

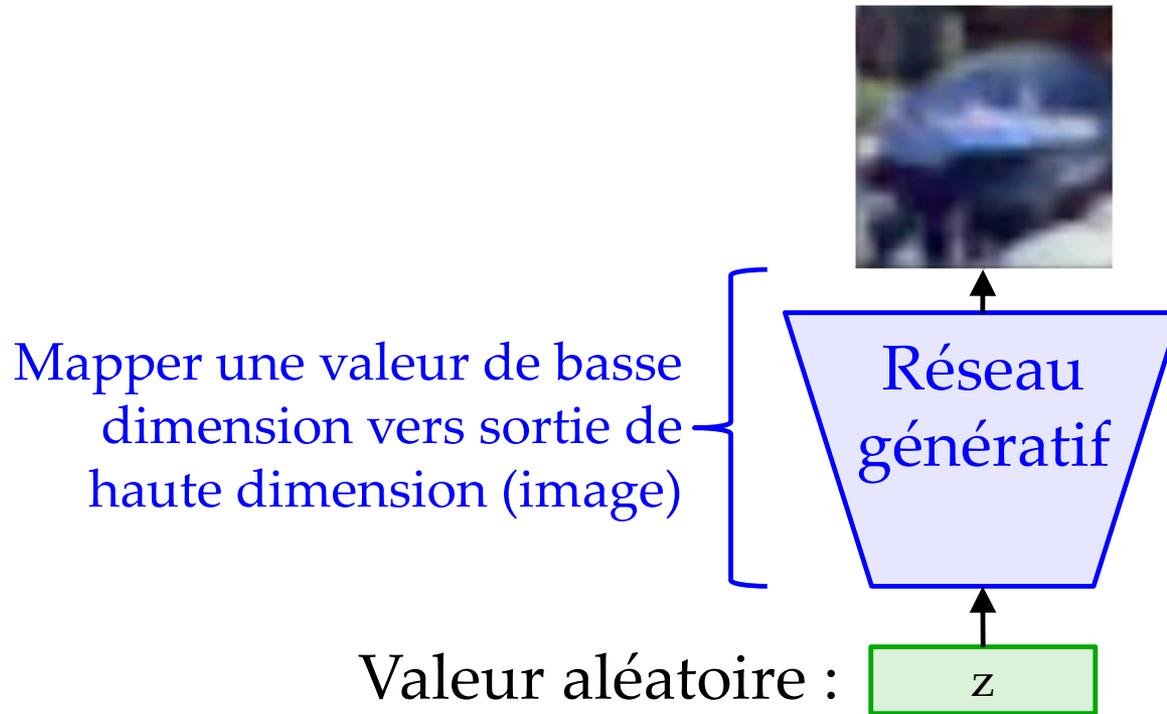


UNIVERSITÉ
LAVAL

GLO-4030/7030 APPRENTISSAGE PAR RÉSEAUX DE NEURONES PROFONDS

GAN : Generative Adversarial Networks

Réseau génératif



Exemple : Variational Autoencoder (VAE)
GAN

GAN

- Approche inspirée de la théorie des jeux
- **Réseau génératif** : essaie de confondre le réseau discriminatif
- **Réseau discriminatif** (critique) : essaie de distinguer entre les images réels et les fausses



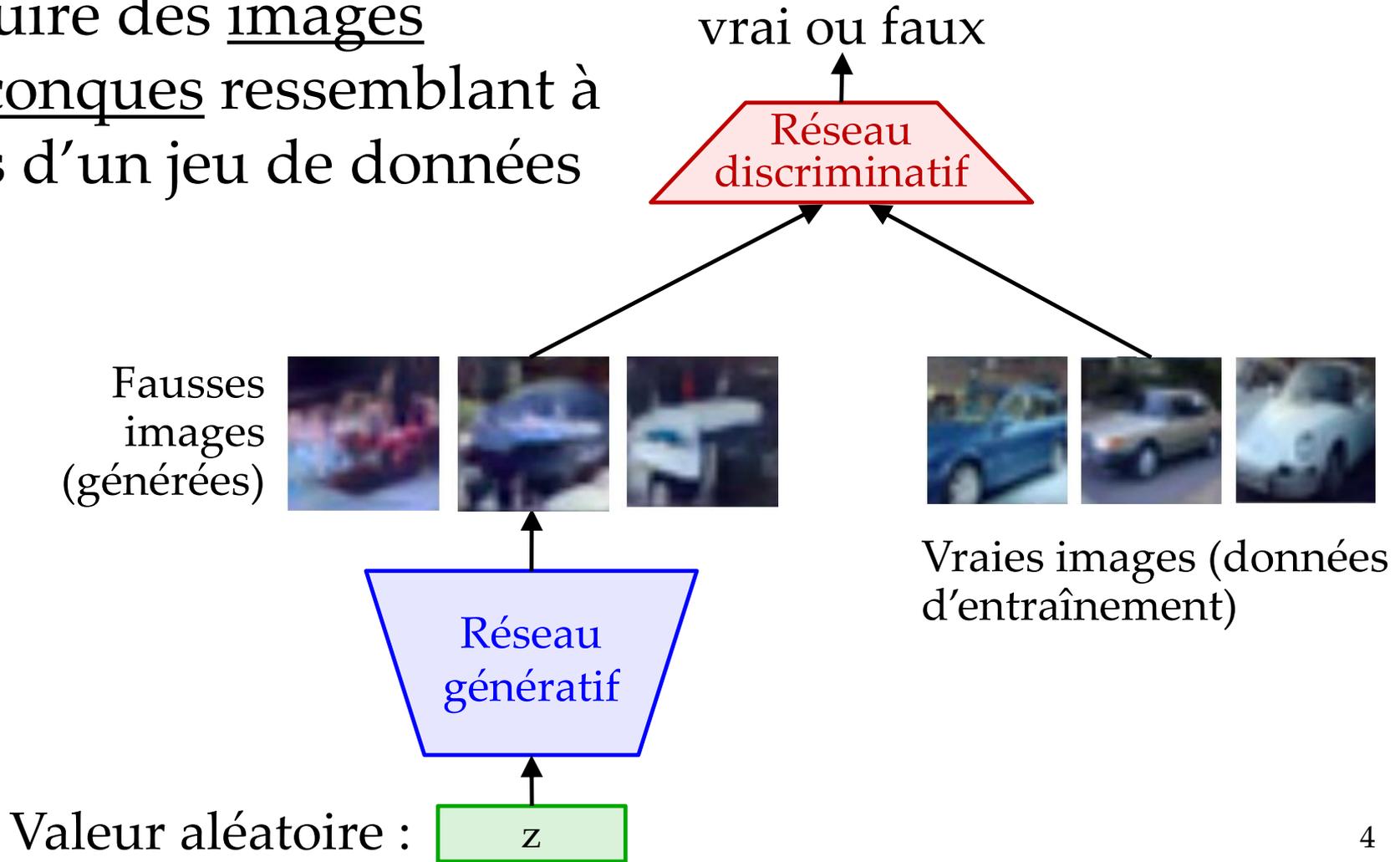
?



?

