L'imagerie

Il existe deux types de format :

- – Le format matriciel (jpg, gif, png,bitmap…) ;
- – Le format vectoriel (ai, eps).

Image vectorielle

Image matricielle

Une image vectorielle est redimensionnable sans perte de qualité, contrairement à une image matricielle

Image vectorielle

Une **image vectorielle** (ou **image en mode trait**), en informatique, est une image numérique composée d'objets géométriques individuels, des primitives géométriques (segments de droite, arcs de cercle, courbes de Bézier, polygones, etc.), définis chacun par différents attributs (forme, position, couleur, remplissage, visibilité, etc.) et auxquels on peut appliquer différentes transformations (homothéties, rotations, écrasement, mise à l'échelle, extrusion, inclinaison, effet miroir, dégradé de formes, morphage, symétrie, translation, interpolation, coniques ou bien les formes de révolution). Elle se différencie en cela des images matricielles (ou images *bitmap*), qui sont constituées de pixels.

Image matricielle

Exemple d'image matricielle.

Une **image matricielle**, ou « **carte de points** » (de l'anglais ***bitmap***), est une image constituée d'une matrice de points colorés. C'est-à-dire, constituée d'un tableau, d'une grille, où chaque case possède une couleur qui lui est propre et est considérée comme un point. Il s'agit donc d'une juxtaposition de points de couleurs formant, dans leur ensemble, une image.

Cette expression est principalement utilisée dans les domaines de l'imagerie numérique (infographie, informatique, photographie numérique, etc.) afin de marquer l'opposition de ce concept avec celui des images vectorielles. Dans ces domaines, les points de couleurs les constituant s'appellent des *pixels* (pour « picture element », soit, littéralement : « élément d'image »).

Quel est la différence entre une image matricielle ou vectorielle ?

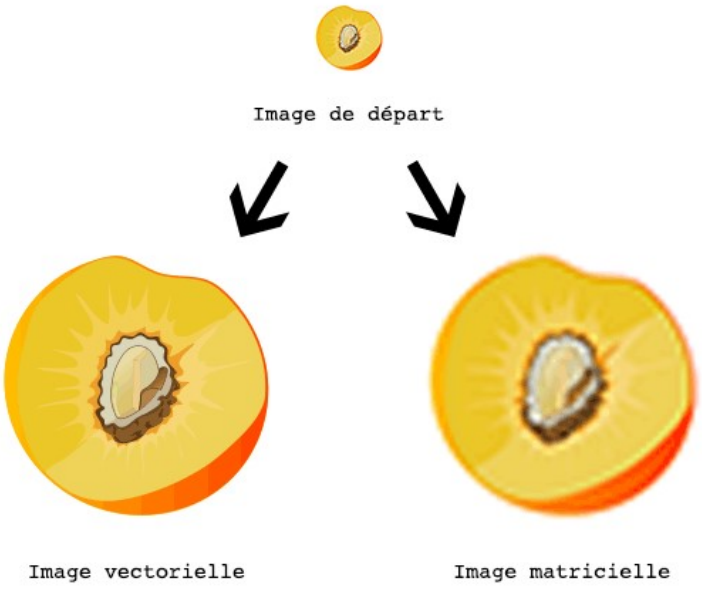
La principale différence entre ces deux formats est qu'une image vectorielle peut être agrandie sans perdre sa qualité alors qu'une image matricielle perd en netteté à l'agrandissement.

L'image matricielle (ou bitmap): Elle est composée de petits points appelés « pixels » que l'on ne voit pas à l'œil nu. Lors de l'agrandissement d'une image matricielle, cette dernière devient floue car les pixels ressortent, ce sont les carrés qui apparaissent sur l'écran.

L'image vectorielle : Elle est composée de lignes de segments qui sont liés entre eux par des formules mathématiques. Il s'agit d'un système de proportionnalité et de coordonnées. Grâce à la vectorisation, chaque

élément a une place bien définie ce qui empêche la déformation de l'image.

Les professionnels (graphistes, illustrateurs ou concepteurs) réalisent la majorité de leurs visuels en vectoriel afin de pouvoir les modifier à volontés sans les altérer.



Différence image matriciel – image vectorielle

Comment créer un fichier vectoriel ?

Pour vectoriser un fichier, il ne suffit pas de le créer sur Photoshop ou Illustrator. En effet la vectorisation est un peu plus compliquée.

Pour vectoriser, il vous faut un logiciel de publication assistée par ordinateur (PAO) comme Illustrator, Corel Draw ou Photoshop, puis couleur par couleur, forme par forme, vous devez définir un à un tous les vecteurs et lignes de l'image.

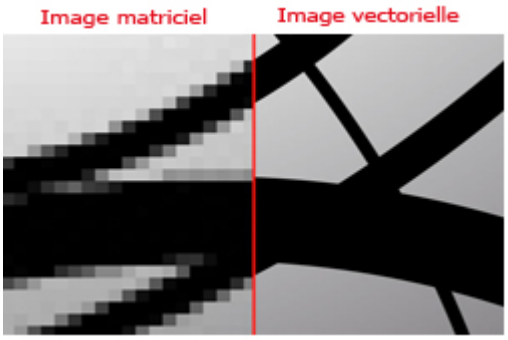
Une fois la vectorisation finie, vous pouvez enregistrer votre image sous le format que vous souhaitez : ai, eps, tiff, png….

Attention : Le format jpeg ne permet pas de conserver la vectorisation de l'image. En effet le jpeg est une image. Lorsque vous enregistrer un fichier en jpeg vous enregistrer une photo prise à un moment précis.

L'image ainsi enregistrée redevient matricielle et perd toute les caractéristiques de la vectorisation.

Les avantages de l'image vectorisée

L'avantage de la vectorisation est de pouvoir agrandir ou réduire une image à volonté sans qu'elle ne perde en qualité. Les lignes vectorielles qui composent l'image étant créées par des formules mathématiques, ces dernières sont recalculées et réadaptées à chaque changement de taille. Cette technique permet de garantir à 100 % la qualité de l'image.



Peut-on tout vectoriser ?

On ne peut pas tout vectoriser. En effet, les photos et les dégradés de couleurs ne se vectorisent pas. La vectorisation aplatie les couleurs et élimine les dégradés.

Exemple de photo vectorisée

Pourquoi est-il essentiel de vectoriser un fichier pour l'impression grand format ?

Afin de garantir une impression de haute qualité correspondant à votre fichier, il est essentiel de vectoriser votre fichier. Cela permet d'éviter de mauvaise surprise comme une police de texte qui change ou le déplacement des éléments du fichier. Grâce à la vectorisation vous obtiendrez toujours une impression haute définition.

Où obtenir son logo en format vectorisé ?

Beaucoup d'entreprise ne possèdent pas de logos ou d'images en format vectoriels. Un fichier matriciel est insuffisant pour la plupart des réalisations graphiques comme l'impression grand format.

En tant que société, les personnes qui peuvent vous fournir votre logo dans un format vectorisé sont:

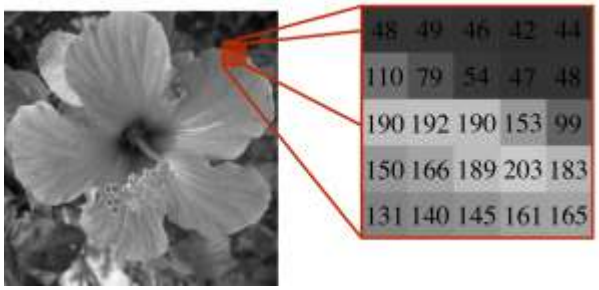
- – L'agence de communication ou le graphiste qui a créé votre logo ;
- – Votre service de communication ;
- – Votre imprimeur ;
- – Baches-publicitaires.com : Notre équipe PAO créée vos fichiers vectorisés.

Le traitement numérique des images

Une image numérique en niveaux de gris est un tableau de valeurs. Chaque case de ce tableau, qui stocke une valeur, se nomme un pixel. En notant n le nombre de lignes et p le nombre de colonnes de l'image, on manipule ainsi un tableau de $n \times p$ pixels.
La figure ci-dessous montre une visualisation d'un tableau carré avec $n=p=240$, ce qui représente $240 \times 240 = 57600$ pixels. Les appareils photos numériques peuvent enregistrer des images beaucoup plus grandes, avec plusieurs millions de pixels.

Une image en niveaux de gris

Les valeurs des pixels sont enregistrées dans l'ordinateur ou l'appareil photo numérique sous forme de nombres entiers entre 0 et 255, ce qui fait 256 valeurs possibles pour chaque pixel.
La valeur 0 correspond au noir, et la valeur 255 correspond au blanc. Les valeurs intermédiaires correspondent à des niveaux de gris allant du noir au blanc.
La figure ci-dessous montre un sous-tableau de 5×5 pixels extrait de l'image précédente. On peut voir à la fois les valeurs qui composent le tableau et les niveaux de gris qui permettent d'afficher l'image à l'écran.



Sous image de taille 5×5

Stocker une image

Stocker de grandes images sur le disque dur d'un ordinateur prend beaucoup de place. Les nombres entiers sont stockés en écriture binaire, c'est-à-dire sous la forme d'une succession de 0 et de 1. Chaque 0 et chaque 1 se stocke sur une unité élémentaire de stockage, appelée bit.
Pour obtenir l'écriture binaire d'un pixel ayant comme valeur 179, il faut décomposer cette valeur comme somme de puissances de deux. On obtient ainsi

$179=27+25+24+2+1,$

où l'on a pris soin d'ordonner les puissances de deux par ordre décroissant. Afin de faire mieux apparaître l'écriture binaire, on ajoute « 1× » devant chaque puissance qui apparaît dans l'écriture, et « 0× » devant les puissances qui n'apparaissent pas

$179=1\times27+0\times26+1\times25+1\times24+0\times23+0\times22+1\times2+1\times20.$

L'écriture binaire de la valeur 179 du pixel est ainsi (1,0,1,1,0,0,1,1), où chaque 1 et chaque 0 correspond au facteur multiplicatif qui apparaît devant chaque puissance.
On peut écrire toute valeur entre 0 et 255 de cet manière, ce qui nécessite d'utilisation de 8 bits. Il y a en effet 256 valeurs possibles, et 256=28. Pour stocker l'image complète, on a donc besoin de

$n\times p\times8$ bits.

