

Catholic2Gogh

(by CycleGAN)



수학전공

201720014 강민주

201820973 김정태

201820995 박은비

I. 서론

1. 연구동기
2. 배경이론
 1. 딥러닝
 2. 오토인코더
 3. GAN

II. 본론

1. CycleGAN
2. 우리의 목표
3. 구현 코드 및 결과

III. 결론

1. 고찰

IV. 참고문헌

I. 서론

1. 연구동기

최근에 여러 가지 카메라 어플리케이션들이 개발되고 있다. 그 중에서도 사진을 찍으면 순정만화나, 애니메이션처럼 그림체를 바꿔주는 것들이 유행한다. 이러한 어플리케이션들에도 수학, 특히나 딥러닝이 사용된다는 것을 알게 되었다. 더 관심이 갔던 이유는 우리가 주로 공부해왔던 딥러닝에서는 분류를 위해 사용하는 분별 모델들이 많았는데, 이러한 기법들로는 새로운 것을 생성하거나 변환할 수 없다는 생각이 들었기 때문이다.

이미지를 생성하고, 색을 입히고, 그림체를 바꾸는 데에는 생성모델이 이용된다. 아래의 표는 분별 모델과 생성모델의 차이를 보여준다.

모델	학습단계가 할 일	예측 단계가 할 일	지도여부
분별모델	$P(y x)$ 추정	$f: x \mapsto y$	지도학습
생성모델	$P(x)$ 또는 $P(x y), P(x, y)$ 추정	$f: \text{씨앗} \mapsto x$ 또는 $f: \text{씨앗}, y \mapsto x,$ $f: \text{씨앗} \mapsto x, y$	분별모델

표1. 분별모델과 생성모델 비교

분별모델은 특징벡터의 확률 분포를 알아낼 필요가 없이 조건부확률 $P(y|x)$ 를 정확히 추정한다면 분류나 회귀문제를 높은 성능으로 풀 수 있는 데에 비해 생성 모델은 벡터 x 의 확률 분포를 추정한다. 이러한 모델은 레이블 정보가 필요 없는 비지도 학습에 해당하며, 이러한 성질 때문에 사람의 학습에 가깝다고 평가된다.

우리는 이러한 생성 모델 중에서도 GAN (Generative Adversarial Network)에 집중하였다. 비지도 학습임에도 불구하고 생성해낸다는 것이 의문이었는데, 이것은 굿펠로우의 생성자 G 와 판별자 D 를 대립시켜 학습하는 아이디어에 의해 설명된다. 여기서 생성자는 가짜 샘플을 생성하여 판별자가 진위구별을 못하게 하는 역할을 하며 판별자는 훈련 집합의 진짜 샘플과 생성자가 만든 가짜 샘플을 구별하는 역할을 한다.

우리는 GAN의 여러 가지 기법 중 CycleGAN을 이용하여 학교 건물 사진, 동양화를 포함한 다양한 그림들에 고흐의 화풍을 입히는 것에 도전하였다. CycleGAN을 포함한 대부분의 GAN들은 성능을 향상시키는 것이 어렵다고 평가된다. 우리는 그 이유를 손실함수에 있다고 생각하였고, 따라서 수학적 접근을 통해 손실함수의 최적화에 대하여 고민해보고 성능향상을 시도해 보려고 한다.

2. 배경이론

2-1. 딥러닝

2-1-1. 딥러닝의 정의

딥러닝(Deep Learning)이란 기계학습의 한 분야로 이야기된다. 여기서의 기계 학습이란 특정한 응용 영역에서 발생하는 데이터를 이용하여 문제를 해결하는 컴퓨터 프로그램을 만드는 작업을 뜻한다. 즉, 데이터를 중심으로 하는 접근방식을 가진다. 기계 학습은 학습, 훈련을 통하여 예측문제를 해결하는데, 여기서의 학습, 훈련이란 성능을 개선하면서 최적의 상태에 도달하는 작업이다. 예측문제에는 크게 회귀와 분류가 있다. 실숫값을 예측하는 문제를 회귀(regression)라고 하며, 어느 부류에 속하는지 예측하는 문제를 분류(classification)라고 한다. 딥러닝은 다층 퍼셉트론(MLP)에 은닉층을 많이 추가하여 깊어진 ‘깊은 신경망’을 학습하는데 사용한다.

2-1-2. 신경망

신경망은 인간이 뇌를 통해 문제를 처리하는 방법과 비슷한 방법으로 문제를 해결하기 위해 컴퓨터에서 채택하고 있는 구조를 말한다. 인간의 뇌가 기본 구조 조직인 뉴런이 서로 연결되어 일을 처리하는 것처럼, 수학적 모델로서의 뉴런이 상호 연결되어 네트워크를 형성할 때 이를 신경망이라 한다.

퍼셉트론(Perceptron)은 학습이 가능한 초창기 신경망 모델이다. 여기서 노드, 가중치, 층 등의 새로운 개념이 도입되었으며, 현대 신경망은 퍼셉트론을 병렬 구조와 순차 구조로 결합한 형태이다.

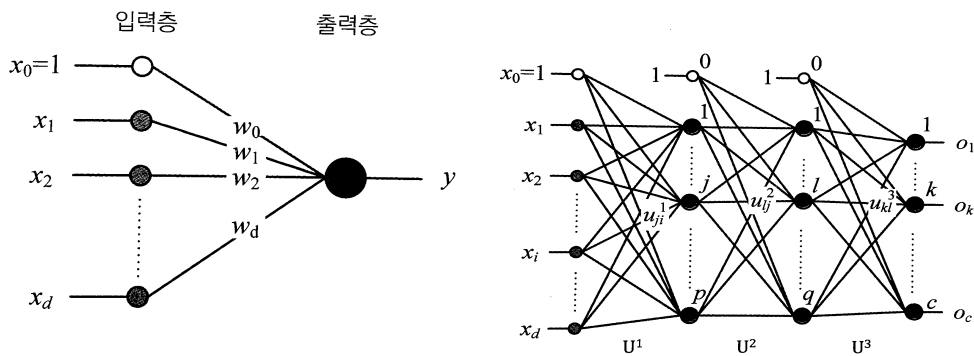


그림 1, 2. 퍼셉트론의 예시, 다층 퍼셉트론의 예시

퍼셉트론의 구조는 위의 그림과 같다. 입력층 (x_1, x_2) 과 출력층(y)이라 부르는 2개의 층이 있는데, 입력층은 아무런 연산을 하지 않으므로 층의 개수를 셀 때 제외한다. 따라서, 퍼셉트론의 층은 1개이다. 그림에서의 원은 ‘노드’라고 하며, 입력층에 있는 입력 노드 하나는 특징

벡터의 특징 하나에 해당한다. 따라서 특징 벡터를 $x = (x_1, x_2, \dots, x_d)^T$ 로 표기한다면, 입력층은 d 개의 노드를 가진다. 그리고 맨 위에 $x_0 = 1$ 로 표기된 바이어스라 부르는 여분 노드가 있어 총 $d+1$ 개의 입력 노드가 있다. 출력층은 y 로 표기된 하나의 노드를 가진다. 입력노드는 모두 출력 노드와 에지로 연결되어 있는데 에지는 w_i 로 표기된 가중치를 가진다.

퍼셉트론은 입력층에 특징 벡터가 들어오면, 연결된 특징값과 가중치를 곱한 결과인 $s = w_0 + \sum_{i=1}^d w_i x_i$ 를 활성화 함수의 입력으로 넣고 계산하는데, 활성화함수의 출력이 퍼셉트론의 최종 출력이 된다.

다층 퍼셉트론은 여러 개의 퍼셉트론을 결합한 다층 구조를 이용하는 것인데, 선형분리가 불가능한 상황을 해결한다. 은닉층을 두며, 오류 역전파 알고리즘을 사용한다는 특징이 있다.

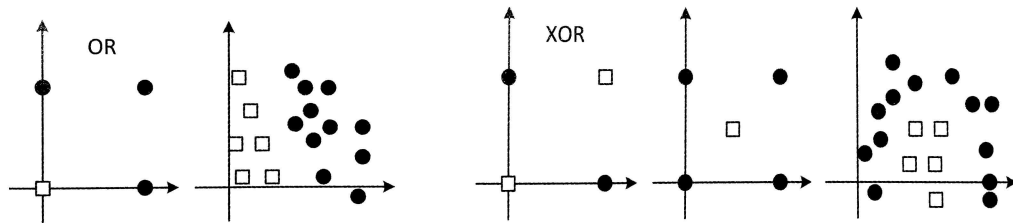


그림 3, 4. 선형분류가 가능한 예시, 선형분류가 불가능한 예시

2-1-3. 활성화함수

활성화 함수란 입력 신호의 총합을 출력신호로 변환하는 함수이다. 활성화 함수로 자주 활용되는 함수에는 계단 함수와 시그모이드 함수, ReLU함수가 있다. 세 함수 모두 출력값을 0과 1사이의 값을 갖는다. 각 함수의 식과 그래프는 아래와 같다. 이 중에서 계단 함수는 많이 사용되지 않는다. 그 이유는 계단함수는 대부분의 점에서 기울기가 0이지만, 시그모이드 함수의 기울기는 연속적으로 변화하며 어느 점에서든 0이 되지 않기 때문이다. 기울기가 0이 되지 않기 때문에 신경망이 올바르게 학습할 수 있다.

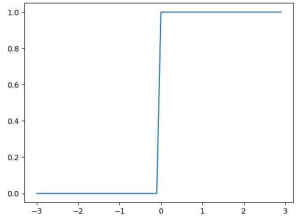
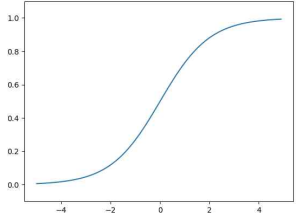
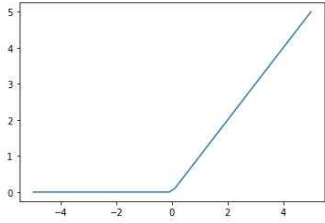
함수이름	함수식	그래프
계단함수	$\tau(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$	
시그모이드	$\tau(s) = \frac{1}{1 + e^{-as}}$	
ReLU	$\tau(s) = \max(0, s)$	

표2. 활성화 함수

신경망에서는 반드시 비선형 함수를 활성화 함수로 사용해야 하는데, 그 이유는 선형함수를 사용하면 신경망에서 층을 이루는 이유가 사라지기 때문이다.

