# GAN:
# Theories & Application

# Contents:

# Contents:

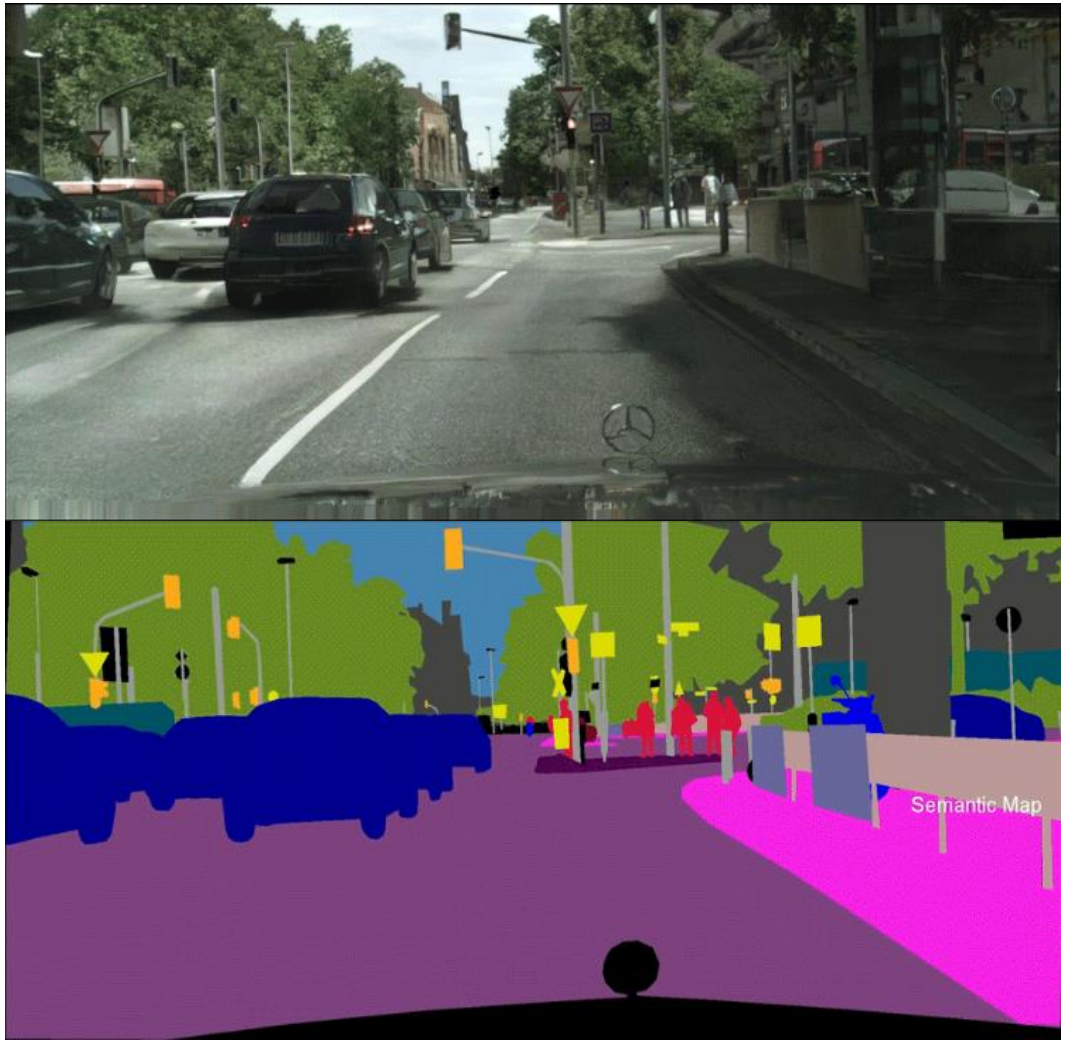
# I. GAN: Theories

# Create by Algorithm:

# Create by Algorithm:

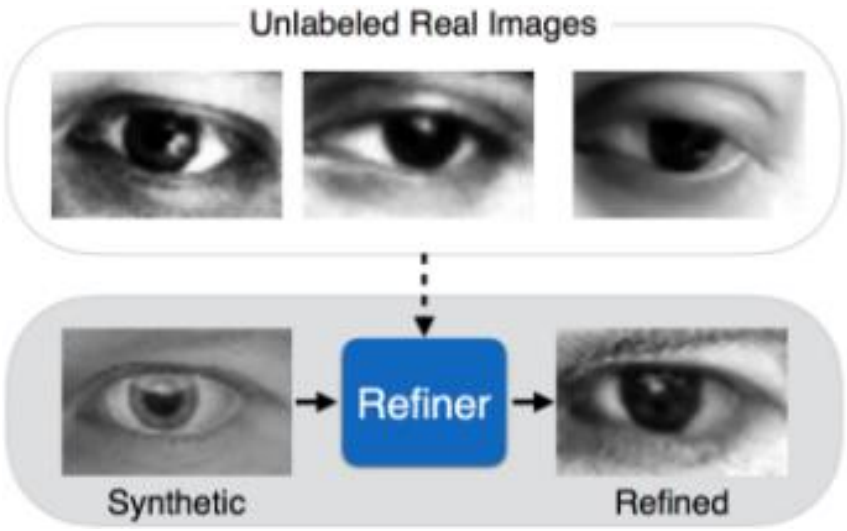
# Create by Algorithm:

# Create by Algorithm:



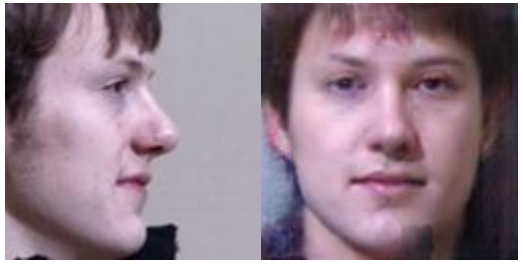
# Create by Algorithm:

# Create by Algorithm:



# Create by Algorithm:

# Create by Algorithm :



# Create by Algorithm :


(a) Good. (b) Thread-line. (c) Brushing. (d) Tin-resuide. (e) Wound

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.1 Questions: Can We Generate Image from DL?

Yes! By Generative Adversarial Network. [对抗生成网络]

It's a METHODOLOGY. NOT A SPECIFIC NETWORK STRUCTURE

### A.2 How to solve?

a. From ideological perspective

Generator & Discriminator: Learning Adversarially.

Discriminator: Decide whether the input is a Real image or a Fake (Generated) one

Generator: Generate an image to try to fool the discriminator

?

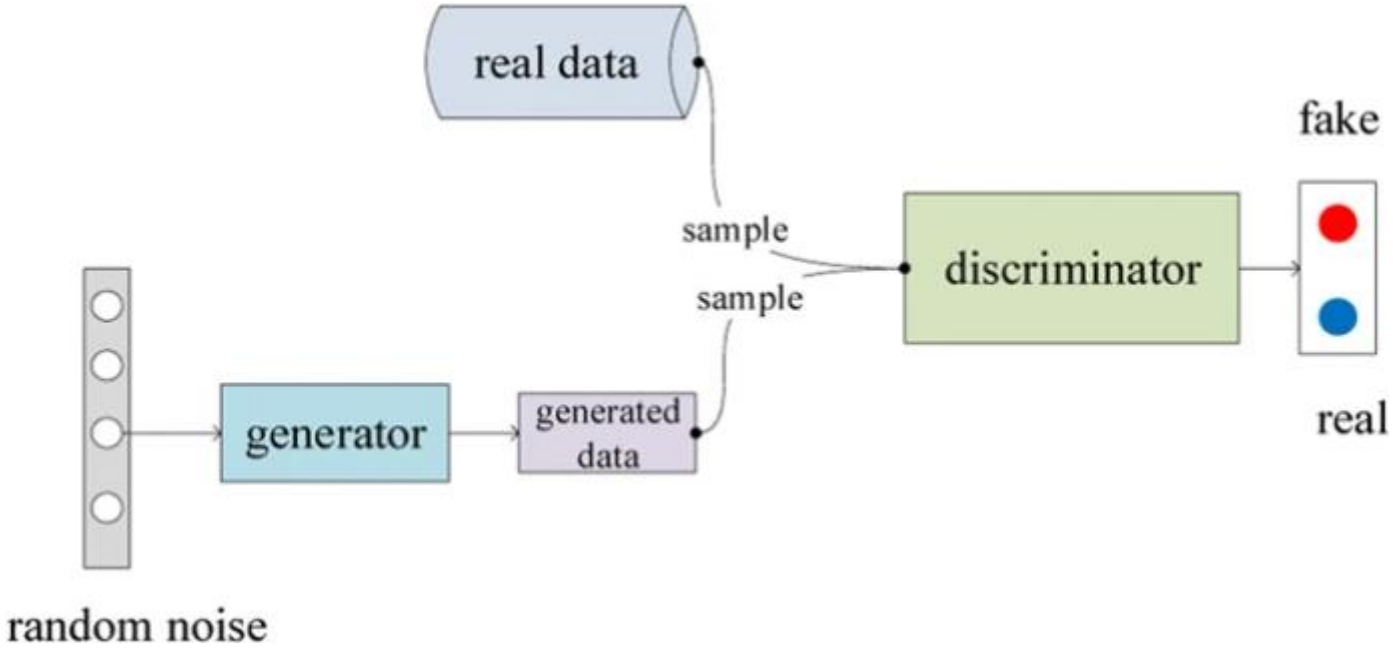
It seems when G & D achieves to equilibrium that we can get a best result.

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.2 How to solve?

#### b. From procedural perspective

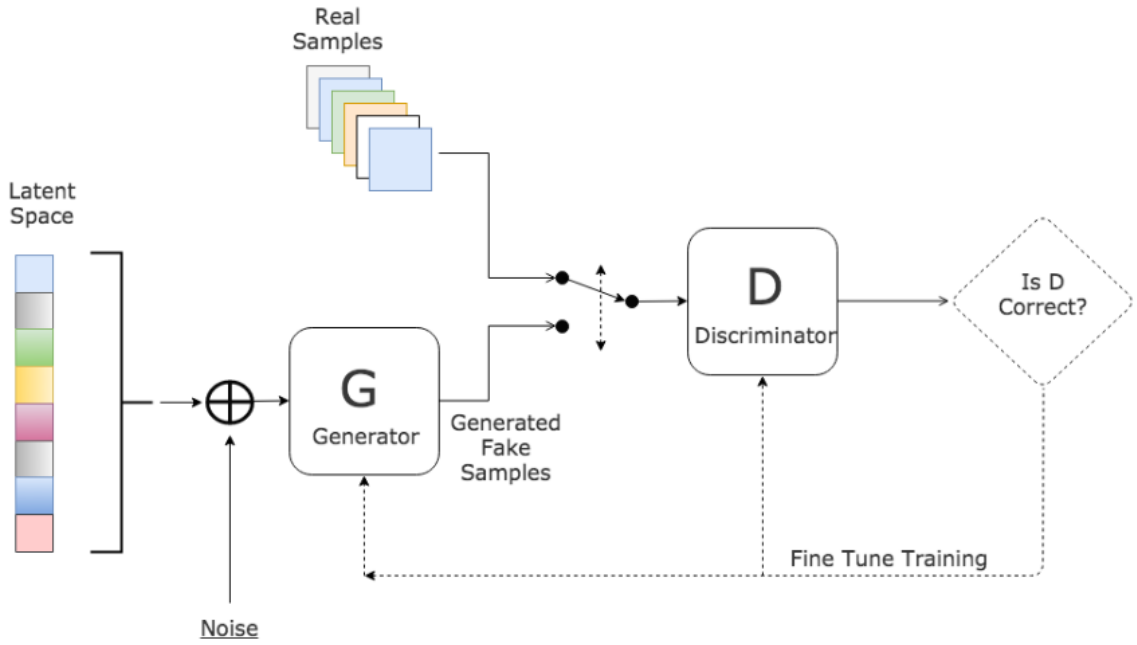


# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.2 How to solve?

#### b. From procedural perspective



# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.2 How to solve?

#### c. From mathematical perspective

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

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.2 How to solve?

#### c. From mathematical perspective

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.3 Explanation of the formula

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

① $Value\ Funcation$: [价值函数]

- **Different from Cost Function. A concept from Reinforcement Learning**
- **RL:** 强化学习是想让一个智能体(agent)在不同的环境状态(state)下，学会选择那个使得奖赏(reward)最大的动作(action)。

  **Agent**在 t 时刻，通过观测环境得到自己所在的 **状态(state)**，接下来agent根据 **策略(policy)** 进行决策后，做出一个 **动作(action)**。这个action就会使得agent在 **环境(environment)** 中转移到一个新的状态，并且在转移时获得一个 **即时奖励(reward)** 值，这样agent又可以在新state中重新选择动作。

  这样就可以累积很多reward值（$R_0, R_1, ..., R_t, ..., R_T$）。agent的目标是希望在**达到终点**的时候获得的**累积reward最大**。
- **VF:** $v^{\pi}(s) = E\left[\sum_{i=1}^{T} \gamma^{i-1} r_i\right], \forall s \in S,$ *we always have* $v^*(s) = \max_{\pi} v^{\pi}(s)$

  *policay* *weight* *reward* *state*

  *We'll get max value* $v^*(s)$ *when we under policy* $\pi$
- **D, G: Learnt states (Learnt discriminator & generator) to maximize the value**
- **$\pi$ : Learnt policy (parameters & structure) of D & G [It's our target to find $\pi$ (parameters & structures) of G & D]**

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.3 Explanation of the formula

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

② $\max_D$: *We hope to get max V under (policy) D*

- **What is D?**

D { **Structure / Parameters: Policy** —*In paper*→ **MLP** —*other dev structures*→ **Fancy Struct** ; **Effect: State** } **Discriminator**

- **Why maximize V?**

**An ideal D can achieve** { **D(x) —>1** ; **D(G(z)) —>0** } ⇒ { ④—>1 ; ⑤—>0 } ⇒ ***maximize V***

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.3 Explanation of the formula

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

③ $\min_G$: *We hope to get min V under (policy) G*

- What is G?

G { Structure / Parameters: Policy —*In paper*→ MLP —*other dev structures*→ Fancy Struct } Generator

Effect: State

- Why minimize V?

An ideal G can achieve ⇒ D(G(z)) —>1 ⇒ ⑤—>1 ⇒ ***minimize V***

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.3 Explanation of the formula

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

⑥ $x\sim p_{data}(x)$ : $x$ *follows the distribution* $p_{data}$

- **x is real data (images), which under the possibility distribution denoted by $p_{data}$**
- **For each real image (x), it's a specific sample. All samples (real images) form a random variable which has the possibility distribution $p_{data}$**

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.3 Explanation of the formula

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

⑦ $z\sim p_z(z)$ : $x$ $follows$ $the$ $distribution$ $p_z$

- **z: Noises. From where the generated images are coming.**
- **$p_z$: Possibility distribution of z.**
- **For each noise sample (z), it's a specific noise sample. All samples (noises) form a random variable which has the possibility distribution $p_z$**

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.3 Explanation of the formula

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

⑧ $\mathbb{E}$ : *Expectation*

- $\boldsymbol{E(x) = \int_{-\infty}^{\infty} xp(x)\,dx}$
- $\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] = \int_{-\infty}^{\infty} p_{data}(x) log D(x)\,dx = \int_x p_{data}(\boldsymbol{x}) \log\big(D(\boldsymbol{x})\big)\,dx$
- $\mathbb{E}_{z\sim p_z(z)}\left[\log\left(1-D\left(G(z)\right)\right)\right] = \int_{-\infty}^{\infty} \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1-D\left(G(z)\right)\right)\right] dz = \int_x p_g(\boldsymbol{x}) \log\big(1-D(\boldsymbol{x})\big)\,dx$

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.3 Explanation of the formula

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

$$\Rightarrow$$

$$V(D,G) = \int_x p_{data}(\boldsymbol{x}) \log\big(D(\boldsymbol{x})\big)\, dx + \int_x p_g(\boldsymbol{x}) \log\big(1-D(\boldsymbol{x})\big)\, dx$$

$$= \int_x p_{data}(\boldsymbol{x}) \log\big(D(\boldsymbol{x})\big) + p_g(\boldsymbol{x}) \log\big(1-D(\boldsymbol{x})\big)\, dx$$

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.3 Explanation of the formula

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

***Summary***:

- Generate a discriminator (D) & a generator (G) step by step
- The target of the D is to try its best to discriminate real and fake images while the target of the G is to try its best to generate fake images to fool the D.
- It seems we can get a global optimality (equilibrium) by dragging $p_g \rightarrow p_{data}$

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

**A.4 Can we get the global optimality at $p_g = p_{data}$?**

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

***Questions I*: Is the global optimality achieving at $p_g = p_{data}$?**

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

**A.4 Can we get the global optimality at $p_g = p_{data}$?**

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

$$\Leftrightarrow$$

$$\max_D V(D) = \int_x p_{data}(\boldsymbol{x})\log\big(D(\boldsymbol{x})\big) + p_g(\boldsymbol{x})\log\big(1-D(\boldsymbol{x})\big)\,dx$$

$$\Leftrightarrow$$

$$\max_D f(D) = a\log(D) + b\log(1-D)$$

$$\Rightarrow$$

$$\frac{\partial f}{\partial d} = \frac{a}{D} - \frac{b}{1-D} = 0 \Rightarrow D^* = \frac{a}{a+b} = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

**Optimal Discriminator**

***Questions I*:**
**Is the global optimality achieving at $p_g = p_{data}$?**

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.4 Can we get the global optimality at $p_g = p_{data}$?

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

$$\Leftrightarrow$$

$$\max_D V(D) = \int_x p_{data}(\boldsymbol{x}) \log\big(D(\boldsymbol{x})\big) + p_g(\boldsymbol{x}) \log\big(1-D(\boldsymbol{x})\big)\, dx$$

$$\Leftrightarrow$$

$$\max_D f(D) = \int_x p_{data}(\boldsymbol{x}) \log\left(\frac{2p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x})+p_g(\boldsymbol{x})} - log2\right) + p_g(\boldsymbol{x}) \log\left(\frac{2p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x})+p_g(\boldsymbol{x})} - log2\right) dx$$

$$= -log4 + \int p_{data} \log\left(\frac{2p_{data}}{p_{data}+p_g}\right) dx + \int p_g \log\left(\frac{2p_g}{p_{data}+p_g}\right) dx$$

***Questions I*:**
**Is the global optimality achieving at $p_g = p_{data}$?**

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

**A.4 Can we get the global optimality at $p_g = p_{data}$?**

$$\max_D f(D)$$

$$= \int_x p_{data}(\boldsymbol{x}) \log\left(\frac{2p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} - log2\right) + p_g(\boldsymbol{x}) \log\left(\frac{2p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} - log2\right) dx$$

$$= -log4 + \int p_{data} \log\left(\frac{2p_{data}}{p_{data} + p_g}\right) dx + \int p_g \log\left(\frac{2p_g}{p_{data} + p_g}\right) dx$$

$$= -log4 + D_{KL}\left(p_{data} \parallel \frac{p_{data} + P_g}{2}\right) + D_{KL}\left(p_g \parallel \frac{p_{data} + P_g}{2}\right)$$

- $\boldsymbol{KL - Divergence}$:

$$D_{KL}(P \parallel Q) = \sum_i P(i) log \frac{P(i)}{Q(i)} = \int_x P(x) log \frac{P(x)}{Q(x)} dx$$

$\boldsymbol{Questions\ I}$:
Is the global optimality achieving at $\boldsymbol{p_g = p_{data}}$?

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

**A.4 Can we get the global optimality at $p_g = p_{data}$?**

$$\max_D f(D)$$

$$= \int_x p_{data}(\boldsymbol{x}) \log\left(\frac{2p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} - log2\right) + p_g(\boldsymbol{x}) \log\left(\frac{2p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} - log2\right) dx$$

$$= -log4 + D_{KL}\left(p_{data} \parallel \frac{p_{data} + P_g}{2}\right) + D_{KL}\left(p_g \parallel \frac{p_{data} + P_g}{2}\right)$$

$$= -log4 + 2JSD(p_{data} \parallel p_g)$$

- $\boldsymbol{JS - Divergence}$:

$$JSD(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel \frac{P+Q}{2}) + \frac{1}{2} D_{KL}(Q \parallel \frac{P+Q}{2})$$

$\boldsymbol{Questions\ I}$:
Is the global optimality achieving at $\boldsymbol{p_g = p_{data}}$?

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.4 Can we get the global optimality at $p_g = p_{data}$?

- $\boldsymbol{KL - Divergence}$: **相对熵**

$$D_{KL}(P \parallel Q) = \sum_i P(i) log \frac{P(i)}{Q(i)} = \int_x P(x) log \frac{P(x)}{Q(x)} dx$$

**KL散度**又称为**相对熵**，信息散度，信息增益。KL散度是两个概率分布P和Q **差别**的非对称性的度量。P为真实概率分布，Q为拟合分布。若PQ相同，则值为零。KL散度的个大问题：非对称

- $\boldsymbol{JS - Divergence}$:

$$JSD(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel \frac{P+Q}{2}) + \frac{1}{2} D_{KL}(Q \parallel \frac{P+Q}{2})$$

**JS散度**度量了两个概率分布的相似度，基于KL散度的变体，解决了KL散度非对称的问题。一般地，JS散度是对称的，其取值是0到1之间。且，是对称的

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

**A.4 Can we get the global optimality at $p_g = p_{data}$?**

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

$$\Leftrightarrow$$

$$f(D^*) = -log4 + 2JSD(p_{data} \parallel p_g)$$
$$\geq -log4 + 2 \times 0$$
$$= -log4 \ (p_{data} = p_g)$$
$$\because [-log4 + \int p_{data} \log\left(\frac{2p_{data}}{p_{data}+p_g}\right)dx + \int p_g \log\left(\frac{2p_g}{p_{data}+p_g}\right)dx]$$

***Questions I*:**
**Is the global optimality achieving at $p_g = p_{data}$?**

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

**A.4 Can we get the global optimality at $p_g = p_{data}$?**

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

$$\Leftrightarrow V(D^*) = -log4 \ (p_{data} = p_g)$$

***Questions I*: Is the global optimality achieving at $p_g = p_{data}$?**

**YES**

***When $D = D^*$:***

$$Original\ Formula =$$
$$C(G) = -log4 + 2JSD(p_{data} \parallel p_g)$$
$$\geq -log4$$
$$= -log4 \ (p_{data} = p_g)$$

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.5 How to solve?

#### d. From algorithm perspective

**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:

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

**end for**

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

- Q1: why ascending & descending?
- Q2: where is $\mathbf{p_{data}}$ & $p_g$
- Q3: Any prob. with generator's updates?

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.5 How to solve?

#### d. From algorithm perspective

**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:

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

**end for**

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

- Q1: why ascending & descending?

  Ans: $\min_G \max_D$, update to opposite direction

- Q2: where is $\mathbf{p_{data}}$ & $p_g$

  Ans: $\mathbf{1} \cdot logD(\boldsymbol{x}) + \mathbf{(1-0)} \cdot \log(1 - D(G(\boldsymbol{z})))$

- Q3: Any prob. with generator's updates?

