Dual Style GAN

Pastiche Master: Exemplar-Based High-Resolution Portrait Style Transfer

Shuai Yang, Liming Jiang, Ziwei Liu, Chen Change Loy

[Submitted on 24 Mar 2022]

논문 구현

CloseAI팀 이상헌 박준혁 김유철 이정훈

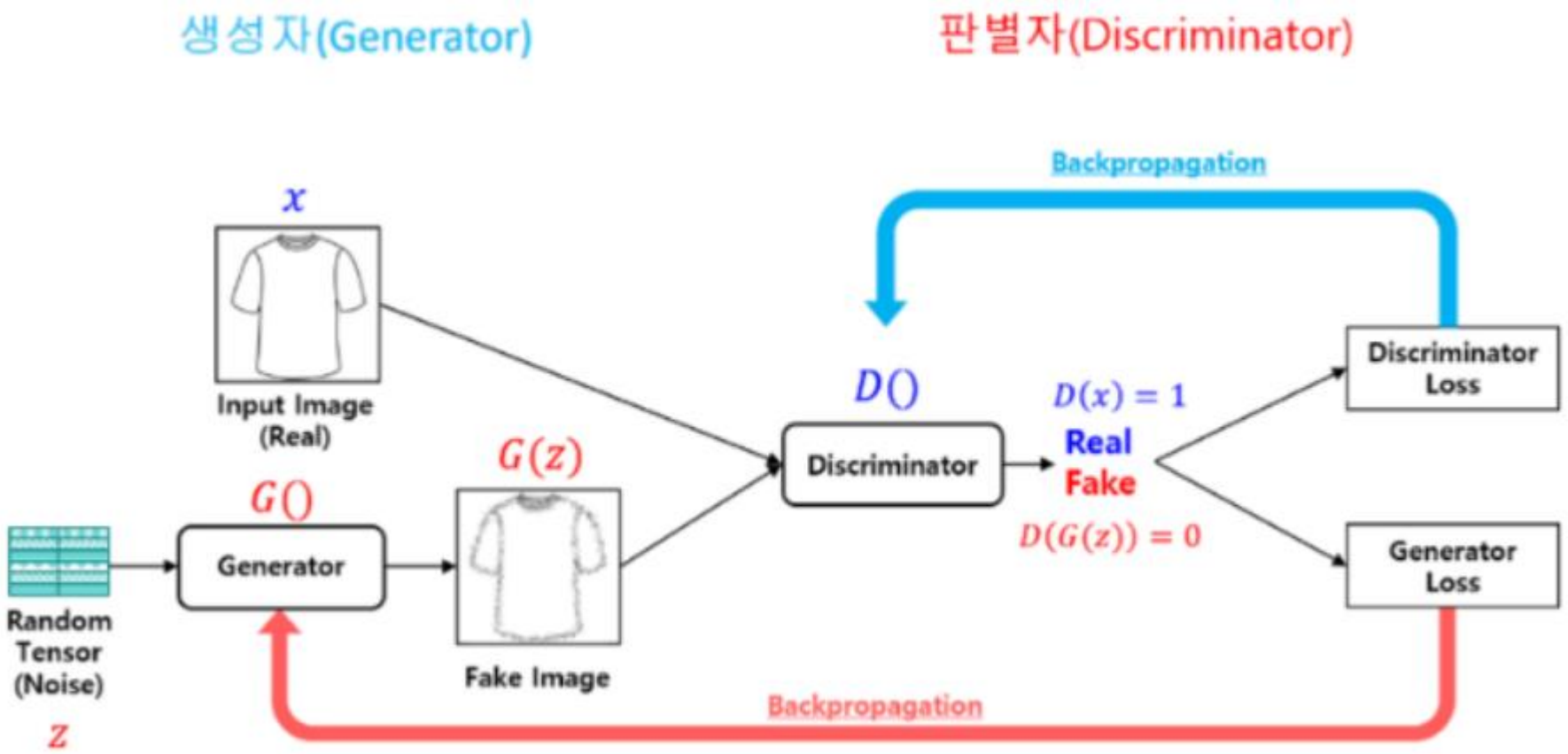
목차

- GAN
- PG GAN
- Style GAN1
- Style GAN2
- Dual Style GAN
- 코드 구현

GAN

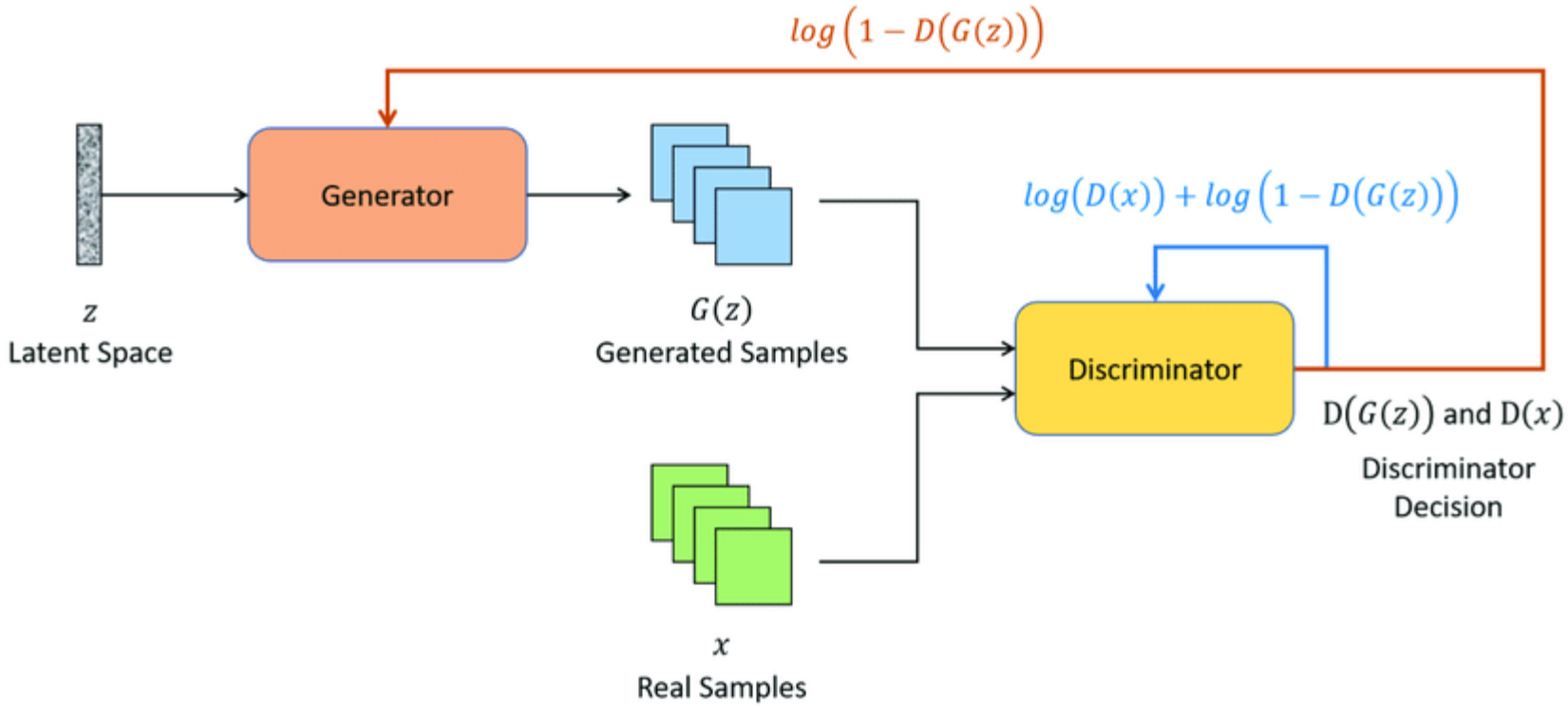
GAN (Generative Adversarial Network)

GAN (적대적 생성모델) : 딥러닝을 통해 임의의 노이즈로부터 가상의 데이터를 생성하는 네트워크(**Generator**)가 실제와 같은 데이터를 생성할 수 있도록 진위 여부를 판별 할 수 있는 네크워크(**Discriminator**)를 경쟁시켜 학습시키는 모델

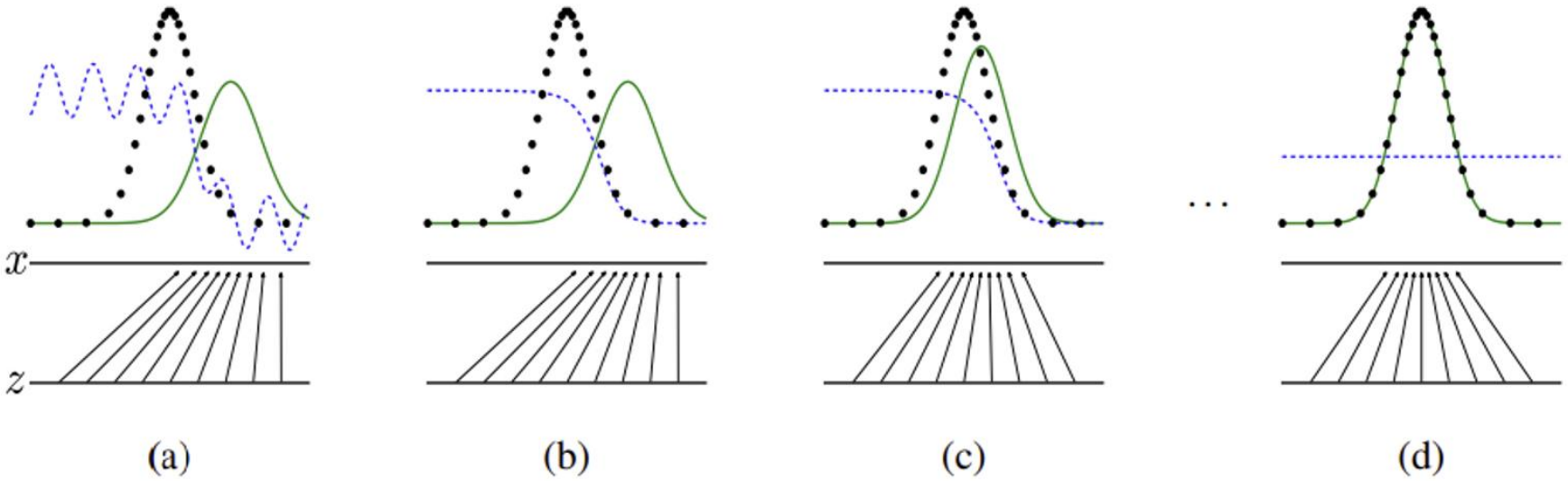


GAN 학습과정(2)

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{x\sim Pdata(x)}[logD(x)] + \mathbb{E}_{z\sim Pz(z)}[\log(1-D(G(z)))]$$



GAN 학습과정(1)



파란색 점선 : *D*, discriminative distribution (판별 모델의 분포)
검정색 점선 : *px*, 데이터에서 생성된 분포 (원본 데이터의 분포)
초록색 실선 : *pg* , generative distribution (생성 모델의 분포)
z → x 화살표 : *x*=*G*(*z*) 매핑, 분포 *pg*로 변환되는 과정