2-1-4. 출력층 활성화 함수

마지막 층인 출력층에서는 앞서 소개한 함수들과는 다른 활성화 함수를 써야 한다. 데이터가 어느 부류에 속하는지에 대한 분류(classification) 문제는 소프트맥스 함수를, 연속적인 수치를 예측하는 회귀(regression) 문제에서는 항등함수를 사용하게 된다. 특히, 소프트맥스 함수의 출력은 반드시 0과 1사이의 실수이며 출력의 총합은 1인데, 이는 확률과 비슷하다. 즉 어떤 클래스로 속할 확률이 얼마인가를 나타내주는 것이 소프트맥스 함수의 출력이라고 할 수 있다.

함수이름	함수식
소프트맥스	$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)}$
항등함수	$y = x$

표3. 출력층 활성화 함수

2-1-5. 손실함수

신경망은 ‘하나의 지표’를 기준으로 최적의 매개변수 값을 탐색한다. 이 때 사용하는 지표를 손실함수(Loss function)라고 한다. 정확도가 아닌 손실함수 값을 지표로 이용하는 이유는 정확도는 매개변수의 작은 변화에는 거의 반응을 보이지 않거나 갑자기 변화할 수 있기 때문이다. 손실함수는 일반적으로 평균제곱오차(mean squared error, MSE)와 교차 엔트로피 오차(cross entropy error, CEE)를 사용한다.

함수이름	함수식
평균제곱오차	$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
교차엔트로피오차	$E = - \sum_i y_i \log \tilde{y}_i$

표4. 손실함수

우리는 손실함수의 값을 가능한 한 작게 하는 최적의 매개변수를 탐색해야 한다. 매개변수의 미분을 계산하고, 그 미분 값을 단서로 매개변수의 값을 서서히 갱신하는 과정을 반복한다. 이때 경사하강(Gradient Descent)알고리즘을 사용한다. 즉, 미분을 이용하여 여러 가중치 값을 대입하여, 그래프의 기울기를 가장 작게 하는 최적의 매개변수를 찾는 것을 최적화(optimization)라 한다.

2-2. 오토인코더

오토인코더[1]란 단순히 입력을 출력으로 복사하는 신경망이며 데이터를 자동으로 인코딩할 수 있도록 도와준다. 오토인코더는 아래 그림과 같은 구조로 이루어져 있다. 아래 그림에서 볼 수 있듯이 인코더(encoder)와 디코더(decoder), 두 부분으로 구성되어 있다. 먼저 ‘인코더(encoder)’란 인지 네트워크라고도 하며, 입력을 내부 표현으로 변환한다. 그다음 ‘디코더(decoder)’란 생성 네트워크라고도 하며, 내부 표현을 출력으로 변환한다. 오토인코더는 입력

층과 출력층의 뉴런수가 동일하다는 것만 제외하면 일반 다층 퍼셉트론과 동일한 구조이다. 또 가운데 층을 보면 엄청나게 작은 사이즈로 압축되어 있다는 것을 알 수 있다. 이 부분에서 주의해야 할 점은, 작게 압축하여 차원의 수를 작게 만들 경우에는 최대한 원본의 특징을 잘 살리며 차원을 낮춰나가야 한다는 것이다. 그렇게 한다면 디코더 부분을 통과했을 때 우리가 원하는 결과를 나오게 만들 수 있다. 이를 이용해 어떤 데이터를 작은 차원으로 만들어 더욱 효율적으로 보관하고 이동이 가능하도록 만들 수 있고 원본과 완벽히 같은 형태의 이미지는 만들어 낼 수 없으므로 원본과 비슷하게 생긴 여러 가지 형태의 이미지를 생성하는 것도 가능해진다.

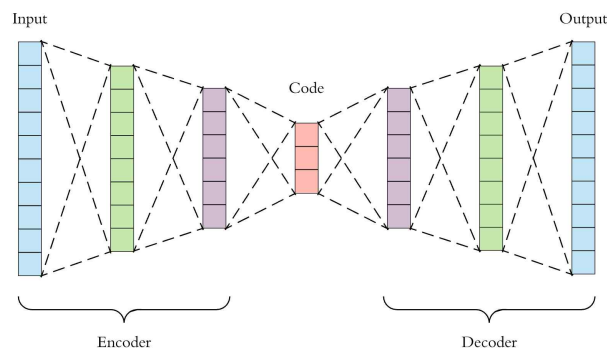


그림5. 오토인코더의 구조

2-3. GAN

2-3-1. GAN

GAN(generative Adversarial Network)[2]이란 적대적 생성 신경망으로, 동시에 두 개의 모델을 훈련하는 머신러닝의 한 종류이다. 두 개의 모델 중 하나인 생성자(Generator)는 가짜 데이터를 생성하도록 훈련되고, 다른 모델인 판별자(Discriminator)는 실제 샘플과 가짜 샘플을 구분하도록 훈련한다.

Generative(생성적)는 GAN의 목적을 잘 설명하는 용어이다. 즉 새로운 데이터를 생성한다는 의미이다. GAN이 생성하기 위해 학습할 데이터는 훈련 세트에 따라서 결정된다. 예를 들어 GAN을 이용해 고희가 만든 것 같은 작품을 만들고 싶다면 고희의 작품들을 데이터셋으로 사용해야 한다.

Adversarial(적대적)은 생성자와 판별자의 경쟁 구도를 나타낸다. 생성자의 목표는 판별자가 데이터셋에 있는 실제 데이터와 구분이 안 될 정도로 유사한 샘플을 만드는 것이다. 판별자의 목표는 생성자가 만든 가짜 데이터를 실제 데이터와 구별하는 것이다. 이렇게 생성자와 판별자는 서로를 이기려는 경쟁을 지속한다. 생성자가 더 그럴싸한 데이터를 생성할수록 판별자 역시 가짜 데이터와 진짜 데이터를 구별하는 일을 더 잘할 줄 알아야 한다.

마지막으로 Network(신경망)는 생성자와 판별자를 만드는 데 가장 널리 사용하는 머신러닝 모델 중 한 종류이다.

GAN의 수학적 기반은 복잡하지만 현실 세계의 비유를 활용하면 더 쉽게 이해할 수 있다. 형사를 판별자, 지폐 위조범을 생성자라고 생각해보자. 위조범이 위조지폐를 더 그럴듯하게 만들수록 형사가 지폐를 판별하는 능력 또한 더 뛰어나야 한다. 반대상황에도 마찬가지로 형사가 위조지폐를 찾는 능력이 뛰어날수록 위조범 역시 잡히지 않기 위해 더 지폐와 구별하기 어려운 위조지폐를 만들어내야 한다.

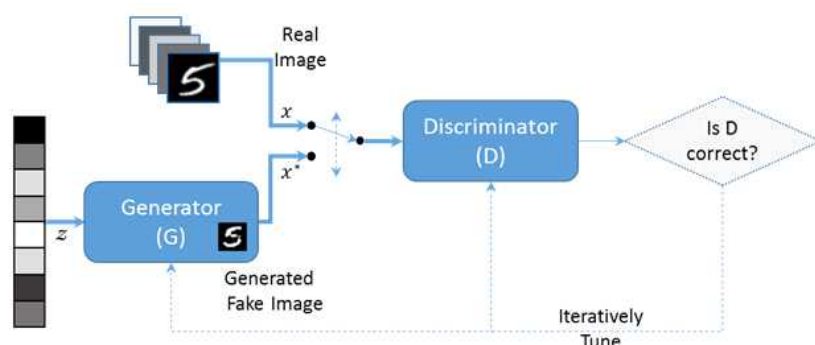


그림6. GAN의 구조

실제 이미지(Real Image)는 생성자가 그림을 모방하기 위해서 학습하는 진짜 샘플 데이터셋이다. 위 사진에선 MNIST 데이터셋으로 이루어져 있다. 이 데이터는 판별자의 훈련을 위해서 판별자의 입력(x)으로 제공된다.

z 는 생성자 네트워크에 입력으로 이 입력부터 생성자가 가짜 샘플 합성을 시작한다. 생성자는 무작위로 숫자 벡터(z)를 받아서 가짜 샘플(x^*)을 출력한다. 생성자의 목표는 훈련 데이터셋의 진짜 샘플과 구별이 안 되는 가짜 샘플을 생성하는 것이다. 판별자는 훈련 데이터셋의 진짜 샘플(x)과 생성자가 만든 가짜 샘플(x^*)을 입력으로 받고 각 샘플이 진짜일 확률을 계산해 출력한다. 이 모든 과정을 계속 반복하여 생성자와 판별자를 훈련시킨다. 판별자는 진짜 샘플(x)과 가짜 샘플(x^*)을 제대로 구분하도록 훈련하고, 생성자는 판별자가 가짜 샘플(x^*)을 진짜로 잘못 분류할 확률을 최대화하도록 훈련한다.

훈련은 생성자가 실제 데이터와 구별이 안 되는 데이터를 생성하여, 판별자가 두 데이터를 50 대 50의 확률로 랜덤하게 추측하게 됐을 때 훈련을 종료하게 된다. 이 이상으로 훈련을 진행하게 되면 과대적합이 진행이 되어 오히려 나쁜 데이터를 생성하게 된다.

다음은 GAN의 손실함수이다.

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

G 가 고정되어 있을 때, 생성자 G 에 대한 최적의 판별자 D 는 다음 식과 같다.

