Analyzing and Improving the Image Quality of StyleGAN

StyleGAN v2 논문리뷰

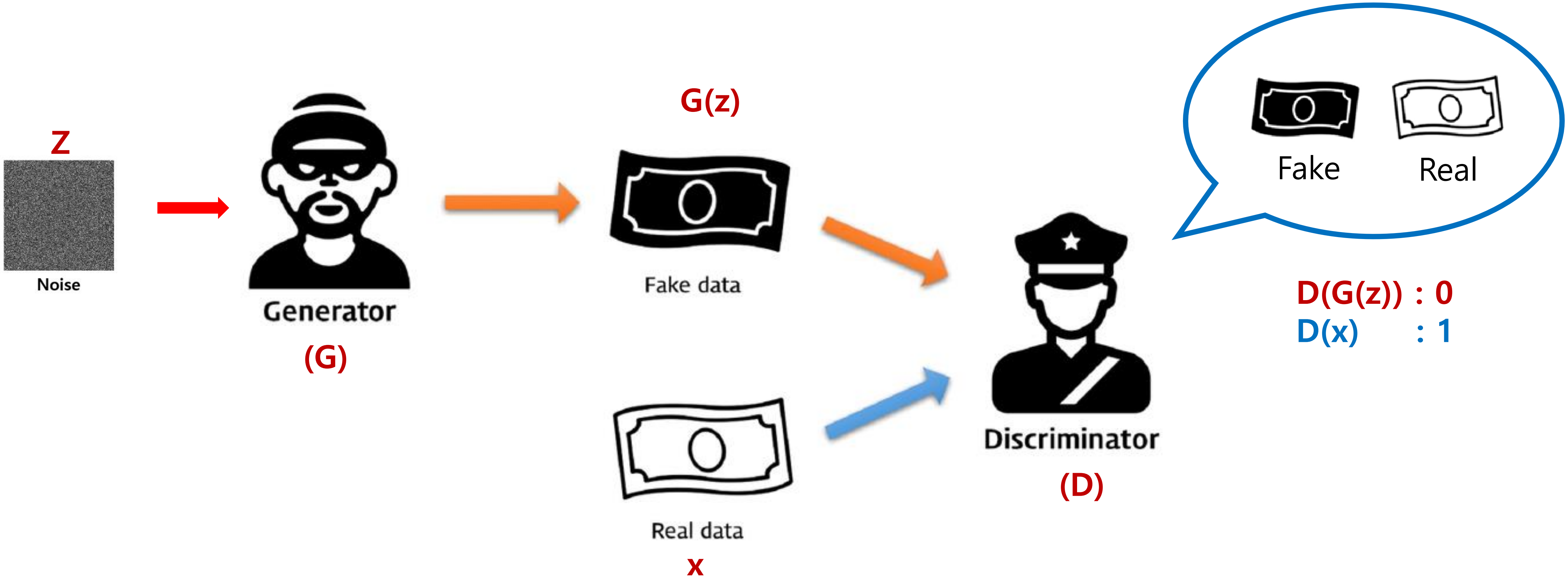
발표자 : 진아람

INDEX

Generative Adversarial Network(GAN)

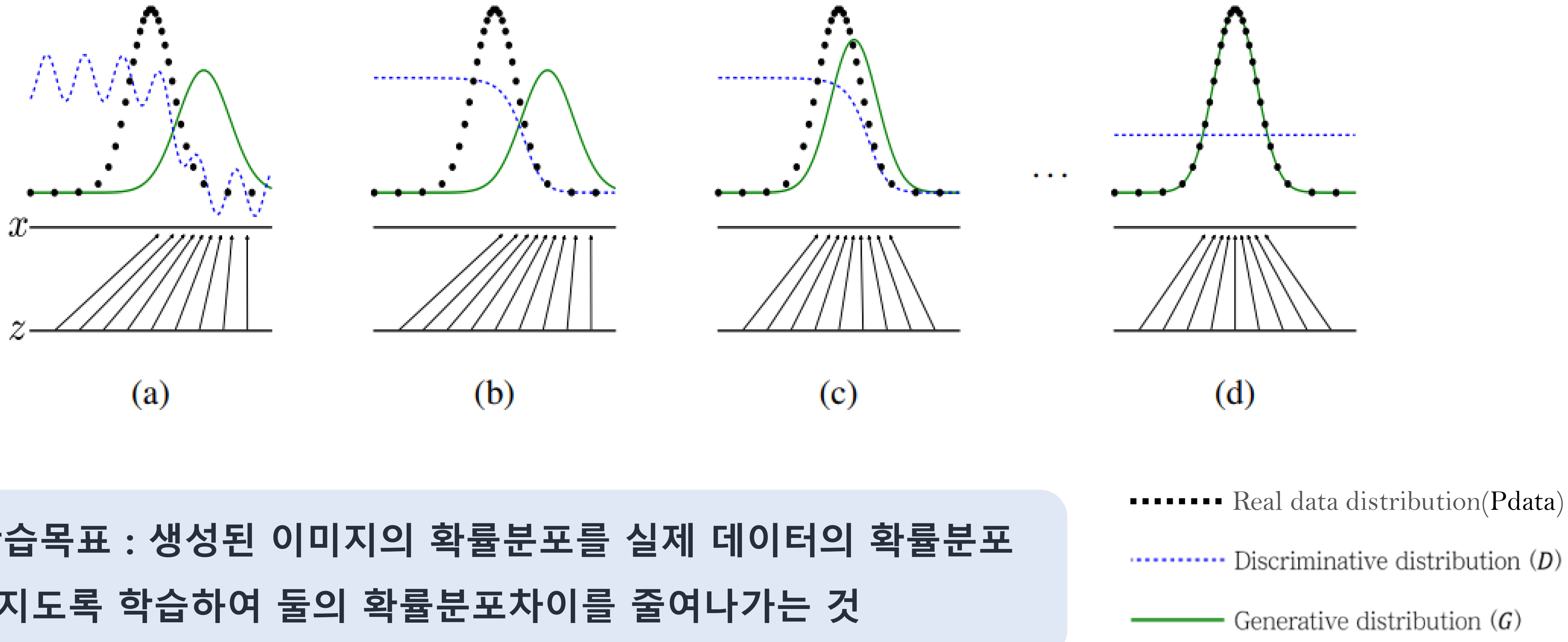
- NIPS 2014

Generative Adversarial Network (GAN)



- 생성자(Generator), 판별자(Discriminator) 네트워크를 활용한 생성 모델
- 생성자는 판별자를 활용하여 이미지 분포를 학습하며, 이미지 생성

학습과정



GAN의 학습목표 : 생성된 이미지의 확률분포를 실제 데이터의 확률분포와 근사해지도록 학습하여 둘의 확률분포차이를 줄여나가는 것

Generative Adversarial Network (GAN)

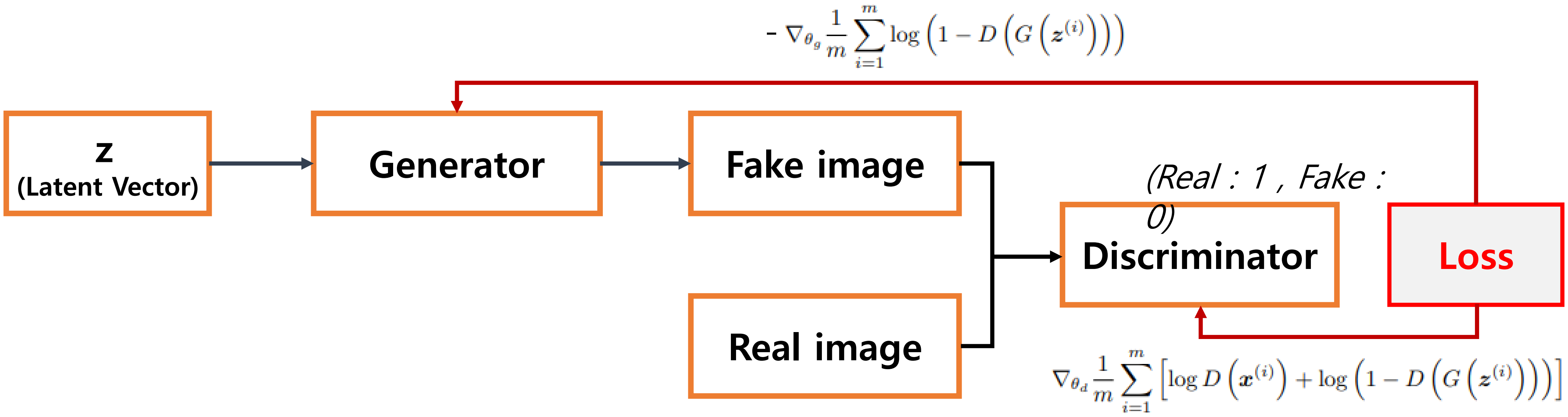
value function $V(G, D)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Generator G(z) : new data instance

Discriminator D(x) = Probability *(Real : 1 ~ Fake : 0)*

D(x)=1 일때 logD(x) = 0 (최대)
D(x)=0 일때 logD(x) = -∞ (최소)



● GAN 의 한계점

- BlackBox method

  → Fully Connected layer로 구성된 Neural Network(MLP)사용하기 때문에
  Neural Network 자체의 한계를 지님