GAN 단점

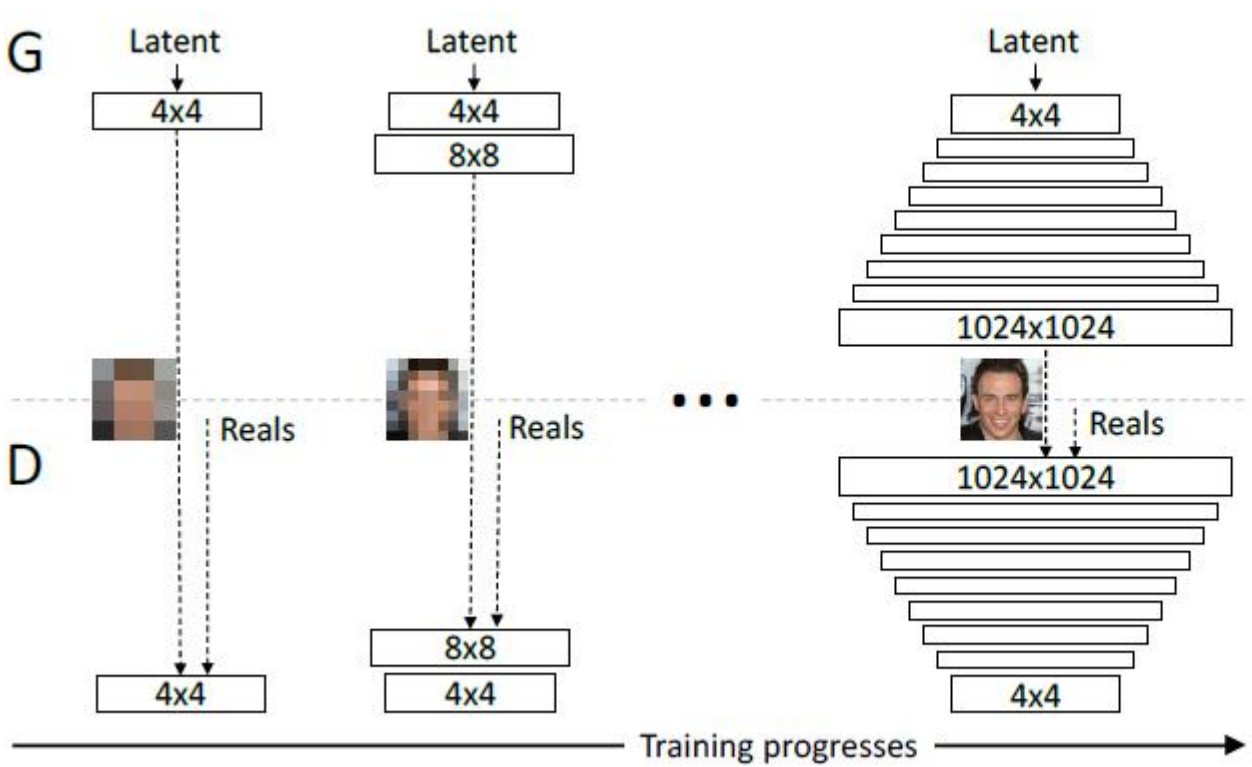
- Mode Collapse(실제 데이터의 분포를 모두 커버 하지 못하고 다양성을 잃어버리는 현상
- 고해상도 이미지를 만들기 위해 더 작은 미니 배치를 사용해야 하는데 이러면 훈련 안전성이 떨어지는 현상
- 고해상도 이미지를 생성할수록, 가짜 이미지라고 판별하기 쉬워지는 현상
- 실제 데이터 분포와 생성자모델의 분포의 겹치는 부분이 적다면 분포들간의 거리를 측정할 때 grdient가 랜덤한 방향을 가리 킬 수 있는 현상

PG GAN

PG GAN(Progressive Growing of GANs)

이러한 GAN의 단점들을 보완하고자 PGGAN에서는 생성자와 판별자를 점진적으로 학습

즉, 만들기 쉬운 저해상도부터 시작하여 새로운 layer를 조금씩 추가하여 고해상도의 이미지를 생성



PG GAN의 핵심 키워드 및 장점

Progressive Growing

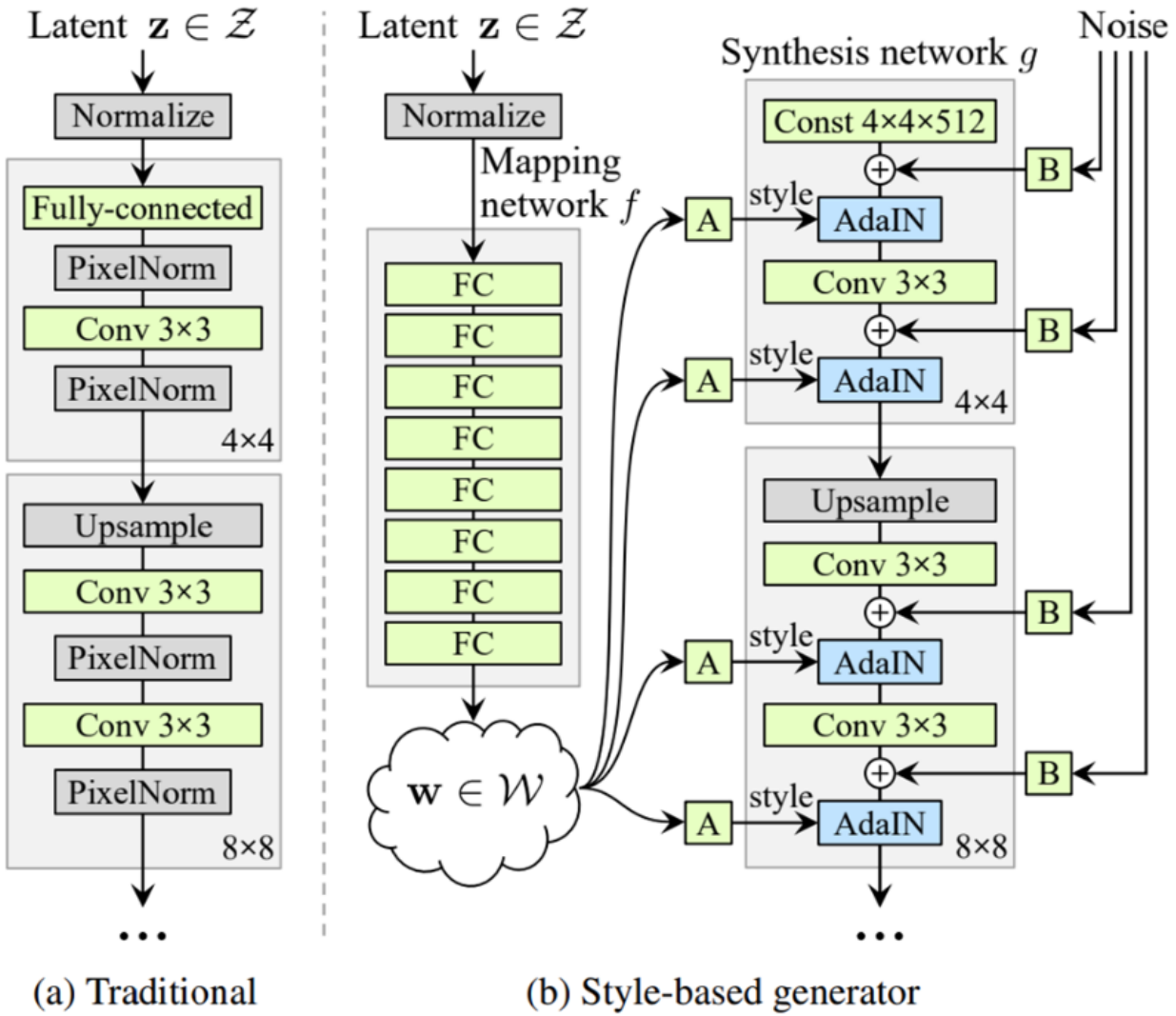
- 점진적으로 해상도를 높여가며 학습(학습 속도 2~6배 빨라짐)

