



Neural Style Transfer 딥러닝을 활용한 스타일 전이

Daehyun Cho
Cognitive System Lab
A.I. Dept, Korea Univ.

Index



NST의 원리 1: CNN-based



NST의 원리 2: GAN-based



NST 실습해보기



작지만 소중한 팁



Backups & References



들어가기 전에...

학습목표

- ✓ 인공지능을 제대로 사용하려면 밑바닥까지 아는 것이 정석입니다.
 - אין 근데 시간 없어요.
 - 없는 Django 분량도 많아서 여유도 없어요.
 - 그리고 강의에 이미 OpenCV로 가볍게 배우셨더라고요
- ✓ 그래서 목표를 **단계 별로** 설정해봤습니다. 내가 할 수 있는 데 까지만 해봅시다.



들어가기 전에...

학습목표

1단계 Neural Style Transfer의 **기본 원리 이해**하기

2단계 Pre-trained weights 불러서 **Django**에 올려 보기

3단계 **처음부터 훈련**하는 방법 배우기



들어가기 전에...

Disclaimer

이 밖에도 NST 논문/연구는 지천에 널렸습니다.

이들 중 아주 일부만 소개하고,

이해를 돋기 위해 **일부 표현 및 설명이 일반적이지 않을 수 있다**는 점 미리 알립니다.



1

Neural Style Transfer (NST) 의 원리 1: CNN-based

- ✓ Neural Style Transfer가 무엇인가요?
- ✓ 초석이 되는 논문 몇 개 부셔보기

▣ Convolutional Neural Network 기반

- ❖ Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
- ❖ Huang, Xun, and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization." Proceedings of the IEEE international conference on computer vision. 2017.

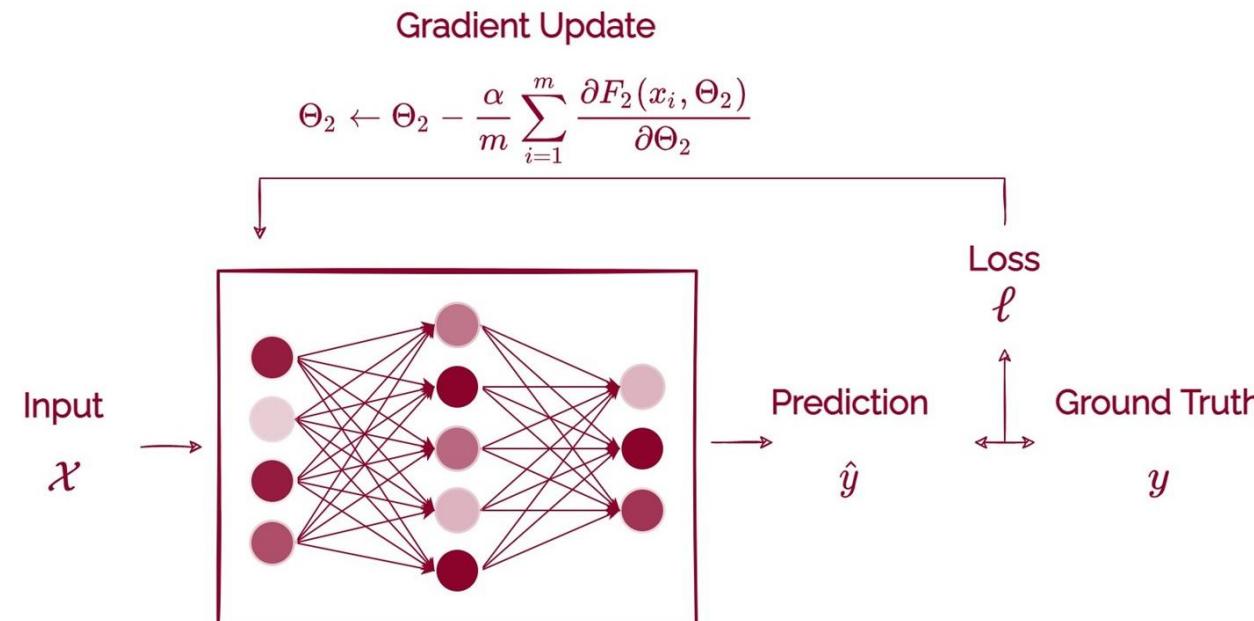
- 한 이미지의 **내용(Content)**에 다른 이미지의 **스타일(Style)**을 입히는 것을 뜻한다.
 - 내용은 이미지 내에서 전체적인 구성요소, 뼈대들. (앵무새와 나뭇가지 등)
 - 스타일은 이미지 내의 전반적인 색채요소들. (노을을 구성하는 색깔 등)
- 그래서 아래와 같이 두 이미지를 합치면 맨 오른쪽과 같은 결과물이 ta-da. (제가 만들었는데 이쁜 것 같음:)



+



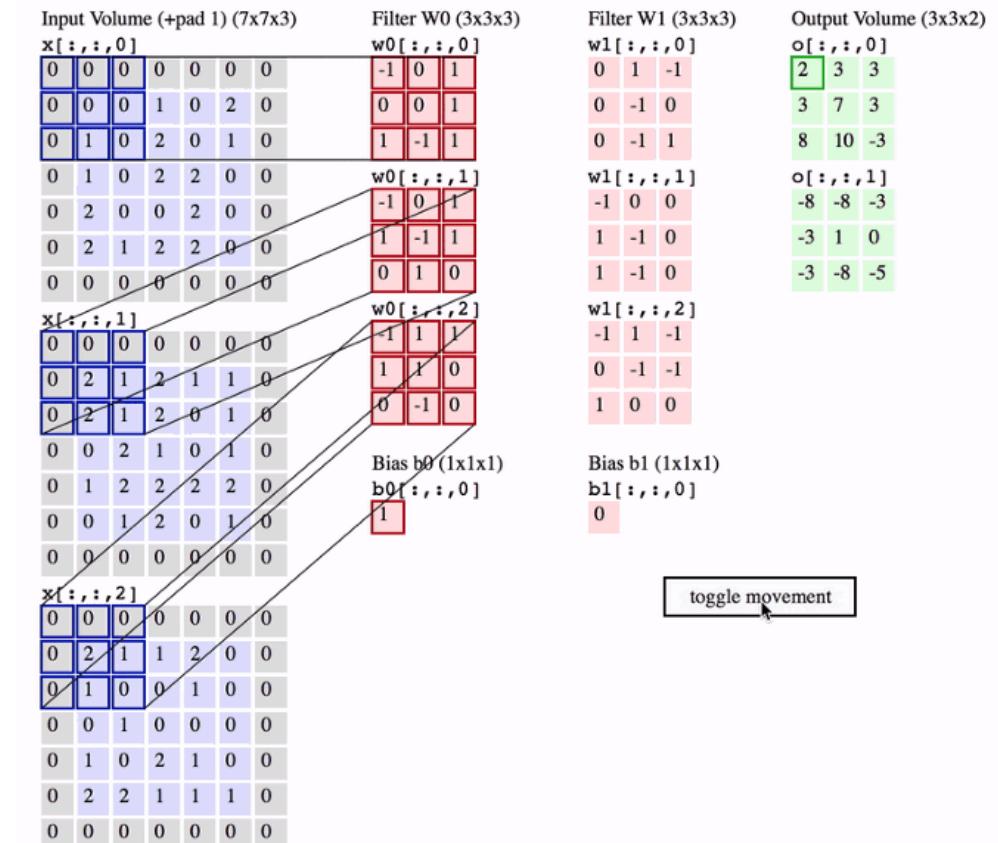
- Computer vision 영역에서 다양한 분야의 연구들이 진행됐고
그 중 사람의 시각에 관해 연구를 하던 한 연구자는 **Convolutional Neural Network를 통해** Style Transfer를 수행했다.
- 처음부터 Style Transfer를 노리고 나온 건 아니고, 이미지 내에서 Texture를 뽑아내려는 연구에서 시작되었다.
- 이를 이해하기 위해서 **CNN을 잠깐 다시 살펴봐야 할 것 같다.**
- 아래는 딥러닝 모델의 기본 구조이다.



- 냅다 Convolution 연산 방법만 살펴보기

*연산에만 집중할 계획이기 때문에 전체적인 구조에서 사용된 Max pool, Padding 같은 요소는 생략 (자세히 알고 싶으신 분들은 따로 연락 주세요)

1. 작은 사이즈의 필터(혹은 커널)를 만들어준다.
2. 이 필터를 이미지에서 순차적으로 돌아가면서 연산을 걸어준다.
3. 이 때 필터를 통과하고 낸을 때의 결과는 어떤 의미를 가질까?

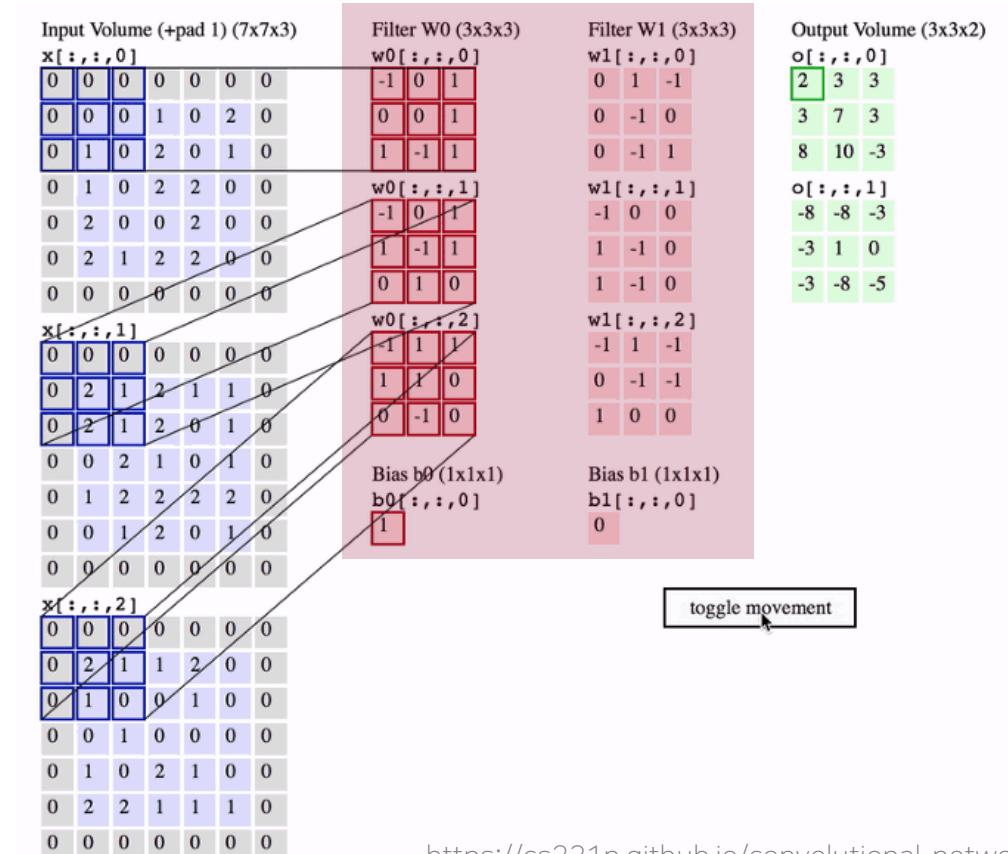


- 냅다 Convolution 연산 방법만 살펴보기

*연산에만 집중할 계획이기 때문에 전체적인 구조에서 사용된 Max pool, Padding 같은 요소는 생략 (자세히 알고 싶으신 분들은 따로 연락 주세요)

1. 작은 사이즈의 필터(혹은 커널)를 만들어준다.

1. 이 필터를 이미지에서 순차적으로 돌아가면서 연산을 걸어준다.
2. 이 때 필터를 통과하고 낸을 때의 결과는 어떤 의미를 가질까?



- 냅다 Convolution 연산 방법만 살펴보기

*연산에만 집중할 계획이기 때문에 전체적인 구조에서 사용된 Max pool, Padding 같은 요소는 생략 (자세히 알고 싶으신 분들은 따로 연락 주세요)

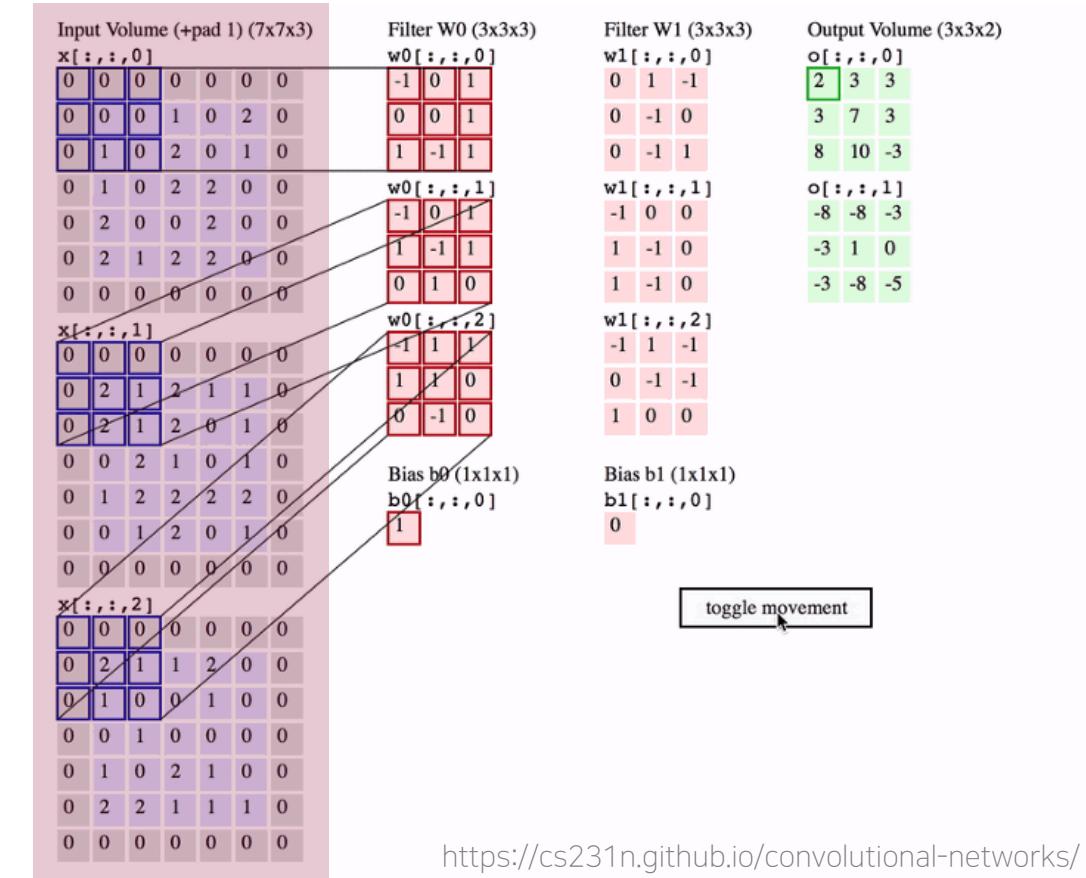
1. 작은 사이즈의 필터(혹은 커널)를 만들어준다.

2. 이 필터를 이미지에서 순차적으로 돌아가면서 연산을 걸어준다.

3. 이 때 필터를 통과하고 낸을 때의 결과는 어떤 의미를 가질까?

Input	Kernel	Output
$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} *$	$= \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$

Fig. 6.2.1 Two-dimensional cross-correlation operation. The shaded portions are the first output element as well as the input and kernel tensor elements used for the output computation: $0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$.

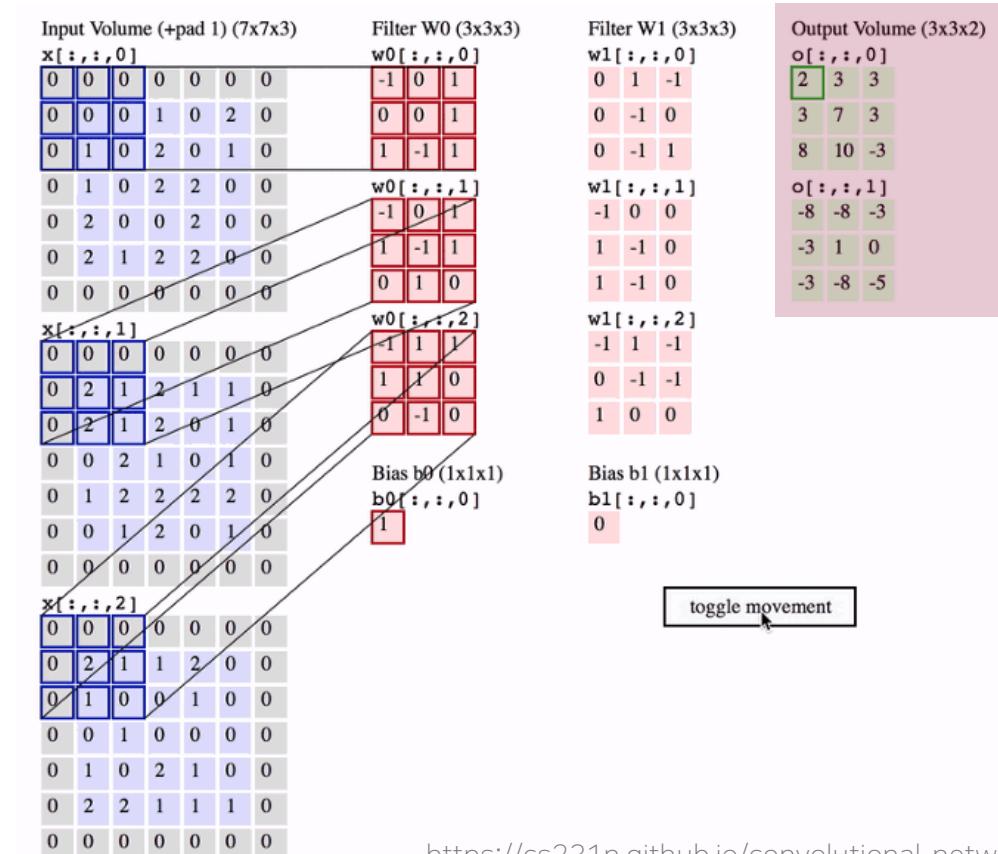


- 냅다 Convolution 연산 방법만 살펴보기

*연산에만 집중할 계획이기 때문에 전체적인 구조에서 사용된 Max pool, Padding 같은 요소는 생략 (자세히 알고 싶으신 분들은 따로 연락 주세요)

1. 작은 사이즈의 필터(혹은 커널)를 만들어준다.
2. 이 필터를 이미지에서 순차적으로 돌아가면서 연산을 걸어준다.
3. 이 때 필터를 통과하고 낸을 때의 결과는 어떤 의미를 가질까?

*보통 이 결과물들을 Feature Map이라 부른다.

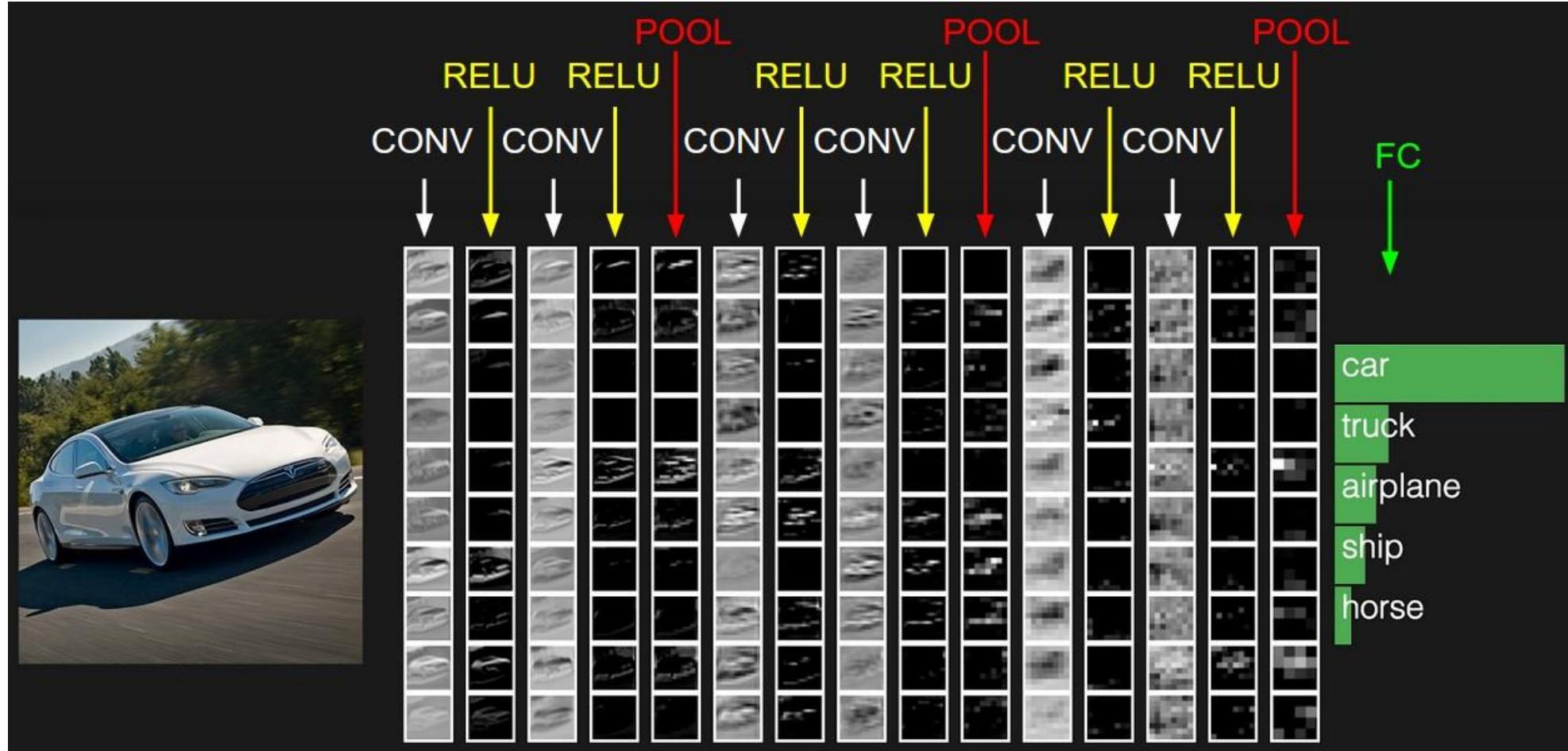


Recap: Convolution

Neural Style Transfer의 원리1: CNN-based

1

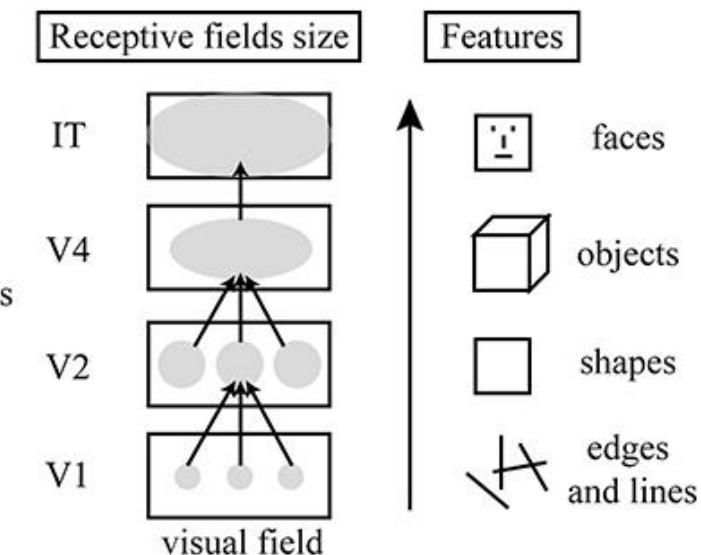
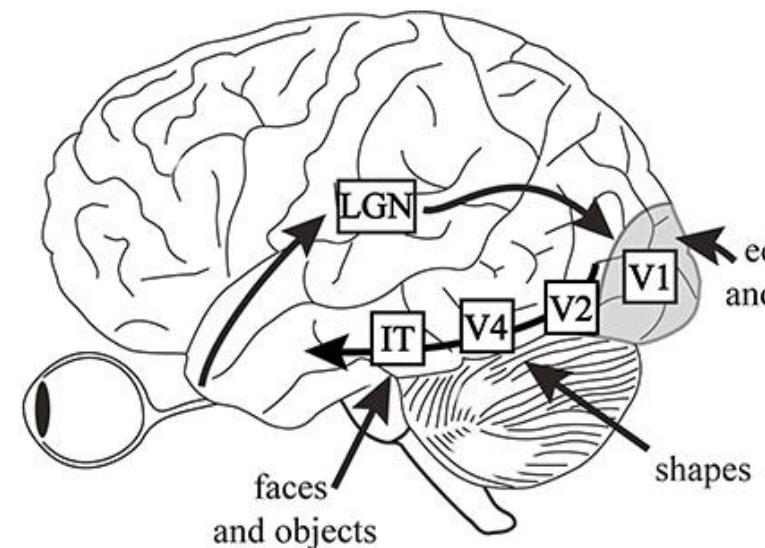
- 이 때 필터를 통과하고 낸을 때의 결과는 어떤 의미를 가질까?
- 잠시 의미를 찾아보기 위해 노를 열어보자 (?)



<https://cs231n.github.io/convolutional-networks/>

- 우리의 뇌에서 시각 시스템이 정보를 처리하는 방식은 **계층적**이다.
- 먼저 빛이 눈으로 들어와서 시신경을 타고 후두엽으로 흘러간다.
후두엽에서 측두엽으로 넘어가는 과정에서 이 정보들이 점점 변화하게 된다.
- 주요 변화는, 제일 먼저 처리되는 시각 정보에서는 저차원이며 아주 작은 요소들을 잡아낸다.
뒤로 처리될수록 더 넓은 범위의 정보들을 잡아낸다.

