|  |  |  |
| --- | --- | --- |
| **UNIVERSITE BADJI Mokhtar– ANNABA**  **FACULTE DES SCIENCES DE L’INGENIORAT**  **DEPARTEMENT D’INFORMATIQUE** |  | Intelligence Artificielle et Traitement de l’information **(IATI)**  Méthodes de Prétraitement de l’Information **(MPI)**  Réalisé par **: A. Melouah** |

**Filtrage d’Images**

**Introduction**

Le traitement d’images en python est possible à travers plusieurs packages comme Pillow ou OpenCV. Dans ce cours on s’intéressera particulièrement au package Pillow. Fréquemment, d’autres package sont sollicités avec Pillow comme ceux figurant dans le tableau ci-dessous.

|  |  |
| --- | --- |
| **Package** | **Utilisation** |
| * Pillow – utilisation des images * Numpy – utlisation des matrices * Matplotlib – utilisation des graphes | * **from** PIL **import** Image * import numpy as np * **from** pylab **import** \* |

Les actions d’ouverture et de conversion image-tableau sont :

|  |  |
| --- | --- |
| **Action** | **Exemple** |
| * Ouvrir une image * Affichage * Conversion image 🡪 matrice * Conversion image 🡨 matrice | * ima=Image.open(‘image.jpg’) * ima # par console anaconda   ima.show() # par le system  imshow(ima) # par Matplotlib   * imtab=np.array(im) # par numpy * im=Image.fromarray(imtab) # par pillow |

Le module **ImageFilter** permet d'appliquer un filtre sur une image.

from PIL import Image

from PIL import ImageFilter

**1.  Les filtres Kernel**

**1.1  PRÉSENTATION**

Les filtres suivants sont implantés :

**KERNEL(SIZE, KERNEL, SCALE=NONE, OFFSET=0)**

Crée un noyau de convolution de la taille donnée. La taille (**size**) de ce noyau doit être (3, 3) ou (5, 5), et l'argument **kernel** doit être un tuple contenant 9 ou 25 nombres, entiers ou réels.

* Si l'argument **scale** est fournis, le résultat de l'application du noyau à chaque pixel sera divisé par cette valeur d'échelle. L'échelle par défaut correspond à la somme de tous les poids du noyau.
* Si l'argument **offset** est donné, sa valeur sera ajoutée au résultat précédent.

import Image

import ImageFilter

image = Image.open('ima.jpg')

im2 = image.filter(ImageFilter.Kernel((3,3),

(-2, -2, -2, -2, 32, -2, -2, -2, -2),

scale = 16,

offset = 0))

im2.show()

**1.2  L'ATTRIBUT FILTERARGS**

L'attribut **filterargs** permet de récupérer les différents attributs :

ImageFilter.Kernel((3, 3),

(-1, -1, -1, -1, 10, -1, -1, -1, -1),

offset = 0,

scale=2).filterargs

((3, 3), 2, 0, (-1, -1, -1, -1, 10, -1, -1, -1, -1))

Remarquer que l'attribut **filterargs** range les attributs dans l'ordre suivant (différent de Kernel) : **size**, **scale**, **offset**, et enfin **kernel**.

**2.  Les noyaux prédéfinis**

Plusieurs noyaux de convolution sont prédéfinis :

**2.1  IMAGEFILTER.BLUR**

Filtre de flou. Il correspond aux arguments suivants passés à Kernel

((5, 5), 16, 0, (1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1))

Exemple d'utilisation :

im1 = image.filter(ImageFilter.BLUR)

**2.2  IMAGEFILTER.CONTOUR**

Filtre d'accentuation des contours. Il correspond aux arguments suivants passés à **Kernel**

((3, 3), 1, 255, (-1, -1, -1, -1, 8, -1, -1, -1, -1))

**2.3  IMAGEFILTER.DETAIL**

Filtre d'accentuation des détails. Il correspond aux arguments suivants passés à **Kernel**

((3, 3), 6, 0, (0, -1, 0, -1, 10, -1, 0, -1, 0))

**2.4  IMAGEFILTER.EDGE\_ENHANCE**

Marquer les bords des objets de l'image. Il correspond aux arguments **Kernel** suivants...

