# Generative Adversarial Network

DMQA

# 목차



- 참고문헌, 10.5pt



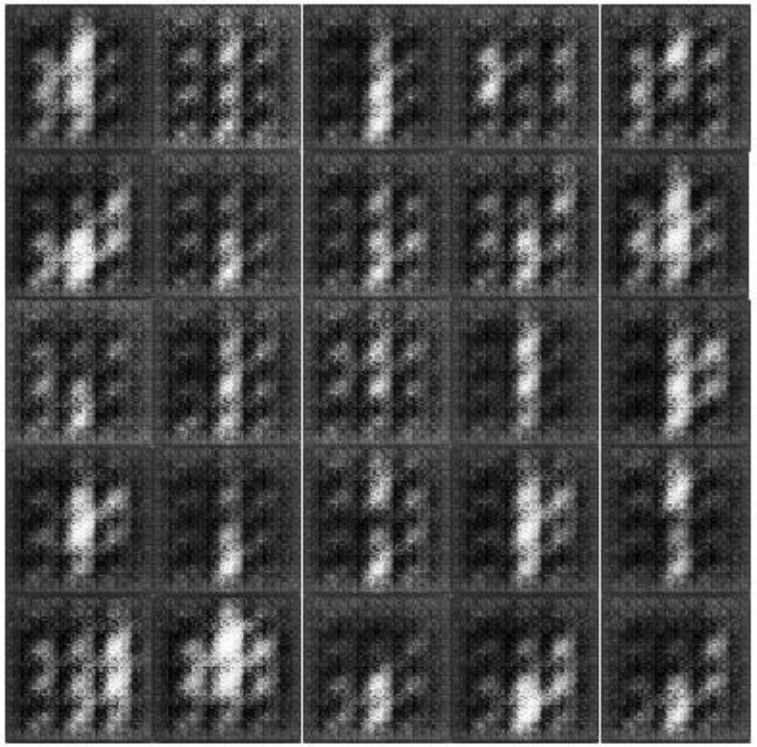


DMQA

# 1. Application of GAN

❖ GAN(Generative Adversarial Network)의 응용

- 과거에는 아래와 같이 이미지를 생성하는데 집중
- 공개되어 있는 이미지 데이터 셋이 부족한 상황



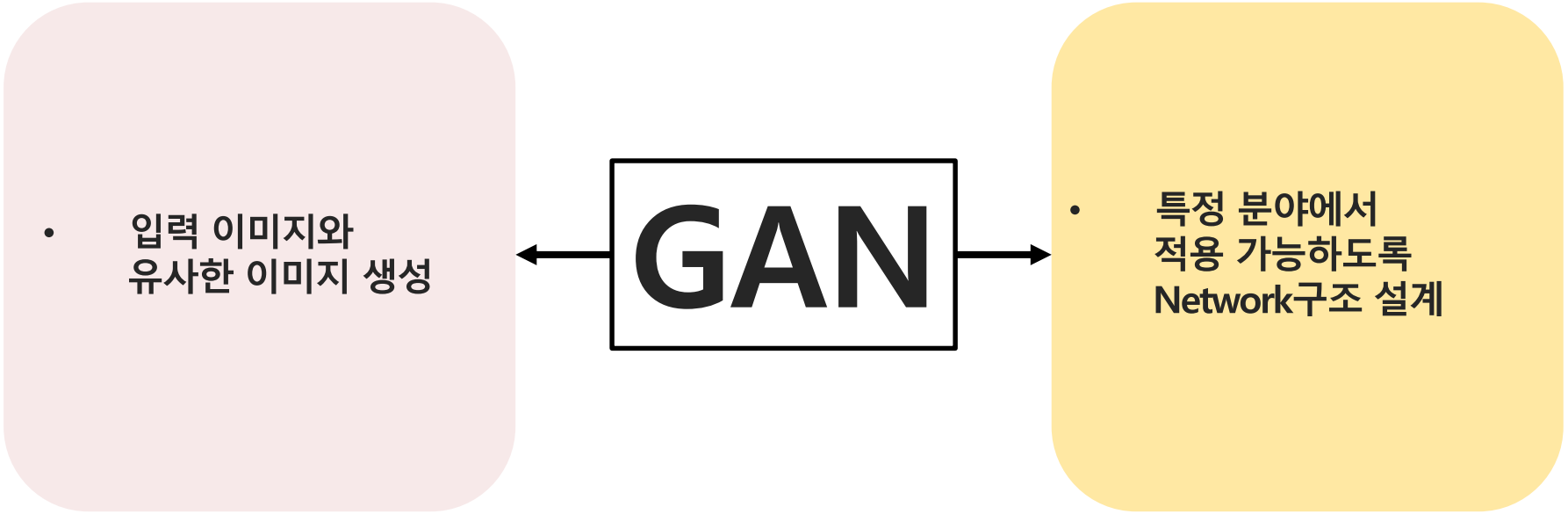
Epoch 1

- 참고문헌, 10.5pt

# 1. Application of GAN

❖ GAN(Generative Adversarial Network)의 응용

- 최근 GAN의 연구 동향은 크게 2가지로 나누어져 있음






DMQA

# 1. Application of GAN

❖ CycleGAN

- 특정 도메인에서 자주 등장하는 화풍을 실제 사진 이미지에 적용하는 연구
- 또한 비슷한 형태지만, 색깔이 다른 경우 이를 바꿔주는 연구


- 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.