- 고해상도 이미지를 생성하기 쉽지 않음

  → 평가기준이 명확하지 않기때문에

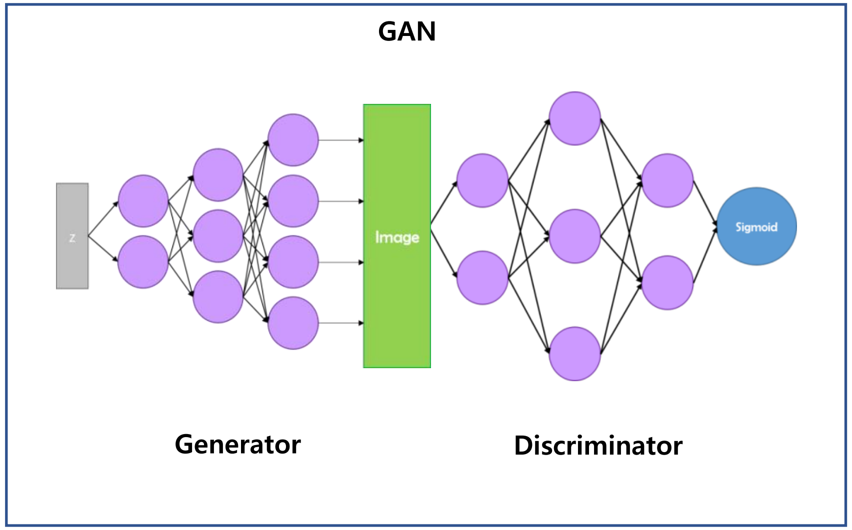
- 모델 훈련의 불안정성 (instability)

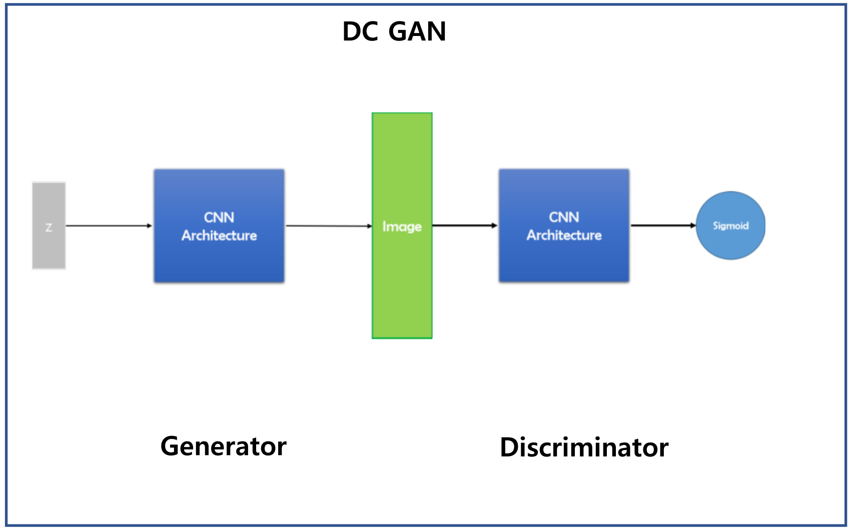
  → GAN의 구조 자체가 불안정

Deep Convolutional Generative Adversarial Network(DCGAN) – ICLR 2016

핵심 성능

- GAN의 Fully Connected Layer들을 Convolution Layer로 대체
- GAN의 불안정함을 없애고 대부분의 상황에서 안정적으로 학습이 되는 모델
- Generator가 이미지를 외워서 출력하는 것이 아니라는 것을 확인
- 기존 GAN보다 더욱 고해상도의 세밀한 이미지 생성





Deep Convolutional Generative Adversarial Network (DCGAN)

- **Architecture : Deep Convolutional Layers 이용**

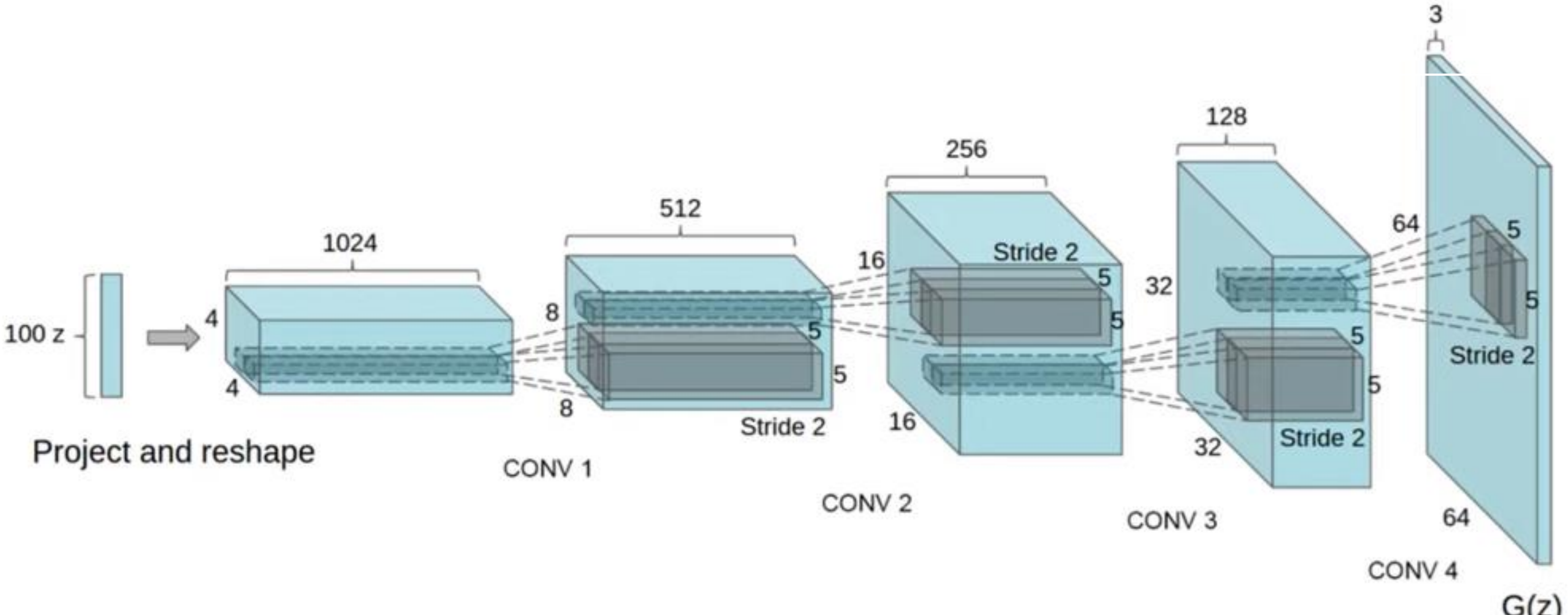


Fig. DCGAN generator used for LSUN scene modeling.

➡ **Image Domain에서 높은 성능**

- Deep Convolutional Layers

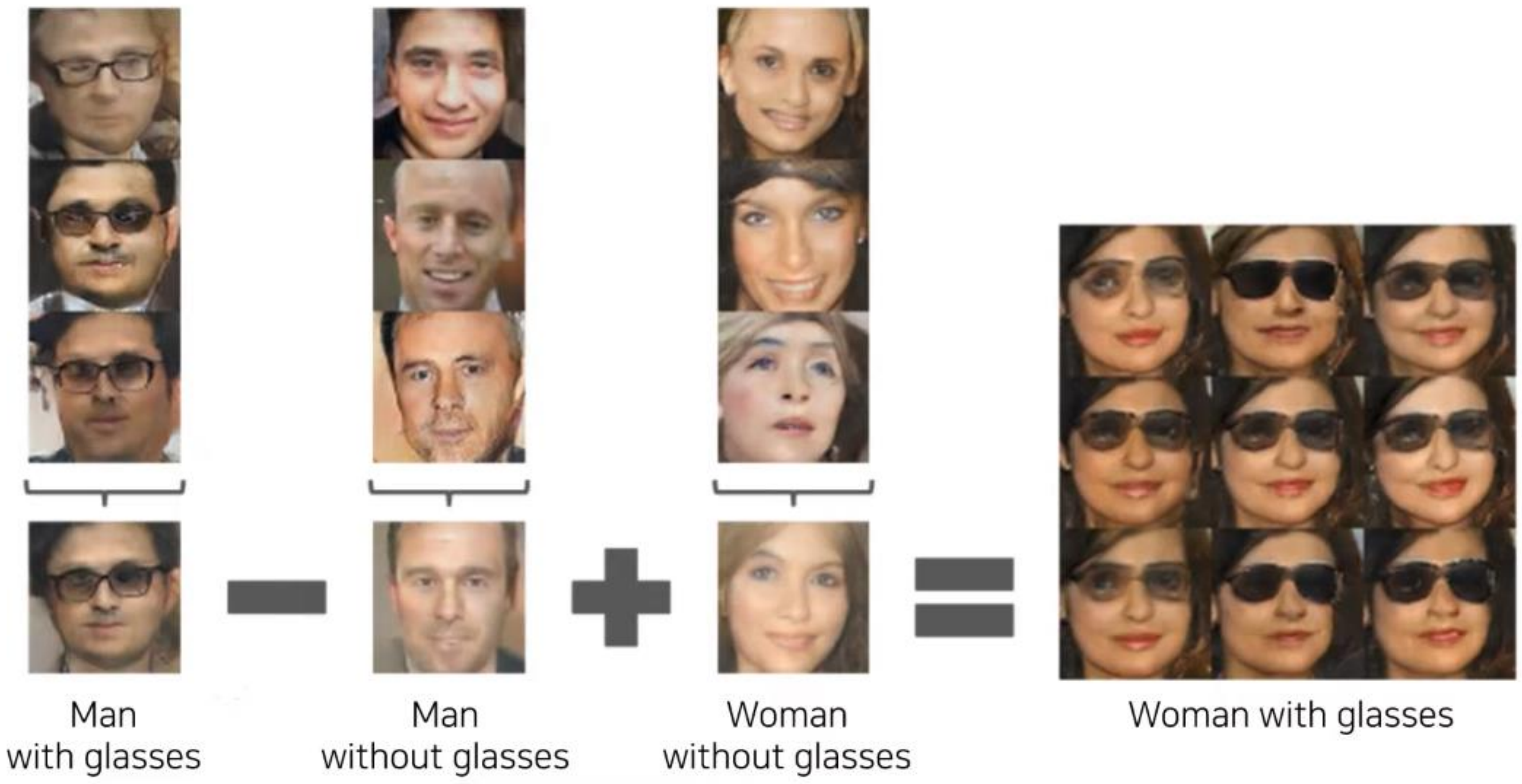
→ BlackBox method 문제해결

Random filters　　　　Trained filters

Deep Convolutional Generative Adversarial Network (DCGAN)

