

# Transport Optimal



RÉJANE JOYARD

Décembre 2023

# Table des matières

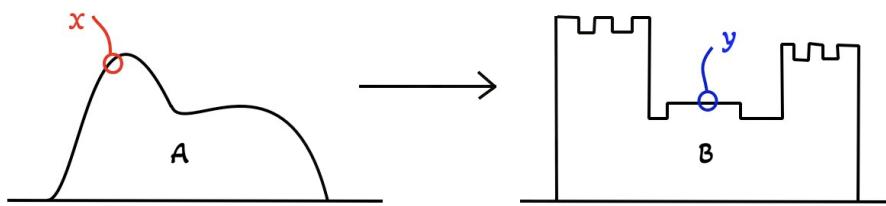
<b>1 Partie 1 : Introduction au transport optimal</b>	<b>1</b>
1.1 Problème de Gaspard Monge (1781) . . . . .	1
1.2 Cadre général pour le problème de Monge . . . . .	1
1.3 Le problème de Monge-Kantorovich . . . . .	2
1.4 Un cadre métrique . . . . .	4
1.5 Cas particulier de Brenier, un coût quadratique dans l'espace euclidien . . . . .	5
1.6 Transport par tranches en dimension $d$ . . . . .	5
<b>2 Partie 2 : TP Tranfert de couleurs</b>	<b>7</b>
2.1 Code Python associé au transfert de couleurs sur des images . . . . .	7
2.2 Optimisation du code . . . . .	10
2.3 Fonctionnalités avancées . . . . .	11
2.3.1 Régularisation . . . . .	11
2.3.2 Interpolation . . . . .	13
2.4 Animation . . . . .	14
2.5 Application à d'autres images . . . . .	15
2.6 Pour aller plus loin . . . . .	16
<b>Références</b>	<b>17</b>

# 1 Partie 1 : Introduction au transport optimal

## 1.1 Problème de Gaspard Monge (1781)

À la fin du XVIII<sup>e</sup> siècle, l'empire napoléonien s'étend rapidement et la construction de forteresses devient la clé de la domination géographique. Gaspard Monge, un mathématicien proche de Napoléon, avait remis quinze ans plus tôt un mémoire sur la théorie de *déblais* et *remblais*. Ce concept se manifestait lors du déplacement de sable d'une dune vers un autre lieu : le volume de sable à transporter était appelé *déblais*, tandis que l'espace qu'il devait occuper après le transport était appelé *remblais*.

Si  $x$  est un lieu dans le *déblais*, et  $y$  est un lieu dans le *remblais*, alors  $T(x, y)$  est le transport de la quantité de sable de  $x$  vers  $y$ . On note  $c(x, y)$  le coût du transport qui est proportionnel à la masse de sable transportée et à la distance parcourue.



Le mathématicien Monge conjecture l'existence d'une solution qui minimise le coût total du transport de ce sable : cette solution dite optimale serait plus économique que les autres solutions. C'est ainsi qu'est née la théorie du transport optimal.

Pour modéliser ce phénomène nous attribuons à la distribution de sable sur la dune  $A$ , la mesure  $\mu$  et à la distribution de sa répartition  $B$ , la mesure  $\nu$ . Le transport optimal correspond à la minimisation du problème avec un coût de transport pouvant être différent de la simple distance. Le problème revient alors à minimisé le critère :

$$\int_{\Omega} c(x, T(x)) d\mu$$

où  $T$  est l'application de transport  $T_*(\mu) = \nu$ .

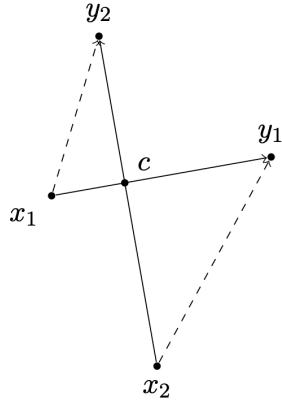
## 1.2 Cadre général pour le problème de Monge

Nous nous plaçons dans un espace métrique  $(E, d)$  séparable et complet et on se donne une fonction de coût :

$$c : E \times E \longrightarrow [0, +\infty[$$

qui définit le coût de transport du point  $x$  au point  $y$ .

Nous noterons  $\mathcal{P}(E)$  l'ensemble des mesures de probabilités boréliennes sur  $E$ .



On a ici une figure qui illustre un raisonnement heuristique. On observe que lorsqu'on veut envoyer  $(x_1, x_2)$  sur  $(y_1, y_2)$  il est plus plausible et économique d'emprunter le chemin en pointillé que de croiser les trajectoires.

Tout cela nous amène à posé la définition suivante :

**Définition 1.2.1** Soit  $\mu$  et  $\nu$  appartenant à  $\mathcal{P}(E)$ . Pour toute application de transport  $T$  de  $\mu$  à  $\nu$ , on définit :

$$C(T) = \int_{\Omega} c(x, T(x))d\mu$$

### 1.3 Le problème de Monge-Kantorovich

Dans les années 1940, le mathématicien et économiste russe Leonid Kantorovich reprend la théorie du transport optimal en voulant élargir le problème de Monge.

Le problème de transport optimal selon Monge peut effectivement être mal posé. Par exemple, dans le cas où  $\mu$  est une mesure de Dirac et que  $\nu$  n'en est pas une, il n'y a pas d'application  $T$  qui vérifie  $T_*(\mu) = \nu$ . En d'autres termes, il y a un problème lorsque la nature des mesures de départ et d'arrivée ne sont pas compatibles pour permettre un transport optimal. Cela peut rendre la recherche d'une solution difficile voire impossible.

**Exemple :** (nature des mesures différentes)

On se place dans  $\mathbb{R}^2$  muni de la fonction de coût  $c_p(x, y) = |x - y|^p$  avec  $x, y$  dans  $\mathbb{R}^2$ .

On considère les mesures suivante :

$$\mu = \delta_0 \otimes m \quad \text{et} \quad \nu = \frac{1}{2}\delta_{-1} \otimes m + \frac{1}{2}\delta_1 \otimes m$$

où  $m$  est la probabilité uniforme sur  $[0, 1]$ . Donc si  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  est une fonction borélienne bornée, nous avons :

$$\begin{aligned} \int f(x)\mu(dx) &= \int f(0, x_2)m(dx_2) \\ \text{et} \\ \int f(y)\nu(dy) &= \frac{1}{2} \int f(-1, y_2)m(dy_2) + \frac{1}{2} \int f(1, y_2)m(dy_2) \end{aligned}$$

D'un point de vue probabiliste,  $\mu$  est la loi de  $(0, X)$  où  $X$  suit la loi uniforme sur  $[0, 1]$ , et  $\nu$  est la loi de  $(\epsilon, X)$  où  $\epsilon$  est une variable aléatoire indépendante de  $X$ , de loi de Rademacher.

Montrons que  $C_p(\mu, \nu) = 1$  mais qu'aucune application  $T$  ne réalise l'infinimum. Dans un premier temps, si  $T$  est une application de transport de  $\mu$  sur  $\nu$  alors nous avons pour tout  $x_2 \in [0, 1]$ / D'où  $T(0, x_2) \in \{-1\} \times [0, 1] \cup \{1\} \times [0, 1]$  et donc  $|(0, x_2) - T(0, x_2)|^p \geq 1$ . On conclut que  $C_p(T) \geq 1$  et que  $C_p(\mu, \nu) \geq 1$ .

