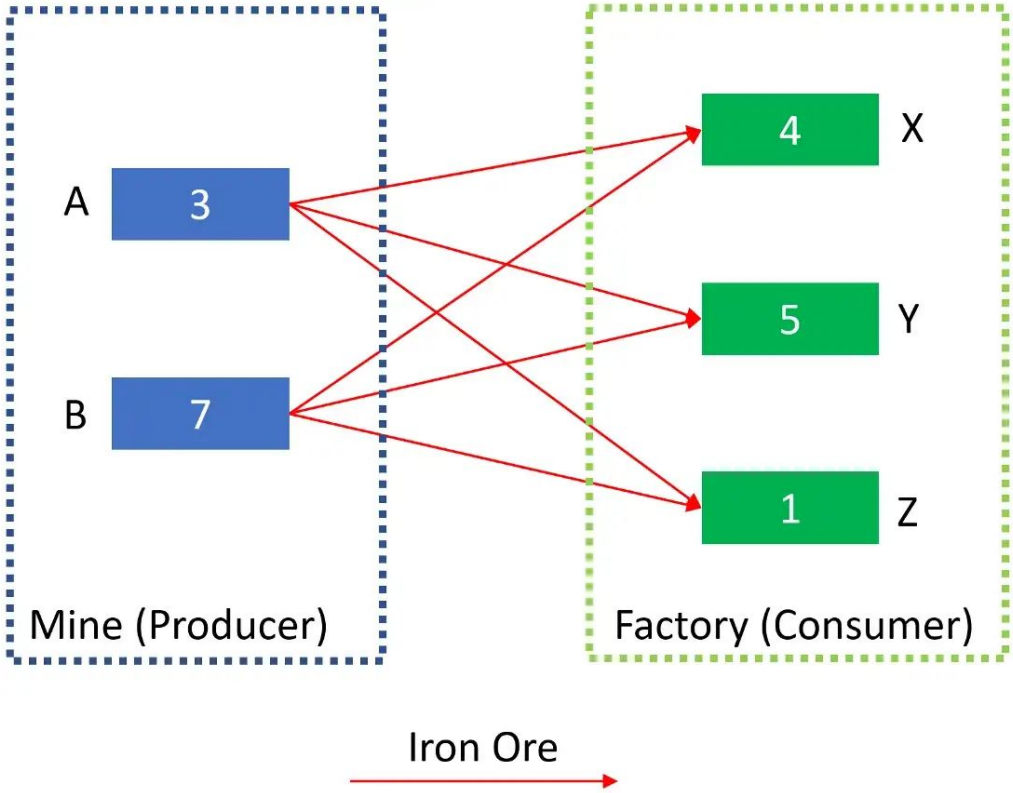
Transfert de couleur par Transport Optimal

Sommaire


1
2
3
4
6
7
Introduction
Algorithme de Sinkhorn
Distance de Wasserstein
Filtre bilatéral
Filtre guidé
Résultats

Introduction - Transport Optimal

Problème de Monge :



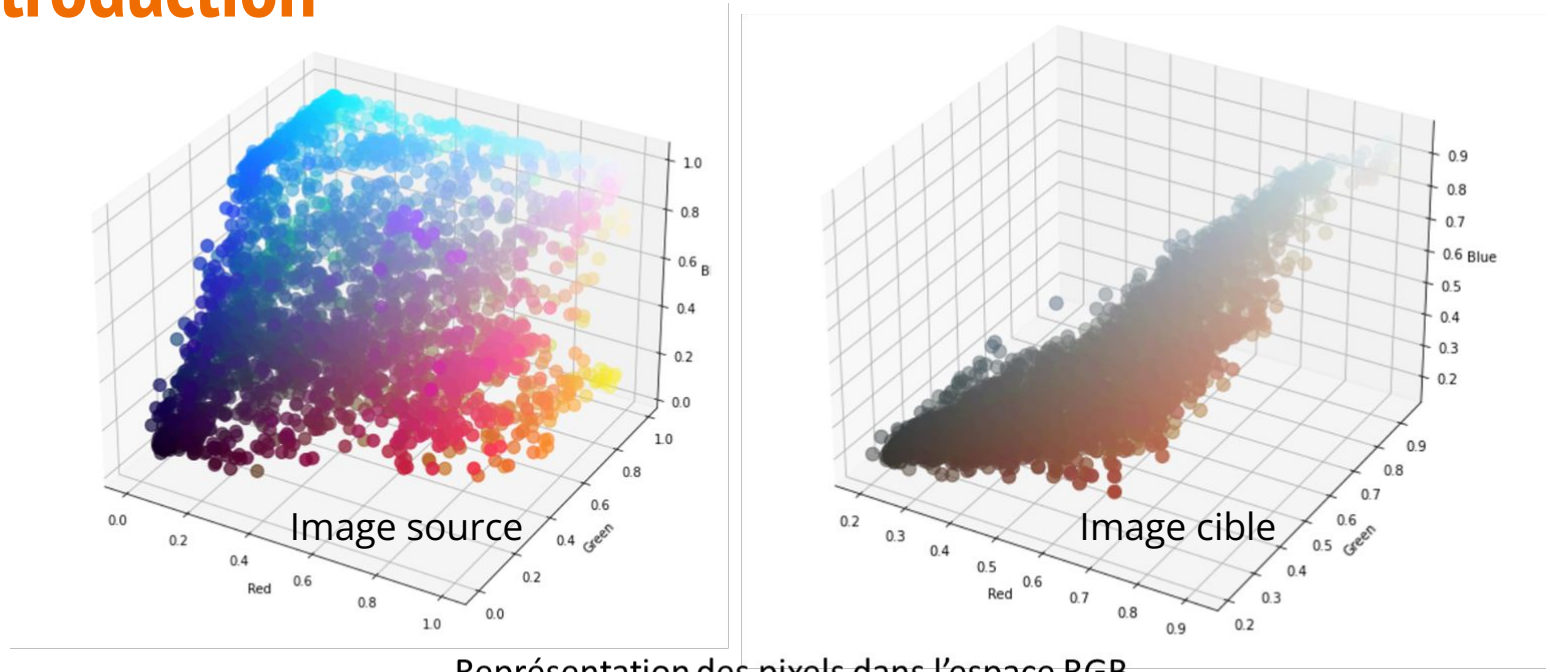
Introduction - Lien avec Transfert de couleur

Image B (source)

Image A (cible)

Objectif : Transférer les pixels de l'image source vers l'image cible

Introduction



Représentation des pixels dans l'espace RGB

Problème : Quelle est la meilleure répartition des pixels ?

