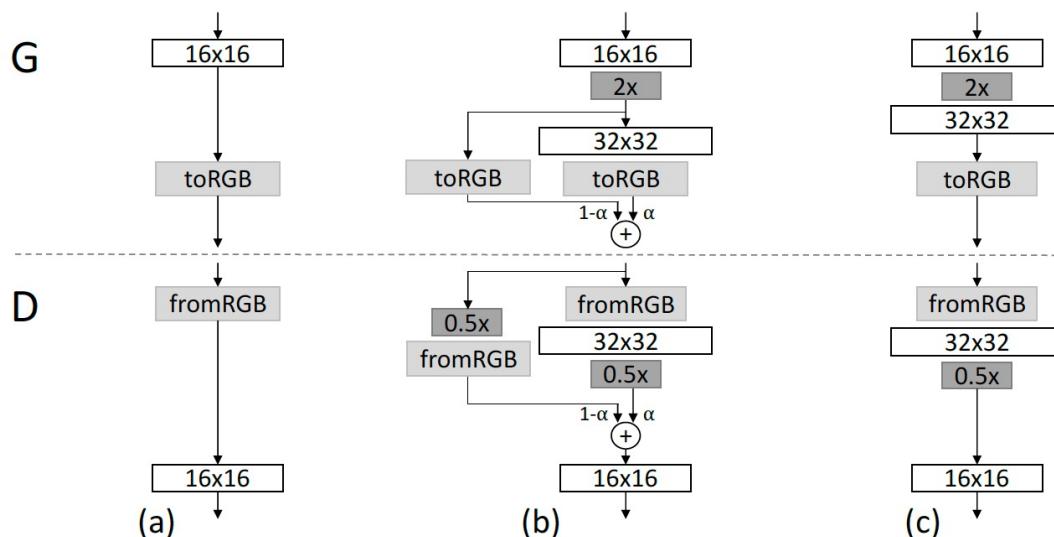
Progressive training

- Incrementally add layers to G and D
- All layers are trainable throughout the training
- Training is faster up to 2~6 times

PGGAN: How to Add a New Layer

Doubling the layer smoothly (a) → (b) → (c)

- The weight α increases linearly from 0 to 1
- The parameters in the 32x32 layer are gradually learned from 0 to 1



- $16 \times 16/32 \times 32$: output layer (2 CONV + leaky ReLU)
- toRGB/fromRGB: the layer that projects feature vectors to RGB colors vice versa
- 2x: nearest neighbor interpolation
- 0.5x: average pooling

PGGAN: Other Heuristics for Stable Training

Neither learnable parameters nor hyperparameters

1. Minibatch standard deviation

- Goal: Increase the variation of output samples
- Add one additional constant feature map at the end of the discriminator
- Compute a single number from the standard deviation of all features in a batch across spatial locations

PGGAN: Other Heuristics for Stable Training

2. Equalized learning rate

- Goal: Keep the weights in each network at a similar scale during the training
- Scale the weights at each layer with a constant (i.e. per-layer normalization constant)

3. Pixel-wise normalization

- Goal: Keep the magnitude of the feature map within a certain range
- Normalize the feature vector in each pixel to a unit length in the generator after each convolutional layer

PGGAN: Results (128x128) – Metrics

Training configuration	CELEBA					MS-SSIM	LSUN BEDROOM					MS-SSIM		
	Sliced Wasserstein distance $\times 10^3$						Sliced Wasserstein distance $\times 10^3$							
	128	64	32	16	Avg		128	64	32	16	Avg			
(a) Gulrajani et al. (2017)	12.99	7.79	7.62	8.73	9.28	0.2854	11.97	10.51	8.03	14.48	11.25	0.0587		
(b) + Progressive growing	4.62	2.64	3.78	6.06	4.28	0.2838	7.09	6.27	7.40	9.64	7.60	0.0615		
(c) + Small minibatch	75.42	41.33	41.62	26.57	46.23	0.4065	72.73	40.16	42.75	42.46	49.52	0.1061		
(d) + Revised training parameters	9.20	6.53	4.71	11.84	8.07	0.3027	7.39	5.51	3.65	9.63	6.54	0.0662		
(e*) + Minibatch discrimination	10.76	6.28	6.04	16.29	9.84	0.3057	10.29	6.22	5.32	11.88	8.43	0.0648		
(e) Minibatch stddev	13.94	5.67	2.82	5.71	7.04	0.2950	7.77	5.23	3.27	9.64	6.48	0.0671		
(f) + Equalized learning rate	4.42	3.28	2.32	7.52	4.39	0.2902	3.61	3.32	2.71	6.44	4.02	0.0668		
(g) + Pixelwise normalization	4.06	3.04	2.02	5.13	3.56	0.2845	3.89	3.05	3.24	5.87	4.01	0.0640		
(h) Converged	2.42	2.17	2.24	4.99	2.96	0.2828	3.47	2.60	2.30	4.87	3.31	0.0636		

