Interactive Image Segmentation with GrabCut

Albert Haque Fahim Dalvi
Computer Science Department, Stanford University
{ahaque,fdalvi}@cs.stanford.edu

Abstract

In this project, we improve GrabCut for image segmentation by using several optimization techniques. We propose an entropy-based gamma computation using spectral histograms for each image. We then employ a one-time user interaction step to correct for mislabeled alpha maps. Our method demonstrates state-of-the-art performance on the GrabCut dataset, achieving an accuracy of 98.1% and Jaccard of 90.7%.

1. Introduction & Related Work

There have been significant past research efforts exploring image segmentation. Initial work done by Shi and Malik [8] used the idea of normalized graph cuts to segment a given image. They provided an approximate solution to the problem of graph partitioning which could be applied to a graph with nodes representing pixels in an image, and the edges representing the similarity between an two given nodes. Boykov and Jolly [1] then introduced another idea in which a user provides "seeds" to indicate regions containing objects or spanning the background. They use these seeds as hard constraints, while introducing softer constraints between the nodes. They then find the optimal solution using the max flow algorithm. Grabcut is heavily based on this technique. Felzenszwalb et al. [2] further build upon this idea of representing images as graphs to solve the problem where there is high variability in a given component of the image. They introduce the idea of coarseness to formally distinguish between good and bad segmentations. Another contribution of their paper was the greedy approach they defined, as it was significantly faster than the older approaches, like the one by Shi and Malik [8].

Lempitsky et al. [5] introduced a prior on the bounding box prior from Grabcut. They proposed that a bounding box can provide more information than just marking the pixels outside the bounding box as background. They defined a notion of tightness so that the process does not result in excessively small segmentations. They ensure that the bounding box "tightly" surrounds their resultant segmentations.

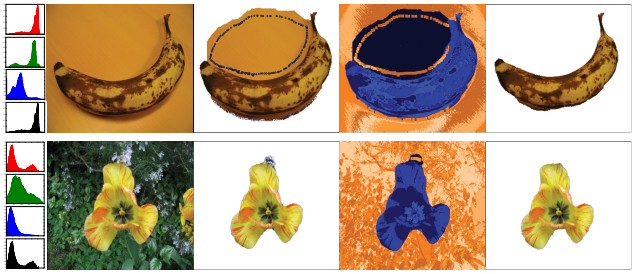


Figure 1: Our Approach. We optimize gamma using the Shannon-entropy of the spectral histogram. Combined with a one-step user interaction step, we achieve state-of-the-art results.

Extending Grabcut with this results in a lower segmentation error.

Additional work was done by the authors in [4] to push image segmentation towards an unsupervised task. One idea, introduced by Joulin *et al.*, used multiple images of the same object to perform automatic segmentation. They used similarity between pixels across two images to define their energy. For example, they look at spatial similarity and discriminative agreement to decide if two pixels (one from each image) belong to the same object. They also provide an optimization solution to minimize the defined energy and achieve a good segmentation.

Finally, Guillaumin et al. [3] introduced the idea of using semantic similarity between images to help with segmentation. Using a segmented source image and a semantically similar target image (i.e. containing a similar object) to transfer the segmentation from the source image and obtain a segmentation for the target image. They use ImageNet, a collection of images grouped together by their semantic similarity.

2. GrabCut

2.1. Algorithm

The Grabcut algorithm by Rother et al. [6] builds upon the idea presented by Shi and Malik [8]. It uses a similar

graph structure, where each node represents a pixel. Each node is connect to the nodes representing its pixel neighbors, and also to a foreground and background node. The idea is that after performing mincut on the graph, the nodes connected to the foreground node will indicate the pixels in the foreground, and the nodes connected to the background node will indicate the pixels in the background.

The crux of the algorithm lies in the way the weights of the edges are assigned, as that is the information that the mincut algorithm uses to partition the image graph. For the rest of the discussion, we will consider *unary* weights to be the weights of the edges from each node to the foreground/background node, and *pairwise* weights to be the weights on the edges connecting neighboring pixels.