Pour l'image montrée à la première figure, on a ainsi besoin de

$240\times240\times8=460800$ bits.

On utilise le plus souvent l'octet (8 bits) comme unité, de sorte que cette image nécessite *57,6*ko (kilo octets).

La résolution d'une image

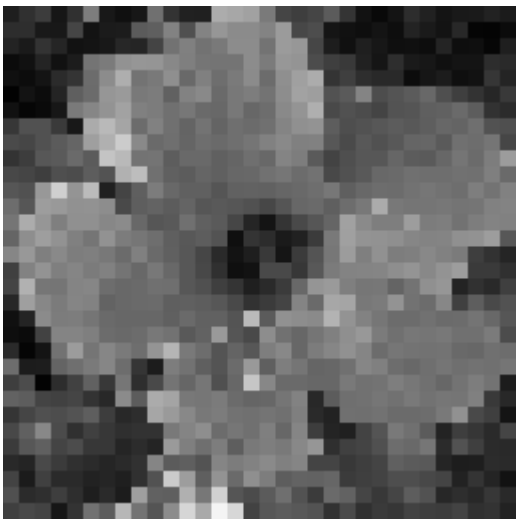
Afin de réduire la place de stockage d'une image, on peut réduire sa résolution, c'est-à-dire diminuer le nombre de pixels.
La façon la plus simple d'effectuer cette réduction consiste à supprimer des lignes et des colonnes dans l'image de départ.
La figure suivante montre ce que l'on obtient si l'on retient une ligne sur 4 et une colonne sur 4.

Une ligne/colonne sur 4

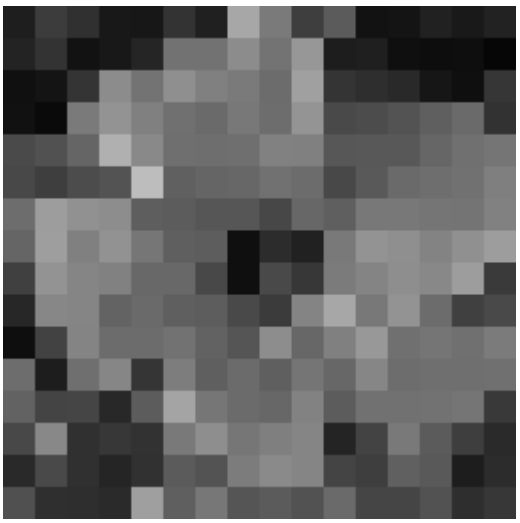
On a ainsi divisé par 4×4=16 le nombre de pixels de l'image, et donc également réduit par 16 le nombre de bits nécessaires pour stocker l'image sur un disque dur.
La figure suivante montre les résultats obtenus en gardant de moins en moins de lignes et de colonnes. Bien entendu, la qualité de l'image se dégrade vite.

Une ligne/colonne sur 2

Une ligne/colonne sur 4



Une ligne/colonne sur 8



Une ligne/colonne sur 16

Quantifier une image

Une autre façon de réduire la place mémoire nécessaire pour le stockage consiste à utiliser moins de nombres entiers pour chaque valeur.

On peut par exemple utiliser uniquement des nombres entiers entre 0 et 3, ce qui donnera une image avec uniquement 4 niveaux de gris.

On peut effectuer une conversion de l'image d'origine vers une image avec 3 niveaux de valeurs en effectuant les remplacements :

- les valeurs dans 0,1,…,63 sont remplacées par la valeur 0,
- les valeurs dans 64,65,…,127 sont remplacées par la valeur 1,
- les valeurs dans 128,129,…,191 sont remplacées par la valeur 2,
- les valeurs dans 192,193,…,255 sont remplacées par la valeur 3.

Une telle opération se nomme quantification.

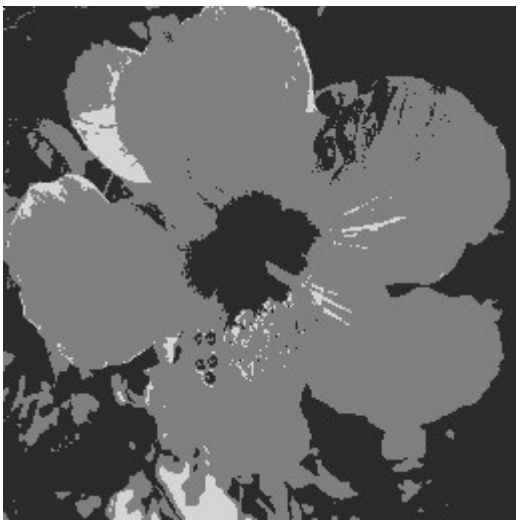
La figure suivante montre l'image résultante avec 4 niveaux de gris. Les 4 valeurs sont affichées en utilisant 4 niveaux de gris allant du noir au blanc.

16 niveaux de gris

Nous avons déjà vu que l'on pouvait représenter toute valeur entre 0 et 255 à l'aide de 8 bits en utilisant l'écriture binaire. De façon similaire, on vérifie que toute valeur entre 0 et 3 peut se représenter à l'aide de 2 bits. On obtient ainsi une réduction d'un facteur 8/2=4 de la place mémoire nécessaire pour le stockage de l'image sur un disque dur.
La figure suivante montre les résultats obtenus en utilisant de moins en moins de niveaux de gris.

16 niveaux de gris

4 niveaux de gris



3 niveaux de gris

2 niveaux de gris

Tout comme pour la réduction du nombre de pixels, la réduction du nombre de niveaux de gris influe beaucoup sur la qualité de l'image. Afin de réduire au maximum la taille d'une image sans modifier sa qualité, on utilise des méthodes plus complexes de compression d'image. La méthode la plus efficace s'appelle JPEG-2000. Elle utilise la théorie des ondelettes. Pour en savoir plus à ce sujet, vous pouvez consuler cet article d'Erwan Le Pennec.

Changer le contraste d'une image

Il est possible de faire subir différentes modifications à l'image afin de changer son contraste.

Un exemple simple consiste à remplacer chaque valeur a d'un pixel d'une image par $255-a$ ce qui correspond au niveau de gris opposé. Le blanc devient noir et vice-versa, ce qui donne un effet similaire à celui des négatifs d'appareils photos argentiques.

Négatif

Sans aller jusqu'à des modifications aussi extrêmes, on peut assombrir une image en remplaçant la valeur a de chaque pixel par son carré $a2=a\times a$.

Ce faisant, les valeurs résultantes ne sont plus dans 0,…,255 mais dans 0,…,2552=65025. Afin d'afficher l'image à l'écran on va donc utiliser des niveaux de gris allant du noir pour 0 au blanc pour 65025.

Carré

Afin d'éclaircir l'image, on peut remplacer chaque valeur a par sa racine carrée $b=a--\sqrt{}$. Cette valeur b est un nombre, qui n'est plus nécessairement entier, qui satisfait $b\times b=a$.
La figure suivante montre l'éclaircissement obtenu. Les valeurs de l'image éclaircie sont dans $0,\ldots,255---\sqrt{}\approx16$, et on utilise donc des niveaux de gris allant du noir (pour 0) au blanc (pour 16).

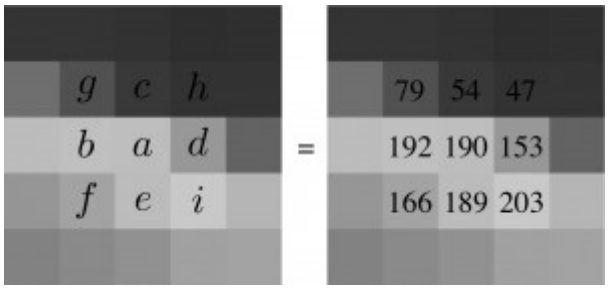
Racine carrée

On pourra noter que l'on a
$a--\sqrt{}\times a--\sqrt{}=a\text{et}a\times a-----\sqrt{}=a$
de sorte que si l'on réalise un éclaircissement suivi d'un assombrissement (ou dans le sens inverse) on retrouve l'image d'origine. Ces deux opérations sont inverses l'une de l'autre.

Enlever le bruit par moyennes locales
Les images sont parfois de mauvaise qualité. Un exemple typique de défaut est le bruit qui apparaît quand une photo est sous-exposée, c'est-à-dire qu'il n'y a pas assez de luminosité. Ce bruit se manifeste par de petites fluctuations aléatoires des niveaux de gris. La figure ci-dessous montre une image bruitée.

Image bruitée

Afin d'enlever le bruit dans les images, il convient de faire subir une modification aux valeurs de pixels. L'opération la plus simple consiste à remplacer la valeur *a* de chaque pixel par la moyenne de *a* et des 8 valeurs *b*,*c*,*d*,*e*,*f*,*g*,*h*,*i* des 8 pixels voisins de *a*.
La figure suivante montre un exemple de voisinage de 9 pixels.



Exemple d'un voisinage 9 pixels

On obtient ainsi une image modifiée en remplaçant *a* par
a+*b*+*c*+*d*+*e*+*f*+*g*+*h*+*i*9
puisque l'on fait la moyenne de 9 valeurs.
Dans notre exemple, cette moyenne vaut
190+192+79+54+47+153+203+189+1669≈141,4.
En effectuant cette opération pour chaque pixel, on supprime une partie du bruit, car ce bruit est constitué de fluctuations aléatoires, qui sont diminuées par un calcul de moyennes. La figure ci-dessous montre l'effet d'un tel calcul.


