Traitement d'images (Partie 3)

Objectifs

- comprendre le fonctionnement d'un filtre de convolution
- travailler les listes de listes
- appliquer un filtre de convolution à une image
- mettre en œuvre un filtre de détection de contours
- savoir introduire puis réduire du bruit numérique dans une image
- aplatir, trier et déterminer la médiane d'une liste

Connaissances et savoir-faire à maîtriser

Nous avons vu dans le TP précédent une technique simple pour appliquer un filtre à une image. Il existe une autre technique donnant des résultats très intéressants mais plus difficile à mettre en œuvre : **le filtrage par convolution**.

Remarque 1 *: par souci de simplification, nous allons considérer des images en niveaux de gris, ce qui signifie que le contenu d'un pixel est une valeur entre 0 et 255.*

1) Définition du filtrage par convolution

Pour appliquer un tel filtre à une image, considérons :

- un pixel quelconque noté p d'une image contenant une certaine valeur de gris que l'on souhaite modifier ;
- une matrice carrée de dimension 3×3 qu'on appellera le **noyau** (on dit aussi **masque** ou **matrice**) de convolution et qu'on notera K pour *kernel* ou *noyau* en français ;
- une matrice V de même taille qui contiendra les valeurs de gris des pixels au voisinage de p.

Le principe du filtrage par convolution est de combiner linéairement les valeurs de K et de V pour obtenir la nouvelle valeur de gris du pixel p.

Par exemple, sur le schéma ci-dessous, le pixel encadré en rouge est le pixel p dont on souhaite modifier la valeur. Celle-ci, avant application du filtre, est égale à 127 et les valeurs des pixels voisins sont indiquées en noir et sont les coefficients de la matrice V. Le carré bleu quant à lui représente la matrice K de convolution, ses coefficients ici sont tous égaux à 1.

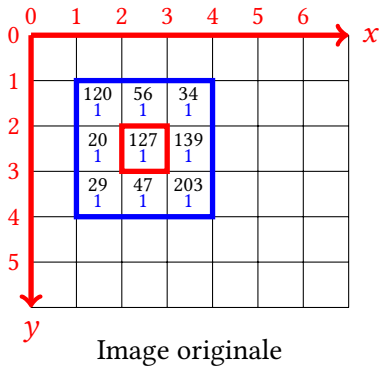

Image originale

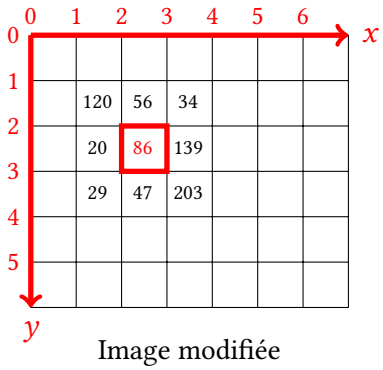

Image modifiée

Si on note $K * V$ le produit de convolution de K par V celui-ci se calcule en sommant les produits de chaque coefficient de K par le coefficient correspondant de V :

$$K * V = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 120 & 56 & 34 \\ 20 & 127 & 139 \\ 29 & 47 & 203 \end{bmatrix}$$
$$= 120 \times 1 + 56 \times 1 + 34 \times 1 + 20 \times 1 + 127 \times 1 + 139 \times 1 + 29 \times 1 + 47 \times 1 + 203 \times 1 = 775$$

La valeur obtenue étant supérieure à 255, on dit qu'on la **normalise** en la divisant par la somme des coefficients positifs de K (on divise donc par 9 dans notre exemple) et on obtient donc $775 \div 9 = 86{,}11$ soit un code de gris égal à 86 que l'on affecte à la valeur du pixel p de la nouvelle image.

On répète cet algorithme sur chaque pixel de l'image originale mais là encore, par souci de simplification, on ne traitera pas les pixels en bordure de l'image du fait que le masque de convolution déborderait alors des limites de celle-ci.

2) Mise en œuvre en Python

Il est nécessaire dans un premier temps de convertir l'image en niveaux de gris. Vous pouvez utiliser la fonction développée dans le premier TP sur le traitement d'images ou utiliser une méthode d'image mise à disposition par la librairie Pillow et qui s'avère bien plus rapide à l'usage :

```
from PIL import Image, ImageOps
img = Image.open("mon-image.jpg")
img_gris = ImageOps.grayscale(img)
```

a. La matrice V

Comme nous l'avons vu dans le TP sur les listes, une matrice peut-être codée via une liste de listes. On récupère alors les niveaux de gris des pixels au voisinage de p via la fonction :

```
def voisins(pimg, px, py):
    """
    Description : construit la matrice carrée 3 par 3 des voisins du pixel (px, py)
    Paramètres  : type(pimg) => PIL.Image ; type(px, py) => (int, int)
    Retour      : liste de listes 3 par 3
    """
    V = [] # matrice des voisins
    for l in range(-1, 2): # 3 lignes indexées de -1 à 1
        ligne_V = [] # chaque ligne de V est une liste
        for c in range(-1, 2): # 3 colonnes indexées de -1 à 1
            ligne_V.append(pimg.getpixel((px + c, py + l)))
        V.append(ligne_V) # on ajoute la ligne à V
    return V
```

