Traitement d'image

Une image numérique peut être représentée par une matrice dans laquelle chaque case représente un pixel (pour *picture element*), unité minimale adressable par le contrôleur vidéo et supposé de couleur uniforme. Le *format* de l'image (png, jpeg, etc) désigne la manière dont cette matrice est représentée en mémoire.

- Pour une image en noir et blanc, chaque pixel ne peut prendre que deux valeurs, ce qui permet de l'encoder sur un seul bit ;
- pour une image en niveau de gris, chaque teinte de gris (en passant du noir au blanc) est encodée sur un octet, ce qui permet de représenter $2^8 = 256$ nuances de gris ;
- pour une image en couleur, chaque pixel et encodé sur trois octets, chaque octet représentant une nuance de couleur. Par exemple, dans le cas du format RGB (*Red*, *Green*, *Blue*), le premier octet dose la quantité de rouge, le second la quantité de vert et le troisième la quantité de bleu pour une synthèse additive ;
- enfin, les pixels d'une image en couleur peuvent être encodés par quatre octets au format RGBA (A pour *alpha*), le quatrième octet codant la transparence du pixel.

Manipulation d'image en Python

Durant ce TP nous aurons besoin de trois modules : numpy pour la manipulation des matrices, matplotlib.pyplot pour la visualisation des images et enfin imageio pour la conversion image/matrice. Commençons donc par les importer :

```
import numpy as np
import matplotlib.pyplot as plt
import imageio as io
```

– La fonction `io.imread`(*source*) permet de convertir une image au format courant (png, jpeg, etc) en un tableau numpy. Elle prend en argument une chaîne de caractère décrivant un fichier présent sur votre ordinateur ou l'adresse http d'une image présente sur le net.

– La fonction `io.imwrite`(*cible*, *tab*) permet au contraire de convertir un tableau numpy (l'argument *tab*) en un fichier image (décrit par la chaîne de caractères *cible*).

– La fonction `plt.imshow`(*tab*) permet de visualiser l'image décrite par un tableau. À noter, pour visualiser une image en niveau de gris il faut préciser l'échelle chromatique à utiliser, ici : `plt.imshow`(*tab*, `cmap='gray'`).

Le script qui suit doit vous permettre de récupérer sur le net l'image qui nous servira de modèle, la visualiser puis l'enregistrer dans le répertoire courant.

```
im = io.imread('http://info-llg.fr/commun-mp/images/picasso.png')
plt.imshow(im)
plt.show()
io.imwrite('./picasso.png', im)
```

Ainsi définie `im` est tableau numpy dont les éléments sont de type *np.uint8* (entiers non signés représentés sur 8 bits, autrement dit un entier compris entre 0 et 255). La méthode shape donne les dimensions de ce tableau : `im.shape = (n, p)` pour une image en niveau de gris, `im.shape = (n, p, 3)` pour une image couleur, `im.shape = (n, p, 4)` pour une image avec composante alpha. L'entier n est le nombre de lignes de l'image et p le nombre de colonnes, le pixel de coordonnées $(0, 0)$ est situé en haut à gauche de l'image.

Remarque. En réalité, le format de l'objet importé par la commande `io.imread` est plus complexe qu'un simple tableau numpy (il contient des informations complémentaires sur le contenu de l'image) et la manipulation de ces images s'en trouve ralentie. Pour accélérer la vitesse d'exécution des fonctions que vous aurez à écrite, vous aurez intérêt à transformer ces dernières en simples tableaux numpy en rajoutant au code précédent la commande :

```
im = np.array(im)
```

1. Fonctions élémentaires

Question 1. Rédiger une fonction `symetrie` qui prend en argument une image et retourne une nouvelle image, obtenue par symétrie autour d'un axe vertical passant par le milieu de l'image.
Indication : la fonction `np.zeros_like(m)` crée un tableau nul de même type et de mêmes dimensions que le tableau `m`.

Question 2. Rédiger une fonction `rotation` qui prend en argument une image et retourne une nouvelle image, obtenue par rotation d'un angle $\pi/2$ autour du centre de l'image.
Indication : la fonction `np.zeros((n, p, s), dtype=np.uint8` crée un tableau vide de dimensions $n \times p \times s$ dont les éléments sont de type *np.uint8*.

Question 3. Rédiger une fonction `negatif` qui prend en argument une image et retourne son négatif, c'est-à-dire l'image dans laquelle chaque composante de couleur de chaque pixel est remplacée par sa valeur complémentaire dans l'intervalle $[0, 255]$.