- SWD (Sliced Wasserstein Distance) vs. MS-SSIM (multi-scale structural similarity)
- Existing metric (MS-SSIM) measures only the variation between outputs not similarity to training set
- Propose SWD: building Laplacian pyramid and randomly select patches from multiple layers and compute the Wasserstein distance between training and generated images
- Numbers: Levels of Laplacian pyramid

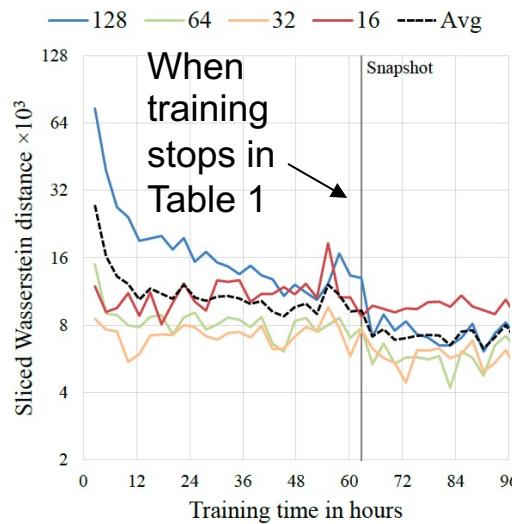
PGGAN: Results (128x128) – Variants

Training configuration	CELEBA					MS-SSIM	LSUN BEDROOM					MS-SSIM		
	Sliced Wasserstein distance $\times 10^3$						Sliced Wasserstein distance $\times 10^3$							
	128	64	32	16	Avg		128	64	32	16	Avg			
(a) Gulrajani et al. (2017)	12.99	7.79	7.62	8.73	9.28	0.2854	11.97	10.51	8.03	14.48	11.25	0.0587		
(b) + Progressive growing	4.62	2.64	3.78	6.06	4.28	0.2838	7.09	6.27	7.40	9.64	7.60	0.0615		
(c) + Small minibatch	75.42	41.33	41.62	26.57	46.23	0.4065	72.73	40.16	42.75	42.46	49.52	0.1061		
(d) + Revised training parameters	9.20	6.53	4.71	11.84	8.07	0.3027	7.39	5.51	3.65	9.63	6.54	0.0662		
(e*) + Minibatch discrimination	10.76	6.28	6.04	16.29	9.84	0.3057	10.29	6.22	5.32	11.88	8.43	0.0648		
(e) Minibatch stddev	13.94	5.67	2.82	5.71	7.04	0.2950	7.77	5.23	3.27	9.64	6.48	0.0671		
(f) + Equalized learning rate	4.42	3.28	2.32	7.52	4.39	0.2902	3.61	3.32	2.71	6.44	4.02	0.0668		
(g) + Pixelwise normalization	4.06	3.04	2.02	5.13	3.56	0.2845	3.89	3.05	3.24	5.87	4.01	0.0640		
(h) Converged	2.42	2.17	2.24	4.99	2.96	0.2828	3.47	2.60	2.30	4.87	3.31	0.0636		

