Image segmentation using GrabCut

Tomas Malmer
tf06tm0@student.lth.se

May 12, 2010

1 Abstract

GrabCut is a way to perform 2D segmentation in an image that is very user friendly. The user only need to input the a very rough segmentation between foreground and background. Typically this is down by drawing a rectangle around the object of interest.

The way that this is accomplish technically is by using a combination of Graph Cuts and statistical models of the foreground and background structure in the color space. The GrabCut algorithm has also been implemented professional software like the newest version of Microsoft Office.

2 Introduction

Graph Cut is an algorithm that have recently gained popularity for use in segmentation tasks in computer vision. During this project I had the opportunity to experiment with it further in order to understand it's use better and combined it with statistical models. The result is a GrabCut implementation in Matlab that's using Graph Cuts techniques as a part of its refining process of an initial user segmentation between foreground and background.

GrabCut is an iterative algorithm that combines statistics and Graph Cut in order to accomplish detailed 2D segmentation with very limited input, originally developed at Microsoft Research Cambridge, UK. The work flow for the user goes as follows: The user selects a image which should be used to perform the segmentation on and draws a rectangle around the object of interest that should be segmented. That's all input that is needed by the GrabCut algorithm to perform a segmentation. Afterwards the user can do a final touch up of the image if needed or use the image again input by selecting a new rectangle.

3 GrabCut algorithm overview

The basic steps for the GrabCut algorithm is as follows. Each step will be explained in more details later on.

1. The user input three things: The foreground, background, and the unknown part of the image that can be either foreground or background. This is normally done by selecting the a rectangle around the object of interest and mark the region inside that rectangle as unknown. Pixel outside this rectangle will then be marked as known background.

2. The computer creates an initial image segmentation, where the unknown pixels are placed in the foreground class and all known background pixels are classified as background.

3. The foreground and background are modeled as Gaussian Mixture Models (GMMs) using the Orchard-Bouman clustering algorithm.

4. Every pixel in the foreground assigned most probable Gaussian component in the foreground GMMs. The same process id done with the pixels in the background but with components of the background GMMs.

5. New GMMs are learned from the pixel sets that where created in the previous step.

6. A graph is built and Graph Cut is used to find a new classification of foreground and background pixels.

7. Repeat step 4-6 until the classification converges

4 Modeling foreground and background

The initial information given about the foreground and the background are given by the user as a rectangular selection around the object of interest. Pixels outside this selection are treated as known background and the pixels inside are marked as unknown. From this data we want to create a model that we can use to determine if the unknown pixels are either foreground or background.

In the GrabCut algorithm this is done by creating K components of multivariate Gaussian Mixture Models (GMM) for the two regions. K components for the known background and K components for the region that could be the foreground, giving a total $2K$ components.

The GMM components has the same dimensions as the color space and are derived from the color statistics in each cluster. In order to get good segmentation we want to find components with low variance since this makes the cluster easier to separate from the others. There are a lot of of proposed ways to create clusters with this property, I decided to test the color quantization technique described by Orchard and Bouman that were suggested in Implementing GrabCut by Justin F. Talbot and Xiaoqian Xu [1] which works well.

4.1 Color clustering

In order to to calculate the GMMs the pixels need to be clustered somehow in order to determine the statistics. This was done by using a binary tree quantization algorithm described by Orchard and Bouman. All pixels are placed in the same cluster in the beginning. The cluster is then slitted around it's mean values projection on the first principal component.

The parameters to the clustering algorithm is the number of clusters, K, to use and the first cluster, C_1.

1. Initialize first cluster, $C_1 = Unknown \cup Foreground$
2. Calculate the mean, μ_1, and the covariance matrix Σ_1 of the cluster C_1
3. For $i = 2$ to K do
4. Find the cluster, C_n, with the largest eigenvalue and its associated eigenvector e_n
5. Split C_n in two sets along the mean values projection on the eigenvector, $C_i = \{x \in C_n : e_n^T z_n \leq e_n^T \mu_n\}$ and update the original cluster with the other half $C_n^* = C_n - C_i$
6. Compute μ_n^*, Σ_n^*, μ_i and Σ_i

5 Graph Cut

Graph cuts can be used to efficiently solve a wide area of low-level problems in computer vision, such as image smoothing and many other problems that can be formulated in terms of energy minimization. The energy minimization problem can then be formulated as maximum flow problem in a graph.

