

Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

ICCV (2017)

2022.01.15 이지현

GAN 모델 주요 개념

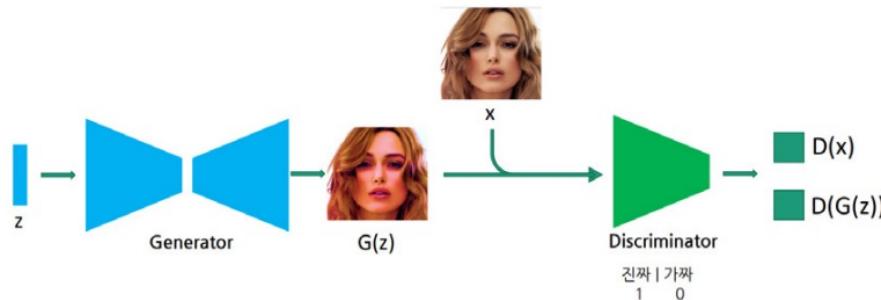
GAN

"Generative Adversarial Nets", Ian Goodfellow (2014)

딥러닝 모델 중 이미지 생성에 널리 쓰이는 모델.
즉, GAN 모델은 데이터셋과 유사한 이미지를 만들도록 학습.



GAN 이미지 편집 및 생성 활용 사례



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

수식 (minimax two-player game)

- 첫 번째 항: real data x 를 discriminator 에 넣었을 때, 나오는 결과를 log 취한 기댓값
- 두 번째 항: fake data z 를 generator 에 넣은 결과 (생성된 가짜 이미지) 를 discriminator 에 넣었을 때, 그 결과를 log 취한 기댓값

D 의 입장) 구별을 잘 하면 좋음. $D(x) = 1, D(G(z)) = 0$ 일 때, 최댓값은 0 (max)

G 의 입장) 구별을 잘 못하면 좋음. $D(x) = 0, D(G(z)) = 1$ 일 때, 최솟값은 -무한대 (min)

'경쟁' 하는 과정을 통해 generative model 을 추정하는 프레임워크 (two-player minmax game)

생성 모델 (G)과 판별 모델 (D), 두 가지 모델을 학습하며,

G 는 실제 training data 의 분포를 모사하며 그와 비슷한 데이터를 생성하려 하고, D 는 실제 데이터와 G 가 생성한 데이터를 구별하려는 경쟁적인 과정으로 이루어져 있음.

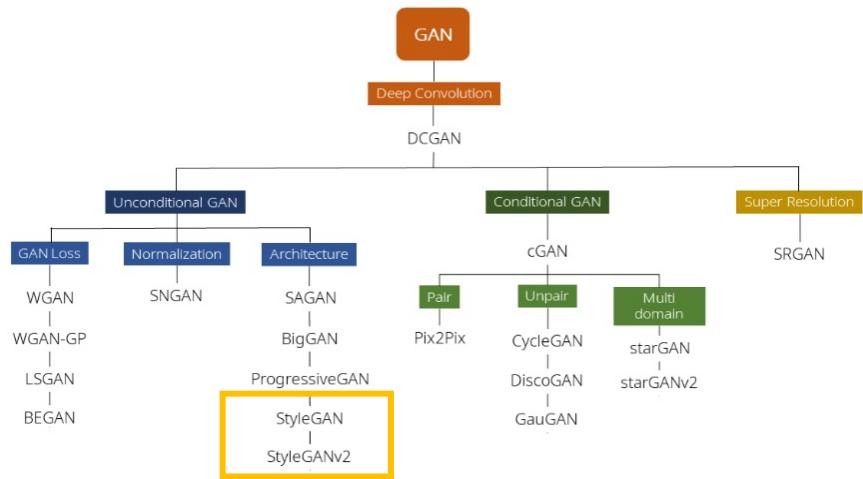
1) Generative model (생성 모델), G : training data 의 분포를 모사. Discriminative model 이 구별하지 못하도록 함.

2) Discriminative model (판별 모델) D : sample 데이터가 G 로부터 나온 데이터가 아닌 실제 training data 로부터 나온 데이터일 확률 추정.

GAN 모델 주요 개념

GAN의 발전

GAN과 관련된 연구가 폭발적으로 증가하고 있음.
종류- GAN, DCGAN, StyleGAN, WGAN, CycleGAN



StyleGAN

이미지 style 을 scale-specific control 가능하게 하는 모델 아키텍처.

- Style transfer

두 이미지 (content image & style image) 가 주어졌을 때,
그 이미지의 주된 형태는 content image 와 유사하게 유지하면서,
스타일만 우리가 원하는 style image 와 유사하게 바꾸는 것.