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \cdot \cdot \cdot (2)$$

위의 식은 다음과 같이 증명될 수 있다. 임의의 생성자 G 가 주어진 판별자 D 에 대한 학습 기준은 $V(G, D)$ 를 최대화 하는 것이다.

$$\begin{aligned} V(G, D) &= \int_x p_{data}(x) \log(D(x)) dx + \int_z p_z(z) \log(1 - D(g(z))) dz \\ &= \int_x p_{data}(x) \log(D(x)) + p_g(x) \log(1 - D(x)) dx \cdot \cdot \cdot (3) \end{aligned}$$

어떤 $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$ 에 대해 $y \rightarrow a \log(y) + b \log(1 - y)$ 함수는 $[0, 1]$ 에서 최솟값이 $\frac{a}{a+b}$ 이

므로 위 함수의 최솟값은 $\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$ 가 된다.

D 에 대한 교육 목표는 조건부 확률 $P(Y=y|x)$ 추정에 대한 로그 우도(log-likelihood)를 최대화하는 것으로 해석할 수 있으며, 여기서 Y 는 x 가 $p_{data}(y=1)$ 에서 왔는지 또는 $p_g(y=0)$ 에서 왔는지를 나타낸다. 따라서 식(1)은 이제 다음과 같이 재구성될 수 있다.

$$\begin{aligned} C(G) &= \max_D V(G, D) \\ &= E_{x \sim p_{data}} [\log D_G^*(x)] + E_{z \sim p_z} [\log(1 - D_G^*(G(z)))] \\ &= E_{x \sim p_{data}} [\log D_G^*(x)] + E_{x \sim p_g} [\log(1 - D_G^*(x))] \\ &= E_{x \sim p_{data}} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \right] + E_{x \sim p_g} \left[\log \frac{p_g(x)}{p_{data}(x) + p_g(x)} \right] \cdot \cdot \cdot (4) \end{aligned}$$

여기서 $p_g = p_{data}$ 일 때 식(2)에 의해 $D_G^*(x) = \frac{1}{2}$ 이다. 또한 식(4)에 의해

$C(G) = \log \frac{1}{2} + \log \frac{1}{2} = -\log 4$ 을 구할 수 있다. 따라서 $C(G)$ 의 극값은 $p_g = p_{data}$ 일 때이며 이

때 $D_G^*(x) = \frac{1}{2}$, $C(G) = -\log 4$ 이다. 여기서 $D_G^*(x) = \frac{1}{2}$ 을 살펴보면, 판별자 D 의 정확도가 0.5일 때 최적의 해라고 할 수 있으며 즉 가장 좋은 모델이라고 말할 수 있는 것이다.

2-3-2. DCGAN

GAN이 발표된 이후, GAN에 대해서 많은 연구가 진행되고 있었지만 그 때마다 구조의 불안정성이라는 문제가 있었다. 그래서 GAN을 안정화하기 위해 여러 연구를 진행하던 중, 그 문제를 가장 잘 해결한 알고리즘이 바로 DCGAN(Deep Convolutional GAN)[3]이다. 이름에서 알 수 있듯이 대부분의 상황에서 안정적으로 학습이 되는 합성곱(Convolution) 구조를 GAN에 녹여 넣었다. DCGAN의 구조는 다음 그림과 같다.

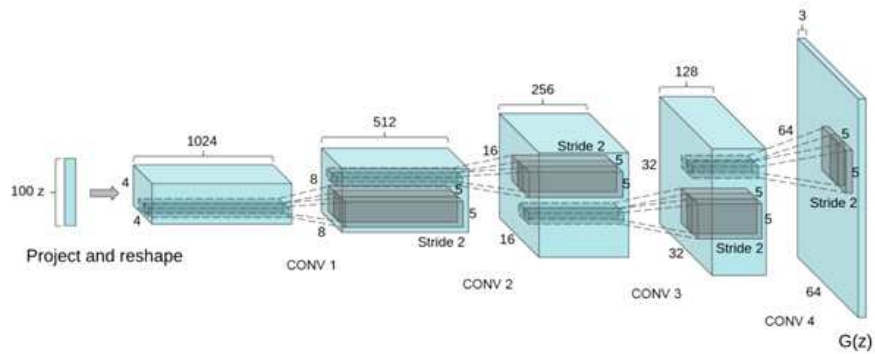


그림7. DCGAN의 구조

2-3-3. SGAN

SGAN(Semi-supervised GAN)[4]은 지금까지 살펴본 알고리즘들과는 큰 차이점이 있다. 다른 GAN들과 오토인코더는 대부분 훈련을 마친 뒤 이미지 생성을 위한 생성자를 사용한다. 하지만 SGAN은 판별자를 사용한다는 점이 다르다고 할 수 있다. 이미지 분류 모델을 학습시킬 때 대부분 지도학습으로 진행한다. 지도학습의 경우 정확도가 가장 높게 나올 수 있는 학습법이지만 모든 데이터들에 라벨링(labeling)을 진행해야 한다는 점에서 비용적, 시간적인 손해가 매우 크다. 따라서 SGAN은 이름에도 알 수 있듯이 지도학습이 아닌 준지도학습으로 학습을 진행한다. 준지도학습은 비지도 학습과 지도학습의 사이라고 볼 수 있다. 준지도학습은 훈련 데이터셋의 일부에만 클래스 레이블을 가지고 있으며, 데이터에 감춰진 내부 구조를 사용해 일부 레이블된 데이터 포인트를 일반화하고, 효율적인 방식으로 이전에 본 적 없는 새로운 샘플을 분류한다. 모든 데이터에 라벨링을 진행할 필요 없이 일부분의 데이터에만 라벨링을 하면 되기 때문에 소요되는 시간과 비용을 줄일 수 있다.

SGAN의 구조는 다음 그림과 같다. 생성자가 훈련하는 방법은 이전의 GAN들과는 다른 점이 거의 없다. 하지만 판별자가 훈련하는 방식이 이전의 GAN과는 다르다. 해야 할 일이 하나가 더 추가되었다. 우선 생성자가 만든 이미지와 실제 이미지를 받은 뒤 두 이미지가 실제 이미지인지 가짜 이미지인지 분류하는 작업을 한다. 이 작업에서 생성자의 이미지를 실제 이미지로 분류하도록 하는 것이 생성자의 목표이다. 그 후 실제 이미지로 분류한 이미지들에 대해서 분류를 진행한다.

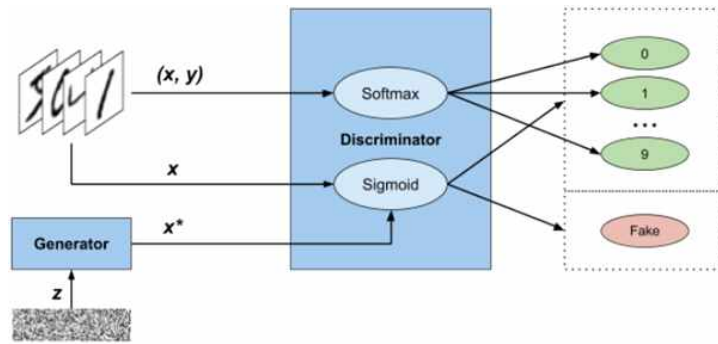


그림8. SGAN의 구조

2-3-4. CGAN

이런 GAN을 실생활에 적용하여 원하는 대로 사용하기에는 한 가지 큰 단점이 있다. 바로 내가 원하는 형태의 이미지를 뽑아낼 수 없다는 것이다. MNIST를 예로 들어보자. MNIST 데이터셋 모두를 훈련시켰을 때 이전의 GAN들로는 내가 원하는 숫자를 뽑아낼 수 없다. 내가 원하는 숫자를 4라고 했을 때 만약 4라는 숫자를 뽑아내려면 4라는 숫자가 나올 때까지 여러 번 생성자를 실행시켜서 얻어내야 한다. MNIST에서는 0~9까지 단지 10개의 라벨밖에 없기 때문에 별 문제가 되지 않는다고 생각할 지도 모른다. 하지만 빨간머리의 소녀 이미지를 만들어내야 할 때 얼마나 여러 번 생성자를 실행시켜야 할지를 생각해보면 이 점이 얼마나 큰 문제인지를 느낄 수 있을 것이다. 이런 큰 단점을 고쳐 만들어 낸 것이 CGAN(Conditional GAN)[5]이다. 생성할 데이터의 종류를 결정할 수 있다면 조건에 맞는 이미지를 생성할 수 있으므로 많은 어플리케이션이 가능해진다.

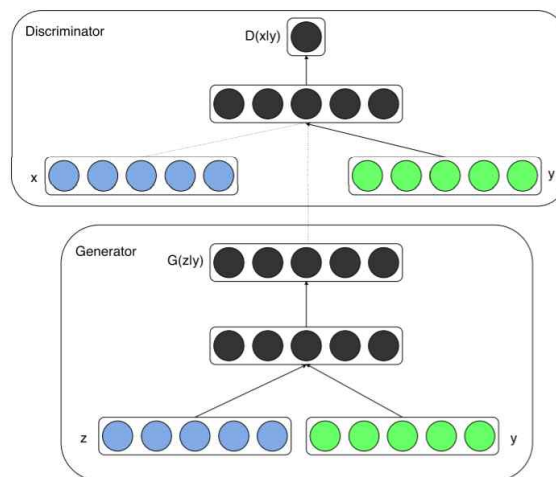


그림9. CGAN의 구조

우선 생성자가 만든 이미지와 실제 이미지를 받은 뒤 생성자가 만든 이미지가 어떤 라벨에

속하는 실제처럼 보이는 이미지인지 가짜처럼 보이는 이미지인지 분류하는 작업을 한다. 이 작업에서 생성자의 이미지를 실제 이미지로 분류하는 것이 생성자의 목표이다. 이전의 알고리즘과는 달리 생성자가 이미지를 생성할 때 무작정 이미지를 생성하는 것이 아니라 라벨에 해당하는 이미지를 생성하도록 한다. 이 변경된 점 때문에 훈련을 마친 뒤 내가 원하는 모양의 이미지를 생성해낼 수 있게 되는 것이다. 이런 알고리즘을 이용하여 다음 설명할 pix2pix, CycleGAN와 같은 알고리즘을 만들 수 있게 되었다.

다음은 CGAN의 손실함수이다. GAN과 동일하게 생성자와 판별자 간의 minmax 게임을 통해 학습을 하며 y 가 조건부로 들어가게 된다.

