

Activité 7

Images, traitement d'images

1 Quelques aspects « théoriques »...

1.1 Un tableau (ou une matrice) comme liste de listes

Les types vecteur et matrice n'existent pas en python ; deux solutions s'offrent à nous :

- soit utiliser une bibliothèque annexe : **numpy** en l'occurrence ; nous aborderons quelques aspects de cette bibliothèque ultérieurement ;
- soit implémenter le type vecteur par une liste python et le type matrice par... une liste de liste.

Dans cet activité nous allons utiliser une matrice comme structure de données (chaque élément contient un pixel de l'image) ; nous verrons ultérieurement quelques éléments d'algèbre linéaire.

a Création

Considérons, par exemple la matrice $M = \begin{bmatrix} 5 & 3 \\ 0 & 7 \\ 4 & 1 \end{bmatrix}$ à trois lignes et deux colonnes.

Cette matrice sera créée par $M = \underbrace{[[\overbrace{5,3}^{2 \text{ colonnes}}], [\overbrace{0,7}^{2 \text{ colonnes}}], [\overbrace{4,1}^{2 \text{ colonnes}}]]}_{3 \text{ lignes}}$.

Plus généralement, on aura, pour une matrice de nl lignes et nc colonnes :

$M = [ligne_0, ligne_1, \dots, ligne_i, \dots, ligne_{nl-1}]$ c'est à dire : $M = \begin{bmatrix} ligne_0 \\ \dots \\ ligne_i \\ \dots \\ ligne_{nl-1} \end{bmatrix}$

Chaque ligne ayant nc éléments : $ligne_i = [a_{i,0}, a_{i,1}, \dots, a_{i,j}, \dots, a_{i,nc-1}]$.

Si on veut créer une matrice, il faut commencer par créer les listes « intérieures » puis faire un *append* sur la liste principale. Par exemple, pour créer une matrice de 3 lignes et 4 colonnes remplies de 0 on peut écrire.

Listing 7.1 – Création d'une matrice comme liste de liste

```
1 # création par append de liste
2 M = []
```

```

3 for l in range(3): # on boucle sur 3 lignes
4     L = [] # une ligne vide
5     for c in range(4): # on boucle sur 4 colonnes
6         L.append(0)
7     M.append(L)
8 # création par compréhension
9 M = [[0 for c in range(4)] for l in range(3)]

```

Plus généralement, lorsqu'on écrit la liste par compréhension, le plus simple est peut-être d'opérer de la façon suivante :

$M = []$

⇒ on veut créer une matrice comme liste de liste

$M = [[\text{for indice_colonne in range(nb_colonne)}]]$

⇒ on gère le nombre de colonnes, c'est à dire le nombre d'éléments des listes « intérieures »

$M = [[\text{for indice_colonne in range(nb_colonne)}] \text{for indice_ligne in range(nb_ligne)}]$

⇒ on gère le nombre de lignes, c'est à dire le nombre de listes « intérieures »

$M = [[\text{fonction de } i_ligne \text{ et } i_colonne \text{ for } i_colonne \text{ in range(nb_colonne)}] \text{for } i_ligne \text{ in range(nb_l)}]$

⇒ on gère l'élément à ajouter, fonction de l'indice de la ligne et de la colonne.

b Accès aux éléments

ATTENTION : on a une liste de liste et non un tableau à deux dimensions ; on ne peut donc pas accéder un élément via l'instruction : $a = M[i, j]$

L'accès à un élément de la matrice se fait par $a = M[\text{indice de la ligne}][\text{indice de la colonne}]$.

Ainsi, par exemple : $M[2][1]$ vaut 1. Lors de l'exécution de cette instruction, on commence par récupérer $M[2]$ c'est à dire la troisième ligne : $[4, 1] \dots$ dont on récupère le second élément, de valeur 1.

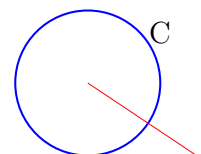
c Dimensions de la matrice

Réfléchissons...

- $\text{len}(M)$ retourne le nombre d'éléments dans la liste M , il s'agit donc du nombre de lignes ;
- pour avoir le nombre de colonnes, il suffit d'accéder à une ligne quelconque et de calculer la longueur de cette liste : $\text{len}(M[0])$ retourne donc le nombre de colonnes.