*말이 그렇지 사실 우리의 뇌는 훨씬 복잡하다. 아이디어만 얻어간 것으로 이해하자.

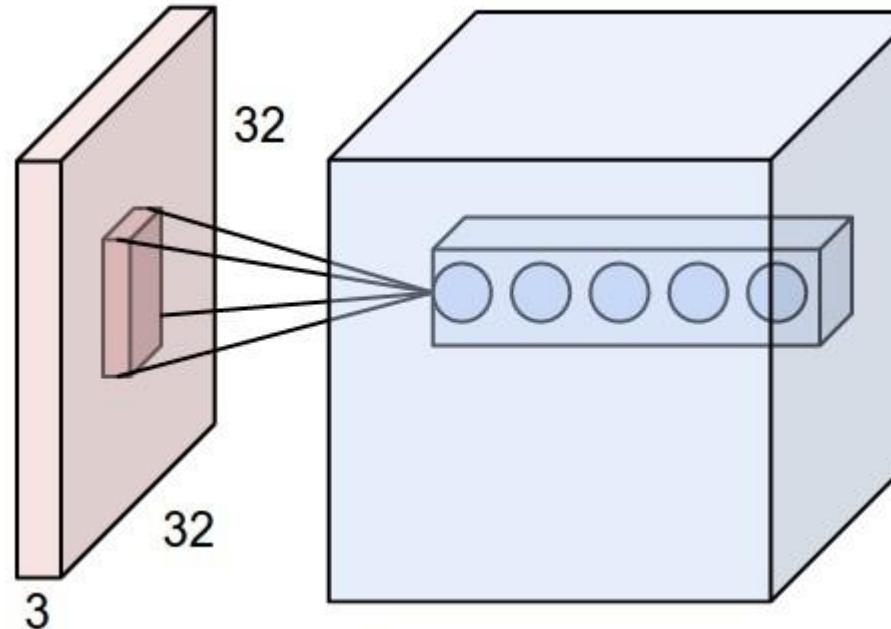


Herzog, Michael H., and Aaron M. Clarke. "Why vision is not both hierarchical and feedforward." Frontiers in computational neuroscience 8 (2014): 135.

- Convolution이 처음 나오게 된 배경이 바로 이런 처리 방법이다. 이제야 뭔가 이해가 되는 듯 아닌 듯 하다.
- 크게 두 가지 방면에서 모방하려 했는데
 - 이미지 전체를 보는 것이 아니라 일부 영역에 연산을 걸어주는 것
 - 뒤로 흘러가는 데이터에는 더 많은 영역의 연산을 포함하도록 하는 것.



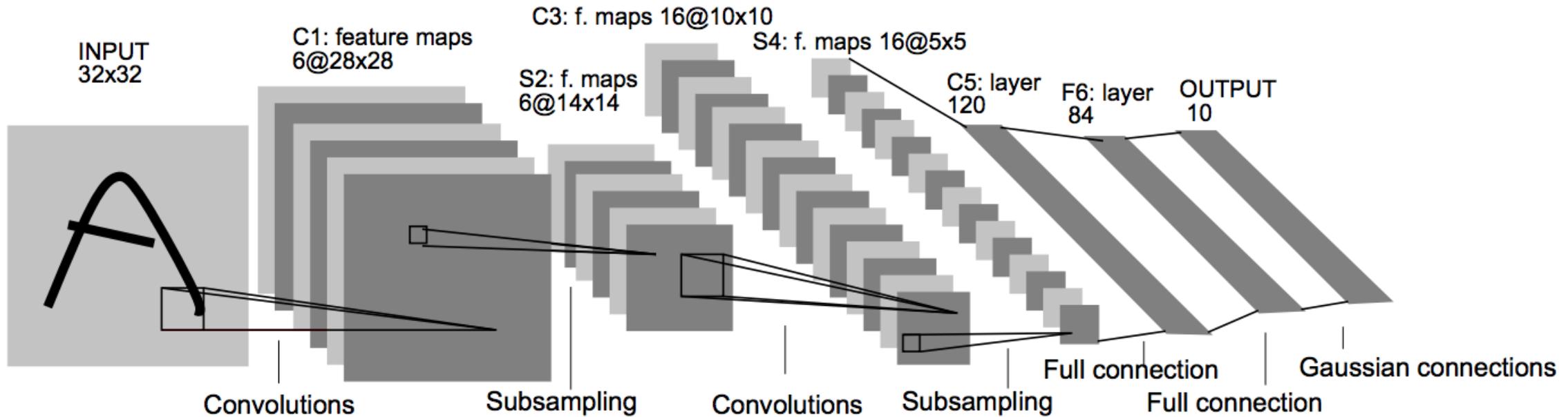
- Convolution이 처음 나오게 된 배경이 바로 이런 처리 방법이다. 이제야 뭔가 이해가 되는 듯 아닌 듯 하다.
- 크게 두 가지 방면에서 모방하려 했는데
 - 이미지 전체를 보는 것이 아니라 일부 영역에 연산을 걸어주는 것
 - 뒤로 흘러가는 데이터에는 더 많은 영역의 연산을 포함하도록 하는 것.



Input Volume (+pad 1) (7x7x3)	Filter W0 (3x3x3)	Filter W1 (3x3x3)	Output Volume (3x3x2)
$x[:, :, 0]$	$w0[:, :, 0]$	$w1[:, :, 0]$	$o[:, :, 0]$
0 0 0 0 0 0 0 0 0 0 1 0 2 0 0 1 0 2 0 1 0	-1 0 1 0 0 1 1 -1 1	0 1 -1 0 -1 0 0 -1 1	2 3 3 3 7 3 8 10 -3
0 1 0 2 2 0 0 0 2 0 0 2 0 0 0 2 1 2 2 0 0	-1 0 1 1 -1 1 0 1 0	-1 0 0 1 -1 0 1 -1 0	o[:, :, 1] -8 -8 -3 -3 1 0 -3 -8 -5
0 0 0 0 0 0 0 0 2 1 2 1 1 0 0 2 1 2 0 1 0	1 1 1 1 1 0 0 -1 0	-1 1 -1 0 -1 -1 1 0 0	Bias b1 (1x1x1) $b1[:, :, 0]$ 0
0 0 2 1 0 1 0 0 1 2 2 2 2 0 0 0 1 2 0 1 0	1 1 1 1 1 0 0 -1 0	1 1 1 1 1 0 0 -1 0	toggle movement
0 0 0 0 0 0 0 0 2 2 1 2 0 0 0 1 0 0 1 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	

<https://cs231n.github.io/convolutional-networks/>

- Convolution이 처음 나오게 된 배경이 바로 이런 처리 방법이다. 이제야 뭔가 이해가 되는 듯 아닌 듯 하다.
- 크게 두 가지 방면에서 모방하려 했는데
 - 이미지 전체를 보는 것이 아니라 일부 영역에 연산을 걸어주는 것
 - **뒤로 흘러가는 데이터에는 더 많은 영역의 정보를 포함하도록 하는 것.**



LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." The handbook of brain theory and neural networks 3361.10 (1995): 1995.

- Convolution 구조로 우리가 여러 이미지 관련 task에서 좋은 성능을 거둘 수 있다. 가 결론이 아니고 -
- 이 Convolution 연산을 통해서 이미지들을 계층적으로 학습할 수 있다는 것이 핵심이다.
 - 여기서 계층적 학습의 결과는 구조 중간에 있는 Feature map들이다.
 - 여담으로 딥러닝 연구로 가게 된다면 성능을 올릴 수 있는 새로운 모델을 제시하는 것도 좋은 연구이지만, 이게 왜 될까?를 검증하기 위해 다양하고 타당한 실험 설계를 통해 결과를 보여주는 연구들이 최고다!
- **이 Feature map으로 무언가 얻어낼 수 없을까?** 하고 시작된 연구가 현재의 Style Transfer 논문이다.



- Style Transfer에 아직 도착 못했다. 자세히 보면 오늘 끝나지 못할 것 같으니 간단하게 살펴보자.

Gatys, Leon, Alexander S. Ecker, and Matthias Bethge.

"**Texture synthesis using convolutional neural networks.**"

Advances in neural information processing systems 28 (2015).



Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge.

"**A neural algorithm of artistic style.**"

arXiv preprint arXiv:1508.06576 (2015).



Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge.

"**Image style transfer using convolutional neural networks.**"

Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Convolution 모델에서 **질감을 합성**해내기

선행 연구에서 더 나아가서 **기존 이미지에 다른 질감을 입히기**

선행 연구를 개정해서 CVPR에 제출
(사실 같은 논문 개정판)

- 저자는 Feature map에 분명히 이미지를 구성하는 요소들을 포함할 것이라 생각했다.
- 특히 질감에 관련한 요소들은 모든 층 전반에 걸쳐서 가지고 있다고 생각.

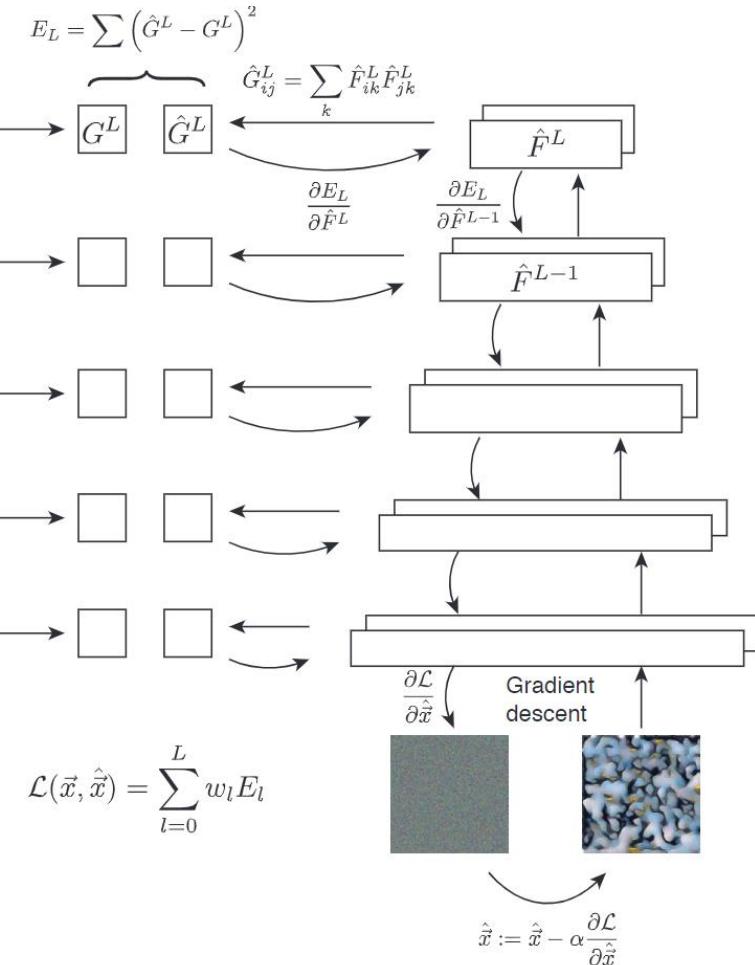
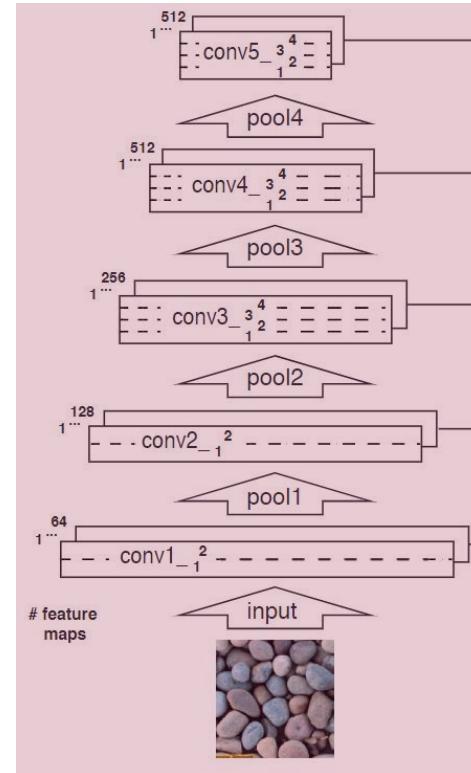
1. 질감을 원하는 이미지를 잘 훈련된 CNN 구조에 통과시킨다.

- 2015년이라 사실 VGG19말고 없다.
- 여러분이 잘 아는 ResNet도 2016년...

2. 이 때 Layer 별로 Feature Map을 뽑아낸다.

3. 새로운 Random Noise를 모델에 통과시킨다.

- 여기서 모델을 어떻게 학습시킬까?
 - 현재 똑같은 이미지를 만들어내는게 목적이 아니기 때문에 단순히 같은 이미지를 복원하는 방향으로 만들 수는 없다.



Gatys, Leon, Alexander S. Ecker, and Matthias Bethge. "Texture synthesis using convolutional neural networks." Advances in neural information processing systems 28 (2015).

- 저자는 Feature map에 분명히 이미지를 구성하는 요소들을 포함할 것이라 생각했다.
- 특히 질감에 관련한 요소들은 모든 층 전반에 걸쳐서 가지고 있다고 생각.

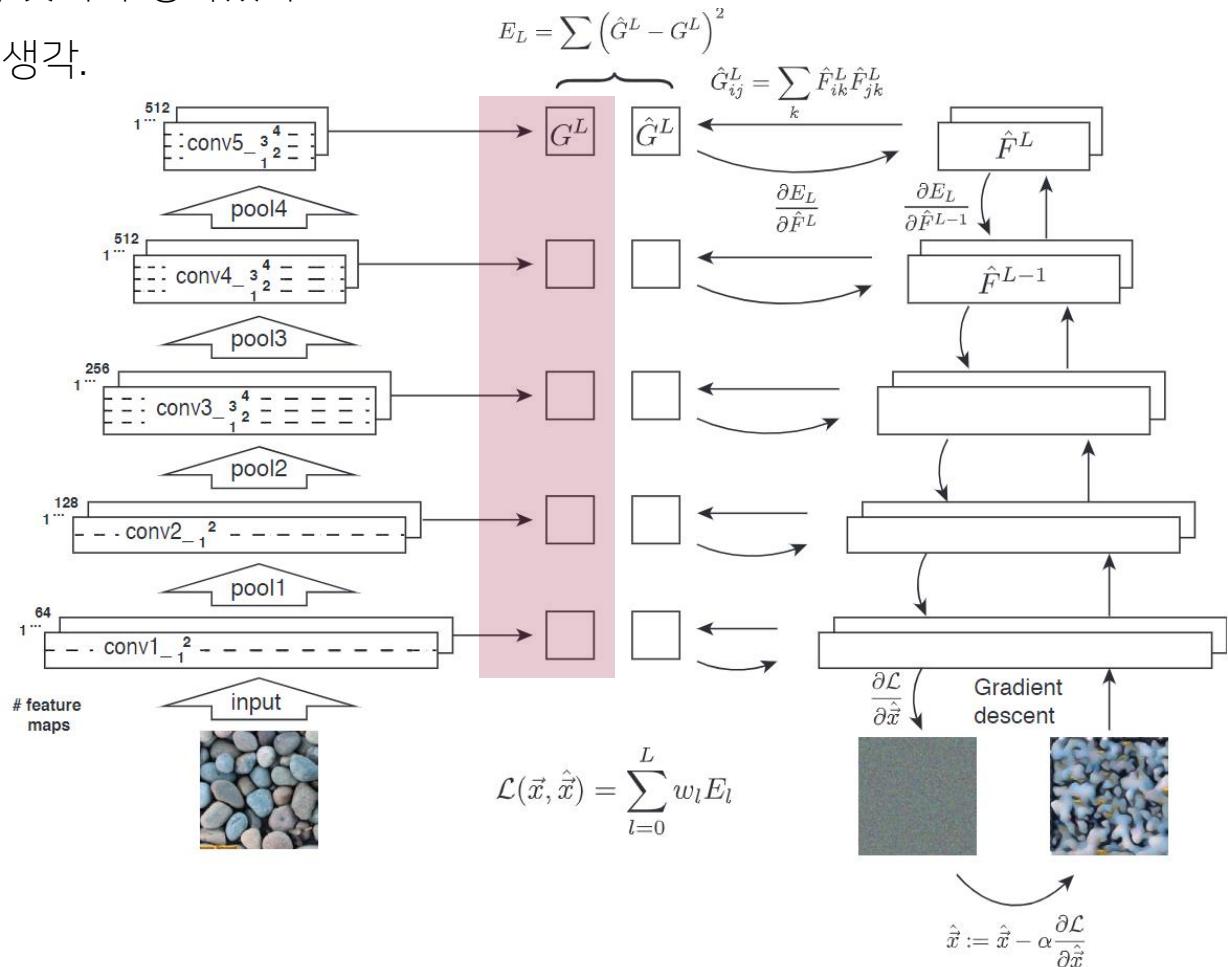
1. 질감을 원하는 이미지를 잘 훈련된 CNN 구조에 통과시킨다.

- 2015년이라 사실 VGG19말고 없다.
- 여러분이 잘 아는 ResNet도 2016년...

2. 이 때 Layer 별로 Feature Map을 뽑아낸다.

3. 새로운 Random Noise를 모델에 통과시킨다.

- 여기서 모델을 어떻게 학습시킬까?
 - 현재 똑같은 이미지를 만들어내는게 목적이 아니기 때문에 단순히 같은 이미지를 복원하는 방향으로 만들 수는 없다.



Gatys, Leon, Alexander S. Ecker, and Matthias Bethge. "Texture synthesis using convolutional neural networks." Advances in neural information processing systems 28 (2015).

- 저자는 Feature map에 분명히 이미지를 구성하는 요소들을 포함할 것이라 생각했다.
- 특히 질감에 관련한 요소들은 모든 층 전반에 걸쳐서 가지고 있다고 생각.

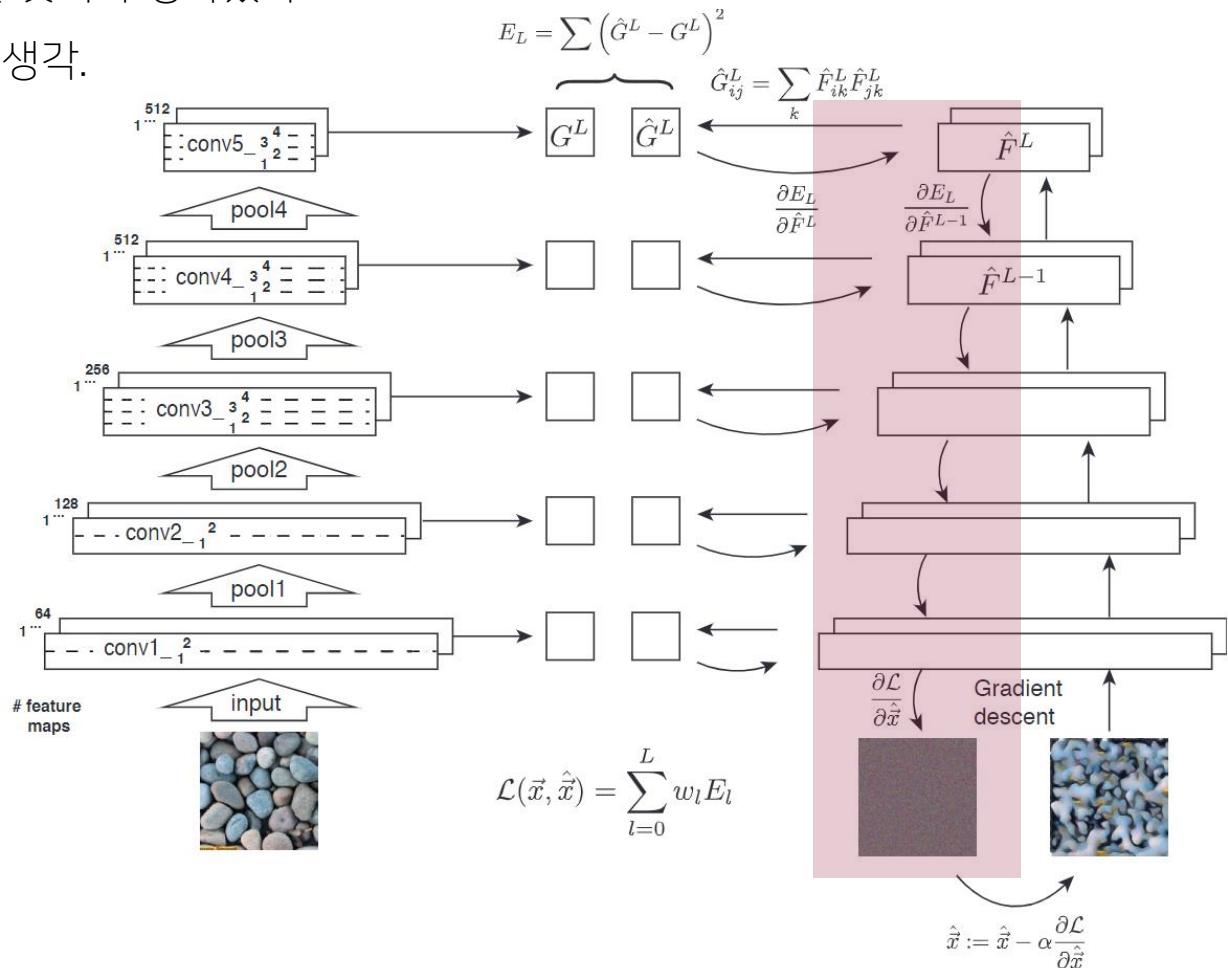
1. 질감을 원하는 이미지를 잘 훈련된 CNN 구조에 통과시킨다.

- 2015년이라 사실 VGG19말고 없다.
- 여러분이 잘 아는 ResNet도 2016년...

2. 이 때 Layer 별로 Feature Map을 뽑아낸다.

3. 새로운 Random Noise를 모델에 통과시킨다.

- 여기서 모델을 어떻게 학습시킬까?
 - 현재 똑같은 이미지를 만들어내는게 목적이 아니기 때문에 단순히 같은 이미지를 복원하는 방향으로 만들 수는 없다.



Gatys, Leon, Alexander S. Ecker, and Matthias Bethge. "Texture synthesis using convolutional neural networks." Advances in neural information processing systems 28 (2015).

- 저자는 Feature map에 분명히 이미지를 구성하는 요소들을 포함할 것이라 생각했다.
- 특히 질감에 관련한 요소들은 모든 층 전반에 걸쳐서 가지고 있다고 생각.

1. 이미지를 잘 훈련된 CNN 구조에 통과시킨다.

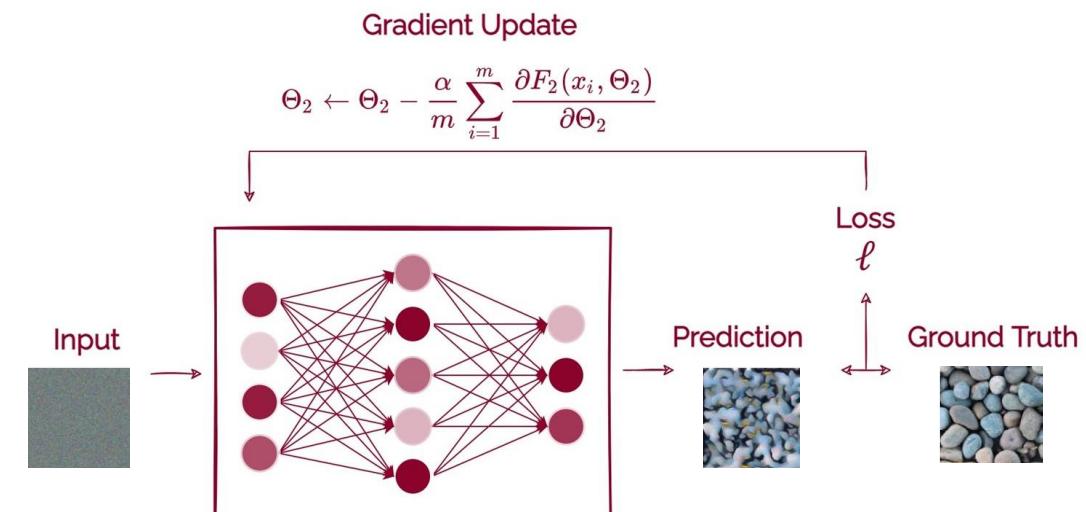
- 2015년이라 사실 VGG19말고 없다.
- 여러분이 잘 아는 ResNet도 2016년...

2. 이 때 Layer 별로 Feature Map을 뽑아낸다.

3. 새로운 Random Noise를 모델에 통과시킨다.

▪ 여기서 모델을 어떻게 학습시킬까?

- 현재 똑같은 이미지를 만들어내는게 목적이 아니기 때문에 단순히 같은 이미지를 복원하는 방향으로 학습시킬 수는 없다.