Image bruitée **Moyenne sur 9 pixels**

Tout le bruit n'a pas été enlevé par cette opération. Afin d'enlever plus de bruit, on peut moyenner plus de valeurs autour de chaque pixel. La figure suivante montre le résultat obtenu en moyennant de plus en plus de valeurs.


Moyenne sur 9 pixels **Moyenne sur 25 pixels**


Moyenne sur 49 pixels **Moyenne sur 81 pixels**

Le calcul de moyenne de pixels est très efficace pour enlever le bruit dans les images. Malheureusement il détruit également une grande partie de l'information de l'image. On peut en effet s'apercevoir que les images

obtenues par moyennes sont floues. Ceci est en particulier visible près des contours.

Enlever le bruit par médianes locales

Afin de réduire le flou introduit par les moyennes locales, il faut remplacer le calcul de moyenne par une opération un peu plus complexe, que l'on nomme médiane.

Etant donné la valeur *a* d'un pixel, et les valeurs *b*,*c*,*d*,*e*,*f*,*g*,*h*,*i*, on commence par les classer par ordre croissant.

Dans l'exemple du voisinage de 9 pixels utilisé à la section précédente, on obtient les 9 valeurs classées

47,54,79,153,166,189,190,192,203.

La médiane des neuf valeurs *a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i* est la 5*e* valeur de ce classement (c'est-à-dire la valeur centrale de ce classement).

Dans notre cas, la médiane est donc 166. Notez que ce nombre est en général différent de la moyenne, qui vaut, pour notre exemple *141,4*.

La figure ci-dessous compare le débruitage obtenu en effectuant la moyenne et la médiane de 9 pixels voisins.

Moyenne sur 9 pixels

Médiane sur 9 pixels

Afin d'enlever plus de bruit, il suffit de calculer la médiane sur un nombre plus grand de pixels voisins, comme montré à la figure suivante.

Médiane sur 9 pixels

Médiane sur 25 pixels

Médiane sur 49 pixels

Médiane sur 81 pixels

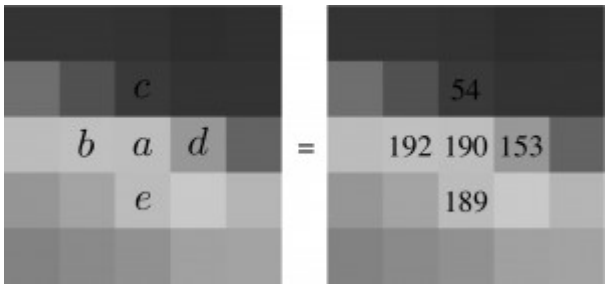
On constate que cette méthode est plus performante que le calcul de moyennes, car les images résultantes sont moins floues. Cependant, tout comme avec le calcul de moyennes, si l'on prend des voisinages trop grands, on perd aussi de l'information de l'image, en particulier les bords des objets sont dégradés.

Détecter les bords des objets

Afin de localiser des objets dans les images, il est nécessaire de détecter les bords de ces objets. Ces bords correspondent à des zones de l'image où les valeurs des pixels changent rapidement. C'est le cas par exemple lorsque l'on passe du pétale de la fleur (qui est clair, donc avec des valeurs grandes) à l'arrière plan (qui est sombre, donc avec des valeurs petites).

Afin de savoir si un pixel avec une valeur a est le long d'un bord d'un objet, on prend en compte les valeurs b,c,d,e de ses quatre voisins (deux

horizontalement et deux verticalement), qui sont disposés par rapport à *a* comme illustré à la figure suivante.



Exemple d'un voisinage de 5 pixels

Notons que l'on utilise ici seulement 4 voisins, ce qui est différent du calcul de moyennes et de médianes où l'on utilisait 8 voisins. Ceci est important afin de détecter aussi précisément que possible les bords des objets.

On calcule une valeur ℓ suivant la formule

$$\ell=\sqrt{(b-d)2+(c-e)2}.$$

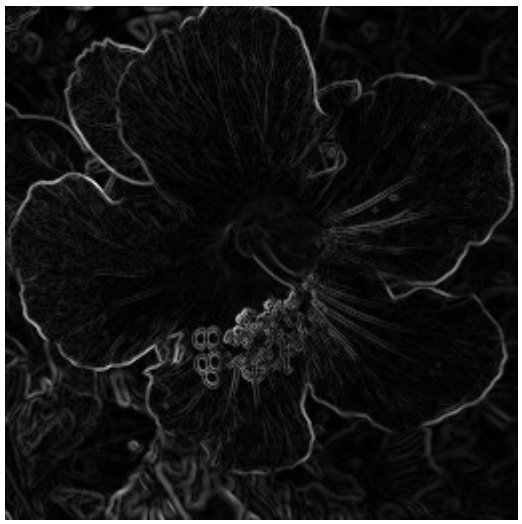
Dans notre exemple, on obtient donc

$$\ell=\sqrt{(192-153)2+(189-54)2}=\sqrt{19746}\approx 140{,}5.$$

On peut remarquer que si $\ell=0$, alors on a $b=d$ et $c=e$. Au contraire, si ℓ est grand, ceci signifie que les pixels voisins ont des valeurs très différentes, le pixel considéré est donc probablement sur le bord d'un objet.

La figure suivante montre l'image obtenue en calculant la valeur ℓ associée à chaque pixel. On a affiché ces valeurs avec du noir quand $\ell=0$, du blanc quand ℓ atteint sa valeur maximale.

Image



Carte de contours ℓ

On peut voir que dans l'image de droite, les contours des objets ressortent en blanc, car ils correspondent aux grandes valeurs de ℓ.

Les images couleurs

Une image couleur est en réalité composée de trois images, afin de représenter le rouge, le vert, et le bleu. Chacune de ces trois images s'appelle un canal. Cette représentation en rouge, vert et bleu mime le fonctionnement du système visuel humain.

La figure suivante montre la décomposition d'une image couleur en ses trois canaux constitutifs.

Image numérique couleur

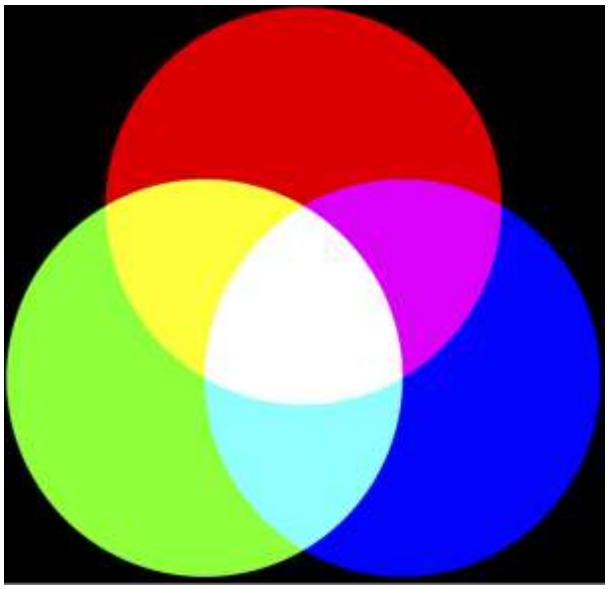
Canal rouge

Canal vert

Canal bleu

Chaque pixel de l'image couleur contient ainsi trois nombres (r,v,b), chacun étant un nombre entier entre 0 et 255. Si le pixel est égal à $(r,v,b)=(255,0,0)$, il ne contient que de l'information rouge, et est affiché comme du rouge. De façon similaire, les pixels valant (0,255,0) et (0,0,255) sont respectivement affichés vert et bleu.

On peut afficher à l'écran une image couleur à partir de ses trois canaux (r,v,b) en utilisant les règles de la synthèse additive des couleurs. La figure suivante montre les règles de composition cette synthèse additive des couleurs. Un pixel avec les valeurs (r,v,b)=(255,0,255) est un mélange de rouge et de vert, il est ainsi affiché comme jaune.



Synthèse additive des couleurs

On peut calculer une image en niveaux de gris à partir d'une image couleur en moyennant les trois canaux. On calcule donc une valeur

$$a=r+v+b3$$

qui s'appelle la luminance de la couleur.

La figure suivante montre l'image de luminance associée à une image couleur.

Luminance de l'image

Une autre représentation courante pour les images couleurs utilise comme couleurs de base le cyan, le magenta et le jaune. On calcule les trois nombres (c,m,j) correspondant à chacun de ces trois canaux à partir des canaux rouge, vert et bleu (r,v,b) comme suit
$c=255-r, m=255-v, j=255-b$.
Par exemple, un pixel de bleu pur $(r,v,b)=(0,0,255)$ va devenir $(c,m,j)=(255,255,0)$. La figure suivante montre les trois canaux (c,m,j) d'une image couleur.

Image numérique couleur

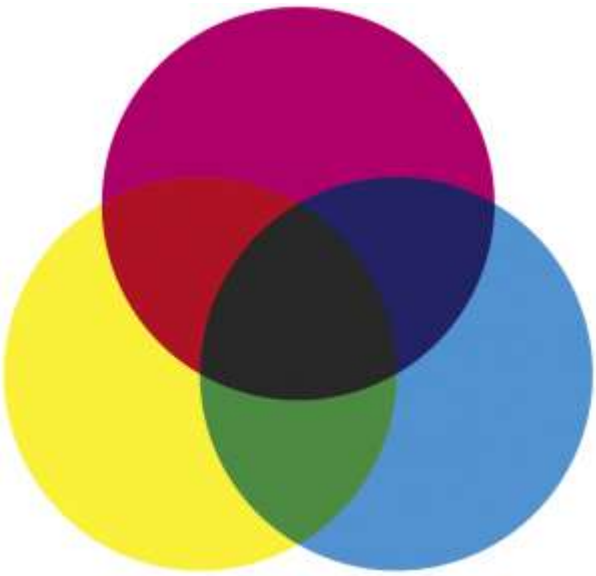
Canal cyan

Canal magenta

Canal jaune

Afin d'afficher une image couleur à l'écran à partir des trois canaux (c,m,j), on doit utiliser la synthèse soustractive des couleurs. La figure suivante montre les règles de composition cette synthèse soustractive. Notons que ces règles sont celles que l'on utilise en peinture, lorsque l'on

mélange des pigments colorés. Le cyan, le magenta et le jaune sont appelés couleurs primaires.