1.2 Image bitmap

La plus belle image au monde est certainement le dernier selfie que vous avez réalisé. C'est beaucoup moins fun, mais ce qui est représenté ci-contre est également une image !



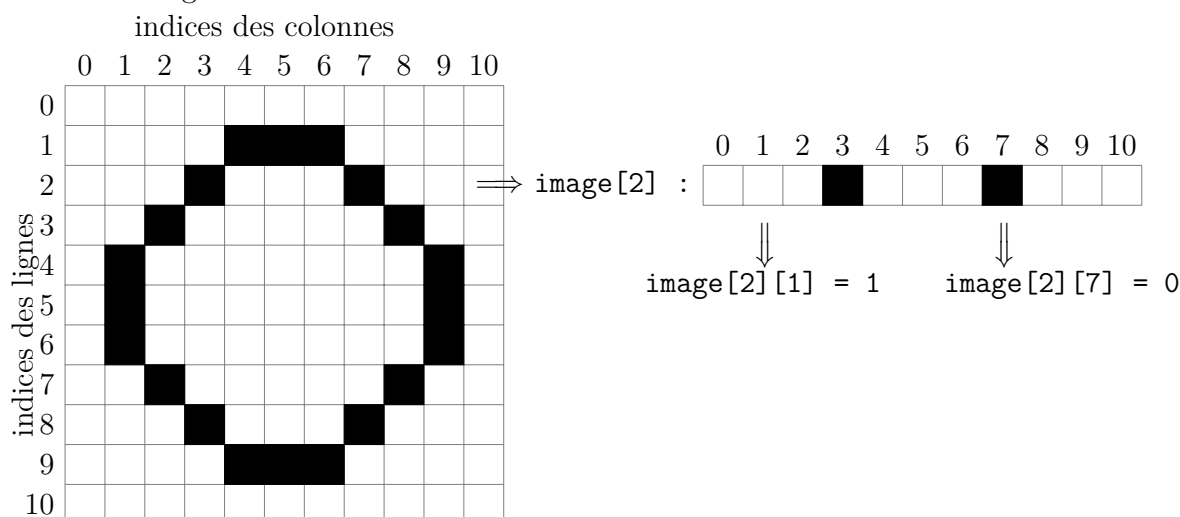
Dans le premier cas, il n'existe guère qu'une solution pour stocker l'information : décrire l'image comme un ensemble de points. Le plus simple des systèmes de codage des couleurs pour chacun de ces pixels est le mode RGB (ou RVB en français). La valeur de chacune des couleurs primaires correspond à un entier variant entre 0 (absence) et 255 (intensité maximale). Le codage de chaque couleur est alors possible sur un octet (8 bits). La combinaison des 3 entiers R , G et B permet ainsi de décrire : $256^3 = 16\,777\,216$ couleurs. Précisons également qu'il s'agit d'une synthèse additive des couleurs. **Le noir sera codé par (0,0,0) ; le blanc par (255,255,255).**

Nous allons aborder, ici, une manipulation de « bas niveau » de l'image en traitant l'image pixel par pixel à l'aide de code python !

Nous utiliserons pour cela la bibliothèque `matplotlib.pyplot` pour afficher les images. La façon la plus simple de préciser la « couleur » d'un pixel est alors de donner une liste de trois nombres réels `[r,g,b]` correspondant respectivement au rouge, vert, bleu (chaque nombre étant compris entre 0 et 1); ce qui revient à diviser la valeur RGB par 255!

a De la matrice de points à l'image, et vice versa

L'image sera implémentée sous forme de liste de listes (une liste de lignes donc!). Prenons l'image d'un cercle.



