Traitement élémentaire des images

1 Introduction aux tableaux numpy

Exécutez le code suivant :

```
import numpy as np
T = [[0,1],[2,3]] # des tableaux dont les éléments sont des tableaux.
U = [[4,5],[6,7]]
print(T+U) # concaténation
print(2*T) # repetition
```

Exécutez maintenant :

```
T = np.array(T) # conversion en tableau numpy.
U = np.array(U)
print(T+U) # addition terme à terme.
print(2*T) # multiplication de tous les éléments par 2.
```

Remarquez qu'avec des tableaux numpy, le produit T*U ne produit pas d'erreur et a bien un sens ici : celui du produit terme à terme des tableaux, comme pour l'addition.

Attention, pour avoir un sens, les opérations précédentes doivent se faire sur des tableaux ayant même taille :

```
T = np.array([[0,1],[2,3],[4,5]])
U = np.array([[4,5],[6,7]])
print(T+U) #addition d'un tableau 3*2 avec un tableau 2*2: erreur.
```

Pour copier un tableau dans une nouvelle variable, il faut utiliser la fonction copy de Numpy.

```
T = np.array([[0,1],[2,3],[4,5]])
T2 = T # si un élément de T ou T2 est modifié
# alors cette modification sera aussi appliquée à l'autre tableau
T3 = np.copy(T) # si un élément de T ou T3 est modifié
# alors cette modification ne sera pas appliquée à l'autre tableau
T[0][0] = 6
print(T)
print(T2)
print(T3)
```

2 Décodage d'une image en couleur

Une image en couleur RGB a été sauvegardée dans un fichier `mystere.txt` sous la forme d'une chaine de caractères. (on pourra consulter le contenu de ce fichier avec un simple editeur de texte). On cherche à afficher l'image correspondante. Le codage a été réalisé en utilisant la convention suivante :

- Chaque pixel est codé par ses trois intensités de couleur exprimées en décimal et séparées par un espace
- Les pixels formant une ligne sont séparés par un point-virgule `;`
- Les lignes sont séparées par un retour à la ligne

Evidemment, stocker une image sous forme de texte n'est pas du tout optimal (on procède ainsi uniquement pour l'ergonomie du TP).

Question : mettre en place un programme qui charge en mémoire vive le fichier `mystere.txt`, l'exploite pour obtenir un tableau de type `array` contenant des `uint8`, correspondant à l'image recherchée (l'image est de forme carrée). Puis afficher l'image. Question bonus : transformer l'image en version 1up.

3 Décomposition d'une image en pixels

Les images qu'on va utiliser sont des images dites matricielles. Elles sont composées d'une matrice (tableau) de points colorés appelés pixels. Les formats d'images matricielles qu'on utilisera seront BMP, JPG ou JPG en « couleurs vraies ». Chaque pixel est un triplet de nombres entre 0 et 255 : un nombre pour chaque couleur primaire rouge, vert, bleu. Un tel nombre est représentable sur 8 bits et s'appelle un octet. Il y a donc $2^{24} = 16777216$ couleurs possibles. On utilise ici la synthèse additive des couleurs : le triplet (0 ; 0 ; 0) correspond à un pixel noir alors qu'un pixel blanc est donné par (255 ; 255 ; 255). Un pixel « pur rouge » est codé par (255 ; 0 ; 0).

Voici un code permettant d'ouvrir une image à l'aide du module `PIL` et de la stocker dans un tableau numpy :

```
from PIL import Image #importation du sous-module Image du module PIL
im = Image.open("image.jpg") #ouverture d'une image au format jpg dans Python.
tab = np.array(im)
```

Nous allons travailler avec l'image "`lisboa.jpg`".

Question 1. Ouvrez l'image et chargez-la. Exécutez et commentez également le code suivant :

```
print(tab) #
print(im.size) #
print(len(tab)) #
print(len(tab[0])) #
print(len(tab[0][0])) #
print(tab.shape) #
```

On vérifiera que l'image obtenue a bien un profil RGB, et on appellera le professeur dans le cas contraire.

Voici un code permettant de créer une image à partir d'un tableau numpy :

```
nouvelle_image = Image.fromarray(tab)
nouvelle_image.save("nom_image.jpg") # pour l'enregistrer au format voulu
```

Pour afficher l'image dans la console, il suffit d'appeler l'image en mémoire : `nouvelle_image`
Pour afficher l'image dans un graphe ou une fenêtre, voir l'annexe du poly de cours.

