TP 10 : traitement d'images

PTSI 2024/2025

POINT COURS : IMPORTATION DE MODULES, MODULE `NUMPY`

- D'une manière générale on peut importer un module ou des fonctions/méthodes complémentaires à celles fournies par Python deux manières comme ci-dessous pour le module `numpy` qui est un module de calcul mathématique :
  - `import numpy` ou pour faire plus court ensuite `import numpy as np`. Une fonction de ce module est ensuite accessible par la syntaxe : `numpy.fonction` (ou `np.fonction` si on a déclaré l'alias `np`).
  - `from numpy import log` : ici seule la fonction `log` du module numpy sera utilisable ensuite et ce par la syntaxe `log(2)` par exemple (on ne rappelle pas que la fonction `log` est celle du module `numpy`)
  - La méthode précédente peut être étendue pour importer plusieurs fonctions :

```
from numpy import log, cos
```

Script 1 : *les tableaux numpy*

Niveau : 1

Les tableaux dans `numpy` offrent la possibilité de manipuler des objets s'apparentant à des listes, ou des listes de listes, mais de manière plus souple et plus performante. Ces objets sont de type `ndarray`. Faire les essais suivants dans une console Python après avoir importé le module `numpy` :

1. `t = numpy.array([2,-4,1.9, 3, -3])`
2. Tester les commandes suivantes appelées commande de ***slicing*** :

```
t[1:3],t[2:] ..
```

3. Créer un nouvel objet de type `ndarray` par les commandes suivantes :

```
A = [[1,  2, 3], [4,  5,  6],[7,  8,  9]]
ta = numpy.array(A)
```

   puis tester les commandes suivantes :

```
ta[1][2], ta[1,2], ta+ta, 1/2*ta, ta*ta.
```

   Commenter par rapport aux usages possibles sur la liste `A`.

4. Tester les commandes suivantes puis commenter :

```
ta[1:3], ta[1:][0:2], ta[1: , 0:2]
```

5. Par ailleurs, les tableaux `numpy` peuvent être utilisés comme variables de certaines fonctions de la variable réelle qui alors s'appliquent à chaque élément du tableau. Tester par exemple

```
>>> t = [3.3, 2.2, -1]
>>> np.floor(t)
>>> int(t)
```

   Commenter.

- On pourra dans la suite du TP utiliser le matériel `numpy` suivant :
  - `numpy.sum(<tableau>)` renvoie la somme des éléments du tableaux.
  - `numpy.floor(<tableau>)` applique la fonction partie entière à tous les éléments du tableau et renvoie le tableau des partie entières.
  - `numpy.shape(<tableau>)` renvoie les dimensions du tableau.

- ▹ `numpy.ones((l,h))` créer un tableau `numpy` bi-dimensionnel de dimensions `l` lignes et `h` colonnes. (de même avec `numpy.zeros`)

- Attention. Il n'y a pas d'équivalent de la méthode `append` des listes pour les tableau `numpy`.

POINT COURS : CODAGE DES IMAGES AU FORMAT BITMAP (BMP)

- Dans ce format, l'image est représenté comme un tableau à deux dimensions dont chaque cellule correspond à un pixel et contient l'information concernant la couleur de celui-ci.
- Toute couleur est vue comme une synthèse de trois couleurs primaires : Red, Green, Blue et le niveau de chacune de ses couleur dans un pixel sera représenté par un entier en 0 et 255 (dans le format dit "24 bits")
- ce qui donne le schéma suivant pour 9 pixels :

(255,0,0)	(0,255,0)	(0,0,255)
(112,48,160)	(196,89,17)	(255,255,255)
(254,176,131)	(91,153,213)	(83,59,123)
(0,0,0)	(255,255,0)	(255,51,153)

- ce qui va correspondre informatiquement à un tableau contenant des triplets de nombre entiers pour représenter la couleur de chaque pixel ce qui donne pour l'exemple ci-dessus en utilisant une liste de liste ou un tableau `numpy` :

```
image =
[[[255,0,0], [0,255,0],[0,0,255]],
[[112,48,160], [196,89,17],[255,255,255]],
[[254,176,131], [91,153,213],[83,59,123]],
[[0,0,0], [255,255,0],[255,51,153]]]
```

(en remarquant que le pixel en bas à droite est repéré dans la liste par `image[3][2]` : la numérotation des lignes allant "de haut en bas ").

