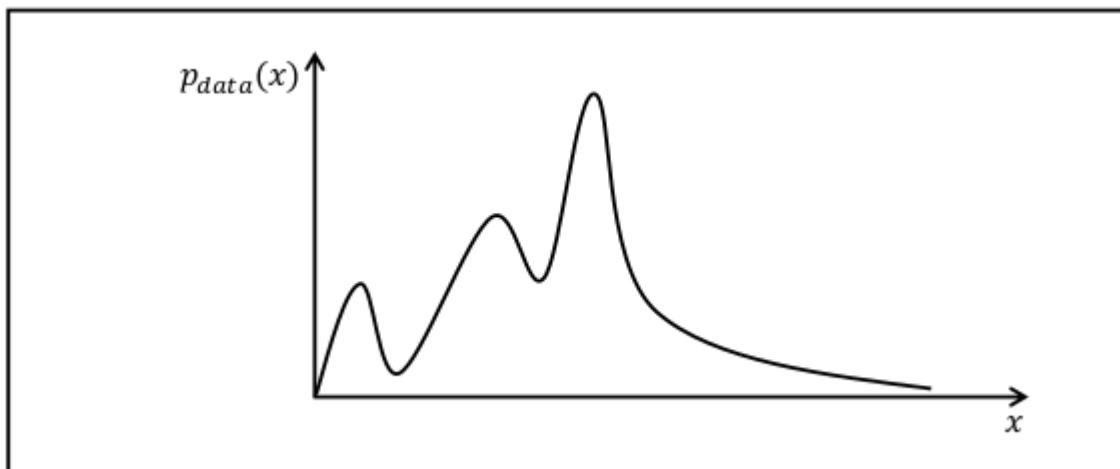


1시간 만에 GAN(Generative Adversarial Network) 완전 정복하기

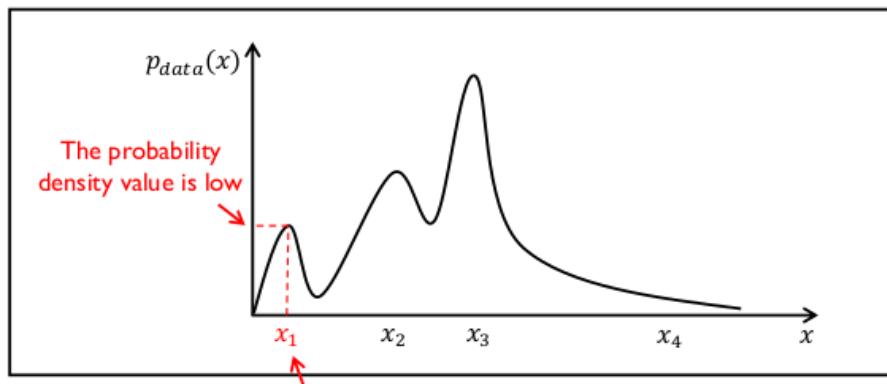
1시간 만에 GAN(Generative Adversarial Network) 완전 정복하기 (https://www.youtube.com/watch?v=odpj7_tGY0&t=2839s) 의 노트입니다.

Probability Distribution (확률 분포)

실제 이미지에 대한 확률 분포(확률밀도함수)가 존재할 때 아래의 분포를 가지는다고 가정한다면,



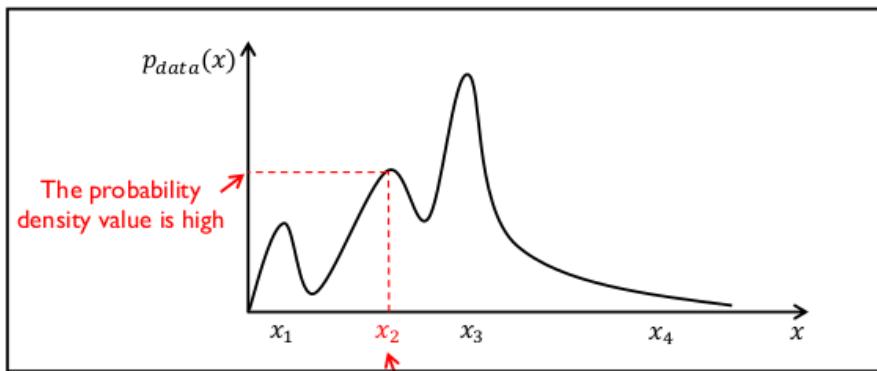
Let's take an example with human face image dataset.
Our dataset may contain few images of **men with glasses**.



x_1 is a $64 \times 64 \times 3$ high dimensional vector
representing a man with glasses.

사람 얼굴 데이터 셋에서 안경을 낀 남자가 좀 상대적으로 적게 등장했다고 하면, 그 안경을 낀 남자에 해당하는 픽셀을 대표하는 가스을 x_1 의 어떤 벡터라고 했을 때 그 x_1 에 해당하는 확률 밀도 가스은 상대적으로 작게 됩니다.

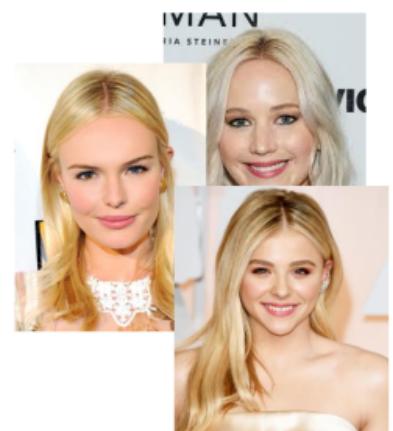
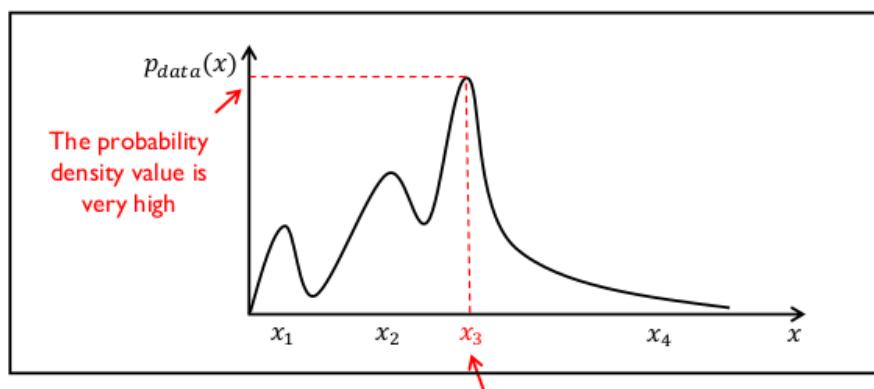
Our dataset may contain many images of women with black hair.



x_2 is a $64 \times 64 \times 3$ high dimensional vector representing a woman with black hair.

흑발의 여자는 안경을 낀 남자보다 학습데이터에서 비교적 많이 등장을 했다면, 흑발의 여자 이미지에 해당하는 확률 밀도 가스은 상대적으로 높게 됩니다.

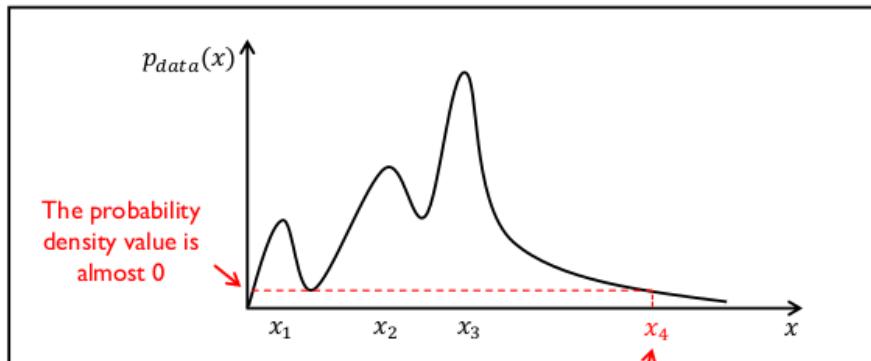
Our dataset may contain very many images of women with blonde hair.



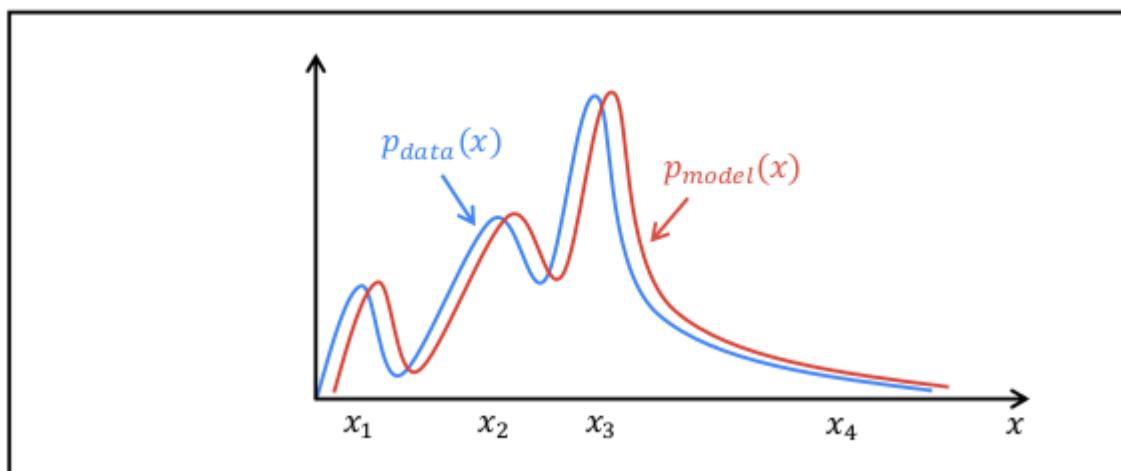
x_3 is a $64 \times 64 \times 3$ high dimensional vector representing a woman with blonde hair.

마찬가지로 금발의 여자 데이터가 많이 나왔다면 금발의 여자 데이터에 해당하는 벡터를 x_3 라고 했을 때 x_3 에 대한 확률 밀도가스은 상당히 높게됩니다. (예를 들어 1차원 상으로 설명되었지만, 실제로는 상당히 고차원입니다.)

Our dataset may not contain **these strange images**.



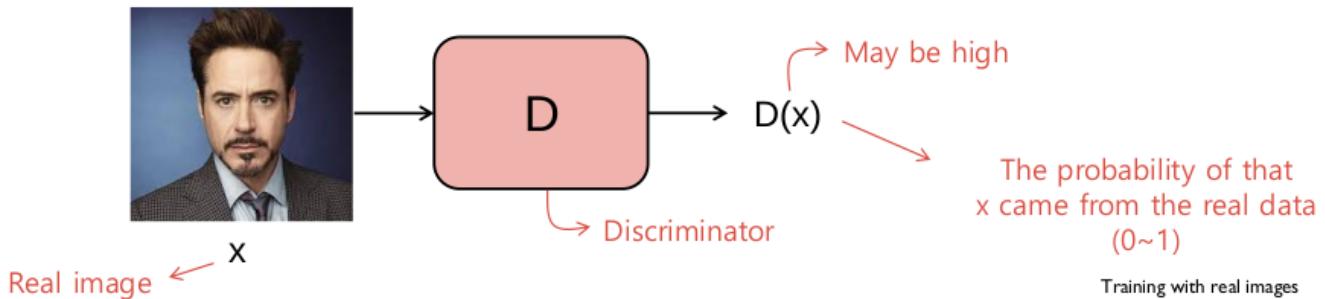
괴상한 이미지는 실제 학습 데이터에 존재할 수 없기 때문에 이를 x_4 라고 하면 이에 대한 확률밀도가 0도 굉장히 낮습니다. 이에 대한 $64 \times 64 \times 3$ 크기의 고차원 벡터는 픽셀가스이기 때문에 픽셀가스들을 잘 조정하다보면, 이런 괴상한 이미지를 만들어 낼 수 있게 됩니다.