StyleGAN 초기 연구

"Image Style Transfer Using Convolutional Neural Networks"



StyleGAN 활용 예시

Content 이미지와 Style 이미지를 네트워크에 통과시킬 때 나온 각각의 feature map 을 저장하고, 새롭게 합성될 영상의 feature map 이 content 이미지와 style 이미지로부터 나온 feature map 과 비슷한 특성을 가지도록 최적화.

- 장점: 이미지 2장 (content image & style image) 으로 style transfer 가 가능.
- 단점: 매번 이미지를 새롭게 최적화 해야 하므로 시간이 오래 걸림.

StyleGAN | Batch Normalization

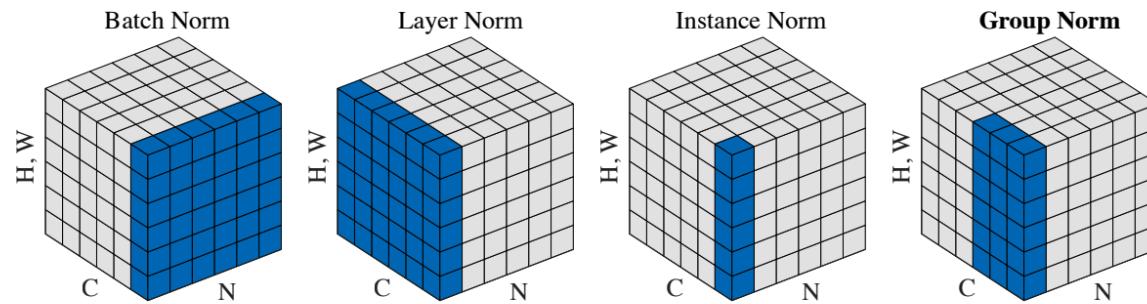
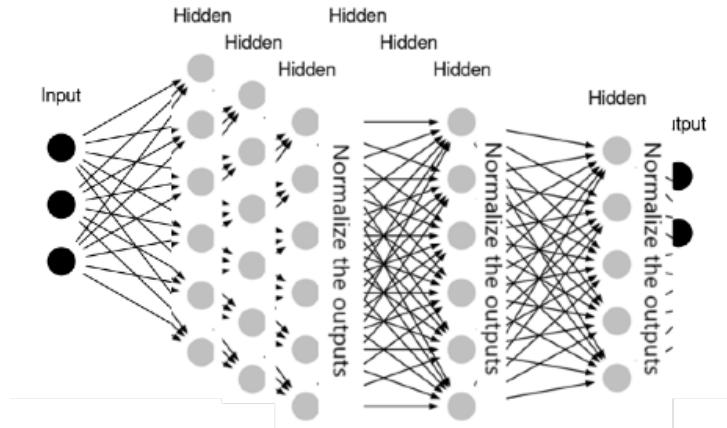


Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.



1. Batch normalization

- 채널별로 (channel-wise) 정규화 수행

$$\text{BN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

$$\mu_c(x) = \frac{1}{NHW} \sum_{n=1}^N \sum_{h=1}^H \sum_{w=1}^W x_{nchw}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW} \sum_{n=1}^N \sum_{h=1}^H \sum_{w=1}^W (x_{nchw} - \mu_c(x))^2 + \epsilon}$$

2. Instance normalization

- 개별적인 이미지 (인스턴스) 대하여 각 채널별로 (channel-wise) 정규화 수행
- 개별적인 샘플(이미지)에 대하여 수행한다는 점이 BM 과의 차이점
- Style transfer 수행하는 생성 모델에 사용되었을 때, 성능을 높일 수 있음.

$$\text{IN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

$$\mu_{nc}(x) = \frac{1}{HW} \sum_{h=1}^H \sum_{w=1}^W x_{nchw}$$

$$\sigma_{nc}(x) = \sqrt{\frac{1}{HW} \sum_{h=1}^H \sum_{w=1}^W (x_{nchw} - \mu_{nc}(x))^2 + \epsilon}$$

StyleGAN | 조건부 인스턴스 정규화 (Conditional Instance Normalization)

3. Conditional Instance Normalization (임의의 style transfer)

Instance Normalization (IN) 을 수행할 때, 조건에 따라 다른 feature statistics 를 갖도록 함.
각각의 스타일마다 서로 다른 감마와 베타값을 사용하자.

- 서로 다른 affine parameters 를 적용해, 완전히 다른 스타일 (style) 생성 가능
 - Dumulin 의 논문에서 CIN 을 제안하였고, 32개의 스타일(s)에 대하여 성공적으로 학습.

$$\text{CIN}(x; s) = \gamma^s \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta^s$$



CIN을 이용한 스타일 전송(style transfer)을 위해

2FS만큼의 추가적인 파라미터가 필요합니다.