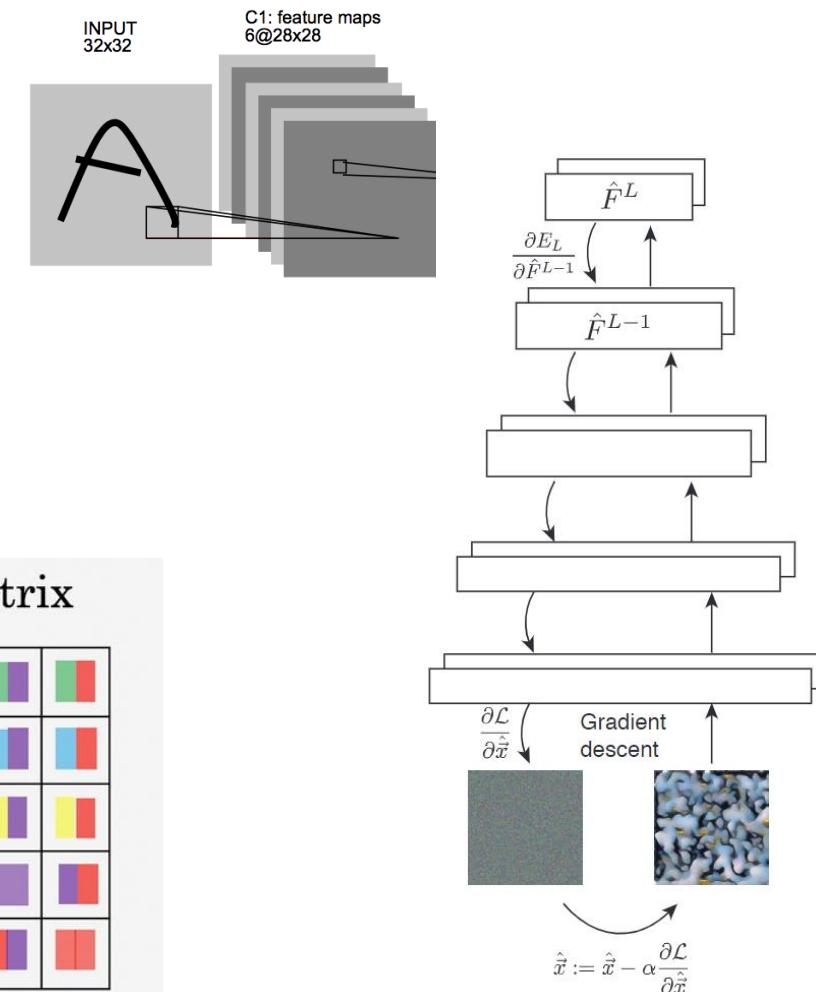
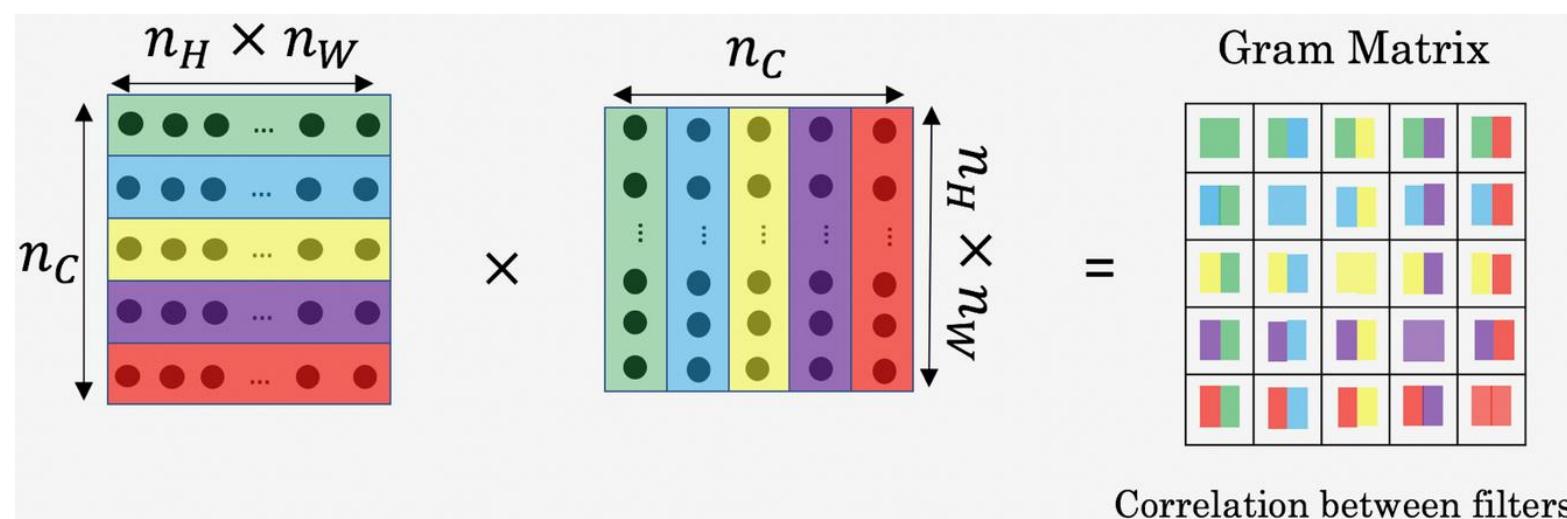


CNN으로 “질감”을 학습해보자.

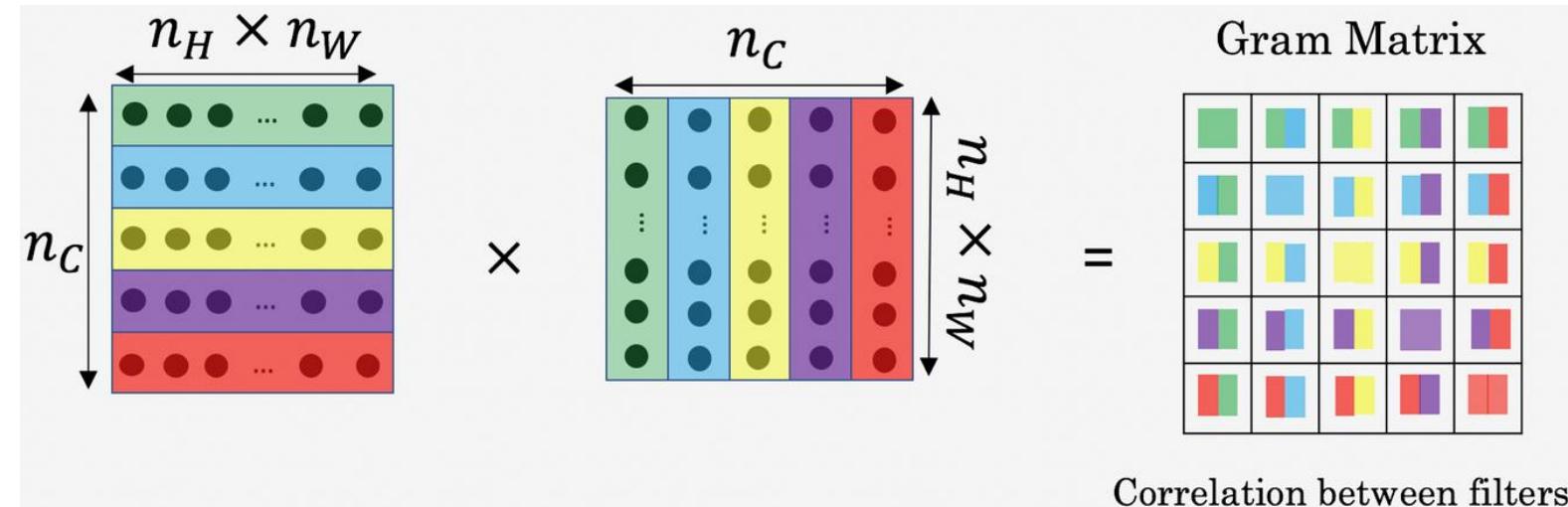
1

Neural Style Transfer의 원리1: CNN-based

- 여기서 Gram Matrix라는 개념이 등장한다. 아이고 두야
- 레이어 별로 Feature map이 나타날 것이다.
 - 이 때 Feature map의 형태는 (채널 수, 가로, 세로)인 3차원 텐서 형태를 가진다.
 - 이를 (채널 수, 가로 * 세로)인 2차원 행렬 형태로 변환해준 뒤 전치를 걸어서 곱해준다.
*솔직히 말로만 들으면 이해할 수 없다. 그림으로 최대한 이해를 해보자.
 - 결과물이 의미하는 바는 무엇일까?



- 하나의 Feature map과 다른 Feature map 사이를 내적했으니
이는 일종의 Feature map들 사이 상관 관계를 뜻한다.
- 이 상관 관계를 나타내는 결과물은
원본 이미지가 아니면서 동시에 내부적인 요소들을 **내재**한다. *내재한다는 표현이 모호하지만… 최선입니다.



*여기서 중요한 전제는 이 Feature map에 의미가 생기려면,
이를 만들어주고 있는 네트워크의 성능이 좋아야 한다.
그래서 pre-trained된 VGG를 사용하는 것이다.



CNN으로 “질감”을 학습해보자.

Neural Style Transfer의 원리1: CNN-based

- 하나의 Feature map과 다른 Feature map 사이를 내적했으니
이는 일종의 Feature map들 사이 상관 관계를 뜻한다.
- 이 상관 관계를 나타내는 결과물은
원본 이미지가 아니면서 동시에 내부적인 요소들을 **내재화**한다.
- 즉, **새롭게 생성한 이미지는 원본 이미지 재구성이 아닌
이 내부요소를 재구성하는 방향으로 학습시킨다.**
- 하나의 레이어에서 나온 Feature map만 맞추는 것이 아니라,
종합적으로 여러 정보들을 합침

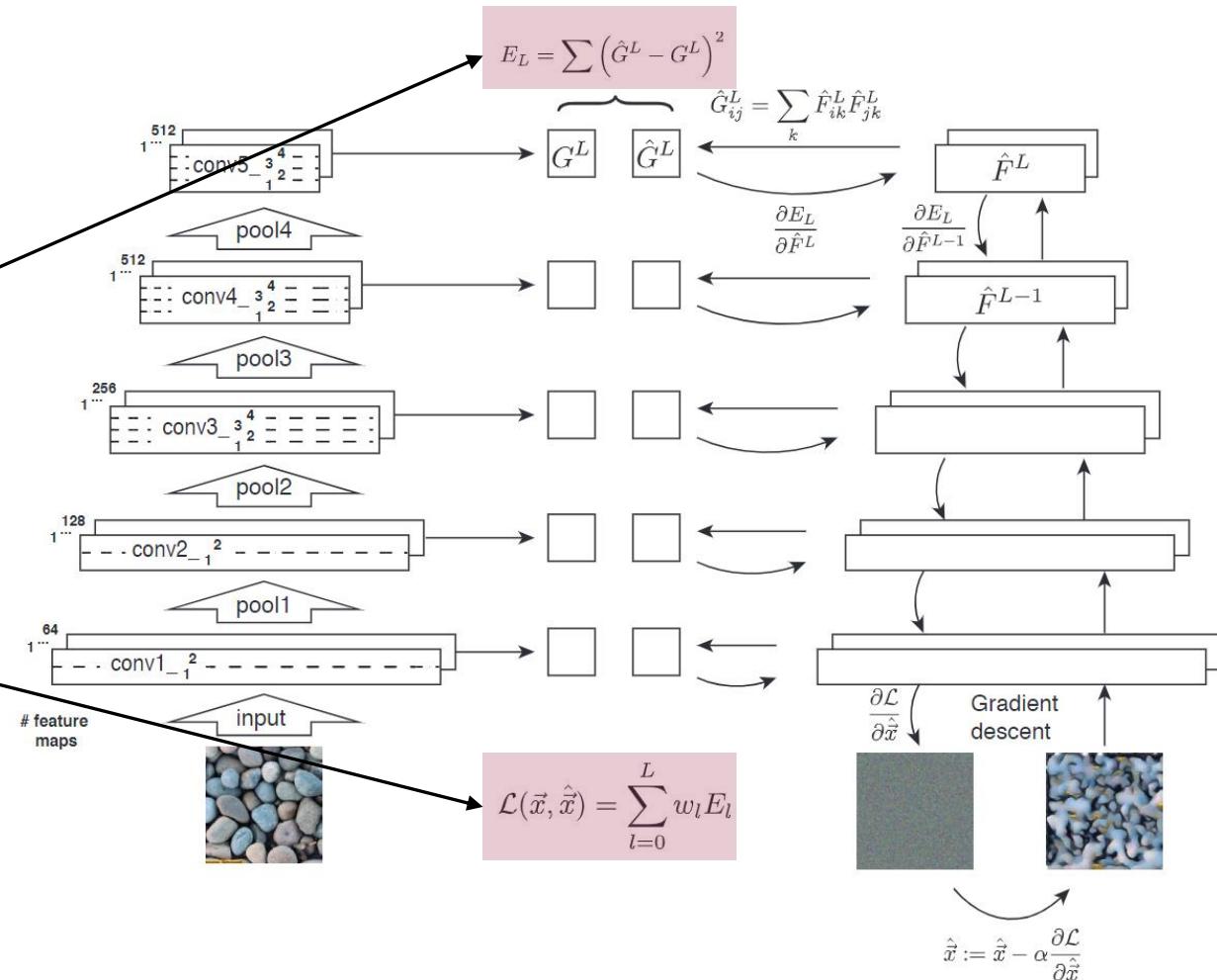
Notation

F^L, \hat{F}^L : 원본/생성 이미지의 L 번째 레이어에서 만들어낸 Feature Map

G^L, \hat{G}^L : 원본/생성 이미지의 L 번째 레이어에서 만들어낸 Gram Matrix

E_L : L 번째 레이어에서 Gram Matrix의 MSE

\mathcal{L} : 입력값과 재생성한 질감 사이의 최종 손실함수. 이를 최소화 해야함.



Gatys, Leon, Alexander S. Ecker, and Matthias Bethge. "Texture synthesis using convolutional neural networks." Advances in neural information processing systems 28 (2015).

CNN으로 “질감”을 학습해보자.

- 이렇게 해서 학습한 질감들이다.
- 레이어가 위로 갈수록 잘 나온다.
- 아래 레이어는 조금 더 색채감 위주로 표현



Neural Style Transfer의 원리1: CNN-based

CNN으로 “질감”을 학습해보자.

Neural Style Transfer의 원리1: CNN-based

- 이렇게 해서 학습한 질감들이다.

A ~1k parameters



~10k parameters



~177k parameters



~852k parameters

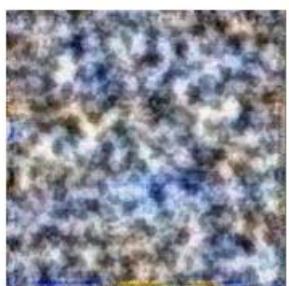


original



매개변수가 많을 수록 더 잘 나오는 편

B conv1



conv2



conv3



conv4



conv5

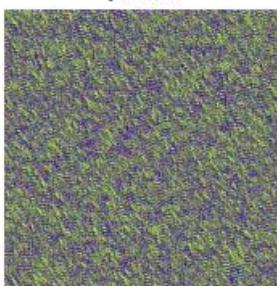


VGG말고 다른 네트워크 사용했을 때의 경우.
이 당시에는 VGG가 제일 잘 나갈 때…

C conv1_1



pool1



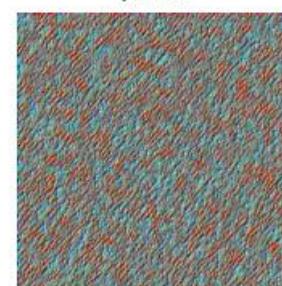
pool2



pool3



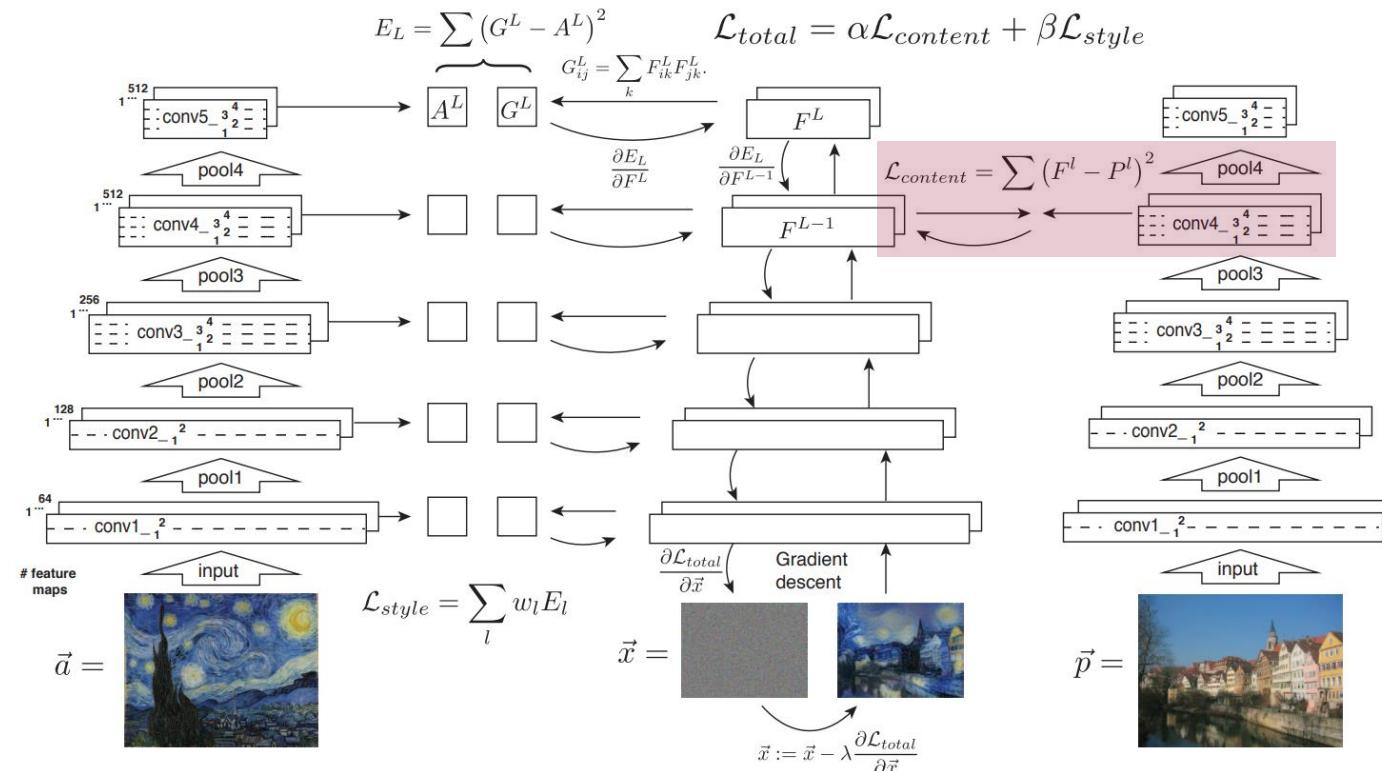
pool4



아예 새로운 네트워크를 들고 열어본 결과
질감 1도 표현 못하는 중

Gatys, Leon, Alexander S. Ecker, and Matthias Bethge. "Texture synthesis using convolutional neural networks." Advances in neural information processing systems 28 (2015).

- 질감을 뽑을 때는 모든 레이어의 Feature map을 학습하려 했다.
- 그러나 앞서 살펴본 Convolution의 특성상, **고차원의 요소들은 후반 레이어들에서 뽑아낼 수 있기 때문에** Content의 경우는 **마지막 레이어의 Feature map만 학습하여** 사용한다.
- 최종적으로 이 모델은 Content를 만들기 위한 손실 함수와 Style을 만들기 위한 손실 함수를 최적화하여 결과물을 만들어낸다.



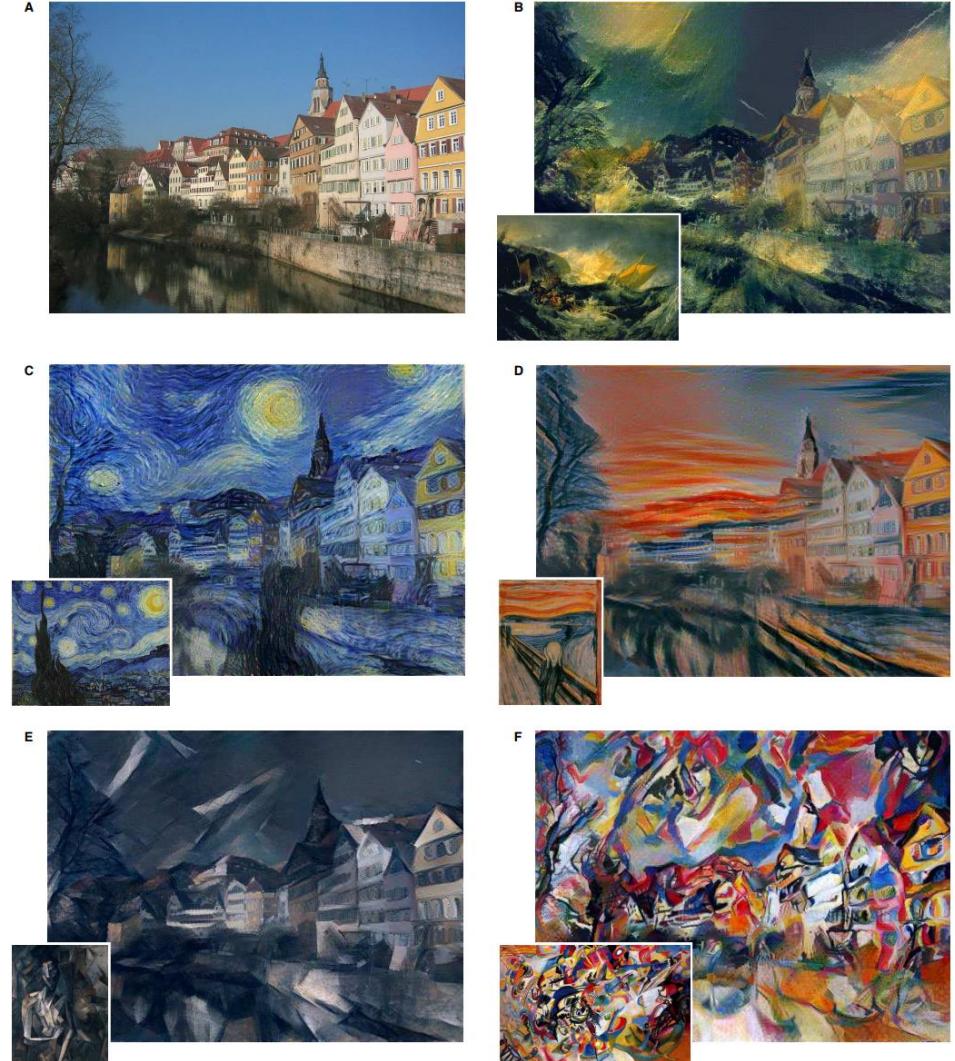
Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

그렇게 등장한 첫 딥러닝 기반의 NST 모델

Neural Style Transfer의 원리1: CNN-based

1

- 이전에도 Style Transfer를 시도한 노력들은 많이 있었지만, 딥러닝을 기반으로 + 눈으로 봤을 때도 잘 나온 연구가 되었다.
- 사실 이 저자는 독일에서 신경과학을 공부하던 박사인데 졸업논문으로 어마 무시한 걸 내버렸다 심지어 CVPR. (나도 박사과정인데…)
- 여기서 더 나아가 Deeparts라는 스타트업을 출시했고 지금까지 잘 돌아가네요. 진짜 인생한방
- 인터넷에 Neural Style Transfer를 검색했을 때 나오는 대부분의 블로그나 설명은 이 논문을 설명한다..
- 근데 **구현해 놓은 코드는 이 논문이 아닌 경우가 많다. 왜일까?**



Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

역시 마크1은 쓰는 거 아니야

1

Neural Style Transfer의 원리1: CNN-based

- 이 모델은 처음으로 NST를 잘 수행했지만 문제가 하나 있다.
- 바로 **처음 본 질감을 Transfer를 할 수는 없다.**
 - 즉, 질감이 바뀔 때마다 새로 학습해야 한다.
 - 하드웨어가 좋아져서 새로 학습하는 게 큰 문제는 아니지만, 보통 AI에서는 늘 일반화를 향해 간다.
 - 그리고 종종 이상한 스크래치가 나타나는 현상도 있었다.
- 그래서 이 논문을 시작으로 굉장히 다양한 연구들이 진행되었다.
 - **일반화**를 위한 노력이 많았다.
 - Gram Matrix 부분이 정확히 이해가 안 갔다면 정상이다.
이 원리를 밝혀 내기 위한 연구들도 많다.
 - VGG-19뿐 아니라 **다른 모델**을 사용한 경우도 많다.



여태 살펴본 논문으로 만든 결과물
하단에 kg 받는 무언가 있다.



AdalIN Pre-trained로 만들어낸 결과물
훨씬 이쁘다...

- 일반화 논문 중에 제일 널리 알려진 것 중 하나는 AdaIN이다. (또 CVPR)
 - Tensorflow에서 제공하는 Pre-trained weight도 AdaIN이다.
 - Neural Style Transfer 검색했을 때 나오는 대부분의 설명글 또한 -
- 여기서 정말 온갖 새로운 테크닉을 넣었는데 모두 살펴볼 시간이 없기 때문에 핵심적인 구조만 살펴보자.