- 벡터 연산 지원

Latent Vactor1 Latent Vactor2 Latent Vactor3



PGGAN(Progressive Growing of GANs for Improved Quality, Stability, and Variation)

메인 아이디어

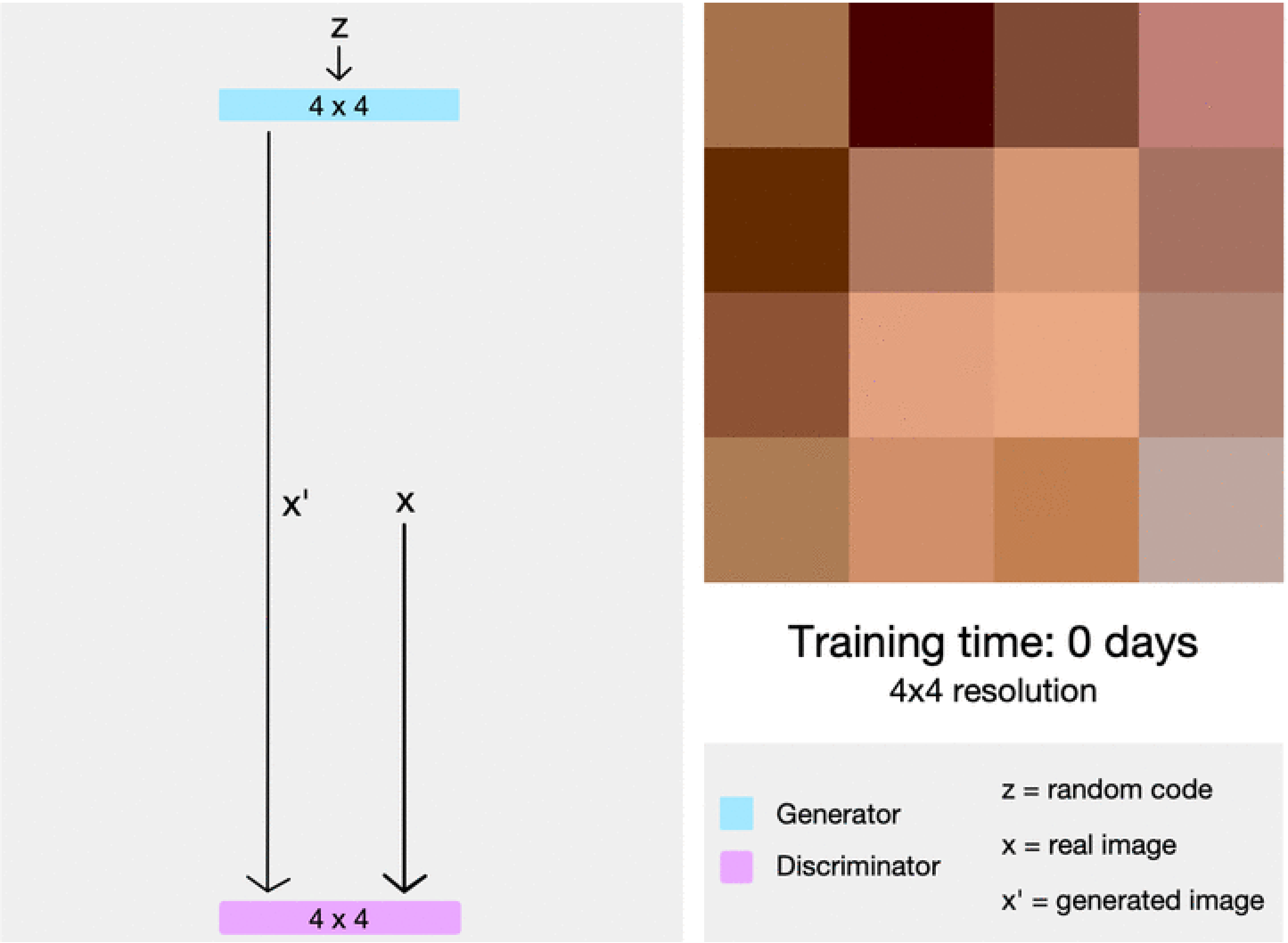
학습 과정에서 레이어를 추가

고해상도 이미지 학습 성공

한계점

이미지의 특징 제어가 어려움

StyleGAN에서 개선



핵심 성능

- Generator와 Discriminator를 점진적으로 학습시키는 것
- 훈련속도를 향상시키고 훨씬 안정화함과 동시에 가장 좋은 화질의 이미지들을 생성
- 한번에 전체 크기의 image feature들을 학습 하는 것이 아닌 4X4 저해상도로 시작
- Large-scale structure 를 찾아내고 점차 detail-scale을 찾음

  -> 1024 X 1024 고해상도로 높아지는 것에 더 고효율
- 안정성을 위해 WGAN-GP loss 사용

StyleGAN – CVPR 2019

(A Style-Based Generator Architecture for Generative Adversarial Networks)

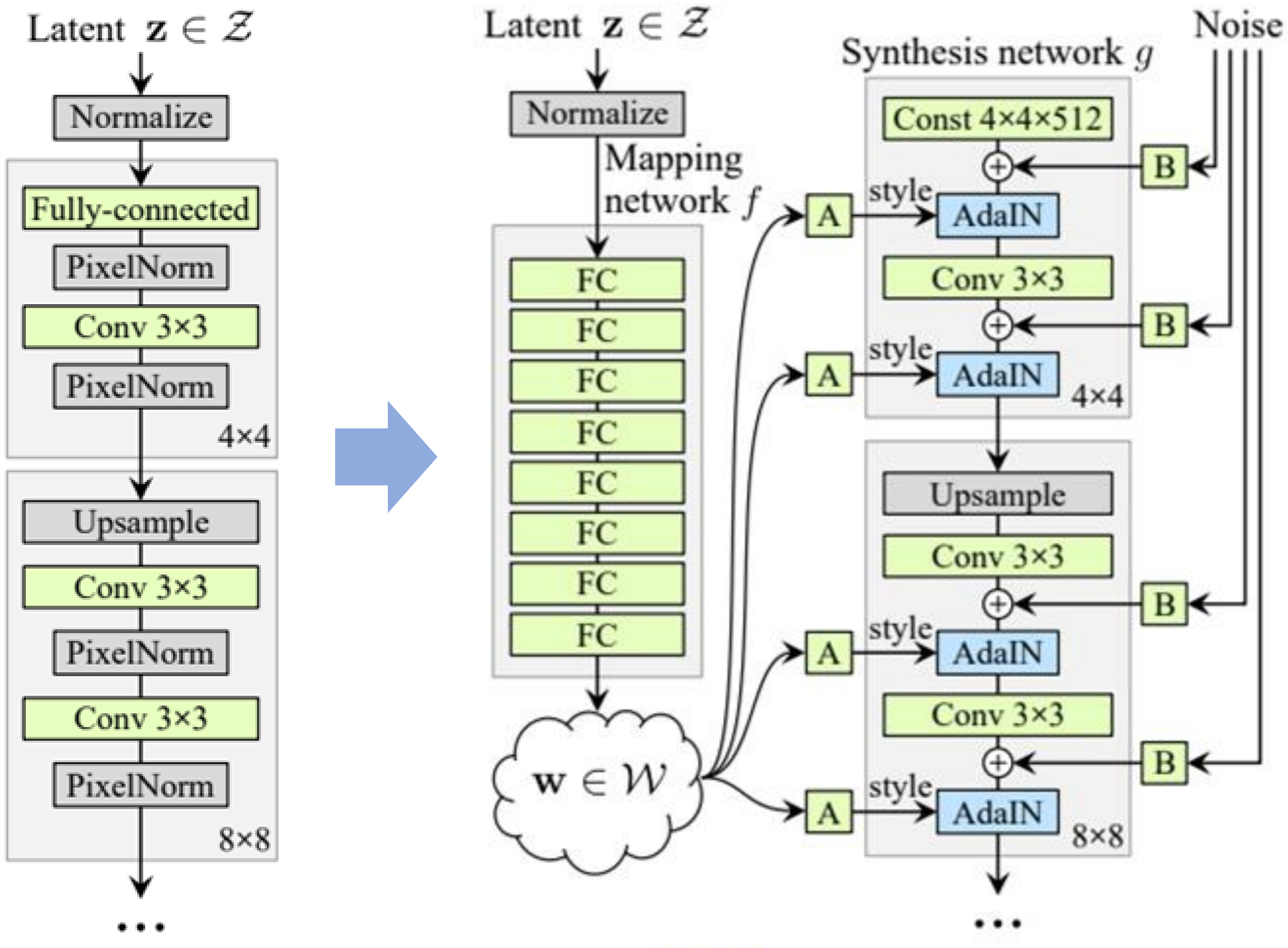
- **Style transfer + GAN = StyleGAN**

[장점]

