TP 12 : Traitement d'images : transformations, rotations, applications de filtres et détection de contours

Dans ce TP, on continue de manipuler des images. On ne va travailler qu'avec des images en noir et blanc pour ce TP, qui seront représentées par des tableaux à deux dimensions (et non pas trois comme pour des images couleur au format RGB).

1 Création d'un répertoire de travail

Avant de commencer ce TP, créez un autre répertoire spécifique dans votre espace personnel, par exemple `TP_Image2`. Récupérer l'annexe sur le site web, que vous pouvez placer dans ce répertoire, ainsi que l'image `lena_gris.png`, et `lena_carre.png`. Changer le répertoire de travail :

```
os.chdir("U://Documents/chemin/vers/votre/répertoire/TP_Image2") #à modifier dans l'annexe.
```

2 Syntaxe pour les images

fonction	description
`im=Image.open('image.png')`	Ouvre l'image `image.png`, stocke le résultat dans la variable `im`.
`im.show()`	affiche l'image stockée dans `im`
`tab=np.array(im)`	calcule le tableau de pixels associé à l'image, le résultat est stocké dans `tab`.
`tab.shape`	la forme du tableau (couple (h, ℓ) pour une image en noir et blanc).
`Image.fromarray(tab)`	image associée à un tableau
`np.zeros((h,l), dtype=np.uint8)`	tableau 2D rempli de zéros de type `np.uint8`
`tab.copy()`	copie du tableau `tab`
`im.save('sauvegarde.png')`	sauvegarde une image.
`np.uint8(x)`	convertit x (flottant, entier...) en entier sur 8 bits.

3 Quelques transformations

3.1 Symétrie

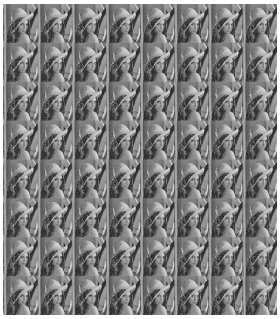
Question 1. *Symétrie verticale*. Écrire une fonction `symetrie_verticale(T)` renvoyant un nouveau tableau obtenu par symétrie verticale de `T`. Exécuter `test_symetrie_verticale()` et vérifier que vous obtenez bien l'image suivante.

3.2 Transformation du photomaton

Cette transformation a été introduite en 1997 par Jean-Paul Delahaye et Philippe Mathieu dans la revue *Pour la Science*. Cette transformation n'est définie que pour des tableaux dont les dimensions (hauteur et largeur) sont paires (c'est le cas de notre image de Léna). L'idée est de considérer les pixels par blocs de taille 2×2, le pixel en haut à gauche du bloc ira dans le quart supérieur gauche de l'image, de même pour les trois autres pixels qu'on répartit dans les trois autres quarts de l'image, comme sur la figure suivante :

Question 2. Écrire une fonction `transformation_photomaton(T)` renvoyant un nouveau tableau obtenu par cette transformation sur `T`. Exécuter `test_photomaton()` et vérifier que vous obtenez bien l'image suivante (notez que l'on n'a pas 4 fois la même image : tous les pixels initiaux se retrouvent dans la transformation).

Question 3. Écrire une fonction `itere_photomaton(T, n)` appliquant n fois la transformation du photomaton. Tester pour de petits n avec `test_itere_photomaton(n)`. Voici le résultat avec $n = 3$.



Comme on itère une bijection, il existe un plus petit entier $n > 0$ (appelé la période de la transformation), tel qu'itérer n fois cette transformation ramène à l'image initiale.

Question 4. Trouver ce plus petit n pour l'image carrée de taille 256×256 `lena_carre.png`. Il n'est pas très élevé. *Indication :* le test d'égalité entre deux tableaux Numpy de mêmes tailles `T==T2` fournit un tableau de booléens de la même taille, chaque booléen étant le résultat d'un test d'égalité entre deux éléments situés aux mêmes indices. On peut utiliser `(T==T2).all()` pour vérifier que tous ces tests donnent `True`. Inversement, `(T!=T2).any()` permet de vérifier si au moins deux entrées de `T` et `T2` sont différentes.

Remarque : la période pour `lena_gris.png` est 2016, ce qui prend un peu de temps pour être calculé naïvement. Il est en fait facile de calculer cette période à partir des dimensions de l'image, on pourra consulter l'article [1] Wikipédia.

4 Rotation d'une image

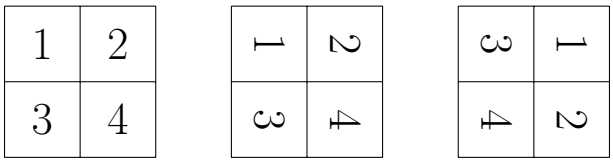
4.1 Nouveau tableau explicite

Question 5. Écrire une fonction `rotation_1(T)` prenant en entrée un tableau Numpy, et renvoyant un nouveau tableau obtenu par rotation de 90 degrés vers la droite de `T`. L'appel `test_rotation_1()` doit produire l'image ci-dessous :

1. `https://fr.wikipedia.org/wiki/Transformation_du_clich%C3%A9_Photomaton`

4.2 Un algorithme récursif pour la rotation

Conseil : passer cette section dans un premier temps, pour y revenir en fin de TP.
Petit défi ici : écrire un algorithme effectuant la rotation d'un tableau *carré* dont la hauteur (égale à la largeur) est une puissance de 2, avec la contrainte d'avoir une complexité $O(1)$ en mémoire en plus de la pile d'appels de fonction. On ne peut donc pas utiliser d'autre tableau. L'algorithme proposé est récursif, le principe est résumé dans la figure ci-dessous.