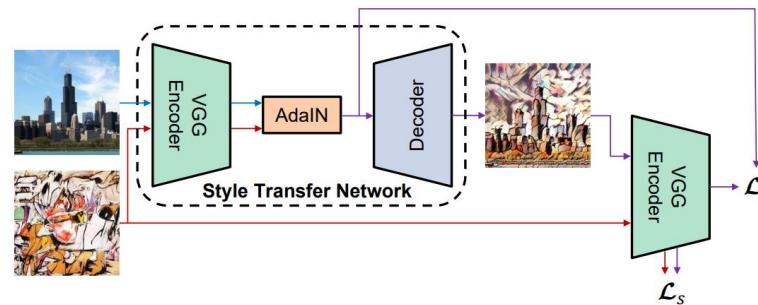


Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same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

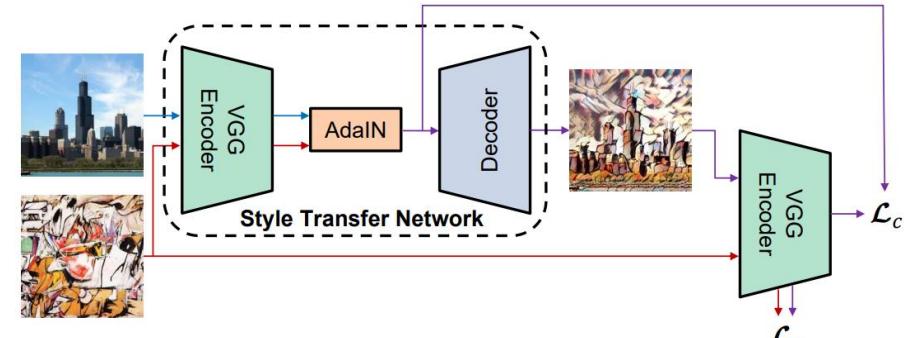


Huang, Xun, and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization." Proceedings of the IEEE international conference on computer vision. 2017.

Adaptive Instance Normalization

- 기존 방식대로 Content와 Style을 넣어준다.
 - 아까 배웠던 대로 똑같은 방식으로 학습해준다.
 - Feature map 몇 개 뽑아서 Content도 만들고 Style도 만들고 ~
 - 여기서 이 **인코딩된 Content에 Style을 입혀준다.**
 - 옆의 수식에서 Content가 x , Style이 y
 - 하나의 이미지 단위에서 일어나기 때문에 이 기법을 Adaptive Instance Normalization이라 부르기로 했다.
 - Style이 입혀진 Content를 디코더에 넣어서 Upsampling을 한다.
- 훈련을 하는 방식은 기존 방식과 유사하다.
 - 최종 스타일이 입혀진 결과물을 다시 인코더에 넣어서 Content를 뽑아낸다.
 - 그리고 이 Content가 처음에 입혀진 Content랑 유사한지 살펴본다
 - Style의 경우는 인코더에

Neural Style Transfer의 원리1: CNN-based

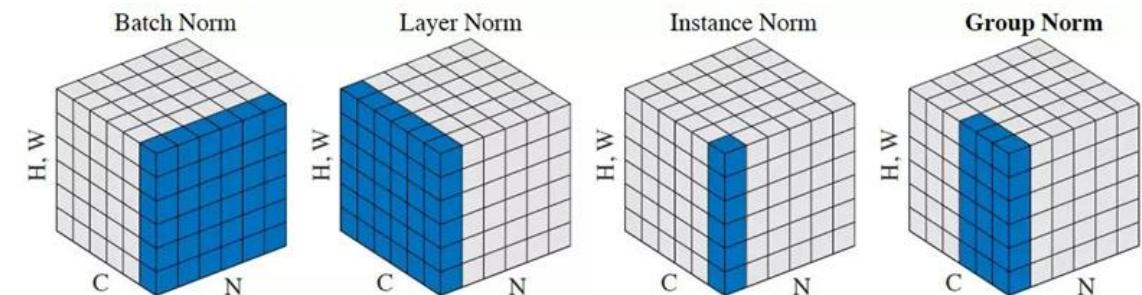


$$\text{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

$$\begin{aligned} \mathcal{L}_c &= \|f(g(t)) - t\|_2 \\ \mathcal{L} &= \mathcal{L}_c + \lambda \mathcal{L}_s \\ \mathcal{L}_s &= \sum_{i=1}^L \|\mu(\phi_i(g(t))) - \mu(\phi_i(s))\|_2 + \\ &\quad \sum_{i=1}^L \|\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))\|_2 \quad (13) \end{aligned}$$

Huang, Xun, and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization." Proceedings of the IEEE international conference on computer vision. 2017.

- 이미지 분야에서 널리 쓰이는 기법 중 하나는 **Batch Normalization**이다.
 - Feature 단위로 값들을 조정해주고 다음 레이어로 보내는 기술이다.
 - 여기서 조정은 같은 배치 내에 있는 다른 데이터들과 Feature별로 표준화를 해주고
 - 학습이 가능한 값으로 스케일링을 해준다는 뜻이다.
 - 이게 필요한 이유는, 레이어가 학습할 때마다 너무 차이가 나는 분포를 받아서 학습을 하다 보니 값이 너무 튀어서 작은 학습률(learning rate)을 사용할 수 밖에 없었다.
 - 실제로 CNN은 BN이 생기기 전과 후로 나뉠 정도로 성능과 속도개선에 지대한 영향이 있었던 논문이다.
 - 이게 CNN Activation 이후에 연산이 도입되면서 CNN 모델에서 큰 학습률을 사용할 수 있게 되었다.
- Style Transfer에서도 먹힐까?
 - BN이 잘 되긴했는데, 다른 후속 연구 중에 IN, Instance Normalization이 더 잘된다는 연구가 나왔다.
 - 이 논문의 저자도 이를 캐치하고 IN을 도입했고 특별히 Style Transfer를 위해서 저 조정값들을 Style에서 배운다.



Features

Batch

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu_{batch}}{\sigma_{batch}}$$

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

2

Neural Style Transfer (NST) 의 원리2: GAN-based

- ✓ GAN도 많이 쓴다던데… GAN은 또 뭔가요
- ✓ 초석이 되는 논문 몇 개 짹먹

■ Generative Adversarial Network 기반

- ❖ Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017. ([CycleGAN](#))
- ❖ Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019. ([StyleGAN](#))



- 생성 기반 Style Transfer도 파벌이 있다.
 - 앞서 살펴본 CNN 계열의 경우, 이미지 자체의 표현을 활용해서 기존 이미지에 스타일을 “입힌다” “합성한다”는 개념에 가깝다.
 - 반면, 새롭게 생성형식으로 진행되는 모델들도 있는데 보통 Generative 모델(생성 모델)이라 부른다.
- 여기도 약간의 계보가 있다.

Isola, Phillip, et al.

"Image-to-image translation with conditional adversarial networks (pix2pix)"

Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

Goodfellow, Ian, et al.

"Generative adversarial nets. (GAN)"

Advances in neural information processing systems 27 (2014).

Zhu, Jun-Yan, et al.

"Unpaired image-to-image translation using cycle-consistent adversarial networks. (CycleGAN)"

Proceedings of the IEEE international conference on computer vision. 2017.

Karras, Tero, et al.

"Progressive growing of gans for improved quality, stability, and variation. (PGGAN)"

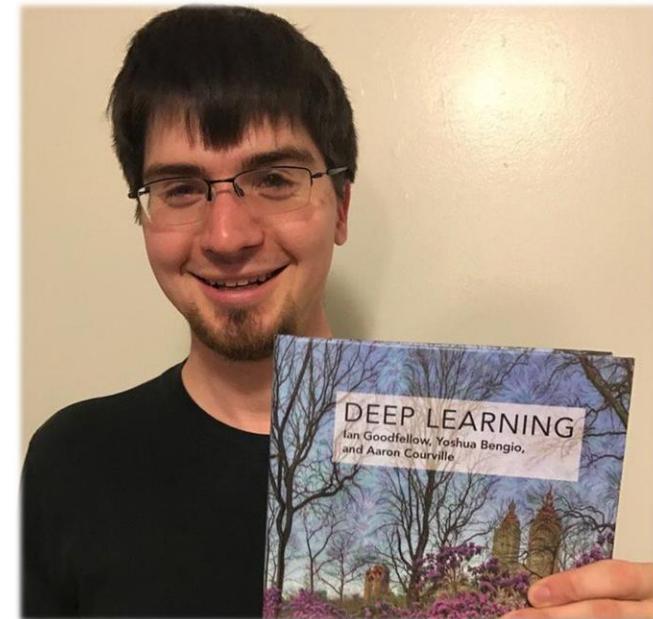
arXiv preprint arXiv:1710.10196 (2017).

Karras, Tero, Samuli Laine, and Timo Aila.

"A style-based generator architecture for generative adversarial networks. (StyleGAN)"

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

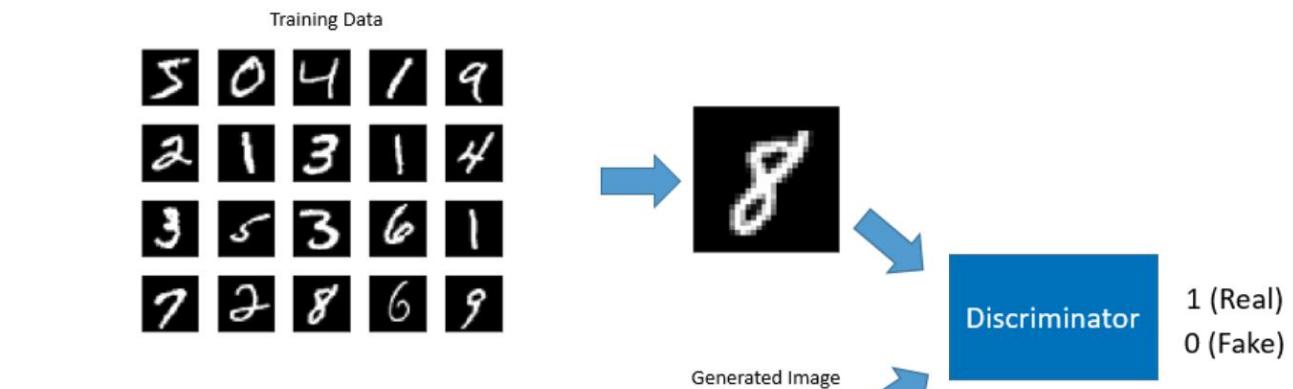
- GAN은 인공지능계의 거성 Ian Goodfellow가 2014년에 발표한 새로운 네트워크 형태이다.
- Generative Adversarial Networks 의 줄임말로
 - 일단 “**생성**” 모델이고
 - “**적대적**”이라는 말도 들어간다.
 - 이 두 단어를 머리 속에 심어 놓고 진행하자.
- 생성 모델은 우리가 전에 수업시간에 배웠던 모델들과는 구분된다.
 - 분류 모델의 경우는 데이터를 분류하는 것을 목표로 학습한다.
 - 반면 생성 모델은 **한 분류의 데이터를 만들어내는 것**으로 시작한다.



이 바이블의 저자

- GAN 기본 구조는 신경망이 두 개가 포함된다는 점이다.
 - 그리고 이름에 포함된 “적대적”은 이 두 신경망이 싸우고 있다는 뜻이다.
 - 하나는 이미지를 계속해서 만들어내는 신경망이라 **생성자**라 부른다.
 - 다른 하나는 실존하는 데이터와 만들어내진 가짜 데이터를 구분하는 **판별자**라 부른다.
- 딥러닝은 항상 학습 목표가 있어야 한다.
 - 생성자는 판별자를 속이도록 학습한다.
 - 판별자는 짹퉁을 구분하도록 학습한다.

이렇게 학습하면 결과적으로 생성자는 판별자가 구분을 못하도록 열심히 학습을 하기 때문에 **“그럴싸한” 이미지를 만들어낼 수** 있다.
- 이런 학습 방식은 최적화가 조금 힘들었지만 극복하기 위한 다양한 연구가 나타나면서 여러 분야에 적용되었다.
 - 그 중 하나가 Style Transfer 영역이다.
 - 수많은 GAN-based 논문이 있지만
대표적인 CycleGAN과 StyleGAN만 간단하게 살펴보자.



Latent Sample

$$\begin{array}{c} -0.19972104, \ 0.42638235, \ -0.71335986, \\ 0.27617624, \ 0.04559994, \ -0.82961057, \\ 0.74493232, \ 0.305852, \ -0.81311934, \\ 0.02522898, \ 0.60752668, \ 0.42092858, \\ 0.86428853, \ 0.14307071, \ 0.42457545, \\ 0.84422774, \ 0.26477474, \ -0.398651, \\ -0.41719925, \ 0.71651308, \ 0.26192929, \\ -0.88549566, \ 0.65559716, \ -0.18518651, \end{array}$$

CycleGAN

Neural Style Transfer의 원리2: GAN-based

- 얼룩말을 말로 바꿔주는 생성자 G 가 있다고 하자. (즉, $G: X \rightarrow Y$) + 그리고 말로 잘 바뀌었는지 맞추는 판별자 D_Y 가 있다고 하자.
- 반대로 말을 얼룩말로 바꿔주는 생성자 F 가 있다고 하자. (즉, $F: Y \rightarrow X$) + 얼룩말로 잘 바뀌었는지 맞추는 판별자 D_X 가 있다고 하자.
- 이 둘이 각자의 역할을 잘 맡으려면 어떻게 해야 할까?
 - 아까 학습이 잘되려면 어떻게 해야 하는지를 떠올려보자.
 - 말 \rightarrow 얼룩말로 변환되었을 때 이 얼룩말이 찐인지 가짜인지 **잘 판별하는 것을 목표**로 한다.
 - 방금 변환한 얼룩말을 다시 말로 변환해본다. 그 때 이 말이 **원래의 말로 잘 돌아왔는지도 목표**로 한다.
 - 반대의 경우도 똑같이 진행해준다.

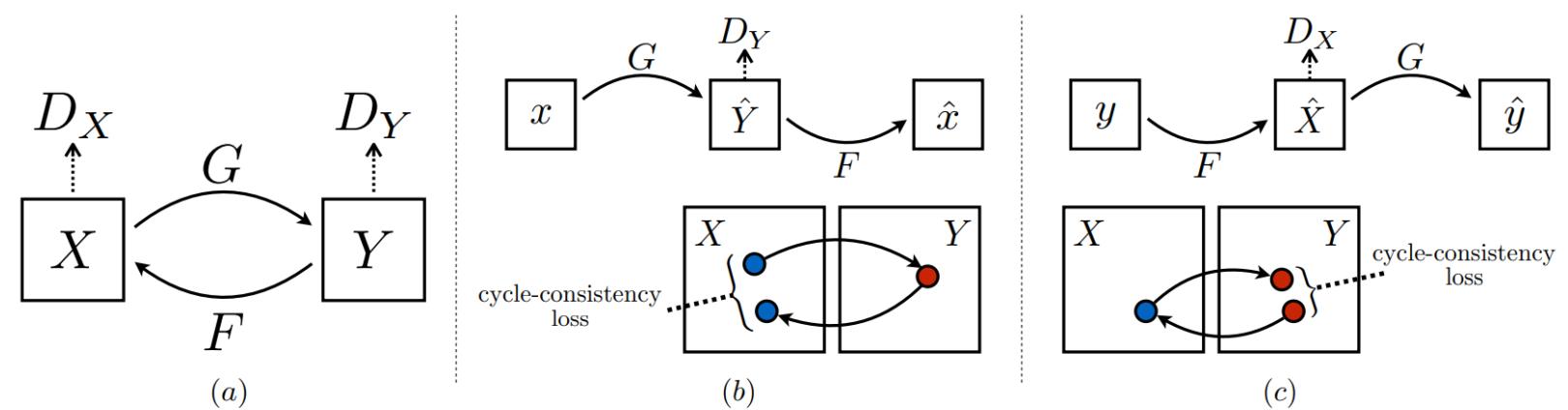
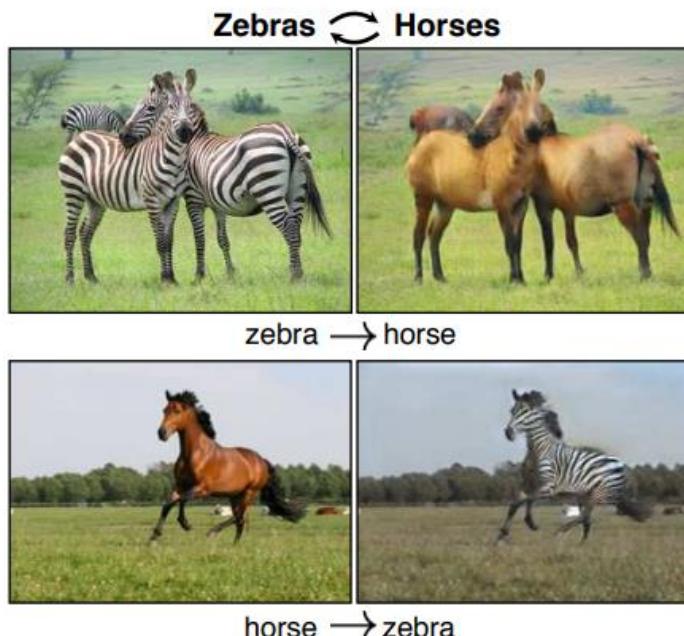


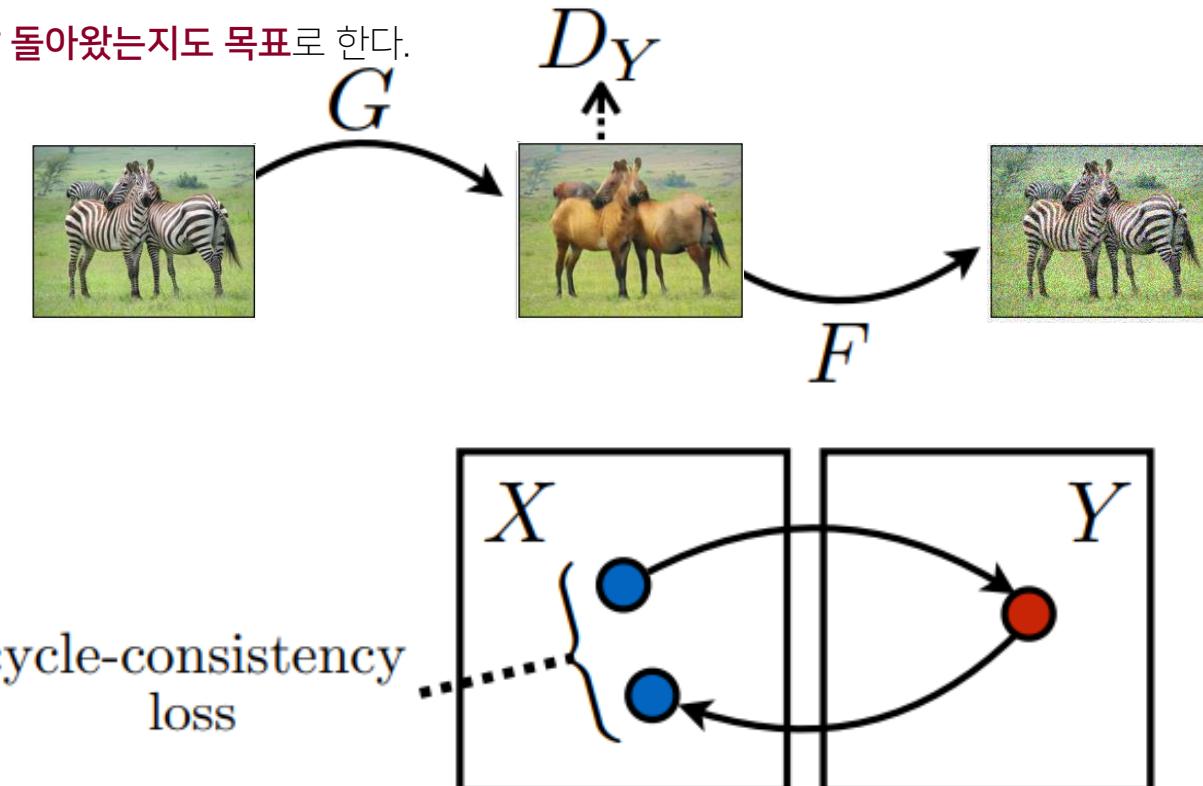
Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X . D_Y encourages G to translate X into outputs indistinguishable from domain Y , and vice versa for D_X , F , and X . To further regularize the mappings, we introduce two “cycle consistency losses” that capture the intuition that if we translate from one domain to the other and back again we should arrive where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017. (CycleGAN)

CycleGAN

Neural Style Transfer의 원리2: GAN-based

- 얼룩말을 말로 바꿔주는 생성자 G 가 있다고 하자. (즉, $G: X \rightarrow Y$) + 그리고 말로 잘 바뀌었는지 맞추는 판별자 D_Y 가 있다고 하자.
- 반대로 말을 얼룩말로 바꿔주는 생성자 F 가 있다고 하자. (즉, $F: Y \rightarrow X$) + 얼룩말로 잘 바뀌었는지 맞추는 판별자 D_X 가 있다고 하자.
- 이 둘이 각자의 역할을 잘 맡으려면 어떻게 해야 할까?
 - 아까 학습이 잘되려면 어떻게 해야 하는지를 떠올려보자.
 - 말 \rightarrow 얼룩말로 변환되었을 때 이 얼룩말이 찐인지 가짜인지 **잘 판별하는 것을 목표**로 한다.
 - 방금 변환한 얼룩말을 다시 말로 변환해본다. 그 때 이 말이 **원래의 말로 잘 돌아왔는지도 목표**로 한다.
 - 반대의 경우도 똑같이 진행해준다.

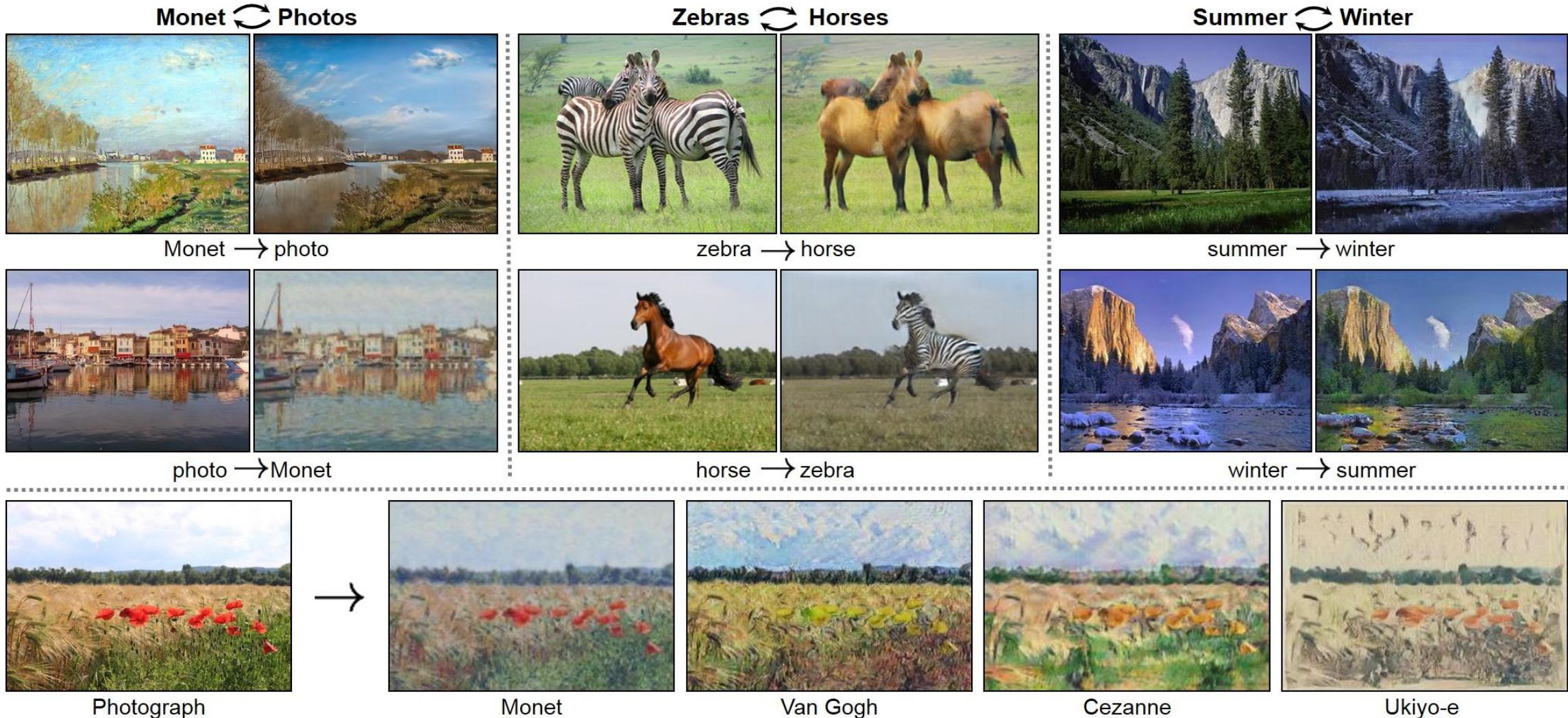


Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017. (CycleGAN)

CycleGAN

Neural Style Transfer의 원리2: GAN-based

- 결과는 제법 잘 나오는 편.



Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017. (CycleGAN)

이론은 완벽해!