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)} [\log D(x|y)] + E_{z \sim p_z(z)} [\log(1 - D(G(z|y)))] \cdot \cdot \cdot (5)$$

2-3-5. pix2pix

이미지 대 이미지 변환 문제에 대한 범용적 해결책으로 조건부 적대적 네트워크를 찾아본 결과, 이들 네트워크는 입력 이미지에서 출력 이미지까지의 매핑을 배울 뿐만 아니라, 이 매핑을 훈련시키기 위한 손실 기능도 학습한다. 이것은 전통적으로 매우 다른 손실 제형(formulation)이 필요할 수 있는 문제에 동일한 일반적 접근법을 적용하는 것을 가능하게 한다. 이러한 접근방식이 라벨 지도에서 사진을 합성하고, 가장자리 지도에서 사물을 재구성하며, 다른 작업 중에서도 이미지를 색칠하는 데 효과적이라는 것을 증명한다. 이로 인해 더 이상 매핑 함수를 직접 설계하지 않으며, 손실 함수도 직접 설계하지 않고도 합리적인 결과를 얻을 수 있음을 나타낸다.[6]

많은 GAN에서 파생된 모델들은 무작위 잡음 벡터 z 에서 $G: z \rightarrow y$ 인 출력 이미지 y 가 나오게 매핑을 학습하는 생성 모델이다. 대조적으로 CGAN은 관측된 이미지 x 와 무작위 잡음 벡터 z 로부터 y 로의 매핑인 $G: \{x, z\} \rightarrow y$ 을 학습한다. 생성자 G 는 적대적으로 훈련된 판별자 D 에 의해 실제 이미지와 구별될 수 없는 출력을 생성하도록 훈련되며, 판별자 D 는 이 출력 이미지는 생성자의 가짜 이미지를 탐지하도록 훈련된다. 학습 절차는 다음 그림과 같다.

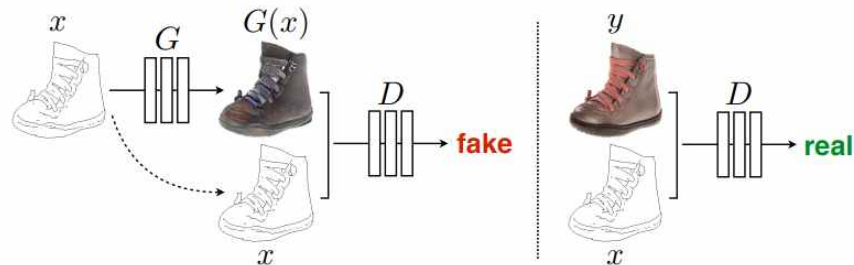


그림10. pix2pix의 학습 절차

CGAN의 목적함수는 다음과 같이 표현된다. 여기서 G 는 이를 최대화하려는 적대적인 D 에

대해 이 함수를 최소화하려고 한다. 즉 $\operatorname{argmin}_G \max_D L_{cGAN}(G, D)$ 로 표현된다.

$$L_{cGAN}(G, D) = E_{x,y}[\log D(x, y)] + E_{x,z}[\log(1 - D(x, G(x, z)))] \cdot \cdot \cdot (6)$$

또한 판별자 조절의 중요성을 시험하기 위해 판별자가 x 를 준수하지 않는 조건이 없는 변형과 비교한다.

$$L_{GAN}(G, D) = E_y[\log D(y)] + E_{x,z}[\log(1 - D(G(x, z)))] \cdot \cdot \cdot (7)$$

이전의 접근방식은 GAN 목표를 $L2$ 거리와 같은 손실과 혼합하는 것이 유익하다는 것을 발견했다. 판별자의 일은 변함이 없지만, 생성자는 판별자를 속이는 것뿐만 아니라 $L2$ 의 의미로 실제 출력과 가까이에 있어야 한다. 또한 $L2$ 가 아닌 $L1$ 거리를 사용하여 다음과 같이 모호성을 줄인다.

$$L_L(G) = E_{x,y,z}[\|y - G(x, z)\|_1] \cdot \cdot \cdot (8)$$

따라서 최종 목적함수는 다음과 같다.

$$G^* = \operatorname{argmin}_G \max_D L_{cGAN}(G, D) + \lambda L_L(G) \cdot \cdot \cdot (9)$$

다음은 생성자의 아키텍처인 U-net[7]인데, U-net은 오토인코더의 모양에서 따온 구조이다. 인코딩 하는 부분은 오토인코더와 다른 부분이 없지만 디코딩 과정에서 이전의 이미지를 알려주며, 생성자가 원래의 이미지의 구조를 기억하며 새로운 이미지를 생성하도록 하는 역할을 한다.

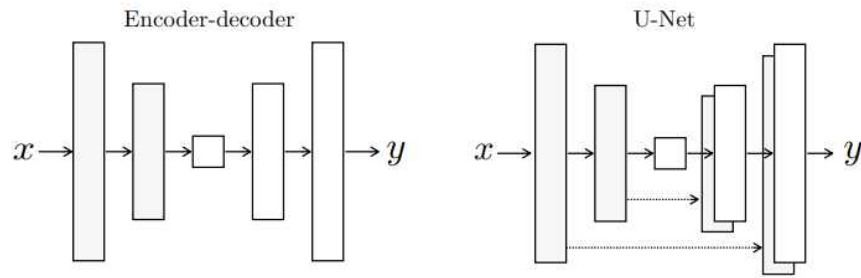


그림11. 오토인코더와 U-Net의 구조

II. 본론

1. CycleGAN

1-1. 이미지 대 이미지 변환

이미지 대 이미지(image-to-image) 변환은 짝진 형태의 이미지 훈련 세트를 이용해 입력 이미지와 출력 이미지를 매핑하는 것이 목표인 컴퓨터 비전과 그래픽의 한 분야이다. 그러나 많은 작업의 경우 짝이 지어진 학습 데이터를 얻는 것은 어렵다. 예를 들어 흑백 이미지의 경우 먼저 컬러 사진을 로드하여 흑백 필터를 적용한다. 그 다음 원본 이미지를 한 도메인으로 사용하고 흑백 필터를 적용한 이미지를 다른 도메인으로 사용한다. 이렇게 하면 양쪽 도메인에 모두 존재하는 이미지를 준비할 수 있고, 그 다음 훈련된 GAN을 다른 곳에 적용할 수 있다. 하지만 완벽한 이런 쌍을 준비하기 쉽지 않다면 모델을 학습하기 어렵다고 할 수 있다.

짝지어진 훈련 데이터가 없는 경우, 한 이미지 컬렉션의 특수 특성을 잡아내고 이러한 특성이 다른 이미지 컬렉션으로 어떻게 바꿀 수 있는지 알아내는 것이 요점이다. 이 문제는 더 광범위하게 이미지를 주어진 장면의 한 표현에서 다른 표현으로 바꾸는 이미지 변환으로 설명될 수 있다. 그러나 짝지어진 훈련 데이터를 얻는 것은 어렵고 비용이 많이 들 수 있다.

1-2. CycleGAN

CycleGAN[8]은 두 가지 훈련 데이터가 없는 경우, 짝지어진 입력과 출력 데이터 없이 도메인 간 이미지를 변환하는 방법을 학습하기 위한 접근방식을 제시한다. 우선 도메인들 사이에 몇 가지 근본적인 관계가 있다고 가정한다. 예를 들어 이미지들은 동일한 근본적인 장면의 서로 다른 렌더링이라고 생각하고 그 관계를 학습한다. 여기서 렌더링이란 컴퓨터 프로그램을 사용하여 모델 또는 이들을 모아놓은 장면인 씬 파일로부터 영상을 만들어내는 과정을 말한다. 비록 쌍체 데이터의 형태로 지도는 부족하지만 세트 레벨에서 지도할 수 있다. 우선 도메인 X 의 이미지 세트와 도메인 Y 의 다른 세트를 받는다. 그리고 출력 $\hat{y} = G(x), x \in X$ 가 \hat{y} 과 y 를 구분하도록 훈련된 적수에 의해 이미지 $y \in Y$ 와 구별되지 않도록 매핑 $G: X \rightarrow Y$ 를 훈련한다. 이론적으로 이 목표는 경험적 분포 $p_{data}(y)$ 와 일치하는 \hat{y} 에 대한 출력 분포를 유도할 수 있다. 따라서 최적의 G 는 도메인 X 를 Y 와 동일하게 분산된 \hat{Y} 로 변환한다. 즉 CycleGAN의 목표는 $G(x)$ 의 이미지분포가 적대적 손실(Adversarial loss)을 이용해 분포 Y 와 구별할 수 없도록 매핑 $G: X \rightarrow Y$ 를 학습하는 것이다. 이 매핑은 제약이 적기 때문에 역매핑 $F: Y \rightarrow X$ 과 함께 진행했고, $F(G(X)) \approx X$ 을 시행하기 위한 주기 일관성 손실(Cycle Consistency loss)을 도입한다.

1-3. 손실 함수

1-3-0. 소개

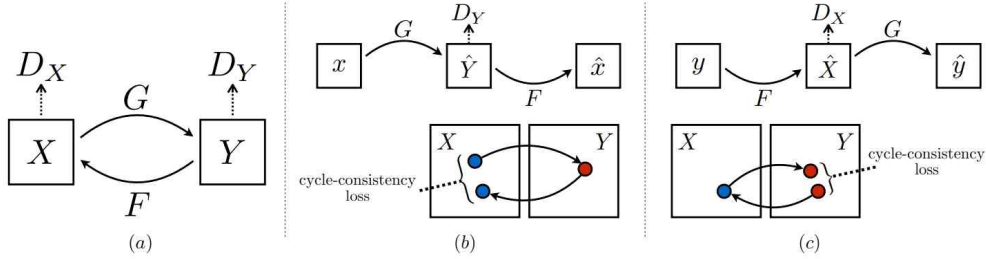


그림12. CycleGAN의 구조

CycleGAN의 목표는 두 도메인 X 와 Y 간의 매핑을 학습하는 것이다. 위의 그림12-(a)와 같이 이 모델은 두 개의 매핑함수 $G: X \rightarrow Y$ 와 $F: Y \rightarrow X$ 를 포함한다. x 는 X 에 속하는, y 는 Y 에 속하는 샘플이고 데이터 분포는 $x \sim p_{data}(x)$, $y \sim p_{data}(y)$ 로 나타낼 수 있으며 두 개의 적대적 판별자 D_X 와 D_Y 를 도입한다. 여기서 D_X 은 이미지 $\{x\}$ 와 변환된 이미지 $\{F(y)\}$ 를 구별하고, D_Y 는 이미지 $\{y\}$ 와 변환된 이미지 $\{G(x)\}$ 를 구별한다.