  Ans:
  $D(G(\boldsymbol{z}))$ 初始应该很小(因为太假了, 很好分)
  所以如果按照原始, 则几乎学不到东西。
  所以不如:
  $\log\left(1 - D\left(G(\boldsymbol{z})\right)\right) \Leftrightarrow -\log(D(G(\boldsymbol{z})))$
  减小$\left(1 - D(G)\right)$的梯度等价于提高$D(G)$的梯度

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

### A.5 How to solve?

#### e. From code perspective

```
optimizer_G.zero_grad()

# Sample noise as generator input
z = Variable(Tensor(np.random.normal(0, 1, (imgs.shape[0], opt.latent_dim))))

# Generate a batch of images
gen_imgs = generator(z)

# Loss measures generator's ability to fool the discriminator
g_loss = adversarial_loss(discriminator(gen_imgs), valid)

g_loss.backward()
optimizer_G.step()
```

```
# ---------------------
#  Train Discriminator
# ---------------------

optimizer_D.zero_grad()

# Measure discriminator's ability to classify real from generated samples
real_loss = adversarial_loss(discriminator(real_imgs), valid)
fake_loss = adversarial_loss(discriminator(gen_imgs.detach()), fake)
d_loss = (real_loss + fake_loss) / 2

d_loss.backward()
optimizer_D.step()
```

# I. GAN: Theories

## B. GAN: Problems

**Almost all people are suffering:**

- **It's really hard to train**
- **It's prone to generate similar images**
- **Generated images are blurred and not real**
- **The Training procedure is unstable**

# I. GAN: Theories

## A. GAN [2014 Goodfellow]

**A.6 Can we get the global optimality at $p_g = p_{data}$?**

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

③ ② ① ⑧ ⑥ ④ ⑧ ⑦ ⑤

***Questions II*: Can we get/converge to the global optimality?**

# I. GAN: Theories

## B. GAN: Problems

**Almost all people are suffering:**

- **It's really hard to train**
- **It's prone to generate similar images**
- **Generated images are blurred and not real**
- **The Training procedure is unstable**

**WHY!**

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

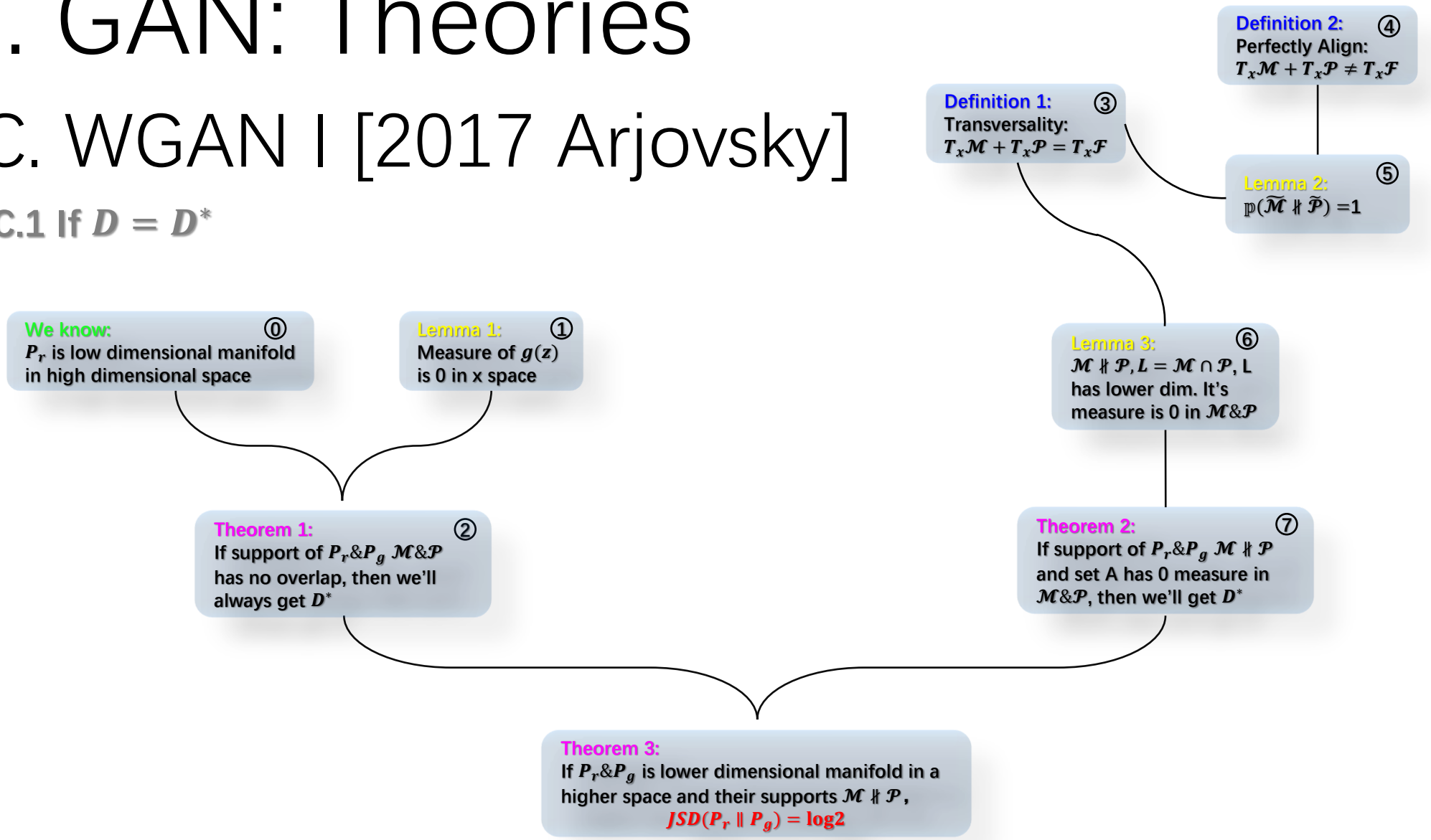
**Argument:**

**It's all because of the optimal discriminator ($D^*$)**

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

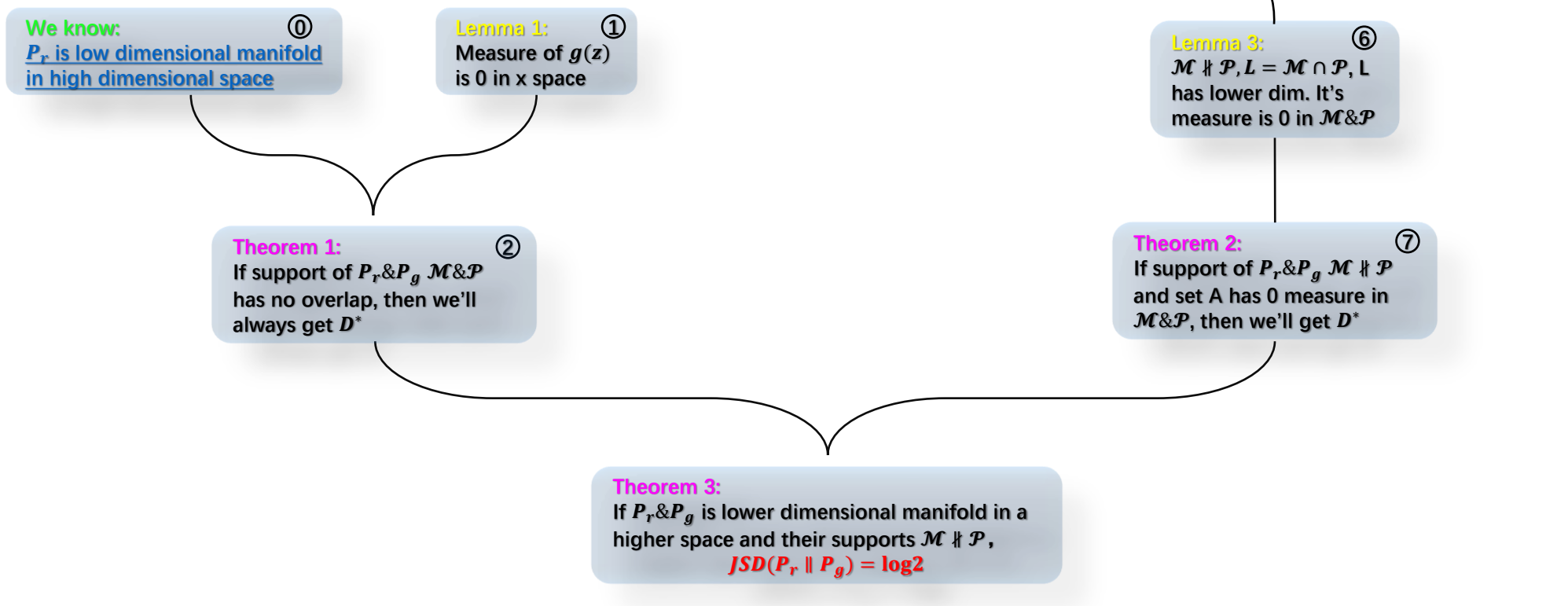


# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

There is strong empirical and theoretical evidence to believe that $\mathbb{P}_r$ is indeed extremely concentrated on a low dimensional manifold (Narayanan & Mitter, 2010).



# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

**Lemma 1.** *Let $g : \mathcal{Z} \to \mathcal{X}$ be a function composed by affine transformations and pointwise nonlinearities, which can either be rectifiers, leaky rectifiers, or smooth strictly increasing functions (such as the sigmoid, tanh, softplus, etc). Then, $g(\mathcal{Z})$ is contained in a countable union of manifolds of dimension at most $\dim \mathcal{Z}$. Therefore, if the dimension of $\mathcal{Z}$ is less than the one of $\mathcal{X}$, $g(\mathcal{Z})$ will be a set of measure 0 in $\mathcal{X}$.*

**We know:** ⓪
$P_r$ is low dimensional manifold in high dimensional space

**Lemma 1:** ①
Measure of $g(z)$ is 0 in x space

1. $g(z)$ is a lower dimensional manifold in a higher space
2. Usually $d(z) < d(x)$;
   It doesn't matter if $d(z) \geq d(x)$ because it doesn't mean $d(g(z)) \geq d(x)$

**Theorem 1:** ②
If support of $P_r \& P_g$ $\mathcal{M} \& \mathcal{P}$ has no overlap, then we'll always get $D^*$

**Definition 1:** ③
Transversality:
$T_x\mathcal{M} + T_x\mathcal{P} = T_x\mathcal{F}$

**Definition 2:** ④
Perfectly Align:
$T_x\mathcal{M} + T_x\mathcal{P} \neq T_x\mathcal{F}$

**Lemma 2:** ⑤
$\mathbb{P}(\widetilde{\mathcal{M}} \nparallel \widetilde{\mathcal{P}}) = 1$

**Lemma 3:** ⑥
$\mathcal{M} \nparallel \mathcal{P}, L = \mathcal{M} \cap \mathcal{P}$, L has lower dim. It's measure is 0 in $\mathcal{M} \& \mathcal{P}$

**Theorem 2:** ⑦
If support of $P_r \& P_g$ $\mathcal{M} \nparallel \mathcal{P}$ and set A has 0 measure in $\mathcal{M} \& \mathcal{P}$, then we'll get $D^*$

**Theorem 3:** ⑧
If $P_r \& P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$,
$JSD(P_r \| P_g) = \log 2$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

**Definition 2:** ④
Perfectly Align:
$T_x\mathcal{M} + T_x\mathcal{P} \neq T_x\mathcal{F}$

**Definition 1:** ③
Transversality:
$T_x\mathcal{M} + T_x\mathcal{P} = T_x\mathcal{F}$

**Lemma 2:** ⑤
$\mathbb{p}(\widetilde{\mathcal{M}} \nparallel \widetilde{\mathcal{P}}) = 1$

**We know:** ⓪
$P_r$ is low dimensional manifold in high dimensional space

**Lemma 1:** ①
Measure of $g(z)$ is 0 in x space

**Lemma 3:** ⑥
$\mathcal{M} \nparallel \mathcal{P}, L = \mathcal{M} \cap \mathcal{P}$, L has lower dim. It's measure is 0 in $\mathcal{M}\&\mathcal{P}$

**Theorem 1:** ②
If support of $P_r\&P_g$ $\mathcal{M}\&\mathcal{P}$ has no overlap, then we'll always get $D^*$

1. If $P_r\&P_g$ has supports $\mathcal{M} \cap \mathcal{P} = \emptyset$, We'll always have optimal discriminator $D^*$
2. $P_r(D^*(x=1)) = 1, P_g(D^*(x=0)) = 1$

**Theorem 2:** ⑦
If support of $P_r\&P_g$ $\mathcal{M} \nparallel \mathcal{P}$ and set A has 0 measure in $\mathcal{M}\&\mathcal{P}$, then we'll get $D^*$

**Theorem 2.1.** *If two distributions $\mathbb{P}_r$ and $\mathbb{P}_g$ have support contained on two disjoint compact subsets $\mathcal{M}$ and $\mathcal{P}$ respectively, then there is a smooth optimal discrimator $D^* : \mathcal{X} \to [0, 1]$ that has accuracy 1 and $\nabla_x D^*(x) = 0$ for all $x \in \mathcal{M} \cup \mathcal{P}$.*

**Theorem 3:**
If $P_r\&P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$,
$JSD(P_r \parallel P_g) = \log 2$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

1. $T_x$: Tangent Space
2. $\mathcal{F} \in \mathbb{R}^d$
3. $\mathcal{M}$&$\mathcal{P}$ is submanifold of $\mathcal{F}$

**Definition 1:** ③
Transversality:
$T_x\mathcal{M} + T_x\mathcal{P} = T_x\mathcal{F}$

**Definition 2:** ④
Perfectly Align:
$T_x\mathcal{M} + T_x\mathcal{P} \neq T_x\mathcal{F}$

**Lemma 2:** ⑤
$\mathbb{p}(\widetilde{\mathcal{M}} \nparallel \widetilde{\mathcal{P}}) = 1$

**Definition 2.1.** We first need to recall the definition of transversallity. Let $\mathcal{M}$ and $\mathcal{P}$ be two boundary free regular submanifolds of $\mathcal{F}$, which in our cases will simply be $\mathcal{F} = \mathbb{R}^d$. Let $x \in \mathcal{M} \cap \mathcal{P}$ be an intersection point of the two manifolds. We say that $\mathcal{M}$ and $\mathcal{P}$ intersect transversally in $x$ if $T_x\mathcal{M} + T_x\mathcal{P} = T_x\mathcal{F}$, where $T_x\mathcal{M}$ means the tangent space of $\mathcal{M}$ around $x$.

**We know:** ⓪
$P_r$ is low dimensional manifold in high dimensional space

**Lemma 1:** ①
Measure of $g(z)$ is 0 in x space

**Lemma 3:** ⑥
$\mathcal{M} \nparallel \mathcal{P}, L = \mathcal{M} \cap \mathcal{P}$, L has lower dim. It's measure is 0 in $\mathcal{M}$&$\mathcal{P}$

**Theorem 1:** ②
If support of $P_r$&$P_g$ $\mathcal{M}$&$\mathcal{P}$ has no overlap, then we'll always get $D^*$

**Theorem 2:** ⑦
If support of $P_r$&$P_g$ $\mathcal{M} \nparallel \mathcal{P}$ and set A has 0 measure in $\mathcal{M}$&$\mathcal{P}$, then we'll get $D^*$

**Theorem 3:**
If $P_r$&$P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$,
$JSD(P_r \| P_g) = \log 2$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

**Definition 2.2.** We say that two manifolds without boundary $\mathcal{M}$ and $\mathcal{P}$ **perfectly align** if there is an $x \in \mathcal{M} \cap \mathcal{P}$ such that $\mathcal{M}$ and $\mathcal{P}$ don't intersect transversally in $x$.
We shall note the boundary and interior of a manifold $\mathcal{M}$ by $\partial M$ and Int $M$ respectively. We say that two manifolds $\mathcal{M}$ and $\mathcal{P}$ (with or without boundary) perfectly align if any of the boundary free manifold pairs $(\text{Int}\,\mathcal{M}, \text{Int}\,\mathcal{P})$, $(\text{Int}\,\mathcal{M}, \partial\mathcal{P})$, $(\partial\mathcal{M}, \text{Int}\,\mathcal{P})$ or $(\partial\mathcal{M}, \partial\mathcal{P})$ perfectly align.

1. 4 types of perfectly align.
2. $\partial\mathcal{M}/\,\partial\mathcal{P}$ — 皮/皮
   $\partial\mathcal{M}/\,\text{Int}\mathcal{P}$ — 皮/瓤（瓤/皮）
   $\text{Int}\mathcal{M}/\text{Int}\mathcal{P}$ — 瓤/瓤

**Definition 2:**
Perfectly Align:
$T_x\mathcal{M} + T_x\mathcal{P} \neq T_x\mathcal{F}$

**Definition 1:** ③
Transversality:
$T_x\mathcal{M} + T_x\mathcal{P} = T_x\mathcal{F}$

**Lemma 2:** ⑤
$\mathbb{P}(\widetilde{\mathcal{M}} \nparallel \widetilde{\mathcal{P}}) = 1$

**We know:** ⓪
$P_r$ is low dimensional manifold in high dimensional space

**Lemma 1:** ①
Measure of $g(z)$ is 0 in x space

**Lemma 3:** ⑥
$\mathcal{M} \nparallel \mathcal{P}, L = \mathcal{M} \cap \mathcal{P}$, L has lower dim. It's measure is 0 in $\mathcal{M}$&$\mathcal{P}$

**Theorem 1:** ②
If support of $P_r$&$P_g$ $\mathcal{M}$&$\mathcal{P}$ has no overlap, then we'll always get $D^*$

**Theorem 2:** ⑦
If support of $P_r$&$P_g$ $\mathcal{M} \nparallel \mathcal{P}$ and set A has 0 measure in $\mathcal{M}$&$\mathcal{P}$, then we'll get $D^*$

**Theorem 3:**
If $P_r$&$P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$,
$JSD(P_r \parallel P_g) = \log 2$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

**Lemma 2.** *Let $\mathcal{M}$ and $\mathcal{P}$ be two regular submanifolds of $\mathbb{R}^d$ that don't have full dimension. Let $\eta, \eta'$ be arbitrary independent continuous random variables. We therefore define the perturbed manifolds as $\tilde{\mathcal{M}} = \mathcal{M} + \eta$ and $\tilde{\mathcal{P}} = \mathcal{P} + \eta'$. Then*

$$\mathbb{P}_{\eta,\eta'}(\tilde{\mathcal{M}} \text{ does not perfectly align with } \tilde{\mathcal{P}}) = 1$$

**Definition 2:** ④
Perfectly Align:
$T_x\mathcal{M} + T_x\mathcal{P} \neq T_x\mathcal{F}$

**Definition 1:** ③
Transversality:
$T_x\mathcal{M} + T_x\mathcal{P} = T_x\mathcal{F}$

**Lemma 2:** ⑤
$\mathbb{p}(\widetilde{\mathcal{M}} \nparallel \widetilde{\mathcal{P}}) = 1$

1. $\widetilde{\mathcal{M}} = \mathcal{M} + \eta$ / $\widetilde{\mathcal{P}} = \mathcal{P} + \eta'$
2. It's hardly possible that $P_r$ is perfectly align to $P_g$

**We know:** ⓪
$P_r$ is low dimensional manifold in high dimensional space

**Lemma 1:** ①
Measure of $g(z)$ is 0 in x space

**Lemma 3:** ⑥
$\mathcal{M} \nparallel \mathcal{P}, L = \mathcal{M} \cap \mathcal{P}$, L has lower dim. It's measure is 0 in $\mathcal{M}$&$\mathcal{P}$

**Theorem 1:** ②
If support of $P_r$&$P_g$ $\mathcal{M}$&$\mathcal{P}$ has no overlap, then we'll always get $D^*$

**Theorem 2:** ⑦
If support of $P_r$&$P_g$ $\mathcal{M} \nparallel \mathcal{P}$ and set A has 0 measure in $\mathcal{M}$&$\mathcal{P}$, then we'll get $D^*$

**Theorem 3:**
If $P_r$&$P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$,
$JSD(P_r \parallel P_g) = \log 2$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

**Lemma 3.** *Let $\mathcal{M}$ and $\mathcal{P}$ be two regular submanifolds of $\mathbb{R}^d$ that don't perfectly align and don't have full dimension. Let $\mathcal{L} = \mathcal{M} \cap \mathcal{P}$. If $\mathcal{M}$ and $\mathcal{P}$ don't have boundary, then $\mathcal{L}$ is also a manifold, and has strictly lower dimension than both the one of $\mathcal{M}$ and the one of $\mathcal{P}$. If they have boundary, $\mathcal{L}$ is a union of at most 4 strictly lower dimensional manifolds. In both cases, $\mathcal{L}$ has measure 0 in both $\mathcal{M}$ and $\mathcal{P}$.*

**Definition 2:** ④
Perfectly Align:
$T_x\mathcal{M} + T_x\mathcal{P} \neq T_x\mathcal{F}$

**Definition 1:** ③
Transversality:
$T_x\mathcal{M} + T_x\mathcal{P} = T_x\mathcal{F}$

**Lemma 2:** ⑤
$\mathbb{P}(\widetilde{\mathcal{M}} \pitchfork \widetilde{\mathcal{P}}) = 1$

**We know:** ⓪
$P_r$ is low dimensional manifold in high dimensional space

**Lemma 1:** ①
Measure of $g(z)$ is 0 in x space

**Lemma 3:** ⑥
$\mathcal{M} \pitchfork \mathcal{P}, L = \mathcal{M} \cap \mathcal{P}$, L has lower dim. It's measure is 0 in $\mathcal{M}\&\mathcal{P}$

**Theorem 1:** ②
If support of $P_r\&P_g$ $\mathcal{M}\&\mathcal{P}$ has no overlap, then we'll always get $D^*$

**Theorem 2:** ⑦
If support of $P_r\&P_g$ $\mathcal{M} \pitchfork \mathcal{P}$ and set A has 0 measure in $\mathcal{M}\&\mathcal{P}$, then we'll get $D^*$

**Theorem 3:**
If $P_r\&P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \pitchfork \mathcal{P}$,
$JSD(P_r \parallel P_g) = \log 2$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

**Definition 2:** ④
Perfectly Align:
$T_x\mathcal{M} + T_x\mathcal{P} \neq T_x\mathcal{F}$

**Definition 1:** ③
Transversality:
$T_x\mathcal{M} + T_x\mathcal{P} = T_x\mathcal{F}$

**Lemma 2:** ⑤
$\mathbb{p}(\widetilde{\mathcal{M}} \nparallel \widetilde{\mathcal{P}}) = 1$

**We know:** ⓪
$P_r$ is low dimensional manifold in high dimensional space

**Lemma 1:** ①
Measure of $g(z)$ is 0 in x space

**Lemma 3:** ⑥
$\mathcal{M} \nparallel \mathcal{P}, L = \mathcal{M} \cap \mathcal{P}$, L has lower dim. It's measure is 0 in $\mathcal{M}$&$\mathcal{P}$

**Theorem 2.2.** *Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions that have support contained in two closed manifolds $\mathcal{M}$ and $\mathcal{P}$ that don't perfectly align and don't have full dimension. We further assume that $\mathbb{P}_r$ and $\mathbb{P}_g$ are continuous in their respective manifolds, meaning that if there is a set $A$ with measure 0 in $\mathcal{M}$, then $\mathbb{P}_r(A) = 0$ (and analogously for $\mathbb{P}_g$). Then, there exists an optimal discriminator $D^* : \mathcal{X} \to [0, 1]$ that has accuracy 1 and for almost any $x$ in $\mathcal{M}$ or $\mathcal{P}$, $D^*$ is smooth in a neighbourhood of $x$ and $\nabla_x D^*(x) = 0$.*

**Theorem 1:** ②
If support of $P_r$&$P_g$ $\mathcal{M}$&$\mathcal{P}$ has no overlap, then we'll always get $D^*$

**Theorem 2:** ⑦
If support of $P_r$&$P_g$ $\mathcal{M} \nparallel \mathcal{P}$ and set A has 0 measure in $\mathcal{M}$&$\mathcal{P}$, then we'll get $D^*$

Just like Theorem 1

1. If $P_r$&$P_g$ has supports $\mathcal{M} \cap \mathcal{P} = \emptyset$, We'll always have optimal discriminator $D^*$
2. $P_r(D^*(x = 1)) = 1, P_g(D^*(x = 0)) = 1$

**Theorem 3:**
If $P_r$&$P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$,
$JSD(P_r \parallel P_g) = \log 2$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

**We know:** ⓪
$P_r$ is low dimensional manifold in high dimensional space

**Lemma 1:** ①
Measure of $g(z)$ is 0 in x space

**Theorem 1:** ②
If $P_r$&$P_g$support of $\mathcal{M}$&$\mathcal{P}$ has no overlap, then we'll always get $D^*$

**Definition 1:** ③
Transversality:
$T_x\mathcal{M} + T_x\mathcal{P} = T_x\mathcal{F}$

**Definition 2:** ④
Perfectly Align:
$T_x\mathcal{M} + T_x\mathcal{P} \neq T_x\mathcal{F}$

**Lemma 2:** ⑤
$\mathbb{p}(\widetilde{\mathcal{M}} \nparallel \widetilde{\mathcal{P}}) = 1$

**Lemma 3:** ⑥
$\mathcal{M} \nparallel \mathcal{P}, L = \mathcal{M} \cap \mathcal{P}$, L has lower dim. It's measure is 0 in $\mathcal{M}$&$\mathcal{P}$

**Theorem 2:** ⑦
If support of $P_r$&$P_g$ $\mathcal{M} \nparallel \mathcal{P}$ and set A has 0 measure in $\mathcal{M}$&$\mathcal{P}$, then we'll get $D^*$

***Questions II*: Can we get/converge to the global optimality?**

**Theorem 2.3.** *Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions whose support lies in two manifolds $\mathcal{M}$ and $\mathcal{P}$ that don't have full dimension and don't perfectly align. We further assume that $\mathbb{P}_r$ and $\mathbb{P}_g$ are continuous in their respective manifolds. Then,*

$$JSD(\mathbb{P}_r \| \mathbb{P}_g) = \log 2$$
$$KL(\mathbb{P}_r \| \mathbb{P}_g) = +\infty$$
$$KL(\mathbb{P}_g \| \mathbb{P}_r) = +\infty$$

**Theorem 3:**
If $P_r$&$P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$,
$JSD(P_r \| P_g) = \log 2$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

**Theorem 3:**
**If $P_r$&$P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$, $JSD(P_r \parallel P_g) = \log 2$**

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

**Theorem 3:**
**If $P_r$&$P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$, $JSD(P_r \parallel P_g) = \log 2$**

$$f(D^*) = \int_x p_{data}(\boldsymbol{x}) \log\left(\frac{2p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} - log2\right) + p_g(\boldsymbol{x}) \log\left(\frac{2p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} - log2\right) dx$$

$$= -log4 + D_{KL}\left(p_{data} \parallel \frac{p_{data} + P_g}{2}\right) + D_{KL}\left(p_g \parallel \frac{p_{data} + P_g}{2}\right)$$

$$= -log4 + 2\boldsymbol{JSD}\left(\boldsymbol{p_{data}} \parallel \boldsymbol{p_g}\right) = -log4 + 2\boldsymbol{log2}$$

$$\boldsymbol{= 0}$$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

**C.1 If $D = D^*$**

**Theorem 3:**
**If $P_r$&$P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$, $JSD(P_r \parallel P_g) = \log 2$**

**Theorem 3:**
**If we have $D^*$, then we can LEARN NOTHING**

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

C.1 If $D = D^*$

**Theorem 3:**
**If $P_r$&$P_g$ is lower dimensional manifold in a higher space and their supports $\mathcal{M} \nparallel \mathcal{P}$, $JSD(P_r \parallel P_g) = \log 2$**

**WHY!**

# I. GAN: Theories

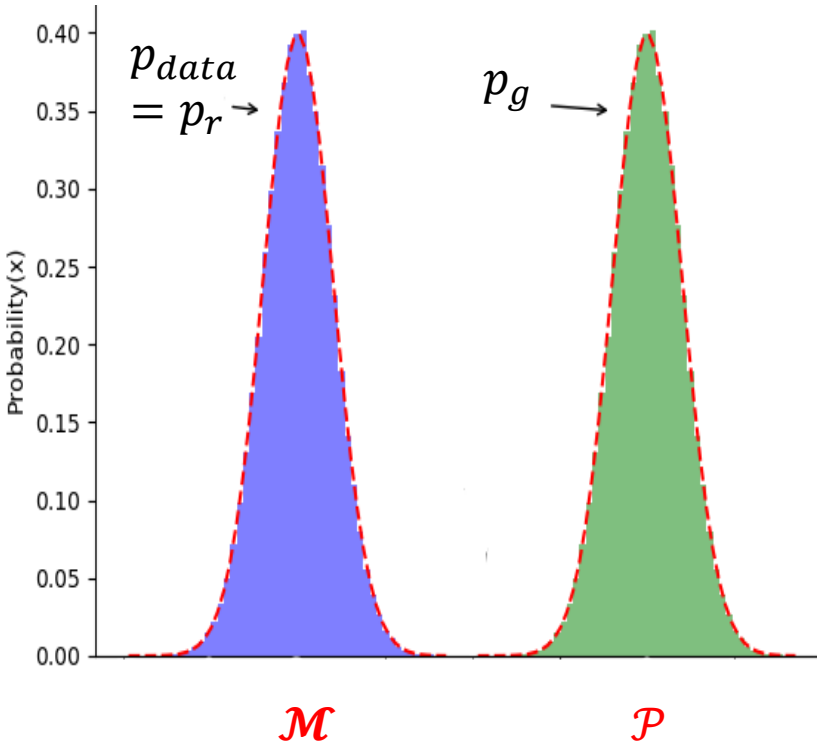
## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

$$
\begin{aligned}
& f(D^*) \\
&= -log4 + \int p_{data} \log\left(\frac{2p_{data}}{p_{data} + p_g}\right) dx + \int p_g \log\left(\frac{2p_g}{p_{data} + p_g}\right) dx \\
&= -log4 + 1 \cdot \int_{x \in \mathcal{M}} \log\left(\frac{2p_{data}}{p_{data} + 0}\right) dx + 1 \cdot \int_{x \in \mathcal{P}} \log\left(\frac{2p_g}{0 + p_g}\right) dx \\
&= -log4 + 1 \cdot log2 \cdot \int_{x \in \mathcal{M}} 1 dx + 1 \cdot log2 \cdot \int_{x \in \mathcal{P}} 1 dx \\
&= -log4 + 2log2 \\
&= 0
\end{aligned}
$$

$$\boldsymbol{JSD(P_r \parallel P_g) = \log 2}$$

***Questions II*:**
**Can we get to the global optimality?**



# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

***Questions II*: Can we get/converge to the global optimality?**

**No**

***When $D = D^*$:***

$$Original\ Formula =$$

$$C(G) = -log4 + 2JSD(p_r \parallel p_g)$$

$$\geq -log4$$

$$= -log4 + 2 * log2\ (p_r \nparallel p_g)$$

$$= 0$$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.1 If $D = D^*$

***Questions II*: Can we get/converge to the global optimality?**

**No**

$$\boldsymbol{When\ D = D^*:}$$

$$\begin{aligned} &Original\ Formula = \\ &C(G) = -log4 + 2JSD(p_r \parallel p_g) \\ &\quad \geq -log4 \\ &\quad = -log4 + 2 * log2\ (p_r \nparallel p_g) \\ &\quad = 0 \end{aligned}$$

**There is no loss at all.**

**When we have the best discriminator, we cannot train at all.**

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.2 If $D \to D^*$

$\boldsymbol{Questions\ III}$: You are talking about $\boldsymbol{D = D^*}$. But it's too strict.
What about $\boldsymbol{D \to D^*}$?

