Leçon d'informatique : Gestion d'image

S. Benlhajlahsen - PCSI$_1$

Sommaire

Extrait du programme :

Thèmes	Exemples d'activité, au choix du professeur et non exigibles des étudiants. Commentaires
Matrices de pixels et images.	Algorithmes de rotation, de réduction ou d'agrandissement. Modification d'une image par convolution : flou, détection de contour, etc. *Les images servent de support à la présentation de manipulations de tableaux à deux dimensions.*

Préambule : Nous utiliserons dans ce cours deux bibliothèques :

- `import numpy as np` : pour la manipulation rapide de matrices.
- `import matplotlib.pyplot as plt` : pour l'affichage d'image et leur importation.

I Tableaux à deux dimensions

I.A présentation

À retenir : Un tableau à deux dimensions sera dans toute la suite une **liste de liste** `[`L_0`, `L_1`, ..., `L_n`]` où toutes les sous-listes sont de même taille p. On pourra se reporter à l'exemple de la figure 1.

```
T = [[9, 7, 5, 6, 2],
     [5, 0, 6, 1, 9],
     [1, 8, 4, 3, 0] ]
```

$$\begin{pmatrix} 9 & 7 & 5 & 6 & 2 \\ 5 & 0 & 6 & 1 & 9 \\ 1 & 8 & 4 & 3 & 0 \end{pmatrix}$$

$n = 3$ lignes

$p = 5$ colonnes

FIGURE 1 – À gauche, le tableau à deux dimensions (c'est une liste de taille 3 dont les sous-listes sont de tailles 5). À droite, la matrice associée à trois lignes et 5 colonnes

Dimensions du tableau : On accède aux nombres de lignes par `len(T)` (c'est la taille n de la liste principale.)
On accède aux nombre de colonnes par `len(T[0])` (c'est la taille p de chaque sous-liste. On a pris arbitrairement la première).
La méthode `shape` issue du module `numpy` donne aussi les dimensions de la matrice. Ainsi, `np.shape(T)` fournit le *tuple* `(3, 5)`. Bien évidemment, le nombre de lignes sera alors `np.shape(T)[0]` et le nombre de colonnes `np.shape(T)[1]`.

I.B manipulation

Indexations doubles : Un élément du tableau sera indexé par un indice de ligne (`i` dans la suite) et un indice de colonne (`j` dans la suite). Un élément du tableau `T` sera alors obtenu par `T[i][j]`.

Boucles imbriquées : Si on souhaite balayer tous les éléments de `T`, on peut exécuter deux boucles `for` imbriquées sur les indices de ligne et de colonnes (voir figure 2).

```
n, p = len(T), len(T[0]) # ou encore (n,p) = np.shape(T)
for i in range(n):
    for j in range(p):
        ... instructions sur T[i][j]
```

FIGURE 2 – Boucles imbriquées.

I.C tableaux `numpy`

Méthodes de bases : On rappelle que les tableaux `numpy` sont des tableaux homogènes et taille fixes. La méthode `np.array(T)` convertit automatiquement une liste python en tableau numpy.
De plus :

- `M = np.zeros([n,p])` fournit un tableau nul de taille $n \times p$;
- `M = np.ones([n,p])` fournit un tableau de taille $n \times p$ composé du flottant 1 ;
- On accède à l'élément de `i`-ème ligne et de la `j`-ème colonne par `M[i][j]` ou `M[i,j]`
- On rappelle que, pour un tableau `numpy`, les opérations sur les tableaux sont « transférés » aux éléments. Ainsi, `T**2` fournit un tableau de même dimensions où tous les éléments ont été mis au carré.
- Enfin, on peut faire du *slicing* ou hachage par une syntaxe du type ou `T[a:b, c:d]`. Ainsi, `T[1:3, 1:4]` fournit le sous-tableau où on a conservé les 2ème et 3ème lignes et les colonnes 2, 3 et 4. Cela donne :

$$\begin{array}{c} \\ \texttt{i=1} \\ \\ \texttt{i=3} \end{array}\!\!\begin{array}{c} \begin{array}{cccc} \texttt{j=1} & & & \texttt{j=4} \end{array} \\ \left(\begin{array}{cccc} 9 & 7 & 5 & 6 & 2 \\ 5 & 0 & 6 & 1 & 9 \\ 1 & 8 & 4 & 3 & 0 \end{array}\right) \end{array} \Rightarrow \begin{pmatrix} 0 & 6 & 1 \\ 8 & 4 & 3 \end{pmatrix}$$

Attention `T[a:b][c:d]` ne fournit pas le même résultat.

