Feuille de TP n°1

Licence 3 MApI3, module Signal, Fourier, image

On donne le fichier `imtools.py`, qui fournit quelques fonctions utiles pour la manipulation d'images. On peut le trouver à l'adresse `http://www.math.univ-toulouse.fr/~vfeuvrie/l3mapi3`.

Par « image », on entend un tableau numpy à deux dimensions qui contient des nombres compris entre 0 et 255. Voici quelques exemples d'utilisation.

Listing 1 – Chargement et affichage d'une image

```
from imtools import *

i = open_image('lena.png')
display_image(i)
```

Listing 2 – Affichage d'une image en négatif

```
from numpy import *

i = open_image('lena.png')
j = array([])
j.resize(i.shape)
for x in range(i.shape[0]):
    for y in range(i.shape[1]):
        j[x, y] = 255 - i[x, y]
display_image(j)
```

Listing 3 – Enregistrement d'une image

```
i = array([])
i.resize(64, 64)
for x in range(64):
    for y in range(64):
        i[x, y] = 3 * x + y
save_image(i, 'test.png')
```

Listing 4 – Affichage de plusieurs images sur la même fenêtre

```
i1 = open_image('lena.png')
i2 = open_image('barbara.png')
i3 = open_image('peppers.png')
i4 = open_image('mandrill.png')

#Affichage des quatre images sur 2 lignes et
    3 colonnes
plt.subplot(2, 3, 1)
display_image(i1)
plt.subplot(2, 3, 2)
display_image(i2)
plt.subplot(2, 3, 3)
display_image(i3)
plt.subplot(2, 3, 5)
display_image(i4)
```

Listing 5 – Accès par bloc

```
i = open_image('lena.png')
#Une bande horizontale noire
i[10:20, :] = 0
#Une bande verticale blanche
i[:, 50:80] = 255
#Accès à un pixel sur six
i[::2, ::3] = 128
display_image(i)
```

Exercice 1 – Fenêtrage et échantillonnage.

1) Effectuer et afficher le fenêtrage de l'image `lena.png` sur la grille $\{235, \dots, 295\} \times \{240, \dots, 300\}$.

2) Effectuer et afficher l'échantillonnage de l'image `barbara.png` aux facteurs 2, 3, 4 et 5.

3) Décrire qualitativement les résultats obtenus.

Exercice 2 – Filtrage. Il est relativement facile de créer de nouvelles fonctions en Python. Voici quelques exemples

Listing 6 – Fonctions numériques

```
def sqr(x):
    return x * x

def abs(x):
    if x < 0:
        return -x
    else:
        return x
```

Listing 7 – Fonctions sur tableaux

```
def gray_copy(i, val):
    j = array([])
    j.resize(i.shape)
    for x in range(j.shape[0]):
        for y in range(j.shape[1]):
            j[x, y] = val
    return j
```

1) Définir une fonction `moyenne(i, n)` qui renvoie la nouvelle image obtenue en faisant la convolution par une fenêtre carrée de taille $(2n+1) \times (2n+1)$. On effectuera la pondération nécessaire pour s'assurer que le résultat est à valeurs dans $[0, 255]$.

Pour périodiser l'accès aux pixels d'une image, il est possible d'utiliser l'opérateur « modulo » qui s'écrit avec le symbole % en Python. L'expression `a % b`, où a et b sont des entiers avec $b > 0$, renvoie l'unique entier de la forme $a + kb$ compris entre 0 et $b - 1$.

2) Réaliser et afficher la convolution pour les tailles $n = 1, 2, 3$ et 4.

3) Décrire qualitativement les résultats obtenus.

Exercice 3 – Filtrage et échantillonnage.

1) Définir une fonction `echantillonnage2x(i, n)` qui renvoie l'image obtenue après convolution par fenêtre de taille $(2n + 1)$, puis sous-échantillonnage d'un facteur 2.

2) Tester sur `barbara.png` avec des fenêtres pour les tailles $n = 1, 2, 3$ et 4.

3) Quelle taille préconiseriez-vous pour un sous-échantillonnage de facteur 2 ?

4) Adapter l'expérience des questions précédentes pour déterminer la meilleure taille de fenêtre pour un sous-échantillonnage de facteur 3.

Exercice 4 – Bruit. La fonction `random.normal(0, sigma, size=(N,N))` génère un bruit gaussien, centré en zéro, et d'écart-type σ sous forme d'une image de taille $N \times N$.

1) Créer une fonction `bruitage(i, sigma)` qui renvoie l'image i après ajout d'un bruit gaussien d'écart-type σ.

2) Visualiser l'effet du bruit sur `barbara.png` pour des valeurs d'écart-type de 2, 5, 10, 25, 50, 100 et 200.

3) Décrire qualitativement les résultats obtenus.

Exercice 5 – Quantification à pas constant. Pour l'affichage des images quantifiées, on utilisera la fonction `display_image` avec les paramètres supplémentaires `vmin=0` et `vmax=255`. Ceux-ci garantissent que la valeur 0 sera affichée comme du noir et 255 comme du blanc. Dans le cas contraire, Python procède à un ajustement automatique avec les valeurs minimales et maximales de l'image, ce qui n'est pas souhaitable ici.

1) Écrire une fonction `quantify1(i, n)` qui effectue la quantification de l'image i à pas constant sur n niveaux. On pourra utiliser l'expression `int(round(z))` qui, lorsque z est un nombre flottant, renvoie l'entier le plus proche.

2) Quantifier et visualiser les images `barbara.png` et `io.png` aux niveaux 5, 10, 25, 50, 100 et 200.

3) Décrire qualitativement les résultats obtenus.

Exercice 6 – Égalisation d'histogramme. La fonction `plt.plot(X, Y)`, où X et Y sont deux tableaux unidimensionnels, affiche le graphe de Y (en ordonnée) en fonction de X (en abcisse).

1) Écrire une fonction `histogramme(i)` qui renvoie l'histogramme cumulé d'une image i sous la forme d'un tableau à 256 éléments.

2) Représenter les histogrammes des images `lena.png`, `barbara.png` et `io.png`.

3) Implémenter l'algorithme d'égalisation d'histogramme vu en TD sous la forme d'une fonction `egalise(i)`.

4) Tester avec les images mentionnées plus haut et décrire qualitativement les résultats obtenus.

Exercice 7 – Quantification à pas variable.

1) Écrire une fonction `quantify2(i, n)` qui effectue une quantification sur n niveaux en découpant l'histogramme de l'image en n classes d'effectif comparables, et en attribuant à chaque classe sa valeur moyenne.

2) Quantifier avec cette méthode et visualiser les images `barbara.png` et `io.png` aux niveaux 5, 10, 25, 50, 100 et 200.

3) Comparer le bruit de quantification de cette méthode avec celle de l'exercice précédent.

4) Décrire qualitativement les résultats obtenus.