다음은 두 종류의 손실이다. 첫 번째로 적대적 손실(Adversarial loss)은 생성된 이미지의 분포를 대상 도메인의 데이터 분포와 일치시키기 위한 것이다. 두 번째로 주기 일관성 손실(Cycle Consistency loss)은 학습된 매핑 G 와 F 가 서로 모순되지 않기 위한 것이라고 할 수 있다.

1-3-1. 적대적 손실 (Adversarial Loss)

생성자 G 의 모든 변환은 이에 해당하는 판별자 D_Y 가 있고 생성자 F 는 판별자 D_X 가 있다. D_X 가 도메인 X 로 변환할 때 진짜 사진처럼 보이는지 항상 테스트한다고 생각할 수 있다. D_Y 는 그 반대로 사용한다. 이전 구조와 같은 아이디어이지만 이제는 손실이 두 개 있으므로 판별자가 두 개이다. 사과에서 진짜처럼 보이는 오렌지로 변환하는 것뿐만 아니라 변환된 오렌지를 진짜처럼 보이는 사과로 재구성해야한다. 따라서 적대 손실이 두 번 등장한다. 함수 $G: X \rightarrow Y$ 와 D_Y 에 대한 적대적 손실은 다음 식과 같은 함수를 사용한다.

$$L_{GAN}(G, D_Y, X, Y) = E_{y \sim p_{data}(y)} [\log D_Y(y)] + E_{x \sim p_{data}(x)} [\log(1 - D_Y(G(x)))] \cdot \cdot \cdot (10)$$

정리하면 G 는 생성된 이미지 $G(x)$ 를 도메인 Y 와 비슷하게 보이도록 노력을 하며 D_Y 는 변환된 샘플인 $G(x)$ 와 실제 샘플인 y 와의 구별을 목표로 한다. 즉 G 는 위의 함수를 최소화시키려고 하며 반대로 D 는 위의 함수를 최대화시키고자 하는데, 이는 식(11)로 나타낼 수 있다. 또한 $F: Y \rightarrow X$ 와 D_X 에 대해서도 유사한 적대적 손실을 적용하고, 이는 식(12)로 나타낼 수

있다.

$$\min_G \max_{D_Y} L_{GAN}(G, D_Y, X, Y) \cdot \cdot \cdot (11)$$

$$\min_F \max_{D_X} L_{GAN}(F, D_X, Y, X) \cdot \cdot \cdot (12)$$

적대적인 훈련은 이론적으로 각각 대상 도메인 Y 와 X 와 동일하게 분포된 출력을 생성하는 매핑 G 와 F 를 배울 수 있다. 그러나 충분한 용량이 있는 네트워크는 동일한 입력 이미지 집합을 대상 도메인에서 임의의 이미지의 무작위 순열(random permutation)에 매핑할 수 있으며, 여기서 학습된 매핑 중 하나가 대상 분포와 일치하는 출력 분포를 유도할 수 있다. 따라서 적대적 손실만으로는 학습된 함수가 개별 입력 x_i 를 원하는 출력 y_i 에 매핑할 수 있다는 것을 보장할 수 없다.

1-2-2. 주기 일관성 손실 (Cycle Consistency Loss)

이러한 이미지 변환은 \hat{y} 에 걸쳐 동일한 분포를 유도하는 매핑 G 가 무한히 많기 때문에 개별 입력 x 와 출력 y 가 의미 있는 방식으로 쌍을 이루도록 보장하지는 않는다. 더욱이 실제로 대립적 목표를 분리하여 최적화하는 것이 어렵다는 문제가 발생한다. 표준 절차(standard procedure)로 인해 모드 붕괴(mode collapse)라고 알려진 문제가 종종 발생하는데, 여기서 모든 입력 이미지가 동일한 출력 이미지에 매핑되고 최적화가 진전되지 않는다.

이러한 문제들로 인해 적대적 손실 단독으로는 제대로 된 학습을 보장하기 어렵기 때문에 목적 함수에 더 많은 구조를 추가해야 한다. 가능한 매핑 함수의 공간을 줄이기 위해 위 그림 (b), (c)와 같이 학습된 매핑함수는 주기 일관성(cycle consistent)이 있어야 한다. 예를 들어 영어에서 프랑스어로 문장을 번역한 다음 다시 프랑스어로 번역하면 본래의 영어 문장으로 돌아가야 한다는 점이다. 수학적으로 설명한다면 변환 $G: X \rightarrow Y$ 와 또 다른 변환 $F: Y \rightarrow X$ 이 있다면 G 와 F 는 서로 반대여야 하고, 두 매핑 모두 전단사가 되어야 한다. 변환에 지도 G 와 F 를 동시에 학습하고, $F(G(x)) \approx x$ 와 $G(F(y)) \approx y$ 가 되게 하는 주기 일관성 손실을 추가함으로써 이러한 구조적 가정을 적용한다.

원본 도메인의 이미지 x 와 두 번 변환된 이미지 \hat{x} 간의 차이에 대한 함수이다. 원본 이미지와 두 번 변환된 이미지는 동일해야 한다. 그렇지 않다면 $x - y - x$ 매핑에 일관성이 없다고 할 수 있다. 원본 도메인의 이미지 y 와 두 번 변환된 이미지 \hat{y} 간의 관계도 마찬가지이다. 앞에 소개한 손실은 정방향 주기 손실 (Forward Cycle Consistency), 그 다음 소개한 손실은 역방향 주기 손실(Backward Cycle Consistency)라고 한다. 정방향 주기 손실과 역방향 주기 손실은 각각 식(13), 식(14)으로 나타낼 수 있다

$$x \rightarrow G(x) \rightarrow F(G(x)) \approx x \text{ 일 때 } E_{x \sim p_{data}(x)} [\|F(G(x)) - x\|_1] \cdot \cdot \cdot (13)$$

$$y \rightarrow F(y) \rightarrow G(F(y)) \approx y \text{ 일 때 } E_{y \sim p_{data}(y)} [\|G(F(y)) - y\|_1] \cdot \cdot \cdot (14)$$

따라서 주기 일관성 손실함수는 다음 식과 같이 나타낼 수 있다.

$$L_{cyc}(G, F) = E_{x \sim p_{data}(x)} [\|F(G(x)) - x\|_1] + E_{y \sim p_{data}(y)} [\|G(F(y)) - y\|_1] \cdot \cdot \cdot (15)$$

재구성된 이미지 $F(G(x))$ 는 입력 이미지 x 와 근접하게 일치하게 된다. 다음 그림에서 완전한 손실의 감소와 비교한다. GAN 손실을 제거하면 주기 일관성 손실이 제거되는 것과 마찬가지로 결과가 상당히 저하된다. 따라서 두 손실 모두 모델의 결과에 매우 중요하다고 결론짓는다.

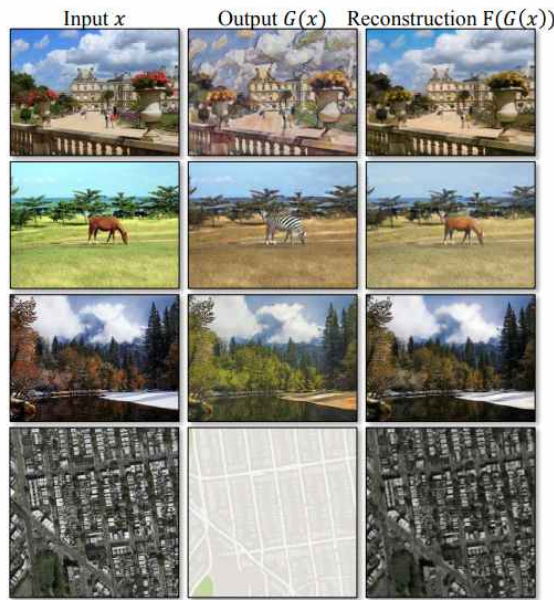


그림13. 주기 일관성 손실

1-2-3. 동일성 손실

CycleGAN은 사진의 전반적인 색 구성(또는 온도)을 유지하기 원한다. 이미지에 여러 필터를 적용한 후에도 원본 이미지를 복제 가능하게 만드는 것으로 생각할 수 있다. 따라서 입력 데이터와 출력 데이터의 색 구성을 보존하기 위해 추가적인 손실인 동일성 손실(identity loss)을 도입한다. 이 손실은 타깃 도메인의 실제 샘플, 즉 그림이 아닌 사진이 입력으로 들어왔을 때는 사진 자기 자신을 출력으로 산출하도록 생성자를 정규화(regularize)한다. 그 식은 다음과 같다. 동일성 손실은 CycleGAN 작동에 필수적이지는 않지만 완벽을 기하기 위해 포함시켰다고 할 수 있다. 이는 경험적으로 더 나은 결과를 만들고 타당하게 보이는 제약을 부과하기 때문이다.

$$L_{identity}(G, F) = E_{y \sim p_{data}(y)} [\|G(y) - y\|_1] + E_{x \sim p_{data}(x)} [\|F(x) - x\|_1] \cdot \cdot \cdot (16)$$

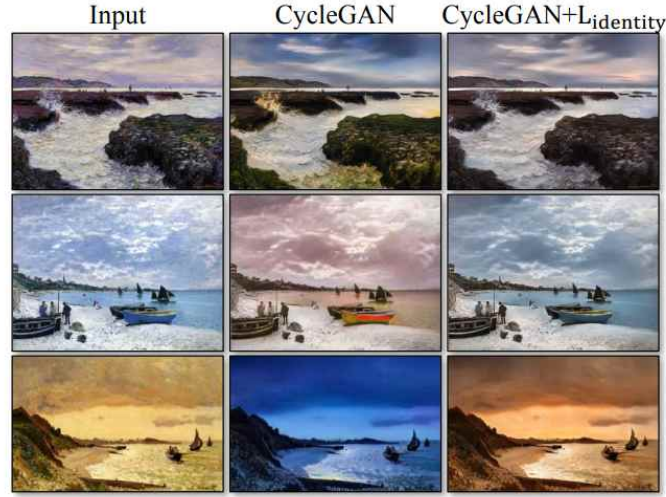


그림14. 동일성 손실

1-4. 전체 목적 함수

이 손실을 도메인 X 와 Y 의 적대적 손실과 결합하면, 짝이 없는 데이터의 이미지 대 이미지 변환을 야기할 수 있으므로 결과적으로 전체 목적 함수가 완성된다. 전체적인 목적 함수는 다음 식과 같다.