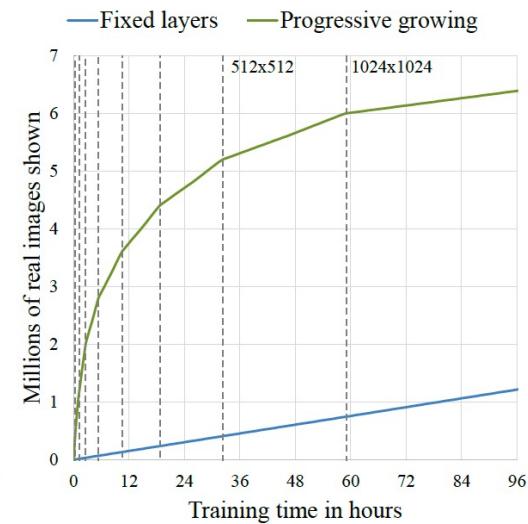
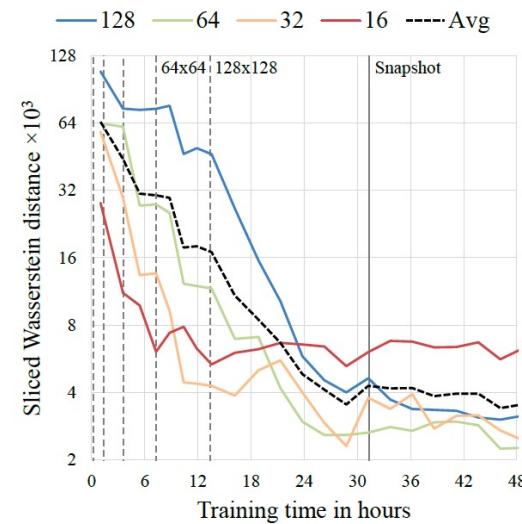
- (c) decrease minibatch size from 64 to 16
- (d) Adjust the hyperparameters as well as by removing batch/layer normalization
- (e)(f)(g): the proposed heuristics
- Non-converged: terminating the training once the D has been shown a total of 10M real images

PGGAN: Speed and Convergence

Wasserstein GANs



Progressive growing enabled



On a single GPU (NVIDIA Tesla P100)

Raw training speed
with 1024x1024
(dashed: double the resolution of G and D)

PGGAN: Summary

It's much easier for D to capture unrealistic artifacts in high-resolution

This leads to overwhelming D , which contributes unstable training

Progressive growing both stabilizes and accelerates the training

BigGAN

Successfully train a large-scale GAN with some tricks

- A simple sampling technique (truncation trick) allows fine-grained control of the trade-off between sample variety and fidelity

GANs benefit dramatically from scaling

- Train models with 2-4x as many parameters and 8x batch size compared to prior art
- Huge improvement on class-conditional ImageNet compared to StyleGAN: IS $52.52 \rightarrow 166.3$ ($\times 3.19$) and FID $18.65 \rightarrow 9.6$ ($\times 1.94$)



BigGAN: Scaling Up GANs

Use SA-GAN architecture as a baseline

- Both increased model / batch size improve FID and IS

Using skip connections from the latent z to multiple layers of G

Stable to 10^6 iterations, or it collapses at the given iteration

Batch size	# units per layer	Shared embedding	Orthogonal Regularization	Itr $\times 10^3$	FID	IS
256	64	81.5	SA-GAN Baseline	1000	18.65	52.52
512	64	81.5	✗	1000	15.30	58.77(± 1.18)
1024	64	81.5	✗	1000	14.88	63.03(± 1.42)
2048	64	81.5	✗	732	12.39	76.85(± 3.83)
2048	96	173.5	✗	295(± 18)	9.54(± 0.62)	92.98(± 4.27)
2048	96	160.6	✓	185(± 11)	9.18(± 0.13)	94.94(± 1.32)
2048	96	158.3	✓	152(± 7)	8.73(± 0.45)	98.76(± 2.84)
2048	96	158.3	✓	165(± 13)	8.51(± 0.32)	99.31(± 2.10)
2048	64	71.3	✓	371(± 7)	10.48(± 0.10)	86.90(± 0.61)

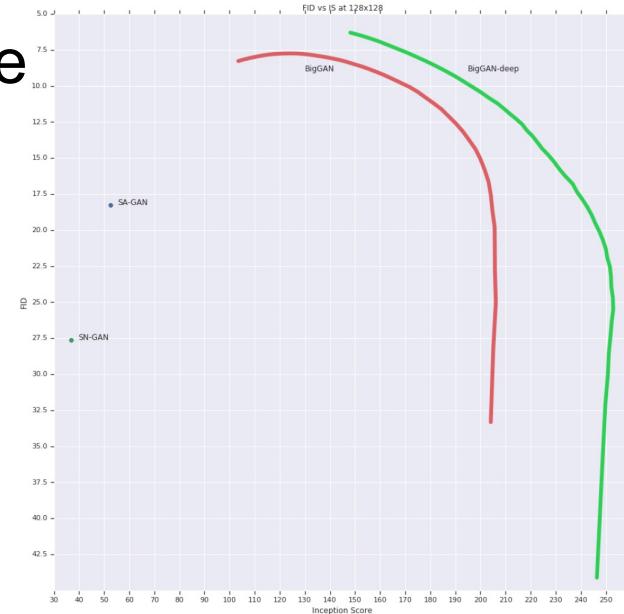
BigGAN: Scaling Up GANs

Truncating elements of z during the test time gives a huge improvement on IS

- The model is trained with $z \sim N(0, I)$ and samples z at test time so that the value that falls outside a range is resampled to be inside that range