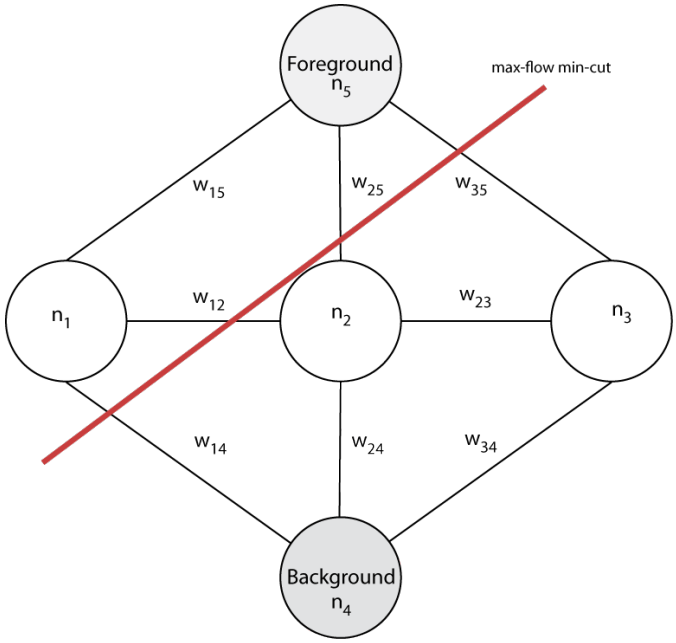


Figure 1: Example of a graph to segment to foreground and background

The graph consists of nodes n_i that has links between them with weights , w_{ij}. The weights hat describes how hard they are connected or related to each other. The min-cut between two nodes are a way to separate the graph in two distinct parts with minimal effort by minimizing $\sum_{i \in I} w_i$ where I is the set of links between the nodes that were cut off. In the example in figure 1, n_1 would be classified as foreground and all the other nodes as background.

I didn't implement my of version of Graph Cuts since it would be time consuming and its already done by others. Vladimir Kolmogorov (`http://www.cs.cornell.edu/people/vnk/software.html`) has written a good implementation in C that has a MATLAB wrapper around it and works perfectly well.

5.1 Usage in GrabCut

The graph consists of two parts. The first part describes how much each pixel, m and n, is connected to it's neighborhood, the N-links. How well a pixel is connected to it's neighborhood is calculated using (1). How the pixels are connected to each other are not changed during the iterations and therefore it's only needed to be constructed once.

$$N(m,n) = \frac{\gamma}{dist(m,n)} e^{-\beta||z_m - z_n||^2} \tag{1}$$

The second part of the graph consists of links to the foreground and the background note, the source and the sink of cut. The weight of these links is the "labeling cost", $L(m)$ for labeling a pixel m as foreground or background. (These links are marked with black node weights in the previous example on graphs)

The scheme to construct the weights to this part of the graph is found in the following table:

Pixel type	Background T-link	Foreground T-link
$m \in foreground$	0	$L(m)$
$m \in background$	$L(m)$	0
$m \in unknown$	$D_{fore}(m)$	$D_{back}(m)$

If the labeling cost exceeds the maximum sum of weight that a pixel can have to it's neighborhood the pixel will be forced to labeled as the chosen class. More specificity of weights for the pixel m is chosen in the following way

$$L(m) > \sum N(m,n)$$

you can force a pixel to be a part of either the foreground or background. Since I'm using a 8-neighborhood I can use

$$L(m) = 8\gamma + 1$$

The values D_{fore} and D_{back} are function of the likelihood that the pixel m belongs to the foreground and background GMMs respectively. They can be computed as follows:

