R5.A.12 TP 3

Introduction aux traitements mathématiques des images

1er octobre 2025 – B. COLOMBEL

Dans ce TP, nous utiliserons la bibliothèque `scikit-image` disponible sur les postes de l'IUT dans la distribution `anaconda`. Le module `numpy` nous sera également utile.

```
import numpy as np // pour les maths
import matplotlib.pyplot as plt // pour charger et afficher les images
from skimage import data, io, exposure, filters, util // pour le traitement des images
```

1 Introduction à scikit-image

```
image = plt.imread('paris.png')
plt.imshow(image)
plt.show()
```

L'image importée est un vecteur de classe `numpy.ndarray` contenant des valeurs `uint8` de trois dimensions dont la forme est

HAUTEUR, LARGEUR, CANAL (ROUGE, VERT, BLEU)

```
print(np.shape(image))
```

- le premier coefficient est le nombre de lignes ;
- le second, le nombre de colonnes ;
- le troisième est le nombre de composantes associées à chaque pixel (3 pour une image couleur, une pour une image en niveau de gris)

```
im1[58, 70]
```

```
type(im1[58, 70])
```

Les élements de la matrices sont de type `uint8`, c'est-à-dire des entiers codés sur 8 bits.
Certaines bibliothèques accepte aussi les images normalisées, avec des coefficients compris entre 0 pour noir et 1 pour blanc. Cela peut parfois faciliter les calculs et la manipulation des images.

Le module `data` de `scikit-image` qui contient des images pré-chargées (images standards de démonstration qui peuvent être utilisées pour effectuer des tests).

```
im1 = data.astronaut()
im2 = data.coffee()
plt.imshow( im1 )
plt.show()

plt.imshow( im2 )
plt.show()
```

On peut aussi sauvegarder une image avec la syntaxe `plt.imsave('chemin', image)`.

1.1 Création d'images

Réalisons une image constituée de bandes verticales allant du noir (0) au blanc (255), en passant par les 256 nuances de gris possibles.
Pour cela, on écrit une matrice constituée de ℓ colonnes de 0, ℓ colonnes de 1, ℓ colonnes de 2, etc. où ℓ représente la largeur (en pixels) d'une bande de hauteur h.
Recopier cette fonction dans la cellule suivante de votre notebook Jupyter :

```
def degrade(h, l) :
    res = 255 * np.ones((h, 256*l), dtype=np.uint8)
    for i in range(h) :
        for j in range(1, 256*l+1) :
            res[i, j-1] = np.trunc((j-1)/l)  # Partie entière
    return res
```

puis afficher l'image `Mire` obtenue avec la commande `Mire = degrade(60, 2)` ; s'agissant d'une image en niveau d gris, penser à ajouter l'argument `cmap = 'gray'` dans la commande `imshow()`.

Exercice 1.

1. Créer une image en niveau de gris `X1` codée sur 8 bits ayant $M = 200$ lignes et $N = 300$ colonnes. Cette image a un fond gris clair et contient une bande horizontale de couleur noire et d'une largeur de 10 pixels.
2. Créer une image en niveau de gris `X2` codée sur 8 bits ayant $M = 200$ lignes et $N = 300$ colonnes. Cette image a un fond gris foncé et contient un carré gris clair de taille 30×30. Faire en sorte que le carré soit situé vers le milieu de l'image.

1.2 Image couleur

Lorque l'on prend une photo avec un appareil numérique, ayant par exemple deux millions de pixels, cela signifie qu'il comporte deux millions de capteurs ; chaque capteur mesure les quantités de lumière rouge, de lumière verte et de lumière bleue reçues, et les enregistre sous forme de nombres : ainsi à chaque capteur est associé un triplet (R,V,B) de nombres entiers compris entre 0 et 255, et à une image en couleur sont associées trois matrices (la première pour le rouge, la deuxième pour le vert et la dernière pour le bleu). Chaque matrice comporte, dans ce cas, deux millions de termes. Nous appellerons hypermatrice cet ensemble de trois matrices.