Si la variable `image` est associée à la matrice précédente on peut :

- accéder à un élément de l'image par `image[indice ligne][indice colonne]` (attention à bien respecter l'ordre : l'instruction se lit de la gauche vers la droite, on accède d'abord à la ligne puis à la colonne!);
- accéder au nombre de ligne par `len(image)`;
- accéder au nombre de colonne par `len(image[0])` (en fait on demande combien il y a d'éléments dans la première ligne!).

b Création d'une image à l'aide de code python

b.1 Image noire et blanc ou niveau de gris

On code le niveau de gris par un entier compris entre 0 et 1 : 0 correspond au noir et 1 au blanc. Pour créer une matrice (de petite taille) on pourra écrire quelque chose comme :

`image1 = [[ligne 1], [ligne 2], ..., [ligne nbLig - 1]]` et pour chaque ligne : `[colonne 0], [colonne 1], ..., [colonne nbCol-1]`.

C'est un peu fastidieux...

On peut aussi, créer une matrice qui ne contient, par exemple que des 1 (on a ainsi un fond blanc) et modifier les points un par un. `M = [[1 for c in range(nbCol)] for l in range(nbLig)]`, toujours dans cet ordre puisqu'on crée (en lisant de l'extérieur vers l'intérieur) `nbLig` listes qui contiennent `nbCol` valeurs.

Une fois l'image créée, on l'affiche à l'aide de l'instruction : `plt.imshow(image, cmap='gray')` (pour une image en niveau de gris).

b.2 Image en couleur Une liste de trois valeurs (réelles comprises entre 0 et 1) permet de définir les composantes rouge-vert-bleu : $[r, g, b]$...une image en couleur est donc une liste de listes de listes !

Pour afficher l'image *image*, on utilisera l'instruction `plt.imshow(image)`.

b.3 Importer une image Après avoir importé la bibliothèque `matplotlib.image` à l'aide de l'alias `img`, on peut importer une image de format `.jpg` ou `.png` à l'aide de l'instruction `img.imread(adresse de l'image)`.

1.3 Traitement d'image

Après avoir importé une image dans un logiciel de traitement d'image, on peut aisément modifier certains de ses aspects : luminosité, contraste, couleurs...

1.4 Niveau de gris

Convertir une image en niveaux de gris revient à :

- n'utiliser qu'une seule valeur pour coder le niveau de gris ;
- utiliser un triplet de trois valeurs identiques si on désire conserver le format d'origine de l'image rgb.

Pour convertir une image couleur en niveau de gris :

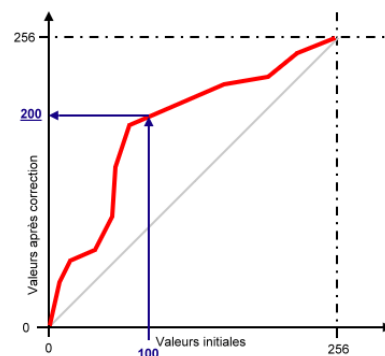
- soit on fait la moyenne des trois valeurs : $gris = \frac{(r + g + b)}{3}$;
- soit on utilise une formule un peu plus sophistiquée qui tient compte de la sensibilité de l'oeil humain. La plus courante est : $gris = 0,299 \times rouge + 0,587 \times vert + 0,114 \times bleu$.

a Luminosité, contraste

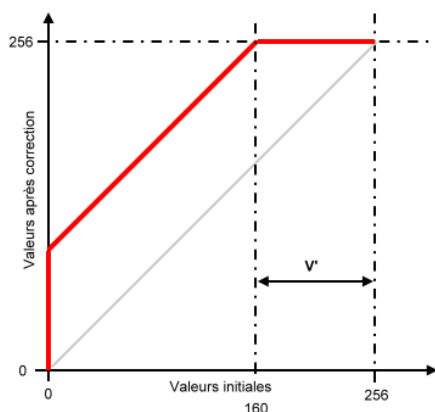
Retoucher une image revient à modifier les valeurs de certains pixels. On peut le faire localement (à un endroit précis de l'image) ou globalement. Dans ce dernier cas, on utilise un outil appelé « courbe tonale » (dessin ci-contre ^a).

