TD Segmentation – CETS8AH

12 avril 2024
Erwan Kerrien

Seule la section 2 est notée. Vous rendrez ce TD, à faire seul, pour le 23 avril sous format électronique : envoyez par mail à `Erwan.Kerrien@inria.fr` les M-fichiers que vous aurez écrits, ainsi qu'un texte de réponse (format Word, OpenOffice ou Latex, ou autre si pas trop exotique). N'hésitez pas à me poser des questions par mail.

La section 1 a pour objectif de vous faire implanter l'algorithme des K-means sur des images à niveaux de gris et de mieux comprendre comment l'histogramme peut être exploité. Il n'est pas noté car il ne présente par de grande difficulté, par ailleurs la classification par K-means est disponible sous Matlab via la fonction `kmeans` (étonnant, non ?)

La section 2 est notée. Elle a pour objectif de vous faire implanter un snake, dans sa formulation la plus simple, à savoir celle donnée par Kass, Witkins et Terzopoulos dans leur article "Snakes : Active contour models", International Journal of Computer Vision, pp. 321-331, 1988. Cette section aura aussi pour but de vous présenter de nouvelles fonctionnalités sous matlab, à savoir la définition interactive de points et la mise à jour dynamique d'une figure.

1 K-moyennes (Exercice non-noté)

Cette première section a pout but de vous faire implanter l'algorithme des K-means dans le cas particulier où les données (caractéristiques) peuvent se synthétiser sous la forme d'un histogramme. Plus précisément, nous considérons une image en niveaux de gris et la caractéristique est donc mono-dimensionnelle : c'est l'intensité de chaque pixel.

Rappel de l'algorithme :

1. initialiser des centres au hasard
2. associer chaque donnée au centre le plus proche
3. calculer la moyenne des données par centre
4. mettre à jour les centres avec cette moyenne
5. si pas convergence, recommencer en 2

De quoi a-t-on besoin ?

1. de l'histogramme d'une image en niveaux de gris. Vous pouvez le générer avec la commande Matlab `imhist`.
2. de générer des centres aléatoires. Comme les données sont des scalaires, il suffit d'utiliser la fonction `rand` qui renvoie un résultat entre 0 et 1 (exclus). Il suffit donc de multiplier par 256 pour avoir un résultat entre 0 et 256. Par exemple, la commande `rand(1,5)*256` génère 5 valeurs entre 0 et 256 (exclus).
3. d'associer chaque point à un centre. À nouveau, comme ce sont des scalaires, il suffit de calculer les milieux des intervalles. La figure 1 vous en donne le principe. D'un point de vue calcul, si `C` contient les centres, alors on trouve les bornes des intervalles par la formule `(C(1:end-1)+C(2:end))*0.5` : ces bornes sont les seuils à appliquer à l'image.
4. de calculer une valeur moyenne. Ce point est spécifique à l'exploitation d'un histogramme. Normalement les K-means considèrent les données, c'est-à-dire les intensités de pixel. Le nombre de "points" (données) est donc égal au nombre de pixel. Cela fait beaucoup et on peut exploiter l'histogramme pour réduire la complexité. En effet, imaginons qu'on veuille calculer la moyenne des intensités sur l'image. La méthode naïve consiste à passer en revue tous les pixels et en faire la moyenne :

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1}$$

Soit $N-1$ sommes et une division à faire, où N est le nombre de pixels. Imaginons à présent qu'on ait un histogramme de ces valeurs (par exemple si les x_i sont des valeurs discrètes), par exemple sur 256 niveaux de 0 à 255 (les seules intensités, donc valeurs des x_i qu'on peut rencontrer dans le jeu de données). Je peux reformuler ma moyenne sous la forme :

$$\bar{x} = \frac{1}{N}\sum_{k=0}^{255}\sum_{i:x_i=k} x_i \tag{2}$$

Soit

$$\bar{x} = \frac{1}{N}\sum_{k=0}^{255}\sum_{i:x_i=k} k = \frac{1}{N}\sum_{k=0}^{255} k \sum_{i:x_i=k} 1 \tag{3}$$