Trade-off between FID and IS can be adjusted with the truncation level

- Frechet Inception Distance (FID) penalizes lack of diversity but Inception Score (IS) does not
- Reducing the truncation threshold diminishes variety
- A smaller threshold leads to a higher IS but a lower FID



BigGAN-deep:
4x layers

BigGAN: Results

ImageNet

Model	Res.	w/o Truncation	At the best FID setting	With validation data	At the best IS setting
		FID/IS	(min FID) / IS	FID / (valid IS)	FID / (max IS)
SN-GAN	128	27.62/36.80	N/A	N/A	N/A
SA-GAN	128	18.65/52.52	N/A	N/A	N/A
BigGAN	128	8.7 ± .6/98.8 ± 3	7.7 ± .2/126.5 ± 0	9.6 ± .4/166.3 ± 1	25 ± 2/206 ± 2
BigGAN	256	8.7 ± .1/142.3 ± 2	7.7 ± .1/178.0 ± 5	9.3 ± .3/233.1 ± 1	25 ± 5/291 ± 4
BigGAN	512	8.1/144.2	7.6/170.3	11.8/241.4	27.0/275
BigGAN-deep	128	5.7 ± .3/124.5 ± 2	6.3 ± .3/148.1 ± 4	7.4 ± .6/166.5 ± 1	25 ± 2/253 ± 11
BigGAN-deep	256	6.9 ± .2/171.4 ± 2	7.0 ± .1/202.6 ± 2	8.1 ± .1/232.5 ± 2	27 ± 8/317 ± 6
BigGAN-deep	512	7.5/152.8	7.7/181.4	11.5/241.5	39.7/298

JFT-300 (Google Internal dataset with 18K categories)

Ch.	Param (M)	Shared	Skip- z	Ortho.	FID	IS	(min FID) / IS	FID / (max IS)
64	317.1	✗	✗	✗	48.38	23.27	48.6/23.1	49.1/23.9
64	99.4	✓	✓	✓	23.48	24.78	22.4/21.0	60.9/35.8
96	207.9	✓	✓	✓	18.84	27.86	17.1/23.3	51.6/38.1
128	355.7	✓	✓	✓	13.75	30.61	13.0/28.0	46.2/47.8

BigGAN: Summary

ResNet architecture is powerful enough to achieve SOTA on ImageNet

Both increased model / batch size improves FID and IS

Truncation trick gives a huge improvement on IS

StyleGAN

Proposes a new generator architecture inspired from style transfer literature

- Use the AdaIN (Adaptive Instance Normalization) operations of [2]

Achieves SOTA on CelebA-HQ and FFHQ (Flickr-Faces-HQ, proposed)

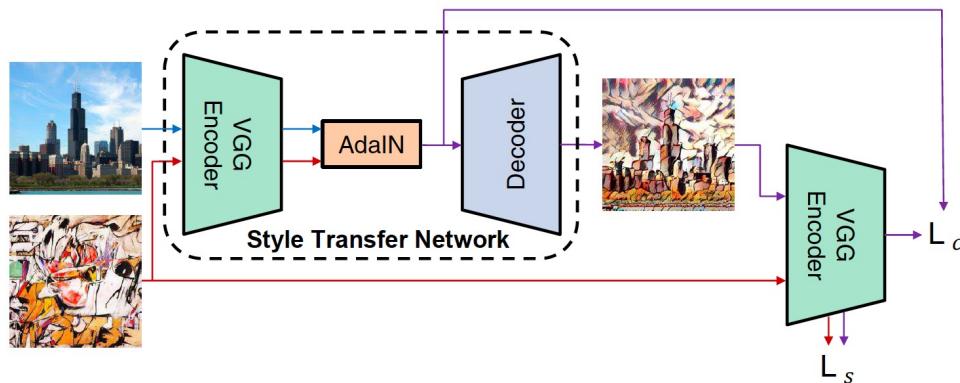


StyleGAN: Review of AdaIN

Adaptive Instance Normalization (AdaIN)