The model described by Rother et al. uses Gaussian Mixture Models to represent the color space in the images. We have two GMMs in play, one that represents the foreground and one that represents the background. The number of components in each GMM is set such that the variability in the foreground/background can be captured appropriately. For example, if the background is composed of two colors, yellow and red, one component of the GMM would fit on the yellow pixels, while the other one will fit on the red. To initialize these GMMs, we basically use some form of clustering. In the case of our implementation, we decided to use K-means, as it is relatively fast and clusters similar colors into a single component. We also use a fixed number of gaussian components, namely five, as this seems to work well on a vast number of images.

We first use the bounding box supplied by the user to set our initial guess for the foreground and background pixels. Everything inside the bounding box is considered to be foreground, and everything outside it is marked as background. As mentioned earlier, we run K-means on this initial segmentation to construct the GMMs.

The algorithm then goes through an iterative process of improving the segmentation. We first assign each pixel a component from the background and the foreground GMM. This is done by simply checking which gaussian component in each of the GMMs retursns the highest probability for any given pixel. Next, using our current segmentation (from the previous iteration, or from the bounding box for the first iteration) we recompute the parameters of each GMM. Basically, we take all the pixels currently segmented as foreground and update the mean and covariance of the foreground GMM, and similarly we use the background pixels for the background GMM. We then assign the weights to the edges of the graph as follows: The unary weights come from each of the gaussian. They are computed as the negative of the log probability of a pixel belonging to the GMM. To reduce computational overhead, the paper suggests that we compute the probability of only the assigned component for every pixel. Let u_f represent the weight from the foreground GMM and u_b represent the weight from the background GMM. The pairwise weights, $p_{(i,j)}$ are computed from the difference between the two pixels. Specifically, $p_{(i,j)} = \gamma \exp -\beta \|z_i - z_j\|^2$, where z_i and z_j represent the pixel indexed by i and j. γ here is set to 50 from empirical observations, while β is computed from the expected difference between pairs of pixels from the image. Note that a higher similarity between the nodes would result in a bigger pairwise term. The example graph in 2 shows how the weights are set for each node:

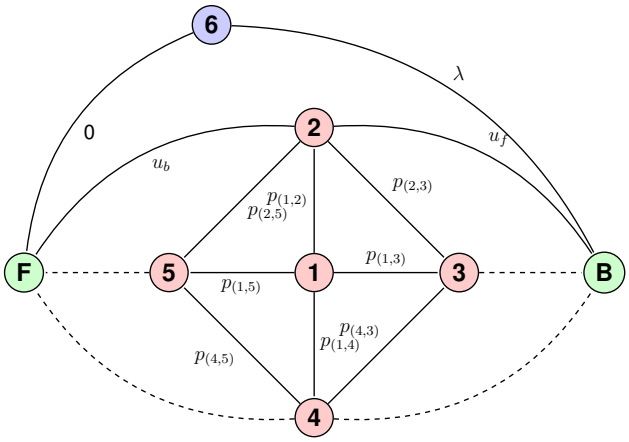


Figure 2: Simplified illustration of GrabCut's internal graph representation. Red nodes indicate pixels inside the bounding box. Blue nodes represent a pixel outside the bounding box (i.e. definitely background). Green nodes represent special foreground and background nodes respectively (i.e. source and sink).

Notice that not all edges are shown in the graph for clarity. For the nodes that are inside the bounding box (red nodes), we assign the unary term from the foreground GMM to the edge between a node and the background node, and the unary term from the background GMM is assigned to the edge between a node and the foreground term. The reasoning behind this is that since the unary terms are negative log probabilities, a lower value indicates a higher probability. Therefore, if the node has a higher probability of being in the background (i.e. a higher probability from the background GMM), u_b would be lower, and the mincut algorithm would cut the edge weighted u_b, thus partitioning the node with the background node. For the nodes outside the bounding box (blue nodes), we assign 0 weight to the edge to the foreground node, and λ weight to the edge to the background node. The idea is to set λ to a very high value so that the edge weighted 0 is always cut, and the node is always assigned as background.