Montrons maintenant que  $C_p(\mu, \nu) \leq 1$ . Nous allons construire une suite  $T_n$  transportant  $\mu$  sur  $\nu$  telle que  $C_p(T_n) \leq (1 + \frac{1}{2n})^p$ ,  $n \in \mathbb{N}^*$ .

Pour la suite de cet exemple, nous noterons  $A = \{0\} \times [0, 1]$ ,  $B = \{-1\} \times [0, 1]$  et  $C = \{1\} \times [0, 1]$ . Divisons le segment  $A$  en sous-segments verticaux  $I_1, \dots, I_{2n}$  de longueur  $\frac{1}{2n}$ . De manière horizontale, nous allons segmenter  $B$  et  $C$  et les noter respectivement  $J_1, \dots, J_n$  et  $K_1, \dots, K_n$  avec une longueur de  $\frac{1}{n}$ .

Ainsi, pour chaque  $p = 1, \dots, n$ , l'application  $T_n$  envoie les points  $(0, x_2)$  appartenant à  $I_{2p}$  vers  $(-1, f_{2p}(x_2))$  où  $f_{2p}$  est l'unique application affine croissante envoyant  $I_{2p}$  sur  $J_p$ .

De même, pour chaque  $p = 1, \dots, n$ , l'application  $T_n$  envoie les points  $(0, x_2)$  appartenant à  $I_{2p+1}$  vers  $(1, f_{2p+1}(x_2))$  où  $f_{2p+1}$  est l'unique application affine croissante envoyant  $I_{2p+1}$  sur  $K_p$ . Puis, vérifions que  $T_n$  transporte la mesure  $\mu$  sur la mesure  $\nu$ . Cela implique que la masse totale dans chaque sous-segment de départ est transportée de manière cohérente vers les sous-segments d'arrivée.

On remarque que  $J_p = K_p = I_{2p} \cup I_{2p+1}$ , on a donc toujours  $|x_2 - f_p(x_2)| \leq \frac{1}{2n}$  pour tout  $x_2 \in I_p$ . D'où  $|(0, x_2) - T_n(0, x_2)| = \sqrt{1 + |x_2 - f_p(x_2)|^2} \leq 1 + \frac{1}{2n}$ . Par suite,  $C_p(T_n) \leq (1 + \frac{1}{2n})^p$  ce qui prouve que  $C_p(\mu, \nu) \leq 1$ .

Montrons enfin par l'absurde qu'aucune application  $T^*$  existe.

On suppose que  $T^*$  est une application de transport optimale. La fonction  $x_2 \mapsto |(0, x_2) - T(0, x_2)|$  est minorée par 1, en revanche par optimalité nous avons :

$$\int_0^1 |(0, x_2) - T(0, x_2)|^p = 1,$$

elle vaut donc 1 Lebesgue presque sûrement.

Par conséquent, il existe une fonction mesurable  $u : [0, 1] \longleftarrow \{-1, 1\}$  telle que  $T(0, x_2) = (u(x_2), x_2)$  pour presque tout  $x_2$ . On notera  $I_{\pm}$  les ensembles où  $u$  prend la valeur  $\pm 1$ . Alors,

$$\int 1_{\{-1\} \times I_+} = \frac{1}{2}m(I_+) \neq 0 = \int 1_{\{-1\} \times I_+} dT_{\#}\mu$$

Ainsi, il y a contradiction.

■

Leonid Kantorovich généralise et améliore le problème : on peut trouver une mesure  $\gamma$  sur  $X \times Y$  qui atteint l'infinimum suivant :

$$\inf \left\{ \int_{X \times Y} c(x, y) \mid \gamma \in \Gamma(\mu, \nu) \right\}$$

où  $\Gamma(\mu, \nu)$  est l'ensemble des mesures définies sur  $X \times Y$ .

Détaillons un peu plus l'origine de cette expression.

La quantité de sable déplacé du point  $x$  doit être égal à la quantité de sable qui était présente au départ en ce point, mathématiquement on traduit cela par l'expression suivante :

$$\mu(x) = \int_Y \gamma(x, y) dy$$

La quantité de sable amenée au point  $y$  doit être égal à la quantité finale souhaitée de sable en ce point :

$$\nu(y) = \int_X \gamma(x, y) dx$$

Ainsi, le coût total d'un plan de transport  $\gamma$  est :

$$\int_Y \int_X c(x, y) \gamma(x, y) dx dy = \int_{X \times Y} c(x, y) d\gamma(x, y)$$

Dans notre cas, le but est de trouver un plan de transport optimal qui minimiserait le coût total donné par la formule ci-dessus. Autrement dit on cherche  $C$  telle que :

$$C = \inf_{\gamma \in \Gamma(\mu, \nu)} \int_{X \times Y} c(x, y) d\gamma(x, y)$$

L'ensemble  $\Gamma(\mu, \nu)$  rend plus agréable la recherche de solutions du problème de Monge. Ici, le transfert de la masse est soumis à la seule condition d'avoir pour mesures  $\mu$  et  $\nu$ .

Le plan  $\Gamma(\mu, \nu)$  n'est jamais vide car le produit tensoriel  $\mu \otimes \nu$  des deux mesures convient toujours. Cela correspond intuitivement à ce que chaque unité de masse déplacée est répartie sur tout l'espace d'arrivée équitablement. Nous pourrons également préciser que ce plan est convexe ce qui entraîne la propriété de la stabilité.

## 1.4 Un cadre métrique

Dans cette partie nous parlerons de la construction d'une métrique sur l'espace des mesures de probabilité basée sur le transport optimal. Lorsque les espaces d'arrivée et de départ sont les mêmes (nous les noterons  $X$ ), il est possible de construire une distance entre les mesures  $\mu$  et  $\nu$  à partir du coût de transport optimal associé à condition que nous aillons  $c(x, y) = d(x, y)^p$  où  $d$  est une distance sur  $X$ . Nous définirons uniquement cette distance dans les cas où  $p \in [1; \infty)$ .

Le but d'établir un cadre métrique est de quantifier l'écart entre deux mesures de probabilités selon une fonction de coût donnée. On se demande dans quelle mesure cela permet de définir une distance sur l'espace des probabilités.

**Définition 1.4.1 (La distance de Wasserstein)** Soit  $(X, \delta)$  un espace métrique, mesurable et complet ( $\delta$  désigne la distance associée). Pour tout  $p \in [1; \infty)$ , nous définissons l'ensemble  $\mathcal{P}_p^\delta(X, \mathbb{B})$  des mesures de probabilités sur  $\mu$  sur  $X$  telles que, en prenant  $a \in X$ , nous avons :

$$\int_X \delta(a, x)^p < +\infty$$

Pour tous  $\mu, \nu \in \mathcal{P}_p^\delta(X)$ , définissons comme dans la partie 1.3 l'ensemble  $\Gamma(\mu, \nu)$  des mesures de probabilité de Lebesgue  $\gamma$  sur  $X \times X$  tel que l'on puisse définir la grandeur suivante :

$$W_p^\delta(\mu, \nu) = \inf_{\gamma \in \Gamma(\mu, \nu)} \left( \int_{X \times X} \delta(x, y)^p d\gamma(x, y) \right)^{\frac{1}{p}}$$

$W_p^\delta$  correspond à une distance sur l'espace  $\mathcal{P}_p^\delta(X)$ . Cette distance, basée sur le transport optimal, est appelé **distance de Wasserstein**.