*F: Feature Map의 개수



전체 feature map 의 수 * 스타일의 수 * 2 (채널마다 감마와 베타)

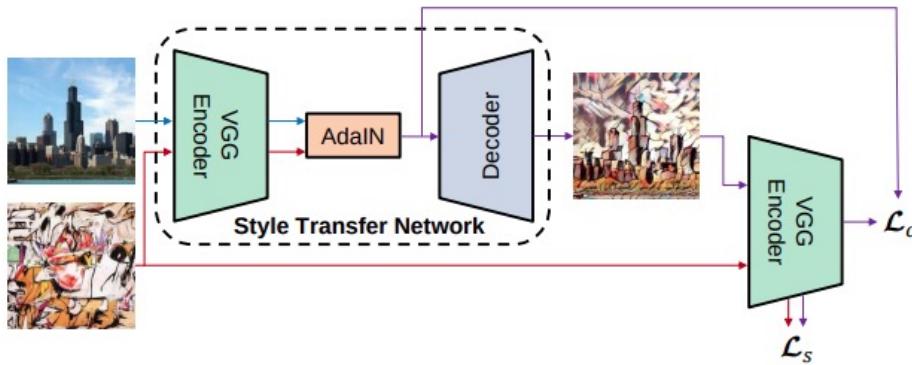
StyleGAN | AdaIN Style Transfer

"Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization" (ICCV 2017)

AdaIN을 이용하여 이미지 특징 (feature) 의 statistics 를 변경할 수 있으며, 이를 이용해 스타일 전송 (style transfer) 진행.

장점

1. 자신이 원하는 임의의 (arbitrary) 스타일 이미지로부터 스타일 정보를 가져올 수 있음.
2. 실시간으로 (real-time) 빠르게 스타일 전송 (style transfer) 진행할 수 있음. (한 번의 Feed-Forward)



Statistics information 이란?

- 간단하게 평균 (mean) 과 분산 (variance).
- 초기 연구인 Gatys 방식에서 VGG 에서 나온 feature 들에 Gram Matrix 를 사용해서 Style representation 이 가능함을 실험적으로 보임.
- Gram Matrix 가 대표적인 feature space 상의 statistics 를 추출해내는 방법인데,
- 이후 많은 연구가 이루어지면서 feature space 상, 여러 statistics 가 Style 을 표현하는데 유용하다는 것이 실험적으로 많이 밝혀짐.

본 논문에서는 feature statistics 를 normalizing 시킴으로써, Instance Normalization 이 style normalization 을 수행한다고 함.

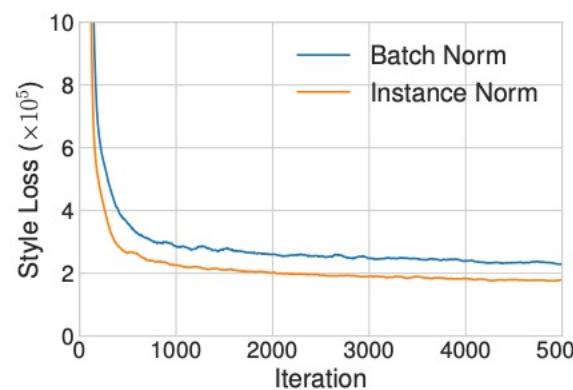
- 하나의 네트워크를 학습시킨 이후에, 어떠한 이미지에서도 바로 스타일 정보를 추출해서 적용할 수 있음.
- 이미지의 Style 과 Contents 에 대한 Statistics 정보를 VGG Encoder 에서 추출.
- **Style Transfer 진행 (AdaIN)**
 - 내가 원하는 Contents 를 담고 있는 이미지의 feature x 에서, 이미지의 스타일을 빼주고,
 - 내가 입하고 싶은 Style 을 더해줌.

StyleGAN | AdaIN Style Transfer

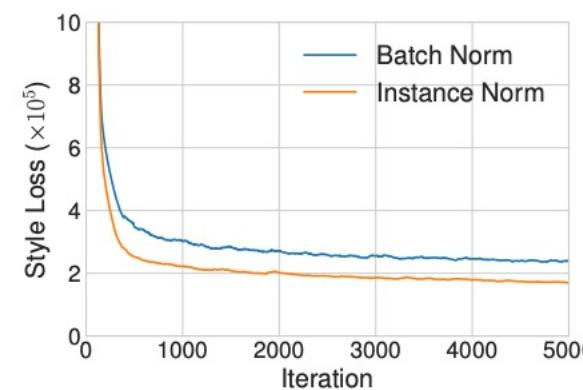
Statistics information 이란?