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.2 If $D \to D^*$

**$Questions\ III$: You are talking about $D = D^*$. But it's too strict. What about $D \to D^*$?**

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

$$\mathbb{E}_{z\sim p_z(z)}[-\log D(G(z))]$$

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.2 If $D \to D^*$

***Questions III***: **You are talking about $D = D^*$. But it's too strict. What about $D \to D^*$?**

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

$$\mathbb{E}_{z\sim p_z(z)}[-\log D(G(z))]$$

**Theorem 4:**

If $\| D - D^* \| < \varepsilon$ and $\| J_\theta G(z) \|_2 \leq M^2$, then

$$\| \nabla_\theta \mathbb{E}_{z\sim p_z(z)} \left[ log\left(1 - D\left(G(z)\right)\right)\right] \|_2 < M\frac{\varepsilon}{1-\varepsilon}$$

**Meaning:**

When D is Getting better,
Our loss is heading to zero.

**Vanishing Gradient**

**Corollary 1:**

$$\lim_{\|D - D^*\| \to 0} \nabla_\theta \mathbb{E}_{z\sim p_z(z)} \left[ log\left(1 - D\left(G(z)\right)\right)\right] = 0$$

**The better the discriminator,**
**The harder to train the system**

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.2 If $D \to D^*$

***Questions IV*: Why mode collapse?**

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

$$\mathbb{E}_{z\sim p_z(z)}[-\log D(G(z))] \quad ②$$

**Let's do some calculation!**

Let $D = D^* = \frac{p_r}{p_g + p_r}$, then we have:

$\mathbb{E}_{x\sim p_r}[log D^*(x)] + \mathbb{E}_{x\sim p_g}[\log(1-D^*(x))]$

$= -2log2 + 2JSD(p_{g_\theta} || p_r)$

*And because*

$KL(p_g || p_r) = \mathbb{E}_{x\sim p_g}\left[\log \frac{p_g}{p_r}\right] =$

$\mathbb{E}_{x\sim p_g}\left[\log \frac{\frac{p_g}{p_r+p_g}}{\frac{p_r}{p_r+p_g}}\right] = \mathbb{E}_{x\sim p_g}\left[\log \frac{1-D^*}{D^*}\right] =$

$\mathbb{E}_{x\sim p_g}[\log(1-D^*)] - \mathbb{E}_{x\sim p_g}[\log(D^*)]$

**Then**

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.2 If $D \to D^*$

***Questions IV*: Why mode collapse?**

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

$$\mathbb{E}_{z\sim p_z(z)}[-\log D(G(z))] \quad ②$$

**Theorem 5:**

If $D^* = \frac{p_r}{p_{g_\theta}+p_r}$, then

$$\mathbb{E}_{z\sim p_z(z)}\left[-\nabla_\theta \log D^*(G(z))\big|_{\theta=\theta_0}\right]$$
$$= \nabla_\theta [KL(p_g \,||\, p_r) - JSD(p_g \,||\, p_r)]|_{\theta=\theta_0}$$

Ⓐ Ⓑ

**Problem I:**

Ⓐ ↑

Ⓑ ↓

**Problem II:** $KL(p_g \,||\, p_r) = \int_x p_g \log \frac{p_g}{p_r} dx$

- $p_g \to 0, p_r \to 1, KL(\cdot) \to 0$
- $p_g \to 1, p_r \to 0, KL(\cdot) \to \infty$

**Mode Collapse**

# I. GAN: Theories

## C. WGAN I [2017 Arjovsky]

### C.2 If $D \to D^*$

***Summary***:

$\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$ ①

**When D is Getting better,**
**Our loss is heading to zero.**

$\mathbb{E}_{z\sim p_z(z)}[-\log D(G(z))]$ ②

**KL Divergence is not reasonable.**
**2 Reasons**

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.1 Wasserstein Distance [Earth-Mover Distance]

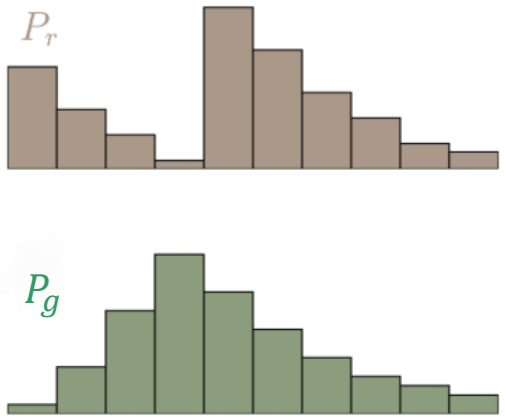
***Wasserstein Distance*:** Compare distance of 2 distributions

$$W(\mathbb{P}_r, \mathbb{P}_g) = \underset{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)}{\inf} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

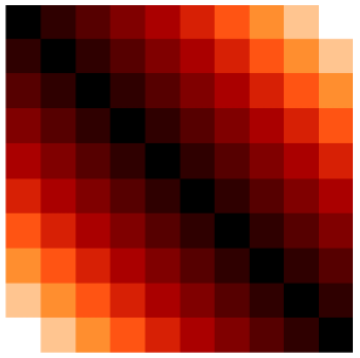
③ ② ①

$$① \ \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|] = \int_y \int_x \gamma(x,y) \| x - y \| dx\, dy = \sum_{x,y} \| x - y \| \gamma(x,y)$$

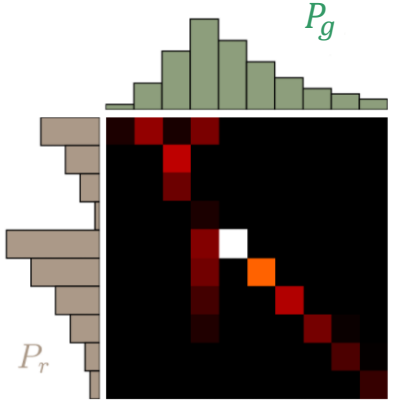
- **What is $x, y$? $\gamma(x,y)$? $\| x - y \|$? Why E?**




$D$: Distance Matrix



D




Π

$\Pi = \gamma(x,y)$

由$\mathbb{P}_r(\mathbb{P}_g)$变成$\mathbb{P}_g(\mathbb{P}_r)$所需要的变换。

$\sum_x \gamma(x,y) = \mathbb{P}_r(y))$

$\sum_y \gamma(x,y) = \mathbb{P}_g(x))$

① x与y距离的加权和;
权: $\gamma(x,y)$

所以用E[期望]来表示

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.1 Wasserstein Distance [Earth-Mover Distance]

***Wasserstein Distance***: Compare distance of 2 distributions

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|] \quad \left【 \sum_{x,y} \| x - y \| \gamma(x,y) 】\right.$$

③ ② ①

② $\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)$:

- What is $\Pi$?

  $\boldsymbol{\Pi}$: is the set of all transforms from $\mathbb{P}_r(\mathbb{P}_g)$ to $\mathbb{P}_g(\mathbb{P}_r)$.
  We have myriads of ways. It means ALL POSSIBILITIES.
  That's why we need

- What is $\gamma$?

  $\boldsymbol{\gamma}$: is the one joint distribution whose margins are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$

③ inf

- infimum: 下确界。集合中的最小值
  In order to get the minimum value (distance) between $\mathbb{P}_r$ *and* $\mathbb{P}_g$

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.2 How to Solve

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

[supremum]

$$= \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

Why?

$$= \max_{w\in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

Why??

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.2 How to Solve

$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \max_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)] \qquad \text{Why?}$$

$$= \max_{w \in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))] \qquad \text{Why??}$$

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.2 How to Solve

$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \max_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

**Kantorovich-Rubinstein Duality,**
original explanation, 中文对应详解版本

$$= \max_{w \in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

**Lipschitz**

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.2 How to Solve

$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \max_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

**Kantorovich-Rubinstein Duality,**
original explanation, 中文对应详解版本

$$= \max_{w \in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

**Lipschitz**

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.3 KR Duality

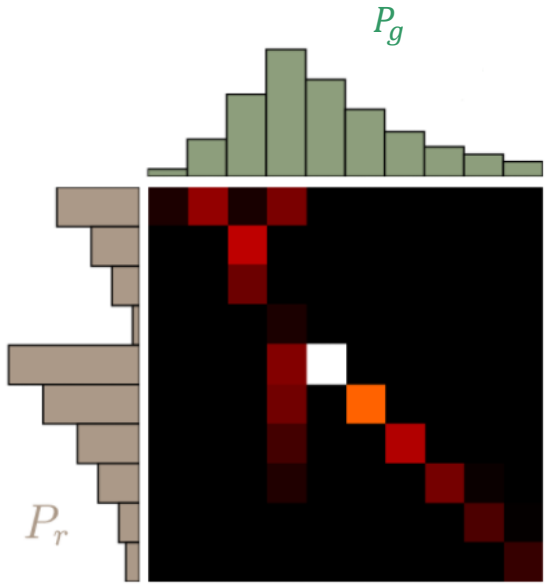
$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \int\int \gamma(x,y) \| x - y \| \, dxdy$$

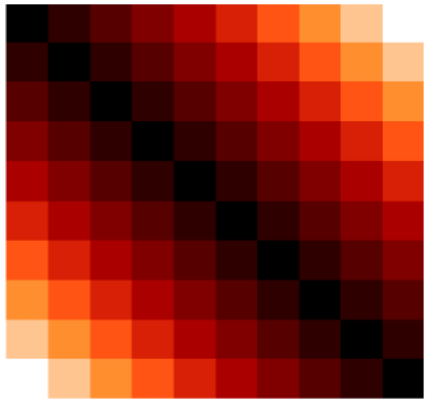
$$\text{s.t.} \int \gamma(x,y)\, dy = \mathbb{P}_r(x), \qquad \sum_x \gamma(x,y) = \mathbb{P}_r$$

$$\int \gamma(x,y)\, dx = \mathbb{P}_g(y), \qquad \sum_y \gamma(x,y) = \mathbb{P}_g$$

$$\gamma(x,y) \geq 0$$




$\gamma(x,y)$



$\| x - y \|$

D

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.3 KR Duality

$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \int\int \gamma(x,y) \| x - y \| \, dxdy$$

$$= \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \{\langle \Pi, D \rangle | A\Pi = b, \Pi \geq 0\}$$

$$\text{s.t.} \sum_y \gamma(x,y) = \mathbb{P}_r(x)$$

$$\sum_x \gamma(x,y) = \mathbb{P}_g(y)$$

$$\gamma(x,y) \geq 0$$

$$\Pi = \begin{pmatrix} \gamma(x_1,y_1) \\ \gamma(x_1,y_2) \\ \vdots \\ \gamma(x_2,y_1) \\ \gamma(x_2,y_2) \\ \vdots \\ \gamma(x_n,y_1) \\ \gamma(x_n,y_2) \\ \vdots \end{pmatrix} \quad D = \begin{pmatrix} d(x_1,y_1) \\ d(x_1,y_2) \\ \vdots \\ d(x_2,y_1) \\ d(x_2,y_2) \\ \vdots \\ d(x_n,y_1) \\ d(x_n,y_2) \\ \vdots \end{pmatrix}$$

$$\underbrace{\begin{pmatrix} 1 & 1 & \dots & 0 & 0 & \dots & \dots & 0 & 0 & \dots & \dots \\ 0 & 0 & \dots & 1 & 1 & \dots & \dots & 0 & 0 & \dots & \dots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots \\ 0 & 0 & \dots & 0 & 0 & \dots & \dots & 1 & 1 & \dots & \dots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots \\ \hline 1 & 0 & \dots & 1 & 0 & \dots & \dots & 1 & 0 & \dots & \dots \\ 0 & 1 & \dots & 0 & 1 & \dots & \dots & 0 & 1 & \dots & \dots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots \\ 0 & 0 & \dots & 0 & 0 & \dots & \dots & 0 & 0 & \dots & \dots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} \gamma(x_1,y_1) \\ \gamma(x_1,y_2) \\ \vdots \\ \gamma(x_2,y_1) \\ \gamma(x_2,y_2) \\ \vdots \\ \gamma(x_n,y_1) \\ \gamma(x_n,y_2) \\ \vdots \end{pmatrix}}_{\Pi} = \underbrace{\begin{pmatrix} p_r(x_1) \\ p_r(x_2) \\ \vdots \\ p_r(x_n) \\ \vdots \\ p_g(y_1) \\ p_g(y_2) \\ \vdots \\ p_g(y_n) \\ \vdots \end{pmatrix}}_{b}$$

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.3 KR Duality

$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \int\int \gamma(x,y) \| x - y \| dxdy$$

$$= \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \{\langle \Pi, D \rangle | A\Pi = b, \Pi \geq 0\}$$

$\Leftrightarrow$

$$\min_x = \{c^T x | Ax = b, x \geq 0\}$$

$$=$$

$$\max_y = \{b^T y | A^T y \leq c\}$$

**This is a general linear programming task.**

**This is the target KR Duality.**

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.3 KR Duality

$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \int\int \gamma(x,y) \| x - y \| dxdy$$

$$= \min_{\Pi}\{\langle \Pi, D\rangle | A\Pi = b, \Pi \geq 0\}$$

$\Leftrightarrow$

$$= \max_{F}\{\langle b, F\rangle | A^T F \leq D\}$$

$$\langle b, F\rangle = \sum_i p_r(x_i) f(x_i) + \sum_i p_g(y_i) g(y_i)$$

$\Leftrightarrow$

$$\Leftrightarrow \forall x, y, f(x) + g(y) \leq d(x,y)$$
$$\Leftrightarrow \forall x, f(x) + g(x) \leq d(x,x) = 0$$
$$\Leftrightarrow \forall x, g(x) \leq -f(x)$$
$$\Leftrightarrow \forall x, p_r(x)f(x) + p_g(y)g(y) \leq p_r(x)f(x) - p_g(x)f(x)$$

$$= \max_{f}\{\int [p_r(x)f(x) - p_g(x)f(x)]dx \,|f(x) - f(y) \leq d(x,y)\}$$

$$\min_x = \{c^T x | Ax = b, x \geq 0\}$$
$$=$$
$$\max_y = \{b^T y | A^T y \leq c\}$$

$$b = \begin{pmatrix} p_r(x_1) \\ p_r(x_2) \\ \vdots \\ p_r(x_n) \\ \vdots \\ \hline p_g(y_1) \\ p_g(y_2) \\ \vdots \\ p_g(y_n) \\ \vdots \end{pmatrix}$$

$$\underbrace{\left(\begin{array}{cccc|cccc} 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 1 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \dots & 1 & 1 & 0 & \dots & 0 \\ 0 & 0 & \ddots & 1 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right)}_{A^\top} \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \\ \vdots \\ \hline g(y_1) \\ g(y_2) \\ \vdots \\ g(y_n) \\ \vdots \end{pmatrix}}_{F} \leq \underbrace{\begin{pmatrix} d(x_1, y_1) \\ d(x_1, y_2) \\ \vdots \\ \hline d(x_2, y_1) \\ d(x_2, y_2) \\ \vdots \\ \hline \vdots \\ \hline d(x_n, y_1) \\ d(x_n, y_2) \\ \vdots \\ \hline \vdots \end{pmatrix}}_{D}$$

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.3 KR Duality

$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|] = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \int\int \gamma(x,y) \| x - y \| dxdy$$

$$= \min_{\Pi}\{\langle \Pi, D\rangle | A\Pi = b, \Pi \geq 0\}$$

$$\Leftrightarrow$$

$$= \max_{F}\{\langle b, F\rangle | A^T F \leq D\}$$

$$\Leftrightarrow$$

$$= \max_{f}\{\int [p_r(x)f(x) - p_g(x)f(x)]dx \,|f(x) - f(y) \leq d(x,y)\}$$

$$= \max_{f, \|f\|_L \leq 1} \mathbb{E}_{x\sim p_r(x)}[f(\boldsymbol{x})] - \mathbb{E}_{x\sim p_g(x)}[f(\boldsymbol{x})]$$

这里指f满足L约束 (Lipschitz Constraint)

$$\min_{x} = \{c^T x | Ax = b, x \geq 0\}$$
$$=$$
$$\max_{y} = \{b^T y | A^T y \leq c\}$$

$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \max_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

$$= \max_{w \in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.4 Lipschitz Constraint

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

**Kantorovich-Rubinstein**

$$= \max_{w\in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

**Lipschitz Constraint**

中文解析版本
英文详解版本

**If we have function $f(x)$ and there is an $L$ which is greater than $0$ to make**

$$\| f(x_1) - f(x_2) \| \leq L \| x_1 - x_2 \|$$

**Then we say the function satisfies Lipschitz constraint where $L$ is called Lipschitz constant.**

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.4 Lipschitz Constraint

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

**Kantorovich-Rubinstein**

$$= \max_{w\in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

**Lipschitz Constraint**

**If we have function $f(x)$ and there is an $L$ which is greater than $0$ to make**

$$\| f(x_1) - f(x_2) \| \leq L \| x_1 - x_2 \|$$

**Then we say the function satisfies Lipschitz constraint where $L$ is called Lipschitz constant.**

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.4 Lipschitz Constraint

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

**Kantorovich-Rubinstein**

$$= \max_{w\in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

**Lipschitz Constraint**

**If we have function $f(x)$ and there is an $L$ which is greater than $0$ to make**

$$\| f(x_1) - f(x_2) \| \leq L \| x_1 - x_2 \|$$

**Then we say the function satisfies Lipschitz constraint where $L$ is called Lipschitz constant.**

**Here we have**

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

**If** $\| f \|_L \leq K\ (K - Lipschitz)$, **then we have** $K \cdot W(\mathbb{P}_r, \mathbb{P}_g)$

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.4 Lipschitz Constraint

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

**Kantorovich-Rubinstein**

$$= \max_{w\in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

**Lipschitz Constraint**

**If we have function $f(x)$ and there is an $L$ which is greater than $0$ to make**

$$\| f(x_1) - f(x_2) \| \leq L \| x_1 - x_2 \|$$

**Then we say the function satisfies Lipschitz constraint where $L$ is called Lipschitz constant.**

**Here we have**

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

**If $\| f \|_L \leq K$ $(K - Lipschitz)$, then we have $K \cdot W(\mathbb{P}_r, \mathbb{P}_g)$**

**So if we have a set of $\{f_w\}_{,w\in W}$ where they all under K-Lipschitz**
**Then**

$$W(\mathbb{P}_r, \mathbb{P}_g) = \max_{w\in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.4 Lipschitz Constraint

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

$$= \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

**Kantorovich-Rubinstein**

$$= \max_{w\in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

**Lipschitz Constraint**

If we have function $f(x)$ and there is an $L$ which is greater than $0$ to make

$$\| f(x_1) - f(x_2) \| \leq L \| x_1 - x_2 \|$$

Then we say the function satisfies Lipschitz constraint where $L$ is called Lipschitz constant.

Here we have

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

If $\| f \|_L \leq K$ $(K - Lipschitz)$, then we have $K \cdot W(\mathbb{P}_r, \mathbb{P}_g)$

So if we have a set of $\{f_w\}_{,w\in W}$ where they all under K-Lipschitz
Then

$$W(\mathbb{P}_r, \mathbb{P}_g) = \max_{w\in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f_w(g_\theta(z))]$$

# I. GAN: Theories

## D. WGAN II [2017 Arjovsky]

### D.5 Algorithm

**Algorithm 1** WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

**Require:** : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size.
$n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
**Require:** : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.

**while** $\theta$ has not converged **do**
  **for** $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  **end for**
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
**end while**

**Wasserstein Distance**

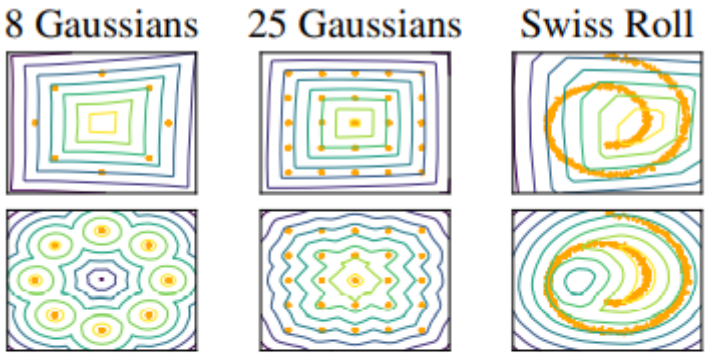
**Lipschitz Constraint**

# I. GAN: Theories

## E. WGAN III-[WGAN-GP] [2017 Gulrajani]

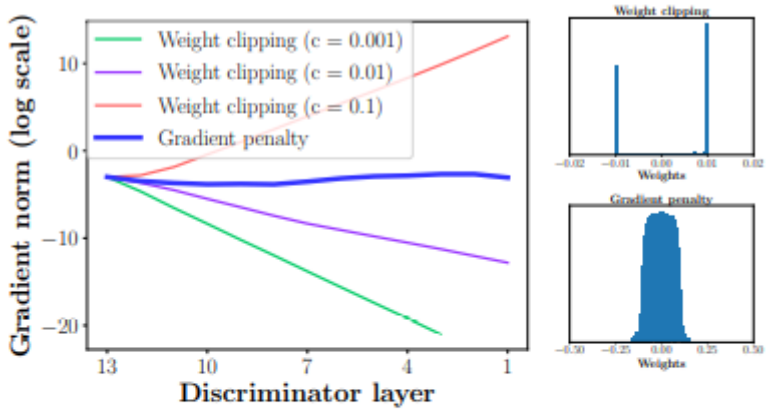
### E.1. WGAN Problems

- **Capacity underuse（模型泛化能力弱）**




(a) Value surfaces of WGAN critics trained to optimality on toy datasets using (top) weight clipping and (bottom) gradient penalty. Critics trained with weight clipping fail to capture higher moments of the data distribution. The 'generator' is held fixed at the real data plus Gaussian noise.

- **Exploding & vanishing gradients**



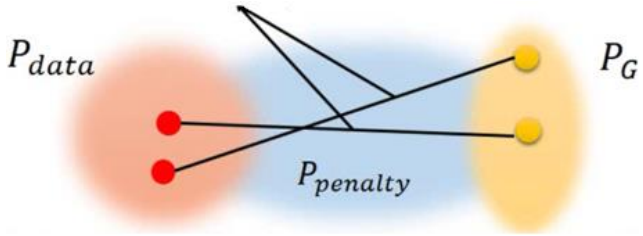

(b) (left) Gradient norms of deep WGAN critics during training on the Swiss Roll dataset either explode or vanish when using weight clipping, but not when using a gradient penalty. (right) Weight clipping (top) pushes weights towards two values (the extremes of the clipping range), unlike gradient penalty (bottom).

# I. GAN: Theories

## E. WGAN III-[WGAN-GP] [2017 Gulrajani]

### E.2. Solution: Gradient Penalty

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \overbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_g}[(\| \nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}}) \|_2 - 1)^2]}^{\text{Our gradient penalty}}$$




**Algorithm 1** WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

**Require:** The gradient penalty coefficient $\lambda$, the number of critic iterations per generator iteration $n_{\text{critic}}$, the batch size $m$, Adam hyperparameters $\alpha, \beta_1, \beta_2$.
**Require:** initial critic parameters $w_0$, initial generator parameters $\theta_0$.