- Pour gérer cela sous Python on va utiliser deux modules :
  - `numpy` pour gérer les tableaux à l'aide des objets de type `array`.
  - `pyplot` qui permet d'importer des images au format BMP en les implémentant sous forme de tableau `numpy` et de les afficher :
    - ▹ `pyplot.imread<fichier.bmp>` pour lire un fichier image et le récupérer sous forme de tableau `numpy`.
    - ▹ `pyplot.imshow(image)` puis `pyplot.show()` pour visualiser l'image correspondante à un tableau `numpy` correctement formé pour représenter une image bitmap.

Script 2 : *création d'images aléatoires* — Niveau : 1

1. Expliquer la commande suivante :

```
img = np.array([[[0,0,0] for _ in range(h)] for _ in range(l)])
```

si `l,h` sont des entiers strictement positifs.

2. Écrire une fonction `creer_image` qui prend en entrée la hauteur et la largeur d'une image à créer et renvoie un tableau `numpy` représentant une image BitMaP dont les pixels sont constitués par des triplets de valeurs aléatoires entières dans $[\![0\,;\,255]\!]$.

3. Faire un test avec une image de taille 6 sur 4 pixels et visualiser à l'aide des commandes `plt.imshow(<tableau>)` et `plt.show()`.

Script 3 : *Niveaux de gris* — Niveau : 1

1. Récupérer dans le cahier de prépa le fichier python de base du TP 9 (`squelette.py`) ainsi que l'image à charger (`essai.bmp`) et la stocker dans votre dossier personnel. Modifier alors le nom du fichier dans la commande `plt.imread` pour pouvoir charger l'image que vous avez sauvée.
2. a. Vérifier le type de l'objet `image` puis afficher ses caractéristiques à l'aide de la méthode `shape` des tableaux `numpy`. Commenter.
   b. Évaluer la taille en Ko de cette image au format 24 bits à l'aide des données précédentes.
3. Les couleurs grises correspondent au cas où les trois couleurs sont représentées de manière identique. Modifier le tableau `image` en égalisant pour chaque pixel les trois couleurs à l'aide d'une moyenne :

$$c_r \times R + c_g \times G + c_b \times B$$

   où R, C, B sont les valeurs des trois entiers définissant la couleur du pixel et c_r, c_g, c_b des pondération telles que : $c_r + c_g + c_b = 1$.
4. Afficher l'image ainsi modifiée.

Script 4 : *floutage par convolution.* — Niveau : 2

Un *floutage par convolution* est une opération dans laquelle on recalcule la valeur des couleurs d'un pixel à l'aide d'une moyenne pondérée de celles des pixels contigus. Les coefficients permettant le calcul de la moyenne sont stockés dans une matrice :

$$G = \begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} \\ c_{1,0} & c_{1,1} & c_{1,2} \\ c_{2,0} & c_{2,1} & c_{2,2} \end{pmatrix}$$

pour s'appliquer au 9 pixels entrant en jeu dans le recalcul de la valeur du pixel centre $p_{l,c}$

$$A = \begin{pmatrix} p_{l-1,c-1} & p_{l-1,c} & p_{l-1,c+1} \\ p_{l,c-1} & p_{l,c} & p_{l,c+1} \\ p_{l+1,c-1} & p_{l+1,c} & p_{l+1,c+1} \end{pmatrix}$$

où (l, p) est le couple d'entiers repérant dans l'image le pixel central qu'on veut "flouter" à l'aide en en recalculant la valeur à l'aide de la formule :

$$G * A = \sum_{i=0}^{2} \sum_{j=0}^{2} c_{i,j} p_{l+i-1,c+j-1}.$$

L'opération $*$ étant appelée ***convolution*** des deux matrices.

1. Écrire une fonction `convolution` qui prend en entrée deux matrices de taille 3 comme ci-dessus et qui renvoie le résultat de leur produit de convolution.
2. Écrire une fonction `floutage` qui prend en entrée une image sous forme de tableau `numpy` et renvoie l'image traitée par convolution sous forme d'un tableau `numpy`. Pour simplifier, la fonction n'effectuera les opérations de floutage que pour les pixels qui ne sont pas au bord.
3. a. Évaluer le nombre de tours de boucles effectués par votre fonction en fonction des paramètre de l'image.
   b. Évaluer le nombre d'opérations élémentaires effectuées par votre fonction en fonction des paramètres de l'image.