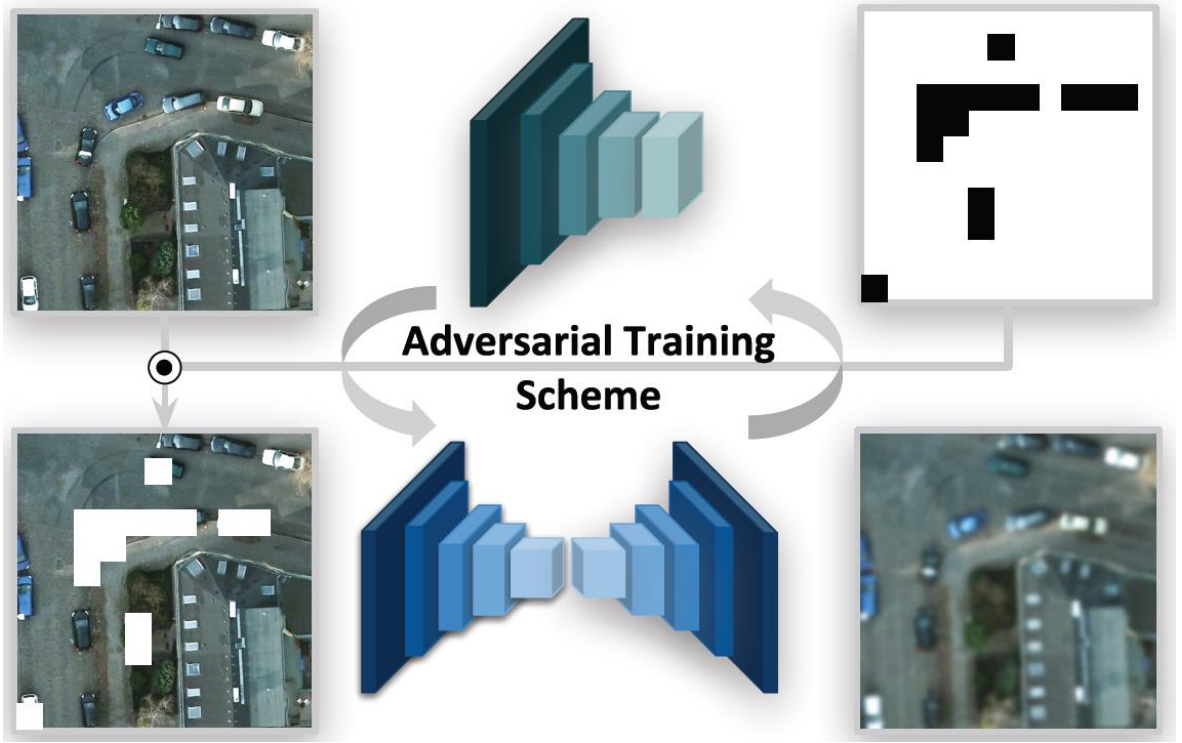


DMQA

# 1. Application of GAN

❖ 이미지 내부에 지워진 부분을 복구하는 연구

- Discriminator 와 Generator로 구성
- 두 Network를 경쟁적으로 학습시키면서 이미지에 존재하는 픽셀들의 분포를 파악




- Singh, S., Batra, A., Pang, G., Torresani, L., Basu, S., Paluri, M., & Jawahar, C. V. (2018).
- Self-Supervised Feature Learning for Semantic Segmentation of Overhead Imagery. In BMVC (p. 102).



DMQA

# 1. Application of GAN

❖ Animation 산업에서의 GAN의 적용

- 사람이 모든 만화를 그리기 어렵기 때문에, GAN을 이용해 만화 캐릭터 생성
- 이를 기반으로 만화가의 반복 작업 횟수를 줄일 수 있음


- Jin, Yanghua, et al. "Towards the automatic anime characters creation with generative adversarial networks." arXiv preprint arXiv:1708.05509 (2017).

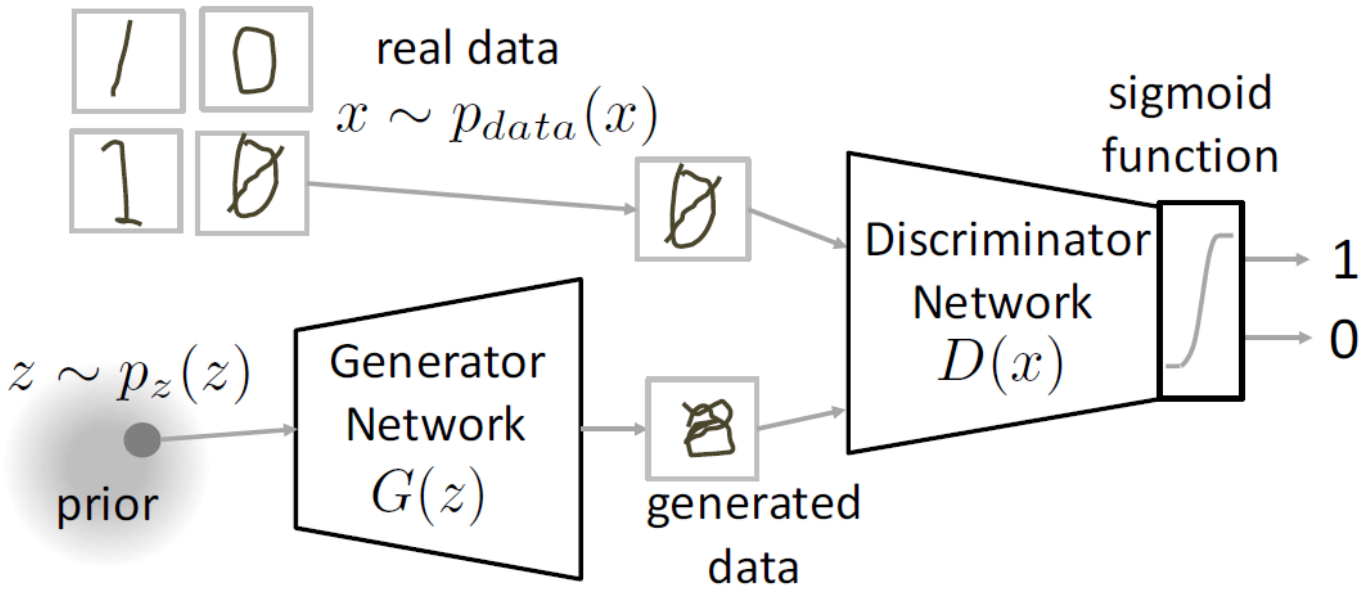


DMQA

# 2. GAN 복습

❖ GAN(Generative Adversarial Network)

- 생성 모델(Generative Model)의 새로운 패러다임을 불러온 모델
- 경쟁적 학습을 통해 Sample데이터와 매우 유사한 Sample을 생성
- 학습 과정을 통해, 인공 신경망 모형은 Sample의 분포를 학습하게 됨



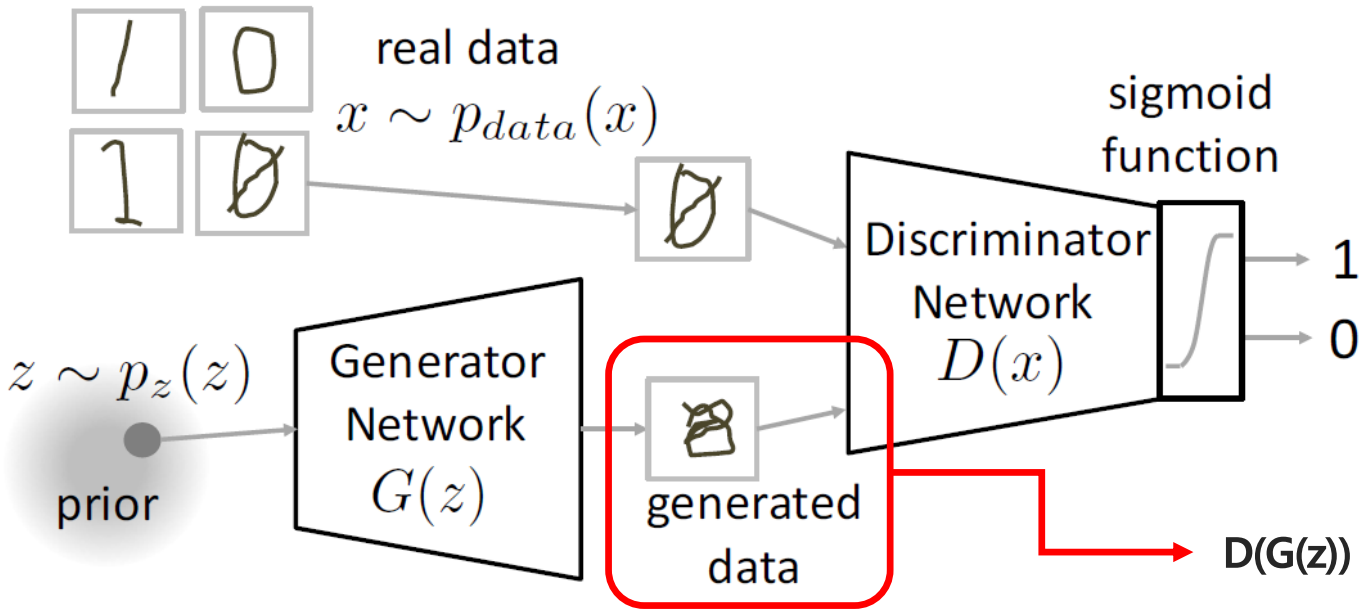



DMQA

# 2. GAN 복습

❖ GAN(Generative Adversarial Network)