Algorithme de Sinkhorn

Coût du transfert du pixel x vers y : $c(x,y) = |\, x - y \,|^2 = (x_{rouge} - y_{rouge})^2 + (x_{vert} - y_{vert})^2 + (x_{bleu} - y_{bleu})^2$

$$argmin_{P \in U(\alpha,\beta)} \sum_{i,j} C_{i,j} P_{i,j} - \varepsilon H(P)$$

On souhaite trouver la meilleure répartition possible des pixels tout en minimisant le coût

- U ensemble des combinaisons possibles
- α et β les distributions des images cibles et sources
- $H(P)$ entropie de la matrice P
- ε paramètre de régularisation
- $C_{i,j}$ matrice des coûts

Algorithme de Sinkhorn :
(algo de projection alternée)

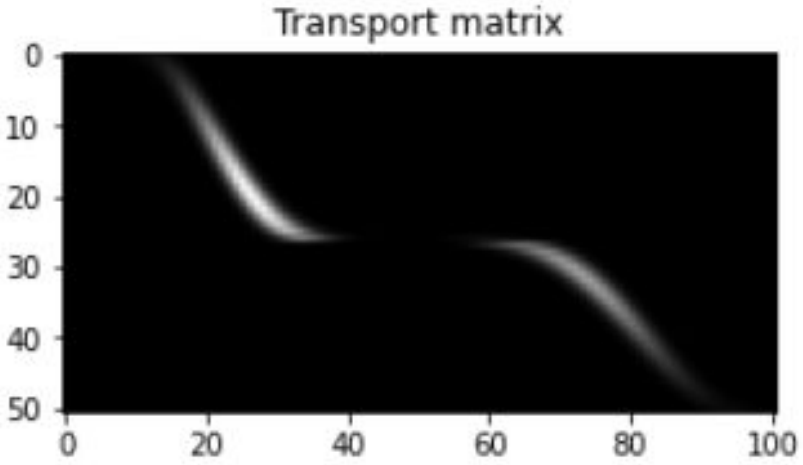
$$\boldsymbol{P^* = diag(u^*) K diag(v^*)}$$

Où : - $K = e^{-\frac{1}{\varepsilon C}}$

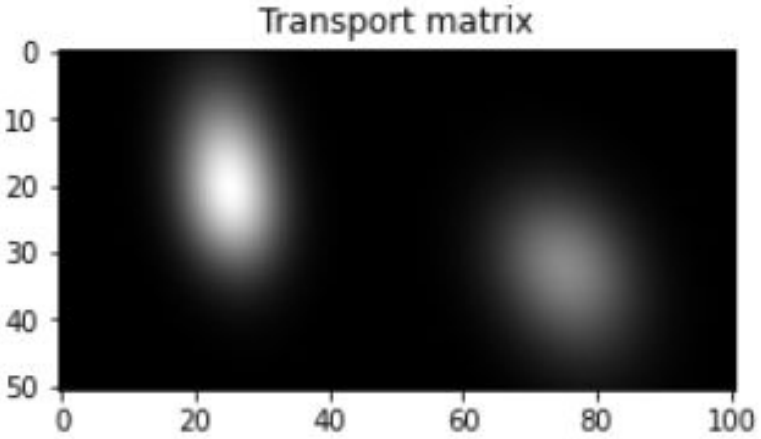
- $u_{k+1} = \frac{a}{K v_k}$ et $v_{k+1} = \frac{b}{K^T v_k}$

Algorithme de Sinkhorn

Influence du paramètre epsilon



epsilon = 0.001



epsilon = 0.1

Algorithme de Sinkhorn

Coût du transfert du pixel x vers y : $c(x,y) = |x - y|^2 = (x_{rouge} - y_{rouge})^2 + (x_{vert} - y_{vert})^2 + (x_{bleu} - y_{bleu})^2$

$$argmin_{P \in U(\alpha,\beta)} \sum_{i,j} C_{i,j} P_{i,j} - \varepsilon H(P)$$

On souhaite trouver la meilleure répartition des pixels tout en minimisant le coût

- U ensemble des combinaisons possibles
- α et β les distributions des images cibles et sources
- $H(P)$ entropie de la matrice P
- ε terme de régularisation
- $C_{i,j}$ matrice des coûts

Algorithme de Sinkhorn :
(algo de projection alternée)