The pairwise terms propagate the information given by the unary terms. For example, if we know a node is definitely background, and a neighboring node is connected

(a) After 5 iterations (b) After 10 iterations

Figure 3: Illustration of two user interaction steps

with a high pairwise weight, we would like the mincut algorithm to not cut this edge, as the neighbor is probably a part of the background.

We then run mincut on this graph, and use the resulting partition to assign each pixel to either the foreground or the background, and re-iterate to get a better segmentation.

2.2. Extensions

Extensions we did including gamma optimization, user interaction, and hyperparameter tuning. Results for each extension are shown in Section 4.

2.2.1 Hyperparameter Tuning

To further improve results, we tuned several GrabCut hyperparameters and show their results in Section 4:

1. Number of iterations
2. Number of GMM components, K
3. Neighborhood size

We adopt a grid-search approach and do not modify the original GrabCut implementation for this extension.

2.2.2 Tight versus Loose Bounding Box

A tighter bounding box not only gives us more pixels that can be marked as definitely background, but also gives our background GMM more information to learn from. Hence, we decided to extract tight bounding boxes from the given ground truth segmentations and run our grabcut algorithm on these bounding boxes.

2.2.3 User Interaction for Alpha Map

Color-skewed images presents an interesting challenge: how do you separate the background from the foreground if they are the same color? To address this, we employ a single user interaction step. After five iterations, we show the current segmentation to the user and prompt them to draw lines on the foreground, whose pixels actually belong the the background. This is illustrated in Figure 3.

Using this new information, we treat it as ground truth and permanently assign the user selected pixels to the background. The alpha trimap is updated and GrabCut continues the iterative minimization step.

2.2.4 Entropy-Based Gamma Optimization

We further address the color-skew problem by employing an entropy-based gamma optimization technique. The motivation is that unary energies capture color information better than the pairwise terms. Pixels with a similar color will have a small pairwise term but may belong to completely different GMM components. We propose to reduce gamma for color skewed images through an automatic entropy-based gamma optimization.

$$H = -\sum_i p_i \log(p_i) \tag{1}$$

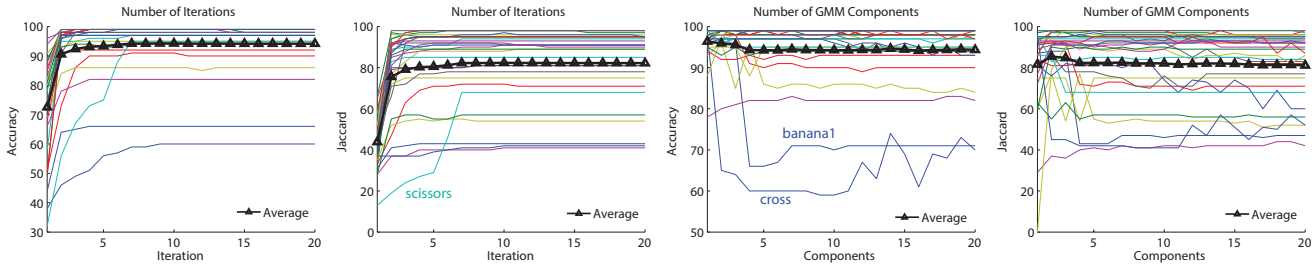
For each image, we compute the spectral histogram by combining the individual RGB color channel distributions. We then compute the Shannon entropy [7] (see Equation 1) and scale gamma accordingly.

3. Code Details

The code is divided primarily into two different sections; the implementation of the grabcut algorithm which can be found in `grabcut.py` and the evaluation code, which can be found in `ml.py`. Inside `grabcut.py`, we have two implementations of each step in the algorithm - a vectorized version which is optimized for speed, and a normal method which is optimized for clarity. For each step (like computing pairwise weights, computing beta etc.) the output of each of the implementation is identical. The evaluation code contained in `ml.py` basically loads the test dataset and runs grabcut on each of the images, comparing the resulting segmentations with the ground truth data to compute the accuracy and jaccard measures. Hence, to get the reported scores, it is sufficient to run:

```
python ml.py -h
```