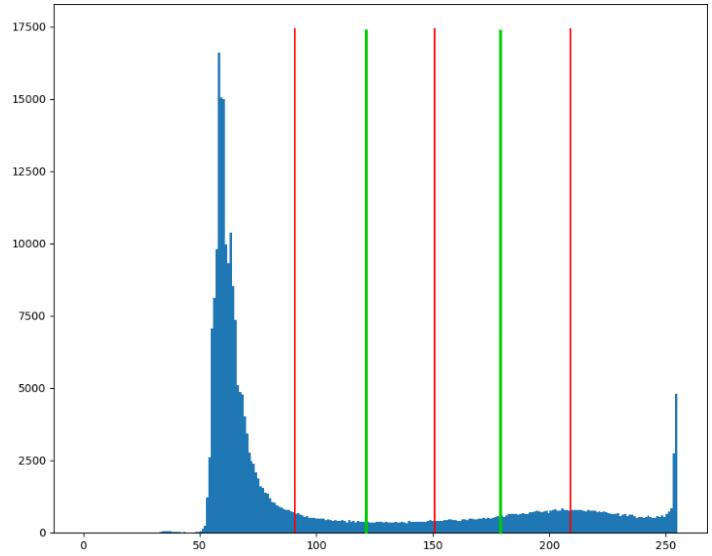

FIGURE 1 – Calcul des bornes des intervalles (en vert) permettant d'associer une valeur à un centre (en rouge). Celles-ci sont les valeurs moyennes entre deux centres successifs.

Or, l'histogramme $H(k)$ donne le nombre de données x_i qui valent k, c'est-à-dire :

$$H(k) = \sum_{i:x_i=k} 1 \tag{4}$$

Donc j'en déduis

$$\bar{x} = \frac{1}{N} \sum_{k=0}^{255} kH(k) \tag{5}$$

Et en plus je peux dire :

$$N = \sum_{k} H(k) \tag{6}$$

Donc

$$\bar{x} = \frac{\sum_{k=0}^{255} kH(k)}{\sum_{k=0}^{255} H(k)} \tag{7}$$

Reprenons à présent notre calcul de moyenne, mais uniquement sur un intervalle $k_{\min}, k_{\max}$. Cette moyenne restreinte vaut :

$$\bar{x}_r = \frac{\sum_{k=k_{\min}}^{k_{\max}} kH(k)}{\sum_{k=k_{\min}}^{k_{\max}} H(k)} \tag{8}$$

Question : à partir de ces éléments, implantez l'algorithme des K-means pour une image en niveaux de gris. On donnera comme paramètre le nombre de classes que l'on désire.

2 Ouroboros (Exercice noté)

L'ouroboros est un serpent qui se mord la queue, et en effet nous n'allons dans cet exercice que nous intéresser à des snakes fermés.

2.1 Quelques notations et rappels du cours

Nous allons noter $\mathcal{C}$ la courbe qui décrit le bord de l'objet. Nous faisons l'hypothèse que cette courbe est paramétrée par s, telle que chaque valeur de s donne un point en 2D $\mathcal{C}(s) = (x(s), y(s))$ qui appartient à la courbe $\mathcal{C}$. En faisant

varier s de 0 à 1, nous décrivons successivement tous les points de la courbe. La courbe est supposée fermée, c'est-à-dire $\mathcal{C}(0) = \mathcal{C}(1)$.

Avec cette notation, la dérivée $\mathcal{C}'(s) = \partial\mathcal{C}/\partial s$ est un vecteur tangent à $\mathcal{C}$ au point $\mathcal{C}(s)$, et la dérivée seconde $\mathcal{C}''(s) = \partial^2\mathcal{C}/\partial s^2$ est liée à la courbure en ce même point.

Le snake décrira bien les contours de l'objet présent dans l'image lorsque l'énergie suivante sera minimisée

$$E_{\text{snake}} = \frac{1}{2}\int_0^1 \alpha(s)|\mathcal{C}'(s)|^2 ds + \frac{1}{2}\int_0^1 \beta(s)|\mathcal{C}''(s)|^2 ds - \gamma\int_0^1 |\nabla I(\mathcal{C}(s))|^2 ds \tag{9}$$

2.2 Analyse du problème

L'objectif étant de minimiser cette énergie, nous pouvons adopter une approche par descente de gradient, les paramètres étant un peu particuliers puisque c'est la courbe $\mathcal{C}$. Mais si nous faisons momentanément abstraction de cette particularité, l'algorithme est simple :

1. Initialiser la courbe : celle-ci est donnée par une suite ordonnée de points 2D c_i
2. calculer le gradient de l'énergie E_{snake} en chaque point c_i, que je note $\delta E_i = \nabla_{\mathcal{C}} E_{\text{snake}}(c_i)$
3. mettre à jour chaque point c_i en lui soustrayant δE_i, multiplié par un pas λ : $c_i^{t+1} = c_i^t - \lambda\delta E_i$
4. si convergence (aucun c_i ne bouge de manière significative), ou si le nombre maximal d'itérations est atteint, alors arrêter. Sinon, recommencer en 2.

