TP n° 6 – Manipulation & traitement d'images

Les manipulations d'images en `python` demandent la lecture de fichiers au format `*.png` d'un répertoire donné. Pour y accéder, il est nécessaire de définir le répertoire de travail du *shell* avec la commande `chdir` de la bibliothèque `os`.

MÉMO – **Répertoire de travail**

`import os`

`os.chdir("chemin")` ..... changer de répertoire

sur windows, remplacer chaque *backslash* \ par une paire \\ ou mettre `r` avant la chaîne du chemin

`os.listdir()` ..... lister le contenu du répertoire courant

Une fois dans le bon répertoire, on charge une image `"fichier.png"` sous forme de tableau `numpy` d'entiers non signés sur 8 bits (type `uint8`, valeurs dans $[\![0, 255]\!]$). avec la commande :

```
import numpy as np
from PIL import Image
T = np.asarray(Image.open("image.png"))
```

Si l'image est en niveaux de gris, `T` est un tableau d'entiers non signés de type `uint8` compris entre 0 (noir) et 255 (blanc) contenant n lignes et p colonnes. Si l'image est en couleur, `T` est un tableau de dimension $n \times p \times c$ contenant n lignes, p colonnes et $c = 3$ informations de couleur associées à l'intensité L_R du rouge, L_G du vert et L_B du bleu (codage RGB), toutes codées sur un octet (8 bits) et donc comprises entre 0 et 255. Dans le cas où la transparence est prise en compte (cas du RGBA), on a $c = 4$, la quatrième information étant celle du canal α associé à la transparence (figure 1).

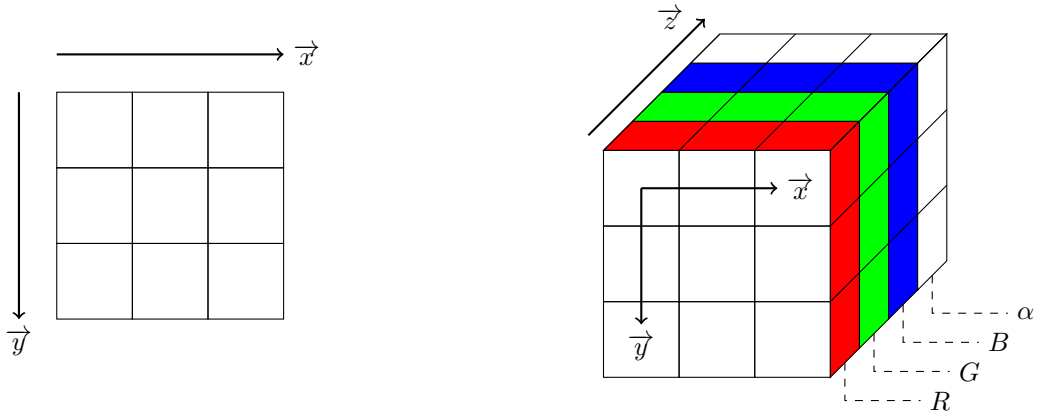

FIGURE 1 – Principe de stockage des images en niveaux de gris (gauche) et en couleur (droite).

MÉMO – **Manipulation d'images**

`import numpy as np` ..... bibliothèque de gestion des tableaux

`from PIL import Image` ..... bibliothèque de gestion des images

`import matplotlib.pyplot as plt` ..... bibliothèque de gestion des graphiques

`T = np.asarray(Image.open("image.png"))` ..... charge une image dans un tableau

`plt.figure()`
`plt.imshow(T)`
`plt.axis("off")`
`plt.show()`
..... affiche l'image de tableau `T`

`Image.fromarray(T).save("fichier.png")` ..... enregistrer une image en `"fichier.png"`