**while** $\theta$ has not converged **do**
  **for** $t = 1, ..., n_{\text{critic}}$ **do**
    **for** $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    **end for**
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  **end for**
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
**end while**

# II. GAN Applications -Preparation

# II. GAN: Applications

## F. Training pipeline

### Training order of G(enerator) & D(iscriminator)

```
def optimize_parameters(self):
    self.forward()                   # compute fake images: G(A)
    # update D
    self.set_requires_grad(self.netD, True)  # enable backprop for D
    self.optimizer_D.zero_grad()     # set D's gradients to zero
    self.backward_D()                # calculate gradients for D
    self.optimizer_D.step()          # update D's weights
    # update G
    self.set_requires_grad(self.netD, False)  # D requires no gradients when optimizing G
    self.optimizer_G.zero_grad()        # set G's gradients to zero
    self.backward_G()                   # calculate graidents for G
    self.optimizer_G.step()             # update G's weights
```

- **Train D & G separately**
- **Training order of D & G is not required**
  not necessary to D first, vice versa
- **Relative order of inner G & D is needed**
  could add other contents between each line
- **Training times for D & G is not required**
  In earlier time, people tend to train > 1 times, now, people tend to train both G & D 1/loop
- **Link of this example**
  Link, from line 116

# II. GAN: Applications

## F. Training pipeline

**model.zero_grad or optimizer.zero_grad**

```
def optimize_parameters(self):
    self.forward()                   # compute fake images: G(A)
    # update D
    self.set_requires_grad(self.netD, True)  # enable backprop for D
    self.optimizer_D.zero_grad()     # set D's gradients to zero
    self.backward_D()                # calculate gradients for D
    self.optimizer_D.step()          # update D's weights
    # update G
    self.set_requires_grad(self.netD, False)  # D requires no gradients when optimizing G
    self.optimizer_G.zero_grad()        # set G's gradients to zero
    self.backward_G()                   # calculate graidents for G
    self.optimizer_G.step()             # udpate G's weights
```

```
netD.zero_grad()
real_cpu, label = data
batch_size = real_cpu.size(0)
if opt.cuda:
    real_cpu = real_cpu.cuda()
input.data.resize_as_(real_cpu).copy_(real_cpu)
dis_label.data.resize_(batch_size).fill_(real_label)
aux_label.data.resize_(batch_size).copy_(label)
dis_output, aux_output = netD(input)

dis_errD_real = dis_criterion(dis_output, dis_label)
aux_errD_real = aux_criterion(aux_output, aux_label)
errD_real = dis_errD_real + aux_errD_real
errD_real.backward()
```

- **If 1 model vs 1 optimizer, then no difference**
- **If multiple model vs multiple optimizer, then it matters**
  model.zero_grad set all parameters non-trainable
- **Link for the new example**
  Link, from line 176

# II. GAN: Applications

## F. Training pipeline

**parameters.require_grad=True/False**

```
def optimize_parameters(self):
    self.forward()                   # compute fake images: G(A)
    # update D
    self.set_requires_grad(self.netD, True)  # enable backprop for D
    self.optimizer_D.zero_grad()     # set D's gradients to zero
    self.backward_D()                # calculate gradients for D
    self.optimizer_D.step()          # update D's weights
    # update G
    self.set_requires_grad(self.netD, False)  # D requires no gradients when optimizing G
    self.optimizer_G.zero_grad()        # set G's gradients to zero
    self.backward_G()                   # calculate graidents for G
    self.optimizer_G.step()             # udpate G's weights
```

```
def set_requires_grad(self, nets, requires_grad=False):
    """Set requies_grad=Fasle for all the networks to avoid unnecessary computations
    Parameters:
        nets (network list)   -- a list of networks
        requires_grad (bool)  -- whether the networks require gradients or not
    """
    if not isinstance(nets, list):
        nets = [nets]
    for net in nets:
        if net is not None:
            for param in net.parameters():
                param.requires_grad = requires_grad
```

- **Set parameters in a model trainable or not**
- **Not required for this specific**
  2 optimizers are used to train 2 different models
- **Needed if use 1 optimizer to train multiple models**
- **Link for the new example**
  Link, from line 219

# II. GAN: Application

## F. Training pipeline

**Training with original GAN: D is classifier**

- **Relative order of — and - - is required.**
  **Order / Position of -[ is not required**

- **Train G:** fake —> true
  **Train D:** fake —> false
  real —> true

- **d_loss could backward separately**
  d_loss.backward ⇔ real_loss.backward, fake_loss...
  Link, line 189 & 212
  therefore, /2 is not required

- **Link of this example**
  This link is really good. Tons of examples are there

```
optimizer_G.zero_grad()

# Sample noise as generator input
z = Variable(Tensor(np.random.normal(0, 1, (imgs.shape[0], opt.latent_dim))))

# Generate a batch of images
gen_imgs = generator(z)

# Loss measures generator's ability to fool the discriminator
g_loss = adversarial_loss(discriminator(gen_imgs), valid)

g_loss.backward()
optimizer_G.step()

# ---------------------
#  Train Discriminator
# ---------------------

optimizer_D.zero_grad()

# Measure discriminator's ability to classify real from generated samples
real_loss = adversarial_loss(discriminator(real_imgs), valid)
fake_loss = adversarial_loss(discriminator(gen_imgs.detach()), fake)
d_loss = (real_loss + fake_loss) / 2

d_loss.backward()
optimizer_D.step()
```

BCELoss

# II. GAN: Applicatio

## F. Training pipeline

**Training with WGAN: D is MSE**

- **Relative order of — and - - is required.**
  **Order / Position of { is not required**
- **No BCELoss, just simple "-"**
- **"Clamp" for WGAN**
  Line 140
  **Gradient-Penalty for WGAN-GP**
  Line 149 & 169
- **Link of this example**
  This link is really good. Tons of examples are there

```
# ---------------------
#  Train Discriminator
# ---------------------

optimizer_D.zero_grad()

# Sample noise as generator input
z = Variable(Tensor(np.random.normal(0, 1, (imgs.shape[0], opt.latent_dim))))

# Generate a batch of images
fake_imgs = generator(z).detach()
# Adversarial loss
loss_D = -torch.mean(discriminator(real_imgs)) + torch.mean(discriminator(fake_imgs))

loss_D.backward()
optimizer_D.step()


# -----------------
#  Train Generator
# -----------------

optimizer_G.zero_grad()

# Generate a batch of images
gen_imgs = generator(z)
# Adversarial loss
loss_G = -torch.mean(discriminator(gen_imgs))

loss_G.backward()
optimizer_G.step()
```

# II. GAN: Applications

## F. Training pipeline

**Practical / Real code**

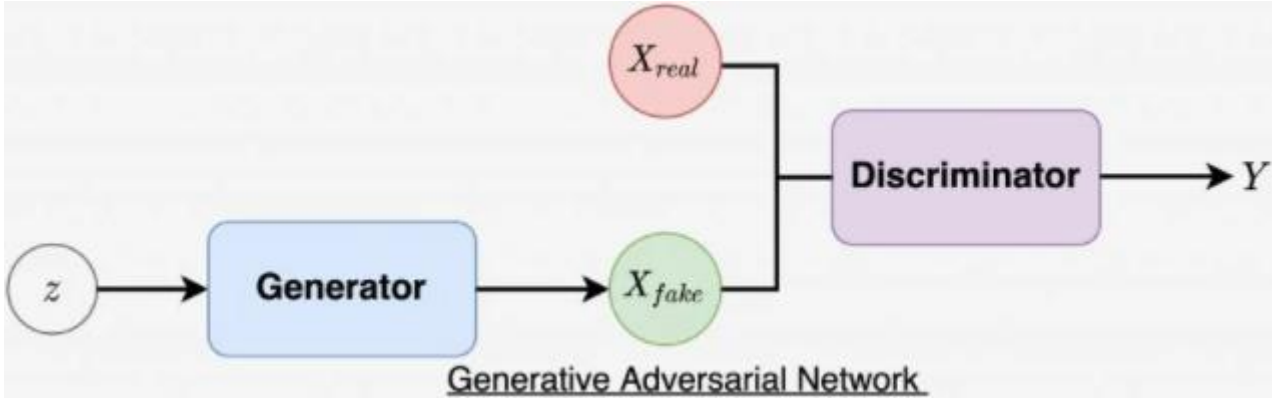
- **Customized Generator(U-Net+FPN) & Discriminator (modified PatchGAN)**
- **WGAN-GP**
- **Multiple Losses: adv loss + l1/l2 loss + perceptual loss + gp loss**
- **Can be used as your starting point**

# II. GAN: Applications

## G. Conditional GAN [2014, Mehdi Mirza]

**One drawback of original GAN**

- **No control on modes of the data being generated**
  input is random noise vector

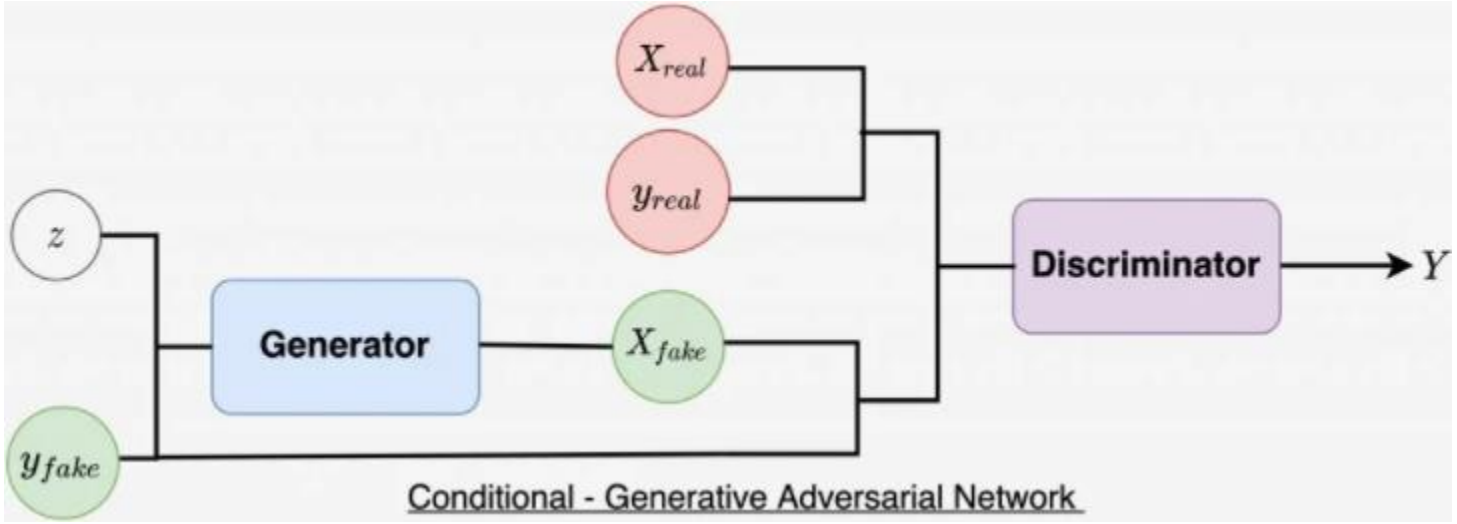


# II. GAN: Applications

## G. Conditional GAN

**One drawback of original GAN**

- **No control on modes of the data being generated**
  input is random noise vector
- **Condition the model on additional information can direct data generation**
  additional input info can control our output

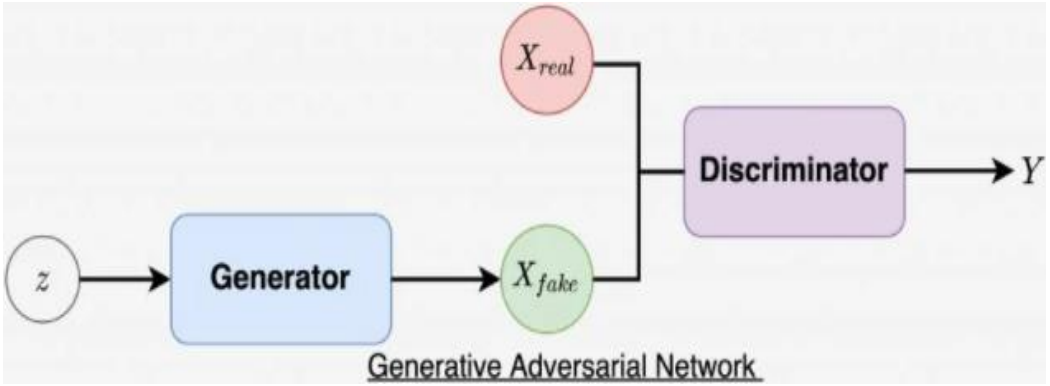


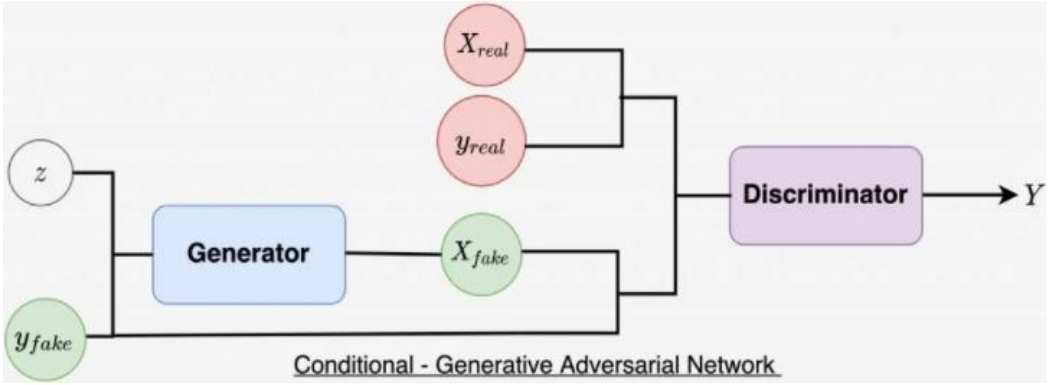
# II. GAN: Applications

## G. Conditional GAN

### Some tips of cGAN

- **G & D could be any form: NN or CNN**
- **Input of G could be any form (z)**
  here is random noise vector
  but could be other forms like images, audios etc.
- **Conditions of G & D could be any form (y)**
  **$y$ in original work:**
  For training:
  $y_{real}$: class label [one-hot vector]
  $y_{fake}$: class label [randomly generate]
  For referencing:
  $y$: target class label [one-hot vector]
  **$y$ in other works:** could be any form like images or so
  $y_{real}$ and $y_{real}$ could be same



# II. GAN: Applications

## G. Conditional GAN

### Implementations of cGAN

- **No differences from original training pipeline**

- **Use class labels as additional info**
  Use labels as conditions for real samples
  Use created fake labels [gen_labels] as conditions for generated samples

- **Details of G & D**
  Next page

- **Link**

```
# ----------------
#  Train Generator
# ----------------

optimizer_G.zero_grad()

# Sample noise and labels as generator input
z = Variable(FloatTensor(np.random.normal(0, 1, (batch_size, opt.latent_dim))))
gen_labels = Variable(LongTensor(np.random.randint(0, opt.n_classes, batch_size)))

# Generate a batch of images
gen_imgs = generator(z, gen_labels)

                    BCELoss
# Loss measures generator's ability to fool the discriminator
validity = discriminator(gen_imgs, gen_labels)
g_loss = adversarial_loss(validity, valid)

g_loss.backward()
optimizer_G.step()

# ---------------------
#  Train Discriminator
# ---------------------

optimizer_D.zero_grad()

# Loss for real images
validity_real = discriminator(real_imgs, labels)
d_real_loss = adversarial_loss(validity_real, valid)

# Loss for fake images
validity_fake = discriminator(gen_imgs.detach(), gen_labels)
d_fake_loss = adversarial_loss(validity_fake, fake)

# Total discriminator loss
d_loss = (d_real_loss + d_fake_loss) / 2

d_loss.backward()
optimizer_D.step()
```

# II. GAN: Application

## G. Conditional GAN

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()

        self.label_embedding = nn.Embedding(opt.n_classes, opt.n_classes)

        self.model = nn.Sequential(
            nn.Linear(opt.n_classes + int(np.prod(img_shape)), 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 512),
            nn.Dropout(0.4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 512),
            nn.Dropout(0.4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 1),
        )

    def forward(self, img, labels):
        # Concatenate label embedding and image to produce input
        d_in = torch.cat((img.view(img.size(0), -1), self.label_embedding(labels)), -1)
        validity = self.model(d_in)
        return validity
```

- nn.Embedding: vectorizing label

```
>>> class_num, batch_size = 4, 2
>>> gen_labels = Variable(LongTensor(np.random.randint(0, class_num, batch_size)))
>>> gen_labels
tensor([3, 0])
>>> embedding = nn.Embedding(class_num, class_num)
>>> embedding(gen_labels)
tensor([[ 0.5038,  0.2772,  0.5829, -0.8652],
        [ 1.9970,  1.7157,  0.6585, -0.7063]], grad_fn=<EmbeddingBackward>)
```

- nn.Embedding: transfer 1-dim info to target-dim vectors
- nn.Embedding: not common in CV
- .scatter_: more common in CV
  translate label scalar to one-hot vector

```
>>> gen_labels
tensor([3, 0])
>>> gen_y = torch.zeros(batch_size, class_num)
>>> gen_y = Variable(gen_y.scatter_(1, gen_labels.view(batch_size, 1), 1))
>>> gen_y
tensor([[0., 0., 0., 1.],
        [1., 0., 0., 0.]])
```

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        self.label_emb = nn.Embedding(opt.n_classes, opt.n_classes)

        def block(in_feat, out_feat, normalize=True):
            layers = [nn.Linear(in_feat, out_feat)]
            if normalize:
                layers.append(nn.BatchNorm1d(out_feat, 0.8))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *block(opt.latent_dim + opt.n_classes, 128, normalize=False),
            *block(128, 256),
            *block(256, 512),
            *block(512, 1024),
            nn.Linear(1024, int(np.prod(img_shape))),
            nn.Tanh())

    def forward(self, noise, labels):
        # Concatenate label embedding and image to produce input
        gen_input = torch.cat((self.label_emb(labels), noise), -1)
        img = self.model(gen_input)
        img = img.view(img.size(0), *img_shape)
        return img
```

# II. GAN: Applications

## H. AC-GAN [2016 Augustus Odena]

### AC: auxiliary classifier

- **Official code: Link**

```
############################
# (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
###########################
# train with real
netD.zero_grad()
dis_output, aux_output = netD(input)

dis_errD_real = dis_criterion(dis_output, dis_label)
aux_errD_real = aux_criterion(aux_output, aux_label)
errD_real = dis_errD_real + aux_errD_real
errD_real.backward()

# train with fake
fake = netG(noise)
dis_label.data.fill_(fake_label)
dis_output, aux_output = netD(fake.detach())
dis_errD_fake = dis_criterion(dis_output, dis_label)
aux_errD_fake = aux_criterion(aux_output, aux_label)
errD_fake = dis_errD_fake + aux_errD_fake
errD_fake.backward()

optimizerD.step()
```

1

gt: class label

[1, 0, 0, | 0.3472, -0.209, ...]

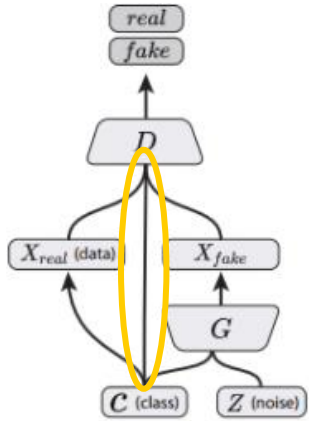
concat(aux_label, random noise)

Not necessary to be unconditional for D

0

Randomly generated class label

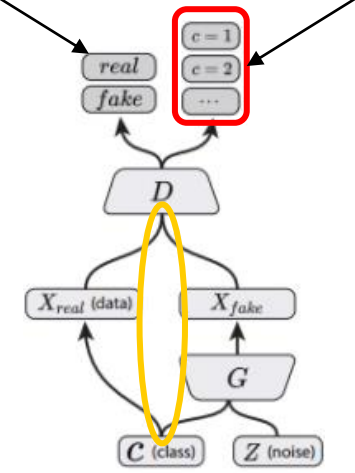
```
############################
# (2) Update G network: maximize log(D(G(z)))
###########################
netG.zero_grad()
dis_output, aux_output = netD(fake)
dis_errG = dis_criterion(dis_output, dis_label)
aux_errG = aux_criterion(aux_output, aux_label)
errG = dis_errG + aux_errG
errG.backward()
optimizerG.step()

class _netD(nn.Module):
    def __init__(self, ngpu, num_classes=10):
        # discriminator fc
        self.fc_dis = nn.Linear(13*13*512, 1)
        # aux-classifier fc
        self.fc_aux = nn.Linear(13*13*512, num_classes)
        # softmax and sigmoid
        self.softmax = nn.Softmax()
        self.sigmoid = nn.Sigmoid()
    def forward(self, input):
        fc_dis = self.fc_dis(flat6)
        fc_aux = self.fc_aux(flat6)
        classes = self.softmax(fc_aux)
        realfake = self.sigmoid(fc_dis).view(-1, 1).squeeze(1)
        return realfake, classes
```




Conditional GAN
(Mirza & Osindero, 2014)

$L_S = E[\log P(S = real \mid X_{real})] + E[\log P(S = fake \mid X_{fake})]$

$L_C = E[\log P(C = c \mid X_{real})] + E[\log P(C = c \mid X_{fake})]$



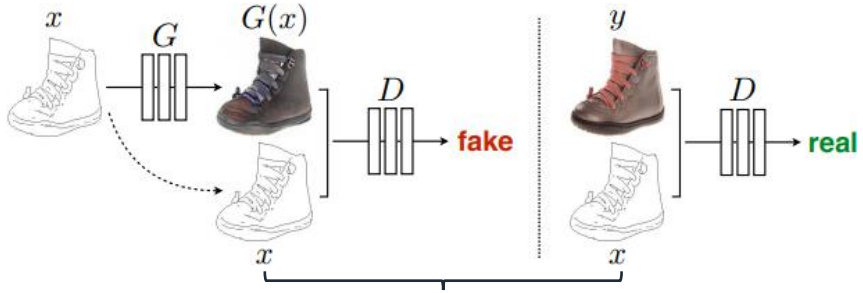

AC-GAN
(Present Work)

# II. GAN: Applications

## I. Pix2pix [2016-2018 Phillip Isola]

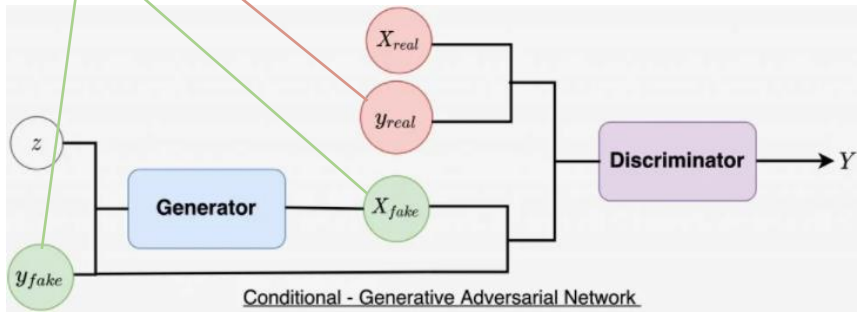
### Improvements: image condition + PatchGAN

- Image condition: original code, simplified version




x: image used as condition
they are same image

$y_{fake}$ & $y_{real}$: randomly generated label vector used as condition
they are different vectors




```
def forward(self, img_A, img_B):
    # Concatenate image and condition image by channels to produce input
    img_input = torch.cat((img_A, img_B), 1)
    return self.model(img_input)
```

```
# ------------------
#  Train Generators
# ------------------

optimizer_G.zero_grad()

# GAN loss
fake_B = generator(real_A)
pred_fake = discriminator(fake_B, real_A)
loss_GAN = criterion_GAN(pred_fake, valid)
# Pixel-wise loss
loss_pixel = criterion_pixelwise(fake_B, real_B)

# Total loss
loss_G = loss_GAN + lambda_pixel * loss_pixel

loss_G.backward()

optimizer_G.step()
```

real
fake

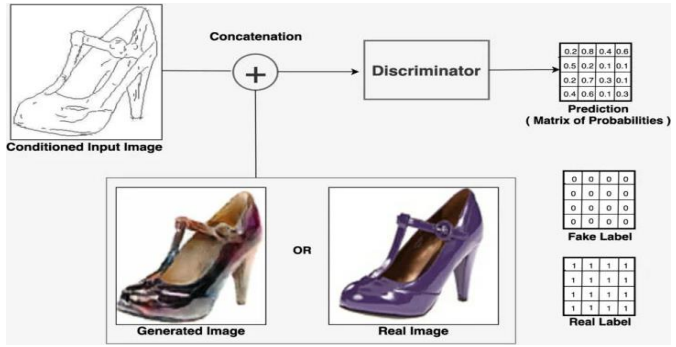
```
# ----------------------
#  Train Discriminator
# ----------------------

optimizer_D.zero_grad()

# Real loss
pred_real = discriminator(real_B, real_A)
loss_real = criterion_GAN(pred_real, valid)

# Fake loss
pred_fake = discriminator(fake_B.detach(), real_A)
loss_fake = criterion_GAN(pred_fake, fake)

# Total loss
loss_D = 0.5 * (loss_real + loss_fake)

loss_D.backward()
optimizer_D.step()
```

# II. GAN: Applications

## I. Pix2pix

### Improvements: image condition + PatchGAN

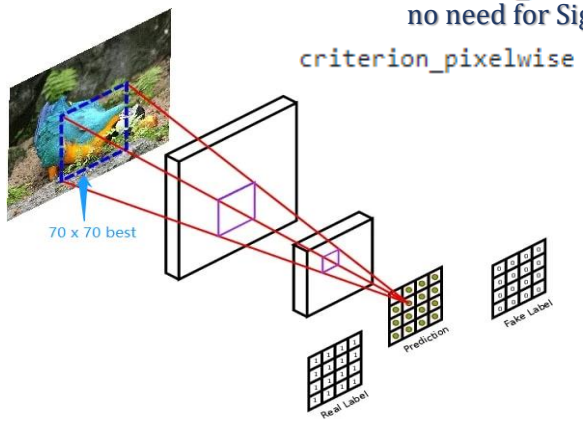
- PatchGAN: better than original discriminator




```
criterion_GAN = torch.nn.MSELoss()
```

no need for Sigmoid/Softmax in D

```
criterion_pixelwise = torch.nn.L1Loss()
```




```
def forward(self, img_A, img_B):
    # Concatenate image and condition image by channels to produce input
    img_input = torch.cat((img_A, img_B), 1)
    return self.model(img_input)
```

```
# ------------------
#  Train Generators
# ------------------

optimizer_G.zero_grad()

# GAN loss
fake_B = generator(real_A)
pred_fake = discriminator(fake_B, real_A)
loss_GAN = criterion_GAN(pred_fake, valid)
# Pixel-wise loss
loss_pixel = criterion_pixelwise(fake_B, real_B)

# Total loss
loss_G = loss_GAN + lambda_pixel * loss_pixel

loss_G.backward()

optimizer_G.step()
```

```
# ----------------------
#  Train Discriminator
# ----------------------

optimizer_D.zero_grad()

# Real loss
pred_real = discriminator(real_B, real_A)
loss_real = criterion_GAN(pred_real, valid)

# Fake loss
pred_fake = discriminator(fake_B.detach(), real_A)
loss_fake = criterion_GAN(pred_fake, fake)

# Total loss
loss_D = 0.5 * (loss_real + loss_fake)

loss_D.backward()
optimizer_D.step()
```

# II. GAN: Applications

## I. Pix2pix

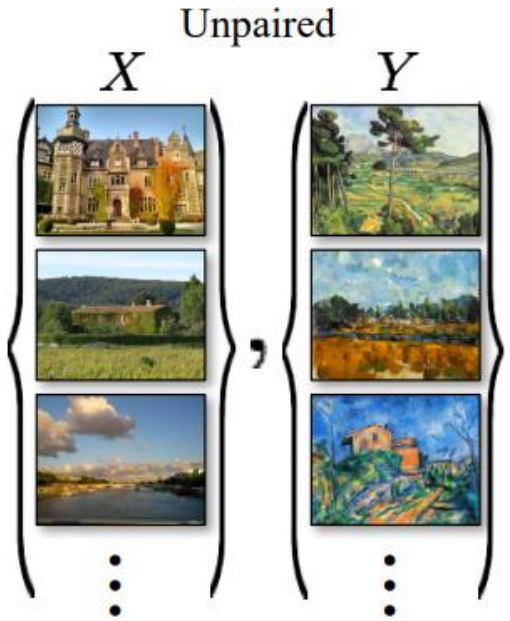
**Practical / Real code**

- **PatchGAN: used in WGAN(-gp)**
- **UNet-structure for G**
- **Other tricks for training GAN**