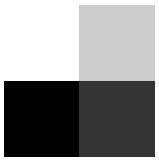
II Images en noir et blanc, images en couleurs

À retenir : Une image est représentée dans un ordinateur sous la forme d'une mosaïque de petits carrés appelés pixels [a]. La définition d'une image correspond à ses dimensions. Par exemple, une image en 1920×1280 a une largeur [b] de 1920 pixels et une hauteur [c] de 1280 pixels.

a. **pi**cture **el**ements.
b. Nombre de colonnes.
c. Nombre de lignes.

Représentation matriciel d'une image en noir et blanc : On représentera dans la suite les images en noir et blanc par une matrice de taille $n \times p$. Un carré noir correpond à 0, un carré blanc à 1 et une valeur entre 0 et 1 correspond à un niveau de gris.

$$\begin{pmatrix} 1 & 0.8 \\ 0 & 0.2 \end{pmatrix} \iff$$



Synthaxe : La méthode `imshow` de la bibliothèque `matplotlib.pyplot` permet de convertir une matrice en une image pixelisée. Le paramètre facultatif `cmap = "gray"` permet d'avoir un dégradé de gris mais on pourrait avoir un dégradé de bleu (voir figure 3).

```
import numpy as np
import matplotlib.pyplot as plt
T = np.array([[1, 0.8],[0, 0.2]])
plt.figure() # facultatif mais utile pour recommencer à zéro
plt.imshow(T, cmap="gray") # converti la matrice en niveau de gris
plt.show() # affiche
```

FIGURE 3

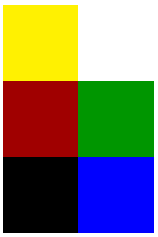
Image en couleur

Pour une image en couleur, on va attribuer à chaque pixel un triplet de valeurs (R,G,B) pour *red*, *green* et *blue*. Les valeurs peuvent être entre 0 et 1 ou entre 0 et 255 suivant les normes.

- `[0, 0, 0]` correspond à un pixel noir ;
- `[255, 0, 0]` correspond à un pixel rouge ;
- `[0, 255, 0]` correspond à un pixel vert ;
- `[0, 0, 255]` correspond à un pixel bleu ;
- `[255, 255, 0]` correspond à un pixel jaune ;
- `[255, 255, 255]` correspond à un pixel blanc ;

On donne ci-dessous un exemple :

$$\begin{pmatrix} [1,\ 1,\ 0] & [1,\ 1,\ 1] \\ [1,\ 0,\ 0] & [0,\ 1,\ 0] \\ [0,\ 0,\ 0] & [0,\ 0,\ 1] \end{pmatrix} \Longleftrightarrow$$



II.A Importation

Format d'image : Les images pixelisées sont principalement au format JPEG dont l'extension est `.jpg` ou `.jpeg`. On trouve aussi le format `.png`.

Synthaxe : On utilisera la méthode `imread` de la bibliothèque `matplotlib.pyplot`. On importe le fichier sous la forme `plt.imread("chemin/nom")` (voir figure 4). Cet méthode renvoie :

- un tableau de taille $n \times p$ pour une image en niveau de gris ;
- un tableau de taille $n \times p \times 3$ pour une image en couleur ;
- un tableau de taille $n \times p \times 4$ pour une image en couleur avec un effet de transparence (standard RGBA).

Dans l'exemple de la figure 4, chaque pixel est encodé par un triplet (R,G,B) de valeur entre 0 et 255 [a].

a. On rappelle que $255 = 256 - 1 = 2^8 - 1$. Les pixels sont encodé sous la forme de 3 octet.

Remarque : On notera qu'une image en couleur sera représentée par un **tableau à 3 dimensions** :

- la hauteur correspond au nombre de lignes ;
- la largeur correspond au nombre de colonnes ;
- la profondeur correspond au triplet `[R, G, B]`. Ainsi, `T[0, 0, 1]` fournit la valeur de vert du pixel en haut à gauche de l'image.

```python
import numpy as np
import matplotlib.pyplot as plt

T = plt.imread("tigre.jpg")
print(np.shape(T))
# renvoie (1280, 1920, 3)
print(T[0, 0])
# renvoie [72 75 32]
print(T[0, 0, 2])
# valeur de bleu du pixel en (0,0)
```

FIGURE 4 – Importation d'une image. À droite, une image au format `jpeg`.

III Manipulations d'images

Remarque : Dans toute la suite, on suppose que l'image a été importé sous la forme d'un tableau de dimensions $n \times p$. La troisième dimension sera précisée si l'image est en couleur.