- 간단하게 평균 (mean) 과 분산 (variance).
- 초기 연구인 Gatys 방식에서 VGG 에서 나온 feature 들에 Gram Matrix 를 사용해서 Style representation 이 가능함을 실험적으로 보임.
- Gram Matrix 가 대표적인 feature space 상의 statistics 를 추출해내는 방법인데,
- 이후 많은 연구가 이루어지면서 feature space 상, 여러 statistics 가 Style 을 표현하는데 유용하다는 것이 실험적으로 많이 밝혀짐.

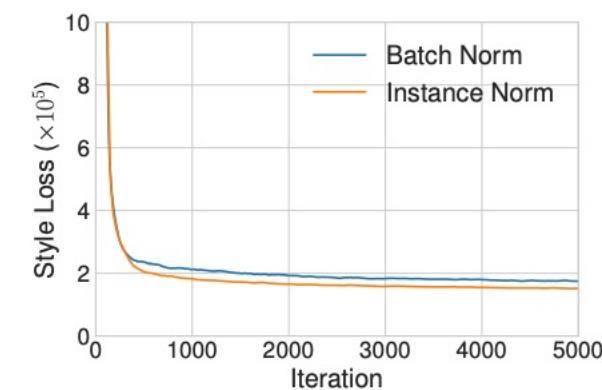
본 논문에서는 feature statistics 를 normalizing 시킴으로써, Instance Normalization 이 style normalization 을 수행한다고 함.



(a) Trained with original images.



(b) Trained with contrast normalized images.



(c) Trained with style normalized images.

Figure 1. To understand the reason for IN's effectiveness in style transfer, we train an IN model and a BN model with (a) original images in MS-COCO [36], (b) contrast normalized images, and (c) style normalized images using a pre-trained style transfer network [24]. The improvement brought by IN remains significant even when all training images are normalized to the same contrast, but are much smaller when all images are (approximately) normalized to the same style. Our results suggest that IN performs a kind of style normalization.

StyleGAN | AdaIN Style Transfer

4. Adaptive Instance Normalization

기존의 감마와 베타였던 파라미터 값 -> 다른 이미지의 임의의 스타일

$$\text{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y) \quad (8)$$

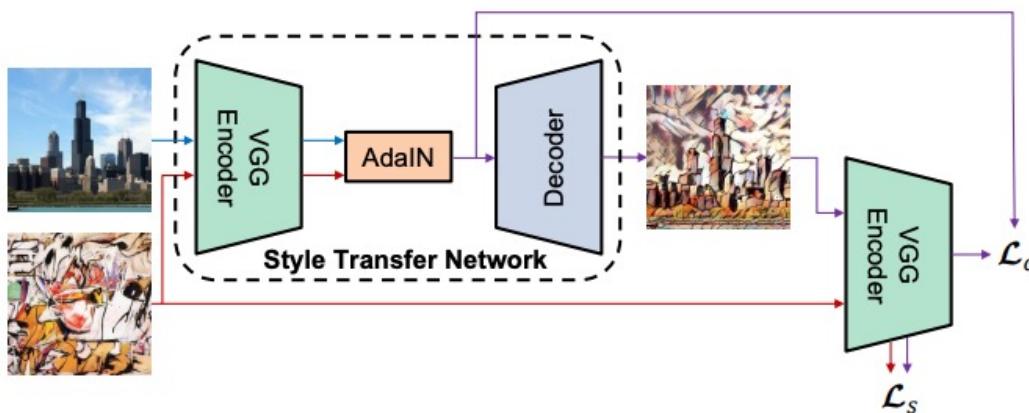


Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same VGG encoder to compute a content loss \mathcal{L}_c (Equ. 12) and a style loss \mathcal{L}_s (Equ. 13).

x: 원본 이미지

y: 스타일 이미지

Style 이미지로부터 feature statistics 정보를 추출한 후, Style transfer

Encoder

- pretrained VGG network (중간에 있는 특정 layer 까지만 사용)
- 특징 추출 목적으로 사전 학습되어 고정된 (fixed) 네트워크

Decoder

- 학습할 네트워크
- 결과 이미지를 생성

AdaIN layer

- style transfer 그 자체를 수행하는 레이어

$$t = \text{AdaIN}(f(c), f(s))$$

(9) Style Transfer 를
수행한 feature 정보

A randomly initialized decoder g is trained to map t back to the image space, generating the stylized image $T(c, s)$:

$$T(c, s) = g(t) \quad (10)$$

디코더 (Decoder) 를 거쳐 생성된 결과 이미지

StyleGAN | AdaIN Style Transfer

4. Adaptive Instance Normalization

Loss function

- 최종 목적 함수: $\mathcal{L} = \mathcal{L}_c + \lambda \mathcal{L}_s$
- 콘텐츠 손실: $\mathcal{L}_c = \|f(g(t)) - t\|_2$
- 스타일 손실: $\mathcal{L}_s = \sum_{i=1}^L \|\mu(\underline{\phi_i(g(t))}) - \mu(\phi_i(s))\|_2 + \sum_{i=1}^L \|\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))\|_2$
i번째 스타일 feature

$$t = \text{AdaIN}(f(c), f(s)) \quad (9) \quad \begin{array}{l} \text{Style Transfer 를} \\ \text{수행한 feature 정보} \end{array}$$