- Receive a content input x and a style input y
- Affine transformation (with no learnable parameter) to align the channel-wise mean and variance of x to match those of y

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



Encoder: first few layers of a fixed VGG-19

Decoder: mirrors the encoder, with all pooling layers replaced by nearest up-sampling to reduce checkerboard effects

L_c : content loss, L_s : style loss

StyleGAN: A New Generator Architecture

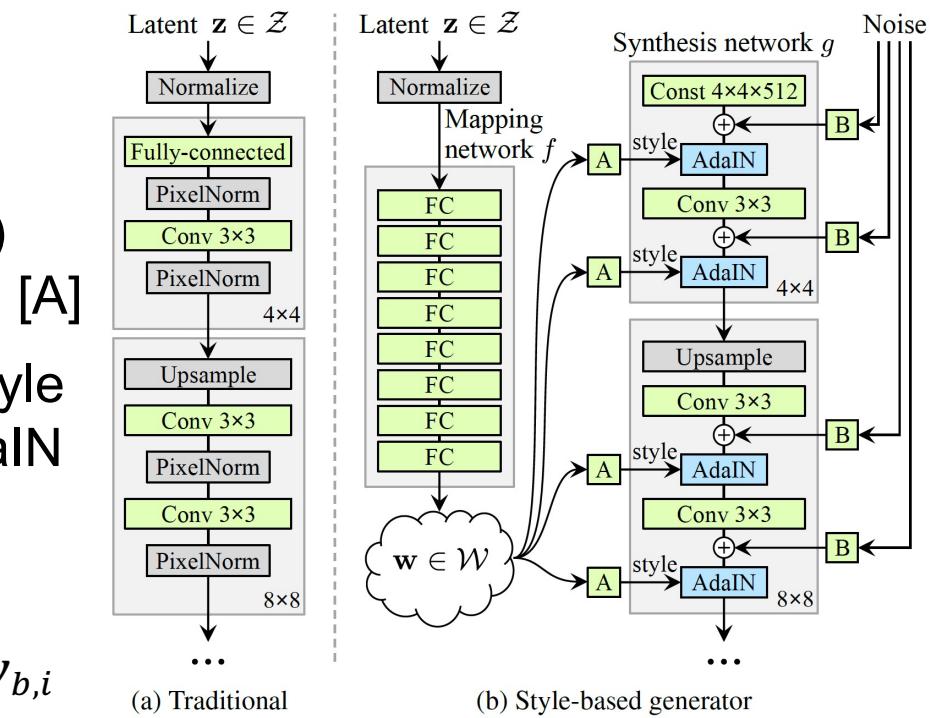
An 8 FC-layered Mapping network

- Transforms a latent code z to the embedded latent code w
- Traditionally, the latent code is provided to the generator through an input layer