# II. GAN: Applications

## J. Cycle-GAN [2017-2020 Jun-Yan Zhu]

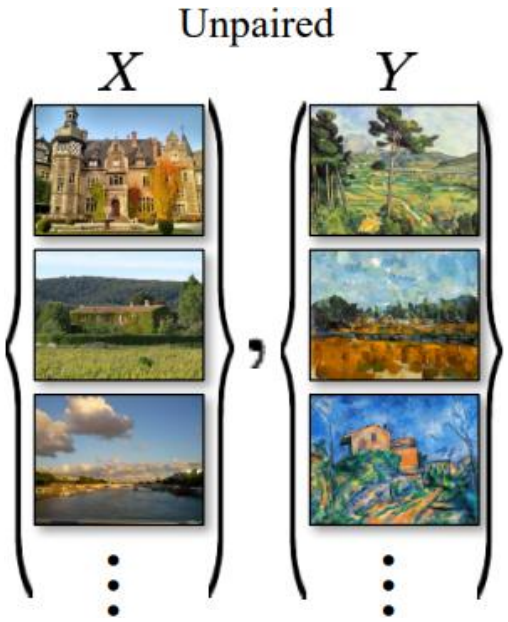
**Huge progress: unpair data for training**

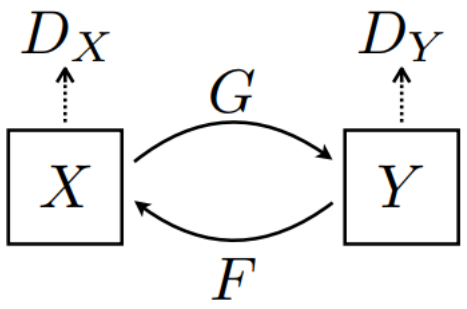



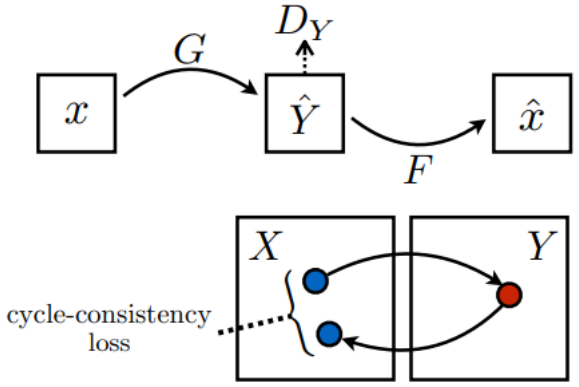
# II. GAN: Applications

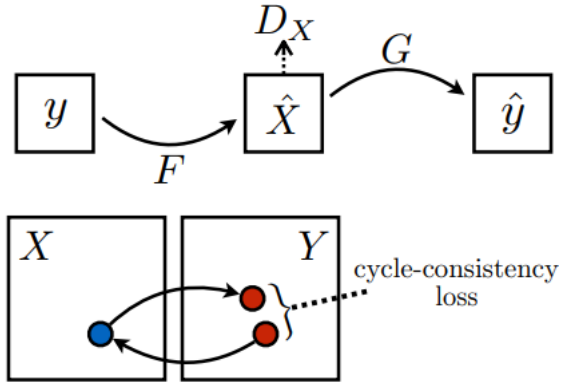
## J. Cycle-GAN [2017-2020 Jun-Yan Zhu]

**Huge progress: unpair data for training**



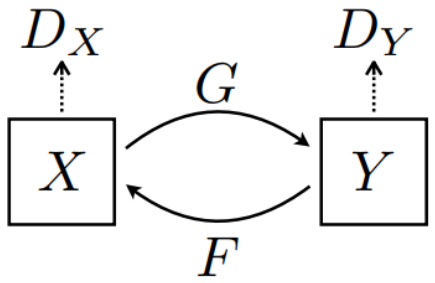



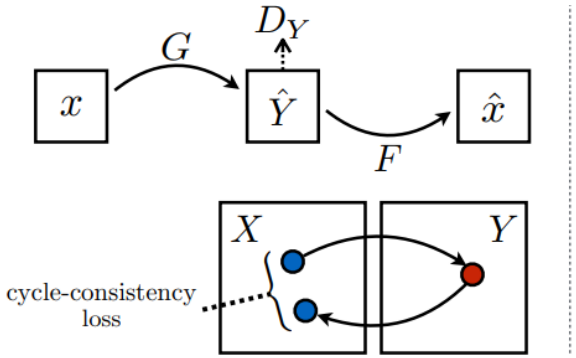

- **2 Generators forms a cycle**
  not only we train one G & D
  we train 2 pairs of G & D [G, Dy: Domain 1 -> 2 / F, Dx: Domain 2 -> 1]

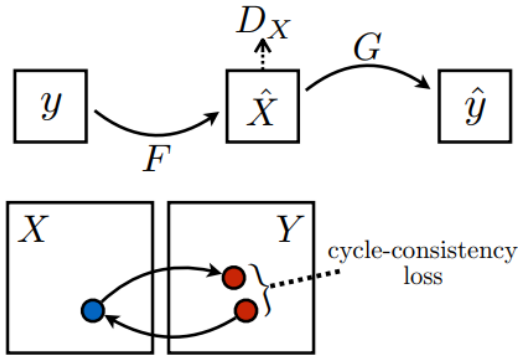
# II. GAN: Applications

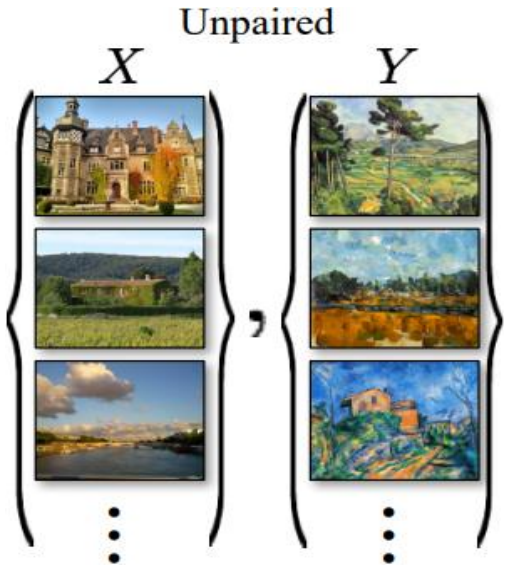
## J. Cycle-GAN

### Cycle Consistency




- **Equations**

  **For** $G: X \rightarrow Y$

  $$\min_G \max_{D_Y} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y)$$

  **For** $F: Y \rightarrow X$

  $$\min_F \max_{D_X} \mathcal{L}_{\text{GAN}}(F, D_X, Y, X)$$

  $$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))]$$

- **Cycle Consistency**

  $$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\| F(G(x)) - x \|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\| G(F(y)) - y \|_1]$$

- **Full objective**

  $$\mathcal{L}(G, F, D_x, D_y) = \mathcal{L}_{\text{GAN}}(G, D_y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_x, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F) + [\mathcal{L}_{\text{identity}}(G, F)]$$

# II. GAN: Applications

## J. Cycle-GAN

**Code: original version (start from line 180), simplified version (start from line 155)**

```
# ------------------
#  Train Generators
# ------------------

G_AB.train()
G_BA.train()

optimizer_G.zero_grad()

# GAN loss
fake_B = G_AB(real_A)
loss_GAN_AB = criterion_GAN(D_B(fake_B), valid)
fake_A = G_BA(real_B)
loss_GAN_BA = criterion_GAN(D_A(fake_A), valid)

loss_GAN = (loss_GAN_AB + loss_GAN_BA) / 2

# Cycle loss
recov_A = G_BA(fake_B)
loss_cycle_A = criterion_cycle(recov_A, real_A)
recov_B = G_AB(fake_A)
loss_cycle_B = criterion_cycle(recov_B, real_B)

loss_cycle = (loss_cycle_A + loss_cycle_B) / 2

# Total loss
loss_G = loss_GAN + opt.lambda_cyc * loss_cycle

loss_G.backward()
optimizer_G.step()
```

nn.MSELoss()

PatchGAN

nn.MSELoss()

nn.L1Loss()

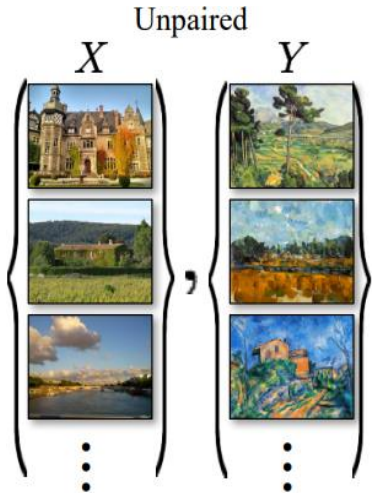
nn.L1Loss()

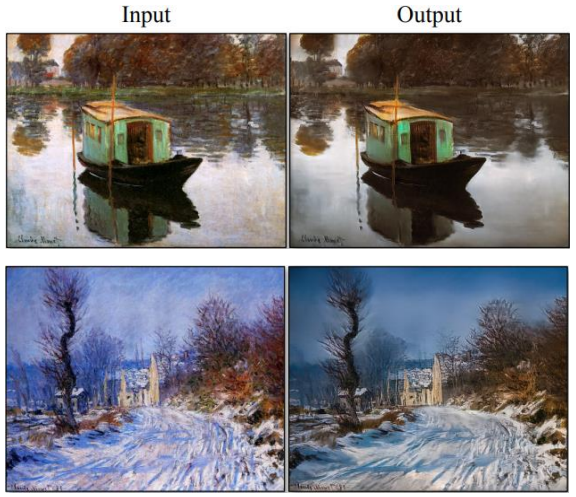
```
optimizer_D_A.zero_grad()

# Real loss
loss_real = criterion_GAN(D_A(real_A), valid)
# Fake loss (on batch of previously generated samples)
fake_A_ = fake_A_buffer.push_and_pop(fake_A)
loss_fake = criterion_GAN(D_A(fake_A_.detach()), fake)
# Total loss
loss_D_A = (loss_real + loss_fake) / 2

# ----------------------
#  Train Discriminator A
# ----------------------

loss_D_A.backward()
optimizer_D_A.step()

optimizer_D_B.zero_grad()

# Real loss
loss_real = criterion_GAN(D_B(real_B), valid)
# Fake loss (on batch of previously generated samples)
fake_B_ = fake_B_buffer.push_and_pop(fake_B)
loss_fake = criterion_GAN(D_B(fake_B_.detach()), fake)
# Total loss
loss_D_B = (loss_real + loss_fake) / 2

# ----------------------
#  Train Discriminator B
# ----------------------

loss_D_B.backward()
optimizer_D_B.step()
```

nn.MSELoss()

nn.MSELoss()

nn.L1Loss()




- **Not necessary: 1 optimizer for G & 2 optimizer for D**
  also work: 2optimizers for both
  1 optimizer for both

- **Usage of buffer, size=50**
  extract 1 from last 50 images and push into D
  reason: reduce model oscillation
  inspiration: 2017, Shrivastava, Apple, Link

- **Combine with WGAN**
  discussion for WGAN+CycleGAN, Link

# II. GAN: Applications

## J. Cycle-GAN

### Identity Loss

- **Equations**

  $\mathcal{L}_{\text{identity}}(G, F) =$
  $\mathbb{E}_{y \sim p_{\text{data}}(y)}[\| G(y) - y \|_1] +$
  $\mathbb{E}_{x \sim p_{\text{data}}(x)}[\| F(x) - x \|_1]$

- **Reason**

  if image$(y)$ has already been in $Y$, then $G(y)$ should $G(y) = y$

- **Inspiration**

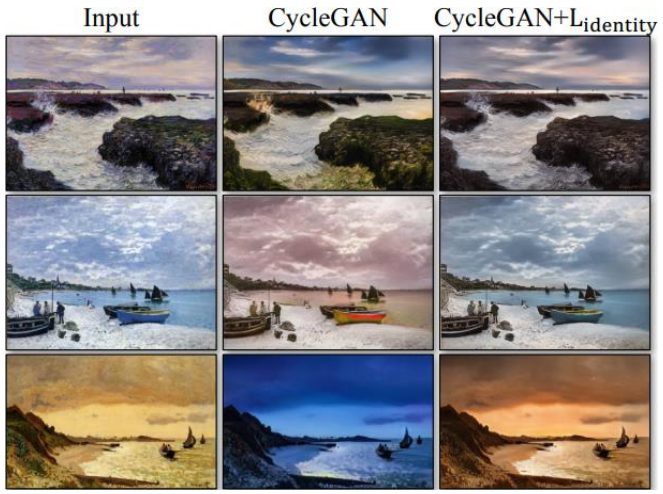
  Link, ICLR 2017, Taigman

  $$R_{\text{CONST}} = \mathbb{E}_{x \sim \mathcal{D}_S}\, d(f(x), f(G(x)))$$

- **Without this**

  color may shift

**Problem**


```
# ------------------
#  Train Generators
# ------------------

G_AB.train()
G_BA.train()

optimizer_G.zero_grad()

# Identity loss
loss_id_A = criterion_identity(G_BA(real_A), real_A)
loss_id_B = criterion_identity(G_AB(real_B), real_B)

loss_identity = (loss_id_A + loss_id_B) / 2

# GAN loss
fake_B = G_AB(real_A)
loss_GAN_AB = criterion_GAN(D_B(fake_B), valid)
fake_A = G_BA(real_B)
loss_GAN_BA = criterion_GAN(D_A(fake_A), valid)

loss_GAN = (loss_GAN_AB + loss_GAN_BA) / 2

# Cycle loss
recov_A = G_BA(fake_B)
loss_cycle_A = criterion_cycle(recov_A, real_A)
recov_B = G_AB(fake_A)
loss_cycle_B = criterion_cycle(recov_B, real_B)

loss_cycle = (loss_cycle_A + loss_cycle_B) / 2

# Total loss
loss_G = loss_GAN + opt.lambda_cyc * loss_cycle + opt.lambda_id * loss_identity

loss_G.backward()
optimizer_G.step()
```

nn.MSELoss()

PatchGAN

nn.MSELoss()

nn.L1Loss()

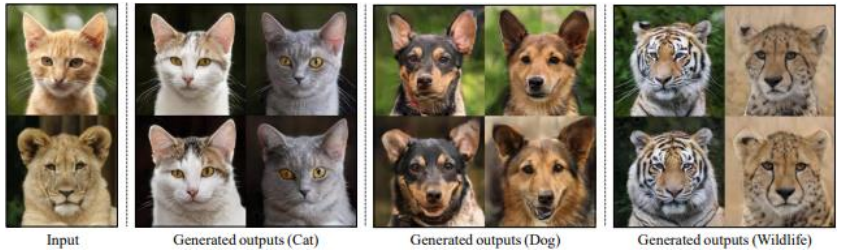
nn.L1Loss()

# II. GAN: Applications

## K. StarGAN [2018, Choi]

**Huge progress:**

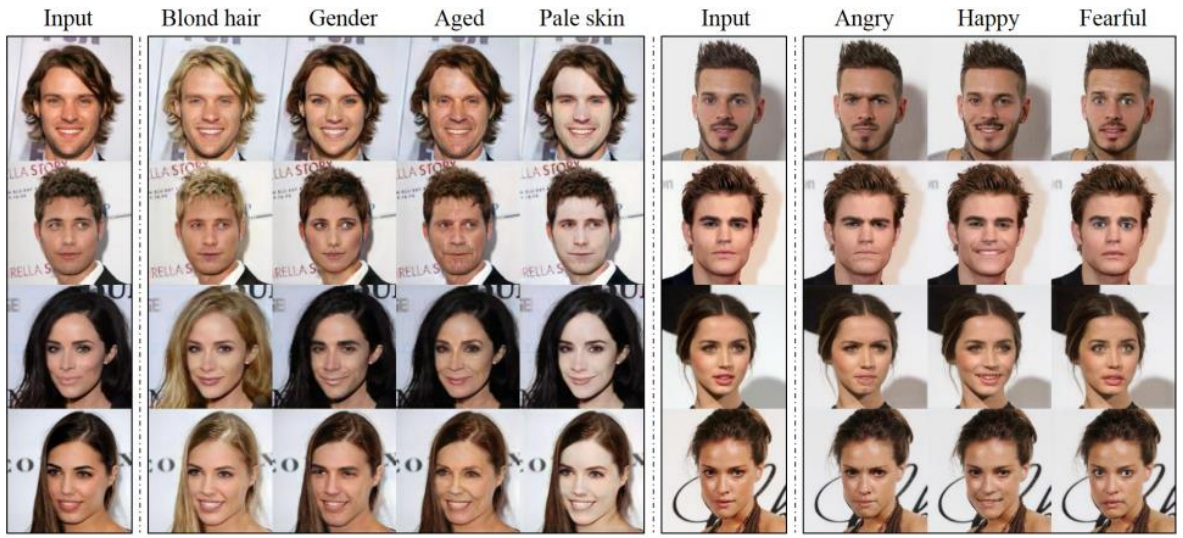
**multi-domain translation**
**partial labeled GT** } **for unpair data training**




- **Previous difficulties**
  translation can only happen between 2 domains
  labels are complete between 2 domains

- **Solution**
  class label condition + auxiliary classifier
  + cycle consistency
  mask vector

- **Sources**
  v1: paper, original code, simplified code
  v2 [2020, Choi]: paper, original code



# II. GAN: Applications

## K. StarGAN

**Huge progress:**

**multi-domain translation**

**partial labeled GT**

- **Solution**

  class label condition + auxiliary classifier + cycle consistency

```python
class Discriminator(nn.Module):
    """Discriminator network with PatchGAN."""
    def __init__(self, image_size=128, conv_dim=64, c_dim=5, repeat_num=6):
        super(Discriminator, self).__init__()
        layers = []
        layers.append(nn.Conv2d(3, conv_dim, kernel_size=4, stride=2, padding=1))
        layers.append(nn.LeakyReLU(0.01))

        curr_dim = conv_dim
        for i in range(1, repeat_num):
            layers.append(nn.Conv2d(curr_dim, curr_dim*2, kernel_size=4, stride=2, padding=1))
            layers.append(nn.LeakyReLU(0.01))
            curr_dim = curr_dim * 2

        kernel_size = int(image_size / np.power(2, repeat_num))
        self.main = nn.Sequential(*layers)
        self.conv1 = nn.Conv2d(curr_dim, 1, kernel_size=3, stride=1, padding=1, bias=False)
        self.conv2 = nn.Conv2d(curr_dim, c_dim, kernel_size=kernel_size, bias=False)

    def forward(self, x):
        h = self.main(x)
        out_src = self.conv1(h)
        out_cls = self.conv2(h)
        return out_src, out_cls.view(out_cls.size(0), out_cls.size(1))
```

combined with WGAN-GP

convert tensor shape to (n, c, 1, 1)

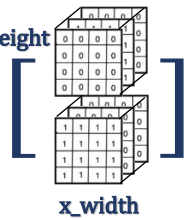
auxiliary classifier

convert tensor shape to (n, c)

condition on label
(label: domain id)

assure output is limited
[not necessary actually]

c : one-hot matrix
e.g.:
[[0, 1, 0]
[1, 0, 0]]
batch_size = 2,
class_num = 3




```python
class Generator(nn.Module):
    """Generator network."""
    def __init__(self, conv_dim=64, c_dim=5, repeat_num=6):
        super(Generator, self).__init__()

        layers = []
        layers.append(nn.Conv2d(3+c_dim, conv_dim, kernel_size=7, stride=1, padding=3, bias=False))
        layers.append(nn.InstanceNorm2d(conv_dim, affine=True, track_running_stats=True))
        layers.append(nn.ReLU(inplace=True))

        # Down-sampling layers.
        curr_dim = conv_dim
        for i in range(2):
            layers.append(nn.Conv2d(curr_dim, curr_dim*2, kernel_size=4, stride=2, padding=1, bias=False))
            layers.append(nn.InstanceNorm2d(curr_dim*2, affine=True, track_running_stats=True))
            layers.append(nn.ReLU(inplace=True))
            curr_dim = curr_dim * 2

        # Bottleneck layers.
        for i in range(repeat_num):
            layers.append(ResidualBlock(dim_in=curr_dim, dim_out=curr_dim))

        # Up-sampling layers.
        for i in range(2):
            layers.append(nn.ConvTranspose2d(curr_dim, curr_dim//2, kernel_size=4, stride=2, padding=1, bias=False))
            layers.append(nn.InstanceNorm2d(curr_dim//2, affine=True, track_running_stats=True))
            layers.append(nn.ReLU(inplace=True))
            curr_dim = curr_dim // 2

        layers.append(nn.Conv2d(curr_dim, 3, kernel_size=7, stride=1, padding=3, bias=False))
        layers.append(nn.Tanh())
        self.main = nn.Sequential(*layers)

    def forward(self, x, c):
        # Replicate spatially and concatenate domain information.
        # Note that this type of label conditioning does not work at all if we use reflection padding in Conv2d.
        # This is because instance normalization ignores the shifting (or bias) effect.
        c = c.view(c.size(0), c.size(1), 1, 1)
        c = c.repeat(1, 1, x.size(2), x.size(3))
        x = torch.cat([x, c], dim=1)
        return self.main(x)
```

# II. GAN: Applications

## K. StarGAN

**Huge progress:**

**multi-domain translation**

**partial labeled GT**

- **Solution**

  class label condition + auxiliary classifier + cycle consistency

```
class Discriminator(nn.Module):
    """Discriminator network with PatchGAN."""  combined with WGAN-GP
    def __init__(self, image_size=128, conv_dim=64, c_dim=5, repeat_num=6):
        super(Discriminator, self).__init__()
        layers = []
        layers.append(nn.Conv2d(3, conv_dim, kernel_size=4, stride=2, padding=1))
        layers.append(nn.LeakyReLU(0.01))

        curr_dim = conv_dim
        for i in range(1, repeat_num):
            layers.append(nn.Conv2d(curr_dim, curr_dim*2, kernel_size=4, stride=2, padding=1))
            layers.append(nn.LeakyReLU(0.01))
            curr_dim = curr_dim * 2

        kernel_size = int(image_size / np.power(2, repeat_num))
        self.main = nn.Sequential(*layers)
        self.conv1 = nn.Conv2d(curr_dim, 1, kernel_size=3, stride=1, padding=1, bias=False)
        self.conv2 = nn.Conv2d(curr_dim, c_dim, kernel_size=kernel_size, bias=False)
        convert tensor shape to (n, c, 1, 1)
    def forward(self, x):
        h = self.main(x)
        out_src = self.conv1(h)
        out_cls = self.conv2(h)
        return out_src, out_cls.view(out_cls.size(0), out_cls.size(1))
```

auxiliary classifier

convert tensor shape to (n, c)

```
# ============================
# Train the discriminator
# ============================

# Compute loss with real images.
out_src, out_cls = self.D(x_real)
d_loss_real = - torch.mean(out_src)
d_loss_cls = self.classification_loss(out_cls, label_org, self.dataset)

# Compute loss with fake images.
x_fake = self.G(x_real, c_trg)
out_src, out_cls = self.D(x_fake.detach())
d_loss_fake = torch.mean(out_src)

# Compute loss for gradient penalty.
alpha = torch.rand(x_real.size(0), 1, 1, 1).to(self.device)
x_hat = (alpha * x_real.data + (1 - alpha) * x_fake.data).requires_grad_(True)
out_src, _ = self.D(x_hat)
d_loss_gp = self.gradient_penalty(out_src, x_hat)

# Backward and optimize.  adv loss of WGAN  classification loss (cross entropy)  adv loss of gradient penalty
d_loss = d_loss_real + d_loss_fake + self.lambda_cls * d_loss_cls + self.lambda_gp * d_loss_gp
self.reset_grad()
d_loss.backward()
self.d_optimizer.step()
```

condition on label

(label: domain id)

c : one-hot matrix

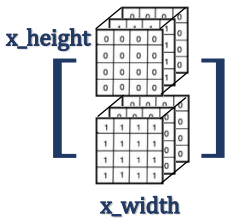
e.g.:

[[0, 1, 0]

[1, 0, 0]]

batch_size = 2,

class_num = 3




```
        layers.append(nn.Conv2d(curr_dim, 3, kernel_size=7, stride=1, padding=3, bias=False))
        layers.append(nn.Tanh())
        self.main = nn.Sequential(*layers)

    def forward(self, x, c):
        # Replicate spatially and concatenate domain information.
        # Note that this type of label conditioning does not work at all if we use reflection padding in Conv2d.
        # This is because instance normalization ignores the shifting (or bias) effect.
        c = c.view(c.size(0), c.size(1), 1, 1)
        c = c.repeat(1, 1, x.size(2), x.size(3))
        x = torch.cat([x, c], dim=1)
        return self.main(x)
```

# II. GAN: Applications

## K. StarGAN

**Huge progress:**

**multi-domain translation**

**partial labeled GT**

- **Solution**

  class label condition + auxiliary classifier + **cycle consistency** [Cycle-GAN]

```
# Original-to-target domain.
x_fake = self.G(x_real, c_trg)
out_src, out_cls = self.D(x_fake)
g_loss_fake = - torch.mean(out_src)
g_loss_cls = self.classification_loss(out_cls, label_trg, self.dataset)

==================
Train the generator
==================

# Target-to-original domain.
x_reconst = self.G(x_fake, c_org)
g_loss_rec = torch.mean(torch.abs(x_real - x_reconst))

# Backward and optimize.
g_loss = g_loss_fake + self.lambda_rec * g_loss_rec + self.lambda_cls * g_loss_cls
self.reset_grad()
g_loss.backward()
self.g_optimizer.step()
```

adv loss of WGAN — cycle loss — classification loss

[WGAN(-GP)]

```
class Discriminator(nn.Module):
    """Discriminator network with PatchGAN."""
    def __init__(self, image_size=128, conv_dim=64, c_dim=5, repeat_num=6):
        super(Discriminator, self).__init__()
        layers = []
        layers.append(nn.Conv2d(3, conv_dim, kernel_size=4, stride=2, padding=1))
        layers.append(nn.LeakyReLU(0.01))

        curr_dim = conv_dim
        for i in range(1, repeat_num):
            layers.append(nn.Conv2d(curr_dim, curr_dim*2, kernel_size=4, stride=2, padding=1))
            layers.append(nn.LeakyReLU(0.01))
            curr_dim = curr_dim * 2

        kernel_size = int(image_size / np.power(2, repeat_num))
        self.main = nn.Sequential(*layers)
        self.conv1 = nn.Conv2d(curr_dim, 1, kernel_size=3, stride=1, padding=1, bias=False)
        self.conv2 = nn.Conv2d(curr_dim, c_dim, kernel_size=kernel_size, bias=False)
    def forward(self, x):
        h = self.main(x)
        out_src = self.conv1(h)
        out_cls = self.conv2(h)
        return out_src, out_cls.view(out_cls.size(0), out_cls.size(1))
```

combined with WGAN-GP

convert tensor shape to (n, c, 1, 1)

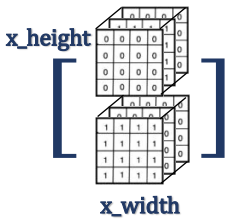
convert tensor shape to (n, c)

**PatchGAN** [pix2pix]

[cGAN] **condition on label** (label: domain id)