Neural Style Transfer의 원리2: GAN-based

- 그럴싸해 보이는데 문제가 살짝 있다.
 - 학습이 어렵게 안된다. 응당 CycleGAN만의 문제는 아니라 큰 이슈는 아니다.
 - 구조를 바꾸진 못한다.
 - 무슨 뜻이냐면, 말이랑 얼룩말은 구조적으로 비슷하고 무늬만 다르다. 수행하면 잘 돌아간다.
근데 개랑 고양이는 무늬만 다른 게 아니라 구조 자체가 다르다. **모양을 변화는 못한다**는 뜻.
 - 마찬가지로 사과랑 오렌지도 그렇다. (오렌지는 꼬다리가 없다)
 - 데이터 분포도 파악하지 못한다.
 - 저 끔찍한 반인반수 모양을 보면 바로 이해할 수 있다.
 - 연구에서 데이터셋 문제라고 하기는 한다…

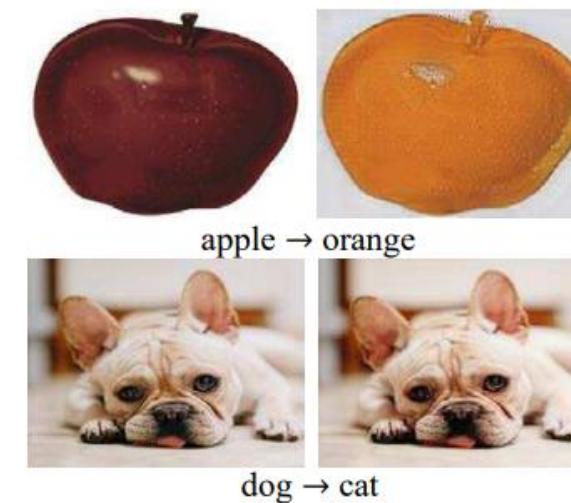
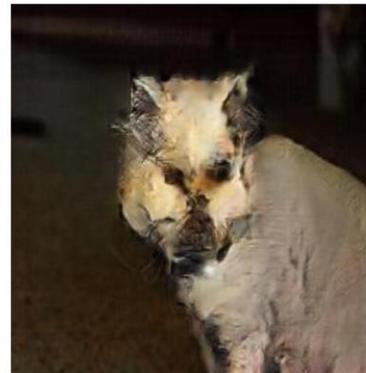
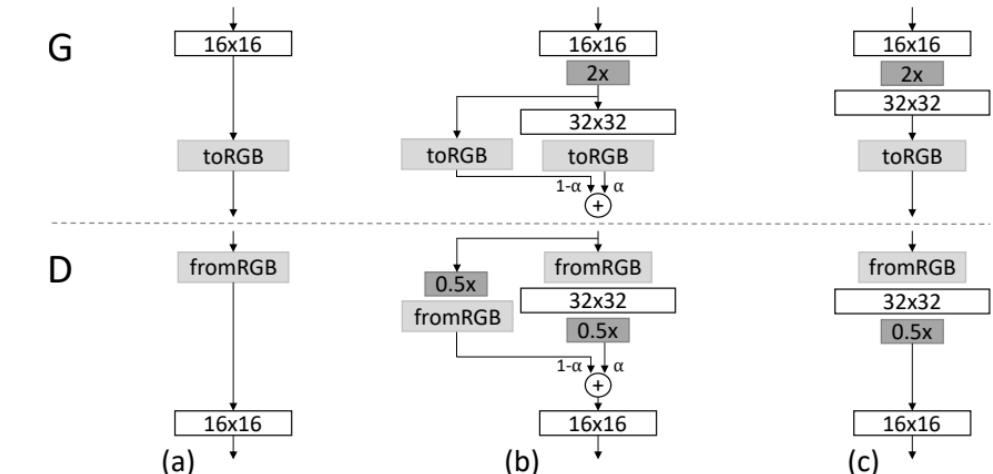
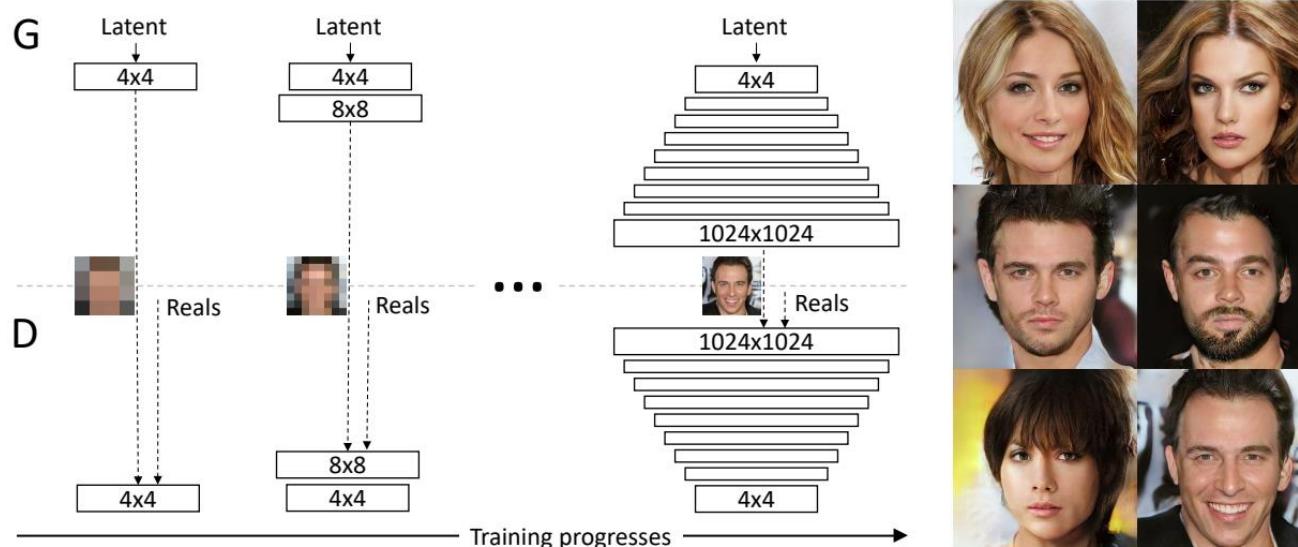


Figure 12: Some failure cases of our method.

Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017. (CycleGAN)

- Style Transfer 분야를 NVIDIA도 연구를 하기 시작했다. 그 결과 현재 Style Transfer 검색 하면 **StyleGAN이 거의 대표**로 나온다.
- StyleGAN을 알려면 이들의 선행연구인 PGGAN의 핵심을 살짝 알아야한다. 어렵지 않으니 빠르게 살펴보자.
- 생성자와 판별자가 대칭적인 구조를 가진다.
- **처음에는 저해상도의 이미지를 생성**하고 이를 판별한다.
- **학습이 진행될수록 레이어를 추가하여 생성되는 이미지의 해상도를 높인다.**
 - 해상도를 높이는 과정에서 갑자기 난데없는 레이어가 추가되면 성능이 널뛴다.
 - 그래서 기존의 저해상도 레이어에서 나온 이미지와 **새로운 레이어의 결과를 섞어서 학습**시킨다. (Smooth Fade-in)



Karras, Tero, et al. "Progressive growing of gans for improved quality, stability, and variation. (PGGAN)" arXiv preprint arXiv:1710.10196 (2017).

PGGAN으로 만들어낸 이미지

- GAN에서 자주 사용되는 데이터 중에 하나가 CelebA라는 셀럽들 얼굴 모아 놓은 데이터셋이 있다.
- PGGAN을 통해서 CelebA-HQ라는 고해상도 데이터셋을 만들어낸 것이 이 논문의 또다른 기여포인트다.
 - 화질이 높으니까 꽤나 진짜 같다. 소름 끼쳐
- 이미지가 잘은 나왔는데 **내가 조절할 수 없다는 것**이 극복해야 할 문제였다.



Figure 5: 1024×1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.

Neural Style Transfer의 원리2: GAN-based



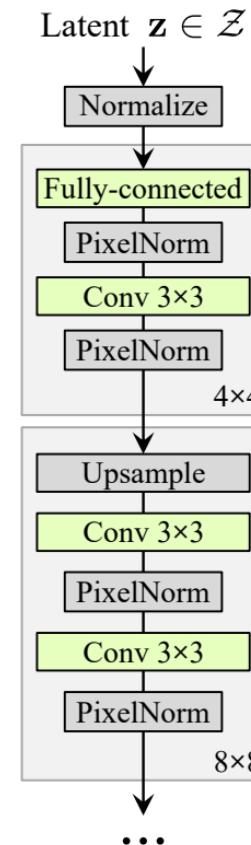
Figure 7: Selection of 256×256 images generated from different LSUN categories.

Karras, Tero, et al. "Progressive growing of gans for improved quality, stability, and variation. (PGGAN)" arXiv preprint arXiv:1710.10196 (2017).

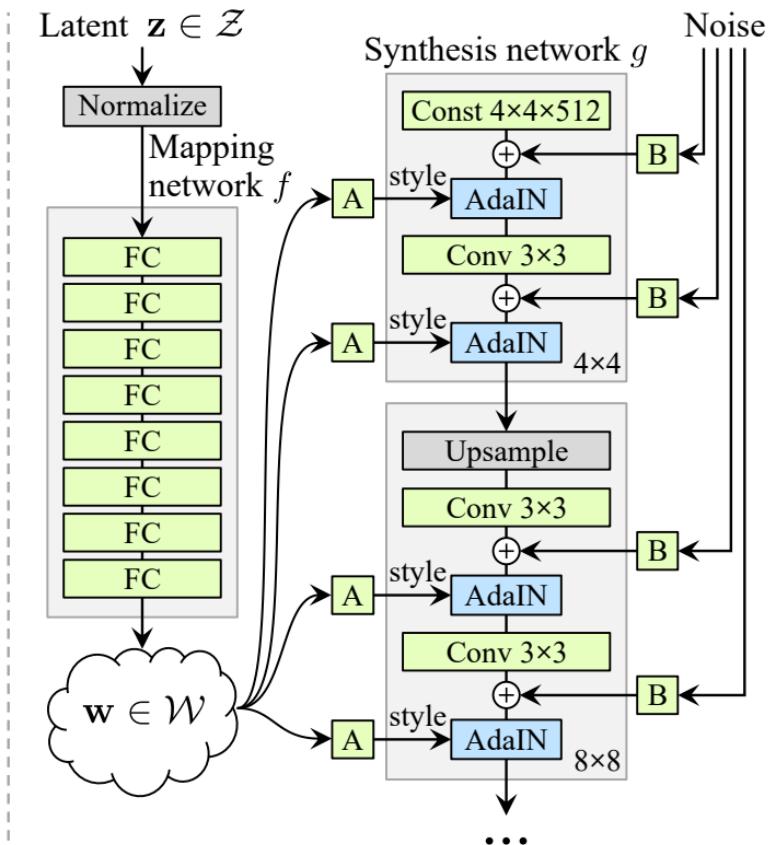
StyleGAN의 등장

Neural Style Transfer의 원리2: GAN-based

- 사실상 GAN계열의 Style Transfer를 평정했다 말할 수 있을 정도로 널리 알려지고 성능이 좋은 StyleGAN을 출시했다.
- Generation 성능이 좋은 것도 주목할 만한 점이지만, 이 논문은 **여러 방면에서 좋은 방법론**들을 제시했다.
 - 하지만 다 다루고 갔다가는 인공지능을 접겠다는 말이 나올 것 같으니 간단하게 보고가자.
- 모델 구조에서 (a)가 기존 GAN Generator의 구조이다.
 - 아까 PGGAN에서도 살펴보았듯이, GAN은 어떤 잠재 공간(latent space)의 벡터를 Generator에 보내서 이미지를 생성하는 방식이다.
 - StyleGAN의 포인트는 이 벡터를 그냥 사용하는 것이 아니라, 비선형 완전 연결망에 보내서 (Fully Connected Layer) **w라는 새로운 벡터로 Mapping**을 해준다.
 - 이 벡터를 Generator에서 사이사이에 넣어주게 된다.
 - 계속해서 Upsampling하면서 해상도가 올라가는 구조인데 **여기에서 w를 입혀서 스타일을 조절**할 수 있다.
 - 참고로 여기서 w를 입혀주는 방식이 아까 봤었던 AdaIN 방식이다
 - 해상도에 따라 입혀주는 **스타일의 미세함을 조절할 수 있다는 점**이 큰 포인트다.
- 이 밖에도 딥러닝에서 큰 화두 중 하나인 Disentanglement를 잘 구현해냈다는 점, GAN을 Generator의 구조만 변경해서 개선했다는 점, CelebA에 이어 새로운 고해상도 얼굴 데이터셋인 FFHQ를 출시했다는 점 등 관전 포인트가 많은 논문.



(a) Traditional

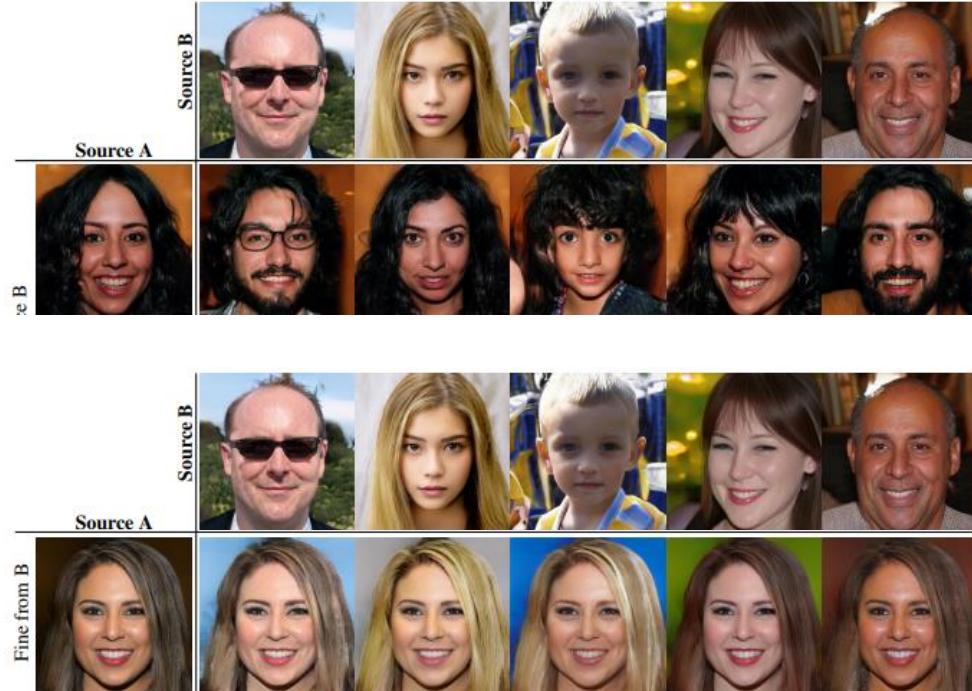


(b) Style-based generator

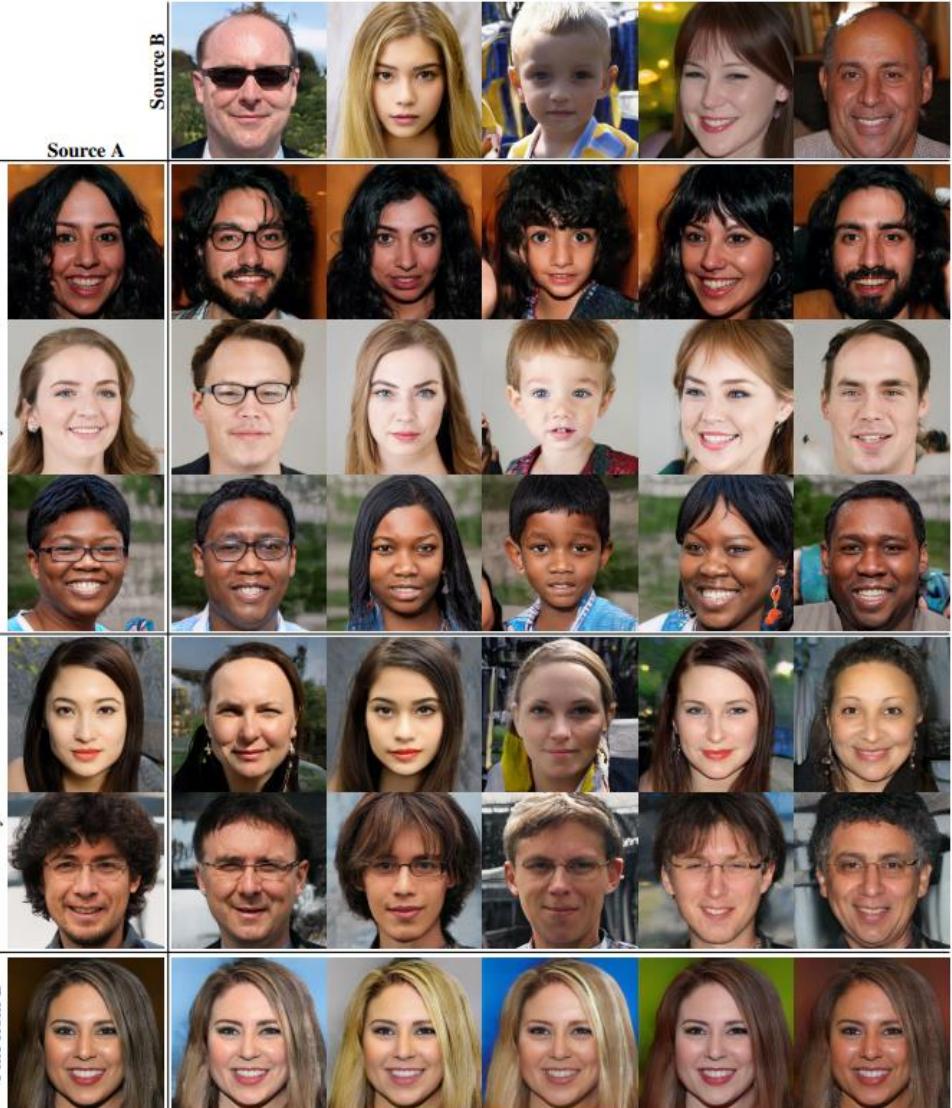


StyleGAN의 등장

- 결과를 보면 좀 놀랍다.
- Source B의 스타일을 Source A로 입히는 방식이다.
 - 그러니까 A가 Content, B가 Style
- 가로열은 내려갈 수록 B의 미세한 점들을 입힌다.
 - 윗줄은 B의 큰 스타일들, 아랫줄은 B의 작은 스타일들



Neural Style Transfer의 원리2: GAN-based



Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks" Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

- StyleGAN이 대박을 치고 그 뒤로 v3까지 출시되었다.
- 자세한 개선 포인트는... 저도 NST를 이렇게까지 파보는 건 처음이라 궁금하시면 같이 공부합시다.

Karras, Tero, Samuli Laine, and Timo Aila.

"A style-based generator architecture for generative adversarial networks."

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.



Karras, Tero, et al.

"Analyzing and improving the image quality of stylegan."

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.



Karras, Tero, et al.

"Alias-free generative adversarial networks."

Advances in Neural Information Processing Systems 34 (2021).

Style Transfer, 해치웠나?

구글에 Style Transfer로 검색했을 때 등장하는 대부분의 포스팅은
오늘 주로 살펴본 4개의 논문일 정도로 굉장히 대표적인 것들을 살펴봤다.

제일 대표적인 논문을 최대한 수학 없이 다뤄보았지만
미세한 부분들을 건드리면서 코드를 돌리려면 직접 논문을 봐야 할 수 있다!



3

Neural Style Transfer 실습하기

- ✓ 일단은 Pre-trained 불러서 돌려보기 (흥미유발)
- ✓ Django에 올려 보기
- ✓ 훈련은 어떻게 할까?



- 어떻게 돌아가는지 이론을 배워봤다.

이제 pre-trained된 모델을 불러서 사용하는 방법을 살펴보자.

1. 모델을 불러온다.
2. 이미지도 불러온다.
3. 모델에 이미지를 넣어준다.

- 배웠던 것에 비해 너무 단순한 것이 아닌가 싶다.

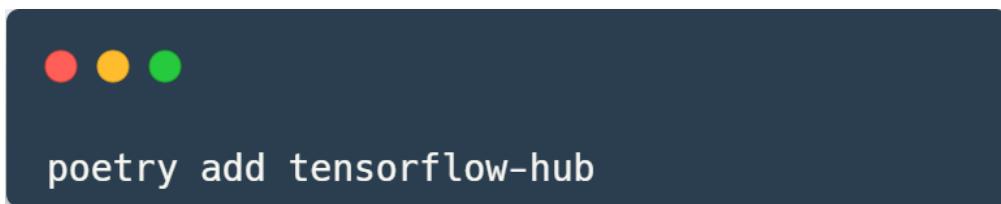
- 사실 이 단순한 코드를 가로막는 데에는 **여러 문제**가 있는데

- 하나의 모델을 불러오는 건 쉬운데, 다른 모델이 쓰고 싶으면 코드가 변경되어야 하는 점
- 이미지를 모델에 넣어 주기 까지 처리 과정
- 그리고 모델에서 나온 이미지를 다시 프론트로 보내는 과정
- 웹에서 모델은 어떻게 올려놓지?

등 자잘한 고려사항들이 많다. (~~원래 코딩은 자잘한 잔 에러 때문에 킹받는다~~)



- Tensorflow는 인공지능에 익숙하지 않은 사람들을 위해, 혹은 리소스가 한정적이어서 어떤 큰 모델을 훈련하기 힘든 사람들을 위해 **미리 저장된 모델을 제공**한다.
 - TF뿐 아니라 이런 pre-trained된 weight나 구조를 제공하는 프레임워크는 많다.
(e.g. PyTorch, HuggingFace, FairSeq, rwightman/pytorch-image-models 등)
- 홈페이지에 가면 다양한 task에 대해 여러 pre-trained weight를 제공한다.
 - 메인 페이지에 우리의 task인 스타일 전이도 있다 (5개 밖에 없으면서)
- 이를 실제로 사용하려면 우선 **tensorflow-hub 라이브러리를 설치**해야한다.



TensorFlow Hub, 학습된 머신러닝 모델의 저장소

TensorFlow Hub은 어디서나 미세 조정 및 배포 가능한 학습된 머신러닝 모델의 저장소입니다. 몇 줄의 코드만으로 BERT 및 Faster R-CNN과 같은 학습된 모델을 재사용할 수 있습니다.

```
import tensorflow_hub as hub
model = hub.KerasLayer("https://tfhub.dev/google/nlpm-en-dim128/2")
embeddings = model(["The rain in Spain.", "falls", "mainly", "In the plain!"])
print(embeddings.shape) # (4, 128)
```

모델 보기

사용 사례에 맞는 학습된 TF, TFLite 및 TFPJ 모델을 찾아보세요.

모델

TFHub.dev에서 TensorFlow 커뮤니티의 학습된 모델 찾기

BERT	객체 감지	스타일 전이	기기 내 음식 분류기
Class Label BERT Single Sentence BERT 텍스트 분류 및 질문 답변 등 NLP 작업에서 BERT를 확인해 보세요. 모델 보기	이미지에서 객체를 감지하려면 Faster R-CNN Inception ResNet V2 640x640 모델을 사용하세요. 모델 보기	이미지 하나의 스타일을 이미지 스타일 전이 모델을 사용하여 다른 이미지로 전이하세요. 모델 보기	이 TFLite 모델을 사용하여 휴대기기의 음식 사진을 분류하세요. 모델 보기

- 사용하고 싶은 모델을 찾아서 링크를 찾아준다.
 - 우리가 사용하려는 모델도 오른쪽 이미지를 보면 Copy URL이 있다.
 - 복사해서 가져오자.
- 그리고 Tensorflow-hub 모듈로 불러오면 알아서 모델을 불러온다.
- 필요한 이미지들을 “잘 처리해서” 넣어주면 변환된 이미지가 나온다.

```
import tensorflow_hub as hub

hub_module = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/1')

stylized_image = hub_module(tf.constant(content_image), tf.constant(style_image))[0]
tensor_to_image(stylized_image)
```



TensorFlow Hub

Search for models, collections & publishers

Back

magenta/arbitrary-image-stylization-v1-256

Fast arbitrary image style transfer.

Publisher: Google License: Apache-2.0

Architecture: Dataset:

Other Multiple

Overall usage data

313.5k Downloads

Model formats

TF TFLite (magenta/arbitrary-image-stylization-v1-256/fp16/prediction) V2

Hub module (v2)

Usage data: 85.6k Downloads V2

Fine tunable: No License: Apache-2.0

Format: Hub module

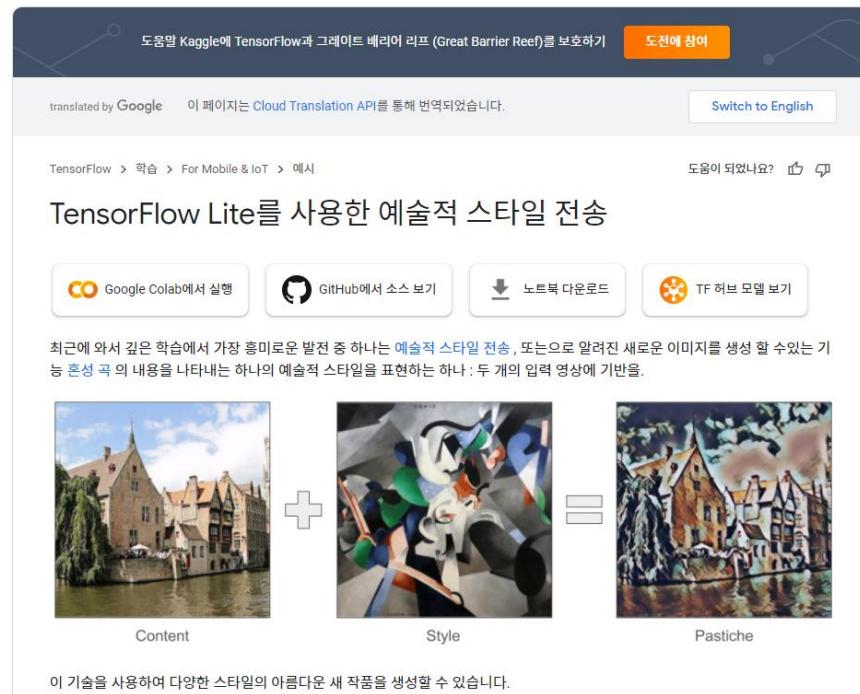
Fast arbitrary image style transfer.

