Image Segmentation using Grabcut

Vivardhan Kanoria
Stanford University
Management Science & Engg. Department
vkanoria@stanford.edu

Kevin Truong
Stanford University
Computer Science Department
ktruo001@stanford.edu

Abstract

We implement the grabcut algorithm for image segmentation, as proposed by Rother et al [2]. The algorithm produces state of the art results on several images including known challenging images. This paper discusses the subject of image segmentation broadly, explains the approach taken by Rother et al [2], gives an overview of our implementation and code and discusses some of our results.

.

1. Introduction

We implement an algorithm for image segmentation, using grabcut, based on the work of Rother et al [2] and Boykov and Jolly [1]. Our algorithm produces state of the art results on a data set of 30 images with an accuracy of 92.32% on average for default parameter settings. We present our method, highlighting departures from the implementation of Rother et al [2] if any.

2. Image Segmentation

The project considers the problem of getting accurate pixel level segmentation of a desired object in an image with minimal user interaction. The background of the image cannot be easily subtracted in most settings. This makes the problem a nontrivial one.

2.1. Previous Work

Image segmentation has received a lot of interest from the vision community. A lot of work in the field has used graph cuts, notably, the work of Shi and Malik [4]. Rother et al [2] also give a brief overview of other successful methods in image segmentation, some of which have become part of standard software like Adobe Photoshop or CorelDraw. Grabcut overcomes some issues in these approaches, e.g. it achieves excellent segmentation with minimal user interaction. The algorithm also achieves the global minimum of the segmentation objective whereas some other methods only guarantee a local minimum.

2.2. Problem Formulation

The input to our algorithm is an image with a rectangular bounding box specified by the user, around the object of interest. The desired output is a binary map which assigns a value of 1 to a pixel if it belongs to the foreground (object) or 0 if it belongs to the background. The user input can then be viewed as a set of hard constraints and a prior. The hard constraint is that every pixel outside the bounding box is always background and the prior is that everything inside the bounding may possibly be foreground.

Given this user input, the goal is to create an algorithm that updates the labels of the pixels inside the bounding box so that eventually only the object of interest is labeled as foreground and everything else is background.

2.3. Grabcut Model Formulation

The grabcut approach sets up segmentation as energy minimization problem. The total energy is described by assignment of pixels to a set of 2 Gaussian Mixture Models in color space, one for foreground and one for background. The following notation will be used throughout this paper:

N: Total number of pixels in image
$\mathbf{Z} = \{z_1, \ldots, z_n, \ldots, z_N\}$: Set of pixel RGB values
M: Total number of Gaussian mixture components
$\mathbf{k_n} = \{1, 2, \ldots, M\}$: Gaussian mixture component index of pixel
$K = \{k_1, \ldots, k_n, \ldots, k_N\}$: Pixel component assignments
$\boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_n, \ldots, \alpha_N\}$: Set of segment values, $\alpha_n \in \{0, 1\}$
$\boldsymbol{\pi}_{(i,k)}$: prior probability of cluster k in segment $i \in \{0, 1\}$
$\boldsymbol{\mu}_{(i,k)}$: mean of cluster k in segment $i \in \{0, 1\}$
$\boldsymbol{\Sigma}_{(i,k)}$: covariance of cluster k in segment $i \in \{0, 1\}$
$\boldsymbol{\theta} = \{\pi, \mu, \Sigma\}$: Gaussian Mixture Model parameters
$\mathbf{D_n}$: Unary energy of pixel n
C: Set of neighboring pixels of a given pixel
$\mathbf{V_n}$: Interaction energy between pixel n and its neighbors
$\mathbf{U} = \Sigma_n D_n$: Total unary energy of image
$\boldsymbol{\gamma}$: Parameter for importance of interaction energy
$\boldsymbol{\beta}$: Normalizing parameter for pairwise energy calculation
$\mathbf{E} = U + \Sigma_n V_n$: Total energy of image

The energy for a pixel n has two terms, a unary term (depending only on that pixel) U_n and a binary term (depending on interaction of the pixel with its neighbors) V_n.

