Convolution

Les exercices à réaliser sont situés dans la base de code que vous récupérez en vous inscrivant sur le lien GitHub classroom reçu par mail [1]. Lisez bien le readme du dépôt pour comprendre comment l'utiliser. La majorité des fonctions demandées existent déjà dans OpenCV : **le but n'est pas d'utiliser les fonctions d'OpenCV mais de les coder vous même !** Nous utiliserons donc uniquement les conteneurs de base d'OpenCV et les fontions d'entrée/sortie.

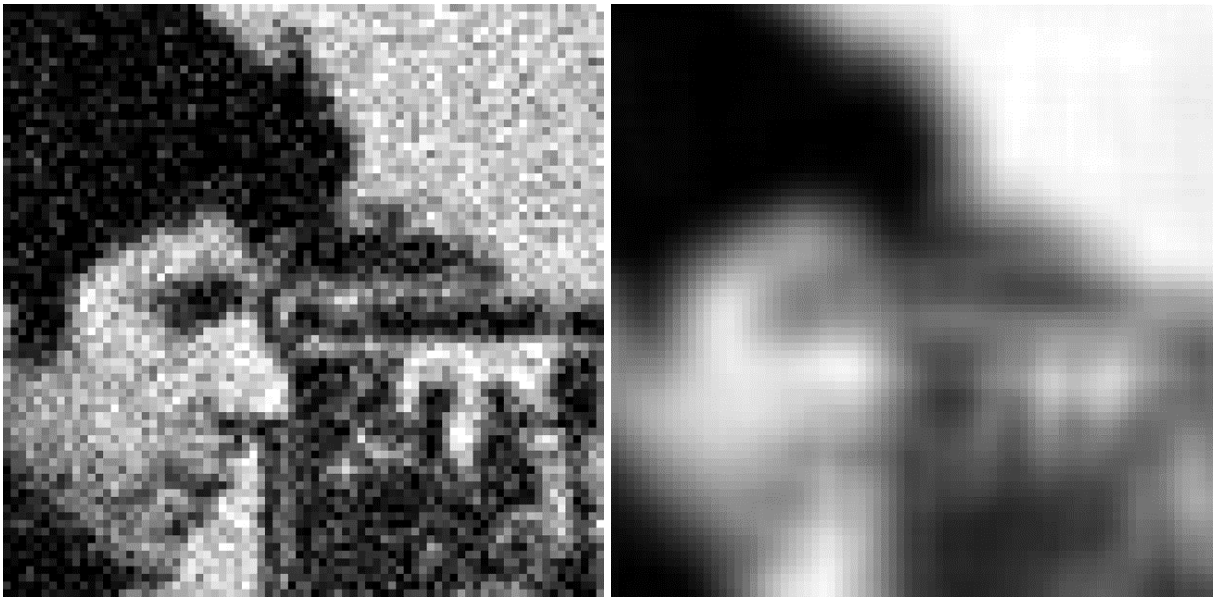
Important

Au cours de ce chapitre, vous compléterez le fichier ``tpConvolution.cpp`` que vous devrez pousser sur votre dépôt git avant la prochaine séance (cf. consignes détaillées envoyées par mail).

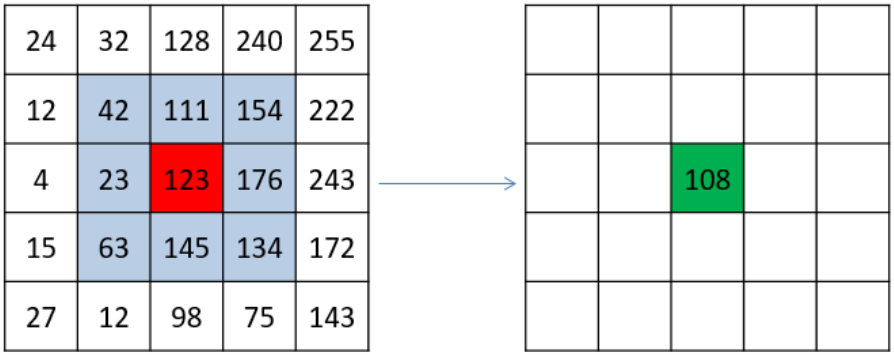
Filtre moyenneur

Le filtre moyenneur est une opération de traitement d'images utilisée pour réduire le bruit dans une image et/ou flouter une image. Par exemple, l'application d'un filtre moyenneur sur l'image de gauche donne l'image de droite :

En zoomant, on peut voir en détail les effets du filtre; le bruit clairement visible dans le ciel a bien été réduit mais les détails du visage et de la caméra sont floutés :