Question 2. Dans les questions suivantes, nous serons amenés à ouvrir des images contenues dans un fichier dont le nom est de la forme "nom.jpg" et à le renommer sous la forme "nom_transformee.jpg".

Écrire une fonction `renomme(fichier,transformee)` qui prend en argument deux chaînes de caractères, la première donnant le nom du fichier de départ, la seconde le nom de la transformation à appliquer, et qui renvoie le nom du nouveau fichier sous forme d'une chaîne de caractères.

Par exemple, `renomme("toto.jpg","bleu")` renverra `"toto_bleu.jpg"`

Question 3. Écrire une fonction qui prend en argument une chaîne de caractères, indiquant le nom de l'image à traiter, et qui crée une nouvelle image ne contenant que les composantes rouges de l'image à traiter. Le nom de l'image renvoyée sera constitué de la chaîne de caractères de départ, du caractère souligné et de la chaîne de caractères "red" ainsi que de la même extension : ainsi "toto.jpg" sera traité en "toto_red.jpg"

Question 4. Une image négative est une image dont les couleurs ont été inversées par rapport à l'originale ; par exemple le rouge devient cyan, le vert devient magenta, le bleu devient jaune et inversement. Les régions sombres deviennent claires, le noir devient blanc.

Les couleurs primaires étant codées sur un octet (256 nuances), la couleur opposée à x se calcule par : $255 - x$

Écrire une fonction `inverse` ayant pour argument une chaîne de caractères étant le nom du fichier image qui crée l'image négative dont le nom est fichier_negatif.jpg.

Question 5. En augmentant la taille d'image, créer un cadre noir de 5 pixels autour de l'image. Pour cela créer une fonction `cadre_noir` ayant pour argument une chaîne de caractères étant le nom du fichier image et un entier indiquant l'épaisseur en pixels du cadre. Cette fonction doit créer une nouvelle image dont le nom est `fichier_cadre.jpg`.

2. De la couleur vers les niveaux de gris

Dans une image en niveaux de gris, chaque pixel est noir, blanc, ou a un niveau de gris entre les deux. Cela signifie que les trois composantes R, V, B ont la même valeur.

Les niveaux de rouge, vert et bleu étant identiques, les informations sont redondantes si l'on sait que l'on a une image en niveau de gris. Il est possible de sauvegarder la matrice des pixels non pas en associant à chaque pixel un triplet de niveau (R,V,B) mais une valeur unique égale au niveau de gris

Question 6. Pour transformer une image en couleur en image en niveaux de gris, nous allons dans un premier temps nous contenter de calculer la moyenne pixel par pixel des trois composantes du triplet (R,V,B).

Écrire une fonction `niveau_gris` ayant pour argument une chaîne de caractères étant le nom du fichier de l'image initiale. Cette fonction doit créer une nouvelle image en niveau de gris à partir d'une matrice ne contenant que le niveau de gris et pas les niveaux R, V, B. Le nom du fichier résultant sera *fichier_gris.jpg*

Question 7. Écrire une fonction `histogramme` ayant pour argument une chaîne de caractères étant le nom du fichier d'une image dont on souhaite tracer l'histogramme d'intensité lumineuse. Cette fonction doit afficher l'histogramme et renvoyer une liste ou un array contenant les compteurs de chaque intensité lumineuse.

Question 8. L'œil est plus sensible à certaines couleurs qu'à d'autres. Le vert (pur), par exemple, paraît plus clair que le bleu (pur). Pour tenir compte de cette sensibilité dans la transformation d'une image couleur en une image en niveaux de gris, on ne prend généralement pas la moyenne arithmétique des intensités de couleurs fondamentales, mais une moyenne pondérée. La formule standard donnant le niveau de gris en fonction des trois composantes est :

$$\text{gris} = \lfloor\ 0.299 * \text{rouge} + 0.587 * \text{vert} + 0.114 * \text{bleu}\ \rfloor$$

Écrire une fonction `niveau_gris_pondere` ayant pour argument une chaîne de caractères étant le nom du fichier de l'image initiale. Cette fonction doit créer une nouvelle image en niveau de gris à partir d'une matrice contenant les niveaux R, V, B. Le nom du fichier résultant sera *fichier_gris_pondere.jpg*

3. Modifications de la luminosité

Lorsqu'une image est trop sombre (ou trop claire), nous pouvons souhaiter modifier la luminosité de celle-ci. Nous allons travailler sur l'image en niveaux de gros créée à la question précédente et étudier deux solutions possibles.

