TP 6: Texture Segmentation using Graph cuts and Gabor Filters

Ahmed Besbes *Rola Harmouche*

April 16, 2009

1 Introduction

In the previous lab, textured images were segmented using k-means clustering and Gabor features. In this lab, we will be using the same Gabor features and a Graph cuts algorithm to perform the same type of segmentation, with some improvements.

2 Graph cuts for texture segmentation

Graph cuts are used for labeling by energy minimization. They are used for several computer vision applications that can be formulated as labeling problems. This is also the case for texture segmentation, where we wish to label each image pixel as belonging to some texture class C.

An $s-t$ graph is composed of terminal nodes and non-terminal or internal nodes. The terminal nodes consist of a source node (s) and a sink node (t). In our case, nodes s and t will each represent a different texture class. The image pixels are represented by non-terminal nodes or internal nodes. Terminal nodes are connected to non-terminal nodes via edges called t-links, and non-terminal nodes are connected to their neighbours via edges called n-links given a neighbourhood system. Each edge $< p, q >$ between two nodes p and q has a non-negative capacity $cap(p, q)$ representing the amount of possible directed flow through that edge. Equivalently, these edge capacities or weights can be perceived as the cost of cutting the corresponding edge. Minimizing the global energy consists of finding the minimum cut of the graph (which has been shown in class to be equivalent to finding the maximum flow over the graph).

In our case, we define the t-link capacities (cap_t) and the n-link capacities (cap_n) in the following manner:

$$\begin{aligned} cap_t &= -log(p(I|C)) \ , \\ cap_n &= \beta \ , \end{aligned} \tag{1}$$

where I is the feature vector, C is the texture label, $p(I|C)$ is the probability distribution of the feature vector I for a given label C, and β is a constant value. This probability is defined using a normal distribution:

$$p(I|C) = \frac{1}{(2\pi)^{(N/2)}|\Sigma|^{(1/2)}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)) \ , \tag{2}$$

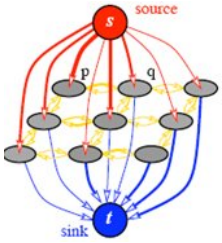

Figure 1: Example of a graph G with 4-connected neighbourhoods between pixels. Source: Boykov et al.

where N is the dimension of the feature vector, and μ and Σ are the means and covariances of the class distributions, respectively. After minimization, the resulting segmentation should be smoother than in the case of k-means clustering due to the use of neighbourhood information. In order to segment the textures using graph cuts, these steps have to be followed:

- Extract the features from the image.
- Compute the statistics of both classes, using the extracted features. The k-means result (provided as the "mask.pgm" image) is used to define the initial classes. To obtain this segmentation, 4 directions, 2 frequency levels and an initial sigma value of 5 were used as parameters of the Gabor filters.
- Define the graph.
- Create the non-terminal nodes (equal to the number of pixels in the image that need to be segmented).
- Create the t-links and the n-links and define their capacities.
- Apply the maximum flow / minimmum cut algorithm.The output of the algorithm is a labeling for each pixel to "1" or "0" depending on whether it belongs to the source or the sink.

3 Work to be submitted

The code provided to you includes a "tp6.cpp" file containing the main function. It also uses the Max-Flow code written by Vladimir Kolmogorov (see "graph.h" and the readme file for details). For this work, you should complete the code provided to you in order to segment the test image "twoTextures.pgm". To this end, you should complete the following sections denoted by `\\TODO` in the main function:

- Write the code that stores the feature vectors in the CImgLists `allFeaturesObject1` and `allFeaturesObject2`, for the pixels belonging respectively to class 1 and class 2.
- Using the previously stored features, compute the mean vectors `fMean1` and `fMean2` corresponding to μ_1 and μ_2 and the covariance matrices `fCov1` and `fCov2` corresponding to Σ_1 and Σ_2.

Figure 2: From left to right: (a) the texture image to be segmented (b) the segmentation obtained using k-means (c) example of a result using graph cuts (d) example of a result using graph cuts with non homogenous n-links capacities.

- Complete the part of the code witch adds the t-links and calculates their capacities `capsource` and `capsink`. Recall that the capacity of an edge is the cost of cutting the edge.
- **Bonus** In the code, the n-links are assumed to be homogeneous, i.e. constant throughout the image and equal to β (`beta` in the code). Modify the n-links in a way to obtain a result similar to Figure 2(d). Notice that a part of the image is smoothly segmented (similar to Figure 2(c)) and that the other part has a segmentation similar to that in Figure 2(b).
- **Bonus** The code available runs for only one graph-cut iteration. Modify the code in order to make it run for several iterations. The class mask and statistics should be updated in the current iteration using the resulting segmentation of the previous one.

You should also submit a `.pdf` report containing the segmentation results. You should show the following:

- The initial segmentation using only k-means.
- Your best segmentation result using graph cuts.
- Segmentations using 4-connected and 8-connected neighbourhoods.
- Segmentations using different n-link capacities.

Comment on how changing the graph connectivity and capacities affect the segmentation results.

Bonus: Comment on how varying the number of iterations for the algorithm affects the segmentation results.

References

[1] Yuri Boykov and Olga Veksler: "Graph Cuts in Vision and Graphics: Theories and Applications" . In "Math. Models of C.Vision: The Handbook", edts. Paragios, Chen, Faugeras available: http://luthuli.cs.uiuc.edu/ daf/courses/Optimization/Combinatorialpapers/ Nikos-Yuri-Olga.pdf

[2] The code for calculating the maxflow was taken from the following site: http://www.cs.ucl.ac.uk/staff/V.Kolmogorov/software.html