Le filtre moyenneur fait parti de la catégorie des filtres d'images locaux car pour calculer la nouvelle valeur d'un pixel, il regarde la valeur des pixels proches. Concrètement, la valeur filtrée d'un pixel p est égale à la moyenne des valeurs des pixels proches de p. En général, on définit les « pixels proches de p » comme l'ensemble de pixels contenus dans un carré de largeur k centré sur p :



Avec un filtre moyenneur de largeur 3, pour calculer la nouvelle valeur du pixel rouge de l'image originale de gauche, on calcule la valeure moyenne des pixels situés dans un carré de dimension 3×3 centré sur ce pixel. Cela donne la nouvelle valeur du pixel sur l'image transformée (pixel vert sur l'image de droite) : $\frac{42+111+154+23+123+176+63+145+134}{9} = 108$

Cette opération est répétée pour tous les pixels de l'image. On parle de *fenêtre glissante* pour caractériser le carré sur lequel est calculé la moyenne des pixels et qui se déplace sur l'image :

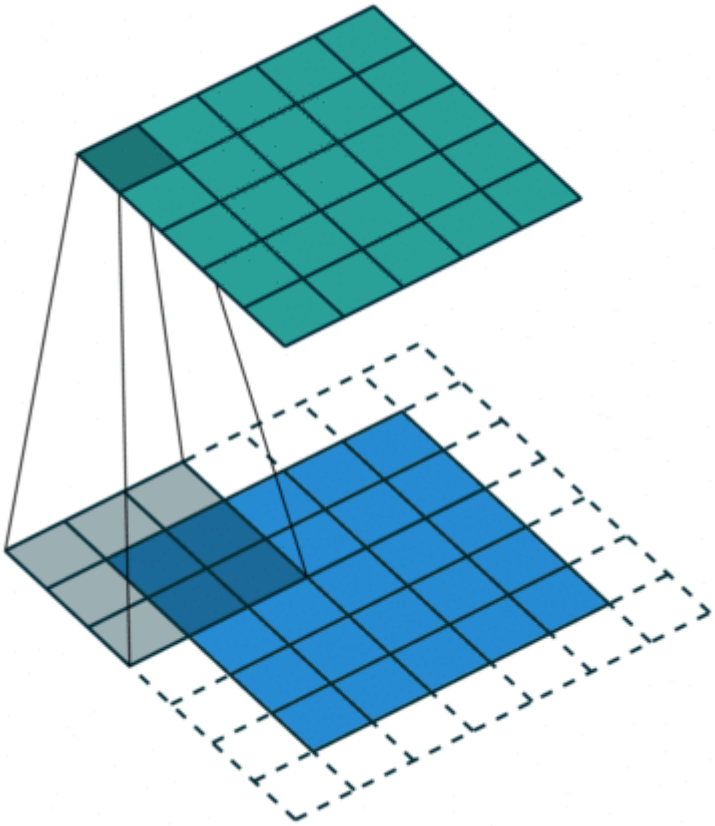


Illustration du principe de la fenêtre glissante. La fenêtre se déplace sur l'image du bas (en bleu) pour calculer les valeurs de la nouvelle image en haut (en vert). Source https://github.com/vdumoulin/conv_arithmetic.

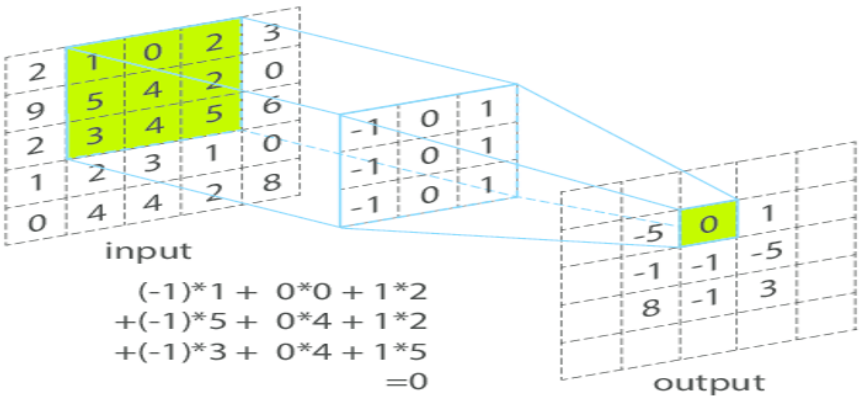
Exercice 1 : Filtre moyenneur

Implémentez le filtre moyenneur dans la fonction `meanFilter` du fichier `tpConvolution.cpp`. Pensez à valider votre implémantion avec la commande `test`.