Download 81.66MB

[Copy URL](https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2)

<https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2>

- Tensorflow가 그냥 일반 TF 모델도 많이 푸는데
이상하게 **Style Transfer의 경우는 TFLite모델이 대다수**다
- 애초에 모바일을 겨냥하고 경량화로 모델 크기를 제대로 갈아버린 모델이라
그냥 TF에서 쓰는게 어려울 수 있다.
- 대신에 TFLite 전용 Tutorial이 따로 있다 (심지어 Style Transfer로)
- 별도로 실습을 같이 진행하지 않을 예정이지만, 관심이 있다면 선택지에 둘보자!



```

# Run style transform on preprocessed style image
def run_style_transform(style_bottleneck, preprocessed_content_image):
    # Load the model.
    interpreter = tf.lite.Interpreter(model_path=style_transform_path)

    # Set model input.
    input_details = interpreter.get_input_details()
    interpreter.allocate_tensors()

    # Set model inputs.
    interpreter.set_tensor(input_details[0]["index"], preprocessed_content_image)
    interpreter.set_tensor(input_details[1]["index"], style_bottleneck)
    interpreter.invoke()

    # Transform content image.
    stylized_image = interpreter.tensor(
        interpreter.get_output_details()[0]["index"]
    )()

    return stylized_image

# Stylize the content image using the style bottleneck.
stylized_image = run_style_transform(style_bottleneck, preprocessed_content_image)

# Visualize the output.
imshow(stylized_image, 'Stylized Image')

```

https://www.tensorflow.org/lite/examples/style_transfer/overview

- 끝은 맞는데 이제 –
 - 이미지 어떻게 넣어줄 것인지? TF는 까다롭다구
 - 결과물은 어떻게 할 것인지?
 - 모델의 크기가 좀 되는데 Django에서 어떻게 처리하지?



*웹에서 이미지를 어떻게 통신할지는 별도로 다루지 않았습니다.

1. 자료형 변환

- Tensorflow의 모든 데이터는 라이브러리 내에서 사용하는 **tf.Tensor 타입을 사용**해야 한다.
 - 일반적으로 Python에서 이미지는 PIL Library의 객체로 사용하게 된다.
 - PIL에서 한 번에 Tensor로 넘기면 좋겠지만, 애석하게도 그 기능을 하는 함수는 없다.
 - tf.keras.utils.img_to_array**로 PIL → np.ndarray로 한 번 넘겨주고
 - tf.convert_to_tensor**를 통해 np.ndarray → tf.Tensor로 넘겨준다.
 - 사실 이건 자료형 짹어보면서 하나씩 차분하게 변환해주고 .py에 저장하면된다.

2. 사이즈 조절하기

- 모델에 따라 받는 입력의 크기가 다르다. 우선은 한쪽 변이 512가 되면서 비율을 유지하도록 짜보자.

3. 값 스케일링하기

- 딥러닝 모델은 큰 값보다는 작은 값을 선호한다.
- 또한 데이터를 Centering해주는 것이 중요하다. 하지만 여기서는 생략해도 관계 없다.

4. 배치 처리를 고려한 차원수 조절

- Tensorflow 모델은 항상 배치 처리를 상정하기 때문에 차원수에도 유의해야 한다.
 - 이미지는 (가로, 세로, 채널 수)의 3차원 데이터를 가지지만, 배치 처리 때문에 (배치, 가로, 세로, 채널 수)의 4차원 형태를 넘겨줘야 한다.
 - 하나의 이미지만 사용한다면 배치=1로 사용하면 된다.

```
● ● ●

from PIL.Image import Image
import numpy as np
import tensorflow as tf

# 이미지 경로
image_path: str = "image.jpeg"

# PIL로 불러오기
pil_image: Image = Image.open(pil_image)

# PIL > np.ndarray
np_image: np.ndarray = tf.keras.utils.image_to_array(pil_image)

# np.ndarray > tf.Tensor
tf_image: tf.Tensor = tf.convert_to_tensor(np_image, dtype=tf.float32)

# tf.Tensor 형변환하기
# 1. 모델에 맞게 Resize 해주기
# 2. 차원수 늘려주기

# Resize: 더 긴 변을 512로 맞춰주기
shape = tf.cast(tf.shape(tf_image)[ :-1], tf.float32)

max_dim = 512
long_dim = max(shape)
scale = max_dim / long_dim

new_shape = tf.cast(shape * scale, tf.int32)

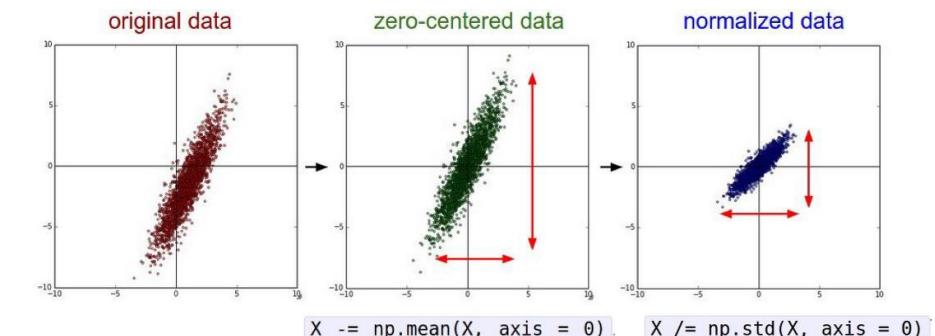
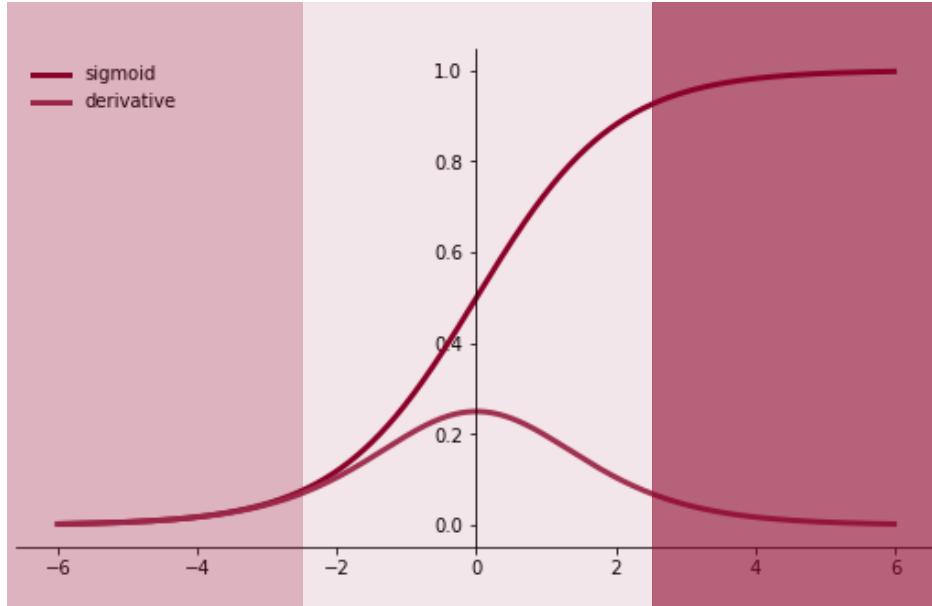
resized_image = tf.image.resize(tf_image, new_shape) / 255.0

# 차원수 늘려주기
expanded_image = resized_image[tf.newaxis, :]
```

- 딥러닝 모델들은 **작은 값을 받아야 학습이 잘** 된다.
- 자세하게 설명하기엔 너무 길어지겠지만 **가장 대표적인 이유는 활성화 함수** 때문이다.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

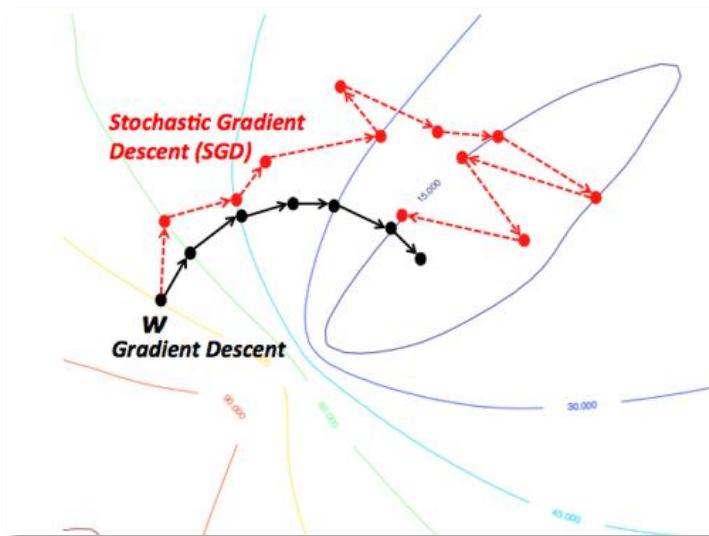
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



- 딥러닝 모델이 데이터를 하나하나 받고 있으면 학습이 끝날 수가 없다.
- 그래서 프레임워크는 **데이터를 모아서 한 번에 처리하도록 구현**되어 있다.
- 한 번에 모은 데이터의 수를 batch size라 부른다.

Batch Size가 클 수록 학습이 잘되나요?

- 또 설명이 길어질 수 있는 부분인데 이는 딥러닝을 최적화하는 방식에서 어느 정도 답이 될 수 있다.



Help protect the Great Barrier Reef with TensorFlow on Kaggle [Join Challenge](#)

TensorFlow > API > TensorFlow Core v2.8.0 > Python 도움이 되었나요? [Upvote](#) [Downvote](#)

tf.nn.conv2d

[TensorFlow 1 version](#) [View source on GitHub](#)

Computes a 2-D convolution given `input` and 4-D `filters` tensors.

```
tf.nn.conv2d(  
    input, filters, strides, padding, data_format='NHWC', dilations=None,  
    name=None  
)
```

The `input` tensor may have rank 4 or higher, where shape dimensions `[:-3]` are considered batch dimensions (`batch_shape`).

Given an input tensor of shape `batch_shape + [in_height, in_width, in_channels]` and a filter / kernel tensor of shape `[filter_height, filter_width, in_channels, out_channels]`, this op performs the following:

1. Flattens the filter to a 2-D matrix with shape `[filter_height * filter_width * in_channels, output_channels]`.
2. Extracts image patches from the input tensor to form a *virtual* tensor of shape `[batch, out_height, out_width, filter_height * filter_width * in_channels]`.

- 아까 다룬 일련의 명령어들을 *image2tensor*라는 함수로 묶어버리자.
- 그리고 아까 불러온 모델에 그대로 넣어준다.
- 여기서 드는 의문: Django 서버에서 아래 hub.load **메소드를 언제 어디서 사용하는 게 좋을까?**
 - 그냥 일반 views.py에다 넣으면 Transfer를 할 때마다 모델을 불러오게 생겼다.
 - **그래서 apps.py가 있다!**

```
content_path = "tfapp/static/content_images/content1.jpeg"
style_path = "tfapp/static/style_images/style1.jpeg"

content_image = image2tensor(content_path)
style_image = image2tensor(style_path)

hub_module = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/1')
stylized_image = hub_module(content_image, style_image)[0]
tensor_to_image(stylized_image)
```



- 항상 Django-admin startapp <앱이름>을 통해 새로운 앱을 만들었다.
- 이 앱의 기본정보는 app.py 내에 자동으로 생성된 <앱이름>Config 형태로 보관되어 있다.
- Django의 runserver를 수행하면 INSTALLED_APPS 내에 있는 **앱들을 찾아서 불러온다.**
 - 여기서 INSTALLED_APPS 목록에 있는 앱 중에 우리가 만든 앱들을 찾으려면 App별로 설정값이 별도로 보관되어 있어야한다.
 - 이걸 보관하는 것이 apps.py에서 기본생성되는 AppConfig의 역할 중 하나이다.
 - 이 apps.py는 백그라운드에서 계속 실행되기 때문에 지속적으로 사용해야하는 모델은 이 안에서 불러오면 반복적으로 호출되지 않고 사용될 수 있다.

```
# tfapp/apps.py

from django.apps import AppConfig
import tensorflow_hub as hub

class TfappConfig(AppConfig):

    default_auto_field = 'django.db.models.BigAutoField'
    name = 'tfapp'
    model = hub.load("https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/1")
```

```
# tfapp/views.py
from tfapp.apps import TfappConfig, image2tensor, tensor2image
...

def transfer(request):
    ...
    transferred_tensor = TfappConfig.model(content_tensor, style_tensor)[0]
    ...
```

- 앞에서 소개한 아주 다양한 논문들은 **구현 방법도 제각각**이다.
- 최대한 바로 사용할 수 있는 코드들을 찾아보려 했는데 많이 없다.
 - 유행할 당시가 2017~2019년인데 이 때가 Tensorflow v1을 사용할 때고 (악마의 언어)
연구분야에서는 PyTorch로 넘어갈 당시이다보니, Tensorflow v2를 사용해 구현된 것이 많이 없는 게 아니고 그냥 전멸
 - 현재로써 가장 쉽게 사용할 수 있는 모델은 **아까 소개한 tensorflow-hub에 올라간 AdaIN 구조**다.

그럼 우째요

- 위에서 제공한 모델을 사용한다.
- CycleGAN, StyleGAN의 경우는 **그래도 무지성으로 훈련할 수 있게 제공된 코드**들이 있다.
 - 여기서 훈련한 걸 직접 옮겨보는 방법도 나쁘지 않다.
 - 리소스가 문제가 될 수 있는데 StyleGAN의 경우는 고해상도 이미지를 만들어내려면 최소 VRAM 16GB가 달려 있어야한다.
(대충 스펙이 엄청 좋은 GPU가 있어야 한다는 뜻)



- 진짜 개인 연구 코드보다도 열심히 찾아본 것 같은데
정말 Tensorflow 2.x로 나온 건 찾기가 힘들다. (대부분 TF 1.x or PyTorch 아이고)

AdaIN

- ✓ [Elleryqueenhomels/arbitrary_style_transfer](#)

NST

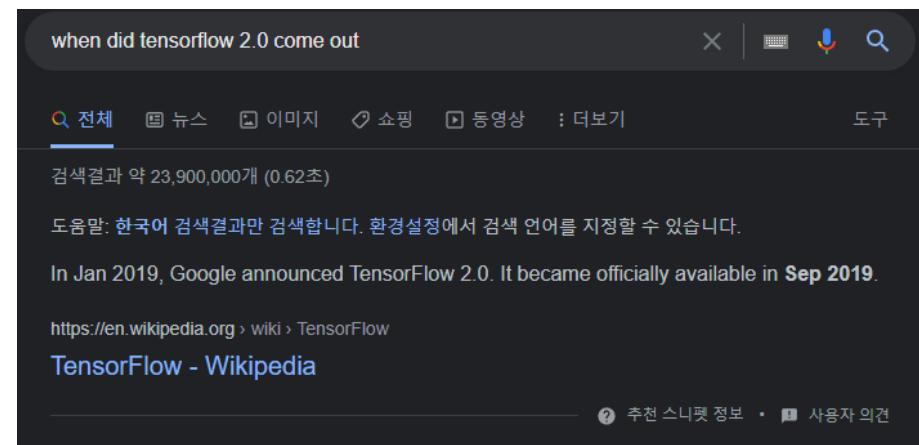
- ✓ [Keras Neural Style Transfer Tutorial](#)
- ✓ [Sckimynwa/cs231/styletransfer](#)

CycleGAN

- ✓ [Tensorflow CycleGAN Tutorial](#)
- ✓ [Keras CycleGAN Tutorial](#)
- ✓ [AakashKumarNain/CycleGAN_TF2 Pre-trained Models](#)

StyleGAN

- ✓ [Nvlabs/stylegan2](#)
- ✓ [Keras Face Image Generation with StyleGAN](#)



4

작지만 소중한 팁

- ?
 다른 NST 논문 찾아볼 때 팁이 있나요?
- ?
 논문 별로 구현된 코드를 찾아볼 수 있나요?
- ?
 준비하면서 겪은 고난과 역경
- ?
 미세먼지 팁



OpenCV vs. TensorFlow

- 두 모듈 모두 좋은 라이브러리다. 다만 용도가 조금 다르다.



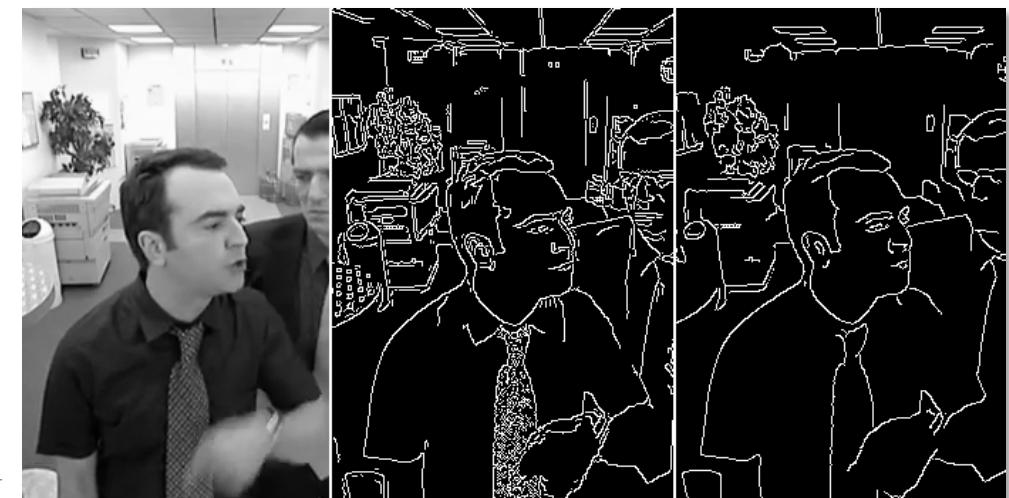
- 다양한 비전 관련된 함수들을 사용할 수 있음
 - 얼굴 랜드마크 뽑기
 - 경계선 찾아서 없애기
- 예전보다 Pre-trained weight가 많이 늘어났음
- 인공지능 모델 관련해서 자유도는 많이 떨어짐
- 설치가 화남



- 딥러닝 네트워크를 더 자세하게 Customize 할 수 있다.
- 근데 pre-trained만 불러서 쓴다면 굳이 쓸 필요는 없다.



얼굴의 Landmark 추출
SNOW 앱 만들기 가능



- 인공지능도 분야가 아주 많습니다. 처음 보는 분야를 갑자기 봐야하는 상황이 생겼을 때 어떻게 시작하면 좋을까?

- 일단 어지간한 논문을 모두 조회할 수 있는 www.scholar.google.co.kr에 접속
(학교 별로 방법이 살짝 다르지만 라이선스가 있으면 다수의 논문을 무료 조회할 수 있음)

- 가령 Neural Style Transfer를 공부하고 싶으면 검색창에
"neural style transfer **survey**"라고 검색!

- 논문도 다양한 종류가 있는데
 - 새로운 모델/방법론을 제시한 논문 (거의 주류)
 - 새로운 데이터셋을 제시한 논문 (우리가 아는 데이터셋은 죄다 논문이 나와있다)
 - **Survey 논문**: 한 분야에 대해 관련 연구를 모두 모아 놓고 이를 정리하는 논문
(Review Paper라고도 함)

The screenshot shows the Google Scholar search interface. At the top, there's a search bar with the query "neural style transfer survey". Below it, there are two radio buttons: "모든 언어" (All languages) and "한국어 필터" (Filter by Korean). To the right of the search bar is a magnifying glass icon. On the left, there's a sidebar titled "추천 학술자료" (Recommended academic papers) with a list of five papers. Each entry includes the title, authors, journal, and a "PDF" link. The titles include "Distance-based Probabilistic Data Augmentation for Synthetic Minority Oversampling", "A Novel Deep Learning Bi-GRU-I Model for Real-Time Human Activity Recognition Using Inertial Sensors", "Training Vision Transformers with Only 2040 Images", "Noninvasive Human Activity Recognition Using Millimeter-Wave Radar", and "Improved Human Activity Recognition Using Majority Combining of Reduced-Complexity Sensor Branch Classifiers". On the right side of the search results, there's a vertical sidebar with a "로그인" (Login) button at the top.

그 와중에 추천 그만해... 못 읽은 게 더 많아

관심 분야 다른 논문 찾아보기

Backups



- 검색을 해보면 다양한 Survey 논문을 볼 수 있다. (Review도 마찬가지)
- Survey 논문은 사실 아주 유명한 연구자가 하는 경우는 드물다.
- 이런 리뷰 논문을 고를 때는
 - 저자가 평소에 어떤 연구를 했는지 (자연어 하던 사람이 난데 없이 비전 리뷰를? Ban)
 - 인용횟수 (사실 이것만 봐도 됨)
- 고려하지 않아도 되는 점은 어디에 실렸는지 인데,

서베이의 경우는 분야랑 상관없이 뜯금 없는 학회/저널에 게재되는 경우가 많다 (대체 왜지)

The screenshot shows a search results page with the query "neural style transfer survey". The first result is a PDF titled "Neural style transfer: A review" by Jing Yang, Feng Ye, Yu... from IEEE transactions on..., 2019 - ieeexplore.ieee.org. The second result is a PDF titled "Demystifying neural style transfer" by Li Wang, Liu, Hou from arXiv:1701.01036, 2017 - arxiv.org. The third result is a PDF titled "Deep learning for text style transfer: A survey" by Jin, Jin, Hu, Vechtomova from Computational..., 2021 - direct.mit.edu. The fourth result is a PDF titled "Advanced deep learning techniques for image style transfer: a survey" by Liu, X, RR, Ji, W Ma from Signal Processing: Image Communication, 2019 - Elsevier. The fifth result is a PDF titled "Structure-preserving neural style transfer" by Cheng, XC Liu, J Wang, SP Lu... from Image Processing, 2019 - ieeexplore.ieee.org. The sixth result is a PDF titled "A survey on image style transfer approaches using deep learning" by Zhao, C from Journal of Physics: Conference Series, 2020 - iopscience.iop.org. The seventh result is a PDF titled "Locally controllable neural style transfer on mobile devices" by Reimann, M Klingbeil, S Pasewaldt, A Semmo... from The Visual..., 2019 - Springer. The eighth result is a PDF titled "Projecting emotions from artworks to maps using neural style transfer" by Bogucka, L Meng from Proceedings of the ICA, 2019 - pdfs.semanticscholar.org. The ninth result is a PDF titled "Photorealistic style transfer with screened poisson equation" by Mechrez, E Shechtman, L Zelnik-Manor from arXiv preprint arXiv..., 2017 - arxiv.org.



- 사실 **이해는 뒷전이고 구현이 먼저인 경우가 많다.**

구현된 걸 보고 논문이 이해되는 경우도 많기 때문에 틀린 방법은 아니다.

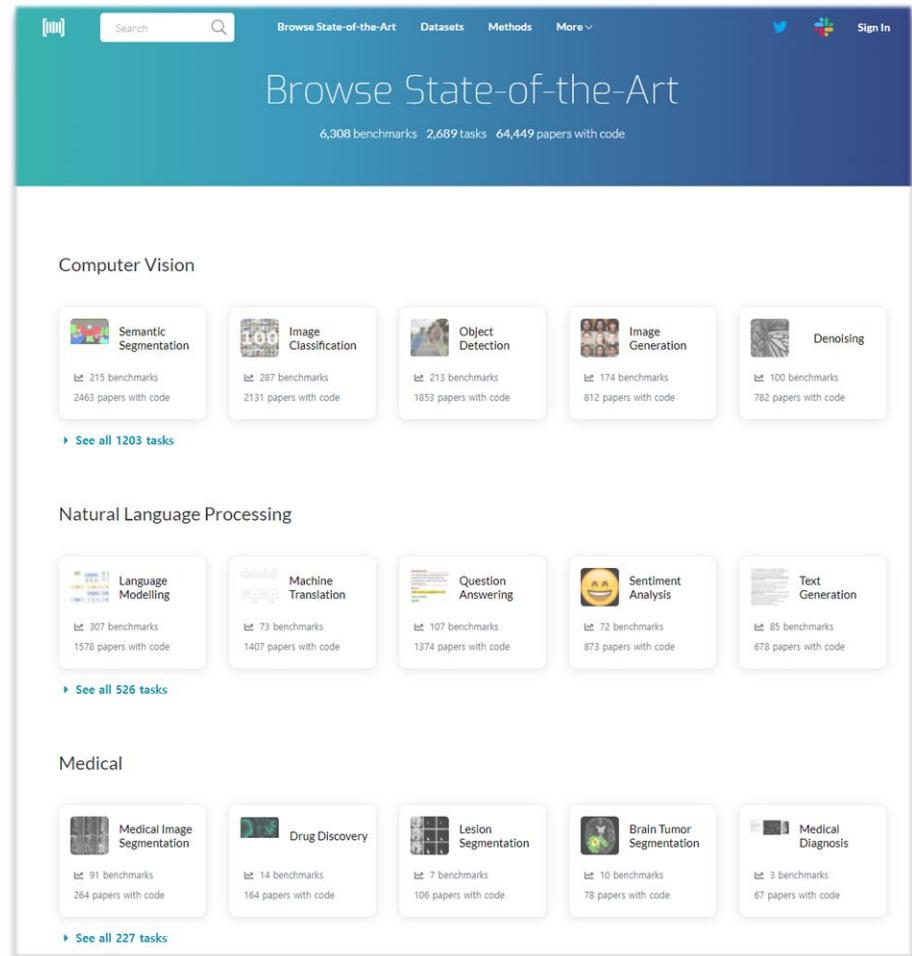
- 인공지능 논문의 경우 많이들 논문 내에 **자신들의 github 주소를 남겨준다.**

so a texture model needs to be agnostic to spatial information. A summary statistic that measures the spatial information in the feature maps is given by the correlations between the responses of

'Source code to generate textures with CNNs as well as the rescaled VGG-19 network can be found at <http://github.com/leongatys/DeepTextures>

3

- 하지만 논문 저자 외에 다른 사람들이 더 쉽게 구현한 경우도 있는데 이를 검색하려면?
- 구글링할 때 “논문 제목 + Github”만 쳐도 잘 나오긴 한다.
- **하지만 www.paperswithcode.com 과 함께라면!** 어디든지!