Generative Model 이 하고자 하는 목표가 이 시점에서 나오게 되는데, 파란 색이 실제 학습 데이터 셋의 분포이며, Generative Model은 실제 이 데이터의 분포를 잘 근사하는 모델을 만들고자 합니다. 빨간색은 모델이 생성에 대한 이미지 데이터 분포라고 생각할 수 있습니다. 두개의 확률분포가 높은 차이를 줄여주는 게 이제 **Generative Model**의 목표입니다.

Generative Adversarial Networks (GAN)

Discriminator Model



진짜 이미지를 진짜 이미지로 구별하고 가짜 이미지를 가짜로 구별할 수 있도록 학습합니다.

- 입력 : 64 x 64 크기의 고차원 벡터
- 출력 : binary classification

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

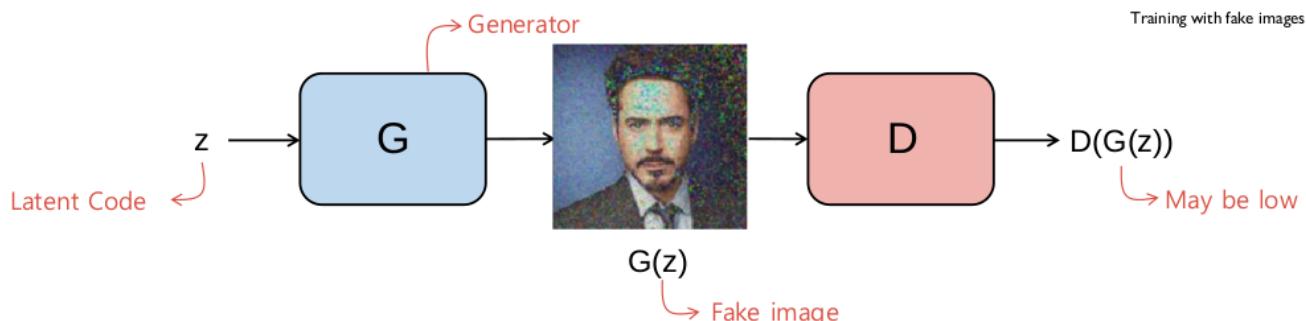
↑ ↑

Maximum when $D(x) = 1$ Maximum when $D(G(z)) = 0$

D should maximize $V(D, G)$

Sample x from real data distribution Sample latent code z from Gaussian distribution

Generator



랜덤한 코드를 입력 받아서 이미지를 생성하고 **Discriminator**를 속여야합니다. Discrimator의 출력이 1이 나오도록 학습을 시켜야합니다. Generator가 학습을 할 수록 진짜가짜은 가짜 이미지를 만들어 내게 됩니다.

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

↑ ↑

G is independent of this part Minimum when $D(G(z)) = 1$

G should minimize $V(D, G)$

Code

Define the discriminator

input size: 784
hidden size: 128
output size: 1

```
1 import torch
2 import torch.nn as nn
3
4
5 D = nn.Sequential(
6     nn.Linear(784, 128),
7     nn.ReLU(),
8     nn.Linear(128, 1),
9     nn.Sigmoid())
10
```

Define the generator

input size: 100
hidden size: 128
output size: 784

```
5 D = nn.Sequential(  
6     nn.Linear(784, 128),  
7     nn.ReLU(),  
8     nn.Linear(128, 1),  
9     nn.Sigmoid())  
10  
11 G = nn.Sequential(  
12     nn.Linear(100, 128),  
13     nn.ReLU(),  
14     nn.Linear(128, 784),  
15     nn.Tanh())  
16
```

```
11 G = nn.Sequential(  
12     nn.Linear(100, 128),  
13     nn.ReLU(),  
14     nn.Linear(128, 784),  
15     nn.Tanh())  
16  
17 criterion = nn.BCELoss()
```

Optimizer for D and G

```
16  
17     criterion = nn.BCELoss()  
18  
19     d_optimizer = torch.optim.Adam(D.parameters(), lr=0.01)  
20     g_optimizer = torch.optim.Adam(G.parameters(), lr=0.01)
```

```
22 # Assume x be real images of shape (batch_size, 784)  
23 # Assume z be random noise of shape (batch_size, 100)
```

```
24  
25     while True:  
26         # train D  
27         loss = criterion(D(x), 1) + criterion(D(G(z)), 0)  
28         loss.backward()  
29         d_optimizer.step()
```

Forward, Backward and Gradient Descent

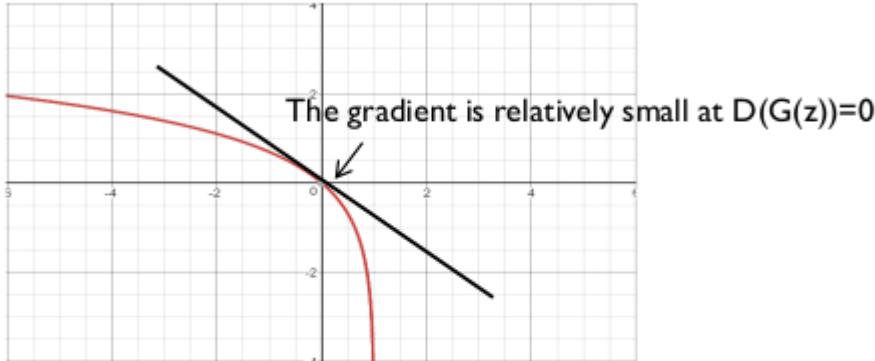
Forward, Backward and Gradient Descent →

```
31      # train G  
32      loss = criterion(D(G(z)), 1)  
33      loss.backward()  
34      g_optimizer.step()
```

실제로 돌아가는 코드는 <https://github.com/yunjey/pytorch-tutorial> 레포지토리에 있습니다.

Non-Saturating Game

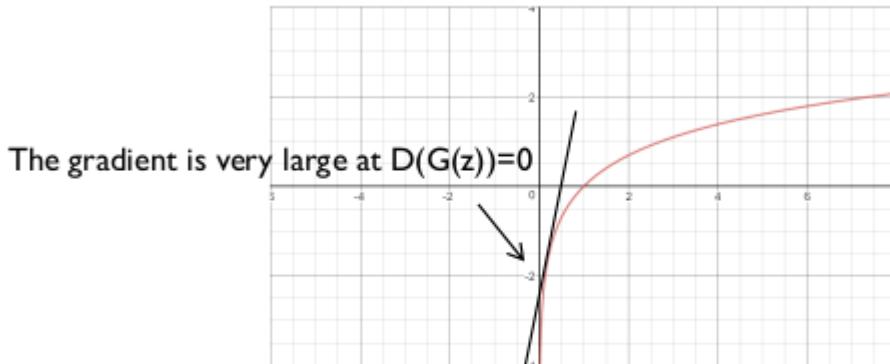
Generator는 처음에는 매우 형편없는 이미지를 만들게 됩니다. Discreminator의 입장에서는 이런 형편없는 이미지에 대해서 가짜임을 확신하게 됩니다. 그렇다면, $D(G(z))$ 의 가수이 0에 가깝게 된다. 그런데 그때의 기울기가 생가보다 작습니다. (아래의 그래프를 참고합니다.)



$$y = \log(1 - x)$$

그래서 $\log(1 - x)$ 를 최소화하는 것이 아니라 $\log(x)$ 를 최대화하는 방향으로 바꾸게 됩니다. (Heuristically motivated)

$$\begin{array}{c} \cancel{\min_G E_{z \sim p_z(z)} [\log(1 - D(G(z))]} \\ \downarrow \text{Modification (heuristically motivated)} \\ \max_G E_{z \sim p_z(z)} [\log D(G(z))] \end{array}$$



$$y = \log(x)$$

기울기가 무한대로 나오게 됩니다. Generator 의 결과물이 안좋은 상황을 최대한 빨리 벗어나려고 노력하려고 해서 Discreminator가 햇가르려할만한 데이터를 생성해내려고 하게 됩니다.

$$\min_G E_{z \sim p_z(z)}[-y \log D(G(z)) - (1-y) \log(1 - D(G(z)))]$$

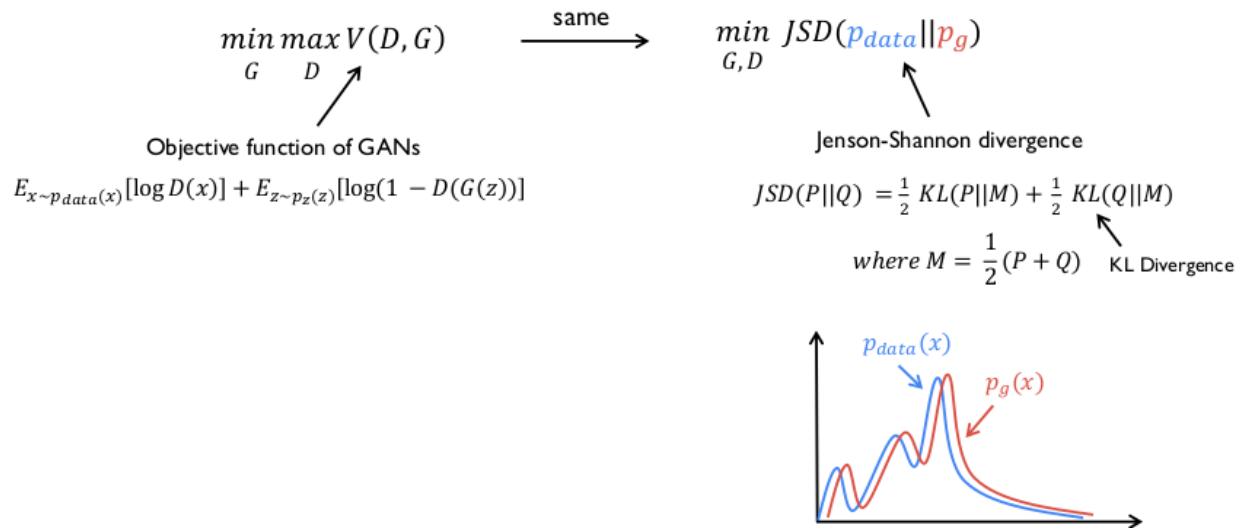
$$\downarrow \quad y = 1$$

$$\min_G E_{z \sim p_z(z)}[-\log D(G(z))]$$

Theory in GAN

Why does GANs work?

Because it actually minimizes the distance between the **real data distribution** and the **model distribution**.