1. **특질분리 - 세부적인 특질을 컨트롤 할 수 있음(인종, 포즈 등)**
2. **부분변화 샘플링 - 주요 특질은 보존한채 세부적인부분(피부상태, 머리카락의 위치 등)의 변화만 주는 샘플링 가능**
3. **성능개선**

A Style-Based Generator Architecture for Generative Adversarial Networks

- **StyleGAN의 구조**



(a) Traditional

(b) Style-based generator

Architecture

(1) **Mapping network** 사용

z 도메인에서 w 도메인으로 매핑

(2) 비선형 변환을 거친 **w 벡터**를 각 Block마다 넣어줌

(3) Affine transformation 후 , **AdaIN layer**사용

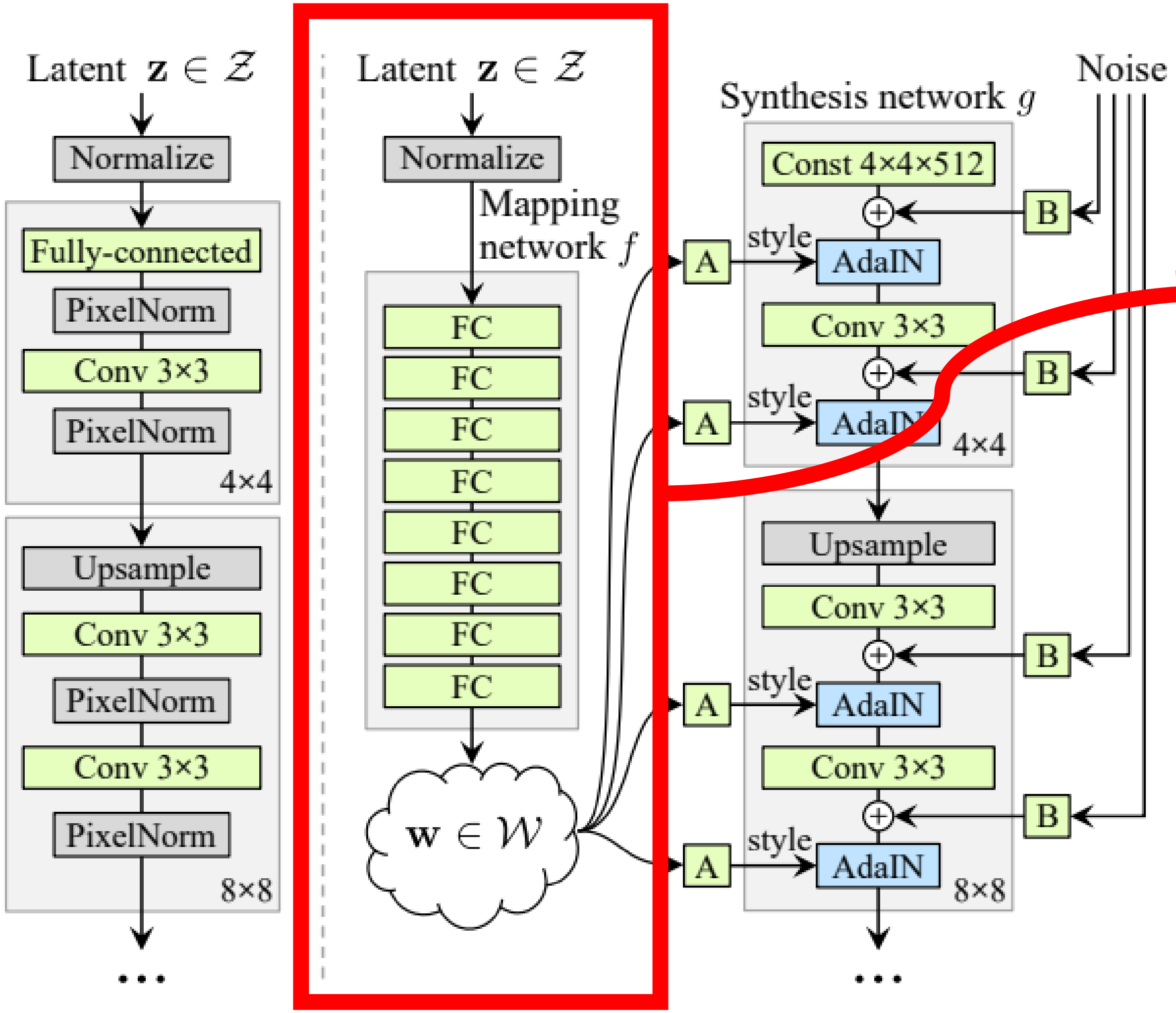
(4) Noise 추가

- **StyleGAN**

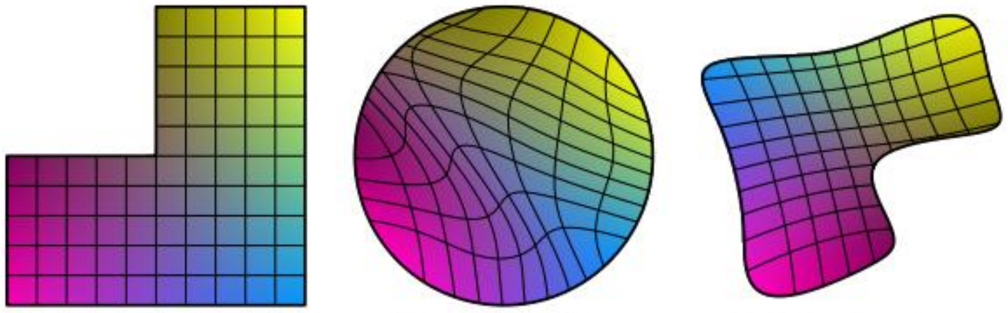
서로 다른 이미지를 만들어 내는 스타일들을 섞어서 이미지를 만들어 내면,
스케일마다 다른 특성을 가지는 새로운 얼굴 이미지를 생성할 수 있음.

A Style-Based Generator Architecture for Generative Adversarial Networks

- **StyleGAN의 장점(1) 특질분리**



(a) Traditional (b) Style-based generator


(a) Distribution of features in training set (b) Mapping from $\mathcal{Z}$ to features (c) Mapping from $\mathcal{W}$ to features