A randomly initialized decoder g is trained to map t back to the image space, generating the stylized image $T(c, s)$:

$$T(c, s) = g(t) \quad (10)$$

디코더 (Decoder) 를 거쳐 생성된 결과 이미지

StyleGAN | AdaIN Style Transfer

Experiments

- Dataset
 - MS-COCO
 - 80,000 training examples
- Optimizer
 - adam
- Augmentation
 - Resize to 512 while preserving the aspect ratio
 - Randomly crop region sof size 256*256

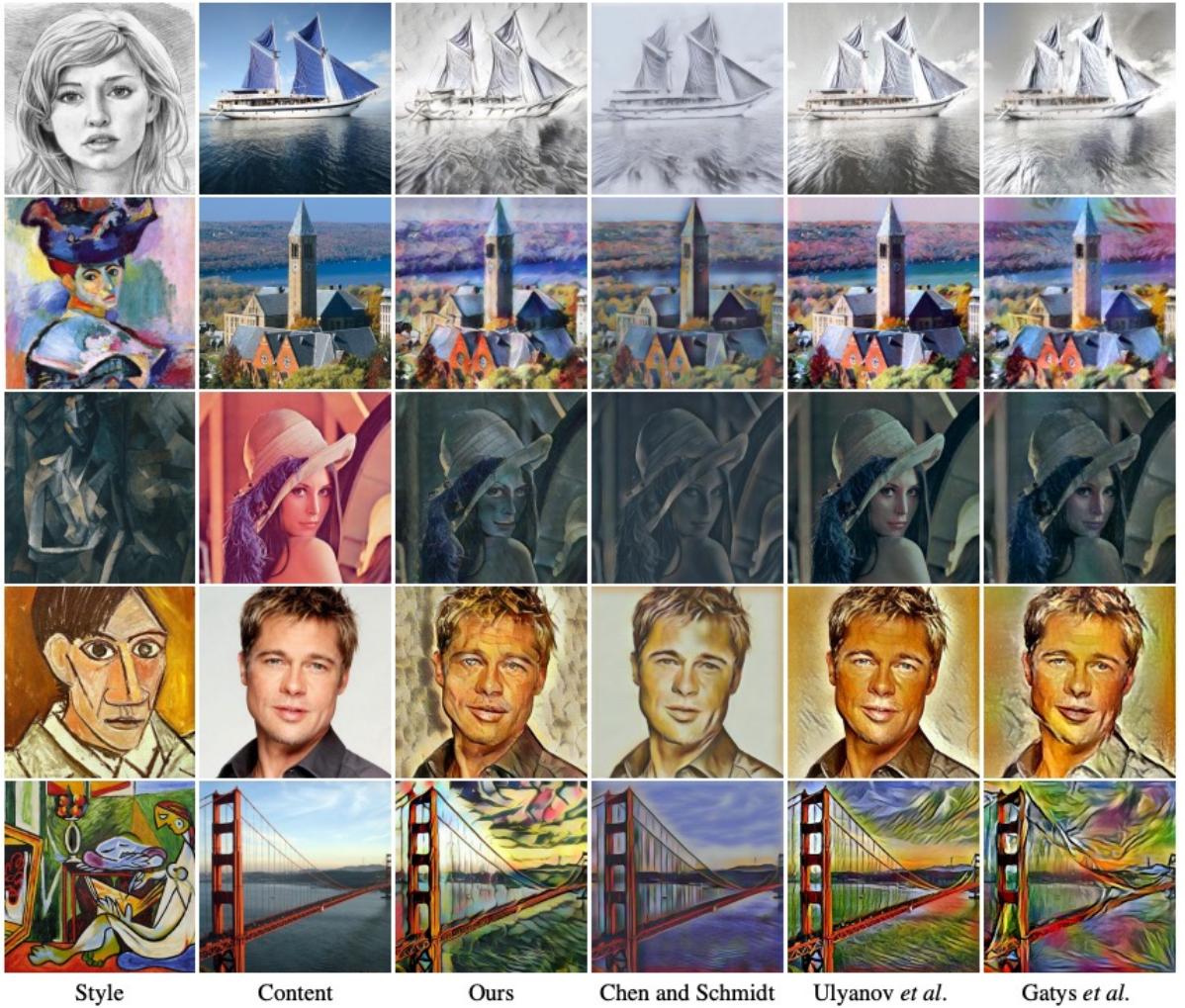


Figure 4. Example style transfer results. All the tested content and style images are never observed by our network during training.

