Image Segmentation using Min s-t Cut and Multiway Cut

Francesco Sgherzi

November 2019

Contents

1 Introduction

Image Segmentation is the process of *partition* an image into *relevant* sets, where each element in a given set is characterized by similar values of a particular metric.

Traditionally, *Image Segmentation* has found numerous application in medical imaging (cancer and other *foreign bodies* detection), entity detection (face detection, edge detection) and also as first stage to more sophisticated recognition algorithms. Due to this research has been driven to push toward more efficient and *portable* implementations of the algorithms solving *Image Segmentation*.

1.1 Problem Definition

In this project, we will explore *Graph based* techniques in solving the *Image Segmentation* problem. In its simplest form, *Image Segmentation* is the process of distinguishing foreground from background, that is, given an N by M digital image, output an image of the same dimensions where each pixel is black (white) if the original pixel belonged to background, white (black) otherwise.

Other, more sophisticated, forms of *Image Segmentation* aim to distinguish different *regions* of the image, therefore grouping pixel in sections of similar features (e.g. color) rather than assigning them to only two sets.

Note that, due to the strong correlation between a matrix representation (e.g. adjacency) and a graph representation, the aforementioned problems can be modeled as graph partition problems. More specifically: *background-foreground distinction* can be modeled as an instance of the *Min Cut* problem and *region distinction* as an instance of the *Multiway Cut* problem.

2 Traditional approaches

2.1 Thresholding

The *Thresholding* approach is easily one of the most common and simple method of tackling this problem. The general idea is to use a certain threshold value (e.g. 50% black) and assign a pixel to foreground (background) if its grayscale value is above (below) the threshold. However, it is easy to see the naiveness of this approach leads to a non trivial amount of false positives. For instance, take an image with just two colors and suppose that each color occupies 50% of the image. If the greyscale representation for the two color is above (below) the threshold the algorithm would assign both region to the background (foreground). A more refined version of this approach would set the threshold *dynamically*, for instance by averaging the most common colors in the image.

2.2 Convolution kernel

Traditionally used for edge detection, the *Convolution kernel* approach can be applied to *Image Segmentation* if postprocessing of the resulting image is an option. When the countours of a figure in an image are detected, those can be used as boundaries to distinguish foreground and background. While this is a viable and portable approach due to the fact that the convolution operation involves simple vector operations and can be easily parallelized and therefore accelerated on commodity hardware like phones GPUs the postprocessing required to transform the problem from *Edge Detection* to *Image Segmentation* can become fairly intensive.

2.3 Clustering

The *Clustering* approach is the most classic technique for *Image Segmentation* due to the easiness of conversion between two problems (in fact, image segmentation can be seen as a *Clustering* problem). While every *Clustering* algorithm acheives similar results the most used is certainly *K-Means*, since it fairly low on resource utilization and all the common shortcomings that *K-Means* encounters are generally not applicable. In general, *K-Means* requires prior knowledge of the number of clusters and it is sensible to the initialization of the cluster value. By applying it to *Image Segmentation*, the number of clusters is fixed and known *a priori* (in the most general case $k = 2$) and this greately reduces sensibility to the initial state. However, *K-Means* struggles greatly in recognizing *non-globular* shapes, which implies that, in order to get results on more complex images, one should first run the algorithm with a *high* number of clusters, and then postprocess (merge) them. While there are *Clustering* algorithms that don't encounter such problems (like *DBSCAN*) those are more complex and require a higher resource utilization.

3 Algorithms overview

In this section we will explore the theoretical background required for the application of the aforementioned problems to *Image Segmentation*.

3.1 Min s-t Cut

Let $G = (V, E)$ be a graph and $s, t \in V$ with $s \neq t$. Let c be the capacity associated to every edge such that $\forall e \in E : c(e) \geq 0$. Find a cut $(X, V \backslash X)$ in the graph such that s and t are disconnected and the capacity of the cut (the sum of the capacity of all edges crossing the two vertex sets) is minimized.

3.2 Multiway Cut

Let $G = (V, E)$ be a graph and let $S = \{s_1, s_2, ...s_k\}$ be a set of k vertices. Let c be the capacity associated to every edge such that $\forall e \in E : c(e) \geq 0$. Remove egdes to separate every pair of nodes $s_i, s_j : i \neq j$ while minimizing the sum of capacities of the cut edges.