**auxiliary classifier** [AC-GAN]

c : one-hot matrix
e.g.:
[[0, 1, 0]
[1, 0, 0]]
batch_size = 2,
class_num = 3




```
        layers.append(nn.Conv2d(curr_dim, 3, kernel_size=7, stride=1, padding=3, bias=False))
        layers.append(nn.Tanh())
        self.main = nn.Sequential(*layers)

    def forward(self, x, c):
        # Replicate spatially and concatenate domain information.
        # Note that this type of label conditioning does not work at all if we use reflection padding in Conv2d.
        # This is because instance normalization ignores the shifting (or bias) effect.
        c = c.view(c.size(0), c.size(1), 1, 1)
        c = c.repeat(1, 1, x.size(2), x.size(3))
        x = torch.cat([x, c], dim=1)
        return self.main(x)
```

# II. GAN: Applications

## K. StarGAN

**Huge progress:**

**multi-domain translation**

**partial labeled GT**

- **Solution**

  mask vector [used to generate comprehensive label]

```
========================
1. Preprocess input data
========================

if dataset == 'CelebA':
    c_org = label_org.clone()
    c_trg = label_trg.clone()
    zero = torch.zeros(x_real.size(0), self.c2_dim)
    mask = self.label2onehot(torch.zeros(x_real.size(0)), 2)
    c_org = torch.cat([c_org, zero, mask], dim=1)
    c_trg = torch.cat([c_trg, zero, mask], dim=1)
elif dataset == 'RaFD':
    c_org = self.label2onehot(label_org, self.c2_dim)
    c_trg = self.label2onehot(label_trg, self.c2_dim)
    zero = torch.zeros(x_real.size(0), self.c_dim)
    mask = self.label2onehot(torch.ones(x_real.size(0)), 2)
    c_org = torch.cat([zero, c_org, mask], dim=1)
    c_trg = torch.cat([zero, c_trg, mask], dim=1)
```

```
[[0, 1, 0 | 0, 0, 0, 0 | 1, 0]
 [1, 0, 0 | 0, 0, 0, 0 | 1, 0]]

data A   data B   mask vector

[[0, 0, 0 | 0, 0, 0, 1 | 0, 1]
 [0, 0, 0 | 0, 1, 0, 0 | 0, 1]]
```

multi-dataset version

```
========================
2. Train the discriminator
========================

# Compute loss with real images.
out_src, out_cls = self.D(x_real)
out_cls = out_cls[:, :self.c_dim] if dataset == 'CelebA' else out_cls[:, self.c_dim:]
d_loss_real = - torch.mean(out_src)
d_loss_cls = self.classification_loss(out_cls, label_org, dataset)
```

original version

```
# Compute loss with real images.
out_src, out_cls = self.D(x_real)
d_loss_real = - torch.mean(out_src)
d_loss_cls = self.classification_loss(out_cls, label_org, self.dataset)
```

```
======================
3. Train the generator
======================

# Original-to-target domain.
x_fake = self.G(x_real, c_trg)
out_src, out_cls = self.D(x_fake)
out_cls = out_cls[:, :self.c_dim] if dataset == 'CelebA' else out_cls[:, self.c_dim:]
g_loss_fake = - torch.mean(out_src)
g_loss_cls = self.classification_loss(out_cls, label_trg, dataset)

# Original-to-target domain.
x_fake = self.G(x_real, c_trg)
out_src, out_cls = self.D(x_fake)
g_loss_fake = - torch.mean(out_src)
g_loss_cls = self.classification_loss(out_cls, label_trg, self.dataset)
```

# II. GAN: Applications

## K. StarGAN

**v2 version: StarGAN v2 [2020, Choi]**

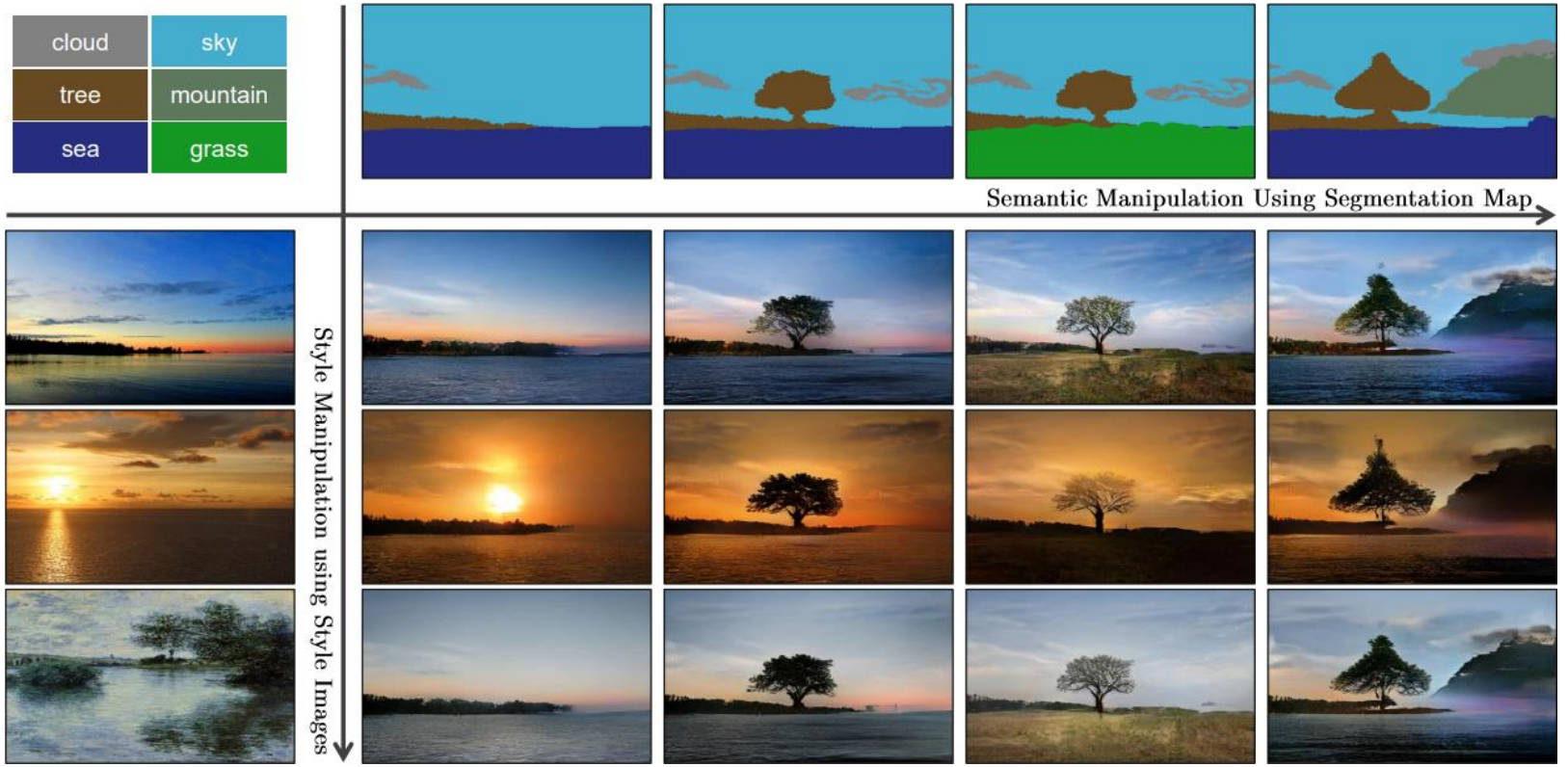
paper: StarGAN v2: Diverse Image Synthesis for Multiple Domains

code

# II. GAN: Applications

## L. SPADE [2019, Park]

**semantic image synthesis with spatially-adaptive normalization: SPatially Adaptive DEnormalization**



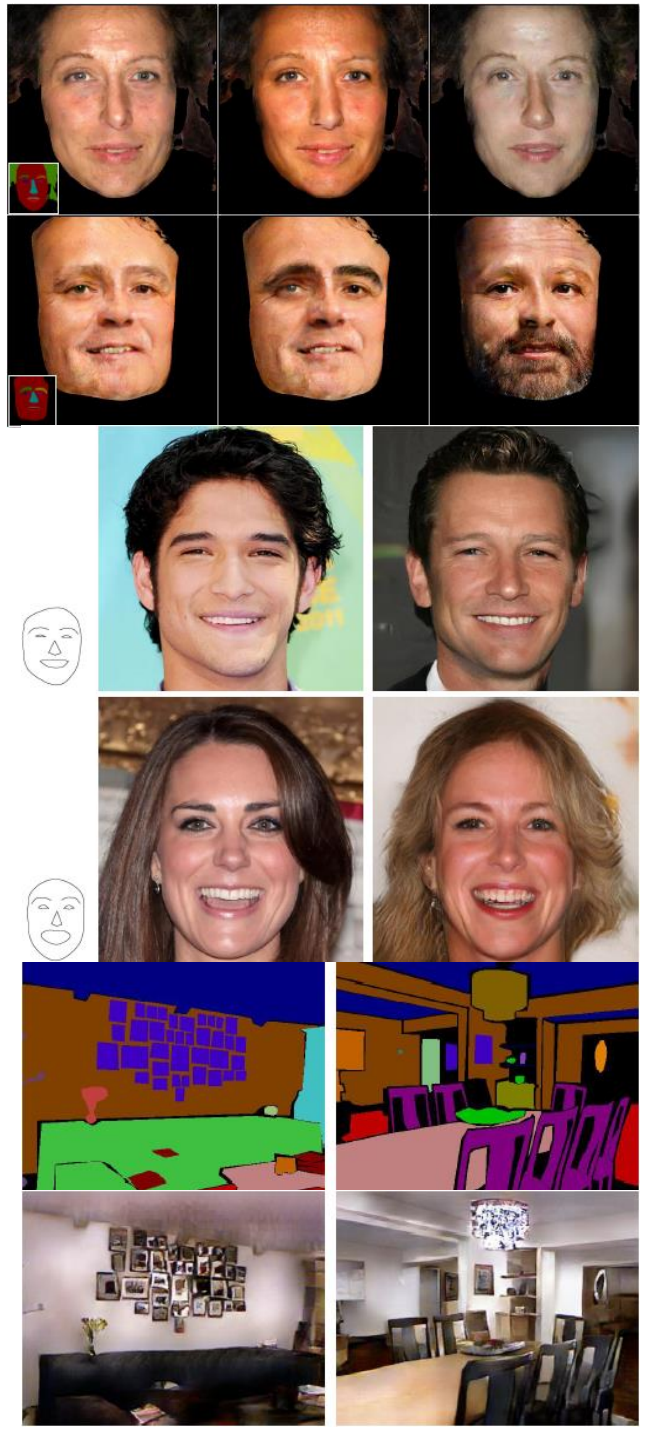
# II. GAN: Applications

## L. SPADE

**Before SPADE: pix2pixHD [2018, Wang, great links]**

- Generate higher resolution (2048 x 1024) than before

- Semantic manipulating generated images



# II. GAN: Applications

## L. SPADE

### Before SPADE: pix2pixHD [2018, Wang]

- **Generate higher resolution (2048 x 1024) than before**

  Multiscale G & D

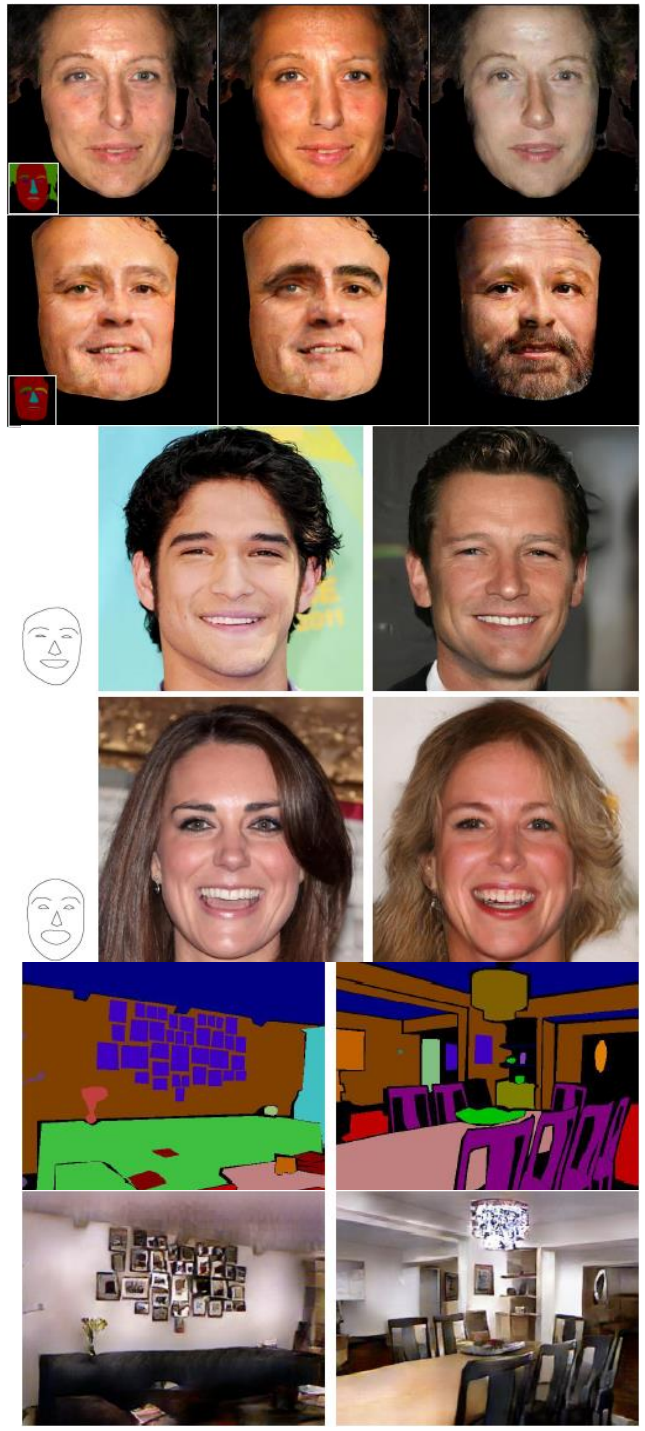
  Advanced objective functions: D —> Feature matching loss + perceptual loss

  Instance Map

- **Semantic manipulating generated images**

  Improved structure: feature encoder network + instance-wise pooling

  Improved input: label map + feature map



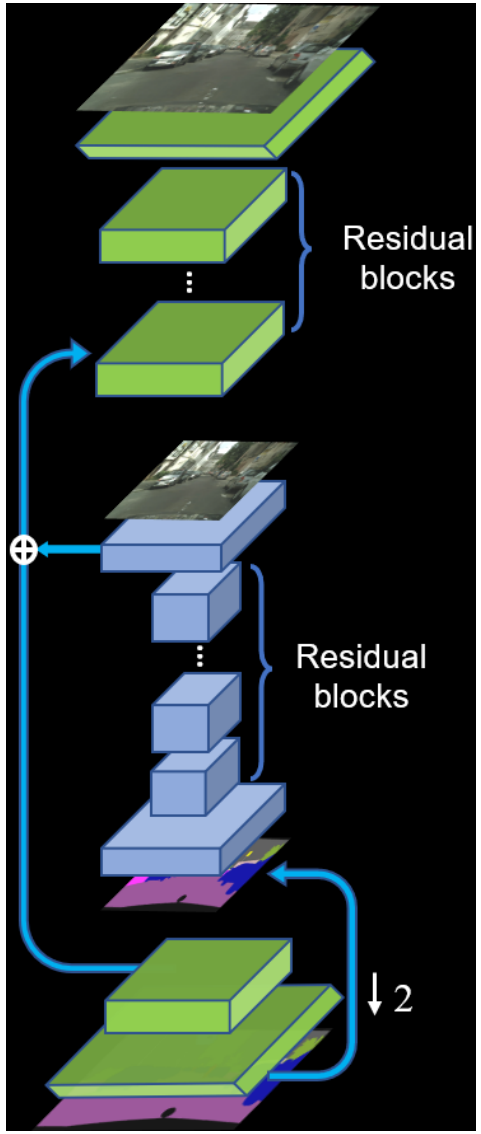
# II. GAN: Applications

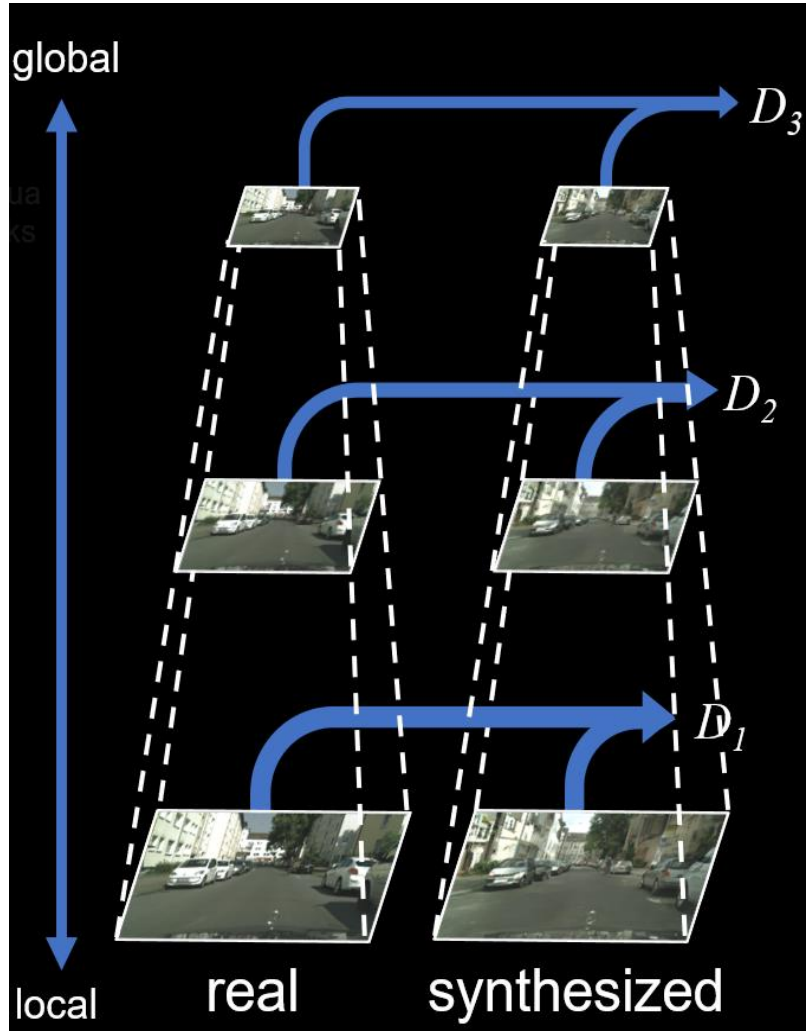
## L. SPADE

### Before SPADE: pix2pixHD [2018, Wang]

- **Generate higher resolution (2048 x 1024) than before**
  **Multiscale G & D**
  Advanced objective functions
  Instance Map

- **Semantic manipulating generated images**
  Improved structure
  Improved input




$$\min_G \max_{D_1,D_2,D_3} \sum_{k=1,2,3} \mathcal{L}_{\text{GAN}}(G, D_k)$$

# II. GAN: Applications

## L. SPADE

### Before SPADE: pix2pixHD [2018, Wang]

- **Generate higher resolution (2048 x 1024) than before**

  Multiscale G & D

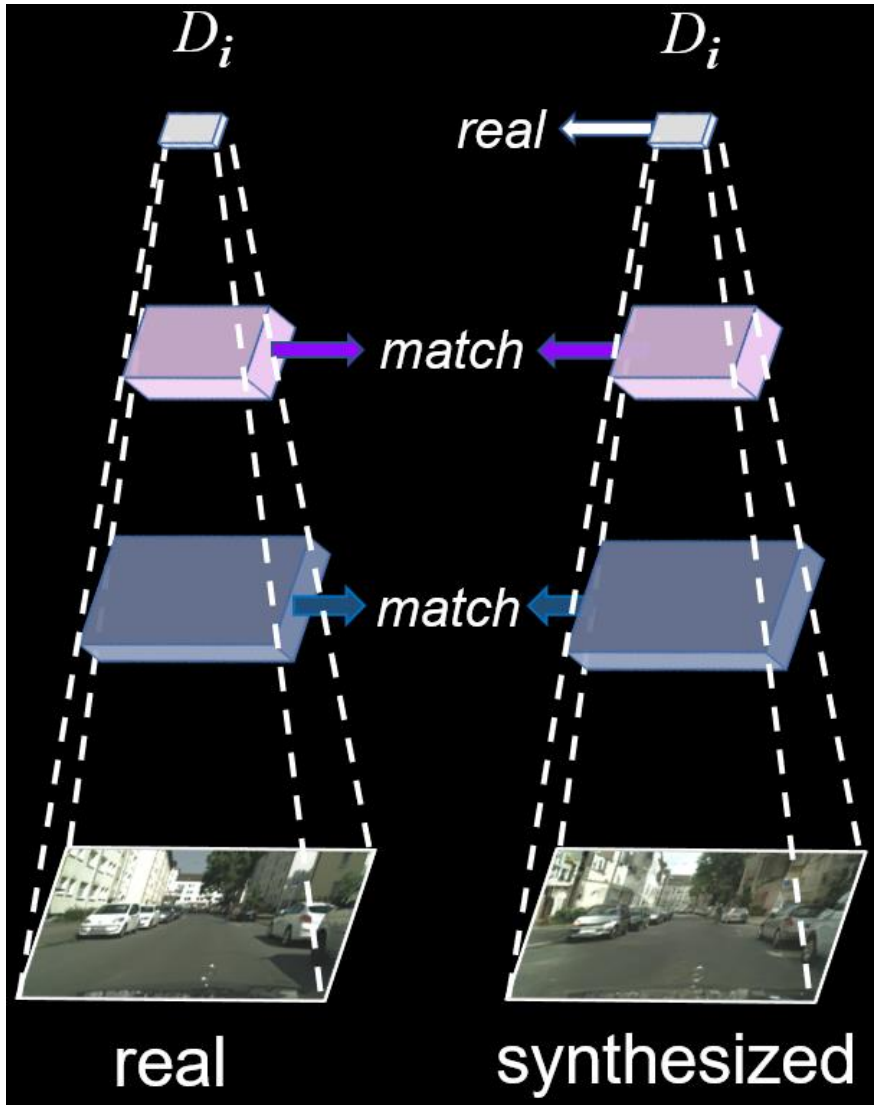
  **Advanced objective functions: FM Loss (D)** + Perceptual Loss

  Instance Map

- **Semantic manipulating generated images**

  Improved structure

  Improved input




$$\mathcal{L}_{\text{FM}}(G, D_k) = \mathbb{E}_{(\mathbf{s},\mathbf{x})} \sum_{i=1}^{T} \frac{1}{N_i} [||D_k^{(i)}(\mathbf{s}, \mathbf{x}) - D_k^{(i)}(\mathbf{s}, G(\mathbf{s}))||_1]$$

# II. GAN: Applications

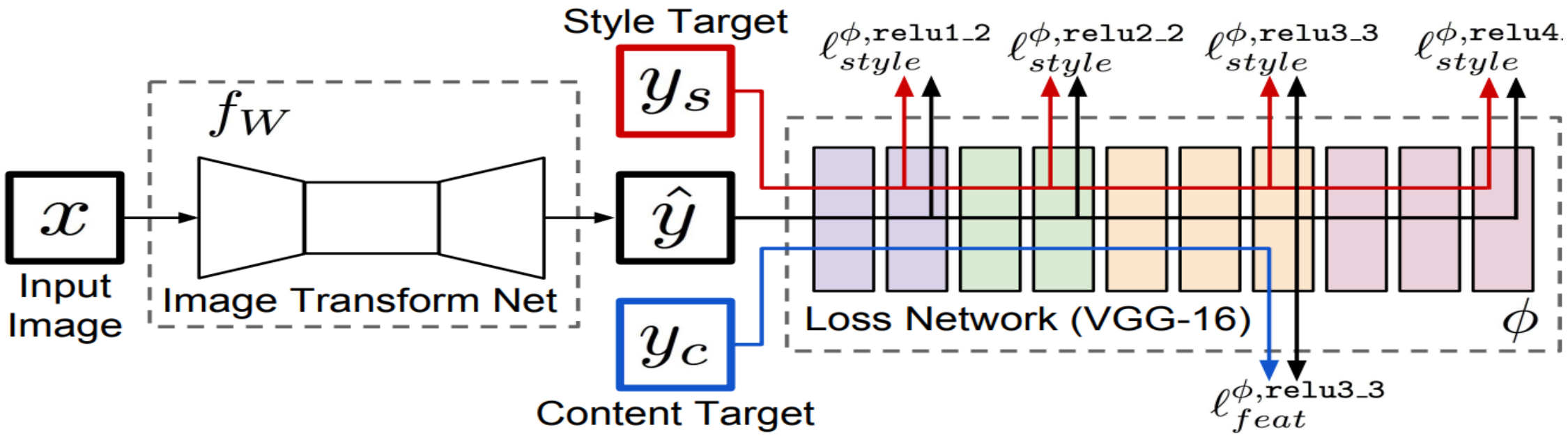
## L. SPADE

### Before SPADE: pix2pixHD [2018, Wang]

- **Generate higher resolution (2048 x 1024) than before**

  Multiscale G & D

  **Advanced objective functions**: FM Loss + Perceptual Loss (G(s))



# II. GAN: Applications

## L. SPADE

**Before SPADE: pix2pixHD [2018, Wang]**

- **Generate higher resolution (2048 x 1024) than before**

  Multiscale G & D

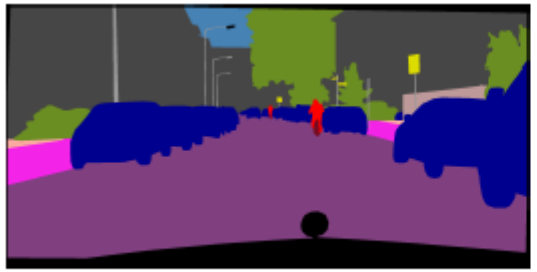
  Advanced objective functions

  Instance Map

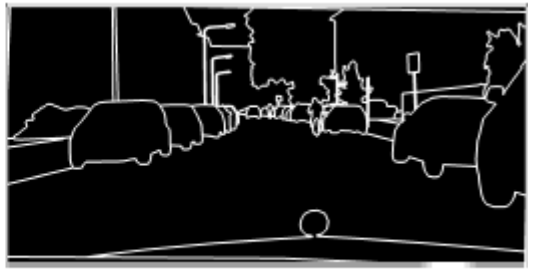
- **Semantic manipulating generated images**