GAN

Générateur cherche à produire des images quelconques ressemblant à celles d'un jeu de données



GAN

- Ne cherche pas à modéliser explicitement la densité (manifold)
- Approche inspirée de la théorie des jeux : jeu minimax à 2 joueurs

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

GAN

- Discriminateur D (sortie 0 à 1) change ses poids afin de maximiser V

$$\min_G \max_D V(D, G) = \underbrace{\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}}_{\text{Vraies données}} [\log \overbrace{D(\mathbf{x})}^{\uparrow 1}] + \underbrace{\mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}}_{\text{Fausses données}} [\log(1 - \overbrace{D(G(\mathbf{z}))}^{\downarrow 0})]$$

$$\begin{aligned}\log(1) &= 0 \\ \log(0.1) &= -1 \\ \log(0) &= -\text{Inf}\end{aligned}$$

GAN

- Générateur G (sortie image) cherche à confondre le discriminateur D, afin de minimiser V

$$\min_G \max_D V(D, G) = \underbrace{\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}}_{\text{Vraies données}} [\log \overbrace{D(\mathbf{x})}^{\downarrow 0}] + \underbrace{\mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}}_{\text{Fausses données}} [\log(1 - \overbrace{D(G(\mathbf{z}))}^{\uparrow 1})]$$

$$\log(1) = 0$$

$$\log(0.1) = -1$$

$$\log(0) = -\text{Inf}$$

Entraîner le GAN

- Pour cette fonction objective V

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

- Alternance entre :
 - **Montée** du gradient pour le **discriminateur**

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

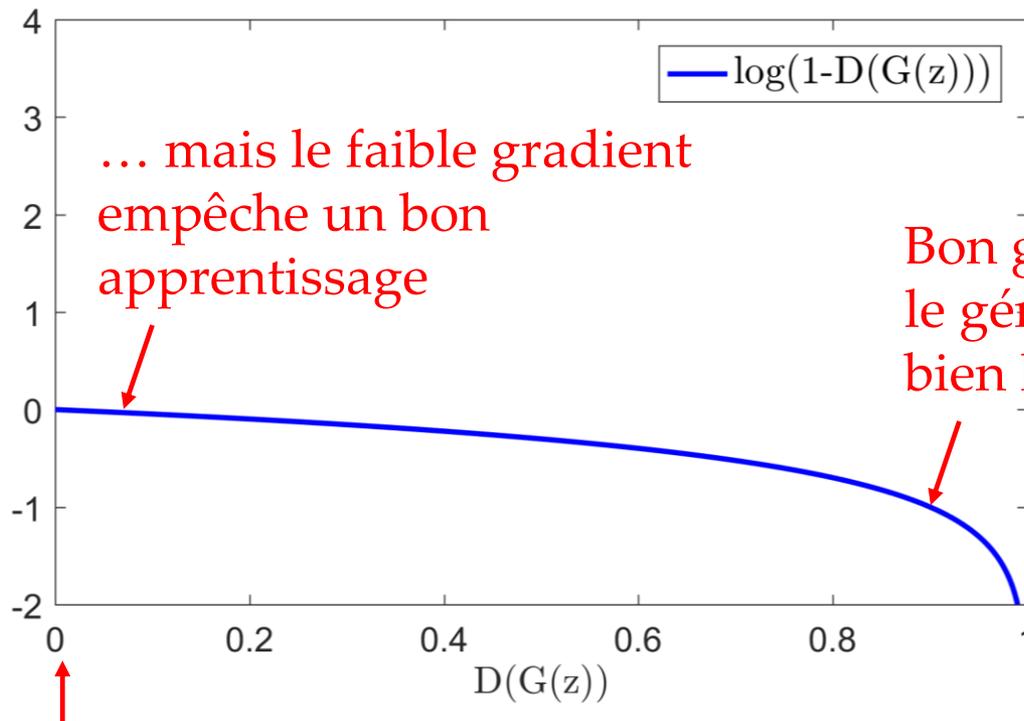
- **Descente** du gradient pour le générateur

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

Instabilité entraînement

- Cette fonction de perte de G est peu commode

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



... mais le faible gradient empêche un bon apprentissage

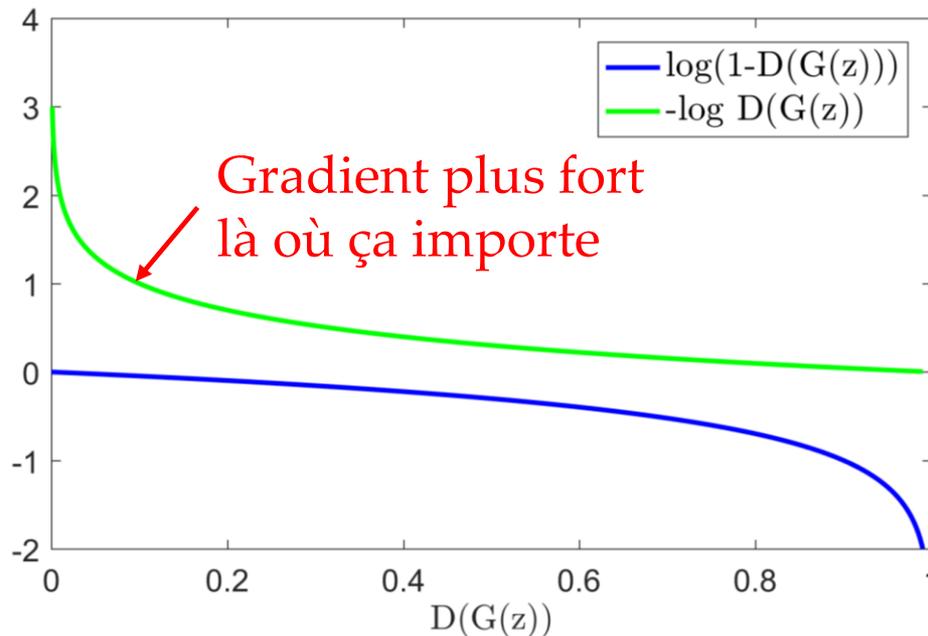
Bon gradient inutile, car le générateur confond bien le discriminateur

Discriminateur a démasqué le générateur

Version améliorée 1

- **Montée** de gradient pour le générateur

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



En bref, la forme de la fonction de perte importe beaucoup!

