

## TD08 : Traitement d'images

### 1 Les modules de traitement d'image

Modules spécialisés : PIL ou pillow (Python Imaging Library) ; scikit-image.

Pages de référence :

- Pillow : <https://pillow.readthedocs.org/>
- Scikit-image : <http://scikit-image.org/>

Note rapide : Information sur les modules installés : WinPython Control Panel > uninstall packages (situé dans le répertoire d'installation de WinPython, par exemple C:\WinPython-64bit-3.3.5.0) ou bien :

```
>>> help('modules') #avec un long temps de chargement
```

#### 1.1 Les types de fichiers images

Un descriptif assez complet se trouve à la page :

<https://pillow.readthedocs.org/handbook/image-file-formats.html>

Ceci étant le module scikit-image (abrégié en `skimage`) permet de lire un grand nombre de formats d'images sans avoir à s'en préoccuper, par exemple par la commande `imread`.

#### 1.2 Manipulation d'une image avec `skimage`

##### 1.2.1 Chargement, sauvegarde, visualisation d'une image

Pour charger une image à partir d'un fichier on importera le sous-module `io` du module `skimage` :

```
>>> from skimage import io
```

Le chargement de l'image (quelque soit son type de départ .jpg, .bmp, etc.) se fait par l'instruction :

```
>>> im=io.imread('joconde.png')
```

L'image est alors chargée sous la forme d'un tableau `numpy` (`ndarray`) et pour la sauvegarder (sous un autre nom, par exemple, ou après traitement, au format .png nécessairement) :

```
>>> io.imsave('nouveau.png', im)
```

Pour visualiser l'image :

```
>>> io.imshow(im) #ou io.imshow(im[...,0]) pour afficher en niveaux de gris
>>> io.show() #imshow(im) prépare, show() affiche dans une fenêtre à l'écran
```

##### 1.2.2 Caractéristiques d'une image (tableau `numpy`)

Ces caractéristiques sont données par les attributs `.shape` (format de la matrice), `.size` (nombre d'éléments), `.dtype` (type des éléments).

```
>>> f=io.imread('SN REFSDAL_hs-2015-08-c-full.tif.tif')
>>> type(f),f.dtype,f.shape,f.size
(<class 'numpy.ndarray'>, dtype('uint8'), (940, 940, 3), 2650800)
```

Les tableaux `numpy` pour une image sont des tableaux à trois dimensions la troisième correspondant à la couleur (trois composantes, R, V, B, codées chacune sur 256 bits ou par un flottant entre 0 et 1) :

```
>>> f
array([[[ 64,  59,  38],
        [ 64,  59,  38],
        [ 64,  59,  38],
        ...,
        [ 36,  32,  34],
        [ 36,  32,  34],
        [ 36,  32,  34]]], dtype=uint8)
```

La conversion se faisant par :

```
>>> import skimage as sk
>>> ff = sk.img_as_float(f)
#et inversement f= sk.img_as_uint(ff)
```

```
>>> ff
array([[[ 0.25098039,  0.23137255,
 0.14901961],
        [ 0.25098039,  0.23137255,
 0.14901961],
        [ 0.25098039,  0.23137255,
 0.14901961],...,
        [ 0.14117647,  0.1254902 ,
 0.13333333],
        [ 0.14117647,  0.1254902 ,
 0.13333333],
        [ 0.14117647,  0.1254902 ,
 0.13333333]]])
```

**Exercice 1.** Créer, en respectant sa proportion de 8:5, une image du drapeau de la Suède, dont les caractéristiques sont données ci-dessous

Les dimensions du drapeau suédois sont de 5:2:9 horizontalement et 4:2:4 verticalement. On prendra une largeur de 320 pixels, et donc une hauteur de 200 pixels et des bandes de 40 pixels de large.