Le point délicat ici est de calculer le gradient de l'énergie. Ça n'est pas follement compliqué, et gérer un paramètre sous forme de courbe, est similaire à un paramètre sous forme de vecteur. Je passe les calculs et vous donne le résultat que vous accepterez pour le moment [1]. L'énergie E_{snake} se décompose en trois termes. Les deux premiers définissent l'énergie interne au snake. Si on suppose α et β constants, pour une courbe fermée, on obtient (se fait par intégration par parties, d'où le signe - pour la première) :

$$\frac{\partial}{\partial\mathcal{C}}\left(\frac{1}{2}\alpha|\mathcal{C}'|^2\right) = -\alpha\mathcal{C}'' \tag{10}$$

$$\frac{\partial}{\partial\mathcal{C}}\left(\frac{1}{2}\beta|\mathcal{C}''|^2\right) = \beta\mathcal{C}'''' \tag{11}$$

Pour calculer les dérivées en un point c_i, vous emploierez l'approximation discrète classique par différences finies :

$$\mathcal{C}''(c_i) = c_{i-1} - 2c_i + c_{i+1} \tag{12}$$

$$\mathcal{C}''''(c_i) = c_{i-2} - 4c_{i-1} + 6c_i - 4c_{i+1} + c_{i+2} \tag{13}$$

La dérivée de l'énergie externe, celle qui fait intervenir les données image, est plus complexe. Une manière simple consiste à passer par des approximations numériques pour :

1. calculer l'image de la norme du gradient $|\nabla I|^2$
2. calculer le gradient de cette image : une image pour la dérivée horizontale ($\partial I/\partial x$) et une image pour la dérivée verticale ($\partial I/\partial y$)

Le premier TP vous donne tous les éléments pour ce faire (exercice sur Sobel). Mais si on applique Sobel, nous aurons un problème de convergence car le noyau de Sobel est petit (3×3) et son influence ne s'étend donc pas au-delà de ce petit voisinage : le snake devra être initialisé dans ce voisinage si on veut que l'énergie externe ait une influence. Nous allons donc ici suivre une autre manière de faire et utiliser un filtrage gaussien. En effet, il est aisé de prouver l'égalité suivante :

$$\nabla\left(G_\sigma * I\right) = \left(\nabla G_\sigma\right) * I \tag{14}$$

où $*$ est la convolution et G_σ est une gaussienne d'écart-type σ. On peut donc calculer le gradient d'une image, rendue continue par interpolation gaussienne, en calculant la convolution avec la dérivée d'une gaussienne. La "largeur" du gradient peut alors se moduler par σ : plus cette valeur sera grande, plus le signal de gradient sera étalé (mais aussi plus flou et donc moins précis).

Et il se trouve que le gradient d'une gaussienne est très facile à calculer. Commençons par rappeler la formule de la gaussienne en 2D (nous travaillons sur l'image !).

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{15}$$

1. `https://mat-web.upc.edu/people/toni.susin/files/SnakesAivru86c.pdf` donne tous les détails du calcul

Nous avons alors

$$\frac{\partial}{\partial x} G_\sigma(x,y) = -\frac{x}{\sigma^2} G_\sigma(x,y) \tag{16}$$

$$\frac{\partial}{\partial y} G_\sigma(x,y) = -\frac{y}{\sigma^2} G_\sigma(x,y) \tag{17}$$

2.3 Implémentation sous Matlab

Les deux calculs au cœur de l'algorithme sont donc d'une part le calcul des dérivées de la courbe (équations 12 et 13) et le calcul du filtre de dérivées de gaussienne (équations 16 et 17).

La courbe sera représentée par un ensemble de N points 2D sous la forme d'une matrice `C` de taille $2 \times N$. Vous pouvez l'initialiser par des points régulièrement disposés sur un cercle, et la section suivante vous explique comment le faire de manière interactive sous matlab.

Pour calculer les dérivées de la courbe, il ne faut pas oublier que la courbe est fermée. Les équations 12 et 13 font intervenir des indices de -1 jusqu'à $N+2$. Une astuce est donc d'étendre la matrice `C` en lui ajoutant deux colonnes au début (dans lesquelles on copiera les deux derniers points) et deux colonnes à la fin (dans lesquelles on copiera les deux premiers points). Ceci se fait par la commande (`eC` est la matrice "étendue") :

```
>> eC=[C(:,end-1:end),C,C(:,1:2)]
```

On peut alors utiliser avec discernement les opérations d'extraction de coupes dans les matrices (avec `:` dans les indices) pour implémenter les équations 12 et 13.