Exercice 2.

1. Charger l'image `test-couleurs.png`. L'appeler I_1.
2. Créer les trois matrices `R = I1[:, :, 0]`, `V = I1[:, :, 1]` et `B = I1[:, :, 2]` puis les afficher. À quoi correspondent ces trois matrices ?

Exercice 3. Écrire un script en `python` réalisant un damier alternant des pixels rouges, ayant pour coordonnées RVB : (255,0,0) et des pixels verts, ayant pour coordonnées RVB : (0,255,0). La matrice du rouge est donc une alternance de 255 et de 0, celle du vert de 0 et de 255, celle du bleu ne contient que des 0.
Les paramètres m et n représentent respectivement le nombre de lignes et le nombre de colonnes du damier.

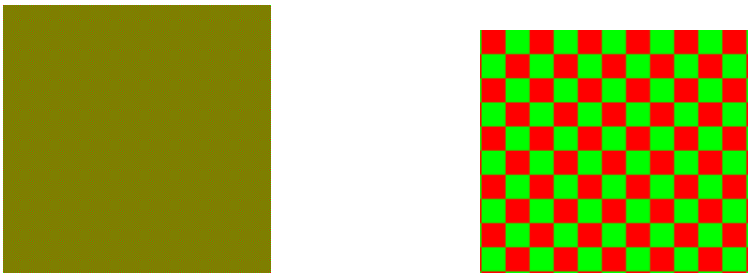

FIGURE 1 – Le damier et un zoom

2 Traitement ponctuel

2.1 Histogramme d'une image

L'histogramme représente la répartition des pixels selon leur intensité (de 0 à 255). Les trois canaux (rouge, vert et bleu) ont chacun un histogramme.

L'histogramme est créé avec la fonction `histogram()` du module `exposure`.

```
fig, axs = plt.subplots(nrows = 1, ncols = 3, figsize = (13,4))
canals = ['rouge', 'vert', 'bleu']

for i, canal in enumerate(canals) :
    hist = exposure.histogram( image[ : , : , i] )
    axs[i].plot(hist[0])
    axs[i].set_title(canal)
```

Exercice 4.

1. Ouvrir l'image `lena.png` et la stocker dans la variable `lena`.
2. La convertir en niveau de gris avec la commande :

   ```
   from skimage import color

   lena = color.rgb2gray(lena)
   ```

3. Exécuter la commande `print(lena)` pour vérifier que l'image a été renormalisée (tous les coefficients sont compris entre 0 et 1).
4. Afficher son histogramme.
5. Une image étant un tableau nous pouvons lui appliquer une fonction (bien définie) pour en modifier les niveaux de gris. Par exemple, si on peut utiliser les fonctions suivantes :

   $$f(x) = \sqrt{x} \quad \text{et} \quad g(x) = x^2$$

   Quel(s) effet(s) procure ces fonction sur l'image ?

6. Écrire une fonction `teinte(image, h)` qui ajoute la valeur h à chaque pixel de l'image image pour $h \in [-1;1]$ (attention à ne pas dépasser les valeurs autorisées!).
   Tester votre fonction avec quelques valeurs de h et afficher les nouveaux histogrammes.

2.2 Look Up Table

Soit $I(x,y)$ le niveau de gris du point P de l'image source et $I'(x,y)$ le niveau de gris de l'image résultat. L'opération ponctuelle réalise l'application suivante φ de $\mathbb{R}_+$ dans $\mathbb{R}_+$:

$$I'(x;y) = \varphi\left[I(x;y)\right]$$

Les coordonnées du point résultat sont supposée identiques à celle du point source dans cette étude. L'opération peut se représenter par un graphe ou une table (LUT).