On remarque que l'on retrouve l'expression du problème de Monge-Kantorovich lorsque  $p = 1$ . En effet, dans notre cas la distance choisie est le coût de transport,  $\delta(x, y) = c(x, y)$ , et nous avons alors :

$$W_1^c(\mu, \nu) = \inf_{\gamma \in \Gamma(\mu, \nu)} \left( \int_{X \times X} c(x, y)^1 d\gamma(x, y) \right)^{\frac{1}{1}} = \inf_{\gamma \in \Gamma(\mu, \nu)} \int_{X \times X} c(x, y) d\gamma(x, y)$$

## 1.5 Cas particulier de Brenier, un coût quadratique dans l'espace euclidien

C'est dans les années 1990 que Yann Brenier, un mathématicien français, établit un lien clair entre le problème de Monge et celui de Kantorovich.

En effet, il est possible de résoudre le problème de Monge dans un cas particulier, *i.e.*, le plan de transport optimal du problème de Kantorovich est issu d'une application de transport.

**Théorème 1.5.1 (Brenier, 1991)** *Soit  $\mu, \nu$  deux mesures de probabilités sur  $\mathbb{R}^d$  avec sa norme euclidienne canonique noté  $|.|$ . Si  $\mu$  est absolument continue par rapport à la mesure de Lebesgue et si :*

$$\int |x|^2 \mu(dx) < +\infty \text{ et } \int |y|^2 \nu(dy) < +\infty,$$

*alors il existe une application de transport  $T^*$  de  $\mu$  sur  $\nu$ , telle que  $C_2(T^*) = C_2(\mu, \nu)$ .*

*Cette application  $T^*$  s'écrit comme le gradient d'une fonction convexe :  $T^* = \nabla \Phi$  où  $\Phi : \mathbb{R}^d \rightarrow \mathbb{R} \cup \{+\infty\}$ .*

**Remarque :** Cette application optimale est unique : si  $T$  est une application de transport telle que  $C_2(T) = C_2(\mu, \nu)$ , alors  $T(x) = T^*(x)$  pour  $\mu$ -presque tout  $x$ .

## 1.6 Transport par tranches en dimension $d$

Le calcul des plans de transport optimal est un problème numérique coûteux en dimension supérieure à 1. Pour répondre à cette problématique, nous pouvons utiliser la technique du transport par tranches.

Lorsque l'on travaille avec des dimensions supérieures à 1, le problème du transport devient plus complexe : il implique le déplacement d'une masse dans un espace multidimensionnel. Le transport par tranches est une technique qui consiste à traiter chaque dimension indépendamment les unes des autres.

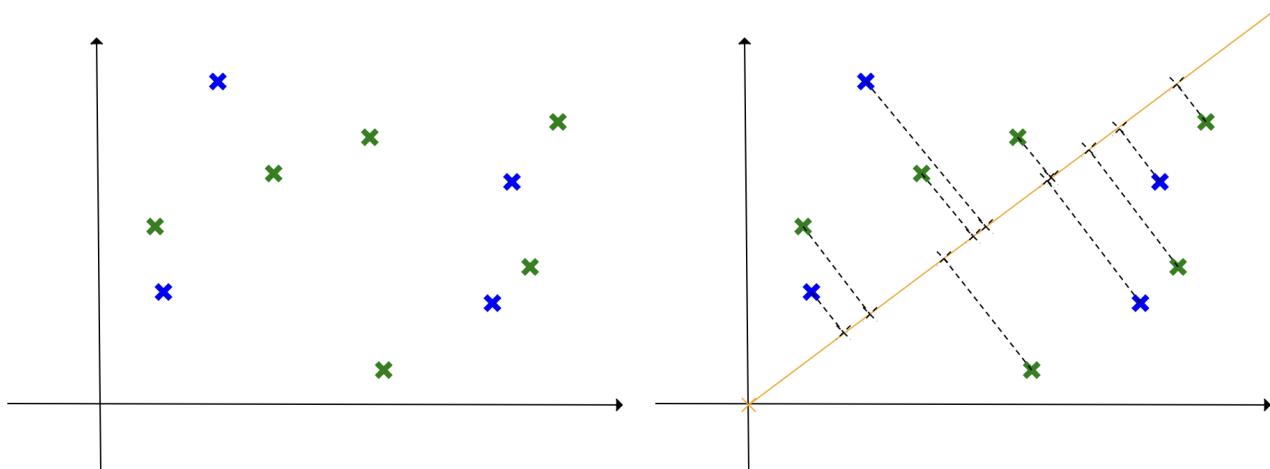
Cette technique demande dans un premier temps de diviser chaque distribution de probabilité en tranche, le long de chaque dimension. Après avoir appliquer le transport optimal sur chaque paire de tranches (dans les distributions  $\mu$  et  $\nu$ ), il faut combiner alors les résultats pour obtenir le transport optimal dans l'espace multidimensionnel complet.

Ceci peut être répétée jusqu'à ce que la solution converge vers une solution optimale dans notre espace à plusieurs dimensions.

**Définition 1.6.1** *Le problème d'appariement équilibré suppose d'écrire les distributions  $\mu$  et  $\nu$  de la façon suivante :*

$$\mu := \frac{1}{n} \sum_{i=1}^n \delta_{x_i} \quad \text{et} \quad \nu := \frac{1}{n} \sum_{i=1}^n \delta_{y_i}$$

Pour la suite, nous noterons  $N$  et  $M$  les quantités de points des distributions respectives  $\mu$  et  $\nu$ . Illustrons les distributions discrètes ainsi que leurs projections sur la droite  $\theta$  pour mieux comprendre.



À gauche nous distinguons la distribution  $\mu$  en vert et la distribution  $\nu$  en bleu et à droite leurs projections par rapport à  $\theta$  (droite orange). On note les projections  $(p_\theta)_\# \mu$  et  $(p_\theta)_\# \nu$  selon la direction  $\theta$ .

On suppose ici que le coût est défini de la manière suivante :  $c(x, y) = \|x - y\|^2$  où  $\|\cdot\|$  est la norme euclidienne dans  $\mathbb{R}^d$  (on peut généraliser en notant  $c(x, y) = \|x - y\|^p$  avec  $p \geq 1$ ).

De ce fait nous pouvons remplacer le problème de minimisation par,

$$SW_2^2(\mu, \nu) := \int_{S^d} W_2^2((p_\theta)_\# \mu, (p_\theta)_\# \nu)) d\theta \text{ où } S^d \text{ est l'hypersphère unitaire en dimension } d.$$

Ici, la projection  $(p_\theta)_\# \mu$  est l'image de la mesure  $\mu$  par  $p_\theta$  et de même la projection  $(p_\theta)_\# \nu$  est l'image de la mesure  $\nu$  par  $p_\theta$ , i.e. :

$$(p_\theta)_\# \mu = \frac{1}{N} \sum_{i=1}^N \delta_{< x_i, \theta > \theta} \quad \text{et} \quad (p_\theta)_\# \nu = \frac{1}{M} \sum_{i=1}^M \delta_{< y_i, \theta > \theta}$$

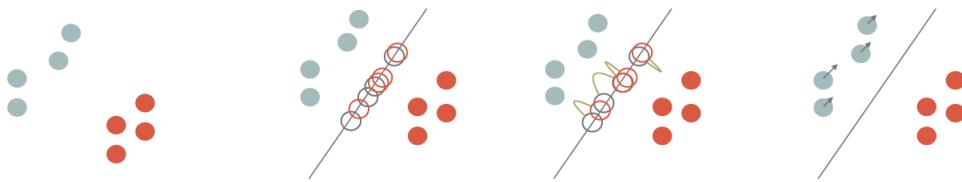
Étant donné que les projections sont faites sur le même axe, la formulation précédente ne fait intervenir le transport optimal qu'en dimension 1. Puis, en ordonnant les points projetés : c'est-à-dire en calculant la différence entre le premier point projeté de  $\mu$  avec le premier point projeté de  $\nu$ , le second avec le second, etc... Nous pouvons trouver deux permutations  $\sigma_\theta$  et  $\kappa_\theta$  optimales et nous pouvons ainsi écrire :

$$SW_2^2(\mu, \nu) = \int_{S^d} (| < x_{\sigma_\theta(i)} - y_{\kappa_\theta(i)}, \theta > |)^2 d\theta$$

Numériquement, nous échantillonons l'ensemble des directions  $S^d$  et considérons donc un nombre fini de tranches.