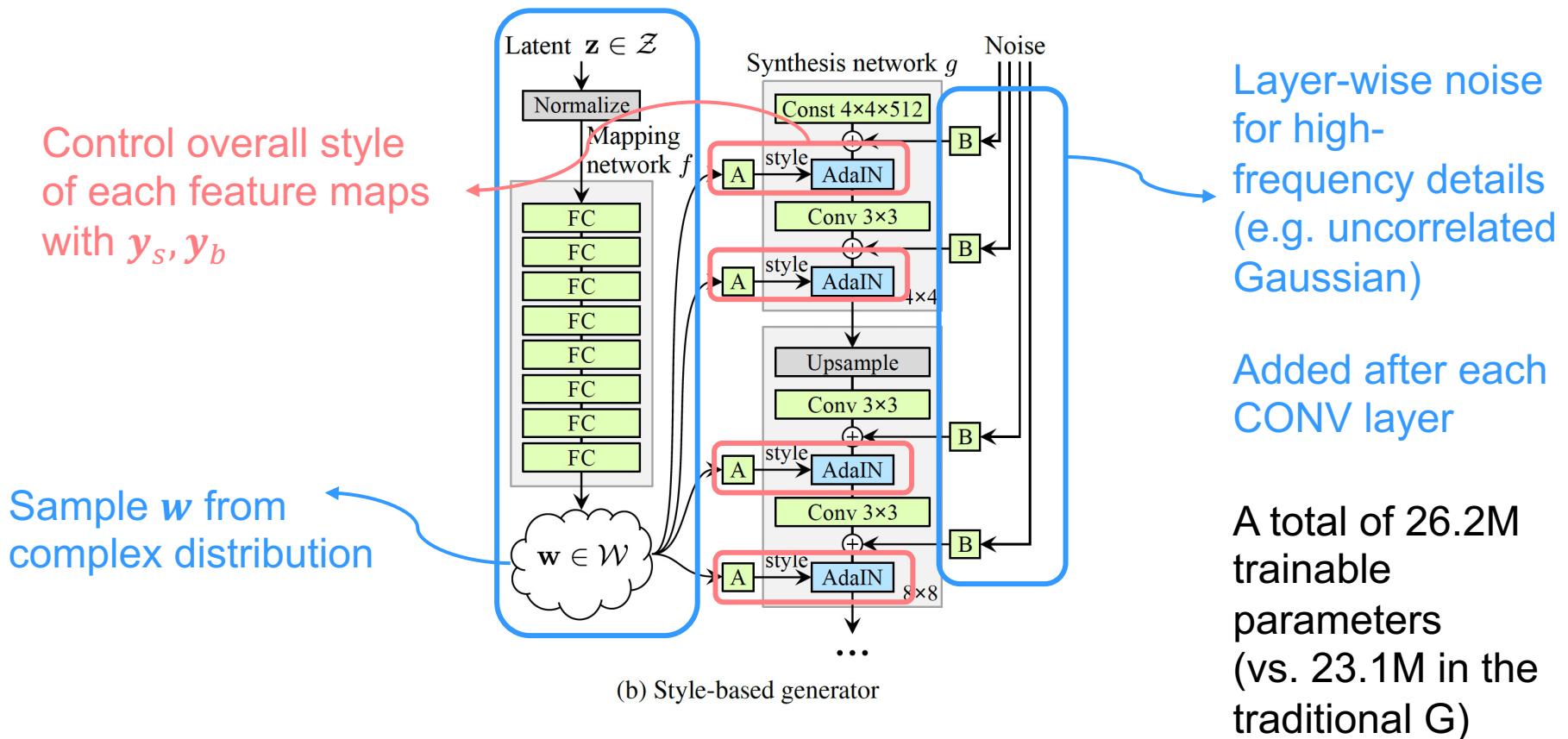
For each layer

- Latent w is transformed to layer-specific styles (y_s, y_b) via an affine transformation [A]
- Then (y_s, y_b) control the style of feature maps x_i with AdaIN operation

$$\text{AdaIN}(x_i, y) = y_{s,i} \left(\frac{x_i - \mu(x_i)}{\sigma(x_i)} \right) + y_{b,i}$$