Algorithme ca. 2016

for number of training iterations **do**

for k steps **do**

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

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

end for

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

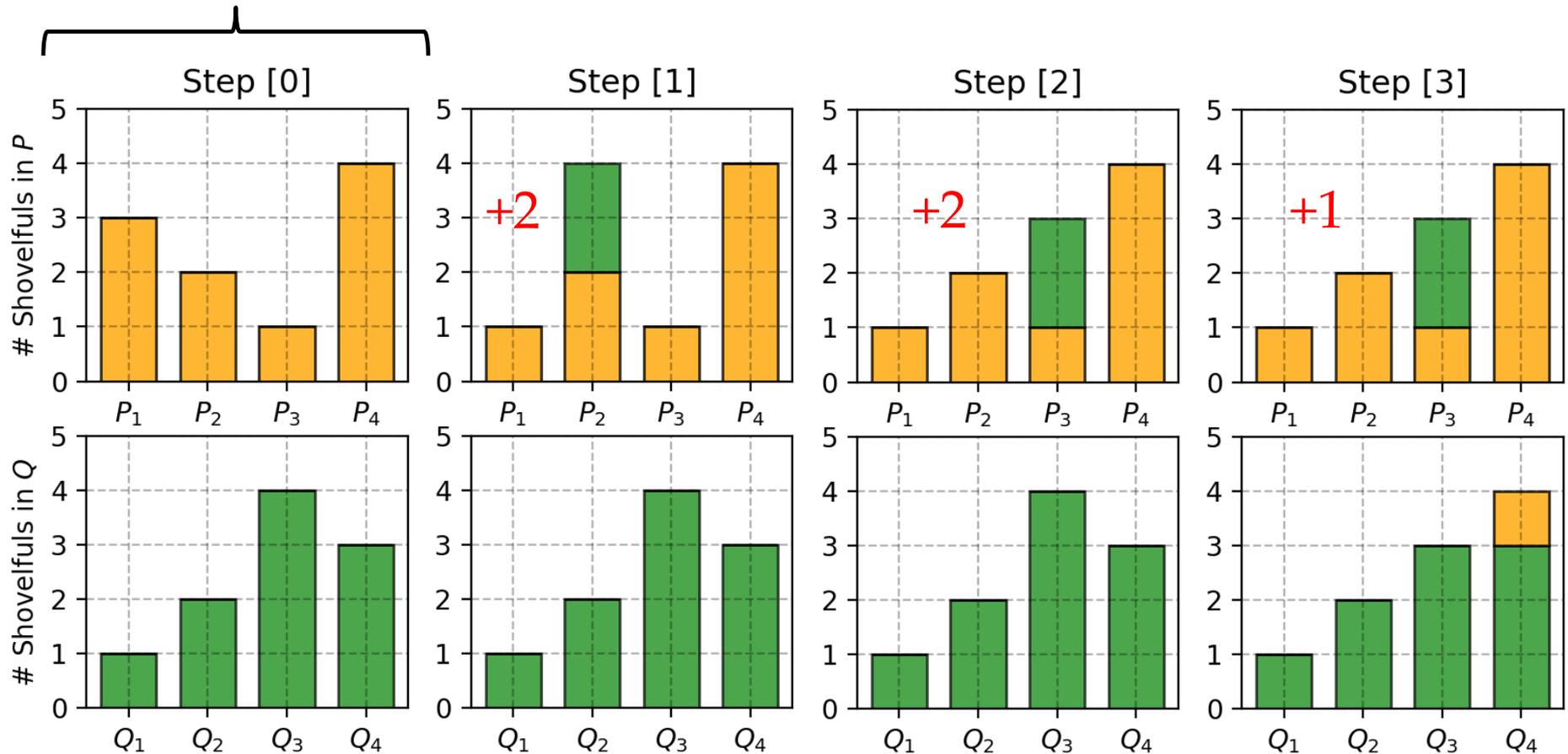
Version améliorée 2

- Fonction perte précédente difficile à entraîner
- Autre interprétation du GAN : on veut que les distributions $D(G(z))$ et $D(x)$ soient les plus proches possibles
- Métrique de distance sur distributions
- Perte sur distance Wasserstein
 - Earth mover distance (distance du cantonnier)
 - (quantité déplacée) x (distance déplacement)

Distance de Wasserstein

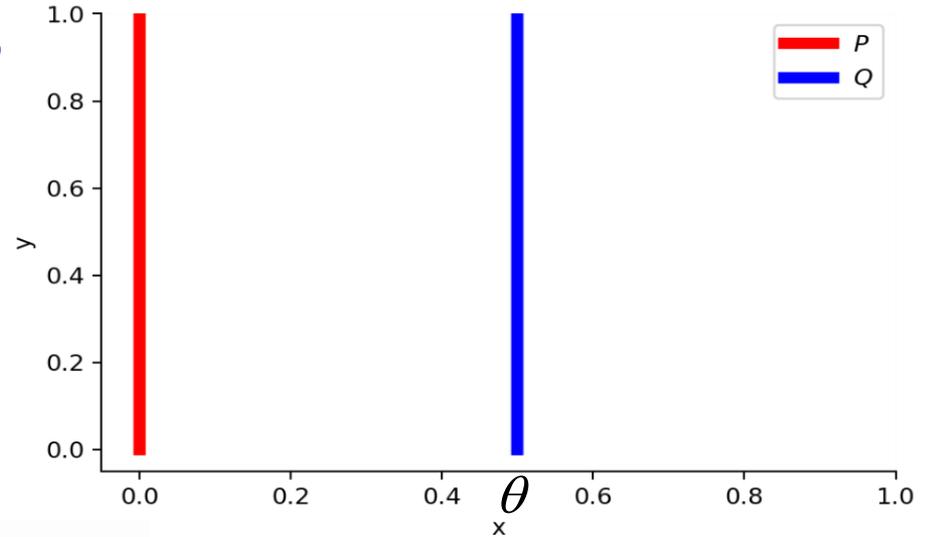
Mesurer la distance entre ces 2 distributions

Distance Wasserstein discrète = $2+2+1 = 5$



Comparaison des distances

KL : Kullback-Leibler
 JS : Jensen-Shannon
 W : Wasserstein



$$\theta = 0 \left\{ \begin{array}{l} D_{KL}(P||Q) = D_{KL}(Q||P) = D_{JS}(P, Q) = 0 \\ W(P, Q) = 0 = |\theta| \end{array} \right.$$

$$\theta \neq 0 \left\{ \begin{array}{l} D_{KL}(P||Q) = \sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty \\ D_{KL}(Q||P) = \sum_{x=\theta, y \sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty \\ D_{JS}(P, Q) = \frac{1}{2} \left(\sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=\theta, y \sim U(0,1)} 1 \cdot \log \frac{1}{1/2} \right) = \log 2 \\ W(P, Q) = |\theta| \end{array} \right.$$

**Pas besoin d'avoir
le même support**

WGAN

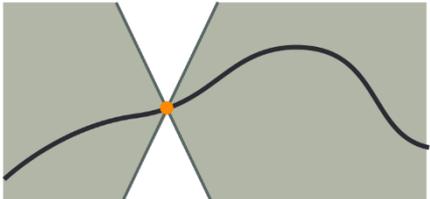
Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α , the learning rate. c , the clipping parameter. m , the batch size. n_{critic} , the number of iterations of the critic per generator iteration.

Require: : w_0 , initial critic parameters. θ_0 , initial generator's parameters.

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

doit être K-lipschitzienne



“Weight clipping is a clearly terrible way to enforce a Lipschitz constraint”

A real-valued function $f : \mathbb{R} \rightarrow \mathbb{R}$ is called K -Lipschitz continuous if there exists a real constant $K \geq 0$ such that, for all $x_1, x_2 \in \mathbb{R}$,

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

WGAN-GP (Gulrajani et al., 2017)

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 λ , the number of critic iterations per generator iteration n_{critic} , the batch size m , Adam hyperparameters α, β_1, β_2 .