StyleGAN | AdaIN Style Transfer

Experiments

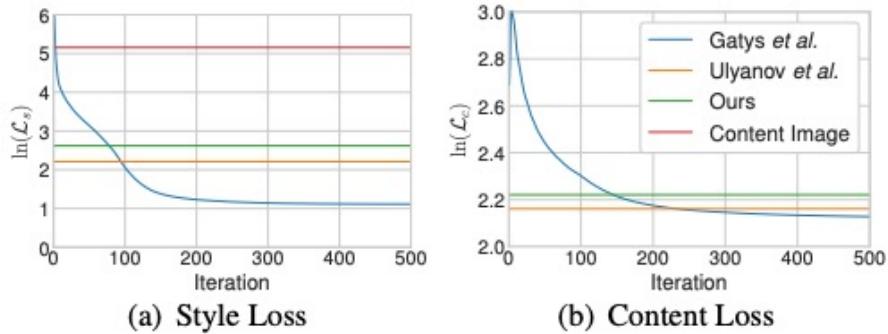


Figure 3. Quantitative comparison of different methods in terms of style and content loss. Numbers are averaged over 10 style images and 50 content images randomly chosen from our test set.

Method	Time (256px)	Time (512px)	# Styles
Gatys <i>et al.</i>	14.17 (14.19)	46.75 (46.79)	∞
Chen and Schmidt	0.171 (0.407)	3.214 (4.144)	∞
Ulyanov <i>et al.</i>	0.011 (N/A)	0.038 (N/A)	1
Dumoulin <i>et al.</i>	0.011 (N/A)	0.038 (N/A)	32
Ours	0.018 (0.027)	0.065 (0.098)	∞

Table 1. Speed comparison (in seconds) for 256×256 and 512×512 images. Our approach achieves comparable speed to methods limited to a small number styles [52, 11], while being much faster than other existing algorithms applicable to arbitrary styles [16, 6]. We show the processing time both excluding and including (in parenthesis) the style encoding procedure. Results are obtained with a Pascal Titan X GPU and averaged over 100 images.

StyleGAN | AdaIN Style Transfer

Additional Experiments

1. Decoder에 IN/BN layer 추가한 경우

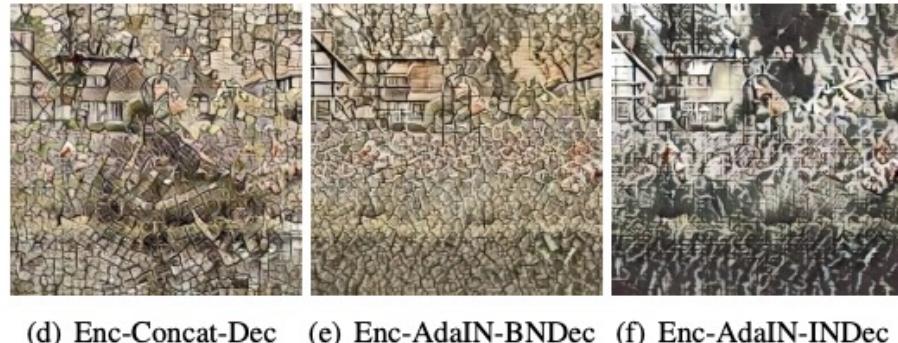


Figure 5. Comparison with baselines. AdaIN is much more effective than concatenation in fusing the content and style information. Also, it is important *not* to use BN or IN layers in the decoder.

2. Style과 Content 정보 중 특정 이미지에 가중치를 두는 경우

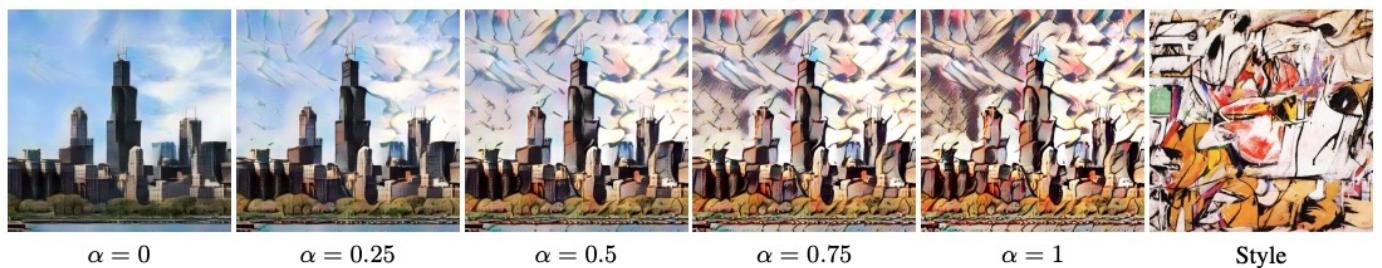


Figure 7. Content-style trade-off. At runtime, we can control the balance between content and style by changing the weight α in Equ. 14.