The unary term is proportional to the negative log probability of assigning that pixel to a given Gaussian mixture component. It has the following expression:

$$D_n(\alpha_n, K, \theta, z) = -\log\pi(\alpha_n, k_n) - \log p(z_n|\alpha_n, k_n, \theta)$$

Where:

$$p(z_n|\alpha_n, k_n, \theta) = \frac{\exp\left(-\frac{1}{2}[z_n - \mu(\alpha_n, k_n)]^T \Sigma(\alpha_n, k_n)^{-1}[z_n - \mu(\alpha_n, k_n)]\right)}{(\det\Sigma(\alpha_n, k_n))^{1/2}}$$

The binary term of the energy for pixel n is evaluated as follows:

$$V_n = \gamma \sum_{(m,n)\in C} [I(\alpha_n \neq \alpha_m)]\exp(-\beta\|z_m - z_n\|^2)$$

where I is the indicator function, i.e. the term within the summation is 0 if the two pixels m and n belong to the same segment, otherwise it depends on the exponent of the negative squared Euclidean distance between the RGB values of the pixels.

The constant β is calculated by randomly selecting a given number of image pixels and finding the following value:

$$\beta = \frac{2}{E[\|z_m - z_n\|^2]}$$

Where E[x] denotes an expectation. We compute the expectation across all pairs of pixels m and n in our random set.

The total energy E (different from E for expectation above) of the entire image for a given Z, K, θ and α is then defined as:

$$E = \sum_{n=1}^{N} D_n + V_n$$

This energy when iteratively minimized by successively updating K, θ and α using grabcut algorithm gives a good segmentation α on convergence.

3. Algorithm

We describe the important components of the algorithm here. There are 3 main steps after initialization, which are repeated until convergence. Each step in an iteration is minimization of the total energy E with respect to one of the three parameters K, θ or α.

3.1. Initialization

Given the user specified bounding box, we initialize the background to be all the hard constrained pixels ($\alpha_n = 0$, for $n \notin$ bounding box) and the foreground to be all the pixels within the bounding box ($\alpha_n = 1$, for all $n \in$ bounding box).

In the first iteration, there is no set of initialized Gaussian mixtures. Therefore, initialization of the Gaussian Mixture Model component assignment for each pixel is done using the K Means algorithm.

3.2. Assigning pixels to Gaussian Mixture Models

After the first iteration, the pixels in each segment assigned to 1 of M Gaussians in that segment. This is done by finding the energy of the pixel for assignment to each component in that segment and choosing the component with the least energy. This corresponds to the largest probability, i.e. for every pixel n, we compute D_n for k = {1… M} successively, using the parameters {π (α_n, k), μ (α_n, k), Σ (α_n, k)} and choose the k for which D_n is the least. The new set K thus obtained corresponds to an update of the cluster assignments.

3.3. Updating GMM Parameters θ

Having updated K, we now update the Gaussian Mixture Model parameters θ. This is done by looping through each combination of α = {0, 1} and k = {1, …, M}, selecting all the pixels corresponding to that set and recalculating π, μ and Σ. Let $F_{(i,k)}$ be the number of pixels in the set $z_{(i,k)}$ such that:

$$z_{(i,k)} = \{n: \alpha_n = i, k_n = k\}$$

Then we update θ as follows:

$$\pi_{(i,k)} = \frac{F_{(i,k)}}{\sum_{k=1}^{K} F_{(i,k)}}$$

$$\mu_{(i,k)} = \frac{1}{F_{(i,k)}} \sum_{n\in z_{(i,k)}} z_n$$

$$\sigma_{(i,k)} = \frac{1}{F_{(i,k)} - 1} \sum_{n\in z_{(i,k)}} (z_n - \mu_{(i,k)})\,(z_n - \mu_{(i,k)})^T$$

3.4. Updating segmentation α

With the new values of K and θ, we now update the segmentation by minimizing energy over α. This is done using the max flow min cut algorithm [5].

We set up the graph to have one node corresponding to each pixel. In addition, there is a source node S and sink node T, i.e. there are N+2 nodes in the graph. In order to set the weights of the edges, we require some energy terms.

Let D_{n1} be the energy of assigning pixel 1 to a component in the foreground GMM, such that the least energy component is chosen. Similarly, let D_{n2} be the corresponding energy for assignment to the background GMM. Then the weight on the edge joining the pixel to the source node is D_{n2} (background energy) and the weight on the edge joining the pixel to the sink node is D_{n1} (foreground energy). This defines all edges between the pixels and the terminal (source and sink) nodes.

The weight on the edge between 2 pixels is defined as the pairwise interaction term between them, i.e. the individual term evaluated within the expression for V_n. Concretely for neighboring pixels m and n, the weight on the edge between them is:

$$w_{m,n} = \gamma[I(\alpha_n \neq \alpha_m)]\exp(-\beta\|z_m - z_n\|^2)$$

This defines the weights of all edges between pixels.