  Improved structure

  Improved input



(a) Semantic labels



(b) Boundary map

(a) Using labels only

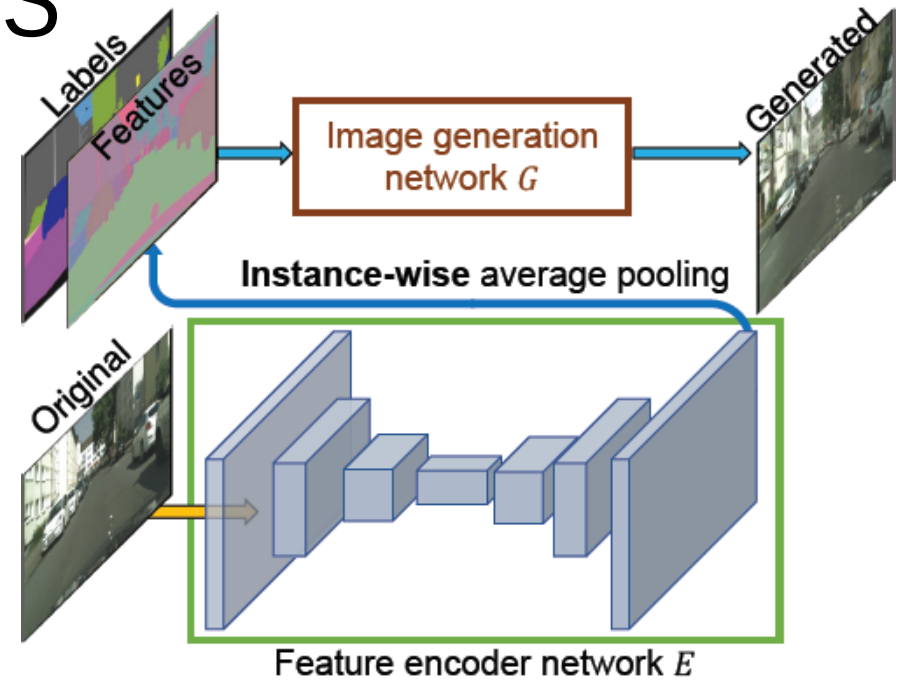
(b) Using label + instance map

# II. GAN: Applications

## L. SPADE

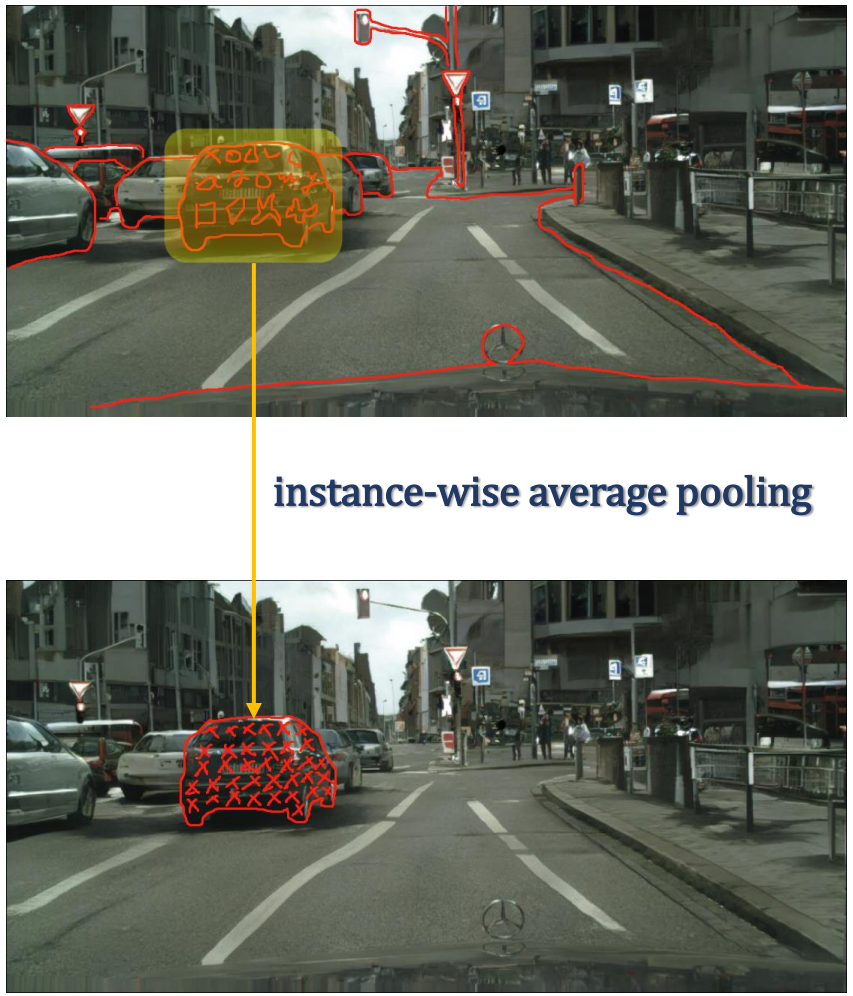
### Before SPADE: pix2pixHD [2018, Wang]

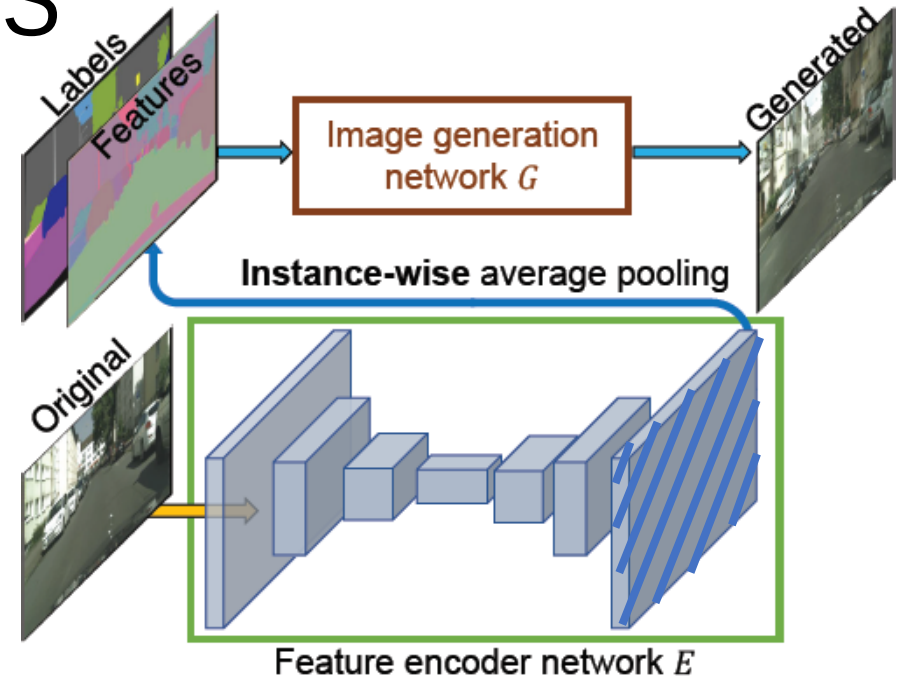
- **Generate higher resolution (2048 x 1024) than before**
  - Multiscale G & D
  - Advanced objective functions
  - Instance Map
- **Semantic manipulating generated images**
  - **Improved structure:**
    - **feature encoder network**
    - Instance-wise pooling
  - Improved input




1. **train simultaneously with G**
2. **not used during inferencing**
3. **the target is trying to get features for each instances**
4. **channel of the last tensor is 3 [in paper]**

# II. GAN: Applications



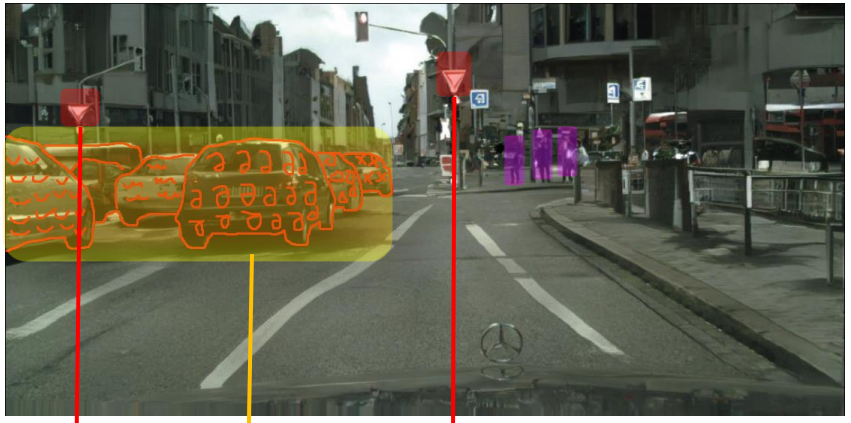

**During training:**

1. **instance-wise average pooling for last tensor**
2. **concat the generated 3-channel tensor to label map**

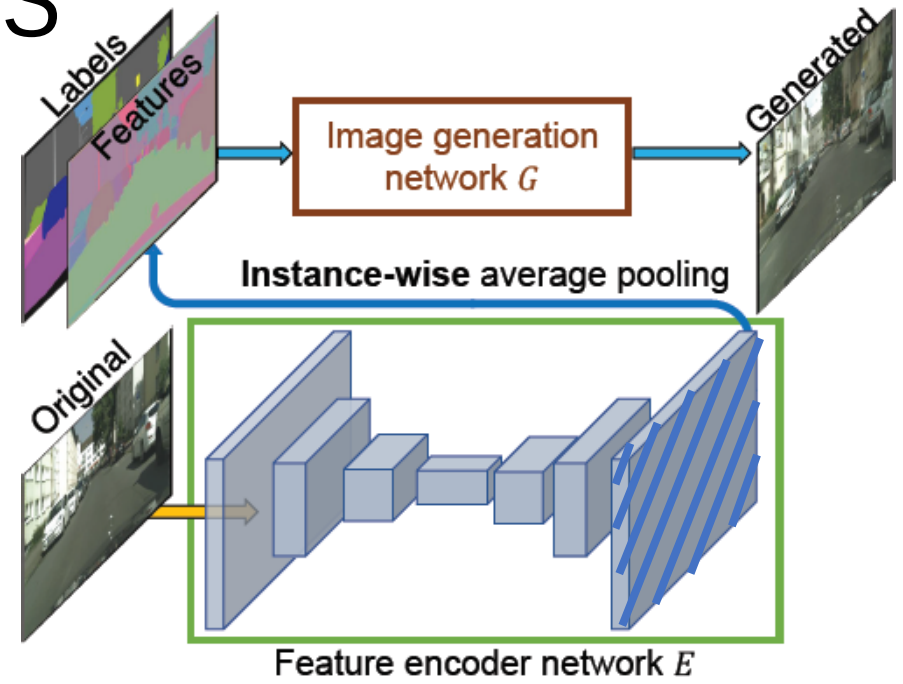
During inferencing:

1. instance-wise average pooling for last tensor
2. k-means to get mean features per category
3. feature for each instance can be manipulated

# II. GAN: Applications



get mean features per category by k-means



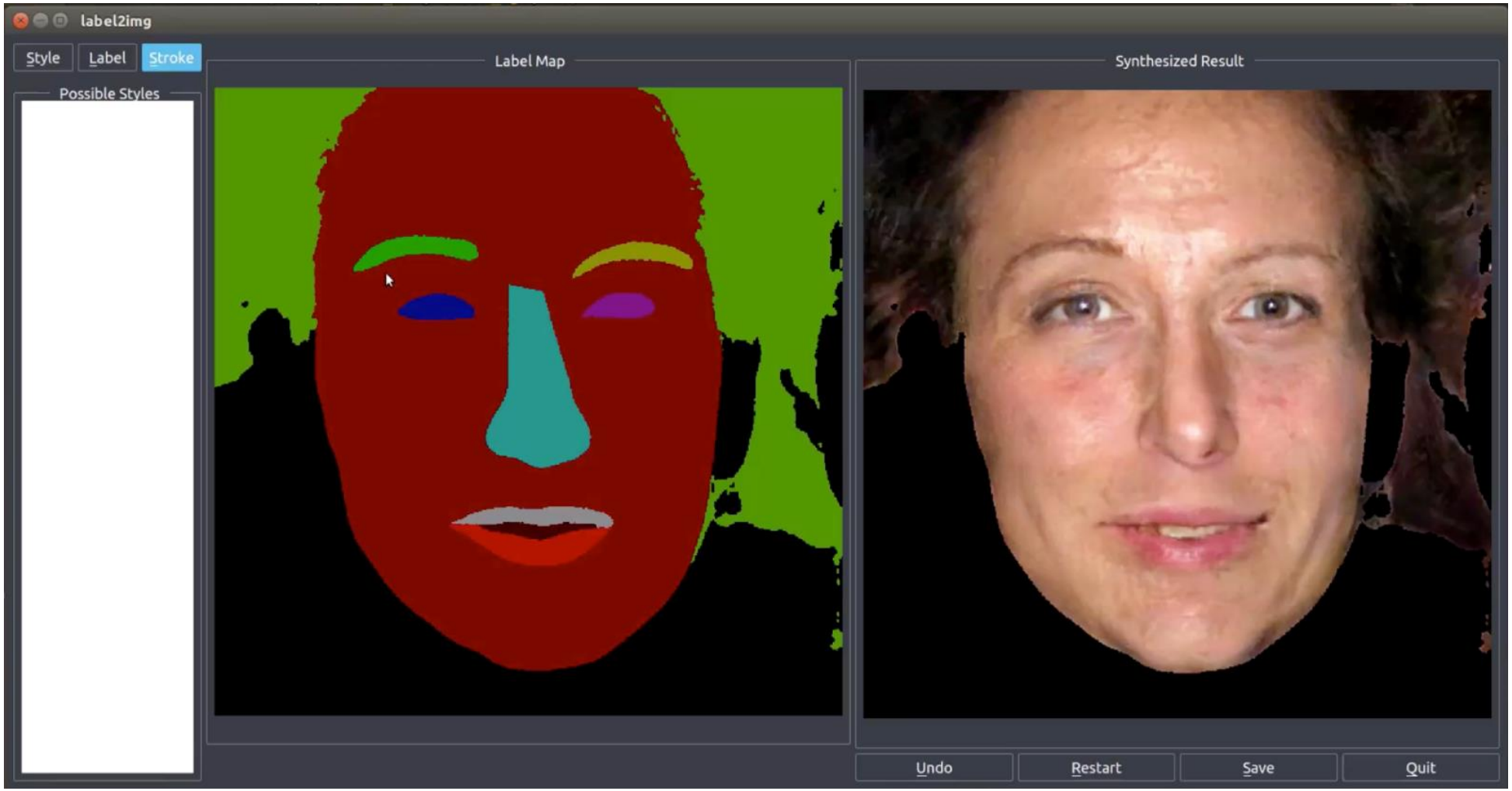

During training:

1. instance-wise average pooling for last tensor
2. concat the generated 3-channel tensor to label map

**During inferencing:**

1. **instance-wise average pooling for last tensor**
2. **k-means to get mean features per category**
3. **feature for each instance can be manipulated**

# II. GAN: Applications

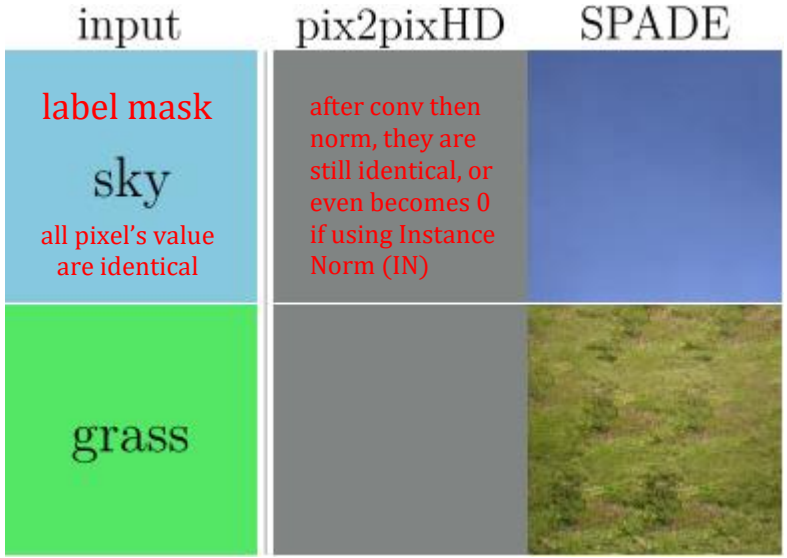


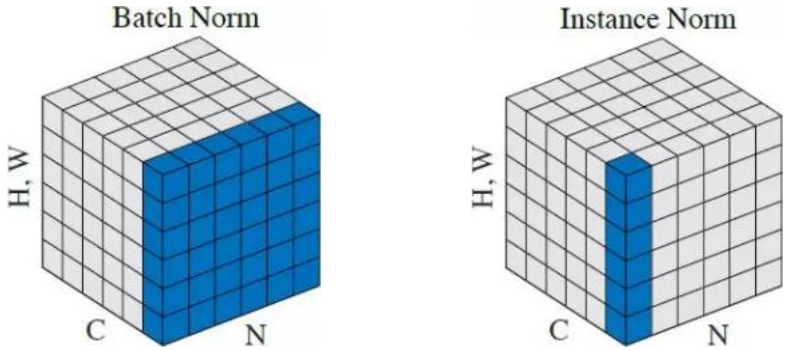
# II. GAN: Applications

## L. SPADE

### SPADE: Drawbacks of pix2pixHD

- **Theoretically**
  normalization can remove semantic information
- **Practically**
  Encoder-Decoder structure is heavy (no need for encoder)
  Results could be improved

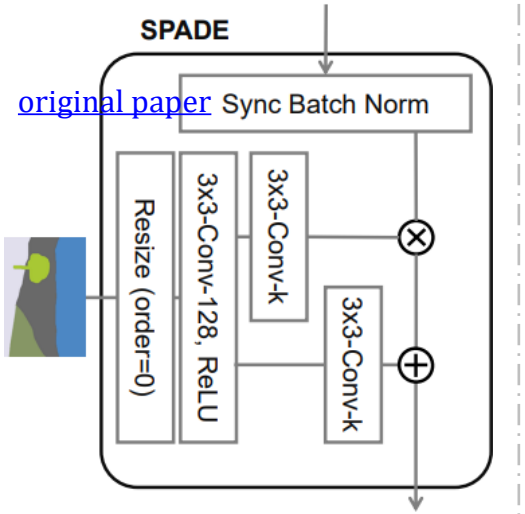


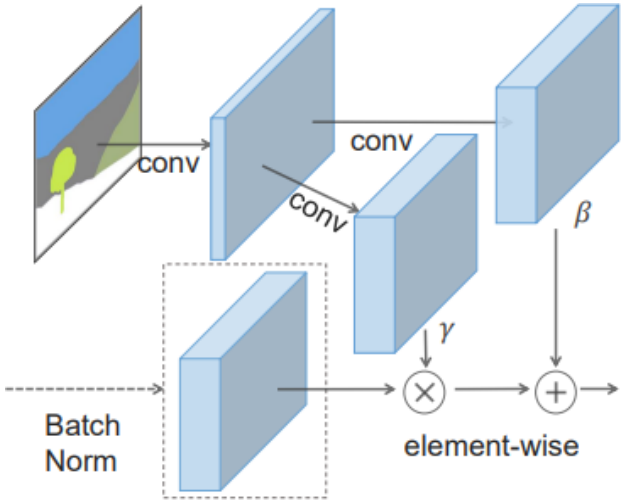



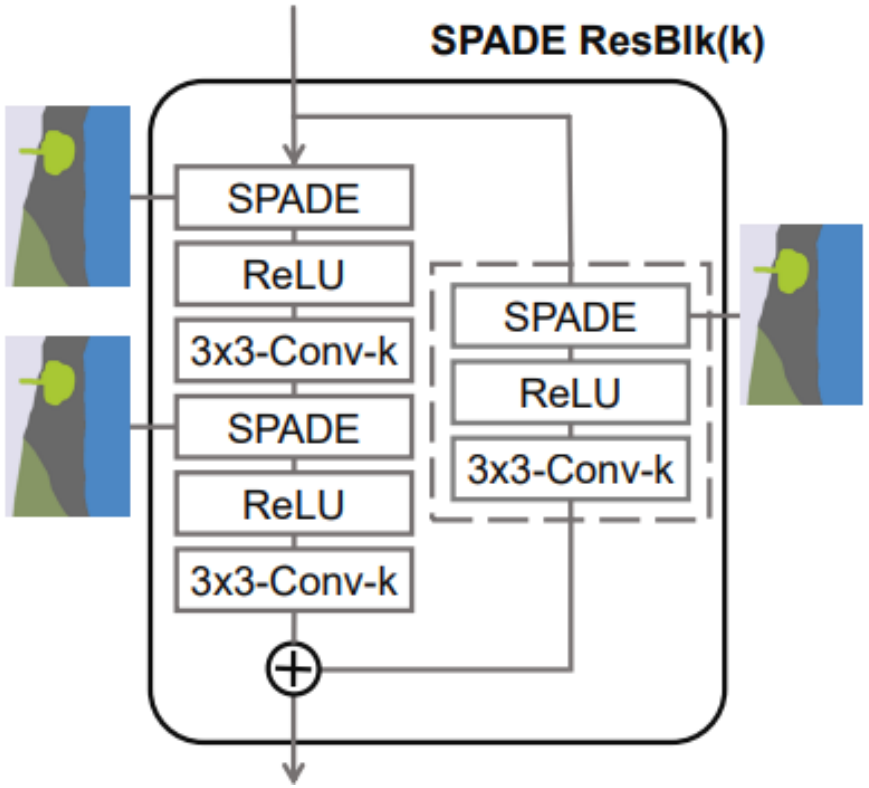
# II. GAN: Applications

## L. SPADE

### SPADE: structures (SPADE / SPADE ResBlock)




[original paper](#)

$n \in N, c \in C^i, y \in H^i, x \in W^i$

$$\gamma^i_{c,y,x}(\mathbf{m})\frac{h^i_{n,c,y,x} - \mu^i_c}{\sigma^i_c} + \beta^i_{c,y,x}(\mathbf{m})$$

$$\mu^i_c = \frac{1}{NH^iW^i}\sum_{n,y,x} h^i_{n,c,y,x}$$

$$\sigma^i_c = \sqrt{\frac{1}{NH^iW^i}\sum_{n,y,x}\left((h^i_{n,c,y,x})^2 - (\mu^i_c)^2\right)}$$

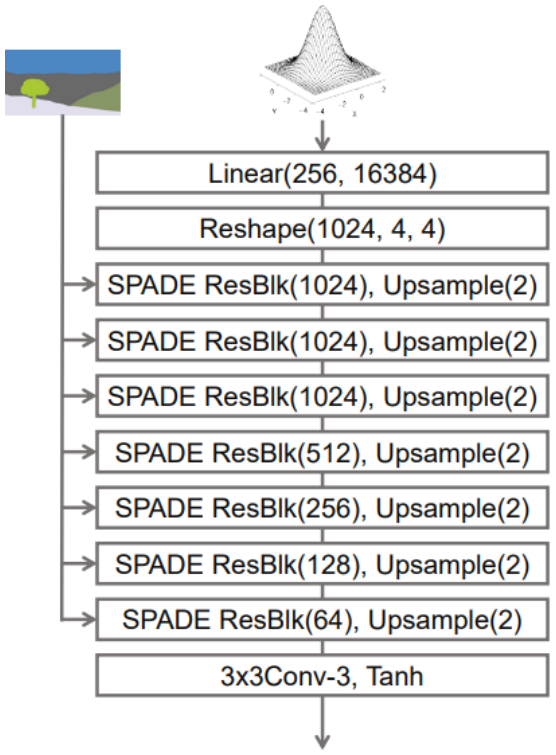
$m \in \mathbb{L}^{H \times W}$:
semantic segmentation mask

$\gamma^i_{c,y,x}(m)$ and $\beta^i_{c,y,x}(m)$:
learned modulation parameters of the normalization layer

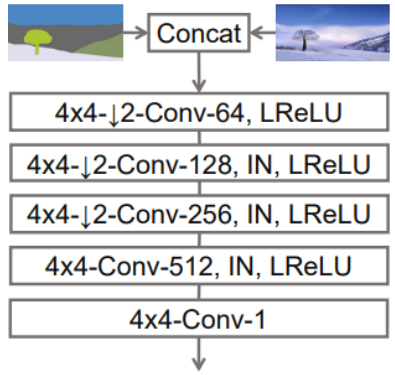
# II. GAN: Applications

## L. SPADE

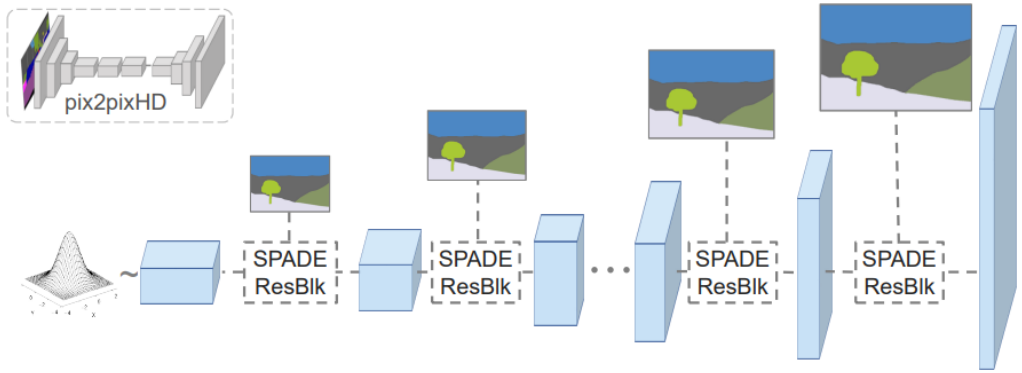
### SPADE: structures (Generator / Discriminator)




- No need for encoder.
- Random input is enough.
- Because SPADE will help to encode spatial information
- [my personal experience] No need to use Tanh