Pour calculer la dérivée de l'énergie externe, l'opération de base est de pouvoir convoluer l'image avec la dérivée d'une gaussienne. On pourra utiliser le code suivant obtenir les deux noyaux (`HSIZE` est la demi-taille du noyau, qui aura donc une taille `2*HSIZE+1`, et `SIGMA` est l'écart-type du noyau gaussien) :

```
function [Gx,Gy] = DerivGauss(HSIZE, SIGMA)
   [x,y] = meshgrid(-HSIZE:HSIZE,-HSIZE:HSIZE);
   G=fspecial('gaussian',2*HSIZE+1,SIGMA);
   Gx = - x/(SIGMA*SIGMA) .* G;
   Gy = - y/(SIGMA*SIGMA) .* G;
end
```

Le bout de code suivant charge une image `I`, et en obtient les dérivées horizontale `Ix` et verticale `Iy`, calculées au moyen d'une gaussienne d'écart-type 1 :

```
>> I = imread('motifs.jpg');
>> [Gx,Gy] = DerivGauss(3,1);
>> Ix = imfilter(double(I),Gx,'conv');
>> Iy = imfilter(double(I),Gy,'conv');
>> subplot(1,3,1), imshow(I);
>> subplot(1,3,2), imshow(Ix,[]);
>> subplot(1,3,3), imshow(Iy,[]);
```

Quelques remarques sur ce code :

1. il faut convertir l'image en double (`double(I)`) avant de faire le filtrage, sinon le résultat est incorrect. En effet, `imfilter` produit en résultat une image qui est du même type que l'image d'entrée. Comme l'image renvoyée par `imread` est en `uint8`, on ne pourra stocker toutes les valeurs des gradients horizontal et vertical.
2. `imfilter` fait du filtrage par corrélation. Or nous avons ici des filtres qui sont asymétriques (plus précisément ils sont antisymétriques). Il faut donc soit leur appliquer une symétrie centrale (ce qui se fait par un simple changement de signe dans ce cas précis), soit spécifier à `imfilter` qu'on désire faire du filtrage par convolution. C'est ce que nous avons choisi de faire ici (option `'conv'`).
3. comme les images `Ix` et `Iy` sont stockées sur des doubles, il faut faire attention à bien normaliser leur visualisation. Cela se fait par un tableau vide `[]` passé à `imshow`, qui applique automatiquement cette normalisation entre les min et max de l'image.

2.4 Question notée

Une seule question : implémentez l'algorithme de snake décrit ci-dessus, testez-le et faites-en un commentaire (ce qui marche, ce qui ne marche pas, les difficultés d'initialisation et de paramétrage). Vous aurez besoin d'extraire, à des positions pixel non entières (les c_i) des valeurs dans des images (gradient horizontal de la norme du gradient de l'image et gradient vertical de la norme du gradient de l'image). Pour ce faire, vous pourrez utiliser la fonction Matlab `interp2` (interpolation linéaire). Un ordre d'idée de paramètres qui fonctionnent (assez) bien sur l'image `motifs.jpg` est : $\alpha = 0.01, \beta = 0.001, \gamma = 0.1$ avec `HSIZE` $= 5$ et `SIGMA`$=4$ (mais le résultat est très lissé).

2.5 Fonctionnalités utiles sous Matlab

Pour voir comment votre courbe évolue, il peut être intéressant de non pas tout redessiner sur la figure, mais de simplement la mettre à jour. La commande `drawnow` vous permet de faire cela [2].

Pour initialiser les points de votre courbe, vous pouvez par exemple les définir sur un cercle régulièrement échantillonné. Mais une manière plus sympa de faire est de placer les points à la main. La commande Matlab `ginput` permet de faire cela.

Par exemple, ce bout de code permet rentrer une courbe à la main, et de voir le résultat mis à jour en temps réel :

```
function C=pickContour()
    hold on;
    [x,y,b] = ginput(1);
    if b ~= 1
     hold off
        return
    end
    C=[x;y]
    h=plot(C(1,:),C(2,:),'r-');
    g=plot(C(1,:),C(2,:),'ro');
    while b==1
        [x,y,b] = ginput(1);
        if b == 1
            C=[C [x;y]];
            set(h,'XData',[C(1,:) C(1,1)]);
            set(h,'YData',[C(2,:) C(2,1)]);
            set(g,'XData',[C(1,:) C(1,1)]);
            set(g,'YData',[C(2,:) C(2,1)]);
            drawnow;
        end
    end
    hold off;
end
```

Vous pouvez l'appeler par

```
>> I = imread('motifs.jpg');
>> imshow(I);
>> C = pickContour();
```

Les points se définissent par un clic gauche et vous terminez par un clic droit (ou tout sauf gauche). Vous remarquerez que l'ajout de `C(2,1)` et `C(1,1)` dans les commandes `set` permet de fermer le dessin du contour.

Attention : un snake demande d'avoir une bonne densité de points, et que ces points soient régulièrement espacés sur le contour. Par conséquent, prenez soin, même si c'est pénible, de rentrer beaucoup de points. Et pensez à sauvegarder votre ensemble de points durement acquis (revoir le premier TP).

Ou bien, écrivez un bout de code qui rééchantillonne uniformément une courbe polygonale sur N points...

2. Cette commande est inutile dans certaines versions de Matlab