Informatique - TP 9

Traitement d'images

M. Marmorat, M. Morel

21 Mars 2024

Le but de ce TP est de manipuler des images. On verra qu'il est pratique de représenter une image comme un tableau Numpy.

1 Introduction

Les images seront codées sous la forme d'un tableau T à 3 dimensions. Si $n \times p$ est la dimension de l'image, on accède au pixel de coordonnées (i, j) (avec $i \in [\![0, n-1]\!]$ et $j \in [\![0, p-1]\!]$) par $T[i][j]$. $T[i][j]$ est lui-même un tableau (unidimensionnel) à 3 entrées représentant les nuances de rouge, vert et bleu dans chaque pixel. Ces nuances sont codées par des entiers dans l'intervalle $[\![0, 255]\!]$.

Par exemple, le tableau $[0, 0, 0]$ représente le blanc, $[255, 0, 0]$ le rouge, $[0, 255, 0]$ le vert, $[0, 0, 255]$ le bleu et $[255, 255, 255]$ le noir. La commande $T[i][j][c] = x$ attribue donc au pixel (i, j) la valeur x pour la couleur correspondant à c (rouge pour 0, vert pour 1 et bleu pour 2).

1.1 Chargement, visualisation et sauvegarde d'image

Pour les tests, on téléchargera le fichier *'panda.png'* disponible sur cahier-de-prepa > Informatique > TPs > Données.

On commence par créer un dossier *TP11* dans lequel on met tous les fichiers qu'on manipule dans ce TP (les images et le fichier Python). On utilise les bibliothèques suivantes :

```
1 import matplotlib.pyplot as plt     #pour visualiser les images
2 import matplotlib.image as mpimg    #pour charger et sauvegarder les
     images
3 import numpy as np      #pour le traitement des tableaux
```

Pour charger l'image :

```
1 image = mpimg.imread("panda.png")
```

En cas d'erreur, il faut aller dans les paramètres de l'environnement de programmation et demander que le répertoire courant soit changé lors de l'exécution du fichier.

La variable *image* est le tableau à 3 dimensions codant l'image. Si ses entrées sont des flottants plutôt que des entiers entre 0 et 255, on change l'encodage :

```
1 if image.dtype == np.float32:
2     image = np.uint8(255*image)
```

Pour visualiser l'image :

```
plt.imshow(image)
plt.show()
```

L'image doit apparaître, avec des axes de coordonnées pour repérer les pixels.
Pour sauvegarder l'image :

```
plt.imsave(nom_fichier,image)
```

L'image codée par le tableau *image* est sauvegardée dans le dossier sous le nom spécifié.

Exercice 1 Utiliser les commandes précédentes pour charger l'image *"panda.png"*, la visualiser et en créer une copie nommée *"jumeau.png"*.

1.2 Deux commandes utiles

On aura besoin de créer une image *vide* (c'est-à-dire dont tous les pixels sont noirs). On écrit :

```
image = np.zeros((n,p,3),dtype=np.uint8)
```

avec n et p les dimensions de l'image.
On rappelle enfin qu'on peut accéder aux dimensions d'un tableau multidimensionnel *image* par la commande shape :

```
n,p = image.shape[0],image.shape[1]
```

Ainsi, n et p seront le nombre de pixels en abscisse et en ordonnée de l'image codée par le tableau.

2 Gestion des couleurs

Exercice 2 Créer une image représentant le drapeau fran cais (format 300×200).

On pensera à sauvegarder cette image et les autres dans notre dossier.

Exercice 3 Écrire une fonction `negatif`(image), renvoyant le négatif de l'image passée en argument : les nuances des trois couleurs de chaque pixel sont remplacées par leur complément à 255.

Exercice 4 Écrire une fonction `nuances_gris`(image), renvoyant l'image en nuances de gris. Pour chaque pixel, on fait la moyenne pondérée des trois couleurs avec les coefficients 0.2126, 0.7152 et 0.0722 (*recommandation 709*) et on inscrit cette moyenne pour les trois couleurs du nouveau pixel.

3 Manipulation de l'image

Exercice 5 Écrire une fonction `miroir`(image) renvoyant l'image inversée selon la verticale, de l'image passée en argument.

Exercice 6 Écrire une fonction `rotation`(image) renvoyant l'image obtenue par rotation d'angle 90 degrés (sens trigonométrique), de l'image passée en argument.

Exercice 7 Écrire une fonction `quart`(image) renvoyant la même image, avec dimensions moitié en hauteur et largeur. On gardera un pixel sur deux, en hauteur et en largeur.

Exercice 8 Écrire une fonction `quatre`(image) renvoyant la même image, avec dimensions doublées en hauteur et en largeur. Chaque pixel sera dédoublé en hauteur et en largeur.

4 Algorithme de détection des contours

(*d'après* http://www.tangentex.com/TraitementImages.htm)

On va maintenant réaliser un algorithme de détection des contours. Le but est de renvoyer une image en noir et blanc, avec du noir aux endroits où l'image a des contours. On procède de la fa con suivante :

- On convertit l'image de départ en nuance de gris.
- Pour chaque pixel qui n'est pas sur un des bords de l'image, on calcule à quel point les nuances de gris des pixels voisins sont *proches* (en un sens à préciser).
- On crée une image de même dimension, noire au début. On rend blanc chaque pixel qui a passé le test précédent.

Exercice 9

Écrire une fonction `voisins`(i,j,image) prenant en argument une image en nuances de gris et des coordonnées (i, j) – correspondant à un point qui n'est pas sur le bord de l'image – et renvoyant les nuances de gris de ce point et de ses 4 voisins directs (sous la forme d'un quintuplet de valeurs).

Attention ! les nuances de gris ne sont pas de type integer. *Il faudra les convertir.*

Exercice 10 Écrire une fonction `norme`(X) prenant en argument un quintuplet de valeurs et renvoyant une mesure de l'éloignement de ses valeurs.

Si $X = (x_0, x_1, x_2, x_3, x_4)$, on renvoie

$$\sqrt{(x_0 - x_1)^2 + (x_0 - x_2)^2 + (x_0 - x_3)^2 + (x_0 - x_4)^2}.$$

Exercice 11 Écrire une fonction `contour`(image,seuil) réalisant l'algorithme décrit en début de

partie. Pour le test du deuxième point, on dira que les nuances de gris des voisins sont proches si la norme correspondante est inférieure au seuil fixé.

Tester cette fonction avec un seuil fixé à 30.