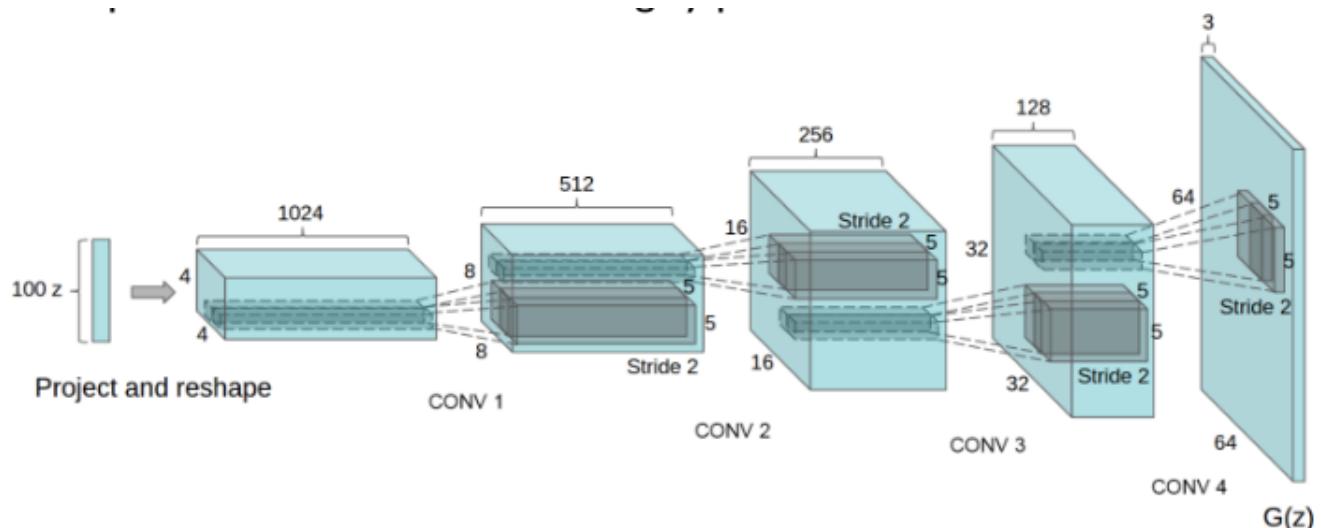


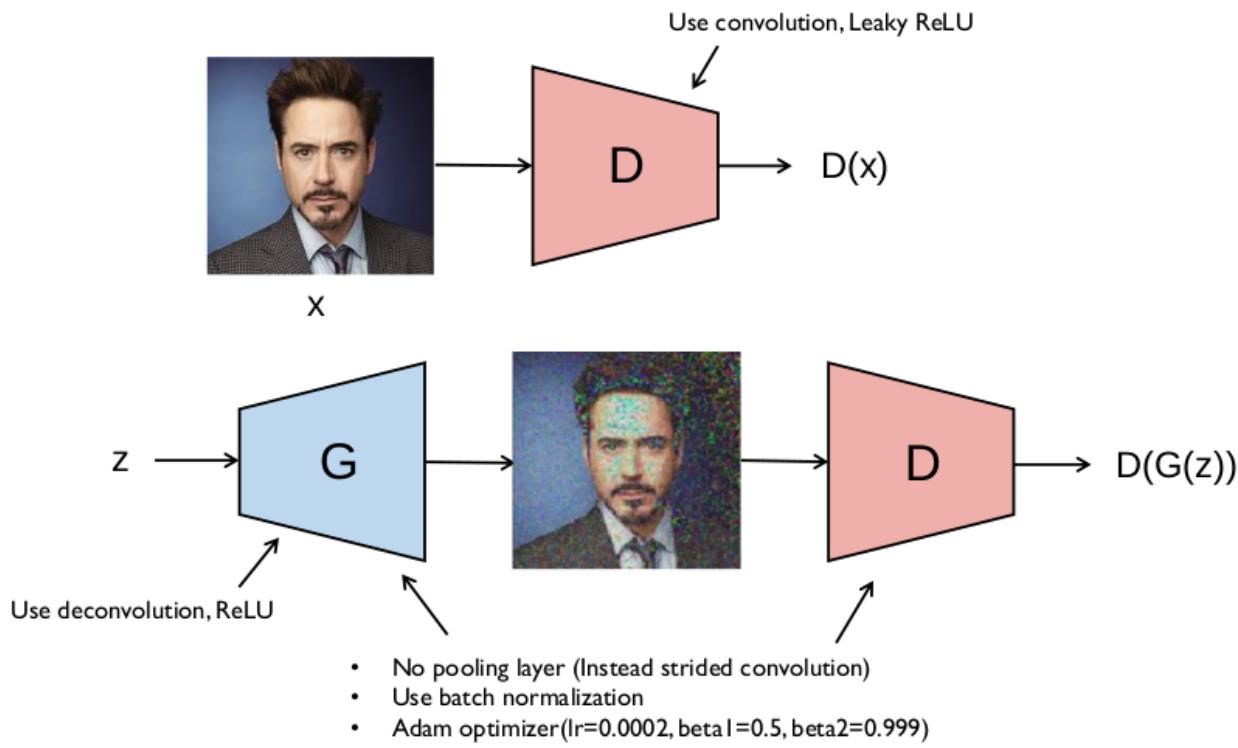
이론적으로 왜 GAN이 잘 작동하는지 : 최적화하는 것은 실제로 두 확률분포 간의 차이를 줄여주는 것이기 때문에 Generator가 진짜와 가까운 이미지를 만들 수 있다.

Variant of GAN

DCGAN (Deep Convolutional GAN, 2015)

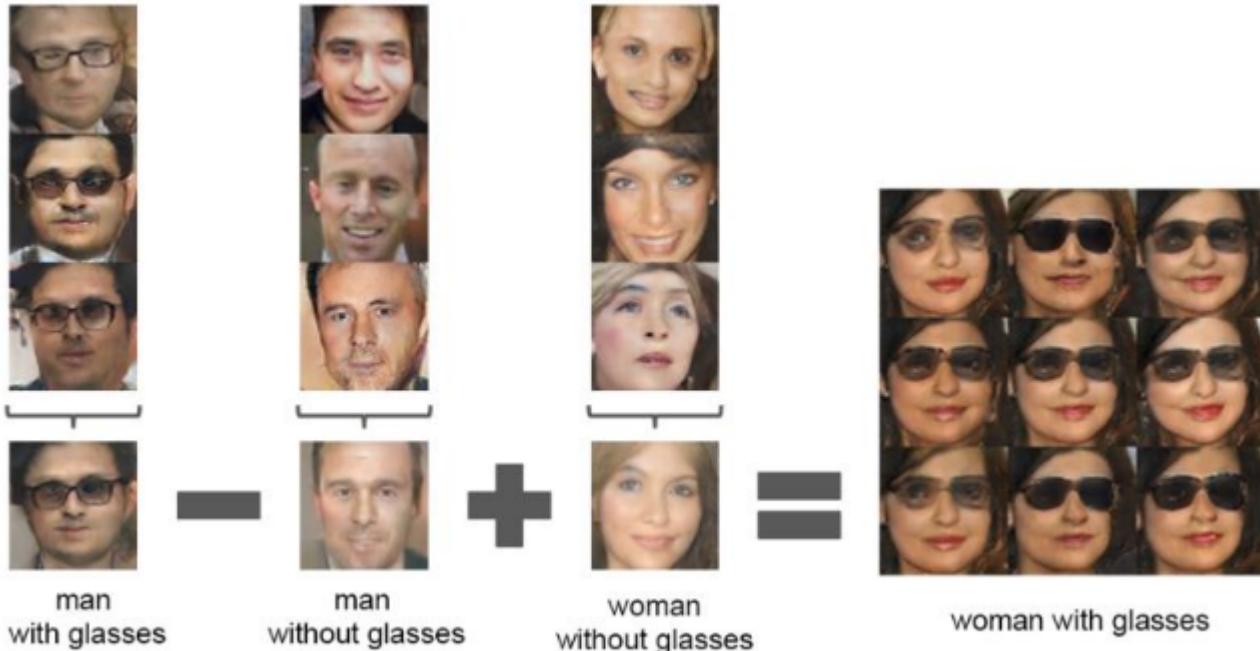
Deep Convolutional Network를 통해서 Generator를 만들어냈다.





DCGAN에서의 핵심은 **Pooling Layer**를 사용하지 않았다는 것이다. Pooling Layer를 사용하게 되면 unpooling을 할 때의 결과물이 blocky하게 된다. 학습을 안정화시키기 위해서 Batch normalization을 적용하였으며, Adam optimizer를 사용하였다.

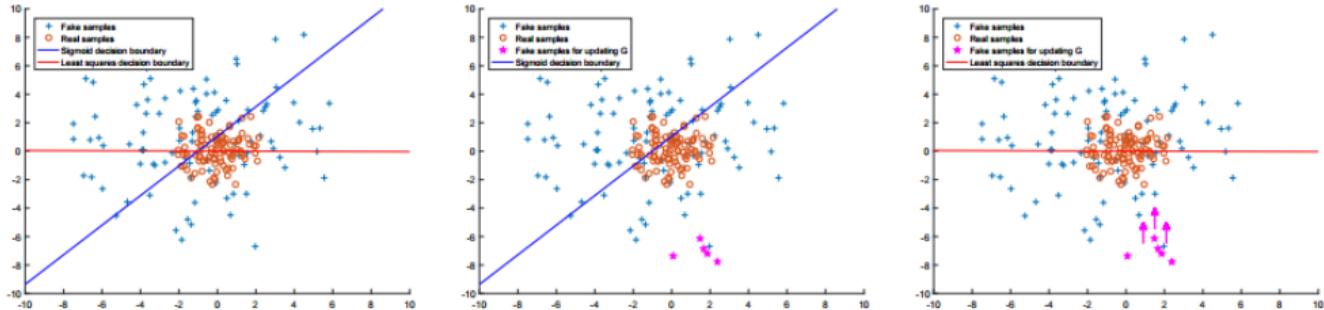
64 x 64크기의 이미지를 만들 때 보통 4개의 Convolutional Layer를 이용해서 만들게 되는데, 저 term들의 결과가 좋게 나온다. 다른 논문들에서도 저 가ㅅ을 고정적으로 사용한다.



DCGAN에서 하나 더 재밌는 포인트는, Generator의 인풋으로 들어가는 Latent vector의 산술적 연산이 가능하다는 점이다.

LSGAN (Least Squares GAN)

기존 GAN가트은 경우는 Discreminator만 속이면 됐지만... **Gradient Vanishing**



파란 선이 Discriminator의 Decision boundary를 의미합니다. 선 아래쪽이 Discriminator가 True라고 예측한 이미지 데이터입니다. 빨간색 점이 진짜 데이터고 파란 점이 가짜 데이터입니다. 빨간색에 가까운 파란색은 굉장히 잘 만들어진 가짜 이미지입니다. 핑크색 데이터는 Discriminator를 완벽하게 속인 이미지 데이터입니다.

그러나, 핑크색 데이터가 좋은 데이터나?라고 했을 때는 그렇지 않습니다. 빨간색 점 데이터랑 가까운 데이터가 실제로 좋은 데이터이다. (Gradient Vanishing : Discriminator를 완벽해 속이더라도 실제 이미지 데이터랑 비슷하다고 보장할 수 없다.)

Vanilla GAN

```

1 D = nn.Sequential(
2     nn.Linear(784, 128),
3     nn.ReLU(),
4     nn.Linear(128, 1),
5     nn.Sigmoid())

```

Remove sigmoid
non-linearity
in last layer

```

1 D = nn.Sequential(
2     nn.Linear(784, 128),
3     nn.ReLU(),
4     nn.Linear(128, 1))

```

```

7 G = nn.Sequential(
8     nn.Linear(100, 128),
9     nn.ReLU(),
10    nn.Linear(128, 784),
11    nn.Tanh())

```

Generator is the
same as original

```

7 G = nn.Sequential(
8     nn.Linear(100, 128),
9     nn.ReLU(),
10    nn.Linear(128, 784),
11    nn.Tanh())

```

```

7 G = nn.Sequential(
8     nn.Linear(100, 128),
9     nn.ReLU(),
10    nn.Linear(128, 784),
11    nn.Tanh())
12
13 # Loss of D
14 D_loss = - torch.mean(torch.log(D(x))) - torch.mean(torch.log(1 - D(G(z))))
```

Replace cross entropy loss
to least squares loss (L2)

```

7 G = nn.Sequential(
8     nn.Linear(100, 128),
9     nn.ReLU(),
10    nn.Linear(128, 784),
11    nn.Tanh())
12
13 # Loss of D
14 D_loss = torch.mean((D(x) - 1)**2) + torch.mean(D(G(z))**2)
```

D(x) gets closer to 1
D(G(z)) gets closer to 0
(same as original)

```

11     nn.Tanh()
12
13 # Loss of D
14 D_loss = - torch.mean(torch.log(D(x))) - torch.mean(torch.log(1 - D(G(z))))
15
16 # Loss of G
17 G_loss = - torch.mean(torch.log(D(G(z))))
```