Les couleurs du drapeau ont officiellement les côtes suivantes dans le « Natural Color System » :

NCS 0580-Y10R pour le jaune (RGB(249, 205, 48)),  
NCS 4055-R95B pour le bleu (RGB(22, 101, 161)).



### 1.2.3 Conversion en niveaux de gris

On convertit une image en niveaux de gris en prenant la moyenne des trois composantes R, G, B.

L'opération est facile si l'on utilise les fonctions adaptées de `numpy` :

```
>>> import skimage as sk ; from skimage import io ; import numpy as np
>>> imRGB = io.imread('0_monalisaRGB.jpg')
>>> print(type(imRGB), imRGB.shape, imRGB.size, imRGB.dtype)
<class 'numpy.ndarray'> (1024, 687, 3) 2110464 uint8
>>> imNG = np.mean(im, axis=2)
>>> print(type(imNG), imNG.shape, imNG.size, imNG.dtype)
<class 'numpy.ndarray'> (1024, 687) 703488 float64
```

**Exercice 2.** Écrire une fonction `convertNG` qui prend en argument une image en couleurs et renvoie cette image convertie en une image en niveaux de gris, sans recourir à la fonction `np.mean`.

Note : on pourra utiliser les [coefficients recommandés](#) :

$$\text{Intensité} = 0.2126 \cdot \text{rouge} + 0.7152 \cdot \text{vert} + 0.0722 \cdot \text{bleu}$$

**Exercice 3.** Écrire une fonction `negatifNG` qui prend en argument une image en niveaux de gris et renvoie le négatif de cette image.

### 1.2.4 Conversion en noir et blanc

Il s'agit ici de convertir une image en niveaux de gris en une image en noir et blanc (pour une image en `'float64'`, les pixels sont à 0 ou 1, pour une image en `'uint8'`, les pixels sont à 0 ou 255).

#### 1.2.4.a Conversion par seuillage

Une première façon de faire consiste à définir un seuil de gris en deçà duquel les pixels sont mis à 0 et au-delà duquel ils sont mis à 1 (ou 255).

**Exercice 4.** Écrire une fonction `NBseuil` prenant en argument une image en niveaux de gris au format `'uint8'` (format par défaut des images chargées par la fonction `io.imread`) et une valeur de seuil (entre 0 et 255) et renvoyant cette image convertie en noir et blanc par seuillage.

#### 1.2.4.b Conversion par tramage par diffusion d'erreur

En commençant en haut à gauche de l'image, on met le pixel à 0 ou à 1 (255) par rapport à un seuil fixé  $\sigma$ . Si le pixel était à la valeur  $k$  et est mis à la valeur  $k'$ , on commet une erreur  $\delta = k - k'$ , que l'on « diffuse » en ajoutant la moitié de l'erreur au pixel situé à sa droite, et le quart de l'erreur au pixel au-dessous et au pixel au-dessous et à droite. On réitère alors le procédé sur le pixel à droite, et ainsi de suite, en traitant les lignes les unes après les autres.

**Exercice 5.** Écrire une fonction `tramage` qui implémente ce procédé.

### 1.2.5 Filtrage



La plupart des opérations élémentaires en traitement d'image peuvent se réduire à l'application de filtres à la matrice des intensités de l'image. Un filtre  $F$  se présente sous la forme d'une matrice  $n \times p$  de coefficients (usuellement et ici :  $n = p = 3$ ) que l'on applique, par *convolution*, à la matrice  $A$  représentant l'image, le produit de convolution  $A$  étant défini par :

$$A_{i,j}^f = (A * F)_{i,j} = \sum_{k=0}^2 \sum_{l=0}^2 A_{i-1+k, j-1+l} F_{k,l}.$$

### 1.2.5.a Filtrage par floutage

**Exercice 6.** Écrire une fonction qui implémente l'application du filtre uniforme  $F$ , défini ci-dessous à une image :

$$F = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Filtre uniforme sur une image faible résolution	Filtre uniforme sur une image haute résolution
	

## 1.2.6 Pixellisation : réduction et agrandissement

### 1.2.6.a Réduction

On souhaite compresser une image en couleur en remplaçant quatre pixels contigus et formant un carré, par un seul pixel, pour lequel les intensités suivant les trois canaux de couleurs sont la moyenne des niveaux de couleurs sur les quatre pixels voisins remplacés.