Notez que pour une valeur de paramètre k, la fenêtre de calcul à utiliser est de taille $2k + 1$: cette pratique commune permet de garantir que la fenêtre considérée est de dimension impaire et donc que son centre tombe précisément sur un pixel de coordonnées entières.

Convolution

La *convolution*, ou *produit de convolution*, est une généralisation du filtre moyenneur où l'on considère cette fois une moyenne pondérée. La fenêtre glissante est alors elle même une image qui contient les coefficients de pondération. On l'appelle généralement *noyau de convolution* ou *masque de convolution* (*kernel* ou *mask* en anglais) :



Le noyau de convolution (au centre) contient les coefficients de pondération. Le principe est alors similaire au filtre moyenneur : pour calculer la nouvelle valeur d'un pixel à droite, on calcule la moyenne des pixels de l'image originale (à gauche) se trouvant sous le masque de convolution pondérée par les valeurs du masque.

Formellement, le produit de convolution est une opération entre deux images en niveau de gris f et g, notée $f * g$ défini par:

$$\forall (x, y) \in \mathbb{Z}^2, (f * g)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i, j) g(x - i, y - j)$$

Dans cette expression on peut reconnaitre:

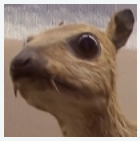
- $g(x - i, y - j)$: qui comporte une opération de translation par le vecteur (x, y) et une symétrie centrale (inversion des coordonnées i et j) : il s'agit du déplacement de l'image g à la position (x, y) (g joue donc le rôle de fenêtre glissante). La symétrie centrale est présente pour une raison technique et n'a pas vraiment d'importance en pratique,
- la multiplication *point à point* de f par la translatée de g : c'est l'opération de pondération,
- la sommation du tout : c'est l'opération de moyennage.

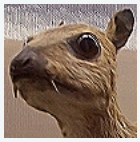
Le produit de convolution est une application bilinéaire, associative et commutative. C'est-à-dire que pour toutes images f, g, h et pour tout scalaire λ, on a :

- $f * (g + \lambda h) = (f * g) + \lambda (f * h)$;
- $(f * g) * h = f * (g * h)$;
- $f * g = g * f$.

Le choix du noyau de convolution va permettre d'obtenir différents effets :

Exemple de convolutions (source images: Wikipédia)

Effet	Noyau	Résultat
Identité (ne fait rien...)	$\begin{bmatrix} 1 \end{bmatrix}$	

Effet	Noyau	Résultat
Filtre moyenneur (lissage)	$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	
Filtre gaussien 3×3 (lissage)	$\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$	
Filtre gaussien 5×5 (lissage)	$\frac{1}{256}\begin{bmatrix}1&4&6&4&1\\4&16&24&16&4\\6&24&36&24&6\\4&16&24&16&4\\1&4&6&4&1\end{bmatrix}$	
Filtre réhausseur (renforce les contours)	$\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix}$	
Filtre Laplacien (détecteur de contours)	$\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$	

Exercice 2 : Produit de convolution

Implémentez le produit de convolution dans la fonction `convolution` du fichier `tpConvolution.cpp`. Pensez à valider votre implémantion avec la commande `test`.

Note

Le produit de convolution est intimement lié à la transformée de Fourier par le *théorème de la convolution*. En effet si on note par F et F^{-1} la tranformée de Fourier et la transformée de Fourier inverse, on a l'égalité suivante :

$$f * g = F^{-1}(F(f) \circ F(g))$$

ou $\circ$ représente la multiplication point à point, aussi appelé *produit d'Hadamard* (on multiplie les 2 valeurs d'un même pixel dans les 2 images).

Autrement dit, pour calculer $f * g$, on peut procéder ainsi :

1. on calcule les transformées de Fourier des images f et g,
2. on multiple les 2 résultats des transformées de Fourier entre eux,
3. on prend la transformée de Fourier inverse du résultat de la multiplication.

Cette approche présente une complexité algorithmique inférieure au calcul *naïf* grâce à l'algorithme de transformée de Fourier rapide (Fast Fourier Transform (FFT)). Néanmoins, la constante multiplicative est plus élevée qu'avec l'approche naïve et la convolution de Fourier n'est intéressante que lorsque le noyau de convolution est relativement grand (plusieurs dizaines de pixels de côté).

