TP8 – Segmentation par *GrabCut*

La méthode de segmentation *GrabCut* présente des points communs avec la méthode de segmentation par champ de Markov du TP5 de TAV. Dans les deux cas, il s'agit de *classification supervisée*. La méthode *GrabCut* permet de segmenter en $N = 2$ parties (segmentation « fond-forme ») une image en niveaux de gris $x = (x_s)_{s\in\mathcal{S}}$, où $\mathcal{S}$ désigne l'ensemble des pixels. La raison pour laquelle elle est *supervisée* est que l'utilisateur doit initialiser le traitement par un détourage grossier de l'objet à segmenter.

Chacune des $N = 2$ classes est modélisée par un mélange de n lois normales (la valeur $n = 5$ est recommandée). L'estimation des paramètres d'un mélange de lois normales a déjà été effectuée, dans le TP5 de TIM, par la méthode itérative EM (Espérance-Maximisation). Dans ce TP, c'est une autre méthode itérative qui sera utilisée, moins précise mais plus rapide : la méthode des k**-moyennes** (*k-means*).

Une fois que les paramètres des $N = 2$ mélanges de lois normales ont été estimés, la vraisemblance de chaque pixel, relativement à chaque classe (fond ou forme), peut être calculée. Plutôt que d'affecter chaque pixel à la classe qui maximise sa vraisemblance, l'étiquetage des pixels est vu comme un champ de Markov $k = (k_s)_{s\in\mathcal{S}}$. Un a priori de type « lissage » pénalise les changements de classe entre pixels voisins (*modèle de Potts*). La configuration optimale $\widehat{k} = (\widehat{k_s})_{s\in\mathcal{S}}$ est celle qui maximise la probabilité a posteriori de la configuration k, sachant l'image x. Or, d'après le théorème de Bayes :

$$p(k|x) = \frac{p(x|k)\,p(k)}{p(x)} \propto p(x|k)\,p(k) \tag{1}$$

L'hypothèse d'indépendance des données permet d'écrire la vraisemblance sous forme de produit :

$$p(x|k) = \prod_{s\in\mathcal{S}} p(x_s|k_s) \tag{2}$$

Quant à la probabilité a priori de la configuration k, elle s'écrit (modèle de Potts) :

$$p(k) \propto \exp\left\{-\beta \sum_{\{s,t\}\in\mathcal{C}_2} [1 - \delta(k_s, k_t)]\right\} \tag{3}$$

où β est un « hyper-paramètre » du modèle, $\mathcal{C}_2$ désigne l'ensemble des cliques de cardinal 2, c'est-à-dire l'ensemble des paires $\{s, t\}$ de pixels voisins, et $\delta(k_s, k_t)$ vaut 1 si $k_s = k_t$ et 0 sinon (symbole de Kronecker).

Maximiser la probabilité a posteriori (1) revient à minimiser l'opposé de son logarithme, donc à résoudre le problème d'optimisation suivant :

$$\widehat{k} = \underset{k=(k_s)}{\arg\min} \left\{ -\sum_{s\in\mathcal{S}} \ln\left[p(x_s|k_s)\right] + \beta \sum_{\{s,t\}\in\mathcal{C}_2} [1 - \delta(k_s, k_t)] \right\} \tag{4}$$

Il est impensable de tester les $N^{\mathrm{card}(\mathcal{S})}$ configurations k possibles. Dans le TP5 de TAV, un problème d'optimisation similaire à (4) a été résolu par *recuit simulé*. Il existe une preuve de convergence du recuit simulé vers le minimum global, mais elle impose de faire décroître la température tellement lentement qu'il est quasi-impossible, en pratique, de pouvoir affirmer que le minimum global a bien été atteint. Or, il existe une autre méthode d'optimisation qui permet de garantir la convergence vers le minimum global : la **coupure de graphe** (*graph cut*). De plus, cette méthode est beaucoup plus rapide que le recuit simulé.