Replace cross entropy loss
to least squares loss (L2)

```

11     nn.Tanh()
12
13 # Loss of D
14 D_loss = torch.mean((D(x) - 1)**2) + torch.mean(D(G(z))**2)
15
16 # Loss of G
17 G_loss = torch.mean((D(G(z)) - 1)**2)
```

D(G(z)) gets closer to 1
(same as original)

LSGAN의 결과 (LSUN DATASET)



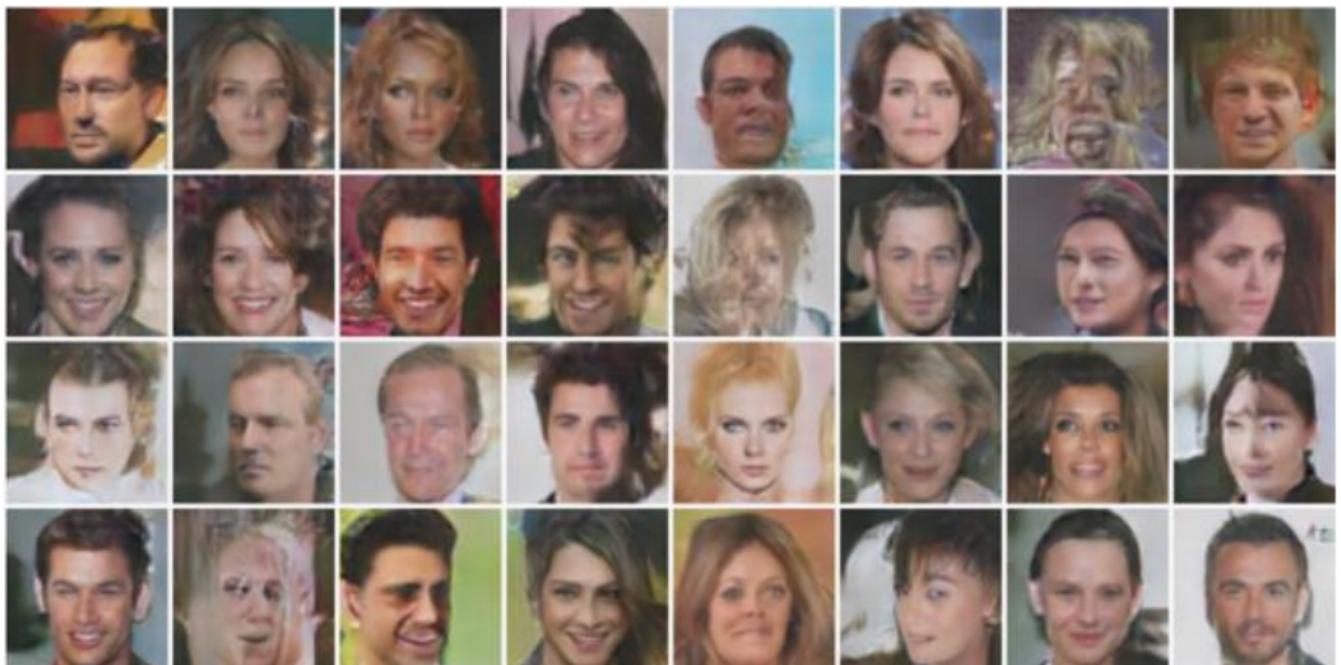
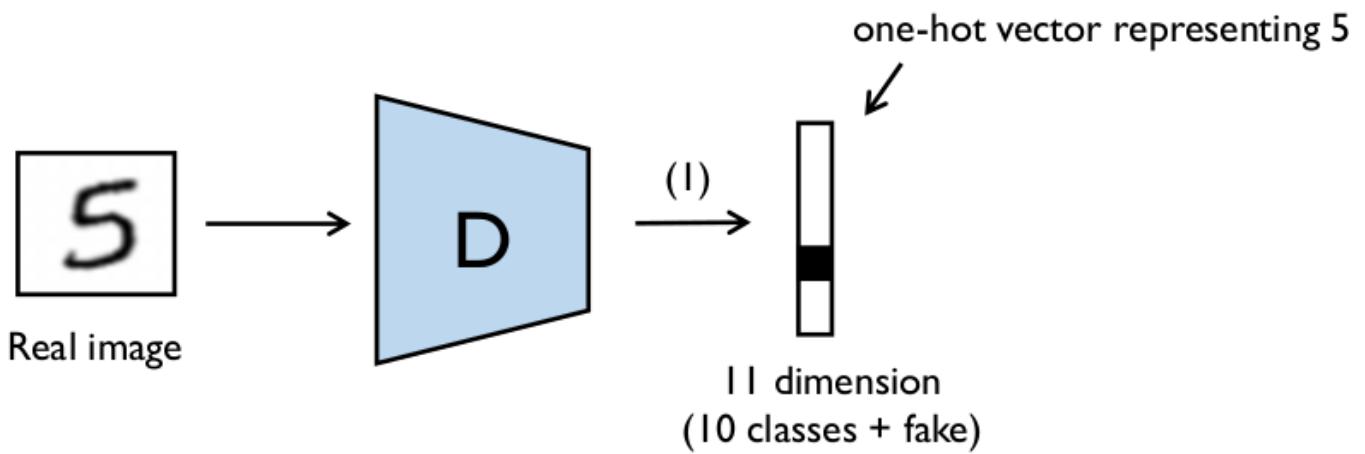
(a) Church outdoor.



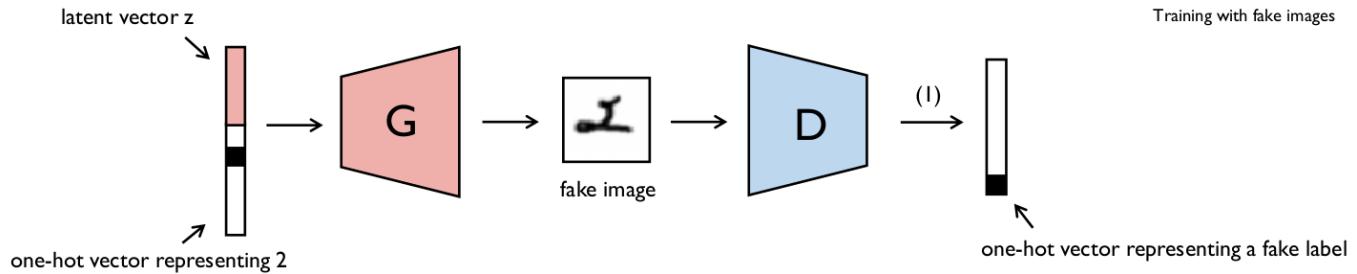
(b) Dining room.



(c) Kitchen.

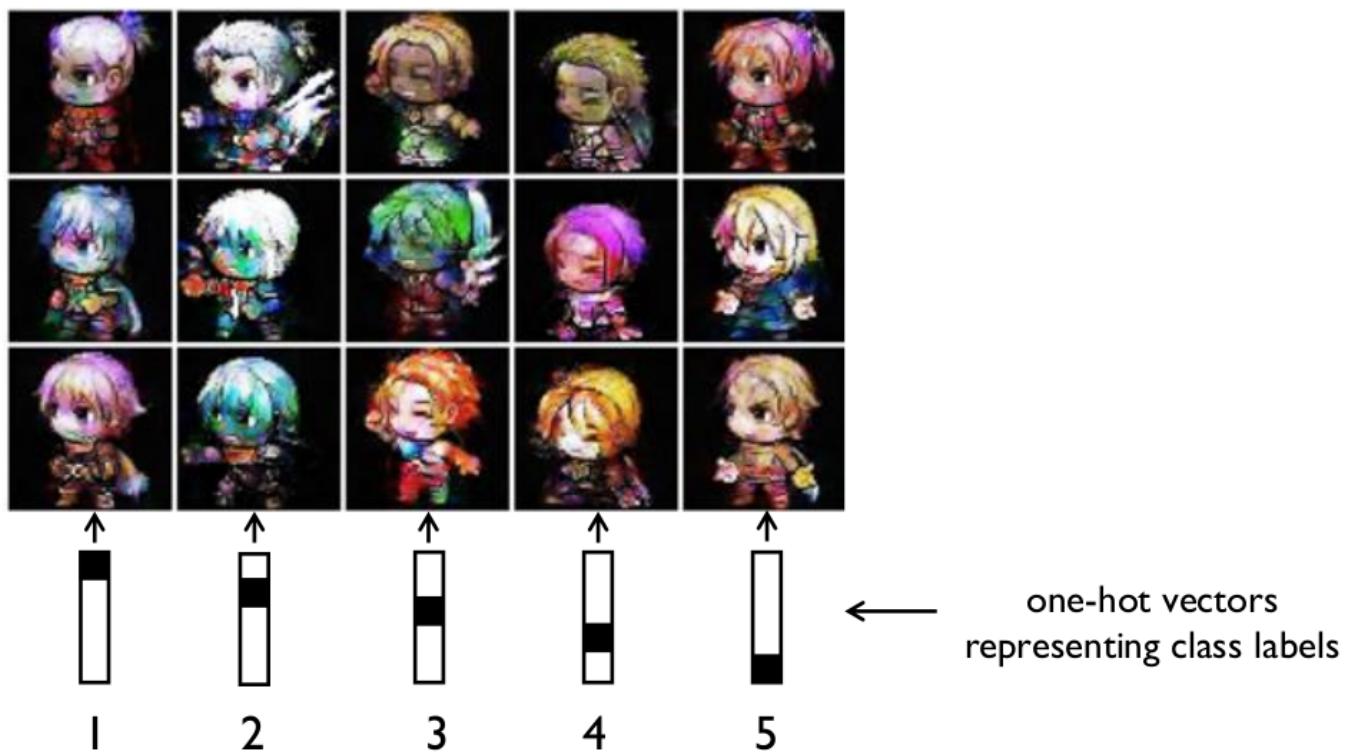
LSGAN의 결과 (CelebA)**SEMI-SUPERVISED GAN**

Discriminator가 더이상 진짜/가짜를 구별하는 것이 아니라 어떤 클래스를 구분하게 된다. 기존의 Supervised Learning의 경우 MNIST 데이터를 판별할 때 10개의 클래스로 판별하게 되지만, Semi-supervised GAN에서는 **10개 + Fake** 클래스로 판별하게 됩니다.



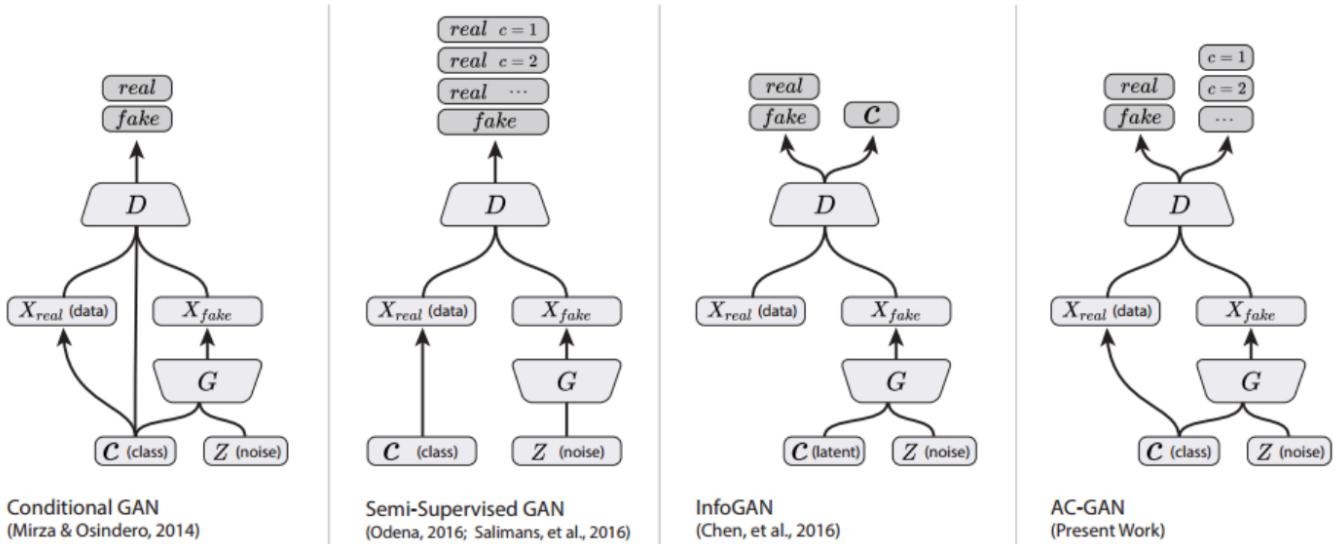
2를 나타내는 One-hot 벡터를 Generator에 입력을 하면 가짜이미지 2를 생성합니다. 이 이미지를 Discriminator는 Fake 클래스로 구분해내야 합니다. 반대로 Generator는 Discriminator를 속여야 하기 때문에 입력으로 준 One-hot 벡터와 Discriminator의 출력이 가느도록 학습을 진행합니다.

