TD #1 – CETS8AH

4 mars 2025
Erwan Kerrien

Ce TD est noté et est à faire seul. **Seuls les exercices 2 et 4 sont à rendre et seront notés.** Vous me le rendrez pour le 18 mars sous format électronique : envoyez par mail à `Erwan.Kerrien@inria.fr` les M-files que vous aurez écrits, ainsi qu'un texte de réponse (format Word, Libre/OpenOffice ou Latex, ou autre si pas trop exotique). Des images pour appuyer vos conclusions et interprétations seront les bienvenues.

Les autres exercices (1 et 3) sont optionnels. Je vous conseille cependant de commencer aujourd'hui par l'exercice 1 qui vous explique comment fonctionne le filtrage d'image sous MATLAB. Ceci vous sera nécessaire pour faire l'exercice 2. L'exercice 3 est plutôt pour vous assurer que vous avez bien compris et faire un petit filtrage sympa. L'exercice 4 est indépendant du reste.

Ce TD impliquera sans aucun doute du travail hors cours. Je reste cependant disponible pour répondre par mail à toutes les questions que vous vous posez. Je ne résoudrai bien entendu pas le TD pour vous, mais mon objectif est de m'assurer que vous avez compris. N'hésitez donc pas à me solliciter.

1 Le filtrage sous MATLAB *(Non noté, bonus)*

Il existe trois fonctions pour effectuer une convolution en 2D sous MATLAB : `conv2`, `filter2` et `imfilter`. Pour bien comprendre leurs options ainsi que leurs différences, nous devons rentrer dans le détail de la convolution en pratique.

Si vous regardez la formule de convolution, vous pouvez remarquer qu'il est possible d'obtenir des valeurs non-nulles pour une image plus grande que l'image originale. Clarifions cela avec un exemple en 1D. Si vous prenez le vecteur (faisant office d'image) `I=[1,1,1,1]` et que vous le convoluez avec le filtre lisseur `h=1/4*[1,2,1]`, vous obtenez `J` tel que ;

```
J(i) = 0.25*I(i-1)+0.5*I(i)+0.25*I(i+1)
```

Vous voyez qu'il y a un problème aux bords : par exemple, pour calculer `J(1)`, il faut une valeur `I(0)`. Quelle doit être cette valeur ? Plusieurs possibilités existent, qui sont, soit dit en passant, liées au problème d'aliasing dans le calcul de la transformée de Fourier. Les deux principales sont :

1. plonger `I` dans un champ de zéros, de telle sorte que `I(0) = 0`
2. prolonger `I` par une opération miroir, de telle sorte que `I(0) = I(2)`

Dans les deux cas, un prolongement à l'infini de l'image est possible. On peut donc a priori calculer un vecteur infini. Seulement, dans le premier cas, `J` prend des valeurs nulles assez rapidement, et dans le deuxième cas, `J` devient aussi périodique.

MATLAB fait la première hypothèse. Dans ce cas, `J` est formé de trois zones :

1. une zone centrale de valeurs qui sont telles que les valeurs de `I` impliquées dans leur calcul sont valides (c'est-à-dire que leur indice est compris entre 1 et `size(I)`).
2. une zone externe formée uniquement de zéros car les valeurs de `I` utilisées pour leur calcul sont nulles car toutes non-valides.
3. une zone intermédiaire de valeurs calculées à partir de valeurs de `I` dont une au moins est valide et une au moins est invalide.

Cependant, la zone centrale a une taille plus petite que le vecteur `I` en entrée (`J(1)` appartient à la zone intermédiaire), et si on lui adjoint la zone intermédiaire, on obtient un vecteur plus grand que le vecteur en entrée (`J(0)` appartient aussi à la zone intermédiaire).

Il y a donc trois résultats intéressants en sortie de la convolution. Chacun de ces résultats correspond à une option `'shape'` sous MATLAB :

1. juste la zone centrale (juste les valeurs fiables) : option `'valid'`
2. la zone centrale + la zone intermédiaire (toute valeur potentiellement non-nulle) : option `'full'`
3. une zone centrale, mais comprenant également des éléments de la zone intermédiaire de telle façon que le vecteur résultat `J` ait la même longueur que le vecteur en entrée `I` : option `'same'`

   Cette zone est centrale dans le sens où le nombre d'éléments de la zone intermédiaire ajoutés à gauche et à droite est le même. Dans le cas où la taille de `h` est paire, ceci n'est pas strictement possible, aussi un élément de plus

qu'à gauche est ajouté à droite. Par exemple, si `I=[1 2 3 4]` et `h=[1 2]`, le résultat de `conv2(I,h,'same')` est le vecteur `[4 7 10 8]`, ce qu'on obtient en ajoutant un 0 en fin de vecteur `I`, mais sans en mettre au début.