- 생성 모델(Generative Model)의 새로운 패러다임을 불러온 모델
- 경쟁적 학습을 통해 Sample데이터와 매우 유사한 Sample을 생성
- 학습 과정을 통해, 인공 신경망 모형은 Sample의 분포를 학습하게 됨



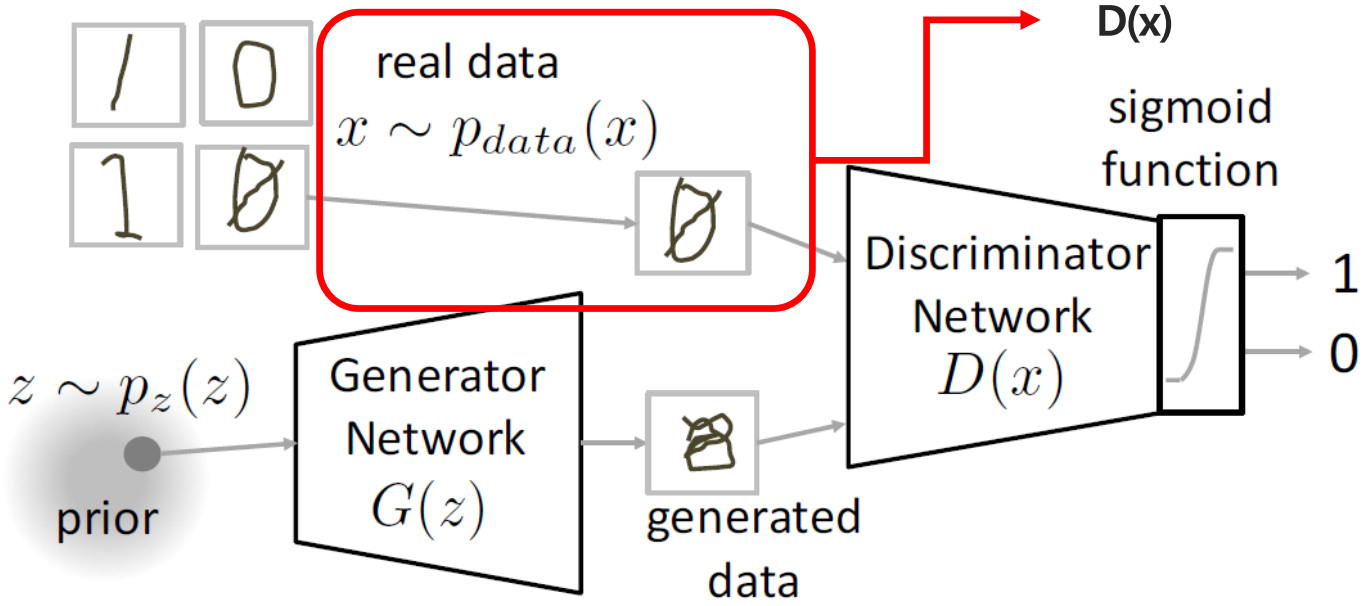



DMQA

# 2. GAN 복습

❖ GAN(Generative Adversarial Network)

- 생성 모델(Generative Model)의 새로운 패러다임을 불러온 모델
- 경쟁적 학습을 통해 Sample데이터와 매우 유사한 Sample을 생성
- 학습 과정을 통해, 인공 신경망 모형은 Sample의 분포를 학습하게 됨



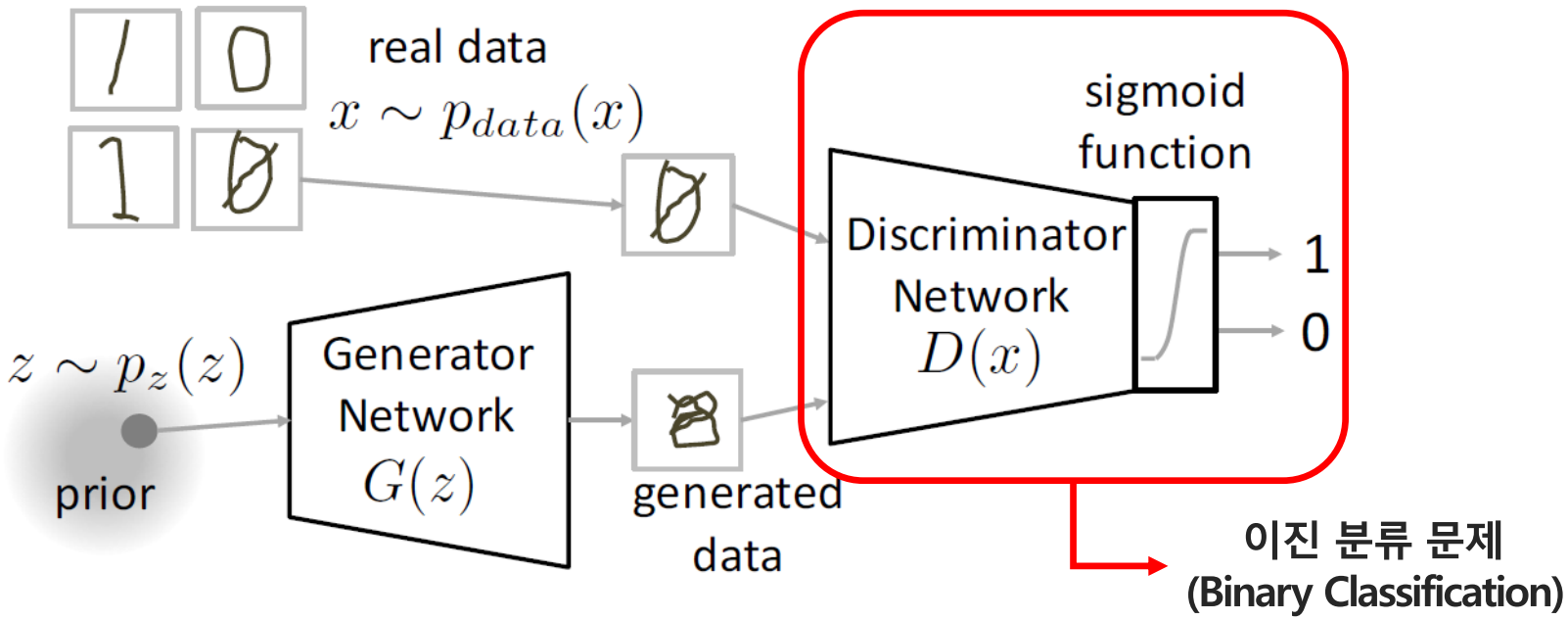



DMQA

# 2. GAN 복습

❖ GAN(Generative Adversarial Network)