Synthèse soustractive des couleurs

On peut donc stocker sur un disque dur une image couleur en stockant les trois canaux, correspondant aux valeurs (*r*,*g*,*b*) ou (*c*,*m*,*j*). On peut modifier les images couleur tout comme les images en niveaux de gris. La façon la plus simple de procéder consiste à appliquer la modification à chacun des canaux.

Conclusion

Cet article n'a fait qu'effleurer l'immense liste des traitements que l'on peut faire subir à une image. Le traitement mathématique des images est un domaine très actif, où les avancées théoriques se concrétisent sous la forme d'algorithmes rapides de calcul qui ont des applications importantes pour la manipulation des contenus numériques.

Outil pour transformer des images en noir et blanc (niveaux de gris). Une image en niveau de gris est une image dont les couleurs varient du blanc au noir.

Réponses aux Questions

Comment transformer une image en noir et blanc ?

! Noir et blanc s'entend ici comme dégradé de niveux de gris. Pour la transformation d'une image en binaire noir ou blanc, voir l'outil image binaire.

Chaque pixel couleur d'une image est généralement défini par un triplet de valeurs de rouge, vert et bleu, mais peut aussi être défini par un triplet

teinte, saturation, luminosité. Cette valeur de luminosité correspond à la position du pixel entre le noir et le blanc.
Pour calculer cette luminosité, plusieurs algorithmes/référentiels de couleurs sont possibles, comme les recommandations 709, 601 ou 2100 utilisés comme standard en photographie ou vidéo (et issus de la CIE Commission Internationale de l'Éclairage).
Si une erreur est retournée, il peut s'agit d'un fichier altéré, d'une mauvaise image, ou que le format du fichier ne correspond pas à son extension.

Qu'est ce que la recommandation CIE 709 ?
La recommandation 709 est utilisée comme standard pour la télévision HDTV. La luminance est calculée Gris = 0,2125 * Rouge + 0,7154 * Vert + 0,0721 * Bleu
Exemple : Une couleur représentant la couleur Orange constituée de Rouge=255, Vert=127, Bleu=0. Alors la valeur de gris est 255*0.2125+127*0.7154+0*0.0721=145

Qu'est ce que la recommandation CIE 601 ?
La recommandation 601 était utilisée comme standard pour la télévision analogique Hertzienne. La luminance est calculée Gris = 0,299 * Rouge + 0,587 * Vert + 0,114 * Bleu
Exemple : Une couleur représentant la couleur Orange constituée de Rouge=255, Vert=127, Bleu=0. Alors la valeur de gris est 255*0.299+127*0.587+0*0.0114=151

2. Lecture d'une image
2.a. Codage d'une image en couleur

Les images fournies par les appareils photo sont généralement en couleur. Une image en couleur est contituée de trois couches : une couche rouge (R), une couche verte (V), une couche bleue (B). Nous n'allons pas ici expliquer comment sont représentées les couleurs. Voir à ce sujet : Espace des couleurs RVB et la simulation Espace des couleurs RVB et triangle de Maxwell.
Soit N_x le nombre de colonnes de l'image et N_y le nombre de lignes. Le nombre de pixels total est $N=N_xN_y$. Chaque couche est une matrice comportant N_y lignes et N_x colonnes. Le plus souvent, cette matrice contient des entiers codés sur 8 bits (les valeurs vont de 0 à 255). Pour l'image en couleur complète, il y a donc 24 bits par pixels, à multiplier par le nombre de pixels pour obtenir l'occupation totale en mémoire. Chaque couche peut être vue comme une image en niveaux de gris. Le niveau 0 est le noir, le niveau 255 est le blanc, le niveau 128 est un gris moyen.
On voit sur la figure le pixel (5,4) dont les niveaux de gris des trois couches sont (100,20,200), ce qui donne une couleur violette. Par convention, on notera en premier l'indice qui repère les colonnes, conformément au repérage des coordonnées d'un point sur un plan (x,y). On remarque néanmoins que l'origine se trouve sur le coin supérieur gauche de l'image.

2.b. Lecture d'un fichier image

Voici l'image en couleur :

Pour lire un fichier d'image (PNG, JPEG, etc), on utilise le module imageio :

```
import imageio
import numpy
```

```
from matplotlib.pyplot import *
img =
imageio.imread("../../../../figures/image/niveaux/images/babouin.png"
)
```

Voici les dimensions du tableau et le type de données :

```
print(img.shape)
--> (256, 256, 3)
print(img.dtype)
--> dtype('uint8')
```

La valeur d'un pixel pour une couche est codée sur 8 bits : c'est un entier compris en 0 et 255.
La séparation des trois couches se fait par :

```
rouge = img[:,:,0]
vert = img[:,:,1]
bleu = img[:,:,2]
```

On se contentera ici de travailler sur la couche rouge, qui est une image en niveaux de gris. On peut voir la valeur d'un pixel particulier, celui d'indices (120,100) :

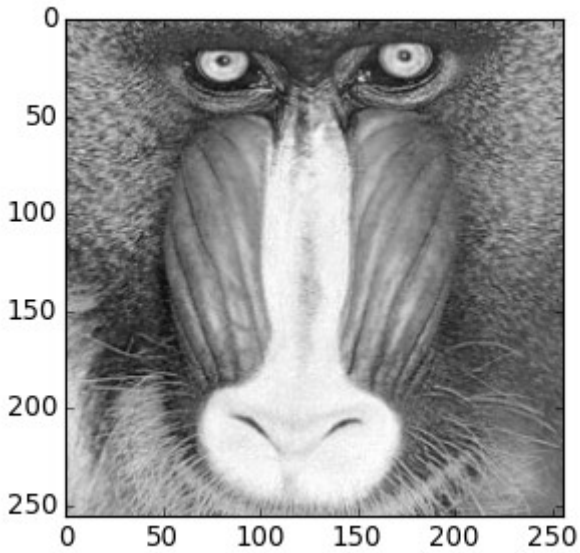
```
print(rouge[100,120])
--> 238
```

Remarquons que le premier indice d'accès au tableau indique la ligne.

2.c. Affichage d'une image

La fonction matplotlib.pyplot.imshow permet d'afficher un tableau sous forme d'une image :

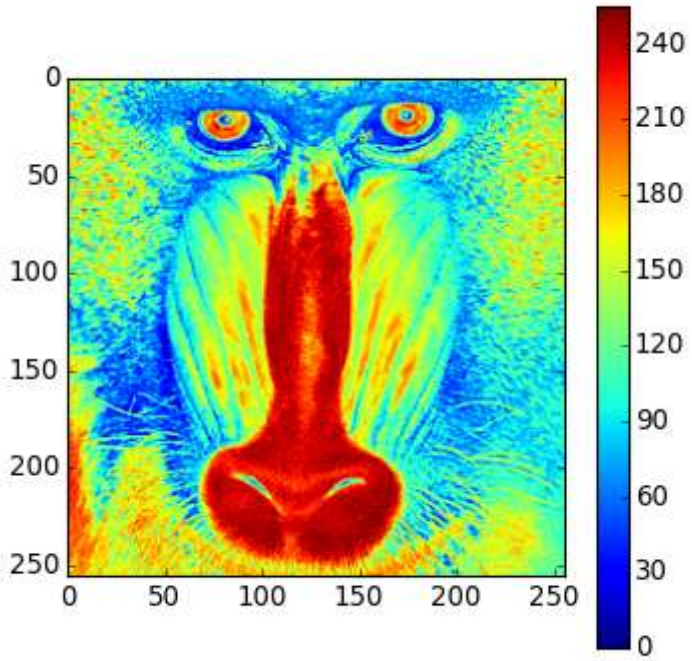
```
figure(figsize=(4,4))
imshow(rouge,cmap='gray')
```



Le pixel de coordonnées (i,j) est l'élément rouge[j,i].
L'argument cmap='gray' permet de représenter les valeurs en niveaux de gris. La fonction matplotlib.pyplot.imshow ainsi utilisée convertit la

plage de valeurs [min,max] du tableau en valeurs dans l'intervalle [0,255] pour l'affichage en image.
Il peut être intéressant de représenter l'image en niveaux de gris avec une échelle de couleurs, car nous identifions plus facilement une couleur qu'un niveau de gris :

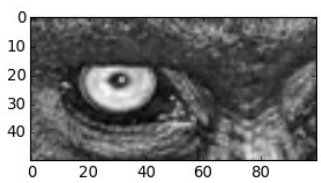
```
figure(figsize=(5,5))
imshow(rouge)
colorbar()
```