## 2 Partie 2 : TP Transfert de couleurs

Pour revenir à notre problème de transfert d'histogramme, les centres de Dirac  $\{x_i\}$  sont des points dans l'espace RVB (un Dirac par pixel de l'image d'entrée), et  $\{y_i\}$  sont les couleurs de l'image *target*. La mise en correspondance de l'histogramme peut être considérée comme un transport du nuage de points  $\mu$  vers  $\nu$  dans  $\mathbb{R}^3$  (l'espace colorimétrique RVB). L'idée est d'advecter les points de  $\mu$  de manière à minimiser  $SW(\mu, \nu)$ . Cela revient à projeter les points sur une direction aléatoire  $\theta$ , à aligner les projections triées et à déplacer  $\mu$  dans la direction  $\theta$  par  $|\langle x_{\sigma_\theta(i)} - y_{\kappa_\theta(i)}, \theta \rangle|$ .



### 2.1 Code Python associé au transfert de couleurs sur des images

1. Dans un premier temps, il est nécessaire d'importer les bibliothèques que nous allons utiliser ainsi que les images à traiter.

```

1 # Necessary libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import cv2
5 from tqdm import tqdm
6
7 # Load images
8 image_input = cv2.imread('pexelsA-0.png')
9 image_target = cv2.imread('pexelsB-0.png')
```

2. La fonction *vector\_random* génère un vecteur aléatoire de dimension  $d$  et le normalise afin d'assurer que le vecteur retourné soit un vecteur unitaire.

```

11 # Function allowing to generate a random vector of dimension d
12 def vector_random(d):
13     random_vector = np.random.normal(loc: 0, scale: 1, d)
14     normalization = random_vector / np.linalg.norm(random_vector)
15
16     return normalization
```

3. La fonction *sort* parcourt tous les pixels de l'image, récupère la couleur du pixel à la position  $(i, j)$  et effectue la projection de cette couleur sur la direction définie par le vecteur  $\theta$  (qui dans notre cas sera un vecteur généré par la fonction précédente). La projection est calculée en prenant le produit scalaire entre la couleur du pixel et le vecteur  $\theta$ .

Cette fonction va retourner une liste triée contenant les projections des couleurs des pixels d'une image sur la direction  $\theta$ .

```

18 # Function to sort the projection of the samples of one image onto a random direction
19 def sort(image, theta):
20     h, l, _ = image.shape
21     L = []
22     for i in range(h):
23         for j in range(l):
24             color = image[i,j]
25             projection = np.dot(color, theta)
26             L.append((projection,(i,j))) # Pixel position storage
27     sorting = sorted(L, key=lambda x: x[0])
28     return sorting

```

4. Il s'en suit la fonction principale de notre code qui va implémenter une opération de transport optimal entre nos deux images.

Dans un premier temps, nous trions les pixels de l'image *input* et de l'image *target* à l'aide de la fonction *sort*, puis initions notre image *output* à l'aide d'un tableau de zéros, de la même taille que nos images.

Nous allons parcourir les pixels triés des images *input* et *target* en même temps et pour chaque pixel de l'image *input* nous récupérons sa position. Une fois que nous aurons vérifié si la position du pixel est dans les limites de l'image *input*, nous allons pouvoir appliquer la formule résultante du transport optimal.

Étant donné que j'utilise *OpenCV*, il est nécessaire de réaliser un *np.clip* afin que s'assurer que les valeurs sont dans l'intervalle [0, 255].

Cette fonction retourne l'image *output* contenant les pixels advectés de l'image *input* vers l'image *target* le long de la direction  $\theta$ .

Pour ce travail, j'ai choisi de traiter le transport pixel par pixel, c'est une méthode qui prend un peu plus de temps que d'autres car chaque pixel doit être traiter individuellement. Mais elle peut permettre des résultats plus précis : elle convient dans les cas où il doit y avoir une correspondance fine entre les pixels comme pour la reconstruction d'images.

```

30 # Function allowing optimal transport
31 def transport(image1, image2, theta):
32     sorted_source = sort(image1, theta)
33     sorted_target = sort(image2, theta)
34     advected_samples = np.zeros_like(image1, dtype=np.uint8)
35
36     for i in range(len(sorted_source)):
37         source_pixel = sorted_source[i][1]
38
39         if 0 <= source_pixel[0] < image1.shape[0] and 0 <= source_pixel[1] < image1.shape[1]:
40             difference = sorted_source[i][0] - sorted_target[i][0]
41             advected_sample = image1[source_pixel[0], source_pixel[1]] - difference * theta
42
43             advected_sample = np.clip(advected_sample, a_min: 0, a_max: 255).astype(np.uint8)
44             advected_samples[source_pixel[0], source_pixel[1]] = advected_sample.astype(np.uint8)
45
46     return advected_samples