**Exercice 7.** Implémenter cet algorithme sous la forme d'une fonction `réduction4`, prenant en argument une matrice `numpy` correspondant à une image en couleur et renvoyant la matrice `numpy` correspondant à l'image réduite (en couleur). On donnera une implémentation utilisant un parcours explicite de l'image et une implémentation utilisant les opérations sur les matrices `numpy`. Tester avec une image au format bitmap (.bmp) le gain en mémoire.

### 1.2.6.b Agrandissement

On souhaite agrandir une image en couleur en intercalant entre chaque ligne et chaque colonne une nouvelle ligne et une nouvelle colonne. Les intensités suivant chaque canal de couleur pour les nouveaux pixels seront égaux à la moyenne des intensités des deux, ou quatre, anciens pixels qu'ils touchent.

**Exercice 8.** Implémenter cet algorithme sous la forme d'une fonction `agrandissement4`, prenant en argument une matrice `numpy` correspondant à une image en couleur et renvoyant la matrice `numpy` correspondant à l'image agrandie (en couleur). On donnera une implémentation utilisant un parcours explicite de l'image.

## 1.2.7 Histogramme des couleurs

On souhaite produire les histogrammes des niveaux de couleurs suivant chaque canal pour une image en couleurs. On utilise un nombre  $n$  de classes de même largeur pour les niveaux de couleurs.

**Exercice 9.** Q9.1. Implémenter la construction des listes des effectifs des classes pour chaque couleur. Q9.2. À l'aide des instructions usuelles, implémenter la construction effective de l'histogramme à l'aide de `matplotlib`.

Q9.3. Implémenter la construction de l'histogramme en utilisant la fonctionnalité dédiée de `matplotlib`, dont la description peut être trouvée ici : <http://www.python-simple.com/python-matplotlib/histogram.php>.

### 1.2.8 Gradient de couleur et détection de contours

Chaque pixel de l'image présentant une intensité notée  $I(i, j)$ , on peut calculer le gradient d'intensité selon les deux directions, notées  $x$  et  $y$ , de l'image.

On utilise ici l'algorithme de Sobel dans lequel le gradient  $G_x$ , selon  $x$  (respectivement  $y$ ), est évalué par rapport à l'écart d'intensité entre les pixels encadrant le pixel en position  $i$  (respectivement  $j$ ), ainsi :

$$G_x(i, j) = \frac{I(i+1, j) - I(i-1, j)}{2} ; \quad G_y(i, j) = \frac{I(i, j+1) - I(i, j-1)}{2}.$$

Afin d'uniformiser les valeurs obtenues (pour réduire l'effet des bruits) on utilise une pondération pour effectuer une moyenne des valeurs du gradient sur 3 pixels adjacents dans la direction orthogonale à celle du gradient évalué, ainsi les valeurs des gradients selon les deux directions pour le pixel  $(i, j)$  sera donné par les formules :

$$G_x^m(i, j) = \frac{G_x(i, j-1) + 2G_x(i, j) + G_x(i, j+1)}{4} ; \quad G_y^m(i, j) = \frac{G_y(i-1, j) + 2G_y(i, j) + G_y(i+1, j)}{4}.$$