$$\boldsymbol{P^* = diag(u^*) K diag(v^*)}$$

Où : - K = $e^{-\frac{1}{\varepsilon C}}$

- $u_{k+1} = \frac{a}{K v_k}$ et $v_{k+1} = \frac{b}{K^T v_k}$

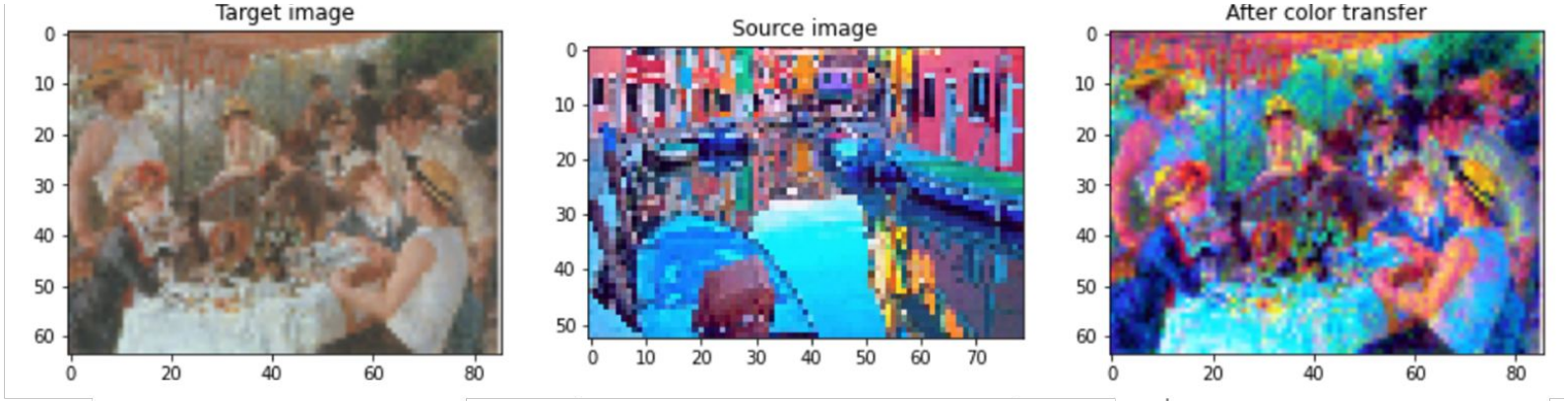
Algorithme de Sinkhorn

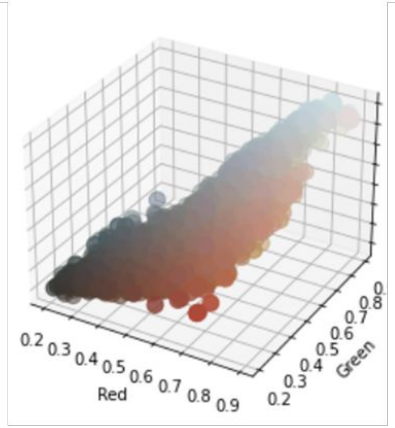
L'algorithme de Sinkhorn nécessite le calcul de la matrice des coûts C

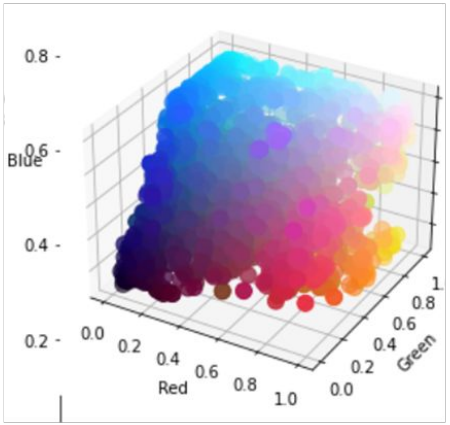
Images 256 x 256 → 4,29 x 10^9 éléments à calculer

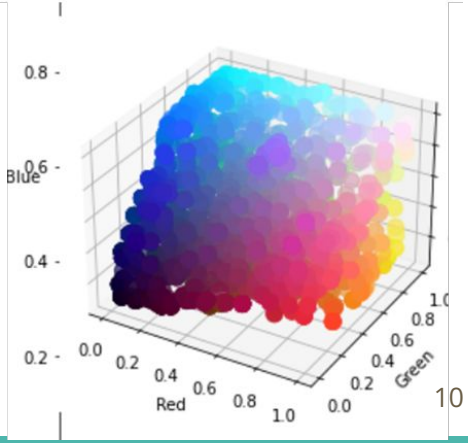
→ **On rassemble les pixels par blocs**

Algorithme de Sinkhorn

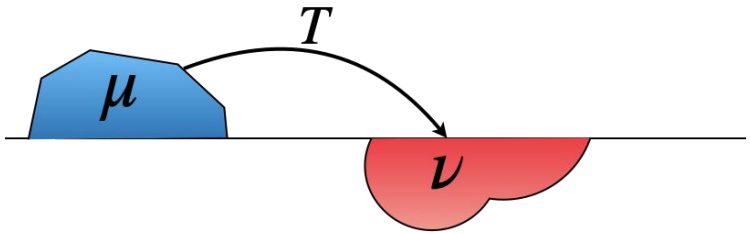








La distance de Wasserstein



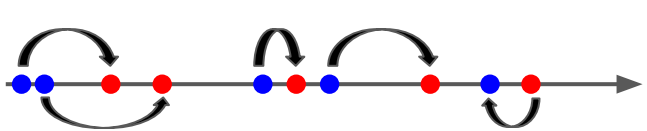
Vecteur $X, Y \in \mathbb{R}^{N,d}$, N = #pixels, d = #colors

Trouver des permutations σ^* de sorte que: $\sigma^\star \in \underset{\sigma \in \Sigma_N}{\operatorname{argmin}} \sum_i \|X_i - Y_{\sigma(i)}\|^p$

⟹ La distance quadratique de Wasserstein:

$$W_\sigma(\mu_X, \mu_Y) = \sum_{i=0}^{N-1} \|X_i - Y_{\sigma(i)}\|^2$$

Solution explicite pour la distribution 1D



X_i
Y_i

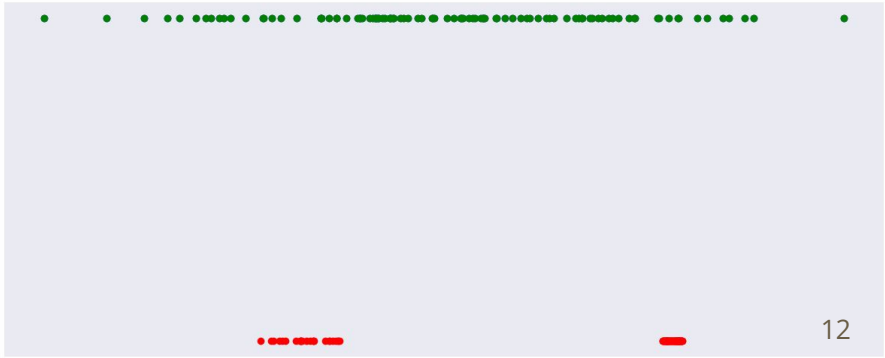
Trier les valeurs: $X_{\sigma_X(1)} \leq X_{\sigma_X(2)} \leq \cdots \leq X_{\sigma_X(N)},$