- 생성 모델(Generative Model)의 새로운 패러다임을 불러온 모델
- 경쟁적 학습을 통해 Sample데이터와 매우 유사한 Sample을 생성
- 학습 과정을 통해, 인공 신경망 모형은 Sample의 분포를 학습하게 됨



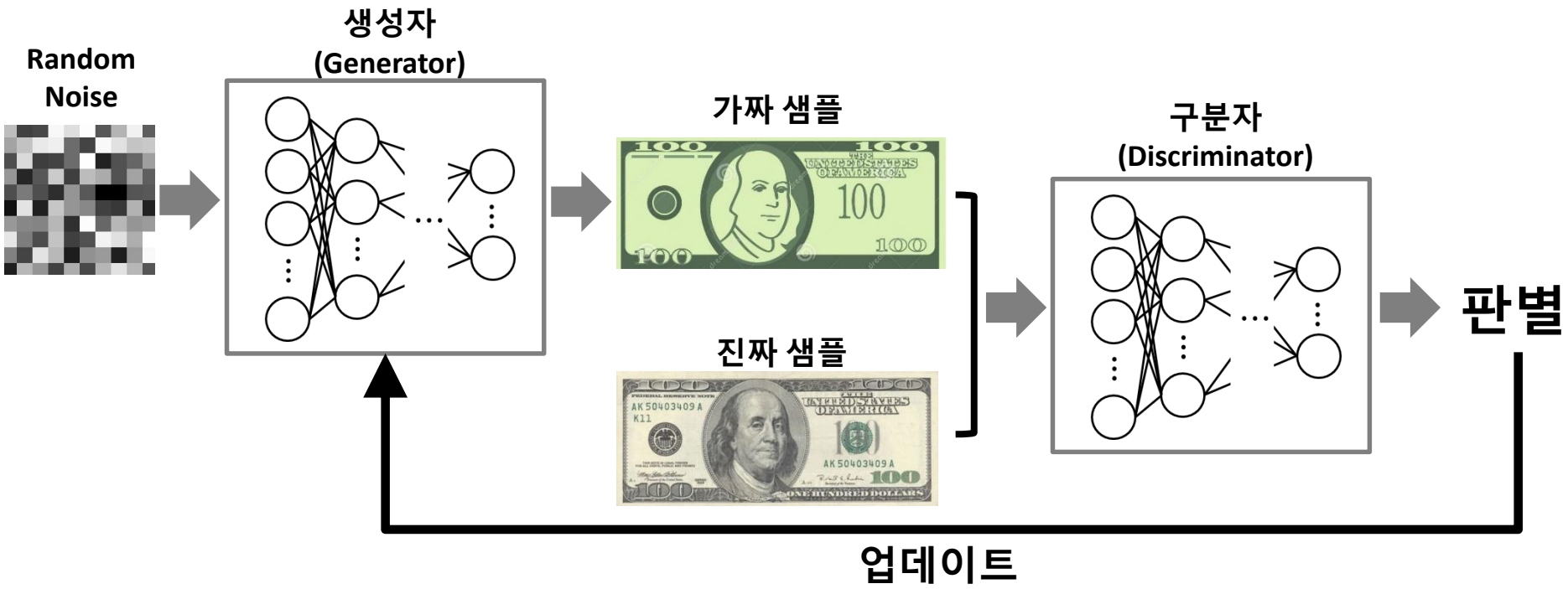



DMQA

# 2. GAN 복습

## ❖ GAN 작동 방식

1. 무작위 노이즈(Noise) 생성
2. 생성자를 통해 가짜샘플을 생성
3. 구분자를 통해 진짜/가짜 샘플을 판별
4. 반복적인 학습을 통해 생성자를 학습(진짜 샘플과 유사한 데이터를 생성하도록)



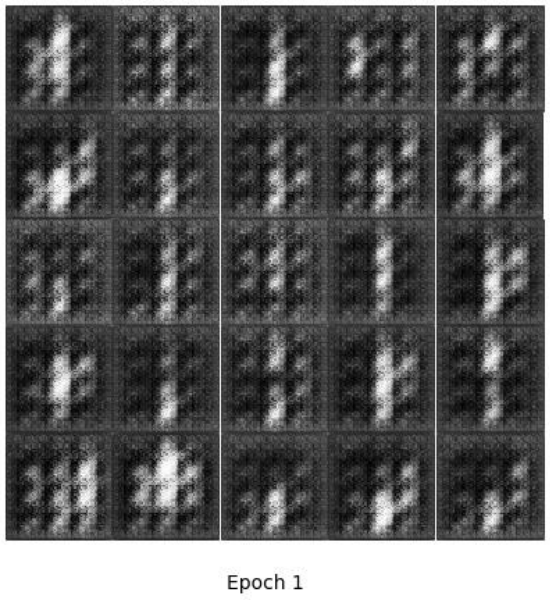


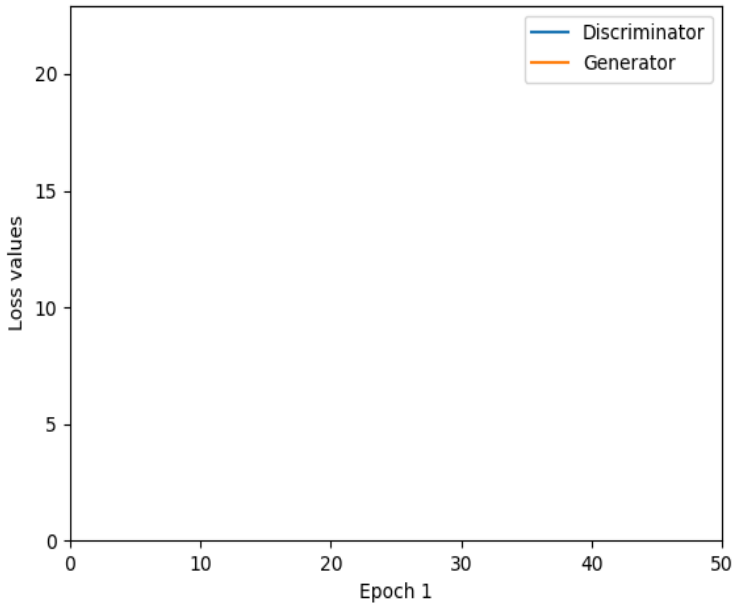

DMQA

# 2. GAN 복습

❖ GAN(Generative Adversarial Network)

- Discriminator는 실제 데이터와 생성 데이터를 잘 구별하도록 학습
- Generator는 생성 데이터가 실제 데이터와 유사하도록 생성
- 두 신경망 모형의 성능을 높이는 방향으로 학습하면서, loss 값이 수렴하는 것을 확인




- Generative Adversarial Networks(2014)
Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio



DMQA

# 3. GAN 1D Example 실습

❖ 실습

# 감사합니다.

- 참고문헌, 10.5pt

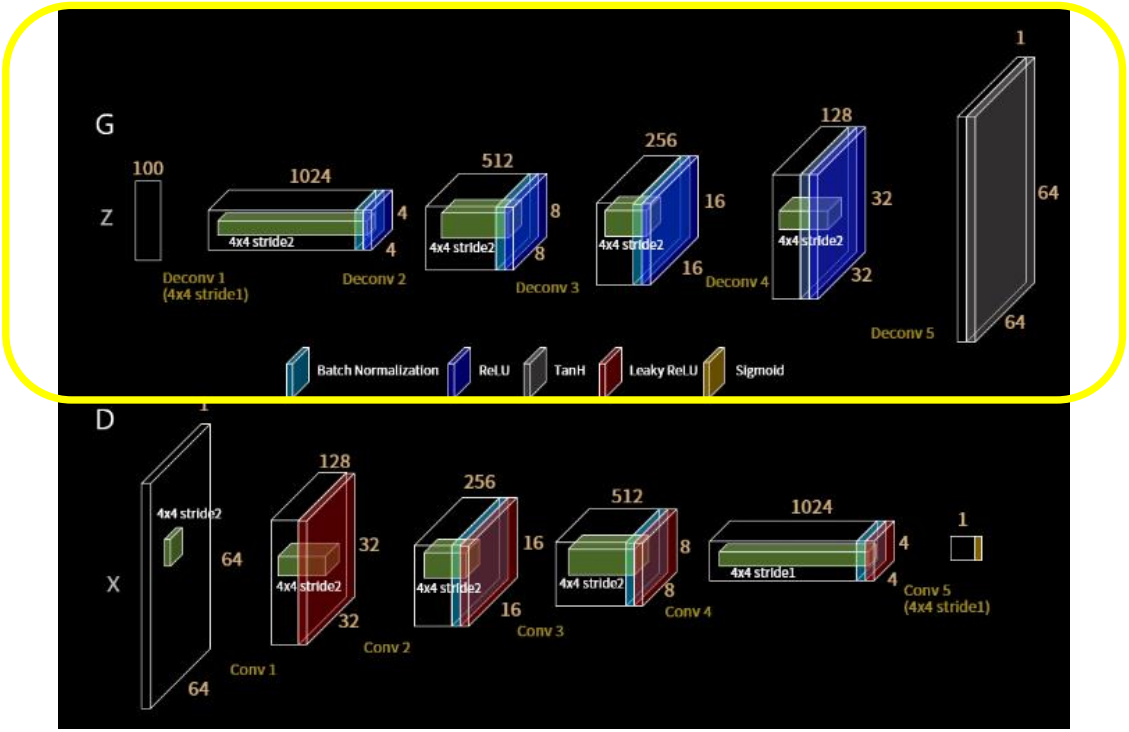
# 별첨

- 참고문헌, 10.5pt

# Convolution & Transposed convolution 연산

❖ Transposed convolution 연산

- CNN 기반의 Encoder - Decoder의 구조를 가진 Network에서 주로 사용
- Decoder를 통해 원본 사이즈로 되돌릴 때, Transposed convolution 연산을 진행




**Transposed Convolution 연산 진행**
(DCGAN)

- https://github.com/znxlwm/pytorch-MNIST-CelebA-GAN-DCGAN



DMQA

# Convolution & Transposed convolution 연산

❖ Convolution 연산 복습

- 입력 행렬과 Kernel 행렬의 원소들을 element-wise 곱의 합
- Convolution 연산을 통해 입력 행렬의 정보를 요약

| | | | |
|---|---|---|---|
| 3 | 5 | 0 | 4 |
| 6 | 7 | 1 | 2 |
| 7 | 0 | 0 | 9 |
| 9 | 8 | 8 | 1 |

입력 행렬

○

| | | |
|---|---|---|
| 1 | -1 | 1 |
| 1 | 0 | 0 |
| 1 | -1 | -1 |

Kernel 행렬

=

| | |
|---|---|
| | |
| | |

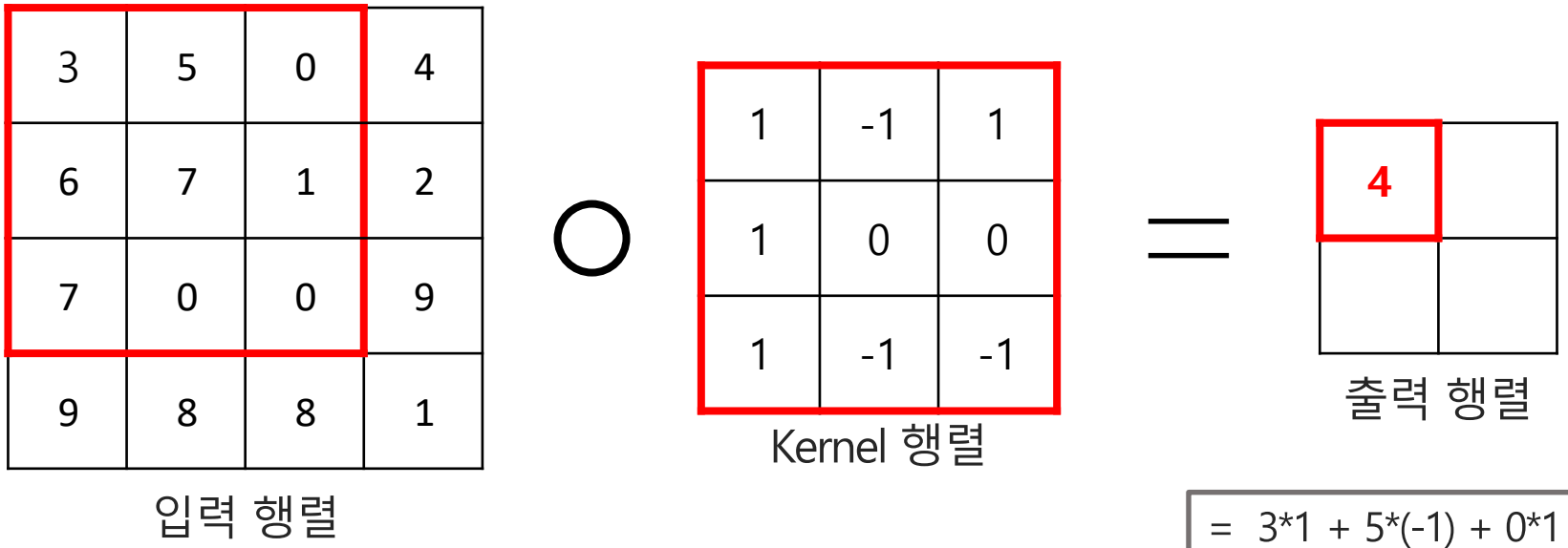
출력 행렬

- https://zzsza.github.io/data/2018/06/25/upsampling-with-transposed-convolution/