Require: initial critic parameters w_0 , initial generator parameters θ_0 .

```
1: while  $\theta$  has not converged do
2:   for  $t = 1, \dots, n_{\text{critic}}$  do
3:     for  $i = 1, \dots, m$  do
4:       Sample real data  $\mathbf{x} \sim \mathbb{P}_r$ , latent variable  $\mathbf{z} \sim p(\mathbf{z})$ , a random number  $\epsilon \sim U[0, 1]$ .
5:        $\tilde{\mathbf{x}} \leftarrow G_{\theta}(\mathbf{z})$ 
6:        $\hat{\mathbf{x}} \leftarrow \epsilon \mathbf{x} + (1 - \epsilon) \tilde{\mathbf{x}}$  Plus de weight clipping!
7:        $L^{(i)} \leftarrow D_w(\tilde{\mathbf{x}}) - D_w(\mathbf{x}) + \lambda(\|\nabla_{\hat{\mathbf{x}}} D_w(\hat{\mathbf{x}})\|_2 - 1)^2$ 
8:     end for
9:      $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$ 
10:  end for
11:  Sample a batch of latent variables  $\{\mathbf{z}^{(i)}\}_{i=1}^m \sim p(\mathbf{z})$ .
12:   $\theta \leftarrow \text{Adam}(\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m -D_w(G_{\theta}(\mathbf{z})), \theta, \alpha, \beta_1, \beta_2)$ 
13: end while
```

Mode collapse

- Rappel : le GAN génère des images quelconques (on ne peut spécifier explicitement le type d'image désiré)
- Générateur risque de perfectionner un/quelques styles d'image



Significant degree of mode collapse in the GAN MLP [1]

Métrique d'évaluation

- **Classification d'image** : précision
- **GAN** : quantifier le taux de réalisme!
- Problématique similaire à la traduction
 - BLEU score n'est pas toujours représentatif
- Parfois recourt à Amazon Mechanical Turk, pour exploiter jugement humain

Inception score IS

- Réseau Inception préentraîné sur ImageNet
- IS Corrèle bien avec scores humains sur CIFAR-10

images générées p est calculé avec Inception

$$IS(G) = \exp \left(\mathbb{E}_{\mathbf{x} \sim p_g} D_{KL} \left(\overbrace{p(y|\mathbf{x}) \parallel p(y)} \right) \right)$$

- Plus IS est grand, mieux c'est
- Relié à l'information mutuelle entre marginal et conditionnel :
 - Inception est confiant qu'il n'y a qu'un objet dans l'image $\rightarrow p(y|\mathbf{x})$ a une faible entropie
 - Générateur produit une grande variété d'image $\rightarrow p(y)$ a une grande entropie

GAN MLP → CNN

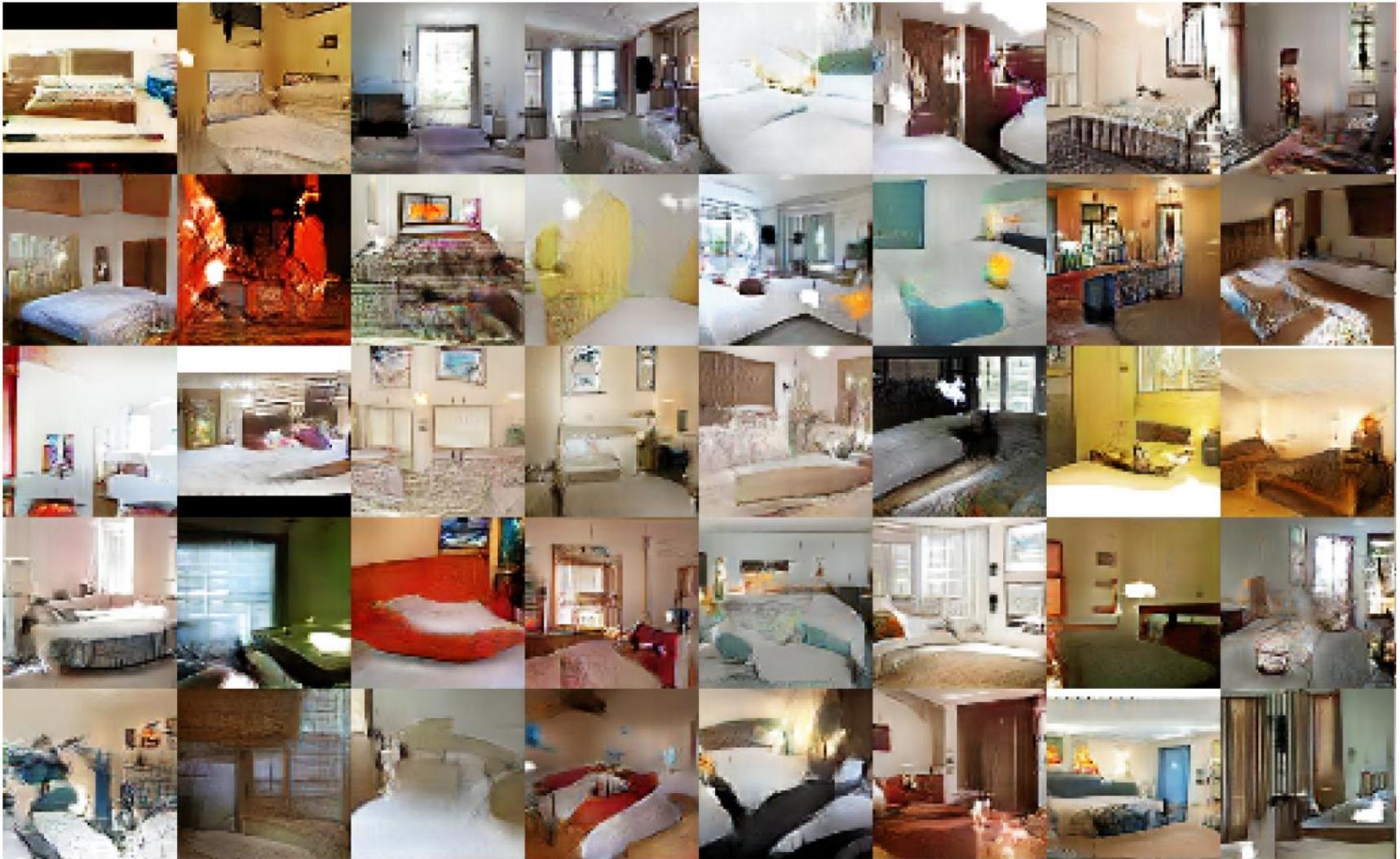
- Approches précédentes basées sur MLP
- Radford et al. proposent des règles pour utiliser des CNN : DCGAN