Conversion en niveau de gris

Pour convertir une image couleur en niveau de gris, un pixel représenté par ses composantes (r, g, b) doit être remplacé par un pixel à une seule composante $y \in [\![0, 255]\!]$ appelée *luminance*. Cette quantité, qui traduit la sensation visuelle de luminosité, dépend de manière inégale des trois composantes RGB. La formule communément recommandée pour cette conversion est la suivante :

$$y = 0,2126\, r + 0,7152\, g + 0,0722\, b \qquad (y \text{ étant arrondi à l'entier le plus proche})$$

(pour un œil humain le vert parait plus lumineux que le rouge, lui-même plus lumineux que le bleu).

Question 4. Rédiger une fonction `niveaudegris` qui prend an argument une image couleur et retourne une nouvelle image convertie en niveau de gris.

2. Traitement d'image

La pluspart des filtres de traitement d'images utilisent une convolution par un *masque* pour réaliser une modification. Ces masques sont des matrices de petites tailles, en général 3×3 (taille que nous allons considérer par la suite) ou 5×5 :

$$\mathrm{C} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

Le filtre associé à ce masque transforme la matrice $\mathrm{M} = (m_{ij})$, associée à une certaine image, en la matrice $\mathrm{M} \otimes \mathrm{C} = (m^*_{ij})$ par une opération appelée *produit de convolution* et définie par la relation : $\forall i, j$,

$$m^*_{ij} = c_{11} m_{i-1,j-1} + c_{12} m_{i-1,j} + c_{13} m_{i-1,j+1} + c_{21} m_{i,j-1} + c_{22} m_{i,j} + c_{23} m_{i,j+1} + c_{31} m_{i+1,j-1} + c_{32} m_{i+1,j} + c_{33} m_{i+1,j+1}$$

Dans le cas d'une image en couleur, on peut choisir d'appliquer le même masque à chacune des trois composantes RGB de l'image ou leur appliquer des masques différenciés, en fonction de l'effet désiré. Dans la suite de ce sujet, et dans un but simplificateur, nous appliquerons le même masque à chacune des trois composantes de couleur.

Remarque. Lorsque le pixel initial est sur un bord, une partie de la convolution porte en dehors des limites de l'image. Là encore, nous conviendrons pour simplifier de laisser inchangés ces pixels.

Question 5. Rédiger une fonction `convolution` qui prend en arguments deux matrices M (associée à une image) et C et retourne la matrice $\mathrm{M} \otimes \mathrm{C}$. Les éléments de la matrice C seront *a priori* de type *float* tandis que ceux des matrices M et $\mathrm{M} \otimes \mathrm{C}$ seront de type *np.uint8*. Il conviendra donc, lorsque $m^*_{ij} < 0$ ou $m^*_{ij} > 255$ de remplacer les valeurs calculées par respectivement 0 et 255. Ne pas oublier non plus que les pixels d'une image en couleur possèdent trois composantes et que chacune de ces composantes doit être traitée.

Exemples de masques

Lissage Pour obtenir un effet de lissage on utilise le masque C_1 : chaque pixel est remplacé par la moyenne de lui-même et de ses huit voisins ; pour cette raison on parle de filtre *moyenneur*.

Augmentation du contraste Au contraire, pour augmenter le contraste on utilise le masque C_2.

Repoussage Enfin, le masque C_3 donne un effet de relief à l'image, appelé *repoussage*.

$$C_1 = \frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix} \qquad C_2 = \begin{pmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{pmatrix} \qquad C_3 = \begin{pmatrix}-2 & -1 & 0\\ -1 & 1 & 1\\ 0 & 1 & 2\end{pmatrix}$$

Figure 1 – Trois filtres classiques en traitement d'images.

Question 6. Rédiger trois fonctions `lissage`, `contraste` et `repoussage` qui prennent toutes trois en argument une image et retournent une nouvelle image obtenue en appliquant respectivement un effet de lissage, d'augmentation de contraste et de repoussage, et observer le résultat sur l'image test.

3. Détection de contours

On détecte les contours dans une image en repérant les pixels qui correspondent à un changement brutal d'intensité lumineuse. Pour réaliser ceci, nous allons partir d'une image en niveau de gris : ainsi, la matrice associée à l'image contient pour chaque pixel (x,y) la luminance $\mathrm{I}(x,y)$ de ce dernier.

Pour trouver ces points nous allons définir l'analogue discret du gradient de l'intensité de chaque pixel :

$$\operatorname{grad}\mathrm{I}(x,y) = \frac{\partial \mathrm{I}}{\partial x}(x,y)\vec{e}_x + \frac{\partial \mathrm{I}}{\partial y}(x,y)\vec{e}_y.$$

Ce vecteur indique la direction de la plus forte variation du clair vers le sombre, et son module l'intensité de la variation. Ainsi, on peut penser que les contours correspondent aux pixels pour lesquels $\|\operatorname{grad}\mathrm{I}(x,y)\|^2$ est important : on conviendra de colorer en blanc (luminance = 255) les pixels pour lesquels cette quantité dépasse un certain seuil, et en noir (luminance = 0) les autres.