2.d. Extraction d'une partie d'une image

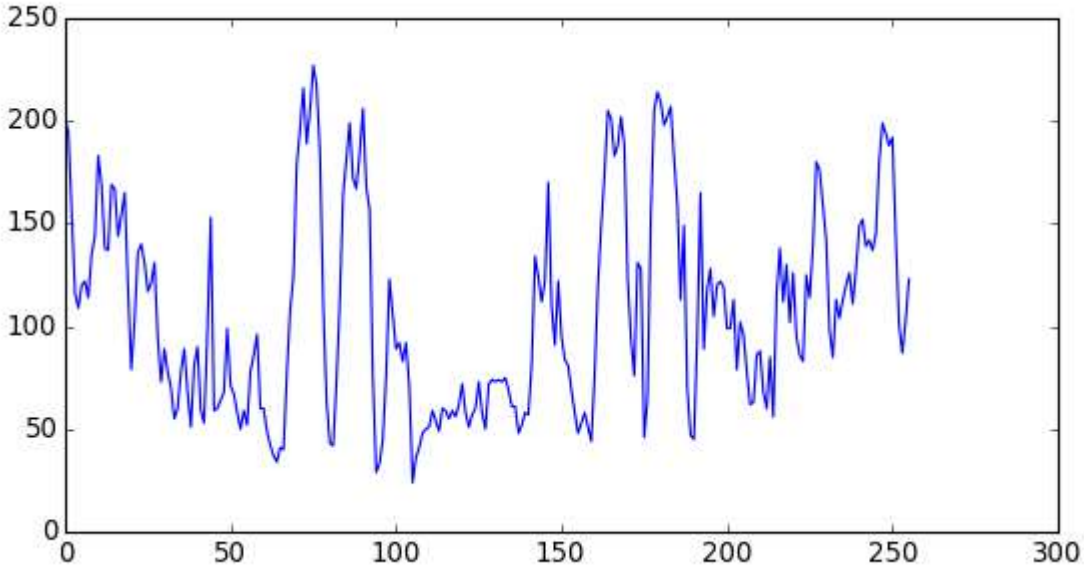
Une partie rectangulaire d'une image peut être extraite de la manière suivante, en faisant attention à indiquer les lignes sélectionnées en premier :

```
partie = rouge[0:50,50:150]
figure(figsize=(4,4))
imshow(partie,cmap='gray')
```



Une opération courante est la sélection d'une seule ligne (ou d'une seule colonne) :

```
ligne20 = rouge[20,:]
figure(figsize=(8,4))
plot(ligne20)
```



On obtient ainsi le profil de niveaux de gris sur cette ligne.

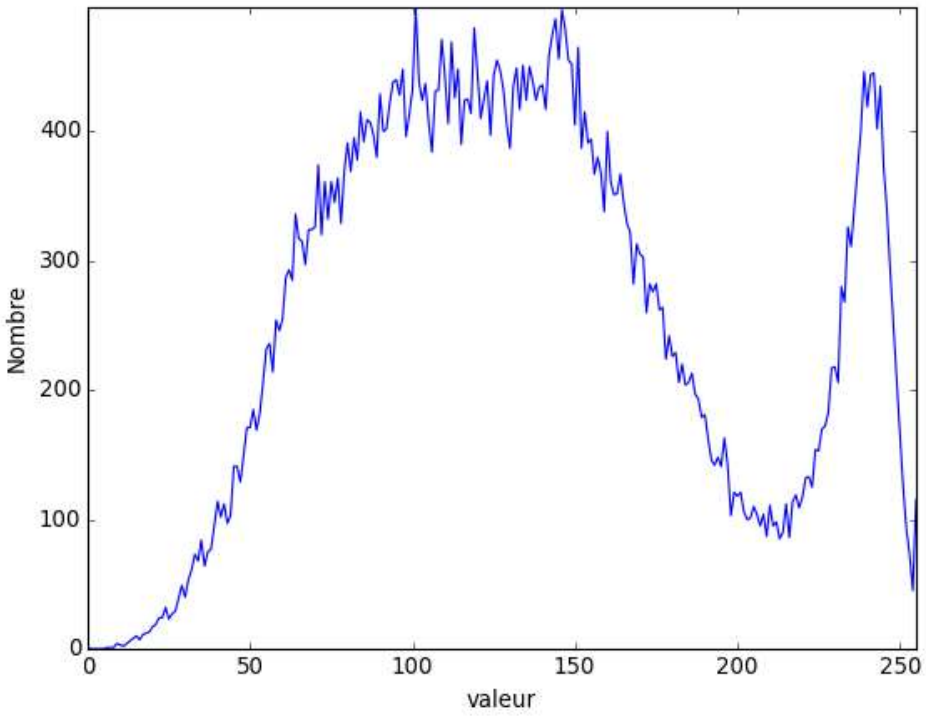
3. Étude statistique des niveaux de gris

Voici comment obtenir la moyenne des valeurs d'une image et l'écart-type :

```
print(rouge.mean())
--> 137.09773254394531
print(rouge.std())
--> 56.967793119151594
```

Pour obtenir plus d'informations, on peut calculer un histogramme. Pour chaque valeur entière comprise entre 0 et 255, on compte le nombre de pixels qui ont cette valeur.

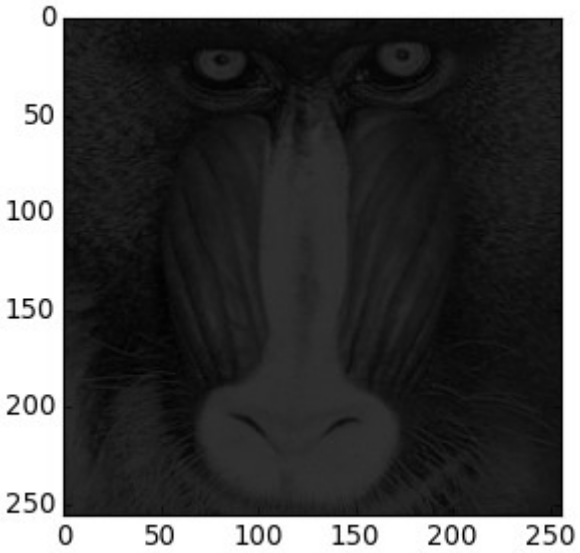
```
def histogramme(image):
    h = numpy.zeros(256,dtype=numpy.uint32)
    s = image.shape
    for j in range(s[0]):
        for i in range(s[1]):
            valeur = image[j,i]
            h[valeur] += 1
    return h

h = histogramme(rouge)
figure(figsize=(8,6))
plot(h)
axis([0,255,0,h.max()])
xlabel("valeur")
ylabel("Nombre")
```



4. Modification des niveaux de gris

Voyons ce qu'il se passe si l'on divise les valeurs d'une image par 5 (division entière) :

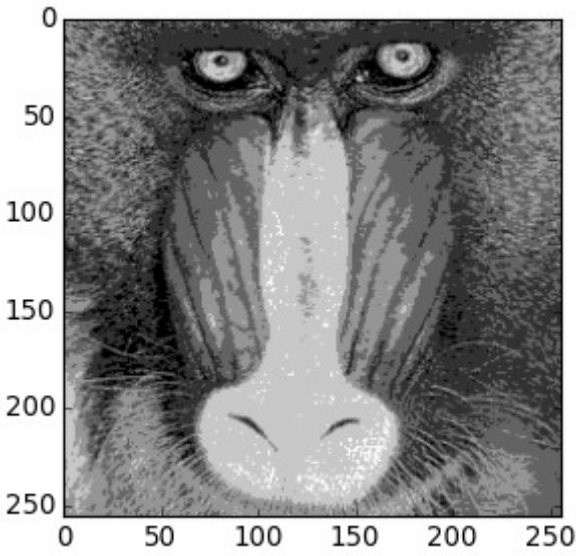
```
im1 = rouge/5
figure(figsize=(4,4))
imshow(im1,cmap='gray',vmin=0,vmax=255)
```



On a précisé les valeurs minimale et maximale, pour éviter qu'une normalisation des valeurs soit effectuée. Comme attendu, l'image est beaucoup plus sombre.

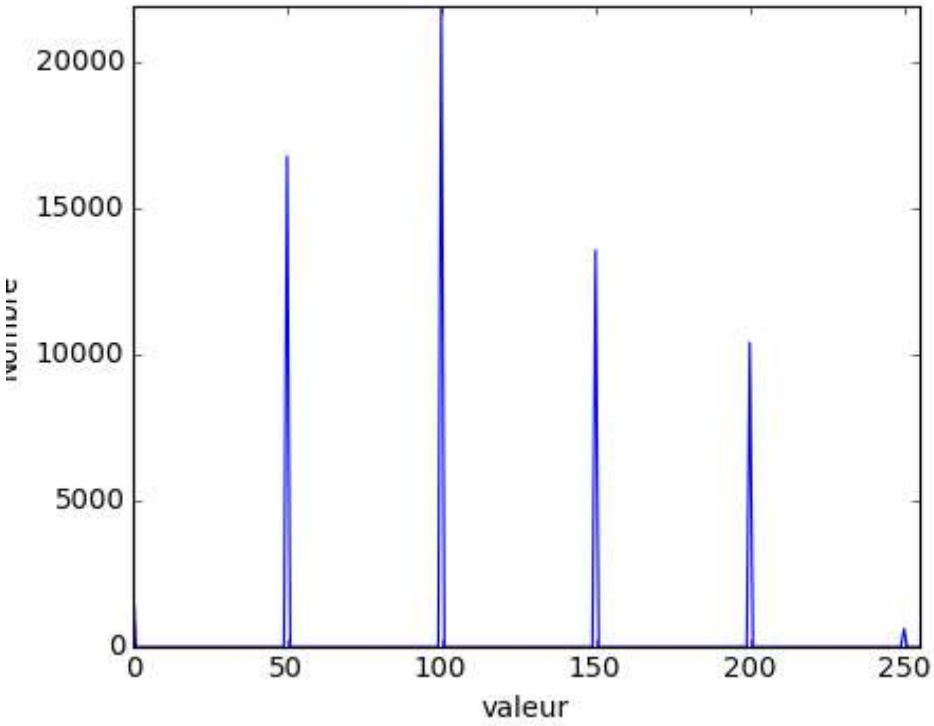
Voici ce qu'il arrive si l'image est divisée par 50, puis multipliée par 50 pour retrouver l'image initiale :

```
im2 = rouge/50
im3 = im2*50
figure(figsize=(4,4))
imshow(im3,cmap='gray',vmin=0,vmax=255)
```



L'image finale ne comporte plus qu'un nombre restreint de niveaux de gris. C'est une conséquence de la division entière des valeurs. Voyons l'histogramme de cette image :

```
h = histogramme(im3)
figure(figsize=(6,5))
plot(h)
axis([0,255,0,h.max()])
xlabel("valeur")
ylabel("Nombre")
```



On voit ainsi que les calculs faits sur des entiers peuvent conduire à une perte d'information sur les niveaux de gris de l'image. Pour éviter cet inconvénient, il suffit de convertir l'image en tableau de flottants :

```
rouge = rouge*1.0

print(rouge.dtype)
--> dtype('float64')
```

Les flottants sont codés sur 64 bits (flottants double précision), à comparer aux 8 bits de l'image initiale. Tous les traitements d'image (par exemple le filtrage) seront effectués avec des flottants, pour éviter les pertes d'informations dues aux divisions entières. Les valeurs seront converties en entiers au moment de l'enregistrement dans un fichier.