StyleGAN: A New Generator Architecture



StyleGAN: Ablation Study

FID scores for various generator designs

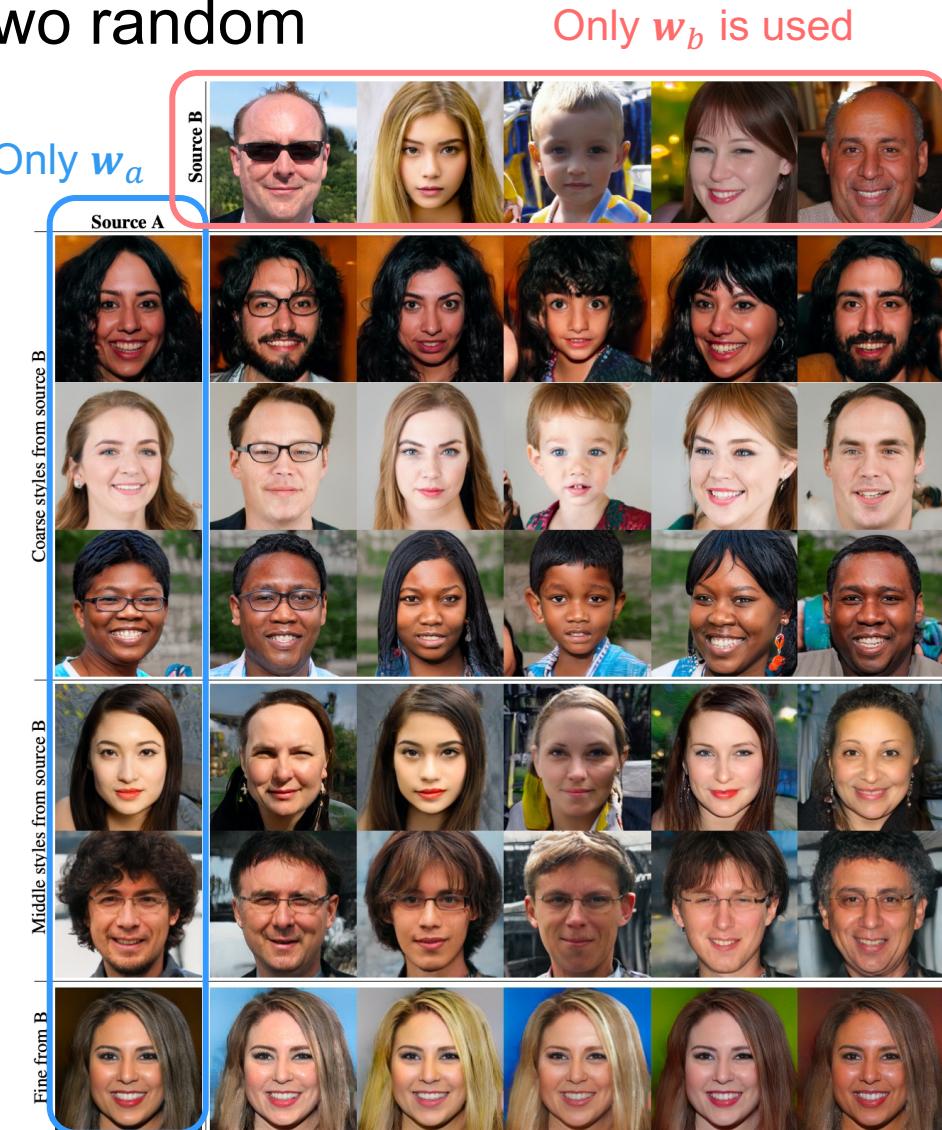
- (a) Baseline and (b) its improved version (e.g. longer training, bilinear up/down sampling)
- (c) Adding AdaIN operations
- (d) Simplifying architecture by removing the traditional input layer but starting from w
- (e) Introducing the noise input
- (f) Mixing regularization (see later)

Method	CelebA-HQ	FFHQ
A Baseline Progressive GAN [30]	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

StyleGAN: Ablation Study

Generate an image using two random latent codes

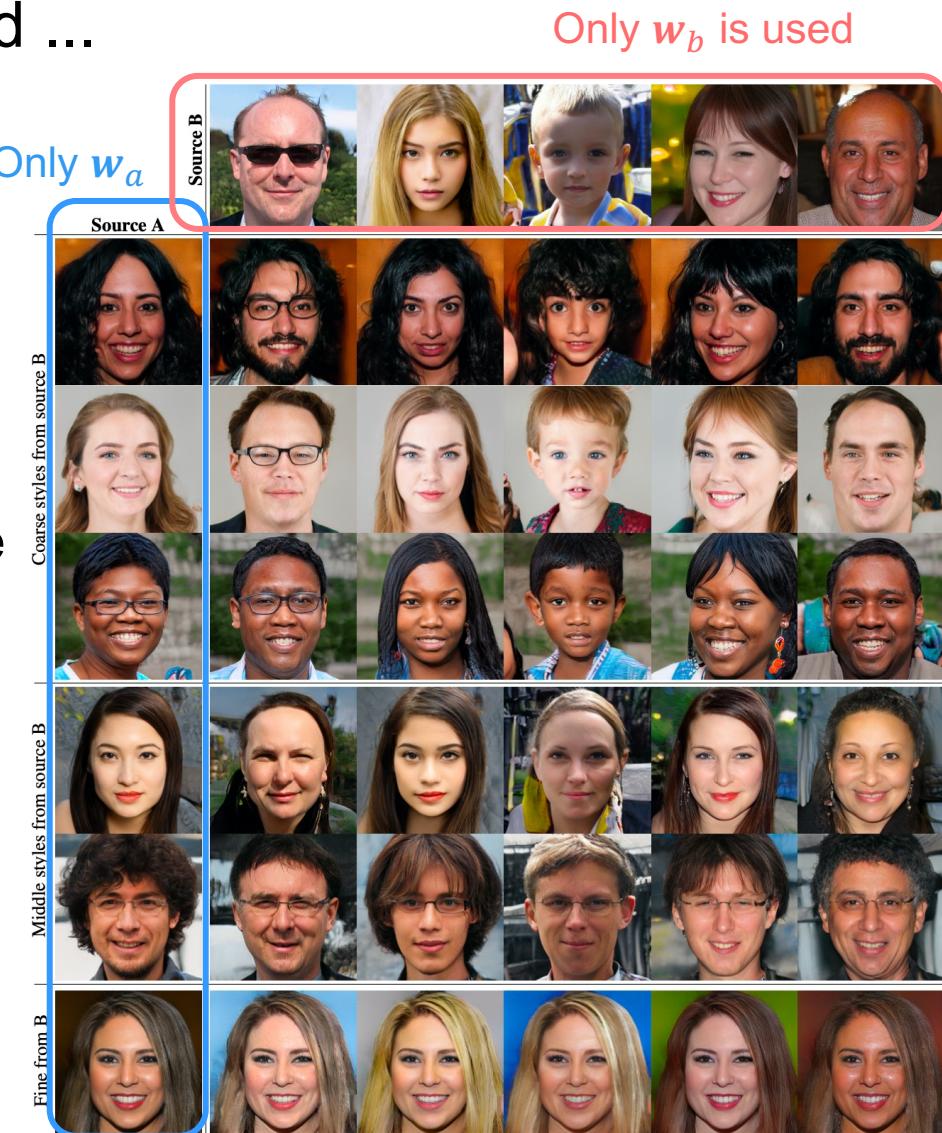
- Obtain w_a and w_b from two latent codes
- A split point in the synthesis network is chosen and w_a is used for AdaIN operations before the point while w_b is after the point