III.A Manipulations des couleurs

Négatif d'une image en noir et blanc : Si le niveau de gris est encodé entre 0 et 1, alors on peut inverser chaque pixel en prenant son complémentaire à 1. Ainsi, 0 devient 1, 0.2 devient 0.8. . .. On pourra se reporter à la figure 5.

```python
n = np.shape(T)[0]
p = np.shape(T)[1]
T2 = np.zeros((n,p), dtype = int)
for i in range(n):
    for j in range(p):
        T2[i,j] = 1-T[i,j]
# variante pour les tableaux numpy
T2 = 1 - T
plt.imshow(T2, cmap="gray")
plt.show()
```

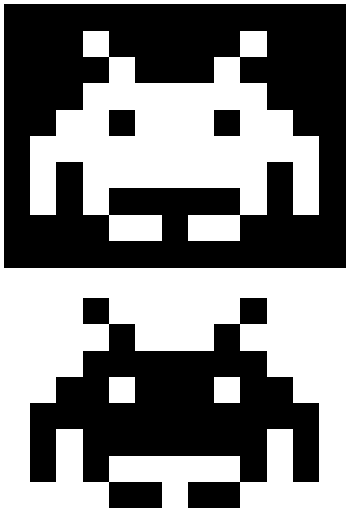


FIGURE 5 – Une image et son négatif.

Négatif d'une image en couleur : Si les couleurs sont encodés entre 0 et 255, alors il faut inverser les 3 couleurs en prenant son complémentaire à 255. On pourra se reporter à la figure 6.

III.B Manipulations géométriques

Pour ces manipulations, on va se concentrer sur des images en noir et blanc.

Symétrie : On part d'un tableau `T` représentant l'image 1 de la figure 7. Si on note n le nombre de ligne et p le nombre de colonne. On prend un repère cartésien (Cxy) avec C le centre de l'image. On constate alors que :

```
T = plt.imread("rose.jpg")
T2 = np.zeros(np.shape(T), dtype= int)
for i in range(np.shape(T)[0]):
    for j in range(np.shape(T)[1]):
        T2[i,j, 0] = 255-T[i,j, 0]
        T2[i,j, 1] = 255-T[i,j, 1]
        T2[i,j, 2] = 255-T[i,j, 2]
```

FIGURE 6 – Une image et son négatif.

- la symétrie par rapport à l'axe (Cy) revient à modifier l'indice de colonne `j` pour qu'il devienne `p-j-1`, cela donne l'image 2 et la fonction `symetrieX`;
- la symétrie par rapport à l'axe (Cx) revient à modifier l'indice de ligne `i` pour qu'il devienne `n-i-1`, cela donne l'image 3 et la fonction `symetrieY`;
- on peut composer ces deux opérations pour obtenir l'image 4. Il suffit de calculer et la fonction `symetrieX(symetrieY(T))`.

```
def symetrieX(T):
    n = np.shape(T)[0]
    p = np.shape(T)[1]
    S = np.zeros([n,p])
    for i in range(n):
        for j in range(p):
            S[i][j] =
            T[i][p-j-1]
    return S
def symetrieY(T):
    n = np.shape(T)[0]
    p = np.shape(T)[1]
    S = np.zeros([n,p])
    for i in range(n):
        for j in range(p):
            S[i][j] =
            T[n-i-1][j]
    return S
```

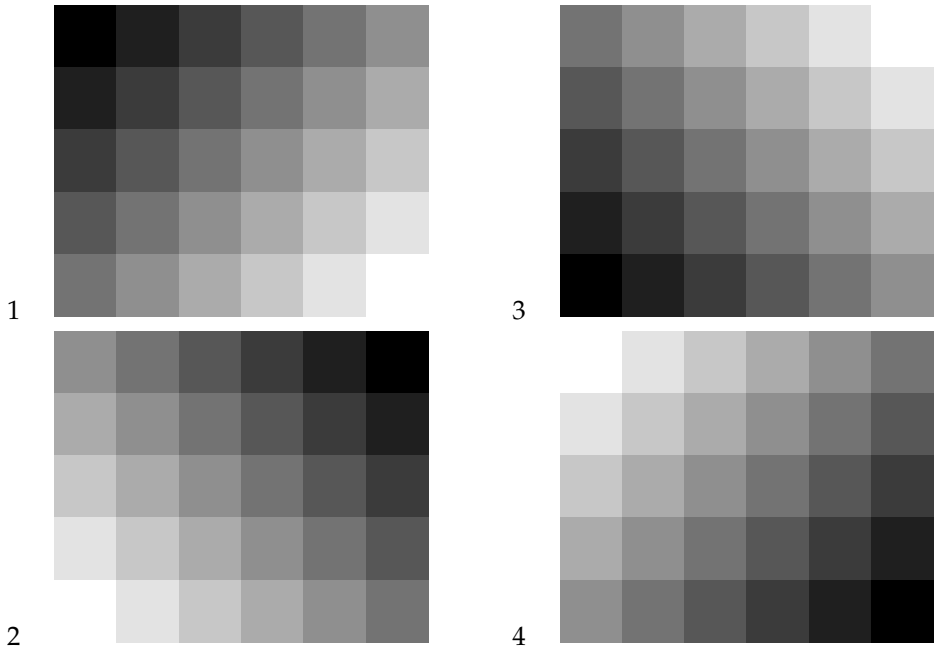
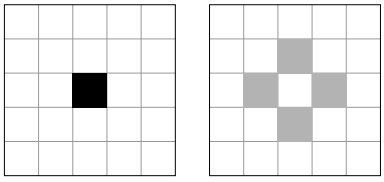


FIGURE 7 – Symétrie d'images

Flou ou filtre passe-bas :

Flouter une image revient à lisser certains détails : c'est l'application d'un filtre passe-bas spatial ! Un algorithme simple revient à remplacer chaque pixel par la moyenne des 4 pixels qui l'entourent. On pourra se reporter à la figure 8.