Détection de contours

La détection de contours consiste à chercher les courbes continues le long des zones de fortes variations dans l'image. Les expériences en neurosciences ont en effet montrés que la détection de contours est une des premières étapes réalisées par le cortex visuel, suggérant leur importance pour les processus d'analyse d'images.

Note

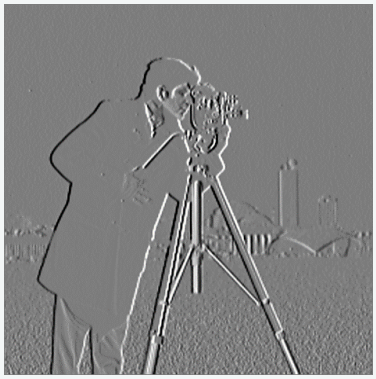
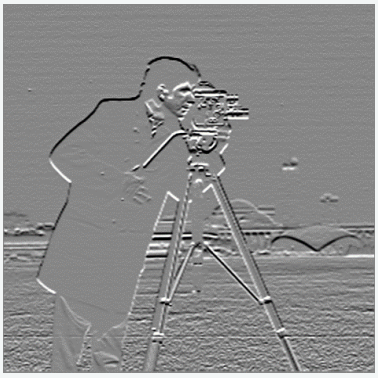
L'étude du cortex visuel à notamment progressé grâce à une série d'expériences menées sur des chats dans les années 60. Une des expériences constistait par exemple à élever un chaton dans un environnement où seules des lignes verticales sont visibles; après quelques semaines, on constatait alors l'absence de réponse du cortex visuel en présence de lignes horizontales, montrant ainsi que le cerveau du chat, en l'absence de stimulation appropriée, n'a pas pû « entrainer son détecteur de contours » sur l'axe horizontal. Plus d'information sur ce site

La détection de zones de variation des niveaux de gris de l'image correspond à l'opération de dérivation. Comme une image numérique n'est pas une fonction continue, la notion de dérivée n'est pas formellement définie et on utilisera un analogue appelé *gradient*. Comme une image a 2 dimensions, le gradient de l'image f, notée ∇f, est une image vectorielle, donnée par les deux dérivées partielles :

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right).$$

Sur l'exemple ci-dessous, on peut observer une image et ses 2 dérivées partielles. On constate que les variations horizontales (respectivement verticales) apparaissent dans la dérivée partielle $\frac{\partial f}{\partial x}$ (respectivement $\frac{\partial f}{\partial y}$).

Gradient de l'image camera

		
f	$\frac{\partial f}{\partial x}$	$\frac{\partial f}{\partial y}$

On peut également regarder le vecteur du gradient sous sa forme polaire $(||\nabla f||, dir(\nabla f))$, avec :

- $||\nabla f||$ la norme du gradient : $||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$ ou, pour simplifier les calculs $||\nabla f|| = \left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|$
- et $dir(\nabla f)$ la direction du gradient : $dir(\nabla f) = \text{atan2}\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right)$

Cela donne les images suivantes (notez que les niveaux de gris de $dir(\nabla f)$ représente des angles et son interprétation visuelle n'est donc pas évidente) :

Gradient de l'image camera (forme polaire)

		
f	$\|\|\nabla f\|\|$	$dir(\nabla f)$

En pratique, il n'existe pas de définition unique pour le calcul des dérivées partielles $\frac{\partial f}{\partial x}$ et $\frac{\partial f}{\partial y}$. Plusieurs solutions exprimables sous forme de produit de convolution ont été proposées, celle que nous allons voir est appellée *gradient de Sobel*.

La méthode de Sobel définit les dérivées partielles de la manière suivante :

$$\frac{\partial f}{\partial x} = f * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

et

$$\frac{\partial f}{\partial y} = f * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Le choix de ces noyaux peut s'expliquer en décomposant les noyaux de convolution selon leur contribution dans chacune des dimensions. Par exemple, pour la dérivée partielle selon x, on a :

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Dit autrement, le produit de convolution étant un application linéaire, on peut réécrire la définition de $\frac{\partial f}{\partial x}$ sous la forme :

$$\frac{\partial f}{\partial x} = \left(\left(f * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \right) * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \right)$$

C'est à dire qu'on commence par convoluer f avec le noyau vertical, puis on convolue avec le noyau horizontal :

- la première convolution réalise un lissage de l'image sur l'axe vertical (le noyau utilisé est la colonne centrale du noyau gaussien 3×3),
- la deuxième convolution calcule, pour chaque pixel p, la différence entre le pixel à droite de p et le pixel à gauche de p (c'est donc une mesure de variation).