$Y_{\sigma_Y(1)} \leq Y_{\sigma_Y(2)} \leq \cdots \leq Y_{\sigma_Y(N)},$

$\Rightarrow O(N \log(N))$

La permutation optimale qui minimise la distance de Wasserstein

$$\sigma^* = \sigma_Y \circ \sigma_X^{-1}$$



! Pas adapté pour les images **couleurs**

Approximation: Sliced Wasserstein Distance

La distance de Wasserstein en dimension **supérieure que 1** est **trop exigeant en calcul**

⟹ Approchée par **Sliced Wasserstein Distance**

$$SW(X,Y)^2 = \int_{\theta\in\Omega} W(X_\theta, Y_\theta)^2 d\theta \qquad \text{où} \qquad X_\theta = \{\langle X_i, \theta\rangle\}_{i=0,\dots,N-1}, Y_\theta = \{\langle Y_i, \theta\rangle\}_{i=0,\dots,N-1}$$

$$= \int_\Omega \left(\sum_{i=0}^{N-1} |\langle X_{\sigma_{X,\theta}(i)} - Y_{\sigma_{Y,\theta}(i)}\rangle|^2 \right) d\theta$$

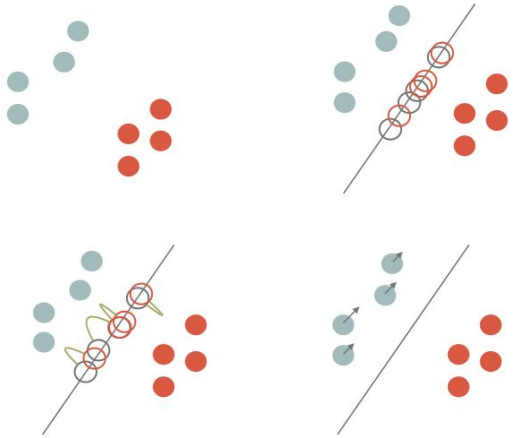
Agréger la distance de Wasserstein **en 1D** des **nuages de points projetés.**

Approximation: Descente de gradient stochastique

$E_Y(X) \overset{\text{def.}}{=} SW(X,Y)$ est lisse

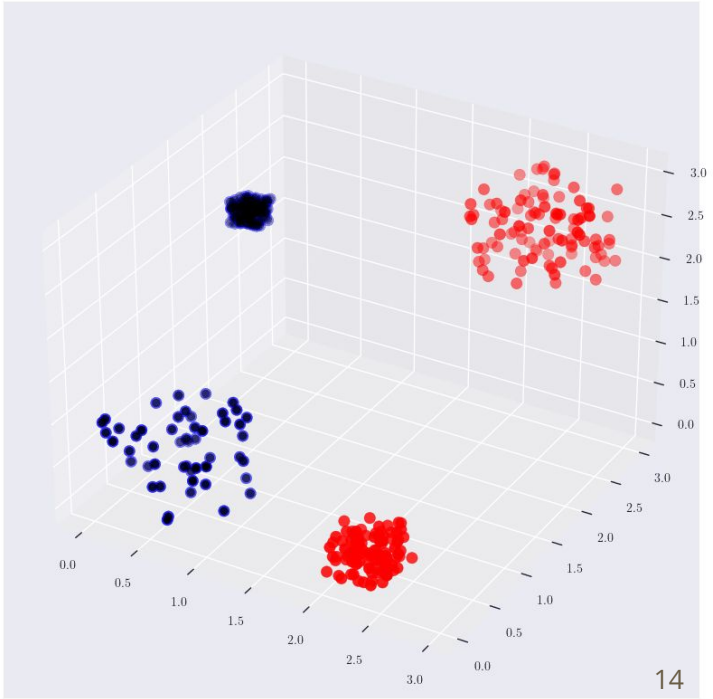
$$\rightarrow \quad \nabla E_Y(X)_i = 2 \int_\Omega \left(\langle X_i - Y_{\sigma_{Y,\theta} \circ \sigma^{-1}_{X,\theta}(i)}, \theta \rangle \theta \right) d\theta$$

Pour un pas suffisamment petit τ, l'itération de descente de gradient stochastique s'écrit :



$$X \leftarrow X - \tau \nabla E_Y(X)$$

A la convergence, X et Y ont la même distribution.



Approximation: Descente de gradient stochastique

Calculer le gradient par Monte Carlo à chaque itération est trop cher en calcul

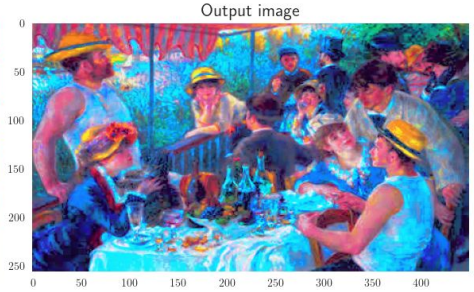
$$\nabla E_Y(X)_i = 2 \int_{\Omega} \left(\langle X_i - Y_{\sigma_{Y,\theta} \circ \sigma^{-1}_{X,\theta}(i)}, \theta \rangle \theta \right) d\theta$$