Note that this problem is NP-Hard for $k \geq 3$. Only approximate algorithms achieve a polinomial time. In the implementation a $(2 - \frac{2}{k})$-approximation has been used.

4 From the image to the graph

The conversion from image to the graph is quite straightforward. Given a digital image, we represent it as a N by M array of 3-dimensional vectors (being the *Red*, *Green*, *Blue* values), we then create a vertex for each pixel in the image and an edge between every *horizontally* and *vertically* adjacent vertices of some capacity.

We are now going to discuss the metrics used to compute the capacity of the edges.

4.1 Euclidean Distance

Given two points, $x, y \in [0...255]^3$ the *Euclidean Distance* is defined as:

$$d_e = \sqrt{(x_r - y_r)^2 + (x_g - y_g)^2 + (x_b - y_b)^2}$$

4.2 "Brightness" Distance

Since the human eye does not perceive the three main colors equally the following *distance* has been developed to better represent changes in luminance in certain colors (weighting more, for instance, green instead of blue).

$$r = \frac{x_r + y_r}{2}$$

$$d_l = \sqrt{(2 + \frac{r}{256}) * (x_r - y_r)^2 + 4 * (x_g - y_g)^2 + (2 + \frac{255 - r}{256}) * (x_b - y_b)^2}$$

Distances are then converted to similarities and normalized to represent probabilities.

In order to properly represent the problem as a *Min s-t Cut* or a *Multiway-cut* problem it is mandatory to add the characteristic *source* and *sink* vertices (or k terminal vertices in the case of *Multiway-cut*) to the representation.

Those vertices are then connected to every pixel vertex in the graph by a capacity representing the *probability* of that vertex to belong to a given cluster (being it a color cluster in the case of *Multiway-cut* or the foreground or background cluster in the case of *Min s-t Cut*).

5 Experimental results

The test set consists of 5 sample images taken from the Berkley Image Segmentation Dataset

All tests were running against the `OpenCV` version of *KMeans*, with a number of clusters properly adjusted to reflect the number of colors that would have resulted from the application of *Min s-t cut* and *Multiway Cut* on the image graph ($k = 2$ and $k = 4$ respectively).

After that, a (percentage) mean square error and a (percentage) mean distance evaluation was performed between the two images.

A side note. In order to *help* the algorithm, in the `clustered` (`k-cuts`) method the source and sink node (the terminal nodes) are given color values according to the most common colors. On the other hand, the `min-cut` method uses *black* and *white* as foreground and background colors.

The results follow:

image	technique	similarity	percentage mean distance	mean square error
66075	clustered	euclidean sim	17.275	3.639
66075	clustered	luminance sim	19.634	4.193
lenna	clustered	euclidean sim	27.059	7.986
lenna	clustered	luminance sim	26.350	7.770
310007	clustered	euclidean sim	18.607	4.085
310007	clustered	luminance sim	18.294	3.983
317080	clustered	euclidean sim	35.499	14.087
317080	clustered	luminance sim	36.210	14.419
197017	clustered	euclidean sim	46.813	22.175
197017	clustered	luminance sim	46.814	22.175
66075	k-cuts	euclidean sim	4.584	0.513
66075	k-cuts	luminance sim	4.476	0.488
lenna	k-cuts	euclidean sim	8.683	1.220
lenna	k-cuts	luminance sim	7.516	0.993
310007	k-cuts	euclidean sim	4.687	0.550
310007	k-cuts	luminance sim	4.754	0.565
317080	k-cuts	euclidean sim	6.425	0.991
317080	k-cuts	luminance sim	6.091	0.976
197017	k-cuts	euclidean sim	6.519	0.831
197017	k-cuts	luminance sim	6.530	0.772
66075	min-cut	euclidean sim	48.341	48.341
66075	min-cut	luminance sim	48.314	48.314
lenna	min-cut	euclidean sim	39.703	39.703
lenna	min-cut	luminance sim	39.284	39.284
310007	min-cut	euclidean sim	24.303	24.303
310007	min-cut	luminance sim	22.460	22.460
317080	min-cut	euclidean sim	27.472	27.472
317080	min-cut	luminance sim	46.096	46.096
197017	min-cut	euclidean sim	40.317	40.317
197017	min-cut	luminance sim	39.503	39.503