Detailed help is available upon running each of the file with the `-h` flag. Note that the code depends on the following libraries: `numpy`, `matplotlib`, `argparse`, `sklearn`, `cython` and `pymaxflow` (provided in source package). Detailed instructions are available by running `check_dependencies.py`. We also have a distributed implementation in `ml_remote.py` that uses the corn cluster and a parallel implementation in `ml_parallel.py` to reduce processing time. The normal `ml.py` implementation runs in about $259s$ ($8.6s$ per image on average), `ml_parallel.py` finishes in $365s$ ($12.1s$ per image on average) and `ml_remote.py` finishes in $25s$



(a) Iterations (Accuracy) (b) Iterations (Jaccard) (c) Components (Accuracy) (d) Components (Jaccard)

Figure 4: Hyperparameter tuning results. We experimented with the number of iterations, number of components, neighborhood size, and bounding box. We use 10 iterations, 10 components, a neighborhood size of four, and a loose bounding box (except when tuning a specific parameter).

($0.8s$ per image on average). Note that the parallel implementation was slower on our test machine, but maybe be useful when a machine has several cores.

4. Experiment

4.1. Hyperparameter Tuning

Iterations. We vary the number of iterations from 1 to 20. The results are shown in Figures 4a and 4b. The scissor image begins with a low Jaccard value. This can be attributed to the size of the ground truth segmentation in the image plane. As a percentage of the total image pixels, the ground truth segmentation represents a small percentage. Since the initial foreground estimate is the bounding box, the union is a large value while the intersection is small. This results in a very low Jaccard value. This implies that for images which the foreground object is small (defined by the number of pixels), we should relax the convergence criteron for the number of iterations.

GMM Components. Let K denote the number of components in each Gaussian mixture model. We analyzed the performance of K ranging from 1 to 20 (see Figures 4c and 4d). Some images exhibit very high accuracies when K is small (e.g. cross and banana1). This can be attributed to the color skew of these images. Because few colors represent the image (yellow for banana1 and blue/gray for cross), when $K < 3$, performance is very high. However, when $K > 5$, performance decreases. In a sense, large K causes "overfitting" by assigning multiple components to the sky (the background for cross) when it is one "object." The implication of this experiment is that for images with large color skew, a small value of K should be used.

Neighborhood Size. As expected, increasing the neighborhood size from four to eight pixels increases performance. Four neighbors generates an accuracy and Jaccard of 93.5 ($\pm$9.2) and 80.0 ($\pm$19.5), respectively. Eight neighbors generates an accuracy and Jaccard of 94.2 ($\pm$9.4) and 82.3 ($\pm$17.9), respectively. Standard deviations are reported across images. Because we are able to improve performance without significant computational overhead (implementation is entirely vectorized), we use eight neighbors for our pairwise energy computation.

