TP12 - Images

→ *Q.1) Récupérez le fichier* `TP12.py` *sur cahier de prépa dans lequel vous pourrez sauvegarder votre travail.*

A) Matrices de pixels

Une image matricielle est un rectangle composé de pixels (picture elements).
Lorsque l'image est en niveau de gris, chaque pixel est un entier compris entre `0` et `255`, ou encore un flottant entre `0` et `1` : plus le nombre est grand, plus le pixel est lumineux.
Pour les images en couleurs, il existe plusieurs codage : nous allons utiliser le système RGB. La couleur d'un pixel est déterminée par trois entiers entre `0` et `255` (ou trois flottants entre `0` et `1`) en utilisant une synthèse additive : chaque nombre représente l'intensité de rouge, vert, bleu dans la couleur du pixel. Par exemple, `(255, 0, 0)` représente le rouge, `(0, 255, 0)` le vert et `(0, 0, 255)` le bleu. On retrouve les nuances de gris en prenant un pixel de la forme `(n, n, n)`.

Pour nous, une image en niveaux de gris sera représentée par un tableau `numpy` de taille `h x l` rempli d'entiers. Une image en couleurs sera un tableau à trois dimensions de taille `h x l x 3`. Pour cela :

- on importe la bibliothèque : `import numpy as np`
- `im = np.zeros((p, q), dtype='uint8')` crée un tableau de taille `p x q` rempli de 0
- `im.shape` donne la taille de chaque dimension du tableau
- `im[i, j]` donne l'élément en position `i, j`
- `im[i, :]` donne la ligne `i`
- `im[:, j]` donne la colonne `j`

On utilisera des entiers entre `0` est `255` pour encoder les couleurs.
Pour afficher l'image, on utilise ici la bibliothèque `matplotlib` :

- pour afficher une image en niveaux de gris, on utilise : `plt.imshow(image, cmap='gray')`
- pour afficher une image en couleurs, on utilise : `plt.imshow(image)`

→ *Q.2) Créer une image de taille* 2×2 *dont le pixel en haut à gauche est blanc et les autres sont noirs, puis l'afficher.*

→ *Q.3) Créer un tableau de zéros nommé* `im1` *de taille* 100×150*, puis mettre à* 255 *tous les pixels* (x, y) *tels que* $x \in [20, 60]$ *et* $y \in [100, 120]$.
Afficher l'image `im1`.

→ *Q.4) Créer un tableau de zéros de taille* $100 \times 150 \times 3$ *nommé* `im2`*, puis*

- *mettre à* 255 *la première coordonnée de tous les pixels* (x, y) *tels que* $x \in [40, 60]$ *et* $y \in [20, 130]$ *;*
- *mettre à* 255 *la deuxième coordonnée de tous les pixels* (x, y) *tels que* $x \in [20, 80]$ *et* $y \in [40, 60]$ *;*
- *mettre à* 255 *la troisième coordonnée de tous les pixels* (x, y) *tels que* $x \in [20, 80]$ *et* $y \in [100, 120]$ *;*

Afficher `im2`*, puis sa composante bleue en niveau de gris. On peut utiliser* `plt.figure()` *pour créer une nouvelle figure avant d'afficher la deuxième image.*

→ *Q.5) Que renvoie la commande* `np.random.rand(20, 20, 3)` *?*

→ *Q.6) Afficher une image couleur aléatoire de taille* 30×30*, puis sa composante rouge en niveau de gris.*

→ *Q.7) Afficher une image de taille* 100×100 *faisant apparaître un fond blanc et un disque bleu centré en* $(49, 49)$ *et de rayon* 10 *pixels.*

Pour ouvrir une image et la stocker dans un tableau, on utilise la fonction `imread` du module `pyplot` : `im = plt.imread('fichier.png')` stocke l'image sous forme d'un tableau numpy dans la variable `im`.

→ *Q.8) Charger l'image* `lena.png` *trouvable sur cahier de prépas. Afficher sa taille puis l'afficher avec* `matplotlib`.