- **Mapping network 사용**

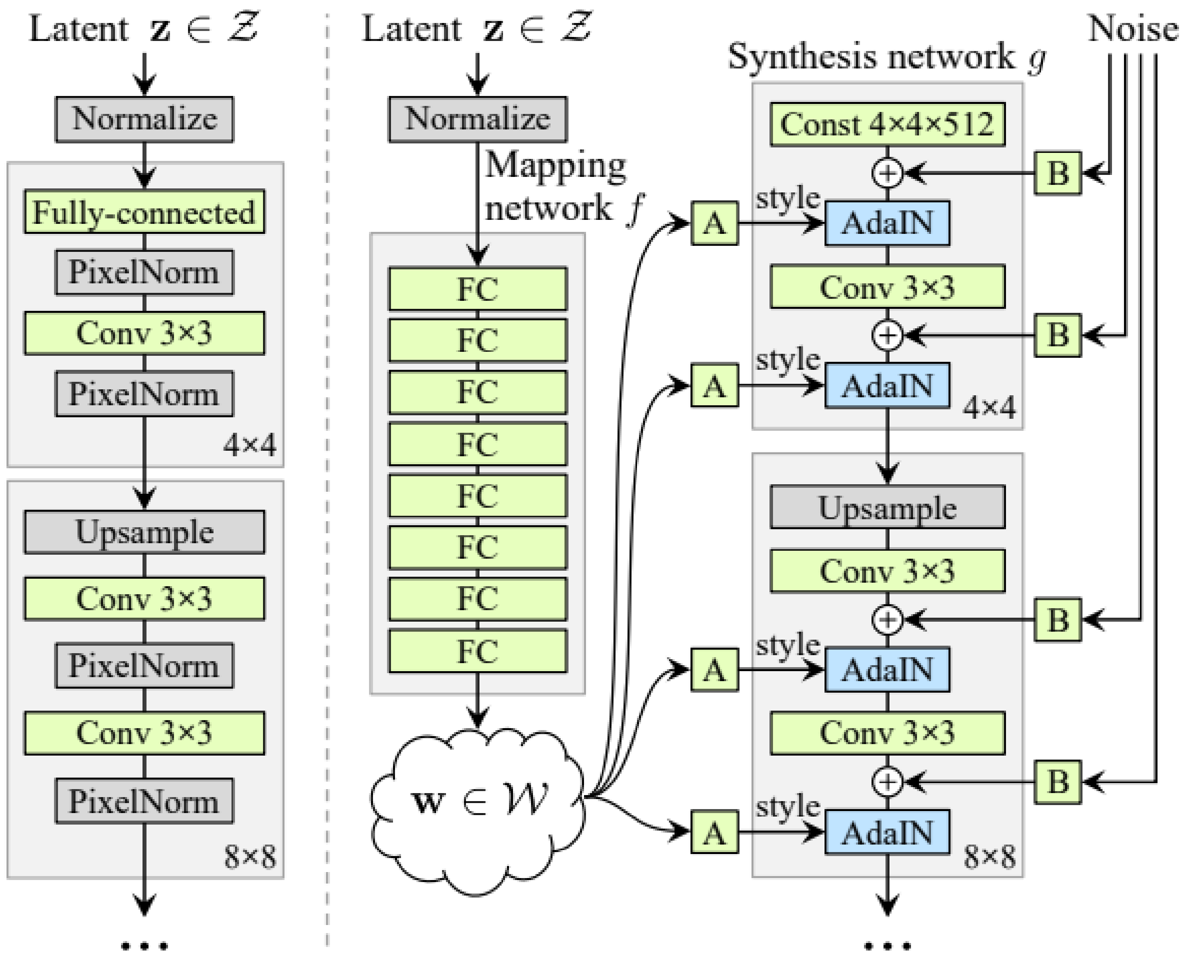
  기존 모델의 z 도메인에서 w 도메인으로 매핑

- **Feature들의 Entangled된 확률분포를**

  더욱 linear하고 덜 entangled하게 만들어줌

A Style-Based Generator Architecture for Generative Adversarial Networks

- **StyleGAN의 장점(1) 특징분리**



(a) Traditional (b) Style-based generator

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$

- Normalization기법으로

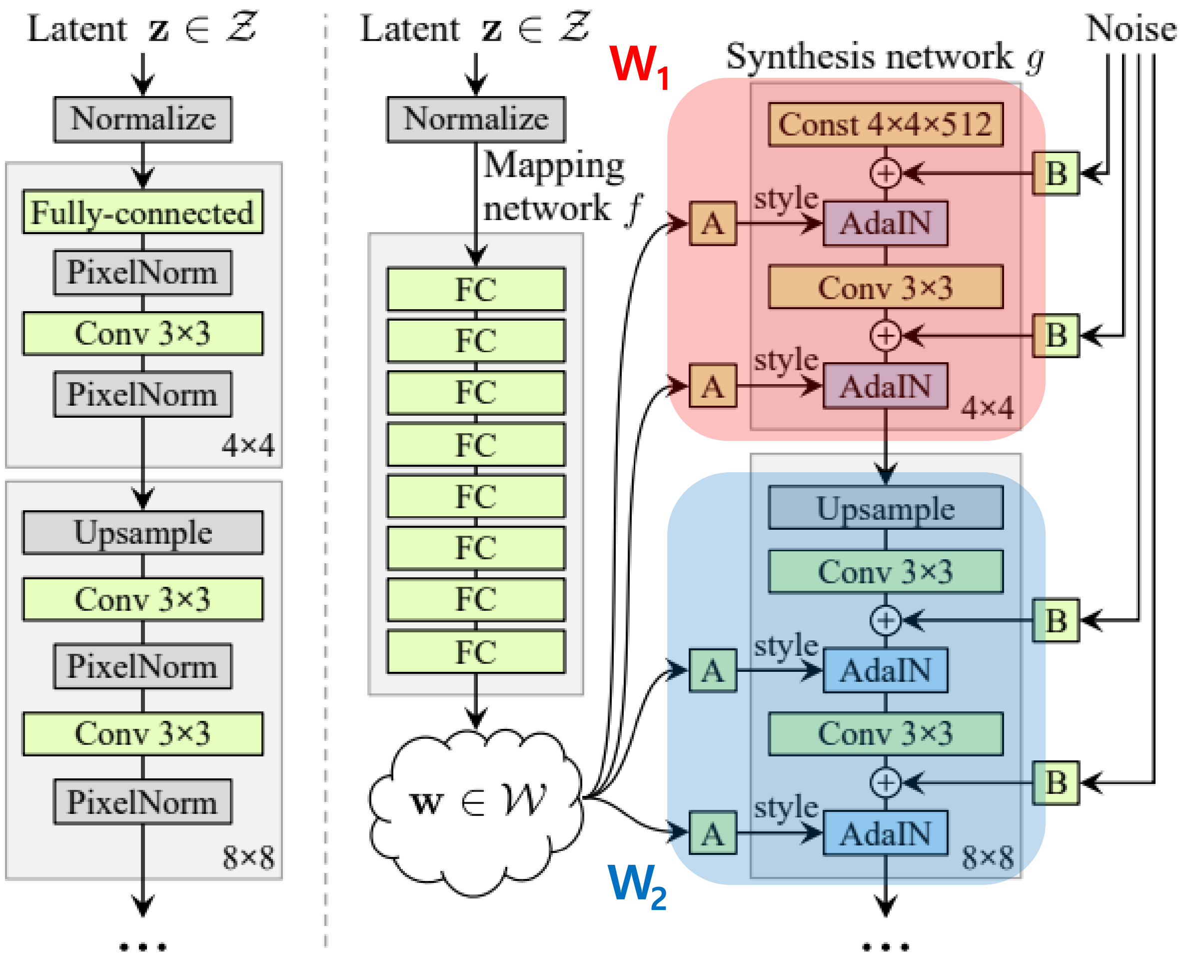
  Adaptive Instance Normalization(AdaIN) 사용

- style끼리 분리되기위해서는 input되는

  w가 서로 영향을 끼치면 안되기 때문에

- Normalization 할때마다 다른 w가 기여함

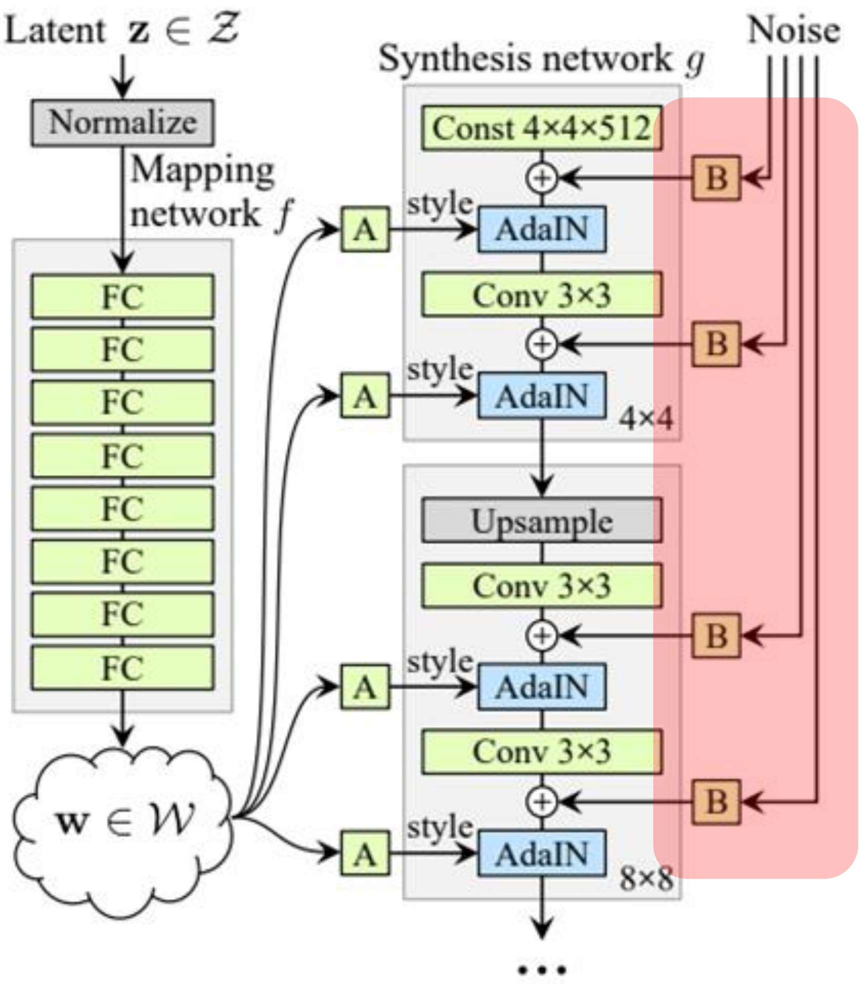
- **StyleGAN의 장점(1) 특질분리**



(a) Traditional (b) Style-based generator

- **Mixing regularization**
- **서로 다른 두 z1, z2를 섞어서 사용하는 것으로 인접한 스케일의 특질 사이의 correlation을 낮춤**

● StyleGAN의 장점(2) 부분 변화 샘플링



(b) Style-based generator

▪ Stochastic variation

매 채널마다 **Noise**를 넣어주어 확률적인 특성

이 결과 이미지에 반영될 수 있게 해줌

- 피부상태(여드름, 주근깨)

- 머리카락의 방향 등

- **StyleGAN의 장점(2) 부분 변화 샘플링**

(a) Generated image (b) Stochastic variation (c) Standard deviation

StyleGAN 의 한계점

1. Blob-like(Droplet) Artifacts: 물방울 형태의 인공물

Figure 1. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but if we look at the activations inside the generator network, the problem is always there, in all feature maps starting from the 64x64 resolution. It is a systemic problem that plagues all StyleGAN images.

2. Phase Artifacts : 이미지 고정값 이슈

Figure 6. Progressive growing leads to "phase" artifacts. In this example the teeth do not follow the pose but stay aligned to the camera, as indicated by the blue line.