Exercice 3 : Détecteur de contours de Sobel

Implémentez la fonction `edgeSobel` du fichier `tpConvolution.cpp` qui calcule la norme du gradient selon la méthode de Sobel. Pensez à valider votre implémantion avec la commande `test`.

Filtre bilatéral

L'objectif de ce dernier exercice est de réutiliser les éléments acquis dans les exercices précédents pour comprendre et implémenter une transformation décrite dans un autre contexte.

L'opération de convolution est efficace pour débruiter une image et lisser les zones texturées. Malheureusement, elle a également une forte tendance à flouter les contours. Il existe une variation de l'opération de convolution appelée filtre bilatéral qui résoud ce problème en introduisant une seconde pondération afin de ne pas donner trop de poids à un pixel dont la valeur est éloignée de la valeur du pixel courant.

Exercice 4 : Filtre bilatéral

Implémentez le filtre bilatéral dans la fonction `bilateralFilter` du fichier `tpConvolution.cpp`. Pensez à valider votre implémantion avec la commande `test`. Cette méthode est décrite sur de nombreux sites Web, par exemple https://people.csail.mit.edu/sparis/bf_course/slides/03_definition_bf.pdf

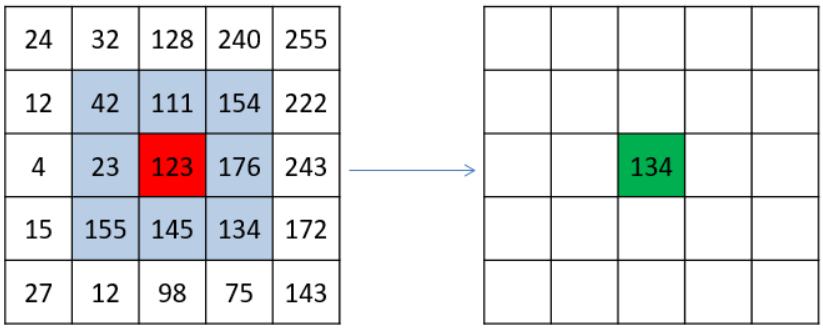
[1]

[1] La base de code est également récupérable ici

Filtre médian

L'idée du *filtre médian* reprend le principe de la fenêtre glissante mais ne s'intéresse pas à une combinaison linéaire des valeurs des pixels dans la fenêtre (contrairement au produit de convolution). Au lieu de cela, on va trier les pixels situés sous la fenêtre par ordre de niveau de gris croissant et prendre le pixel se trouvant au milieu : on parle d'élément *médian*.

Par exemple, si l'on considère la liste de valeur $(1, 3, 6, 9, 223)$: cette liste est triée et comporte 5 éléments; le médian de cette liste est le 3ème élément, c'est-à-dire le 6. Par rapport à la moyenne, le médian à l'avantage d'être robuste aux valeurs extrêmes qui ne sont pas forcément représentatives de la distribution des valeurs dans la liste. Dans le cas précédent, la moyenne vaut 48.4, elle est fortement influencée par la valeur 223 qui n'est pas vraiment représentative du reste des éléments de la liste. Alors que le médian à 6 semble effectivement mieux représenter les valeurs de la liste.



$$23 \leq 42 \leq 111 \leq 123 \leq 134 \leq 145 \leq 154 \leq 155 \leq 176$$

Calcule du filtre médian sur une image. La fenêtre est représentée en bleue sur l'image d'origine (à gauche). Les valeurs dans la fenêtre sont triées (ligne du bas) et l'élément médian (134) est utilisé comme nouvelle valeur sur le résultat (à droite).

Note

Les statistiques de l'INSEE sur les salaires en France présentent toujours des valeurs médianes et non des moyennes, sinon les très hauts revennus influenceraient fortement le résultat sans pour autant être représentatifs de la population.

Le filtre médian est particulièrement efficace pour réduire le bruit *poivre et sel* où les pixels sont aléatoirement transformés en pixels blancs ou noirs :

Exemple d'application du filtre médian pour réduire le bruit poivre et sel d'une image. A gauche : image originale. Au centre : image artificiellement bruitée par du bruit poivre et sel, 1/10ème des pixels ont été aléatoirement transformés en pixels noirs ou blancs. A droite : résultat d'application d'un filtre médian 3×3 sur l'image bruitée. On constate que l'image est bien débruitée. Le filtre médian introduit moins de flou que le filtre moyenneur.

Exercice 5 : Filtre médian

Implémentez le filtre médian dans la fonction `median` du fichier `tpMorphology.cpp`.
Pensez à valider votre implémantion avec la commande `test`.