# Convolution & Transposed convolution 연산

❖ Convolution 연산 복습

- 입력 행렬과 Kernel 행렬의 원소들을 element-wise 곱의 합
- Convolution 연산을 통해 입력 행렬의 정보를 요약



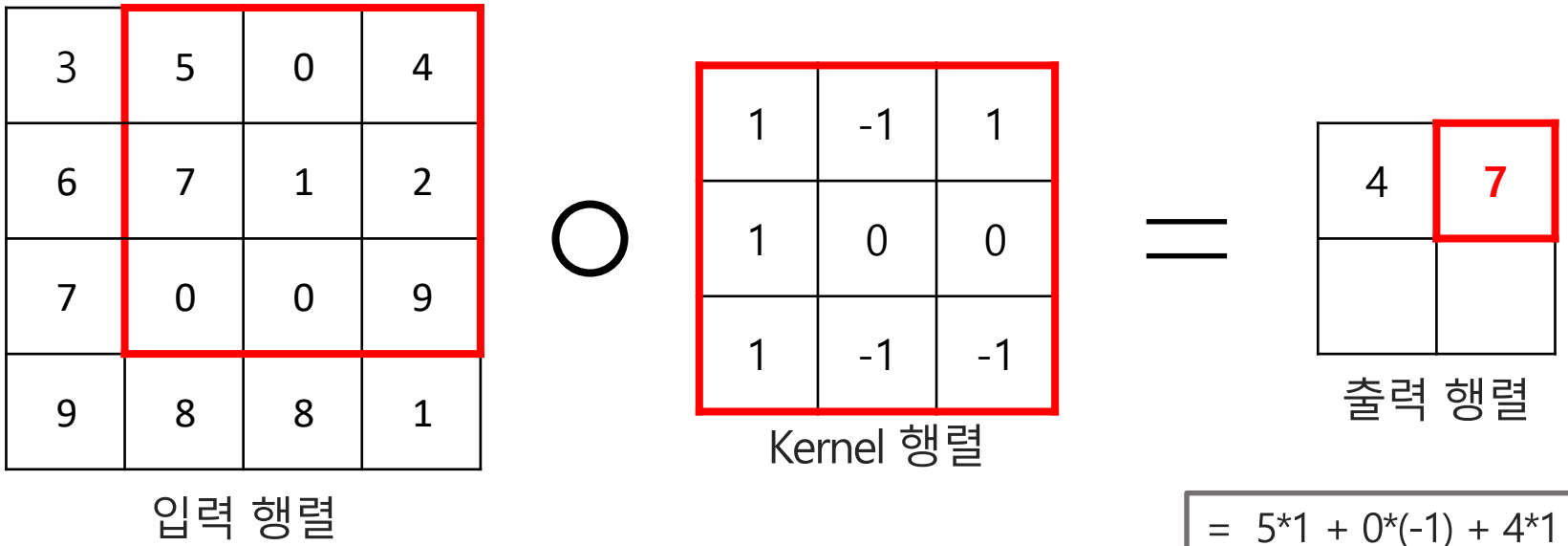

= 3*1 + 5*(-1) + 0*1
+ 6*1 + 7*0 + 1*0
+ 7*0 + 0*(-1) + 0*(-1)

- https://zzsza.github.io/data/2018/06/25/upsampling-with-transposed-convolution/

# Convolution & Transposed convolution 연산

❖ Convolution 연산 복습

- 입력 행렬과 Kernel 행렬의 원소들을 element-wise 곱의 합
- Convolution 연산을 통해 입력 행렬의 정보를 요약



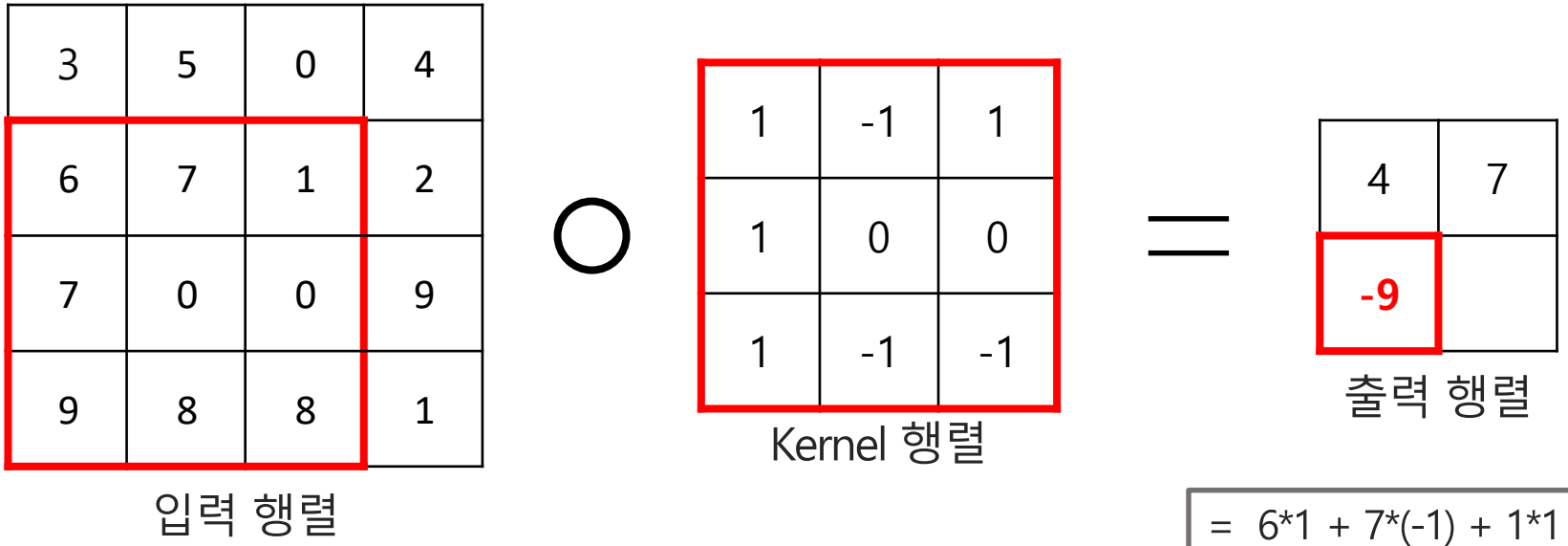

= 5*1 + 0*(-1) + 4*1
+ 7*1 + 1*0 + 2*0
+ 0*0 + 0*(-1) + 9*(-1)

- https://zzsza.github.io/data/2018/06/25/upsampling-with-transposed-convolution/