Result (Game character)



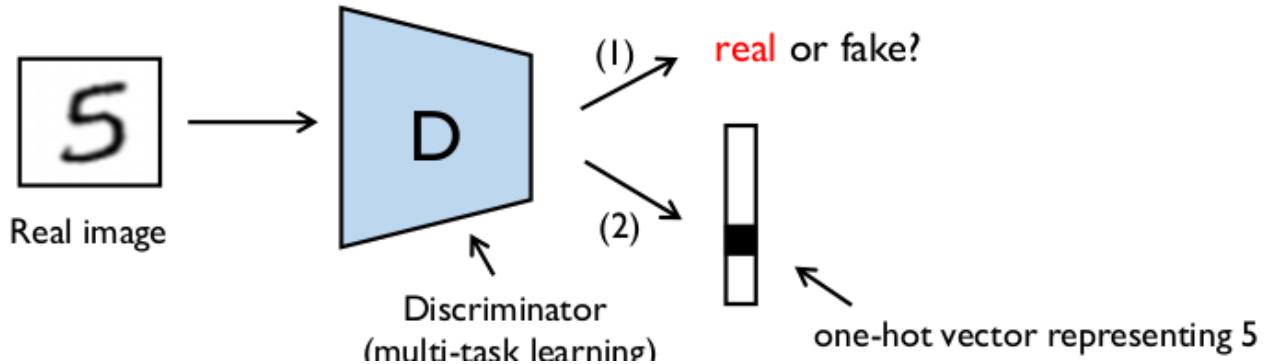
ACGAN (Auxiliary Classifier GAN, 2016)

Semi-supervised Learning에 ACGAN을 적용시킬 수 있지 않는가..



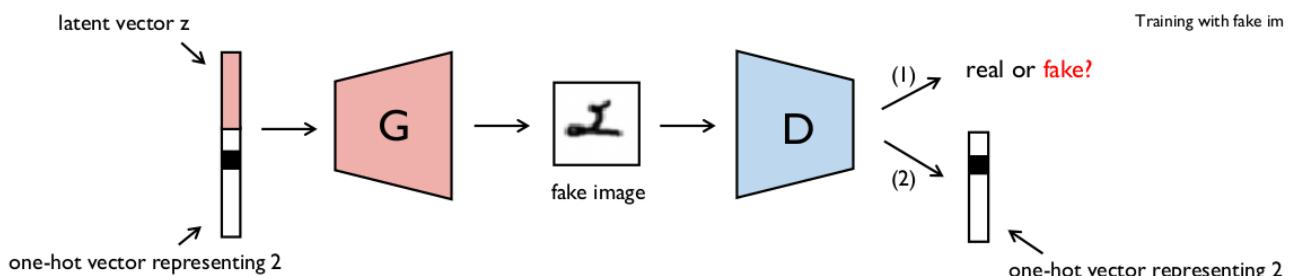
Discriminator가 해야하는 테스크가 2가지가 있습니다.

1. Training with real images



- 1. FC Layer with sigmoid : **real or fake?**
- 2. FC Layer with softmax : $0 \sim 9$

2. Training with fake images



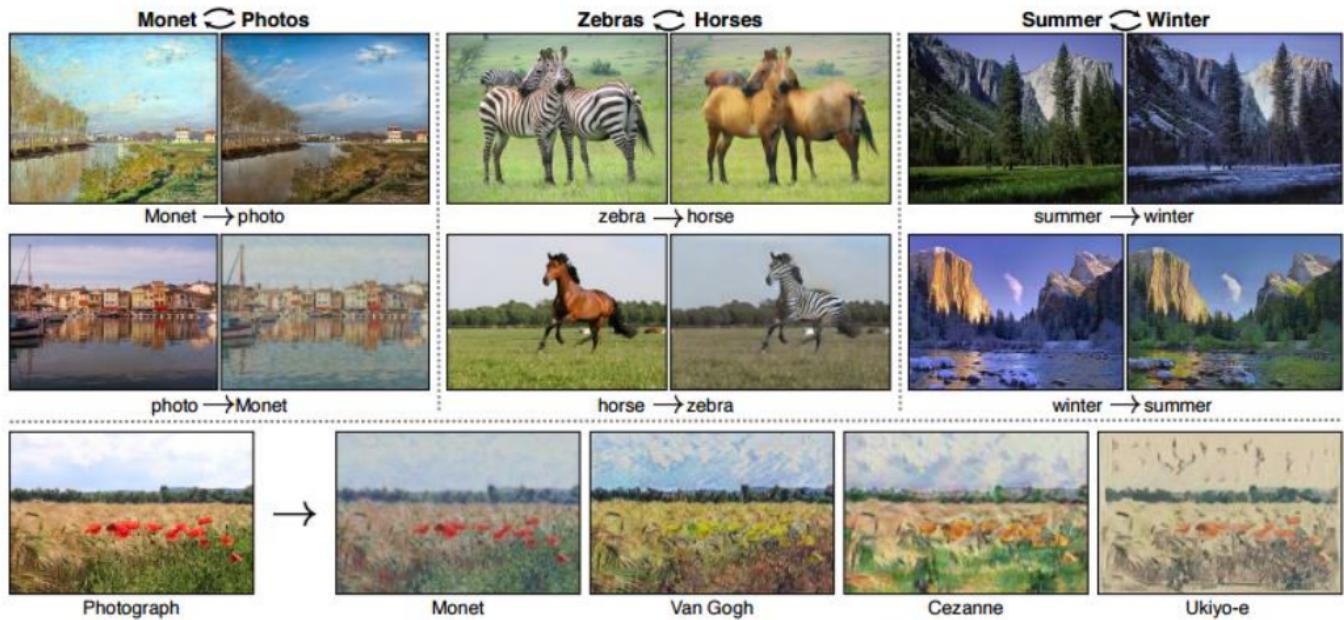
Generator에 2에 해당하는 one-hot 벡터를 입력하면 가짜 이미지를 생성해내고, Discreminator는 이 이미지를 가짜라고 판단해야 합니다. 그럼에도 불구하고 Discreminator는 가짜 이미지를 2로 분류해야 합니다. 여태까지의 GAN은 Generator에 집중을 했지만, ACGAN은 Discreminator에 집중을 한다. Generator는 Data Augmentation하는 역할을 하고 노이즈가 포함된 데이터를 입력하더라도 Classification이 되면 Discreminator의 성능이 좋아진게 아닌가.