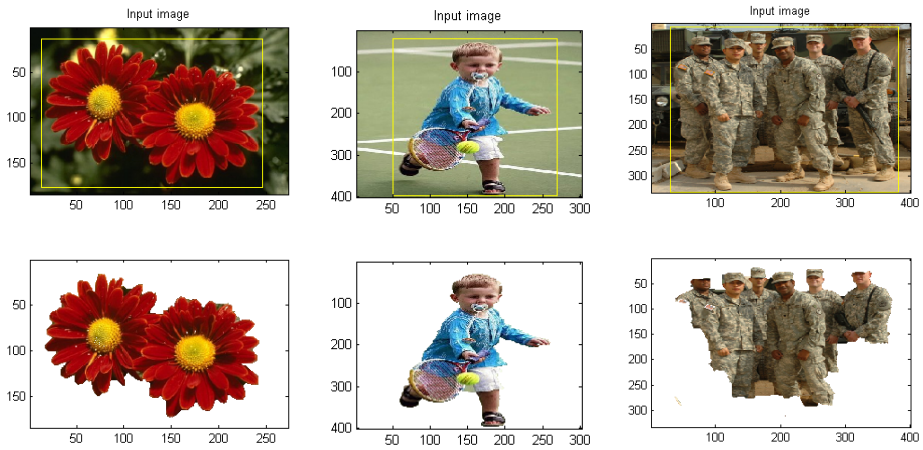
$$D(m) = -\log \sum \pi_i \frac{1}{\sqrt{\det \Sigma_i}} e^{\frac{1}{2}[z_m - \mu_i]^T \Sigma_i^{-1} [z_m - \mu_u]} \tag{2}$$

where the summation goes over the foreground Gaussian components for D_{fore} and over the background Gaussian components for D_{back}.

6 Results

The implementation in MATLAB were tested using some different types of images in order to evaluate the performance and and the correctness of the segmentation.

The result is pretty good but you can see that there are some small parts of the background left around the flowers. This is because there is a big color difference near it that is bigger than the color difference between the flower and the background. There



(a) Flowers (b) A little tennis player (c) Hard image to segment

Figure 2: GrabCut in action

is also some pixels in the middle of the flower that has been classified as background. The reason for this is again that the color in the middle of the flower can variate quite much and are therefore not so hard related to it's neighborhood.

The segmentation of figure 2b went quite well apart from some remaining background under the tennis ball in figure. The line under him interferes a little with the segmentation because of the low cost of cutting along a line.

Figure 2c with soldiers is a very hard image to segment because the models for the foreground and background are very similar.

Running my GrabCut implentation in MATLAB on figure 2a took about 5 min.

7 Conclusion

GrabCut works well when the object of interest has an another color distribution compared to the background. If that's not the case the segmentation could be problematic, at least with the statistical models that I'm using. There are some cases that makes GrabCut fail completely. When I tried to segment out only one soldier in figure 2c GrabCut didn't get any result at all. This is because the models for the background and foreground are very much the same. The result from the algorithm could be adjusted by a final touch up by the user that may improve the result. It will be necessary to do so for certain images.

8 Future work

My implementation is rather slow since speed wasn't my priority this time. However it would gain tremendous in speed if it was rewritten in a vectorizied manner in order to take advantages of MATLABs fast matrix calculations. The total execution time would then decrease a significant amount. An alternative would be to implement in in another language, C++ for example.

Many of the parts that now are slow could be written in such a way that they could be done in parallel. For example the construction of the neighborhood graph and the calculation of the a pixels probability to be assigned to a certain clusters a good candidates for parallelization. It would be fun to see how big images GrabCut can handle if it was implemented to take advantage of parallelization using nVidia CUDA toolbox.

In the original paper they suggested a method to make fading translation between by fitting lines to the around the edges and set the transparency by a function of the distance from that line. That could be useful depending of want you want do do after the segmentation.

It would probably be possible to develop a more sophisticated model for the background and foreground then a Gaussian Mixture Model that only take accounts for the color in each pixel.

References

[1] Justin F. Talbot, Xiaoqian Xu *Implementing GrabCut*. Brigham Young University, Revised: April 7, 2006.

[2] Carsten Rother, Vladimir Kolmogorov, Andrew Blake *"GrabCut" - Interactive Foreground Extraction using Iterated Graph Cuts*. Microsoft Research Cambridge, UK

[3] ORCHARD, M. T., AND BOUMAN, C. A. 1991. *Color Quantization of Images*. IEEE Transactions on Signal Processing 39, 12, 2677–2690