We now apply the max flow min cut algorithm on the graph so defined. The cut obtained is such that in the residual graph, there exists exactly one edge between each pixel node and a terminal node. If there is an edge to the source node, we set $\alpha = 1$ for the pixel and if there is an edge to the sink node, we set $\alpha = 0$. This gives us the updated segmentation for the entire image.

4. Code

Our codes were implemented with the following structure. We set our model parameters and some useful global variables in the file **initGlobalVariables.m.** The main user interface function is **grabcutRun.m**. This function calls the 3 update steps below, until convergence is achieved:

a. **updateK.m –** function for updating the cluster assignments of pixels
b. **updateGMM.m** – function for updating the Gaussian mixture model parameters
c. **updateAlpha.m –** function that updates the alpha value for each pixel. This function calls the max flow min cut subroutine which is implemented in the mex file **segment_full.cpp**

Our code in grabcut.m is used for random sampling of each grabcut runs, which we will explain in the results section.

4.1. Model Initialization

Our default setting is to use 5 components for the Gaussian Mixture Models for both background and foreground and $\gamma = 50$, as suggested by the Rother et al [2].

4.2. Updating cluster assignments: updateK.m

This file looks at two cases. In the first iteration of grabcut, it initializes the cluster assignments using helper function initK. initK calls kMeans.m or a random initialization routine, depending on the value of the variable initType in initGlobalVariables.m. For subsequent iterations, the updateK.m reassigns pixels to the correct Gaussian Mixture component based on the assignment which has least energy.

The function that computes the energy is computeD (implemented within updateK.m). This function computes an energy matrix of size K x N where K is the number of components (same for foreground and background) and N is the number of pixels. This function is efficiently implemented, requiring just one for loop over components. The correct segment (foreground or background) parameters are chosen by using binary vectors of foreground pixels (alpha_fg_idx) and background pixels (alpha_bg_idx). We use bsxfun along with these two binary vectors to efficiently select the correct Gaussian parameter μ to be used for each pixel. σ and π are selected for each pixel by simply multiplying the binary vectors with the respective foreground and background σ and π values.

Using the energy matrix D returned by computeD, we need only to compute the index of the row of D that has the minimum energy for each column. This set of minimum indices gives the new cluster assignments for the pixels.

4.3. Updating GMM Parameters: updateGMM.m

The file updateGMM.m now updates the parameters of the Gaussian Mixture Model in the obvious way, given the current foreground and background assignment and Gaussian mixture component assignment of each pixel.

We find an indicator vector for all the foreground and background pixels. We loop through each of the components and find pixels in a particular component and segment using a simple & operation on the two binary vectors. This immediately gives us π. We use the new binary vectors to select all the pixels in a given segment and components and there for compute their mean which is μ and their covariance which is σ.

The entries in σ may be very small, in which case we run into an error when we compute its inverse to get the energy of each pixel D. To prevent this, we add a small epsilon to the diagonal of sigma.

4.4. Update Segmentation: updateAlpha.m

This file calls the max flow min cut algorithm. However, we first need to compute the arguments to the max flow min cut mex file.

We first require the unary energy term corresponding to assignment of a pixel to the lowest energy foreground component as well as the lowest energy background cluster. To do this, we use a modified version of the computeD function from the updateK.m file in section 6.2. The modification is required because we now need the minimum energy for both foreground and background, where as previously we only required one of the two. This time computeD returns a 2 x N matrix of energy values. The first row is the energy corresponding to assignment to the lowest energy foreground component. The second row is the analogous term for the background.

We also need to compute V, the pairwise interaction energy between pixels. We implemented both 8 connected as well as 16 connected models (8 directions pixels extend up to 2 pixel units each away). We compute the energy for each pair of pixels with the formula for V_n given previously. The function computeV returns a tensor of size C x N x 2, where C is the number of neighbors (4 or 8) and N is the number of pixels. The entry V (a, i, 1) gives an energy and the entry V (a, i, 2) gives the neighbor pixel index. i.e. the energy between pixel i and pixel V(a, i, 2) is V(a, i, 1).

The arguments D and V are passed to the mex file segment_full.cpp to run the max flow min cut algorithm. We initialize the graph g to have N nodes (excluding the source and sink) where N is the number of pixels. The edge weights between the pixels and the terminals are initialized using the matrix D. The weights between pixel and i and the source and sink are D (i, 2) and D (i, 1), respectively.