Question 9. Dans un premier temps nous allons décaler la luminosité de tous les points d'un même niveau. Cette transformation ne sera toutefois pas bijective.

Écrire une fonction `change_luminosite`, qui prend en argument une chaine de caractère étant le nom du fichier image à traiter, image en niveau de gris décrite par une matrice dont chaque pixel est représenté seulement par le niveau de gris et un entier entre -255 et 255, valeur du décalage du niveau de gris. Cette fonction doit créer une nouvelle image *fichier_lum.jpg*

On pourra utiliser les fonctions `min` et `max` pour s'assurer que les couleurs ne sortent pas de l'intervale $[0; 255]$.

Question 10. Afin de disposer d'une transformation se rapprochant davantage d'une transformation bijective et, "brûlant" donc moins les images, on introduit la fonction suivante, appelée fonction d'exposition de paramètre γ

$$f(i) = \min\left(255, \left\lfloor 255 * \exp\left(\ln\left(\frac{i}{255}\right) * \frac{1}{\gamma}\right)\right\rfloor\right) \text{ si } i > 0 \text{ et 0 si i=0}$$

Lorsque $\gamma > 1$ on augmente la luminosité, et lorsque $\gamma < 1$ on la diminue.

Écrire une fonction `change_exposition`, qui prend en argument une chaine de caractère étant le nom du fichier image à traiter, image en niveau de gris décrite par une matrice dont chaque pixel est représenté seulement par le niveau de gris et un paramètre gamma, comme décrit ici. Cette fonction doit créer une nouvelle image *fichier_exposition.jpg*

4. Seuillage et détection de contours

L'objectif de cette partie est de compter le nombre de colonies de bactéries dans l'image `"bacteries.png"`. Pour cela, il faudra d'abord détecter les contours des colonies (par produit de convolution puis seuillage). Puis un dénombrement des ensembles connexes de pixels dans l'image permet de connaitre le nombre de colonies.

Question 11. Écrire une fonction `seuillage` : cette fonction prend une chaîne de caractères étant le nom du fichier en niveaux de gris à traiter et un entier *seuil*. Tous les pixels de luminosité inférieure à *seuil* deviennent noirs ; tous les autres blancs.

Cette fonction doit créer une nouvelle image *fichier_seuillage.png*

Question 12. Écrire une fonction `contour` : cette fonction prend une chaîne de caractères étant le nom du fichier en niveaux de gris à traiter. Elle applique un produit de convolution avec la matrice ci-dessous. Le résultat attendu est un tableau représentatif d'une image décrivant les contours de chaque colonie en gris clair sur fond noir.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Question 13. On suppose dans cette question que l'on dispose d'une image uniquement en noir et blanc : ses pixels sont donc des entiers, égaux à 0 ou à 255.

L'objectif de la question est de déterminer les coordonnées des pixels d'une composante connexe associée à un pixel `depart` donné, à savoir la liste des points `pix` telle qu'il existe un chemin (composé de déplacements verticaux et horizontaux) qui passe exclusivement par des pixels noirs reliant `depart` et `pix`.

On se propose d'écrire une fonction traitant ce problème de la manière suivante (il s'agit en réalité d'une mauvaise version d'un parcours de graphe en largeur que l'on étudiera ultérieurement) :

- on initialise une liste `CC` et une liste `L` à `[]` si image[depart]=0 et à [depart] sinon
- Tant que L n'est pas vide on répète les opérations suivantes :
  - on enlève le premier élément de `L` et on le stocke dans une variable `pix`
  - Si image[pix]=0, pix n'appartient pas à la composante connexe, on ne fait rien
  - Si image[pix]=255, pix appartient à la composante connexe de depart, on considère alors deux cas :

— Si pix est dans CC, il a déjà été parcouru, on ne fait rien
— Sinon on ajoute pix à CC et on ajoute également les voisins de pix à la liste L des candidats. On prendra garde à ne pas sortir des limites de l'image.

Question 14. Utiliser les fonctions ci-dessus pour déterminer le nombre de colonies dans l'image fournie. Pour cela, on traitera cette image de sorte à obtenir une image en noir et blanc, à l'aide des fonctions précédentes. On fera le décompte du nombre de colonies de bactéries, qui sera égal au nombre de composantes connexes noires dans l'image obtenue après traitement.