On peut aussi utiliser une fonction plus explicite :

```
def voisins(im, x, y):
    """
    Description : construit la matrice carrée 3 par 3 au voisinage du pixel (x, y)
    Paramètres  : type(im) => PIL.Image ; type(x, y) => (int, int)
```

```
    Retour      : liste de listes 3 par 3
    """
    return  [
            [im.getpixel((x-1,y-1)), im.getpixel((x,y-1)), im.getpixel((x+1,y-1))],
            [im.getpixel((x-1,y  )), im.getpixel((x,y  )), im.getpixel((x+1,y  ))],
            [im.getpixel((x-1,y+1)), im.getpixel((x,y+1)), im.getpixel((x+1,y+1))]
            ]
```

b. La matrice K

Vous pourrez tester dans les exercices les différents noyaux de convolution suivants :

```
# Flou                  # Flou Gaussien          # Réhausseur de contraste
K_f = [[1, 1, 1],       K_g = [[1, 2, 1],       K_c = [[ 0, -1,  0],
       [1, 1, 1],              [2, 4, 2],              [-1,  5, -1],
       [1, 1, 1]]              [1, 2, 1]]              [ 0, -1,  0]]
```

Remarque 2 : vous veillerez à ce que la valeur du pixel modifié soit bien dans l'intervalle $[0\,;255]$.

3) Filtres de détection de contours

Il existe plusieurs filtres de ce type, nous allons étudier dans ce TP le filtre de *Sobel* qui donne des résultats très intéressants comme l'illustre l'exemple ci-dessous :

FIGURE 1 – Original

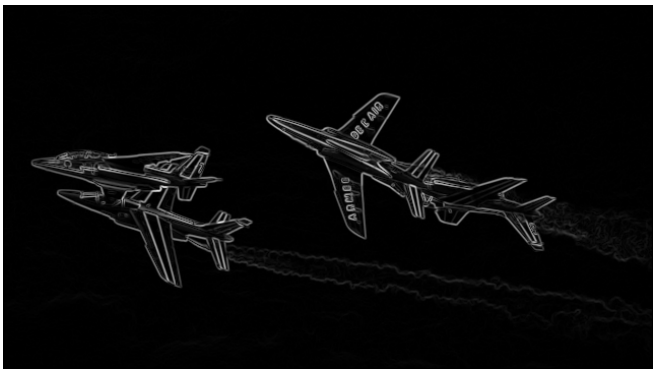

FIGURE 2 – Sobel

a. Principe général

Le fonctionnement d'un tel filtre repose sur sa capacité à pouvoir détecter de brusques variations de niveaux de gris à l'intérieur d'une image afin de déduire les bords d'un objet ou d'une forme à l'intérieur de celle-ci.

Remarque 3 : en mathématiques, l'opérateur permettant de calculer cette variation d'intensité et dans quel sens varie cette intensité (du clair vers le sombre ou inversement) se nomme ***le gradient****.*

b. Mise en œuvre

On définit d'abord les deux matrices de convolution suivantes :

```
# Sobel en x                # Sobel en y
Sx = [[-1, -2, -1],         Sy = [[-1, 0, 1],
      [ 0,  0,  0],               [-2, 0, 2],
      [ 1,  2,  1]]               [-1, 0, 1]]
```

Puis pour chaque pixel p de l'image, on charge son voisinage V et on calcule le gradient horizontal, noté gx, et vertical, noté gy :

```
gx = int(convolution(V, Sx) / 4) # 4 est la somme des coefficients positifs
gy = int(convolution(V, Sy) / 4)
```

Enfin, on calcule le module du gradient et on l'affecte au pixel de la nouvelle image :

```
g = int(math.sqrt(gx * gx + gy * gy)) # g = sqrt(gx^2 + gy^2)
img.putpixel((x, y), g)
```

4) Filtre de réduction de bruit

Le **bruit numérique** est une dégradation que subit l'image lors de son acquisition ou de sa numérisation. Il se traduit par la présence de pixels indésirables dans des zones aléatoires de l'image.
Il existe un filtre capable d'éliminer ce bruit, c'est le filtre médian. Son fonctionnement est quelque peu différent des précédents puisqu'il ne fait pas intervenir de masque de convolution. Si on considère un pixel p et son voisinage V alors la valeur de p est la médiane de la liste triée des voisins de p. Dans notre exemple du début de TP, la liste triée des voisins de p est :

$$20 - 29 - 34 - 47 - \boxed{56} - 120 - 127 - 139 - 203$$

La médiane est la valeur centrale, soit 56.