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

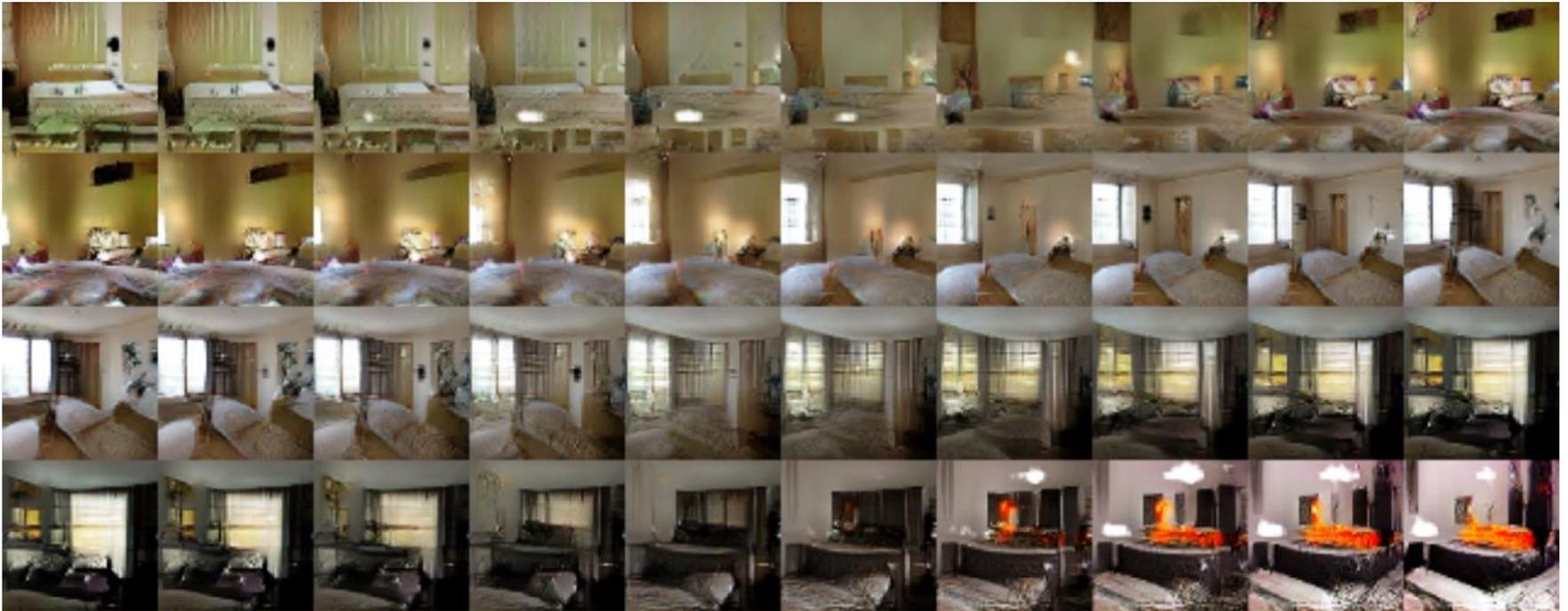
Résultats (64x64 pixels)



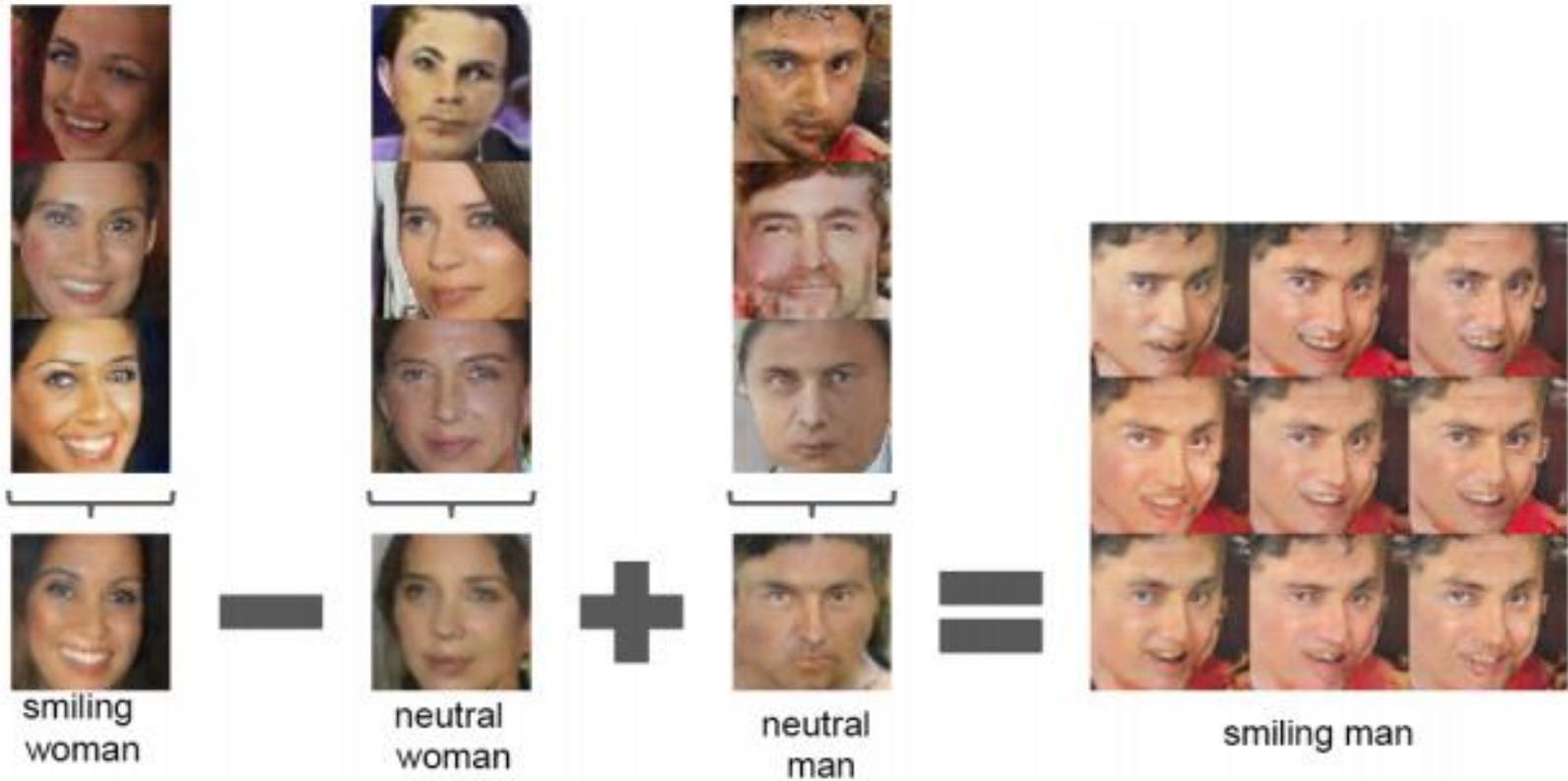
Entraîné sur LSUN bedroom dataset, 3 millions d'images.