Les flottants obtenus sont dans l'intervalle [0,255]. Pour certains calculs, il est plus commode de travailler avec des flottants dans l'intervalle [0,1], ce qui s'obtient par :

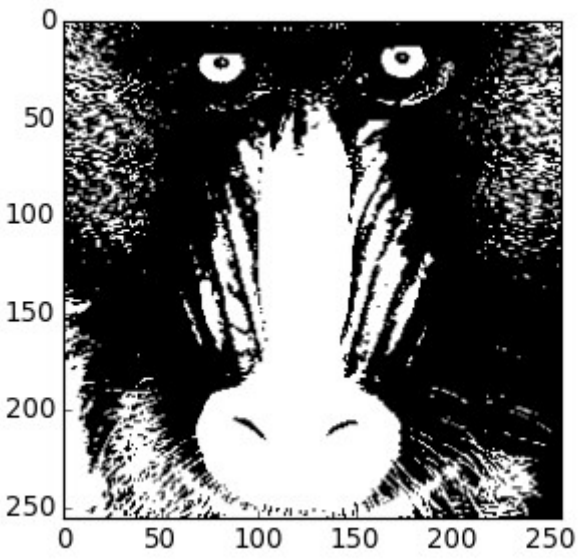
```
rouge01 = rouge/255.0
```

Cette convention de représentation des images est souvent utilisée, car elle donne un codage indépendant de la profondeur de l'image initiale (8 bits ou 16 bits).

Une opération courante de traitement d'image est le seuillage, qui consiste à repérer les pixels dont la valeur dépasse un certain seuil.

```
def seuillage(image,seuil):
    resultat = image.copy()
    s = image.shape
    for j in range(s[0]):
        for i in range(s[1]):
```

```
            if image[j,i] > seuil:
               resultat[j,i] = 255
            else:
                resultat[j,i] = 0
    return resultat

im4 = seuillage(rouge,150)
figure(figsize=(4,4))
imshow(im4,cmap='gray',vmin=0,vmax=255)
```

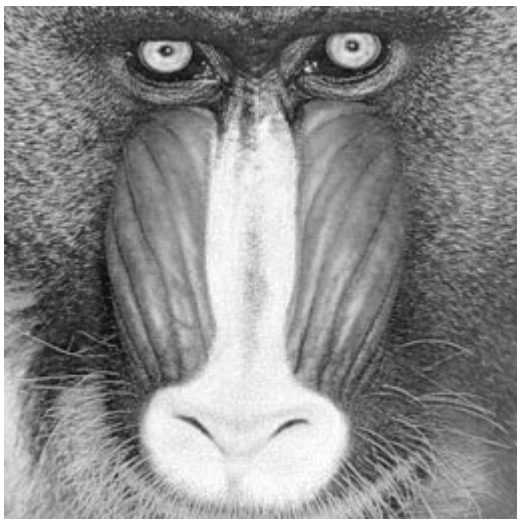


5. Enregistrement d'une image

L'image précédente peut être enregistrée avec la fonction imageio.imwrite :

```
imageio.imwrite("../../../../figures/image/niveaux/images/imA.png",ro
uge)
```

Voici l'image obtenue avec ce fichier :



Il s'agit d'une image en couleur dont les trois couches sont identiques, ce qui donne des niveaux de gris.

On peut aussi reconstituer une image en couleur à partir de ses trois couches. Pour obtenir une image différente de l'image initiale, on réduit les valeurs de la couche rouge. On doit tout d'abord créer un tableau à trois dimensions et le remplir avec les couches.

```
s = rouge.shape
couleur = numpy.zeros((s[0],s[1],3),dtype=numpy.uint8)
couleur[:,:,0] = rouge*0.5
couleur[:,:,1] = vert
couleur[:,:,2] = bleu
imageio.imwrite("../../../../figures/image/niveaux/images/imB.png",co
uleur)
```

Fonction

uint8

Tableaux d'entiers non signés 8 bits

Description

Variables dans MATLAB® de données, type (classe) `uint8` sont stockés sous forme de 1 octet (8 bits) des entiers non signés. Par exemple :

```
>>y = uint8(10);

>>whos y

Name        Size              Bytes  Class Attributes

 y          1x1                   1      uint8
```

Conversion image couleur en biveau de gris

```
function ConversionRGBtoGris(file,cofR,cofG,cofB)
   im=imread(file);
   Fig1 = figure('Position',[1260,210,450,450],'Toolbar','none','Menubar','none',
                 'Name','Image originale','NumberTitle','off');
   imshow(im);
   out = uint8(zeros(size(im,1), size(im,2), size(im,3)));

   %Composants R, G, B de l'image d'entrée
   R = im(:,:,1);
   G = im(:,:,2);
   B = im(:,:,3);
   %la recommandation CIE 601
   out(:,:,1) = R*cofR ;
   out(:,:,2) = G*cofG ;
   out(:,:,3) = B*cofB ;
   Fig =figure('Position',[200,210,450,450], 'Toolbar','none','Menubar','none',
               'Name', 'Niveau de gris','NumberTitle','off');
   imshow(out);
end
```

Execution

La recommandation CIE 601

ConversionRGBtoGris('im9.png',0.299 ,0.587,0.114);

La recommandation CIE 709

ConversionRGBtoGris('im9.png',0.2125,0.7154,0.0721);

La conversion simple $Gris = \frac{(Rouge + Vert + Bleu)}{3}$

ConversionRGBtoGris('im9.png',1/3 ,1/3,1*3);

La fonction de conversion en matlab

```
I=rgb2gray(RGB);
```

Exemple

```
RGB = imread('peppers.png');
Fig1 = figure('Position',[1260,210,450,450],'Toolbar','none','Menubar','none',
              'Name','Image originale','NumberTitle','off');

imshow(RGB) ;
```

%Convertir une image en niveaux de gris de l'image RVB et l'afficher.

```
I = rgb2gray(RGB);
 Fig =figure('Position',[200,210,450,450], 'Toolbar','none','Menubar','none',
             'Name', 'Niveau de gris','NumberTitle','off');
imshow(I);
```

Equilibrage des couleurs

```
function out = EquilibrageCouleur(file)
    % Équilibrage des couleurs à l'aide de l'assomption du monde gris
    %   I - Image RVB 24 bits
    %   out - Image RVB 24 bits équilibrée en couleurs
     im=imread(file);
     Fig1 =figure('Position',[1260,210,450,450],'Toolbar',
```

```
                    'none','Menubar','none','Name','Image
                     originale','NumberTitle','off');
              imshow(im);
              out = uint8(zeros(size(im,1), size(im,2), size(im,3)));

            %R,G,B components of the input image
            R = im(:,:,1);
            G = im(:,:,2);
            B = im(:,:,3);

            % Inverse des valeurs moyennes des valeurs R, V et B

            mR = 1/(mean(mean(R)));
            mG = 1/(mean(mean(G)));
            mB = 1/(mean(mean(B)));

            % Plus petite valeur moyenne (MAX parce que nous avons
               affaire à l'inverse)
            maxRGB = max(max(mR, mG), mB);

            %Calculate the scaling factors
            mR = mR/maxRGB;
            mG = mG/maxRGB;
            mB = mB/maxRGB;

            %Scale the values
             out(:,:,1) = R*mR;
             out(:,:,2) = G*mG;
             out(:,:,3) = B*mB;
             Fig =
figure('Position',[200,210,450,450],'Toolbar','none','Menubar','none'
,'Name','Équilibrage des couleurs en utilisant l''hypothèse du monde
gris','NumberTitle','off');
              imshow(out);
        end
```

Conversion RGB vers Hsv

hsv = rgb2hsv(rgb)
Convertis valeurs de RVB à la teinte appropriée, la saturation, et coordonne les valeurs (TSV).. hsv est la même taille que rgb.

Attribut	Description
Hue	Valeur de `0` à `1` qui correspond à la position de la couleur sur une roue chromatique. Comme teinte passe de `0` à `1`, les transitions de couleur du rouge à l'orange, jaune, vert, cyan, bleu, magenta et enfin de retour au rouge.
Saturation	Montant de départ de neutre ou de teinte. `0` indique une teinte neutre, tandis que `1` indique la saturation maximale.
Valeur	Valeur maximale entre les composantes rouges, verts et bleus d'une couleur spécifique.