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cyc}(G, F) \cdot \cdot \cdot (17)$$

여기서 λ 는 두 목적이 상대적 중요성을 제어하게끔 만들어 준다. 결국 CycleGAN은 다음 식을 해결하는 것을 목표로 한다고 할 수 있다.

$$G^*, F^* = \operatorname{argmin}_{G, F} \max_{D_X, D_Y} L(G, F, D_X, D_Y) \cdot \cdot \cdot (18)$$

이 모델은 두 개의 오토인코더를 훈련시키는 것으로 볼 수 있다. 즉, 오토인코더 $F \circ G: X \rightarrow X$ 를 다른 오토인코더 $G \circ F: Y \rightarrow Y$ 와 함께 훈련하는 것이다. 두 오토인코더는 이미지를 다른 영역으로 변환하는 중간 표현을 통해 이미지를 자신에게 매핑한다. 그러한 설정은 또한 임의의 표적 분포(arbitrary target distribution)와 일치하도록 오토인코더의 병목층을 훈련시키기 위해 적대적 손실(adversarial loss)을 사용하는 적대적 오토인코더(adversarial auto-encoder)의 특별한 사례로 볼 수 있다. X 오토인코더의 목표분포는 도메인 Y 의 목표분포이다. 또한 한 방향으로만 사이클 손실만을 가지고 이러한 방법을 평가하고, 한 사이클로는 이 절제되지 않은 문제에 대한 훈련을 정규화하기에 충분하지 않다는 것을 보여준다.

2. 우리의 목표

우리는 GAN의 여러 가지 기법 중 CycleGAN을 이용하여 학교 건물 사진에 고흐를 포함한 여러 화가들의 화풍을 입히는 것에 도전하였다. CycleGAN을 포함한 대부분의 GAN들은 성능을 향상시키는 것이 어렵다고 평가된다. 우리는 그 이유를 손실함수에 있다고 생각하였기 때문에 수학적 접근을 통해 손실함수의 최적화에 대하여 고민해보고 성능향상을 시도해 보려고 한다. 또한 작성한 코드에 대하여 학습률, epoch, 활성화 함수, 최적화 함수, 손실 함수, 생성자와 판별자의 층을 쌓는 방식, 입력 이미지의 크기 등 파라미터들의 조절을 통한 성능향상 또한 도전해 보려고 한다.

이를 통하여 우리는, 짝지어진 데이터가 없는 경우에서의 한 이미지 컬렉션의 특수 특성을 잡아내고 다른 이미지 컬렉션으로 변환되는 과정과, 성능이 향상되면서 그전의 결과와 어떤 차이가 있는지 관찰하려고 한다.

3. 구현 코드 및 결과

3-1. 네트워크 구조

데이터셋은 실제 이미지는 Flickr에서 landscape와 landscapephotography 키워드로 검색해서 나온 6853장의 사진이고, 그림 이미지는 wikiart에서 다운받은 Van Gogh의 401장의 그림이다.

conv2d는 Conv2d층을 strides = 2, padding="same"으로 통과하고 alpha가 0.2인 LeakyReLU층을 통과 하도록 설정한 block이다. deconv2d는 size = 2로 업샘플링한 뒤, Conv2D층을 strides = 1, padding="same", activation="relu"로 통과한 뒤 InstanceNormalization을 하여 반환해주는 block이다.

두개의 generator에서는 U-Net구조를 사용하여 모델을 작성하였다. 이미지의 크기만큼 인풋으로 받고, strides = 2, padding = "same"으로 kernel_size를 2×10까지 늘려가며 다운샘플링(down sampling)을 하였다. 업 샘플링(up sampling) 과정에는 strides = 1, padding = "same", activation = "relu"를 이용하여 다시 원본의 사이즈로 업 샘플링을 하였다.

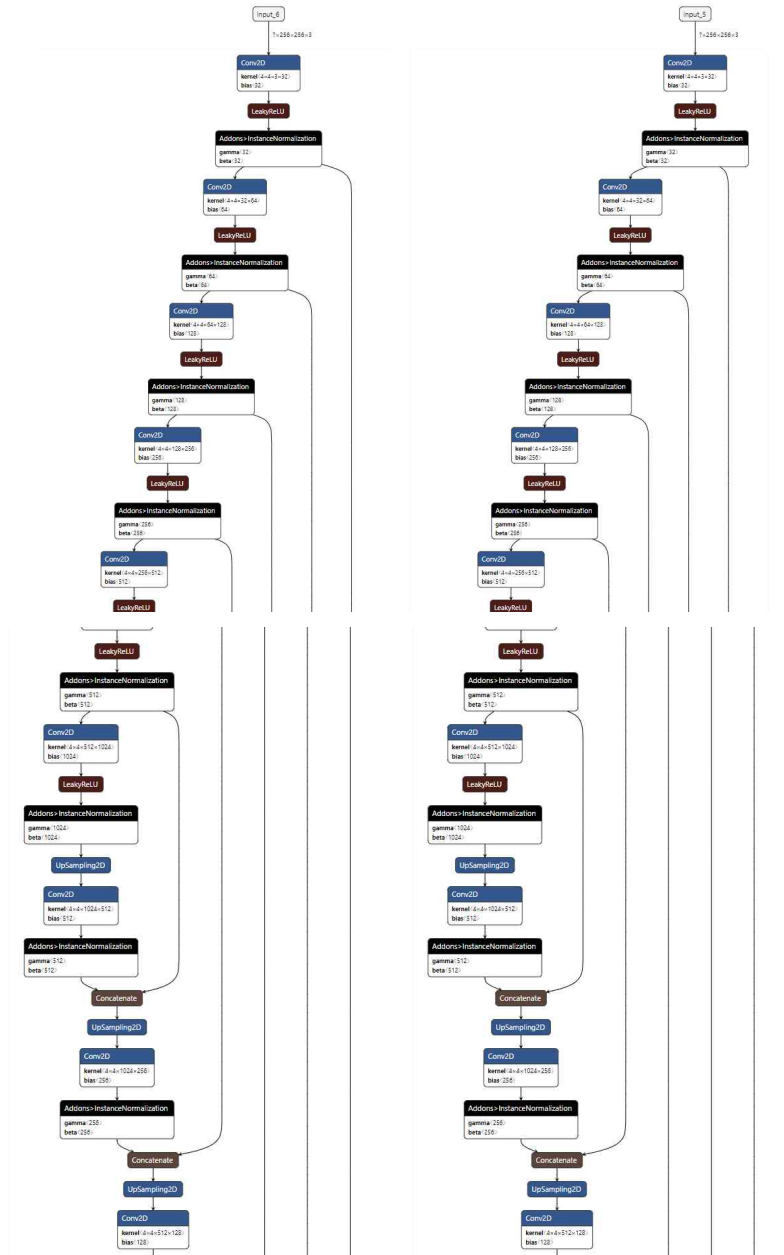
deconv2d block을 이용해 이미지가 진짜인지 가짜 이미지인지, 그리고 진짜 이미지라면 어떤 레이블인지 판별하는 두개의 판별자 모델을 만든다. 판별자는 loss = "mse", optimizer는 lr이 0.0002이고, beta_1 = 0.5인 Adam을 이용해 컴파일하였다. 생성자는 loss = "mse", "mae"를 이용하였고 loss 가중치로 1, 10, 9를 주었다. (사이클 일관성 손실 : 10.0, 동일성 손실 : 0.9 * 사이클 일관성 손실) optimizer로는 판별자와 같은 Adam을 주어 컴파일 하였다. Batch_size를 32로 하여 200번 반복하여 훈련 하였다.

다음은 훈련 상세 설명이다. 첫째로 L_{GAN} 을 음의 로그 가능도(negative log likelihood)보다 최소 제곱 손실 (least-squared loss)이 안정적이기 때문에 바꿨다. G 와 F 는 최소화해야 하는 각각의 함수가 존재하는데 이는 각각 다음과 같은 식(19), 식(20)과 같다.

$$\text{minimize } E_{x \sim p_{\text{data}}(x)} [(D(G(x)) - 1)^2] \cdot \cdot \cdot (19)$$

$$\text{minimize } E_{y \sim p_{\text{data}}(y)} [(D(y) - 1)^2] + E_{x \sim p_{\text{data}}(x)} [D(G(x))^2] \cdot \cdot \cdot (20)$$

둘째 모델 진동(model oscillation)을 줄이기 위해 최신 생성기에서 생산된 것이 아닌 생성된 이미지의 역사(history)를 사용하여 판별자를 업데이트한다.