Sur l'abscisse, on lit les valeurs originales des pixels et sur l'ordonnée les valeurs après modifications. La diagonale représente la courbe où il n'y a aucune modification

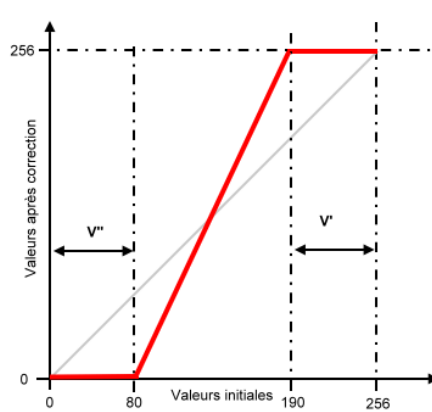
Dans l'exemple ci-contre, par exemple, tous les pixels de valeurs 100 prendront la valeur 200. Ici, l'image sera éclaircie.



^a. Il faudrait, en principe, représenter trois courbes tonales : une pour le rouge, une pour le vert et une pour le bleu.



Augmenter la luminosité

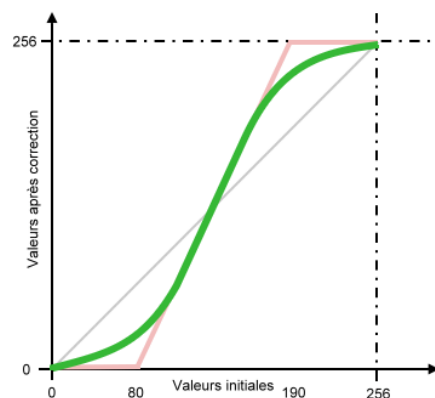


Augmenter le contraste

Le traitement proposé dans les schémas précédents est un peu « brutal » ; il y a perte d'information. Dans l'exemple de droite : tous les points ayant une valeur inférieure à 80 seront noirs et tous ceux ayant une valeur supérieure à 190 seront blancs.

Dans la « vraie vie » il est préférable d'utiliser une courbe lissée dans laquelle la courbe tonale rejoint les axes tangentiellement (comme dans l'exemple ci-contre).

Il nous faut alors connaître l'expression analytique de la fonction permettant cette transformation.



2

A vous de jouer...

2.1

Création d'images

a

Quelques routines utilitaires...

La trame du fichier à exploiter cette semaine contient l'importation des bibliothèques nécessaires ainsi que la définition des variables correspondant aux pixels *blanc*, *noir*, *gris*, *rouge* et *bleu*.

```
1 import matplotlib.pyplot as plt
2 import matplotlib.image as img
3
4 blanc=[1.0,1.0,1.0]
5 noir=[0.0,0.0,0.0]
6 gris=[0.5,0.5,0.5]
7 rouge=[1.0,0.0,0.0]
8 bleu=[0.0,0.0,1.0]
```

1. Ecrire le code d'une fonction *largeur(image)* retournant la largeur de l'image (liste de listes) passée en paramètre.
2. Ecrire le code d'une fonction *hauteur(image)* retournant la hauteur de l'image (liste de listes) passée en paramètre.
3. Ecrire le code d'une fonction *fond(nl,nc,color)* qui à partir de la couleur d'un pixel passé en paramètre crée une image de cette couleur comptant *nl* lignes et *nc* colonnes et retourne cette image.

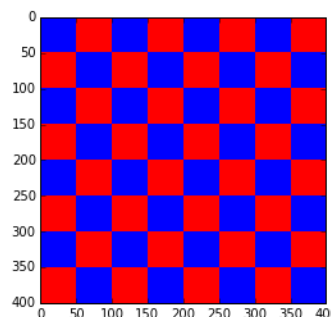
4. Ecrire le code d'une fonction `insert(fond, image, pos_x, pos_y)` qui permet d'insérer l'image `image` dans le fond `fond` aux positions spécifiées (l'origine du repère étant en haut à gauche, l'axe x orienté vers la droite et l'axe y orienté vers le bas). La variable `image` n'est pas modifiée. Le fond, quant à lui, est modifié. Si l'image insérée dépasse du fond, elle doit être tronquée.
5. Tester votre code sur quelques exemples de votre choix.

b**Séquence patriotique**

1. Compléter le programme afin de tracer un drapeau français.
2. Compléter le programme afin de tracer un drapeau japonais. Le rapport entre la hauteur et la largeur du drapeau est de 2/3, et le diamètre du disque rouge est trois cinquièmes de la hauteur du drapeau.