Exemple

```
function [img_hsv,hue,s,v]=imHsv(obj)
```

```
 I=double(obj.im);
 %[hue,s,v]=rgb2hsv(I);
 img_hsv = rgb2hsv(I);
 hue=img_hsv(:,:,1);
 s  =img_hsv(:,:,2);
 v  =img_hsv(:,:,3);
 Fig = figure('Toolbar','none','Menubar','none','Name',
              'Image HSV','NumberTitle','off');
 imshow(img_hsv);
end
```

Convertir VHS en couleurs RVB

```
rgb = hsv2rgb(hsv)
```

Description

rgb = hsv2rgb(hsv) convertit la teinte, saturation et valeur (TSV) coordonnées aux valeurs appropriées de rouge, verts et bleus (RVB). Tableau hsv peut être un tableau de trois colonnes ou m-par-n-en-3. Tableau de sortie rgb est la même taille et type numérique comme hsv.

Exemple

hsv = [.6 1 1; .6 .7 1; .6 .5 1; .6 .3 1; .6 0 1];

Convertir la matrice HSV un colormap en appelant hsv2rgb. Puis utilisez cette palette dans une parcelle de surface.

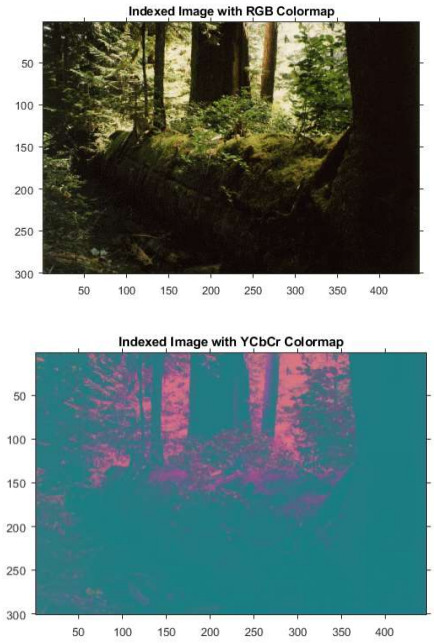
```
rgb = hsv2rgb(hsv);
surf(peaks);
colormap(rgb);
colorbar
```

Convertir RVB en couleurs YCbCr

YCbCr **=rgb2ycbcr**

Convertir des valeurs de couleur RVB en espace de couleurs YCbCr

```
%% Convertir RVB en couleurs YCbCr

function [img_ycbcr,Y,Cb,Cr ]=imYcbcr(obj)
  img_ycbcr = rgb2ycbcr(obj.im);
  Y  = img_ycbcr(:,:,1);
  Cb = img_ycbcr(:,:,2);
  Cr = img_ycbcr(:,:,3);
  Fig = figure('Toolbar','none','Menubar','none','Name',
               'Image Ycbcr','NumberTitle','off');
  imshow(img_ycbcr);
end
```



imcontrast

Régler l'outil de contraste

La fonction imcontrast permet de créer un outil de contraste. L'outil de contraste est un contraste interactif et l'outil de réglage de luminosité que vous pouvez utiliser pour ajuster le mappage de noir à blanc permet d'afficher une image en niveaux de gris.

Exemple

```
function haussement_Contraste(obj)
  I=obj.imGris();
  Fig1 = figure('Toolbar','none','Menubar','none','Name',
                'Image originale','NumberTitle','off');
  imshow(obj.im);
  Fig2 = figure('Toolbar','none','Menubar','none','Name',
                'Haussement Contraste','NumberTitle','off');
  imcontrast(I);
end
```

Définition du Filtrage

1. Introduction

L'étude d'un signal nécessite de supprimer au maximum le bruit parasite dû aux conditions d'acquisition. L'un des buts du filtrage est de nettoyer le signal en éliminant le plus de bruit possible tout en préservant le maximum d'informations.

Le filtrage des images a la même finalité que celui des signaux 1D. Il s'agit essentiellement d'enlever le bruit (parasite) ou de sélectionner certaines fréquences.
Le principe du filtrage est de modifier la valeur des pixels d'une image, généralement dans le but d'améliorer son aspect. En pratique, il s'agit de créer une nouvelle image en se servant des valeurs des pixels de l'image d'origine.
N'entrent pas dans la catégorie du filtrage toutes les transformations de l'image d'origine : zoom, découpage, projections, …
Les images peuvent être entachées de bruits de nature différente. On s'intéressera ici aux bruits
– additif, gaussien
– de flou (convolution)

Filtrage de Gauss

```
function B = filtrageGaus(im)
   Fig1 = figure('Position',[1260,210,450,450],'Toolbar','none',
      'Menubar','none','Name','Image originale','NumberTitle','off');
   imshow(im);
   B=imgaussfilt(im,'FilterSize',5);
   Fig = figure('Position',[200,210,450,450],'Toolbar','none',
         'Menubar','none','Name','Filtre Gauss','NumberTitle','off');
   imshow(B);
end
```

fspecial

Créer le filtre 2D prédéfini

Syntaxe

```
h = fspecial(type)

h = fspecial('average',hsize)

h = fspecial('disk',radius)

h = fspecial('gaussian',hsize,sigma)

h = fspecial('laplacian',alpha)

h = fspecial('log',hsize,sigma)

h = fspecial('motion',len,theta)

h = fspecial('prewitt')

h = fspecial('sobel')
```

Description

exemple de

`h = fspecial(type)` crée un filtre à deux dimensions `h` du `type` spécifié. Parmi les types de filtre ont des paramètres facultatifs supplémentaires, montrés dans les syntaxes suivantes. `fspecial` retourne `h` comme un noyau de corrélation, qui est la forme appropriée d'utiliser avec `imfilter`.

`h = fspecial('average',hsize)` retourne un étalement de filtre `h` de taille `hsize`.

`h = fspecial('disk',radius)` retourne un filtre moyenneur circulaire (tambourin) au sein de la matrice carrée de taille `2*radius+1`.

`h = fspecial('gaussian',hsize,sigma)` retourne un filtre passe-bas gaussien symétrie de révolution de taille `hsize` avec écart `sigma`. Non recommandé. Utilisez plutôt `imgaussfilt` ou `imgaussfilt3` .

`h = fspecial('laplacian',alpha)` renvoie un filtre 3-en-3 se rapprochant de la forme de l'opérateur laplacien bidimensionnelle, `alpha` contrôle la forme du Laplacien.

`h = fspecial('log',hsize,sigma)` retourne une symétrie de révolution laplacien de filtre gaussien de taille `hsize` avec écart `sigma`.

`h = fspecial('motion',len,theta)` retourne un filtre rapprochant, une fois convolé avec une image, le mouvement linéaire d'une caméra. `len` spécifie que la longueur de la motion et `theta` spécifie l'angle du mouvement en degrés dans un sens anti-horaire. Le filtre devient un vecteur pour les mouvements horizontaux et verticaux. La valeur par défaut `len` est `9` et le défaut de `theta` est `0`, ce qui correspond à un mouvement horizontal de neuf pixels.

`h = fspecial('prewitt')` retourne un filtre 3-en-3 qui souligne les bords horizontaux en se rapprochant d'un dégradé vertical. Pour souligner les bords verticaux, transposer le filtre `h'`.

```
[ 1   1   1
  0   0   0
 -1  -1  -1 ]
```

`h = fspecial('sobel')` retourne un filtre 3-en-3 qui met l'accent sur les bords horizontaux à l'aide de l'effet de lissage en se rapprochant d'un dégradé vertical. Pour souligner les bords verticaux, transposer le filtre `h'`.

```
[ 1   2   1
  0   0   0
 -1  -2  -1 ]
```

```
I = imread('cameraman.tif');
```

```
imshow(I);
```

Créer un filtre de requête et l'utiliser à brouiller l'image. Afficher l'image floue.

```
H = fspecial('motion',20,45);
MotionBlur = imfilter(I,H,'replicate');
imshow(MotionBlur);
```

Créer un filtre de disque et utilisez-le à brouiller l'image. Afficher l'image floue.

```
H = fspecial('disk',10);
blurred = imfilter(I,H,'replicate');
imshow(blurred);
```

Arguments d'entrée

`type` : Type de filtre

`'average'` | `'disk'` | `'gaussian'` | `'laplacian'` | `'log'` | `'motion'` | `'prewitt'` | `'sobel'`

Type de filtre, spécifié comme l'une des valeurs suivantes :

Valeur	Description
`'average'`	Filtre moyenneur
`'disk'`	Filtre moyenneur circulaire (casemate)
`'gaussian'`	Filtre gaussien passe-bas. Non recommandé. Utilisez plutôt `imgaussfilt` ou `imgaussfilt3` .
`'laplacian'`	Se rapproche de l'opérateur laplacien bidimensionnelle
`'log'`	Filtre laplacien de Gauss
`'motion'`	Se rapproche du mouvement linéaire d'un appareil photo
`'prewitt'`	Filtre d'insistant sur le bord horizontal de Prewitt
`'sobel'`	Sobel horizontal bord-mettant l'accent sur filtre

Les Types de données : `char` | `string`

imfilter

N-D de filtrage d'images multidimensionnelles

Syntaxe

```
B = imfilter(A,h)

B = imfilter(A,h,options,...)
```

Description

`B = imfilter(A,h)` filtres l'array multidimensionnel `A` avec le filtre multidimensionnelle `h` et renvoie le résultat dans `B`.

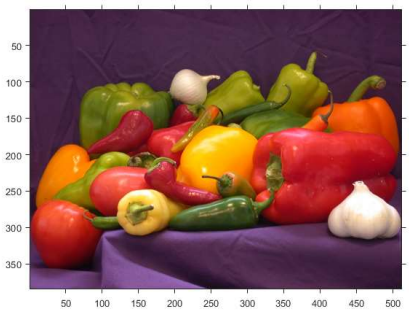
Facultativement, vous pouvez filtrer un tableau multidimensionnel avec un filtre 2-D à l'aide d'un GPU (nécessite Parallel Computing Toolbox™). Pour plus d'informations, consultez Traitement d'images sur un GPU.