Pixelisation d'une image : On peut diminuer la qualité d'une image en diminuant sa résolution. On se donne un facteur de réduction α (par exemple 2 pour l'image ci-dessous). On définit des carrés de taille α comprenant α^2 pixels que l'on

FIGURE 8 – Filtre passe-bas. L'image de droite est moins nette et présente moins de détails que l'image originale.

remplace par un seul niveau de gris qui est la moyenne des α^2 pixels. On pourra alors l'appliquer à une vraie photo comme en figure 9.

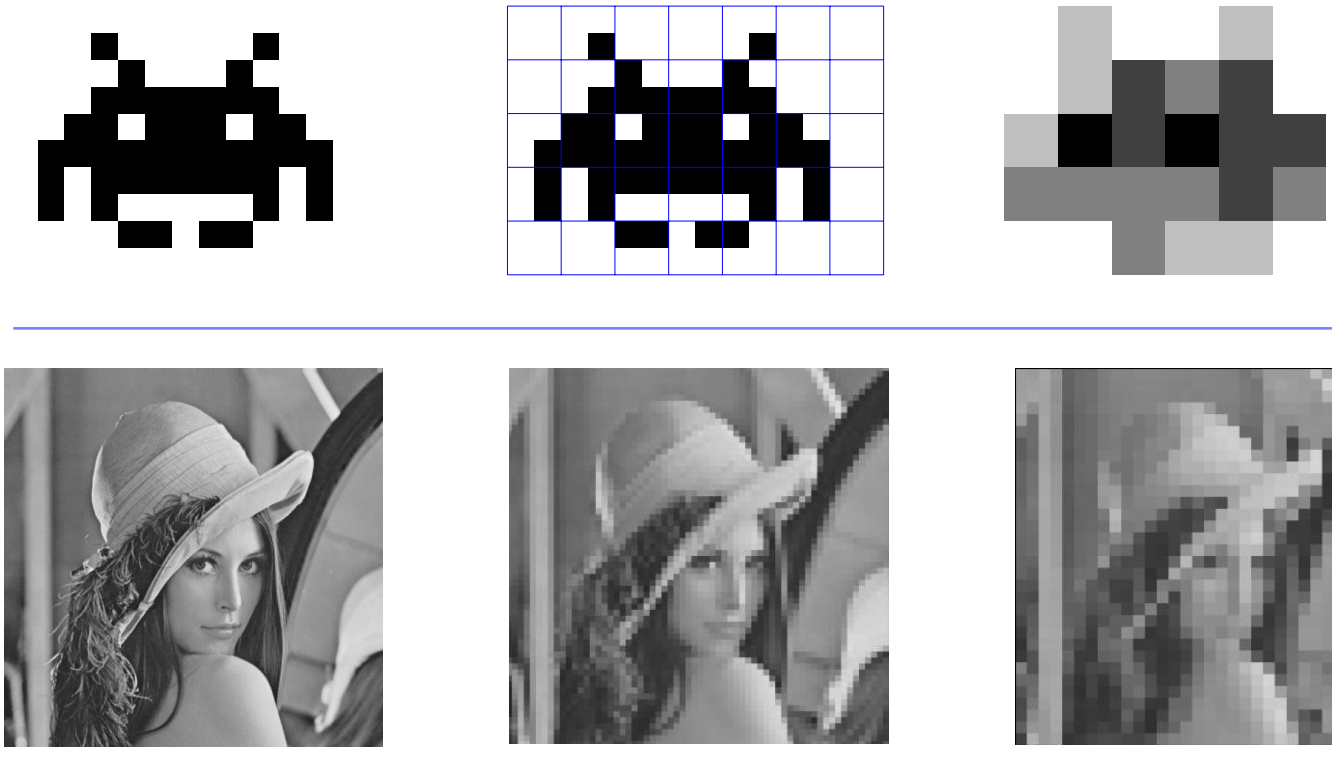


FIGURE 9 – Diminution de la résolution. L'image de départ est en 512×512. On a diminué la résolution d'un facteur 8 pour obtenir l'image du centre (de taille 64×64) et d'un facteur 16 pour l'image de droite (de taille 32×32).