CPGE BCPST 1
Année Académique: 2023 - 2024

TP N°3 d'informatique

Matrices de pixels et images

Anicet E. T. Ebou, ediman.ebou@inphb.ci

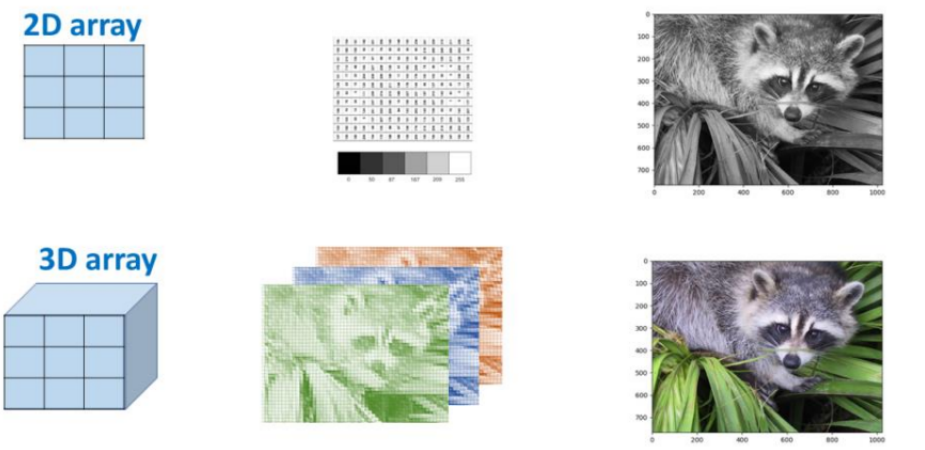
---

1. Manipulation des images avec Python

Le langage `Python` permet le traitement d'images de divers formats (`.png`, `.bmp`, `.jpeg`, ...). Quelque soit la nature de l'image (couleur ou noir et blanc), elle est vue comme un tableau représentant les $n \times p$ *pixels* de l'image de départ (en général une puissance de 2 comme 512, 1024, ...).

Pour ce faire, nous utiliserons la bibliothèque `PIL` pour laquelle une documentation extensive se trouve sur son site internet `https://pillow.readthedocs.io/en/stable/index.html`.

Une image est représentée par un tableau 2D ou 3D de valeurs appelés pixels. Si l'image est en niveau de gris, le tableau est en 2D, si elle est en couleur alors le tableau est en 3D.



Les tableaux à 3 dimensions contiennent les couches Rouge, Vert, Bleu. Chaque élément de ce tableau contient les informations propres au pixel qu'il représente. Ces informations sont de deux natures:

- Pour une image en nuance de gris (noir et blanc), c'est un nombre entre 0 et 255 allant de 255 pour un pixel blanc à 0 pour un pixel noir.
- Pour une image en couleur, c'est une liste de trois nombres qui représentent respectivement les nuances des trois couleurs primaires dans l'ordre rouge (R), vert (V) et bleu (B). Chaque nombre est un entier entre 0 et 255. L'équilibre des trois nombres donne donc la couleur (au sens du mélange) du pixel en question.

Exemples. La liste [0,0,0] code un pixel noir et [255,255,255] un pixel blanc. La liste [255,0,0] représente un pixel totalement rouge primaire, [0,255,0] un pixel totalement vert primaire et [0,0,255] un pixel totalement bleu primaire.

Chaque triplets de tels entiers représentant une couleur différente, il y a donc $255^3 = 16777216$ couleurs ou nuances possibles.

En somme, pour une image couleur, tous éléments `T[i][j]` du tableau T représentant l'image est lui même une liste dont les éléments sont `T[i][j][k]` pour $k = 0, 1, 2$.

Pour toute la suite, on s'intéresse à des images couleurs pour lesquelles on va modifier les informations des pixels, modifier leurs positions etc.

2. Les traitement de l'image

L'importance de l'image

On considère une image `photo` au format `form` se trouvant à l'adresse `chemin` dans l'ordinateur. On importe photo dans le fichier `.py` qui réalise le traitement par les commandes.

```
# on importe la bibliothèque numpy
import numpy as np
# on importe la sous-bibliothèque Image de PIL
from PIL import Image
# on ouvre dans Python l'image souhaitée
im = Image.open("chemin/photo.form")
# on crée le tableau correspondant à l'image
T = np.array(im)
```

`Attention` : il est recommandé d'identifier dans une fenêtre d'ouverture classique le chemin d'accès à `photo`. On le copie, le colle dans le fichier `.py` et certaine version de `Python` exige **le remplacement de tous les anti-slashs par des slashs**, ce que l'on fera en cas d'erreur de lecture. Le format `np.array` est un format spécifque de tableau dans `Python`. Leur numérotation est a priori identique aux listes de listes (tableau de dimension 2).