c**Un damier pour jouer aux échecs**

1. Ecrire le code d'une fonction `damier(n,p,color1,color2)` permet d'afficher un damier de n cases horizontales et n cases verticales alternativement de couleurs `color1` et `color2`; chaque case étant représentée par un carré de $p \times p$ pixels.
2. Ecrire le code permettant de stocker le damier représenté ci-contre dans la variable `echec`, chaque case de l'échiquier étant un carré de 25 pixels de côté.
3. Importer l'image du fichier `cavalier.png` (l'image doit être au même niveau que votre programme sur le disque dur pour éviter à gérer les chemins d'accès). Une conversion de format est nécessaire ici (dans le fichier d'origine, on a un format `rgba`) : il faut écrire le code suivant :



```

1  cavalier=img.imread("cavalier.png","png")
2
3  def rgba2rgb(image):
4      nl = len(image)
5      nc = len(image[0])
6      rgb = fond(nl,nc,blanc)
7      for l in range(nl):
8          for c in range(nc):
9              pixel = image[l][c]
10             rgb[l][c]= [pixel[0],pixel[1],pixel[2]]
11     return rgb
12
13  cavalier2 = rgba2rgb(cavalier)

```

Vérifier que la taille de l'image est bien 50×50 et afficher l'image.

4. A l'aide de l'instruction `randint(0,7)` de la bibliothèque `random`, tirer deux nombres entiers au hasard entre 0 et 7 (inclus) et placer le cavalier à cette position sur l'échiquier.
5. Pour experts... un peu de transparence

On aimerait bien pouvoir voir la couleur de la case sur laquelle est posé l'image du cavalier à l'extérieur de la forme (fermée) qui représente le cavalier.

Pour cela, au lieu de décrire chaque pixel par 3 valeurs $[r, g, b]$ on ajoute un quatrième paramètre a qui gère la transparence ($a = 1$ pour une opacité maximale ; $a = 0$ pour une transparence totale). On a alors un codage RGBA de la couleur par une liste $[r, g, b, a]$.

Modifier les différentes routines précédentes afin d'afficher une image du cavalier comme ci-contre.



6. Et, pourquoi pas maintenant... colorier en gris toutes les cases accessibles par le cavalier !

2.2 Coucou, Lenna



Cette image, libre de droits, sert d'image de test pour les algorithmes de traitement d'image et est devenue de facto un standard industriel et scientifique. Wikipedia vous précisera son origine.

Le code proposé ci-dessous charge l'image dans la variable *lenna* et affiche l'image. Ensuite, on appelle une routine qui retourne l'image verticalement (cet exemple peut servir de modèle pour la plupart des exercices suivants).

```
1 lenna = img.imread("lenna.png")
2 plt.imshow(lenna)
3 plt.show()
```

a Retourner l'image

1. Écrire une routine qui retourne l'image horizontalement.
2. Écrire une routine qui retourne l'image verticalement.

b Niveaux de gris

1. Créer une routine *niveauGris(image)* qui, à partir d'une image passée en paramètre, retourne l'image en niveaux de gris. Pour gagner du temps sur la suite, on codera le niveau de gris sur un seul réel compris entre 0 et 1.
2. Après avoir appelé cette routine et stocké le résultat dans la variable *lennaGris*, afficher cette image (`plt.imshow(lennaGris, cmap='gray')`).

c Retouche d'image

Pour simplifier, on travaillera, ici, sur l'image en niveau de gris *lennaGris*.

1. Proposer une routine *eclaircir* (acceptant éventuellement un paramètre) qui traite une image en niveau de gris afin de l'éclaircir. Tester sur l'image *lennaGris*
2. On pourrait aussi :
 - a. obtenir le négatif de l'image couleur (ou niveaux de gris) en remplaçant toutes les valeurs x de tous les pixels par $1 - x$.
 - b. effectuer un seuillage de l'image (niveaux de gris ou couleur) : on remplace toutes les valeurs x de tous les pixels par 1 si $x < seuil$ et par 0 si $x > seuil$
 - c. ... on peut imaginer aussi son propre traitement d'image : pourquoi pas ceux correspondants aux courbes tonales proposées.

d Application d'un filtre

Un traitement « classique » d'images (que l'on retrouve dans la plupart de logiciels) consiste à appliquer un filtre. Par exemple, à l'aide de Photoshop (ou gimp), pour flouter quelques défauts sur votre selphie !