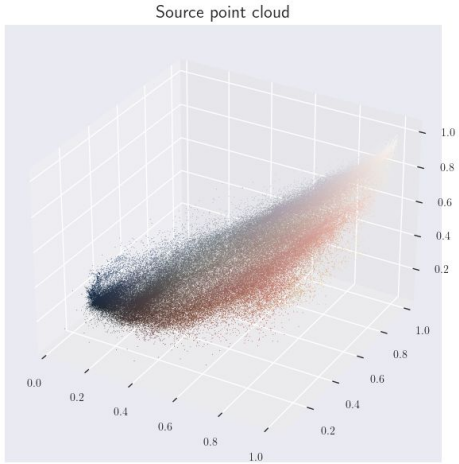
τ

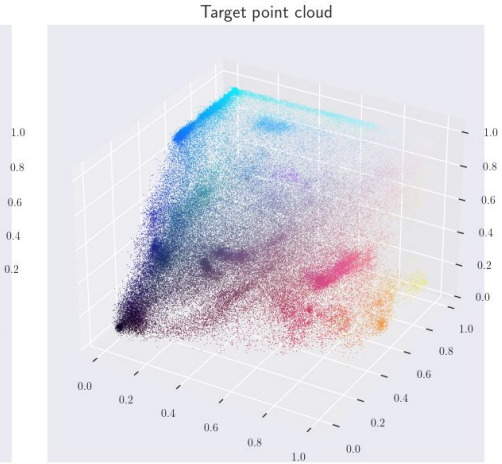
⟹ Décomposer l'estimateur Monte Carlo du gradient en plusieurs itérations

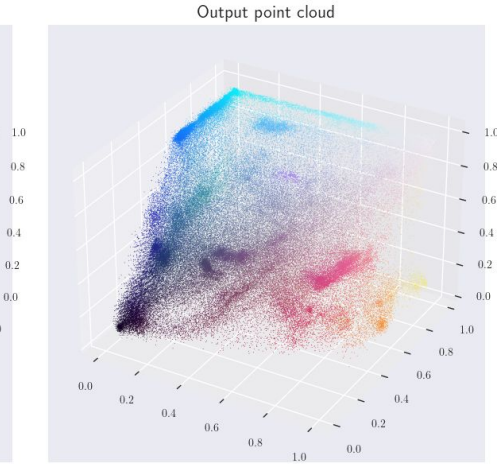
$$X \leftarrow X - \tau \left(\langle X - Y_{\sigma_{Y,\theta} \circ \sigma^{-1}_{X,\theta}}, \theta \rangle \theta \right)$$

Résultats avec la méthode de Wasserstein

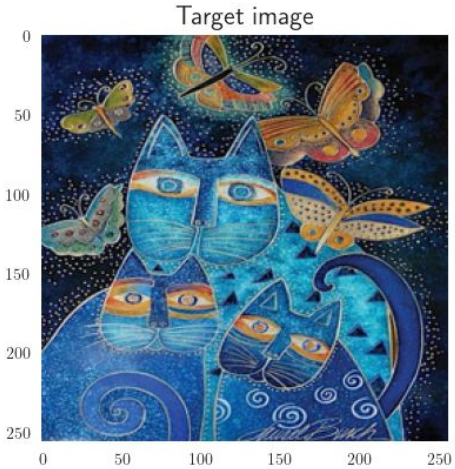


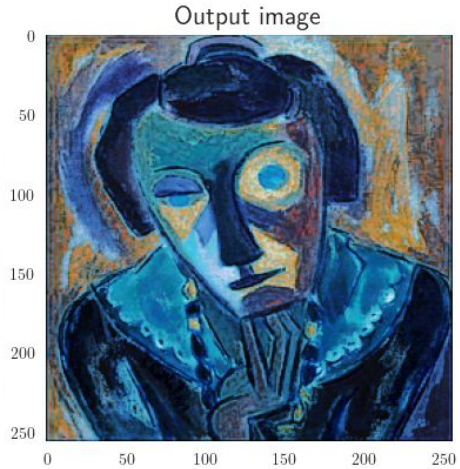






Résultats avec la méthode de Wasserstein





Amélioration et lissage par filtre bilatéral

Filtre appliqué

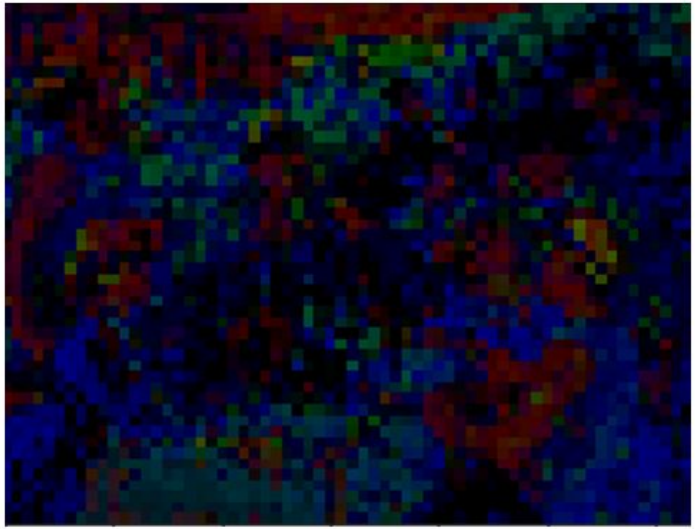


Image sur laquelle est appliquée le filtre

Image obtenue après transfert – image source de départ

Pour chaque pixel x_i :