Minibatch Std

- Discriminator의 마지막 블록에 해당 레이어를 추가하여 mode collaspe문제 보완

Style GAN

Style GAN 아키텍처

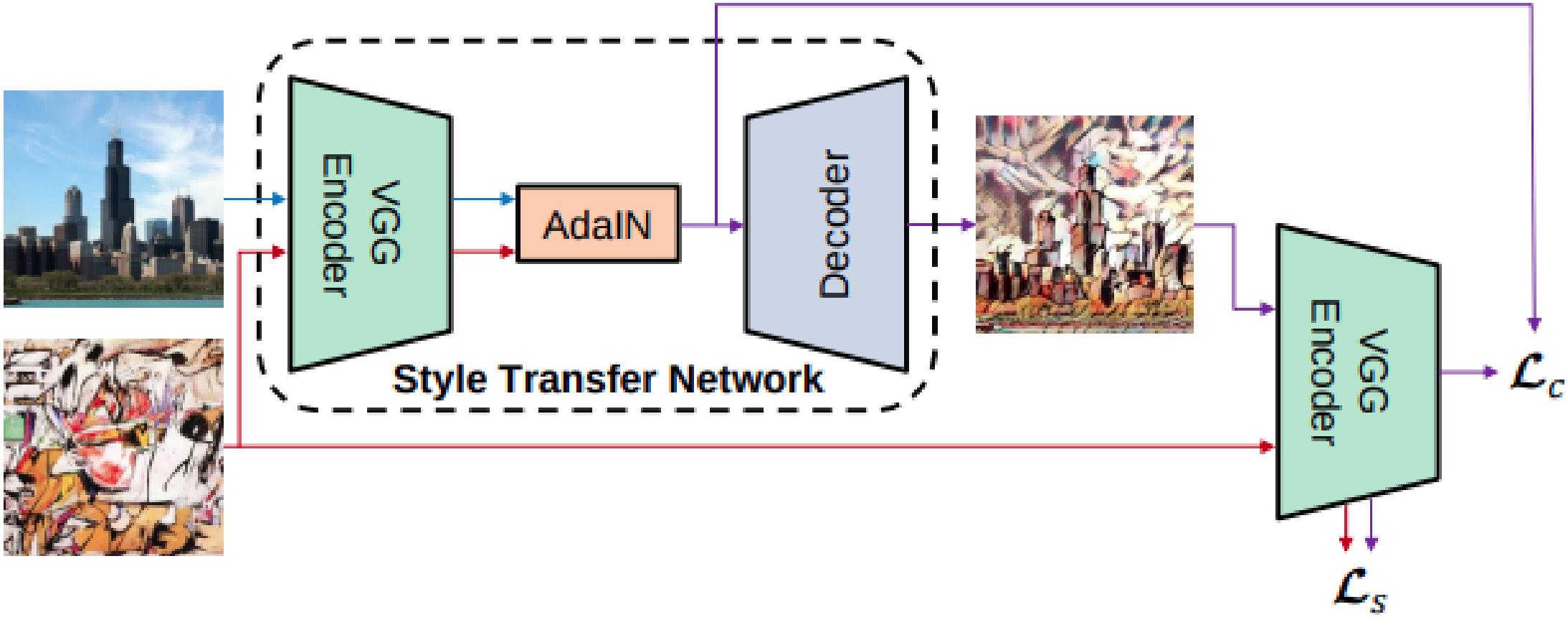


Generator architecture에 초점을 맞춘 새로운 네트워크

생성 모델이 만들어내는 이미지는 우리가 아는 다양한 스타일들의 조합이라고 생각

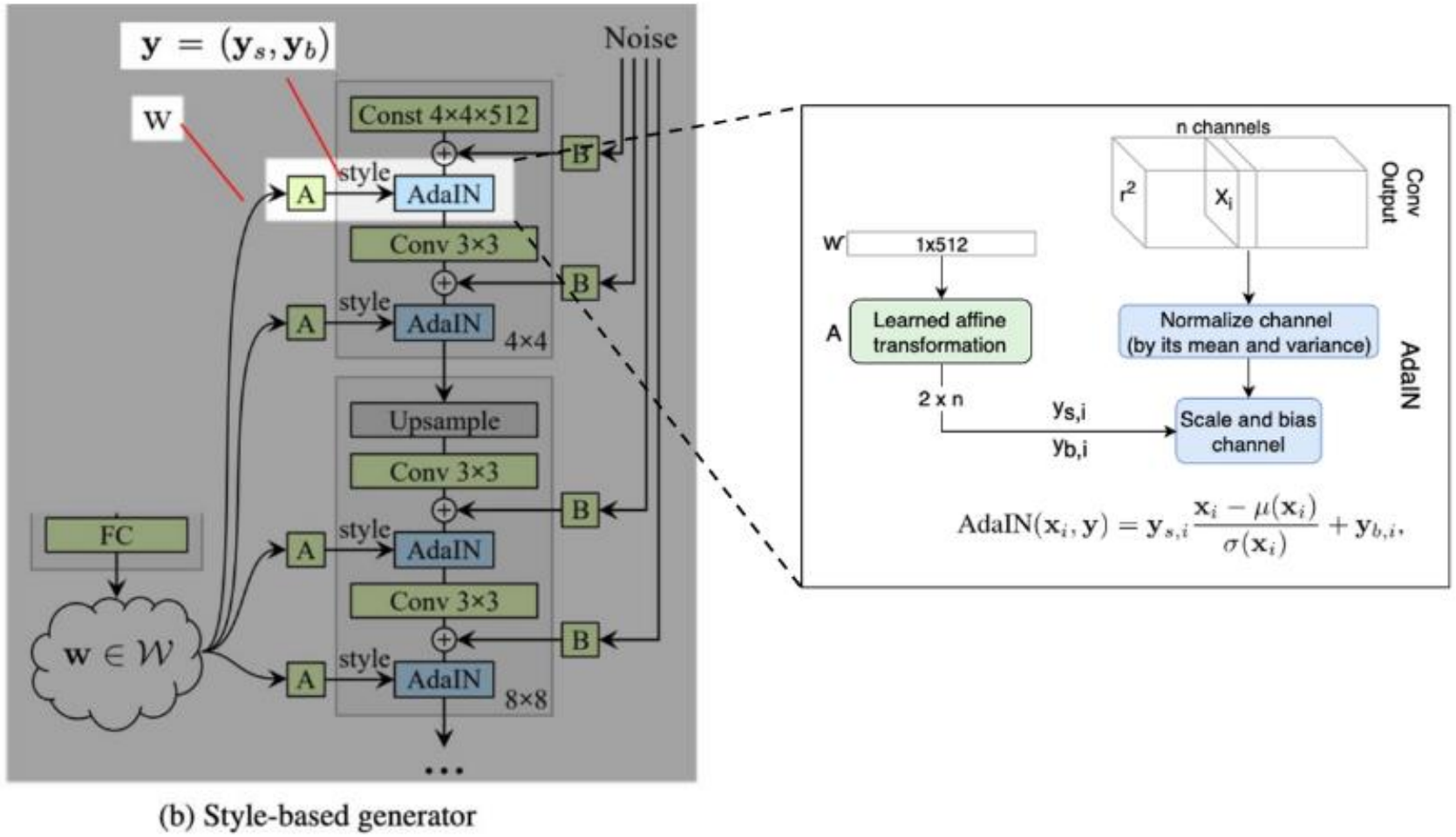
결국 생성되는 사람의 얼굴은 머리스타일, 성별, 피부색 등등 다양한 style들의 조합으로 이루어졌기에, 이미지를 생성함에 있어서 다양한 style들의 조합으로 새로운 fake image를 생성해낼 수 있습니다.

AdaIN



딥러닝 인코더 네트워크로부터 추출된 high-level feature map이 아래의 화풍과 같은 style 이미지의 통계적 특성을 따르게 하도록 변환해주는 AdaIN을 PG GAN의 Image Generator에 도입

AdaIN



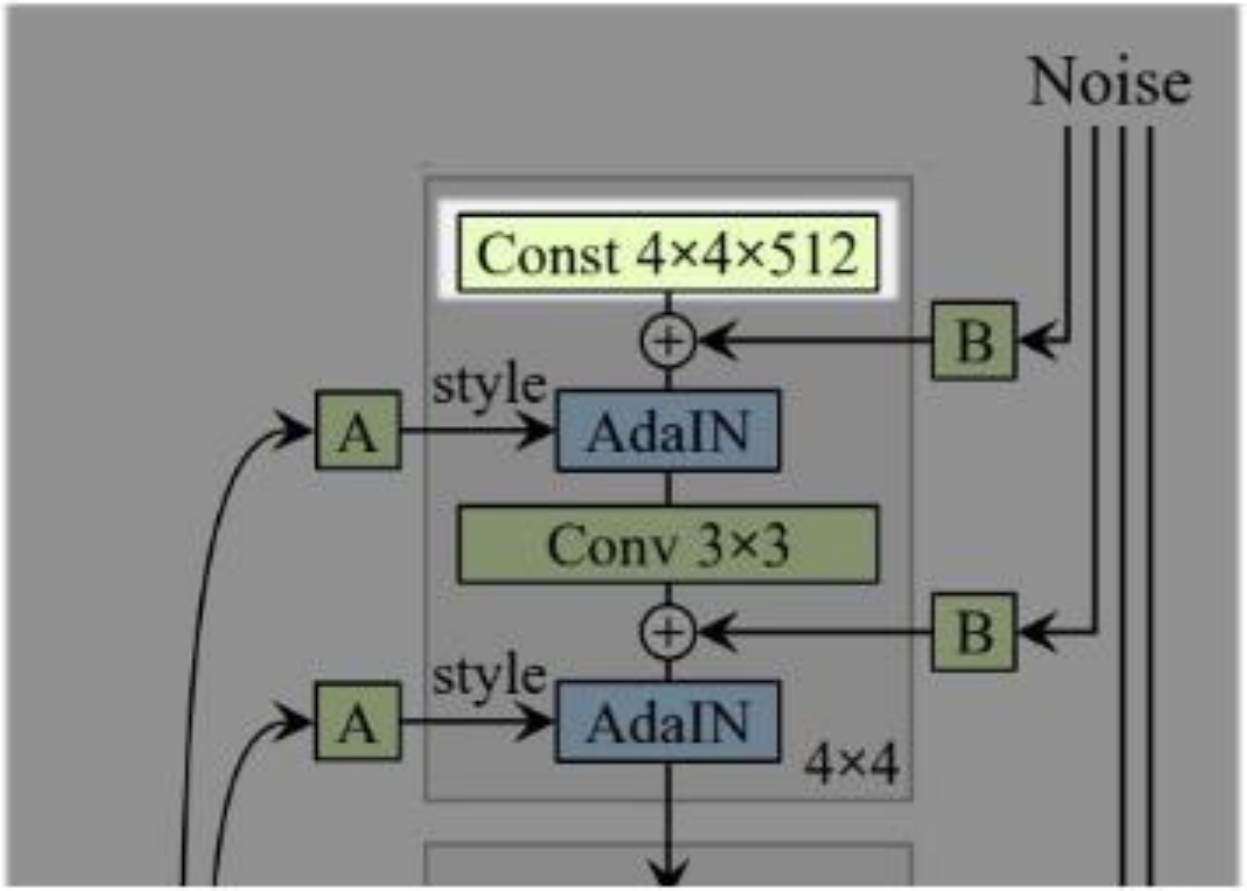
(b) Style-based generator

latent z를 8개의 Full Connected layer(Mapping network)를 통해 좀 더 실제 데이터 분포에 가깝고 disentanglement한 latent w로 만들어줬다.

Style은 latent vector w를 affine transformation을 거쳐 style 정보인 y로 변환한 것

AdaIN을 PG GAN 아키텍처에 도입함으로써 이미지 생성에 Style을 넣을 수 있게 되었다.

Why ‘learned constant’ ?

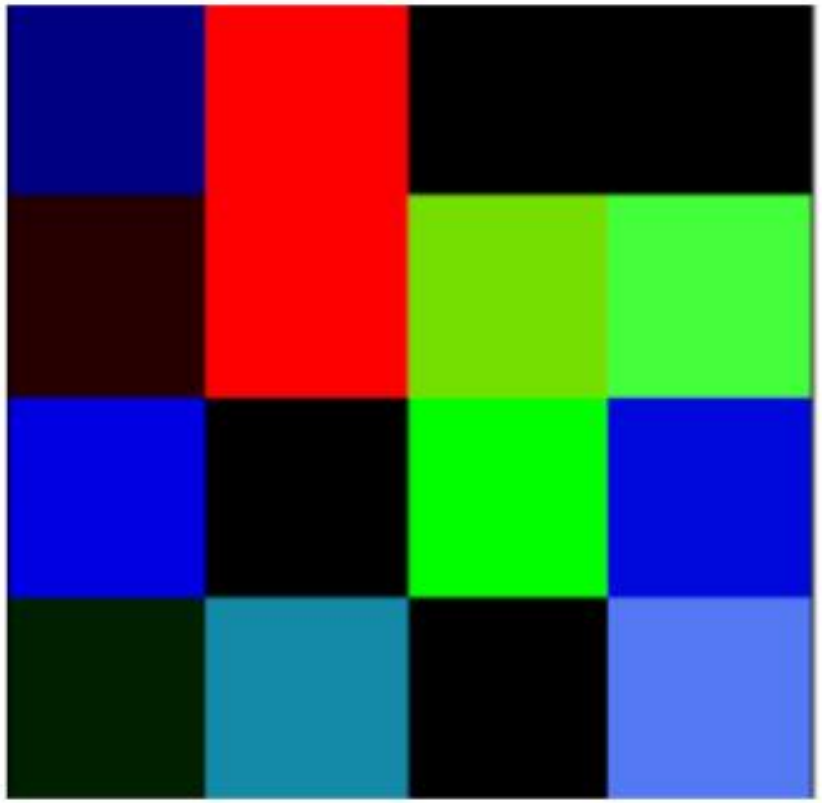


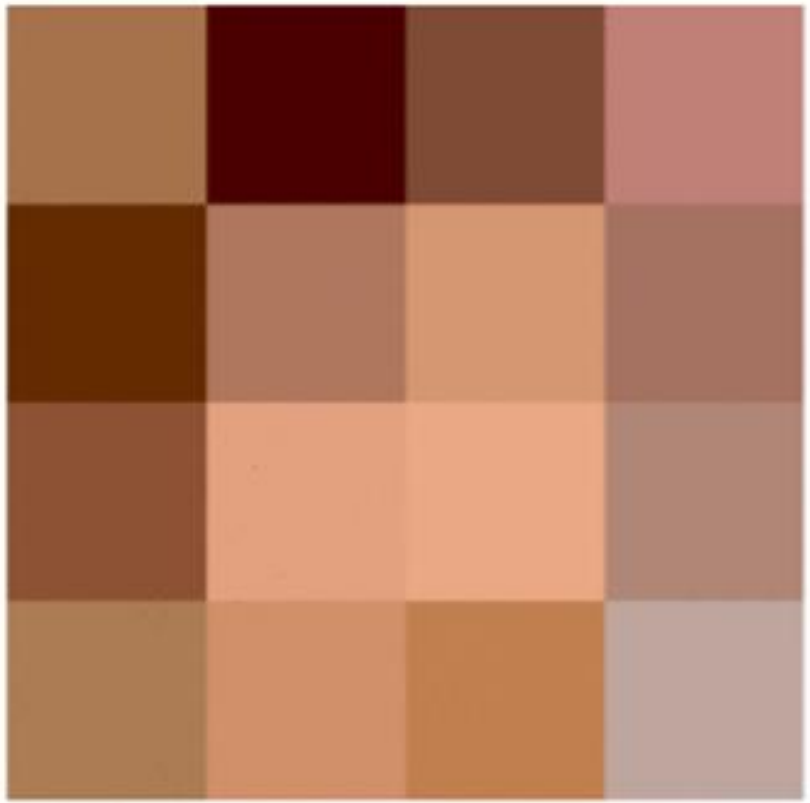
Style GAN이 학습되는 과정에서 learnable하게 학습되고,
Test(혹은 generation) 과정에서 이미지를 생성할 땐, 고정되어 있는 값

초기의 4x4 이미지가 무엇이어야 이미지를 더 잘 생성할 수 있고 학습시킬 수 있을지 배운다.