On effectue alors ce qu'on appelle une convolution.

Prenons, par exemple, l'exemple d'un filtre 3×3 : $F = \begin{pmatrix} F_{0,0} & F_{1,0} & F_{2,0} \\ F_{0,1} & F_{1,1} & F_{2,1} \\ F_{0,2} & F_{1,2} & F_{2,2} \end{pmatrix}$

L'image filtrée J s'obtient en effectuant le produit de convolution entre l'image I et le filtre F . Le pixel de coordonnées (i,j) dans l'image filtrée est obtenu en faisant la moyenne pondérée (par les coefficients du filtre F) du pixel $I_{i,j}$ de l'image initiale et de ses 8 proches voisins.

$$J_{i,j} = F_{0,0}I_{i-1,j-1} + F_{1,0}I_{i,j-1} + F_{2,0}I_{i+1,j-1} \\ + F_{0,1}I_{i-1,j} + F_{1,1}I_{i,j} + F_{2,1}I_{i+1,j} \\ + F_{0,2}I_{i-1,j+1} + F_{1,2}I_{i,j+1} + F_{2,2}I_{i+1,j+1}$$

On s'assure toutefois que $J_{i,j}$ est compris entre 0 et 1 : si $J_{i,j} < 0$, alors $J_{i,j} = 0$ et si $J_{i,j} > 1$ alors $J_{i,j} = 1$.

On propose quelques filtres « classiques » : lissage, accentuation, embossage, convolution (dans l'ordre).

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}; \begin{pmatrix} 0 & -0.5 & 0 \\ -0.5 & 3 & -0.5 \\ 0 & -0.5 & 0 \end{pmatrix}; \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}; \begin{pmatrix} -\frac{1}{6} & -\frac{2}{3} & -\frac{1}{6} \\ \frac{2}{3} & \frac{13}{3} & \frac{2}{3} \\ -\frac{1}{6} & -\frac{2}{3} & -\frac{1}{6} \end{pmatrix}$$

1. Ecrire le code d'une routine *convolution(image, filtre)* qui applique à l'image *image* le filtre *filtre* constitué d'une matrice 3×3 et qui retourne l'image filtrée. On traite une image en niveaux de gris avec le gris codé sur 1 réel.
2. Tester le résultat sur *lennaGris* avec l'un des filtres proposés.
3. On peut aussi proposer son propre filtre (il faut juste s'assurer que la somme des termes vaille soit 0 soit 1).

2.3 S'il reste du temps... stéganographie : une image dans une image

Contrairement à la cryptographie, qui chiffre des messages de manière à les rendre incompréhensibles, la stéganographie cache les messages dans un support, par exemple des images ou un texte qui semble anodin.

On peut cacher un texte dans une image numérique et cela de manière parfaitement invisible à l'œil nu. Cette technique s'appelle le tatouage (watermarking en anglais). Elle est utilisée notamment pour protéger des images par copyright, mais on peut aussi transmettre des messages cachés. Nous proposons ici une méthode simple, mais qui fonctionne seulement avec certains formats d'images. En effet, beaucoup de formats compressent les données et donc modifient les bits de l'image, ce qui a pour effet de détruire le message caché.

On propose de traiter chaque composante x de chaque pixel de la façon suivante :

- on convertit x en nombre entier n compris entre 0 et 255 (dans l'image donnée, x varie entre 0 et 1) ;
- on récupère les 4 bits faibles de n et on les décale de 4 bits vers la gauche.

	bits de poids fort				bits de poids faible			
$n =$	1	0	0	1	0	1	1	0
reste de la division par 16 : $n \% 16 =$	0	0	0	0	0	1	1	0
que l'on multiplie par 16 : $16 \times (n \% 16) =$	0	1	1	0	0	0	0	0

- on divise la valeur obtenue par 255 pour la convertir en réel compris entre 0 et 1.
1. Faire cette conversion pixel par pixel sur l'image mystere.png ;
 2. Découvrir l'image cachée (le calcul est un peu long) !