We initialize the edge weights between pixels using V. The edge weight between pixel i and pixel V (a, i, 2) is V (a, i, 1). We then call the member function g –> maxflow ( ) and then read the new alpha values using the member function g –> what_segment (i).

This file returns a binary vector of size N (number of pixels), where a 1 corresponds to foreground and 0 corresponds to background.

5. Results

5.1. Baseline

The baseline results of our model are obtained using 5 foreground components, 5 background components, 8-connected pixels, $\gamma = 50$, and random initialization of the pixel GMM assignments. We ran the segmentation test on all images and had an average accuracy of 92.32%. Accuracy is calculated by number of correctly labeled pixel over number of pixel in the image. The 4 best performing images are 'sheep' (99.56%), 'stone2' (98.94%), 'flower' (98.55%), and 'fullmoon' (98.26%). The two worst performing images are 'cross' (72.02%) and 'banana1' (69.33%). These are displayed in Fig. 4. Our results generally had high precision, as most of the object pixels were labeled foreground, but low recall as most of our errors were background pixels being labeled as foreground. We suspect this may be because the number of foreground and background components used, as shown in fig. 1. All the incorrectly labeled pixels in the image belonged to a single GMM component.

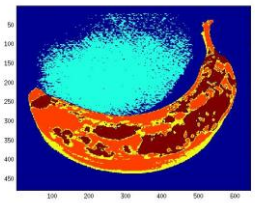


Fig 1. Almost all incorrectly labeled pixels belong to 1 component

5.2. Number of components selection/User Interaction

There are several parameters such as β, γ and M which affect performance of our implementation. Some parameters can be selected based on the image, such as β, whose calculation was described previously. Rother et al have suggested using $\gamma = 50$ and $M = 5$ for both background and foreground.

We tried to find a correlation between the numbers of components that gave optimal performance and the spread in grey values of the user specified foreground and background regions. The idea behind this is that if a given segment is relatively smooth, i.e. the grey values do not vary a great deal, performance will presumably better if we use fewer GMM components to model this low variation. However, this hypothesis turned out to be a poor one. Rather, we found that on hard images (poor pixel accuracy), one of the foreground components, when removed gave nearly perfect segmentation. This can be attributed to having ambiguous segmentation problem instances, where one user might desire the exact output of our algorithm, while others might want to subtract or add certain parts. In general, we have found that the pixels which the user wishes to add or subtract all belong to a single GMM component of the background or foreground respectively. Therefore, the logical way to overcome this problem is just to implement a simple user interaction function that allows the user to toggle all the alpha values of a specific component in either the background or the foreground by clicking anywhere inside that component, e.g. in figure 1, clicking the light blue component would cause it to disappear.

5.3. Number of Neighboring Pixels

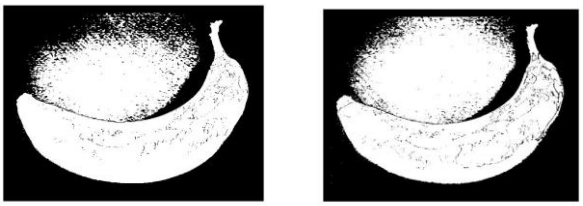
We varied the number of neighboring pixels from 8 (one pixel top/left/bottom/right and diagonals) to 16 (two pixels in each direction) and 0 (no smoothing term). We got the following results:

Image	no smoothing	8 pixels	16 pixels
Sheep	99.25%	99.56%	99.41%
Stone2	98.73%	98.94%	98.91%
Flower	98.36%	98.55%	98.79%
Full moon	98.29%	98.26%	98.33%
Cross	77.08%	72.02%	70.35%
Banana1	75.32%	69.33%	68.19%

Clearly smoothing more only helps slightly in images that we already perform well on, while hurting performance of images that are hard. This indicates that smoothing can have only incremental impact on performance, i.e. it removes some errors caused by hard segmentation in max flow min cut.

In fact, when we had no smoothing, the hard images performed much better while the easy images performed only slightly worse. These effects are shown in the fig. 2, where the

incorrectly labeled background pixels are much more sparse and have more "holes" (pixels inside that are correctly labeled background) in them.



(a) (b)

Fig. 2. No smoothing causes incorrectly labeled background pixels to remain sparse (a) while smoothing makes it dense (b).