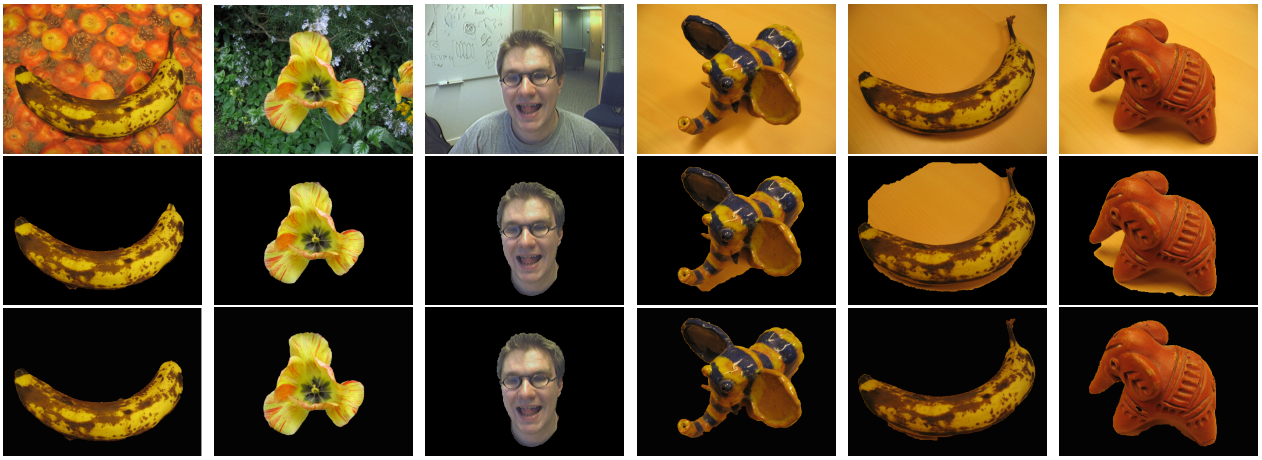
Table 1: Results of optimized gamma

	$\gamma = 50$		Dynamic γ	
Image	Accuracy	Jaccard	Accuracy	Jaccard
fullmoon	99%	92%	98%	75%
stone1	99%	96%	97%	90%
teddy	99%	97%	98%	92%
banana1	66%	43%	68%	44%

Bounding Box. It is important to discuss the definition of a bounding box. In our experiments, we discovered that defining the foreground as strictly inside the bounding box caused performance to decrease on images where the foreground spans the entire width of the image (e.g. cross). Under this definition, the furthest right column will belong to the background. During the iterative optimization phase, this allows the background to "seep" to other areas of the foreground by using this single column as a channel. Redefining the foreground to include pixels on the bounding box edge solved this problem. The results of tight bounding box is shown in Table 2.

4.2. Entropy-Based Gamma Optimization

After running a few experiments, we realized that there is no linear correlation between the entropy of the spectral histogram of an image and an appropriate value of gamma. Our hypothesis was the a lower entropy should hint at a lower gamma value to weight the pairwise term down. Unfortunately, considering some of the low entropy images (see Table 1), performance improved only on one image. This suggests that we need a more informative representation of color in an image, not just the entropy.



(a) banana2 (b) flower (c) person8 (d) ceramic (e) banana1 (f) elefant

Figure 5: Results of GrabCut segmentation with user interaction and gamma optimization. The first four images (columns) perform well. The last two images (far right) perform poorly. First row: original input image. Second row: result after five iterations Third row: segmentation after one step of user interaction. A total of ten iterations have been completed.

Table 2: Algorithm performance on the full dataset

GrabCut + Extension	Accuracy	Jaccard
No Modifications	94.2 ± 9.4	82.3 ± 17.9
Tight Bounding Boxes	95.3 ± 8.0	84.8 ± 16.6
No Modifications (tuned)	95.9 ± 6.8	85.4 ± 15.1
User Interaction	98.1 ± 1.6	90.7 ± 7.4

4.3. Discussion

Table 2 shows the performance of GrabCut with and without extensions. Each of our extensions improves accuracy and Jaccard, with user interaction increasing accuracy and Jaccard by 4% and 8%, respectively, from the baseline GrabCut algorithm.

4.4. Qualitative Analysis

We show six images and their respective segmentations in Figure 5. The successful images (four left columns) do not require any user interaction since their original segmentations are sufficient. The two poorly performing images (two right columns), do benefit from user interaction, however. Note that the poorly performing images have significant color skew, with most pixels lying in the yellow-range. Although the ceramic image contains the same background as banana1 and elefant, it still achieves an acceptable segmentation. The segmentation is not perfect, and the elefant still benefits from user interaction (i.e. the yellow segment underneath the elephant's trunk is gone after the user interaction step).

References

[1] Y. Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *ICCV*, volume 1, pages 105–112. IEEE, 2001. 1

[2] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004. 1

[3] M. Guillaumin, D. Küttel, and V. Ferrari. Imagenet auto-annotation with segmentation propagation. *International Journal of Computer Vision*, 110(3):328–348, 2014. 1

[4] A. Joulin, F. Bach, and J. Ponce. Discriminative clustering for image co-segmentation. In *CVPR*, pages 1943–1950. IEEE, 2010. 1

[5] V. Lempitsky, P. Kohli, C. Rother, and T. Sharp. Image segmentation with a bounding box prior. In *ICCV*, pages 277–284. IEEE, 2009. 1

[6] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG)*, 23(3):309–314, 2004. 1

[7] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mobile Computing and Communications*, 5(1):3–55, 2001. 3

[8] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, Transactions on*, 22(8):888–905, 2000. 1