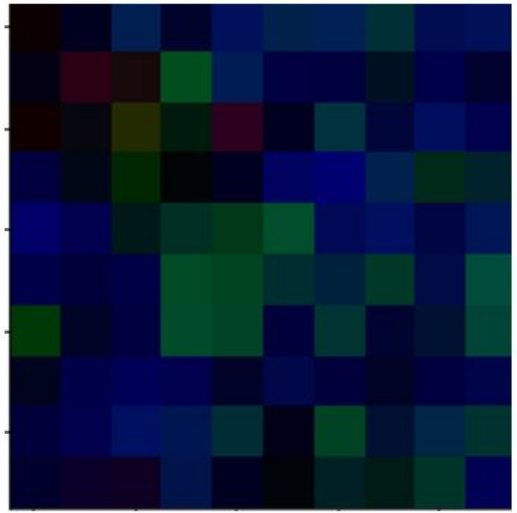
1. Définition d'une **fenêtre** de voisinage
2. Calcul de la **distance RGB** $\boldsymbol{d_i}$ entre pixel x_i et autres pixels de la fenêtre
3. Calcul des **poids** de chaque pixel en fonction d'un paramètre σ:

$$\boldsymbol{w = \exp(\sigma d_i)}$$

Paramètrage de $\boldsymbol{\sigma}$ – plus σ est grand, plus on donne de l'importance aux couleurs proches de x_i

Amélioration et lissage par filtre bilatéral

Filtre appliqué



Exemple de fenêtre

Image obtenue après transfert – image source de départ

Pour chaque pixel x_i :

4. Calcul de la **moyenne pondérée** de ces poids sur chaque fenêtre et attribution de la valeur du pixel

$$x_i = \frac{1}{n}\sum_{i}^{m} w_i f_i$$

Où, f_i: pixel i de la fenêtre

m: nombre de pixels dans la fenêtre,

w: poids,

$n = \sum_i^m w_i$: somme des poids.

Amélioration et lissage par filtre bilatéral

Résultat final

Avant lissage

Après lissage

Non adapté pour les images de grandes tailles

Des solutions adaptées aux images de grandes dimensions

Version de Sinkhorn utilisant des convolutions

- On applique successivement des convolutions

- Pas de besoin de créer des blocs de pixels

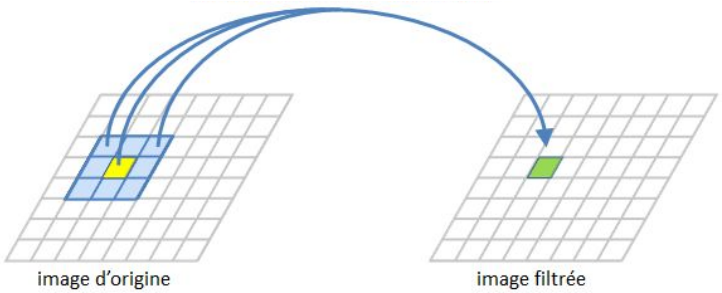
- Ne nécessite pas le calcul de la matrice C

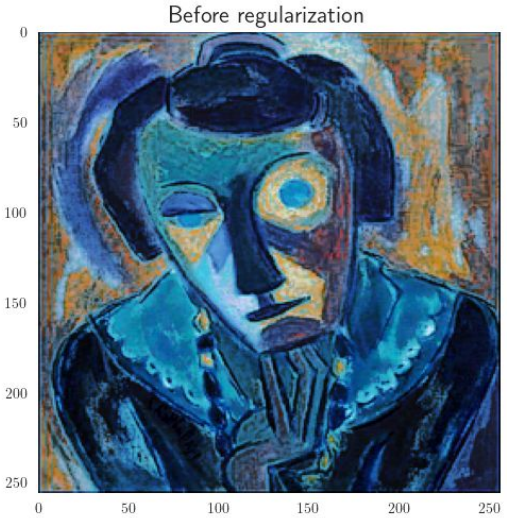
- Plus rapide
- Méthode adaptée aux grandes images

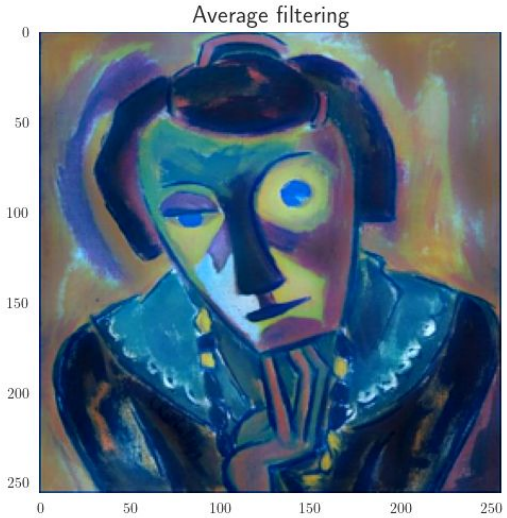
Lissage par filtre moyen

$$H : Z \mapsto h \star Z$$

Calculer la moyenne







Lissage par filtre guidé

p = Sortie de l'algo - Source
I = Source

L'image d'entrée p, l'image de guidage I, rayon r, régularisation ϵ

Image de sortie filtrée q

$$q_i = a_k I_i + b_k, \forall i \in \omega_k$$

fenêtre ω_k centrée sur le pixel k

(a_k, b_k) sont des coefficients linéaires supposés constants dans ω_k

$$a_k, b_k = \text{argmin}_{\omega_k} E(a_k, b_k) := \sum_{i \in \omega_k} ((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2$$

$$\begin{aligned}
&\text{mean}_I = f_{\text{mean}}(I) \\
&\text{mean}_p = f_{\text{mean}}(p) \\
&\text{corr}_I = f_{\text{mean}}(I \odot I) \\
&\text{corr}_{Ip} = f_{\text{mean}}(I \odot p) \\
&\text{var}_I = \text{corr}_I - \text{mean}_I \odot \text{mean}_I \\
&\text{cov}_{Ip} = \text{corr}_{Ip} - \text{mean}_I \odot \text{mean}_p \\
&a = \text{cov}_{Ip} \oslash (\text{var}_I + \epsilon) \\
&b = \text{mean}_p - a \odot \text{mean}_I \\
&\text{mean}_a = f_{\text{mean}}(a) \\
&\text{mean}_b = f_{\text{mean}}(b) \\
&\textbf{return} \quad q = \text{mean}_a \odot I + \text{mean}_b
\end{aligned}$$