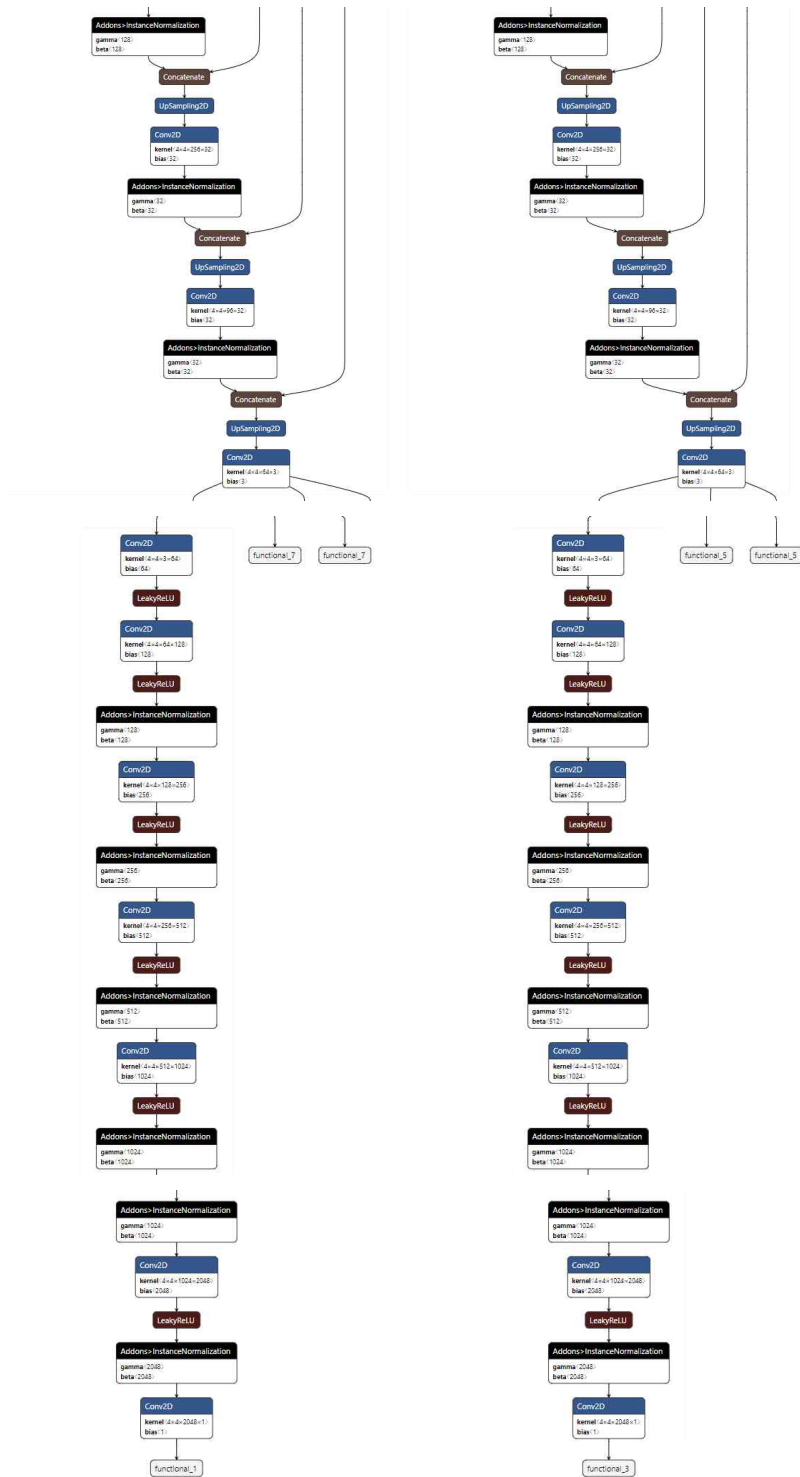


그림15. 네트워크 아키텍처

3-2. 구현 상세 코드

```
class CycleGAN():
    def __init__(self):

        self.img_rows = 256
        self.img_cols = 256
        self.channels = 3
        self.img_shape = (self.img_rows, self.img_cols, self.channels)

        self.dataset_name = 'vangogh2photo'

        self.data_loader = DataLoader(dataset_name=self.dataset_name,
                                       img_res=(self.img_rows, self.img_cols))

        patch = int(self.img_rows / 2**6)
        self.disc_patch = (patch, patch, 1)

        self.gf = 32
        self.df = 64

        # 손실 가중치
        self.lambda_cycle = 10.0
        self.lambda_id = 0.9 * self.lambda_cycle

        self.optimizer_lr = 0.0002
        self.optimizer = Adam(self.optimizer_lr, 0.5)

        self.d_A = self.build_discriminator()
        self.d_B = self.build_discriminator()
        self.d_A.compile(loss='mse',
                        optimizer=self.optimizer,
                        metrics=['accuracy'])
        self.d_B.compile(loss='mse',
                        optimizer=self.optimizer,
                        metrics=['accuracy'])

        self.g_AB = self.build_generator()
        self.g_BA = self.build_generator()

        img_A = Input(shape=self.img_shape)
        img_B = Input(shape=self.img_shape)

        fake_B = self.g_AB(img_A)
        fake_A = self.g_BA(img_B)

        reconstr_A = self.g_BA(fake_B)
        reconstr_B = self.g_AB(fake_A)

        img_A_id = self.g_BA(img_A)
        img_B_id = self.g_AB(img_B)

        self.d_A.trainable = False
        self.d_B.trainable = False

        valid_A = self.d_A(fake_A)
        valid_B = self.d_B(fake_B)

        self.combined = Model(inputs=[img_A, img_B],
                              outputs=[valid_A, valid_B,
                                       reconstr_A, reconstr_B,
                                       img_A_id, img_B_id])

        self.combined.compile(loss=['mse', 'mse',
                                    'mae', 'mae',
                                    'mae', 'mae'],
                              loss_weights=[1, 1,
                                             self.lambda_cycle, self.lambda_cycle,
                                             self.lambda_id, self.lambda_id],
                              optimizer=self.optimizer)
```

CycleGAN 클래스의 `__init__`을 통해 CycleGAN 훈련 시 사용할 여러 가지 인스턴스 변수들을 저장한다. 여기서 `self.gf`와 `self.df`는 각각 생성자와 판별자의 첫 번째 층에 있는 필터의 개수를 의미한다. `self.d_A.trainable`, `self.d_B.trainable`를 `False`로 지정한 이유는 생성자와 판별자가 동시에 훈련하는 것이 아닌 번갈아가면서 훈련하도록 설정하려고 했기 때문이다.


```

class CycleGAN(CycleGAN):
    @staticmethod
    def conv2d(layer_input, filters, f_size=4, normalization=True):
        """다운샘플링하는 동안 사용되는 층"""
        d = Conv2D(filters, kernel_size=f_size,
                    strides=2, padding='same')(layer_input)
        d = LeakyReLU(alpha=0.2)(d)
        if normalization:
            d = InstanceNormalization()(d)
        return d

    @staticmethod
    def deconv2d(layer_input, skip_input, filters, f_size=4, dropout_rate=0):
        """업샘플링하는 동안 사용되는 층"""
        u = UpSampling2D(size=2)(layer_input)
        u = Conv2D(filters, kernel_size=f_size, strides=1,
                    padding='same', activation='relu')(u)
        if dropout_rate:
            u = Dropout(dropout_rate)(u)
        u = InstanceNormalization()(u)
        u = Concatenate()([u, skip_input])
        return u

```

conv2d를 정의해서 이미지를 다운샘플링 할 때 반복해서 사용되는 구조를 저장하여 필요할 때 마다 self.conv2d로 불러와서 사용할 수 있도록 한다. conv2d는 conv2d를 통해 이미지를 다운샘플링 할 때 반복해서 사용되는 구조를 저장하여 사용할 수 있도록 하는 역할을 한다. deconv2d는 deconv2d를 통해 이미지를 업샘플링 할 때 반복해서 사용되는 구조를 저장하여 사용할 수 있도록 하는 역할을 한다.

```

class CycleGAN(CycleGAN):
    def build_generator(self):
        """U-Net 생성자"""
        d0 = Input(shape=self.img_shape)

        d1 = self.conv2d(d0, self.gf)
        d2 = self.conv2d(d1, self.gf * 2)
        d3 = self.conv2d(d2, self.gf * 4)
        d4 = self.conv2d(d3, self.gf * 8)

        d5 = self.conv2d(d4, self.gf * 16)
        d6 = self.conv2d(d5, self.gf * 32)

        u1 = self.deconv2d(d6, d5, self.gf * 16)
        u2 = self.deconv2d(u1, d4, self.gf * 8)
        u3 = self.deconv2d(u2, d3, self.gf * 4)
        u4 = self.deconv2d(u3, d2, self.gf * 1)
        u5 = self.deconv2d(u4, d1, self.gf)

        u6 = UpSampling2D(size=2)(u5)
        output_img = Conv2D(self.channels, kernel_size=4,
                             strides=1, padding='same', activation='tanh')(u6)

        return Model(d0, output_img)

```

build_generator는 생성자를 만드는 함수이다. 이 때 U-net이라는 특별한 구조를 이용해 생성자를 만든다. 이미지를 받아서 새로운 이미지를 생성하는 Model(d0, output_img)을 반환한다.


```

class CycleGAN(CycleGAN):
    def build_discriminator(self):
        img = Input(shape=self.img_shape)

        d1 = self.conv2d(img, self.df, normalization=False)
        d2 = self.conv2d(d1, self.df * 2)
        d3 = self.conv2d(d2, self.df * 4)
        d4 = self.conv2d(d3, self.df * 8)
        d5 = self.conv2d(d4, self.df * 16)
        d6 = self.conv2d(d5, self.df * 32)

        validity = Conv2D(1, kernel_size=4, strides=1, padding='same')(d6)

        return Model(img, validity)

```

build_discriminaor는 판별자를 생성해 내는 함수이다. 이 함수는 이미지를 받아 이전에 선언했던 conv2d함수를 사용해서 이미지의 출력으로 반환하는 판별자 모델이다.

```

class CycleGAN(CycleGAN):
    def sample_images(self, epoch, batch_i):
        r, c = 2, 3

        imgs_A = self.data_loader.load_data(domain="A", batch_size=1, is_testing=True)
        imgs_B = self.data_loader.load_data(domain="B", batch_size=1, is_testing=True)

        fake_B = self.g_AB.predict(imgs_A)
        fake_A = self.g_BA.predict(imgs_B)

        reconstr_A = self.g_BA.predict(fake_B)
        reconstr_B = self.g_AB.predict(fake_A)

        gen_imgs = np.concatenate([imgs_A, fake_B, reconstr_A, imgs_B, fake_A, reconstr_B])

        gen_imgs = 0.5 * gen_imgs + 0.5

        titles = ['Original', 'Translated', 'Reconstructed']
        fig, axs = plt.subplots(r, c)
        cnt = 0
        for i in range(r):
            for j in range(c):
                axs[i,j].imshow(gen_imgs[cnt])
                axs[i, j].set_title(titles[j])
                axs[i,j].axis('off')
                cnt += 1
        fig.savefig("images/%s/%d_%d.png" % (self.dataset_name, epoch, batch_i))
        plt.show()