Ceci étant bien compris, les différences entre les trois fonctions permettant de faire de la convolution/filtrage sous MATLAB sont les suivantes :

- `conv2(A,B)` effectue la convolution de l'image `A` avec le noyau `B`. Il faut donc prendre en compte la symétrie centrale du noyau `B` avant de convoluer l'image. Cette fonction ne s'applique qu'à des images en niveaux de gris et renvoie un résultat avec l'option `'full'` par défaut.
- `filter2(A,B)` effectue la correlation de l'image `B` avec le noyau `A`. On a donc une inversion du sens des arguments. De plus comme le filtrage se fait par correlation, il n'y a pas de symétrie centrale à considérer pour le noyau. Cette fonction ne s'applique qu'à des images en niveaux de gris et renvoie un résultat avec l'option `'same'` par défaut.
- `imfilter(A,B)` effectue le filtrage de l'image `A` par le noyau `B`. Cette fonction s'applique à toute sorte d'image, en particulier en niveaux de gris ou en couleurs. Son comportement par défaut est d'effectuer un filtrage par corrélation et renvoie un résultat avec l'option `'same'`. Ces options peuvent être changées et vous pouvez de plus modifier la manière dont l'image `A` est étendue à l'espace tout entier pour calculer les valeurs intermédiaires. Voir l'aide de cette fonction sous MATLAB pour plus de précision.

La fonction `imfilter` est à favoriser dans le cadre du traitement de l'image car c'est la mieux adaptée. Cependant, cette fonction fait partie d'un toolkit optionnel de MATLAB nommé Image Processing. Il vous faut donc connaître les deux autres fonctions afin de pouvoir vous débrouiller dans un contexte limité.

Enfin, la fonction `fspecial` permet de générer tout une batterie de filtres classiques que vous pouvez ensuite utiliser pour filtrer une image. Lisez l'aide de cette fonction. Attention, cette fonction fait également partie du toolkit Image Processing.

2 Filtre de Sobel *(Noté)*

Cet exercice a pour objectif de retrouver les coefficients du filtre de Sobel. Ce filtre implémente une dérivation discrète sur une image 2D $I(x, y)$. Vous pouvez générer le filtre de Sobel vertical (selon y) par un appel à `fspecial('sobel')`. Son application par convolution produira donc une approximation discrète de la dérivée $\frac{\partial I}{\partial y}$. Le filtre de Sobel horizontal s'obtient par transposition du précédent.

Dans cet exercice, on va commencer par le cas d'un signal 1D $f(x)$ et retrouver la formule de dérivation discrète pour approcher $f'(x)$. On transcrira cette expression sous forme d'un filtre 1D. Puis on analysera le filtre de Sobel pour en comprendre les subtilités de design.

Vous pourrez valider votre implantation en comparant vos résultats avec ceux fournis par la fonction `imgradient` de MATLAB (voir la doc). Cette fonction prend un argument optionnel nommé `'method'` qui est par défaut `'sobel'`.

1. Écrivez le développement de Taylor d'une fonction $f(x+\delta)$, avec δ petit. En posant $\delta = 1$, retrouvez les coefficients d'un filtre dérivateur 1×2.
2. Le filtre précédent ne préserve pas la position du gradient. En écrivant le développement de Taylor de $f(x-\delta)$, trouvez un autre filtre 1×2. En le combinant avec le développement précédent, retrouvez un filtre dérivateur 1×3, version de celui de Sobel en 1D.
3. Le filtre de Sobel vertical, renvoyé par `fspecial('sobel')` s'obtient en convoluant le filtre (colonne) ci-dessus par un filtre (ligne) 1×3. En reprenant le filtre de Sobel obtenu par `fspecial`, donnez les coefficients de ce filtre ligne. Quel est son effet ?

   Note : *il est possible que vous retrouviez un filtre connu à un coefficient multiplicatif près. Ceci est normal car il est difficile de normaliser un filtre dérivateur puisque la somme de ses coefficients vaut 0.*

   Note : *La possibilité de décomposer le filtre de Sobel en deux filtres orthogonaux correspond au fait que le filtre de Sobel est* **séparable**
4. Ecrivez une fonction `sobelNorm` qui renvoie la norme L_2 du gradient. Pour mémoire, le gradient d'une fonction (image) `I` est le champ de vecteurs formé en tout point par `[DxI DyI]`. L'image `DyI` s'obtient en convoluant l'image `I` par le filtre de Sobel vertical, et l'image `DxI` s'obtient en convoluant l'image `I` par le filtre de Sobel horizontal (transposé du filtre vertical). Etant données ces deux images `DxI` et `DyI`, la norme du gradient s'obtient par `sqrt(DxI.^2+DyI.^2)`. Testez cette fonction sur diverses images. Quelles particularités de l'image peut-on espérer détecter avec ce filtre ? Donnez un exemple où cela n'est pas satisfaisant ?