논문 별로 구현 코드 찾아보기: paperswithcode

Backups

4

- 보통 여기엔 대분류/task 형식으로 많이 정리되어 있다.
(가령, 비전에서 Object detection)
- Task의 종류는 굉장히 다양하게 정리 되어 있기 때문에
어지간한 건 찾을 수 있다. 당연히 Style Transfer도 그 중 하나
- Task별 페이지에는 해당 Task와 관련된 논문이 모여 있다.
- 아까 살펴본 Arbitrary Adaptive Style Transfer를 찾아보자.

The screenshot shows the 'Style Transfer' page on paperswithcode.com. At the top, there's a navigation bar with links for 'Search', 'Browse State-of-the-Art', 'Datasets', 'Methods', 'More', 'We're hiring!', and 'Sign In'. Below the navigation, a 'Computer Vision' category is selected. The main content area features a heading 'Style Transfer' with a subtext stating '373 papers with code • 0 benchmarks • 15 datasets'. It describes style transfer as 'the task of changing the style of an image in one domain to the style of an image in another domain.' Below this, there's a section for 'Benchmarks' with a link to 'Add a Result'. A sidebar on the right lists various categories: 'Introduction', 'Benchmarks', 'Datasets', 'Subtasks', 'Libraries', 'Papers' (with sub-options: 'Most implemented', 'Social', 'Latest', 'No code'), and 'Content'. The 'Libraries' section displays a list of repositories with their star counts: martiansideofthemoon/style-transfer... (122★), Yijunmaverick/UniversalStyleTransfer (541★), chuanli11/MGANs (280★), and libreal/neural-painters-x (70★). The 'Datasets' section shows icons for various datasets like DukeMTMC-reID, MPI Sintel, MPIIGaze, iKala, LaMem, Touchdown Dataset, POP909, WebCaricature Dataset, PASTEL, and DeepWriting. The 'Subtasks' section includes icons for Image Stylization, Style Generalization, Face Transfer, and Font Style Transfer. A small note at the bottom right of the page reads 'Cogsys Lab, Daehyun Cho'.

논문 별로 구현 코드 찾아보기: paperswithcode

Backups



- 해당 논문페이지로 넘어가는 링크들이 있고 무려 Code라는 탭에는 모든 구현체들을 볼 수 있다.
- 친절하게 어떤 라이브러리를 메인으로 사용했는지까지 정리되어 있다.
- 확실히 2017년 논문이라 아직 TF와 PyTorch가 싸우는 광경을 목격할 수 있다.
- 이게 구현체가 많다고 좋아할 게 아니다. 이 중에 내가 쓸 줄 아는 거 찾으면 생각 보다 몇 개 안 나온다. (**배제해야하는 경우가 너무 많다**)
 - 나랑 OS가 다른 경우 (빨리 mac/ubuntu로 갈아타자)
 - 모듈 버전 명시 안 해놓고 옛날 모듈 쓰는 경우
 - Tensorflow v1 인 경우 (tf.v1만 봐도 차가 떨리고 집에 근심이 생길 것만 같다)
 - 혹은 토치를 사용하는 사람들의 경우 구-Torch가 등장할 수 있다 (확장자가 .lua) 이것도 지양...
 - 그냥 코드가 거지 같은 경우도 많다

The screenshot shows the paperswithcode website interface. At the top, there's a search bar and navigation links for 'Browse State-of-the-Art', 'Datasets', 'Methods', 'More', and social media links ('We are hiring!', Twitter, LinkedIn, GitHub). Below the header, the specific paper is displayed:

Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization
ICCV 2017 · Xun Huang, Serge Belongie · [Edit social preview](#)

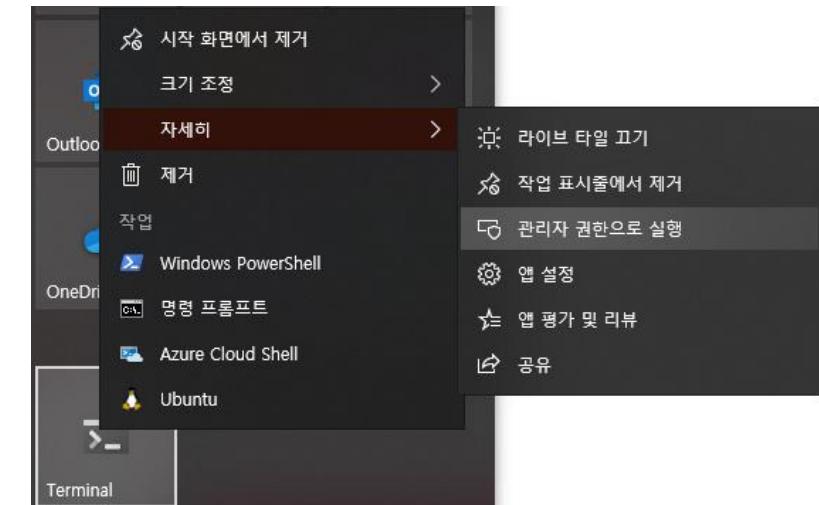
Gatys et al. recently introduced a neural algorithm that renders a content image in the style of another image, achieving so-called style transfer. However, their framework requires a slow iterative optimization process, which limits its practical application. Fast approximations with feed-forward neural networks have been proposed to speed up neural style transfer. Unfortunately, the speed improvement comes at a cost: the network is usually tied to a fixed set of styles and cannot adapt to arbitrary new styles. In this paper, we present a simple yet effective approach that for the first time enables arbitrary style transfer in real-time. At the heart of our method is a novel adaptive instance normalization (AdaIN) layer that aligns the mean and variance of the content features with those of the style features. Our method achieves speed comparable to the fastest existing approach, without the restriction to a pre-defined set of styles. In addition, our approach allows flexible user controls such as content-style trade-off, style interpolation, color & spatial controls, all using a single feed-forward neural network.

Below the abstract, there are four download links: PDF, Abstract, ICCV 2017 PDF, and ICCV 2017 Abstract.

The main content area lists various code implementations for this task:

Code	Tasks
xunhuang1995/Adain-style	Style Transfer
vincent-thevenin/Realistic-Neural-T...	PyTorch
Yijunmaverick/UniversalStyleTransfer	TensorFlow
eridgd/WCT-TF	TensorFlow
KaiyangZhou/mixstyle-release	PyTorch
bethgelab/stylize-datasets	PyTorch
PacktPublishing/Hands-On-Image-Gene...	TensorFlow
KaiyangZhou/ssdg-benchmark	PyTorch
GuoShi28/Caffe-Instance-Normalizati...	PyTorch
CellEight/Pytorch-Adaptive-Instance...	PyTorch
J3698/AdainN-reimplementation	PyTorch
srihari-humbarwadi/adain-tensorflow...	TensorFlow
ZVK/talking_heads	PyTorch
gs18113/Adain-TensorFlow2	TensorFlow
ZVK/Talking-Heads	PyTorch
aadhithya/Adain-pytorch	PyTorch
AlbanSeurat/keras-style-transfer	TensorFlow

- 애초에 연구를 PyTorch로 진행하고 모든 작업이 Ubuntu에서 진행되고 그마저도 원격으로 작업할 때는 macOS로 진행되다보니까 ... Tensorflow + Windows + Poetry 폭격에 정신을 못 차리는데 ...
- 진짜 발표 준비보다 poetry 건드리는 시간이 더 오래 걸렸을 정도로 모듈 설치가 힘들었...습니다....
- Windows에서 Poetry를 사용하면서 뜨는 몇 가지 에러들과 저의 경우에는 해결되었던 해결책 몇 가지 소개하면
 - WinError2 보통 파일/디렉토리가 존재하지 않거나 틀렸을 때 발생
Poetry 설치 중에 발견했는데 일단 제가 이름을 틀릴리 없으니 (Poetry가 하니까 ...) 에러를 자세히 읽어보니 **pip 모듈을 업그레이드 하라더군요. 합시다...**
 - WinError5 권한이 없는 경우에 발생. 널리 알려진 해결책이 많긴 한데 대표적인건
 - Cmd를 관리자 권한으로 실행. 혹은
 - 컴퓨터를 껐다 키기 '^'
 - WinError32 파일이 사용 중일 때 발생. 개발이 아니더라도 윈도우에서 빈번하게 뜨는 에러
 - ...**껐다 키는 것 말고**는 아직까지도 해결책을 못 찾았다!