`B = imfilter(A,h,options,...)` effectue un filtrage multidimensionnelle selon un ou plusieurs des options spécifiées.

Créer le filtre et l'appliquer

Lire une image couleur dans l'espace de travail et l'afficher.

```
originalRGB = imread('peppers.png');
imshow(originalRGB)
```



Créer un filtre de flou à l'aide de la fonction `fspecial` .

```
h = fspecial('motion', 50, 45);
```

Appliquer le filtre à l'image d'origine pour créer une image avec le flou de mouvement. Notez `imfilter` est plus efficace de mémoire que certaines autres fonctions de filtrage dans qu'elle génère un tableau du même type de données que le tableau de l'image d'entrée. Dans cet exemple, la sortie est un tableau de `uint8`.

```
filteredRGB = imfilter(originalRGB, h);
figure, imshow(filteredRGB)
```

Filtrer l'image, cette fois en spécifiant l'option de limite répétées.

```
boundaryReplicateRGB = imfilter(originalRGB, h, 'replicate');
figure, imshow(boundaryReplicateRGB)
```

Filtre à l'aide des Images imfilter par Convolution

Par défaut, `imfilter` utilise corrélation parce que les fonctions de conception de filtre de boîte à outils produisent des grains de corrélation. Utiliser le paramètre facultatif à utiliser la convolution.

Créer une matrice de l'échantillon.

```
A = magic(5)
A = 5×5

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Créer un filtre.

```
h = [-1 0 1];
```

Filtrer à l'aide de la corrélation, la valeur par défaut.

```
imfilter(A,h)
ans = 5×5

    24   -16   -16    14    -8
     5   -16     9     9   -14
     6     9    14     9   -20
    12     9     9   -16   -21
    18    14   -16   -16    -2
```

Filtre à l'aide de convolution, spécifiant `imfilter` avec le paramètre optionnel.

```
imfilter(A,h,'conv')
ans = 5×5

   -24    16    16   -14     8
    -5    16    -9    -9    14
    -6    -9   -14    -9    20
   -12    -9    -9    16    21
   -18   -14    16    16     2
```

Convertir Image classe afin d'éviter des valeurs négatives de sortie

Dans cet exemple, la sortie de `imfilter` a des valeurs négatives, lorsque l'entrée est de classe `double`. Pour éviter des valeurs négatives, convertir l'image en un autre type de données avant d'appeler `imfilter`. Par exemple, lorsque le type d'entrée est `uint8`, `imfilter` tronque les valeurs de sortie à `0`. Il serait également approprié convertir l'image en un type entier signé.

```
A = magic(5)
A = 5×5

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Filtrer l'image avec `imfilter`.

```
h = [-1 0 1];
imfilter(A,h)
ans = 5×5

    24   -16   -16    14    -8
     5   -16     9     9   -14
     6     9    14     9   -20
    12     9     9   -16   -21
    18    14   -16   -16    -2
```

Notez que le résultat a des valeurs négatives. Pour éviter des valeurs négatives dans l'image de sortie, convertissez l'image d'entrée `uint8` avant d'effectuer le filtrage. Depuis l'entrée de `imfilter` est de classe `uint8`, la sortie est également de classe `uint8`et `imfilter` tronque les valeurs négatives pour `0`.

```
A = uint8(magic(5));
imfilter(A,h)
ans = 5x5 uint8 matrix

   24     0     0    14     0
    5     0     9     9     0
    6     9    14     9     0
   12     9     9     0     0
   18    14     0     0     0
```

Créer un filtre et l'appliquer sur un GPU

Lire une image couleur dans l'espace de travail comme un `gpuArray` et l'afficher.

```
originalRGB = gpuArray(imread('peppers.png'));
imshow(originalRGB)
```

Image d'origine

Créer un filtre, `h`, qui permet de rapprocher les mouvements de caméra linéaire.

```
h = fspecial('motion', 50, 45);
```

Appliquer le filtre, à l'aide `imfilter`, à l'image `originalRGB` pour créer une nouvelle image, `filteredRGB`. L'image est retournée comme un `gpuArray`.

```
filteredRGB = imfilter(originalRGB, h);
figure, imshow(filteredRGB)
```

Image filtrée

Notez `imfilter` est plus efficace de mémoire que certaines autres opérations de filtrage dans qu'elle génère un tableau du même type de données que le tableau de l'image d'entrée. Dans cet exemple, la sortie est un tableau de `uint8`.

```
whos
Name                Size                Bytes  Class       Attributes

  filteredRGB       384x512x3             108  gpuArray
  h                  37x37              10952  double
  originalRGB       384x512x3             108  gpuArray
```

Essayez l'opération de filtrage, cette fois en spécifiant l'option limite de `replicate` .

```
boundaryReplicateRGB = imfilter(originalRGB, h, 'replicate');
figure, imshow(boundaryReplicateRGB)
```

Image avec limite répétée

Arguments d'entrée

A — Image à filtrer
tableau numérique

Image pour être filtrée, spécifiée comme un tableau de toute classe et de la dimension.

Les Types de données : `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`h` — Multidimensional filtrer
Tableau de N-D des données de type `double`

Filtre multidimensionnel, spécifié comme un tableau de N-D de type de données `double`.

Pour effectuer le filtrage d'image utilisant un GPU, spécifiez `h` comme un vecteur ou matrice 2D de type de données `double`.

Les Types de données : `double`

`options` : Options afin de contrôler l'opération de filtrage
vecteur de caractère | chaîne scalaire | scalaire numérique

Options qui contrôlent l'opération de filtrage, spécifiée comme un vecteur de caractère, chaîne numérique ou scalaire. Le tableau suivant répertorie toutes les options prises en charge.

Options de limites

Option	Description
Options de remplissage	
scalaire numérique, `X`	Tableau d'entrée des valeurs en dehors des limites du tableau reçoivent la valeur `X`. Lorsque aucune option de remplissage est spécifiée, la valeur par défaut est `0`.
`'symmetric'`	Tableau d'entrée des valeurs en dehors des limites du tableau sont calculées par miroir réfléchissant le tableau à travers la frontière de tableau.
`'replicate'`	Tableau d'entrée des valeurs en dehors des limites du tableau sont censés égale à la valeur de bordure de tableau le plus proche.
`'circular'`	Tableau d'entrée des valeurs en dehors des limites du tableau sont calculées en supposant implicitement que le tableau d'entrée est périodique.
Taille de sortie	
`'same'`	Le tableau de sortie est la même taille que le tableau d'entrée. Il s'agit du comportement par défaut lorsque aucune option de taille de sortie n'est spécifiées.
`'full'`	Le tableau de sortie est le résultat filtré complète et donc est plus grand que le tableau d'entrée.
Options de Convolution et de corrélation	
`'corr'`	`imfilter` effectue un filtrage multidimensionnelles à l'aide de corrélation, qui est de la même que façon que `filter2` effectue le filtrage. Lorsque aucune corrélation ou convolution option est spécifiée, `imfilter` utilise la corrélation.

Option	Description
'conv'	imfilter effectue multidimensionnelle de filtrage à l'aide de convolution.

Exemple
Filtre laplacian

```
function BW =filtre_laplacian(im)
  clc;
  % create a new figure to show the image.
  I=imread(im)
  J=Rgb2Gray(I);
  f1=figure('Toolbar','none','Menubar','none','Name',
            'Image originale','NumberTitle','off');
  imshow(I);

  f1=figure('Position',[200,210,450,450],'Toolbar','none',
     'Menubar','none','Name','Filtre Laplacien','NumberTitle','off');
  % create laplacian filter.
  H = fspecial('laplacian');
  % apply laplacian filter.
  BW = imfilter(J,H);
  mshow(BW); title(' Image filtré par le laplacien')
end
```

Exemple
Filtre moyen

```
function BW =filtre_Moyen(obj)
 clc;
 % create a new figure to show the image .
 I=imread(im)
 J=Rgb2Gray(I);
 f1=figure('Toolbar','none','Menubar','none','Name',
           'Image originale','NumberTitle','off');
 imshow(I);

 f2=figure('Position',[200,210,450,450],'Toolbar','none',
'Menubar','none','Name','Filtre moyenne','NumberTitle','off');
 % create laplacian filter.
 H = fspecial('average');
 % apply average filter.
 BW = imfilter(J,H);
 imshow(BW); title('filtrage d'image par le filtre moyen')
end
```

medfilt2

medfilt2

2-D de filtrage médian

Syntaxe

```
J = medfilt2(I)

J = medfilt2(I,[m n])

J = medfilt2(___,padopt)
```

Description

`J = medfilt2(I)` effectue un filtrage médian de l'image `I` en deux dimensions. Chaque pixel de sortie contient la valeur médiane dans un quartier 3-en-3 autour du pixel correspondant dans l'image d'entrée.

Facultativement, vous pouvez calculer la normalisée corrélation croisée à l'aide d'un GPU (nécessite Parallel Computing Toolbox™). Pour plus d'informations, consultez

`J = medfilt2(I,[m n])` effectue un filtrage médian, où chaque pixel de sortie contient la valeur médiane dans le `m`- par -`n` quartier autour le pixel correspondant à l'image d'entrée.

Exemple

```
function BW=filtreMediane(obj)
  % create a new figure to show the image .
  f1=figure('Toolbar','none','Menubar','none','Name',
            'Image originale','NumberTitle','off');
  % show the loaded image.
  I=imread(im)
  imshow(I);
  J=Rgb2Gray(I);
  % apply median filter using the internal matlab function medfilt2 .
  BW=medfilt2(J,[5 5]);

  f2=figure('Position',[200,210,450,450],'Toolbar','none',
       'Menubar','none','Name','Filtre médiane','NumberTitle','off');
  % show image after applying the filter
  imshow(BW);
  % write the new image to the file system .
  imwrite(BW,'newimage.gif','GIF');
 end
```