   Note : *Attention à la visualisation du résultat. L'application de la racine carrée change le type numérique pour vos images, et les passe en double. La fonction* `imshow` *va donc considérer par défaut que les valeurs sont dans*

$[0,1]$, *ce qui a pour effet d'afficher une image avec des pixels noirs ou blancs, mais pas en niveaux de gris. Pour bien visualiser l'image, indiquez la plage de valeurs à considérer, soit explicitement (ex :* `imshow(I,[0,255])`*), soit implicitement (ex :* `imshow(I,[])`*).*

3 Unsharp masking *(Non noté)*

Il existe plusieurs techniques pour réaliser un unsharp masking. Nous allons brièvement en explorer une ici.

L'unsharp masking correspond à la transformation

$$g(x,y) = f(x,y) - \gamma \Delta f(x,y)$$

où γ est un paramètre de gain strictement positif et $\Delta f(x,y)$ est le laplacien de f donné par :

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

f est l'image donnée en entrée, et g est l'image résultat.

1. écrivez une fonction `unsharp` qui implémente cette transformation, et prenant en paramètre une image et un gain. Le laplacien sera calculé par convolution avec le noyau suivant :

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix}$$

   Testez le résultat sur l'image `lena.jpg`, ainsi que d'autres, pour différentes valeurs du gain. (N.B. : γ n'est pas forcément un entier ! Faites aussi attention à la classe de votre image.).

   Note : la fonction `fspecial('laplacian')` donne une expression légèrement plus précise que le filtre ci-dessus. Encore une fois, ceci est à un facteur multiplicatif près car le normalisation du filtre est délicate (la somme des coefficients est nulle).
2. une autre manière d'implanter le laplacien d'une fonction est de l'approximer par la différence de gaussiennes :

$$\Delta f \approx (G_{K\sigma} - G_\sigma) * f$$

   Cette approximation est particulièrement précise quand $K = 1.6$. Écrivez une nouvelle fonction `unsharpGauss` qui implémente la transformation. Cette fonction prendra un paramètre supplémentaire qui est σ l'écart-type pour les gaussiennes. Testez cette nouvelle fonction sur `lena.jpg`, ainsi que d'autres images, pour différentes valeurs du gain et de l'écart-type.

   Note : Vous utiliserez la fonction `fspecial` avec l'option `'gaussian'` pour générer un filtre gaussien. Cette fonction prend deux paramètres supplémentaires qui sont `HSIZE` la taille du filtre ainsi que `SIGMA`, l'écart-type de la gaussienne. Afin d'avoir une taille de filtre adaptée à l'écart-type de la gaussienne, vous considérerez la relation `HSIZE=8 * sqrt(2) * SIGMA` et ferez en sorte que `HSIZE` soit un entier impair. Attention, vous devrez également faire en sorte que les deux filtres gaussiens soient de la même taille.
3. Quel est l'effet de cette transformation ? Vous pouvez comparer vos résultats avec le filtre donné par un appel à `fspecial('unsharp',ALPHA)` pour différentes valeurs de `ALPHA` (voir l'aide de `fspecial`). Encore une fois, le résultat est-il toujours satisfaisant ?

4 Transformée de Fourier *(Noté)*

1. Soit la fonction `FT` :

```
function s=FT(I)
  s=fftshift(fft2(I));
end
```

   `FT` calcule la transformée de Fourier d'une image `I` en s'assurant que les coefficients pour les fréquences nulles sont placés au centre de l'image résultat. De quel type sont les éléments de `s` ?
2. Lisez la doc de la fonction matlab `abs`. Que représente `abs(s)` (en terme de signal) ?

3. Chargez l'image `bureau.jpg` dans la variable `I` et calculez `s=FT(I)`. Si vous affichez l'image `abs(s)`, vous ne voyez qu'un point blanc sur un fond noir : le max de `abs(s)` est très grand et très concentré, ce qui empêche une visualisation correcte des choses. Il faut saturer cette image : affichez l'image `abs(s)` au moyen de la commande `imshow(abs(s),[0 100000])`. Puis faites la même chose avec l'image `barbara.jpg`. Quelle différence faites-vous entre les deux images finales ? Comment l'expliquez-vous ? Les questions suivantes utilisent la valeur de `s` obtenue avec `barbara.jpg`.
4. Mettez à zéro les éléments `s(i,j)` tels que $1 \leq i \leq 170$ ou $1 \leq j \leq 170$ ou $343 \leq i \leq 512$ ou $343 \leq j \leq 512$. Puis inversez la transformée de Fourier `s` en lui appliquant la fonction :

   ```
   function iI=iFT(s)
     iI=abs(ifft2(fftshift(s)));
   end
   ```

   Affichez le résultat au moyen de la commande `imshow(uint8(iI))`. Vous devriez remarquer deux phénomènes apparemment contraires. Quels sont-ils ? Pourquoi apparaissent-ils ?