Interpolation sur z

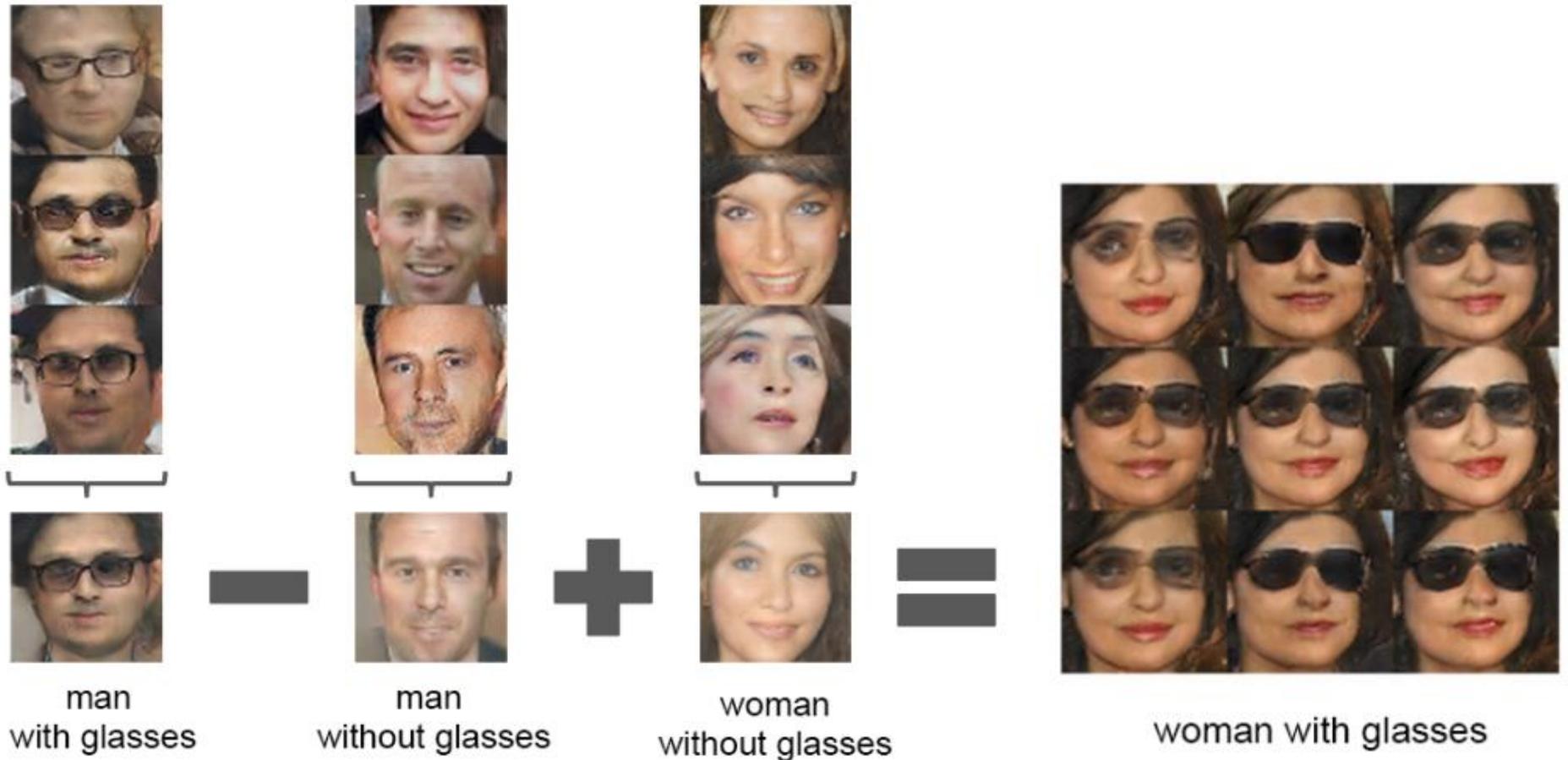
- Permet d'évaluer si le réseau a appris par cœur (☹) ou non (☺) les données d'entraînement
- Absence de transitions brusques est bon signe!