$$T(c, s, \alpha) = g((1 - \alpha)f(c) + \alpha\text{AdaIN}(f(c), f(s))) \quad (14)$$

StyleGAN | AdaIN Style Transfer

Additional Experiments

3. 여러 개의 style 에 대해서 style interpolation



Figure 8. Style interpolation. By feeding the decoder with a convex combination of feature maps transferred to different styles via AdaIN (Equ. 15), we can interpolate between arbitrary new styles.

$$T(c, s_{1,2,\dots,K}, w_{1,2,\dots,K}) = g\left(\sum_{k=1}^K w_k \text{AdaIN}(f(c), f(s_k))\right) \quad (15)$$

4. Color control & Spatial control



Figure 9. Color control. Left: content and style images. Right: color-preserved style transfer result.



Figure 10. Spatial control. Left: content image. Middle: two style images with corresponding masks. Right: style transfer result.

StyleGAN | AdaIN Style Transfer Code

■ 필요한 라이브러리 불러오기 및 환경 설정

```
# 필요한 PyTorch 라이브러리 불러오기
import torch
import torch.nn as nn

from torchvision import transforms
from torchvision.utils import save_image

from PIL import Image
from IPython.display import Image as display_image
```

<https://colab.research.google.com/drive/1KqeksZ9q3fC-vDNk-Tn-PHrh1ZI-vOM>

https://github.com/ndb796/Deep-Learning-Paper-Review-and-Practice/blob/master/code_practices/AdaIN_Style_Transfer_Tutorial.ipynb

StyleGAN | AdaIN Style Transfer Code

■ AdaIN module 구현

```
def adaptive_instance_normalization(content_feat, style_feat):
    assert (content_feat.size()[:2] == style_feat.size()[:2])
    size = content_feat.size()
    style_mean, style_std = calc_mean_std(style_feat)
    content_mean, content_std = calc_mean_std(content_feat)

    # 평균(mean)과 표준편차(std)를 이용하여 정규화 수행
    normalized_feat = (content_feat - content_mean.expand(size)) / content_std.expand(size)
    # 정규화 이후에 style feature의 statistics를 가지도록 설정

    return normalized_feat * style_std.expand(size) + style_mean.expand(size) # return t
```

```
def calc_mean_std(feat, eps=1e-5): # feat.shape (N: 배치 크기, C: 채널 크기, H: 높이, W: 너비)
    size = feat.size()
    assert (len(size) == 4)
    N, C = size[:2]
    feat_var = feat.view(N, C, -1).var(dim=2) + eps # view 함수 https://wikidocs.net/52846
    (뷰(View) - 원소의 수를 유지하면서 텐서의 크기 변경)
    feat_std = feat_var.sqrt().view(N, C, 1, 1)
    feat_mean = feat.view(N, C, -1).mean(dim=2).view(N, C, 1, 1)

    return feat_mean, feat_std
```

Adaptive Instance Normalization (AdaIN 구현)

Content feature 의 스타일을 style feature 의 스타일로 변경.

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

Instance Normalization 을 위한 평균 (mean)과 표준편차 (std) 계산

StyleGAN | AdaIN Style Transfer Code

■ 인코더 네트워크 구현

```
# 인코더(Encoder) 정의
vgg = nn.Sequential(
    nn.Conv2d(3, 3, (1, 1)),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(3, 64, (3, 3)),
    nn.ReLU(), # relu1-1
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(64, 64, (3, 3)),
    nn.ReLU(), # relu1-2
    nn.MaxPool2d((2, 2), (2, 2), (0, 0), ceil_mode=True),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(64, 128, (3, 3)),
    nn.ReLU(), # relu2-1
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(128, 128, (3, 3)),
    nn.ReLU(), # relu2-2
    nn.MaxPool2d((2, 2), (2, 2), (0, 0), ceil_mode=True),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(128, 256, (3, 3)),
    nn.ReLU(), # relu3-1
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 256, (3, 3)),
    nn.ReLU(), # relu3-2
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 256, (3, 3)),
    nn.ReLU(), # relu3-3
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 256, (3, 3)),
    nn.ReLU(), # relu3-4
    nn.MaxPool2d((2, 2), (2, 2), (0, 0), ceil_mode=True),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 512, (3, 3)),
    nn.ReLU(), # relu4-1, this is the last layer used
    nn.ReflectionPad2d((1, 1, 1, 1)), ...)
```

Encoder: VGG 형식의 네트워크를 사용하여 이미지로부터 특징 (feature) 을 추출 (extract).