Explications de l'algorithme :

- Si le tableau à rotationner possède seulement 1 pixel, il n'y a rien à faire (c'est le cas de base).
- Sinon, on découpe le tableau (rappel : il est carré, de taille une puissance de 2) en 4 morceaux. Via 4 appels récursifs, on effectue des rotations sur les 4 morceaux. Une fois ceci effectué, on permute intelligemment les pixels 4 par 4 pour obtenir la rotation du tableau initial.

Pour écrire l'algorithme, on va utiliser une fonction auxiliaire `aux(i,j,nb)`, dont le but est de travailler sur le tableau de taille nb $\times$ nb dont le pixel en haut à gauche est (i, j).

Question 6. Compléter la fonction `rotation_2(T)`. Vérifier ensuite que l'appel `test_rotation_2()` s'effectue correctement (il s'agit d'effectuer une rotation sur l'image `lena_carre.png` dont le tableau associé est de taille $2^8 \times 2^8$).

5 Application d'un filtre

On peut modifier une image en lui appliquant différents traitements (lissage, flou, etc...). Il suffit pour cela d'appliquer un *filtre* à l'image. La plupart des filtres utilisent une *convolution* par un *masque* pour réaliser une modification. Ces masques sont des matrices de petites tailles, en général 3×3 ou 5×5. Dans la suite, nous ne considérerons que des masques de taille 3×3 :

$$M = \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1,0} & m_{1,1} & m_{1,2} \\ m_{2,0} & m_{2,1} & m_{2,2} \end{pmatrix}$$

Le filtre associé à ce masque transforme un tableau $T = (t_{i,j})_{0 \leq i < h, 0 \leq j < \ell}$ associé à une image, en le tableau $T' = T \otimes M$, où

$$t'_{i,j} = m_{0,0}t_{i-1,j-1} + m_{0,1}t_{i-1,j} + m_{0,2}t_{i-1,j+1} + m_{1,0}t_{i,j-1} + m_{1,1}t_{i,j} + m_{1,2}t_{i,j+1} + m_{2,0}t_{i+1,j-1} + m_{2,1}t_{i+1,j} + m_{2,2}t_{i+1,j+1}$$

On remarque que cette transformation n'est pas bien définie pour un pixel du bord ($i \in \{0, h-1\}$ ou $j \in \{0, \ell - 1\}$), on convient de laisser le pixel inchangé [2]

Question 7. Écrire une fonction `convolution(T,M)` qui prend en arguments un tableau `T` (associé à une image) et un tableau `M` de taille 3×3, et retournant le tableau $T \otimes M$. Attention :

2. En général, suivant le masque à appliquer, on peut donner une formule adéquate pour un point du bord, mais on ne s'embêtera pas ici.

— les coefficients de M peuvent être entiers, flottants... Ceux de T et de $T \otimes M$ seront `np.uint8`, on fera éventuellement des conversions (`round` pour prendre la partie entière), et prenez garde à ce qu'une valeur négative soit convertie en 0, et une valeur supérieure à 255 en 255.

— laissez les pixels du bord inchangés.

Indication : ne pas hésiter à introduire une fonction intermédiaire pour le calcul de $t'_{i,j}$.

Voici deux exemples de masques :

$$M_1 = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix} \quad \text{et} \quad M_2 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

M_1 est appelé *moyenneur* : l'image obtenue sera plus floue que l'image initiale, un pixel devenant la moyenne des pixels alentour. À l'inverse, M_2 est appelé *réhausseur* : il accentue les contrastes.

Question 8. Appeler `test_moyenneur()` et `test_rehausseur()`, vérifiez que vous obtenez les images ci-dessous.

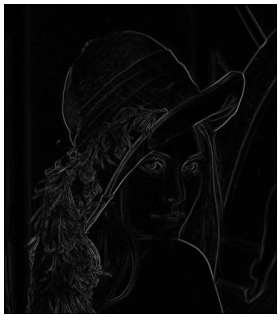
6 Détection de contour

Il s'agit ici de repérer dans l'image les zones où les valeurs de pixels changent brusquement (on parle de *contour*). L'idée consiste en quelque sorte à *dériver* l'image. Par exemple, la valeur de la dérivée en (i, j) du tableau $T = (t_{i,j})$ sur l'axe vertical pourrait être par exemple $\frac{t_{i+1,j} - t_{i-1,j}}{2}$. On pourrait encoder ceci avec un filtre, le problème est que l'intervalle $[\![0, 255]\!]$ n'est pas du tout conservé. On choisit ici de regarder la valeur absolue de la dérivée, et de sommer les dérivées suivant les axes verticaux et horizontaux. En résumé, on calcule pour (i, j) qui n'est pas sur le bord la valeur :

$$e_{i,j} = \left| \frac{t_{i+1,j} - t_{i-1,j}}{2} \right| + \left| \frac{t_{i,j+1} - t_{i,j-1}}{2} \right|$$

qu'on appelle couramment *l'énergie* d'un pixel.

Question 9. Écrire une fonction `calcul_energies(T)` prenant en entrée un tableau et renvoyant (toujours sous forme `np.uint8`) le tableau de même taille contenant les énergies. Attention : convertir chaque entrée en entier standard (`int`) avant d'appliquer la soustraction. On laissera 0 sur le bord. Appeler `test_energie()` et vérifier que vous obtenez l'image suivante (à gauche). Il est peut-être un peu plus parlant de calculer le négatif (valeurs $255 - e_{i,j}$ pour tous i, j), vous obtiendrez l'image de droite.



Indication : là encore, ne pas hésiter à introduire une fonction intermédiaire.