```

sample_images는 훈련이 진행되는 동안 훈련 상황을 예시를 통해 보여준다. 즉 원본 이미지를 다른 도메인으로 변환한 뒤 다시 원본으로 되돌리는 함수이다. 여기서 gen_imgs는 모든 이미지를 정규화(Normalization)한 뒤에 저장한다.

```

class CycleGAN(CycleGAN):
    def train(self, epochs, batch_size=1, sample_interval=50):

        valid = np.ones((batch_size,) + self.disc_patch)
        fake = np.zeros((batch_size,) + self.disc_patch)

        for epoch in range(epochs):
            for batch_i, (imgs_A, imgs_B) in enumerate(self.data_loader.load_batch(batch_size)):

                fake_B = self.g_AB.predict(imgs_A)
                fake_A = self.g_BA.predict(imgs_B)

                dA_loss_real = self.d_A.train_on_batch(imgs_A, valid)
                dA_loss_fake = self.d_A.train_on_batch(fake_A, fake)
                dA_loss = 0.5 * np.add(dA_loss_real, dA_loss_fake)

                dB_loss_real = self.d_B.train_on_batch(imgs_B, valid)
                dB_loss_fake = self.d_B.train_on_batch(fake_B, fake)
                dB_loss = 0.5 * np.add(dB_loss_real, dB_loss_fake)

                d_loss = 0.5 * np.add(dA_loss, dB_loss)

                g_loss = self.combined.train_on_batch([imgs_A, imgs_B],
                                                    [valid, valid,
                                                     imgs_A, imgs_B,
                                                     imgs_A, imgs_B])

                if batch_i % sample_interval == 0:
                    self.sample_images(epoch, batch_i)

```

train은 생성한 이미지로 훈련을 진행하는 함수이다. 판별자의 훈련을 진행하면서 나온, 손실의 평균을 d_loss에 저장한다. 또한 생성자의 훈련을 진행하면서 나온 손실을 g_loss에 저장한다.

3-3. 구현 결과

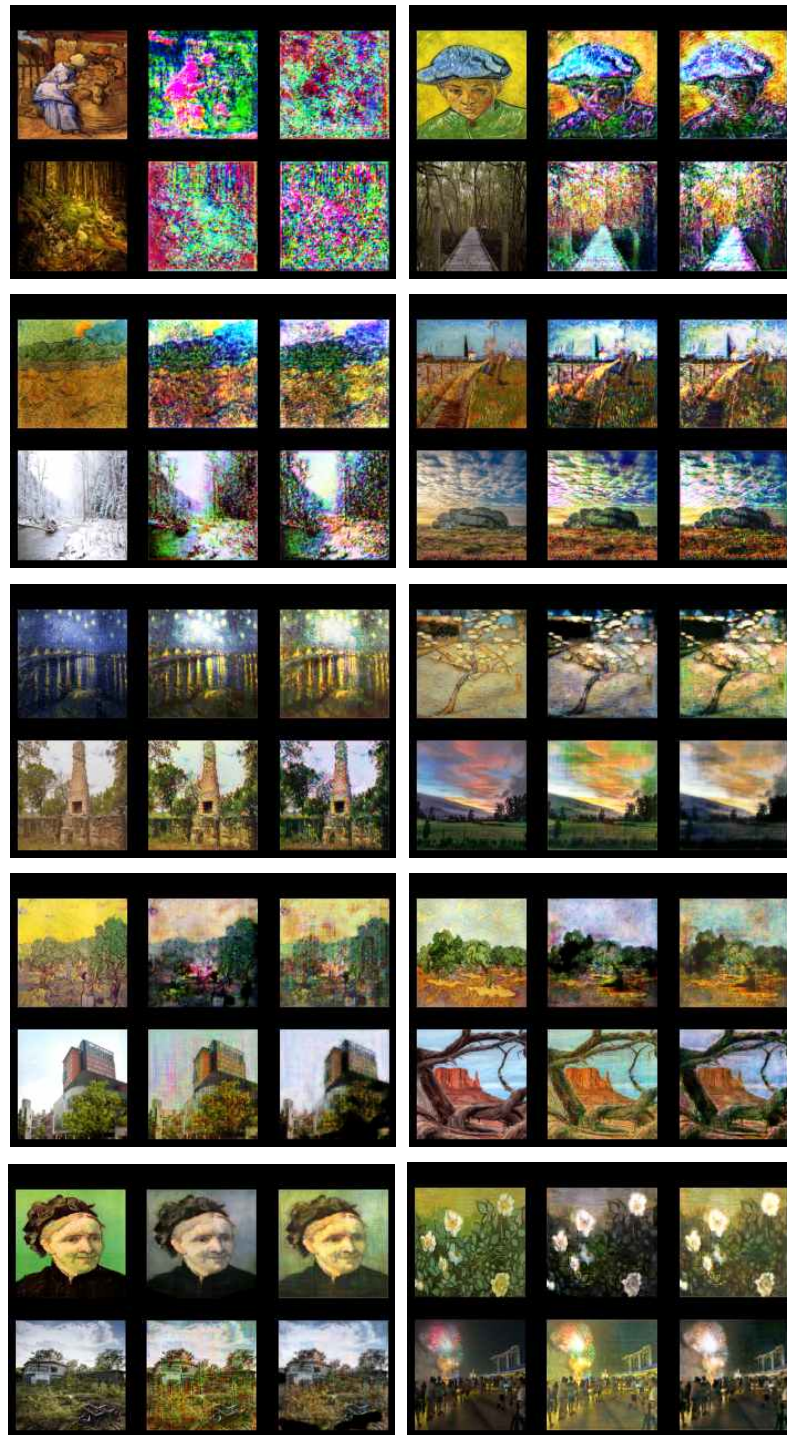


그림16. CycleGAN 학습과정

왼쪽 순서대로 원본 이미지(Original), 변환 이미지(Translated), 재구성된 이미지(Reconstructed)이다. 학습이 진행되면서 이미지 변환 성능이 점점 좋아지는 것을 확인할 수 있다.



그림17. CycleGAN 결과 이미지

Ⅲ. 결론

1. 고찰

GAN은 현재 많은 분야에 사용되고 있는 기술이다. 예를 들면 낡은 옛날 사진을 복원해 준다든지, 흑백 사진에 채색을 해준다든지, 심지어는 눈을 감고 있는 사진을 눈을 뜬 것처럼 바꿀 수도 있다. 최근에는 음성 등 자연어 처리에도 많이 이용되고 있다.

이렇게 성능이 좋아지면서 딥페이크가 주목을 받고 있다. 딥페이크는 딥러닝과 페이크의 합성어로, 허위의 동영상 콘텐츠를 만들거나 변형하는데 사용되는 AI기반 기술을 의미한다. 예시로 영국의 한 회사가 데이비드 벅의 목소리를 9개 언어로 구현해 말라리아 퇴치 캠페인을 홍보한 적이 있고, 단편 영화 <선스프링>은 AI 생성 모델을 활용해 구현했다. 미디어 제작을 위한 배우 섭외, 장소섭외, 촬영 등이 없이도 가능해져 비용 절감 효과까지 기대할 수 있게 된 것이다.

그러나 이와 동시에 악의적인 활용에 대한 우려가 커지고 있다. AI가 생성해낸 가짜뉴스, 합성된 가짜 정치적 선언, 혹은 당사자 모르게 얼굴이 합성된 포르노 등 사람들에게 혼란을 빚거나, 범죄에 이용될 수 있는 사례들이 생겨났다. 실제로 한 연구팀에 딥페이크의 예시로 오바마 전 미국 대통령의 정치적 발언 영상을 만들고 공개해 화제가 된 적이 있다. 우리는 결과물의 이용과 활용 방향에 주의하여, GAN의 기술이 악의적으로 사용되지 않도록 노력해야 할 것이다.

이번 연구를 통해 CycleGAN이란 모델을 가지고 여러 가지 방법으로 훈련시켜보고 결과를 볼 수 없었다는 점이 매우 아쉬웠다. 우선 논문에서 소개된 학습률을 줄이며 학습시키는 과정을 시도해 보았지만 학부생 수준으로 구현하기 힘들었던 것 같다. 추가로 더 높은 화질의 이미지나 영상을 가지고 모델 내부의 층을 쌓아가거나 줄여가고, 여러 가지 파라미터 튜닝을 해보며 더 훈련을 해보고 싶었지만 연산을 할 GPU의 성능과 주어진 시간이 충분하지 못해 해보지 못한 것이 아쉬웠다. 또한 어떤 화가의 화풍을 그림에 입힐지 고민하는 과정에서 화가의 특징이 강하게 드러나는 고흐의 그림으로 학습을 진행하는 것으로 결정하는데 고흐의 작품이 다른 화가들에 비해 데이터가 적어서 학습이 잘 진행되지 않았던 것 같다.

IV. 참고 문헌

- [1] Vukosi N. Marivate, Fulufhelo V. Nelwamodo, Tshilidzi Marwala. **Autoencoder, Principal Component Analysis and Support Vector Regression for Data Imputation**
- [2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. **Generative Adversarial Nets**
- [3] Alec Radford, Luke Metz, Soumith Chintala. **UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS**
- [4] Tatjana Chavdarova, Francois Fleuret. **SGAN: An Alternative Training of Generative Adversarial Networks**
- [5] Mehdi Mirza, Simon Osindero. **Conditional Generative Adversarial Nets**
- [6] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. **Image-to-Image Translation with Conditional Adversarial Networks**
- [7] Olaf Ronneberger, Philipp Fischer, Thomas Brox. **U-Net: Convolutional Networks for Biomedical Image Segmentation**
- [8] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros. **Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.**