StyleGAN: Ablation Study

In the examples, w_b is used ...

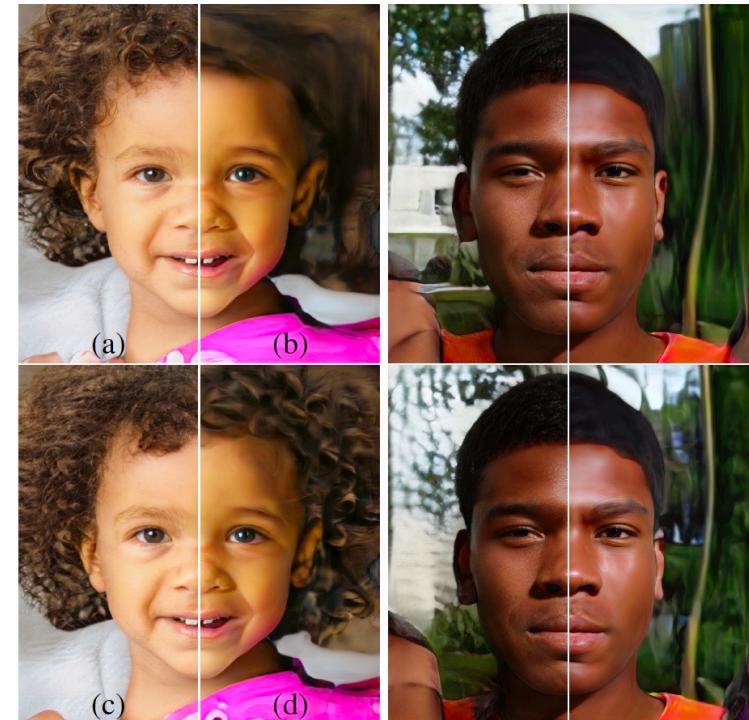
- For coarse resolution ($4^2 - 8^2$) → high-level aspects such as pose, hair style, face shape, and eyeglasses from source B
- For middle resolution ($16^2 - 32^2$) → smaller scale facial features, hair style, eyes open/closed from B, while pose, face shape, and eyeglasses from A
- For high resolution ($64^2 - 1024^2$) → B brings mainly the color scheme and microstructure



StyleGAN: The Effect of Noise

No noise leads to featureless “painterly” look

- (a) Noise is applied to all layers
- (b) No noise
- (c) Noise in fine layers
only ($64^2 - 1024^2$)
- (d) Noise in coarse layers
only ($4^2 - 32^2$)



StyleGAN: Summary

Style-based generator architecture performs better than PGGAN

But a comparison with BigGAN architecture is missing in the paper

In practice, both architectures work well!

Choose between BigGAN and StyleGAN architectures as you like

Conclusion

Scale up the model to have enough capacity and right inductive bias

Choose between BigGAN and StyleGAN architectures for high-resolution

Use self-attention layers if your target images are quite structured

Progressive training helps a lot in terms of convergence speed & stability

Use truncation trick to adjust trade-off between diversity and fidelity of samples