```

5. Nous réalisons une boucle qui, à chaque itération, réalise le transport optimal.

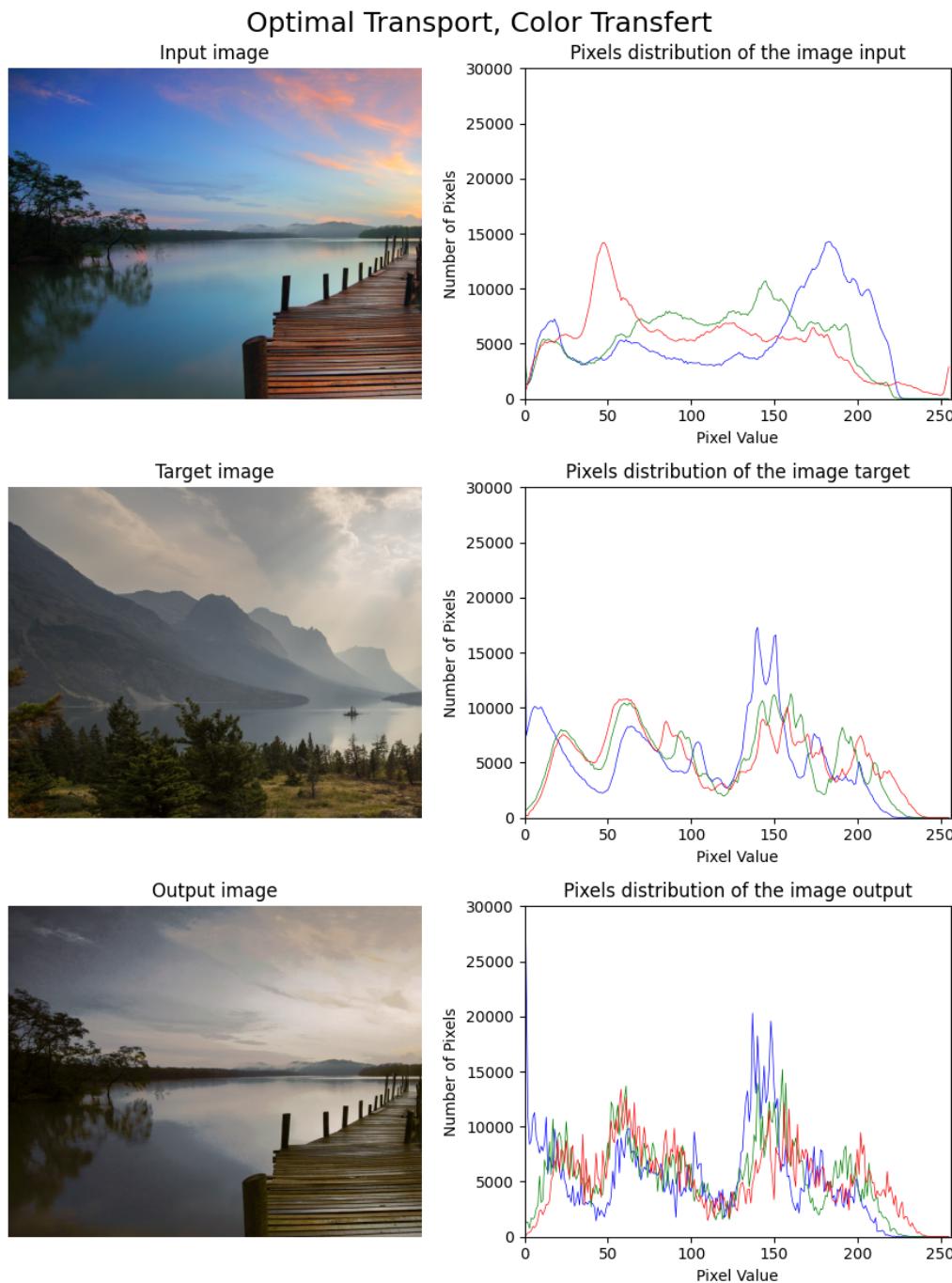
```

48 # The main loop of the sliced approach
49 image_source_copy = image_input.copy()
50 for iteration in tqdm(range(20)):
51     image_result = transport(image_source_copy, image_target, vector_random(3))
52     image_source_copy = image_result

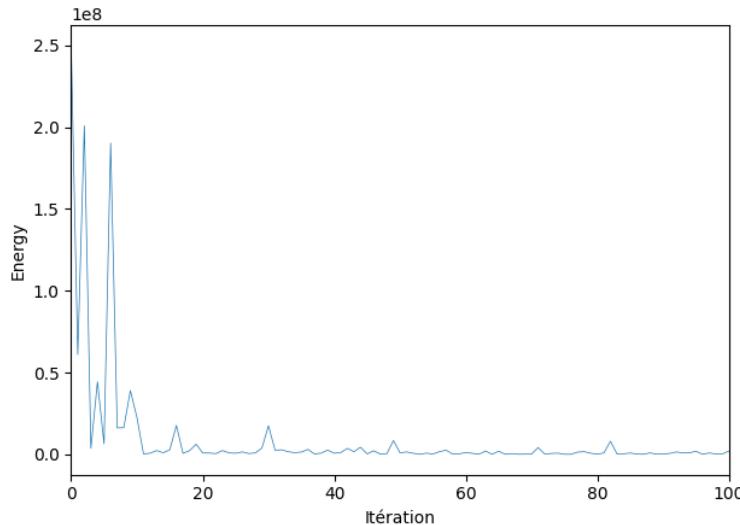
```

Le résultat varie en fonction du nombre d'itérations réalisées : en effet, plus il y a d'itérations plus le résultat correspond à nos attentes.

Voici le résultat sur 50 itérations :



6. Regardons l'énergie de transport au fur et à mesure des itérations pour déterminer à partir de combien d'itérations le résultat pourra être correcte.



Nous pouvons constater qu'à partir de 20 itérations, les couleurs de l'image *output* semblent se rapprocher de celles de l'image *target*.

## 2.2 Optimisation du code

Le code précédent présente des performances peu optimales notamment au sein de la boucle principale où chaque itération prend environ 16 secondes. Cette inefficacité peut poser des problèmes lorsqu'il faut un nombre d'itérations important pour obtenir le résultat que l'on souhaiterait.

Les deux fonctions à optimiser sont les fonctions *sort* et *transport* car elles renferment des boucles qui prennent un temps significatif pour s'exécuter.

Optimisation de la fonction *sort* :

Dans un premier temps on extrait les dimensions de l'image ainsi que les canaux de couleurs RGB. Puis on effectue une projection linéaire des pixels grâce au produit scalaire entre  $\theta$  et chaque pixel. On crée une matrice d'indices correspondant à la position de chaque pixel dans l'image. On combine par la suite les valeurs de projection et les indices en une matrice et chaque ligne contient la valeur de la projection associée. La fonction renvoie la matrice triée, qui contient les valeurs de projection et les indices correspondants ordonnés en fonction de la projection.

```

25 def sort(image, theta):
26     """
27         Fonction permettant de trier la projection des échantillons
28         d'une image sur une direction theta aléatoire
29         :param image: image sur laquelle on souhaite réaliser les projections
30         :param theta: direction aléatoire
31         :return: un tableau trié par couleur en fonction de leur projection dans la direction theta
32     """
33     h, l, _ = image.shape
34     color = image.reshape(-1, 3)
35     projection = np.dot(color, theta)
36     indices = np.arange(h * l).reshape(h, l)
37     sorting = np.column_stack((projection, indices.ravel()))
38     sorting = sorting[sorting[:, 0].argsort()]
39
40     return sorting

```

Optimisation de la fonction *transport* :

Pour remplacer la boucle conditionnelle *if*, on va simplement créer un masque booléen pour s'assurer que les coordonnées des pixels advectés restent à l'intérieur des limites de l'image *input*. La partie principale de la fonction qui advecte les pixels réalise le calcul du transport uniquement sur les pixels qui sont valides selon le masque. Ainsi tous les pixels sont traités en même temps.

```

42 def transport(image1, image2, theta):
43     """
44     Fonction réalisant le transport optimal sur l'image input
45     :param image1: image input
46     :param image2: image target
47     :param theta: direction aléatoire
48     :return: résultat du transport optimal des échantillons de l'image
49     input vers l'image target dans la direction theta
50     """
51
52     sorted_source = sort(image1, theta)
53     sorted_target = sort(image2, theta)
54     advected_samples = np.zeros_like(image1, dtype=np.uint8)
55
56     source_indices = sorted_source[:, 1].astype(int)
57     source_pixels = np.unravel_index(source_indices, image1.shape[:2])
58
59     valid_pixels = np.logical_and.reduce([(0 <= source_pixels[0]),
60                                         (source_pixels[0] < image1.shape[0]),
61                                         (0 <= source_pixels[1]),
62                                         (source_pixels[1] < image1.shape[1]),])
63
64     differences = sorted_source[:, 0] - sorted_target[:, 0]
65
66     advected_samples[source_pixels[0][valid_pixels], source_pixels[1][valid_pixels]] = np.clip(
67         image1[source_pixels[0][valid_pixels],
68                 source_pixels[1][valid_pixels]] - (differences[valid_pixels][:, np.newaxis] * theta),
69         a_min: 0,
70         a_max: 255).astype(np.uint8)
71
    return advected_samples

```

Les résultats sont convaincants car nous sommes passés de 16 secondes par itération à environ 2 secondes. Il sera plus facile pour la suite d'exécuter le programme avec un nombre conséquent d'itérations.