StyleGAN2 – CVPR 2020

(Analyzing and Improving the Image Quality of StyleGAN)

Analyzing and Improving the Image Quality of StyleGAN (StyleGAN2)

1. 문제점 해결

(1) Blob-like(Droplet) Artifacts (물방울형태의 인공물이 발생되는 문제)

원인 : AdaIN style transfer

→ **AdaIN을 제거하고 weight demodulation으로 대체**

(2) Phase Artifacts (일부 요소들이 고정되는 문제)

원인 : Progressive growing

→ **generator의 Progressive growing을 다른 Architecture의 구조로 교체**

2. 성능 개선

큰 네트워크(highest-resolution layer의 feature map개수 2배)를 사용해서 고해상도의 디테일을 살림

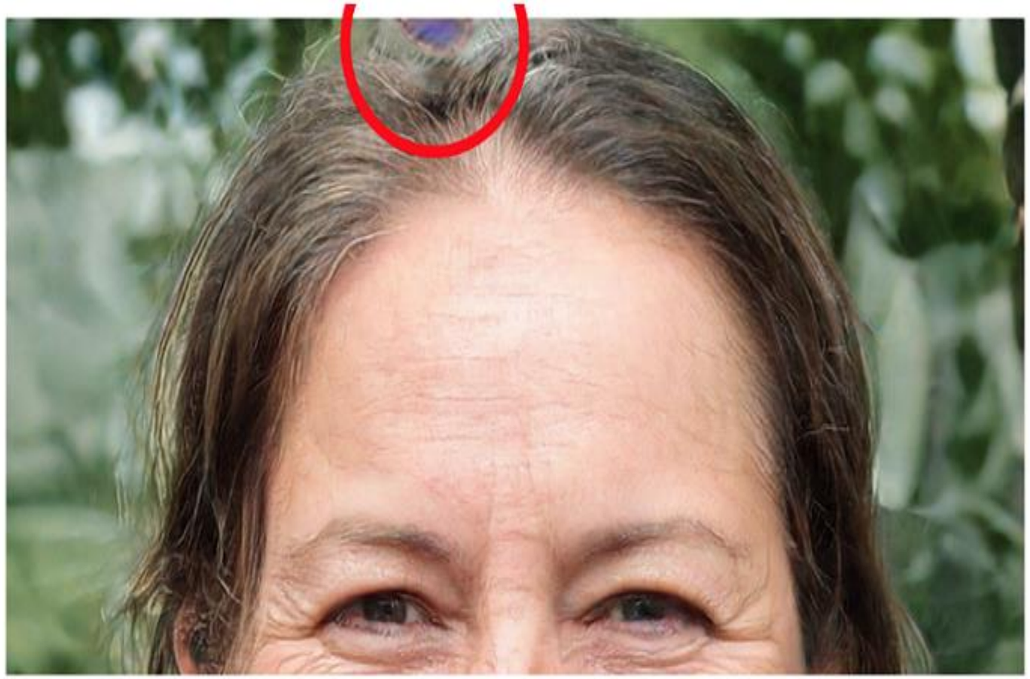
Analyzing and Improving the Image Quality of StyleGAN (StyleGAN2)

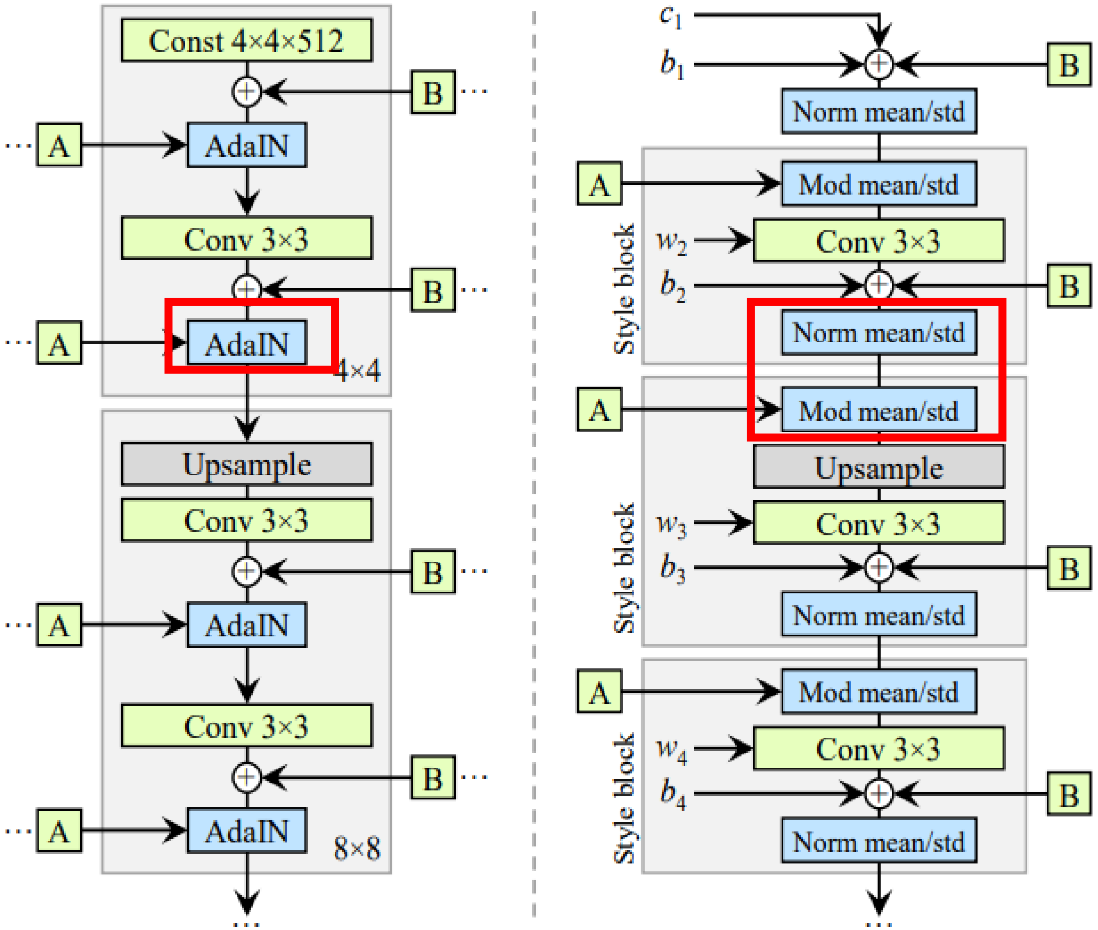
- **StyleGAN의 문제점(1) Blob-like(Droplet) Artifacts**

  *원인 : **Adaptive Instance Normalization(AdaIN)***

  ✓ 보통 Generator의 64X64에서부터 Droplet artifact가 보이기 시작 후 모든 해상도에 발현

  ✓ 해상도가 높아질수록 점점 더 강하게 나타나는 특성



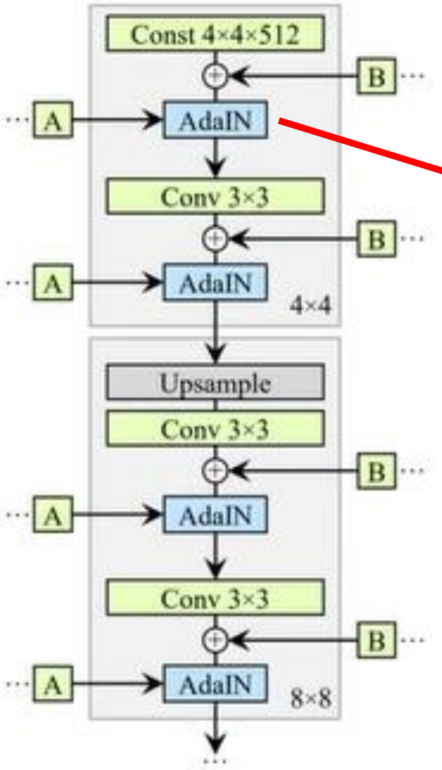


(a) StyleGAN (b) StyleGAN (detailed)

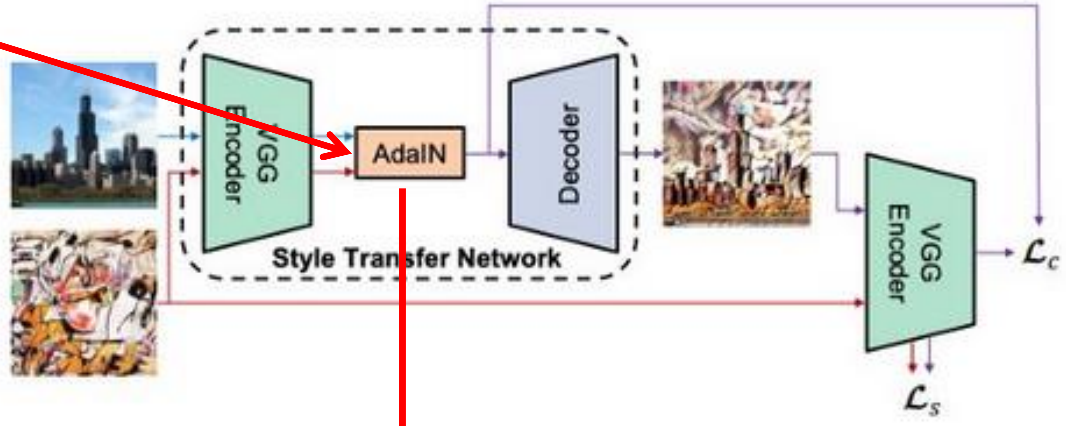
Analyzing and Improving the Image Quality of StyleGAN (StyleGAN2)

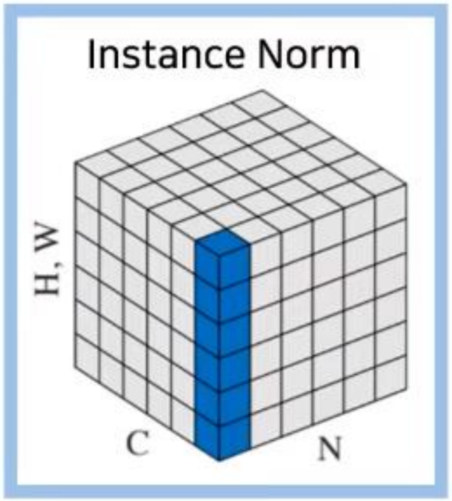
- **StyleGAN의 문제점(1) Blob-like(Droplet) Artifacts**

** <u>AdaIN : 잠재적으로 feature들 사이에 상대적인 크기에서 발견되는 정보들을 파괴</u>*



(a) StyleGAN

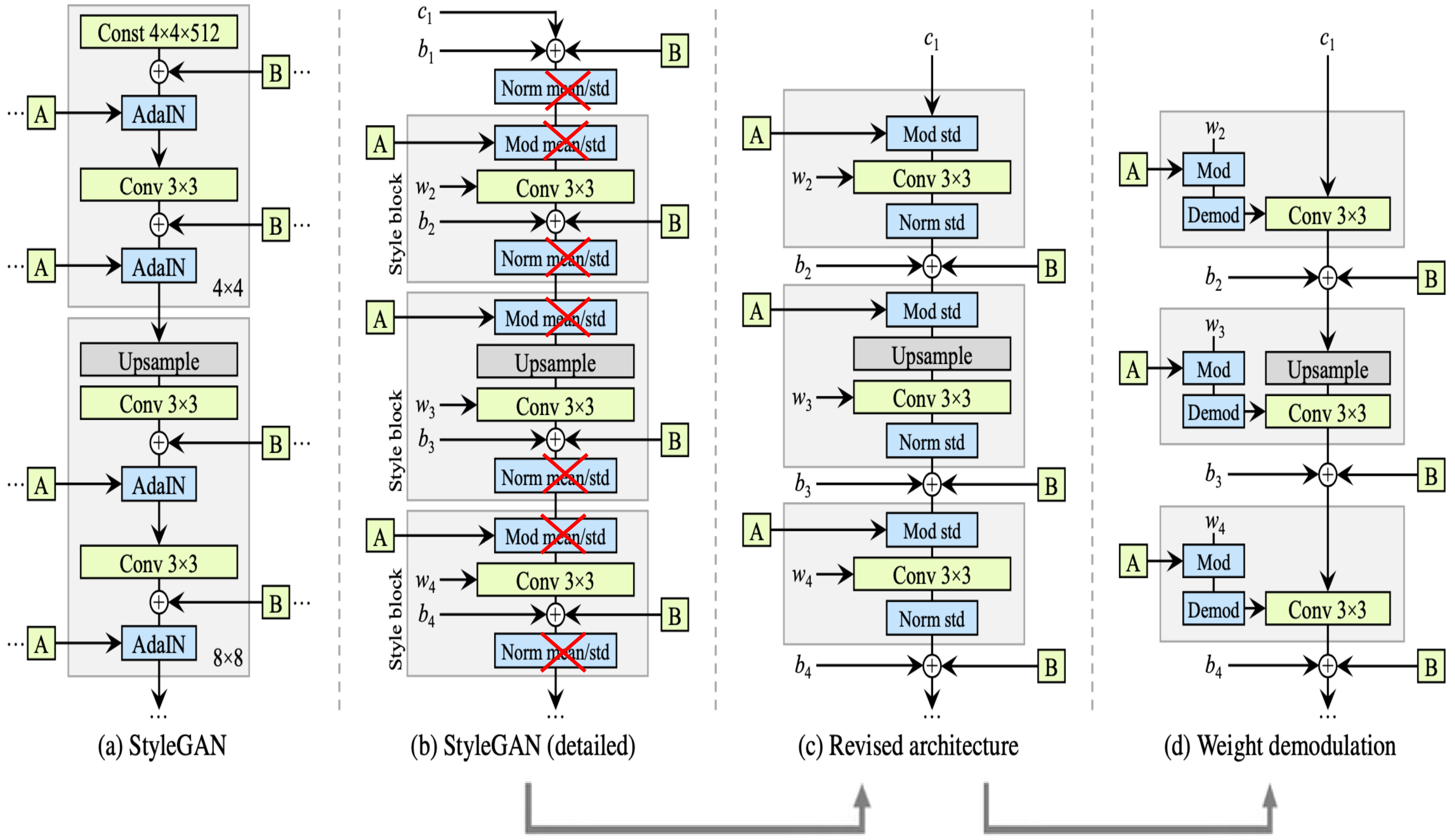




AdaIN : 각 feature map 의 평균과 분산을 normalize

Analyzing and Improving the Image Quality of StyleGAN (StyleGAN2)

1. Blob-like(Droplet) Artifacts Solution: **Normalization 단계 제거**



(a) StyleGAN (b) StyleGAN (detailed) (c) Revised architecture (d) Weight demodulation

핵심아이디어

- Mean(평균)값을 제외하고 Standard deviation(표준편차)만 사용해도 기능상 충분
- 정규화 과정을 block외부에서 진행하여 나온 w값으로 Conv연산 수행
- 이로인해, Style block에서의 일부 feature의 증폭을 방지할 수 있음

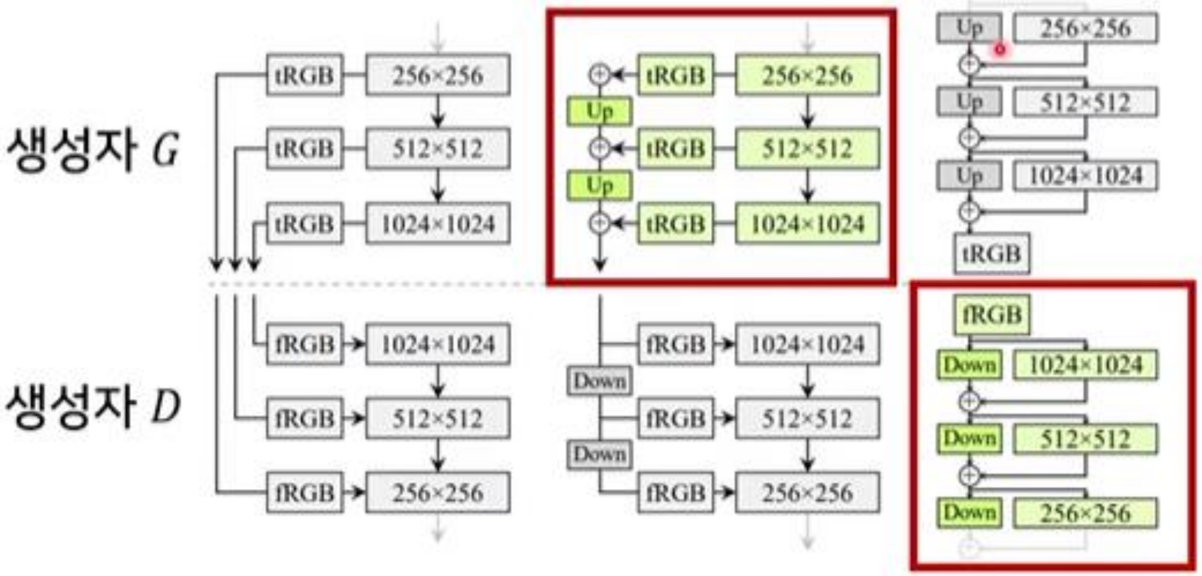
Droplet Artifacts 문제해결

Analyzing and Improving the Image Quality of StyleGAN (StyleGAN2)

- **StyleGAN의 문제점(2) Phase Artifacts**

  원인 : Progressive growing

(2) Phase Artifacts Solution : 다른 Architecture 네트워크 사용



(a) MSG-GAN (b) Input/output skips (c) Residual nets

FFHQ	D original FID	D original PPL	D input skips FID	D input skips PPL	D residual FID	D residual PPL
G original	4.32	265	4.18	235	3.58	269
G output skips	4.33	169	3.77	127	**3.31**	**125**
G residual	4.35	203	3.96	229	3.79	243

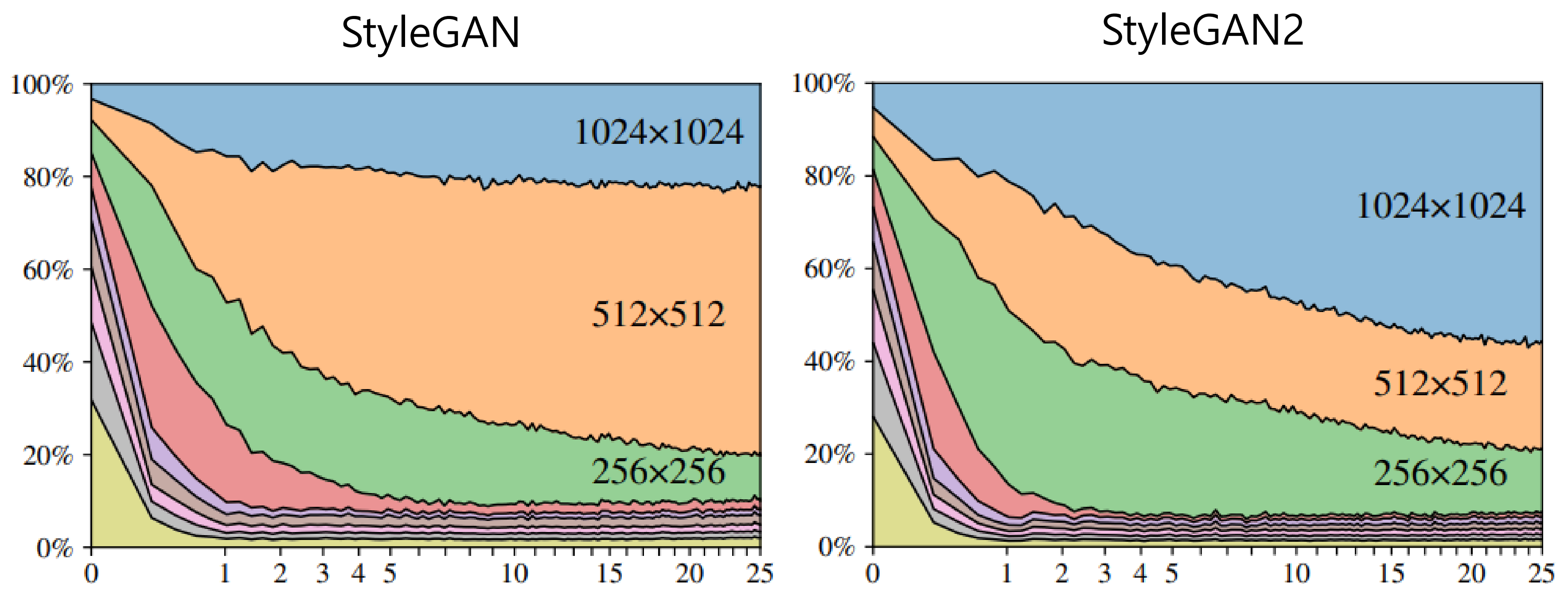
LSUN Car	D original FID	D original PPL	D input skips FID	D input skips PPL	D residual FID	D residual PPL
G original	3.75	905	3.23	758	3.25	802
G output skips	3.77	544	3.86	**316**	3.19	471
G residual	3.93	981	3.40	667	**2.66**	645

- **Generator**(Synthesis Network)는 RGB outputs의 contribution을 업샘플링 하고 합하며 bilinear(쌍선형) 필터링을 사용하는 **output skip** 사용
- **Discriminator**는 **Residual connection**을 사용하도록 수정(LAPGAN과 유사함)

  → 다음과 같이 사용했을 때 **성능**이 좋았음

Analyzing and Improving the Image Quality of StyleGAN (StyleGAN2)

- **StyleGAN의 성능 개선**



(a) StyleGAN-sized (config E) (b) Large networks (config F)

- StyleGAN 과 StyleGAN2의 훈련시간에 따른 해상도의 퀄리티 비교

OUTPUT

Analyzing and Improving the Image Quality of StyleGAN (StyleGAN2)

- StyleGAN2 Style mixing output

Analyzing and Improving the Image Quality of StyleGAN (StyleGAN2)

- StyleGAN2 Style mixing output

핵심 코드

```python
@persistence.persistent_class
class FullyConnectedLayer(torch.nn.Module):
    def __init__(self,
        in_features,                # Number of input features.
        out_features,               # Number of output features.
        bias            = True,     # Apply additive bias before the activation function?
        activation      = 'linear', # Activation function: 'relu', 'lrelu', etc.
        lr_multiplier   = 1,        # Learning rate multiplier.
        bias_init       = 0,        # Initial value for the additive bias.
    ):
        super().__init__()
        self.activation = activation
        self.weight = torch.nn.Parameter(torch.randn([out_features, in_features]) / lr_multiplier)
        self.bias = torch.nn.Parameter(torch.full([out_features], np.float32(bias_init))) if bias else None
        self.weight_gain = lr_multiplier / np.sqrt(in_features)
        self.bias_gain = lr_multiplier
```

```python
@persistence.persistent_class
class MappingNetwork(torch.nn.Module):
    def __init__(self,
        z_dim,                      # Input latent (Z) dimensionality, 0 = no latent.
        c_dim,                      # Conditioning label (C) dimensionality, 0 = no label.
        w_dim,                      # Intermediate latent (W) dimensionality.
        num_ws,                     # Number of intermediate latents to output, None = do not broadcast.
        num_layers      = 8,        # Number of mapping layers.
        embed_features  = None,     # Label embedding dimensionality, None = same as w_dim.
        layer_features  = None,     # Number of intermediate features in the mapping layers, None = same as w_dim.
        activation      = 'lrelu',  # Activation function: 'relu', 'lrelu', etc.
        lr_multiplier   = 0.01,     # Learning rate multiplier for the mapping layers.
        w_avg_beta      = 0.995,    # Decay for tracking the moving average of W during training, None = do not track.
    ):
```

```python
@persistence.persistent_class
class SynthesisLayer(torch.nn.Module):
    def __init__(self,
        in_channels,                    # Number of input channels.
        out_channels,                   # Number of output channels.
        w_dim,                          # Intermediate latent (W) dimensionality.
        resolution,                     # Resolution of this layer.
        kernel_size     = 3,            # Convolution kernel size.
        up              = 1,            # Integer upsampling factor.
        use_noise       = True,         # Enable noise input?
        activation      = 'lrelu',      # Activation function: 'relu', 'lrelu', etc.
        resample_filter = [1,3,3,1],    # Low-pass filter to apply when resampling activations.
        conv_clamp      = None,         # Clamp the output of convolution layers to +-X, None = disable clamping.
        channels_last   = False,        # Use channels_last format for the weights?
    ):
        super().__init__()
        self.resolution = resolution
        self.up = up
        self.use_noise = use_noise
        self.activation = activation
        self.conv_clamp = conv_clamp
        self.register_buffer('resample_filter', upfirdn2d.setup_filter(resample_filter))
        self.padding = kernel_size // 2
        self.act_gain = bias_act.activation_funcs[activation].def_gain
```

```
if in_channels != 0:
    self.conv0 = SynthesisLayer(in_channels, out_channels, w_dim=w_dim, resolution=resolution, up=2,
        resample_filter=resample_filter, conv_clamp=conv_clamp, channels_last=self.channels_last, **layer_kwargs)
    self.num_conv += 1

self.conv1 = SynthesisLayer(out_channels, out_channels, w_dim=w_dim, resolution=resolution,
    conv_clamp=conv_clamp, channels_last=self.channels_last, **layer_kwargs)
self.num_conv += 1
```

해상도를
높여감

```python
@misc.profiled_function
def modulated_conv2d(
    x,                          # Input tensor of shape [batch_size, in_channels, in_height, in_width].
    weight,                     # Weight tensor of shape [out_channels, in_channels, kernel_height, kernel_width].
    styles,                     # Modulation coefficients of shape [batch_size, in_channels].
    noise           = None,     # Optional noise tensor to add to the output activations.
    up              = 1,        # Integer upsampling factor.
    down            = 1,        # Integer downsampling factor.
    padding         = 0,        # Padding with respect to the upsampled image.
    resample_filter = None,     # Low-pass filter to apply when resampling activations. Must be prepared beforehand by calling upfirdn2d.setup_filter().
    demodulate      = True,     # Apply weight demodulation?
    flip_weight     = True,     # False = convolution, True = correlation (matches torch.nn.functional.conv2d).
    fused_modconv   = True,     # Perform modulation, convolution, and demodulation as a single fused operation?
):
    batch_size = x.shape[0]
    out_channels, in_channels, kh, kw = weight.shape
    misc.assert_shape(weight, [out_channels, in_channels, kh, kw]) # [OIkk]
    misc.assert_shape(x, [batch_size, in_channels, None, None]) # [NIHW]
    misc.assert_shape(styles, [batch_size, in_channels]) # [NI]

    # Pre-normalize inputs to avoid FP16 overflow.
    if x.dtype == torch.float16 and demodulate:
        weight = weight * (1 / np.sqrt(in_channels * kh * kw) / weight.norm(float('inf'), dim=[1,2,3], keepdim=True)) # max_Ikk
        styles = styles / styles.norm(float('inf'), dim=1, keepdim=True) # max_I
```

- **미리 Conv Layer 외부에서 Weight 값에 대해 Modulation(Scale), Demodulation(Normalization) 시행**
- **Blob Artifacts 해결**

```python
class SynthesisBlock(torch.nn.Module):
    def __init__(self,
        in_channels,                        # Number of input channels, 0 = first block.
        out_channels,                       # Number of output channels.
        w_dim,                              # Intermediate latent (W) dimensionality.
        resolution,                         # Resolution of this block.
        img_channels,                       # Number of output color channels.
        is_last,                            # Is this the last block?
        architecture        = 'skip',       # Architecture: 'orig', 'skip', 'resnet'.
        resample_filter     = [1,3,3,1],    # Low-pass filter to apply when resampling activations.
        conv_clamp          = None,         # Clamp the output of convolution layers to +-X, None = disable clamping.
        use_fp16            = False,        # Use FP16 for this block?
        fp16_channels_last  = False,        # Use channels-last memory format with FP16?
        **layer_kwargs,                     # Arguments for SynthesisLayer.
    ):
        assert architecture in ['orig', 'skip', 'resnet']
```

- **Generator Architecture를 multi-skip이 가능한 input/output skips 로 변경**
- **Phase Artifact 해결 및 성능 개선(MSG-Gan / skips / resnet 중 skips가 가장 성능이 좋음)**

```python
class DiscriminatorBlock(torch.nn.Module):
    def __init__(self,
        in_channels,                        # Number of input channels, 0 = first block.
        tmp_channels,                       # Number of intermediate channels.
        out_channels,                       # Number of output channels.
        resolution,                         # Resolution of this block.
        img_channels,                       # Number of input color channels.
        first_layer_idx,                    # Index of the first layer.
        architecture        = 'resnet',     # Architecture: 'orig', 'skip', 'resnet'.
        activation          = 'lrelu',      # Activation function: 'relu', 'lrelu', etc.
        resample_filter     = [1,3,3,1],    # Low-pass filter to apply when resampling activations.
        conv_clamp          = None,         # Clamp the output of convolution layers to +-X, None = disable clamping.
        use_fp16            = False,        # Use FP16 for this block?
        fp16_channels_last  = False,        # Use channels-last memory format with FP16?
        freeze_layers       = 0,            # Freeze-D: Number of layers to freeze.
    ):
        assert in_channels in [0, tmp_channels]
        assert architecture in ['orig', 'skip', 'resnet']
```

- **Discrimonator Architecture를 Resdual Network 로 변경**
- **Phase Artifact 해결 및 성능 개선(MSG-Gan / skips / resnet 중 resnet이 가장 성능이 좋음)**

Q & A!