Exercice 1 : estimation des paramètres par les k-moyennes

La vraisemblance d'un pixel de niveau de gris x_s relativement à une classe k_s modélisée par un mélange de n lois normales s'écrit :

$$p(x_s|k_s) = \sum_{i=1}^{n} \frac{\pi_{k_s,i}}{\sigma_{k_s,i}\sqrt{2\pi}} \exp\left\{-\frac{(x_s - \mu_{k_s,i})^2}{2\,\sigma_{k_s,i}^2}\right\} \tag{5}$$

où $\pi_{k_s,i}$, $\mu_{k_s,i}$ et $\sigma_{k_s,i}^2$, $i \in [1,n]$, désignent la proportion, la moyenne et la variance de la $i^{\text{ème}}$ loi du mélange. Pour pouvoir calculer l'expression (5) de la vraisemblance, ce qui est nécessaire à la résolution du problème (4), il faut connaître les paramètres $(\pi_{k_s,i}, \mu_{k_s,i}, \sigma_{k_s,i}^2)$, $i \in [1,n]$. Un problème similaire a été rencontré dans le TP5 de TIM, où deux méthodes d'estimation avaient été testées : l'estimation par tirages aléatoires, qui est facile à mettre en œuvre, mais lente et peu précise ; la méthode EM, qui est bien plus précise, mais itérative, donc lente elle aussi. Une troisième possibilité, moins précise que EM mais plus rapide, est la méthode des **k-moyennes**.

Le script `exercice_1.m` est quasiment identique au script `exercice_1.m` du TP5 de TAV. Néanmoins, au lieu d'être fournis, les paramètres $(\pi_i, \mu_i, \sigma_i^2)$ des $n = 6$ lois normales du mélange décrivant les niveaux de gris de l'image sont estimés. Pour ce faire, la fonction `estimation` procède en deux temps :

- Les niveaux de gris sont d'abord partitionnés en $n = 6$ classes à l'aide de la fonction `kmeans` de Matlab (activez les options `'Emptyaction'` et `'Start'`, avec les valeurs respectives `'singleton'` et `'cluster'`).
- Les paramètres des $n = 6$ lois normales sont ensuite estimés, classe par classe.

Écrivez la fonction `estimation`, puis importez les fonctions `attache_donnees`, `regularisation` et `recuit_simule` que vous avez écrites pour le TP5 de TAV. Le script `exercice_1.m` doit vous permettre d'atteindre un pourcentage de bonnes classifications similaire à celui de l'exercice 1 du TP5 de TAV, mais sans qu'il soit nécessaire de connaître à l'avance les paramètres des $n = 6$ lois normales !

Exercice 2 : optimisation par coupure de graphe

Faites une copie du script `exercice_1.m`, de nom `exercice_2.m`, que vous modifierez de manière à ne plus effectuer la résolution du problème (4) par recuit simulé, mais par **coupure de graphe**. Pour cela, il vous faut faire trois appels successifs à la fonction `GraphCut` (lisez la documentation de cette fonction) :

- `[gch] = GraphCut('open',DataCost,SmoothnessCost);` permet de créer un graphe dont les nœuds sont les pixels de l'image. Le paramètre `DataCost`, qui correspond à l'attache aux données (l'opposé de la log-vraisemblance), a déjà été calculé. Le paramètre `SmoothnessCost`, qui correspond au lissage (c'est-à-dire à l'a priori), nécessite de créer une matrice décrivant les pénalisations du modèle de Potts.
- `[gch,labels] = GraphCut('expand',gch);` lance la minimisation par coupure de graphe. **Attention :** les valeurs du paramètre de sortie `labels` sont comprises entre 0 et $n-1$, et non entre 1 et n.
- `[gch] = GraphCut('close',gch);` est nécessaire pour terminer le traitement.

Non seulement le pourcentage de bonnes classifications est excellent, mais l'optimisation par coupure de graphe est beaucoup plus rapide que par recuit simulé (cela est dû, en partie, à l'utilisation de fichiers `mex`).

Exercice 3 : segmentation par *GrabCut*

Vous disposez de tous les éléments vous permettant d'écrire un script, de nom `exercice_3.m`, mettant en œuvre la méthode *GrabCut*. À la fois le fond et la forme sont modélisés par des mélanges de n lois normales. L'utilisateur doit sélectionner un rectangle englobant de la forme, c'est-à-dire contenant l'objet à segmenter. Cela permet d'initialiser la configuration k. La méthode *GrabCut* répète ensuite la boucle suivante :

1. Estimer les paramètres d'un mélange de n lois normales pour les pixels du fond et pour ceux de la forme.
2. Calculer l'attache aux données de tous les pixels de l'image, relativement aux deux classes, à l'aide d'une copie de la fonction `attache_donnees`, de nom `attache_donnees_n`, que vous modifierez de manière à retourner l'opposé du logarithme de l'expression (5) de la vraisemblance.
3. Trouver la solution $\widehat{k}$ du problème (4) par coupure de graphe.
4. Faire passer dans le fond tout pixel de la forme en lequel $\widehat{k} \neq k$ (« rognage » de la forme).

tant qu'un pixel au moins a été « rogné ». Testez ce script sur les images `avion.jpg`, `jardin.jpg`, `zebre.jpg`.