## 2.3 Fonctionnalités avancées

### 2.3.1 Régularisation

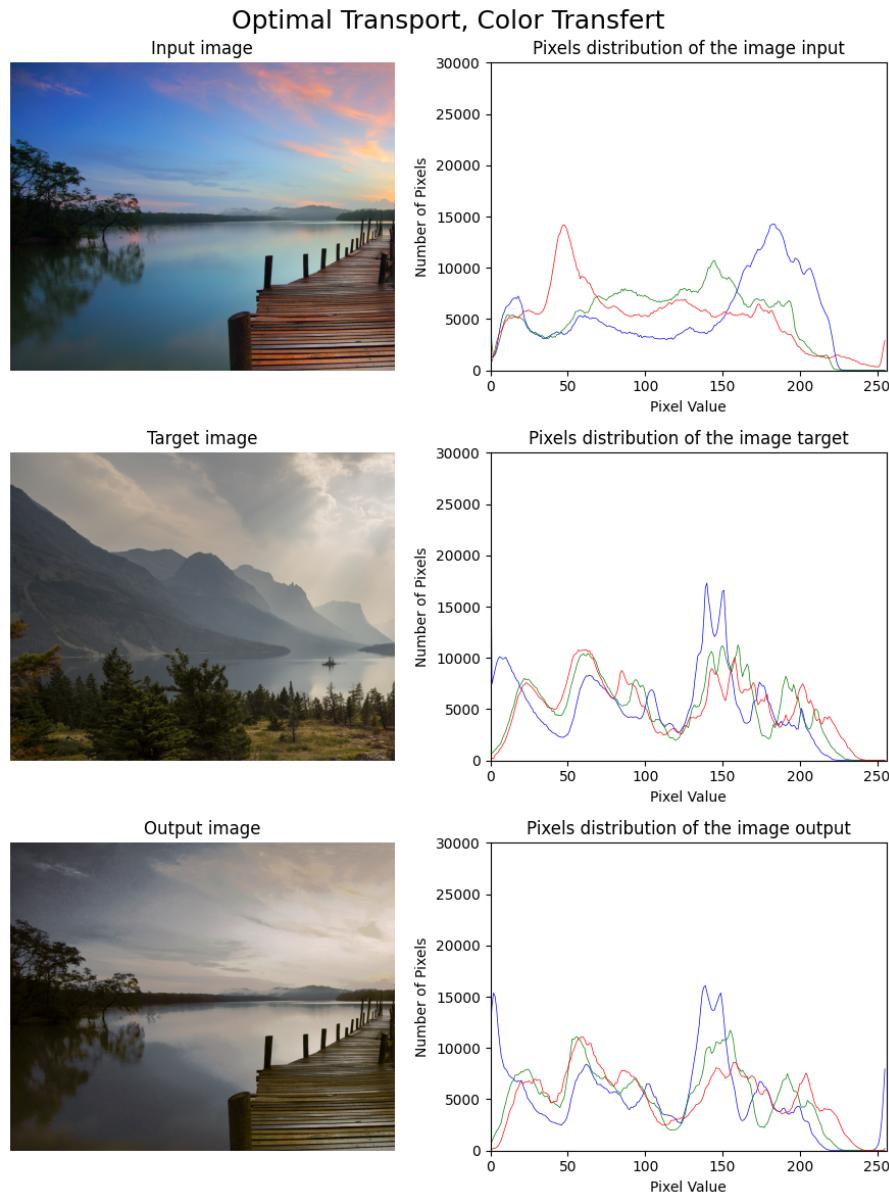
La régularisation est une fonctionnalité qui a pour but d'obtenir des résultats plus lisses et à gérer le bruit. En effet, l'histogramme de l'image *output* présenté dans la partie 2.1 nous montre qu'il y a énormément de bruit.

La méthode est la suivante. Une fois que nous avons la transformation de l'image *input* par rapport à l'image *target*, nous allons exprimer le plan de transport puis le filtrer pour lisser et ajouter le résultat filtré à l'image *input* afin obtenir l'image régularisée et lissée. J'ai choisi de filtrer l'image avec un filtre bilatéral. Pour se faire, j'ai réalisé la fonction *regularization* suivante :

```

73 def regularization(image_source, transported_image, sigma):
74     """
75     Fonction permettant la régularisation (gestion du bruit), afin d'obtenir un résultat plus lisse
76     :param image_source: image input
77     :param transported_image: image output
78     :param sigma: paramètre de régularisation
79     :return: image output lissée
80     """
81     difference = transported_image - image_source
82     filtered_difference = cv2.bilateralFilter(difference, d=15, sigmaColor=sigma, sigmaSpace=sigma)
83
84     final_image = np.clip(image_source + filtered_difference, a_min: 0, a_max: 255).astype(np.uint8)
85
  
```

Dans ce cas, c'est la valeur de *sigma* que l'on fait varier : plus elle sera basse moins le lissage sera important. Nous aurons juste à appeler cette fonction dans la boucle principale et adapter le *sigma* pour obtenir le meilleur résultat possible.



Ce résultat est vraiment convaincant, nous avons un histogramme très ressemblant à celui de l'image *target*.

### 2.3.2 Interpolation

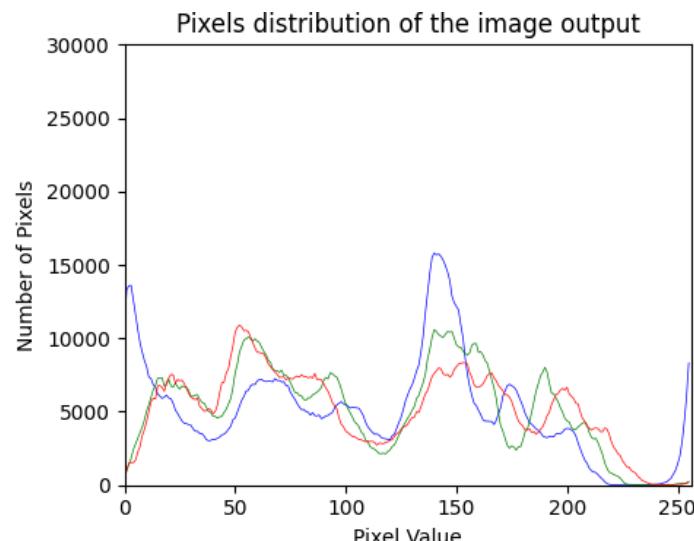
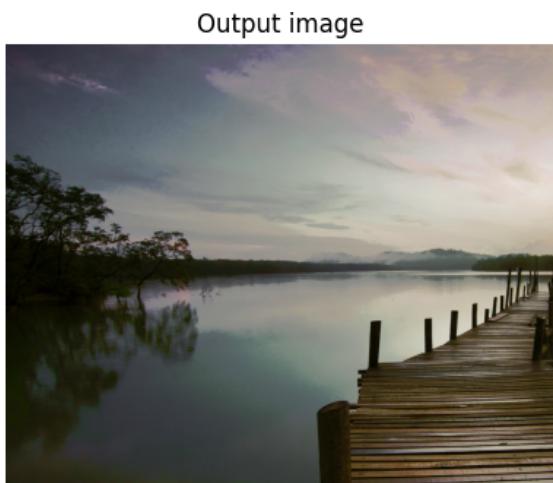
L'interpolation est utilisée pour estimer des valeurs intermédiaires entre deux mesures afin d'ajuster l'étape de l'advection. Au lieu de déplacer la valeur  $x$  vers  $y$  complètement, nous allons déplacer de  $\alpha$  de cette distance à chaque étape d'interpolation. Il suffit simplement de rajouter  $\alpha$  à la fonction *transport*.

```

42     def transport(image1, image2, theta, alpha):
43         """
44             Fonction réalisant le transport optimal sur l'image input
45             :param image1: image input
46             :param image2: image target
47             :param theta: direction aléatoire
48             :return: résultat du transport optimal des échantillons de l'image
49             input vers l'image target dans la direction theta
50         """
51
52         sorted_source = sort(image1, theta)
53         sorted_target = sort(image2, theta)
54         advected_samples = np.zeros_like(image1, dtype=np.uint8)
55
56         source_indices = sorted_source[:, 1].astype(int)
57         source_pixels = np.unravel_index(source_indices, image1.shape[:2])
58
59         valid_pixels = np.logical_and.reduce([
60             (0 <= source_pixels[0]),
61             (source_pixels[0] < image1.shape[0]),
62             (0 <= source_pixels[1]),
63             (source_pixels[1] < image1.shape[1]),])
64
65         differences = sorted_source[:, 0] - sorted_target[:, 0]
66
67         advected_samples[source_pixels[0][valid_pixels], source_pixels[1][valid_pixels]] = np.clip(
68             image1[source_pixels[0][valid_pixels]],
69             source_pixels[1][valid_pixels]] - (alpha * differences[valid_pixels][:, np.newaxis] * theta),
70             a_min: 0,
71             a_max: 255).astype(np.uint8)