# Convolution & Transposed convolution 연산

❖ Convolution 연산 복습

- 입력 행렬과 Kernel 행렬의 원소들을 element-wise 곱의 합
- Convolution 연산을 통해 입력 행렬의 정보를 요약



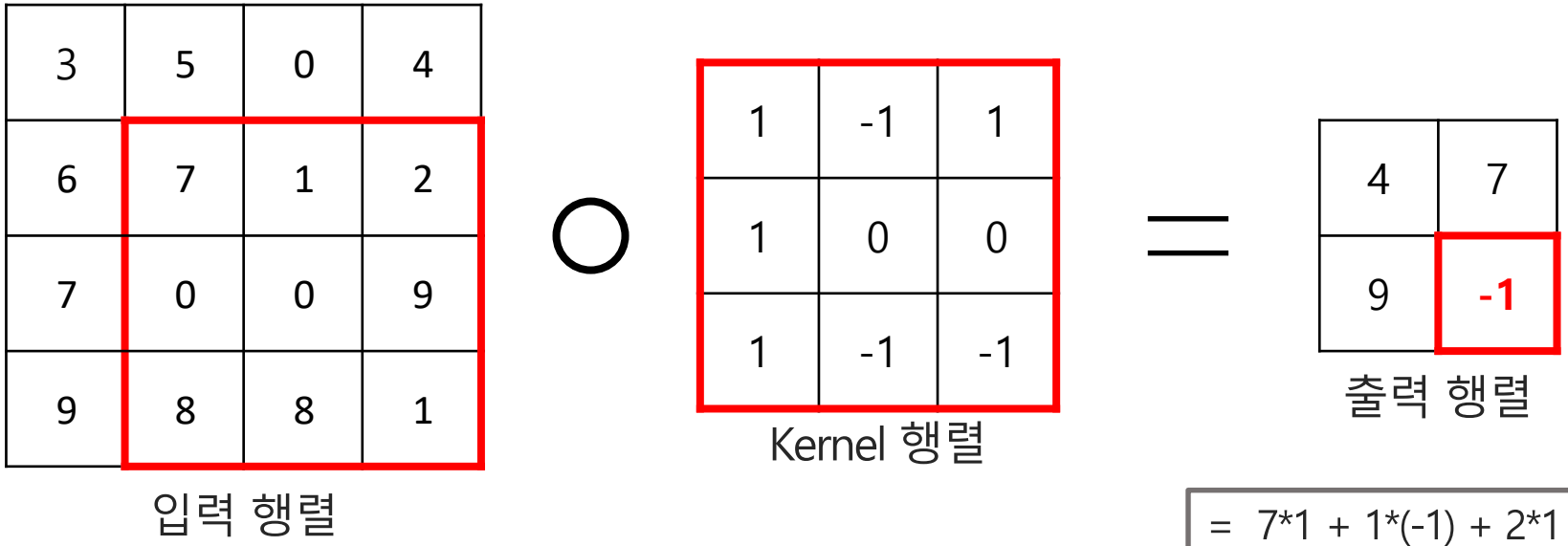

= 6*1 + 7*(-1) + 1*1
+ 7*1 + 0*0 + 0*0
+ 9*0 + 8*(-1) + 8*(-1)

- https://zzsza.github.io/data/2018/06/25/upsampling-with-transposed-convolution/

# Convolution & Transposed convolution 연산

❖ Convolution 연산 복습

- 입력 행렬과 Kernel 행렬의 원소들을 element-wise 곱의 합
- Convolution 연산을 통해 입력 행렬의 정보를 요약



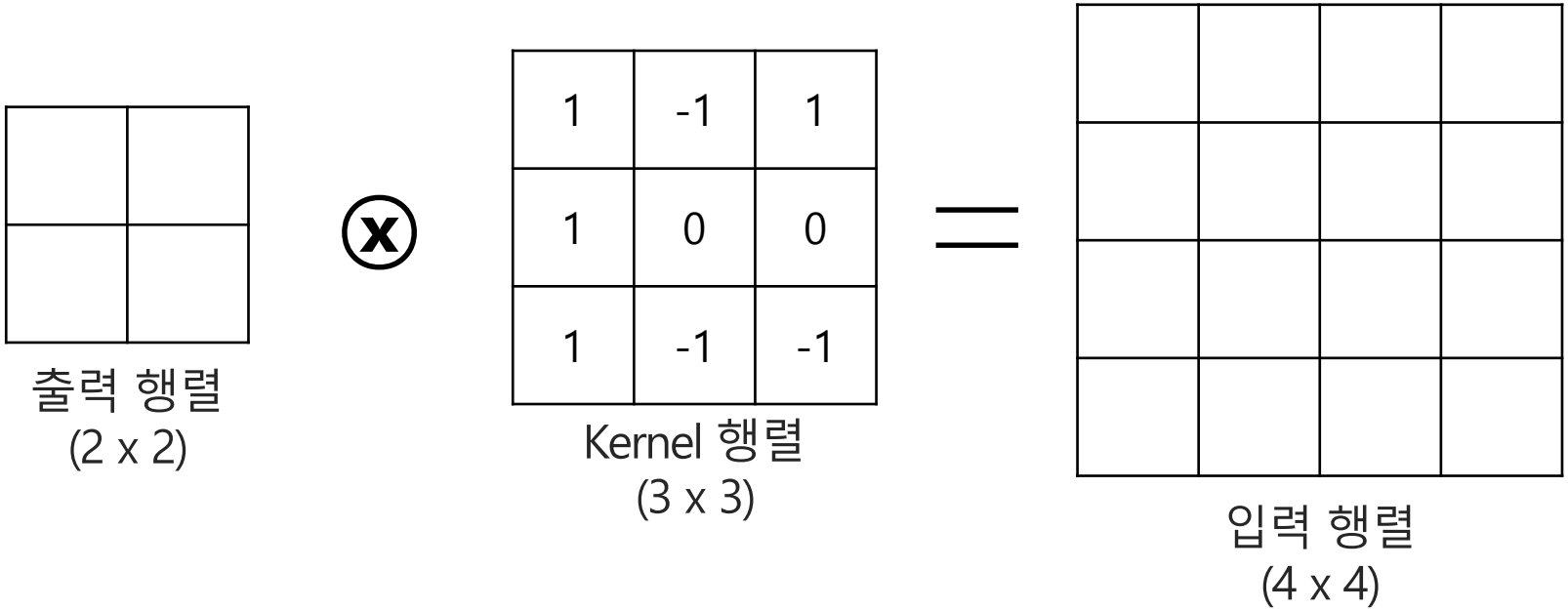

= 7*1 + 1*(-1) + 2*1
+ 0*1 + 0*0 + 9*0
+ 8*0 + 8*(-1) + 1*(-1)

- https://zzsza.github.io/data/2018/06/25/upsampling-with-transposed-convolution/

# Convolution & Transposed convolution 연산

❖ Convolution 연산의 역연산

- Convolution 연산에 대한 역연산으로는 아래의 연산이 불가
- 입력 행렬의 열과 Kernel 행렬의 행이 동일하지 않기 때문




출력 행렬
(2 x 2)

Kernel 행렬
(3 x 3)

입력 행렬
(4 x 4)



DMQA

# Convolution & Transposed convolution 연산

❖ Transposed convolution 연산

- 입력 행렬을 열 벡터로 변환하여 연산을 진행
- Kernel 행렬을 행렬의 곱셈이 가능한 형태로 변환
- Zero padding을 포함한 kernel 행렬이라고 생각하면 이해가 쉬움

| | | |
|---|---|---|
| 1 | -1 | 1 |
| 1 | 0 | 0 |
| 1 | -1 | -1 |

Kernel 행렬
(3 x 3)

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | -1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | -1 | -1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | -1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | -1 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | -1 | -1 | 0 | 1 | -1 | -1 | 0 | 1 | -1 | -1 |

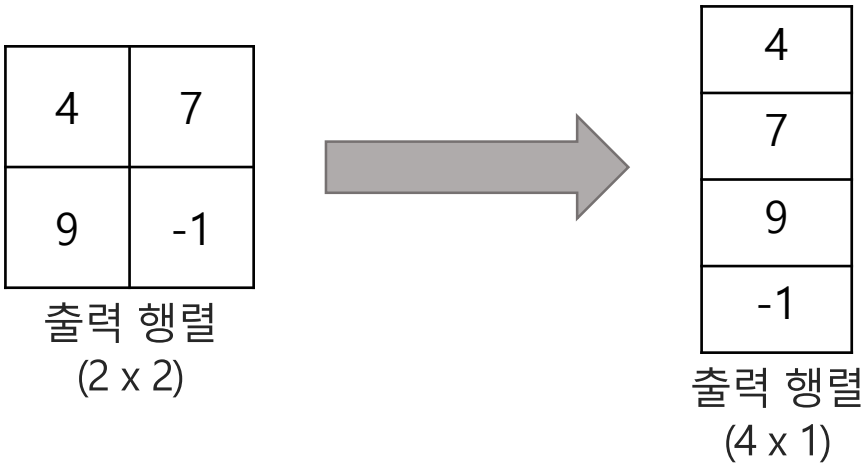
Kernel 행렬
(4 x 16)