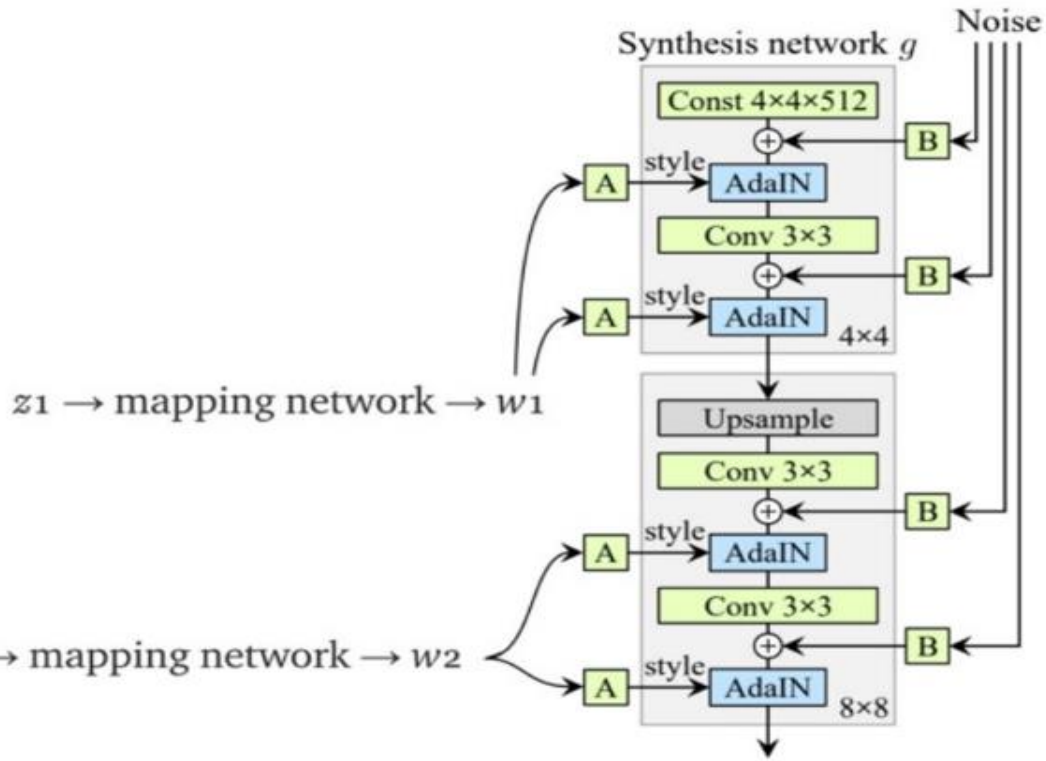
데이터의 전체적인 identity, 앞으로 style이 주입될 때 이미지를 생성하기 가장 좋은 스타팅 포인트를 구축

Why ‘learned constant’ ?





Style Mixing



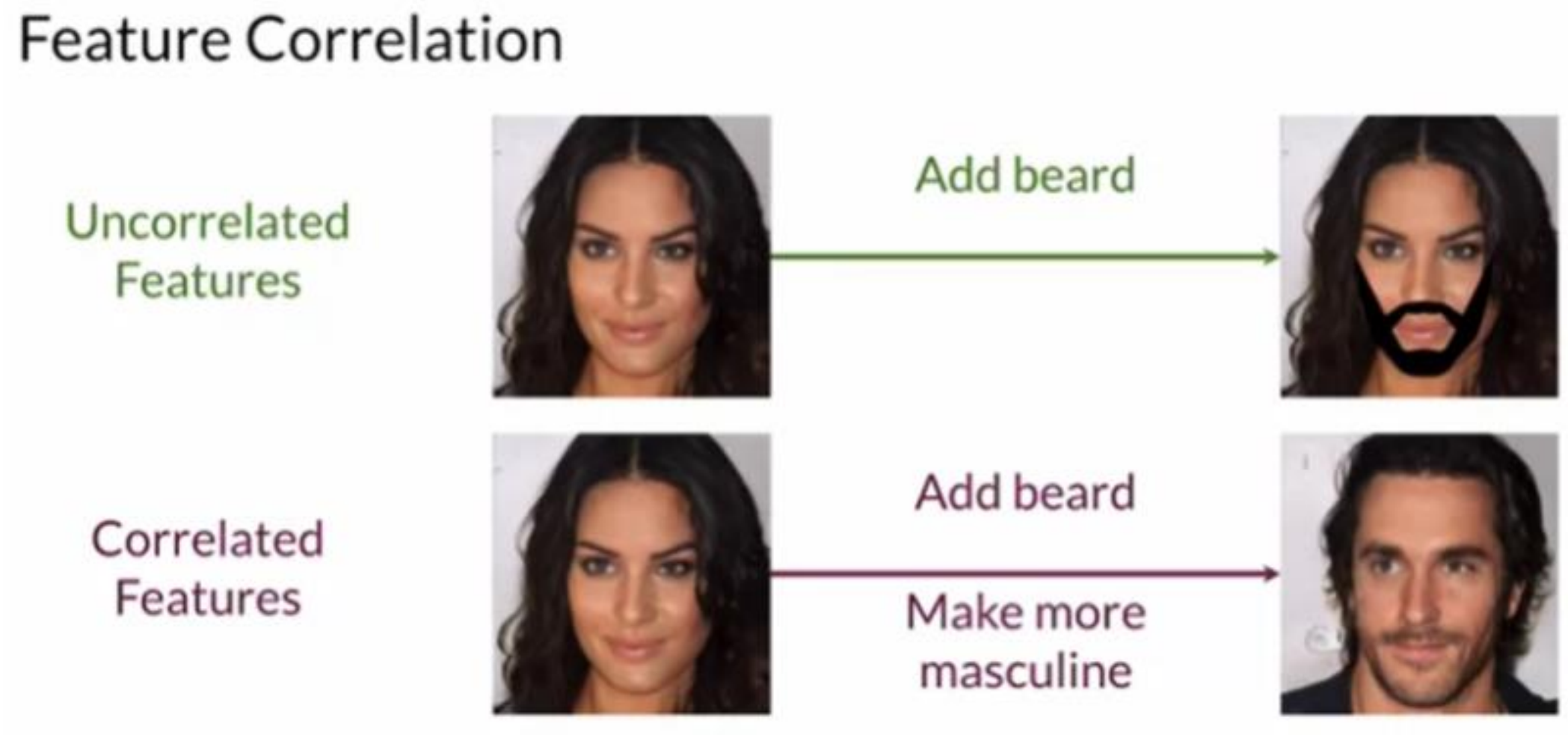
하나의 w로 학습하면 인접해 있는 level의 style 간의 correlation이 생길 수 있다.

이러한 상황을 줄이기 위해 Mixing Regularization을 한다.

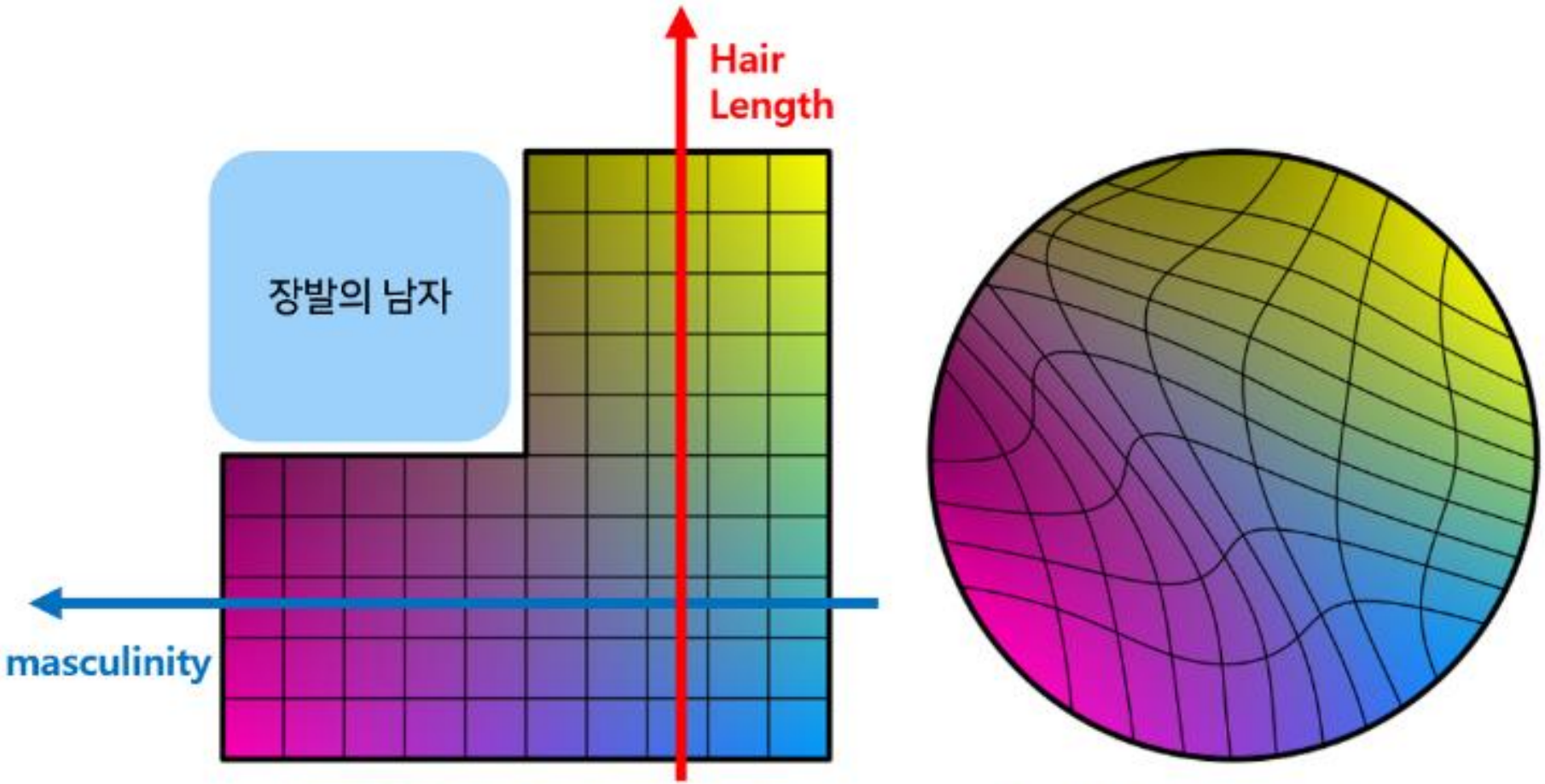
Mixing Regularization은 서로 다른 이미지를 생성하는 w1과 w2를 특정 level(crossover point)을 기준으로 이전까지는 w1의 style정보를, 이후부터는 w2의 style 정보를 넣어 이미지를 생성하도록 학습시키는 것

인접한 두 level의 style간의 correlatio이 생기는 것을 막도록하는 일종의 Regularization을 통해 localization이 개선되며 FID에서도 성능적 향상을 보인다.

Entanglement



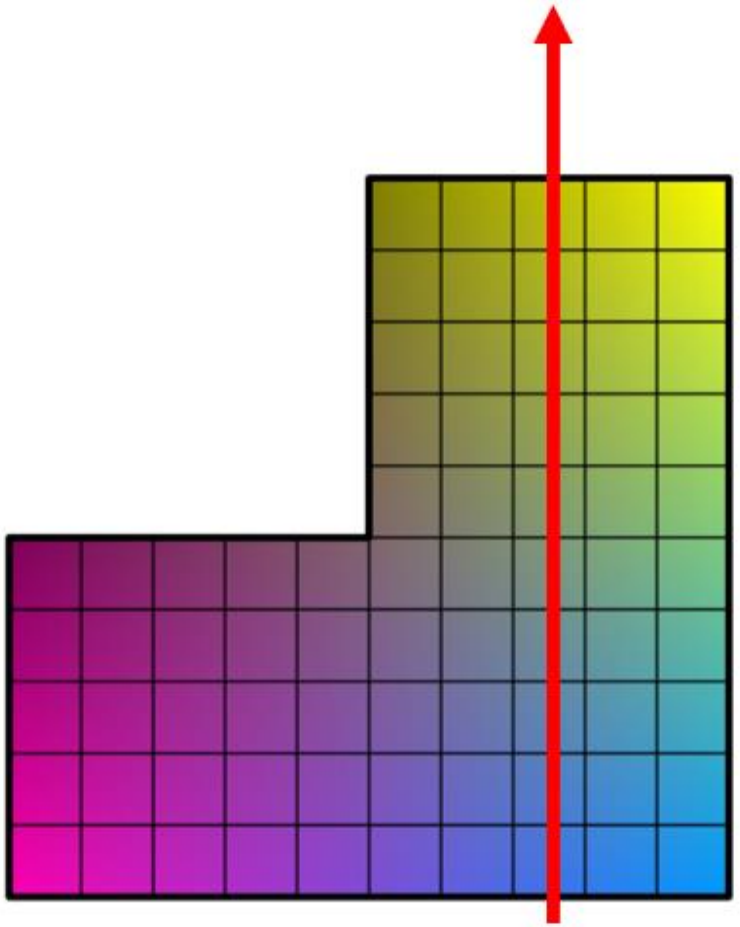
Entanglement



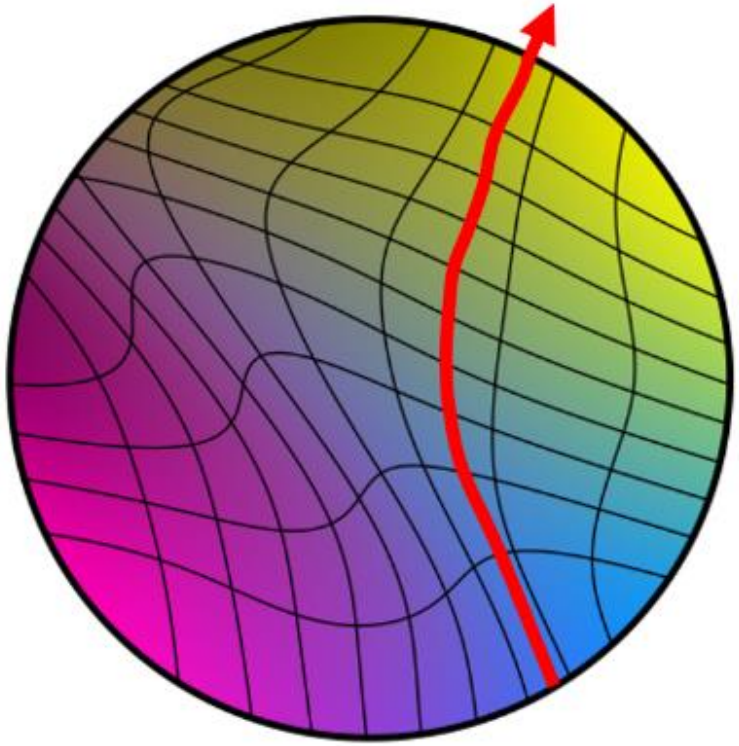
(a) Distribution of features in training set

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

Entanglement

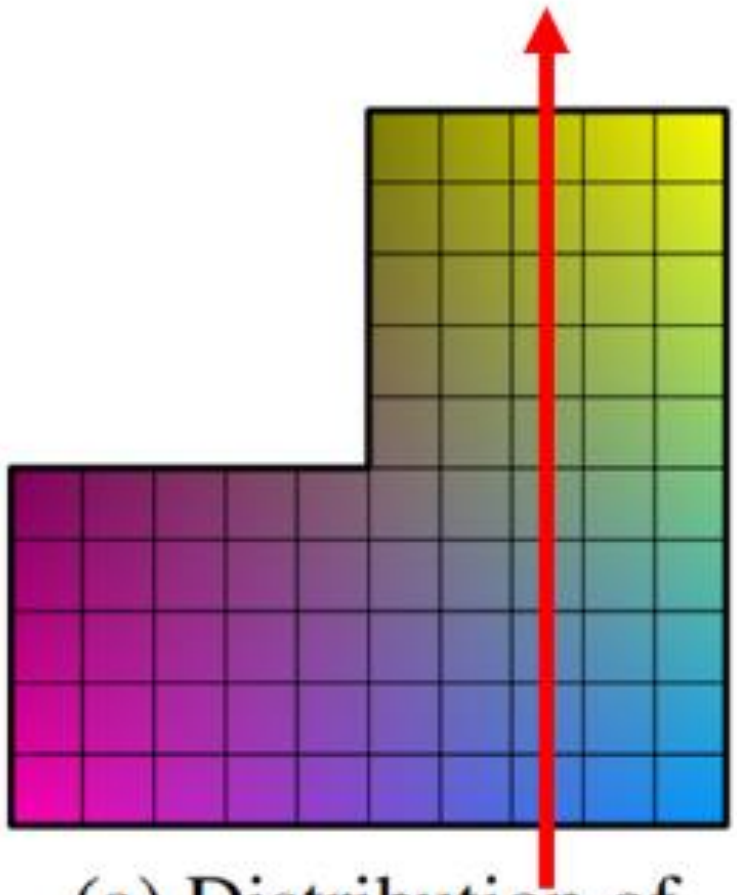


(a) Distribution of features in training set

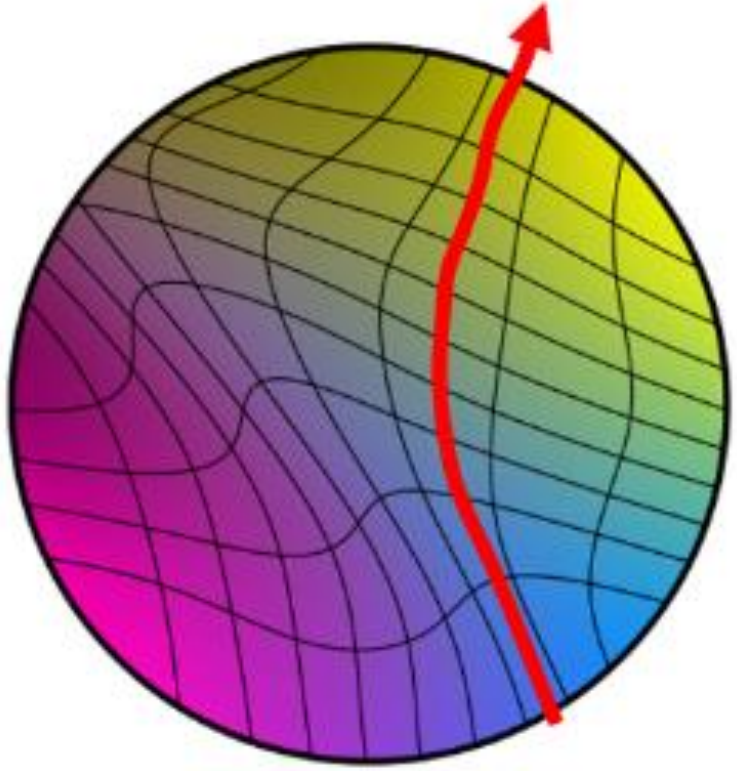


(b) Mapping from $\mathcal{Z}$ to features

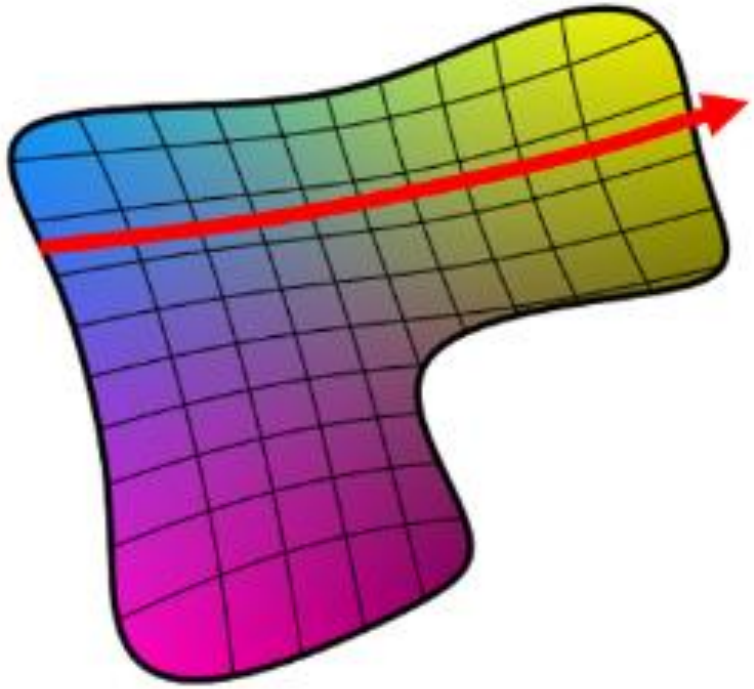
Disentanglement



(a) Distribution of features in training set

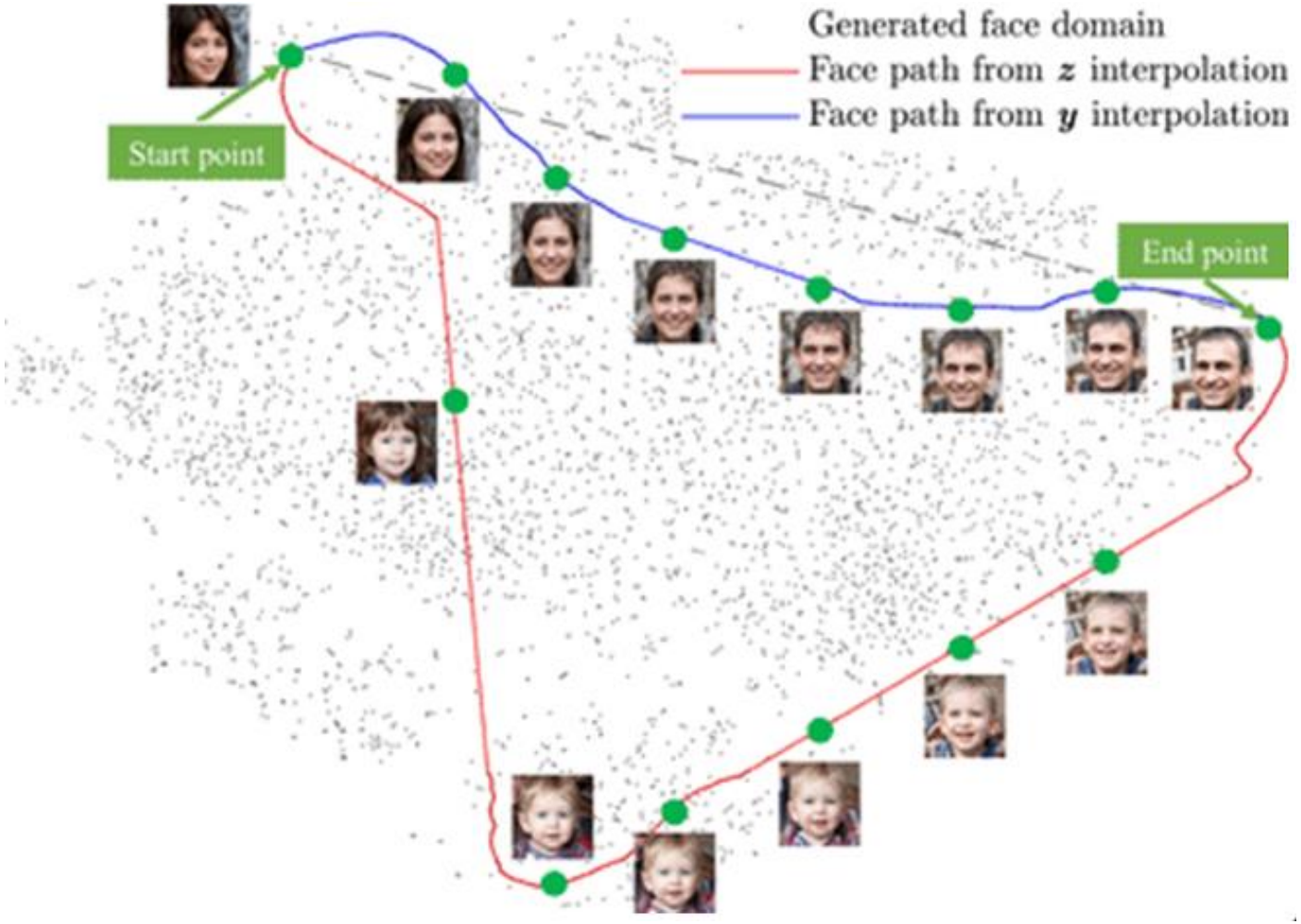


(b) Mapping from $\mathcal{Z}$ to features



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

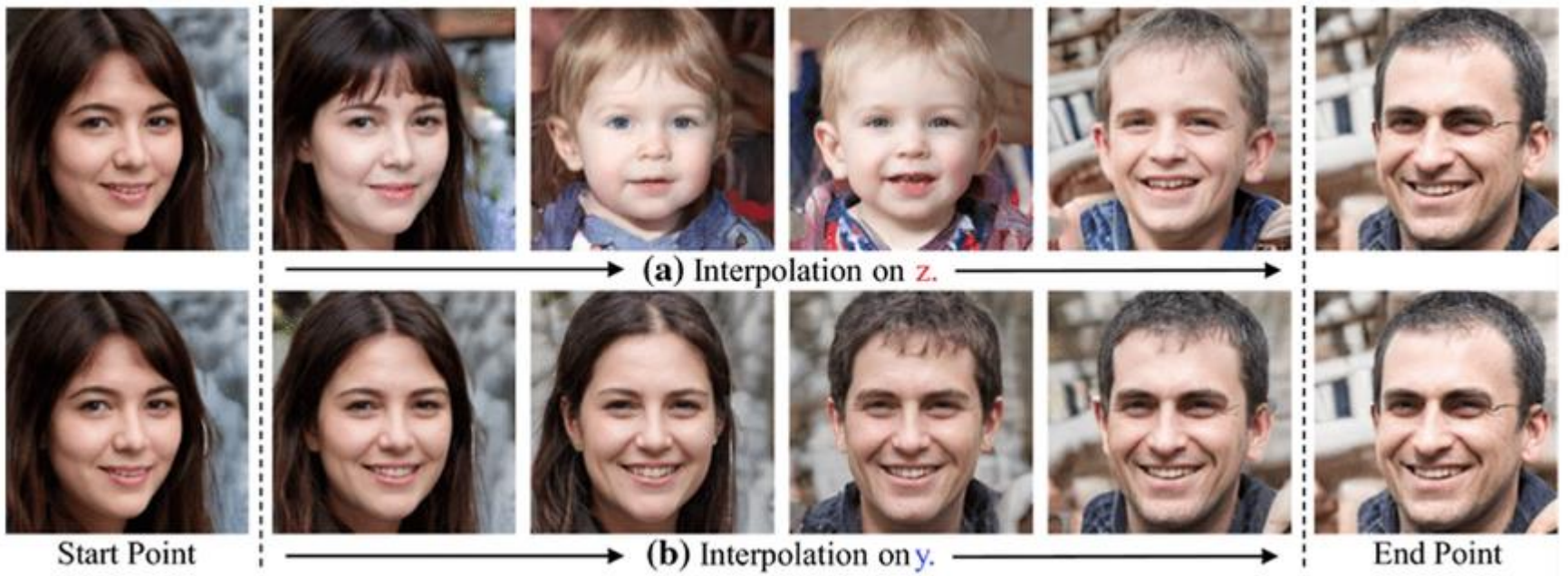
Disentanglement



각 latent space에서 interpolation된 point들을 generator를 통해 face image를 생성한 후 생성된 이미지 분포에서 해당 face image를 찍어가며 interpolation path를 비교한 결과

Latent space Z에서 interpolate한 얼굴 사진은 많이 왜곡되어서 길게 움직이게 된 반면, Disentangled한 latent space W의 interpolation은 실세 생성되는 이미지 분포에서도 선형적으로 유사한 길이만큼 움직이게 된다.

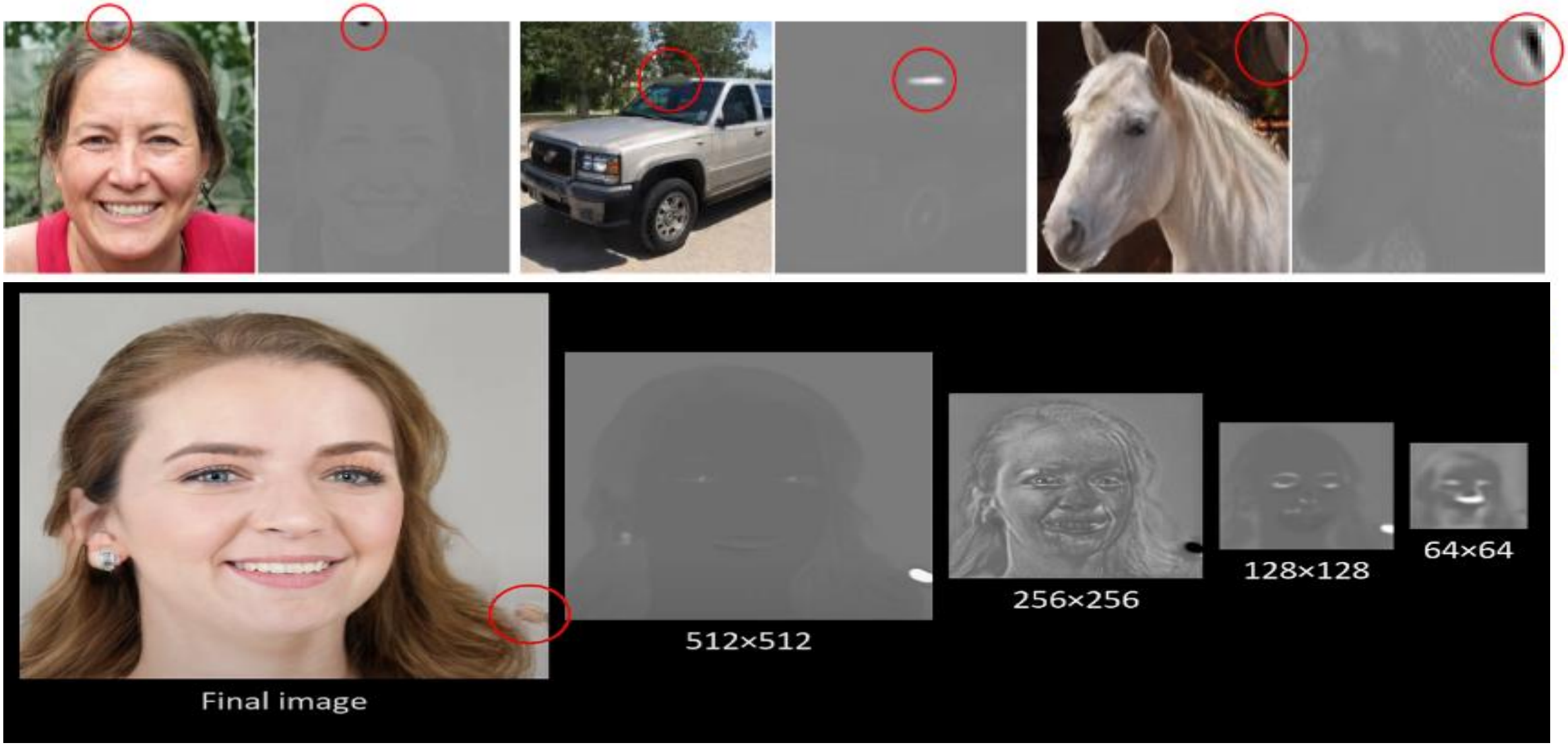
Disentanglement



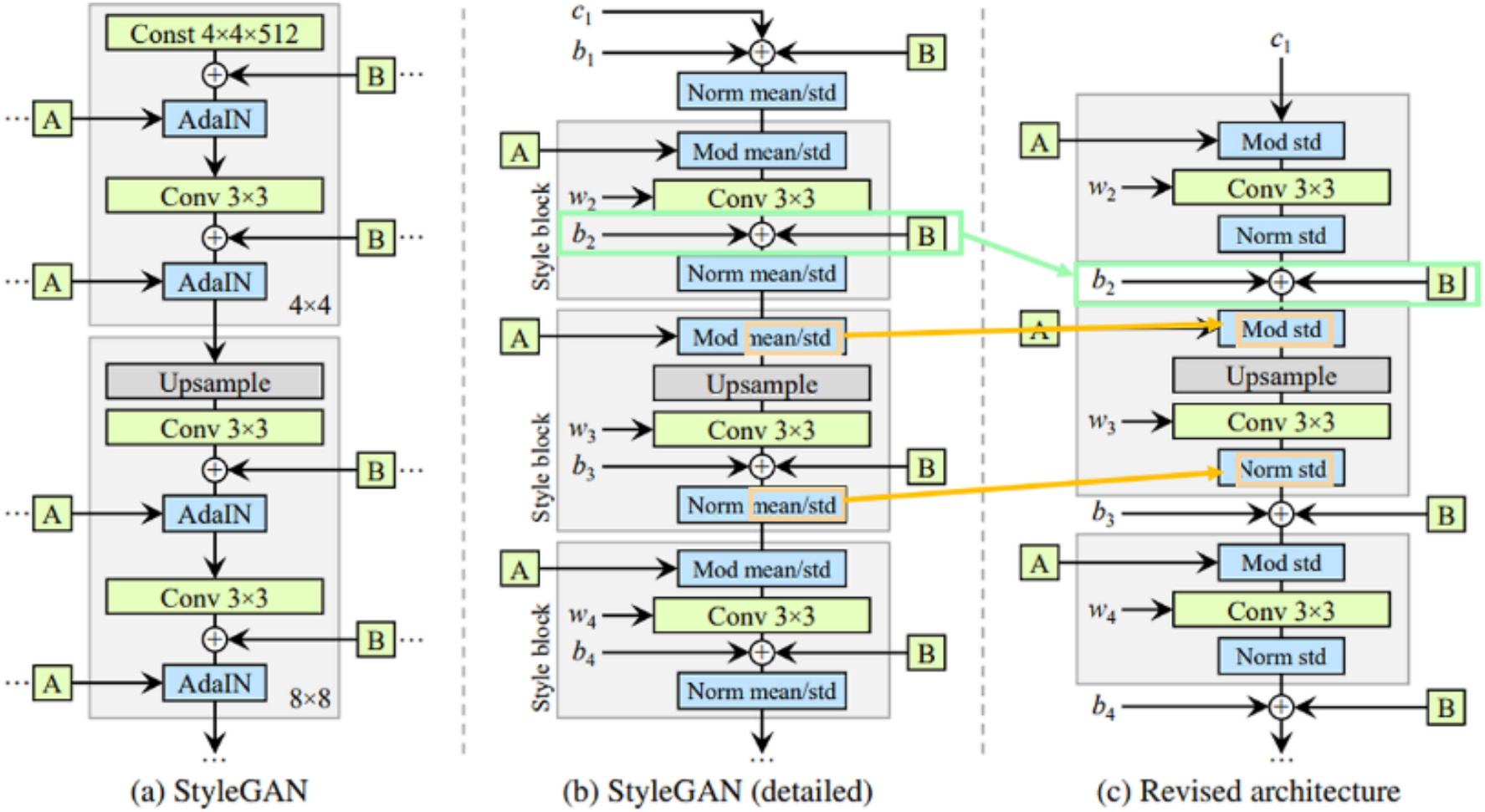
Latent space Z에서 interpolate한 얼굴 사진은 많이 왜곡되어서 빙 돌아 길게 움직이게 된 반면,
Disentangled한 latent space W의 interpolation은 실세 생성되는 이미지 분포에서도 선형적으로 유사한 길이만큼 움직이게 된다.

Style GAN 2

Style GAN의 문제점



Style GAN 2



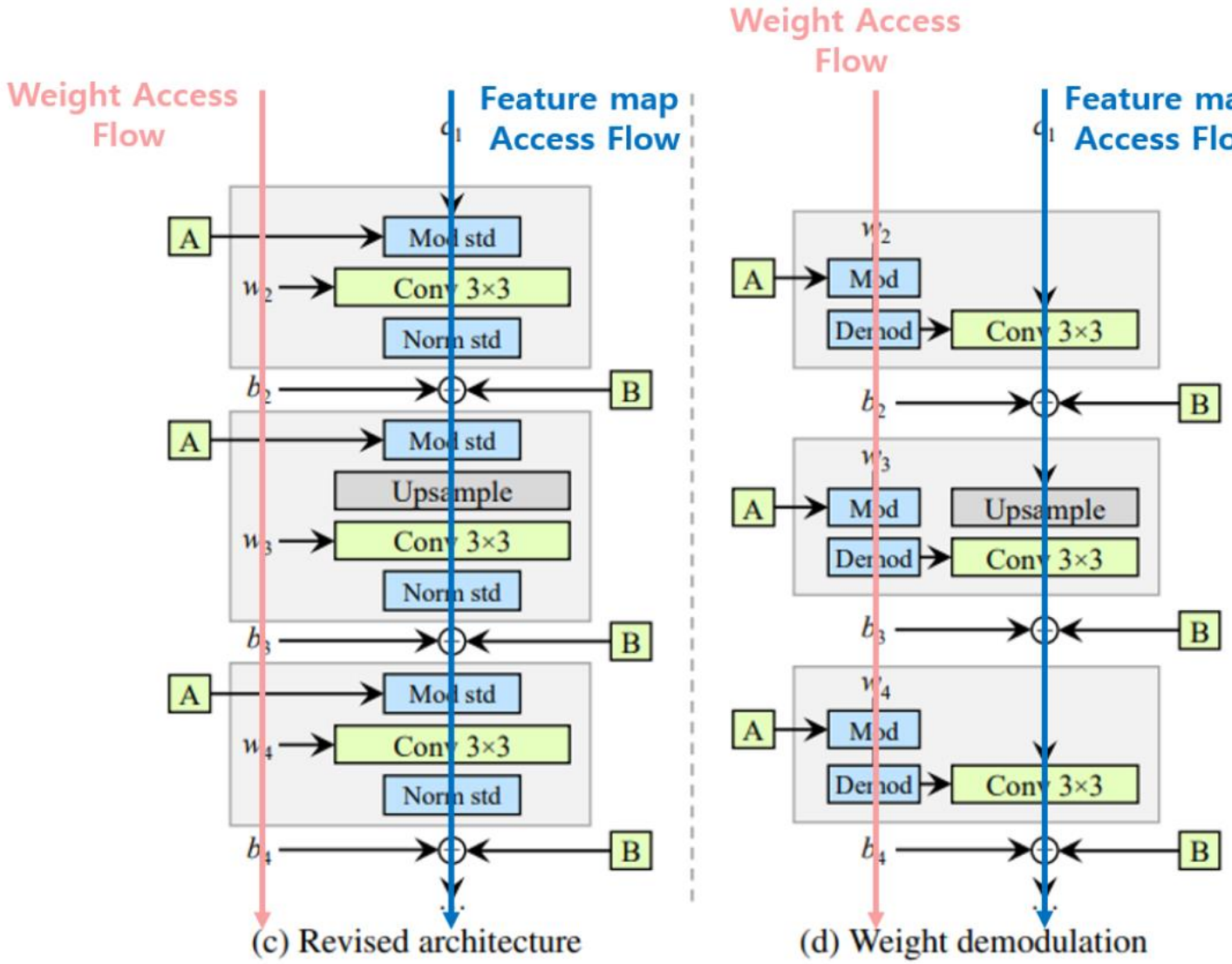
(a) StyleGAN (b) StyleGAN (detailed) (c) Revised architecture

AdaIN 단계에서 각 채널의 feature map들을 normalize 하는 과정에서 각 채널마다 다른 값으로 scaling이 되다가 나중에 채널을 합칠 때 범주를 벗어난 부분(물방울 artifact)이 생긴다.

Generator architecture를 개정해 normalization 문제를 해결함으로써 물방울 artifact 현상 개선

But 근본적인 해결이 아님.

Style GAN 2



각 채널의 feature map에 직접 정규화를 하다보니 feature map 간의 관계가 깨졌다.

Feature map이 아니라 conv layer의 가중치에 정규화를 해줌으로써 간접적으로 normalization

이를 통해 feature map 값의 증폭을 막으면서, normalization 효과

Style GAN 2

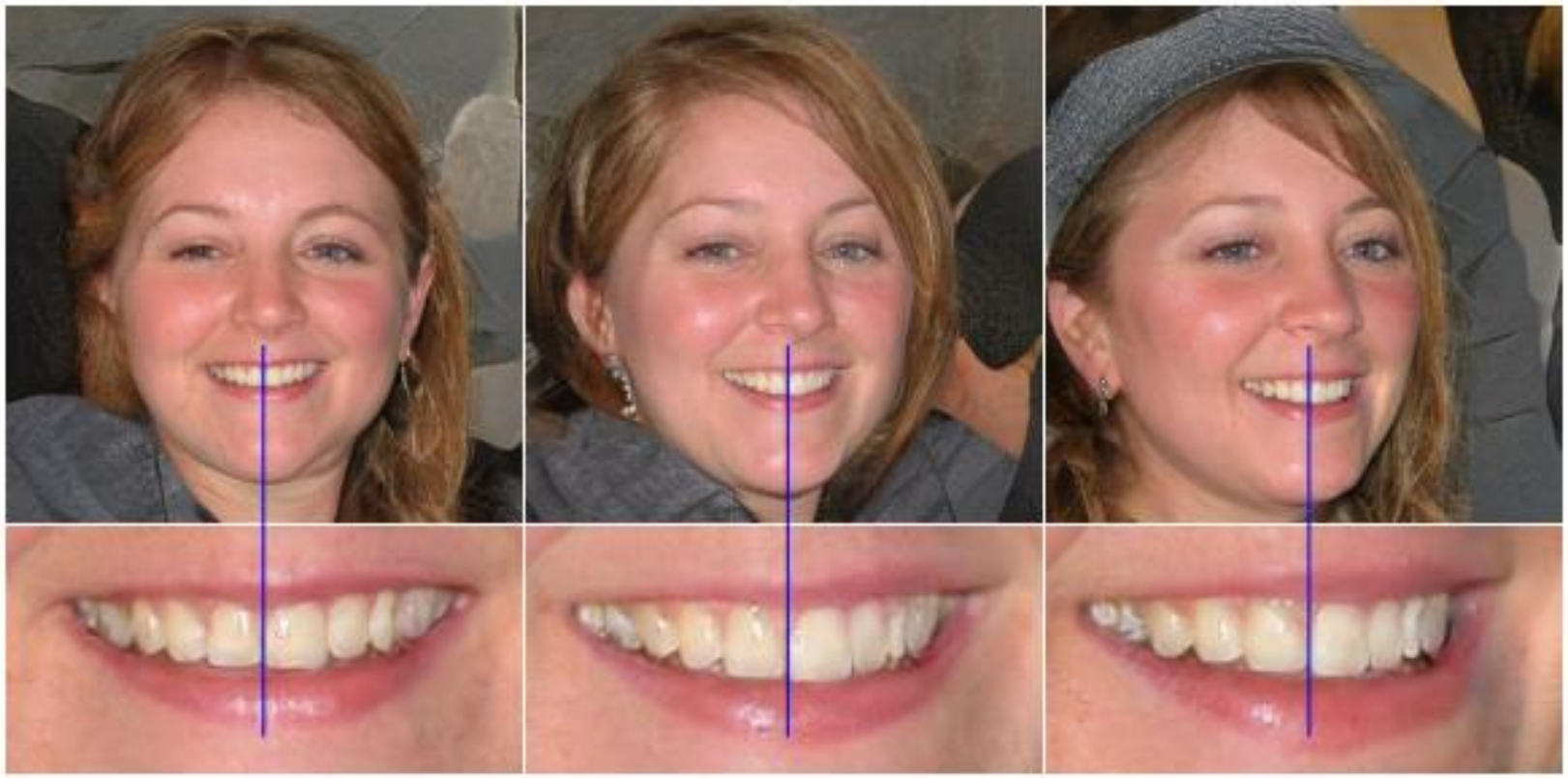
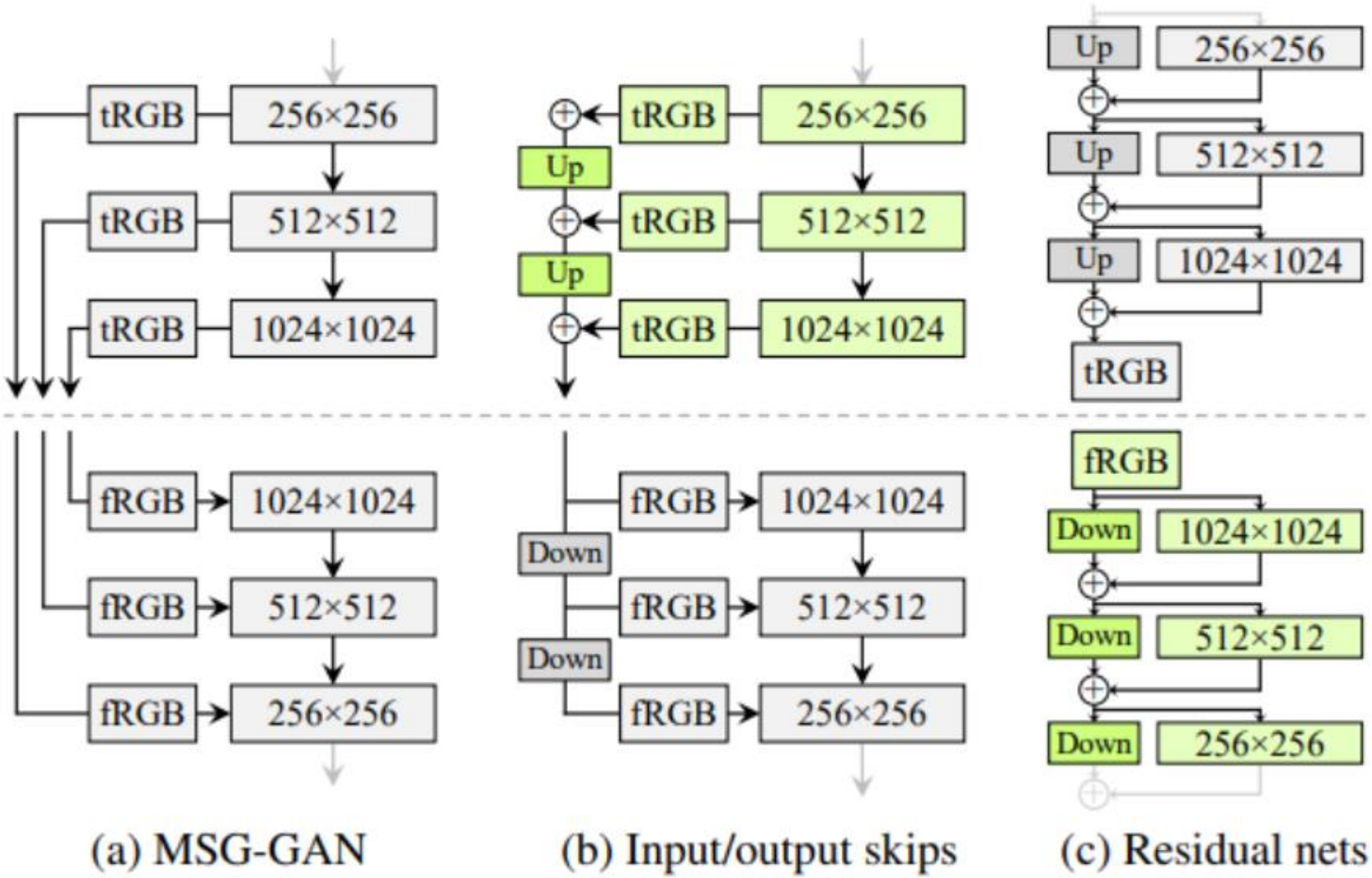


Figure 6. Progressive growing leads to "phase" artifacts. In this example the teeth do not follow the pose but stay aligned to the camera, as indicated by the blue line.

Phase artifact 발생
이빨과 같이 특정 부분이 이미지에서 카메라 pose를 변형시켜도 pose에 따라 변하지 않고, 이미지 상의 특정 부분에 완전히 align 되어 있어 고정된 것.

Progressive growing 방식은 저해상도에서 얻은 특징이 고해상도로 전파될 때 high frequency detail에 집중하기 때문에 최종적으로 보이는 이미지의 어떠한 특징이 한 부분에 고정된다는 부작용이 있다.

Style GAN 2



Artifact 해결을 위해 progressive growing을 사용하지 않고 고해상도 이미지를 생성해야 합니다.