Les caracteristiques de l'image

On obtient les caractéristiques du tableau représentant l'image avec les instructions suivantes:

```
# taille de l'image en pixels
print(im.size)
# format de l'image
print(im.format)
# mode de l'image
print(im.mode)
# nombre de lignes du tableau
print(len(T))
# nombre de colonnes du tableau
print(len(T[0]))
# nombre d'éléments dans une des listes du tableau
print(len(T[0][0]))
```

Création d'une image

Si on dispose d'un tableau dont chaque élément est une liste de trois entiers entre 0 et 255, on peut faire créer l'image qui lui correspond par les lignes de code suivantes.

```
# T est un tableau adéquat
nouvim = Image.fromarray(np.uint8(T))
# on affiche l'image
```

```
nouvim.show()
# on enregistre l'image sous le nom photo1
nouvim.save("chemin/photo1.form")
```

uint8 représente l'encodage des données (le format des caractères du tableau). On ne sera pas tenu d'enregistrer les images créées à chaque étape.

Les manipulations d'une image

Les manipulations d'une image que l'on peut envisager sont nombreuses. On tentera de se concentrer sur quelques points, comme :

- Le changement des composantes de couleurs.
- L'égalisation des composantes de couleurs ("niveau de gris").
- Des permutations de pixels d'un tableau, pour avoir des changements géométriques (symétries, rotations...).

3. Applications et manipulations

Exercice 1

1. Avec `np.array`, créer un tableau 2×2 de triplets représentant chaque couleur primaire avec un triplet supplémentaire pour la couleurdc blanche.
2. Faire afficher l'image correspondante : on doit avoir quatre carré des quatre couleurs données (*il faudra peut-être zoomer !*).

Pour toute la suite, on choisit une image (de petite taille de préférence) ou alors on considère l'image fournie sur laquelle on pratiquera tous les exemples.

Exercice 2:

1. Réaliser l'importation de l'image, puis sa transformation sous forme de tableau.
2. Combien de pixels comporte l'image ? (Taille de l'image).
3. Donner les composantes de couleurs du dernier pixel de l'image (celui en bas à droite)

Exercice 3 :

1. Compléter la fonction `composanterouge(T)`, qui renvoie pour un tableau uniquement les composantes rouges, à savoir `[r,0,0]` pour un triplet `[r,v,b]` (on remarquera que le fonctionnement choisi évite les effets de bords).

```
def composanterouge(T):
    U=[] # le tableau où l'on stockera le résultat
    for i in range(len(T)):
        L=[] # la ligne à remplir
        for j in range(len(T[0])):
            L.append(...) # on ne garde que la composante rouge
        U.append(L) # on complète U avec la ligne L
return U
```

2. Ecrire de même des fonctions `composantevert(T)` et `composantebleu(T)`.

3. Faire afficher les images correspondante avec l'image de départ fournie. On utilisera:

```
# les caractères d'encodage sont au format uint8
nimage=Image.fromarray(np.uint8(composanterouge(T)))
nimage.show()
```

On pourra se servir des fonctions précédentes pour avoir une trame de remplissage du tableau que l'on constitue pour avoir une nouvelle image. De même, on pourra reprendre les instructions qui permettent l'affichage.

Exercice 4 : On appelle négatif de tout `[r,v,b]` le triplet `[255-r,255-v,255-b]`. Ecrire une fonction `negatif(T)` qui renvoie le tableau des négatifs de `T`. Faire afficher l'image négative qui correspond à l'image de départ.

Exercice 5 : Ecrire une fonction `inversioncouleurs(T)` qui permute au moins deux couleurs de chaque pixels (toujours la même permutation, quelque soit le pixel). Faire afficher l'image correspondante.

Exercice 6 : Ecrire une fonction `attenuation(T)` qui divise chaque valeur d'un triplet de couleur par 2. Faire afficher l'image correspondante. Que remarque-t-on ?

Exercice 7:

1. Ecrire une fonction `renverse(T)` qui donne le tableau correspondant à l'image renversée (par symétrie centrale).
2. Ecrire des fonctions `miroir1(T)` et `miroir2(T)` qui donne respectivement les tableaux correspondants à la symétrie d'axe vertical et la symétrie d'axe horizontal.
3. Faire apparaître les trois images transformées.

Exercice 8: Une image en niveau de gris s'obtient, à partir d'une image en couleur) de la façon suivante. Pour des valeurs `r,v,b` de niveaux respectifs du rouge, vert et bleu d'un pixel, l'indice du niveau de gris est donné par

$$g = 0,299r + 0,587v + 0,114b$$

(une formule parmi tant d'autre).

Ensuite, l'image en niveau de gris est l'image obtenue avec des pixels dont les trois nuances de rouge vert et bleu sont toutes égales à l'indice `g` calculé. Ecrire une fonction `niveaugris(T)` qui renvoie le tableau correspondant à l'image en niveau de gris. Faire apparaître l'image en niveau de gris de l'exemple fourni.