VGG 네트워크에서 맥스 풀링 (max pooling)을 총 4번 진행하지만, Style Transfer 에서는 ReLU4_1 까지만 사용.

StyleGAN | AdaIN Style Transfer Code

■ 디코더 네트워크 구현

```
# 인코더(Encoder) 정의
# 디코더(Decoder) 정의
decoder = nn.Sequential(
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(512, 256, (3, 3)),
    nn.ReLU(),
    nn.Upsample(scale_factor=2, mode='nearest'),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 256, (3, 3)),
    nn.ReLU(),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 256, (3, 3)),
    nn.ReLU(),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 256, (3, 3)),
    nn.ReLU(),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 128, (3, 3)),
    nn.ReLU(),
    nn.Upsample(scale_factor=2, mode='nearest'),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(128, 128, (3, 3)),
    nn.ReLU(),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(128, 64, (3, 3)),
    nn.ReLU(),
    nn.Upsample(scale_factor=2, mode='nearest'),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(64, 64, (3, 3)),
    nn.ReLU(),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(64, 3, (3, 3)),
)
```

Decoder: 인코더와 유사한 네트워크이지만, 반대로 해상도(너비*높이)를 키우는 방식으로 동작.

인코더 파트에서 3번의 downsampling을 거치므로, 디코더에서는 3번의 upsampling.

StyleGAN | AdaIN Style Transfer Code

■ Pre-trained 모델 불러오기

```
!wget https://postechackr-my.sharepoint.com/:u/g/personal/dongbinna_postech_ac_kr/Ebs6XES37otLgWW5cavCY9YByGOnXJD5wC0WZgQPp8vAJg?download=1 -O decoder.pth  
!wget https://postechackr-my.sharepoint.com/:u/g/personal/dongbinna_postech_ac_kr/EXzxBZl0seBFuxucQjIshBEBtM7X5-Lmj-ceqZ5Fu96aIA?download=1 -O vgg_normalised.pth
```

■ Style Transfer

```
def style_transfer(vgg, decoder, content, style, alpha=1.0):  
    assert (0.0 <= alpha <= 1.0)  
    content_f = vgg(content)  
    style_f = vgg(style)  
    feat = adaptive_instance_normalization(content_f, style_f)  
    feat = feat * alpha + content_f * (1 - alpha)  
  
    return decoder(feat)
```

StyleGAN | AdaIN Style Transfer Code

■ 이미지 전처리 및 실습

```
def test_transform(size=512):
    transform_list = []
    if size != 0:
        transform_list.append(transforms.Resize(size))
        transform_list.append(transforms.ToTensor())
    transform = transforms.Compose(transform_list)

    return transform

content_tf = test_transform()
style_tf = test_transform()
```

이미지 전처리

```
# content_path = './code_practices/images/content_img_1.jpg'
content_path = '/content/sunflowers_test1.jpg'
style_path = './code_practices/images/style_img_2.jpg'

# 이미지 전처리
content = content_tf(Image.open(str(content_path)))
style = style_tf(Image.open(str(style_path)))

style = style.to(device).unsqueeze(0)
content = content.to(device).unsqueeze(0)

# model -> eval 모드로 변경.
with torch.no_grad():
    output = style_transfer(vgg, decoder, content, style, alpha=1.0)
    output = output.cpu()

save_image(output, 'output.png')
```

StyleGAN | AdaIN Style Transfer Code

Content Image

Result

Style Image ①



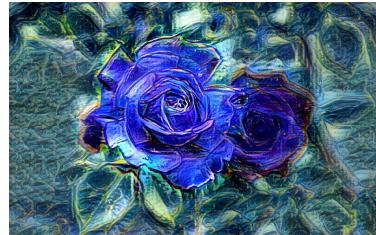
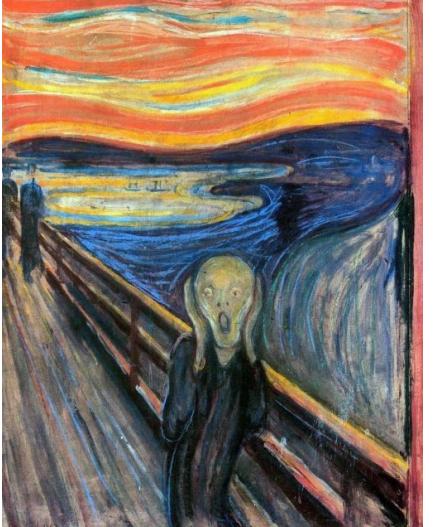
Result Image ①



Result Image ②



Style Image ②



감사합니다.