Définition du gradient discret

Commençons par effectuer un calcul mathématique. Nous avons :

$$\mathrm{I}(x+h,y) = \mathrm{I}(x,y) + h\frac{\partial \mathrm{I}}{\partial x}(x,y) + \frac{h^2}{2}\frac{\partial^2 \mathrm{I}}{\partial x^2}(x,y) + \mathrm{o}(h^2) \quad \text{et} \quad \mathrm{I}(x-h,y) = \mathrm{I}(x,y) - h\frac{\partial \mathrm{I}}{\partial x}(x,y) + \frac{h^2}{2}\frac{\partial^2 \mathrm{I}}{\partial x^2}(x,y) + \mathrm{o}(h^2)$$

donc $\mathrm{I}(x+h,y) - \mathrm{I}(x-h,y) = 2h\dfrac{\partial \mathrm{I}}{\partial x}(x,y) + \mathrm{o}(h^2)$.

Ce calcul nous suggère de poser $\mathrm{G}_x(x,y) = \mathrm{I}(x+1,y) - \mathrm{I}(x-1,y)$ et (pour les mêmes raisons) $\mathrm{G}_y(x,y) = \mathrm{I}(x,y+1) - \mathrm{I}(x,y-1)$ et à définir le gradient discret par : $\operatorname{grad}\mathrm{I}(x,y) = \mathrm{G}_x(x,y)\vec{e}_x + \mathrm{G}_y(x,y)\vec{e}_y$.

On peut observer que le calcul de G_x et de G_y correspond à l'application des masques respectifs :

$$\begin{pmatrix}0 & 0 & 0\\ -1 & 0 & 1\\ 0 & 0 & 0\end{pmatrix} \quad \text{et} \quad \begin{pmatrix}0 & -1 & 0\\ 0 & 0 & 0\\ 0 & 1 & 0\end{pmatrix}.$$

Cependant, l'expérience montre que cette formule reste un peu trop dépendante au bruit d'une image, et qu'on obtient de meilleurs résultats en moyennant autour du point, c'est-à-dire en utilisant les masques

$$\begin{pmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{pmatrix} \quad \text{et} \quad \begin{pmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{pmatrix} \quad \text{(filtre de Sobel)}.$$

Question 7. Rédiger une fonction `gradient` qui prend en argument la matrice des luminances des pixels d'une image et qui retourne la matrice (de type `float`) des normes des gradients discrets calculés à l'aide du filtre de Sobel.

Question 8. En déduire une fonction `contour` qui prend en arguments une image et un seuil et qui retourne une image en noir et blanc dans laquelle tout pixel dont le gradient est inférieur au seuil est coloré en blanc et tout pixel dont le gradient est supérieur au seuil coloré en noir.

4. Stéganographie d'une image

La *stéganographie* est l'art de la dissimulation. Nous allons dans cette dernière partie étudier un procédé permettant de cacher une image au sein d'une autre image.

Dans une image, chaque composante de couleur de chaque pixel est représentée par un entier codé sur huit bits :

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Les quatre premiers bits, dits *de poids forts*, ont plus d'importance (plus de poids) que les quatre bits suivants, dits *de poids faible*, et l'expérience montre qu'on ne change guère l'image en ne gardant que les quatre bits de poids forts :

a_0	a_1	a_2	a_3	0	0	0	0

Dès lors, les quatre bits de poids faibles ainsi libérés vont pouvoir nous servir à cacher les quatre bits de poids forts d'une seconde image :

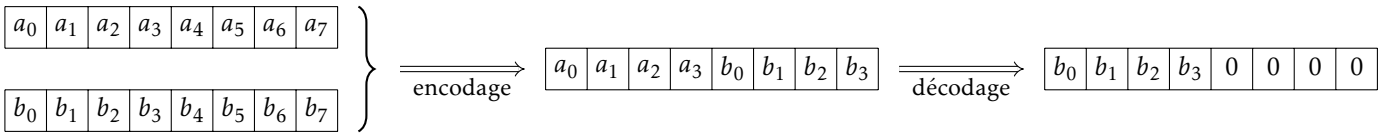

FIGURE 2 – Camouflage d'une image b au sein d'une image a.

Question 9. Rédiger une fonction encodage qui prend en arguments deux images de mêmes tailles a et b en arguments et qui retourne une image dans laquelle b a été camouflée au sein de a suivant le procédé décrit ci-dessus.

L'utiliser pour cacher l'image `'http://info-llg.fr/commun-mp/images/matisse.png'` au sein de l'image de test.

Question 10. Rédiger enfin une fonction decodage qui prend en argument une image qui a été encodée suivant le procédé ci-dessus et qui retourne l'image cachée au sein de celle-ci.

Il vous reste à découvrir l'image que j'ai cachée au sein de l'image `'http://info-llg.fr/commun-mp/images/llg.png'` et vous en aurez terminé.