MÉMO – **Tableaux**

`import numpy as np` ..... chargement du module numpy
`T.dim` ..... nombre de dimensions
`T.shape` ou `np.shape(T)` ..... nombre d'éléments par dimension
`T.size` ..... nombre total d'éléments (produit des dimensions)
`N = T.copy()` ..... recopie des éléments → nouveau tableau
`T.dtype()` ..... type des éléments du tableau (`'int8'`, `'int64'`, `'float64'`, ...)
`N = T.astype('uint8')` ..... changement de type des éléments pour `uint8`
`T = np.zeros((p,q), dtype='uint8')` ..... création d'un tableau nul de p lignes et q colonnes

Coupes

`T[i,j]` ..... élément de la ligne `i` et de la colonne `j`
`T[i,:]` ou `T[i]` ..... vecteur correspondant à la ligne `i`
`T[:,j]` ..... vecteur correspondant à la colonne `j`
`T[a:b,c:d]` ..... sous-tableau formé des lignes `a` à `b-1` et des colonnes `c` à `d-1`

Fonctions : `T` $\mapsto$ tableau de mêmes dimensions que `T`

`np.floor(T)` ..... $\{\lfloor x \rfloor \ : \ x \in T\}$ partie entière
`np.ceil(T)` ..... $\{\lceil x \rceil \ : \ x \in T\}$ entier supérieur, défini comme $\lceil x \rceil = -\lfloor -x \rfloor$

Analyse de données

`T.min()` ou `np.amin(T)` ..... minimum
`T.max()` ou `np.amax(T)` ..... maximum
`T.mean()` ou `np.mean(T)` ..... moyenne
`T.sum()` ou `np.sum(T)` ..... somme
`T.cumsum()` ou `np.cumsum(T)` ..... somme cummulée

Exercice 6.1 (Type d'images, manipulations élémentaires)

1. Télécharger l'archive d'images `Info-TP06_images.zip`, la décompresser puis initialiser le code `python` en définissant le répertoire `images` comme répertoire de travail du *shell*.
2. Définir par compréhension une liste `images` de tableaux associés aux images `lena.png`, `fleur.png` et `des.png`.
3. Définir une fonction `TypeImage` qui prend comme argument le tableau associé à une image et qui renvoie son type (N&B, RGB ou RGBA) et un tuple avec ses dimensions. Tester cette fonction sur les tableaux de la liste `images`.
4. Définir une procédure `Affichage` qui prend comme argument le tableau associé à une image et qui affiche à l'écran cette image avec la commande `imshow`. L'affichage par défaut étant en couleur, pour les images en niveaux de gris, on utilisera l'argument `cmap="gray"`. On notera aussi que pour ne pas afficher les axes, on utilisera la commande `plt.axis('off')`.
5. Définir une procédure `AffichageComposantes` qui prend comme argument le tableau associé à une image et qui affiche chaque composante de couleur dans une sous-figure. Le nombre de sous-figures dépend du type d'image renvoyé par la fonction `TypeImage` (1 pour N&B, 3 pour RGB et 4 pour RGBA). Comme pour les images en niveaux de gris, on utilisera l'argument `cmap="Reds"` pour afficher une carte de rouges avec la fonction `imshow`. Traduire vert et bleu pour les deux autres cartes de couleur et gris pour la carte de transparence. Dans le cas particulier d'images RGBA où la valeur de transparence est toujours à 255, on utilisera la décomposition des images RGB.
6. Trouver un moyen pour afficher seulement l'œil gauche de l'image `lena.png`.

Exercice 6.2 (Compression d'image)

L'objectif de la compression est de réduire la quantité de mémoire nécessaire pour le stockage d'une image ou de manière équivalente de réduire le temps de transmission de celle-ci. Cette compression peut soit conserver l'image intacte, on parle alors de compression sans perte, soit autoriser une dégradation de l'image pour diminuer encore l'empreinte mémoire, on parle ici de compression avec perte. La première méthode est limitée à des taux de compressions (rapport entre la taille mémoire originale et la taille comprimée) de l'ordre de 3 tandis que la seconde permet des facteurs beaucoup plus grands au prix de cette dégradation de l'image.

Parmi les méthodes de compression avec perte, la méthode la plus simple est le changement de résolution en compressant l'information associée à un motif de $N \times N$ pixels en un seul en prenant la

valeur moyenne de l'information de couleur portée par l'ensemble de ces N^2 pixels (figure 2). Le taux de compression est dans ce cas de N^2.