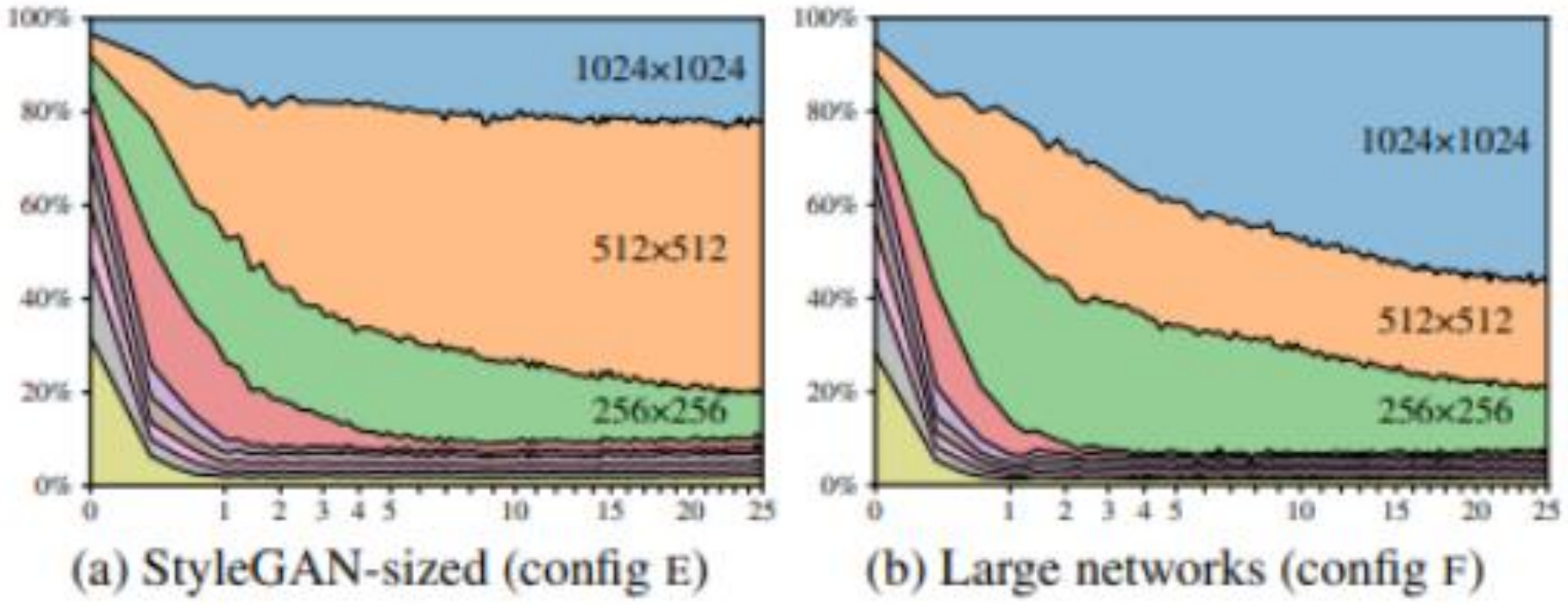
a : 생성자와 판별자의 skip connection

b : 각 해상도 단계의 rgb output을 업샘플링 혹은 다운 샘플링 한 후에 skip connection 을 통해 전달

C : 각 해상도 level 마다 residual connection

고해상도 이미지를 만드는 3가지 방법을 생성자와 판별자에 각각 적용해 생길 수 있는 총 9가지의 조합을 모두 실험한 결과,
생성자에서 output skip connection을 통해 PPL 개선
판별자에서 residual connnection을 통해 좋은 FID 결과를 얻을 수 있다.

Style GAN 2

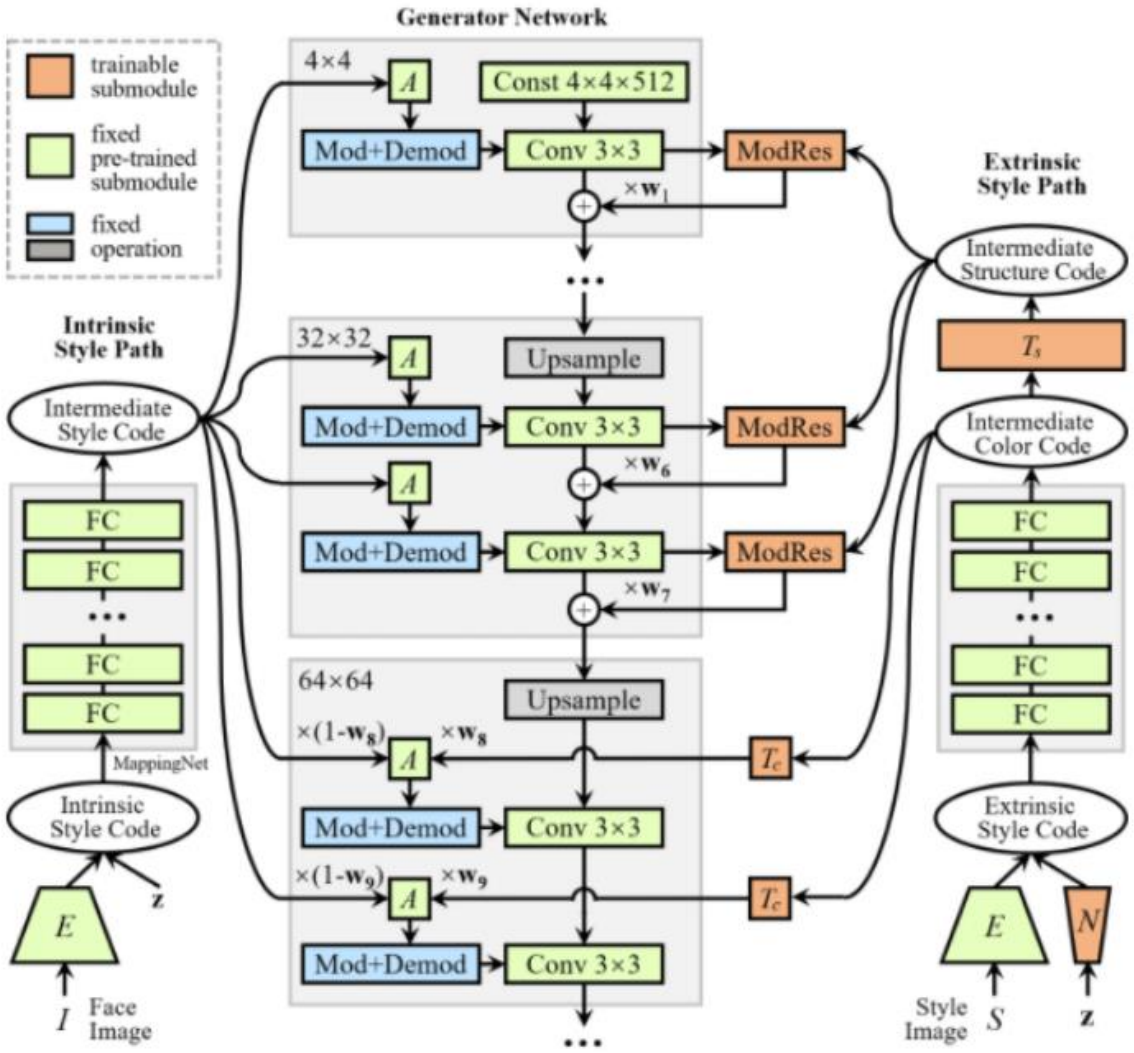


Progressive growing 을 대체한 새로운 아키텍처가 최종 해상도 level인 512x512에서 이미지의 pixel value를 거의 결정한다는 것을 파악.

Feature map의 수(trainable parameter)를 증가시킴으로써 개선

Dual Style GAN

Dual Style GAN



- latent z를 생성자 모델에 넣어 만든 이미지를 사용하던 기존의 방식에서 사용자가 입력한 얼굴 이미지를 딥러닝 인코더를 사용해 스타일 코드를 만든다.
- 내재적 스타일과 외재적 스타일, 2개의 스타일이 합쳐진다.
- 외재적 스타일은 저해상도 단계에서 구조적 스타일을 담고 고해상도 단계에선 style GAN과 같은 색상적 스타일을 담고 있다.

코드구현

```python
class DualStyleGAN(nn.Module):
    def __init__(self,
                 size,                  # 생성될 이미지의 해상도를 지정
                 style_dim,             # 스타일 벡터의 차원을 설정
                 n_mlp,                 # MLP(다층 퍼셉트론)의 층 수를 설정
                 channel_multiplier=2,  # 각 층의 채널 수를 조정하는 계수
                 twoRes=True,           # 두 가지 종류의 스타일 변형 경로를 사용할지 여부를 결정하는 플래그
                 res_index=6):          # 스타일 변형이 적용되는 층의 인덱스를 결정
        super().__init__()              # Python의 상속 기능을 사용하여 nn.Module 클래스의 생성자를 호출함
                                        # 이를 통해 DualStyleGAN이 nn.Module의 모든 속성과 메소드를 상속받아 사용할 수 있음

        layers = [PixelNorm()]          # 스타일 벡터를 정규화하는데 사용되는 PixelNorm 레이어를 리스트에 초기 요소로 추가

        for i in range(n_mlp-6):    # n_mlp에서 6을 빼는 반복문을 설정하고, 각 반복에서 EqualLinear 레이어를 layers 리스트에 추가
            layers.append(EqualLinear(512, 512, lr_mul=0.01, activation="fused_lrelu")) # 학습률을 조정하는 lr_mul
        # structure transform blocks T_s
        self.style = nn.Sequential(*layers)            # 이 층들은 스타일 벡터를 정규화하고 변환하는 데 사용
        # StyleGAN2
        self.generator = Generator(size, style_dim, n_mlp, channel_multiplier)  # 주어진 스타일 벡터를 바탕으로 이미지를 생성
        # The extrinsic style path
        self.res = nn.ModuleList()  # 이미지의 특정 층에 대해 스타일 변형을 적용하는 AdaResBlock 블록들과 색상 변형을 적용하는 EqualLinear 층들로 구성
        self.res_index = res_index//2 * 2       # 주어진 res_index를 2로 나누고 다시 2를 곱하여, res_index가 짝수가 되도록 조정
        self.res.append(AdaResBlock(self.generator.channels[2 ** 2])) # for conv1
        for i in range(3, self.generator.log_size + 1):
            out_channel = self.generator.channels[2 ** i]
            if i < 3 + self.res_index//2:
                # ModRes
                self.res.append(AdaResBlock(out_channel))          # AdaResBlock은 주어진 채널 크기에 맞게 스타일을 적응적으로 조정
                self.res.append(AdaResBlock(out_channel))
            else:
                # color transform block T_c
                self.res.append(EqualLinear(512, 512))             # EqualLinear는 색상 변형을 담당
                # FC layer is initialized with identity matrices, meaning no changes to the input latent code
                self.res[-1].weight.data = torch.eye(512) * 512.0**0.5 + torch.randn(512, 512) * 0.01
                self.res.append(EqualLinear(512, 512))
                self.res[-1].weight.data = torch.eye(512) * 512.0**0.5 + torch.randn(512, 512) * 0.01

        self.res.append(EqualLinear(512, 512)) # for to_rgb7               # 생성자의 다양한 구성 요소와 설정을 클래스 내부 변수로 저장
        self.res[-1].weight.data = torch.eye(512) * 512.0**0.5 + torch.randn(512, 512) * 0.01
        self.size = self.generator.size
        self.style_dim = self.generator.style_dim
        self.log_size = self.generator.log_size
        self.num_layers = self.generator.num_layers
        self.n_latent = self.generator.n_latent
        self.channels = self.generator.channels

        # DualStyleGAN 클래스는 StyleGAN2 아키텍처를 기반으로 하는 이미지 생성기로서,
        # 다층 퍼셉트론(MLP)을 이용해 스타일 벡터를 변환하고,
        # 추가적인 스타일 및 색상 변형 경로를 통해 이미지의 스타일과 색상을 세밀하게 조절할 수 있도록 설계

        # 부가 설명
        # 이 클래스는 스타일 변형을 위한 AdaResBlock과 색상 변형을 위한 EqualLinear 블록을 사용하여 생성된 이미지의 세부 스타일을 조정하고,
        # 전체적으로 더 정교하고 맞춤화된 이미지 생성을 가능하게 함
```

결과

Epoch = 1000

시간 : 138분

레퍼런스

• GAN	2014	Ian Goodfellow
• PGGAN	2017	Tero Karras
• StyleGAN	2018	Tero Karras
• StyleGAN2	2019	Tero Karras
• Dual Style GAN	2022	Shuai Yang

- https://hyoseok-personality.tistory.com/entry/Paper-Review-StyleGAN2-Analyzing-and-Improving-the-Image-Quality-of-StyleGAN
- https://hyoseok-personality.tistory.com/entry/StyleGAN-A-Style-Based-Generator-Architecture-for-Generative-Adversarial-Networks-%EB%A6%AC%EB%B7%B0-1%EC%84%B8%EC%84%B8%ED%95%9C-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0

Q&A