TD : Clustering d'images

1 Application de KMeans à des images

Le but de ce TP est d'implémenter l'algorithme KMeans et de l'appliquer aux pixels d'une image afin de regrouper automatiquement les pixels ayant des couleurs proches. Chaque pixel est composé de trois composantes : rouge, vert et bleue comprises entre 0 et 255. Votre programme prendra en entrée une image et donnera en sortie une image où tous les pixels appartenant au même cluster (au même groupe) auront la même couleur (par exemple la couleur moyenne du cluster). Le nombre de clusters sera donné en paramètre. Votre programme pourra également fournir en sortie une image pour chacun des clusters trouvés.

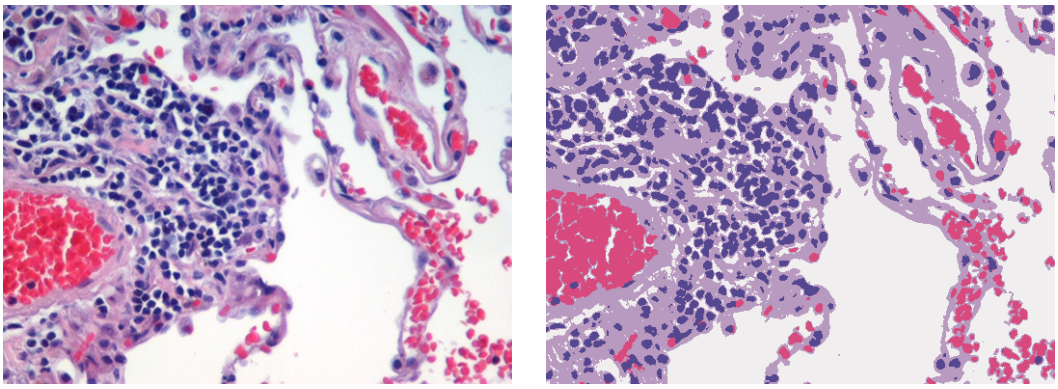

FIGURE 1 – Image d'entrée (gauche), image de sortie (droite)

L'image de test est disponible ici : `https://en.wikipedia.org/wiki/Staining#/media/File:Emphysema_H_and_E.jpg`, mais n'importe quelle image peut être utilisée.

Afin de lire une image, nous allons utiliser la librairie Pillow de Python :

```
from PIL import Image
im = Image.open('Emphysema_H_and_E.jpg')
display(im)
```

Afin de manipuler les données, nous allons utiliser un array Numpy :

```
import numpy as np
rgb = np.array(im.convert('RGB').getdata())
```

2 Classification automatique d'une librairie d'images

Nous allons maintenant utiliser Kmeans afin de faire du clustering de plusieurs images. Il est donc nécessaire de définir une distance entre les images. La distance la plus simple (mais pas très performante) est d'utiliser l'histogramme de couleurs des images.

Nous allons utiliser les données CIFAR10 (`https://www.cs.toronto.edu/~kriz/cifar.html`) disponible dans le package tensorflow.

Dans un premier temps n'utilisez que quelques images de 3 classes afin de développer votre code. Comme il est difficile de calculer une "moyenne" d'images, on utilisera ici le médoïde comme représentant d'un cluster, c'est-à-dire l'objet qui minimise la distance aux autres objets du groupe.

3 Pour aller plus loin

— Coder une méthode afin de garder le meilleur résultat de plusieurs initialisations différentes de l'algorithme KMeans
— Coder une méthode afin de guider le choix du nombre de clusters pour une image donnée
— Implémenter l'algorithme EM vu en cours. Comparer les résultats obtenus pour le clustering d'images entre KMeans e EM.
— Chercher comment définir une distance entre deux modèles composés de lois de probabilités.
— Chercher d'autres données complexes sur lesquelles appliquer vos algorithmes

4 Annexes

Algorithm 1: K-Means Algorithm

Input: $E = \{e_1, e_2, \ldots, e_n\}$ (set of entities to be clustered)
k (number of clusters)
$MaxIters$ (limit of iterations)
Output: $C = \{c_1, c_2, \ldots, c_k\}$ (set of cluster centroids)
$L = \{l(e) \mid e = 1, 2, \ldots, n\}$ (set of cluster labels of E)

foreach $c_i \in C$ **do**
  $c_i \leftarrow e_j \in E$ (e.g. random selection)
end
foreach $e_i \in E$ **do**
  $l(e_i) \leftarrow argminDistance(e_i, c_j) j \in \{1 \ldots k\}$
end

$changed \leftarrow false$;
$iter \leftarrow 0$;
repeat
  foreach $c_i \in C$ **do**
    $UpdateCluster(c_i)$;
  end
  foreach $e_i \in E$ **do**
    $minDist \leftarrow argminDistance(e_i, c_j)\ j \in \{1 \ldots k\}$;
    if $minDist \neq l(e_i)$ **then**
      $l(e_i) \leftarrow minDist$;
      $changed \leftarrow true$;
    end
  end
  $iter++$;
until $changed = true\ and\ iter \leq MaxIters$;

FIGURE 2 – L'algorithme KMeans