TP 8 INFO Manipulation d'images

Dans ce TP, nous allons manipuler des images numériques avec Python.

Pour commencer, récupérez l'ensemble des fichiers liés à ce TP dans la rubrique informatique du site se trouvant à l'adresse `http://vonbuhren.free.fr`, puis placez ces derniers au même emplacement que votre fichier Python.

Partie I Généralités

I.A - Images numériques

Il existe deux grands types d'images numériques : les **images matricielles** et les **images vectorielles**.

Une image matricielle est constituée d'un pavage carré dont chaque élément, appelé **pixel** (pour *picture element* en anglais), est coloré de manière uniforme. Une telle image peut être représentée par une matrice dans laquelle chaque coefficient représente un pixel.

Un pixel peut être codé de différentes manières selon le type d'images considérées :

- pour une image en noir et blanc, chaque pixel est codé sur 1 bit par 0 pour la couleur noire et par 1 pour la couleur blanche ;
- pour une image en niveau de gris, chaque pixel est codé sur 1 octet (i.e. 8 bits) par un entier compris entre 0 (pour la couleur noire) et 255 (pour la couleur blanche) ;
- pour une image en couleurs encodée au format RGB (pour `red, green, blue` en anglais), chaque pixel est codé sur 3 octets par un triplet (R, G, B) où R, G et B sont des entiers compris entre 0 et 255. Ces trois entiers permettent de doser respectivement la quantité de rouge, de vert et de bleu pour obtenir $2^{24} = 16\,777\,216$ de couleurs différentes ;
- certaines images en couleurs sont encodées au format RGBA (pour `red, green, blue, alpha` en anglais) : on utilise un 4^e octet pour coder la transparence du pixel.

A contrario, les images vectorielles sont encodées via des formules géométriques qui vont pouvoir être décrites d'un point de vue mathématique. La principale différence entre le format matriciel et le format vectoriel est qu'une image vectorielle peut être agrandie sans perdre sa qualité alors qu'une image matricielle perd en netteté à l'agrandissement.

I.B - Manipulations d'une image numérique en Python

Dans ce TP, nous n'étudierons que des images matricielles, principalement au format RGB. Afin de pouvoir manipuler des images en Python, nous allons utiliser les deux modules suivants.

```
from PIL import Image
import numpy as np
```

Le module `Image` nous permettra de charger, d'afficher et de sauvegarder une image se trouvant sur l'ordinateur en Python. Voici un exemple d'utilisation des principales fonctions de ce module.

```
from PIL import Image

# Chargement d'une image d'un fichier en une image PIL
img = Image.open('exemple.png')

# Affichage d'une image PIL
img.show()

# Sauvegarde d'une image PIL en fichier image
img.save('resultat.png')
```

Question 1 : En utilisant les instructions ci-dessus, afficher l'image `craies.png` avec l'interpréteur Python.

Le module `numpy` nous permettra de convertir l'image chargée avec `PIL` en une matrice de pixels. Voici un exemple d'utilisation des principales fonctions dont nous aurons besoin.

```
import numpy as np

# Convertion d'une image PIL en tableau numpy
tab = np.array(img_pil)

# Récupérer le format d'un tableau numpy
H, L, P = np.shape(tab)

# Récupérer les composantes RGB du pixel en position (i, j) = (242, 157)
R, G, B = tab[242, 157]

# Convertion d'un tableau numpy en image PIL
new_img = Image.fromarray(tab)
```

Question 2 : Quelles sont les composantes (R,G,B) du pixel en position $(i,j) = (215,559)$? Quelle est la teinte dominante de cette couleur ?

Les deux instructions ci-dessous vous seront utiles dans la suite du TP pour générer rapidement des tableaux nuls.

```
# Créer un tableau nul de même format et de même type que le tableau tab
new_tab = np.zeros_like(tab)

# Créer un tableau nul de format H x L x P contenant des entiers non signés sur 8 bits
# Pour rappel : un entier non signé sur 8 bits est compris entre 0 et 255
new_tab = np.zeros((H, L, P), dtype = np.uint8)
```

Partie II Transformations élémentaires d'une image

Question 3 : Écrire une fonction `symetrie(img)` prenant en entrée une image `img` et qui renvoie l'image obtenue en appliquant une symétrie horizontale à `img`.

Question 4 : Écrire une fonction `rotation(img)` prenant en entrée une image `img` et qui renvoie l'image obtenue en appliquant une rotation d'angle $\pi/2$ à `img`.

Le négatif d'une image est une image dont les couleurs ont été inversées par rapport à l'originale : pour chaque pixel de l'image d'origine, ses composantes (R,G,B) sont remplacées par $(255-R, 255-G, 255-B)$.

Question 5 : Écrire une fonction `negatif(img)` prenant en entrée une image `img` et qui renvoie le négatif de cette image.

Pour terminer cette partie, nous allons étudier la conversion d'une image encodée au format RGB en une image en niveau de gris. Le procédé classique consiste à remplacer les composantes (R,G,B) de chaque pixel par une seule valeur ℓ, appelée **luminance**, indiquant le niveau de gris du pixel, donnée par la formule

$$\ell = \lfloor 0{,}2126 \times R + 0{,}7152 \times G + 0{,}0722 \times B \rfloor.$$

La formule tient compte de la manière dont l'œil humain perçoit les trois composantes rouge, vert et bleu : une lumière verte apparaît plus claire qu'une lumière rouge, et encore plus qu'une lumière bleue.