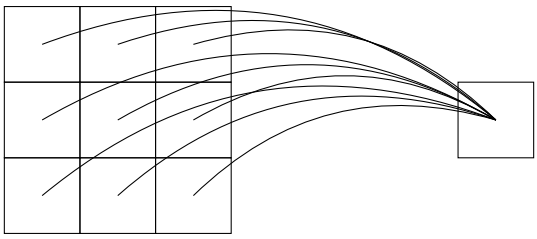

FIGURE 2 – Principe de la compression d'image par changement de résolution.

1. Définir une fonction `Resolution` qui prend comme arguments le tableau associé à une image et la dimension N du motif associé au taux de compression et qui renvoie le tableau de valeurs associé à l'image compressée. Cette fonction doit pouvoir compresser aussi bien des images en niveaux de gris que des images en couleurs (RGB ou RGBA).
2. Définir une procédure `AffichageDuo` qui prend comme argument un tuple de deux tableaux d'images et qui affiche à l'écran les deux images cote à cote. L'utiliser pour les images de la liste `images` pour afficher l'image initiale et sa version compressée avec $N = 10$.
3. Définir une procédure `CompressionImage` qui prend comme arguments le tableau associé à une image, le taux de compression minimal souhaité et le nom du fichier résultant et qui doit successivement compresser l'image en appelant la fonction `Resolution` avec la valeur de N permettant d'obtenir le taux de compression minimal souhaité et l'enregistrer au format `*.png`.
4. Tester cette procédure avec les images de la liste `images` avec différents taux de compression. En affichant dans chaque cas l'image originale et sa transformée compressée, décrire succinctement comment sont compressés les bords des images.

Exercice 6.3 (Filtrage spatial)
Pour limiter le bruit d'une image, on lui applique généralement un filtre passe-bas. Parmi les différents moyens de filtrage, il est possible d'utiliser une convolution discrète d'une cellule de $N \times N$ pixels dite de taille $N = 2\,k + 1$ ($k \in \mathbb{N}$), centrée autour de chaque pixel de position (x, y) et telle que la luminance de l'image filtrée, c'est-à-dire l'intensité lumineuse de chaque pixel, soit :

$$L_{\mathcal{F}}(x, y) = (L * \mathcal{F})(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} L(x - i, y - j) \times \mathcal{F}(i, j)$$

où L est la fonction de luminance associée à chaque pixel de position (x, y) et où $\mathcal{F}$ est la fonction définissant le filtre. Parmi les filtres exploitant cette convolution, on distinguera :

— le filtre de moyenne de taille N :

$$\mathcal{F}(i, j) = \frac{1}{N^2}$$

— le filtre gaussien (ou de Gauss) de taille $N = 6\sigma + 1$:

$$\mathcal{F}(i, j) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-\left(i^2 + j^2\right)}{\sigma^2}\right)$$

Un autre moyen de filtrage très utilisé est le filtre médian qui associe à chaque pixel de position (x, y) la valeur médiane d'une cellule de $N \times N$ pixels avec $N = 2k + 1$ ($k \in \mathbb{N}$), telle que la luminance de l'image filtrée soit :

$$L_{\text{med}}(x, y) = \text{médiane}\left(\{L(i, j) \mid i \in [x - k; x + k], j \in [y - k; y + k]\}\right)$$

où la fonction médiane peut être appelée avec la commande `median` de la bibliothèque `numpy`.

1. Définir deux fonctions `FMoy` et `FGauss`, respectivement associées aux filtres de moyenne et de Gauss et qui prennent comme arguments la taille du filtre N et les coordonnées (i, j) du point et renvoient la valeur du filtre $\mathcal{F}(i, j)$.
2. Définir une fonction `FiltrageConvolution` qui prend comme arguments le tableau de luminance L associé à une image, la taille du filtre N et le nom de la fonction associée au filtre choisi et qui renvoie le tableau de luminance filtrée $L_{\mathcal{F}}$.