3. Loss Function : loss of (1) + loss of (2)

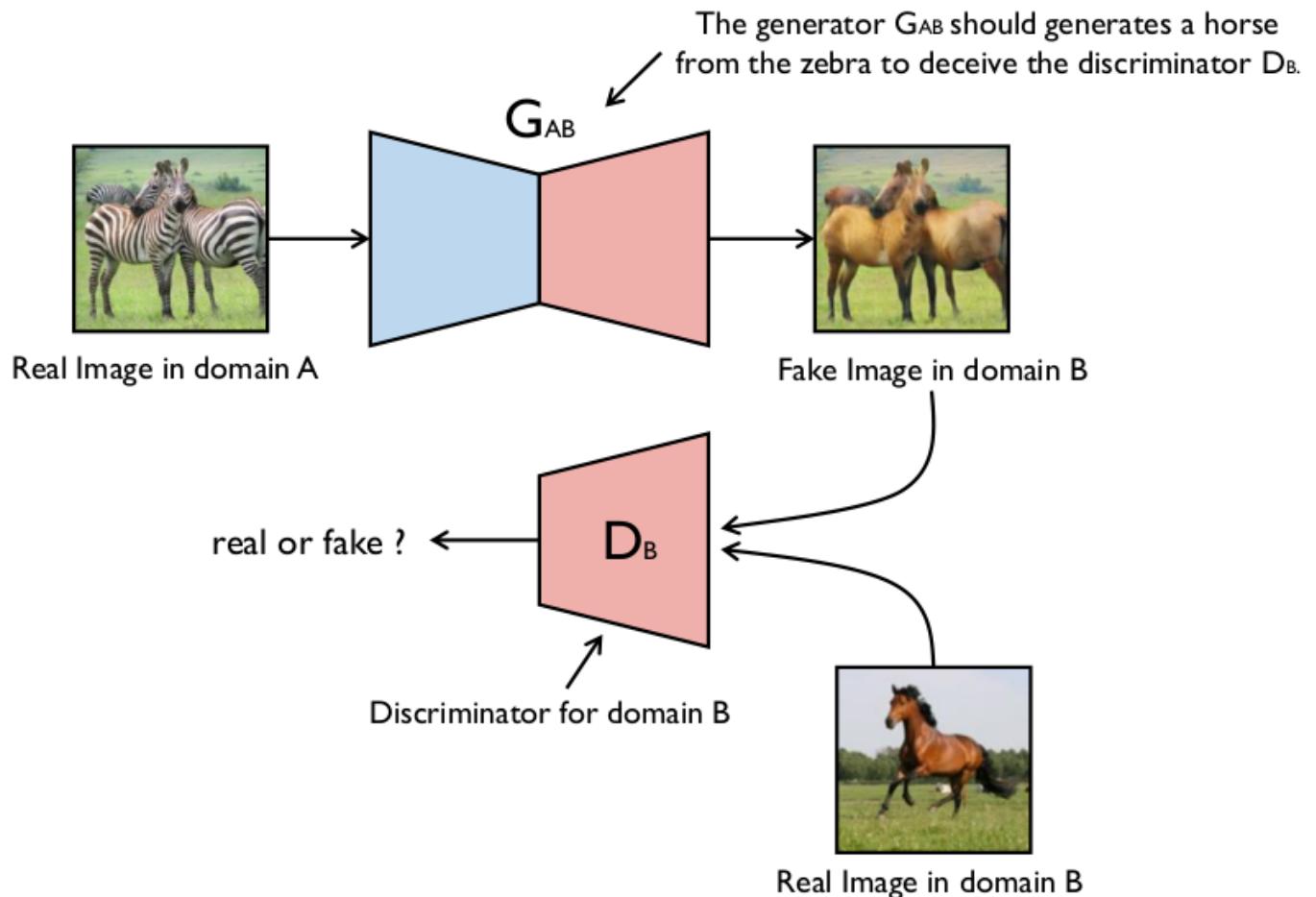
EXTENSIONS OF GAN

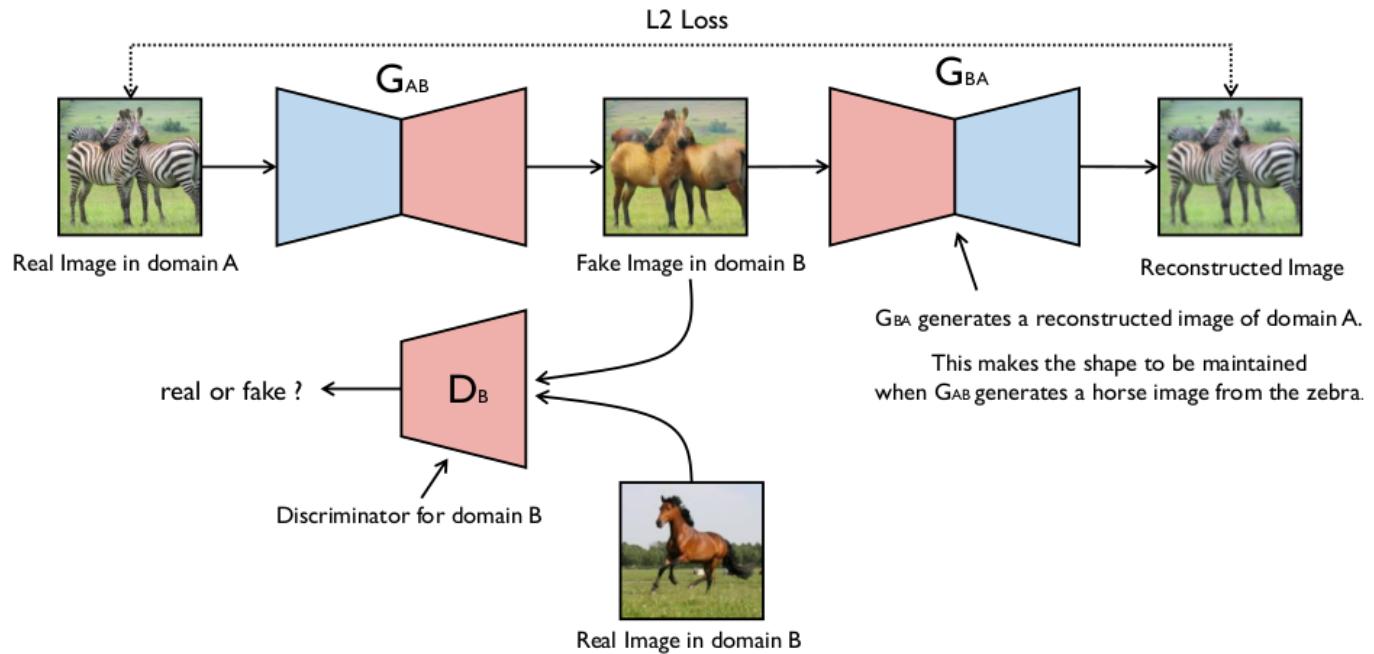
CycleGAN

얼룩말을 일반 말로 바꾸고, 거울 풍경을 여름풍경으로 바꾸고 .. 이미지 도메인을 바꾸는 것. *Parallel Example(Paired Example)*가 없이 *Unsupervised Learning*을 하더라도 이런 것이 될 수 있지 않을까라는 의문을 품고 만들게 됨.

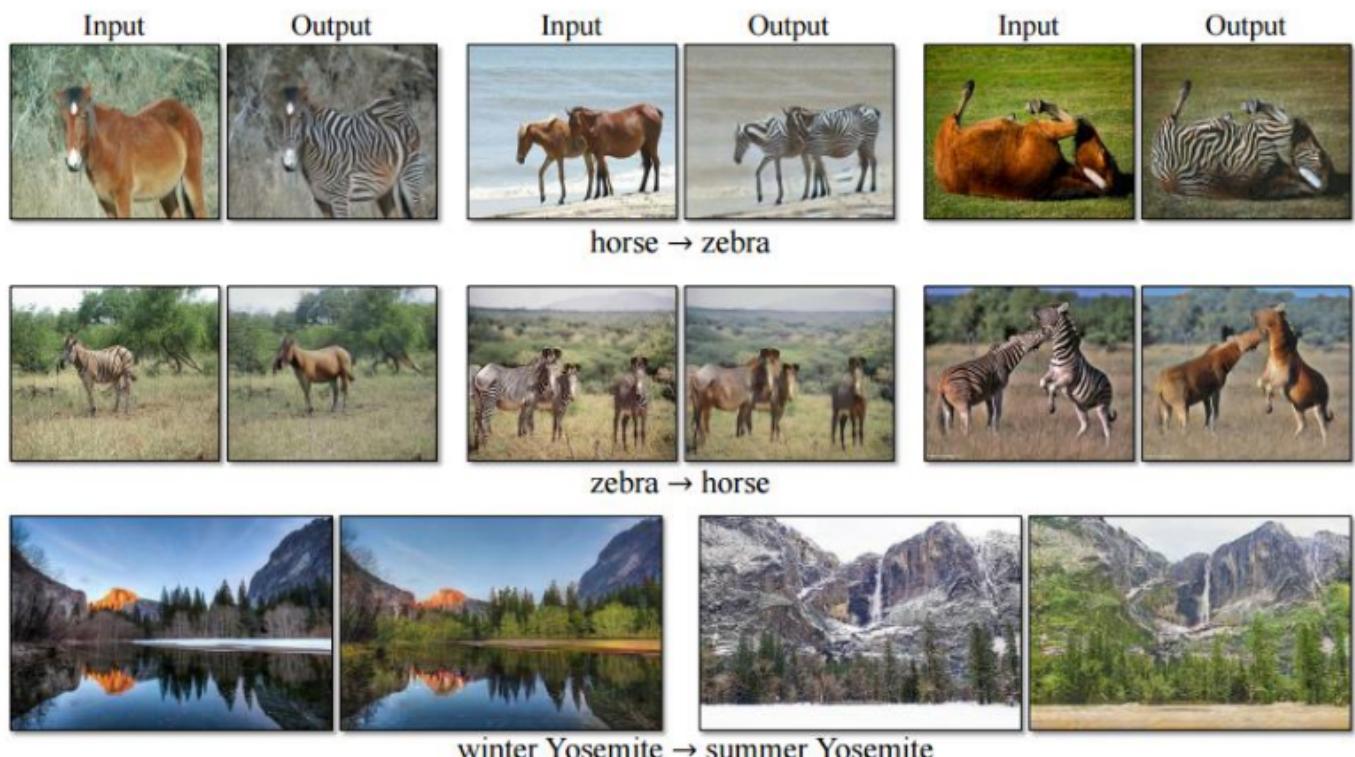


Latent code를 받지 않고 Real Image를 받게 됩니다. Encoder & Decoder 식으로 줄어들었다 늘어나게 됩니다.





Results



데이터가 두 도메인 당 10000개 이상은 있어야함. 논문에서만큼은 잘 나오지는 않지만 의미있는 결과는 볼 수 있다.

Result (SVHN & MNIST)