4. Rotation

La question de la rotation d'une image peut être complexe, nous regarderons qu'une idée rapide. Considérons une matrice de pixel, `carrée, en noir et blanc` (donc une liste de liste d'entiers). On applique le principe récursif suivant : on coupe l'image en 4 cadrans et si on les note (c_1, c_2, c_3, c_4), alors on renvoie (c_2, c_3, c_4, c_1), puis on recommence, jusqu'à que ces cadrans contiennent un pixel.

Exercice 9: Mettre en oeuvre ce principe sur une image carrée de votre choix.

5. Quelques exercices supplémentaires

Exercice 10: Écrire une fonction qui prend en entrée un matrice en RGB, et renvoie la même matrice en effectuant une permutation sur les couleurs de l'image.

Exercice 10 bis: Écrire une fonction python qui prend en argument une matrice carrée de taille paire et renvoie la même image modifiée selon le procéde suivant: La première ligne et la dernière sont fixes mais l'on échange les lignes 1 et 2, puis 3 et 4 puis ... jusqu'à celle `n-3` et `n-2`. Puis on recommence en fixant les deux premières et les deux dernières, puis les trois premières et les 3 dernières, ... jusqu'à la dernière étape où ne sont échangées que les lignes `n/2 - 1` et `n/2`.
La tester plusieurs fois sur une image.

6. Contraste, version naïve

Pour modifier le contraste d'une image RGB, on peut, en premier lieu, commencer par modifier l'ensemble des valeurs des RGB par une fonction f qui aurait un des deux effets suivants :

1. Soit la fonction exacerbe les contrastes en rapprochant les valeurs centrales (celle proche de 122) des extrêmes dont elles sont le plus proche (0 ou 255).
2. Soit la fonction attenue les contraste en rapprochant les valeurs proches des extrêmes du centre.

On propose pour cela les fonctions : $f : x \mapsto 255 \times (arctan(\frac{x-122}{20}) \times \frac{1}{\pi} + \frac{1}{2})$ et $g : x \mapsto tan((\frac{x}{255} - \frac{1}{2}) \times \pi) \times 20 + 122$.

Exercice 11: Comprendre l'intérêt de ces fonctions, et les adapter pour répondre au problème. Le tester sur une image de votre choix.

7. Produit de convolution, transformation d'image

Cette partie est une introduction à une notion bien plus complexe, l'usage du produit de convolution sur une matrice d'image.

On considèrera une matrice d'image en noir et blanc que l'on notera à l'aide d'une suite indexée par deux entiers: $T = (T_{i,j})_{i\in[[0,n-1]],j\in[[0,m-1]]}$ (on peut la voir comme une matrice).

Soit $c = (c_{i,j})_{i\in[[0,d-1]],j\in[[0,d-1]]}$ on définit alors le produit de convolution $T \times c$ de T par c, par la suite indexée par deux entiers tel que pour tout $(i, j) \in [[0, n-1]] \times [[0, m-1]]$:

$$(z \times c)_{i,j} = \sum_{k=0}^{d-1} \sum_{l=0}^{d-1} T_{i-k,j-l} c_{k,l}$$

On dit alors que c est le noyau de convolution.

8. Découverte, lissage

On pose les deux matrices $C_0 = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$ et $C_1 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

Exercice 12: Calculer pour tout i, j, $(T \times C_0)_{i,j}$. Comprendre en quoi est ce que ce produit de convolution remplace une matrice d'une image et la matrice d'une même image mais ajoutée. On

admet dans la suite que le produit de convolution par C_1 renvoie une matrice d'image de contraste accentuée.

Exercice 13: Écrire une fonction qui prend en entrée une matrice de pixel en noir et blanc T et une matrice C et renvoie le produit de convolution $T \times C$. Tester cette fonction avec C_0 et C_1.

Exercice 14: Enfin, on cherche à renvoyer, à partir d'une image de départ, une image contenant uniquement les lignes de contours des formes présentes sur l'image. L'idée est de calculer la variation de l'intensité des pixels et de ne renvoyer que ceux qui sont d'intensité fortement supérieure à celle de ses voisins. Pour cela, on admet que le maximum des valeurs absolues des produits de convolution de T par les matrices de Sobel : $G_x = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$ et $G_y = {}^t G_x$ renvoie une estimation de cette variation d'intensité. Construire une fonction `Sobel(T,s)` qui prend en entrée une matrice de pixels, en noir et blanc, T et un entier seuil s et renvoie la matrice des lignes contours de la matrice originale (on effacera les pixels de variations inférieurs à s). La tester avec s = 60 environ.

Défi: En regardant la fonction de deux variables Z (que l'on supposera C^2) telle que $Z(x_i, y_j)$ est l'intensité du pixel d'index (i,j), pour tout i, j, et les formules de Taylor, montrer que la méthode ci-dessus semble fonctionner, c'est à dire que les produits de convolution de T par G_x et par G_y représente une approximation du gradient de Z en les points x_i, y_j.