```

Voici le résultatat pour  $\alpha = 0.5$  :



## 2.4 Animation

Pour réaliser une petite animation avec la fonctionnalité *FuncAnimation* dans la bibliothèque de *Matplotlib*, nous aurons besoin de définir la fonction *update plot* suivante :

```

15     def update_plot(iteration):
16         """
17             Fonction permettant d'afficher l'évolution de l'image et son histogramme en fonction
18             du transport réalisé
19             :param iteration: nombre d'itérations sur lequel on déroule l'animation
20             """
21         plt.clf()
22
23         if iteration == 0:
24             original_image = image_input
25             plt.subplot(*args: 1, 2, 1)
26             plt.imshow(original_image)
27             plt.title('Image Target')
28         else:
29             current_iteration = iteration - 1
30             current_image = images[current_iteration]
31             plt.subplot(*args: 1, 2, 1)
32             plt.imshow(current_image)
33             plt.title('Image Target')
34             plt.axis('off')
35
36         # Afficher l'histogramme
37         plt.subplot(*args: 1, 2, 2)
38         for color in ['r', 'g', 'b']:
39             plt.plot(*args: histograms[iteration][color], color=color, linewidth=0.5, alpha=1)
40         plt.title('Pixels distribution')
41         plt.xlabel('Pixel Value')
42         plt.ylabel('Number of Pixels')
43         plt.ylim(*args: 0, 30000)

```

Ci-dessous nous avons le code permettant le bon déroulement de l'animation. Nous allons récupérer grâce à la boucle principale, les images et leurs histogrammes correspondants puis nous réalisons la fonction *update plot* sur les données récupérer :

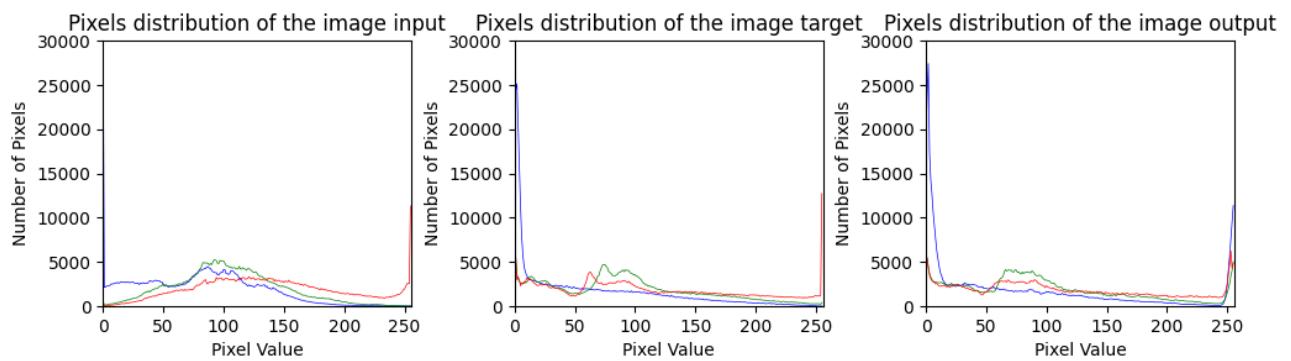
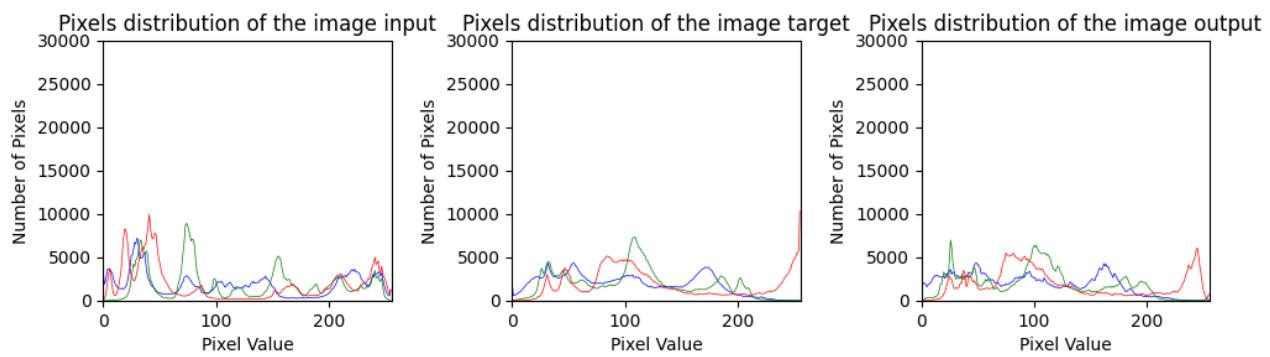
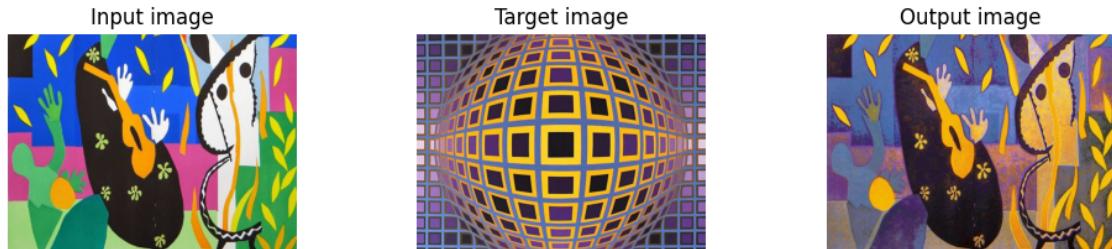
```

45 ▷ 45 if __name__ == '__main__':
46     # Initialisation des données
47     image_input = cv2.imread('pixelA-0.png')
48     image_target = cv2.imread('pixelB-0.png')
49     image_source_copy = image_input.copy()
50     histograms = []
51     images = [image_input]
52     alpha = 0.5
53     energies = [transport_image.calculate_energy(image_input, image_target, transport_image.vector_random(3))]
54
55     # Boucle principale
56     for iteration in tqdm(range(20)):
57         image_result = transport_image.transport(image_source_copy, image_target, transport_image.vector_random(3))
58         images.append(image_result)
59         iteration_histogram = {'b': [], 'g': [], 'r': []}
60         transport_energy = transport_image.calculate_energy(image_source_copy,
61                                                               image_result,
62                                                               transport_image.vector_random(3))
63         energies.append(transport_energy)
64
65         for channel, color in enumerate(['b', 'g', 'r']):
66             hist_channel = cv2.calcHist(images=[image_result[:, :, channel].astype(np.uint8)],
67                                         channels=[0],
68                                         mask=None,
69                                         histSize=[256],
70                                         ranges=[0, 256])
71             iteration_histogram[color] = hist_channel.tolist()
72
73         histograms.append(iteration_histogram)
74         image_source_copy = image_result
75
76         final_image = transport_image.regularization(image_input, image_source_copy, sigma=15)
77         final_image = np.clip(final_image, a_min=0, a_max=255).astype(np.uint8)