Algèbre sur z



Algèbre sur z



CycleGAN

- Pour effectuer des transferts de style

Monet ↔ Photos



Monet → photo

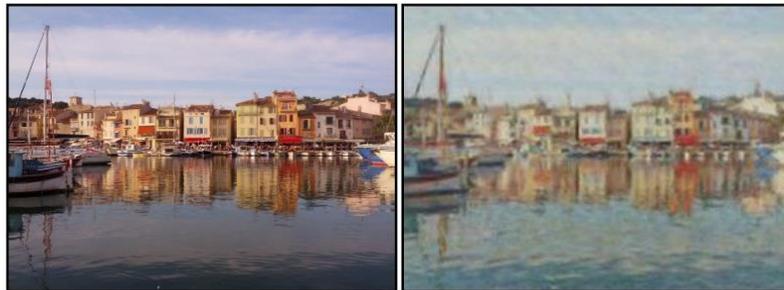


photo → Monet

Summer ↔ Winter



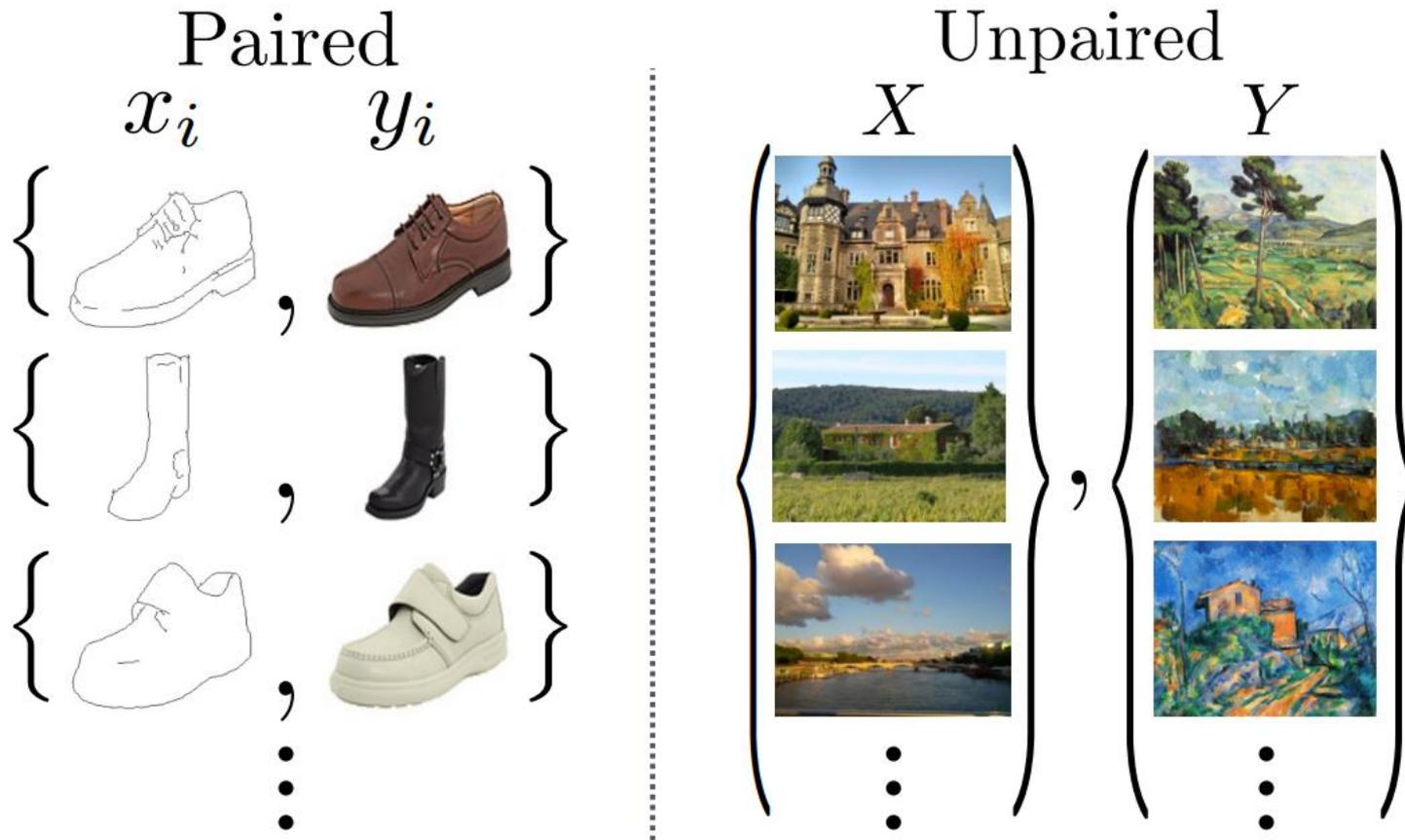
summer → winter



winter → summer

CycleGAN : faiblement supervisé

- Pas besoin d'apparier les images



CycleGAN : faiblement supervisé

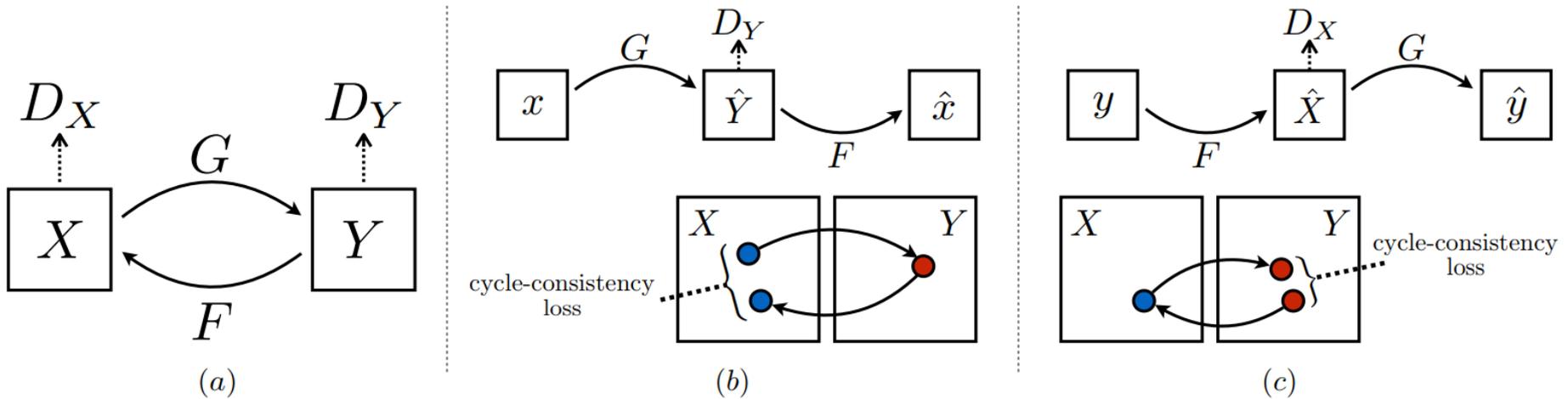
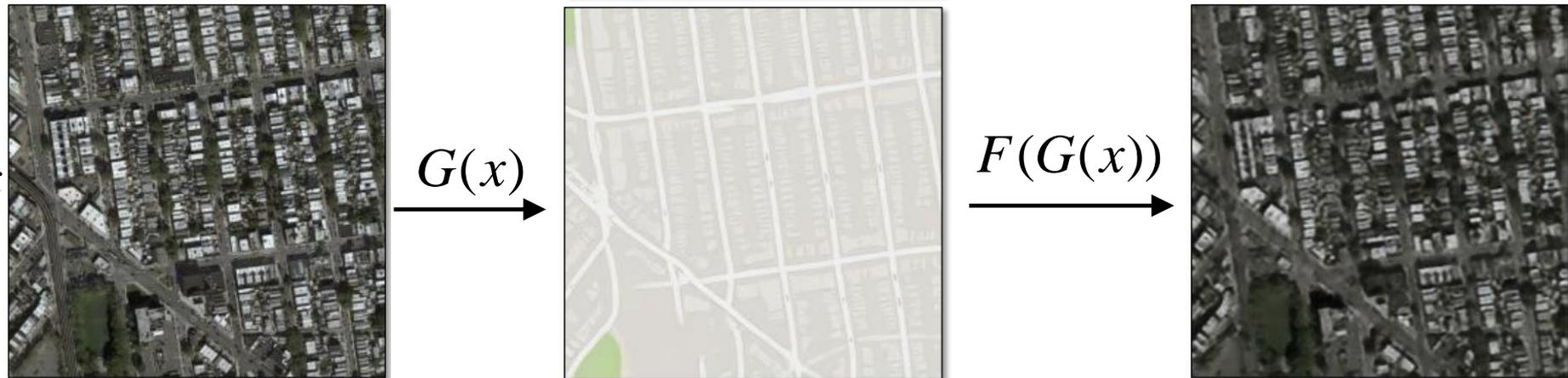


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X . D_Y encourages G to translate X into outputs indistinguishable from domain Y , and vice versa for D_X and F . To further regularize the mappings,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