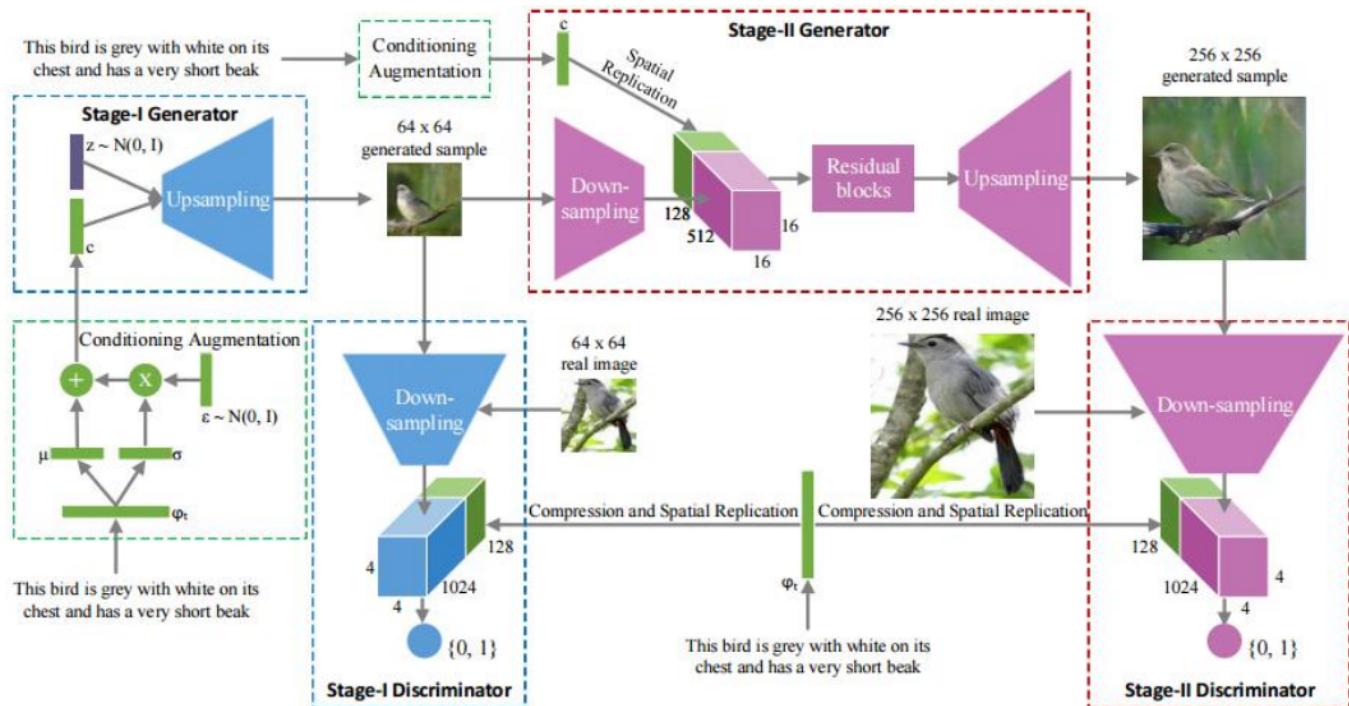


SVHN-to-MNIST



MNIST-to-SVHN

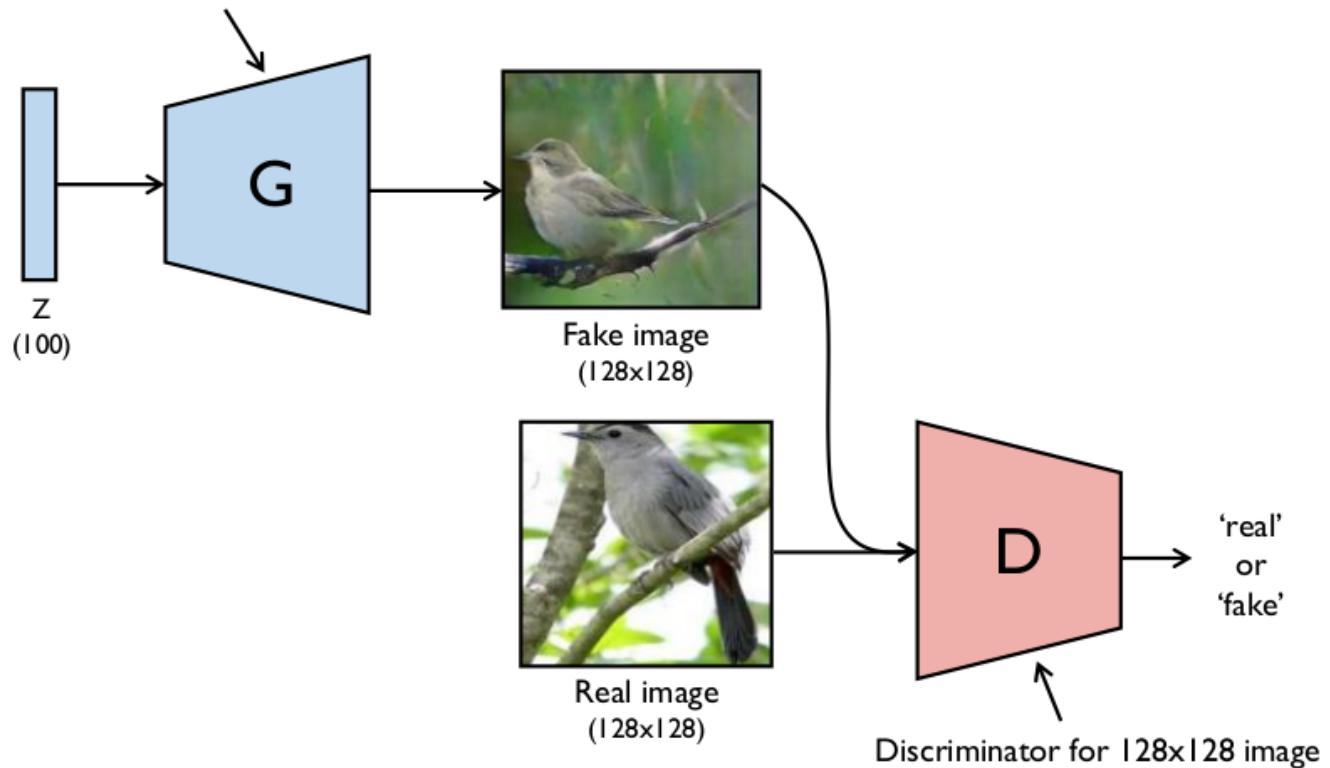
StackGAN



"새가 날라가느다" 텍스트-> 이미지로 변환되는 과정

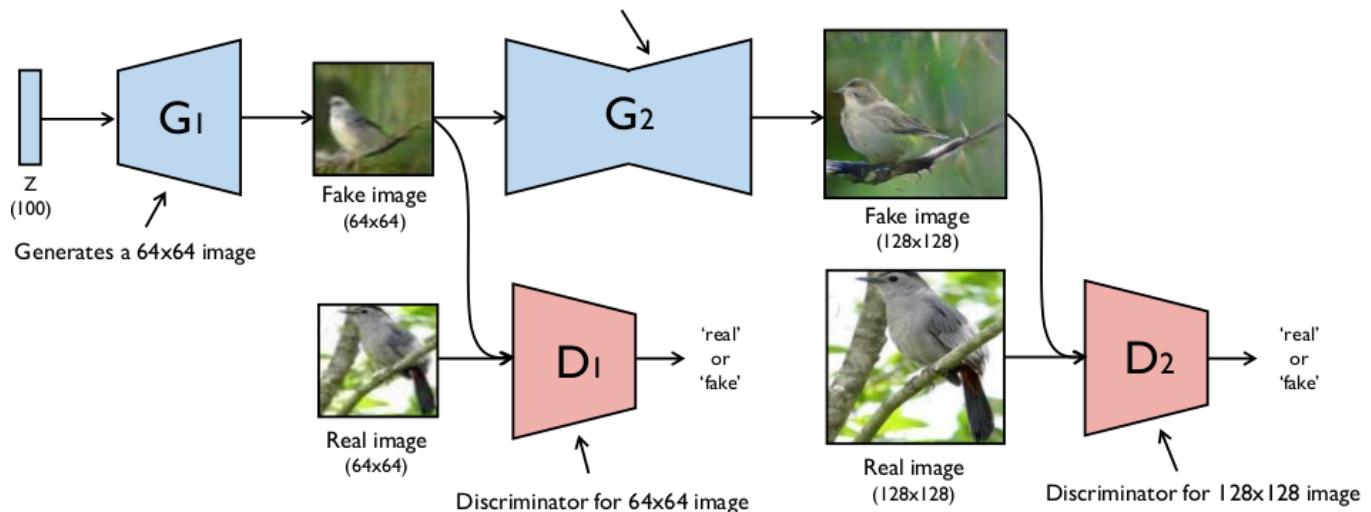
128 x 128 크기의 이미지를 생성 (그 이상의 해상도는 잘 된다는 보장이 없습니다.)

Generates a 128x128 image from scratch (not guarantee good result)

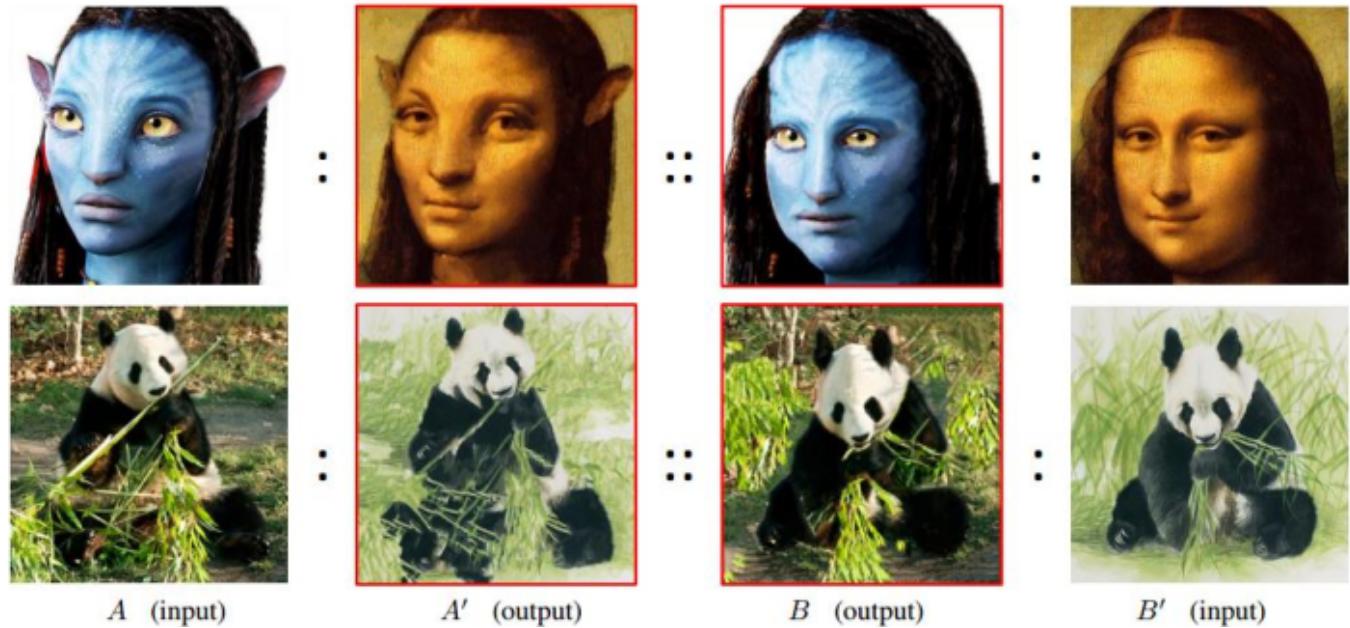


그래서 저해상도의 이미지를 upscale하는 방법을 택함.

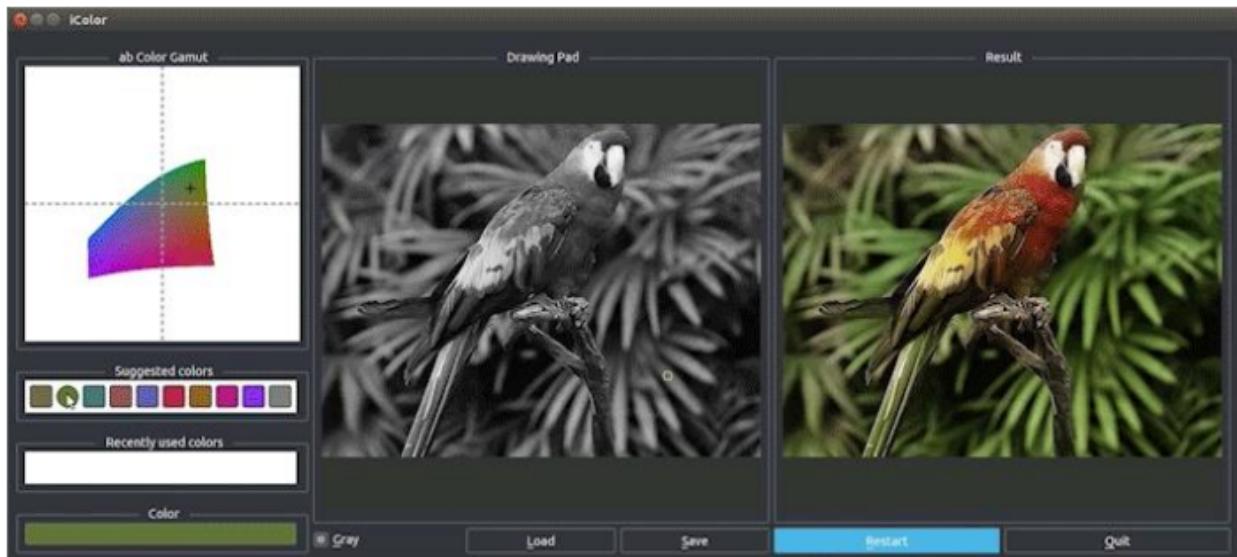
Upscales a 64x64 image to 128x128 (Easier than generating from scratch).



Latest Work



User-Interactive Image Colorization

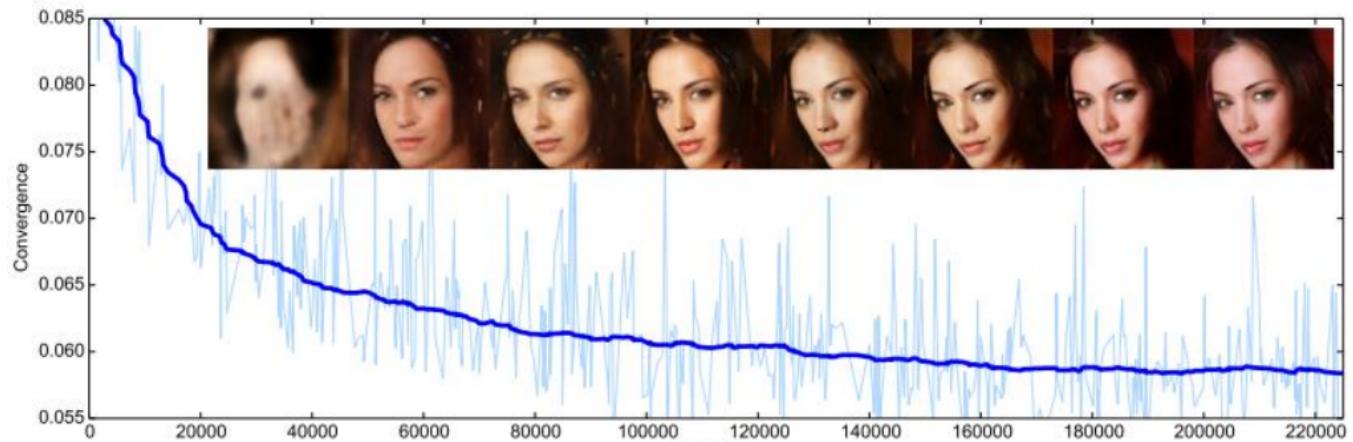


Future of GAN

Boundary Equilibrium GAN (BEGAN)

Convergence measure : Supervised-Learning을 하게 되면 Cross-entropy Loss함수가 있어서 Loss함수가 줄어드는 것을 통해 학습이 잘 되고 있음을 판단할 수 있지만, GAN에는 그런 것이 없었습니다. 실제 이미지를 보고 사람이 학습이 잘 되는지 직접 보고 판단했어야만 했습니다. BEGAN이 처음으로 Convergence Measure가 될만한 것을 제 공합니다. BEGAN의 Discriminator가 Auto Encoder식으로 복잡하게 구현되어 있습니다. 복잡한 구조에서만 사용할 수만 있는 Convergence measure를 제공한다는 한계점을 가지고 있습니다.