- 로컬에서 GPU 환경을 가지고 있다면 tensorflow가 GPU를 사용할 수 있도록 연결해야 한다. (모듈 설치가 끝이 아님)
- 분명 PyTorch GPU 환경을 맞추면 자연스럽게 Tensorflow도 돌아갈 줄 알았는데 전혀 안 돌아가더라고요 ^~^?
 - 사실 인공지능학과에서 우스갯소리로 입학시험에 PyTorch + GPU 환경 설치하는 걸 내야한다고들…
- 일단 tensorflow를 설치할 때 tensorflow-gpu 로 설치해주셔야 합니다.
 - Tensorflow로 설치해도 GPU를 사용할 수 있는 것으로 Documentation에 작성되어 있는데 그래도 tensorflow-gpu로 설치합니다… 무엇이든 확실하게
- 그 다음에는 GPU 환경 설정.
 - 타사GPU로 돌려본 적이 없어서 잘 모르겠지만 대부분 NVIDIA사의 GPU를 사용
 - 순차적으로
 - CUDA Driver
 - cuDNN

```
tf.config.list_physical_devices("GPU")  
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

진짜 이거 보려고 몇 시간을 …

두 가지를 설치해주시고 컴퓨터를 껐다 켰다.

보유하고 있는 GPU에 따라 설치 해야 할 버전이 다르다. Nvidia 홈페이지를 참고.

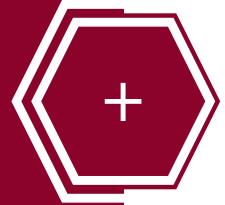


Thank you 🙏
Questions ?

Daehyun Cho
Cogsys Lab, AI Dept
1phantasmas@korea.ac.kr

<https://github.com/1pha/neural-style-transfer>





Backups

- ?
- Who are you
- ?
- Reference⋯⋯ 원래 레퍼 체크 엄청 열심히 하는데 정리할 시간이 없어서 다른 걸 준비해봤습니다.



다른 Framework는 어떤가요?

- 사실 Tensorflow는 연구계에서 빛을 많이 잃었습니다.
- PyTorch가 연구에서 많은 부분을 가져갔습니다.
- 근데 왜 Tensorflow 해요?
 - 여러분들은 연구 안하자나요 ...
 - TF가 개발에서 지금 잘 나가는 이유 중 하나는 **배포에 용이한 아주 많은 라이브러리를 제공하기 때문**
 - TFLite, tf.js, tf recommenders, tf federated, ...
 - <https://www.tensorflow.org/resources/libraries-extensions?hl=ko>

컴퓨터 비전 모델은 뭐가 있나요?

- 딥러닝 강의... 제가 맛 좀 많이 봐 본 편인데요...
 - LeNet, AlexNet, VGG, ResNet, Inception ... 어딜 가도 이거부터 소개하는데 **사실상 이제 할아버지**가 되어버림
 - *(놀랍게도 ResNet은 2016년에 발표한건데 ... 여기가 이렇게 빠릅니다.)
 - 새로운 거 없나요 좀 자극적인거
- 당연히 많습니다. 대표적인 거 몇 가지를 간단하게 흥미 위주로만 소개해보면



MobileNet

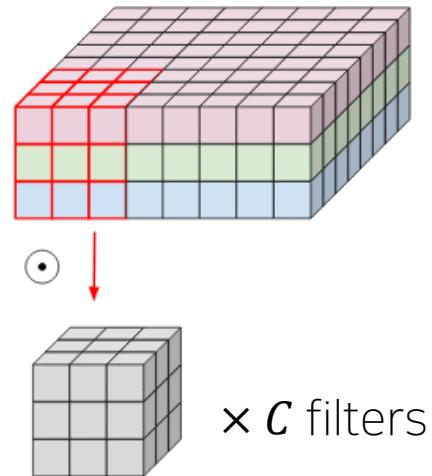
Backups

4

- Convolution 연산 개선을 했다.
- 얘도 좀 계보가 있는데 현재 MobileNet.V3까지 나온 상태이다.
- 이 연산은 비전에서 꽤나 중요한 연산 중 하나
- $K=3$ 의 경우 대략 9배의 연산 개선

- ✓ K =Kernel size
- ✓ C =input channel
- ✓ M =output channel

$$C(K^2M)$$



$$C(K^2 \times M)$$

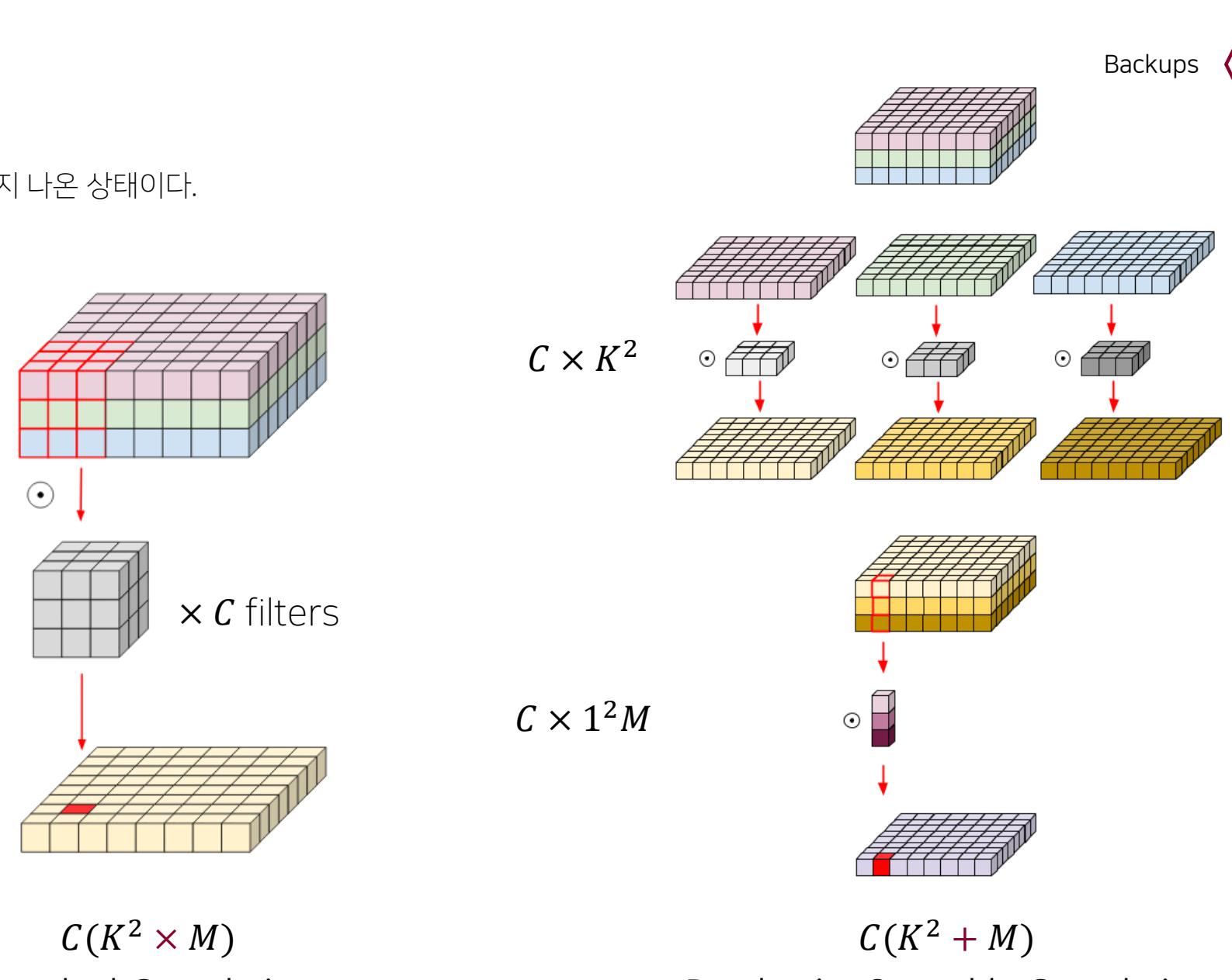
Standard Convolution

$$C \times K^2$$

$$C \times 1^2 M$$

$$C(K^2 + M)$$

Depth-wise Separable Convolution



Transformer (Attention)

- 자연어 처리에서 혜성 같이 등장한 구조.
- 보통 자연어는 아래 그림처럼 순차적으로 단어를 처리하는 구조를 가지는데 이는 문제가 많다.
 - 문장이 길어지면 안 끝난다.
 - 미분이 앞으로 안 가서 업데이트가 안된다.
 - 어떻게든 여러 테크닉으로 해결해보려 했지만 구조를 아예 새로 들고 나온 Transformer의 압승
- 이 모델은 **단어 사이의 연관성을 멀리서도 잡아낼 수 있도록** 설계되었다.
- 아니 근데 왜 갑자기 자연어 모델을 ...
 - 이걸 비전에서도 쓸 수 있을 거라 생각한 Google은 2020년 말에 엄청난 논문을 들고 온다.

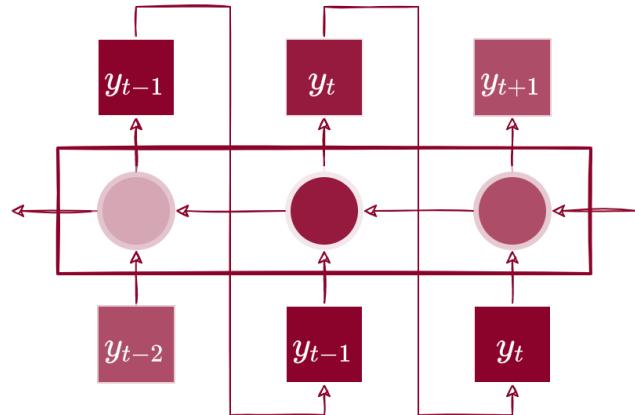


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

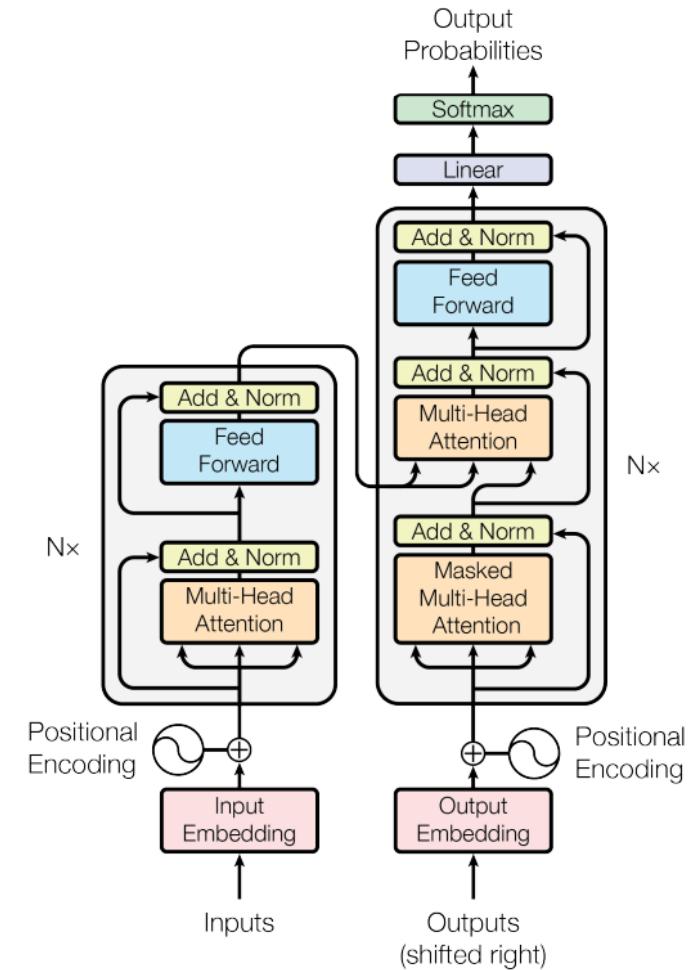
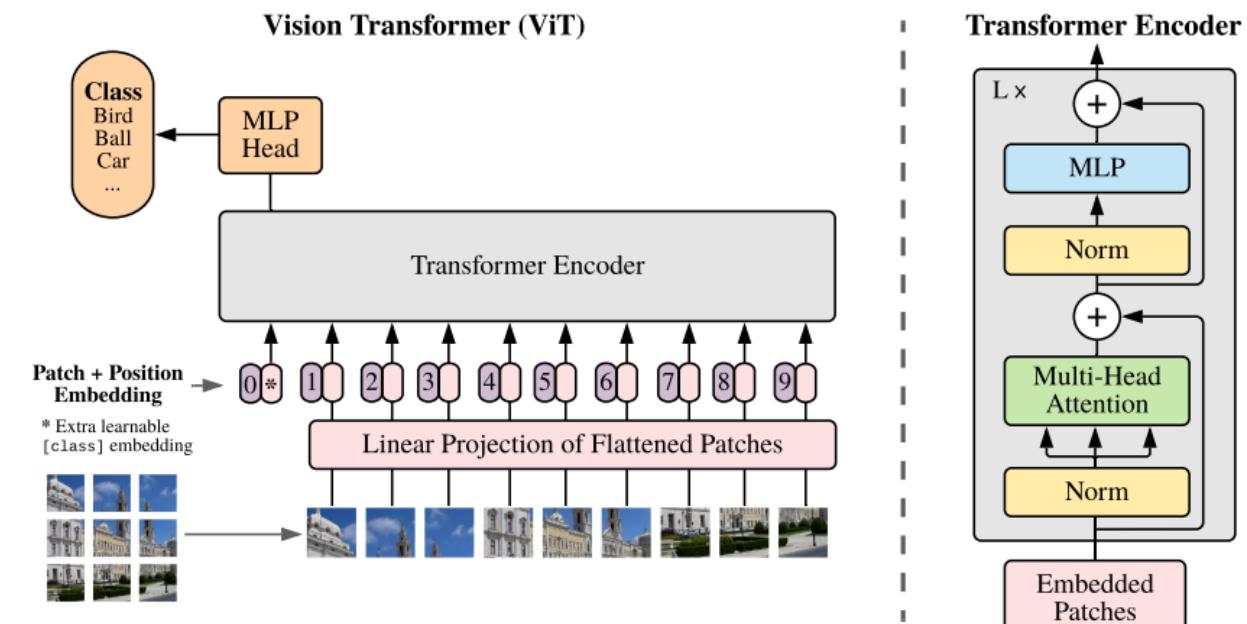


Figure 1: The Transformer - model architecture.

- 트랜스포머는 기본적으로 단어 사이의 연관성을 멀리서도 확인할 수 있도록 짜여진 구조다.
 - 덕분에 데이터는 ~~꽤~~ 많이 필요하지만 성능이 정말 많이 향상되었다.
- 이걸 컴퓨터 비전에서도 쓸 수 있지 않을까? 하는 생각에 Google이 지펴버린 ViT 대란
 - 단어 사이의 연관성을 동시에 볼 수 있도록 설계했다고 했다.
 - 이 단어 자리에 픽셀을 넣으면 어떨까?하는 미쳐버린 생각
 - 보통 자연어에서 문장 길이를 512 정도로 상정하는데 ...
이걸 이미지에서 한다고 생각하면 대표적인 ImageNet 데이터가
224 * 224니까... 어우
 - 그래서 이미지를 쪼개서 패치를 하나의 단어로 가정했다 (천잰가)
- 사실 이 논문은 성능 최고야! 하고 나온 건 아니고
비전에서도 트랜스포머를 적용할 수 있다하고 주장
- 이 뒤로... Vision Transformer는 아주 많이 등장했다고 한다.



✓ Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).

- No convolutional layer can achieve competitive results with convolutional neural networks (e.g. ResNet)
- Trained with distillation process using CNN foundation models as teacher network.
- Use “distillation token” which learns teacher label, while class token is used to train true label.
 - Cosine-similarity between class token and distillation token embedding is high (=0.93), which is not identical.

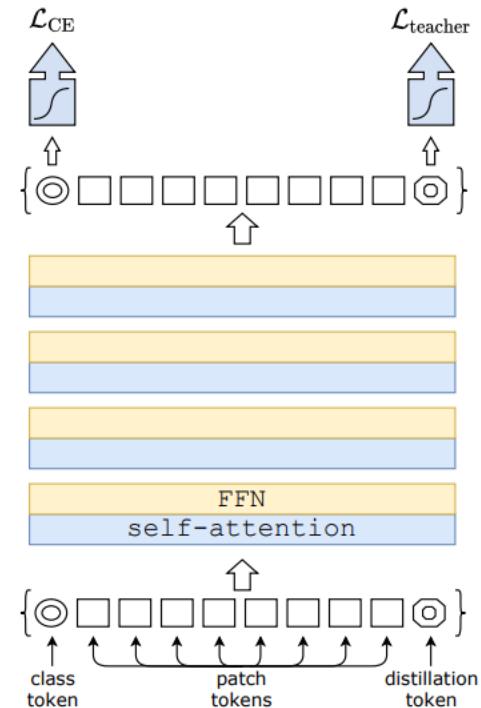


Figure 2: Our distillation procedure: we simply include a new *distillation token*. It interacts with the class and patch tokens through the self-attention layers. This distillation token is employed in a similar fashion as the class token, except that on output of the network its objective is to reproduce the (hard) label predicted by the teacher, instead of true label. Both the class and distillation tokens input to the transformers are learned by back-propagation.

✓ Touvron, Hugo, et al. "Training data-efficient image transformers & distillation through attention." International Conference on Machine Learning. PMLR, 2021.

T2T: Tokens-to-Token (ICCV 2021)

- To have richer meaningful feature extraction compared to naïve ViT and to have lighter computational burden, this model has special trick on tokens.
 - Used Performer-style* self-attention.
 - Concatenate tokens in overlapping scheme, which will reduce image size over the layer
- After embedding image in T2T, use well-known backbone models such as ViT and other CNN-based foundation models as well.

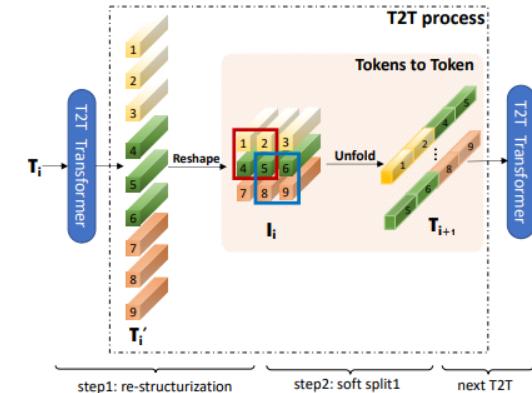


Figure 3. Illustration of T2T process. The tokens T_i are re-structured as an image I_i after transformation and reshaping; then I_i is split with overlapping to tokens T_{i+1} again. Specifically, as shown in the pink panel, the four tokens (1,2,4,5) of the input I_i are concatenated to form one token in T_{i+1} . The T2T transformer can be a normal Transformer layer [37] or other efficient transformers like Performer layer [34] at limited GPU memory.

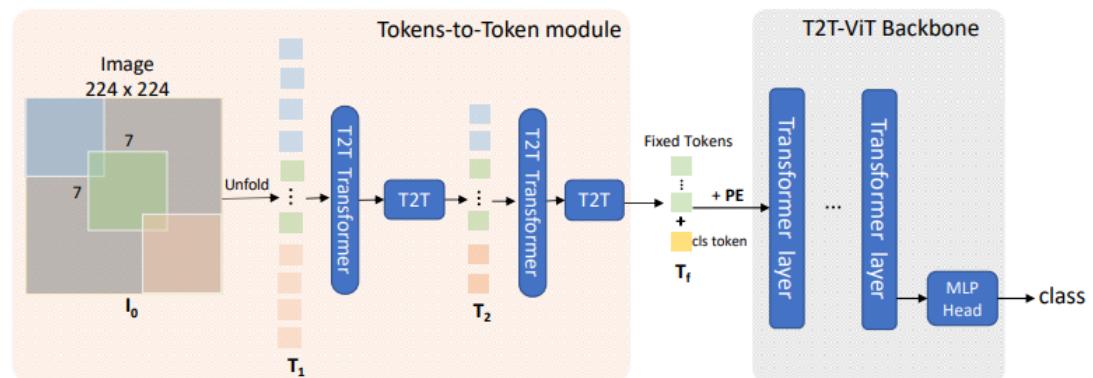


Figure 4. The overall network architecture of T2T-ViT. In the T2T module, the input image is first soft split as patches, and then unfolded as a sequence of tokens T_0 . The length of tokens is reduced progressively in the T2T module (we use two iterations here and output T_f). Then the T2T-ViT backbone takes the fixed tokens as input and outputs the predictions. The two T2T blocks are the same as Fig. 3 and PE is Position Embedding.

- ✓ Yuan, Li, et al. "Tokens-to-token vit: Training vision transformers from scratch on imagenet." arXiv preprint arXiv:2101.11986 (2021).
- ✓ Choromanski, Krzysztof, et al. "Rethinking attention with performers." arXiv preprint arXiv:2009.14794 (2020).

CPVT: Conditional Positional Encodings (Feb 2021)

- Using a fixed positional encoding independent of the input lose translational invariance of the model.
- Here, they proposed a novel **conditional positional encoding scheme (CPE)** and dynamically generates PE.
- Tokens from preceding layer will be reshaped as an image, then forwarded to depth-wise separable convolution to earn position encodings.
- To have translation invariance in the model, **CPVT-GAP** was proposed, which is a model that literally has **Global Average Pooling** layer in the last layer.

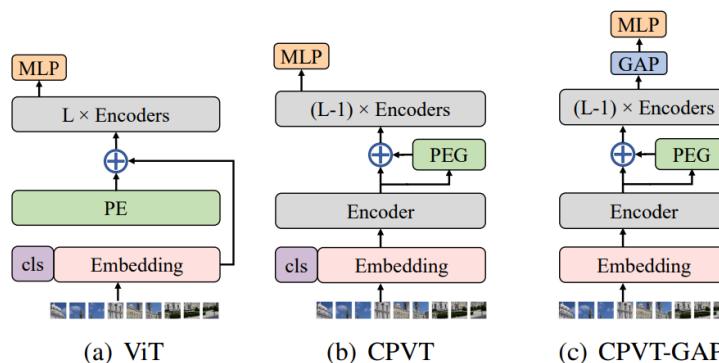


Figure 1. Vision Transformers: (a) ViT [10] with explicit 1D learnable positional encodings (PE) (b) CPVT with conditional positional encoding from the proposed Position Encoding Generator (PEG) plugin, which is the *default* choice. (c) CPVT-GAP without class token (cls), but with global average pooling (GAP) over all items in the sequence. Note that GAP is a bonus version which has boosted performance.

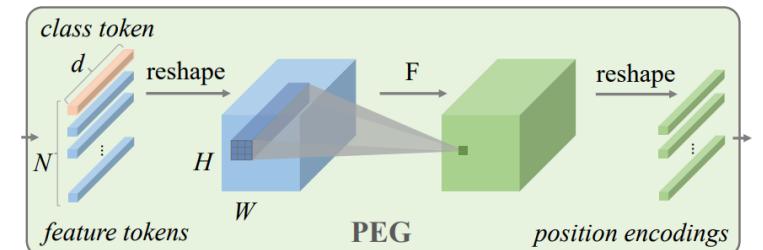


Figure 2. Schematic illustration of Positional Encoding Generator (PEG). Note d is the embedding size, N is the number of tokens. The function \mathcal{F} can be depth-wise, separable convolution or other complicated blocks.

✓ Chu, Xiangxiang, et al. "Conditional positional encodings for vision transformers." arXiv preprint arXiv:2102.10882 (2021).

PvT: Pyramid Vision Transformer (Feb 2021)

- Stemming from CNNs downscaling the input layer by layer, this work tried to shrink image size (literally “pyramid”)
- Through pyramid structure, there are multiscale feature maps that can be used.
 - This enables PVT to do higher resolution task – such as Segmentation or Detection.
 - Compare this to ViT, which has the same dimension feature map, but keep coarsened every layer.
- By downscaling the image, computation burden was reduced.

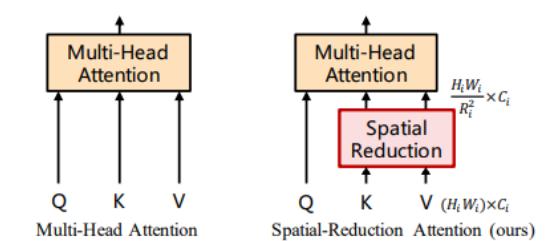
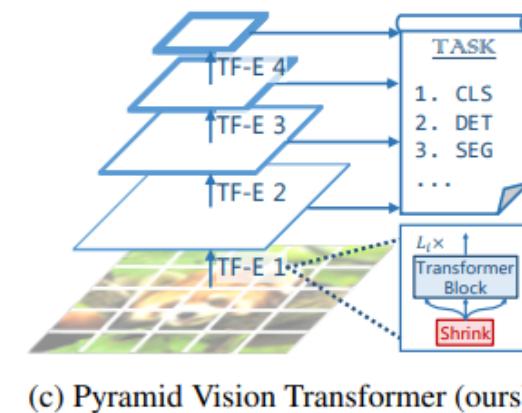
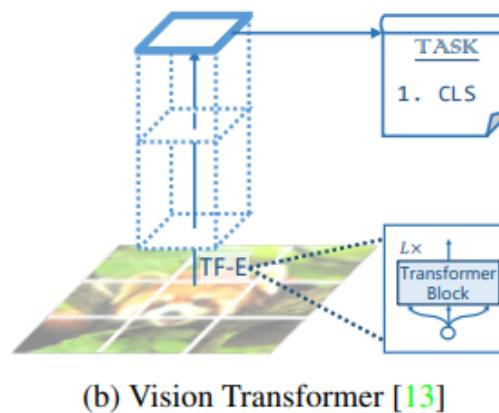
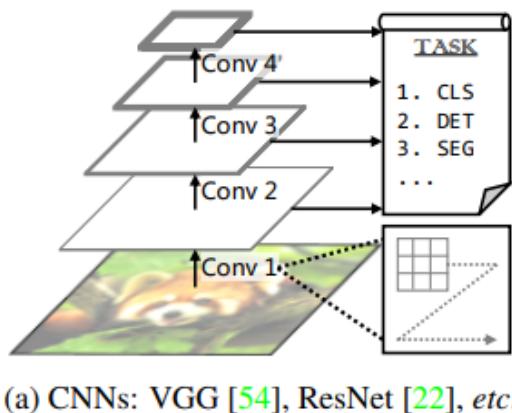


Figure 4: **Multi-head attention (MHA) vs. spatial-reduction attention (SRA).** With the spatial-reduction operation, the computational/memory cost of our SRA is much lower than that of MHA.

✓ Wang, Wenhui, et al. "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions." arXiv preprint arXiv:2102.12122 (2021).

ConViT (Facebook AI, March 2021)

- Introduces “gated positional self-attention (GPSA)”.
 - This allows a form of positional self-attention acts like convolution in the early layer.
 - Later stages attention work in global way, just like ViT
- Attention maps shows more diversity compared to previous work.

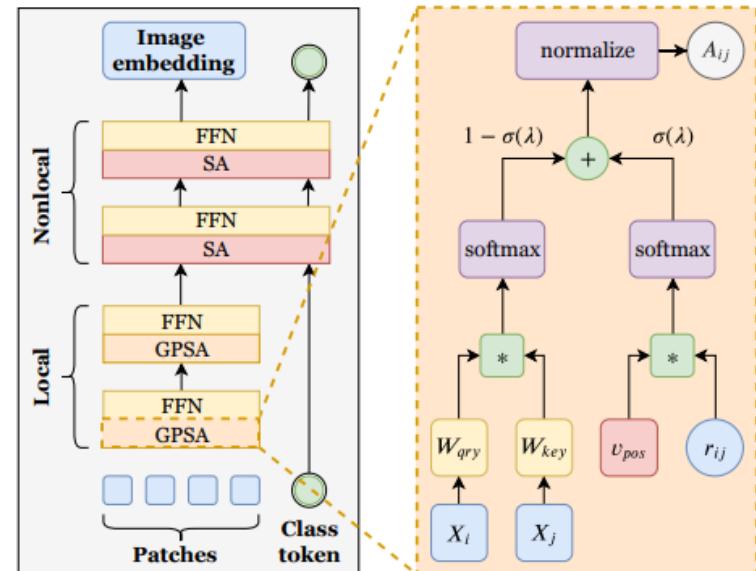
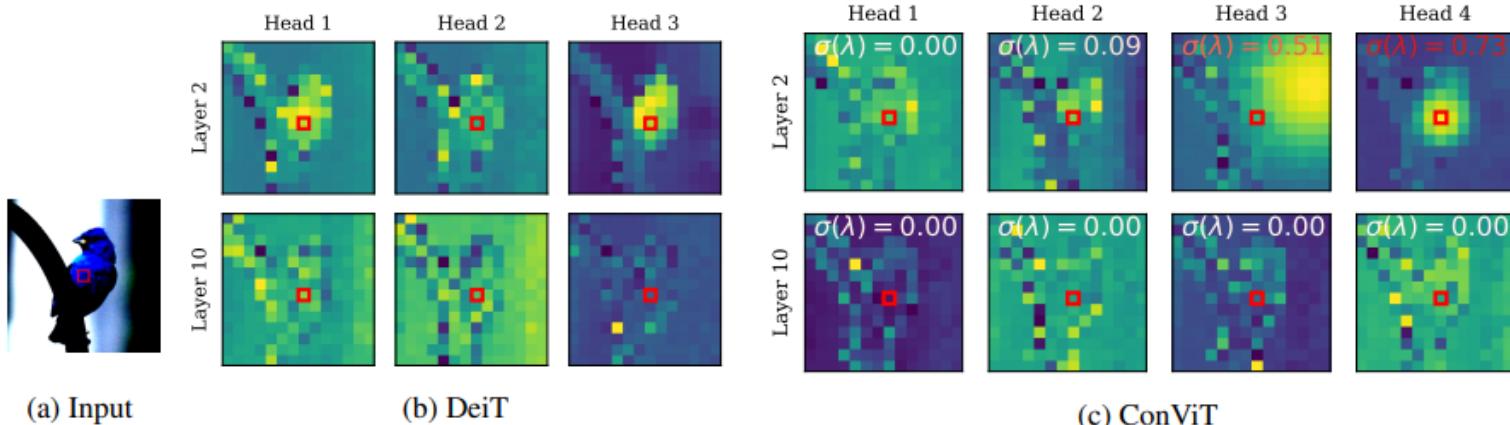


Figure 4. Architecture of the ConViT. The ConViT (left) is a version of the ViT in which some of the self-attention (SA) layers are replaced with gated positional self-attention layers (GPSA; right). Because GPSA layers involve positional information, the class token is concatenated with hidden representation after the last GPSA layer. In this paper, we typically take 10 GPSA layers followed by 2 vanilla SA layers. FFN: feedforward network (2 linear layers separated by a GeLU activation); W_{qry} : query weights; W_{key} : key weights; v_{pos} : attention center and span embeddings (learned); r_{qk} : relative position encodings (fixed); λ : gating parameter (learned); σ : sigmoid function.

✓ d'Ascoli, Stéphane, et al. "Convit: Improving vision transformers with soft convolutional inductive biases." PMLR 139, 2021 arXiv:2103.10697 (2021).

TNT: Transformer in Transformer (NeurIPS 2021)

- Idea is that patch embedding is making a “visual sentence” from a whole image, and then split patches into a smaller sub-patch, thinking these as a “visual word”
- Position encoding is added in both sentence-level and word-level, respectively.
- Far from expectation, computational burden is not high
 - 1.14 \times in FLOPs, and 1.08 \times in #params.
- The work insists TNT learns more diverse than DeiT.

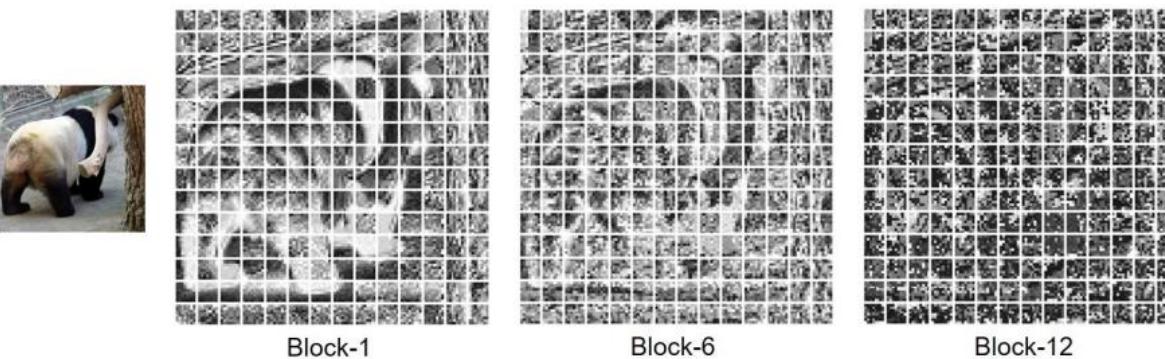


Figure 4: Visualization of the averaged word embeddings of TNT-S.

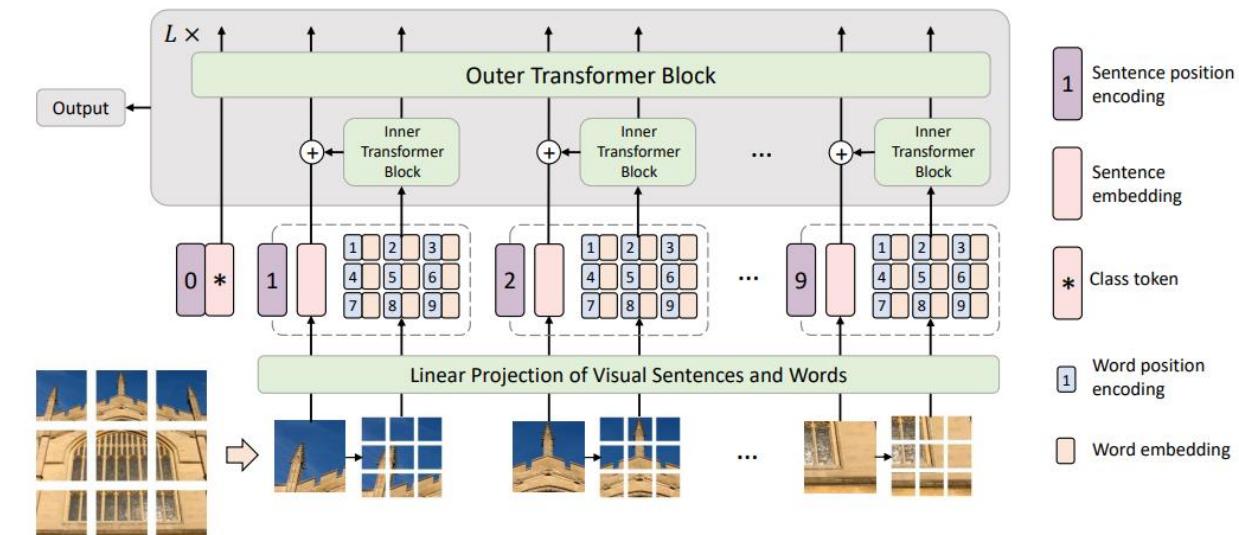


Figure 1: Illustration of the proposed Transformer-iN-Transformer (TNT) framework. The inner transformer block is shared in the same layer. The word position encodings are shared across visual sentences.

✓ Han, Kai, et al. "Transformer in transformer." arXiv preprint arXiv:2103.00112 (2021).

- Optimal combination of Convolution and Attention.
 - Trying to unify depth-wise convolution with self-attention.
 - Top-1 90.88% accuracy in JFT300M
- To make the model feasible, they have tested with following options
 - Down-sampling to reduce spatial size (less burden on attention calculation)
 - Enforce local attention
 - Quadratic softmax attention to linear attention

$$y_i = \sum_{j \in \mathcal{L}(i)} w_{i-j} \odot x_j \quad (\text{depthwise convolution}),$$

$$y_i = \sum_{j \in \mathcal{G}} \underbrace{\frac{\exp(x_i^\top x_j)}{\sum_{k \in \mathcal{G}} \exp(x_i^\top x_k)}}_{A_{i,j}} x_j \quad (\text{self-attention}),$$



$$y_i^{\text{pre}} = \sum_{j \in \mathcal{G}} \frac{\exp(x_i^\top x_j + w_{i-j})}{\sum_{k \in \mathcal{G}} \exp(x_i^\top x_k + w_{i-k})} x_j. \quad (3)$$

✓ Dai, Zihang, et al. "CoAtNet: Marrying Convolution and Attention for All Data Sizes." arXiv preprint arXiv:2106.04803 (2021).

Table 1: Desirable properties found in convolution or self-attention.

Properties	Convolution	Self-Attention
Translation Equivariance	✓	
Input-adaptive Weighting		✓
Global Receptive Field		✓

MobileViT (Apple, Oct 2021)

- Aims to make a light-weight+low latency+general ViT
- Mixture usage of standard & mobile convolution
- Calculation complexity is “in theory” larger than ViTs but in practice, MobileViT is more efficient.
- #params is extremely small compared to current ConvNets

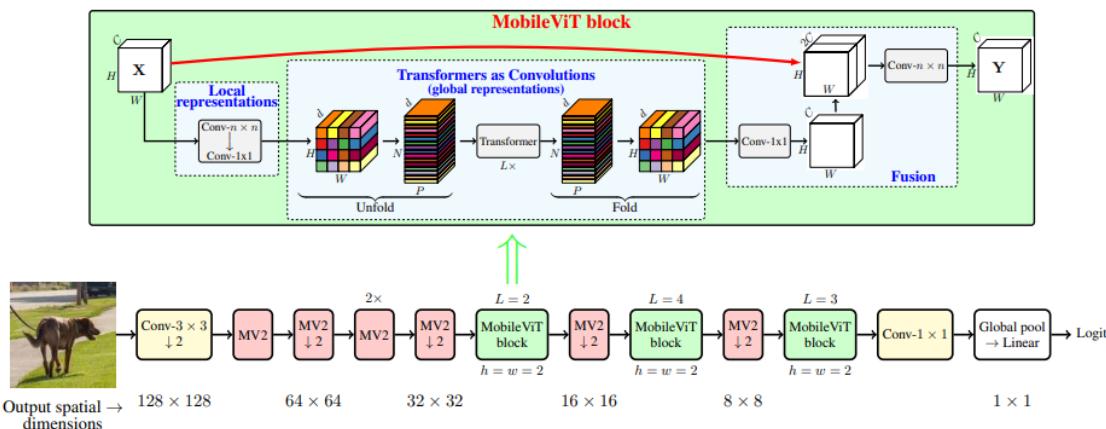


Figure 1: Visual transformers vs. MobileViT

- ✓ Mehta, Sachin, and Mohammad Rastegari. "MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer." arXiv preprint arXiv:2110.02178 (2021).

Model	# Params. ↓	Top-1 ↑
MobileNetv1	2.6 M	68.4
MobileNetv2	2.6 M	69.8
MobileNetv3	2.5 M	67.4
ShuffleNetv2	2.3 M	69.4
ESPNetv2	2.3 M	69.2
MobileViT-XS (Ours)	2.3 M	74.8

(b) Comparison with light-weight CNNs (similar parameters)

Model	# Params. ↓	Top-1 ↑
DenseNet-169	14 M	76.2
EfficientNet-B0	5.3 M	76.3
ResNet-101	44.5 M	77.4
ResNet-101-SE	49.3 M	77.6
MobileViT-S (Ours)	5.6 M	78.4

(c) Comparison with heavy-weight CNNs

Model	# Params. ↓	Time ↓	Top-1 ↑
MobileNetv2 [†]	3.5 M	0.92 ms	73.3
DeiT	5.7 M	10.99 ms	72.2
PiT	4.9 M	10.56 ms	73.0
MobileViT (Ours)	2.3 M	7.28 ms	74.8

Table 3: ViTs are slower than CNNs.

[†]Results with multi-scale sampler (§B).

다른 인공지능 분야는 뭐가 있나요?

Backups

4

- 잠깐 소개한 자연어 분야도 굉장히 핫하다.

- 질의 응답
- 문서 요약
- 대화 생성 (챗봇)
- 여러분 자리를 대체할 코드 짜주는 기계까지
 - 느슨한 개발계에 긴장감을 ...

이거 되는 코인 같은데 뭐부터 해야 하나요?

- 내거에요



- 농담이고 관심 있으신 분들은 따로 상담요청 주세요. 😊

딥러닝 소프트웨어 소스코드 생성기 '알파코드'는 무엇?

▲ 오영주 | ⓒ 입력 2022.02.08 21:55 | 댓글 0

The screenshot shows a user interface for a programming challenge titled "1553_D_ Backspace". The "Solution" section contains the following Python code:

```
// RATING: 1200
// TAGS: *special.implementation
// LANGUAGE IS python3
// CORRECT SOLUTION
// You are given two strings s and t, both consisting of lowercase English letters. You are going to type the string s character by character, from the first character to the last one.
// ...
// When typing a character, instead of pressing the button corresponding to it, you can press the "Backspace" button. It deletes the last character you have typed. If there are multiple characters in the string, they are deleted in reverse order, starting from the current string. For example, if s is "abcd" and you press Backspace instead of typing the first and the fourth characters, you will get the string "bd" (the first press of Backspace deletes no character, and the second press deletes the character 'c'). Another example, if s is "abcas" and you press Backspace instead of the last two letters, then the resulting text is "a".
// ...
// Your task is to determine whether you can obtain the string t, if you type the string s and press "Backspace" instead of typing several (maybe zero) characters of s...
// Input:
// ...
// The first line contains a single integer q, ( $1 \leq q \leq 10^5$ ) = the number of test cases.
// ...
// The first line of each test case contains the string s ( $1 \leq |s| \leq 10^5$ ). Each character of s is a lowercase English letter.
// ...
// The second line of each test case contains the string t ( $1 \leq |t| \leq 10^5$ ). Each character of t is a lowercase English letter.
// ...
int input():
    i in range(t):
        s=input()
        tinput()
        a=[]
        b=[]
        for i in s:
            a.append(i)
        for j in t:
            b.append(j)
        b.reverse()
        a.reverse()
        if len(b)>=0 and len(a)>=0:
            if a[0]==b[0]:
                a.pop(0)
                b.pop(0)
            elif a[0]>b[0] and len(a)>=1:
                a.pop(0)
            elif a[0]<b[0] and len(a)>=1:
                a.pop(0)
        if len(b)==0:
            print("YES")
        else:
            print("NO")
```

AlphaCode의 문제 파싱 및 코드 생성 프로세스의 시작화. 사진 출처: DeepMind