Algorithme du filtre guidé

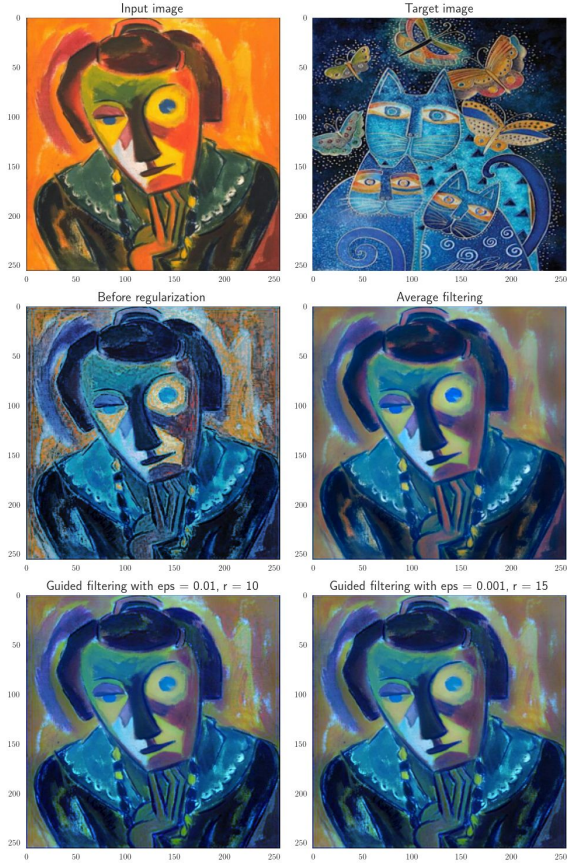
Des solutions adaptées aux images de grandes dimensions

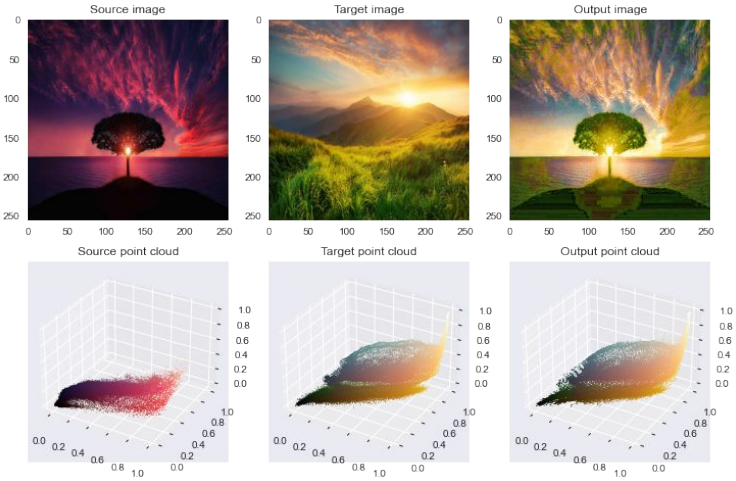
Lissage de l'image par **filtre guidé**

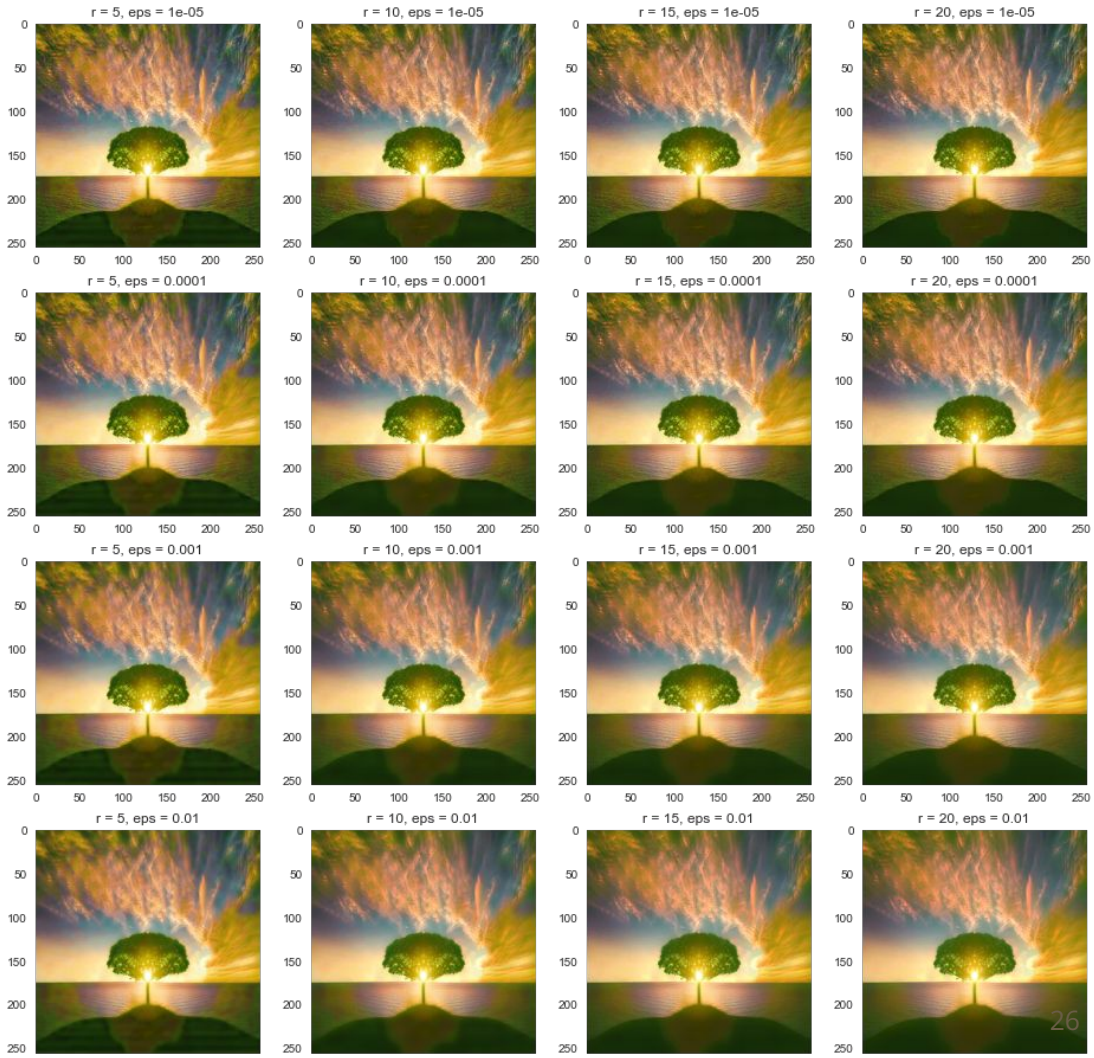
- Basé sur des combinaisons linéaires
- Adaptée aux images de grandes dimensions