((3, 3), 2, 0, (-1, -1, -1, -1, 10, -1, -1, -1, -1))

**2.5  IMAGEFILTER.EDGE\_ENHANCE\_MORE**

Marquer plus encore les bords. Il correspond aux arguments **Kernel** suivants...

((3, 3), 1, 0, (-1, -1, -1, -1, 9, -1, -1, -1, -1))

**2.6  IMAGEFILTER.EMBOSS**

Créer du relief. Il correspond aux arguments **Kernel** suivants...

((3, 3), 1, 128, (-1, 0, 0, 0, 1, 0, 0, 0, 0))

**2.7  IMAGEFILTER.FIND\_EDGES**

Trouver les bords. Il correspond aux arguments **Kernel** suivants...

((3, 3), 1, 0, (-1, -1, -1, -1, 8, -1, -1, -1, -1))

**2.8  IMAGEFILTER.SMOOTH**

Lisser l'image. Il correspond aux arguments **Kernel** suivants...

((3, 3), 13, 0, (1, 1, 1, 1, 5, 1, 1, 1, 1))

**2.9  IMAGEFILTER.SMOOTH\_MORE**

Lisser plus encore. Il correspond aux arguments **Kernel** suivants...

((5, 5), 100, 0, (1, 1, 1, 1, 1, 1, 5, 5, 5, 1, 1, 5, 44, 5, 1, 1, 5, 5, 5, 1, 1, 1, 1, 1, 1))

**2.10  IMAGEFILTER.SHARPEN**

Accentuer l'image. Il correspond aux arguments **Kernel** suivants...

((3, 3), 16, 0, (-2, -2, -2, -2, 32, -2, -2, -2, -2))

**3.  Les filtres RankFilter**

**3.1  RANKFILTER(SIZE, RANK)**

Crée un filtre rang de la taille **size**. Pour chaque pixel de l'image d'origine, le filtre rang trie tous les pixels dans le voisinage **(size,size)** du pixel, et copie la valeur en position **rank** dans l'image de sortie.

import Image

from numpy import \*

tt=Image.new('L',(5,5))

tt.putdata(array(range(25)))

array(tt.getdata()).reshape((5,5))

array([[ 0, 1, 2, 3, 4],

[ 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14],

[15, 16, 17, 18, 19],

[20, 21, 22, 23, 24]])

import ImageFilter

im = tt.filter(ImageFilter.RankFilter(3,1)) # La taille ne peut être que 3 ou 5

array(im.getdata()).reshape((5,5))

array([[ 0, 0, 1, 2, 3],

[ 0, 1, 2, 3, 4],

[ 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14],

[15, 16, 17, 18, 19]])

Explications : prenons le cas du coefficient (2,1) de tt (à savoir : 11). Si on regarde dans son voisinage de taille 3, on a :

[[ 5, 6, 7],

[10, 11, 12],

[15, 16, 17]]

Quand on trie ces valeurs, on trouve : [5, 6, 7, 10, 11, 12, 15, 16, 17]. L'élément en position 1 est donc 6 : c'est cette valeur qui se retrouve dans *im* en position (2,1).

**3.2  MINFILTER(SIZE=3)**

Crée un filtre minimum de la taille donnée. Chaque pixel de l'image originale est replacé, avec ce filtre, par la plus petite valeur qui se trouve dans son voisinage de taille **size**. Ce filtre est un cas particulier, avec pour rang 0.

import Image

image = Image.open('cover.jpeg')

import ImageFilter

im2 = image.filter(ImageFilter.MinFilter(3))

im2.show()

**3.3  MEDIANFILTER(SIZE=3)**

Crée un filtre médian de la taille voulue. Chaque pixel de l'image de départ est remplacée par la valeur médiane des pixels de son voisinage **(size, size)**.

**3.4  MAXFILTER(SIZE=3)**

Crée un filtre maximum (*c.f.* filtre **MinFilter**)

**4.  Le filtre ModeFilter**

**4.1  MODEFILTER(SIZE=3)**

Crée un filtre mode de la taille voulue : chaque pixel de l'image de départ est remplacé par le mode (**i.e.** la valeur la plus fréquente) de son voisinage **(size, size)**. Si aucun pixel n'apparaît plus de deux fois dans ce voisinage, pas de modification.