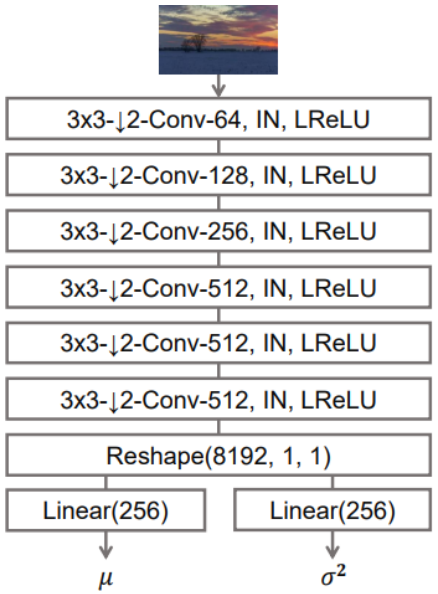

- Conditional GAN
- Multi-Scale
- PatchGAN
- Same with pix2pixHD



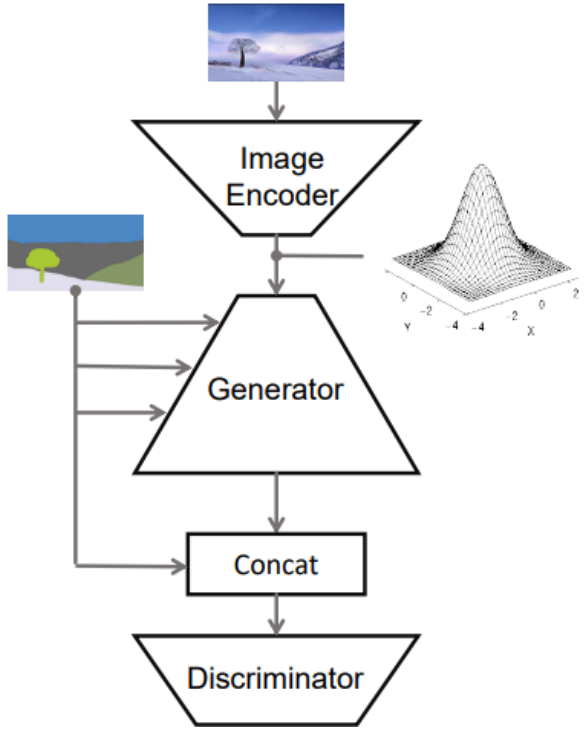
# II. GAN: Applications

## L. SPADE

### SPADE: structures (Encoder / Whole Structure)




- No need for encoder.
  But we could add one to encode desired style
- Several encoding ways
- This work used a very classical way developed in 2013-2014
  original paper
- Introduced a new loss responsible for style learning: KL divergence loss

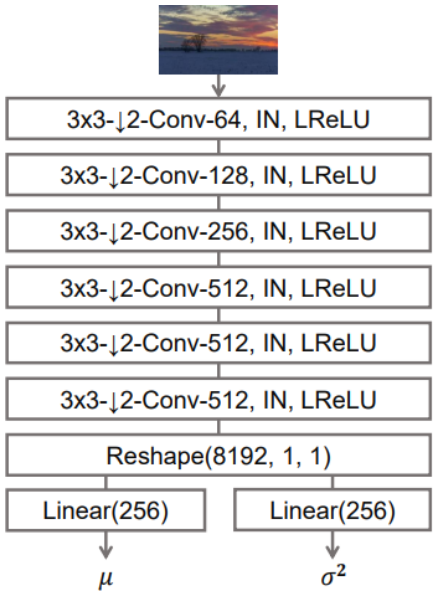



- Losses are same with pix2pixHD
- Only one new loss for style encoding (not a must)
- Could be interactively manipulated as pix2pixHD does
- Original code

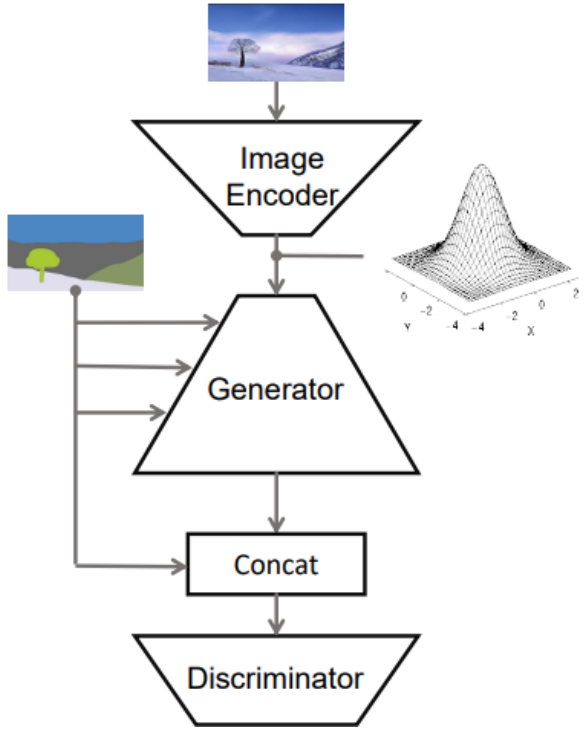
# II. GAN: Applications

## L. SPADE

### SPADE: structures (Encoder / Whole Structure)




- No need for encoder.
  But we could add one to encode desired style
- Several encoding ways
- This work used a very classical way developed in 2013-2014
  original paper
- Introduced a new loss responsible for style learning: KL divergence loss




- Losses are same with pix2pixHD
- Only one new loss for style encoding (not a must)
- Could be interactively manipulated as pix2pixHD does
- Original code

# III. GAN Applications -Defect Generation

# III. GAN: Defect Generation

## M. Defect-GAN [2021 Zhang]

### Introduction

- **Combination of all factors discussed previously**

  Basic training pipeline: Cycle-GAN / StarGAN

  D: PatchGAN + Auxiliary Classifier + WGAN-GP
  (D of StarGAN)

  G: De/Encoder + SPADE + WGAN-GP

  Multi-domains: StarGAN

  New factor: Layer-wise Composition

# III. GAN: Defect Generation

## M. Defect-GAN [2021 Zhang]

### Details

- **Combination**

  **Basic training pipeline: Cycle-GAN / StarGAN**

  D: PatchGAN + Auxiliary Classifier + WGAN-GP

  G: De/Encoder + SPADE + WGAN-GP

  Multi-domains: StarGAN

  New factor: Layer-wise Composition

**Cycle Consistency**

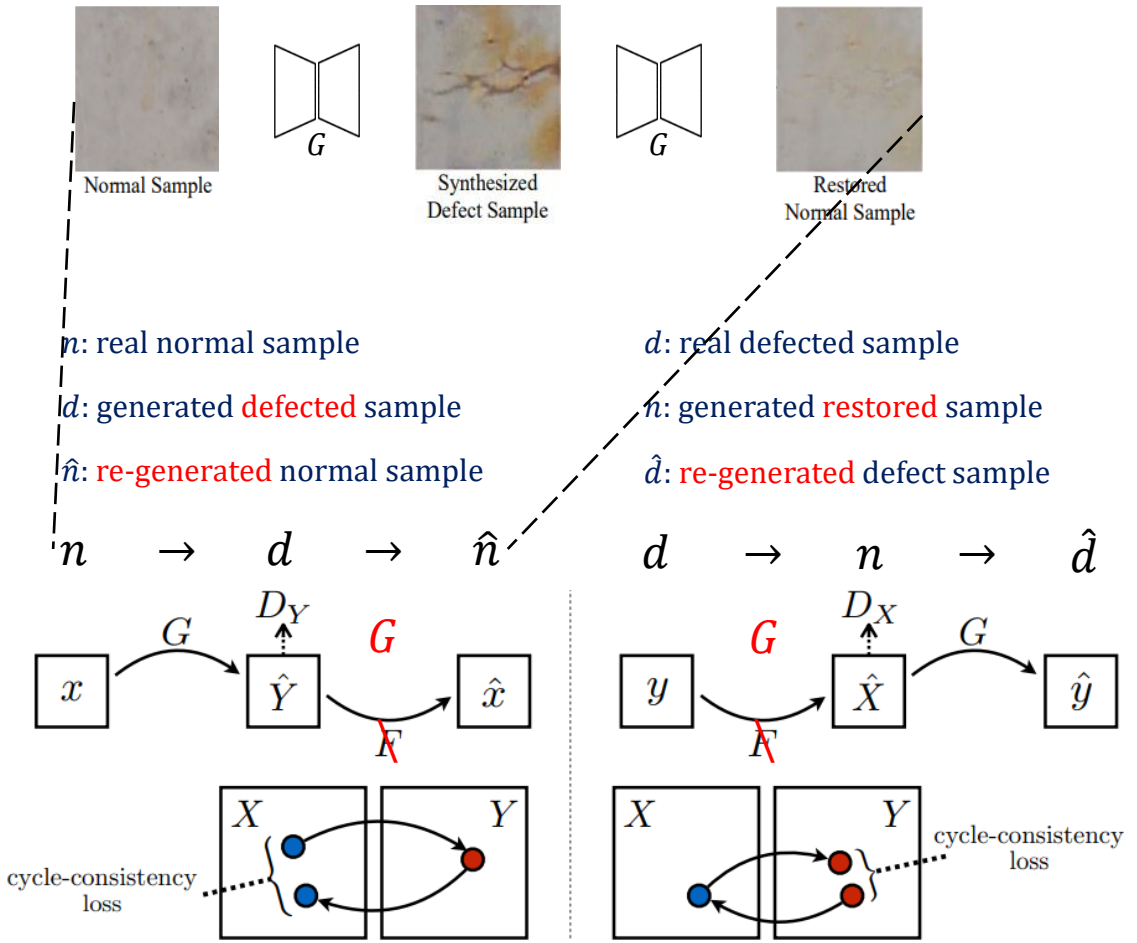
$$\mathcal{L}_{\text{cyc}}(G) = \mathbb{E}_{x\sim p_{data}(x)}[\| G(G(x)) - x \|_1]$$

($\hat{n}/\hat{d}$ and $n/d$ mark the two terms)

**Full objective [according to the pipeline]**

$$\mathcal{L}(G, D_x, D_y) = \mathcal{L}_{\text{GAN}}(G, X, Y) + \mathcal{L}_{\text{GAN}}(G, Y, X) + \lambda\mathcal{L}_{\text{cyc}}(G) + [\mathcal{L}_{\text{identity}}(G)]$$

not used in paper



# III. GAN: Defect Generation

## M. Defect-GAN [2021 Zhang]

### Details

- **Combination**

  Basic training pipeline: Cycle-GAN / StarGAN

  D: PatchGAN + Auxiliary Classifier + WGAN-GP

```
class Discriminator(nn.Module):
    """Discriminator network with PatchGAN."""  combined with WGAN-GP
    def __init__(self, image_size=128, conv_dim=64, c_dim=5, repeat_num=6):
        super(Discriminator, self).__init__()
        layers = []
        layers.append(nn.Conv2d(3, conv_dim, kernel_size=4, stride=2, padding=1))
        layers.append(nn.LeakyReLU(0.01))

        curr_dim = conv_dim
        for i in range(1, repeat_num):
            layers.append(nn.Conv2d(curr_dim, curr_dim*2, kernel_size=4, stride=2, padding=1))
            layers.append(nn.LeakyReLU(0.01))
            curr_dim = curr_dim * 2

        kernel_size = int(image_size / np.power(2, repeat_num))
        self.main = nn.Sequential(*layers)
        self.conv1 = nn.Conv2d(curr_dim, 1, kernel_size=3, stride=1, padding=1, bias=False)
        self.conv2 = nn.Conv2d(curr_dim, c_dim, kernel_size=kernel_size, bias=False)
        convert tensor shape to (n, c, 1, 1)
    def forward(self, x):
        h = self.main(x)
        out_src = self.conv1(h)   auxiliary classifier [AC-GAN]
        out_cls = self.conv2(h)   convert tensor shape to (n, c)
        return out_src, out_cls.view(out_cls.size(0), out_cls.size(1))
```

PatchGAN [pix2pix]

```
# =============================
#  Train the discriminator
# =============================

# Compute loss with real images.
out_src, out_cls = self.D(x_real)
d_loss_real = - torch.mean(out_src)
d_loss_cls = self.classification_loss(out_cls, label_org, self.dataset)

# Compute loss with fake images.
x_fake = self.G(x_real, c_trg)
out_src, out_cls = self.D(x_fake.detach())
d_loss_fake = torch.mean(out_src)

# Compute loss for gradient penalty.
alpha = torch.rand(x_real.size(0), 1, 1, 1).to(self.device)
x_hat = (alpha * x_real.data + (1 - alpha) * x_fake.data).requires_grad_(True)
out_src, _ = self.D(x_hat)
d_loss_gp = self.gradient_penalty(out_src, x_hat)

# Backward and optimize.  adv loss of WGAN  classification loss (cross entropy)  adv loss of gradient penalty
d_loss = d_loss_real + d_loss_fake + self.lambda_cls * d_loss_cls + self.lambda_gp * d_loss_gp
self.reset_grad()
d_loss.backward()
self.d_optimizer.step()
```

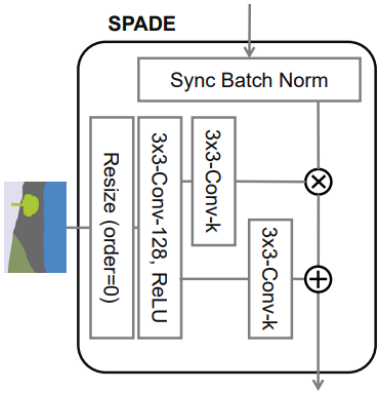
# III. GAN: Defect Generation

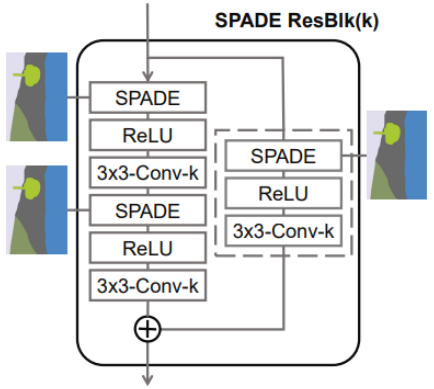
## M. Defect-GAN [2021 Zhang]

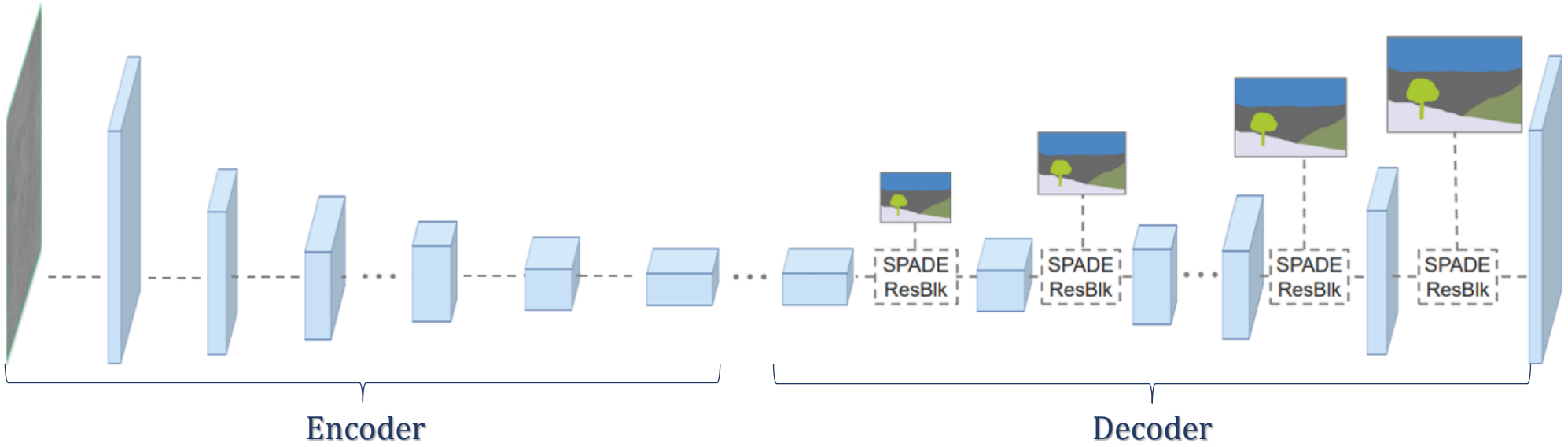
### Details

- **Combination**

  G: De/Encoder + SPADE + WGAN-GP [basic version]



# III. GAN: Defect Generation

## M. Defect-GAN [2021 Zhang]

### Introduction

- **Combination**

  G: De/Encoder + SPADE

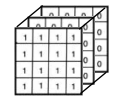
  [advanced version]

  Multi-domains: StarGAN

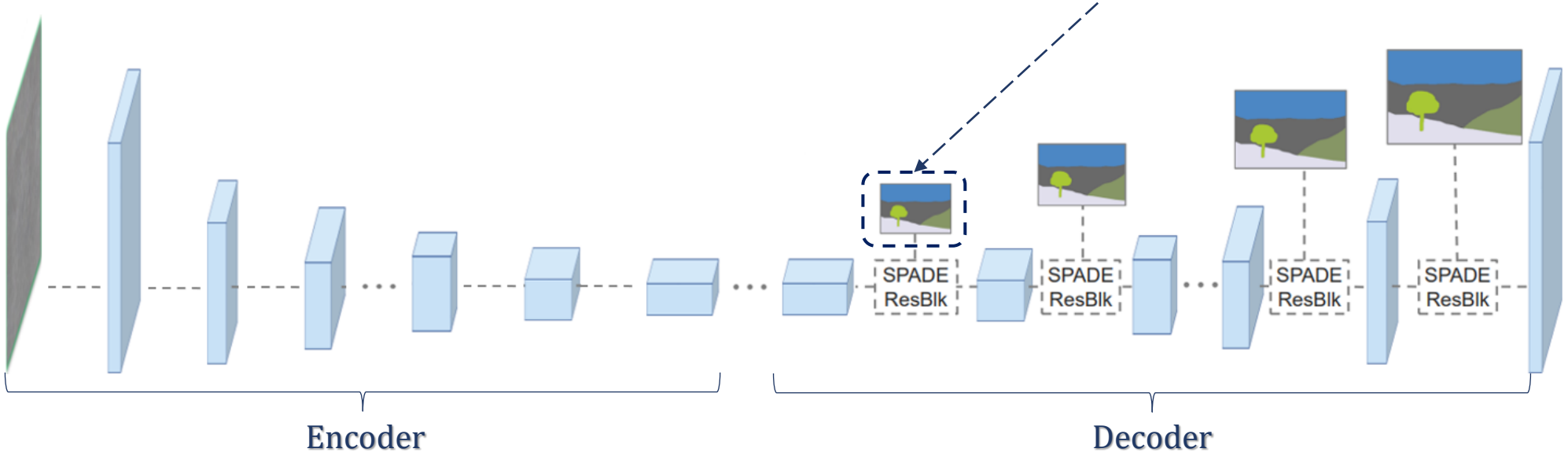
during training stage, the attribute controlling map $A$ should be constant for all locations of the image, i.e., $A$ is acquired by spatial-wisely repeating the target defect label $c \in R^C$. This restriction can be lifted during inference stage, which enables Defect-GAN to add defects at different location(s) in a context-compatible manner.

Spatial & Categorical Control Map:

$A \in \mathcal{R}^{H \times W \times C}$



Phase { Training: / Inferencing:



# III. GAN: Defect Generation

## M. Defect-GAN [2021 Zhang]

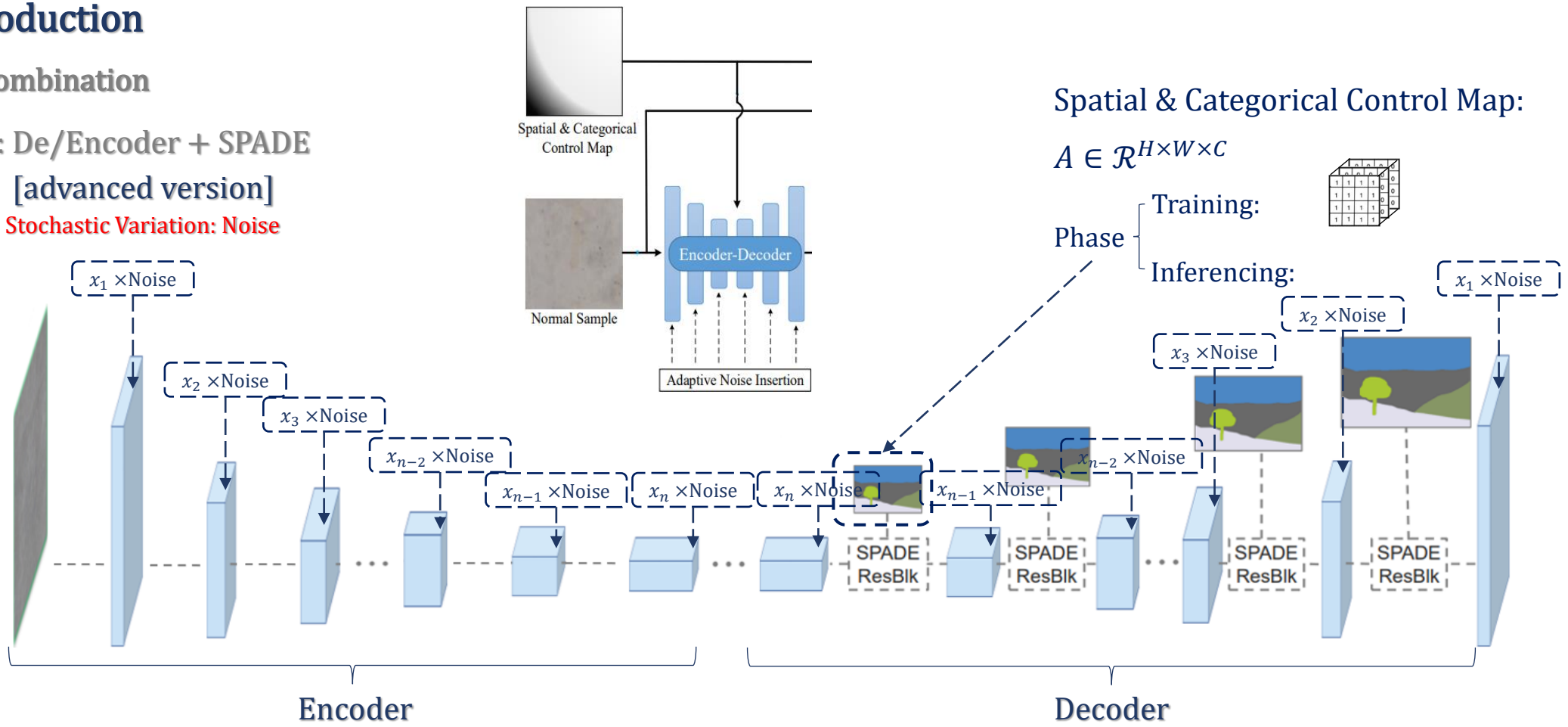
### Introduction

- **Combination**

  G: De/Encoder + SPADE

  [advanced version]

  Stochastic Variation: Noise



# III. GAN: Defect Generation

## M. Defect-GAN [2021 Zhang]

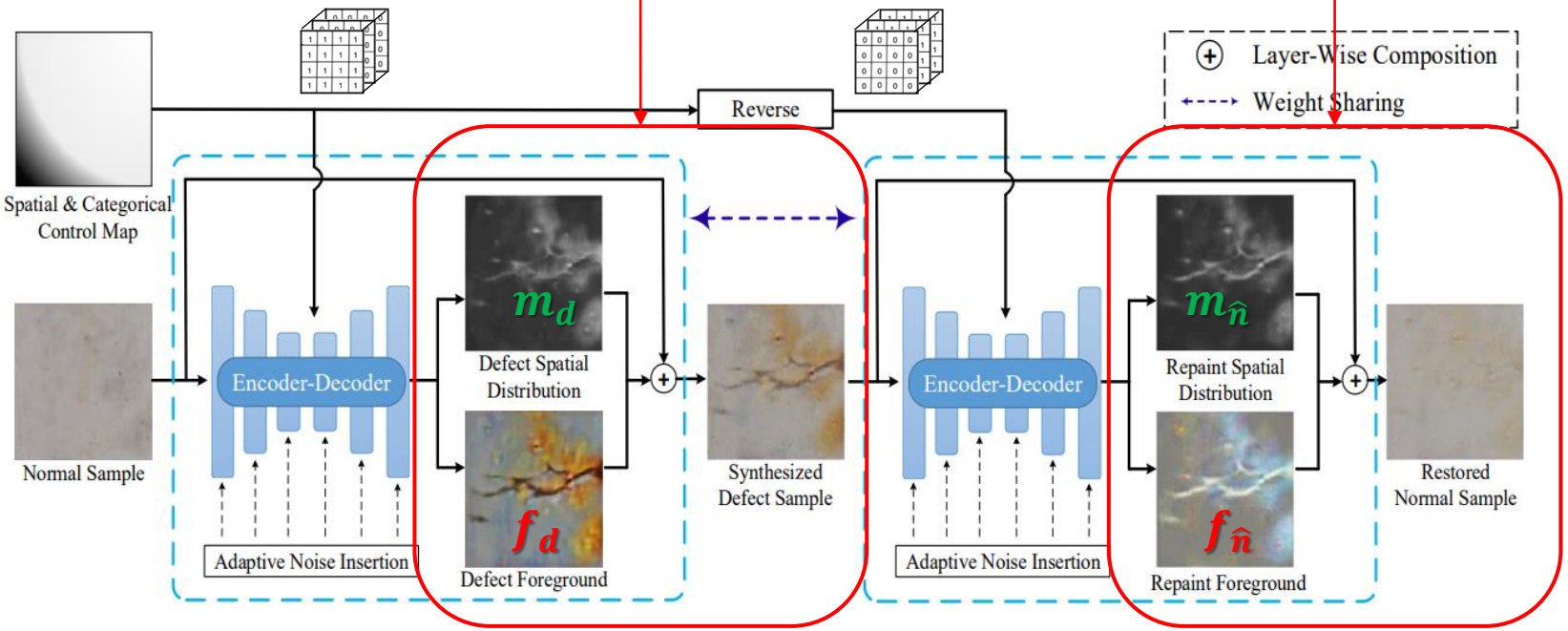
**Introduction**

- **Combination**

$$f_d, m_d = G(n, A_{n\to d})$$
$$d = n \odot (1 - m_d) + f_d \odot m_d$$

$$f_{\hat{n}}, m_{\hat{n}} = G(d, A_{d\to \hat{n}})$$
$$\hat{n} = d \odot (1 - m_{\hat{n}}) + f_{\hat{n}} \odot m_{\hat{n}}$$

New factor: Layer-wise Composition [full version]




**Spatial Distribution Cycle Loss**

$$\mathcal{L}_{sd-cyc}(G) = \mathbb{E}_{m_n, m_{\hat{n}}}[\| m_n - m_{\hat{n}} \|_1]$$

**Spatial Distribution Constrain Loss**

$$\mathcal{L}_{sd-con}(G)$$
$$= \mathbb{E}_{m_n, m_{\hat{n}}}[\| m_n - 0 \|_1 + \| m_{\hat{n}} - 0 \|_1]$$

[just a regularization part to avoid $f_d$ and $f_{\hat{n}}$ be too large]

**Full Loss**

$$\mathcal{L}_D = -\mathcal{L}_{adv} + \lambda^r_{cls}\mathcal{L}^r_{cls}$$
$$\mathcal{L}_G = \mathcal{L}_{adv} + \lambda^f_{cls}\mathcal{L}^f_{cls} + \lambda_{rec}\mathcal{L}_{rec} + \lambda_{con}\mathcal{L}_{sd-cyc} + \lambda_c\mathcal{L}_{sd-con}$$

# III. GAN: Defect Generation