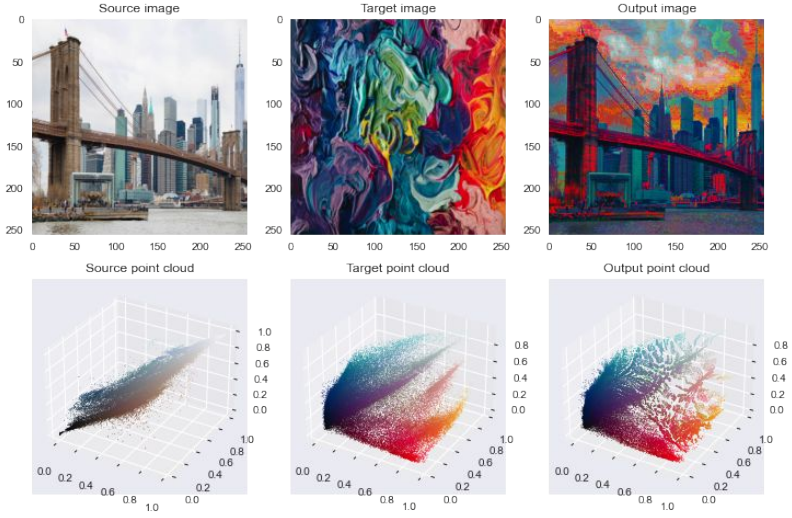
Comparaison avec l'image obtenue avec Sinkhorn traditionnel et un filtre bilatéral

Résultats avec la méthode de Wasserstein et filtre guidé

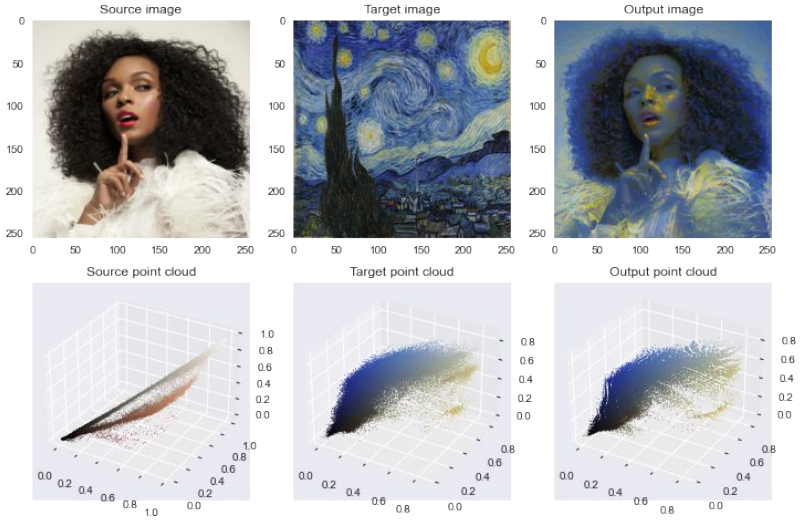



Source image
Target image
Output image
Source point cloud
Target point cloud
Output point cloud


r = 5, eps = 1e-05
r = 10, eps = 1e-05
r = 15, eps = 1e-05
r = 20, eps = 1e-05
r = 5, eps = 0.0001
r = 10, eps = 0.0001
r = 15, eps = 0.0001
r = 20, eps = 0.0001
r = 5, eps = 0.001
r = 10, eps = 0.001
r = 15, eps = 0.001
r = 20, eps = 0.001
r = 5, eps = 0.01
r = 10, eps = 0.01
r = 15, eps = 0.01
r = 20, eps = 0.01


Source image
Target image
Output image
Source point cloud
Target point cloud
Output point cloud


r = 5, eps = 1e-05
r = 10, eps = 1e-05
r = 15, eps = 1e-05
r = 20, eps = 1e-05
r = 5, eps = 0.0001
r = 10, eps = 0.0001
r = 15, eps = 0.0001
r = 20, eps = 0.0001
r = 5, eps = 0.001
r = 10, eps = 0.001
r = 15, eps = 0.001
r = 20, eps = 0.001
r = 5, eps = 0.01
r = 10, eps = 0.01
r = 15, eps = 0.01
r = 20, eps = 0.01


Source image
Target image
Output image
Source point cloud
Target point cloud
Output point cloud


r = 5, eps = 1e-05
r = 10, eps = 1e-05
r = 15, eps = 1e-05
r = 20, eps = 1e-05
r = 5, eps = 0.0001
r = 10, eps = 0.0001
r = 15, eps = 0.0001
r = 20, eps = 0.0001
r = 5, eps = 0.001
r = 10, eps = 0.001
r = 15, eps = 0.001
r = 20, eps = 0.001
r = 5, eps = 0.01
r = 10, eps = 0.01
r = 15, eps = 0.01
r = 20, eps = 0.01