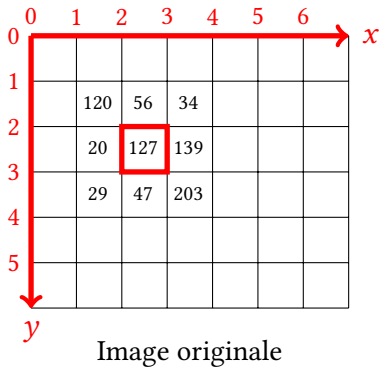

Image originale

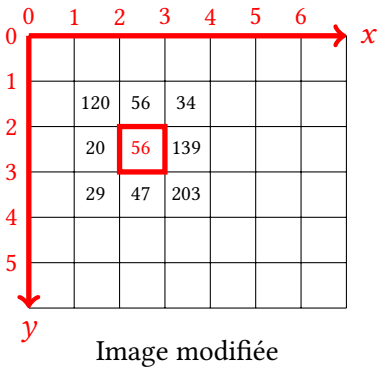

Image modifiée

Remarque 4 *: le voisinage de p étant stocké dans une liste de listes, il faudra veiller à transformer cette liste de listes en une liste unique avant de la trier (on dit qu'on aplatit la liste).*

```
V = [[120, 56, 34], [20, 127, 139], [29, 47, 203]]
V_liste = []
for sous_liste in V: # pour chaque ligne de la matrice V
    for val in sous_liste: # pour chaque valeur de la sous-liste considérée
        V_liste.append(val) # on ajoute cette valeur à la liste unique
# on obtient V_liste = [120, 56, 34, 20, 127, 139, 29, 47, 203]
V_liste.sort() # on trie la liste obtenue
```

Exercices à réaliser

L'objectif est de coder l'ensemble des filtres étudiés dans ce TP. Vous pouvez, si vous vous en sentez capable, vous lancer directement dans ce projet ou bien suivre la progression proposée dans les exercices qui suivent. Vous pouvez travailler sur l'image de votre choix ou utiliser le fichier image `paf.png`. Attention toutefois à bien convertir votre image en niveaux de gris avant d'appliquer les différents filtres.

Exercice 1
Ouvrir le fichier `TP-images-partie3-exo1.py` et répondre aux questions suivantes :

1. Convertir votre image de test en niveaux de gris en utilisant la méthode d'image disponible dans la librairie Pillow. Mettre en place un mécanisme de gestion des exceptions au cas où le fichier image spécifié n'existerait pas :

```
def sauver_gris(pname):
    """
    Description : convertit en gris le fichier image nommé pname et sauvegarde
                  le résultat en préfixant le nom avec gris-
    Paramètres  : type(pname) => str
    Retour      : aucun
    """
```

2. Écrire une fonction qui renvoit la matrice des voisins d'un pixel.
3. Écrire une fonction qui renvoit le produit de convolution des deux matrices K et V.
4. Écrire une fonction qui applique un filtre de flou et un filtre réhausseur de contraste à une image :

```
def filtrer(K, pnorme, pname):
    """
    Description : applique le filtre défini par le noyau K sur le fichier
                  image pname ; pnorme vaut la somme des coefficients
                  positifs de la matrice K
    Paramètres :  type(K, V) => listes de listes 3 par 3
                  type(pnorme) => int
                  type(pname) => str
    Retour :      Image.PIL ou None si le chargement a échoué
    """
```

Figure 3 – Flou

Figure 4 – Contraste ↑

Exercice 2

Ouvrir le fichier `TP-images-partie3-exo2.py` et répondre aux questions suivantes :

1. Écrire une fonction qui applique le filtre de Sobel à une image.
2. Écrire une fonction qui applique le filtre de réduction de bruit à une image.
   Remarques : l'image originale n'étant pas bruitée, le résultat obtenu n'est guère convaincant et n'a pas grand intérêt puisqu'il dégrade l'image originale. Pour comprendre l'intérêt du filtre réducteur de bruit on peut :
   a) *soit l'appliquer à une image bruitée comme par exemple une vieille photo numérisée ;*
   b) *soit introduire volontairement du bruit dans notre image originale en changeant de manière aléatoire la valeur de quelques pixels de l'image ;*
3. Écrire une fonction qui introduit aléatoirement du bruit dans l'image originale sous forme de pixel dont le niveau de gris est lui-même aléatoire puis appliquer le filtre réducteur de bruit.

Figure 5 – Image bruitée à 5%

Figure 6 – Réduction du bruit

Compétences à valider

Encodage		☼	⛅	☁	🌧
Théorie	J'ai compris le fonctionnement d'un filtre de convolution				
	Je sais définir une matrice et un produit de convolution				
	J'ai compris le rôle de l'opérateur gradient				
Pratique	Je sais manipuler une matrice sous la forme d'une liste de listes				
	Je sais charger le voisinage d'un pixel dans une matrice				
	Je sais appliquer un filtre de convolution à une image				
	Je sais introduire du bruit numérique dans une image				
	Je sais aplatir une liste				
	Je sais déterminer la médiane d'une liste				