5.4. Initializations and Random Sampling

To initialize k for each pixel, we tried both kmeans and a uniform random initialization. We found that the average accuracy for all the images is about the same for the kmeans initialization and random initialization. However, after doing multiple random initializations, we noticed that some images exhibited large variance in accuracy (more than 20) while other images changed less than 1%. For the images with large variance, the accuracy usually increased compared to kmeans. The accuracy never decreased more than 1% below that obtained by kmeans. This could indicate that having a strong prior as with kmeans causes the algorithm to converge quickly to a local minimum close to the global minimum. Random initialization presumably prevents this by its very nature, i.e. being random.

Banana1 had the highest variance in score. Using six different random seeds, the scores were: 94.25% 78.65% 68.08% 68.40% 74.65% 69.99%. Major differences between random initializations of k are that the pixels that are background but labeled as foreground. These pixels seem to be different between each random initialization. The foreground pixels, however, remained foreground pixels between each random run. Therefore, we can run multiple iterations of random initializations and vote for a pixel with each iteration. For each pixel, if it is labeled background in any one of the random initialization runs, then we label it background in the final output. It is only labeled foreground if it is labeled foreground in every run. Using this voting scheme, we have the following results: banana1 (93.97%), cross (72.59%), flower (96.98%), fullmoon (98.69%), stone2 (98.81%), sheep (99.49%).

Most of these images have improved scores, some with very large improvements (banana1 jumped 50% in accuracy). The overall performance of this random sampling method was a better fit for our model, because it allowed us to correct many pixels that were background but labeled incorrectly as foreground. Since our algorithm has high precision and low recall, adding this layer of sampling made its recall noticeably better while, the precision had no significant change. An example of this is the banana segmentation with 4 random initializations (fig. 3). Notice that the actual banana pixels that belong to the foreground had few more pixels that were incorrectly labeled background compared to the previous runs on the banana, but the main gain of this method is the removal of false positive background pixels that we labeled as foreground. The false positives have almost completely disappeared.

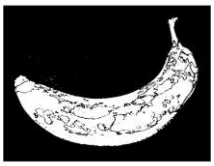
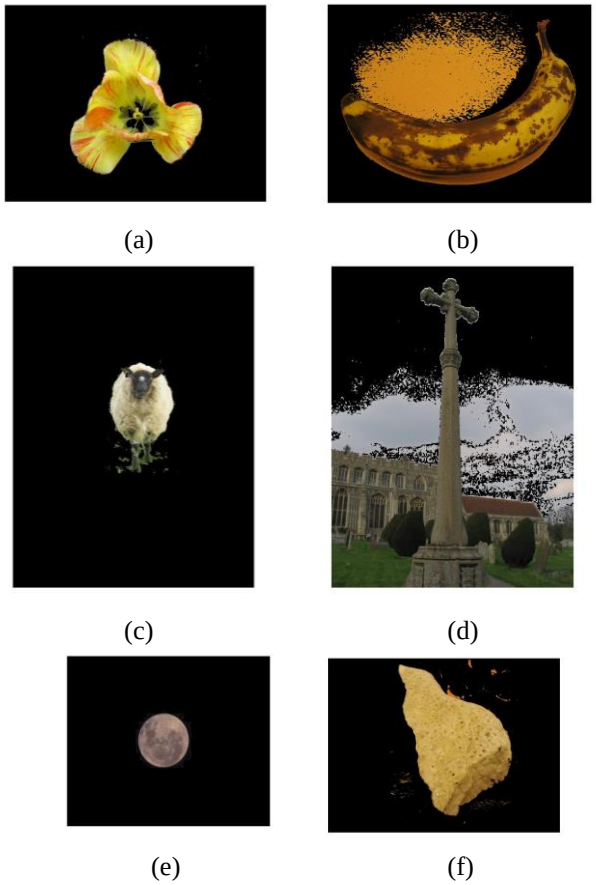


Fig. 3. of voting on multiple random initializations. False positives are very low.



(a) (b)

(c) (d)

(e) (f)

Fig. 4. Results on 4 best {(a), (c), (e) and (f)} and 2 worst {(b) and (d)} images.

6. References

[1] Y. Boykov and M. Jolly, Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images, ICCV 2001.

[2] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut" - Interactive Foreground Extraction using Iterated Graph Cuts. SIGGRAPH 2004.

[3] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-Based Priors, PAMI 2008.

[4] J. Shi and J. Malik, Normalized Cuts and Image Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8), August 2000.

[5] L. Ford and D. Fulkerson, Flows in Networks. Princeton University Press, 1962.