La table LUT peut être utilisée comme suit :

Construction et utilisation d'une Look Up Table (LUT)

Entrées : Une image I de dimension $p \times n$;
Une transformation locale φ ;

```
début
    // Initialisation de la LUT
    pour i variant de 1 à 256 faire
        LUT(i) = φ(i) ;
    fin
    // Utilisation de la LUT
    pour i variant de 1 à p faire
        pour j variant de 1 à n faire
            I'(i,j) = LUT(I(i,j)) ;
        fin
    fin
fin
```

1. **Recadrage de l'histogramme**
   (a) Écrire une fonction qui modifie les niveaux de gris des pixels de telle sorte que la dynamique des niveaux de gris soit comprise entre 0 et 255 (recadrage dynamique).
   On pourra utiliser les fonctions `min(M)` (et `max(M)`) qui retourne la valeur minimale (et maximale) des coefficients de la matrice M.
   (b) Tester la fonction avec l'image `hotel-de-ville.pgm`.
   Afficher les deux histogrammes avant/après traitement.
2. **Égalisation de l'histogramme**
   (a) Écrire une fonction qui modifie les niveaux de gris des pixels par une égalisation d'histogramme et qui affiche l'histogramme et l'histogramme cumulé de l'image résultante.
   (b) Tester votre fonction sur l'image `port.pgm`.
   Afficher les deux histogrammes avant/après traitement.

Remarques. Dans le cas où les images ont été normalisées, on peut utiliser les formules suivantes :

1. La fonction de d'étalement dynamique est donnée par :
$$t(p) = \frac{p-a}{b-a}$$
où a et b représente les valeurs minimales et maximales de niveau de gris de l'image (renormalisée).

2. La fonction dé=’égalisation de l’histogramme est donnée par :

$$j = f(i) = \frac{1}{N_1 \times N_2} \sum_{p=0}^{i} h_p - 1$$

où N_1 et N_2 correspondent aux dimensions de l’image et f_i le nombre de pixel de l’image ayant i comme niveau de gris.

Dans le cas où les images n’ont pas été normalisées, il faut les multiplier par 256.

3 Masque de convolution

Beaucoup de traitement d’images reposent sur la convolution des matrices : chaque pixel de l’image est remplacé par une combinaison linéaire des ses pixels et de ses voisins.
Pour cela, on se donne un noyau appelé *masque de convolution*, qui est une matrice correspondant au poids de chaque pixel voisin dans cette combinaison linéaire lorsque l’on centre la matrice sur le pixel à modifier.
Par exemple, si K est une matrice 3×3 et $I_{i,j}$ le pixel à modifier, on va le changer en :

$$I'(i;j) = \sum_{O \leq k,\ell \leq 2} I_{i+k-1,j+\ell-1} K_{k,\ell}$$

Exemple 1. Avec le filtre $K = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, on obtient $I'(i;j) = -I_{i,j-1} + I_{i,j}$.

Les filtres de convolution les plus classiques sont implémentés dans le module `filters` de `scikit-image`.

⚠ Pour simplifier, nous n’utiliserons que des images **renormalisées** c’est-à-dire que l’on a multiplié les valeurs de chaque pixel pour qu’elles soient comprises entre 0 et 1.
Il suffit pour cela, si l’image est quantifiée sur b bits, de diviser l’image par $2^b - 1$ (qui est la valeur maximale possible pour un pixel).
Dans notre cas, travaillant sur 8 bits, on pourra utiliser la commande :

```
cameraman = io.imread(cameraman.pgm') // image en uint8
print(cameraman)
```

```
cameraman = cameraman / 2**7 // image normalisée
print(cameraman)
```

Exercice 5.

1. Charger l’image `lena_bruit.pgm` puis la visualiser/
2. (a) Explorer la page de scikit-image/filters pour utiliser les filtres susceptibles d’élimiber en partie le bruit de l’image (filtre moyenneur, filtre gaussien, filtre median).
   (b) Appliquer chacun de ces filtres à l’image `lena_bruit.pgm`.
   (c) Quel est le meilleur résultat? Était-ce prévisible?