- https://zzsza.github.io/data/2018/06/25/upsampling-with-transposed-convolution/

# Convolution & Transposed convolution 연산

❖ Transposed convolution 연산

- 출력 행렬을 열 벡터로 변환하여 연산을 진행
- Kernel 행렬을 행렬의 곱셈이 가능한 형태로 변환
- Zero padding을 포함한 kernel 행렬이라고 생각하면 이해가 쉬움

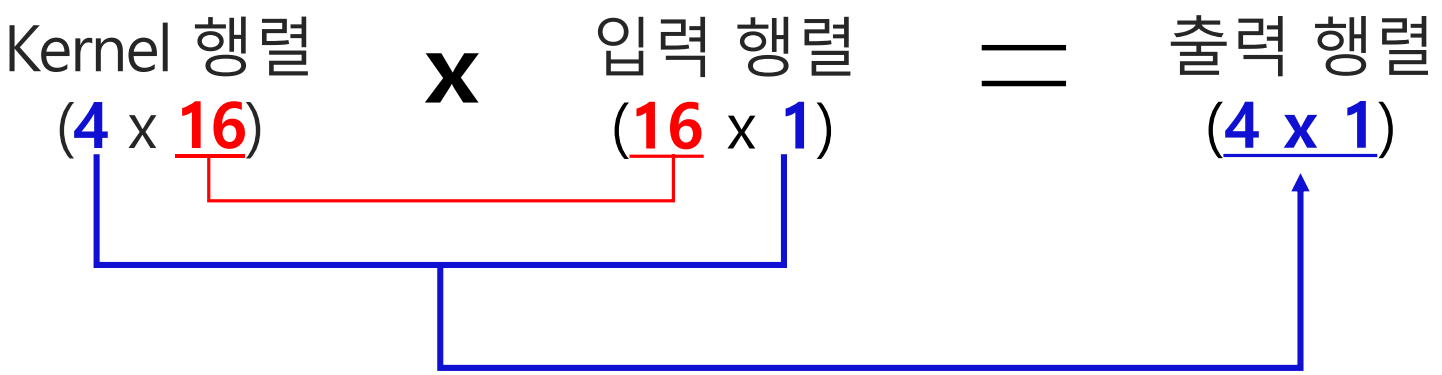



- https://zzsza.github.io/data/2018/06/25/upsampling-with-transposed-convolution/

# Convolution & Transposed convolution 연산

❖ Transposed convolution 연산

- 변환된 kernel 행렬(4x16)과 입력 행렬(16x1)에 대해서 내적 가능
- 출력 행렬의 형태는 (4x1)
- 출력 행렬의 형태를 (2x2) 형태로 변환




- https://zzsza.github.io/data/2018/06/25/upsampling-with-transposed-convolution/

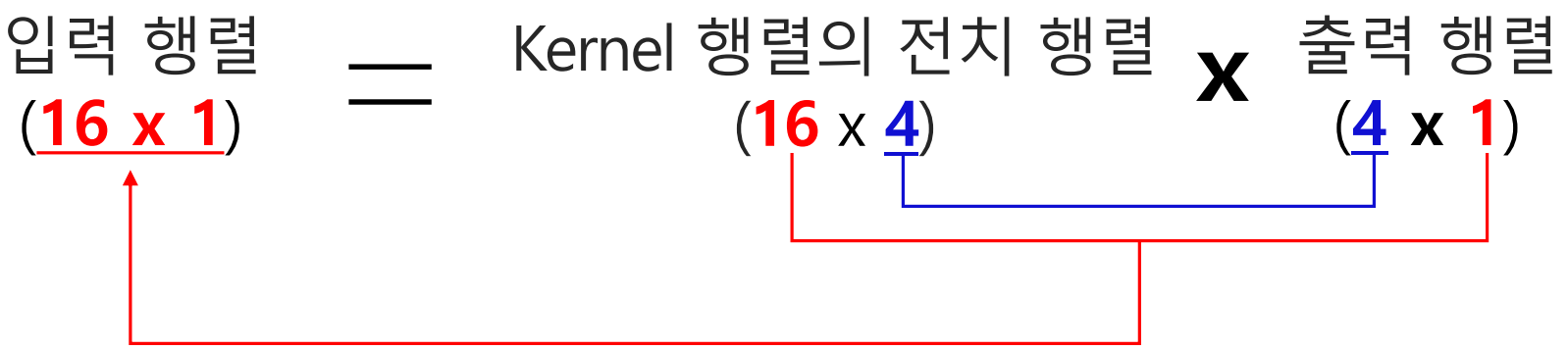


DMQA

# Convolution & Transposed convolution 연산

❖ Transposed convolution 연산

- 진정한 Transposed convolution 연산 시작
- Convolution 연산의 Kernel 행렬과 Transposed convolution 연산의 Kernel 행렬이 불일치
- Kernel 행렬의 전치 행렬을 출력 행렬에 곱함




- https://zzsza.github.io/data/2018/06/25/upsampling-with-transposed-convolution/

# Convolution & Transposed convolution 연산

Kernel 행렬의 전치 행렬

| | | | |
|---|---|---|---|
| **1** | 0 | 0 | 0 |
| **-1** | **1** | 0 | 0 |
| **1** | **-1** | 0 | 0 |
| 0 | **1** | 0 | 0 |
| **1** | 0 | **1** | 0 |
| **0** | **1** | **-1** | **1** |
| **0** | **0** | **1** | **-1** |
| 0 | **0** | 0 | **-1** |
| **1** | 0 | **1** | 0 |
| **-1** | **1** | **0** | **1** |
| **-1** | **-1** | **0** | **-1** |
| 0 | **-1** | 0 | **-1** |
| 0 | 0 | **1** | 0 |
| 0 | 0 | **-1** | **1** |
| 0 | 0 | **-1** | **-1** |
| 0 | 0 | 0 | **-1** |

⊗

출력 행렬

| |
|---|
| 4 |
| 7 |
| 9 |
| -1 |

=

입력 행렬

| |
|---|
| 4 |
| 3 |
| -3 |
| 7 |
| 13 |
| -3 |
| 10 |
| 1 |
| 13 |
| 2 |
| -10 |
| -6 |
| 9 |
| -10 |
| -8 |
| 1 |

압력 행렬(4x4)

| | | | |
|---|---|---|---|
| 4 | 3 | -3 | 7 |
| 13 | -3 | 10 | 1 |
| 13 | 2 | -10 | -6 |
| 9 | -10 | -8 | 1 |

# GAN 복습

❖ GAN(Generative Adversarial Network) – Pseudo code

**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, $k$, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

**for** number of training iterations **do**

**for** $k$ steps **do**

- Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

**end for**

- Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

**log likelihood**
**(Binary Classification Loss 함수)**

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

- Generative Adversarial Networks(2014)
Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio