TP 02 Filtres discrets et detection de contours

A.B. & H.M., ESTIN

Dans ce TP, il vous est demandé d'expérimenter différentes méthodes de détection de contours, en commençant par étudier et comparer des filtres discrets du premier et du second ordre. Se fera ensuite l'étude de suppression de non-maxima pour les filtres de premier ordre et l'évaluation du système élaboré. Enfin, il vous est demandé d'étudier l'effet du lissage en tant que pré-traitement pour la détection de contours.

Pour commencer : téléchargez les fichiers sources sur le dossier TP02 à partir de https://github.com/DSARU-CERIST/BTI. Dans le fichier bti_tp02.ipynb, se trouveront les bibliothèques nécessaires à l'accomplissement de ce TP. Seront remis les fichiers my_functions_tp02.py et bti_tp02.ipynb.

1 Filtres discrets et détection de contours

La détection de contours est une technique de traitement d'images utilisée pour identifier les points d'une image présentant des discontinuités, c'est-à-dire des variations brutales d'intensité. Ces points où l'intensité de l'image varie fortement sont appelés les contours (edges) de l'image. Le processus de détection des contours réduit de manière significative la quantité de données et élimine les informations inutiles, tout en préservant les importantes propriétés structurelles d'une image.

Il existe de nombreuses méthodes de détection de contours, dont la plupart peuvent être regroupées en deux catégories :

- La méthode du *Gradient* qui détecte les contours en utilisant le maximum et le minimum de la dérivée première de l'image.
- La méthode du *Laplacien* qui détecte les contours en utilisant les passages par zéro de la dérivée seconde de l'image.

Le filtre de *Sobel* est un filtre gradient-basé qui peut être utilisé pour détecter les contours d'une image $I(x, y)$. Le filtre est appliqué dans les directions x et y, et fonctionne en convoluant deux matrices S_x et S_y avec l'image originale, puis en prenant la magnitude du résultat $\sqrt{G_x^2 + G_y^2}$, de direction $\arctan(\frac{G_y}{G_x})$. Le filtre Sobel peut être appliqué aux images en niveaux de gris et en couleurs.

$$S_x = \begin{bmatrix} -1 & +0 & +1 \\ -2 & +0 & +2 \\ -1 & +0 & +1 \end{bmatrix} \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ +0 & +0 & +0 \\ +1 & +2 & +1 \end{bmatrix} \quad L = \begin{bmatrix} +0 & -1 & +0 \\ -1 & +4 & -1 \\ +0 & -1 & +0 \end{bmatrix} \tag{1}$$

Le filtre *Laplacien* est un filtre passe-haut qui rehausse les contours et autres détails d'une image. Il est basé sur la dérivée seconde spatiale de l'image. Les valeurs des pixels de sortie sont proportionnelles à la somme des dérivées secondes $(\frac{\partial^2 I}{x^2} + \frac{\partial^2 I}{y^2})$ dans les directions x et y. Cette somme peut être calculée efficacement à l'aide du filtre L.

N.B : Pour effectuer le travail ci-dessous vous aurez besoin des fonctions de **tools.py** et l'image à utiliser sera **flower.png**.

À faire :

- Tout d'abord essayer de se familiariser avec les fonctions de convolution et ses variantes. Lire l'image **flower.jpg** en niveau de gris via *cv2*, puis appliquer le filtre moyenner pour différentes valeurs de taille de filtre (3,7,11) avec padding nécessaire pour conserver la dimension initiale de l'image. Appliquer le stride et la dilatation sans padding et pour différentes valeurs (2,5,11) et (1,3,5) respectivement. Vérifier les dimensions des sorties, combiner quelques différentes variantes de convolution et afficher les images résultantes pour comparaison en utilisant le *subplots*.
- Automatiser le processus précédant. Créer une fonction pour la construction du filtre moyenneur à l'instar de la fonction *gaussian_mask*. Faire une fonction *filter_analysis* qui prend en entrée une image et la fonction du filtre souhaité et qui renvoi la figure des comparaisons.
- Refaire l'expérience précédente sur une image couleur.
- Construir les fonctions permettant de calculer : $G_x = I \circledast S_x$ et $G_y = I \circledast S_y$.
- La fonction permettant de calculer le module du gradient de l'image estimé de chaque pixel G_I et leur direction θ_I.
- A l'aide de la fonction *grad_orientation*, afficher les orientations ajustées sur les 8 directions, voir code suivant :

```
bins = 8
cmap = cmap_discretize('jet', bins+1)
ori_map = orientation_colors()
plt.imshow(ori_map, cmap=cmap, vmin=-1, vmax=bins-1)

plt.colorbar()
plt.axis('off')
plt.title("Orientations")
```

Listing 1: Affichage de la palette de 8 couleurs d'orientation.

- La fonction d'un filtre détecteur de contours du premier ordre en incluant seuillage au module du gradient de l'image calculé.
- La fonction permettant de calculer $L_I = I \circledast L$.
- La fonction déterminant les points de contours (les passages par 0 de L_I). Pour cela, il faut :
  1. Prendre une fenêtre centrée 3×3 sur chaque pixel (i, j), et calculer $\max(L_I)$ et $\min(L_I)$.
  2. Le passage par 0 sera détecté si $\max(L_I) > 0$, $\min(L_I) < 0$ et $\max(L_I) - \min(L_I) > Seuil$ (Le seuil étant déterminé *à priori*).
- Déterminez les différences notables entre les deux approches.

2 Influence du lissage et suppression de non maxima

Le lissage d'une image est un processus permettant de réduire le bruit et les détails d'une image. Il est réalisé en convoluant l'image originale avec un filtre gaussien pour produire une version floue. La fonction gaussienne est contrainte par le paramètre σ définissant l'écart-type de la distribution. La fonction est donné comme suit :

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2}$$

En python, cela peut être fait en utilisant la méthode *scipy.ndimage.gaussian_filter*. La fonction prend une image et l'écart type du filtre gaussien en entrée.

```
from scipy import ndimage
blurred_img = ndimage.gaussian_filter(img, sigma)
```

L'algorithme de suppression de non-maxima locaux (NMS) fonctionne itérativement sur tous les pixels d'une image, en supprimant (annulant) ceux qui ont une magnitude de gradient inférieure à celle de leurs voisins dans la direction du vecteur de gradient. Si nous désignons I comme notre image, ∇I comme son champ de gradient et $|\ .\ |$ comme la fonction de valeur absolue, alors pour chaque pixel (x_i, y_j), et ses voisins $(x_{i\pm m}, y_{j\pm n})$ dans la direction du vecteur de gradient, nous avons :

$$I_{NMS}(x_i, y_j) = \begin{cases} \nabla I(x_i, y_j), & Si \quad |\nabla I(x_i, y_j)| > |\nabla I(x_{i\pm m}, y_{j\pm n})| \\ 0, & Sinon. \end{cases} \tag{3}$$

À faire :

- Ecrire une fonction reproduisant le fonctionnement du NMS. En prenant en compte le fait que :
  1. La fonction prend en entrée la matrice des modules et directions de gradient de chaque pixel.
  2. Effectuer la conversion nécessaire des angles calculés.
  3. Quatre régions sont à définir $(0, \frac{\pi}{4}), (\frac{\pi}{4}, \frac{\pi}{2}), (\frac{\pi}{2}, \frac{3\pi}{4})$ et $(\frac{3\pi}{4}, \pi)$ pour l'évaluation des non-maxima.
  4. La fonction renvoie seulement l'image I_{NMS}
- Comparer les images G et I_{NMS}. Que remarquez-vous ? Déterminez l'effet du NMS sur les contours de l'image.
- Effectuez un lissage de l'image avant et après suppression des non-maxima tout en variant les valeurs de σ. Conclure sur les différentes approches tout en définissant l'influence de chaque paramètre sur les contours générés.