```

## 2.5 Application à d'autres images

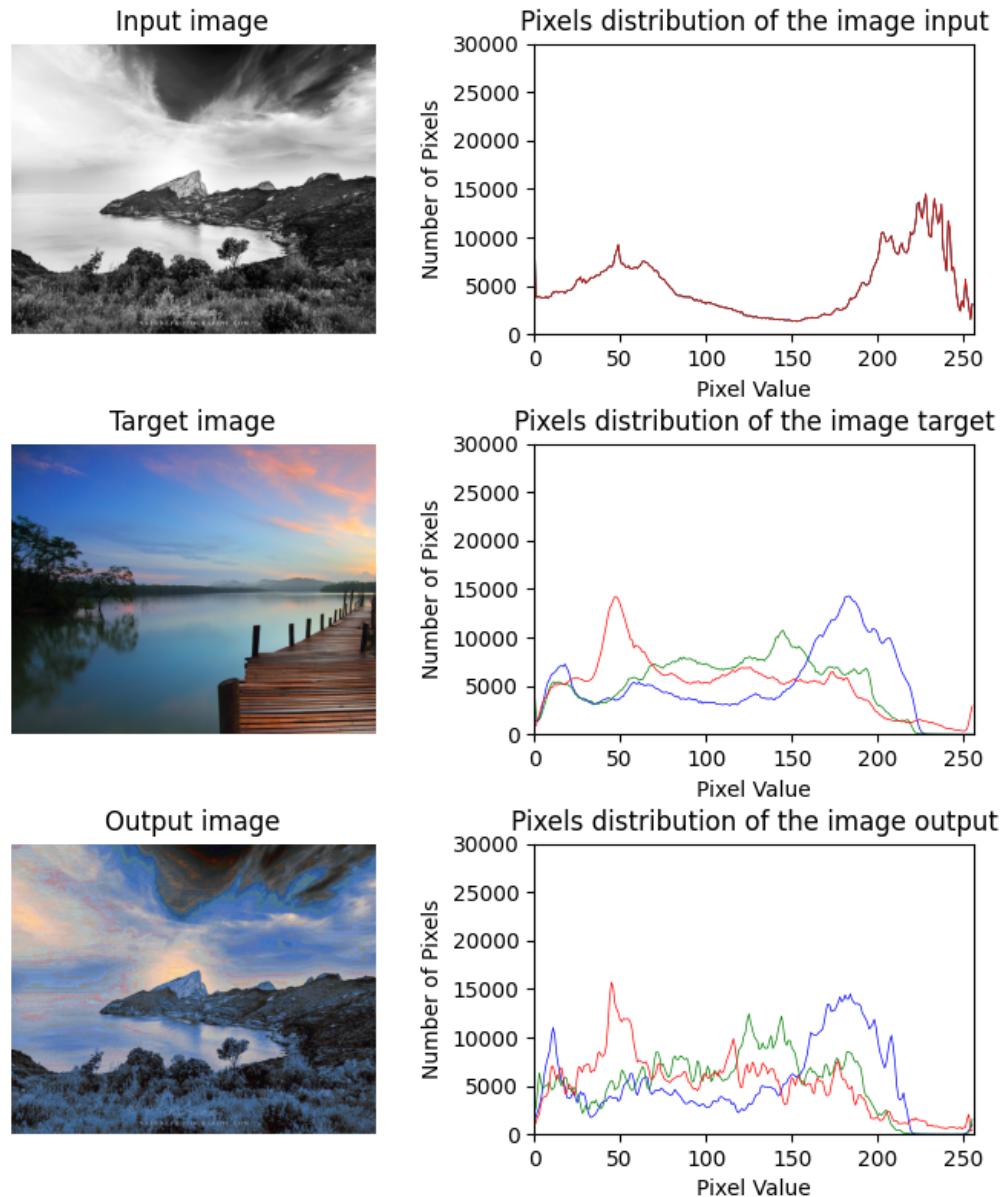
Il est intéressant de regarder si le transport s'applique de la même manière sur des images portant plus de détails.



## 2.6 Pour aller plus loin

Par curiosité j'ai voulu regarder ce que l'algorithme donnait sur le transport d'une image en noir et blanc vers une image en couleurs. Nous nous attendons à avoir un problème car l'histogramme de l'image en noir et blanc ne sera pas composé des 3 canaux.

Cependant voici le résultat :



Ce résultat est mieux qu'espéré, on passe d'un histogramme qui représente les nuances de gris à un histogramme avec les 3 canaux (RGB).

## Références

- David Coeurjolly - *Color Transfer via Discrete Optimal Transport using the sliced approach.*  
<https://codimd.math.cnrs.fr/s/2eRBqV9z1>
- Wikipédia - *Distance de Wasserstein.*  
[https://fr.wikipedia.org/wiki/Distance\\_de\\_Wasserstein#:~:text=La%20distance%20de%20Wasserstein%20est%20un%20moyen%20naturel%20de%20comparer,uniformes%20\(al%C3%A1latoires%20ou%20d%C3%A9terministes\).](https://fr.wikipedia.org/wiki/Distance_de_Wasserstein#:~:text=La%20distance%20de%20Wasserstein%20est%20un%20moyen%20naturel%20de%20comparer,uniformes%20(al%C3%A1latoires%20ou%20d%C3%A9terministes).)
- Rabin Julien, Gabriel Peyré, Julie Delon, Bernot Marc - *Wasserstein Barycenter and its Application to Texture Mixing.* SSVM'11, 2011, Israel. pp.435-446. hal-00476064.
- Nathael Gozlan, Paul-Marie Samson, Pierre-André Zitt - *Notes de cours sur le Transport Optimal*, 21 octobre 2022.  
<https://perso.math.u-pem.fr/samson.paul-marie/pdf/coursM2transport.pdf>
- Cécile Carrère, Didier Lesesvre, Paul Pegon - *Théorie générale du transport et applications.*  
[https://lesesvre.perso.math.cnrs.fr/optimal\\_transportation.pdf](https://lesesvre.perso.math.cnrs.fr/optimal_transportation.pdf)
- Julie Delon - *Optimal Transport for image processing.*  
[https://helios2.mi.parisdescartes.fr/~jdelon/enseignement/cours\\_image\\_m2/COURS\\_M2\\_color-transfer.pdf](https://helios2.mi.parisdescartes.fr/~jdelon/enseignement/cours_image_m2/COURS_M2_color-transfer.pdf)
- Gabriel Peyre - *Le transport optimal numérique et ses applications.*  
<https://www.youtube.com/watch?v=4FtamHah29M>