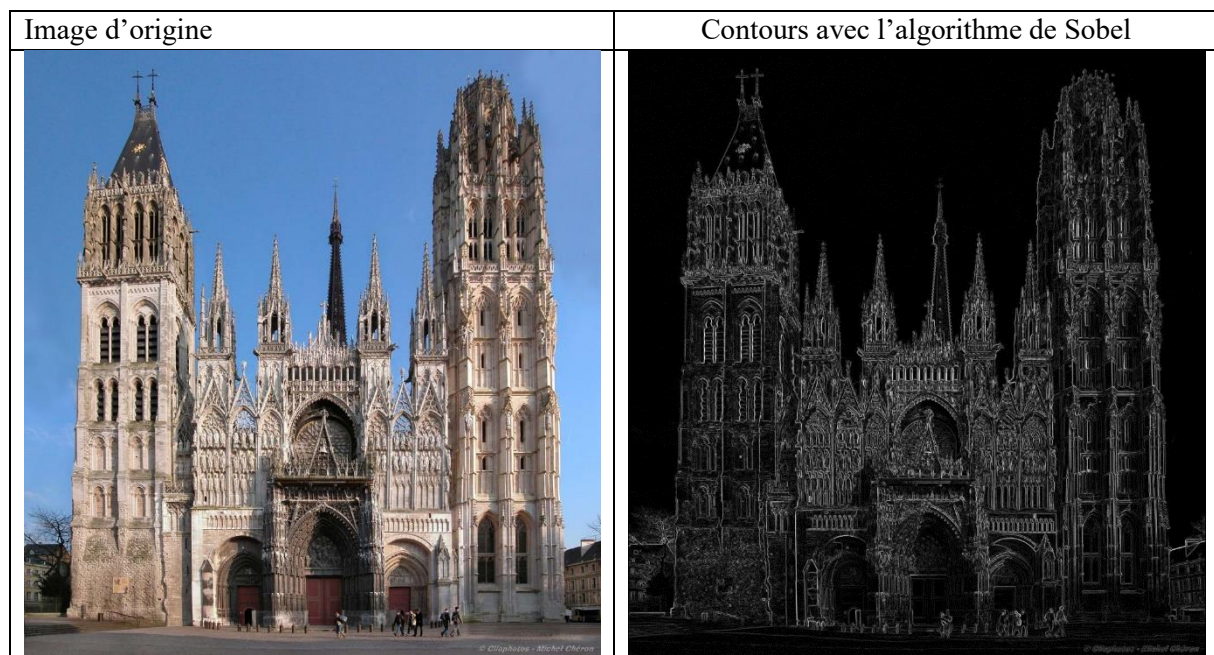
Finalement, en reprenant ces formules, le calcul des gradients selon les deux directions peut se résumer à l'application à l'image de deux filtres, respectivement :

$$F_X = \frac{1}{8} \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} = \frac{1}{8} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \begin{pmatrix} -1 & 0 & 1 \end{pmatrix} \quad \text{et} \quad F_Y = \frac{1}{8} \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ -1 & 2 & 1 \end{pmatrix} = \frac{1}{8} \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 1 \end{pmatrix}.$$

Filtre pour calculer le gradient selon les abscisses      Filtre pour calculer le gradient selon les ordonnées

Afin d'effectuer la détection de contours, on calcule enfin, la matrice  $S$  dont les coefficients,  $S(i, j)$ , sont définis par :

$$S(i, j) = \sqrt{G_x^m(i, j)^2 + G_y^m(i, j)^2}.$$



### 1.2.9 Détection de droites

Utiliser le filtre de Canny du module scikit-image :

[http://scikit-image.org/docs/dev/auto\\_examples/plot\\_canny.html](http://scikit-image.org/docs/dev/auto_examples/plot_canny.html)

pour obtenir les contours de l'objet dans l'image factory-1880261\_960\_720.jpg.

Puis déterminer les droites formant le contour de l'objet photographié à l'aide de la transformée de Hough :

- <http://www.f-legrand.fr/scidoc/srcdoc/opencv/math/hough/hough-pdf.pdf>
- <http://elynxsdk.free.fr/ext-docs/Demosaicing/more/news1/article-graf-espice.pdf>