3. Définir une fonction `FiltrageMedian` qui prend comme arguments le tableau de luminance L associé à une image, la taille du filtre N et qui renvoie le tableau de luminance L_{med}.
4. Appliquer les différents filtres pour les tailles $N = 3$ et $N = 5$ aux images `bateau.png` et `phare.png`. En déduire quel filtre est le plus efficace et discuter le choix de la taille du filtre.

— Complément —

pour les plus rapides

Exercice 6.4 (Optimisation du contraste)
L'intensité lumineuse de chaque pixel d'une image en niveaux de gris est appelée luminance, notée L. Le contraste est un caractère de la répartition lumineuse entre les points d'une image. Le contraste d'une image de $n \times p$ pixels peut être défini comme :

— l'écart-type des variations des niveaux de luminance :

$$C = \sqrt{\frac{1}{n \times p} \sum_{x=0}^{n-1} \sum_{y=0}^{p-1} (L(x,y) - B)^2}$$

avec B la brillance, définie comme la valeur moyenne de la luminance d'une image :

$$B = \frac{1}{n \times p} \sum_{x=0}^{n-1} \sum_{y=0}^{p-1} L(x,y)$$

— la variation entre les niveaux minimum et maximum de luminance (contraste de Michelson) :

$$C_M = \frac{L_{\max} - L_{\min}}{L_{\max} + L_{\min}} \qquad \text{avec} \quad L_{\min} = \min(L) \quad \text{et} \quad L_{\max} = \max(L).$$

Pour améliorer le contraste d'une image, il est nécessaire de modifier la distribution de luminance de l'image. La méthode la plus robuste, dite « d'équalisation d'histogramme », consiste en la création d'une densité cumulative

$$\forall k \in [\![0; 255]\!], \qquad \mathcal{P}(k) = \sum_{i=0}^{k} \frac{h[i]}{n \times p}$$

à partir de l'histogramme h mis sous forme de tableau $h[i]$ contenant le nombre de pixels compris dans chacun des 256 niveaux de luminance ($i \in [\![0; 255]\!]$, codage en 8 bits) de l'image de $n \times p$ pixels. Dans l'expression de la fonction $\mathcal{P}$, le terme $h[i]/(n \times p)$ donne la proportion des pixels qui sont dans la $(i+1)$-ème classe de luminance. La fonction $\mathcal{P}$ renvoie donc la somme cumulée de la proportion des pixels compris entre la 1re et la $(k+1)$-ème classe de luminance ($k \in [\![0; 255]\!]$), telle que $\mathcal{P}(255) = 1$. En accord avec le caractère discret de l'histogramme, la luminance de l'image optimisée en contraste sera définie par :

$$L_{\mathcal{P}}(x,y) = \lfloor 255 \times \mathcal{P}(L(x,y)) \rfloor$$

1. Définir une fonction `BrillanceContraste` qui prend comme argument un tableau de luminance L et qui renvoie les valeurs de brillance B, de contraste C et de contraste de Michelson C_M en utilisant soit les méthodes associées aux listes, soit celles associées au type `ndarray`.
2. Définir une fonction `OptimContraste` qui prend comme argument le tableau de luminance L associé à une image et qui renvoie le tableau de luminance optimisée $L_{\mathcal{P}}$. Pour réaliser le tableau $h[i]$ des 256 niveaux de luminance, on utilisera la fonction `histogram` de la bibliothèque `numpy` sous la forme :

   ```
   hist,bin_edges = np.histogram(L,bins=256,range=[0,255])
   ```

   telle que `hist` soit le tableau de 256 valeurs (`bins=256`), comprises entre 0 et 255 (`range=[0,255]`) correspondant aux 256 intervalles du tableau $h[i]$ de luminance `L` de l'image considérée et où `bin_edges` est un tableau contenant les $256+1$ bornes de ces mêmes intervalles (non utilisé ensuite).
3. Appliquer la fonction `OptimContraste` à l'image `lena.png`.

* *
*