$$\mathcal{M}_{global} = \mathcal{L}(x) + |\gamma \mathcal{L}(x) - \mathcal{L}(G(z_G))|$$

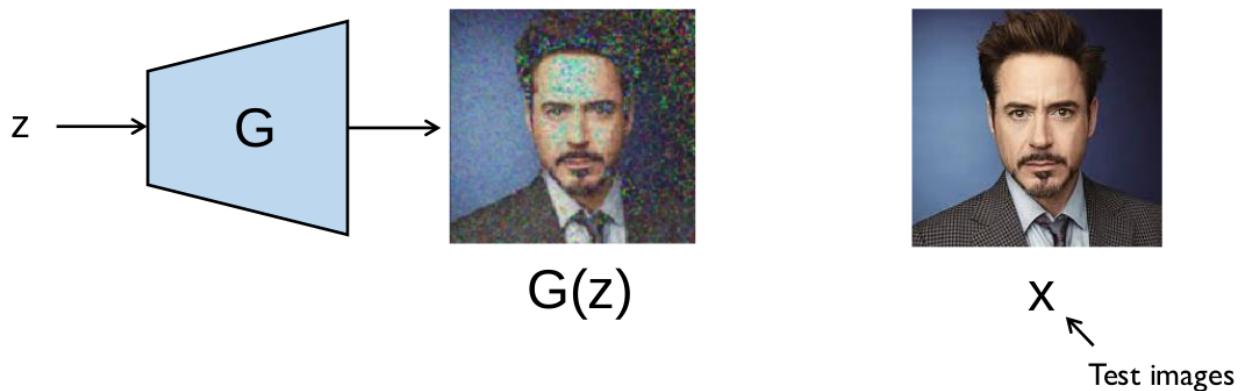


Reconstruction Loss : batch normalization 이 아니라 VT Normalization이 GAN에서 쓰면 좋다는 논문이 발표되었다.

z가스

1. Generator의 Weight를 고정시키고
2. z를 랜덤하게 샘플링하고
3. forward propagation을 해서 이미지를 만듭니다. $G(z)$
4. X와 $G(z)$ 를 빼서 L2 Loss(Reconstruction Loss)를 계산합니다.
5. 계산한 Loss를 Back propagation하고
6. Generator를 학습하는 게 아니라 z가스를 학습하는 중
7. $G(z)$ 와 X가 비슷해지도록 z를 학습하는 것. (Gradient Descent를 활용하여)

$$\mathcal{L}_{rec}(G, X) = \frac{1}{m} \sum_{i=1}^m \min_z \|G(z) - x^{(i)}\|^2$$

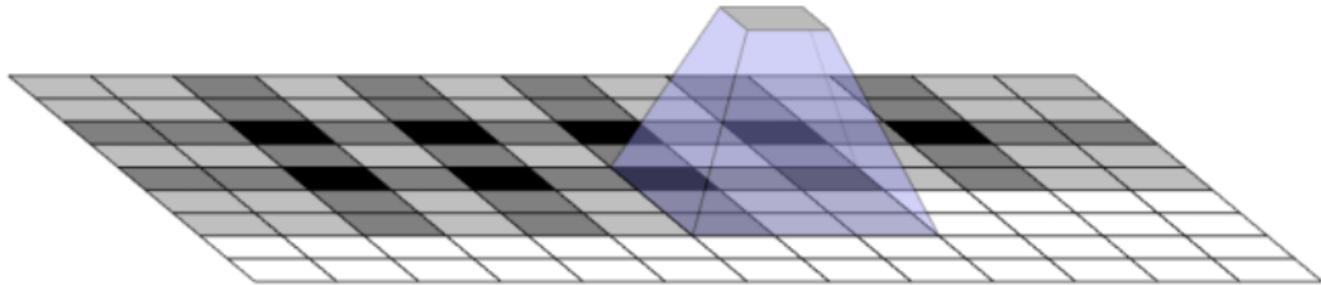


단점이 바로 보인다. z를 학습해야한다. (배보다 배꼽이 더 크게 된다.)

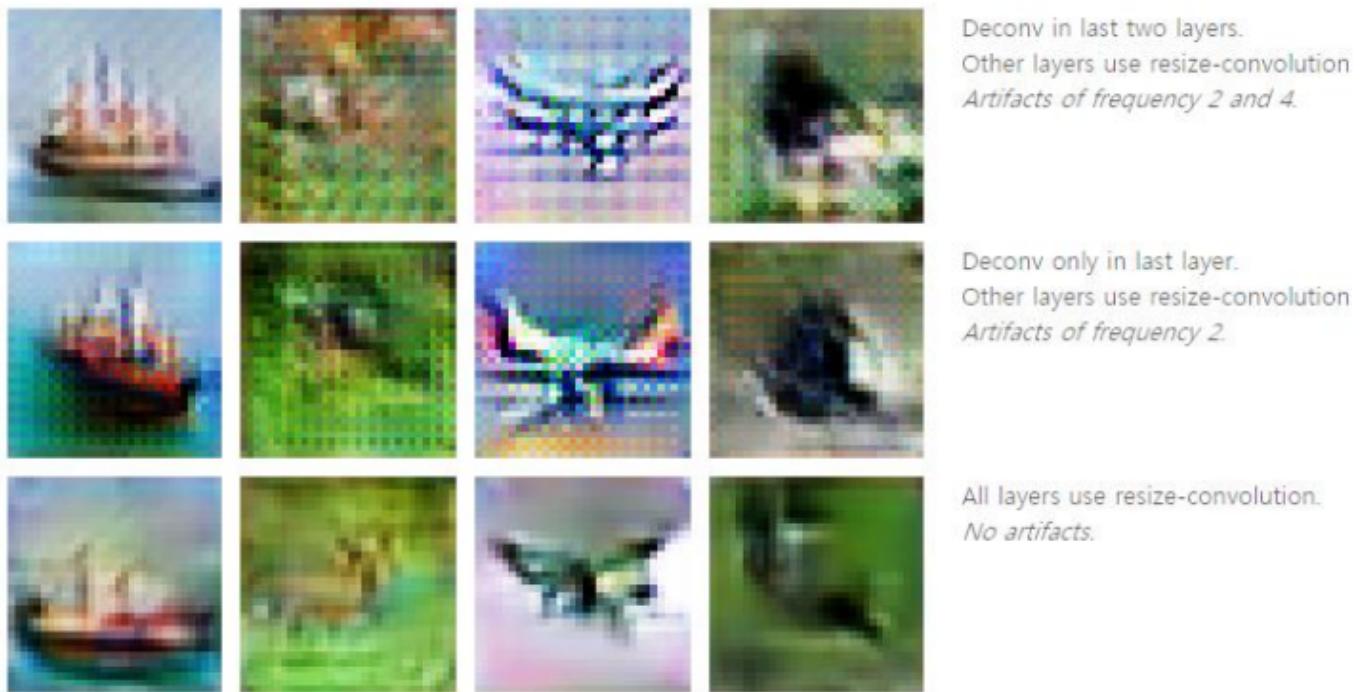
두번째 단점으로는 매우 느리다

Deconvolution Checkboard Artifacts

좀 더 좋은 Up-sampling 방법을 찾아야되지 않을까



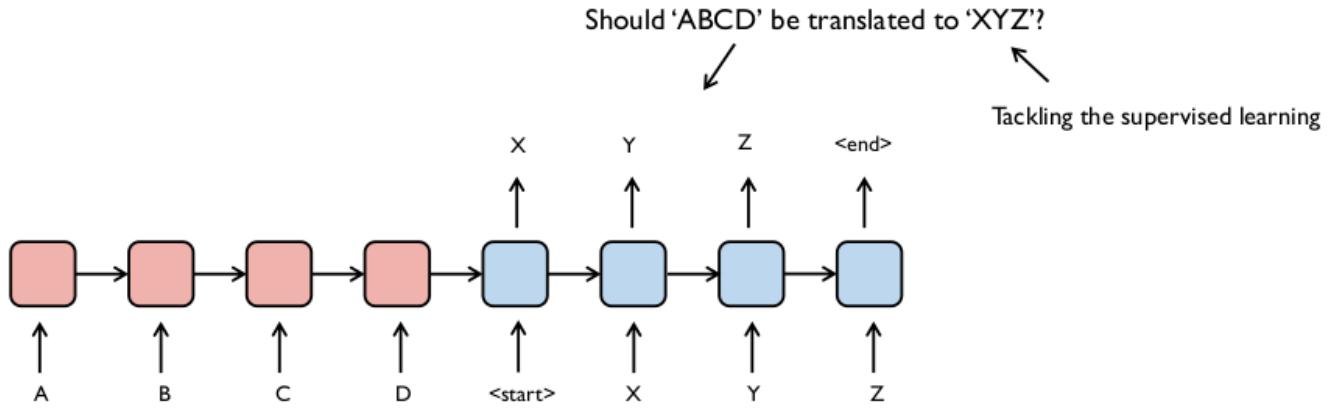
이런 식이면 불균형한 Output을 만들게 된다. (검은색(4) - 진회색 (2)- 회색(1))



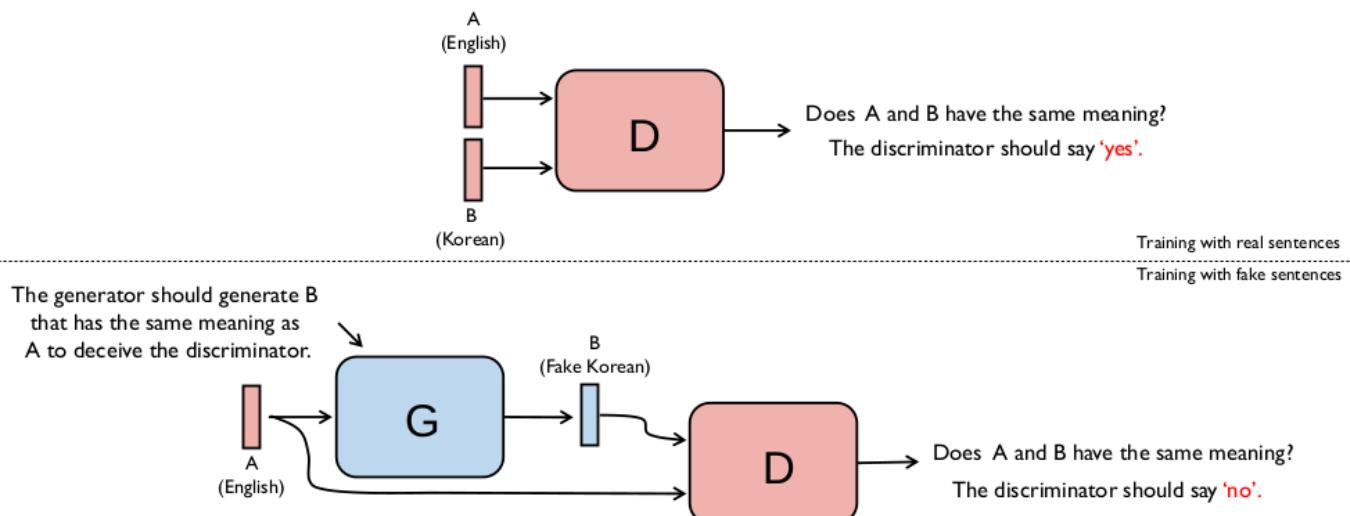
> 이런식이다.

Resize-Convolution : 단지 학습으로 업샘플링하는 deconvolution과는 달리 업샘플링 방식을 rule-based로 한다.
(ex. nearest neighbor) 그 이후의 convolution layer (stride size = 1)로 쌓게 됩니다.

Seq2Seq



- Machine Translation (GANs)



Discriminator는 A, B 두개가 가ㅌ은 뜻을 가짜고 있으면 1에 가까운가스을, 가짜 한글을 만들어내면 0에 가까운 가스으로 판별합니다.