Question 6 : Écrire une fonction `niveau_gris(img)` prenant en entrée une image `img` et qui renvoie la conversion de cette image en niveau de gris.

Partie III Traitement d'une image par convolution

Dans cette partie, nous allons étudier le traitement d'image par l'application de filtres de convolution. Dans ce contexte, on considère une matrice de petite taille appelée **masque**. Dans la suite, nous nous limiterons à des masques de la forme

$$M = \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1,0} & m_{1,1} & m_{1,2} \\ m_{2,0} & m_{2,1} & m_{2,2} \end{pmatrix} \in \mathcal{M}_3(\mathbb{R}).$$

Appliquer le masque M à une matrice $T \in \mathcal{M}_{H,L}(\mathbb{R})$, de coefficients $t_{i,j}$ pour $(i,j) \in [\![0, H-1]\!] \times [\![0, L-1]\!]$, revient à calculer le produit de convolution de T par M. Ce dernier, noté $T \star M$, est définie comme la matrice $R \in \mathcal{M}_{H,L}(\mathbb{R})$, de coefficients $r_{i,j}$ pour $(i,j) \in [\![0, H-1]\!] \times [\![0, L-1]\!]$, définie par :

- en dehors des coefficients sur le bord de la matrice R, on applique la formule

$$\begin{aligned} \forall (i,j) \in [\![1, H-2]\!] \times [\![1, L-2]\!], \qquad r_{i,j} = {} & m_{1,1}\,t_{i-1,j-1} + m_{1,2}\,t_{i-1,j} + m_{1,3}\,t_{i-1,j+1} \\ & + m_{2,1}\,t_{i,j-1} + m_{2,2}\,t_{i,j} + m_{2,3}\,t_{i,j+1} \\ & + m_{3,1}\,t_{i+1,j-1} + m_{3,2}\,t_{i+1,j} + m_{3,3}\,t_{i+1,j+1}; \end{aligned}$$

- les coefficients sur le bord de la matrice R sont nuls.

Remarque 1 : Dans la cadre du traitement d'images, les coefficients des matrices T et R seront des entiers compris entre 0 et 255. Ainsi, lorsque la formule ci-dessus fournit un résultat vérifiant $r_{i,j} < 0$ ou $r_{i,j} > 255$, on remplacera cette valeur respectivement par $r_{i,j} = 0$ ou par $r_{i,j} = 255$.

Question 7 : Écrire une fonction `conv(T, M)` prenant en entrée une matrice `T` et un masque `M` de taille 3×3 et qui renvoie le produit de convolution de `T` par `M`.

Pour traiter une image au format RGB avec le procédé défini ci-dessus, il suffit d'appliquer un masque à chacune des trois composantes RGB de l'image. Dans un souci de simplification, nous appliquerons le même masque aux trois composantes de l'image.

Question 8 : Écrire une fonction `filtre(img, M)` prenant en entrée une image `img` et un masque `M` de taille 3×3 et qui renvoie l'image obtenue en appliquant le masque `M` à chacune des composantes RGB de l'image `img`.

Le choix du masque est à faire en fonction de l'effet recherché sur l'image.

Question 9 : Déterminer l'effet obtenu sur l'image pour chacun des trois masques ci-dessous.

$$M_1 = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad M_2 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}, \qquad M_3 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}.$$

Partie IV Sténographie d'une image

La stéganographie est la technique consistant à dissimuler une information sensible dans un media de couverture d'apparence anodin. Dans le cadre de ce TP, nous allons voir comment cacher une image dans une autre.

Dans une image au format RGB, chaque composante d'un pixel est un entier $x \in [\![0,255]\!]$ codé sur 8 bits : en utilisant la décomposition binaire $x = \sum_{k=0}^{7} x_k 2^k$ où $(x_0,\dots,x_7) \in \{0,1\}^8$, le nombre x est représenté par

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0

Les quatre bits x_7, x_6, x_5 et x_4 (dits de poids forts) ont plus d'importance que les quatre bits x_3, x_2, x_1 et x_0 (dits de poids faibles) sur la valeur de l'entier x. En particulier, la modification des bits de poids faibles des différentes composantes des pixels d'une image n'entraîne qu'une altération très légère de cette image.

En utilisant les remarques ci-dessus, nous pouvons ainsi cacher les quatre bits de poids forts d'une seconde image dans la première image.

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0

$\xrightarrow{\text{cacher}}$

x_7	x_6	x_5	x_4	y_7	y_6	y_5	y_4

Pour récupérer l'image cachée, il suffit d'extraire tous les bits de poids faibles comme ci-dessous.

z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0

$\xrightarrow{\text{révéler}}$

z_3	z_2	z_1	z_0	0	0	0	0

Question 10 : Une image a été cachée dans le fichier `craies.png` en utilisant le procédé décrit ci-dessus. Afficher cette image cachée.