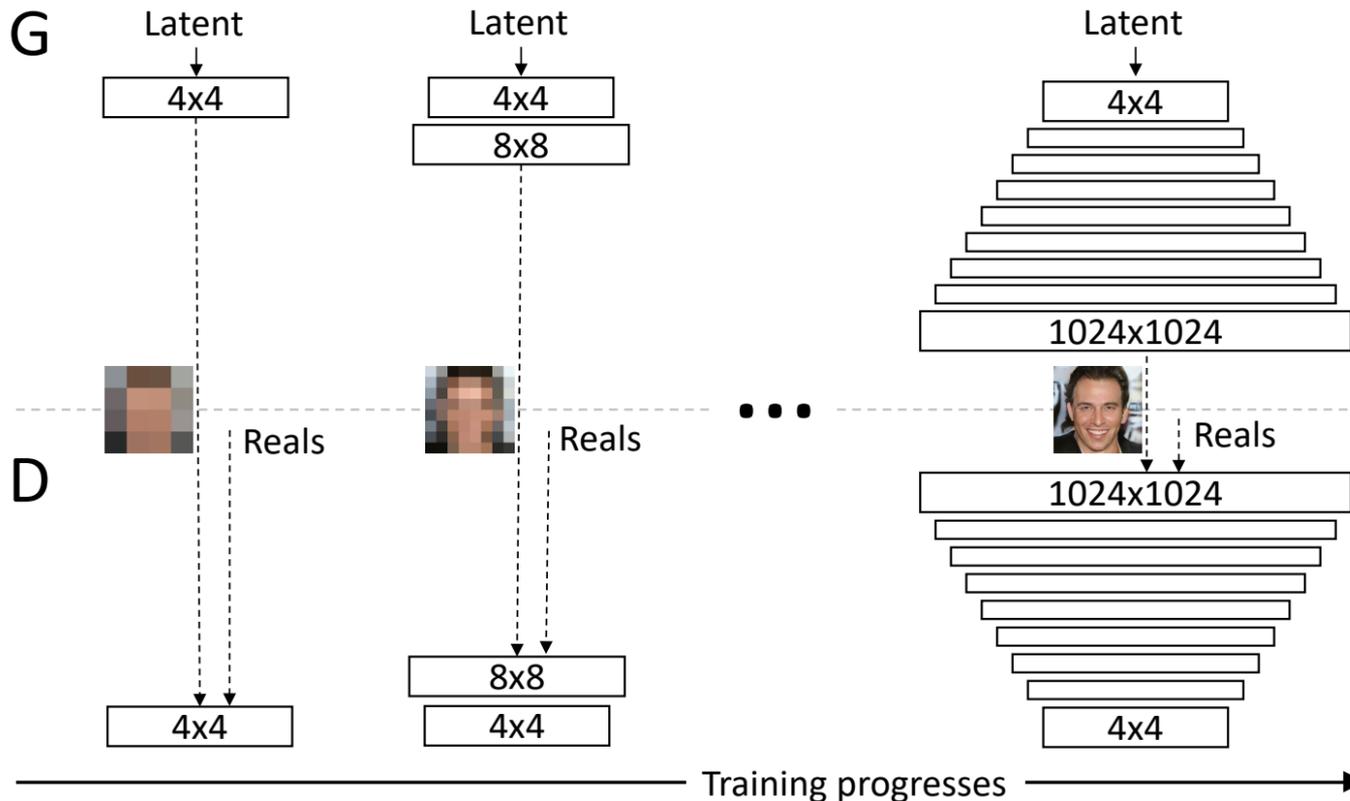
Progressive Growth of GAN

- Recherche à
 - stabiliser l'entraînement des GAN
 - augmenter la qualité
 - la taille des images de sorties : 1024x1024
- Idées maîtresses :
 - faire croître graduellement le GAN
 - ajouter heuristique pour encourager la diversité des images générées (mode collapse)



Croissance progressive

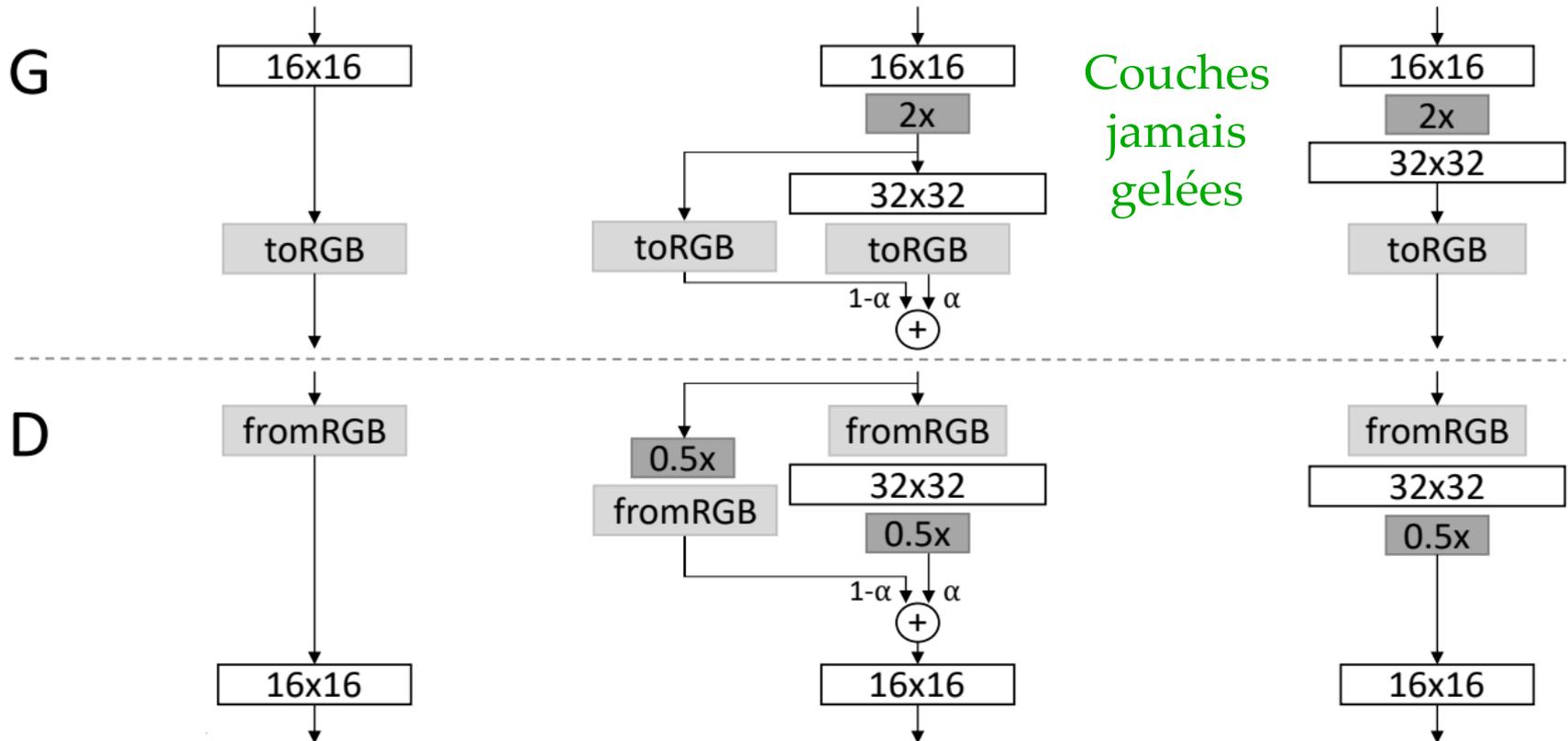
- Permet au réseau de découvrir les structures à grande échelle, puis de raffiner vers le détail
- Plus rapide, car entraînement majoritairement sur des images plus petites (gain 2x-6x)



Note : détails fins sont généralement problématiques pour GAN

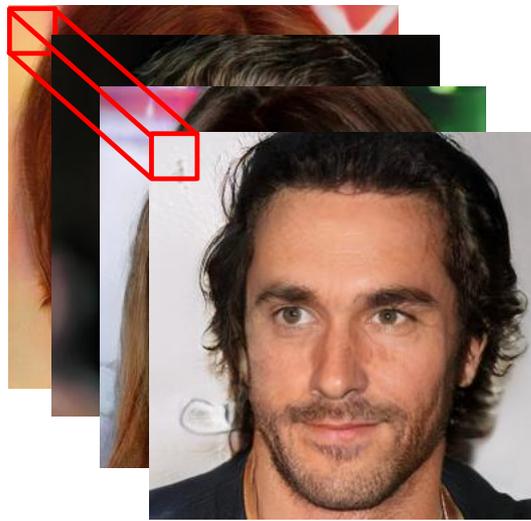
Croissance progressive

- Fondu progressif avec α ($0 \rightarrow 1$) à l'ajout d'une couche
 - éviter les chocs/instabilités au moment de l'ajout
 - simplifie la problématique, au lieu d'essayer de trouver directement $z \rightarrow 1024^2$ (*shaping*)



Contre le *mode collapse*

- Ajout d'un heuristique basé sur des écarts-types pour chaque minibatch



1. Pour chaque feature et chaque endroit, calcule $\sigma_f^{(x,y)}$

2. Moyenne globale unique de $\sigma_f^{(x,y)}$

3. Ajoute un **feature map** dans D contenant cette valeur unique, pour toute la batch (ajouté vers sommet de D)

Discriminator	Act.	Output shape	Params
Input image	–	3 × 1024 × 1024	–
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	–	32 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	–	64 × 256 × 256	–
Conv 3 × 3	–	64 × 256 × 256	14k
Conv 3 × 3	–	64 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Downsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Downsample	–	512 × 4 × 4	–
Minibatch stddev	–	513 × 4 × 4	–
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Conv 4 × 4	LReLU	512 × 1 × 1	4.2M
Fully-connected	linear	1 × 1 × 1	513
Total trainable parameters			23.1M