B) Quelques algorithmes sur les images

On travaille sur des images en niveaux de gris pour simplifier. On pourra tester les algorithmes sur l'image `mandril.png`.

B.1 Transformations

→ *Q.9) Écrire une fonction* `symVert(im)` *qui prend en paramètre une image en niveaux gris et renvoie une copie obtenue en faisant la symétrie axiale par rapport à l'axe vertical qui passe au milieu de l'image.*
On peut utiliser `im.copy()` *pour réaliser une copie de l'image.*

Pour effectuer une rotation de centre (x_c, y_c) et d'angle θ, on commence par créer une image noire de même taille que l'originale, puis pour chaque pixel (x, y), on calcule sa position après rotation :

$$\begin{cases} x' = x_c + (x - x_c)\cos(\theta) + (y - y_c)\sin(\theta) \\ y' = y_c - (x - x_c)\sin(\theta) + (y - y_c)\cos(\theta) \end{cases}$$

sans oublier de convertir les valeurs en entier. Si (x', y') ne sort pas de l'image, on modifie le pixel dans la nouvelle image, sinon, le pixel est perdu.

→ *Q.10) Écrire une fonction* `rotation(im, x, y, theta)` *qui implémente cet algorithme.*

Pour réduire la taille d'un image d'un facteur `n`, on peut remplacer chaque rectangle de pixels de taille `n x n` par son pixel central.

→ *Q.11) Écrire une fonction qui réalise cette réduction.*

Le résultat n'est pas de très bonne qualité : plutôt que de remplacer un rectangle par son centre, on peut le remplacer par la moyenne de ses pixels.

→ *Q.12) Écrire une fonction* `moyenne` *qui prend en argument un tableau à deux dimensions d'entiers et renvoie la moyenne entière des valeurs du tableau.*

→ *Q.13) Écrire la nouvelle version de la réduction.*

B.2 Convolutions

Appliquer un filtre à une image revient à remplacer la valeur de chaque pixel par une moyenne pondérée de ses voisins. Les poids à utiliser sont donnés par la matrice de convolution. Cette matrice est carrée de taille impaire $2p+1$, de sorte que pour le pixel (x, y), on utilise les pixels (x', y') avec $x' \in [\![x-p, x+p]\!]$ et $y \in [\![y-p, y+p]\!]$. Les pixels sur la bande de taille p au bord de l'image ne sont pas modifiés.

→ *Q.14) Écrire une fonction* `conv(im, M, x, y)` *qui calcule la nouvelle valeur du pixel* `(x,y)` *après l'application de la matrice de convolution* `M`.

→ *Q.15) Écrire* `filtre(im, M)` *qui renvoie une nouvelle image obtenue à partir de* `im` *en appliquant la matrice de convolution* `M`.

On prend `M = np.array([[-1/9, -1/9, -1/9], [-1/9, 8/9, -1/9], [-1/9, -1/9, -1/9]])`.

→ *Q.16) Tester la fonction* `filtre` *avec la matrice* `M`. *Que permet de faire ce filtre ?*

B.3 Détection de contours

Une technique simple de détection de contours d'un objet dans un image en niveaux de gris consiste à repérer les zones où les niveaux de gris changent très vite. Pour cela, on calcule la variation de couleur C en x et en y :

$$V_x(i,j) = \max(C(i+1,j), C(i-1,j)) - \min(C(i+1,j), C(i-1,j))$$
$$V_y(i,j) = \max(C(i,j+1), C(i,j-1)) - \min(C(i,j+1), C(i,j-1)).$$

L'amplitude de variation est alors $V(i,j) = \sqrt{V_x(i,j)^2 + V_y(i,j)^2}$.
On crée alors une image en noir et blanc dont les pixels sont blancs si $V(i,j) < s$ et noirs si $V(i,j) \geq s$, où s est un seuil à choisir.

→ *Q.17) Écrire une fonction* `contours(image, seuil)` *qui renvoie une image en noir et blanc à partir de l'image et du seuil.*

→ *Q.18) Tester la fonction sur l'image* `mandril.png` *avec diverses valeurs du seuil.*